Universidad de Valladolid

ESCUELA DE INGENIERÍA INFORMÁTICA

GRADO EN INGENIERÍA INFORMÁTICA
Mención en Ingeniería de Software

Un juego para el aprendizaje del método Gitflow

Alumno: Lara Yu Martín Vázquez

Tutor: Yania Crespo González-Carvajal

A todas las personas que quiero

Agradecimientos

A mis padres, a mi hermana y a mi familia que han estado apoyándome durante toda mi vida. Gracias por estar siempre conmigo.

A mi pareja, que ha sabido darme ese empujón cuando lo necesitaba, por estar y por ser ese espacio seguro.

A mis amigos, gracias por darme un hueco para expresarme y ser yo misma, por ser ese momento de desconexión y comprender que la vida también es disfrutar con los que quieres.

A mi tutora Yania, que desde el principio ha confiado en la idea y me ha dado esa confianza que a veces he necesitado cada semana, por muy ajetreada que estuviese.

Resumen

Este Trabajo de Fin de Grado es un proyecto de gamificación que pretende fomentar el aprendizaje de las personas mediante los videojuegos.

El propósito de este trabajo es desarrollar un juego basado en la web que permita aprender los conceptos fundamentales del método Gitflow utilizando metáforas para facilitar su mejor comprensión.

Este proyecto se ha desarrollado teniendo como framework Angular, HTML5, SCSS y Typescript como lenguaje y siguiendo SCRUM como marco ágil de trabajo.

Abstract

This Final Degree Project is a gamification project that aims to encourage people to learn through video games.

The purpose of this work is to develop a web-based game that allows learning the fundamental concepts of the Gitflow method using metaphors to ease its better understanding.

This project has been developed using Angular framework, HTML5, SCSS and Typescript as language and adapting SCRUM as agile framework.

Índice general

Lista de Figuras

Lista de Tablas

Capítulo 1

Introducción

1.1. Contexto

Los orígenes de la industria de videojuegos pueden ser las primeras consolas Nintendo, hacia el año 1980, que pese a que los gráficos eran sencillos y la jugabilidad limitada, asentó las bases para futuros proyectos. Actualmente, esta industria tiene una presencia enorme, no sólo la mejora en los gráficos y la cantidad de plataformas disponibles han contribuido a su crecimiento, sino también el contexto en el que nos encontramos y la conectividad que se puede tener entre distintas partes del mundo, ha propiciado un aumento significativo que se prevé, que siga aumentando cada año [37].

Teniendo en cuenta el impacto que tienen los videojuegos en la sociedad, es lógico que se hayan implementado nuevas técnicas de aprendizaje dentro de un entorno de juegos. Esta técnica de adquirir conocimientos, promover el aprendizaje y resolver problemas a través del juego se conoce como gamificación.

Hoy en día, el auge de la Informática ha supuesto un aumento en el desarrollo de aplicaciones y servicios. El control de versiones es una pieza fundamental en el mundo del desarrollo de software ya que facilita trabajar con otros desarrolladores y permite compartir y mantener un historial del código. Git, es el sistema de control de versiones distribuido más utilizado por la velocidad con la que los cambios que se quieren hacer se reflejan en el código.

Para este proyecto, se ha propuesto gamificar la experiencia de aprender el método de trabajo Gitflow, método que se emplea a la hora de desarrollar proyectos en los que se necesitan liberar versiones del código durante su desarrollo, para proyectos que utilizan Git como sistema de control de versiones. Los conceptos presentes en esta sección se explicarán más detalladamente en capítulos posteriores.

Este proyecto, que se desarrolla en el contexto de la asignatura de *Trabajo Fin de Grado*, es una propuesta de la estudiante que tiene como base aprender de una manera más lúdica el método de trabajo de Gitflow.

1.2. Control de versiones

El control de versiones o control de código fuente es una práctica del desarrollo de software para rastrear y gestionar los cambios hechos en el código [9]. Es importante dentro de los equipos de desarrollo porque no solo permite gestionar estos cambios sino que también permite a los miembros del equipo trabajar de forma simultánea en el mismo proyecto.

Los sistemas de control de versiones son las herramientas que se emplean para ayudar a los equipos de desarrollo, automatizando el proceso de control de versiones. Estos sistemas facilitan el trabajo en paralelo, previenen pérdidas del trabajo realizado y permiten gestionar los cambios a lo largo del tiempo [53]. Estos sistemas hacen un seguimiento de las modificaciones que se van realizando sobre el código de forma que si, durante el proceso de desarrollo, se comete algún error, se puede volver a una versión anterior del código funcional minimizando las interrupciones para el resto el equipo. Además, permite a los desarrolladores mantener la eficacia y agilidad a medida que el equipo crece.

Utilizar un sistema de control de versiones es una práctica recomendada dentro de cualquier equipo de desarrollo ya que facilita el flujo continuo de cambios en el código, sin bloqueos entre miembros, brinda un seguimiento a lo que hace cada persona dentro del proyecto, permite trabajar en diferentes características/funcionalidades (*features*) del proyecto llevando un registro de qué ficheros se han ido modificando, etc. Dentro de estos sistemas los dos tipos más comunes son:

Sistemas de control de versiones centralizados (CVCS) La idea principal es contar con un repositorio global en el que cada usuario debe subir sus cambios para que se vean reflejados en el repositorio global. A la hora de poder ver esos cambios, cada usuario deberá actualizar el repositorio. El flujo de trabajo en este tipo de sistemas sería parecido a este: Tú subes esos cambios. Los demás actualizan. La principal herramienta que adopta este sistema es Subversion (SVN) [19].

Sistemas de control de versiones distribuidos (DVCS) Estos sistemas se basan en la existencia de múltiples repositorios. Cada usuario tiene su propio repositorio y su copia para poder trabajar. De esta forma los cambios que el usuario realiza no estarán subidos al repositorio global, sino a su local. Para poder reflejar sus cambios al repositorio global el usuario deberá "cargar" los cambios del local al remoto. Para poder ver los cambios realizados, primero se tendrán que traer al repositorio local y luego a la copia de trabajo. El flujo de trabajo para estos sistemas es: Tú cargas esos cambios al local, tú subes esos cambios del local al global. Los demás traen los cambios a su repositorio local. Los demás actualizan su copia local [57]. Git es probablemente la opción más popular dentro de este tipo de sistema.

Git es un proyecto de código abierto diseñado por Linus Torvalds en 2005 [8]. Es un sistema de control de versiones distribuido que establece una linea de tiempo con todos los cambios que se hacen en un proyecto y utiliza para coordinar a equipos de desarrollo durante el desarrollo software. Algunas ventajas al utilizar Git son:

Versiones más rápidas Una de las mayores ventajas de Git son sus capacidades de ramificación y el poco coste que tiene fusionarlas. El uso de ramas permiten un desarrollo flexible y simultáneo. Mientras que la rama principal contiene código estable, las ramas *features* guardan el trabajo en curso en un entorno aislado, que se unen a la rama principal tras su finalización. Al separar la rama principal del desarrollo en curso, es más fácil administrar código y enviar actualizaciones rápidamente. El uso de ramas permite organizarte de forma que cada rama esté asociada a una funcionalidad dentro del marco de trabajo ágil [44].

Desarrollo distribuido Al tener un repositorio local, no necesitas conexión a Internet para guardar cambios, revisar versiones anteriores o comparar cambios en el código. Gracias a tener un desarrollo distribuido cada equipo goza de su copia local y el repositorio global, permitiendo trabajar desde distintos equipos más fácilmente.

Existencias de pull-requests La solicitud para incorporar los cambios realizados antes de poder fusionar la rama a la rama principal de trabajo. De esta forma los desarrolladores pueden revisar el código realizado por otro y pueden generarse debates hasta poder fusionar la rama.

La Escuela de Ingeniería Informática de Valladolid (Universidad de Valladolid) utiliza GitLab como repositorio remoto basado en web para el control de versiones, y Git como sistema de control de versiones. Es por eso que aprender en los primeros años de la carrera la forma correcta de utilizar Git puede reducir costes de tiempo a la hora de organizarse con los compañeros en algún trabajo. Es cierto que al principio la organización de estos puede suplirse sin necesidad de emplear ningún control de versiones, pero conforme se va avanzando en la carrera resulta más cómodo apoyarse en este tipo de control de versiones. Como no existe ninguna asignatura obligatoria común a todas las menciones dedicada a esto, muchas veces se aprende a medida que se va utilizando y para gente que está empezando, conocer las cosas más básicas de Git puede resultar positivo ya que va a hacer que el trabajo en el equipo sea más cómodo. Actualmente, en la asignatura de segundo curso Programación Orientada a Objetos, común a todas las menciones, se promueve el uso de Git y GitLab pero se dedica solamente una sesión de prácticas a esto, y es opcional su uso. La asignatura optativa de tercer curso de la mención de Ingeniería del Software "Tecnologías para el Desarrollo de Software" sí desarrolla este tema y practica mucho al respecto, pero es en una mención e incluso optativa dentro de la misma.

1.3. Conceptos de Git

En esta sección se definen conceptos de Git que se utilizarán en las siguientes secciones. El lector familiarizado con estos conceptos y términos puede saltar la siguiente sección.

Repositorio. Un repositorio de Git es un almacenamiento virtual de un proyecto que permite guardar versiones del código a las que se puede acceder cuando se necesite [4].

Commit/s. Se utilizan para capturar exactamente el estado del proyecto en un momento concreto, después de realizar la captura, esta es almacenada en el repositorio local de Git [33].

Ramas. Las ramas en git son una división del estado del código, se crean en cualquier momento del desarrollo del proyecto [34].

Rama principal. La rama principal se denomina main, aunque en versiones previas de Git se llamaba master. Solo existe una rama main en cada repositorio. Es usual que haya otro tipo de rama principal llamada develop, vinculada a los conceptos que promueve GitFlow (ver Sec. 2.1). El objetivo es que no reciban código de forma directa a través de commit, siempre tienen que recibir código a través de ramas auxiliares. [43].

Rama auxiliar. Ramas a las que los desarrolladores pueden hacer commits y pueden existir varias de ellas a la vez. En GitFlow (ver Sec. 2.1) se tienen diferentes tipos de ramas auxiliares [43].

Comandos. Instrucciones para que Git realice diversas tareas como crear una rama, o borrarla.

git branch. El comando git branch permite crear una rama nueva a partir de una ya existente.

merge. Permite tomar las ramas creadas por git branch e integrarlas en una sola.

pull request/merge request/push request. Es una funcionalidad llamada así en github, gitlab y bitbucket respectivamente, en la que un colaborador pide que revisen sus cambios antes de hacer merge a una rama [17].

1.4. Sistemas de ayuda al aprendizaje para Git

A la hora de aprender una nueva destreza se puede realizar mediante varios métodos. El método más tradicional, en el que, para momentos específicos, buscar algo particular es lo único que se quiere, puede ser de utilidad. Pero normalmente, a la hora de querer aprender desde cero, empezar a buscar en Internet o libros sobre el tema, puede resultar abrumador. Lo mismo sucede para el caso de Git.

Por eso, y en general, existen numerosos tutoriales, cursos y vídeos sobre Git para aprender desde las cosas más básicas hasta algunas más complejas. En Internet se encuentran muchísimas páginas que facilitan al usuario el aprendizaje de Git, que reúnen de forma más visual, y precisa los comandos de Git, ordenando estos por lecciones y por orden de uso en un proyecto real. A continuación se presentan dos de ellas.

1.4.1. Git Inmmersion

Git Immersion [27] es un proyecto que te guía sobre los comandos fundamentales de Git permitiéndote seguir el tutorial aplicando los comandos que se han utilizado explicando la salida que deben tener.

Esta web tiene su base en la conferencia impartida por Jim Weirich [63] en la *Rails Conf 2010* sobre Git [28]. En esta conferencia se explica de forma visual como se construye y diseña de forma correcta un sistema de control de versiones desde cero para comprender los conceptos subyacentes de Git [29]. Esta conferencia se puede encontrar en su página web [26].

Git Immersion agrupa los laboratorios del tutorial impartido por Jim Weirich para acceder de forma más fácil a estos.

1.4.2. Tutorial de BitBucket

Bitbucket [3] es un servicio de alojamiento basado en web, para los proyectos que utilizan el sistema de control de versiones Mercurial y Git [60]. Perteneciente a la empresa Atlassian, BitBucket tiene un tutorial que enseña los conceptos básicos de Git, integrándolo con la plataforma que facilita su uso, Bitbucket Cloud. El tutorial básico dura unos 30 minutos, aunque existe un conjunto de tutoriales que profundizan en los comandos de Git. Estos tutoriales se pueden encontrar aquí [6].

1.5. Gamificación

Gamificación o ludificación traducido al español, proviene de la palabra anglosajona *gamification*. Como refleja su nombre, está relacionado con el ámbito lúdico. Este concepto es relativamente nuevo y ha empezado a ganar popularidad en el siglo XXI como una forma de potenciar y mantener la motivación, el esfuerzo y otros valores que se consigue por medio de los juegos [58].

Gracias a los videojuegos, este término ha cobrado mucha más importancia últimamente. Según un informe elaborado por *Newzoo* en 2022 [52], cuatro de cada cinco españoles de entre 10 a 65 años juegan a algún videojuego dentro de su tiempo libre y, con respecto al año anterior, el sector del videojuego facturó ese año aproximadamente un 32 % más. Siguiendo ese pensamiento, es notorio que el crecimiento de los videojuegos va a seguir aumentado con el paso de los años y por ende, la gamificación también seguirá presente.

Sin embargo, incluso antes de los videojuegos, se utilizaban ciertas métodos como el uso de *rankings* o *leaderboards* mediante los cuales se trataba de motivar a los trabajadores por lo que podemos decir que la gamificación existe desde mucho antes.

No hay que confundir gamificación con juego serios, ya que en ambos casos el eje principal es integración de las mecánicas de los juegos y en ocasiones entender la diferencia que existe entre ellos puede estar difusa.

Los juegos serios son juegos completos diseñados para que conocimientos aplicados en ese juego puedan aplicarse en la vida real. Es decir, coloca dentro de un juego problemas del mundo real para hacerlos más fáciles de entender y entretenidos de resolver [22]. Por otro lado, la gamificación no es un juego en sí, si no que utiliza mecánicas de juego en actividades

cotidianas para motivar a la persona involucrada y presentar una actividad atractiva sobre ámbitos reales que no son juegos. Es decir, toma elementos de los juegos y los lleva a problemas del mundo real. En el caso de este proyecto, el aprender Git es un problema real en el ámbito del desarrollo de Software y mediante el uso de mecánicas de juego, se quiere hacer la experiencia de aprendizaje más divertida.

Entonces, la gamificación [31] consiste en utilizar mecánicas de juego, estética y pensamiento típico de los juegos para involucrar y motivar a las personas, promover el aprendizaje y resolver problemas. Teniendo en cuenta la definición anterior, se desglosa en ciertos puntos que debe tener cualquier desarrollo dedicado a la gamificación:

Juego El objetivo básico es crear un sistema definido por reglas, retroalimentación y resultados cuantificables con el fin de obtener reacciones en el que los jugadores, consumidores, empleados, ..., quieran invertir su tiempo, su energía y sus conocimientos.

Mecánicas Elementos generales que se utilizan en muchos juegos que son importantes a la hora del proceso de gamificación. Se incluyen niveles, obtención de medallas, puntos, restricciones de tiempo, etc.

Estética La interfaz del usuario, el *look and feel*, es importante porque una buena interfaz, no solo estéticamente más bonita, sino fácil de utilizar evoca a los usuarios a tener mayor predisposición para jugar.

Pensamiento típico Es la idea de convertir una experiencia en una actividad que agrupe elementos como competición, exploración, desarrollo de la historia. Es decir, transformar una actividad cotidiana en un juego.

Compromiso Una de los objetivos principales es ganarse la atención del usuario e involucrarle en el juego. Así mismo, la motivación que debe brindarles la aplicación debe ser la suficiente para que el usuario mantenga esa atención. Para ello el juego no debe ser ni muy fácil ni muy difícil para que el usuario quiera seguir dentro del proceso.

Promover el aprendizaje Esto se puede emplear para el aprendizaje ya que muchos elementos están basados en técnicas que se aplican dentro de la enseñanza y que profesores han estado utilizando durante años. La diferencia entre lo que se ha estado haciendo y la gamificación es la integración de motivar y aprender al mismo tiempo dentro del marco de un juego.

Resolver problemas La gamificación se aprovecha de la naturaleza competitiva que tienen los juegos para que los usuarios lo hagan lo mejor posible.

La importancia de implementar el desarrollo de gamificación en el ámbito educativo reside en que los métodos tradicionales de aprendizaje poco a poco están perdiendo popularidad y el tiempo y la atención de los alumnos resulta limitado por lo que encontrar una solución que aproveche al máximo esto es beneficioso para ambas partes, el profesor y el alumno [31].

1.6. Alternativas existentes

1.6.1. Oh My Git!

Oh My Git! [10] es un juego de código abierto que pretende enseñar conceptos básicos de Git mientras se juega. En cada nivel, aparecen una serie de cartas que se pueden seleccionar y mover por la pantalla, y que tras su uso, la interfaz reflejará la acción descrita en la carta. El objetivo principal es lograr los objetivos que están en rojo descritos en cada nivel sean verdes. Esto se puede ver en la Figura 1.1, de forma que se vayan completando sin destruir los otros objetivos presentes que ya hayas conseguido, mostrando al usuario los resultados de las acciones que va tomando inmediatamente mientras juega. De esta forma, el cambio que producen las cartas se refleja automáticamente al utilizarlas y hace que sea una buena experiencia. Cada nivel se introduce mediante una metáfora de viajes en el tiempo y cápsulas temporales y objetos cotidianos de la vida que facilitan y hacen más amena la experiencia de aprender los comandos.

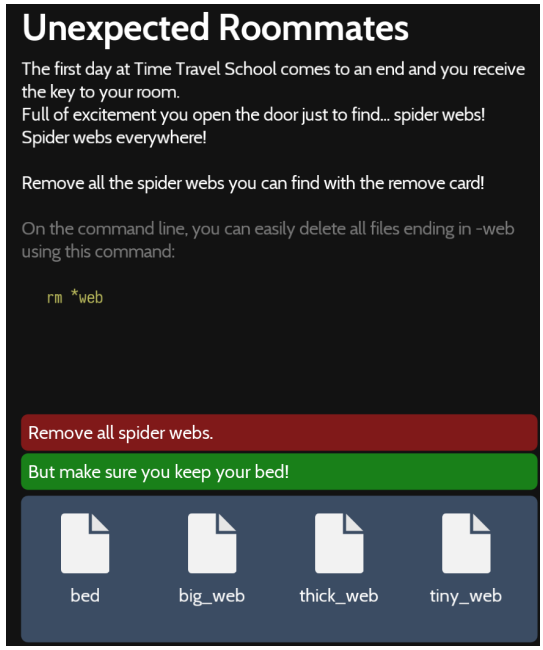


Figura 1.1: Historia del nivel 2.1

Para usuarios más avanzados, se le introduce al usuario una terminal integrada para que pueda ejecutar y probar los comandos. Además, por cada nivel que se complete sin el uso de las cartas se otorgará una chapa dorada por haber completado el nivel solo con la linea de comandos. La Figura 1.2 muestra la estructura que tiene cada nivel.

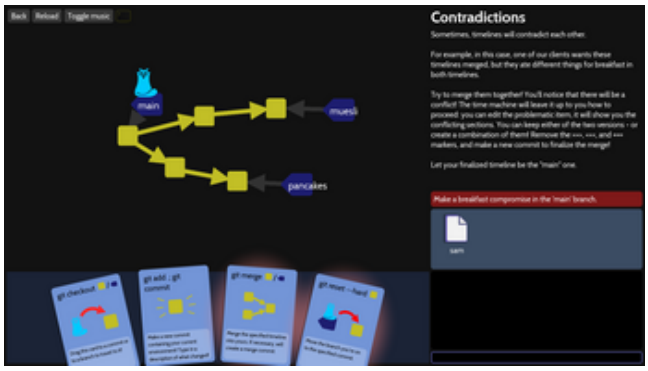


Figura 1.2: Captura de pantalla de un nivel del juego

1.6.2. Learn Git branching

Learn Git Branching [54] es una página web que ofrece una forma más visual para aprender los comandos básicos tanto los principales como los comandos dedicados al remoto. En cada nivel hay un objetivo a cumplir que se muestra en la parte izquierda de la pantalla. Antes de poder introducir una nueva línea en la terminal, hay una demostración previa acompañada de una animación sobre los comandos nuevos que se pueden aplicar. Esta demostración se ve reflejada en la Figura 1.3. En cada lección el usuario debe conseguir el objetivo introduciendo comandos en la terminal sin limite de tiempo y pudiendo revisar el tutorial de los comandos que se han explicado en el nivel.

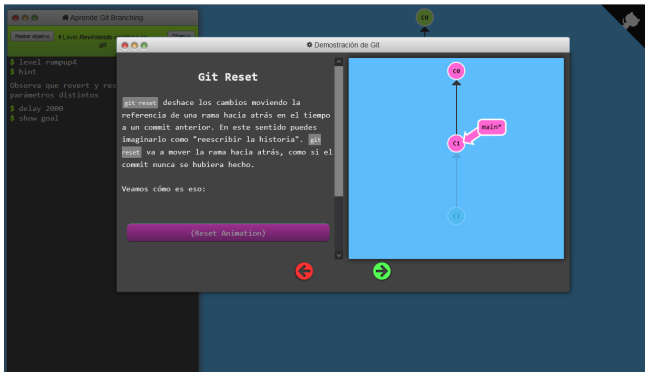


Figura 1.3: Captura de pantalla del nivel 2.4

En LearnGitBranching el objetivo es obtener la figura que se muestra a la derecha de la pantalla utilizando la consola de comandos. Al terminar el nivel, en la Figura 1.4 se refleja la finalización del nivel, en la que se muestra un texto con el número mínimo de comandos que los autores *LearnGitBranching* han utilizado para conseguir el objetivo y el número de comandos que ha hecho el participante. Si es igual o inferior perfecto, pero si es superior, se da la oportunidad de volver a intentarlo para igualar el número.

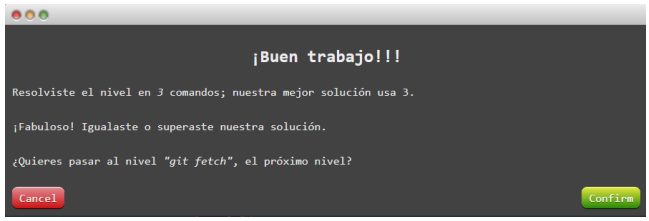
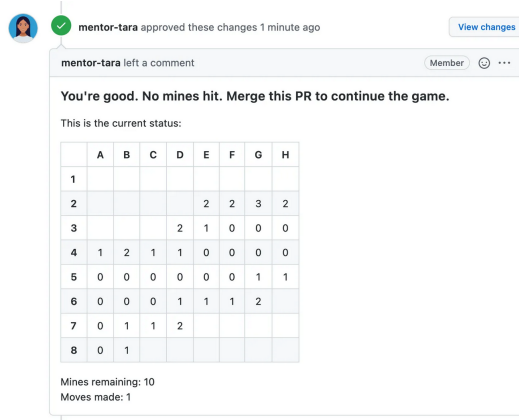


Figura 1.4: Captura de pantalla al finalizar el nivel 6.2

1.6.3. GitHub Minesweeper

GitHub Minesweeper [30] trata Git desde la perspectiva del trabajo en equipo ya que está dedicada al flujo de trabajo que se puede generar entre distintos compañeros actuando sobre un repositorio común. Los desarrolladores en general trabajan en colaboración con otros y se deben conocer las herramientas colaborativas existentes. En esta web se puede practicar esas interacciones que se pueden dar en equipos de trabajo reales, utilizando un bot como compañero de equipo. El juego en sí consiste en ir limpiando las casillas de forma que tu compañero de equipo sea quien te diga si la casilla está cerca de una mina o no. Para cada "movimiento" se realiza su rama y su commit y se espera a la respuesta del bot. Si se evita una mina, el juego continua, se solicita una pull request que será aprobada por el bot y se fusiona la rama creada con el movimiento a la principal.



	A	B	C	D	E	F	G	H
1								
2					2	2	3	2
3				2	1	0	0	0
4	1	2	1	1	0	0	0	0
5	0	0	0	0	0	0	1	1
6	0	0	0	1	1	1	2	
7	0	1	1	2				
8	0	1						

Figura 1.5: Captura de pantalla del tablero al limpiar la casilla A8

Una vez finalizada la parte guiada, se podrá jugar tantas veces como se quiera. De esta manera, se pueden repetir los pasos hasta que no se necesite consultar las guías de referencia proporcionadas y el flujo de trabajo sea algo que se haga de forma natural.

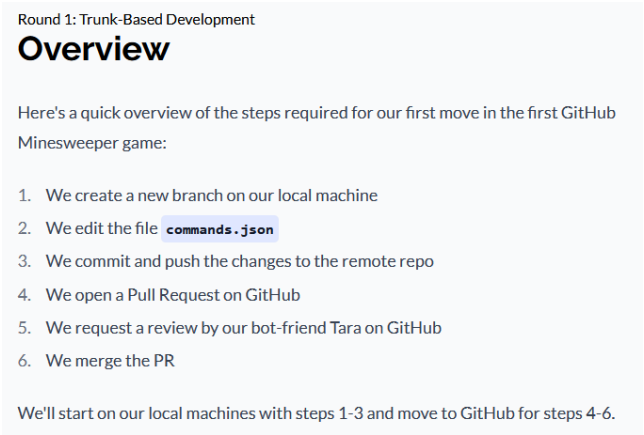


Figura 1.6: Captura del inicio del juego

1.7. Objetivos

Este Trabajo Fin de Grado pretende ayudar a que el usuario conozca y aprenda las bases del método de trabajo Gitflow, transformando ese aprendizaje para que sea lúdico y motivador. Se identifican los siguientes objetivos que pretende cubrir el Trabajo Fin de Grado:

1. Diseñar un juego orientado al aprendizaje del método Gitflow.
2. Creación de temáticas y mecánicas que motiven a los jugadores a participar.
3. Creación de niveles en el que se expliquen los conceptos necesarios.
4. Reflejar logros obtenidos en el juego con el fin de motivar a los jugadores terminar los niveles para completarlos.
5. Diseñar e implementar una plataforma web para la realización del juego.

Los objetivos personales propuestos para este proyecto son los siguientes:

1. Conocer y entender el marco de trabajo SCRUM.
2. Aprender más sobre Git y Gitflow.
3. Conocer más la tecnología Angular.

1.8. Estructura de la memoria

Este documento se estructura de la siguiente forma:

Capítulo 1 Introducción: Este capítulo recoge las nociones básicas e información necesaria para entender el origen de la idea, las alternativas que pueden existir y la motivación y objetivos que hay detrás de este proyecto.

Capítulo 2 Documento de Diseño de Juego: Este capítulo está dedicado a aglomerar toda la información necesaria que se requiere a la hora de definir el concepto del juego que se va a desarrollar como son las metáforas, mecánicas, personajes, niveles, etc.

Capítulo 3 Requisitos y planificación: Este capítulo describe la adaptación del marco de trabajo ágil SCRUM y el proceso de desarrollo seguido en el proyecto, adaptación que necesaria al ser un grupo reducido quienes forman parte del mismo y las características del proyecto y del equipo de trabajo (un TFG realizado por una estudiante que compagina ese trabajo con otras actividades). También se describe la planificación inicial que se ha creado a partir de la elicitación de requisitos inicial, el alcance del proyecto, el presupuesto y el análisis de los riesgos que pueden surgir durante el avance.

Capítulo 4 Análisis: En este capítulo se detallan los modelos de clase y diagramas iniciales que surgen tras el análisis del Documento de Diseño de Juego y los requisitos descritos como historias de usuario.

Capítulo 5 Tecnologías utilizadas: Este capítulo resume las tecnologías que se han utilizando durante el desarrollo del proyecto, abarcando aquellas utilizadas para la comunicación y documentación, análisis, diseño e implementación de la aplicación.

Capítulo 6 Diseño: Durante este capítulo se detalla el diseño arquitectónico seguido, así como los patrones, diagramas y modelos creados que surgen de las decisiones de diseño tomadas.

Capítulo 7 Implementación y pruebas: Este capítulo está dedicado a detallar todo aquello que ha surgido durante la implementación de la aplicación como, por ejemplo, la organización del código, principios seguidos, bocetos, etc, además de todas las pruebas y resultados obtenidos tras la finalización de la misma.

Capítulo 8 Seguimiento: Este capítulo recoge todos los Sprints que se han realizado, y el trabajo ejecutado en cada uno de ellos. Al final del capítulo también se recoge un resumen del proyecto y las diferencias que han existido entre lo ejecutado y la planificación inicial.

Capítulo 9 Conclusiones: En este capítulo se recogen las conclusiones y líneas futuras del proyecto.

Anexo A Manuales: Incluye manuales de instalación, mantenimiento, y de usuario.

Anexo B Resumen de enlaces adicionales: Incluye enlaces de interés sobre el proyecto, como el repositorio de código y otros artefactos.

Capítulo 2

Documento de Diseño de Juego

El Documento de Diseño de Juego, o por sus siglas en inglés, *Game Design Document* (GDD) es un documento que se suele definir en los videojuegos y que recoge muchísima información sobre estos. Este documento está en constante evolución ya que suele crearse en las primeras fases del desarrollo y, a medida que se va avanzando en la definición de lo que va a ser el juego y sus características, el documento se irá modificando [62].

En este Capítulo se describirán las secciones que se han considerado importantes para el GDD de este proyecto.

El objetivo principal del proyecto es utilizar la gamificación para aprender Git haciendo énfasis en aprender el método de trabajo Gitflow. Se quiere que el usuario asimile el flujo de trabajo que existe entre las distintas ramas de una manera más cómoda para que sirva como base y tenga los conocimientos básicos a la hora de abordar un proyecto siguiendo Gitflow.

2.1. Conceptos de Gitflow

En esta sección se definen conceptos con relación al método Gitflow, que no han sido explicados en la Sección 1.3 que se utilizarán durante el documento de diseño de juego. El lector familiarizado con estos conceptos y términos puede saltar la siguiente sección.

Rama main. Es una convención de nombre para la rama principal en un repositorio.

Feature Es el desarrollo de nuevas características o funcionalidades del proyecto.

Rama feature. Un tipo de rama auxiliar, específica para cada feature nueva que quiera implementarse en el proyecto. Estas siempre parten de la rama develop.

Rama develop. Esta rama guarda las ramas feature que van surgiendo a lo largo del proyecto, por lo que, cuando estas se crean, deben tener como base la rama develop y tras su finalización unirse a ella.

Rama release. Un tipo de rama auxiliar, que se bifurca en las situaciones donde la rama develop tenga lo necesario para realizar una publicación [32].

Rama hotfix. Es un tipo de rama auxiliar que se utiliza para la depuración del código cuando se identifica un defecto crítico que debe resolverse durante el ciclo de publicación.

