UNIVERSITAT DE BARCELONA

Trabajo final de grado

GRADO DE INGENIERÍA INFORMÁTICA

Facultad de Matemáticas e Informática

Universitat de Barcelona

EcoSea - Juego de plataformas 2D sobre sostenibilidad en el mar

---

Autor: Manuel Ernesto Martínez Martín

Directoras: **Dra. Anna Puig Puig**
Dra. Inmaculada Rodríguez Santiago

Realizado en: **Departamento de Matemáticas e Informática**

Barcelona, 18 de junio de 2021

Abstract

The aim of this project is to develop a 2D platform game to teach second cycle high school students (14-16 years old) one of the essential aspects that keeps our planet in harmony: the marine ecology.

The player will receive didactic lessons while they are overcoming the different levels of the game, the objective of the game is to keep the oceans clean and thus become aware of the pollution present in our oceans, helping the species that inhabit it so that they can survive in them.

Likewise, the game has been developed with Unity (2019.4.19f1) using C# as programming language and will be adapted for different player profiles: achiever, explorer, killer, philanthropist. Since each player enjoys more playing the same game in a different way. Some players like to be rewarded, others want to test the limits of the game, and still others just want to complete it without pressure. For them, each type of player will unlock special abilities that they can use during the course of their games.

Regarding this final degree project, it is a work shared with another student (Arnau Rovira Pérez) who, together with his and his TFG, add up to the complete game that we want to achieve. This part of the work focuses on the base game.

Resumen

Este proyecto trata sobre el desarrollo de un videojuego 2D de plataformas para enseñar a los estudiantes de segundo ciclo de secundaria (14-16 años) uno de los aspectos indispensables para que nuestro planeta esté en armonía: la ecología marina.

El/La jugador/a recibirá lecciones didácticas mientras va superando los distintos niveles del juego, el objetivo del juego es mantener los océanos limpios y así tomar conciencia de la contaminación presente en nuestros océanos, ayudando a las especies que lo habitan para que puedan sobrevivir en ellos.

Así mismo el juego ha sido desarrollado con Unity (2019.4.19f1) con el lenguaje de programación C# y estará adaptado para los distintos perfiles de jugador: achiever, explorer, killer, filántropo. Ya que cada jugador disfruta más jugando el mismo juego de manera distinta. A algunos jugadores les gusta ser premiados, otros quieren poner a prueba los límites del juego y otros simplemente quieren completarlo sin presión. Por ellos cada tipo de jugador desbloqueará unas habilidades especiales que podrán utilizar durante el transcurso de sus partidas.

Por lo que se refiere a este trabajo final de grado, es un trabajo compartido con otro alumno (Arnau Rovira Pérez) que junto a su TFG y este suman el juego completo que queremos lograr. Esta parte del trabajo se centra en el juego base.

Agradecimientos

Agradezco a mi compañero y amigo Arnau, siempre que he tenido algún problema no ha dudado en ayudarme siempre que ha podido.

Agradezco a Anna Puig y a Inmaculada Rodríguez por estar ahí para cualquier problema que pudiera tener y por tutorizarme.

Agradezco a mi novia Patricia por aguantarme mientras estaba haciendo el proyecto.

Agradezco a mis padres por ayudarme y apoyarme a lo largo de la carrera.

Agradezco a mis amigos David, Ruben, Pau, Aarón y Fran por apoyarme.

Índice

Índice de figuras

Índice de tablas

1. Introducción

El control del impacto medio ambiental es un asunto crucial para el bienestar del planeta. Este efecto incide directamente en la ruptura del equilibrio en la naturaleza y principalmente es generado como resultado de la actividad humana. En el concepto de bienestar también se integra el bienestar de la población, es decir, la biodiversidad y los ecosistemas son esenciales para la vida de los seres vivos. Una simple acción como tirar una botella de agua al mar puede conllevar ciertas consecuencias, pero probablemente no toda la población sea consciente de la importancia de este tipo de actos aparentemente inofensivos.

Distintas organizaciones, fundaciones o empresas se están sumando a diversos proyectos o iniciativas para combatir estos problemas. Tal como explica un articulo de la fundación Mapfre [25], la ruptura de la armonía medioambiental frecuentemente esta causada por la contaminación acústica, del aire, del agua, o bien de los suelos. Además, los residuos que generamos contribuyen a agravar los efectos generando un empobrecimiento de los ecosistemas, de la biodiversidad y causando unos efectos irreversibles.

Eliminar todos los factores que inciden negativamente en el medio ambiente es un proceso muy complejo. Sin embargo, podemos contribuir a la mejora de dicha situación a partir de ciertas rutinas, costumbres o cambiando nuestros hábitos. Hoy en día es preocupante la cantidad de plástico que hay y que se sigue generando en el planeta, además, tal como se explica en Noticias ONU, la pandemia COVID-19 ha aumentado el consumo de plásticos:

"Desde el año pasado el uso de plásticos se ha disparado de manera asombrosa, no solo miles de millones de mascarillas, pero también guantes, desechos médicos y empaques de comida para llevar" [21]

Alrededor del 70 % de los materiales de uso esencial usados durante la pandemia acabarán en el océano, además de los vertederos, produciendo ello un aumento importante en la acumulación de residuos en el medio ambiente. Todo evidencia una cierta incertidumbre sobre qué nos deparará el porvenir. En una publicación de la organización UNICEF, se menciona que en el futuro el impacto medioambiental va a incrementar considerablemente, teniendo incidencia directa en los más jóvenes:

"Las personas menos responsables del cambio climático son los niños; sin embargo, ellos sufrirán las peores consecuencias" [29]

1.1. Contexto

La educación tiene un papel importante en la concienciación de los jóvenes. Sin embargo si la educación se basa solamente en enseñar sin dar incentivos a la persona que está aprendiendo, ésta tiene más dificultades en aprender porque puede no prestar demasiada atención o porque no le motiva el tema o la lección. Por ello, si al estudiante se le da una dosis de ocio relacionado con el tema cada cierto tiempo, combinado con la lección teórica, podrá asimilar mejor los conocimientos. Ya que además de haberlos recibido de parte del profesor de manera teórica los estaría poniendo en práctica a través del propio juego serio[1].

Este proyecto se sitúa en el ámbito de los juegos serios ya que su objetivo es facilitar el aprendizaje y la concienciación de los jóvenes sobre la ecología marina. El proyecto se ha desarrollado en C# usando Unity. Aunque es un lenguaje muy similar a Java, el cual se ha usado en la carrera, ha habido un trabajo de adaptación y de práctica para poder llegar a dominarlo, así como sus componentes del editor.

Este es un proyecto compartido con otro estudiante compañero de la carrera, Arnau [22]. Cada uno se

[1]Son juegos diseñados para un propósito distinto al ocio. Serio se refiere a productos utilizados por industrias como la de defensa, educación, exploración científica, sanidad, emergencias planificación cívica, ingeniería, religión y política [35].

encargará de una parte del juego: Ésta parte se centrará en el videojuego base, su parte es un modo editor añadido al juego. Juntos hacen un juego completo.

Se ha necesitado el conocimiento de varias asignaturas cursadas en la carrera:

- **Programación II**: Ya que es programación orientada a objetos y además hay eventos.
- **Ingeniería de software**: Para gestionar la organización del proyecto en sprints de dos semanas con una metodología agile.
- **Software distribuido**: Ya que se ha usado una api rest para la conexión con la base de datos, es útil saber como funcionan las api rest y saber usarlas.
- **Diseño de software**: Para gestionar las clases necesarias para el proyecto y hacer un buen diseño.
- **Gráficos y visualización de datos**: Aunque es un videojuego 2D, se sigue teniendo componentes vistos en la asignatura como por ejemplo la cámara. También siguen estando presentes aunque de manera implícita las transformaciones geométricas. Sigue existiendo un espacio donde están colocados los gizmos 2D con la misma idea general.

1.2. Motivación

Este proyecto surge de la necesidad de actuar enfrente a un problema a nivel mundial, el exceso de residuos plásticos. A pesar de que este material no es el único que influye negativamente en el bienestar del entorno natural, es imprescindible actuar. La organización de las Naciones Unidas ONU, en 2018 exponía que si no se llevan a cabo unas medidas en 2050 existirán cerca de 12.000 millones de toneladas en los vertederos y el océano [20].

La necesidad de controlar, reducir y tomar consciencia de nuestras acciones es evidente, por ello son muchas las organizaciones que recomiendan tener en cuenta la pauta de "las 4R"que consiste en [32]:

- 1R - Reducir: evitar embalajes o envases de plástico.
- 2R - Reutilizar: volver a utilizar el producto antes de desecharlo.
- 3R - Reciclar: depositar el residuo en el contenedor adecuado para que pueda ser reincorporado.
- 4R - Recuperar: sirve para para obtener un producto nuevo, ya sea generando energía o creando uno nuevo para alargar su vida útil.

La voluntad del presente trabajo es contribuir desde el campo de la informática y las nuevas tecnologías en un proyecto interactivo para concienciar a los jovenes y así favorecer la lucha contra la reducción del plástico y su impacto en el medio ambiente, tanto a corto como a largo plazo.

1.3. Objetivo principal

El objetivo principal del proyecto es diseñar y desarrollar un videojuego serio de plataformas 2D. Que sensibilice a la población más joven y reclame la importancia de reciclar el plástico que deteriora la ecología marina. El jugador deberá interactuar con otros personajes del juego y evitar la contaminación.

1.4. Objetivos específicos

El objetivo general se puede desglosar en los siguientes objetivos específicos:

1. Análisis y diseño del juego:
    - **Estudiar el modelo de jugador**: tipos de jugadores y características de cada uno.
    - **Diseño de los elementos ofrecidos** a cada tipo de jugador.
    - **Recolección y análisis de datos del jugador** para poder adaptar el juego a su estilo, así como la persistencia de estos datos.
    - **Diseño de las mecánicas y elementos del juego**.
2. **Desarrollo del juego** en base a los diseños del punto anterior.
    - **Estudio de Unity**.
    - Creación de **escenas**, **elementos de juego** e implementación de **scripts**.
3. Persistencia:
    - **Análisis, diseño e implementación** de la persistencia de datos.
    - **Diseño de la gestión de usuarios** y que información se necesita para guardar de cada uno.
4. Modo editor:
    - **Análisis, diseño y desarrollo** del modo editor y todos sus elementos.
    - **Diseño de una interfaz usable.** Hay muchos elementos, por lo que se ha de dejar muy claro qué hace cada uno.
    - **Posibilidad de cargar los niveles** diseñados por nosotros mismos para poder partir de alguna base al crear un nivel nuevo
    - **Dar la posibilidad de probar el nivel nuevo** para ver como ha quedado, pasando a ser como una partida del juego base original.
    - **Evitar errores.** El usuario no tiene porque tener conocimientos sobre programación ni sobre el funcionamiento intrínseco del juego, por lo que se ha de evitar que se realicen acciones no válidas y que antes de probar el nivel no falte nada esencial por colocar.

Todos estos objetivos se han distribuido de la siguiente manera:

- Objetivos comunes: Apartado 1.
- Objetivos de Manuel Ernesto: Apartados 2 y 3.
- Objetivos de Arnau: Apartado 4 siguiendo las guías de diseño del apartado 1.

1.4.1. Planificación

Para la planificación se ha hecho uso de un diagrama Gantt. El tiempo estimado es de 20 semanas y se distribuye el tiempo en una unidad por semana. Se ha trabajado en sprints de dos semanas.

El proyecto se inició en Febrero, aunque se empezó a planear a finales del curso anterior. Se realizaron una serie de reuniones iniciales a mediados de Febrero para enfocar el proyecto y se ha tutorizado cada dos semanas.

Como ya se ha mencionado se ha colaborado con otro trabajo final de grado que es una extensión de éste, el cual añade un modo editor donde se pueden construir y jugar nuevos niveles a partir de los ya existentes.

El testeo se ha hecho de manera informal con familiares y amigos usando una build de escritorio con el objetivo de detectar y arreglar errores.

La planificación está separada en cinco partes: formación, diseño, desarrollo, testeo y documentación (ver Figura 1).

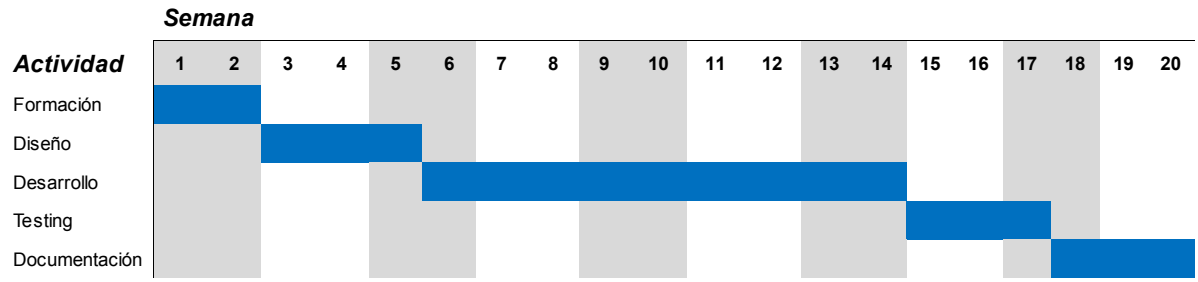


Figura 1: Planificación inicial del proyecto.

Al final del proyecto se hará una comparativa para ver si se ha conseguido la planificación.

1.4.2. Organización del documento

La estructura de esta memoria se desglosa en las siguientes secciones:

- **Introducción**: Tiene el propósito de contextualizar este proyecto con descripciones breves y el objetivo principal de éste.
- **Antecedentes**: Se estudian posibles soluciones, ideas y aplicaciones que ya han abordado los problemas planteados.
- **Análisis**: Se aborda la tipología de jugadores dependiendo de sus perfiles psicosociales, se definen las bases de los requerimientos del proyecto.
- **Diseño del juego**: Se describe el diseño del juego: idea principal, jugabilidad, prototipos de las pantallas de juego, sistema de diálogos y misiones.
- **Diseño del software**: Se detalla el diagrama UML de clases y los patrones de diseño.
- **Implementación del juego**: Se detalla la tecnología utilizada junto a la justificación. Se explican los componentes propios de Unity usados, la arquitectura del sistema, la arquitectura de los diálogos, misiones y habilidades. Finalmente, se explica la serialización de los niveles.
- **Simulaciones y resultados**: Se muestran los resultados del proyecto desarrollado.
- **Conclusiones y trabajo futuro**: Se detallan las conclusiones y el desarrollo del proyecto de cara al futuro.
- **Apéndice A - Manual técnico**: Instalación y requisitos técnicos, guía de usuario y guía de desarrollador.
- **Apéndice B - Descripciones de los casos de uso**: Se detallan las descripciones de los casos de uso como ampliación a la documentación.
- **Apéndice C - Endpoints**: Se detallan los endpoints que se utilizan tanto para la autenticación como para el control de la base de datos.

Como éste es un proyecto compartido, tanto yo como Arnau Rovira tendremos apartados iguales o similares en esta memoria, para identificarlos, estos tendrán el título de color **azul**.

2. Antecedentes

El presente proyecto se centra en el impacto de los plásticos en el entorno, sin embargo hay otros proyectos semejantes que abarcan otros aspectos de la contaminación y el reciclaje.

2.1. Juegos de ecología

2.1.1. Ecoembes

Ecoembes es una organización sin ánimo de lucro que a través del reciclaje y el ecodiseño cuida el medio ambiente. Dicha organización cuenta con diversos recursos medioambientales, como la sección de juegos [9] tradicionales como un puzzle, un parchís o un domino del reciclaje (ver Figura 2).

Figura 2: Juegos de ecoembes

Estos juegos, y otros con la misma temática, están publicados en formato PDF para que el usuario interesado en el juego pueda imprimirlo y jugar donde quiera. Ecoembes informa que estos juegos ayudan a los niños a trabajar en equipo y a reforzar la colaboración y la cooperación, además de inculcar hábitos saludables a sus rutinas diarias.

2.1.2. Energy kids

La página web Energy kids fue premiada en 2012 por la asociación de Web marketing y le dieron también un Gold Screen Award en 2010 y un Adobe Merit Award en 2020.