Ciclo de publicación Cuando develop haya adquirido suficientes funciones para una publicación se crea una rama release a partir de develop. A partir de este punto no pueden añadirse nuevas funciones y en esta rama solo deben corregirse errores, documentación y otras tareas que surjan en relación a la publicación. Cuando está lista para el lanzamiento, la rama release se fusiona en main y se etiqueta con un número de versión. Además, debería volver a fusionarse en develop, ya que esta podría haber progresado desde que se inició ese ciclo.

Fusión Hablamos de fusión para referirse a la unión de cierta rama a otra distinta. En Git se denomina merge que es el nombre del comando que realiza la fusión de ramas.

2.2. ¿Qué es Gitflow?

Gitflow es una forma de organizar el trabajo y tener una estructura bien definida a la hora de desarrollar un proyecto de Software. Es un modelo de trabajo que se utiliza para organizar las ramas y dar una misión a cada una de ellas. De esta forma se asignan funciones específicas a cada rama para luego poder interactuar con las demás [5].

Al iniciar un proyecto existe una rama main y una rama develop en las que se irá guardando el historial del proyecto. La rama main guarda el historial con las distintas publicaciones y la rama develop guarda el historial de las funciones implementadas.

Cada nueva función que se implementa se guarda en su propia rama (ramas feature) y siempre parte de develop. Tras su finalización, se junta en la rama develop de nuevo por lo que las ramas feature solo van a interactuar con esa rama.

Tras la finalización de varias features, se crea una rama release a partir de develop iniciándose en este momento el ciclo de publicación. En este punto no pueden añadirse nuevas funciones y solo se podrían corregir errores que surgiesen. Tras las correcciones, si las hubiera, release se fusiona con main y se etiqueta el número de versión. Esta rama release se fusiona también con develop, ya que al haber podido corregir errores las ramas deben estar actualizadas.

Las ramas para la corrección de errores de la rama main son las hotfix. Se emplean para corregir rápidamente errores de las publicaciones en producción. Tras su corrección en una rama hotfix, se fusionará en main y se etiquetará con una nueva versión. Se fusionará con develop o release dependiendo de que rama exista para actualizarla.

En la Figura 2.1 se refleja el flujo comentado en esta sección.

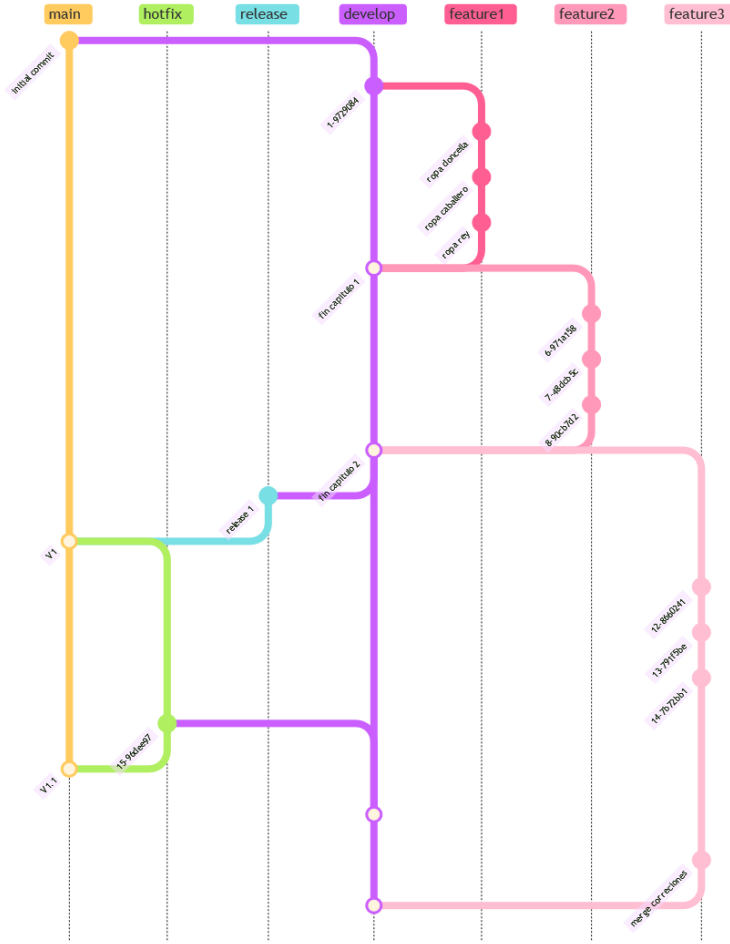


Figura 2.1: Flujo de trabajo de Gitflow

2.3. Adaptación de Gitflow a una historia

La idea inicial de simular una historia y ver reflejado los cambios que el usuario realiza, de cierta forma, viene del juego *Storyteller* [13] en el cual, mediante una serie de escenas predefinidas, unos personajes y la combinación entre ellos, se debe lograr el objetivo del nivel. Se puede ver la estructura de un nivel de ese juego en la Figura 2.2.

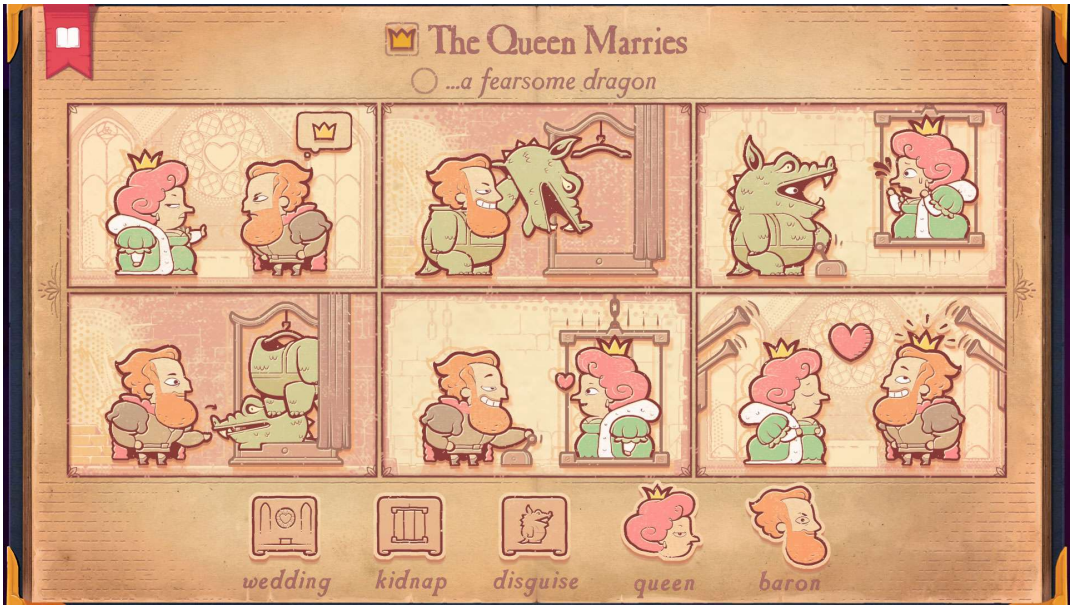


Figura 2.2: Escena del juego: Storyteller

Partiendo de esa idea superficial, se han creado una analogía entre los conceptos relacionados con una historia y las ramas que tiene Gitflow:

Main - Historia. La propia historia de todos los personajes. Recoge todas las aventuras de todos los personajes.

Hotfix - Errores. Ciertos errores que surjan en la historia que no se han podido localizar antes.

Release - Checkpoints. Ya que la rama release se emplea como una forma de controlar lo que se publica en cada entrega, se puede encontrar una similitud en los checkpoints de los juegos. Los checkpoints son marcas que permiten guardar el progreso del jugador en un momento concreto. Los checkpoints suelen aparecer tras una serie de acontecimientos que se ha tenido que realizar previamente.

Develop - Manuscrito. Se puede asemejar a un manuscrito en el que se van a ir añadiendo las *features* necesarias.

Feature - Capitulo. Cada *feature* es una funcionalidad a implementar, por lo que como las historias se subdividen en capítulos podemos emplear los capítulos como si fueran *features* a implementar.

Teniendo esa analogía, podemos intentar darle un significado a cada una de las ramas, ajustando también cómo debería ser el flujo de trabajo siguiendo el flujo de Gitflow:

1. Se crea la rama de trabajo main. La propia historia de los personajes.
2. Se crea una rama develop, que sería el manuscrito de la historia, donde todos los personajes aportan sus ideas. Es un documento en el cual se van a ir añadiendo todos los capítulos.
3. Se crean features, que serían cada capítulo que quiera tener la historia. Teniendo en cuenta que en este caso, los capítulos salen continuados y no habría conflictos.
4. Cuando se acaba esa feature, se unirá a develop. Se irán subiendo capítulos hasta llegar al final de cada nivel.
5. Tras tener todos los capítulos que se crea conveniente se crea una rama release la cual sirve para tener un checkpoint sobre lo que va a englobar cierta versión. Cuando esté listo se fusiona en main para etiquetar el número de versión y luego en develop por si hubiera habido algún cambio con respecto a develop.
6. Si surge algún error, se crea una rama de correcciones desde main y se corrige el error. Una vez corregido, se subirían esos cambios a la rama main y se fusionarían también con develop, para que las ramas estuvieran actualizadas.

El objetivo principal del juego es finalizar los niveles correctamente. Para cada nivel habrá una serie de logros disponibles los cuales se van a ir obteniendo a medida que se avance en el juego o se escojan diversas opciones.

2.4. Mecánicas

En esta sección se explica las distintas mecánicas y la interacción que tiene el usuario con el juego.

En primer lugar se tiene en cuenta las distintas pantallas existentes que va a tener la aplicación. Ya que se está enfocando el juego como una historia con capítulos, en cada nivel se explicará una rama. Dentro de este capítulo habrá una serie de pantallas en las que se describirá el problema u objetivo que se debe resolver. Para ello se han definido los siguientes tipos de pantallas:

Pantallas de introducción (ver Figura 2.3a): En estas pantallas el usuario solo debe leer, ya que serán pantallas en las que solo haya texto. Estas son necesarias, sobre todo al inicio de cada nivel ya que aportan un contexto de lo que se quiere conseguir en cada nivel. Este texto estará enfocado en terminología de la historia y no tanto en terminología de Git y Gitflow.

Pantallas de contexto (ver Figura 2.3b): Pantallas en las que se le introduce el comando que se va a aprender relacionándolo con los comandos de Git.

Pantallas de opción (ver Figura 2.3c). En estas pantallas se le dará al usuario la opción de elegir entre varias opciones, con el fin de simular esas modificaciones por parte del usuario.

Pantallas de flujo (ver Figura 2.3d): Son las pantallas en las que el usuario introduce por linea de comandos los comandos que se requieren. Además en esta existirá una línea en la que se reflejará cada rama y la interacción entre ellas.

Pantalla de información (ver Figura 2.4): Son las pantallas existentes para poder consultar toda la información necesaria para el nivel. Existen durante todo el nivel y siempre son accesibles, exceptuando en las pantallas de introducción. De esta forma se puede consultar fácilmente.

Para cada tipo de pantalla el usuario podrá interactuar con ellas mediante botones o inputs de texto como se indica a continuación.

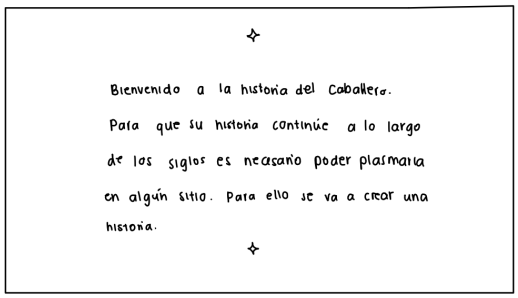
En las **pantallas de introducción** la única interacción que van a tener es cerrar el texto que se muestra. Ya que en estas pantallas lo que se quiere es aportar información y contexto sobre el nivel.

En la **pantallas de contexto** el usuario tendrá que avanzar hacia la siguiente pantalla haciendo clic en algún componente de la pantalla. Además, podrá consultar información útil ya que va a haber un componente dedicado a mostrar ese tipo de información.

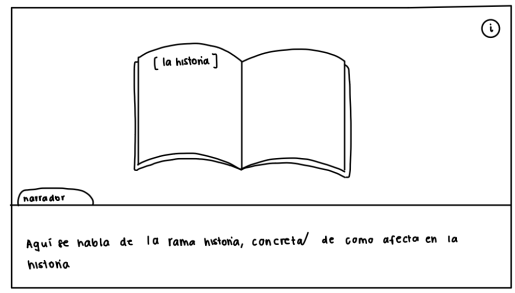
En las **pantallas de opción** el usuario podrá elegir de entre las opciones disponibles una de ellas. El usuario deberá hacer clic en una de las opciones mostradas en pantalla.

En las **pantallas de flujo** el usuario tendrá que escribir los comandos necesarios para poder seguir el flujo que se está explicando.

Otro punto importante es ver la interacción que habrá entre cada una de las pantallas descritas, ya que al ser un juego lineal, estas tendrán un orden similar en cada nivel.

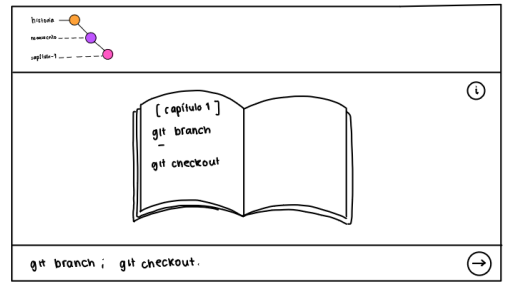


(a) Pantalla de introducción



(b) Pantalla de contexto

(c) Pantalla de opción



(d) Pantalla de flujo

Figura 2.3: Organización existente dentro de cada nivel

1. Todo nivel inicia con una pantalla de introducción para poner al usuario en situación de lo que va a pasar en este.
2. Tras el introducción, puede existir o no, una pantalla de contexto del comando o comandos que se van a emplear en las pantallas de flujo. Si se ha determinado que esa no es necesaria ya que repite conceptos explicados anteriormente, esta se omitirá.
3. La siguiente –que puede o no aparecer–, en función del nivel en el que se encuentre el usuario, es la opción que se le muestra al usuario únicamente cuando se necesita que interactúe aportando una serie de "datos" como si fueran modificaciones de ficheros.
4. A continuación el usuario deberá interactuar con el flujo para poder resolver el objetivo del nivel. Estas suelen ser más de una ya que se le dará al usuario una pantalla de flujo por un número de comandos introducidos, correspondientes al momento en el que se actualiza el grafo. De esta forma no se hace tan abrumador y se crea un grupo de comandos relacionados, para cada pantalla de flujo generada.
5. Tras la pantalla de flujo, podrá aparecer o no, en función de si se necesita, otra vez alguna pantalla de las mencionadas anteriormente.

Este orden se da para cada nivel, pero hay que destacar que exceptuando la primera pantalla de introducción, que inicia cada nivel, el ciclo formado por las fases 2 a 5 se puede repetir durante cada nivel las veces que haga falta, y omitiendo aquellas pantallas no necesarias.

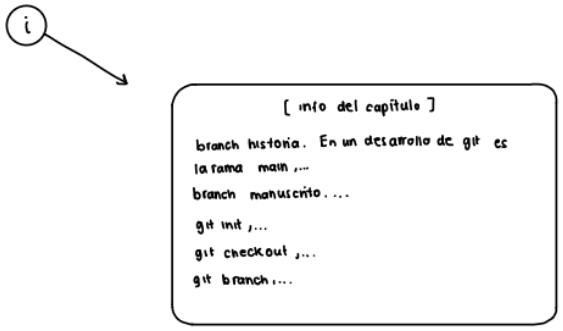


Figura 2.4: Pantalla de información

2.5. Historia

Este juego se está enfocando en seguir el modelo de Gitflow y para ello, se ha querido ajustar al flujo que seguiría una historia. El mundo en el que se desarrolla va a ser el reino "Sunnylane" y sus proximidades. Se va a contar con distintos escenarios por los que pasará el personaje principal. La historia del personaje no va a ser muy complicada ya que lo que se quiere es visualizar de forma clara Gitflow y es preferible no llenar la historia con muchos detalles.

El personaje principal, "Caballero", iniciará su aventura en su casita en el bosque. Dentro, se vestirá para comenzar la jornada de la mañana. Deberá elegir qué ropa ponerse y a continuación saldrá a trabajar.

Pasada la mañana, quedará con "Doncella", la cual se estará preparando para formar parte de los arqueros del reino. En este momento de la tarde "Caballero" le dirá a "Doncella" qué ha estado haciendo para trabajar, por lo que escogerá una de las opciones disponibles.

Mientras tanto, en el castillo, "Rey" habrá pasado toda la tarde buscando su corona ya que no la encuentra.

Cae la noche y ambos se van a dormir a sus casas.

Al día siguiente "Rey" convoca una reunión para cubrir su plantilla de Caballeros reales. El Caballero y la Doncella indican si quieren ser Caballeros o Arqueros.

En este punto la corona del Rey no se encuentra y convocan a todos los personajes a buscarla urgentemente. " Caballero" y "Doncella" la buscan teniendo en cuenta que su elección anterior no se ha vuelto oficial ya que ha surgido un imprevisto.

Ambos, "Doncella" y "Caballero" parten en la búsqueda de la corona, encontrándola en la caseta de "Mascota real", el cual la había tomado como juguete.

Hacen oficial la elección de ambos personajes y "Rey" tras obtener de nuevo la corona nombra a "Caballero" "Caballero real" y a "Doncella" "Arquera real". Ambos felices van en busca de nuevas aventuras.

2.6. Personajes

Varios personajes van a aparecer en la historia, como se ha mencionado anteriormente en la breve descripción de la misma. A continuación se da una breve descripción de los personajes, que pese a no tener mucho impacto es positivo tener una visión de la apariencia de los personajes.

Caballero Es el personaje principal y el personaje que va a "dirigir" el usuario. Tiene 18 años y forma parte de los caballeros del bosque. Es moreno de ojos verdes y tiene una complexión atlética. Es amable pero a veces es demasiado bueno. Su color favorito es el amarillo.

Doncella Personaje secundario que acompaña a "Caballero" durante su travesía. Tiene 18 años y se está entrenando para formar parte de los Arqueros del reino. Es pelirroja de ojos verdes y es muy ágil. Es más perspicaz que "Caballero" y siempre le está dando a su amigo otra perspectiva cuando surge algún problema.

Rey Personaje necesario a la hora de crear el conflicto principal de la historia. Tiene 50 años y mantiene unido al reino gracias a la extensa red de caballeros entrenados para proteger sus tierras. Es justo con las personas que merecen su confianza y siempre intenta mejorar el reino.

Mascota Es el culpable del problema que surge. No tiene mucha aparición en la historia.

2.7. Niveles

Nivel 0. El comienzo de todo.

Este nivel está dedicado a la explicación de las ramas **main y develop**. En primer lugar se le dará al usuario un contexto de lo que va a ser el "juego" y cuál es su misión. Tras esto, primero se le explicará qué es la rama "historia" y cuál es la analogía en un desarrollo con Git aplicando Gitflow. Se le explicará qué es, qué comandos se emplean y, tras esto, se le dará al usuario una línea de comandos para que introduzca el comando adecuado. Tras la creación de esa rama, se creará otra de la misma forma, siendo el "manuscrito" y correspondiéndose a la rama develop. El grafo de flujo que tendrá el nivel al finalizar el nivel será el que se muestra en la Figura 2.5.

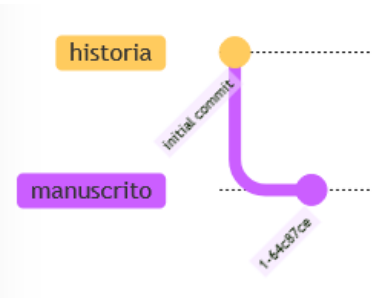


Figura 2.5: Final del nivel 0

Nivel 1. Vuelta a la rutina.

En este nivel se explicará la rama **feature**. para ello se le explicará al usuario la analogía entre features y capítulos y como funciona dentro de Gitflow. Primero habrá una pantalla para introducir el nuevo nivel y, tras esto, se explicará la rama capítulo y su correspondiente nombre en un desarrollo real. Al igual que en el primer nivel, el usuario creará una rama "capítulo1" para poder empezar a desarrollarlo. A continuación se le va dar al usuario una serie de opciones, que se asemejarán a los cambios que se vayan haciendo en los diferentes ficheros. Tras finalizar esa pantalla, el usuario subirá esos cambios y podrá ver los cambios realizados por los demás personajes. Como última parte del nivel se hablará del merge y se hará ese merge del "capítulo1" a "manuscrito". Esto se puede ver reflejado en la Figura 2.6, que muestra el grafo al finalizar el nivel.

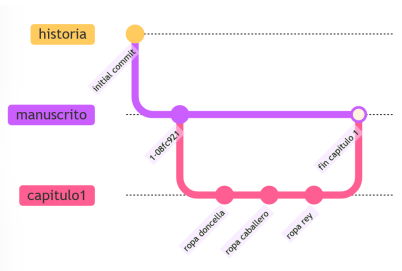


Figura 2.6: Final del nivel 1

Nivel 2. Siguiendo una rutina.

Este nivel muestra lo mismo que el Nivel 1, por lo que será un repaso de esa rama feature. No se va a explicar nada nuevo. La actualización del grafo durante este nivel se encuentra en la Figura 2.7.

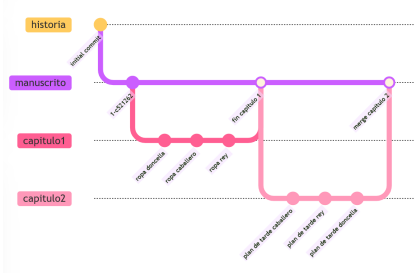


Figura 2.7: Final del nivel 2

Nivel 3. El gran evento.

En este nivel nos vamos a centrar en explicar la rama **release** y en el ciclo de publicación. Se le dará al usuario un contexto de lo que ocurre dentro del mundo en el que se encuentra y luego una explicación relacionada con la rama release dentro de un proyecto real así como lo que es el ciclo de publicación. Tras eso se creará la respectiva rama "checkpoint" y se hará su merge a la rama "historia" etiquetando ese commit con el número de versión. Todo estos cambios se reflejan en la Figura 2.8, que muestra el progreso realizado hasta este nivel.

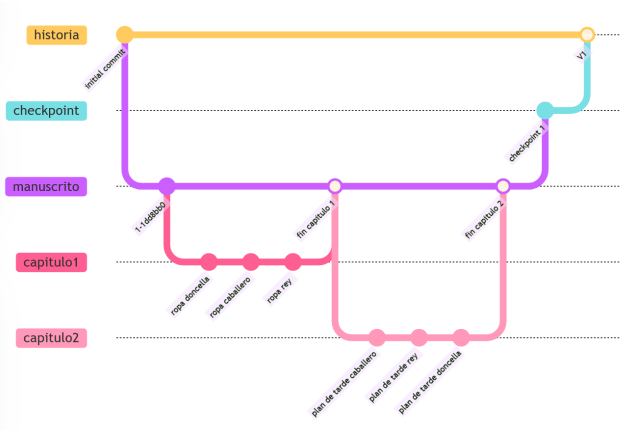


Figura 2.8: Final del nivel 3

Nivel 4. Problemas inesperados.

En este nivel **no se va a explicar ninguna rama** ya que va a servir de puente para el siguiente nivel en el que sí se le explicará al usuario una rama. Este nivel, al igual que el Nivel 1 y el Nivel 2 se centrará en la creación de una nueva rama feature, "capitulo3" de forma que se le introduzca en la historia el problema que va a surgir, y se le da al usuario las opciones disponibles para que pueda elegir. Este nivel termina abruptamente ya que se

quiere dejar el nivel a medias para poder arreglar el conflicto –que se creará a futuro,– entre esta rama y la corrección que se va a dar en el siguiente nivel. El resultado de este nivel está plasmado en la Figura 2.9.

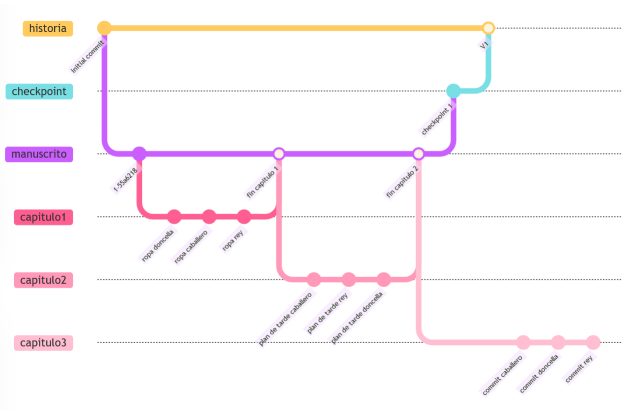


Figura 2.9: Final del nivel 4

Nivel 5. Resolviendo ciclos.

Se va a explicar la rama **hotfix**. Para ello de nuevo hay una introducción de qué pasa dentro del mundo y a continuación la explicación de esa rama. Tras ello, el usuario creará la rama "correcciones" y se le mostrará al usuario una pantalla para que elija una acción, simulando el cambio a realizar. Tras ello, se hace merge a la rama "historia" y a "manuscrito" para que se vean reflejados esos cambios en ambas ramas existentes. Esto se puede ver en la Figura 2.10, que muestra el progreso del flujo tras corregir el error que ha surgido.

Como se estaba realizando el "capítulo3", habrá que resolver conflictos, por lo que tras finalizar la rama "correcciones", se corregirán esos errores en la rama "capítulo3" y tras acabar con esa rama se subirá a develop. Acabará la historia dándole un final al personaje. El estado del diagrama de flujo al finalizar la historia se puede ver en la siguiente Figura 2.11.

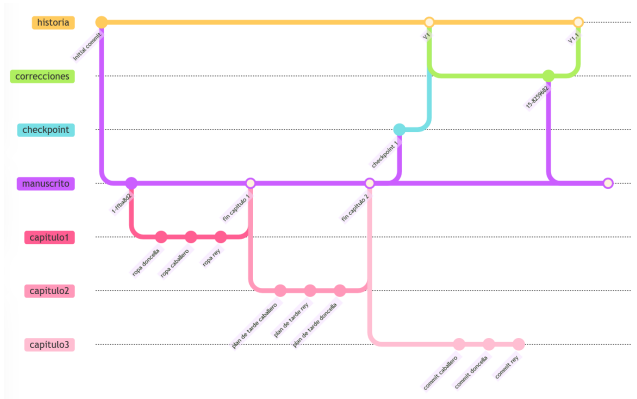


Figura 2.10: Mitad del nivel 5

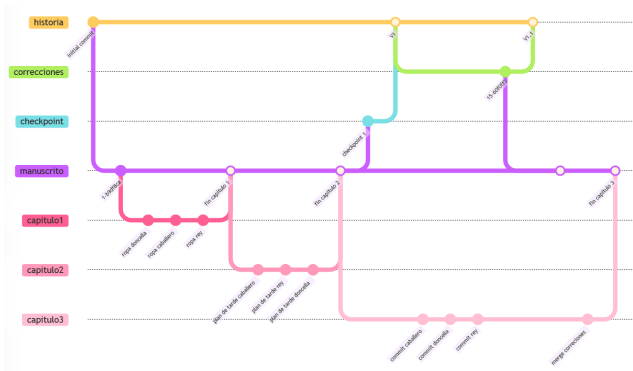


Figura 2.11: Fin del nivel 5

2.8. Logros y objetivos

Uno de los puntos básicos de la gamificación es la implementación de chapas, rankings, puntos,... con el fin de mantener al usuario motivado. Por eso, dentro del juego se podrán obtener distintos logros o chapas durante el transcurso de los capítulos.

Se ha querido enfocar de forma que existan dos tipos de logros.

Logros obligatorios Son logros que se obtienen a medida que se avanza en la historia solo por avanzar.

Logros posibles Son logros que se pueden o no obtener en función de las decisiones que hayas elegido durante los niveles.

En la Tabla 2.1 se resumen los logros que se han definido para este juego.

Número	Descripción
Logros obligatorios (LO)	
LO1: Finalización del Nivel 0	Has finalizado el nivel 0. Hurra! Has empezado una nueva historia.
LO2: Finalización del Nivel 1	Has finalizado el nivel 1. Caballero vestido, caballero feliz.
LO3: Finalización del Nivel 2	Has finalizado el nivel 2. El descanso es el mejor plan de tarde.
LO4: Finalización del Nivel 3	Has finalizado el nivel 3. ¿Dónde está el checkpoint?
LO5: Finalización del Nivel 4	Has finalizado el nivel 4. Turbulencias en el castillo.
LO6: Finalización del Nivel 5	Has terminado el nivel final. Como perros y gatos.
Logros posibles (LP)	
LP1: Seleccionar la ropa del caballero correctamente	Combinando la ropa. Has elegido un color de ropa que le gusta al caballero.
LP2: Seleccionar recoger manzanas al ir a trabajar	Blancanieves. Recogiste manzanas.
LP3: Seleccionar utilizar la espada en la prueba que convoca el rey.	Amigos antes que reyes. Escogiste no enfrentaros.
LP4: Se consigue cuando se elimina alguna rama que no hace falta cuando se une a alguna rama.	Escoba y plumero. Has borrado alguna línea que ya no hacía falta.
LP5: Se consigue cuando se obtiene el 100 % de los logros anteriores	Main character. ¡Se han obtenido todos los logros disponibles!

Tabla 2.1: Lista de logros obtenibles

2.9. Audiencia

Ya que el juego pretende mostrar un nuevo método de trabajo para el control de versiones de un proyecto, la audiencia serían personas que no conozcan este método o tengan dificultad para aprenderlo. Como este método de aprendizaje está relacionado con los juegos, aquellas personas que les guste algo gamificado y estén relacionados con los videojuegos, les resultará más atractivo que a personas que a priori no tengan interés en ellos. Por eso, y ya que los videojuegos están más dedicados a un rango de edad en concreto, esta aplicación está destinada a personas jóvenes.

Se van a emplear comandos básicos, por lo que para personas que no conozcan Git no les va a resultar ningún problema entender el juego, probablemente recién llegados a un grado medio/superior o universitario relacionado con el desarrollo de software.

2.10. Plataformas

La plataforma que se va a emplear va a ser una página web empleando Angular como framework para desarrollarla. Normalmente Angular está pensado para realizar aplicaciones web y no juegos. Sin embargo, en esta ocasión se ha querido utilizar este framework, Angular, como medio para desarrollar un juego por una serie de razones que se describen a continuación:

Portabilidad La web es algo accesible para todo el mundo ya que cualquier navegador puede acceder a ella. Esto es positivo puesto que el juego pasa a estar disponible para aquellos dispositivos que dispongan de un navegador.

Extensibilidad Como Angular está basado en la conjunción de componentes se podrá generar nuevos niveles sin necesidad de cambiar el código existente y pudiendo reutilizar muchos de los componentes que ya hay desarrollados.

Opciones de ampliación Al elegir la página web, en un futuro se pueden implementar nuevas funcionalidades. Además, al ya tener una organización estructural se puede hacer más complejo añadiendo funcionalidades como rankings, puntos, etc.

Datos en servidor Al emplear esta tecnología, se puede hacer uso de peticiones http para guardar en el servidor el estado del jugador y poder guardar el progreso entre dispositivos.