Esta página web se centra en educar a los niños sobre la energía que consumimos, de dónde viene, qué se necesita para generarla, cómo contamina y afecta el medio ambiente, cómo generar energía limpia y muchos otros temas. Es una web realmente completa, pero no solo se limita a dar información, también tiene una sección de juegos [1]. Hay desde adivinanzas y rompecabezas hasta talleres manuales.

2.1.3. Endling

Ecoembes y Energy kids proponen juegos demasiado sencillos, con los que se pueden aprender cosas pero difícilmente te darán ganas de volver a jugar.

Endling [27] es un juego mucho más serio, que si bien es cierto que no está pensado para las aulas, es un muy buen ejemplo de lo que se debe hacer para transmitir mensajes importantes y a la vez entretener al jugador. En este juego te encuentras en la piel del último zorro en un planeta destruido por los humanos, debido a su conducta destructiva hacia la naturaleza. Tienes el objetivo de llevar tus crías a un lugar seguro, lejos de la contaminación y destrucción del ser humano (ver Figura 3).

Figura 3: Gameplay de Endling.

Esta aproximación hace que no sólo se genere empatía hacia el personaje que se controla, sino que además uno sienta que los que lo han conducido a esta situación están errados en su conducta. Es cierto que tal vez es una temática demasiado seria para un juego en las aulas, pero lo que se puede extraer de aquí es el concepto de cómo está pensado el juego. Se necesita que los alumnos se involucren en el juego, tengan motivos para jugar y que, al final, aprendan sin darse cuenta de que están aprendiendo. No es necesario que aprendan sólo aspectos teóricos sobre la contaminación, como qué envase va a qué contenedor, también hay muchos otros valores que se pueden extraer de un juego. Del mismo modo, no es necesario que un juego sólo apunte a tocar la fibra sensible del jugador, puede atraer su interés competitivo, socializador, creativo, etcétera.

2.2. Juegos adaptativos

2.2.1. MUD

MUD o Multi-User Dungeon es un juego de rol y fantasía multijugador creado por Roy Trubshawi Richard Bartle lanzado en 1978. El servidor anfitrión por excelencia de MUD es Mudlet [7], una plataforma de alojamiento de varios juegos de este género. El juego es muy sencillo y se basa sólo en texto (ver Figura 4), pero Richard Bartle estableció los cimientos de lo que más tarde sería el modelo Hexad y redactó un artículo [4] al respecto. Inicialmente él solamente distinguió los jugadores asesinos, triunfadores, exploradores y socializadores, pero estableció las bases para muchos otros modelos similares.

Figura 4: Gameplay de MUD1.

2.2.2. Emergent Personalized Content in Video Games

Simone Guggiari realizó una tesis [14] en el 2019 sobre la generación procedural de niveles en un videojuego dependiendo de la personalidad de los jugadores. Se basó en el modelo de Richard Bartle que se acaba de mencionar.

Comenzó haciendo una versión muy sencilla del juego (ver Figura 5) con la que empezó a construir su propio modelo de jugadores a partir del modelo de Bartle utilizando los primeros datos que podía recoger.

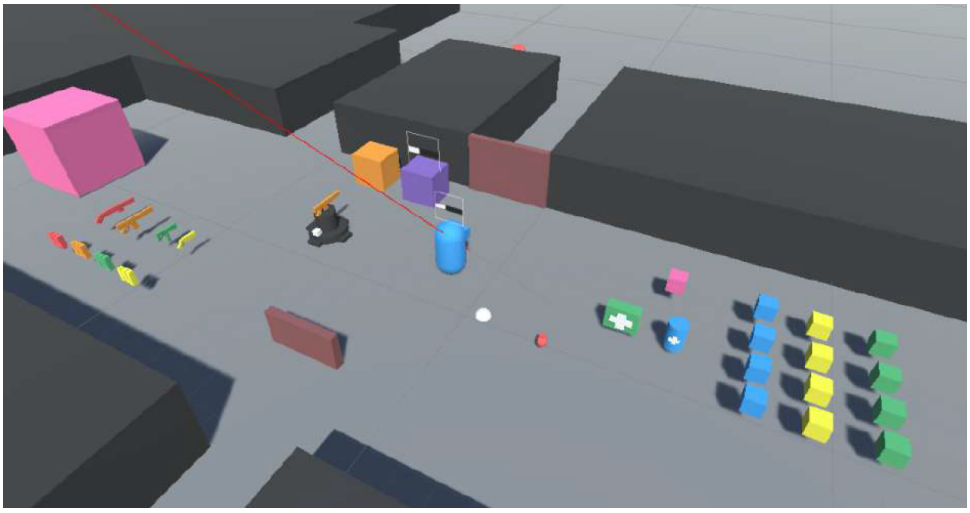


Figura 5: Captura de la primera iteración del juego con la que se comienza a construir un modelo de jugadores en el proyecto "Emergent Personalized Content in Video Games".

Una vez construido el modelo, o al menos la base, se comienzan a desarrollar más las texturas, funcionalidades y jugabilidad del juego (ver Figura 6) y a construir el algoritmo que hará uso de su nuevo modelo de jugadores de manera dinámica para a generar los niveles de manera procedural. El algoritmo de generación de nivel que utiliza esta basado en un algoritmo generativo innovador de propósito general.

Figura 6: Captura del juego final de la tesis "Emergent Personalized Content in Video Games".

Los resultados concluyeron que había una clara correlación entre la adaptabilidad del juego y la valoración de los usuarios, por lo que la tesis confirmó la hipótesis de que se puede mejorar la experiencia general del jugador adaptando el nivel a su estilo.

2.3. Tabla comparativa de los distintos juegos

Nombre	**Descripción / Objetivo**	**Plataformas**
MUD	Es un juego de rol y fantasía de solo texto, en el que se empezó a trabajar con los tipos de jugadores	Disponible en PC
Ecoembes	Juegos tradicionales para imprimir donde se abarcan aspectos como la contaminación y el reciclaje	Disponible en PDF
Energy kids	Módulos (o minijuegos) para educar a los niños sobre los recursos energéticos que consumimos. Explicando por ejemplo, como se genera la energía y como puede llegar a afectar al medio ambiente	Disponible desde su *web*
Endling	Videojuego donde el usuario lleva a un zorro y debe velar por el bienestar de sus crías en un planeta destruido por el ser humano y la contaminación	Disponible en *Steam*

Tabla 1: Comparativa de los distintos juegos.

2.4. ¿Dónde nos situamos nosotros?

Este proyecto no pretende estar al nivel de un juego como Endling o Super Mario Maker, pero sí se quiere alejar de la otra modalidad de juegos educativos anteriormente mencionados. Como estudiantes a los que nos han hecho jugar en el aula, es bien sabido que la gran mayoría de veces estos juegos pasan sin pena ni gloria por nuestras manos. Se espera que éste juego entretenga a todos los alumnos del aula, sea cual sea su personalidad y su conducta a la hora de jugar, que divierta desde el más

competitivo al más amigable e incluso al más disruptor. Para ello, el juego se adaptará a cada uno de los jugadores, dándole habilidades únicas que potenciarán su personalidad y su manera de jugar, haciendo que se sienta más cómodo jugando y no lo vea como una actividad aburrida.

Además, aunque se han diseñado unos niveles y temáticas establecidas, se proporciona un "modo editor", para que los profesores o incluso los alumnos más creativos puedan dejar fluir la imaginación y crear niveles realmente interesantes y completos. Adaptando así el contenido que consideren necesario.

3. Análisis

Se trata de crear un juego de plataformas 2D serio dirigido a alumnos/as que trate sobre el problema que asola a la ecología marina, la contaminación de plásticos de todo tipo. Durante el juego la experiencia del estudiante cambia dependiendo de su perfil de jugador.

3.1. Adaptabilidad al jugador

En todo desarrollo de un juego participan usuarios con diferentes perfiles psicosociales. Uno de los aspectos básicos a la hora de desarrollar un juego es crear un proyecto que intente satisfacer las necesidades de todos los usuarios. para este proyecto se ha escogido el modelo Hexad [18] para el diseño de puntuaciones, recompensas y habilidades.

3.1.1. Tipología de jugador: modelo Hexad

El modelo Hexad propone una serie de tipos de usuarios (ver Figura 7) basándose en las motivaciones, preferencias y personalidades de cada jugador.

Figura 7: Tipos de usuarios - Hexad.

- **Socializadores**: Les motiva la relación con otras personas o personajes. Quieren interactuar con ellos y crear conexiones sociales.
- **Espíritus libres**: Les motiva la autonomía y poder expresarse libremente.
- **Maestros**: Quieren mejorar siempre como jugador, siempre buscan nuevos retos.
- **Filántropos**: Son altruistas, enriquecen la vida de otras personas sin esperar nada a cambio.
- **Jugadores**: Les motiva las recompensas que puedan obtener a nivel personal.
- **Disruptores**: Les motiva el cambio, hacer que pasen cosas que no deberían pasar, cambiar la experiencia clasica del juego.

3.1.2. Adaptación del modelo Hexad al proyecto

Cada usuario comenzará con 3 tipos puntuaciones, una para maestro, una para socializador y otra para disruptor. No necesitará puntuación de espíritu libre ni de filántropo ya que el modo editor y

las misiones opcionales que les interesa a cada uno respectivamente ya están desbloqueadas desde el principio. Estas puntuaciones no se mostrarán nunca al usuario y no tendrá manera de saber su progreso en ninguna rama ni de qué manera obtener los puntos para cada una.

Estas puntuaciones comenzarán teniendo un valor de 0, y a medida que vayan cumpliendo ciertos requisitos detallados en el apartado **4.2**, irá aumentando su puntuación. Se ha marcado un estándar que indica que cada 100 puntos se desbloqueará la rama entera. Así, las tres habilidades de la rama se irán desbloqueando a llegar a 5 (o 10 dependiendo de la rama), 50 y 100 puntos, es decir, un maestro, por ejemplo, al llegar a 10 puntos desbloquea la habilidad de correr, al llegar a 50 puntos desbloquea el doble salto y al llegar a 100 desbloquea el empuje o dash. La primera habilidad requiere pocos puntos para empezar a potenciar el estilo del jugador desde el principio y que no tenga que esperar dos o tres niveles para desbloquear la primera habilidad (ver Tabla 2).

A continuación se muestra un esquema que representa la adaptación del modelo Hexad a este proyecto (ver Figura 8).

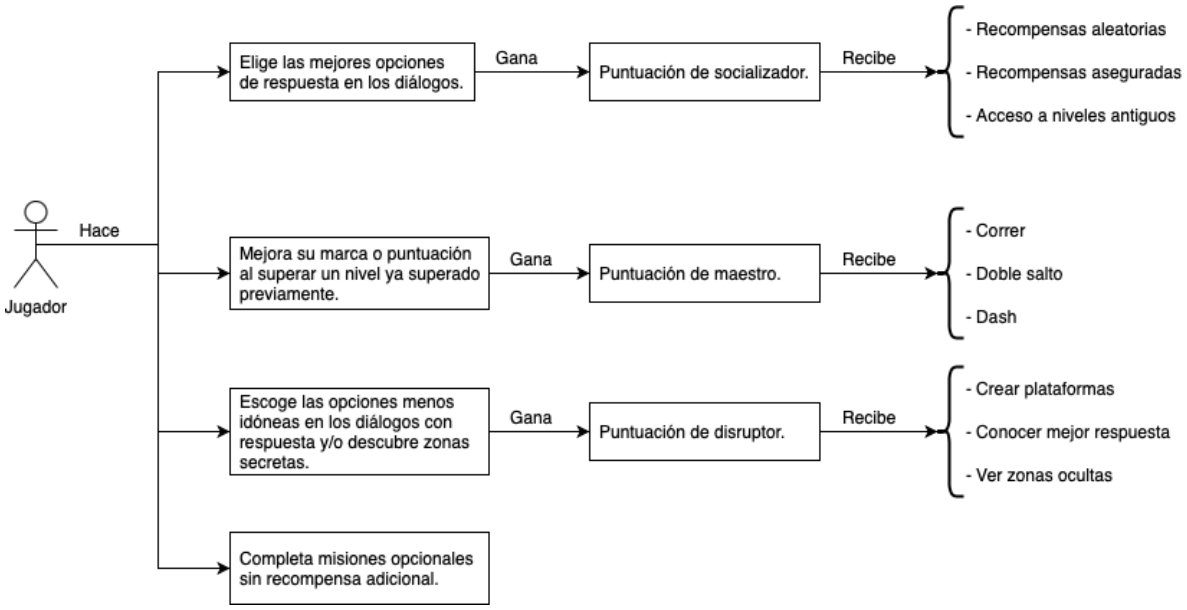


Figura 8: Modelo Hexad adaptado a EcoSea.

Habilidad	**Rama**	**Puntos requeridos**
Correr	Maestro	10
Dash	Maestro	50
Doble salto	Maestro	100
Recompensas de NPC aleatoriamente	Socializador	5
Recompensas de NPC siempre	Socializador	50
Acceso a todos los niveles	Socializador	100
Crear plataforma	Disruptor	5
ejor/peor respuesta en diálogos	Disruptor	50
Mostrar zonas secretas	Disruptor	100

Tabla 2: Habilidades y su puntuación requerida para ser desbloqueadas.

3.2. Diagramas de casos de uso

En éste apartado se detallan los diagramas de casos de uso. Se ha optado por separar a especificación del análisis en dos diagramas:

- Un diagrama de casos de uso de navegación por los menús sin incluir la partida.
- Un diagrama de casos de uso de una partida.

3.2.1. Casos de uso de navegación

Como se muestra en el diagrama de navegación (ver Figura 9), existen dos actores principales, el jugador que no está dado de alta, Jugador Invitado y el Jugador Autenticado.

Al arrancar la aplicación el usuario es Jugador Invitado. Las opciones que tiene son: iniciar sesión y convertirse en un Jugador Autenticado, recuperar su cuenta de usuario si ha olvidado la contraseña, crear una cuenta nueva. Como Jugador Autenticado, se puede avanzar al menú principal. Desde ahí puede: cambiar sus habilidades, cambiar su contraseña, eliminar su cuenta, cerrar la sesión, empezar una nueva partida o elegir un nivel al que jugar.

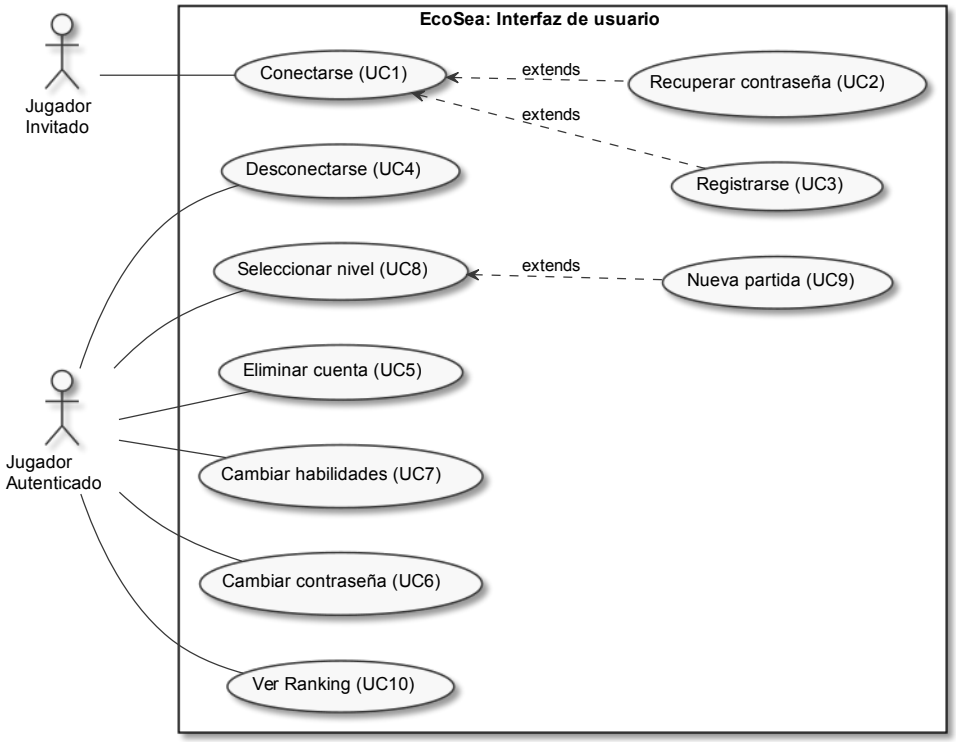


Figura 9: Diagrama de casos de uso de menús e interfaz.