Multijugador A futuro se puede implementar la funcionalidad de trabajar en equipo en la que cada jugador puede aportar a la misma sala de juego.

Elección propia Puesto que es un proyecto personal, y ya conocía esta tecnología, quería ahondar un poco más para conocer cómo sería desarrollar un juego en este tipo de tecnología, sirviéndome como un proyecto que tiene intrínsecamente algo de formación.

Capítulo 3

Requisitos y Planificación

3.1. SCRUM

3.1.1. ¿Qué es Scrum?

Scrum, llamado así como analogía a Scrum en rugby, *all pushing together*, es un sistema de organización de proyectos de metodología ágil que originalmente se diseña para proyectos donde el tiempo que transcurre desde que se concibe un producto hasta que se lanza al mercado, *time to market*, es importante.

Scrum es una implementación del Agile Manifesto, 12 principios que se pueden resumir en estas afirmaciones [7]:

- Las personas y las interacciones por encima de los procesos y las herramientas
- Software funcionando sobre documentación extensiva
- Colaboración con el cliente por encima de negociación de contratos
- Dar respuesta a un cambio por encima del seguimiento de un plan

Partiendo de este manifiesto, Scrum es un sistema que se basa en el aprendizaje continuo y en la adaptación a los factores fluctuantes a medida que se avanza en el proyecto. De esta forma un equipo reducido desarrolla, entrega y mantiene el producto incrementalmente y en periodos cortos y fijos que se han establecido previamente, teniendo en cuenta los posibles riesgos que puedan surgir y haciendo frente a ellos adaptándose lo más posible a los propios miembros del equipo.

El marco de Scrum está formado por una serie de prácticas, principios que los equipos siguen para desarrollar un producto. A continuación se van a detallar los roles que existen,

los "artefactos" o elementos que se producen como resultado de aplicar este sistema y los protocolos y reuniones que se realizan durante todo el proceso.

3.1.2. Roles

Los roles en Scrum son necesarios ya que asignan a cada miembro del equipo una serie de responsabilidades y funciones necesarias para la interacción entre ellos y entre los artefactos [20].

- **Product Owner**. En los equipos de Scrum existe un único *Product Owner*, y es el responsable de maximizar y optimizar el producto o servicio que se entrega. Además, se encarga de establecer el contacto entre los *stakeholders* y el proyecto.
- **Scrum Master**. Es responsable de construir equipos Scrum facilitando las decisiones del equipo de desarrollo. Una de las formas en las que el *Scrum Master* puede ayudar al equipo de desarrollo a mejorar y a reducir los riesgos e impedimentos en el desarrollo del proyecto, es enseñando y entrenando al equipo para que ellos mismos puedan eliminarlos. Este rol se debe al equipo de desarrollo. Por eso, además, se encarga de las labores de formación y de facilitar reuniones y eventos si es necesario.
- **Development Team**. Normalmente formado por 3 a 5 personas. Son los encargados de desarrollar un producto de software "liberable" que se ajuste a la definición de 'Terminado", *Definition of Done*– al finalizar cada Sprint (ver Eventos más adelante). Entonces, el equipo atiende al *Product Owner*.

3.1.3. Artefactos

Los artefactos de Scrum son elementos que aportan información a la hora de que el equipo pueda definir el producto y el trabajo que hay que hacer para crearlo. Es decir, son elementos que representan el valor o el trabajo del equipo y están diseñados para maximizar la transparencia de la información clave [55].

Cada artefacto contiene el compromiso de garantizar que su información proporcione transparencia y un enfoque con el que se pueda medir el progreso. A continuación se explica de forma más detallada los distintos tipos de artefactos que se dan en Scrum [38]:

- **Product backlog**. Es todo lo que sabemos que el producto necesita. Es una lista ordenada de todo lo que sabemos que vamos a necesitar para entregar el producto y las tareas en las que el equipo trabajará próximamente. El Product Owner es quien se encarga del contenido y el orden de estos elementos y son los desarrolladores quienes se encargan de dimensionarlos.
- **Sprint backlog**. Es el plan llevado a cabo para poder entregar los elementos del Backlog de producto seleccionados en el Sprint. El equipo de desarrollo crea este Backlog de

sprint durante la reunión de Planificación de sprint, que contiene una lista con elementos del Backlog del producto que han seleccionado para desarrollar durante el Sprint. Los desarrolladores refuerzan su autonomía abordando el trabajo del Sprint.

- **Increment**. Los pasos a seguir para obtener el producto final consisten en entregar un aumento de producto valioso utilizable en cada Sprint. Es decir, el equipo debe pensar no solo en el objetivo final, sino también en entregar incrementos valiosos a lo largo del proceso produciendo funcionalidades en cada Sprint. Gracias a esto, los *stakeholders* siempre pueden ver en qué está trabajando el equipo llevando a una buena comunicación y entendimiento.

3.1.4. Eventos

Los eventos son reuniones que se introducen con el fin de facilitar el trabajo entre los equipos, los artefactos y la organización de los miembros. Son importantes por la forma en la que están enmarcadas dentro de un límite de tiempo y por la razón concreta por la que existen [14].

Es importante hablar sobre el límite de tiempo dedicado a una actividad ya que para los eventos Scrum es necesario que no se exceda ese límite. De esta forma, el tiempo no se malgasta y trata de enfocarse en lo realmente necesario.

- **Sprint**. El objetivo es convertir las ideas en valor. Los sprints duran entre 1 a 4 semanas y dentro de los eventos en Scrum, este es especial ya que es una especie de Evento Contenedor, en el cual se alojarán los siguientes eventos definidos. No hay espacios entre sprint y sprint y todas las metas y objetivos necesarios para lograr el producto final se desarrollan dentro de estos. Una característica de Scrum es que una vez decidida la duración de cada Sprint, no cambia desde el inicio hasta el final del proyecto.
- **Sprint planning**. Este evento existe para poder planificar el trabajo que se va a realizar en el Sprint. Por ende, es el primer evento existente dentro de un Sprint y a este asisten todos los miembros del equipo de Scrum.
- **Daily scrum**. Su objetivo principal es la de comprobar el progreso que se va haciendo en función del objetivo del Sprint y adaptar el *Sprint Backlog* si es necesario para próximos sprints. Estas reuniones son rápidas y no duran más de 15 minutos.
- **Sprint review**. Esta reunión se da para exponer el resultado del Sprint y adaptar futuros sprints en función de este. A esta reunión asiste todo el equipo Scrum junto con los *stakeholders*, o el *Product Owner* como representante de los *stakeholders*. Así, estos pueden aportar una retroalimentación al equipo sobre el trabajo realizado en ese Sprint.
- **Sprint retrospective**. En estas reuniones se pretende planear formas de mejorar la calidad y la efectividad. Es decir, el equipo busca maneras de mejorar y de colaborar para la mejorar del trabajo. Ya que es algo intrínseco al equipo, a este asistirá únicamente el equipo Scrum.

3.2. Adaptación de Scrum al proyecto

Tras lo visto en la sección anterior, para poder aplicar Scrum a este proyecto es necesario realizar una serie de cambios para adaptar lo mejor posible este sistema dentro de un proyecto en el que hay únicamente dos personas implicadas, la estudiante y la tutora.

Por ello, se va a asignar a la estudiante el rol de Equipo de desarrollo aunque estos suelen estar formados de 3 a 5 personas y la tutora, por su parte, desempeñará el rol de *Product Owner* y *Scrum Master*. De forma el profesor será quién organice las diversas reuniones así como será el encargado de revisar los entregables realizados y la estudiante será quién se encargue de entregar el incremento durante los sprints.

En cuanto a los artefactos existentes, no se aplicarán cambios a los definidos en Scrum ya que estos no dependen del número de personas participantes sino del proyecto o servicio.

Los eventos dentro de este sistema de trabajo son importantes pero, que ajustados a un equipo de dos personas pueden tener una duración excesiva. Por eso, se han modificado de forma que encajen dentro de las dos reuniones definidas por Sprint.

Como se ha comentado anteriormente, se va a establecer dos reuniones por cada Sprint y corresponderán a una reunión por semana. La primera reunión se dedicaría al *Sprint review* y *Sprint retrospective*, y el *Sprint planning* si hubiera el siguiente sprint y la segunda para hacer un seguimiento del incremento en el sprint. Ya que el equipo de desarrollo está formado por la estudiante, la *Daily scrum* se ha eliminado dentro de esta adaptación de Scrum.

Partiendo de que el TFG cuenta con una duración total de 300h, de las cuales se le han restado las horas dedicadas al Sprint 0 (ver Capítulo 8), se ha optado por una duración de 2 semanas por sprint, 35 horas por sprint y por tanto, entre 17h y 18h dedicadas aproximadamente cada semana.

Una vez definida la duración de cada sprint, falta definir cómo se van a puntuar las historias de usuario con el fin de visualizar la cantidad de esfuerzo y horas de trabajo necesarias en cada sprint.

En el marco de Scrum, en general, los equipos suelen utilizar escalas no lineales, como el uso de la sucesión de Fibonacci, de esta forma se ahorran problemas a la hora de colocar un valor u otro ya que, si la distancia entre valores es la misma, puede haber diferencias entre los miembros del equipo. Por otra parte, los intervalos entre los números de esta sucesión son cada vez más amplios y, como resultado, la estimación se vuelve menos precisa a medida que aumenta el valor, lo que refleja que, cuanto más grande sea un proyecto o tarea a realizar, mayor será la incertidumbre y menor será la precisión. Por eso, es importante establecer un límite en la sucesión, como tope de la escala de estimación.

También se debe tener en cuenta que una historia de usuario debe poder ser terminada en un sprint. Por eso, el límite debería ser tal que el esfuerzo para completar la historia más compleja esté enmarcado en lo que se puede realizar en un sprint. En caso de que se perciba que no será así, deben descomponerse las historias de usuario en otras más simples. En la Tabla 3.1 se resumen las escalas más utilizadas para asignar puntos de historia.

Escalas más utilizadas	
Escala lineal	Una serie de números que difieren en la misma cantidad y que estarán separados por la misma distancia.
Sucesión de Fibonacci	Una serie de números cuya suma de los dos últimos dará como resultado el siguiente número y así sucesivamente.
Escala exponencial	Una serie de números cuyo valor es dos veces el número que le precede.

Tabla 3.1: Escalas para asignar los puntos de historia [24].

En este proyecto, se ha optado por emplear la escala de la sucesión de Fibonacci, estableciendo unos límites para asignar los puntos de historia. En principio, el valor mínimo que podrán adquirir las historias de usuario será 1, aunque si en el futuro el equipo de desarrollo quiere velocidad, se considerará medio punto de historia. El máximo será el 5, por lo que los valores posibles serán 0.5, 1, 2, 3 y 5.

Estos puntos pierden su importancia si no se les aporta un valor medible, como lo son las horas de trabajo empleadas, por lo que se le asignará un número de horas a cada punto de historia. Es este tipo de proyectos, por las características del equipo de trabajo (una sola persona, dedicación parcial al proyecto, etc.), el **valor del punto de historia** son 5 horas, por lo que se le asignará esas horas por punto. De esta forma, teniendo en cuenta que los sprints duran 35h, todas por separado pueden abarcarse en un sprint y a mayores, habría que decidir qué historias, u otras tareas colaterales, hacer para poder encajarlas dentro de cada sprint.

En la Tabla 3.2 se refleja el valor que se le da a cada punto de historia de usuario y su peso en horas y complejidad. Así se tendrán en cuenta además de las horas necesarias para realizarlas, otros factores como la complejidad o el esfuerzo que implica la historia.

Matriz de puntos		
Punto de historia	Esfuerzo	Complejidad de la tarea
1	Muy bajo	Muy baja
1	Bajo	Baja
2	Bajo	Media
3	Medio	Alta
5	Alto	Alta

Tabla 3.2: Matriz de puntos de historia

3.3. Planificación inicial del proyecto

Como se ha comentado en la sección anterior, se van a dedicar 35h por Sprint repartiendo esas horas de la forma más conveniente. Así, repartiendo uniformemente las horas, se llega a realizar **2 horas y 30 minutos por día**, un número asequible teniendo en cuenta que se está trabajando paralelamente a la realización de este proyecto.

Quitando el tiempo dedicado al Sprint 0, que son unas 50 horas, se cuenta con **7 u 8 sprints** para llegar a las horas requeridas por la asignatura.

Además, para poder solventar los posibles riesgos que pueden ocurrir, se han añadido dos sprints de refuerzo que se emplearán para completar tareas que no hayan podido realizarse en los sprints a las que pertenecían, como también posibles detalles enfocados a la finalización del proyecto y documentación a entregar.

Si el proyecto sigue la planificación inicial, tendrá una duración de entre 300 y 335 horas aproximadamente, ya que al hacer la conversión de horas a sprints no encaja exactamente con un sprint completo. La idea sería que en el último sprint se añadiesen las horas necesarias para poder llegar a las 300 horas sin necesidad de alargar el proyecto otro sprint más.

Si se deben realizar uno o dos sprints de refuerzo, el total de horas sería entre 335 horas y 370 horas, respectivamente. Sin embargo, ya que esto es la planificación inicial, durante todo el capítulo de seguimiento, explicado en el Capítulo 8, se reflejará más detalladamente los cambios que se hayan ido produciendo durante el proyecto en cuanto a planificación de sprints.

En la siguiente Tabla 3.3 se refleja lo dicho en estos párrafos organizando cada sprint con sus fechas.

Sprint	Inicio	Fin	Comentarios
Sprint 0	28/08/2023	03/11/2023	
Scrum Semanal	28/08/2023 04/09/2023 22/09/2023 06/10/2023 13/10/2023 27/10/2023		Se hicieron varias reuniones a lo largo del sprint para hacer un seguimiento desde el principio sobre el tema del TFG, planificación, contexto,...
Sprint Review, Sprint Retrospective y Sprint Planning	03/11/2023		
Sprint 1	06/11/2023	19/11/2023	
Scrum Semanal	10/11/2023		
Sprint Review, Sprint Retrospective y Sprint Planning	17/11/2023		
Sprint 2	20/11/2023	03/12/2023	
Scrum Semanal	24/11/2023		
Sprint Review, Sprint Retrospective y Sprint Planning	01/12/2023		
Sprint 3	04/12/2023	17/12/2023	
Scrum Semanal	08/12/2023		
Sprint Review, Sprint Retrospective y Sprint Planning	15/12/2023		
Sprint 4	18/12/2023	31/12/2023	
Scrum Semanal	22/12/2023		
Sprint Review, Sprint Retrospective y Sprint Planning	29/12/2023		
Sprint 5	01/01/2024	14/01/2024	
Scrum Semanal	05/01/2024		
Sprint Review, Sprint Retrospective y Sprint Planning	12/01/2024		
Sprint 6	15/01/2024	28/01/2024	
Scrum Semanal	19/02/2024		
Sprint Review, Sprint Retrospective y Sprint Planning	26/01/2024		
Sprint 7	29/01/2024	11/02/2024	
Scrum Semanal	02/02/2024		
Sprint Review, Sprint Retrospective y Sprint Planning	09/02/2024		En caso de no realizarse el primer Sprint de refuerzo, el Sprint Planning no se realizará.
Sprint Refuerzo 1	12/02/2024	25/02/2024	
Scrum Semanal	16/02/2024		
Sprint Review, Sprint Retrospective y Sprint Planning	23/02/2024		En caso de no realizarse el segundo Sprint de refuerzo, el Sprint Planning no se realizará.
Sprint Refuerzo 2	26/02/2024	10/03/2024	
Scrum Semanal	01/03/2024		
Sprint Review, Sprint Retrospective y Sprint Planning	08/03/2024		

Tabla 3.3: Calendarización del proyecto

3.4. Backlog inicial

En esta sección quedan documentadas las distintas épicas y correspondientes historias de usuario que se han descrito inicialmente. Este *Product Backlog* refleja de forma muy general las épicas de usuario que se van a querer desarrollar. Ya que el nivel de detalle es relativamente bajo, las épicas son de grano grueso y conforme se vayan refinando más durante cada Sprint, se podrá reflejar de forma más concisa su trabajo y esfuerzo.

Número	**Épica**
E1	Como Jugador quiero disponer de infamación sobre el juego, documentación, ayuda y mis logros para saber como jugar y poder relacionar los conceptos del juego con el aprendizaje.
E2	Como Jugador quiero seleccionar y cargar el Nivel 0 para poder jugar y completar el Nivel 0
E3	Como Jugador quiero seleccionar y cargar el Nivel 1 para poder jugar y completar el Nivel 1
E4	Como Jugador quiero seleccionar y cargar el Nivel 2 para poder jugar y completar el Nivel 2
E5	Como Jugador quiero seleccionar y cargar el Nivel 3 para poder jugar y completar el Nivel 3
E6	Como Jugador quiero seleccionar y cargar el Nivel 4 para poder jugar y completar el Nivel 4
E7	Como Jugador quiero seleccionar y cargar el Nivel 5 para poder jugar y completar el Nivel 5

Tabla 3.4: Épicas de usuario

Durante el Sprint 0, visto más detalladamente en la Sección 8.1.1, se ha estado desarrollando todo el Documento de Diseño de juego, explicado en el Capítulo 2, y por tanto se ha podido ahondar más en los requisitos que debe cumplir el sistema. Por eso, a continuación se va a dar una lista un poco más detallada sobre lo que abarcaría cada épica, teniendo en cuenta que esta lista no será la versión final del *Product Backlog* y estará sujeto a cambios durante la ejecución de los sprints.

Como la estructura de cada nivel va a ser similar, para cada épica relacionada con la selección de niveles se tendrá una tabla similar a esta, teniendo en cuenta que aquellos niveles que no tengan la escena que se desarrolla en esa historia de usuario, no tendrán esa historia. La Tabla 3.5 muestra una generalización de historias de usuario a tener en cuenta por cada nivel.

Historia de usuario para cada nivel
Como Jugador quiero poder consultar la introducción (definido en la Sección 2.4) del Nivel X para poder leer información relativa a la historia.
Como Jugador quiero poder consultar el contexto (definido en la Sección 2.4) para poder conocer la información asociada al Nivel X.
Como Jugador quiero poder consultar a las opciones disponibles (definido en la Sección 2.4) para seleccionar la opción que quiera.
Como Jugador quiero acceder al flujo (definido en la Sección 2.4) del nivel para poder introducir los datos necesarios y lograr el objetivo del Nivel X.
Como Jugador quiero acceder a la pantalla de información (definido en la Sección 2.4) para conocer los datos técnicos necesarios para el Nivel X.
Como Jugador quiero poder avanzar a la escena siguiente cuando he finalizado la escena que estoy jugando para poder progresar en el juego.

Tabla 3.5: Historias de usuario generalizadas para cada nivel

Para el resto de características de la aplicación, relacionadas con toda la información externa a los niveles y con mostrar las distintas opciones disponibles, se define otro conjunto de historias de usuario definidas que se muestran en la Tabla 3.6.

E1: Información externa a los niveles	
Número	**Historia de usuario**
HU61	Como Jugador quiero acceder al menú para ver la lista de niveles existentes.
HU62	Como Jugador quiero salir del juego y guardar el progreso conseguido hasta el momento para poder volver en otra ocasión y no haber perdido lo realizado.
HU62	Como Jugador quiero consultar la información del juego para conocer todos los datos técnicos que se van a utilizar en todos los niveles.
HU64	Como Jugador quiero poder consultar todos los logros que podría conseguir en el juego para poder realizar mi plan de conseguirlos.
HU65	Como Jugador quiero comprobar los logros que he completado para conocer mi progreso en la historia.
HU66	Como Jugador quiero tener una interfaz en sintonía con la idea del juego para obtener información de manera visual sobre lo que quiere contar el juego.

Tabla 3.6: Historias de usuario para información externa a los niveles

3.5. Product backlog más detallado

Tras el desglose de grano grueso, presentado en la sección anterior, se van a definir historias más concretas o redefinir las que se han definido anteriormente relacionadas con los niveles. Las Tablas 3.7, 3.8 y 3.9 muestran el desglose más detallado de las historias de usuario correspondientes a los diferentes niveles de juego.

Número	Historia de usuario
E2: Nivel 0	
HU01	Como Jugador quiero poder consultar la introducción del Nivel 0 para poder leer información relativa a la historia.
HU02	Como Jugador quiero poder ver el contexto del Nivel para poder entender la rama main y develop que se está explicando.
HU03	Como Jugador quiero poder interactuar con el contexto del Nivel 0 para poder leer el contexto asociado al nivel y poder pasar a la siguiente escena.
HU04	Como Jugador quiero desarrollar los elementos necesarios utilizados en las escenas de opción para poder utilizarlas en niveles posteriores.
HU05	Como Jugador quiero acceder al flujo del nivel 0 para poder introducir los datos necesarios y lograr el objetivo del Nivel 0.
HU06	Como Jugador quiero guardar el flujo del nivel 0 para poder volver hacia atrás durante el nivel y volver a introducir los datos necesarios para lograr el objetivo del Nivel 0.
HU07	Como Jugador quiero tener alternativas en el flujo del nivel 0 para poder introducir una alternativa a los datos necesarios para lograr el objetivo del Nivel 0.
HU08	Como Jugador quiero acceder a la pantalla de información para conocer los datos técnicos necesarios para el Nivel 0.
HU09	Como Jugador quiero poder avanzar a la escena siguiente cuando he finalizado la escena que estoy jugando para poder progresar en el nivel 0 y en el juego.
HU10	Como Jugador quiero poder avanzar al nivel 1 cuando he finalizado la última escena del nivel que estoy jugando para poder progresar en el juego.
HU11	Como Jugador quiero poder guardar los logros que he completado en el Nivel 0 para poder progresar en el juego y verlo reflejado en la pantalla de resumen y en la pantalla de logros.
E3: Nivel 1	
HU12	Como Jugador quiero poder consultar la introducción del Nivel 1 para poder leer información relativa a la historia.
HU13	Como Jugador quiero consultar el contexto del Nivel 1 para entender la rama feature y su importancia en un desarrollo para poder conocer los datos necesarios y avanzar a la siguiente escena.
HU14	Como Jugador quiero poder ver las opciones disponibles durante la escena de opción para leer las opciones disponibles.
HU15	Como Jugador quiero interactuar con las opciones disponibles durante la escena de opción para seleccionar la opción que yo quiera y continuar avanzando.
HU16	Como Jugador quiero acceder al flujo del nivel 1 para poder introducir los datos necesarios y lograr el objetivo del Nivel 1.
HU17	Como Jugador quiero acceder a la pantalla de información para conocer los datos técnicos necesarios para el Nivel 1.
HU18	Como Jugador quiero poder avanzar a la escena siguiente cuando he finalizado la escena que estoy jugando para poder progresar en el nivel 1 y en el juego.
HU19	Como Jugador quiero poder avanzar al nivel 2 cuando he finalizado la última escena del nivel que estoy jugando para poder progresar en el juego.
HU20	Como Jugador quiero poder guardar los logros que he completado en el Nivel 1 para poder progresar en el juego y verlo reflejado en la pantalla de resumen y en la pantalla de logros.

Tabla 3.7: Historias de usuario detalladas para los niveles 0 y 1

Número	Historia de usuario
E4: Nivel 2	
HU21	Como Jugador quiero poder consultar la introducción del Nivel 2 para poder leer información relativa a la historia.
HU22	Como Jugador quiero consultar el contexto del Nivel 2 para afianzar lo aprendido en el Nivel 1 sobre la rama feature y su importancia en un desarrollo para poder conocer los datos necesarios y avanzar a la siguiente escena.
HU23	Como Jugador quiero poder ver y seleccionar una de las opciones disponibles durante la escena de opción para leer las opciones disponibles y poder continuar con el Nivel 2.
HU24	Como Jugador quiero acceder al flujo del nivel 2 para poder introducir los datos necesarios y lograr el objetivo del Nivel 2.
HU25	Como Jugador quiero acceder a la pantalla de información para conocer los datos técnicos necesarios para el Nivel 2.
HU26	Como Jugador quiero poder avanzar a la escena siguiente cuando he finalizado la escena que estoy jugando para poder progresar en el nivel 2 y en el juego.
HU27	Como Jugador quiero poder avanzar al nivel 3 cuando he finalizado la última escena del nivel que estoy jugando para poder progresar en el juego.
HU28	Como Jugador quiero poder guardar los logros que he completado en el Nivel 2 para poder progresar en el juego y verlo reflejado en la pantalla de resumen y en la pantalla de logros.
E5: Nivel 3	
HU29	Como Jugador quiero poder consultar la introducción del Nivel 3 para poder leer información relativa a la historia.
HU30	Como Jugador quiero consultar el contexto del Nivel 3 para entender la rama release y su importancia en un desarrollo para poder conocer los datos necesarios y avanzar a la siguiente escena.
HU31	Como Jugador quiero acceder al flujo del Nivel 3 para poder introducir los datos necesarios y lograr el objetivo del Nivel 3.
HU32	Como Jugador quiero acceder a la pantalla de información para conocer los datos técnicos necesarios para el Nivel 3.
HU33	Como Jugador quiero poder avanzar a la escena siguiente cuando he finalizado la escena que estoy jugando para poder progresar en el Nivel 3 y en el juego.
HU34	Como Jugador quiero poder avanzar al Nivel 4 cuando he finalizado la última escena del nivel que estoy jugando para poder progresar en el juego.
HU35	Como Jugador quiero poder guardar los logros que he completado en el Nivel 3 para poder progresar en el juego y verlo reflejado en la pantalla de resumen y en la pantalla de logros.

Tabla 3.8: Historias de usuario detalladas para los niveles 2 y 3

Número	Historia de usuario
E6: Nivel 4	
HU36	Como Jugador quiero poder consultar la introducción del Nivel 4 para poder leer información relativa a la historia.
HU37	Como Jugador quiero poder ver y seleccionar una de las opciones disponibles durante la escena de opción para leer las opciones disponibles y poder continuar con el Nivel 2.
HU38	Como Jugador quiero acceder al flujo del Nivel 3 para poder introducir los datos necesarios y lograr el objetivo del Nivel 3.
HU39	Como Jugador quiero acceder a la pantalla de información para conocer los datos técnicos necesarios para el Nivel 3.
HU40	Como Jugador quiero poder avanzar a la escena siguiente cuando he finalizado la escena que estoy jugando para poder progresar en el Nivel 3 y en el juego.
HU41	Como Jugador quiero poder avanzar al Nivel 4 cuando he finalizado la última escena del nivel que estoy jugando para poder progresar en el juego.
HU42	Como Jugador quiero poder guardar los logros que he completado en el Nivel 3 para poder progresar en el juego y verlo reflejado en la pantalla de resumen y en la pantalla de logros.
E7: Nivel 5	
HU43	Como Jugador quiero poder consultar la introducción del Nivel 4 para poder leer información relativa a la historia.
HU44	Como Jugador quiero consultar el contexto del Nivel 4 para entender la rama release y su importancia en un desarrollo para poder conocer los datos necesarios y avanzar a la siguiente escena.
HU45	Como Jugador quiero acceder al flujo del Nivel 3 para poder introducir los datos necesarios y lograr el objetivo del Nivel 3.
HU46	Como Jugador quiero acceder a la pantalla de información para conocer los datos técnicos necesarios para el Nivel 3.
HU47	Como Jugador quiero poder avanzar a la escena siguiente cuando he finalizado la escena que estoy jugando para poder progresar en el Nivel 3 y en el juego.
HU48	Como Jugador quiero poder avanzar al Nivel 4 cuando he finalizado la última escena del nivel que estoy jugando para poder progresar en el juego.
HU49	Como Jugador quiero poder guardar los logros que he completado en el Nivel 3 para poder progresar en el juego y verlo reflejado en la pantalla de resumen y en la pantalla de logros.

Tabla 3.9: Historias de usuario detalladas para los niveles 4 y 5

Para expresar algo similar a lo que son requisitos no funcionales, se realiza de forma homogénea como historias de usuario. Para conseguir realizar estas historias de usuario se realizarán tareas de documentación y edición del presente documento, así como otras tareas relacionadas con el proyecto en general. La Tabla 3.10 se enfoca en mostrar aquellas historias que no están relacionadas con el propio desarrollo de la aplicación, sino que pueden ser entendidas como requisitos no funcionales expresados como historias de usuario.

RNF como HU	
Número	**Historia de usuario**
HU71	Como equipo de desarrollo quiero tener actualizada la documentación técnica sobre la memoria del proyecto.
HU72	Como equipo de desarrollo quiero tener una interfaz en sintonía con la idea del juego para mejorar visualmente la aplicación.
HU73	Como equipo de desarrollo quiero un desarrollo web para mostrar la aplicación.

Tabla 3.10: Requisitos no funcionales como Historias de Usuario

3.6. Análisis de riesgos

Durante el proceso de planificación también se debe elaborar un plan de riesgos que permita afrontar los diversos problemas y desafíos que surjan durante toda la realización de este. Un riesgo es un evento que si ocurre puede tener efectos negativos (amenaza) o positivos (oportunidad) que afecten a los objetivos del proyecto. Tener una rápida reacción cuando ocurre algún riesgo puede reducir costes en fases posteriores, por lo que dedicar algo de tiempo en esta fase de planificación para determinar qué riesgos pueden surgir es importante.

Para poder asociar un nivel de probabilidad e impacto sobre cada riesgos, se ha seguido la siguiente matriz, que refleja en las columnas la probabilidad de que ocurra ese riesgo, en las filas, el impacto que tiene sobre el proyecto, y su unión, el riesgo total.

PROBABILIDAD/IMPACTO	BAJO	MEDIO	ALTO
BAJA	BAJO	BAJO	MEDIO
MEDIA	BAJO	MEDIO	ALTO
ALTA	MEDIO	ALTO	ALTO

Figura 3.1: Matriz Probabilidad-Impacto seguida para el análisis de riesgos

Las Tablas 3.11, 3.12, 3.13, 3.14, 3.15, 3.16 y 3.17 definen los riesgos que se han tenido en cuenta durante el proyecto. En dichas tablas se analizan los siguientes aspectos en relación a cada riesgo:

1. Identificación y descripción.

2. Probabilidad, impacto y riesgo total siguiendo la matriz explicada en la Figura 3.1.

3. Especificación del plan de mitigación y contingencia.

R01	
Título	Fallo de la planificación inicial
Descripción	La planificación inicial del proyecto es incorrecta y puede suponer un retraso en las tareas y como consecuencia, en el proyecto.
Probabilidad	Media
Impacto	Alto
Riesgo total	Alto
Plan de mitigación	▪ Ser realista en cuanto al tiempo que va a ocupar cada tarea. ▪ Realizar la funcionalidad base en el sprint.
Plan de contingencia	▪ Dedicarle mas horas de trabajo para llegar a las fechas indicadas. ▪ Reprogramar los Sprints siguientes y eliminar tareas no esenciales. ▪ Ampliar el plazo del entrega del proyecto.

Tabla 3.11: Riesgo 1

R02	
Título	Tutor no disponible
Descripción	El tutor de la Universidad no se encuentra disponible.
Probabilidad	Media
Impacto	Medio
Riesgo total	Medio
Plan de mitigación	▪ Tener planificado con anterioridad las reuniones de seguimiento con el tutor. ▪ Tener una vía de comunicación rápida.
Plan de contingencia	▪ Trabajar en tareas que no requieran del tutor. ▪ Replanificar el proyecto para reducir el impacto que tenga en tareas posteriores.

Tabla 3.12: Riesgo 2

R03	
Título	Alumno no disponible
Descripción	El alumno puede estar de vacaciones, contraer alguna enfermedad, o cualquier otro motivo que impida que el proyecto avance.
Probabilidad	Media
Impacto	Medio
Riesgo total	Medio
Plan de mitigación	▪ Si son ausencias del alumno conocidas con anterioridad, planificar en base a esos días en los que no va a estar disponible. ▪ Si son ausencias que no se conocían en el momento de planificación, planificar teniendo en cuenta que pueden darse esos casos. ▪ Tener Sprints extras para realizar ciertas tareas que no se han completado en sprints anteriores.
Plan de contingencia	▪ Reducir la carga del Sprint actual pasando tareas menos importantes a siguientes sprints. ▪ Si se está enfermo, trabajar en partes del proyecto que no requieran mucho esfuerzo. ▪ Replanificar el proyecto.

Tabla 3.13: Riesgo 3

R04	
Título	Retraso en el desarrollo de las tareas
Descripción	A medida que el proyecto va avanzando las tareas duran mas horas de las estimadas provocando un retraso en las siguientes.
Probabilidad	Media
Impacto	Medio
Riesgo total	Medio
Plan de mitigación	▪ Centrar cada tarea en la funcionalidad mínima.
Plan de contingencia	▪ Realizar mas horas para llegar a las fechas planificadas. ▪ Replanificar el proyecto y mover tareas no esenciales a los Sprints extra.

Tabla 3.14: Riesgo 4

R05	
Título	Trabajar
Descripción	El alumno trabaja durante el mismo tiempo en el que se desarrolla el proyecto y puede ocupar bastante tiempo.
Probabilidad	Alta
Impacto	Bajo
Riesgo total	Medio
Plan de mitigación	▪ Organizar el proyecto teniendo en cuenta las horas laborables. ▪ Reducir la jornada laborar si se pudiera.
Plan de contingencia	▪ Replanificar el proyecto extendiendo el tiempo de cada Sprint y el proyecto. ▪ Realizar más horas al día para compensar el tiempo que se está trabajando.

Tabla 3.15: Riesgo 5

R06	
Título	Pérdida de datos o documentos
Descripción	Parte del trabajo realizado se ve comprometido por perder documentos o el propio ordenador
Probabilidad	Baja
Impacto	Alto
Riesgo total	Medio
Plan de mitigación	▪ Guardar el proyecto en la nube. ▪ Tratar de no perder el portátil.
Plan de contingencia	▪ Volver a desarrollar las partes del proyecto perdido. ▪ Retrasar la fecha final del proyecto. ▪ Realizar más horas para no verse afectada la pérdida del trabajo.

Tabla 3.16: Riesgo 6

R07	
Título	Cambio en los requisitos
Descripción	El proyecto se desarrolla de forma que cambian los requisitos durante fases en las que ya se han realizado tareas.
Probabilidad	Media
Impacto	Alto
Riesgo total	Alto
Plan de mitigación	▪ Realizar antes las tareas más generales que no estén sujetas a cambio. ▪ Tener reuniones frecuentes con el tutor para tener bien definidos los requisitos lo antes posible. ▪ Al principio, tener un análisis exhaustivo de los requisitos para evitar malentendidos y tenerlos lo mas definidos posible.
Plan de contingencia	▪ Realizar los cambios lo antes posible. ▪ Realizar más horas al día con el fin de llegar a las fechas previstas. ▪ Replanificar el proyecto teniendo en cuenta los nuevos requisitos.

Tabla 3.17: Riesgo 7

3.7. Presupuesto

Otra fase importante a realizar antes del desarrollo del proyecto es el plan de presupuestos. Puesto que es distinto un trabajo con todas las características que tendría un proyecto real de un proyecto destinado a ser trabajo de fin de grado se realizarán dos estimaciones teniendo en cuenta costes de hardware, software y de personal.

En el supuesto caso de estar en una empresa, para calcular el presupuesto del equipo informático necesario para realizar el proyecto, se va a emplear el sistema de amortización más utilizado, que es el método lineal o de cuotas fija según la tabla de amortización.

Hay tres tablas diferentes de amortización y se aplican en función del tipo de empresa que se trate. En este caso, vamos a aplicar la tabla de coeficientes de amortización lineal para empresas en la cual, el coeficiente de amortización lineal es del 20 % para equipos electrónicos [2].

Es el coeficiente que estaremos aplicando ya que amortizaríamos el bien en menor tiempo que aplicando 10 años. Teniendo en cuenta esto último se calcula el precio del portátil al mes

de esta forma:

$$CH = \frac{800 € * 0{,}20 \frac{amort}{\text{año}}}{12 \frac{\text{mes}}{\text{año}}} = 13{,}33 €/mes \tag{3.1}$$

El ratón también está dentro de equipos electrónicos por lo que el porcentaje a deducir es el mismo:

$$CH = \frac{117 € * 0{,}20 \frac{amort}{\text{año}}}{12 \frac{\text{mes}}{\text{año}}} = 1{,}95 €/mes \tag{3.2}$$

En el presupuesto simulado también se va a calcular el coste de personal. Honeypot presentó en 2023 un informe europeo sobre los salarios de los desarrolladores para el año 2023, en el cual España tenia una media de 53,500 € anuales [21]. y teniendo en cuenta que el proyecto tiene una duración de 300h a jornada completa sería:

$$CP = \frac{2375 \frac{€}{mes}}{160 \frac{h}{mes} * 12 \frac{mes}{ao}} = 27{,}86 €/h \tag{3.3}$$

Una empresa también tiene gastos fijos. Para calcular el precio de la electricidad, hacemos una media de los últimos 5 meses, obteniendo un coste medio de 0,13 [49]. Sabiendo que el consumo habitual en PYMES oscila entre 12kW y 19kW, podemos tomar como referencia una empresa pequeña y utilizar el primer valor. La jornada va a ser de 8 horas, por lo que las horas en las que la oficina va a estar funcionando será la jornada completa:

$$Elec = 8h * 12kW * 0,13 = 12{,}48 €/\text{día} \tag{3.4}$$

$$Elec = 12{,}48 €/\text{día} * 30 \text{ días} = 374,4 €/\text{mes} \tag{3.5}$$

El porcentaje que paga la empresa por un trabajador con un contrato de tiempo indefinido es del 31,75 % [18]. Partiendo del salario mensual bruto 2375 €, definido anteriormente, se tiene el siguiente gasto:

$$SS = 2375 \frac{€}{mes} * 0,3175 = 754,06 €/h \tag{3.6}$$

La Tabla 3.18 presenta un resumen de la estimación de costes en el caso simulado, mientras que la Tabla 3.19 presenta un resumen de la estimación de costes en el caso real.

Concepto	**Precio**	**Cantidad**	**Total**
Hora de trabajo de desarrollador software	27,86 €/hora	300 horas	8359,37 €
Seguridad Social	754,06 €/mes	4 meses	3016,25 €
Electricidad	374,4 €/mes	4 meses	1497,6 €
Portátil ASUS	13,33 €/mes	4 meses	53,32 €
Ratón Logitech Pro Superlight	1,95 €/mes	4 meses	7.8 €
Total			12934,34 €

Tabla 3.18: Estimación de costes simulado

A la hora de calcular el precio real no se va a tener en cuenta el salario del equipo de trabajo, pues se trata de un estudiante realizando su TFG, ni otros costes que suele tener en cuenta una empresa y únicamente se tendrá en cuenta el coste del material hardware y gastos en electricidad e Internet.

Para la amortización de los equipos informáticos, escogiendo ser autónomo, se va a determinar el rendimiento utilizando la tabla de estimación directa simplificada en el cual se indica que el equipo informático tiene un coeficiente de amortización lineal es del 26 % [1], conforme a las tablas de amortización aplicables, este tipo de activo corresponde al grupo 5: "Equipos para tratamiento de la información y sistemas y programas informáticos".

Al igual que en el cálculo de los equipos informáticos en el presupuesto de costes simulado, se amortizaría antes que pasados los 10 años.

El porcentaje a deducir será el siguiente:

$$CH = \frac{800 € * 0{,}26 \frac{amort}{\text{año}}}{12 \frac{\text{mes}}{\text{año}}} = 17{,}33 €/mes \tag{3.7}$$

$$CP = \frac{117 € * 0{,}26 \frac{amort}{\text{año}}}{12 \frac{\text{mes}}{\text{año}}} = 2{,}54 €/mes \tag{3.8}$$

Así mismo, el gasto en electricidad, se va a tomar como media el precio medio del año 2023, que fue 60,26 €.

El precio por tener Internet en casa se ha sacado teniendo en cuenta tarifa que incluya WiFi más sencilla de Movistar.

Concepto	**Precio**	**Cantidad**	**Total**
Portátil ASUS	17,33 €/mes	4 meses	69,33 €
Ratón Logitech Pro Superlight	2,54 €/mes	4 meses	10,16 €
Electricidad	60,26 €/mes	4 meses	241,04 €
Internet	31,90 €/mes	4 meses	127,6 €
Total			448,13 €

Tabla 3.19: Estimación de costes real

Capítulo 4

Análisis

Este proyecto tiene una carga mayor de definición que de análisis, y en el Capítulo 2 se detalla en profundidad el Documento de Diseño del Juego que recopila información y análisis previo al desarrollo del juego. Sin embargo, en esta sección se recogen diagramas que reflejan modelos de la información y del flujo de actividades que surgen a partir del análisis visto.

4.1. Modelo de dominio inicial

En la Figura 4.1 se refleja el modelo de dominio inicial que analiza la información que maneja el juego y surge como resultado de la fase de análisis de lo documentado en el Capítulo 2.

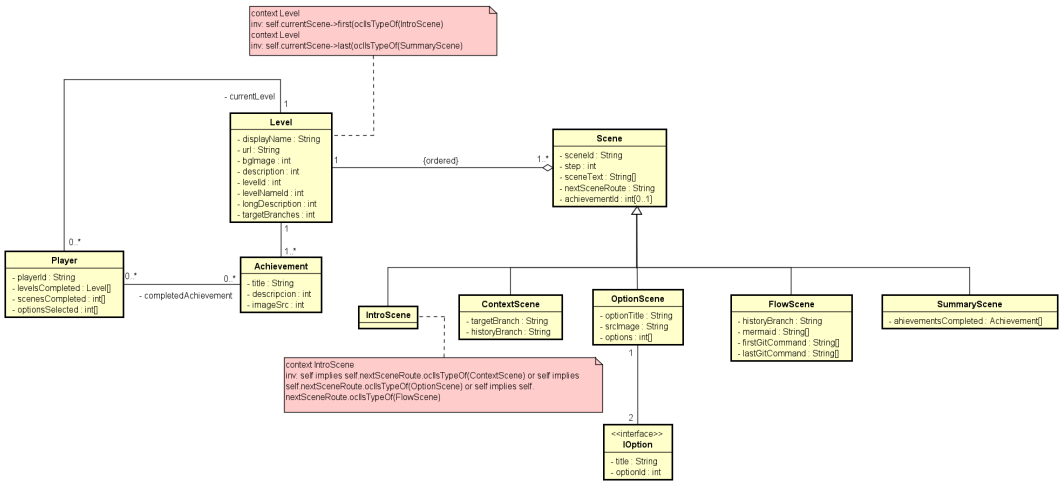


Figura 4.1: Diagrama de dominio inicial

4.2. Diagrama de actividad

Como se ve reflejado en el diagrama de la Figura 4.1, un nivel es un conjunto de escenas ordenadas. Sin embargo, la ordenación no viene dada para todas las escenas, sino para un subconjunto de ellas. Para ilustrar mejor esta organización entre escenas, la Figura 4.2 representa un modelo dinámico en términos de diagrama de actividad relativo al flujo entre escenas.

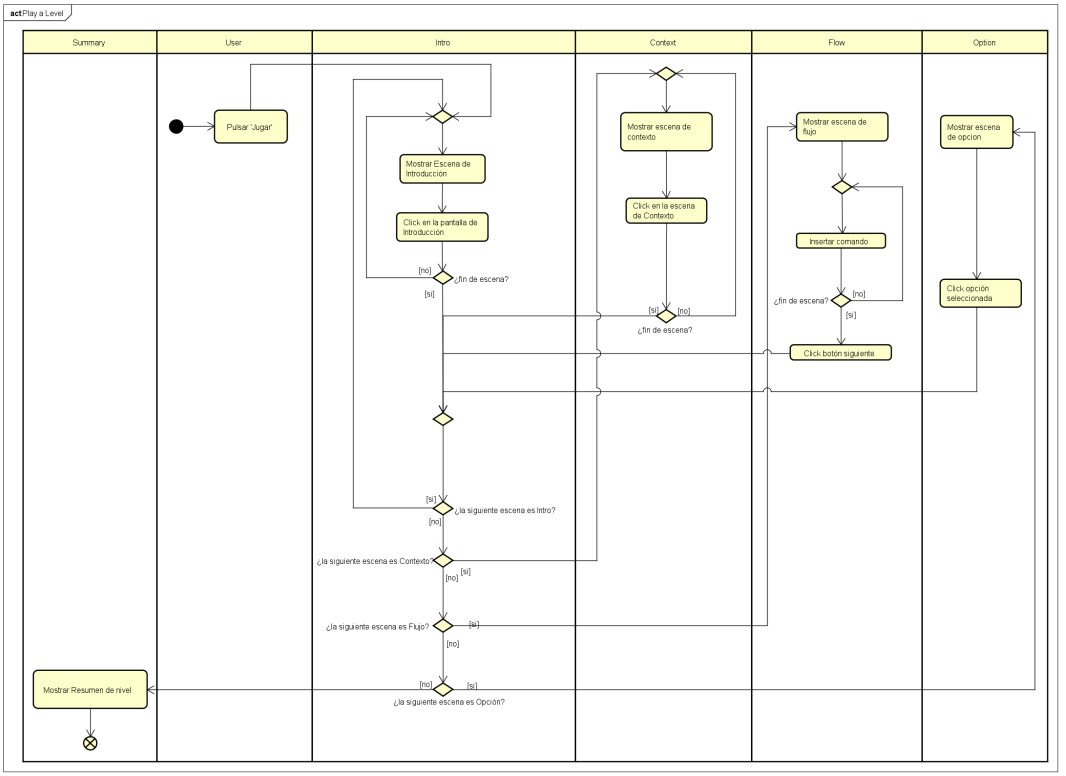


Figura 4.2: Diagrama de actividad: Organización entre escenas

Un nivel siempre empieza con una escena de Introducción con el fin de situar al jugador en la historia. Tras esa primera escena, el tipo de escena que puede mostrarse será del tipo *Intro, Context, Flow, Option* o *Summary*, teniendo en cuenta que esta última debe ser la correspondiente al final de un nivel y, por tanto, la última escena.

Capítulo 5

Tecnologías utilizadas

En este capítulo se presenta un resumen de las tecnologías que se han utilizado durante el desarrollo del proyecto y abarca las herramientas para la documentación y comunicación durante todo el proyecto, análisis, diseño y desarrollo de la aplicación.

5.1. Gestión de proyecto, documentación y comunicación

5.1.1. Notion

Notion [23] es una herramienta de productividad que permite organizar proyectos de diversas índoles debido a su capacidad para personalizar cada plantilla que se quiera usar. Notion funciona como un repositorio infinito donde se pueden crear, editar, compartir contenidos como notas, listas, bases de datos [25],... Además, es una aplicación multiplataforma ya que se puede utilizar desde Web o desde la aplicación móvil en iOS y Android. Además también permite integrar otras aplicaciones como Trello, GitHub,...

En mi caso se ha partido de una plantilla, ya creada por la Comunidad, la cual he adaptado para poder utilizarlo como Sprint Board. Para cada Historia de usuario se ha creado una Tarea dedicada a esta, añadiendo diversas propiedades como fechas de inicio y fin, Sprint en el que se encuentra o la prioridad que tiene.

5.1.2. Clockify

Como herramienta para medir el tiempo he utilizado Clockify [48]. Esta herramienta, además de soportar versiones para escritorio, web y móvil, está integrada con Notion de forma que cada Tarea añadida en Notion tiene un cronómetro asociado a ella. De esta forma, siempre que se quiera trabajar en alguna tarea en específico, se inicia un cronómetro que

empezará a calcular el tiempo empleado en esta. Para finalizar, se vuelve a pulsar el botón y en el tablero de Clockify dedicado a este proyecto se van acumulando las horas para cada tarea.

5.1.3. Overleaf y LaTeX

LaTeX es un sistema de composición de textos muy utilizado para crear documentos de alta calidad tipográfica. Es por ello que LaTeX es especialmente útil para la producción de documentos técnicos y científicos que incluyen fórmulas matemáticas, gráficos, tablas, referencias, etc. Es un estándar de facto para la comunicación y publicación de documentos científicos y es software libre que se puede instalar en diferentes sistemas operativos [36].

Overleaf es un editor online de LaTeX que permite crear y compartir documentos de forma colaborativa y sencilla. Ofrece una gran variedad de plantillas para diferentes tipos de documentos, como artículos, tesis, libros, presentaciones, y permite sincronizar tus proyectos con otros servicios como Dropbox, GitHub o Google Drive.

Esta memoria está escrita bajo el editor Overleaf, junto con todas las características de LaTeX.

5.1.4. Microsoft Teams

Microsoft Teams es una plataforma basada en la nube destinada a la comunicación y colaboración entre personas. Permite crear espacios para conversar entre diversas personas en tiempo real. Esta plataforma se ha estado empleando durante todo el desarrollo del proyecto a la hora de organizar las reuniones o establecer un contacto constante entre la estudiante y la tutora.

5.2. Análisis y diseño

5.2.1. GoodNotes

GoodNotes es una aplicación destinada a dispositivos que tienen como sistema operativo *iOS* que se utiliza como cuaderno digital para poder anotar en estos todo lo que se haría en uno real. Además, tiene la opción de importar PDFs y anotar sobre ellos. Esto resulta de gran utilidad en el ámbito educativo ya que así se pueden hacer anotaciones en los PDFs casi paralelamente a lo que el profesor explica. Como presenta muchas herramientas para la personificación de pinceles, colores, tipografías, etc, también es útil si se quiere bocetar algo no muy complejo.

Por la familiaridad que tengo con la aplicación y la comodidad que supone tener un *Stylus* para dibujar sobre la superficie de la Tablet, se ha optado por crear el *Storyboard* del juego en

esta aplicación. De esta forma se ahorra bastante tiempo a la hora de copiar y pegar dibujos y estructuras de una página a otra y en definitiva, resulta mucho más cómodo que hacerlo a papel y lápiz si va a ser algo sencillo, como es el caso de los bocetos de este proyecto. Este *Storyboard* se puede encontrar en el repositorio, dentro del directorio `/doc`.

5.2.2. Astah

Astah [12], anteriormente JUDE (Java and UML Developers' Environment), es una herramienta de modelado UML. Este tipo de herramienta soporta en gran parte la notación y semántica asociadas a UML como lenguaje de modelado unificado. Es un estándar de modelado de propósito general importante para la ingeniería de Software.

Se ha empleado Astah como aplicación para desarrollar los diversos diagramas que se han realizado durante las fases de Análisis y Diseño.

5.3. Implementación del proyecto

5.3.1. Angular

Angular es un framework para aplicaciones web desarrollado en TypeScript, de código abierto, mantenido por Google, que se utiliza para crear y mantener aplicaciones web de una sola página (SPA) y aplicaciones web progresivas (PWA) [59]. Angular permite crear interfaces de usuario dinámicas y reutilizables mediante el uso de componentes, que son bloques de código que encapsulan la lógica, el estilo y el contenido de una parte de la página. También ofrece una serie de herramientas y servicios para facilitar el desarrollo, la prueba, el despliegue y la optimización de tus aplicaciones web [39].

5.3.2. TypeScript

JavaScript es un lenguaje de programación dinámico débilmente tipado que permite implementar distintas funcionalidades en paginas web. Typescript es un superconjunto de JavaScript, de código abierto desarrollado y mantenido por Microsoft, que esencialmente añade tipos estáticos y objetos basados en clases. TypeScript permite trabajar de manera estructurada y ejecutar grandes proyectos. Se utiliza con el framework de Angular ya que ofrece ventajas como la detección temprana de errores, tipado fuerte y calidad de software.

El núcleo de la parte del cliente de JavaScript consiste en ciertas características que permiten guardar datos en variables, operaciones en bloques de texto y ejecución de código en el momento en el que ciertos eventos ocurren en la página.

5.3.3. SCSS

CSS (Cascading Style Sheets) es un lenguaje que se utiliza para definir el aspecto y la disposición de los elementos de las páginas web. Con CSS se puede determinar, el tipo de letra, el tamaño, el color, el fondo, el margen, el borde o la posición de los elementos HTML [42]. Además, permite separar la información de presentación, que es todo lo que afecta a la apariencia visual de la página como colores, fuentes, margenes, etc, de la información estructural, que es todo lo que define la organización lógica de la página como títulos, listas, párrafos, etc, lo que hace que el código HTML sea más simple y fácil de mantener [61].

SASS (Syntactically Awesome Style Sheets) es un preprocesador de CSS que permite escribir código CSS con más funcionalidades y flexibilidad, como variables, anidamiento, mixins, herencia o funciones. Este lenguaje tiene dos sintaxis. La original, que es la sintaxis con sangrado, que utiliza la sangría para separar bloques y, la más reciente, la sintaxis con llaves (SCSS) que usa el formato de bloques como CSS. SASS se traduce a CSS estándar mediante un compilador [64].

5.3.4. HTML

HTML5 (HyperText Markup Language, versión 5), es la versión más actualizada de *hypertext markup language* es un estándar que sirve para definir la estructura, el diseño y el contenido de una página web [16]. HTML se basa en un conjunto de etiquetas que sirven para definir el texto y otros elementos que compondrán una página web, como imágenes, listas, vídeos, etc. Es decir, la organización lógica de la página. Como se ha comentado, al ser un estándar, se busca que sea un lenguaje capaz de poder ser interpretada de la misma forma en cualquier navegador.

5.3.5. Mermaid

Mermaid es una herramienta para crear gráficos y diagramas, basada en JavaScript, que utiliza definiciones de texto inspiradas en Markdown y un renderizado para crear y modificar diagramas complejos. De esta forma, se dibujan nodos y diagramas de forma sencilla muy personalizables.

Debido a que el proyecto está basado en enseñar el método Gitflow, tener un apoyo gráfico de las distintas ramas existentes, junto con sus respectivos *commits* asociados a cada una de ellas, era algo necesario y con el uso de esta herramienta, y su integración con Angular, se convierte en una tarea sencilla.

Para los diagramas relacionados con el flujo de Gitflow que se muestran en el Capítulo 2 se ha utilizado el editor de texto en línea presente en la web que se puede encontrar aquí [35].

5.3.6. Node.js

Node.js es un entorno de ejecución de un solo hilo, de código abierto, basado en JavaScript, que se utiliza para desarrollar aplicaciones escalables del lado del servidor y de la red. Node.js utiliza un modelo de operaciones de E/S no bloqueantes guiado por eventos que hace que sea súper ligero y eficiente, perfecto para aplicaciones que necesiten pasar muchos datos ya que podrá trabajar de forma simultánea diversas peticiones.

npm (Node Package Manager) es el gestor de paquetes que permite publicar y compartir módulos de node. Existe un repositorio en que almacena paquetes de JavaScript y Node.js.

5.3.7. Paquetes empleados en Angular

Angular tiene muchos paquetes que facilitan las tareas en el ámbito de programación. En este apartado se comentan aquellos que se han añadido a mayores de los que Angular tiene por defecto.

Mermaid

Como se ha comentado anteriormente, se ha empleado Mermaid para mostrar los diagramas relacionados con el flujo Gitflow. Se ha utilizado este paquete para su integración con renderizado dinámico en el código y poder generar diagramas en base a lo que el usuario introduzca por pantalla.

Ionic

Ionic [41] es un marco de desarrollo de aplicaciones de código abierto que facilita la construcción de aplicaciones web nativas y progresivas de primera calidad con tecnologías web.

Los paquetes empleados en el proyecto contienen los componentes web que componen los bloques de construcción de IU reutilizables de Ionic Framework.

RxJS

RxJS [40] es una biblioteca para componer programas asincrónos y basados en eventos mediante el uso de secuencias observables. Proporciona un tipo básico, los observables, (Observador, Schedulers, Subjects) y los operadores inspirados en métodos de `Array` (map, filter, reduce, every, etc) que hacen más fácil la creación de código asíncrono o basado en *callbacks*.

uuid

uuid es un paquete que genera Identificadores únicos universales (UUID) de forma sencilla. Este se emplea para asignar un identificador al jugador. En este caso se ha empleado la versión 4, que se refiere a la concatenación de caracteres al azar pero esta librería soporta otras versiones así que sería fácil cambiar de versión si se requiriese. Este se puede obtener en este enlace [56].

Capítulo 6

Diseño

6.1. Diseño arquitectónico

En el Capítulo 5 se explica de forma esquemática las tecnologías empleadas para el desarrollo del juego. La tecnología principal en la que el juego va a desarrollar forma parte del framework de Angular por lo que se va a explicar de una forma más detallada cómo se ha organizado.

6.1.1. Arquitectura MVVM

La arquitectura *Model View ViewModel* (MVVM), es un patrón arquitectónico que surge como refinamiento de la arquitectura *Model View Controller* (MVC) que permite desacoplar la vista de la lógica de negocio, separando el desarrollo de la interfaz gráfica de usuario. Este patrón divide la aplicación en tres partes: Modelo, Vista y Modelo de Vista.

- Modelo (*Model*). Representa los datos de la aplicación y la lógica de negocio.
- Vista. La vista es la interfaz de usuario de la aplicación y se encarga de mostrar los datos del usuario y las interacciones que tiene el usuario con la aplicación.
- Modelo de vista (*ViewModel*). Sirve como puente entre la vista y el modelo mediante el *data-binding*. Guarda la lógica de la aplicación que vincula la vista al modelo exponiendo los objetos del modelo para gestionar y presentarlos más fácilmente. Remplazaría el Controlador en una arquitectura MVC.

6.2. Arquitectura de carpetas y ficheros en Angular

Partiendo del patrón MVVM, la arquitectura Angular se apoya en ciertos conceptos fundamentales y es necesario comentar primero la estructura base que sigue este framework para poder describir la forma en la que se organizan los ficheros y el cómo se implementa el patrón MVVM.

Los bloques básicos que se pueden crear se denominan componentes y están todos bajo módulos conocidos como *NgModules*.

Los componentes son los bloques básicos reusables para las aplicaciones de Angular que definen la vista y controlan distintas secciones y aspectos de la aplicación. Angular puede seleccionar y modificar estos componentes en función de la lógica y los datos.

Partiendo del patrón MVVM, los componentes son los encargados de la Vista y el Modelo de vista mediante los siguientes archivos:

- Un archivo HTML que es una plantilla que muestra lo que se carga en la página. Complementario y opcional al fichero HTML existe un fichero SCSS que es una hoja de estilos que se aplica al HTML. Esto sería la vista.
- Un archivo TypeScript que define el comportamiento y la interacción entre la plantilla HTML y la estructura DOM. Es el Modelo de vista.
- Un selector CSS que define cómo el componente se utiliza en una plantilla HTML.

Angular CLI genera además un fichero spec.ts para crear tests unitarios del componente y existe uno por cada fichero .ts aunque en este caso no se han utilizado.

En los módulos se encuentran los componentes, anteriormente definidos y otra serie de ficheros necesarios para el desarrollo. Los módulos recogen partes del código con funcionalidad independiente del resto y permiten organizar al equipo de desarrollo asignando a cada miembro una funcionalidad, definir una lista de módulos necesarios para un módulo en concreto (ya que *NgModules* pueden importar funcionalidad descrita por otros así como exportar la funcionalidad necesaria y usarse en otros), desarrollar funciones de manera gradual y crear aplicaciones escalables fácilmente [15] [46].

6.3. Desarrollo basado en componentes

Tras una breve descripción de la estructura que tienen los componentes Angular, podemos hablar de la arquitectura basada en componentes. Este tipo de arquitectura es un enfoque de desarrollo que se centra en convertir elementos en componentes reutilizables. Se trata de un híbrido entre la arquitectura basada en capas y la arquitectura basada en funcionalidades. En vez dividir la aplicación en capas horizontales, como sucede en el enfoque de capas, se

divide verticalmente en componentes modulares, tal y como describe la arquitectura basada en funcionalidades.

Como se ha comentado en la sección anterior, un componente es un elemento reutilizable que agrupa un conjunto de servicios, datos y código relacionado y aporta funcionalidad adicional al mismo tiempo que se acelera el tiempo de desarrollo e implementación de la aplicación [47].

6.3.1. Relevancia de la arquitectura basada en componentes

La arquitectura basada en componentes está teniendo un crecimiento importante dentro de la ingeniería de software ya que está evolucionando para satisfacer las necesidades del usuario y deben ser lo más flexible posible. Este enfoque presenta ciertas cualidades como por ejemplo [51]:

- Reusabilidad. Este enfoque permite la reusabilidad del código, lo que se traduce en que múltiples desarrolladores pueden utilizar ese componente en aplicaciones distintas.
- Extensiblidad. Los componentes pueden extenderse y modificarse fácilmente.
- Reemplazables. Estos componentes pueden cambiarse por otros sin afectar a la aplicación.
- Escalabilidad. Resulta fácil escalar aplicaciones y servicios conforme se va necesitando.

6.4. Componentes en Angular

Angular sigue una arquitectura basada en componentes, es un concepto fundamental y juega un papel crucial a la hora de organizar aplicaciones con Angular. En concreto, se centran en componentes para el front, y por tanto no abarcan todas las capas de una aplicación. Aunque esto se podría complementar con un backend en caso de ser necesario, siguiendo su arquitectura, como puede ser de capas o clean con servicios o microservicios.

Este enfoque ayuda a dividir la aplicación en bloques de construcción más pequeños y reutilizables denominados componentes. Un componente no es más que una clase TypeScript con un decorador (@). Contiene métodos y propiedades que se pueden utilizar en archivos HTML. A continuación se muestran los diagramas de componentes que contiene la aplicación.

En la Figura 6.1 se observa el diagrama de componentes referido a un componente básico. Este está formado por una clase TypeScript, un fichero HTML y su correspondiente hoja de estilos, en este caso, SCSS. En la figura también se refleja el uso de un servicio dentro del componente. Gracias al inyector de servicios, este actúa como una interfaz, dejando que el componente pueda consumir el servicio.