Se pueden consultar las descripciones de los casos de uso de los menús e interfaz en el apéndice **B.1**.

3.2.2. Casos de uso de partida

Una vez autenticado y dentro de una partida (ver Figura 10) se parte de que hay un solo actor, puesto que para llegar a jugar una partida se requiere ser un Jugador Autenticado. El Jugador podrá: controlar a su personaje por el escenario en el eje de las x e y ya que es 2D, saltar, usar habilidades activas especiales que tenga asignadas, hablar con otros personajes, recoger objetos del suelo, pausar la

partida, consultar los objetivos que tiene que completar para terminar una partida, ver las habilidades que tiene asignadas, reiniciar o reanudar la partida y salir de la partida al menú principal.

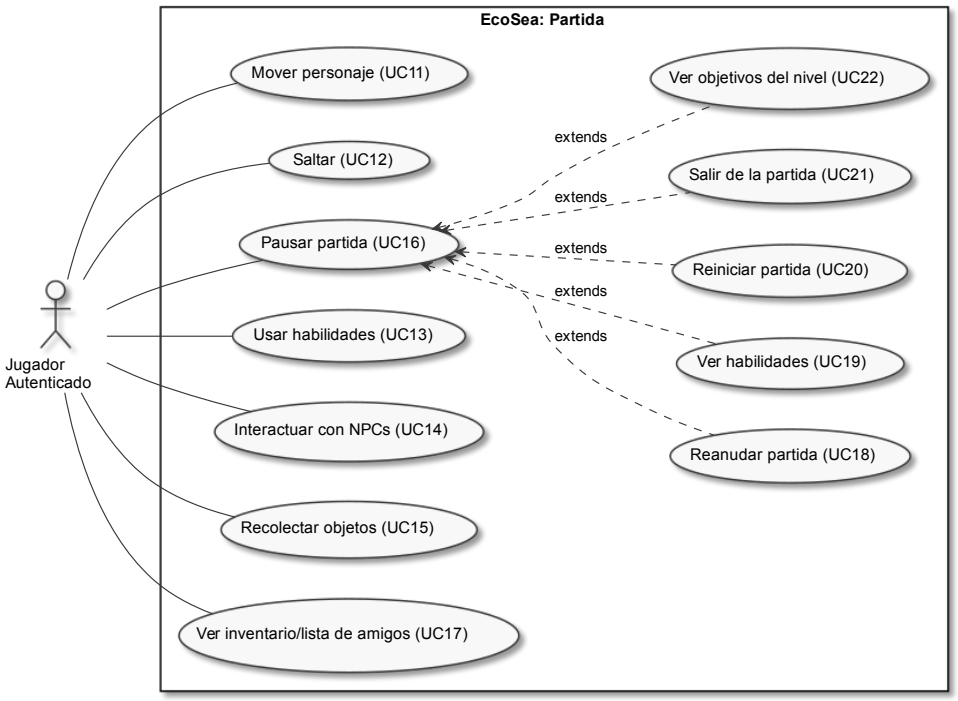


Figura 10: Diagrama de casos de uso de partida.

Se pueden consultar las descripciones de los casos de uso de la partida en el apéndice **B.2**. Los casos de uso del 18 al 22 no se detallarán puesto que son simples acciones desde botones.

4. Diseño del juego

4.1. Objetivos del juego y reglas

El objetivo principal del juego es interactuar con personajes no controlables o NPCs mediante **diálogos**. Estos personajes podrán pedirle al jugador que complete ciertos objetivos como que interactúe con otros personajes y/o que recoja ciertos objetos del escenario y a esto le llamamos **misiones**. Los objetos recogidos pueden ser entregados al personaje a cambio de monedas o mantenidos en el inventario del jugador.

Al final de la partida se obtienen puntos en base a la cantidad de objetos conseguidos, el tiempo en que se ha completado el nivel, NPCs ayudados, misiones opcionales completadas y zonas secretas descubiertas. El jugador podrá visualizar casi todas estas estadísticas mientras juega, ya que habrá un contador de tiempo en el HUD[2] además de un inventario y menú de pausa donde se pueden ver el resto. Lo único que no podrá saber hasta que termine el nivel son las zonas secretas que ha descubierto.

Los **diálogos** tienen mensajes y opcionalmente pueden contener: imágenes junto al texto, respuestas desde 1 hasta 4 que afectarán a la experiencia del jugador y/o misión/es para el jugador. Los diálogos pueden ser requeridos para completar un nivel (no pueden saltarse) y pueden añadirle objetos al inventario del jugador o retirárselos a cambio de monedas.

El jugador podrá utilizar habilidades adicionales (ver apartado **4.2.2**) que irá desbloqueando mientras vaya jugando. Estas habilidades se desbloquearán en base a una *puntuación oculta* que indica qué tipo de jugador es.

Las **misiones** pueden ser optativas u obligatorias. Las optativas pueden no ser completadas, pero las obligatorias son requisito indispensable para poder terminar el nivel.

El juego agrupará los niveles por mundos, existen un total de dos mundos, cada mundo tiene unos niveles. El nivel donde se desarrolla la partida puede estar dividido en secciones o zonas, a las que se accederá a través de puertas o teletransporte que habrá en el nivel. Las puertas requieren que se aprieta la tecla de interacción (flecha arriba o W) para entrar, mientras que los teletransporta activan sólo entrar en contacto. Además, los teletransporte tienen la posibilidad de ser invisibles, pudiendo crear así accesos a zonas secretas o simplemente cambios de escenario "suaves".

EcoSea consta de tres niveles:

- En el primer nivel, el jugador intentará ayudar en el puerto a un marinero que intenta embarcarse con su bote, pero se encuentra con el puerto lleno de plásticos. El jugador deberá ayudar al marinero a recoger los plásticos y además deberá pedir ayuda a los demas marineros del barco.
- En el segundo nivel, el jugador después de ayudar al marinero, llegará al mercado del pueblo y hablara con los distintos mercaderes. Estos le explicarán que a consecuencia de la contaminación no pueden abrir sus puestos, puesto que hay micro-plásticos en los estómagos de los peces. Además los mercaderes le irán mostrando los peligros de la contaminación.
- En el tercer nivel el jugador repasará las 4 erres fundamentales del reciclaje interactuando con los NPCs que representan estas erres (habrá un NPC por erre).

[2]Se le llama Head-Up Display o Visualización Cabeza-Arriba, también conocido como barra de estado a la información que en todo momento se muestra en pantalla durante la partida, generalmente en forma de iconos y números. El HUD suele mostrar el número de vidas, puntos, nivel de salud y armadura, y otros datos, dependiendo del juego [34].

4.2. Jugabilidad

Una partida llegará a su fin cuando el jugador ha completado un número determinado de objetivos obligatorios como por ejemplo: 2 diálogos obligatorios y 3 misiones obligatorias.

4.2.1. Tipo de jugador

Dependiendo de las acciones que realice el jugador en el entorno obtendrá puntos de un tipo u otro que le permitirá desbloquear habilidades especiales, que facilitarán y mejorarán la jugabilidad. Estos puntos, como se ha mencionado anteriormente no son visibles al jugador, sino que sirven para construir el modelo de jugador.

Las acciones que dan puntos para cada tipo de jugador son:

- **Socializador**
  - Elegir opciones más adecuadas (acciones que no falten el respeto ni sean negativas hacia otros personajes) en las respuestas de los diálogos.
- **Maestro**
  - Mejorar el tiempo en completar un nivel que ya había completado. Esto es, mejorar su propia marca.
  - Conseguir más puntos al superar un nivel, de los que ya se consiguieron al superarlo anteriormente.
- **Disruptor**
  - Elegir opciones menos apropiadas en las respuestas de los diálogos (opciones que tienen un carácter negativo en la interacción entre personajes).
  - Descubrir un teletransporte secreto.
- **Filántropo**
  - Completar misiones opcionales.

4.2.2. Habilidades especiales

Cada jugador puede desbloquear una serie de habilidades especiales con las que puede interactuar con el escenario del juego. Éstas habilidades cambian ligeramente la perspectiva de juego porque le aportan beneficios y ventajas al jugador en la partida. Las habilidades pueden ser activas o pasivas:

- Habilidades activas, se activan pulsando una tecla en concreto.
- Habilidades pasivas, repercuten de manera implícita, no hay que darle a ninguna tecla para usarlas.

Las habilidades están relacionadas con el tipo de jugador que interactúe con el juego. Estas se pueden desbloquear al final de un nivel. Concretamente, las habilidades que considera el diseño del juego son:

- Habilidades **activas**
  - **Correr**: El jugador podrá moverse a una velocidad superior a la normal.
  - **Doble salto**: Antes de tocar el suelo el jugador podrá realizar un segundo impulso.

- **Dash**: El jugador podrá realizar un empujón horizontal con cierta velocidad.
- **Crear plataforma**: Aparecerá una plataforma auxiliar donde el jugador desee ponerla (solo puede haber una en un mismo momento).
- **Mostrar zonas secretas**: Los teletransportes secretos serán visibles por unos segundos, después se volverán invisibles de nuevo. Esta habilidad se podrá usar hasta un máximo de 5 veces por partida.

- Habilidades **pasivas**
  - **Recompensa NPC aleatoriamente**: Al acabar un nivel los amigos del jugador a veces le otorgaran puntos.
  - **Recompensa NPC siempre**: Al acabar un nivel los amigos del jugador siempre le otorgaran puntos.
  - **Acceso a todos los niveles**: Se podrán jugar todos los niveles superados (sin la habilidad solo se puede jugar el actual y el anterior).
  - **Mejor/peor respuesta en diálogos**: Al hablar con un NPC que le tengamos que dar una respuesta, podremos ver la mejor (verde) y la peor respuesta (rojo). Durante los diálogos habrá respuestas que pueden ser muy negativas otras que pueden ser muy positivas y algunas neutrales. El valor de la respuesta además de servirle al jugador para ver los diálogos correctos (el jugador puede saltarse diálogos no obligatorios, o responder de mala manera a un NPC, esto puede suponer acabar una conversación, si el jugador desea ver toda una conversación sin acabar mal, deberá elegir la respuesta más positiva), sirve para els sistema de gestión de la tipo de jugador, ya que si el jugador eligen las mejores respuestas se puede llegar a hacer amigo del NPC y si elige las peores se pueden sumar puntos de disruptor a su perfil de jugador (ver Figura 11).

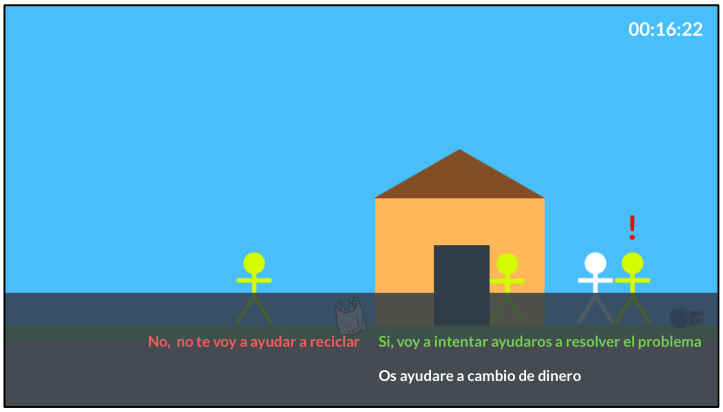


Figura 11: Uso de habilidad de mejor respuesta.

Aunque existe el tipo de jugador que denominamos espíritu libre, este solo obtiene beneficio del modo edición de niveles.

4.3. Diseño de pantallas y escenas

A continuación en la figura 12 se muestra el diagrama de flujo de las escenas del juego conectadas y después los prototipos de las escenas (incluido el modo editor desarrollado por Arnau Rovira).

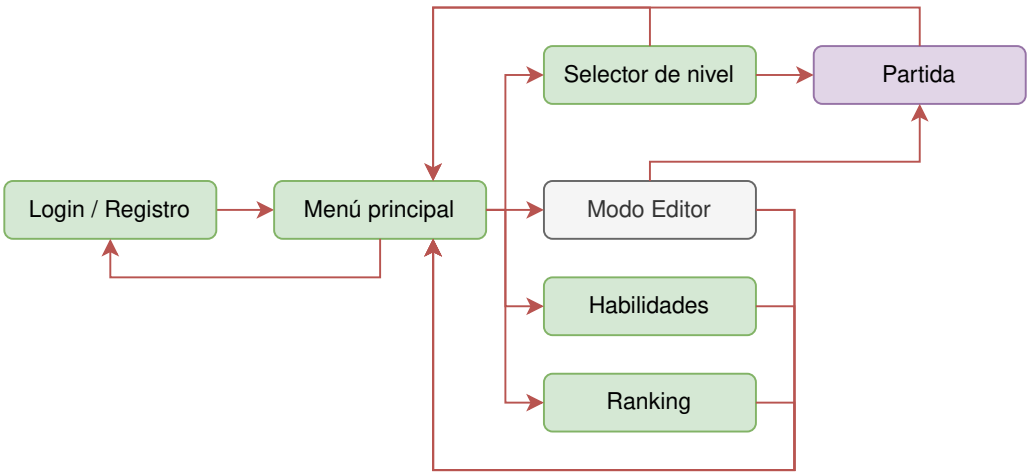


Figura 12: Flujo de las pantallas del juego.

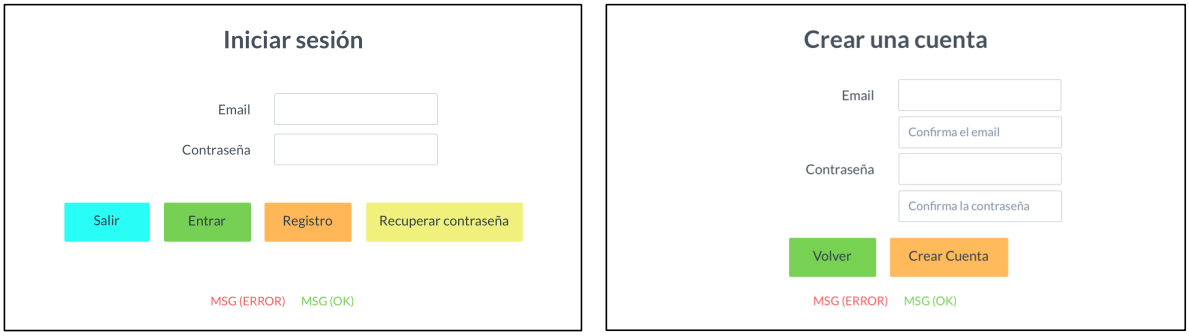
Explicación de las pantallas del juego:

- **Login-Registro**: Es la pantalla principal del juego, desde ella se puede entrar al juego usando una cuenta de usuario, si no se dispone de cuenta se puede crear una cuenta nueva usando su email y una contraseña o restaurar su contraseña si ya tenía una cuenta y la ha perdido (ver Figura 13a y 13b).
- **Menú principal**: Se accede al iniciar sesión, desde ella se puede cerrar sesión o borrar la cuenta de usuario, además de poder acceder al resto de escenas: Selector de nivel, Modo editor, Habilidades, Ranking (ver Figura 14b).
- **Selector de nivel**: Desde esta pantalla el jugador podrá elegir el mundo-nivel al que querrá jugar (ver Figura 16a).
- **Modo editor**: Pantalla desde la cual un usuario puede generar su propio nivel modificando uno o varios niveles existentes (ver Figura 24b).
- **Habilidades**: Desde esta pantalla el jugador podrá ver las habilidades que hay en el juego y podrá equiparse con hasta 3 habilidades máximas, siempre y cuando tenga habilidades desbloqueadas (ver Figura 17a).
- **Ranking**: Desde esta pantalla el jugador podrá ver por nivel o global una lista de los jugadores que mejor han resuelto la partida ordenado por puntos o por tiempo[3]. Los jugadores tendrán un color rojo mientras que si es el propio jugador será de color azul, así podrá identificarse mejor (ver Figura 17b).
- **Partida**: Es la escena de la partida (ver Figura 18b), una partida contiene las siguientes pantallas:
  - **Prólogo**: Pantalla opcional introductoria de una partida en la que se hace una breve introducción con imágenes y texto (ver Figura 18a).
  - **Pause**: Pantalla en la que se detiene el curso de la partida, se pueden consultar las misiones en curso/completadas, las habilidades escogidas, también se puede: volver a la partida en el estado que la había dejado, comenzar de nuevo la partida, salir al menú principal (ver Figura 19a y 19b).
  - **Inventario/Amigos**: Pantalla en la que se muestra el inventario del jugador y la lista de amigos conseguidos (ver Figura 20).