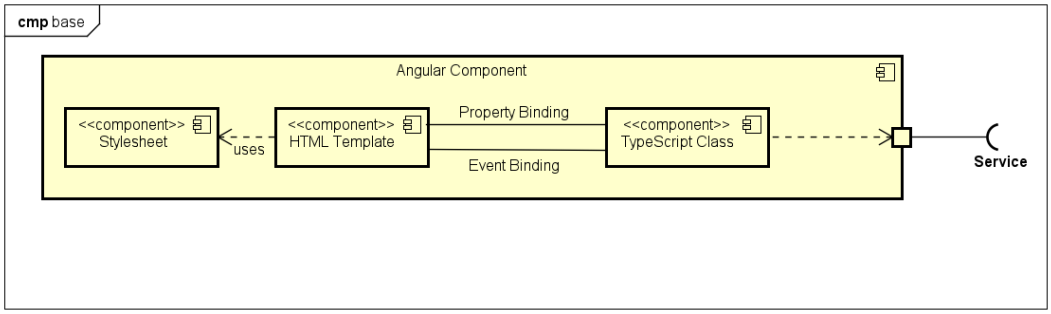


Figura 6.1: Diagrama de componentes: Componente básico

La relación que existe entre los componentes padres e hijos se puede ver en la Figura 6.2. El componente hijo se encuentra definido en la plantilla HTML de forma que este componente se puede comunicar fácilmente con el componente padre.

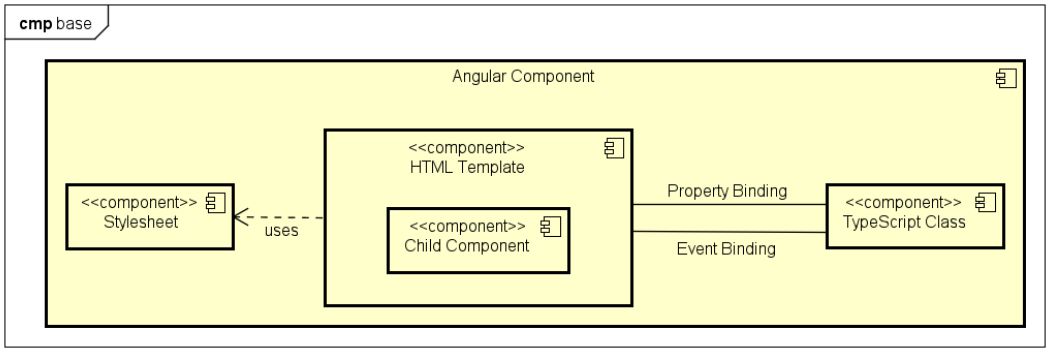


Figura 6.2: Diagrama de componentes: Componente padre e hijo

6.5. Componentes en el proyecto

Este componente, reflejado en la Figura 6.3, es el componente raíz y en el se muestran todos los componentes y actúa como padre para los demás.

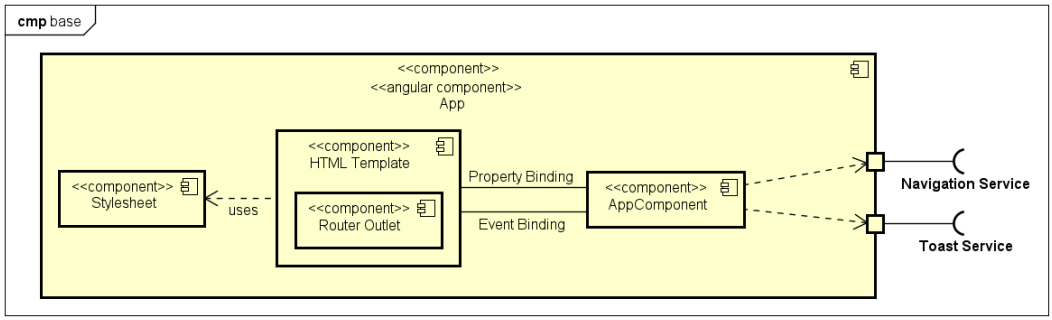


Figura 6.3: Diagrama de componentes: Componente App

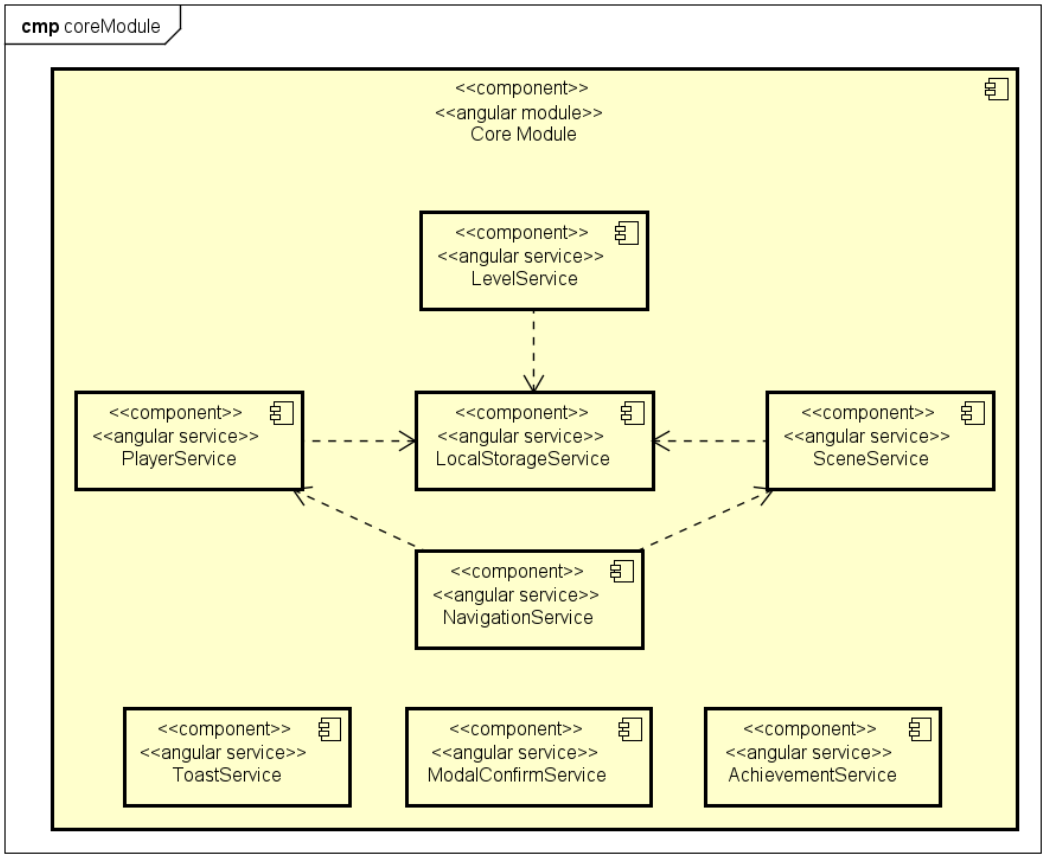


Figura 6.4: Diagrama de componentes: Módulo Core

Bajo el módulo *Core*, representado en la Figura 6.4, se encuentran los servicios que consumen los componentes alojados en el Módulo *Shared*. Se encuentran los servicios encargados

de guardar la información necesaria del usuario y el juego para mantener un progreso en el juego, así como los servicios necesarios para el correcto funcionamiento entre escenas y niveles. También se encuentran otros servicios que manejan el comportamiento de componentes como *Toast* y *ModalConfirm*.

En la Figura 6.5 se reflejan todos los componentes que se emplean en los distintos módulos del módulo *Features*. Son componentes que no tienen mucha lógica asociada, y tan solo se encargan de mostrar la información, *parsear* si se necesita, y devolver al componente padre las acciones que hace el usuario.

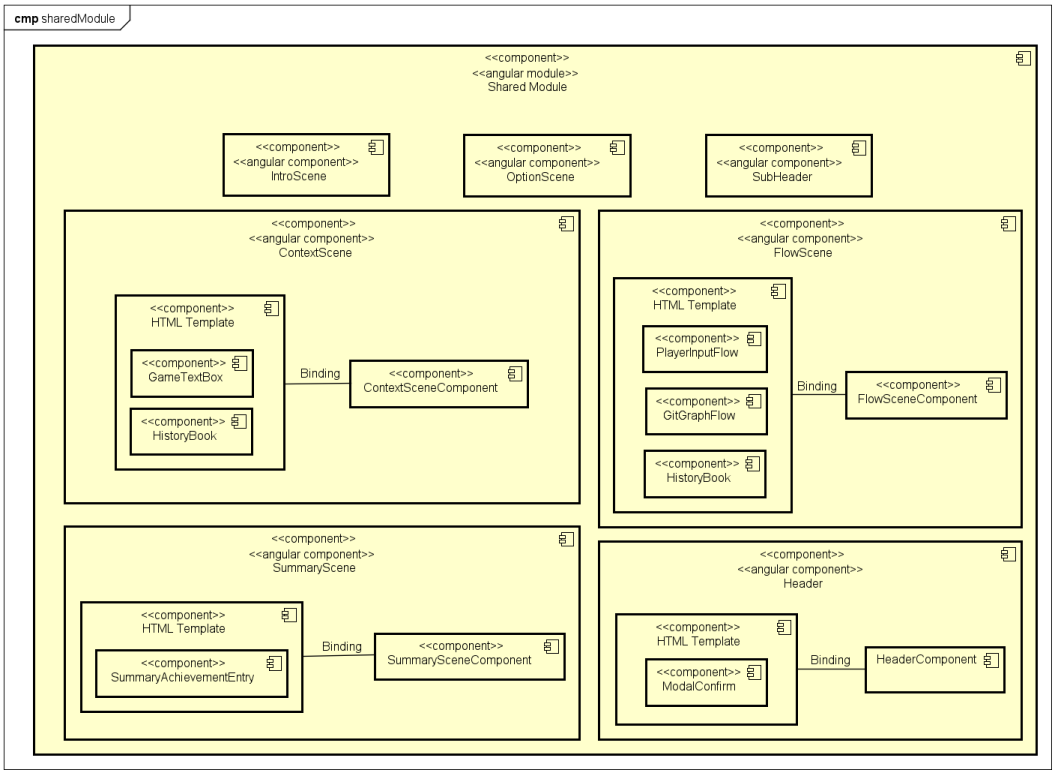


Figura 6.5: Diagrama de componentes: Módulo Shared

En la Figura 6.6 se plasman los módulos relacionados con funcionalidades del juego, sin ser los propios niveles. Se encuentra el módulo *Menu*, correspondiente al componente que se muestra al iniciar la aplicación, el módulo *Achievement* relacionado con los Logros que se van obteniendo, el módulo *LevelList* que recopila todos los niveles existentes así como una breve información relativa a cada nivel y el módulo *Information* que guarda los componentes relacionados con la información extra.

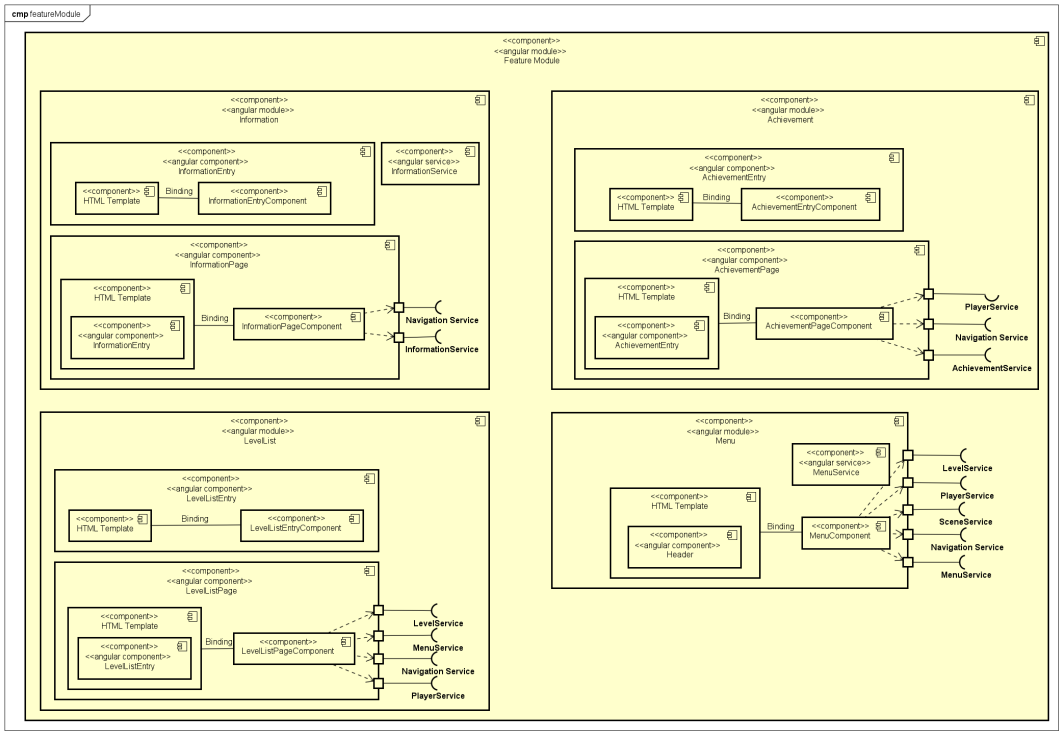


Figura 6.6: Diagrama de componentes: Módulo Features, externo a niveles

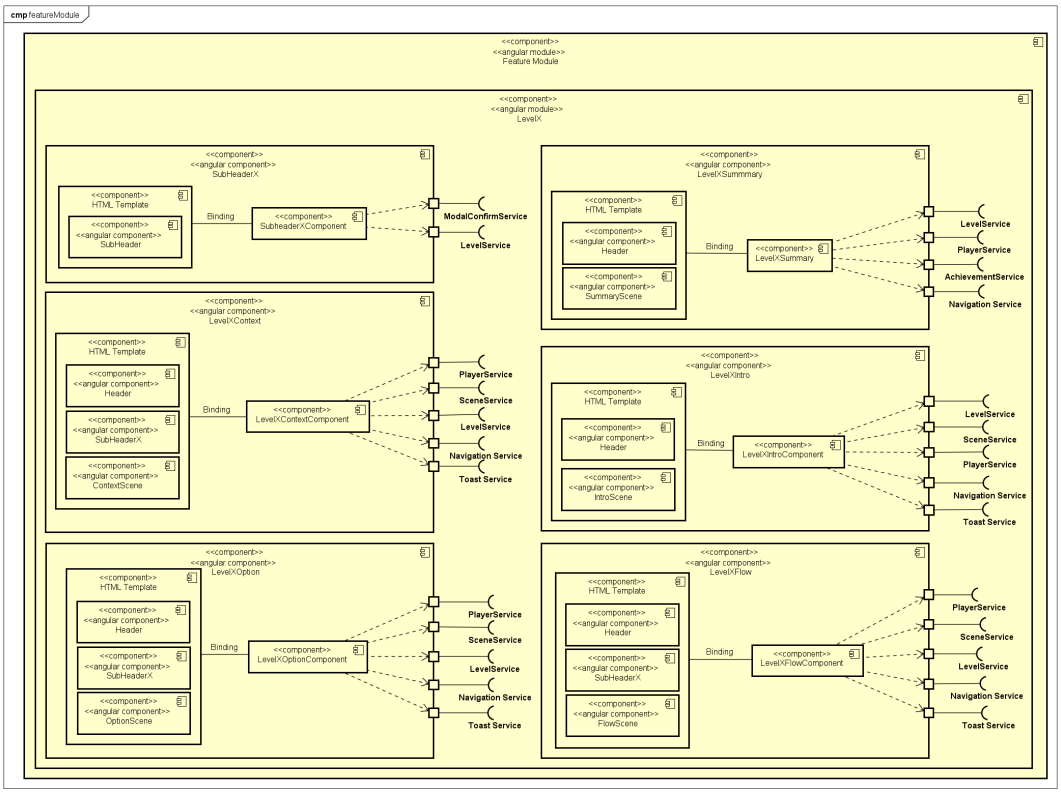


Figura 6.7: Diagrama de componentes: Módulo Features, niveles

Por último, en la Figura 6.7 se representa la estructura que tiene un Nivel. Este diagrama se puede extrapolar a cada nivel existente dentro de la aplicación, pudiendo diferir en ciertos componentes si el nivel no requiere de algún "angular component", como lo sería en el caso del Nivel 0 el cual no tiene esos componentes relacionados con la escena de opción.

6.6. Clases en Angular

Tras definir los componentes que integran esta aplicación, podemos hablar de las clases que se integran dentro de los componentes y permiten modelar los datos que se muestran en el juego, así como los datos que se van guardando durante el progreso del juego.

En primer lugar, hablamos de clases de la aplicación, como los modelos que se utilizan. En la Figura 6.8 se muestran las clases que modelan los datos con los que trabajan los distintos componentes, y la lógica asociada.

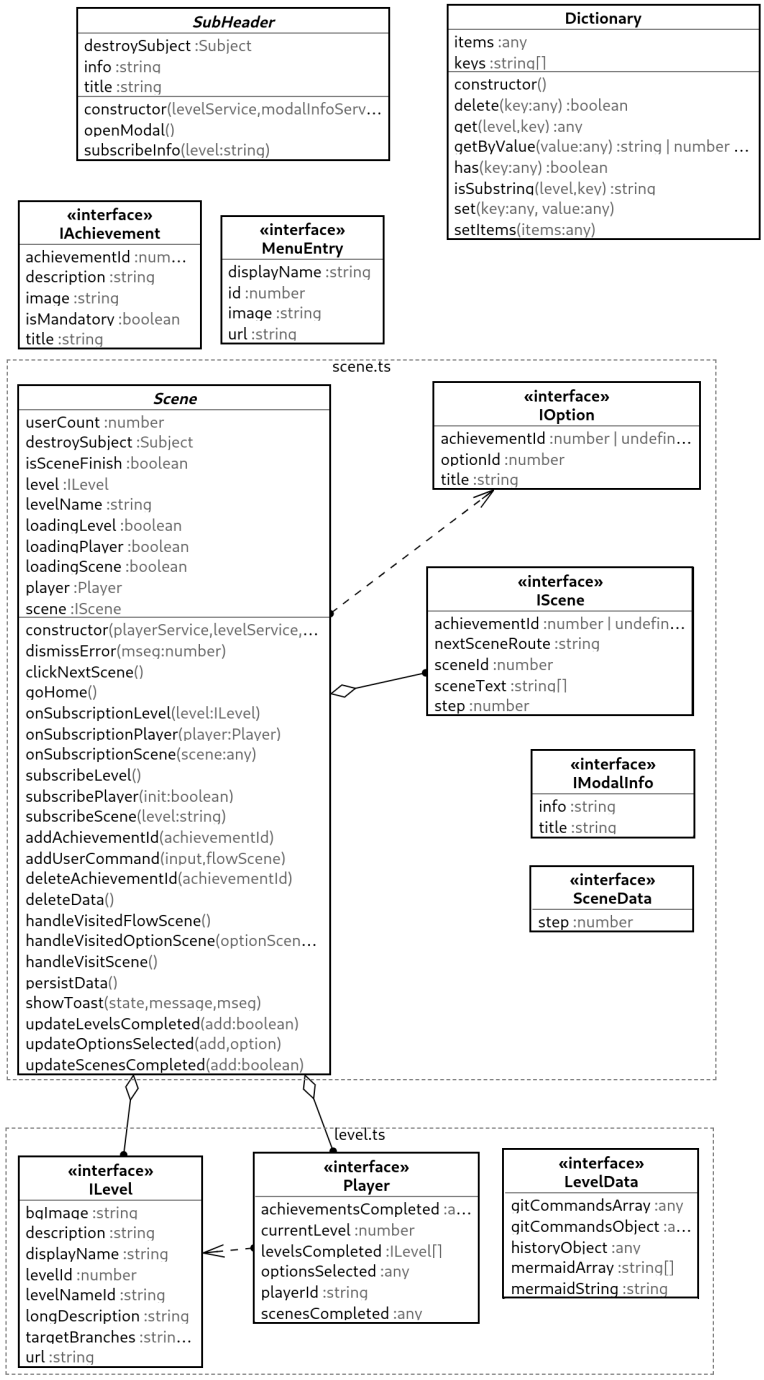
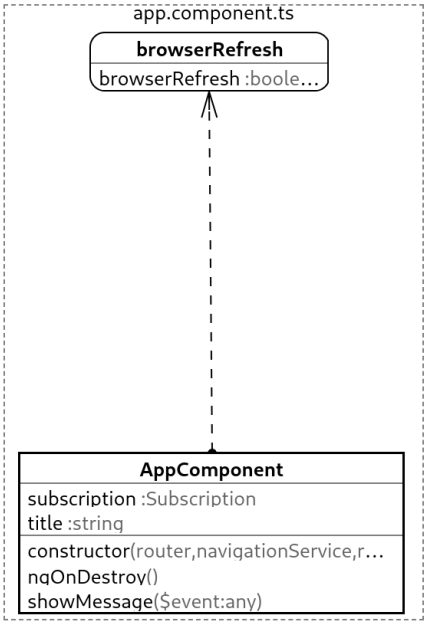
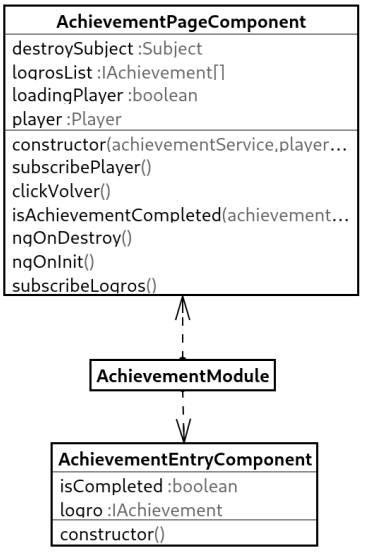


Figura 6.8: Diagrama de clases: Modelos

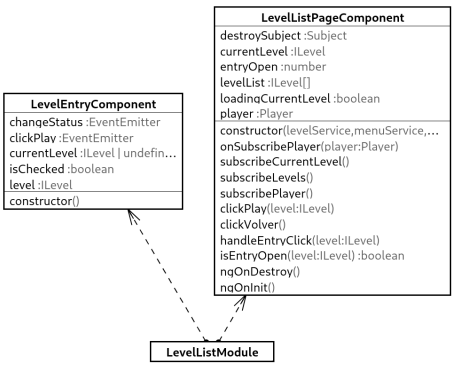
La clase Scene aloja toda la lógica que se emplea en cada componente relacionado con las escenas. De esta forma, se reduce la repetición de código, y solo aquellos métodos que tienen un comportamiento específico entre componentes, se implementan en el propio componente.



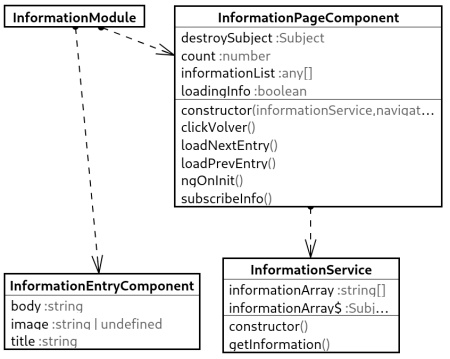
(a) Diagrama de clases: AppModule



(b) Diagrama de clases: Módulo Logros



(c) Diagrama de clases: Módulo Lista de niveles



(d) Diagrama de clases: Módulo Información

Figura 6.9: Diagrama de clases

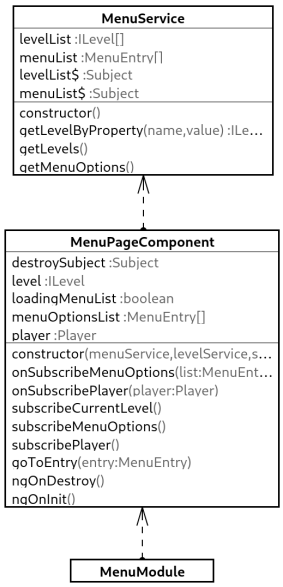


Figura 6.10: Diagrama de clases: Módulo Menú

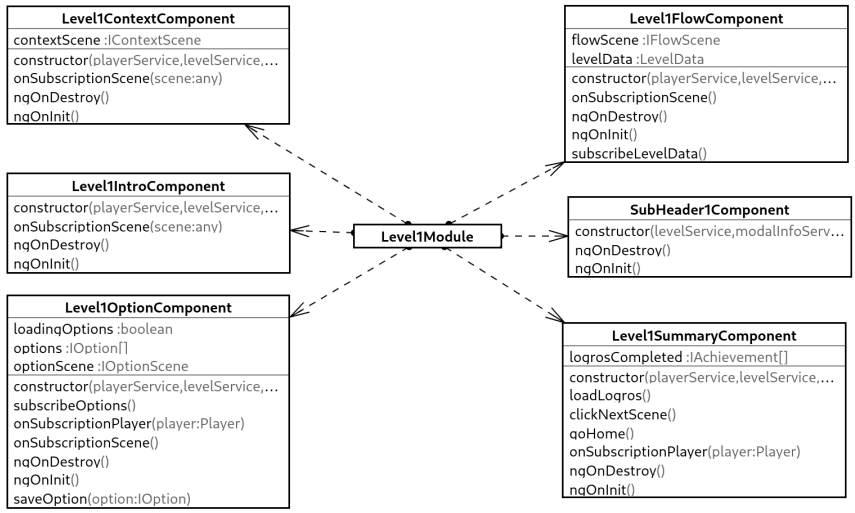


Figura 6.11: Diagrama de clases: Módulo Nivel 1

En las Figuras 6.9a, 6.9b, 6.9d, 6.10 y 6.9c se detallan las clases que ejecutan en tiempo de compilación los componentes presentados en la sección anterior. Se presenta el módulo principal, AppModule, y los distintos módulos que guardan las funcionalidades del juego.

La Figura 6.11 representa al Módulo del Nivel 1, pero, como se ha explicado en la Figura 6.7, esto se puede extrapolar a cada nivel existente en el juego, ya que la estructura y componentes que tienen son similares.

6.7. Despliegue

Por último, a la hora del despliegue de la aplicación, se tiene en cuenta que la aplicación no depende de ninguna funcionalidad de Backend por lo que para desplegar la aplicación solo se necesita un servidor web que compile la aplicación Angular. Esto se muestra en la Figura 6.12.

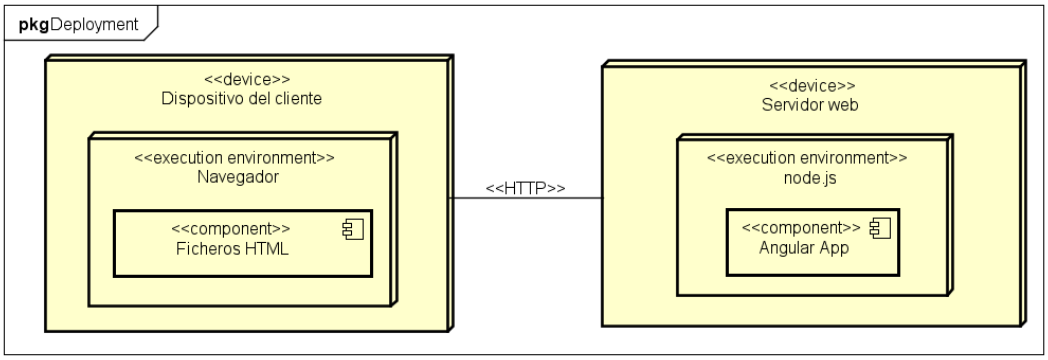


Figura 6.12: Diagrama de despliegue

6.8. Patrones de diseño

En esta sección se recogen los patrones de diseño que se aplican en este proyecto.

6.8.1. Patrón Observador

El patrón de diseño *Observer*, permite ver los cambios que se producen por un objeto y notificar a los observadores de ese objeto que el estado ha sido modificado. Este patrón es uno de los principales patrones de diseño utilizados en interfaces gráficas de usuario, ya que permite desacoplar al componente gráfico de la acción a realizar [50].

El patrón funciona de la siguiente manera. Existe un objeto cuya responsabilidad es mantener una lista de suscriptores y notificarles de alguna actualización, los observadores o

suscriptores, se suscriben y desuscriben de la información a actualizar del sujeto y modifican la información en base a lo que el sujeto haya cambiado. Este patrón es útil cuando muchos elementos dependen de los datos y las actualizaciones que otro les envíe, a esto lo conocemos como una dependencia de uno a muchos. En Angular podemos tener un sujeto que consulte datos y varios componentes que esperen esa información para actualizar la vista.

Existen muchas librerías que implementan este patrón siendo una de ellas RxJS que además de implementar los elementos del Sujeto y los Observadores, agrega operadores que modifican la información que envía el sujeto a sus observadores. En la Figura 6.13 se puede ver la estructura que sigue este patrón.

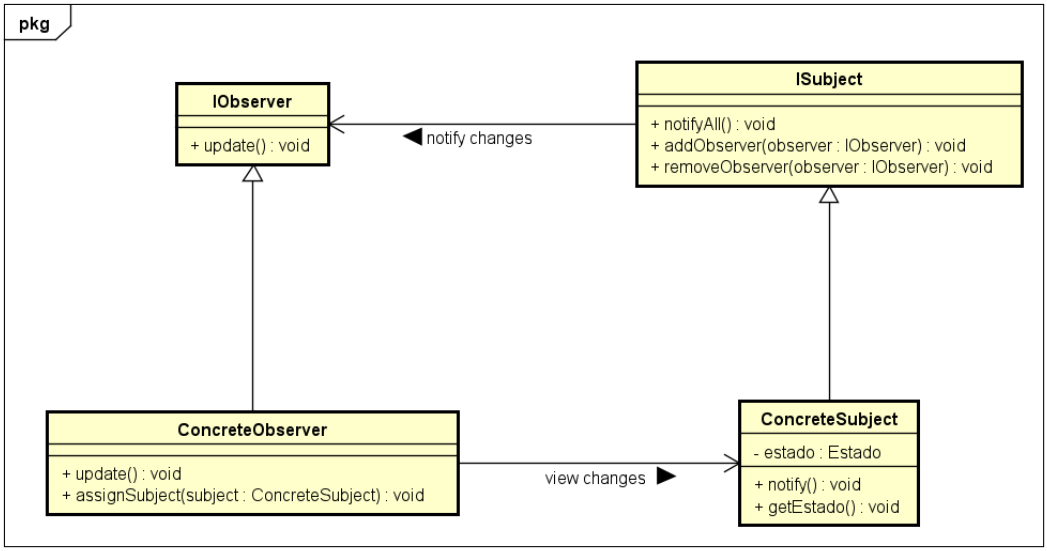


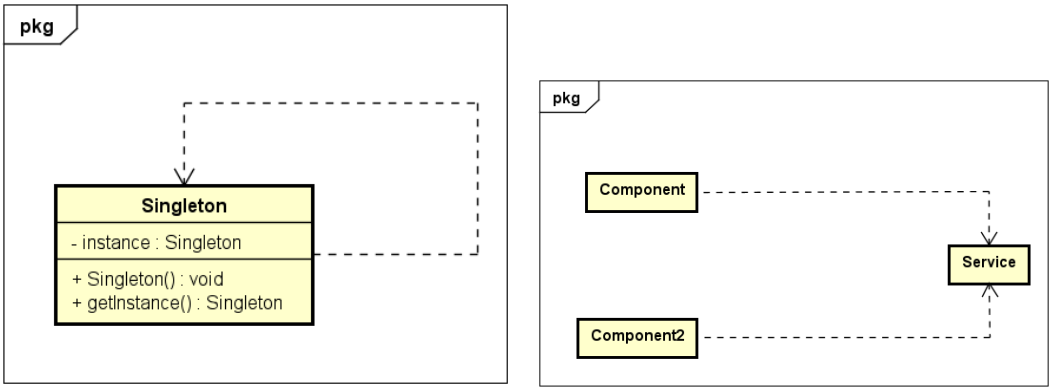
Figura 6.13: Patrón Observador

6.8.2. Patrón Singleton

El patrón *Singleton*, asegura que una clase tiene solo una única instancia y proporciona un punto de acceso global a ella.