[3]Si se ve el ranking global solamente se puede ordenar por puntos no por tiempo.

- **Epílogo**: Pantalla opcional de conclusión de una partida en la que se hace una reflexión con imágenes y texto (ver Figura 22b).
- **Recompensas**[4]: Pantalla con las recompensas en puntos que dan los personajes amigos del jugador al propio jugador.
- **Desbloqueo de habilidades**: Pantalla donde se muestra una habilidad desbloqueada, aparece cuando el personaje consigue los puntos suficientes de una rama[5] con habilidades (ver Figura 23a).
- **Estadísticas**: Pantalla final del nivel donde se muestra la puntuación de cada apartado, puntuación total, tiempo en completar el nivel, botones de: Reiniciar, Continuar al siguiente nivel si existe, Salir al menú principal (ver Figura 24a).

Una vez se entra al juego lo primero que aparece es la pantalla de inicio de sesión (ver Figura 13a), si no se dispone de ninguna cuenta de usuario se ha de ir a la pantalla de registro (ver Figura 13b) y desde ahí se puede crear una cuenta.



(a) Prototipo de la pantalla de inicio de sesión.

(b) Prototipo de la pantalla de registro.

Figura 13: Prototipos de las pantallas de inicio de sesión y registro.

En la pantalla de registro, el jugador rellena los campos de email y contraseña dos veces y si cumplen las condiciones (email válido sin usar, contraseña de longitud máxima de 6 cifras alfanuméricas) muestra un mensaje verde de confirmación diciendo que se ha enviado un email para activar la cuenta, entonces la pantalla cambia y vuelve a la de iniciar sesión, en caso contrario se mostrará un mensaje rojo de error.

En la pantalla de iniciar sesión, cuando se introduce el email y la contraseña pueden pasar 3 situaciones erroneas. En cada caso saldrá un mensaje de error en color rojo. Los casos son:

- El usuario no existe.
- No hay internet.
- El usuario es inválido.

Si ocurre uno de los 3 casos anteriormente mencionados se mostrará un mensaje rojo de error dependiendo del caso.

Si el usuario existe, es decir, que tiene registrado su email pero la cuenta no estaba activa se volverá a mandar un email de activación y se mostrará un mensaje en verde. En caso de que el usuario exista y la cuenta esté activa se comprobará si el usuario tiene nombre de usuario o no. Ya que el email será privado de cada jugador, los demás jugadores verán éste nombre de usuario en lugar del email.

[4]Solo aparecerá la pantalla si el jugador tiene escogida alguna de las dos habilidades de recompensas.

[5]Como hay varias habilidades para los distintos perfiles de jugadores, cada conjunto de habilidades de una tipología de jugador pertenecen a una rama de habilidades, las ramas son: Maestro, Socializador y Disruptor.

Si el usuario no tiene nombre de jugador se pasará a la pantalla de elección (ver Figura 14a) del mismo. Una vez el usuario rellena dos veces el nombre de jugador, escoge un genero (si se escoge masculino se jugará con un personaje chico y si se escoge femenino se usará un personaje chica) y le da a entrar se comprueba si está disponible. Si el nombre está disponible se termina de crear los datos básicos del usuario y se pasa a la pantalla del menú principal (ver Figura 14b), en caso contrario mostrará un mensaje rojo de error.

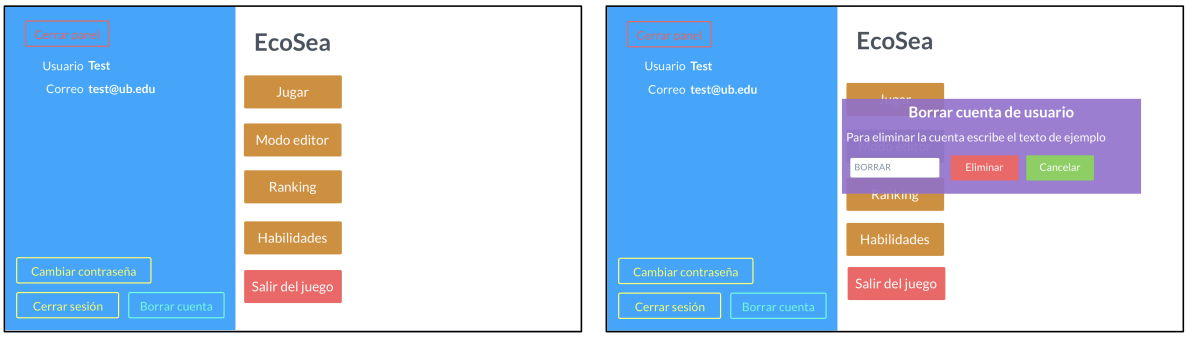
(a) Prototipo de la pantalla de nombre de usuario.

(b) Prototipo de la pantalla de menú principal.

Figura 14: Prototipos de las pantallas de elección de nombre de usuario y menú.

Desde la pantalla del menú principal se acceden a las distintas secciones del juego. También se puede ver la cuenta (ver Figura 15a) pulsando el botón ver cuenta. Desde ahí se puede cambiar la contraseña, cerrar sesión e incluso borrar la cuenta (ver Figura 15b).

Para borrar la cuenta se ha de escribir el texto de ejemplo que aparece en el placeholder del campo.



(a) Prototipo de ver cuenta.

(b) Prototipo de borrar cuenta.

Figura 15: Prototipos de las pantallas de ver y borrar cuenta de usuario.

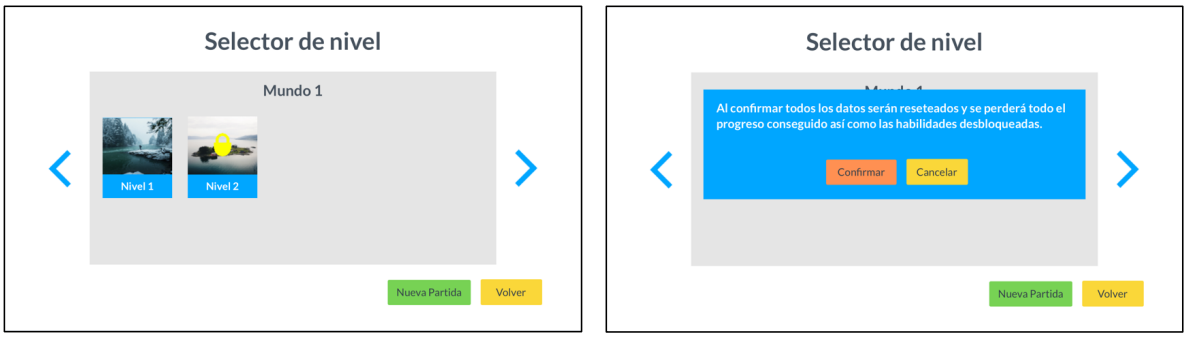
Desde la pantalla de menú principal si se selecciona jugar se accede a la pantalla de selección de nivel (ver Figura 16a), desde ahí se puede cambiar de mundo y elegir un nivel para jugar, posicionando el cursor encima de la tarjeta de un nivel aparecerá un *tooltip*[6] informativo con el título y descripción del nivel. También se puede borrar el progreso del usuario y empezar de nuevo (ver Figura 16b) sin ningún dato guardado. Seleccionando volver se regresa a la pantalla del menú principal.

Los niveles a los cuales no se tiene acceso para jugar aparecerán con el símbolo de un candado, este candado puede aparecer por dos motivos:

- Se trata de un nivel avanzado, el cual requiere haber superado un nivel previo.

[6]También llamado descripción emergente es una herramienta de ayuda visual, que funciona al situar el cursor sobre algún elemento gráfico, mostrando una ayuda adicional para informar al usuario de la finalidad del elemento sobre el que se encuentra [36].

- Un jugador solo puede jugar al nivel actual y al anterior. Para jugar a los niveles antiguos hay que activar la habilidad de **todos los niveles** anteriores al actual.



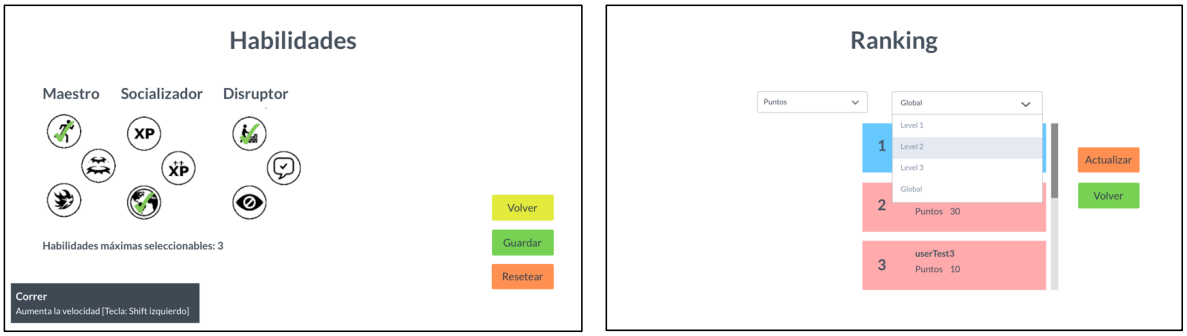
(a) Prototipo de la pantalla de selección de nivel.

(b) Prototipo de nueva partida.

Figura 16: Prototipos de las pantallas de selección de nivel y nueva partida.

Si se selecciona Habilidades se entrará en la pantalla (ver Figura 17a) que contiene las habilidades posibles que el jugador puede escoger, podrá elegir hasta un máximo de 3. Posicionando el cursor encima de alguna se mostrará un *tooltip* informativo de lo que hace la habilidad y como se utiliza. Si pulsa reiniciar se desmarcarán las habilidades. Si se pulsa guardar se guardará la elección del jugador y volverá al menú principal. Si se pulsa volver regresará al menú principal sin alterar nada.

Si se selecciona Ranking desde el menú principal se entrará a la pantalla (ver Figura 17b) que muestra una lista de jugadores ordenada siempre ascendentemente, está lista puede estar ordenada por tiempo en superar el nivel o por puntos conseguidos, si en vez de mostrar un nivel se muestra en global solo se podrá ver por puntuación y no por tiempo.



(a) Prototipo de la pantalla de habilidades.

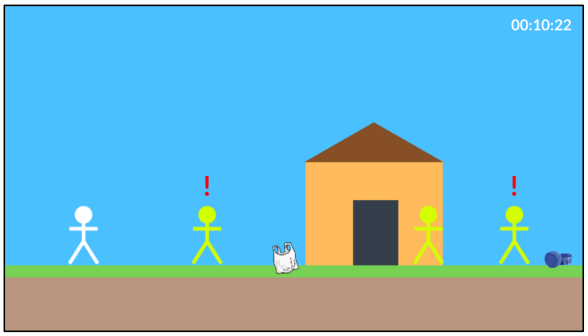
(b) Prototipo de la pantalla de ranking.

Figura 17: Prototipos de las pantallas de habilidades y ranking.

Una vez se ha elegido un nivel donde se desarrollará la partida se podrá ver opcionalmente una pantalla introductoria (ver Figura 18a) que puede tener texto e imágenes. Pulsando el botón comenzará la partida.

La partida en curso se representa con la siguiente imagen (ver Figura 18b).

(a) Prototipo de la pantalla de introducción.



(b) Prototipo de la partida.

Figura 18: Prototipos de las pantallas de prólogo de una partida y partida en curso.

Con la partida empezada, si se pulsa la tecla escape se pausa el juego y aparece un menú con dos columnas, a la izquierda una lista de botones para: seguir, salir, reiniciar el nivel. Y a la derecha una ventana que muestra las misiones activas o completadas (ver Figura 19a), éstas últimas tachadas, si se posiciona el cursor encima de cada misión se mostrará el título y una descripción más completa en un *tooltip*. Si se pulsa el botón de ver habilidades, se mostrarán las habilidades (ver Figura 19b) que hay y las que están elegidas, igual que en la pantalla ranking con su *tooltip* correspondiente, la única diferencia es que en mitad de la partida no se podrán cambiar las habilidades.

Desde el modo editor los botones de reiniciar y salir desaparecen y se muestra un botón de regresar al editor.

(a) Prototipo del pause con misiones.

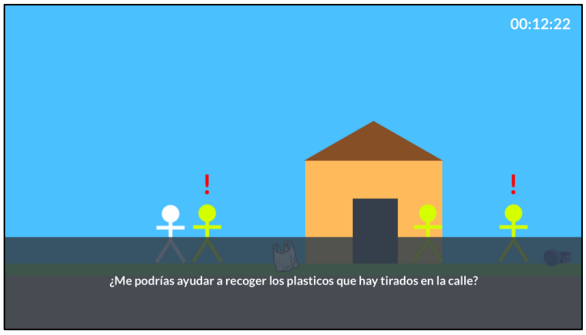
(b) Prototipo del pause con habilidades.

Figura 19: Prototipos de las pantallas de pause.

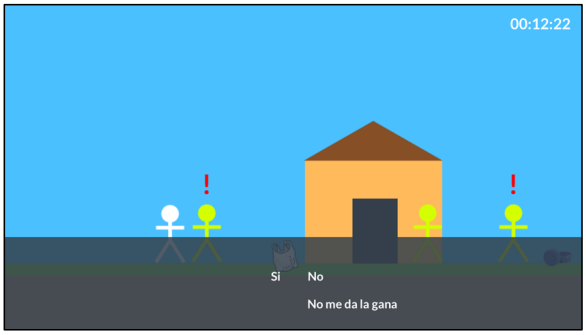
Si en vez de pulsar la tecla escape se mantiene pulsada sin soltar la tecla tabulador, se mostrará otra pantalla con el inventario en el lado izquierdo y los NPCs amigos del jugador en el lado derecho (ver Figura 20).

Figura 20: Prototipo del inventario del jugador.

También con la partida empezada, si el jugador se acerca a un NPC y pulsa la tecla espacio se abrirá una caja de dialogo[7] (ver Figura 21a), el dialogo podrá ser solo de mensajes del NPC con o sin fotos o también en algunos diálogos el jugador podrá responder (ver Figura 21b).



(a) Prototipo de diálogo.



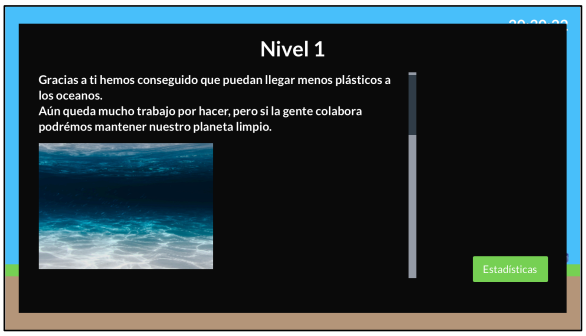
(b) Prototipo de respuesta en diálogo.

Figura 21: Prototipos de las pantallas de diálogo.

Cuando el jugador colisiona con un objeto del escenario en caso de que el objeto se pueda recoger, puede tener opcionalmente diálogos con o sin fotos (ver Figura 22a).

Al acabar un nivel opcionalmente se podrá ver una pantalla similar a la de inicio del nivel (ver Figura 22b).

(a) Prototipo de recolección con diálogo.



(b) Prototipo de la pantalla de conclusión.

Figura 22: Prototipos de las pantallas de recolección con diálogos y epílogo.

Desde la pantalla de conclusión del nivel al pulsar Estadísticas si el usuario puede desbloquear una

[7]La ventana de diálogos se maneja solamente con el cursor del ratón.

habilidad nueva porque ha conseguido puntos suficientes se mostrará una o varias pantallas sucesorias de las habilidades que ha desbloqueado (ver Figura 23a). La pantalla de nueva habilidad desbloqueada desde el modo editor no aparece, ya que en el modo editor la última pantalla que será visible al probar un nivel es la pantalla de conclusiones (ver Figura 22b).

Si el jugador además tiene la habilidad de las recompensas de NPCs amigos (cualquiera de las dos habilidades), se mostrará una pantalla con los premios que le den los amigos NPCs (ver Figura 23b). Desde el modo editor esta pantalla no aparece puesto que la pantalla final corresponderá a la pantalla de conclusión (ver Figura 22b).

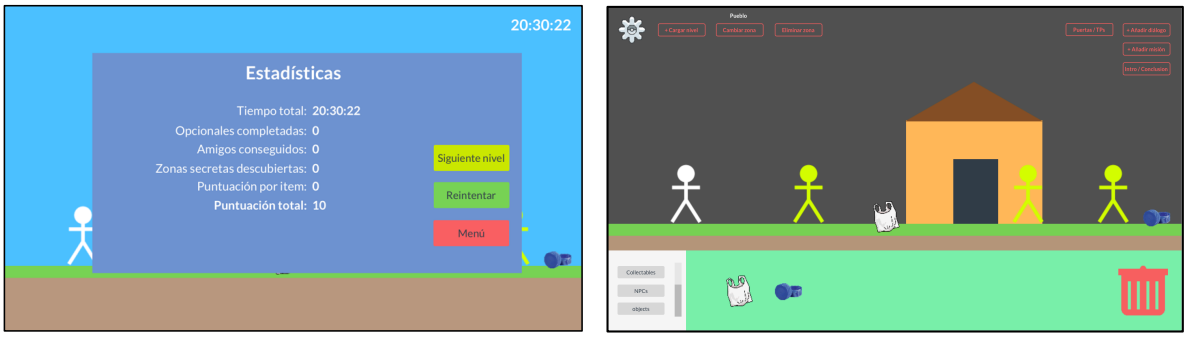
(a) Prototipo de la pantalla de nueva habilidad.

(b) Prototipo de la pantalla de recompensas.

Figura 23: Prototipos de las pantallas de nueva habilidad y recompensas de NPCs.

La última pantalla a la que se accederá al acabar el nivel es la de Estadísticas, que mostrará datos de lo que ha conseguido el jugador a lo largo de la partida (ver Figura 24a). Esto es, amigos conseguidos en esa partida, misiones opcionales que ha completado, zonas ocultas en las que ha entrado, una puntuación por objeto recogido y la puntuación total. El botón siguiente nivel solamente se mostrará si hay un nivel disponible después del actual, desde el modo editor esta pantalla no aparece.

El modo editor (ver Figura 24b) es una escena que se accede desde el menú principal, desde éste se podrán cargar niveles ya hechos y crear un nivel nuevo al que jugar en base a lo que ya existe en el juego.



(a) Prototipo de la pantalla de estadísticas.

(b) Prototipo del modo editor.

Figura 24: Prototipos de las pantallas de estadísticas y modo editor.

4.4. Diseño del sistema de diálogos y misiones

4.4.1. Sistema de diálogos

Un diálogo esta formado por uno o varios mensajes sucesivos donde cada mensaje puede tener opcionalmente una imagen para mostrar. También puede opcionalmente contener respuestas que enlazan a otros objetos diálogo, se contempla que pueda haber desde 1 respuestas hasta un máximo de 4 por objeto diálogo. Además cada diálogo puede tener opcionalmente también misiones que requerirán la participación del jugador (ver Figura 25).

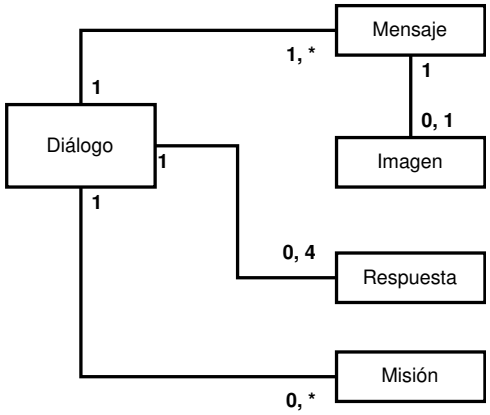


Figura 25: Composición de un diálogo.

Los diálogos se pueden encadenar, ya que las respuestas enlazan con otros objetos diálogos con las mismas propiedades que ellos mismos por tanto una conversación se podría decir que es un árbol de diálogos (ver Figura 26)

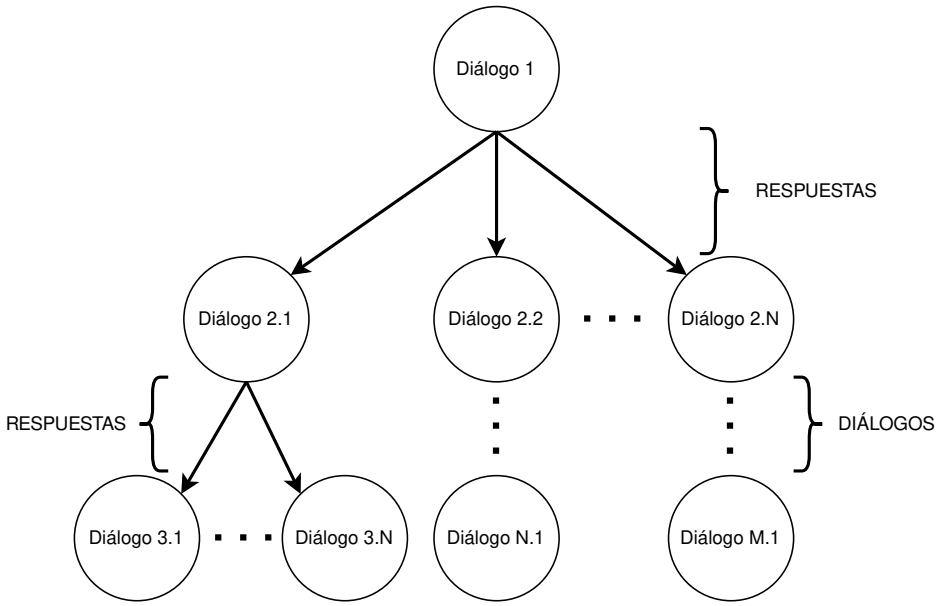


Figura 26: Árbol de diálogos encadenados.

4.4.2. Sistema de misiones

Toda misión en el juego viene dada por un NPC puede ser opcional u obligatoria y contiene un breve mensaje como título, una descripción lo más detallada posible de lo que se ha de hacer y 1 o varios objetivos a completar. Se puede ver una representación simple a continuación (ver Figura 27)

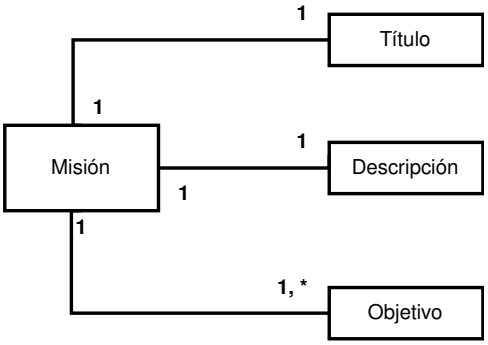


Figura 27: Composición de una misión.

Los objetivos pueden ser de dos tipos: **objetivos de recolección** donde el jugador debe recoger varios objetos y **objetivos de hablar** donde el personaje del jugador interactúa con NPCs con diálogos temporales para dicha misión donde se pueden recibir objetos o entregar objetos opcionalmente a dichos NPCs. El sistema admite una combinación de los dos tipos de objetivos.

4.4.3. Inventario del jugador

El inventario está formado por dos estructuras:

La estructura principal es el inventario Global, en él se almacenan los objetos recolectados por el jugador a lo largo de la partida que no se han devuelto a ningún NPC a cambio de monedas. También se almacenan los objetos le han entregado los NPC al jugador.

La segunda estructura es el inventario temporal que se usa en las misiones cuando hay que recoger X elementos, para hacer que parta de 0 en la recolección. También en el inventario temporal se almacenan los NPC con los que hay que hablar a lo largo de una misión (de los NPC es necesario saber que NPC n mandó al jugador a hablar con el NPC q) (ver Figura 28).

Por ejemplo si el jugador ya ha recogido de otra misión 2 botellas de agua y en una misión nueva le piden que recoja 1 botella de agua, habría un conflicto puesto que el ya tiene 2 en su inventario, habría que empezar a contar de 0 hasta tener esa cantidad de 1.

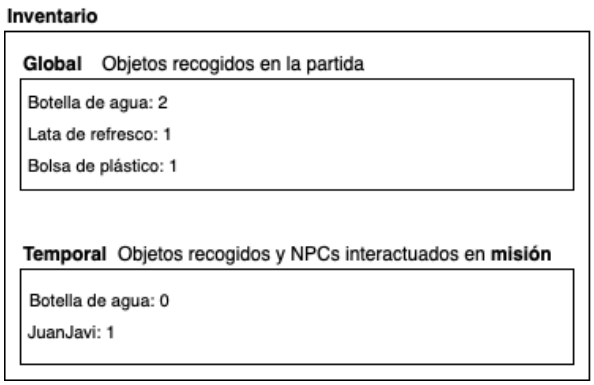


Figura 28: Composición del inventario del jugador.

5. Diseño del software

En este apartado se hablará de: los diagramas de clases UML del proyecto, cómo se ha diseñado el Player a nivel de software, la justificación de los patrones de diseño y la persistencia de datos y la autenticación usando Firebase.

5.1. Diagrama de clases

La Figura 30 muestra el diagrama de clases UML que controlan los menús (Login, Menú principal, Ranking, Habilidades, Selector de nivel) fuera de la partida, actuarían como controllers, aunque algunos componentes de esas escenas tienen sus propios scripts independientes. No están asociados entre ellas porque corresponden a escenas distintas.

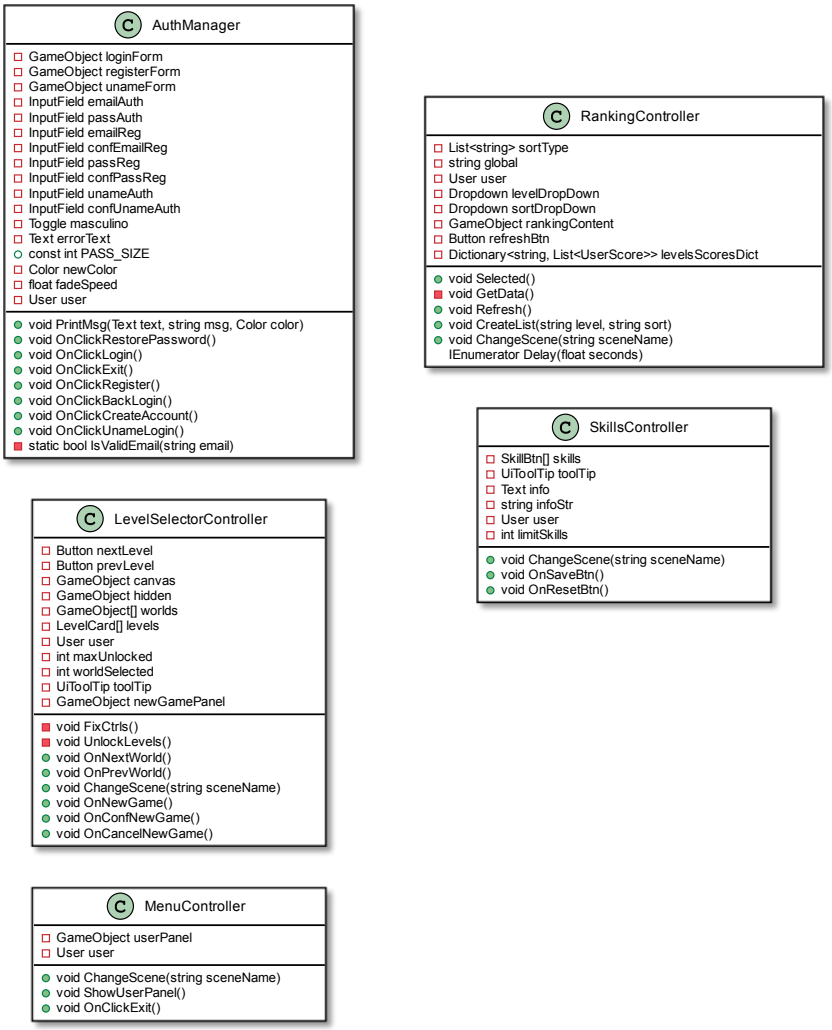


Figura 29: Diagrama UML de los controladores de los Menús.

La Figura 30 muestra el diagrama de clases UML simplificado, con las clases esenciales para una partida. Lo ideal habría sudo poder verlo entero, pero eran demasiadas clases con demasiados atributos y métodos. Por tanto se ha optado por poner las clases más esenciales de una partida. El resto de clases se pueden consultar en el apéndice **D**.

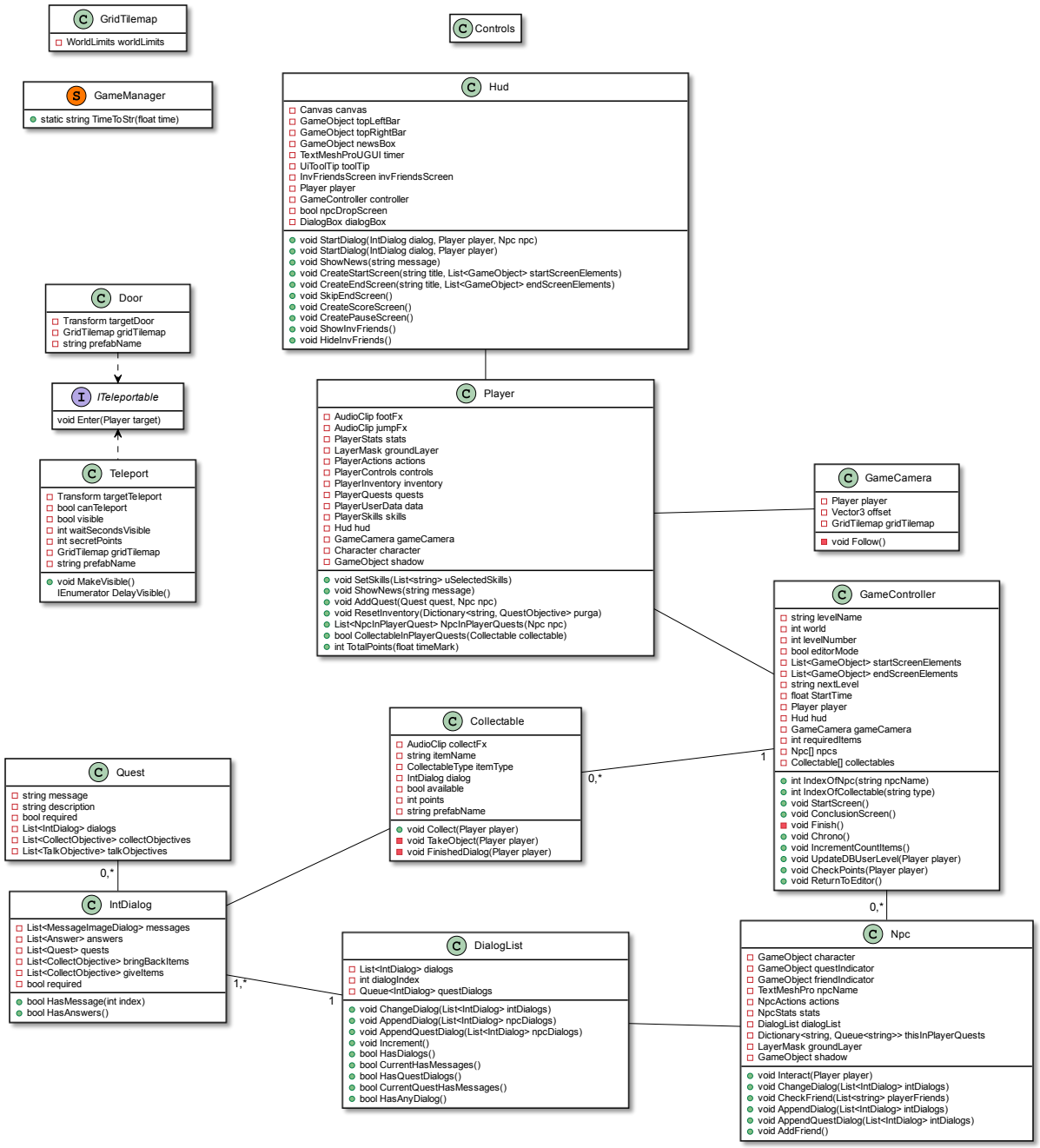


Figura 30: Diagrama UML de la partida (clases esenciales).