Singleton define una función `instancia()` o `getInstancia()`que permite a los clientes acceder a su instancia única y además es responsable de su creación. De esta forma se tiene un acceso controlado a un ejemplar único y se evitan utilizar variables globales.

En Angular, por defecto los servicios en Angular son Singletons siempre y cuando no se indique lo contrario, ya que estos no están obligados a ser Singletons y todo depende del sistema de inyección de dependencias jerárquico que soporta [11]. En el caso de este proyecto, los servicios que existen son Singletons y de cada uno de ellos se tiene solo una instancia. En la Figura 6.14 se ve la estructura del patrón y cómo los componentes de Angular implementan los servicios.



(a) Patrón Singleton (b) Patrón Singleton: Servicios en Angular

Figura 6.14: Patrón Singleton

Capítulo 7

Implementación y pruebas

En este capítulo se detalla todo lo relacionado con la implementación como la organización del código, funcionalidades nuevas que se han tenido en cuenta durante el desarrollo del proyecto, cambios de diseño a lo largo de la ejecución del proyecto y las pruebas realizadas.

7.1. Organización del código

La estructura que sigue el repositorio ha estado dirigida por la organización que suele seguir un proyecto en Angular. A continuación se refleja parte del árbol de directorios seguido en el proyecto:

```
royalgit
├── src
├── doc
├── .editorconfig
├── .gitignore
├── LICENSE
├── README.md
├── angular.json
├── package.json
├── tsconfig.app.json
├── tsconfig.json
└── tsconfig.spec.json
```

El directorio raíz del proyecto tiene la estructura que se ha visto en el árbol de directorios anterior. De este directorio se destaca el directorio `/src`, que es el directorio que contiene la aplicación.

```
src
├── app
│   ├── core
│   │   ├── models
│   │   │   ├── player.ts
│   │   │   ├── scene.ts
│   │   │   └── ...
│   │   ├── services
│   │   │   ├── player.service.ts
│   │   │   ├── scene.service.ts
│   │   │   └── ...
│   │   └── core.module.ts
│   ├── features
│   │   ├── level0
│   │   │   ├── level0-intro
│   │   │   │   ├── level0-intro.component.html
│   │   │   │   ├── level0-intro.component.scss
│   │   │   │   └── level0-intro.component.ts
│   │   │   ├── level0-context
│   │   │   ├── ...
│   │   │   └── level0.module.ts
│   ├── shared
│   │   ├── components
│   │   │   ├── header
│   │   │   │   ├── header.component.html
│   │   │   │   ├── header.component.scss
│   │   │   │   └── header.component.ts
│   │   │   └── ...
│   │   └── shared.module.ts
│   ├── app-component.html
│   ├── app-routing-module.ts
│   ├── app-component.scss
│   └── app-component.ts
├── assets
└── styles
```

En la ruta /**src**/**app** se aloja el código de la aplicación. Dentro de este se encuentra el módulo raíz y 3 módulos que guardan toda la funcionalidad:

- *root*. Es el módulo principal y lo genera el Angular CLI denominándolo *AppModule* y se lanza cuando se inicia la aplicación. Todos los módulos depende de éste, directa o indirectamente, ya que solo existe un módulo raíz.
- *core*. Es el módulo en el que se guardan los modelos de dominio y los servicios que se emplean en toda la aplicación. Se encuentran los archivos relacionados con el Modelo en el patrón MVVM:
    1. *models*. Se encuentran las clases de dominio de la aplicación y se corresponde con

el Modelo en el patrón MVVM.

2. *services*. Servicios necesarios para la obtención de datos y guardado de los mismos.

- *features*. Este módulo es el encargado de guardar todas las funcionalidades implementadas, como lo es cada nivel.
- *shared*. Este módulo está formado por directivas, pipes y componentes. Se encuentran los archivos relacionados con la Vista y el Modelo de vista los cuales se emplearán en las diferentes *features* de la aplicación y por tanto se utilizarán en varios componentes. En este módulo existen estos paquetes:

    1. *components*. En este paquete se encuentran todos los componentes que se utilizarán en distintas *features*.

Para cada funcionalidad dentro del módulo *features*, la estructura que sigue el módulo es la siguiente:

```
feature1
├── component1
│   ├── component1.html
│   ├── component1.scss
│   └── component1.ts
├── component2
├── ...
└── feature1.module.ts
```

- *component-x*. Componente x que aloja parte de la funcionalidad del módulo.
- *module.ts*. Este fichero guarda las rutas de los distintos componentes del módulo.

A continuación se muestra la organización restante que se ha seguido en el proyecto:

```
src
├── assets
│   ├── data
│   │   ├── scenes.json
│   │   ├── lists.js
│   │   └── ...
│   ├── img
│   │   ├── achievements
│   │   │   ├── lvl0.png
│   │   │   └── ...
│   │   ├── gitflow-basic.png
│   │   └── ...
│   └── sound
│       ├── page-flip.mp3
│       └── ...
└── styles
    └── buttons.scss
```

```
└── variables.scss
└── ...
```

En la ruta **/src/assets** se alojan los datos y el contenido multimedia que emplea el proyecto. Se guardan imágenes relacionadas con los logros que se pueden obtener, iconos, sonidos, y los datos relacionados con el juego que se emplean a la hora de mostrar en cada momento los datos correctos.

En la ruta **/src/styles** se encuentran las hojas de estilo que se aplican a diversos componentes y etiquetas de las plantillas .html así como la hoja de variables de colores que se muestra en toda la aplicación.

7.1.1. Principio LIFT

Durante el desarrollo del proyecto, se ha querido seguir el principio LIFT ya que encaja muy bien con la filosofía que tiene este framework a la hora de organizar nombres de ficheros y carpetas. Este principio se basa en cuatro puntos básicos explicados a continuación [45]:

- ***L**ocate code quickly.* Establecer una buena estructura de ficheros para encontrar ficheros de forma fácil e intuitiva cuando no se recuerda exactamente el nombre del fichero. Por eso, mantener ficheros relacionados cerca dentro de esta estructura en una carpeta intuitiva reduce el tiempo de búsqueda y es algo esencial a la hora de poder trabajar eficientemente.
- ***I**dentify the code at a glance.* Darle un nombre descriptivo a los ficheros para saber instantáneamente qué contiene y qué representa el fichero.
- *keep the **F**lattest structure you can.* Mantener una estructura de carpetas y ficheros lo más plana posible considerando crear subcarpetas cuando la carpeta llegue a 7 ficheros o más. Así se evita la creación innecesaria de carpetas que complica escanear de una sola pasada todos los ficheros. Sin embargo, ya que tener muchos ficheros dentro de una carpeta puede ser sobreestimulante, se destaca que a partir de una cantidad de ficheros se separe en subcarpetas.
- ***T**ry to be DRY (T-DRY). Try to be Don't Repeat Yourself.* Es decir, pese a que no repetirse es importante, no es crucial si se sacrifican otros elementos del Principio LIFT por lo que siempre que sea posible se intentará no repetirse hasta que llegue el momento en el que se vean comprometidos uno o más de los elementos descritos arriba.

7.2. Bocetos

A la hora de poder desarrollar una aplicación como esta, es necesario partir de un estudio en cuanto a la interfaz que se va a presentar al usuario. Por ello, antes de poder desarrollar se dedicaron unos días en definir y bocetar una interfaz gráfica que agrupase las distintas

funcionalidades que se necesitaban. La elaboración de estos bocetos se encuadra mejor en las tareas de Diseño pero se ha decidido documentar sobre ellos en este punto, junto antes del desarrollo de la interfaz.

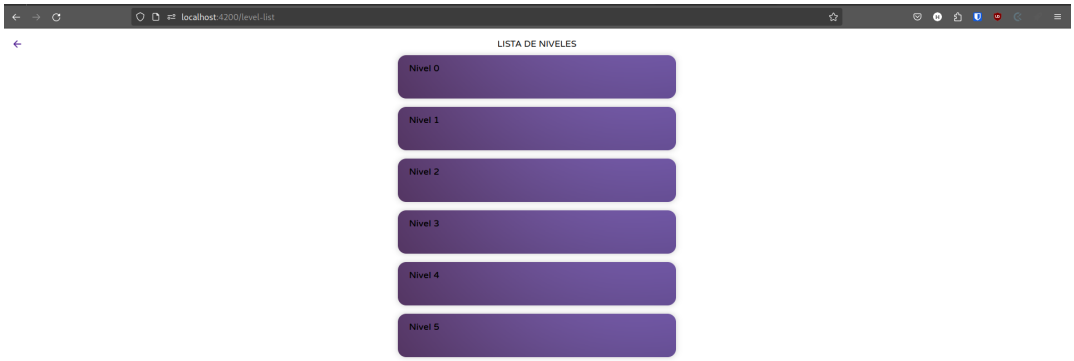


Figura 7.1: Pantalla: Lista de niveles

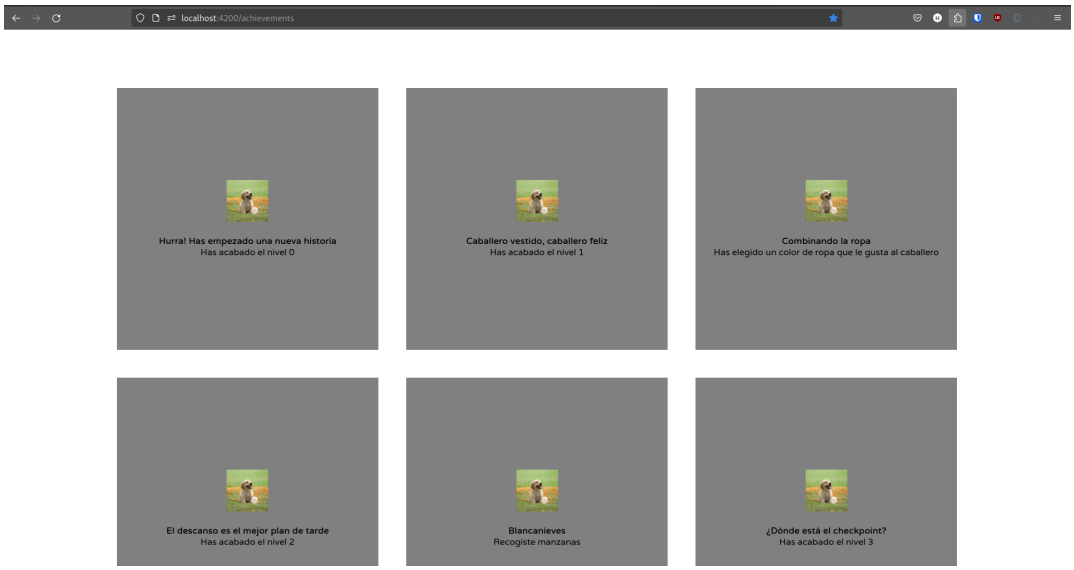


Figura 7.2: Pantalla: Logros

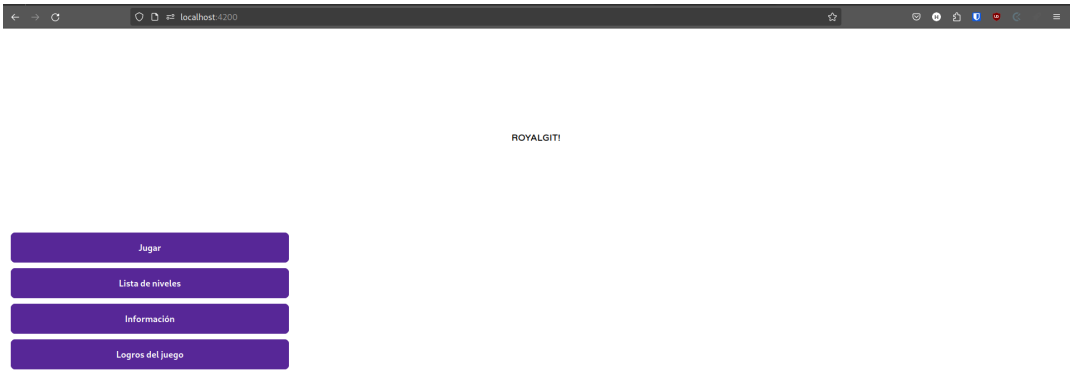


Figura 7.3: Pantalla: Menú

En las imágenes 7.1, 7.2 y 7.3 se puede ver una primera interfaz de lo que sería el juego para la lista de niveles y la pantalla de logros obtenidos. Se priorizó primero la funcionalidad antes que tener una interfaz bonita, por lo que se tenían botones y componentes más rústicos que cumplían con su objetivo.

A su vez, se fueron desarrollando los niveles con el contenido de cada uno, aunque posteriormente sufriese algún cambio.

Para cada escena dentro del nivel, se identificaron los posibles componentes que deberían crearse, pudiendo reutilizar varios componentes en escenas distintas como lo es el libro que aparece tanto en la escena de contexto, como en la de flujo.

7.3. Desarrollo de la interfaz

Tras tener claro un primer boceto de los niveles que iban a existir en la aplicación, se comenzó a crear los componentes necesarios para su implementación. En este momento se debe destacar la existencia de *Dumb Components*, y *Smart Components*. Un *Dumb Component* es un componente simple que tiene únicamente dos responsabilidades que son mostrar la información de entrada y notificar al componente padre de cualquier interacción del usuario como, por ejemplo, hacer click en un botón. El *Smart Component* actúa como padre del mencionado anteriormente y maneja los datos recibidos.

De esta forma, se han ido creado los distintos componentes. En primer lugar, en el módulo *Shared* se encuentran los *Dumb Components*, y estos se utilizan en los componentes que se

han creado en el módulo *Features*, que contendrán toda la lógica y manipulación de datos de la aplicación.

Durante el desarrollo de la aplicación surgieron nuevos componentes que no se habían tenido en cuenta durante la fase de bocetaje. En primer lugar se añadieron en todas las escenas un header que permite salir del nivel independientemente de la escena en la que se encuentre el usuario. Asimismo, el mismo modal que se emplea para mostrar la información extra del nivel, se ha parametrizado de forma que sirva como modal para confirmar salir del nivel.

Como se ha podido ver en el boceto, el cambio entre niveles no se notificaba de ninguna forma, y, puesto que todas las escenas se usan indistintamente sin llevar un orden en concreto y tan solo teniendo en cuenta que un nivel se inicia con una escena de Introducción, por lo que durante el desarrollo del proyecto se consideró añadir un nuevo componente que sirviese como Resumen del Nivel que se acababa de completar. De esta forma surgió una nueva pantalla, que se muestra al final de cada nivel que refleja el progreso del juego mostrando los logros que se han obtenido.

Otro cambio que se llevó a cabo durante la fase de desarrollo del juego fue el nivel de persistencia que se hacía sobre el estado del jugador sobre la escena y nivel en el que se encontraba. En un principio se pretendía guardar la finalización del nivel y de escenas al terminar el nivel. De esta forma, cualquier recarga en la página suponía una recarga del nivel que se traducía en una perdida de todas las escenas completadas del nivel que estaba jugando el jugador.

Durante el desarrollo, se planteó la posibilidad de guardar en el *LocalStorage* las escenas que se iban completando evitando perder las completadas del nivel si se recargaba la página. Teniendo en cuenta esto, al final, las escenas se van guardando progresivamente y, sólo si el jugador decide abandonar el nivel, las escenas de ese nivel se eliminarán, teniendo que empezar el nivel desde el principio la próxima vez que lo retome. De esta forma, se mantiene un estado dentro de las escenas del nivel permitiendo al usuario algo más de libertad.

Cuando se abordó el requisito no funcional definido en la Tabla 3.10 relacionado con el diseño de la interfaz, se realizó un cambio significativo en toda la interfaz, que se podrá ver en las capturas de pantalla que se encuentran en la Sección A.3 del Anexo A.

7.4. Licencia

La licencia que presenta el código es *Attribution-NonCommercial-ShareAlike 4.0 International* que permite adaptar y compartir el material siempre y cuando no sea para un fin comercial, se mantenga la licencia y se le atribuya el crédito correspondiente.

Esta licencia se puede encontrar, bajo el nombre de LICENSE, en el directorio raíz, en el cual se explica ampliamente en qué consiste. Además, en la Figura 7.4, se puede ver qué licencia tiene el proyecto de un vistazo rápido en el recuadro subrayado en amarillo.

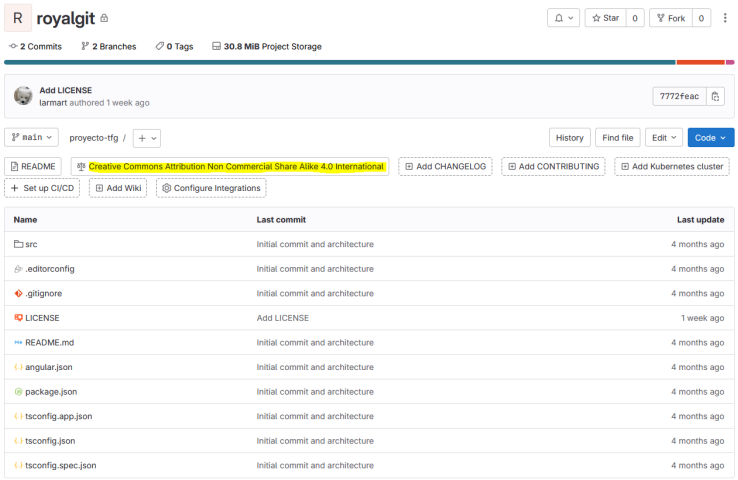


Figura 7.4: Licencia en el repositorio

7.5. Plan de pruebas

7.5.1. Casos de pruebas

Bajo esta sección se agrupan los casos de pruebas que se han probado en la aplicación con el fin de comprobar si el comportamiento es el esperado, y, si no lo es, anotar el resultado obtenido y su solución.

En esta primera parte se definen todos los casos de prueba a los que se ha sometido la aplicación, seguido de las soluciones que se han llevado a cabo para su correcto funcionamiento para aquellos casos de los cuales no se ha obtenido el resultado esperado en una primera ejecución del plan de pruebas.

En la Tabla 7.1 se definen las pruebas llevadas a cabo para el Menú principal de la aplicación.

En la Tabla 7.7 se tienen aquellas pruebas relacionadas con la pantalla de logros.

Para la Pantalla de Información, se tienen los casos de prueba relacionados con esta en la Tabla 7.2.

Los casos de prueba que se han realizado para la Pantalla Lista de Niveles se pueden ver en la Tabla 7.4.

En la Tabla 7.5 se ven los casos de prueba que se han ejecutado para las pantallas de Introducción, lo mismo sucede para las pantallas de contexto, descritos en la Tabla 7.6, opción, que se muestran en la Tabla 7.3 y flujo, que están en en la Tabla 7.8.

La Tabla 7.9 tiene todos los casos de prueba relativos a la pantalla Resumen del Nivel.

Y aquellos casos de pruebas que han sido comunes a varias pantallas se han definido en la Tabla 7.10.

Menú principal	
CP01	
Título	Inicio de la aplicación.
Descripción	Iniciar la aplicación en la ruta raíz.
Resultado esperado	Mostrar las diferentes entradas del menú.
CP02	
Título	Redireccionar a una entrada.
Descripción	Redireccionar a la entrada correspondiente al pulsar en cualquiera de las entradas disponibles.
Resultado esperado	Redireccionar a la url correspondiente y cargar los datos adecuados. Para la entrada de "Jugar" redireccionar al nivel actual que corresponde en cada momento.
CP03	
Título	Recargar el componente.
Descripción	Mantener un estado coherente tras recargar la página.
Resultado esperado	Se borran los datos relacionados con el nivel actual por si se llega de un estado inestable.

Tabla 7.1: Casos de prueba en el Menú principal

Información	
CP04	
Título	Inicio de la pantalla de Información
Descripción	Cargar el componente de Información.
Resultado obtenido	Acceder a los datos iniciales de la información a mostrar.
CP05	
Título	Pulsar en los botones del componente para ir hacia adelante o hacia atrás.
Descripción	Modificar los datos que se muestran en función de la página en la que se encuentre el usuario.
Resultado esperado	Mostrar el contenido de la página en la que se encuentre el usuario.
CP06	
Título	Recargar el componente.
Descripción	Mantener un estado coherente tras recargar la página.
Resultado esperado	Acceder a los datos iniciales de la información a mostrar.

Tabla 7.2: Casos de prueba en la Información

Escenas de Opción	
CP22	
Título	Inicio de la pantalla de Opción.
Descripción	Mostrar el texto correspondiente a la escena y mostrar las opciones disponibles de esa escena.
Resultado esperado	El texto asociado a la escena en la que se encuentra el usuario es el correcto y las opciones disponibles son las asociadas a esa escena.
CP23	
Título	Redirección a la escena siguiente.
Descripción	Tras pulsar en una de las opciones disponibles de la escena guardar los datos necesarios y redireccionar a la escena siguiente.
Resultado esperado	Al pulsar sobre una de las opciones mostradas se guarda la información de la opción escogida y se añade el logro si tiene uno asociado a la opción y se redirecciona a la escena siguiente.
CP24	
Título	Recargar el componente.
Descripción	Mantener un estado coherente tras recargar la página.
Resultado esperado	Se eliminan los datos sobre la escena eliminando la posible opción seleccionada y logro/s que se hayan podido obtener al elegir esa opción. El texto asociado a la escena en la que se encuentra el usuario es el correcto y las opciones disponibles son las asociadas a esa escena.

Tabla 7.3: Casos de prueba Escenas de Opción

Lista de niveles	
CP09	
Título	Inicio de la pantalla de Lista de niveles.
Descripción	Cargar el componente de Lista de niveles.
Resultado esperado	Mostrar la lista de niveles existentes en la aplicación.
CP10	
Título	Mostrar la información relativa a cada nivel.
Descripción	Mostrar la información del nivel al pulsar sobre este. Si el nivel es el nivel por el que se llega el usuario mostrar un botón que permita acceder a él.
Resultado esperado	Mostrar la información del nivel pulsado y a mayores mostrar un botón que permite dirigirte al inicio del nivel por el que se llega el usuario.
CP11	
Título	Listado de niveles tras terminar el juego.
Descripción	Mostrar la lista de niveles.
Resultado esperado	Mostrar la lista de niveles sin ningún nivel seleccionable ya que todos se han completado.
CP12	
Título	Recargar el componente.
Descripción	Mantener un estado coherente tras recargar la página.
Resultado esperado	Mostrar la lista de niveles existentes en la aplicación.

Tabla 7.4: Casos de prueba en Lista de niveles

Escenas de Introducción	
CP13	
Título	Inicio de la pantalla de Introducción.
Descripción	Mostrar el texto adecuado en función de la escena que se quiere mostrar.
Resultado esperado	El texto inicial asociado a la escena en la que se encuentra el usuario es el correcto.
CP14	
Título	Mostrar el texto a medida que se avanza en la escena.
Descripción	Mostrar el texto adecuado de la escena al ir pulsando sobre el texto que aparece en la pantalla.
Resultado esperado	El texto va cambiando cada vez que se pulsa en el texto que aparece.
CP15	
Título	Redirección a la escena siguiente.
Descripción	Tras acabar la escena redireccionar a la escena siguiente.
Resultado esperado	Al hacer click en el último texto de la escena redirecciona a la escena siguiente.
CP16	
Título	Recargar el componente.
Descripción	Mantener un estado coherente tras recargar la página.
Resultado esperado	El texto que es el inicial asociado a la escena en la que se encuentra el usuario es el correcto.

Tabla 7.5: Casos de prueba Escenas de Introducción

Escenas de Contexto	
CP17	
Título	Inicio de la pantalla de Contexto.
Descripción	Mostrar el texto adecuado en función de la escena que se quiere mostrar.
Resultado esperado	El texto inicial asociado a la escena en la que se encuentra el usuario es el correcto.
CP18	
Título	Mostrar el texto a medida que se avanza en la escena.
Descripción	Mostrar el texto adecuado de la escena al pulsar sobre el texto que aparece en la pantalla o sobre el botón con la flecha →.
Resultado esperado	El texto va cambiando cada vez que se pulsa en el texto que aparece o en el botón con la flecha →.
CP19	
Título	Mostrar el texto a medida que se retrocede en la escena.
Descripción	Mostrar el texto adecuado de la escena al pulsar sobre el botón con la flecha ←.
Resultado esperado	El texto cambia al texto que se mostró anteriormente cada vez que se pulsa en el botón con la flecha ←.
CP20	
Título	Redirección a la escena siguiente.
Descripción	Tras acabar la escena redireccionar a la escena siguiente.
Resultado esperado	Al hacer pulsar sobre el botón con la flecha → se redirecciona a la escena siguiente.
CP21	
Título	Recargar el componente.
Descripción	Mantener un estado coherente tras recargar la página.
Resultado esperado	El texto que se muestra que es el inicial asociado a la escena en la que se encuentra el usuario es el correcto.

Tabla 7.6: Casos de prueba Escenas de Contexto

Logros	
CP07	
Título	Inicio de la pantalla de logros.
Descripción	Cargar el componente de Logros.
Resultado esperado	Mostrar la lista de logros obtenibles diferenciando entre logros no obtenidos y logros obtenidos por el usuario.
CP08	
Título	Recargar el componente.
Descripción	Mantener un estado coherente tras recargar la página.
Resultado esperado	Mostrar la lista de logros obtenibles diferenciando entre logros no obtenidos y logros obtenidos por el usuario.

Tabla 7.7: Casos de prueba en los Logros

Escenas de Flujo	
CP25	
Título	Inicio de la pantalla de Flujo.
Descripción	Mostrar el flujo existente creado durante las escenas anteriores.
Resultado esperado	El flujo que se muestra es el que se ha ido creando en escenas anteriores.
CP26	
Título	Recibir un comando correcto.
Descripción	El usuario escribe en la caja de texto el comando que corresponde en ese momento en la escena.
Resultado esperado	El flujo de la escena se actualiza con el/los comando/s recibido/s por el usuario y se actualiza el texto mostrado en la historia del libro relacionado con el nivel.
CP27	
Título	Recibir comandos con opciones.
Descripción	Recibir uno o varios comandos con opciones.
Resultado esperado	Se actualiza el flujo con el/los comando/s recibido/s por el usuario y se actualiza el texto mostrado en la historia del libro relacionado con el nivel.
CP28	
Título	Recibir comandos personalizados.
Descripción	Recibir uno o varios con “commits” personalizados.
Resultado esperado	Se actualiza el flujo mostrando el mensaje personalizado en el flujo y se actualiza el texto mostrado en la historia del libro relacionado con el nivel.
CP29	
Título	Recibir un comando incorrecto escrito correctamente pero no corresponde con el comando que se requiere en el momento.
Descripción	Mostrar un cuadro informativo indicando que el comando existe pero no es el que pide en el momento.
Resultado esperado	Se muestra una notificación indicando que el comando existe pero no es el que pide en el momento.
CP30	
Título	Recibir un comando incorrecto.
Descripción	Mostrar un cuadro informativo indicando que el comando introducido no es correcto.
Resultado esperado	Se muestra una notificacion indicando que el comando no es correcto.
CP31	
Título	Recibir varios comandos separados por “; ”.
Descripción	Despachar los comandos de uno en uno dejando de actualizar el flujo cuando se recibe un comando incorrecto o actualizar el flujo correctamente si todos los comandos introducidos han sido correctos.
Resultado esperado	Se va actualizando el flujo hasta que se recibe un comando incorrecto o actualizar el flujo al recibir todos los comandos sin errores.
CP32	
Título	Redirección a la escena siguiente.
Descripción	Tras acabar los comandos que se solicitan en la escena, mostrar un botón para continuar. Al pulsar sobre el botón, guardar los datos necesarios y redireccionar a la escena siguiente.
Resultado esperado	Tras acabar la escena se habilita un botón que al pulsarlo guarda los datos relacionados con esa escena y redicreciona a la escena siguiente.
CP33	
Título	Recargar el componente.
Descripción	Mantener un estado coherente tras recargar la página.
Resultado esperado	Los comandos que se han introducido durante la escena se eliminan y el flujo que se muestra es el que se ha ido creando en escenas anteriores. El texto que se refleja en el libro con la historia asociado a esa escena se elimina.

Tabla 7.8: Casos de prueba Escenas de Flujo

Escenas de Resumen	
CP34	
Título	Inicio de la pantalla de Resumen.
Descripción	Mostrar los logros obtenidos en el nivel.
Resultado esperado	Los logros mostrados en el resumen corresponden a los obtenidos en el nivel.
CP35	
Título	Redirección al menú principal.
Descripción	Al pulsar sobre el icono x o sobre el texto “Menú principal” guardar los datos pertenecientes al nivel, actualizar el nivel actual y redireccionar al menú principal.
Resultado esperado	Los datos se guardan correctamente y se redirecciona al menú principal.
CP36	
Título	Redirección a la escena siguiente.
Descripción	Al pulsar sobre la flecha >guardar los datos pertenecientes al nivel, actualizar el nivel actual y redireccionar al siguiente nivel.
Resultado esperado	Los datos se guardan correctamente y se redirecciona a la primera escena del siguiente nivel.
CP37	
Título	Recargar el componente.
Descripción	Mantener un estado coherente tras recargar la página.
Resultado esperado	Los comandos que se han introducido durante la escena se eliminan y el flujo que se muestra es el que se ha ido creando en escenas anteriores. Los logros mostrados en el resumen corresponden a los obtenidos en el nivel.

Tabla 7.9: Casos de prueba Escenas de Resumen

Común a las escenas	
CP38	
Título	Redirección al resumen.
Descripción	La escena, que es la última asociada a ese nivel, redirecciona a la pantalla resumen guardando los logros obtenidos al acabar el nivel.
Resultado esperado	Se redirecciona a la escena Resumen del nivel correspondiente tras guardar los logros asociados al nivel.
CP39	
Título	Pulsar sobre el icono de Información
Descripción	Al pulsar sobre el icono de Información se muestra la información relativa al nivel.
Resultado esperado	Al pulsar sobre ese icono se muestra la información relativa al nivel.
CP40	
Título	Salir del modal de Información
Descripción	Al pulsar sobre el botón "Cerrar" o pulsar la tecla ESC la pantalla de información se oculta.
Resultado esperado	Al pulsar sobre el botón "Cerrar" o pulsar la tecla ESC la pantalla de información se oculta y el estado de la escena se mantiene.
CP41	
Título	Redirección al menú principal.
Descripción	Al pulsar sobre el icono x eliminar los datos pertenecientes al nivel y redireccionar al menú principal.
Resultado esperado	Los datos se borran correctamente y se redirecciona al menú principal.

Tabla 7.10: Casos de prueba comunes a escenas

7.5.2. Resultado de las pruebas

En esta sección se reflejan los resultados obtenidos de las pruebas que se han realizado, los posibles fallos que han podido surgir y la solución que se ha aplicado para estos. Estas pruebas se han realizado en los navegadores Firefox y Google Chrome. En la Tabla 7.11 se muestran aquellos casos de prueba que no tuvieron el resultado esperado en la primera ejecución del plan de pruebas y la solución aplicada para corregirlos.

Caso de prueba	Resultado
CP11 7.4	Fallo. Da error y no renderiza la lista de niveles.
CP16 7.5	Fallo. Se borran los datos del juego relacionados con el nivel actual que se está jugando.
CP24 7.3	Fallo. No se obtiene la opción seleccionada si el componente ya ha sido visitado, y por tanto, se ha seleccionado una opción y no elimina el logro asociado a esa opción.

Tabla 7.11: Resultado de los casos de prueba fallidos en una primera ejecución del plan de pruebas

CP11 Para solucionar el problema se vio que el componente no podía cargar esa lista de niveles porque trataba de acceder a los datos alojados del nivel actual, para indicar en

qué nivel se encontraba el jugador, cuando al acabar el juego, el nivel actual no existe, por lo que se tuvo en cuenta y se añadió la condición necesaria para que el componente no diera error al acceder a una variable que no tenía datos.

CP16 Para solucionar ese borrado de datos al recargar el componente, se tuvo en cuenta durante la carga del componente recoger los datos del "LocalStorage", para recoger los datos que hubiera guardados relativos al nivel y no sobrescribir esos datos con la carga inicial de la variable donde se almacena, la cual tiene datos y estructuras sin contenido.

CP24 En este caso, al cargar el componente, se tiene en cuenta si hay una opción ya seleccionada ya que al componente se puede acceder recargando la página, o utilizando los botones del propio navegador. Si esta escena ya tiene guardada una opción, porque ya se ha visitado anteriormente, es necesario eliminarla para evitar un estado incongruente como por ejemplo, tener para la misma escena dos opciones distintas seleccionadas. Inicialmente, la forma en la que se estaba obteniendo la posible opción seleccionada daba error en el código y era erróneo, por lo que se cambió el método para obtener, de forma positiva, la posible opción.

Una vez solucionados estos fallos, la posterior ejecución del plan de pruebas resultó en que todos los casos obtienen el resultado esperado.

Capítulo 8

Seguimiento del proyecto

8.1. Sprints realizados

8.1.1. Sprint 0 (28/08/2023 - 03/11/2023)

Este sprint 0 abarca las semanas dedicadas a tareas relacionadas con la planificación y documentación inicial antes de dar paso a los sprints de desarrollo. Ya que el proyecto se basa en un juego, también se ha necesitado desarrollar la historia y las mecánicas del juego. Este sprint también abarca toda la configuración de entornos de desarrollo y tecnologías necesarias para la futura implementación del proyecto. Dado que el llamado sprint 0 es especial, y previo a los sprints de desarrollo, no se aplica la restricción de duración homogénea como en los sprints de desarrollo.

Tarea	**Tiempo dedicado**
1.2 Motivación	2h
1.4 Sistemas de ayuda y 1.6 Alternativas del proyecto	5h 45m
1.5 Gamificación	2h 30m
3.1 Adaptación de SCRUM al proyecto	4h
3.3 Calendarización del proyecto	2h 30m
3.4 Product Backlog inicial	2h 15m
3.6 Planificación de riesgos	3h
3.7 Presupuesto	1h 25m
2 GDD	16h 30m
5 Tecnologías utilizadas	3h 20m
Configuración de entornos	1h 30m
8.1.1 Seguimiento del sprint 0	0h 25m
Total	**45h**

Tabla 8.1: Tiempo dedicado a cada tarea en el Sprint 0

8.1.2. Sprint 1 (06/11/2023 - 19/11/2023)

Tras toda la información y organización recogida en el Sprint 0, en este sprint se comienza a desarrollar el juego. Durante la reunión de final del sprint anterior, se establecieron las historias de usuario que se abordarían en este sprint, así como empezar a describir la arquitectura que se iba a implementar.

En la siguiente tabla se reflejan las historias que se habían planificado y subrayadas en verde si se han completado, en naranja si están en progreso y se deberán completar en sprints posteriores, o en rojo si no se han podido comenzar. Este código de color se verá reflejado en todas las tablas de seguimiento de ahora en adelante.

Historia	**Tiempo dedicado**	**Tiempo estimado**
3.6 HU61	8h 12m	10h
3.6 HU64	5h 30m	10h
3.5 HU01	1h 30m	10h
Tarea	**Tiempo dedicado**	
6 Definir arquitectura	15h 22m	
8.1.2 Seguimiento del sprint 1	1h	
Análisis de clases	0h 40m	
Reuniones de seguimiento y Sprint Planning	1h 15m	
Total	**33h 29m**	

Tabla 8.2: Tiempo dedicado a cada tarea en el Sprint 1

Como se refleja en la tabla, no se han podido completar todas las historias que se habían planificado ya que en la reunión de final de sprint consideramos hacer un cambio en cuanto al enfoque que queríamos darle a la historia HU61, que se corresponde a mostrar una lista con todos los niveles. Debido a este cambio, la historia HU01, dedicada a la introducción del primer nivel, tampoco se pudo completar, ya que los cambios aplicados para la historia HU61 modificaban el concepto inicial que se tenía con esta historia y no daba tiempo a terminarlo en lo que quedaba de sprint.

En este sprint se ha iniciado el proyecto sin complicaciones, pero a medida que se iban añadiendo funcionalidades, han surgido dudas sobre la arquitectura que se estaba siguiendo. Sin embargo, tras dedicarle unas horas a ver todas las alternativas se ha escogido una arquitectura y estructura de paquetes que en principio puede ser buena. No se descarta más adelante, según vaya evolucionando el desarrollo del sistema, cambiar ligeramente la arquitectura establecida.

8.1.3. Sprint 2 (20/11/2023 - 03/12/2023)

En el Sprint Planning se decide las historias que se quieren abordar. Ya que en el sprint anterior se dejaron incompletas las historias HU61 y HU01, en este sprint se ha fijado como

objetivo terminarlas. Dentro de la finalización de estas dos historias, está implícito desarrollar una tercera historia, HU09, que define cómo se va a realizar la transición entre escenas. Ya que se iba a desarrollar el cambio entre escenas, también se quiso desarrollar la siguiente escena, solo la parte visual de la misma, que corresponde a iniciar la HU02. Al final, se pudo completar esa historia, tanto la parte visual como la lógica.

En la Tabla 8.3 se reflejan las historias de usuario definidas en el Sprint Planning junto con tareas definidas relacionadas con la documentación del proyecto.

Historia	**Tiempo dedicado**	**Tiempo estimado**
3.6 HU61	6h 15m	10h
3.7 HU01	8h	10h
3.7 HU02	7h 30m	5h
3.7 HU09	9h	10h
Tarea	**Tiempo dedicado**	
8.1.3 Seguimiento del sprint 2	0h 30m	
Reuniones de seguimiento y Sprint Planning	1h 10m	
Total	**25h 25m**	

Tabla 8.3: Tiempo dedicado a cada tarea en el Sprint 2

La HU61 era una historia que se había movido del sprint anterior a este, por lo que su carga de trabajo ha sido menor y por tanto las horas empleadas en modificar lo realizado para ajustarlo a lo que se comentó en el Sprint review y restrospective anterior, se han visto reducidas. En este sprint también se ha hecho hincapié en definir y aterrizar la forma en la que se iban a guardar y manejar las escenas, ya que para poder desarrollar la HU09 era necesario tener en cuenta cómo iba a ser la interacción entre escenas. Como es una historia que abarca el nivel que se está desarrollando, ya que definir y desarrollar los saltos entre escenas es algo que no se va a terminar hasta que termine el proceso de desarrollo del Nivel 0, estará en proceso hasta que se cierre el nivel, algo que ocurrirá cuando finalice la historia HU10.

En este sprint lo más destacable ha sido definir qué contiene una escena y cómo interactúa con la siguiente escena, ya que había que tener en cuenta el estado en el que se encontraba el jugador y en un futuro, también se deberá tener en cuenta más parámetros, por lo que poder definirlo correctamente en este sprint ha sido lo más complicado y probablemente por eso haya llevado más tiempo de lo esperado.

8.1.4. Sprint 3 (04/12/2023 - 17/12/2023)

En el Sprint Planning llevado a cabo durante la reunión de final de sprint, 01/12 se decide dentro de todas las historias de usuario definidas en el Product Backlog las que se van a desarrollar en este sprint. Se ha optado por crear los componentes necesarios para poder realizar las escenas de opción de los niveles posteriores, así como el modal de información

que aparece en cada nivel.

Historia	**Tiempo dedicado**	**Tiempo estimado**
3.7 HU03	4h	5h
3.7 HU04	9h	15h
3.7 HU08	5h	5h
3.7 HU09	5h 45m	5h
Tarea	**Tiempo dedicado**	
3.10 HU71	4h 30m	5h
8.1.4 Seguimiento del sprint 3	0h 40m	
Reuniones de seguimiento y Sprint Planning	1h	
Total	**30h 55m**	

Tabla 8.4: Tiempo dedicado a cada tarea en el Sprint 3

Durante este sprint, la reunión semanal fue festivo, por lo que tras hablarlo en el Sprint planning, se tuvo en cuenta que en un principio no iba a realizarse. Debido a esto, tras empezar a desarrollar las historias de usuario, me di cuenta que asigné una historia que tenía como definición desarrollar la escena de opción del nivel era incorrecta y tras su reformulación, HU04, el peso en horas iba a ser menor que el inicialmente supuesto. En cuanto a las demás historias, no surgió ningún problema y se realizaron en el tiempo indicado. Cabe destacar que en este sprint, ha habido unos cuantos días festivos, y por lo dicho anteriormente, el total de horas es significativamente menor a los demás sprints. La Tabla 8.4 resume el plan y la ejecución del sprint.

8.1.5. Sprint 4 (18/12/2023 - 31/12/2023)

En el Sprint Planning que se tuvo el día 15/12 se establecieron las historias de usuario a realizar durante el Sprint. En un principio, se asignaron las HU05, que corresponde a desarrollar las escenas de flujo del nivel, la HU10, que surge de desglosar la historia general de niveles, y es la que se encarga de guardar el estado al acabar un nivel y pasar al siguiente, y la HU12 que es el desarrollo de la introducción del Nivel 1.

Historia	**Tiempo dedicado**	**Tiempo estimado**
3.6 HU62	1h	5h
3.7 HU05	5h	5h
3.7 HU06	10h	5h
3.7 HU07	6h	5h
3.7 HU10	7h 30m	5h
3.7 HU11	3h	5h
3.7 HU12	1h	5h
Tarea	**Tiempo dedicado**	
8.1.5 Seguimiento del sprint 4	50m	
Reuniones de seguimiento y Sprint Planning	1h	
Total	**32h 20m**	

Tabla 8.5: Tiempo dedicado a cada tarea en el Sprint 4

Tras la reunión semanal del sprint, nos dimos cuenta que las tres HU definidas para el sprint se quedaban cortas ya que se habían completado en la mitad de este, por lo que para la segunda mitad del sprint se definieron nuevas HU, que son la HU06, para el guardado del flujo, HU07, para permitir introducir comandos con opciones, la HU11 dedicado a reflejar en la pantalla del menú principal de logros, los logros obtenidos tras acabar el nivel y el desarrollo de la pantalla de Información adicional.

La HU06 fue más complicada de lo previsto, ya que había que tener en cuenta varias cosas y se tuvo que hacer cambios significativos en el código, el tiempo empleado en terminarla no pudo ser empleado en desarrollar las 2 historias que no se han podido acabar y quedarán para el siguiente sprint.

En este sprint ha habido un desequilibro de trabajo, ya que durante la primera mitad del sprint, la carga ha sido considerablemente menor a la segunda mitad, la cual ha estado más completa, no pudiendo abarcar todas las historias. En base a este sprint, se puede aproximar que historias de usuario serán más complicadas de implementar y cuales serán más fáciles, y tendrán un peso menor.

Sin embargo, ya que todas estas historias tienen un comportamiento similar en los niveles siguientes, también se puede estimar que la duración será menor ya que mucho código se reutilizará en las siguiente historias.

8.1.6. Sprint 5 (01/01/2024 - 14/01/2024)

El Sprint Planning no se tuvo como reunión sino de forma asíncrona, debido a los festivos y vacaciones, de forma que lo definido ha sido escrito tras llegar a una conclusión el equipo de desarrollo y el PO.

En este sprint se quieren terminar las dos historias que no se pudieron acabar el anterior

sprint y desarrollar la HU13, que se refiere a desarrollar las escenas de contexto así como empezar a tener la parte visual de las escenas de opción, desarrollado en la HU14.

Además, los cambios de escenas se desarrollarán en la HU18y que se refiere al manejo los datos entre escena y escena.

Historia	**Tiempo dedicado**	**Tiempo estimado**
3.6 HU62	4h	5h
3.7 HU07	10h	5h
3.7 HU13	5h	5h
3.7 HU14	10h	10h
3.7 HU18	1h	5h
Tarea	**Tiempo dedicado**	
8.1.6 Seguimiento del sprint 5	1h	
Reuniones de seguimiento y Sprint Planning	1h	
Total	**29h**	

Tabla 8.6: Tiempo dedicado a cada tarea en el Sprint 5

En este sprint se ha empezado a definir lo necesario para el Nivel 1. Como se refleja en la tabla, la historia relacionada con los cambios entre escenas no se ha podido completar debido a que la HU07 ocupó bastante más tiempo del pensado.

En cuanto a la historia HU62, ya que no está relacionada intrínsecamente con los niveles, puede desarrollarse de forma paralela en cualquier momento y, por ende, no está bloqueada por ninguna historia. Por eso, sirve como historias que se pueden quedar en el backlog y si hubiese algún sprint con menos carga, podría pensarse como una historia abarcable en ese sprint.

Las demás historias se han podido completar con éxito dejando para el siguiente sprint finalizar las asociadas al Nivel 1.

8.1.7. Sprint 6 (15/01/2024 - 28/01/2024)

Durante el Sprint planning del día 12/01, tras el sprint anterior sin reuniones, se pudo poner en común lo realizado hasta el momento. Tras ver el progreso, se definieron las historias que se deberían completar, teniendo en cuenta que los niveles tienen una curva exponencial y por tanto debería reducirse el tiempo que se emplea desarrollando cada nivel.

Por ello para este sprint se ha decidido terminar todo lo relacionado con el Nivel 1 y completar el Nivel 2.

Historia	**Tiempo dedicado**	**Tiempo estimado**
3.7 HU15	2h	5h
3.7 HU16	2h	5h
3.7 HU17	2h	5h
3.7 HU18	1h 30m	2h
3.7 HU19	2h	2h
3.8 HU21	1h 45m	2h
3.8 HU22	2h	2h
3.8 HU23	2h	2h
3.8 HU24	2h	2h
3.8 HU25	2h	2h
3.8 HU26	2h	2h
3.8 HU27	1h 30m	2h
Tarea	**Tiempo dedicado**	
8.1.7 Seguimiento del sprint 6	1h	
Reuniones de seguimiento y Sprint Planning	0h 40m	
Total	**27h 25m**	

Tabla 8.7: Tiempo dedicado a cada tarea en el Sprint 6

Durante este sprint se quiso terminar todo el nivel 1 y nivel 2 ya que, al poder reutilizar componentes el trabajo de los niveles se vería reducido. Gracias a esto, se completaron ambos niveles, relacionando ya cada nivel con los logros que se iban obteniendo en un menor tiempo del que se había pensado en un principio.

En el sprint planning se comentó la idea de jugar al juego en el sprint retrospective y poder leer los textos más pausadamente, por lo que también se le dedicó bastante tiempo a los textos que se mostraban en cada escena para comprobar que las ideas e información que se iban mostrando cobraban sentido y ayudaban a lograr el objetivo de cada nivel.

8.1.8. Sprint 7 (29/01/2024 - 11/02/2024)

En el Sprint Planning del día 26/01/2024 se estuvo viendo el progreso de la aplicación, jugando los niveles que se crearon hasta el momento, que correspondían a los niveles 0, 1 y 2. De esta pequeña demo se pudieron sacar ciertas mejoras que se dejaron para implementar en este sprint. Estas mejoras corresponden a las historias de usuario 11, 20, 28, 35 y 42 en las cuales se define una nueva pantalla al finalizar cada nivel para mostrar al usuario los logros obtenidos en ese nivel e identificar de una forma más cómoda el final de cada nivel.

Además se estuvo hablando de mejoras generales en el documento y cambios relacionados con el *look and feel* de la aplicación.

Historia	Tiempo dedicado	Tiempo estimado
3.7 HU11	2h	2h
3.7 HU20	0h 15m	0h 30m
3.8 HU28	0h 15m	0h 30m
3.8 HU29	3h	1h
3.8 HU30	3h	2h
3.8 HU31	3h	5h
3.8 HU32	2h	2h
3.8 HU33	2h	2h
3.8 HU34	2h 30m	2h
3.8 HU35	2h	1h
3.9 HU36	2h 30m	2h
3.9 HU37	0h 30m	2h
3.9 HU38	0h 30m	2h
3.9 HU39	1h	2h
3.9 HU40	1h	2h
3.9 HU41	1h	2h
3.9 HU42	1h	2h
3.8 HU49	2h	2h
Tarea	**Tiempo dedicado**	
8.1.8 Seguimiento del sprint 7	1h	
Reuniones de seguimiento y Sprint Planning	1h	
Total	**32h 30m**	

Tabla 8.8: Tiempo dedicado a cada tarea en el Sprint 7

En un principio el Sprint abarcaba los cambios comentados en la reunión y el Nivel 3, pero durante el transcurso del sprint el peso de la creación del nivel fue bastante más bajo al estimado, por lo que tomó la decisión de avanzar en los niveles y crear el siguiente.

Además, durante este período se estuvo mejorando la interfaz, añadiendo imágenes características para los logros, los *banners* para cada nivel y la estructura de la información extra, a la que se puede acceder mediante el menú principal.

8.1.9. Sprint 8 (12/02/2024 - 25/02/2024)

En la reunión de este Sprint se estuvo hablando sobre finalizar las funcionalidades, terminando el Nivel 5, y dedicarlo a mayores a correcciones y mejoras de lo que ya estaba implementado. Por lo que en la primera semana del sprint nos hemos centrado en terminar el Nivel 5 y durante la segunda semana mejoras sobre el código y el documento.

Historia	**Tiempo dedicado**	**Tiempo estimado**
3.9 HU43	2h 30m	2h
3.9 HU44	2h	2h
3.9 HU45	2h 30m	2h
3.9 HU46	2h 30m	1h
3.9 HU47	2h 30m	2h
3.9 HU48	4h	2h
3.9 HU49	4h 30m	2h
Tarea	**Tiempo dedicado**	
3.10 HU71	5h	5h
8.1.9 Seguimiento del sprint 8	1h	
Reuniones de seguimiento y Sprint Planning	1h 30m	
Total	**33h 30m**	

Tabla 8.9: Tiempo dedicado a cada tarea en el Sprint 8

Además, en esta reunión se comentó la posibilidad de poder desplegar el juego en una máquina virtual de la Universidad, y el poder probarlo para un pequeño grupo de estudiantes el juego, sin necesidad de tener que cambiar nada y simplemente para tener retroalimentación sobre el proyecto y sobre si el objetivo principal de la aplicación se logra.

Durante el desarrollo del sprint, no hubo complicaciones, e incluso se cambió la forma en la que se persistían los datos, para mejorar la experiencia del jugador durante los niveles, este cambio se describe más ampliamente en la Seccion 7.3.

En la reunión semanal, y debido a la finalización del último nivel, se empezó el Capítulo de Documentación y pruebas, con el fin de comprobar que el flujo del juego era correcto y corregir aquellos fallos que pudieran darse.

8.1.10. Sprint 9 (26/02/2024 - 10/03/2024)

Este Sprint se empleó íntegramente a la documentación del proyecto, haciendo hincapié en la estructura del proyecto, diagramas y secciones que faltaban por completar, o que no podían completarse hasta este momento.

Tarea	**Tiempo dedicado**	
3.10 HU71	27h	
8.1.10 Seguimiento del sprint 9	1h	
Reuniones de seguimiento y Sprint Planning	0h 50m	
Total	**28h 50m**	

Tabla 8.10: Tiempo dedicado a cada tarea en el Sprint 9

Durante el desarrollo de Sprint se estuvo trabajando en la redacción de la memoria del proyecto, figuras necesarias para el análisis y diseño y lectura del documento para corregir fallos o añadir partes que faltaban. La reunión de mitad de Sprint que se tuvo, fue una reunión de seguimiento para comprobar el progreso en las tareas realizadas e indicaciones para ciertas secciones en las que no se sabía como continuar.

Durante esta segunda semana se siguió trabajando en la memoria del proyecto, terminando la misma.

8.2. Resumen de la ejecución del proyecto

Para finalizar este capítulo de Seguimiento se va a proceder a comparar la planificación inicial que se había determinado al inicio del proyecto con la ejecución real del proyecto una vez finalizado. En esta sección se va a ver el tiempo empleado en el proyecto, así como ese cambio en costes en función de los datos reales del tiempo que ha llevado finalizar el proyecto.

8.2.1. Tiempo estimado y tiempo real

En la Sección 3.3 se explicaba la organización y tiempos que se iban a seguir en un principio en el proyecto. Como horas semanales se habían estimado 35h, sin embargo, durante la realización de los sprints se ha visto que la media de horas es de 30h y no 35h como se había estimado. Haciendo un cálculo, los sprints que se deben realizar para poder completar las 300h son unos 8/9 sprints quitando las horas del Sprint 0. Teniendo en cuenta eso, al final, se han realizado 9 sprints que entran dentro de la planificación descrita.

La finalización del proyecto se ha dado el día 10/03/2024, que coincide con el final del segundo Sprint de refuerzo, y por tanto, la planificación, aunque no se ha seguido la organización inicial, el final del proyecto estaba dentro de los límites previstos para el mismo, a partir del análisis de riesgos.

Durante los primeros sprints la velocidad ha sido menor, ya que se debían organizar y crear los componentes desde el principio, y ya durante los Sprints 7 y 8, completar las historias sucedían de forma más fácil y podían completarse mucho más rápido. Gracias a la organización que supone seguir este marco de trabajo Scrum, se han podido ajustar los sprints para completar de forma positiva el proyecto sin necesidad de agregar nuevos sprints. En este caso, estamos contando los sprints de refuerzo que han servido para justamente esto, tener cierto tiempo previsto de tal forma que no afecte demasiado al desarrollo del proyecto.

Al final las horas que se han realizado son 322h, que cumple con el rango entre 300 y 335h, establecido en la planificación inicial, teniendo en cuenta que la conversión de sprints a horas no encajaban.

8.2.2. Costes estimados y costes reales

Puesto que el coste estimado en un principio se había establecido para el trabajo de 4 meses, y al final este se ha alargado a unos 5.25 meses, el presupuesto simulado, siguiendo la fórmula definida en la sección del Presupuesto 3.7, cambia como se muestra en la Tabla 8.11. En este caso, el gasto en lo relacionado con los costes simulados ha aumentado un 15 % lo que supone un aumento considerable por haber finalizado un 1.25 meses después. Sin embargo, este porcentaje ha sido menor que el que se ha dado con los costes reales, reflejado en la FIgura 8.12, que ha aumentado un 24 %, ya que tiene muchos gastos que son fijos y no van en función de un % de amortización como lo es la parte hardware.

Concepto	**Precio**	**Cantidad**	**Total**
Hora de trabajo de desarrollador software	27,86 €/hora	322h horas	8970,92 €
Seguridad Social	754,06 €/mes	5,25 meses	3958,82 €
Electricidad	374,4 €/mes	5,25 meses	1965,6 €
Portátil ASUS	13,33 €/mes	5,25 meses	70 €
Ratón Logitech Pro Superlight	1,95 €/mes	5,25 meses	10,24 €
Total			14975,58 €

Tabla 8.11: Estimación de costes simulado final

Para el cálculo real de los costes, se eliminan las horas de desarrollador, al igual que en la sección de presupuestos inicial (Sección 3.7).

Concepto	**Precio**	**Cantidad**	**Total**
Portátil ASUS	17,33 €/mes	5,25 meses	91 €
Ratón Logitech Pro Superlight	2,54 €/mes	5,25 meses	13,36 €
Electricidad	60,26 €/mes	5,25 meses	316,37 €
Internet	31,90 €/mes	5,25 meses	167,48 €
Total			588,21 €

Tabla 8.12: Estimación de costes real final

Capítulo 9

Conclusiones

Tras la finalización del proyecto puedo concluir que el aprendizaje obtenido durante los meses que ha abarcado dicho proyecto ha sido enriquecedor, ya que me ha permitido conocer más sobre un desarrollo completo abarcando desde las primeras fases de análisis y diseño hasta su desarrollo e implementación. Al seguir un desarrollo ágil, la organización que se ha tenido durante todo el proyecto ha sido estructurada y casa muy bien con la forma en la que yo suelo trabajar por lo que no me ha resultado complicado ajustarme y de hecho, ha mejorado mi capacidad de organización.

Destaco también todo el análisis previo que se ha tenido que hacer para el documento de juego, del cual he aprendido la estructura que debe seguir y el contenido que existe dentro de este.

Al realizar un juego para el aprendizaje de Git con Gitflow, también he profundizado en el flujo Gitflow, que es la idea principal y el origen del proyecto. Es un método que si bien se emplea cuando varios desarrolladores trabajan paralelamente, también puede ajustarse a un trabajo individual ya que este método hace hincapié en organizar las funcionalidades del código y, adicionalmente, ayuda al trabajo simultáneo.

Además, en cuanto a tecnologías, se ha empleado una, de la que tenía cierto conocimiento previo, pero tenía como objetivo conocer más en profundidad. Este objetivo formativo también se ha cumplido, adquiriendo experiencia y habilidades desde iniciar un proyecto Angular desde cero, hasta establecer una estructura de forma que se pueda extender sin necesidad de rehacer código.

En cuanto a los objetivos generales del proyecto que se habían propuesto se han logrado todos:

1. *Diseñar un juego orientado al aprendizaje del método Gitflow.* Este objetivo, es el principal ya que es el motivo por el cual este proyecto existe.
2. *Creación de temáticas y mecánicas que motiven a los jugadores a participar.* La mecáni-

ca que se ha empleado es la colección de logros por lo que este objetivo está completado. Pese a que se comentó la idea de tener otros métodos que motivasen a los jugadores, finalmente no se incluyeron en el proyecto.

3. *Creación de niveles en el que se expliquen los conceptos necesarios.* Los niveles que se habían definido en un principio se han podido completar, terminando un flujo de Gitflow explicando las ramas correspondientes. En un futuro se puede explicar la rama *Bugfix* que desde el inicio de la definición del alcance del juego no se iba a desarrollar.

4. *Reflejar logros obtenidos en el juego con el fin de motivar a los jugadores terminar los niveles para completarlos.* Este objetivo se ha completado ya que para cada nivel se muestran los logros que se han completado y se ha desarrollado una página dedicada a mostrar la lista de logros obtenibles y obtenidos.

5. *Diseñar e implementar una plataforma web para la realización del juego.* Todo el desarrollo se ha realizado para la web, es completamente funcional y ha sido probado para los navegadores Firefox y Google Chrome.

9.1. Líneas de trabajo futuras

La línea de trabajo futura principal es diseñar y ejecutar un plan de evaluación de la eficacia y la eficiencia en el aprendizaje del juego y una evaluación completa de usabilidad.

Ciertas mejoras cuya idea ha surgido durante el desarrollo del proyecto, y que podrían tenerse en cuenta en un futuro, se resumen en los siguientes puntos:

1. Desarrollar más niveles.

2. Guardar los datos del estado del juego en un servidor para poder acceder desde cualquier navegador.

3. Añadir alguna opción nueva para medir la puntuación como un ranking global.

4. Añadir la rama *bugfix* como nueva rama a explicar e integrarla en el método Gitflow.

Bibliografía

[1] Agencia tributaria. Tabla de amortización simplificada. https://sede.agenciatributaria.gob.es/Sede/ayuda/manuales-videos-folletos/manuales-practicos/folleto-actividades-economicas/3-impuesto-sobre-renta-personas-fisicas/3_5-estimacion-directa-simplificada/3_5_4-tabla-amortizacion-simplificada.html. Accessed: 2023-9-19.

[2] Agencia tributaria. Tabla de coeficientes de amortización lineal. https://sede.agenciatributaria.gob.es/Sede/ayuda/manuales-videos-folletos/manuales-practicos/irpf-2022/c07-rendimientos-actividades-economicas-estimacion-directa/fase-1-determinacion-rendimiento-neto/amortizaciones-dotaciones-ejercicio-fiscalmente-deducibles/requisitos-generales/coeficientes-amortizacion-lineal.html. Accessed: 2024-3-25.

[3] Atlassian. Bitbucket. https://bitbucket.org/. Accessed: 2023-9-9.

[4] Atlassian. Configuración de un repositorio. https://www.atlassian.com/es/git/tutorials/setting-up-a-repository. Accessed: 2024-3-20.

[5] Atlassian. Gitflow workflow. https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow. Accessed: 2023-10-13.

[6] Atlassian. Learn git with bitbucket cloud. https://www.atlassian.com/git/tutorials/learn-git-with-bitbucket-cloud. Accessed: 2023-9-9.

[7] Atlassian. Qué es scrum y cómo empezar. https://www.atlassian.com/es/agile/scrum. Accessed: 2023-10-30.

[8] Atlassian. What is git. https://www.atlassian.com/git/tutorials/what-is-git. Accessed: 2024-3-19.

[9] Atlassian. What is version control? https://www.atlassian.com/git/tutorials/what-is-version-control. Accessed: 2023-9-9.

[10] blinry, bleeptrack. Oh my git! an open source game about learning git! https://ohmygit.org/. Accessed: 2023-9-1.

[11] Cecilio Álvarez Caules. Angular services singletons o no? https://www.arquitecturajava.com/angular-services-singletons-o-no/. Accessed: 2024-3-4.

[12] Change Vision. Software design tool – astah professional –. https://astah.net/products/astah-professional/. Accessed: 2024-3-5.

[13] Daniel Benmergui. Storyteller. https://annapurnainteractive.com/en/games/storyteller. Accessed: 2024-3-14.

[14] Derek Davidson. Unlock the power in the five scrum events. https://www.scrum.org/resources/blog/unlock-power-five-scrum-events. Accessed: 2023-10-30.

[15] Emmanuel John. Angular modules: Best practices for structuring your app. https://blog.logrocket.com/angular-modules-best-practices-for-structuring-your-app/. Accessed: 2023-11-06.

[16] Esic. Html5: qué es, características y cómo funciona. https://www.esic.edu/rethink/tecnologia/html5-que-es-caracteristicas-y-como-funciona-c#:~:text=El%20HTML5%20es%20un%20est%C3%A1ndar,%2C%20estilo%20de%20letra%2C%20etc. Accessed: 2024-3-5.