Se pueden consultar las clases con todos sus atributos y métodos (por bloques) en el apéndice **D**.

5.2. Arquitectura del jugador

Controles

Los controles del jugador están en una función de una clase *PlayerControls* que se encarga de comprobar si se ha pulsado alguna tecla, ésta se llama desde el Update de Player.

Estado

El estado del jugador se gestiona desde la clase *PlayerStats*, donde se guarda y se consulta su estado. Esto es, la dirección a donde está mirando, la velocidad de andar, la velocidad de correr o si esta haciendo dash (embestida) entre otras cosas.

Acciones

Para gestionar las acciones del jugador tales como movimientos, interacción, gestionar si cae o salta, se utiliza la clase *PlayerActions*.

Se desea que no se pueda saltar más de una vez (dos si tiene la habilidad de doble salto) hasta que el jugador toque el suelo. Para detectar la colisión se hace con un BoxCast (en IsGrounded()), ya que RayCast falla al ser un solo rayo más estrecho. Se decidió que todos los objetos que puede pisar el jugador sean añadidos a una capa Ground, puesto que para el BoxCast o el RayCast necesitamos de parámetro la capa.

Habilidades

Las habilidades del jugador se gestionan con booleanos en la clase *PlayerSkills*, básicamente al empezar una partida se comprueba qué habilidades tiene y en base a esa comprobación se activan booleanos que se usaran para comprobar cuando puede usar una habilidad o no. Si un jugador no tiene habilidades especiales seleccionadas, solo podrá desplazarse normalmente sin poder usar ninguna.

Datos

En *PlayerUserData* se almacenan y se gestionan los datos de la base de datos del usuario así como el score del nivel de la partida en curso.

Inventario

El inventario del jugador está formado por dos diccionarios, uno sirve de inventarió temporal (ver apartado **4.4.3**, que explica el diseño con más detalle) para el transcurso de las misiones, el otro es el inventario global de lo que tiene el propio jugador guardado, se gestiona en la clase *PlayerInventory*.

Misiones del jugador

Las misiones están formadas por dos listas, una de misiones activas y otra de misiones completadas, estas misiones se construyen usando de base las misiones ScriptableObject que se incluyen en los diálogos, se gestiona en *PlayerQuests*.

5.3. Patrones de diseño

El GameManager que gestiona la sesión se ha decidido hacer singleton, se instancia justo al haber sido autenticado y acompaña a todas las escenas sin destruirse.

Se ha utilizado el patrón observer para detectar cuando hay cambios en atributos, por ejemplo cuando una misión cambia su estado a completada, ésta cambia de la lista de misiones activas a la lista de completadas y así poder mostrar un mensaje en pantalla. Este patrón también se usa a modo de evento para cuando un diálogo se cierra, es útil para hacer que los mensajes de misión completada se muestren justo después de haber interactuado con el objetivo en vez de que se muestre en el mismo momento de interacción sin haber acabado.

Se desechó el patrón factory[8] para los NPCs puesto que al final iban a estar dentro de la escena y que aunque se instanciara de manera procedural, como son Prefabs de unity solo se tendría que llamar a la función Instantiate().

5.4. Persistencia de datos

Para el almacenamiento de datos se ha optado por usar Firebase accediendo con api rest, es decir se hacen peticiones GET, PUT, POST y DELETE. Para ello se ha optado por usar una librería ya hecha que funciona perfectamente, se llama RestClient [6].

Para aprender como funcionaba la base de datos RealTime Database se ha usado una guía [24] de internet, aunque la URL de los endpoints cambia ya que se ha escogido la base de datos europea. Mientras se seguía la guía anterior se ha consultado el funcionamiento en el manual oficial de Firebase [13].

Al realizarse una peticón a la base de datos, se obtiene una cadena JSON que se ha de parsear. Para ello se ha optado por usar FullSerializer [8]. Tiene algunos problemas, ya que por defecto normalmente se parsearían los datos en un diccionario, pero por supuesto si se hace un diccionario de strings y uno de los datos de la respuesta no es de tipo string, la libreria falla pero no lo notifica. Para solucionar esto se ha optado por hacer modelos de datos (clases) para parsear directamente algunas respuestas en objetos.

Dependiendo del tipo de dato a parsear se convertirá directamente en diccionario o en un tipo concreto de dato. Ya que, por ejemplo, se necesita un diccionario de usuarios donde el diccionario tiene de clave id de jugador y de valor el objeto de la clase modelo User. Esto también pasa para los scores de cada jugador. Por ejemplo, se supone que una petición GET inventada retorna:

```
{
  "id": "1234",
  "value": "test",
  "money": 10.4
}
```

Al parsearlo a Diccionario <string, string>se rompería sin avisar puesto que el valor de “money” no es string.

Entonces para arreglarlo se crearía una clase que tuviera de atributos:

- string id.

[8]El patrón factory consiste en utilizar una clase constructora (al estilo del Abstract Factory) abstracta con unos cuantos métodos definidos y otro(s) abstracto(s): el dedicado a la construcción de objetos de un subtipo de un tipo determinado. Es una simplificación del Abstract Factory, en la que la clase abstracta tiene métodos concretos que usan algunos de los abstractos; según usemos una u otra hija de esta clase abstracta, tendremos uno u otro comportamiento [33].

- string value.
- float money.

5.4.1. Base de datos

En la base de datos se guardan dos tipos de datos JSON. Por un lado se guarda de cada jugador:

- Nombre de usuario.
- Score de socializador.
- Score de maestro.
- Score de disruptor.
- Score de filántropo.
- Lista con información de niveles-mundos superados.
- Lista de NPCs amigos.
- Lista de habilidades escogidas.

Por otro lado, se guarda de cada nivel superado por cada jugador, siempre que sea un valor de Score mejor que el anterior o no hubiera datos:

- Tiempo en superar un nivel.
- Puntos conseguidos en un nivel.
- Veces que se ha completado (independientemente si se ha obtenido o no mejor puntuación).

Se pueden consultar los endpoints en el apéndice **C.1**.

5.4.2. Autenticación

Se trabajará con un TOKEN que habrá que ir refrescando cada hora después de realizar la autenticación. Sirve para verificar que el usuario es correcto y evitar que un usuario pueda modificar datos de otro usuario ajeno.

En un primer momento esta parte estaba basada en una guía [23] de internet para entender como se podría gestionar la autenticación, aunque dicha web utilizaba los endpoints de IdentityToolKit [11] de Google y al final se optó por usar los de la documentación oficial de Firebase [12], que también son de Google.

Como se menciona en el apartado de persistencia de datos, si no se hacen modelos para la conversión de las respuestas que obtenemos al hacer las peticiones, se tiene el problema que alguna petición tiene el tiempo de expiración del token en número en lugar de string y se pierden datos.

Para tener mayor seguridad y evitar que usuarios ajenos pudieran modificar datos de otros usuarios además del TOKEN se han establecido unas reglas de lectura y escritura en la base de datos. Dichas reglas están separadas dependiendo de la parte del documento JSON que se quiera acceder. Un usuario cualquiera podrá hacer lecturas de los scores o usuarios, siempre que esté autenticado, no importa que sean datos suyos o de otros usuarios. Estos datos no contienen información crítica. Por otro lado, un usuario no podrá modificar los datos de los scores de otro usuario distinto ni la información de usuario que no le pertenezca a él mismo.

A continuación se muestra el fragmento en JSON de las reglas de la base de datos:

```
{
  "rules" : {
    "users" : {
      ".read" : "auth.uid !== null",
      "$uid" : {
        ".write" : "$uid === auth.uid"
      }
    },
    "scores" : {
      ".read" : "auth.uid !== null",
      "$level" : {
        ".read" : "auth.uid !== null",
        "$uid" : {
          ".write" : "$uid === auth.uid"
        }
      }
    }
  }
}
```

Se puede consultar los endpoints en el apéndice **C.2**.

6. Implementación del juego

6.1. Tecnología utilizada

Una vez decidido lo que se iba a desarrollar, un juego de plataformas 2D para WebGL, se han estado barajando estos posibles motores candidatos:

- **Unity**: Es gratuito, el lenguaje de programación usado es C#. Es un motor muy usado por indies, estudios pequeños y algunos estudios más serios. Permite desarrollar para: Windows, SteamOS, WebGL, Macos, Linux, iOS, Android, PlayStation Vita/4, Xbos 360/One/Series, Wii U, Nintendo 3DS/Switch, Oculus, Tize, Windows phone.
- **Godot**: Es opensource, gratuito y multiplataforma, permite desarrollar usando un lenguaje script parecido a python, también en las últimas versiones incorpora soporte para programar en C#, C++, Python. Permite exportar para: Android, iOS, WebGL, Macos, Windows, Linux.
- **Unreal Engine**: Es opensource, gratuito y se programa en C++. Muy usado por estudios grandes, hay juegos famosos como Gears of War o Fornite. Permite desarrollar para: Windows, Linux, Macos, Xbox One/Series, PlayStation 4/5/VR, Nintendo Switch, Stadia, Oculus, WebGL, iOS, Android.
- **LibGDX**: Es un framework multiplataforma escrito en Java, C y C++. Se programa en Java. Permite exportar para: Windows, Linux, Macos, iOS, Android, WebGL.

6.1.1. Justificación de la tecnología utilizada

Se ha buscado que el motor tenga una buena GUI[9] y buena experiencia de usuario. Se descartó LibGDX porque es un framework en que no hay interfaz de usuario para editar, es decir, no se puede ver como van quedando las cosas mientras se desarrolla, solamente cuando se hace la ejecución completa. También a LibGDX le faltan herramientas para editar tilemaps.

Aunque se quiera exportar el juego a WebGL, se busca un motor con soporte para el mayor número de plataformas posibles. Godo es el que menos soporte tiene para exportar el proyecto, además que al ser de código abierto y no estar hecho por una empresa hay menos información y menos plugins que añadirle. Es muy probable que se tenga que diseñar cualquier elemento que no venga con el motor base, por eso se ha descartado Godot Engine. Aunque como es código abierto siempre se puede extender fácilmente y recompilarlo.

La mayor parte del tiempo en la carrera se ha programado en Python y Java. Unreal es un motor en el que se programa en C++ (no oficialmente se puede añadir algún lenguaje más), es un motor más potente y óptimo que Unity por usar un lenguaje compilado y no hace falta ningún interprete en medio. No obstante, como Unity se programa en C# y se parece mucho a Java, se ha optado por utilizar Unity. Hay mucha más documentación para Unity que para Unreal debido a que muchos más juegos independientes lo utilizan, incluso gente que nada más se dedica al diseño y no sabe de programación.

También mencionar que además Unity dispone de una buena Asset Store donde conseguir paquetes, plugins y assets necesarios para no tener que hacerlos de cero.

Para el diseño de personajes, se ha proporcionado por parte de las directoras del proyecto un paquete de assets llamado HeroEditor [10].

[9] Graphical User Interface o Interfaz Gráfica de Usuario.

6.2. Arquitectura del sistema

La arquitectura de este proyecto consta de dos elementos principales: El videojuego EcoSea y el servicio de autenticación y gestión de la base de datos Firebase. La autenticación se realiza en un servicio de Firebase muentras que el acceso y operaciones CRUD de los datos del jugador se realizan en el servicio RealTime Database. En la Figura 31 se muestra el esquema de la arquitectura que se ha implementado. Como se puede observar el jugador interacciona con la base de datos desde el juego, ya sea para crear una cuenta, conectarse o actualizar sus datos (scores, perfil de jugador, amigos, habilidades escogidas, género del jugador). La interacción se realiza mediante peticiones HTTP y se recibe una respuesta en formato JSON.

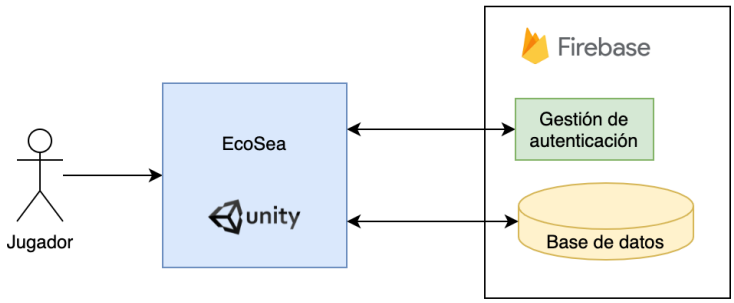


Figura 31: Arquitectura del sistema.

6.3. Implementación con Unity

En un primer momento se pensó en utilizar Modelo-Vista-Controlador. Aunque es posible utilizarla, no termina de encajar bien debido a que existen los componentes GameObject (un GameObject es una clase base para todas las entidades de una escena de Unity, es usado para representar cualquier cosa que puede existir en una escena. Estos no logran nada por sí mismos pero funcionan como contenedoras para componentes, que implementan la verdadera funcionalidad [30]) de Unity que tienen características propias y muy interesantes, es un caso similar a una aplicación desarrollada con Android Studio donde encaja mejor un Modelo-Vista-Presentador.

Al final se ha optado por hacer un desarrollo libre donde cada GameObject puede tener o no un script el cual le proporcionara la funcionalidad deseada al componente. Aún así si que se ha decidido usar un controlador para cada escena del proyecto, aunque éste no tiene la misma responsabilidad ya que como es bien sabido, los objetos de Unity "MonoBehaviour", tienen sus métodos: Awake, Start, Update.

Hasta ahora los métodos Update, Star, Draw y similares se tenían que ejecutar explícitamente desde un hilo paralelo que sería el hilo del juego que actuaría como controlador de la partida. En Unity no hay un hilo de juego explicito, y a diferencia de como se ha programado hasta ahora, estas funciones se ejecutan automáticamente dentro del motor de Unity, solo se puede decidir desde un controlador si desactivar un Update o parar el tiempo por ejemplo.

6.3.1. Elementos de la partida

Todos los niveles del juego incluidos los del modo editor cuentan con una serie de GameObject que son fijos: GameController, GameCamera, Hud, Player, Grid, NPCs, Collectables, Doors, Teleports, Walls, Decorations, Backgrounds, ademas del GameManager (éste último está en todas las escenas del juego). Se decidió usar esta estructura fija ya que el modo editor necesita saber como cargar un nivel para

editarlo, por tanto deben tener cosas en común. Para saber en qué GameObject se encuentran los NPCs, puertas, teletransportes y el resto de objetos de la escena (ver Figura 32).

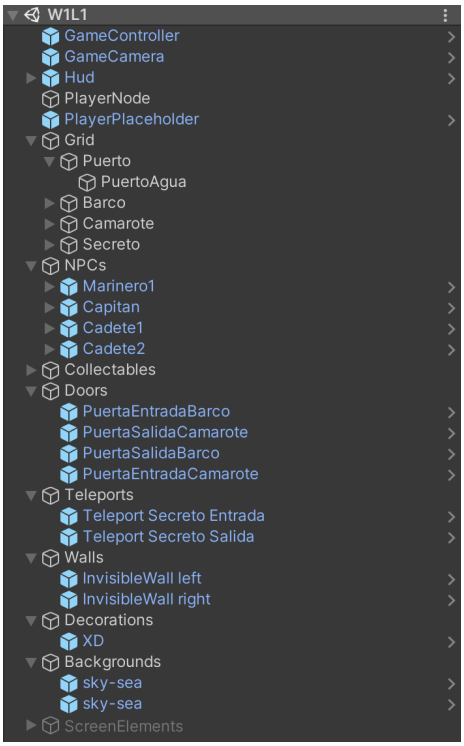


Figura 32: Estructura de GameObjects del nivel 1 del mundo 1.

- **GameController** es la capa más externa que decide cuando una partida llega a su fin, se usa también para asignar componentes entre sí que no estén relacionados desde el inspector de Unity, como por ejemplo asignarle al jugador el HUD o asignarle a la cámara el jugador que ha de seguir.
- **GameCamera** es la cámara principal que tiene el objetivo de seguir al jugador en todo momento, ésta tiene asignada un objeto de tipo GridTilemap lLas secciones de suelo que componen un nivel están formadas por tilemaps, a los tilemaps se les ha asignado un script llamado GridTilemap), este objeto contiene las coordenadas XY límites de la zona, cuando un jugador traspasa una puerta o teletransporte que le lleva a otra zona, la puerta que también tiene un GridTilemap de la zona en la que está se le asigna a la cámara. De esta forma la cámara no seguira moviendose cuando el jugador llegue a un limite.
- **Hud** es un objeto que contiene el Canvas para mostrar los objetos que son de tipo UI, desde el Hud se mostrará el timer de la partida, las pantallas de intro, conclusión, pause, inventario y diálogos así como todo tipo de elementos que sean UI en una partida.
- **Player** es un objeto con físicas, es el personaje controlable por el jugador y tiene un RigidBody2D y un BoxCollider2D. Además para los assets utiliza el pack HeroHeditor que se ha proporcionado por parte de las directoras para la realización de este trabajo. En las escenas en vez del player hay un GameObject llamado playerPlaceHolder, esto es debido a que el player se instancia ingame para dar la posibilidad de entrar como personaje femenino o masculino, y en un futuro si se deciden añadir más modelos de personajes.
- **Grid** es el GameObject donde estarán los tilemaps, hay tilemaps que contienen otros tilemaps a modo de background.
- **NPCs** es un GameObject donde se guardarán los Prefabs instanciados de todos los NPC del nivel, los NPC como el jugador tienen también físicas, se han tenido que hacer dos físicas para

ellos: una con RigidBody2D para que tengan gravedad pero que sean traspasables por el jugador y la otra sin ser RigidBody2D que le permita al jugador interactuar con ellos al colisionar y darle a interaccionar.

- **Collectables** contiene todos los objetos que se podrán recoger del escenario, estos no tienen gravedad.
- **Doors** se encarga de almacenar las puertas traspasables por el jugador.
- **Teleports** se encarga de almacenar los teletransportes traspasables por el jugador.
- **Walls** se encarga de almacenar los muros invisibles para impedir que el jugador se salga del escenario y caiga al vacío.
- **Decorations** se encarga de almacenar los objetos de decoración como puestos de mercado, un barco, etc.
- **Backgrounds** se encarga de almacenar los fondos de cada sección que tiene el nivel.

6.3.2. Componentes de Unity utilizados

En unity como componentes 2D existen dos tipos: UI y Objetos 2D. La diferencia principal es que los UI se colocan en un Canvas[10] y los 2D se colocan en el mundo. También mencionar que para gestionar las coordenadas de posición y tamaños, los componentes UI utilizan el componente RectTransform, mientras que los componentes 2D utilizan el componente Transform.

UI - Interfaz

A continuación se enumeran los componentes de la UI que han sido necesarios para desarrollar las pantallas de login, registro, habilidades, menú principal y HUD del jugador:

- **Canvas** y **Canvas Scaler**: Se utiliza en los GameObjects que van a contener otros GameObjects con componentes de la UI. El Canvas puede tener ajustes para utilizar las coordenadas de mundo o incluso para ajustarse a la resolución y coordenadas que se obtienen con la cámara.
- **Text** y **TextMeshProGUI**: Se utilizan para mostrar texto en todo el juego: menús, avisos dentro del juego e incluso el texto de los botones.
- **Button**: Los botones se utilizan para la interacción del jugador con las distintas opciones del juego, navegación o acciones.
- **LayoutGroups** - **Horizontal**, **Vertical** y **Grid**: Los layouts se utilizan para agrupar elementos de la UI, ya sea texto, botones y otros layouts.
- **Image**: se utiliza para mostrar imágenes que no se trataran como Sprites.
- **Content Size Fitter**: Se utiliza para ajustar el tamaño de los componentes
- **Scrollbar** y **Scroll Rect**: Se utiliza para cuando se tiene una lista de objetos que puede tener un tamaño variable y puede sobrepasar el área donde se muestra.
- **Rect Mask 2D**: Se utiliza conjuntamente con el Scroll para hacer invisibles los objetos que se salen del contenedor. Así la única manera de visualizarlos es interactuar con el Scroll.

[10]Es el área donde todos los elementos UI deben estar. El Canvas es un Game Object con un componente Canvas en él, y todos los elementos UI deben ser hijos de dicho Canvas. Si el Canvas no existe y se crea un objeto UI, se crea automáticamente el Canvas. También utiliza el objeto EventSystem para ayudarle al sistema de mensajes.

Objetos 2D

A continuación se enumeran los componentes 2D que se han usado para crear los distintos niveles del juego:

- **Tilemap**[11] y **Tilemap renderer**: Se usan para pintar el escenario completo, a nivel de suelo e incluso a nivel de background.
- **RigidBody2D**: Se usan para las físicas de los personajes, utilizar y además añadir si se desea gravedad basada en la física de Newton.
- **Colliders - BoxCollider2D**, **Tilemap Collider2D**, **Composite Collider2D**: Se puede complementar con RigidBody2D o usarse sin éste. Son las cajas de colisión que tendrán NPCs, objetos recolectables, puertas, teletransportes y el jugador. Con los colliders se detectan dos tipos de colisiones: la colisión normal y la colisión sin físicas como evento o triggered (si se activa este atributo, permite desencadenar eventos y ser ignorado por el motor de físicas).
- **TextMeshPro**: Se utiliza para poner texto sin usar la interfaz de usuario o el Canvas, como por ejemplo el nombre del NPC encima de él.
- **Sprite** y **Sprite Renderer**: Se utilizan para añadir imágenes 2D a los GameObjects, es usado por todos los elementos visibles del juego.
- **Camera**: Se utiliza para poder visualizar el juego y para seguir al jugador durante la partida.
- **Audio Listener**: Se utiliza para los efectos FX de audio: andar, saltar y recoger un objeto.
- **Sorting Group**: se utiliza para asignar el orden de prioridad de muestreo de imágenes, este componente afecta a los GameObject hijos. Por ejemplo queremos que el player se dibuje delante de las puertas y no al contrario, pero el player tiene muchos componentes de Sprite con distinto orden proporcionados por el HeroEditor, entonces este componente le asigna al GameObject y sus hijos el orden correcto.

6.4. Implementación de diálogos, misiones y habilidades

Los diálogos, misiones y habilidades se han desarrollado utilizando la clase ScriptableObject, un ScriptableObject es un contenedor de datos que puede utilizar para guardar grandes cantidades de datos, independientemente de las instancias de clase. Uno de los principales casos de uso de ScriptableObjects es reducir el uso de memoria del proyecto evitando copias de valores. Esto es útil si el proyecto tiene un Prefabs con estos objetos asignados [31]. Para resumir no son clases normales ni clases MonoBehaviour tradicionales de Unity.

Diálogos

En un primer momento los diálogos se plantearon como clases normales, pero había dificultades para asignárselos desde el editor de Unity a los escenarios base del juego, la única forma era por código.

Se empezaron a hacer pruebas haciendo que los diálogos fuesen MonoBehaviour, ya que estos sí podían asignarse desde el inspector y desde código. La desventaja es que son más pesados puesto que tienen las funciones de Update, Start... Además de todas las funcionalidades heredadas de estos

[11] Un Tilemap es una cuadrícula de mosaicos que se utiliza para crear el escenario de un juego. Hace posible dibujar el mapa pintando con teselas en una cuadrícula, lo que es mucho más rápido que colocar Sprites individuales uno por uno. Además que se le pueden asignar las físicas de forma mas óptima y fácil porque solo hace falta asignárselo al escenario pintado y no a cada elemento por separado [17].

tipos de objetos. También al ser utilizados como un árbol de GameObjects, Unity daba warnings y problemas relacionados con profundidad, por lo que investigando se dio con los ScriptableObjects [3].

La siguiente imagen (ver Figura 33) representa como se ve un diálogo desde el inspector de Unity. El cuál está formado una lista de un mensaje sin imagen, tiene tres respuestas que enlazan con otros diálogos, además cada respuesta cuenta con una puntuación distinta que se usa para la tipología de jugador, éste diálogo no es requerido ni tiene misiones, ni le entrega o recoge objetos del inventario al jugador.

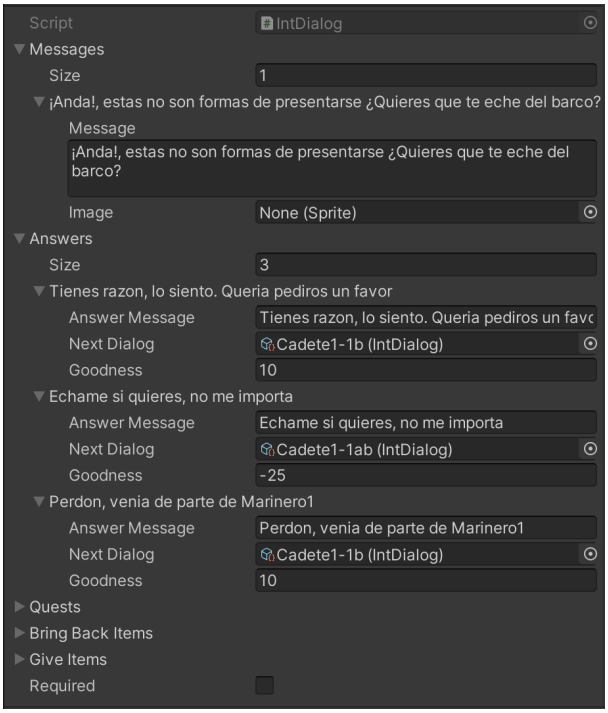


Figura 33: Ejemplo de ScriptableObject de un diálogo.

Se usa la misma estructura para los diálogos con NPCs y con objetos recolectables, la principal diferencia es que el objeto una vez se recoge es destruido, por tanto solo puede tener un único diálogo con la cantidad de mensajes que se requiera.

Un NPC puede tener una cantidad determinada de diálogos, además de que puede tener también diálogos temporales para interactuar con el NPC en una misión en curso. Un NPC siempre tendrá un diálogo en bucle para cuando se vuelva a interactuar con él y siempre será el último que tenga.

Los **diálogos principales** del NPC estarán en una lista con un índice que ira aumentando, los diálogos temporales que solamente aparecerán mientras el jugador tenga una misión, estarán almacenados en una cola que se irá vaciando mientras el jugador siga interactuando.

Los **diálogos temporales** de misión tendrán preferencia sobre los normales a la hora de interactuar con el NPC.

La idea es que el NPC tendrá unos mensajes pero si el jugador tiene una misión que es ir a hablar con ese mismo NPC, ese NPC tendrá que decirle un mensaje relacionado con el contexto de la misión que el jugador tiene y no lo que diría habitualmente.

Por ejemplo, suponiendo que un NPC tiene los diálogos:

- Hola, Me llamo Juan.
- Hace buen tiempo.

En la primera vez que se hable con el NPC se mostrará: *Hola, Me llamo Juan*. El resto de veces que se hable con él se mostrará *Hace buen tiempo*.

Pero si otro NPC llamado Javier te da una misión que es hablar con Juan, a Juan se le añaden unos diálogos temporales para entrar en el contexto de esa misión, al hablar por tanto con Juan se mostrarán los diálogos temporales, por ejemplo: *¿Te ha dicho Javier que me traigas estas bolsas de plástico?*, *Ya le dije que lo hiciera antes de tirarlas*.

Una vez acabados los diálogos temporales se volverán a mostrar sus diálogos normales.

Los diálogos tendrán opcionalmente posibles respuestas que no son más que otros diálogos, haciendo que se forme un árbol o hasta un grafo donde las aristas son las respuestas y estas tienen una puntuación que se usará para el tipo de jugador.

Misiones

En un primer momento las misiones estaban incrustadas dentro de los diálogos, aunque eran clases distintas, las misiones no eran ScriptableObjects pero si objetos serializados[12]. Desde el inspector de cualquier objeto diálogo se iban construyendo las misiones.

Como los ScriptableObjects tienen cierta persistencia de datos, se ha optado por usar estos objetos misiones como modelos de los cuales saldrán en runtime otro tipo de objetos misiones para el jugador. de ésta forma se evita que si se modifica la misión quede alterada la misión original.

Al final se optó porque las misiones fueran ScriptableObjects, esto ayuda a que por un lado creas objetos de diálogo y por otro lado objetos de misión. Cuando esta todo creado, a un diálogo se le asignan las misiones pertinentes desde el inspector (o a código) igual que a un diálogo se le asignan otros diálogos como respuestas.

La siguiente imagen (ver Figura 34) representa como se ve una misión desde el inspector de Unity.

[12]La serialización es un proceso que consiste en convertir un objeto en una secuencia de bytes para almacenarlo o transmitirlo a la memoria, a una base de datos o a un archivo. Su propósito principal es guardar el estado de un objeto para poder volver a crearlo cuando sea necesario. El proceso inverso se denomina deserialización. [19]. Los ScriptableObjects son objetos serializables.

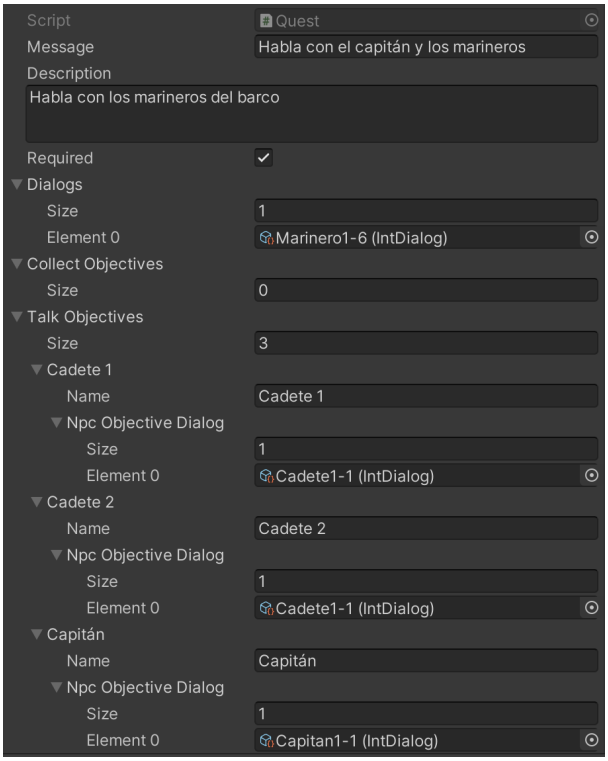


Figura 34: Ejemplo de ScriptableObject de una misión.

Como se mencionó antes, las misiones que tiene el jugador no son ScriptableObject, sino que son un tipo de objeto nuevo y distinto que se crea con los datos proporcionados por este. Para resumir, el objeto ScriptableObject es como un objeto almacenado cuya información se utilizará para hacer un objeto válido para el jugador. Esto es así porque en un primer momento no existía ese otro tipo de objeto misión del jugador y durante el transcurso de una partida **si el ScriptableObject se modificaba quedaba modificado durante todo el runtime del juego**. Por tanto se tuvieron que crear objetos que si se van a **modificar**, como por ejemplo, el **estado de la misión**, si es una misión que esta completa o no. Una misión no se completa hasta que todos sus objetivos se cumplen.

Habilidades

La implementación de las habilidades se ha realizado también con ScriptableObjects. (ver Figura 35) se puede observar como es el ScriptableObject de una habilidad visto desde el inspector de Unity.

Como aclaración, los ScriptableObjects para habilidades solamente se usan a modo de fichero de almacenamiento, donde incluyen: un nombre base de la habilidad, el nombre de la habilidad, una descripción detallada de la habilidad, la rama de talentos a la que pertenece[13], la puntuación de la rama de talentos necesaria para poder desbloquear/usar la habilidad y una imagen de la habilidad para mostrarla en la pantalla de habilidades, pause y nueva habilidad desbloqueada.