[17] Freddy Vega. Utilizando Pull Requests en GitHub. https://platzi.com/clases/1557-git-github/19957-utilizando-pull-requests-en-github/. Accessed: 2024-3-20.

[18] Gestron. Cuánto se paga de seguros sociales por cada trabajador. https://ayudatpymes.com/gestron/cuanto-se-paga-de-seguros-sociales-por-cada-trabajador/#Cuanto_se_paga_de_seguro_social_por_un_trabajador. Accessed: 2024-3-20.

[19] GitLab. What is version control? https://about.gitlab.com/topics/version-control/. Accessed: 2023-9-9.

[20] Hiren Doshi. The various levels of services in the 3 scrum roles. https://www.scrum.org/resources/blog/various-levels-services-3-scrum-roles. Accessed: 2023-10-30.

[21] Honeypot. Developer salary report. https://mcusercontent.com/1bde4eb1f3446b0674c53c3c3/files/fe0a9404-7b19-26f4-5468-a88ea12229a7/Developer_Salary_Report_2023.pdf?utm_medium=email&utm_source=report&utm_campaign=global_universal_salary_report_2023. Accessed: 2024-3-25.

[22] Hugo da Silva. ¿sabes diferenciar entre gamificación y aprendizaje basado en juego? https://blog.criteria.es/sabes-diferenciar-entre-gamificaci%C3%B3n-y-aprendizaje-basado-en-juego. Accessed: 2023-8-28.

[23] Ivan Zhao. Notion. https://www.notion.so/es-es. Accessed: 2023-11-02.

[24] Javier Garzas. Elige una buena escala de puntos historia y te ahorrarás problemas. https://www.javiergarzas.com/2014/11/escala-de-puntos-historia.html. Accessed: 2023-10-30.

[25] Javier Lacort. Qué es notion y cómo usarla como herramienta de productividad. https://www.welcometothejungle.com/es/articles/notion-herramienta-productividad. Accessed: 2023-11-02.

[26] Jim Weirich. |one, step, back| . https://www.onestepback.org/. Accessed: 2023-9-9.

[27] Jim Weirich. Git immersion. https://git-immersion.github.io/gitimmersion/. Accessed: 2023-9-9.

[28] Jim Weirich. Git immersion labs. https://github.com/kEND/git_immersion/blob/master/pdfs/git_tutorial.pdf. Accessed: 2023-9-9.

[29] Jim Weirich. Source control made easy. https://web.archive.org/web/20140209133404/http://pragprog.com/screencasts/v-jwsceasy/source-control-made-easy. Accessed: 2023-9-9.

[30] Johannes Kettmann. Github minesweeper. https://profy.dev/project/github-minesweeper. Accessed: 2023-9-1.

[31] Karl M. Kapp. *The gamification of learning and instruction: Game-based methods and strategies for training and education*. Pfeiffer, 2012.

[32] KeepCoding Team. ¿Qué es la rama release en Git? . https://keepcoding.io/blog/que-es-la-rama-release-en-git/. Accessed: 2024-3-20.

[33] KeepCoding Team. ¿Qué es un Commit en Git? . https://keepcoding.io/blog/que-es-un-commit-en-git/. Accessed: 2024-3-20.

[34] KeepCoding Team. ¿Qué son las ramas en git y para qué sirven? https://keepcoding.io/blog/que-son-las-ramas-en-git/. Accessed: 2024-3-20.

[35] Knut Sveidqvist, et al. Mermaid. diagramming and charting tool. https://mermaid.js.org/. Accessed: 2024-3-5.

[36] LaTeX. Latex – a document preparation system. https://www.latex-project.org/. Accessed: 2023-11-02.

[37] Madhav Agarwal. The evolution and impact of the gaming industry: A comprehensive overview. https://www.linkedin.com/pulse/evolution-impact-gaming-industry-comprehensive-overview/. Accessed: 2024-2-25.

[38] Mary Iqbal. A deep dive into the purpose of each scrum artifact. https://www.scrum.org/resources/blog/deep-dive-purpose-each-scrum-artifact. Accessed: 2023-10-30.

[39] María Coppola. ¿qué es angular? características y ventajas. https://blog.hubspot.es/website/que-es-angular. Accessed: 2023-11-02.

[40] Matthew Podwysocki, et al. Rxjs. reactive extensions library for javascript. https://rxjs.dev/guide/overview. Accessed: 2024-3-5.

[41] Max Lynch and Ben Sperry. Ionic. a new way to build and ship for mobile. https://ionic.io/. Accessed: 2024-3-5.

[42] MDN. What is css? https://developer.mozilla.org/es/docs/Learn/CSS/First_steps/What_is_CSS. Accessed: 2023-11-02.

[43] Miguel Alejandro Esteban Ordoñez. Git flow: Tipos de ramas. https://openwebinars.net/blog/git-flow-tipos-de-ramas/. Accessed: 2024-3-20.

[44] mijacobs, et al. Qué es git. https://learn.microsoft.com/es-es/devops/develop/git/what-is-git. Accessed: 2023-9-9.

[45] Misko Hevery and Adam Abrons. Angular coding style guide. https://angular.io/guide/styleguide. Accessed: 2023-11-06.

[46] Misko Hevery and Adam Abrons. Using angular routes in a single-page application. https://angular.io/guide/router-tutorial. Accessed: 2023-11-06.

[47] nandbox. All you need to know about component-based architecture. https://nandbox.com/all-you-need-to-know-about-component-based-architecture/. Accessed: 2024-2-26.

[48] Nenad Milanović, et al. Clockify. https://clockify.me/es/. Accessed: 2024-3-5.

[49] ocu.org. Evolución del precio de la luz. https://www.ocu.org/vivienda-y-energia/gas-luz/informe/precio-luz. Accessed: 2024-3-20.

[50] Oscar Blancarte. Observer. https://reactiveprogramming.io/blog/es/patrones-de-diseno/observer. Accessed: 2024-3-4.

[51] Ossian Muscad. A guide to component-based architecture: Definition, relevance, benefits, & more. https://datamyte.com/blog/component-based-architecture/. Accessed: 2024-2-26.

[52] Palco23. Los esports crecen en españa y el sector del videojuego facturará 2.380 millones en 2022. https://www.palco23.com/competiciones/los-esports-crecen-en-espana-y-el-sector-del-videojuego-facturara-2380-millones-en-2022. Accessed: 2023-8-30.

[53] Perforce Software. What is a vcs? overview of version control software. https://www.perforce.com/blog/vcs/what-is-version-control. Accessed: 2023-9-9.

[54] Peter Cottle, et al. Learngitbranching. https://learngitbranching.js.org/. Accessed: 2023-9-1.

[55] Responsive Advisors. Intro to scrum (13 of 16): What are scrum artifacts? https://responsiveadvisors.com/blog/what-are-scrum-artifacts/. Accessed: 2023-10-30.

[56] Robert Kieffer and ctavan. uuid. https://www.npmjs.com/package/uuid. Accessed: 2024-3-5.

[57] soumya08. Version control systems. https://www.geeksforgeeks.org/version-control-systems/. Accessed: 2023-9-9.

[58] Spinify. Who started gamification? https://spinify.com/blog/gamification-history/. Accessed: 2023-8-28.

[59] Wikipedia. Angular (framework). https://es.wikipedia.org/wiki/Angular_(framework). Accessed: 2023-11-02.

[60] Wikipedia. Bitbucket. `https://es.wikipedia.org/wiki/Bitbucket`. Accessed: 2023-9-9.

[61] Wikipedia. Css. `https://es.wikipedia.org/wiki/CSS`. Accessed: 2023-11-02.

[62] Wikipedia. Game design document. `https://en.wikipedia.org/wiki/Game_design_document`. Accessed: 2023-10-27.

[63] Wikipedia. Jim weirich. `https://en.wikipedia.org/wiki/Jim_Weirich`. Accessed: 2023-9-9.

[64] Wikipedia. Sass. `https://es.wikipedia.org/wiki/Sass`. Accessed: 2023-11-02.

Apéndice A

Manuales

A.1. Manual de instalación y despliegue

A continuación se detallan los pasos a seguir para el despliegue de la aplicación en una máquina local.

A.1.1. Requisitos previos

- JDK 1.8 o superior
- Angular CLI 16.2.7 o superior
- Node 20.9 o superior
- npm 10.1 o superior
- Git para poder descargar el repositorio

A.1.2. Instalación

Para poder instalar el proyecto en la máquina local se deben seguir los siguientes pasos:

1. Descargar el proyecto alojado en la url `https://gitlab.inf.uva.es/larmart/royalgit.git` en el directorio que se quiera.
2. Utilizar el comando `cd royalgit`
3. Ejecutar el comando `npm install`

A.1.3. Despliegue

Tras tener el proyecto en la máquina local, para poder desplegar el proyecto se debe seguir los pasos listados a continuación:

1. Colocarse en la carpeta raíz del proyecto `/royalgit`
2. Ejecutar el comando `ng serve`

Tras compilar, se tendrá en la dirección `localhost:4200` la aplicación desplegada.

A.2. Manual de mantenimiento

A partir del proceso de instalación indicado en la Sección A.1.2, se podría empezar a desarrollar.

Se tienen dos ramas principales, main y develop. En este punto, develop se ha fusionado con main pero no se ha borrado esa rama, esta puede eliminarse o seguir desarrollando a partir de esta.

En esta sección 7.1 se detalla la organización del código que muestra la la estructura de directorios y ficheros del proyecto.

A la hora de crear un nuevo nivel se deben seguir los siguientes pasos:

1. Ejecutar el comando `ng g module /features/nuevo-nivel`
2. En el fichero `app.module.ts` añadir el módulo `nuevo-nivel` creado.
3. En el fichero `/features/level-list/level-list.module.ts` añadir la ruta de pathing al nuevo nivel correspondiente.
4. Ejecutar el comando `ng g component /features/nuevo-nivel/component1 --module features/nuevo-nivel`
5. En el fichero `/feautres/nuevo-nivel/nuevo-nivel.module.ts` añadir el pathing para los componentes creados si se necesita.

A la hora de crear un nuevo módulo que no es un nuevo nivel se siguen estos pasos:

1. Ejecutar el comando `ng g module /features/nuevo-modulo`
2. En el fichero `app.module.ts` añadir el módulo `nuevo-modulo` creado.
3. En el fichero `app-routing.module.ts` añadir la ruta del pathing del `nuevo-modulo` creado.

4. Ejecutar el comando `ng g component /features/nuevo-modulo/component1 --module features/nuevo-modulo`.

5. En el fichero `/features/nuevo-modulo/nuevo-modulo.module.ts` añadir el pathing para los componentes creados si se necesita.

A.3. Manual de usuario

A continuación se va a mostrar una serie de imágenes que permiten mostrar el uso del proyecto.

Al iniciar la aplicación (Figura A.1), se muestra el menú principal con las opciones que existen. Cada opción redirigirá a la pantalla correspondiente. Se destaca la primera entrada, la cual, en función del nivel en el que se encuentre el jugador, redirigirá a un nivel u otro.

Figura A.1: Pantalla: Menú principal

Esta Figura A.2 es la pantalla que se muestra al seleccionar la opción: Lista de niveles. Esta pantalla coincide con los niveles que se pueden jugar. Además, al hacer clic en cada entrada se muestra un desplegable con información relativa al nivel.

El nivel actual tendrá un botón que al pulsarlo llevará al jugador a la primera escena de dicho nivel. Esto se puede ver en la Figura A.3.

Figura A.2: Pantalla: Lista de niveles

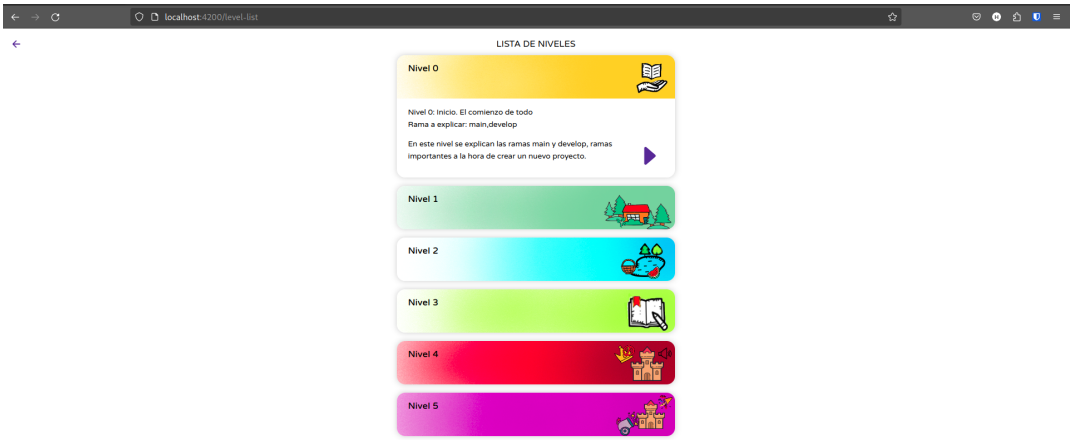


Figura A.3: Pantalla: Lista de niveles al pulsar en el nivel actual

Una vez pulsado en la entrada de Logros, en el menú principal se redirige a la pantalla que se puede ver en la Figura A.4, la cual contiene todos los logros obtenibles en el juego. Además,se indicará en la propia pantalla aquellos logros que se vayan consiguiendo. Este comportamiento se puede ver en la Figura A.5.

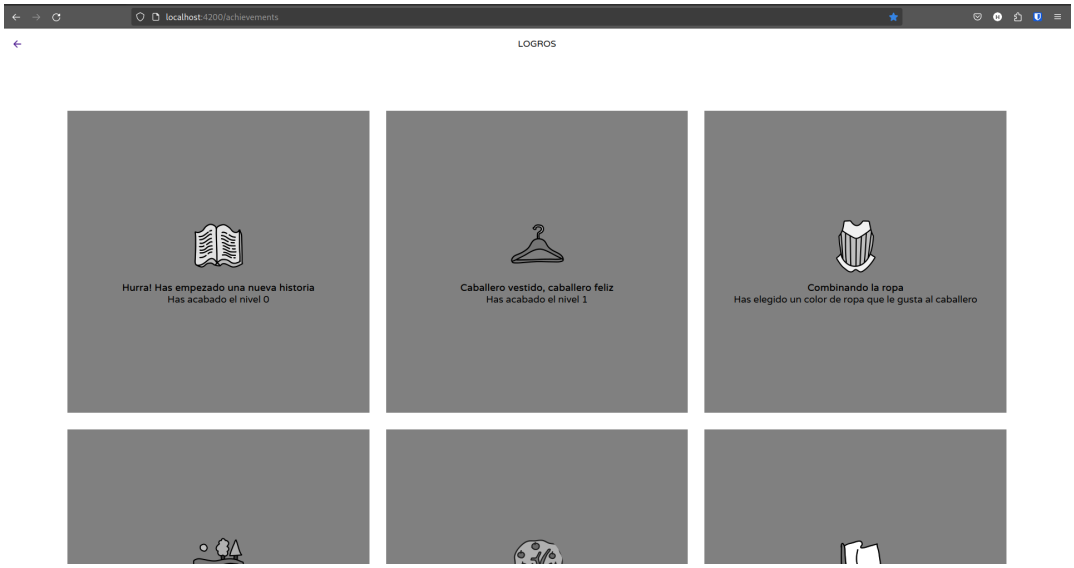


Figura A.4: Pantalla: Logros

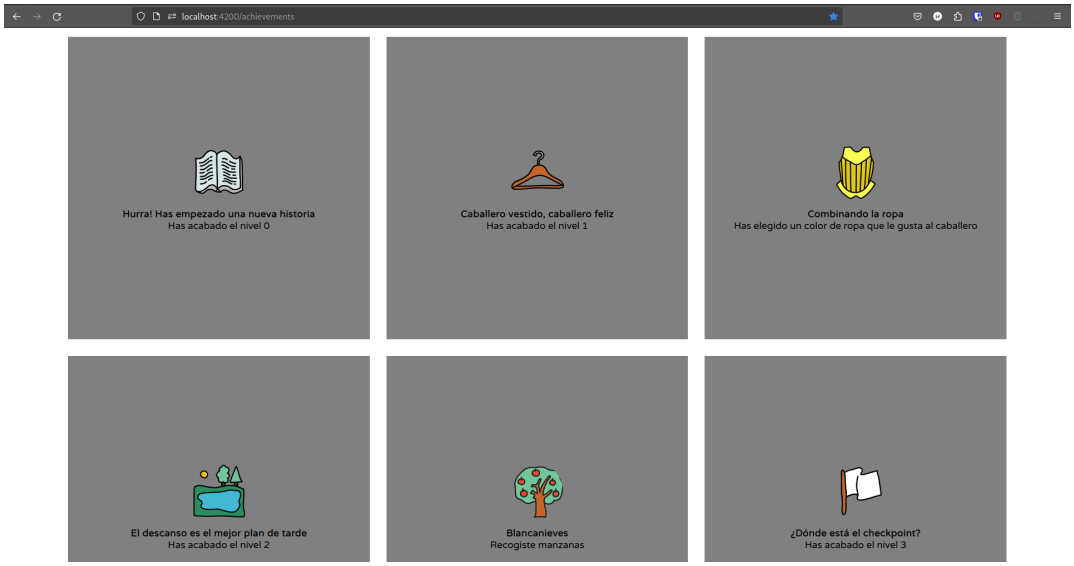


Figura A.5: Pantalla: Logros completos

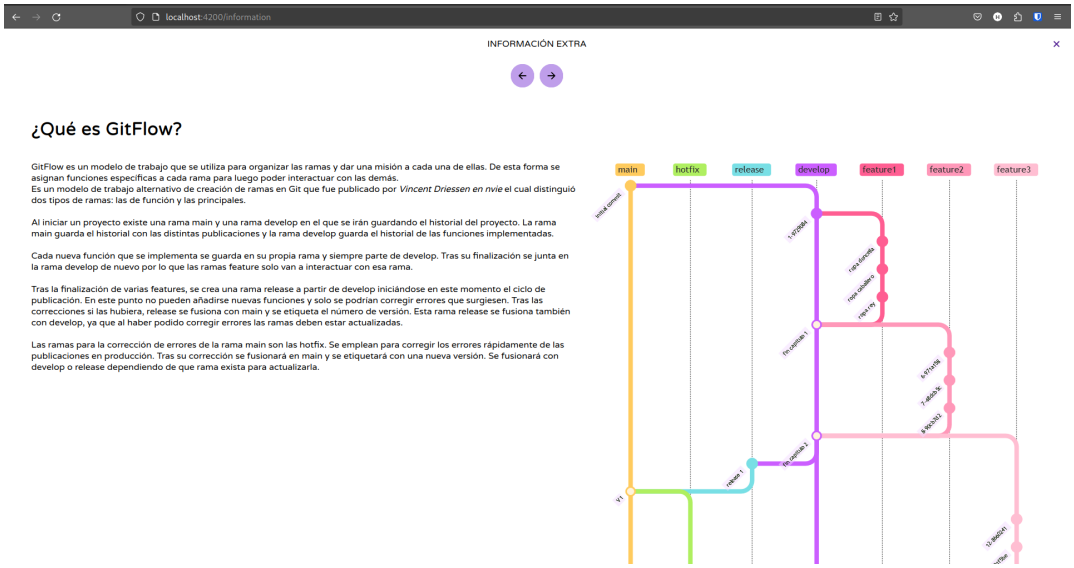


Figura A.6: Pantalla: Información extra

Esta Figura A.6 recoge toda la información necesaria para completar los niveles, de tal forma que en cualquier momento el jugador pueda acceder a ella a modo de consulta. Los botones mostrados en la parte superior de la pantalla permiten avanzar o retroceder en la página y mostrar más contenido.

Una vez se entra a jugar, la primera pantalla que se muestra es la Pantalla de introducción. Al hacer clic sobre esta, se mostrará más texto si lo hubiera, o redirigirá a la siguiente escena.

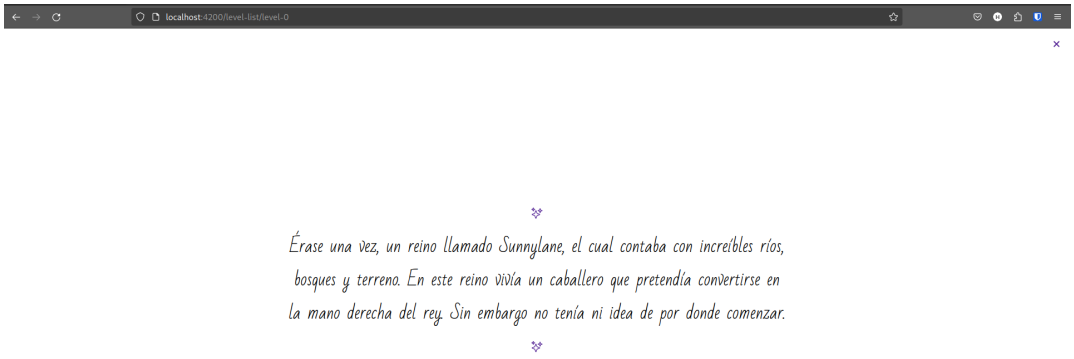


Figura A.7: Pantalla: Escena de Introducción del Nivel 0

La pantalla de contexto, representada en la Figura A.8, permite introducir al usuario información sobre la rama a explicar, así como guías para que el jugador complete los objetivos que se proponen en las escenas de flujo.

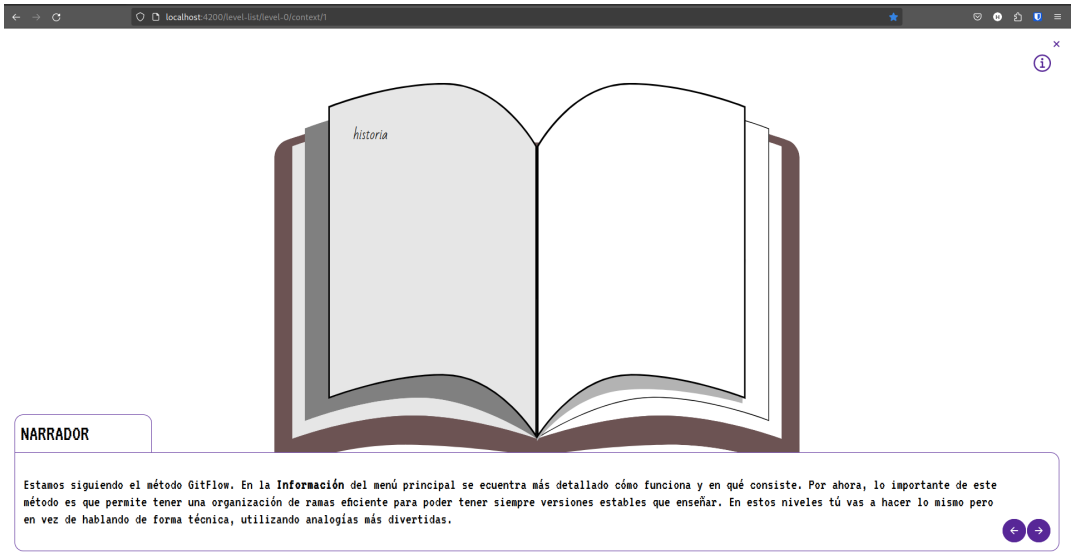


Figura A.8: Pantalla: Escena de Contexto del Nivel 0

Las escenas de flujo, mostradas en las Figuras A.9, A.10 y A.11 muestran distintas escenas

en las que el jugador ha tenido que introducir comandos en el cuadro de texto que se muestra en la parte inferior de la pantalla. En la Figura A.10 y en la Figura A.11 se muestra la escena de flujo según avanzan los niveles.

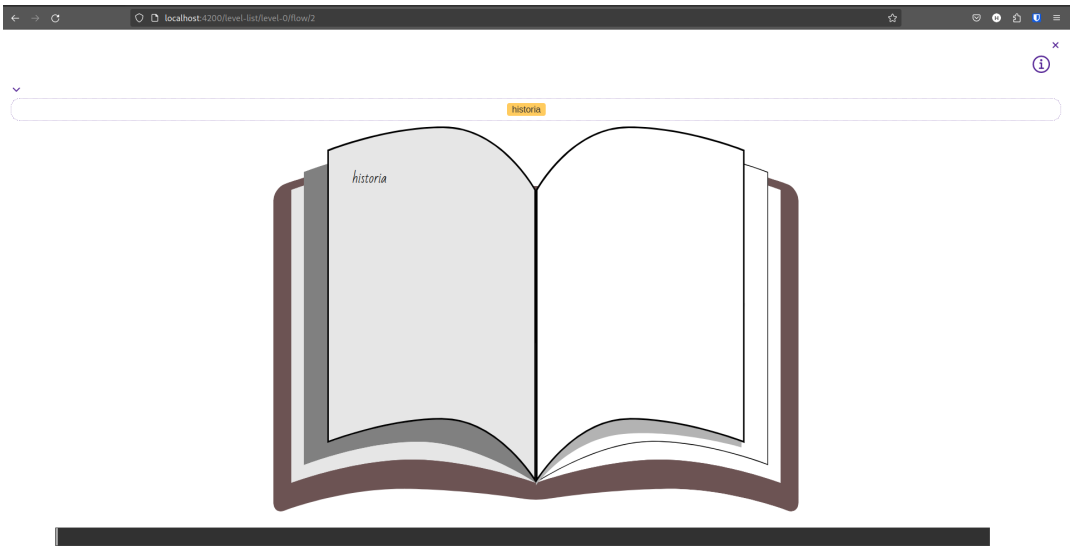


Figura A.9: Pantalla: Escena de Flujo inicial en el Nivel 0

Figura A.10: Pantalla: Escena de Flujo al iniciar la historia en el Nivel 0

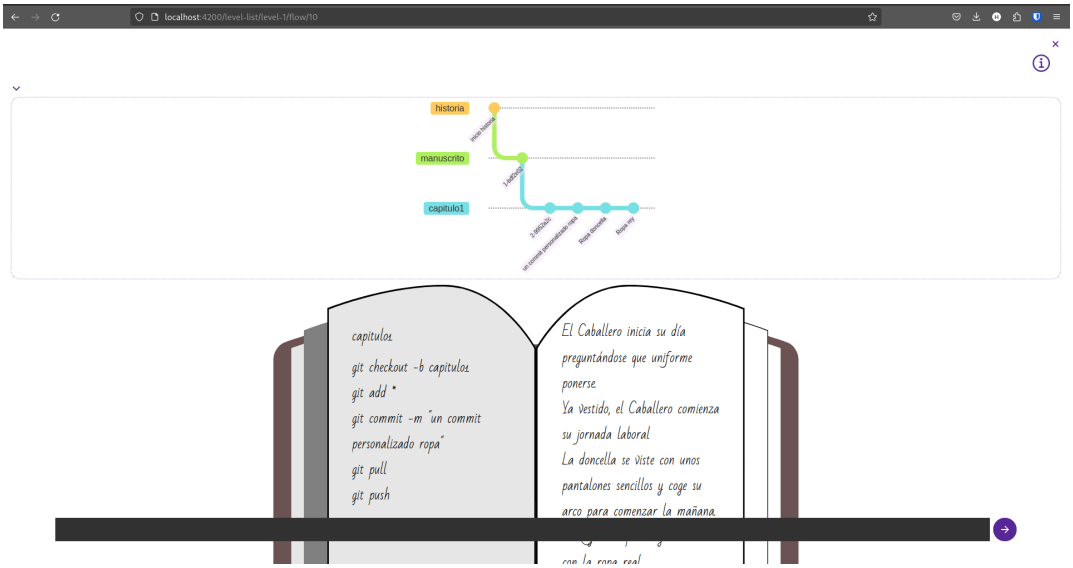


Figura A.11: Pantalla: Escena de Flujo durante el Nivel 1

Al completar la escena de flujo, se le indica al usuario que puede continuar y se le habilita un botón que le permite cambiar de escena.

En la Figura A.12 se ve la estructura que siguen las escenas de opción. Se le da al usuario la posibilidad de elegir entre dos opciones que se muestran, y tras pulsar sobre alguna de ellas, se redirigirá al jugador a la siguiente escena.

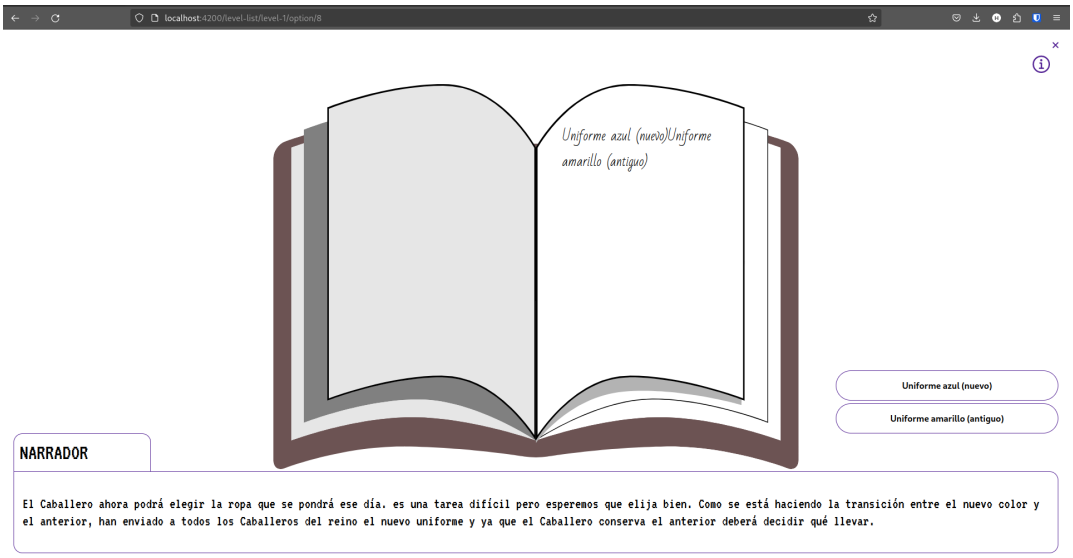


Figura A.12: Pantalla: Escena de Opción para el Nivel 1

Durante las escenas de Contexto, Opción y Flujo se le da al jugador una lista de comandos e información útil del nivel que puede acceder pulsando sobre la i de la parte superior derecha. En la Figura A.13 se puede ver la información útil relativa al nivel 1.

Figura A.13: Pantalla: Modal de Información del Nivel 1

Tras terminar el nivel, la Figura A.14 corresponde con la pantalla que recoge la información con los logros obtenidos en el nivel, además de permitir al usuario salir al menú principal para acceder a la pantalla de logros y ver aquellos que están completos.

Figura A.14: Pantalla: Resumen Nivel 0

En todas las pantallas se da la opción al jugador de poder salir del nivel, (Figura A.15), que se está jugando pulsando la X que se encuentra en la zona superior izquierda. Pulsando ese icono se le mostrará un mensaje al jugador para confirmar que quiere salir del nivel.

Figura A.15: Pantalla: Modal para confirmar salir del nivel

Apéndice B

Resumen de enlaces adicionales

Los enlaces útiles de interés en este Trabajo Fin de Grado son:

- Repositorio del código: `https://gitlab.inf.uva.es/larmart/royalgit`.