Para el transcurso de la partida no se utilizan estos ScriptableObjects de habilidades. Justo al empezar una partida se comprueba una lista de cadenas de texto que son los nombres base de las habilidades que se han elegido en la pantalla de habilidades, en base a esa lista se activan unos booleanos del jugador. Siempre que un booleano de habilidad esté activo se podrá usar dicha habilidad (siempre que no sea una habilidad pasiva).

[13]Cada rama de talentos es de un tipo de jugador: socializador, master, filántropo, disruptor.

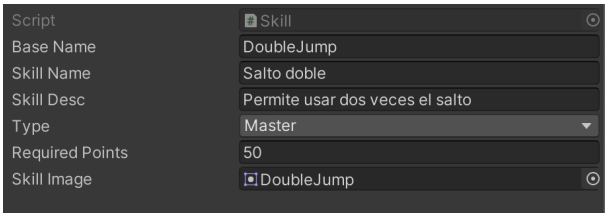


Figura 35: Ejemplo de ScriptableObject de una habilidad.

6.5. Persistencia de los niveles y serialización

La idea original del proyecto era que el jugador, además de poder probar los niveles que crea, puede subirlos a una base datos. De esta manera tanto profesores como alumnos podrían tener acceso a los niveles del resto de usuarios. Esto sería especialmente útil para crear un nivel para una sesión concreta y que los alumnos accedieran a ella mediante la plataforma. La parte de serializar y guardar los niveles sería llevado a cabo por Arnau Rovira y esta parte del proyecto se encargaría de recuperar e interpretar los datos serializados para montar un nivel y poder jugarlo, es decir, cargar un nivel serializado.

A la hora de comenzar con esta parte del proyecto aparecieron varios problemas. Para entenderlos, hay que conocer antes cómo funciona la serialización en Unity:

- **Serialización de objetos, ScriptableObjects y Prefabs**: Unity tiene herramientas para serializar, pero sólo funcionan correctamente con tipos de datos básicos. Al serializar un ScriptableObject como podría ser un diálogo del proyecto, los campos de texto se serializan bien, pero los elementos más complejos como las imágenes no aparecen. Dependiendo del método que se utilice se puede llegar a conseguir que detecte las imágenes, pero sólo es capaz de serializar según su InstanceID, la cual varía en cada ejecución y que, por tanto, no tiene ningún sentido guardar.
- **Serialización de Prefabs**: Hasta el 2017 se podían serializar Prefabs en Unity, pero con la llegada de los Nested Prefabs [28] en la versión 2018.3 dejó de ser posible. Esta característica permite colocar un Prefab dentro de otro Prefab. Es una herramienta muy útil pero tiene la desventaja de que hace imposible su serialización.
- **Guardar en ejecución**: Hay muchos productos que aseguran que pueden serializar Prefabs, ScriptableObjects e incluso escenas enteras, pero lo que hacen en realidad es guardar una copia en tiempo de ejecución. Es una muy buena herramienta para guardar el juego en un estado determinado, cambiar completamente de escena y luego poder seguir donde se había dejado, pero a la hora de hacer persistir estos datos no aportan ninguna solución.
- **UnityEditor**: Hay maneras de serializar la mayoría de elementos de este proyecto con herramientas de Unity como la API UnityEditor, pero sólo funciona en una ejecución desde el inspector de Unity, es decir, una vez compilado el juego para cualquier otra plataforma deja de funcionar. Las herramientas que se han encontrado para serializar o crear Prefabs que utilizaban funciones de UnityEditor no se pueden usar en un producto que se vaya a compilar, solo desde el propio IDE de Unity.

Ante todos estos problemas, se intentó encontrar alguna solución alternativa utilizando paquetes externos especializados en serialización, pero ninguno acabó resultando seleccionados útil. Los motivos fueron los siguientes:

- **Easy Save** [15]: Sólo permite serializar tipos de datos primitivos. Permite guardar Prefab pero sólo en runtime. Se diseñó para la versión 2017.4.40 donde aún no existían los Nested Prefab.

- **Blartenix Save and Load System** [26]: Aunque el paquete asegura poder guardar cualquier elemento de la partida, sólo lo puede hacer durante la ejecución, al pararla se pierden todos los datos y no hay manera de hacerlos persistir.
- **Odin Serializer** [16]: Este es el paquete más completo con diferencia. Permite serializar tipos de datos básicos y hace un muy buen trabajo serializando elementos que Unity no puede serializar. Sin embargo, sigue existiendo el problema de las imágenes y elementos más complejos. Odin serializa todos estos datos utilizando su InstanceID, y proporciona una interfaz para que el desarrollador pueda adaptar el código de tal manera que unos objetos determinados siempre tengan la misma InstanceID al crearse y que, por tanto, los datos serializados sean siempre válidas. El problema de esta solución es que se debería haber invertido una gran cantidad de tiempo en configurar este escenario y habría sido similar que codificar a mano una serialización en JSON, dado que en este proyecto existen muchos tipos de elementos, la mayoría son complejos y, además, al modo editor se crearán muchas instancias de ellos.
- **SerializerFree** [5]: Se recomendó el uso de este paquete porque se había utilizado anteriormente en proyectos similares, pero de la misma manera que EasySave, se desarrolló en 2017 cuando aún no existían los Nested Prefab, por lo que no se podía hacer uso. Además, este paquete utiliza la Api UnityEditor, que como hemos comentado anteriormente no funciona una vez compilado el código para otras plataformas.

Después de toda la investigación realizada se llegó a la conclusión de que no había ninguna solución válida para la cantidad de tiempo de la que se disponía, por lo que se decidió realizar el modo de testeo y dejar la serialización de niveles como trabajo futuro.

7. Resultados

7.1. Planificación final

A continuación se puede observar la planificación final que se ha ejecutado realmente (ver Figura 36). Esta planificación difiere mucho de la que se pensó que sería inicialmente (ver Figura 1). Ya que conforme se ha ido avanzando han ido surgiendo problemas y se ha tenido que ir modificando el código y el diseño de las clases de sprints anteriores.

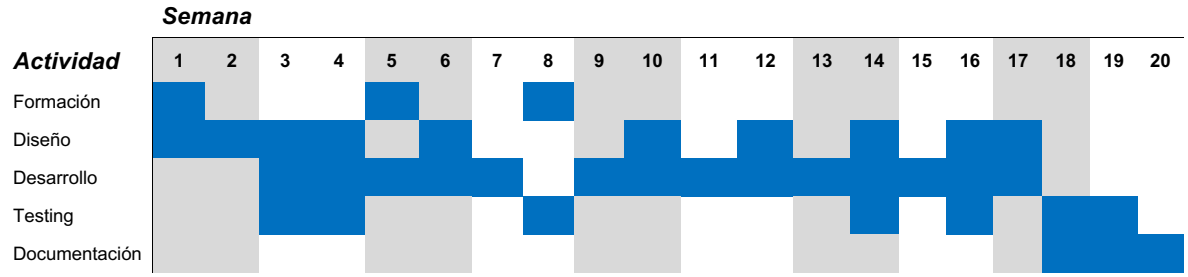


Figura 36: Planificación final del proyecto.

Como se ha comentado, los diagramas de Gantt son medidos en semanas, pero, se ha trabajado en SCRUM con sprints de dos semanas.

El progreso de los sprints será mostrado en dos tablas (tabla 3 y tabla 4) debido al tamaño que ocupa. La siguiente tabla 3 muestra el progreso que ha ido siguiendo el juego de manera resumida en cada sprint desde el principio hasta la mitad de este.

Sprint	Faena realizada
Pre-sprint: **3-10 Feb**	- Investigación de Unity - Aprendizaje de C# - Primera idea del diseño de software
1: **10-24 Feb**	- Diseño del jugador - Movimiento básico - Cámara con seguimiento del objetivo - Testeo de Cámara convencional frente a Cinemachine - Desarrollo básico de los NPCs - Desarrollo básico de puertas y teletransportes - Añadido Canvas y empezada la creación de diálogos
2: **24 Feb - 10 Mar**	- Mejoras en los diálogos + añadido de imágenes a estos - Añadidos los diálogos a los objetos recolectables - Diseño del inventario básico del jugador - Diseño de un sistema de misiones basado en objetivos - Testeo de las misiones - Prueba de controles alternativos nuevos de Unity frente a los convencionales - Creación de un primitivo nivel 1 con un Grid y tilemaps - Problemas con ScriptableObjects y persistencia - HUD del jugador con timer
3: **10-24 Mar**	- Vuelta a los controles tradicionales de Unity - Arreglos de bugs de sprints anteriores - Añadida puntuación al atravesar teletransportes invisibles - Añadido flip con NPC al hablar - Diseño de pantallas finales: estadísticas, nuevas habilidades - Diseño de la pantalla de introducción y final de nivel - Investigación de autenticación y persistencia con Firebase usando api rest - Creación alternativa de objetos misión de jugador generados a partir de los objetos misión de los ScriptableObjects - Utilización de Eventos para comprobar que una lista se ha modificado para mostrar mensaje: amigo añadido, misión añadida, misión completada - Login y Registro empezado
4: **24 Mar - 7 Abr**	- Testing de persistencia: creación de usuario y sus datos - Testing del token de sesión - Investigación y creación de reglas para la base de datos Realtime Database de Firebase - Creación de modelos para parsear consultas a Firebase - Aplicación de Assets del HeroEditor al jugador y a los NPCs - Testeo de animaciones de los personajes - Creación de icono de diálogos disponibles para los NPCs - Creación de pantalla de selección de nombre de usuario

Tabla 3: Tabla de sprints y trabajo realizado, primera mitad.

La siguiente tabla 4 muestra el progreso de desarrollo del juego desde la mitad de éste hasta su finalización.

Sprint	**Faena realizada**
5: **7-21 Abr**	- Creación de la pantalla de Rankings de niveles de los jugadores ordenados por tiempo o por puntuación - Creación del GameController y GameManager - Desarrollo del sistema y pantalla de pause - Arreglos en bugs de las puertas que estaban pegadas a paredes - Comentarios en el código añadidos - Añadida la posibilidad de cerrar sesión - Añadido el nombre del nivel - Diseño de pantalla de habilidades - Añadidos *tooltips* a habilidades
6: **21 Abr - 5 May**	- Pantalla de habilidades acabada - Añadidos *tooltips* a misiones de la pantalla de pause - Añadida la pantalla de habilidades en pause - Mejoras en diálogos: los diálogos tienen una lista con mensajes y cada mensaje puede contener una imagen - Creación de habilidades especiales para el personaje - Creación de pantalla de inventario y amigos NPCs - Cooperación con el TFG del modo editor
7: **5-19 May**	- División del escenario en secciones - Creación de Prefabs nuevos de NPCs - Finalización del nivel 1 del mundo 1 - Diseño del selector de niveles - Creación del nivel 2 del mundo 1 - Habilidades especiales acabadas - Cambios en *tooltips* de habilidades para indicar tecla de uso - Añadidos nuevos tilemaps - Creación de Prefabs nuevos de coleccionables
8: **19 May - 2 Jun**	- Terminado el nivel 1 del mundo 2 - Añadido el nombre del NPC encima y un icono de amigo - Desarrollo de limites de sección de cada nivel para la cámara - Finalización del selector de nivel - Cooperación con el TFG del modo editor - Comienzo de la memoria del TFG - Añadidas pantallas de intro y conclusión a los niveles 1 y 2 del mundo 1
9: **2-16 Jun**	- Testing - Memoria TFG - Diagramas UML finales - Arreglos finales de errores - Comentarios del código
Últimos días: **16-20 Jun**	- Memoria TFG

Tabla 4: Tabla de sprints y trabajo realizado, segunda mitad.

7.2. Resultados visuales del juego

En este apartado se mostrarán los resultados visuales finales obtenidos en EcoSea, se mostrarán algunas pantallas del juego, no todas. Se ha intentado seguir el estilo de los prototipos lo más fielmente

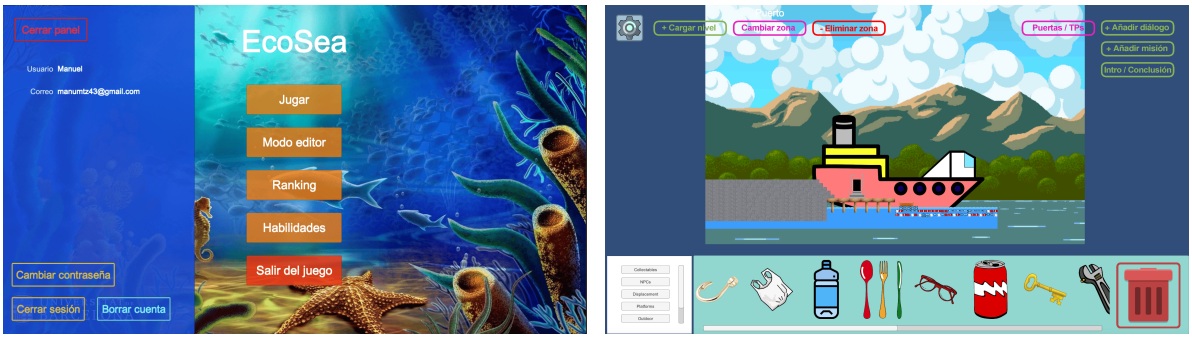
posible.

En la primera imagen (ver Figura 37a) se puede comprobar que ha quedado muy similar a la del prototipo (ver Figura 13a). A la pantalla del menú principal (ver Figura 37b) se le ha añadido un fondo de pantalla que representa el fondo del mar.

(a) Pantalla de inicio de sesión. (b) Pantalla del menú principal.

Figura 37: Pantallas de inicio de sesión y menú principal.

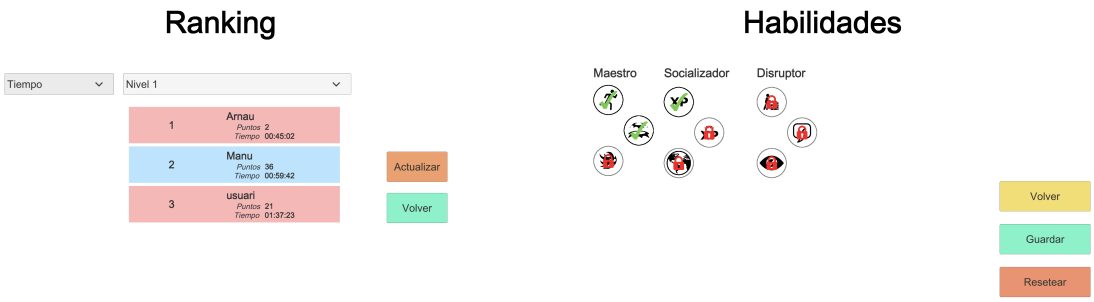
La figura 38a muestra el menú desplegado para ver, borrar, cambiar la contraseña de la cuenta y cerrar sesión. En la figura 38b se muestra el modo editor con un nivel cargado listo para añadirle contenido y probarlo.



(a) Pantalla de ver cuenta. (b) Pantalla del modo editor.

Figura 38: Pantallas de ver cuenta y modo editor.

En la figura 39a se muestra la pantalla de ranking, donde se puede ver la clasificación de los usuarios que han jugado al Nivel 1 ordenado por tiempo. En la figura 39b se muestra la pantalla final de habilidades, con habilidades escogidas (check verde), habilidades bloqueadas (candado rojo).



(a) Pantalla de ranking.

(b) Pantalla de habilidades.

Figura 39: Pantallas de ranking y habilidades.

En la figura 40a se muestra la pantalla de selección de mundo y nivel. En la figura 40b se muestra la pantalla opcional de introducción a un nivel.

(a) Pantalla de selección de nivel y mundo.

(b) Pantalla de introducción.

Figura 40: Pantallas de pause.

En la figura 41a se muestra la pantalla de una partida en curso donde el jugador tiene el inventario abierto y se pueden ver los objetos que ha recogido y los amigos que ha conseguido. En la figura 41b, también se muestra de una partida en curso, un diálogo con múltiples respuestas a escoger por el jugador. También se puede ver el nombre del NPC encima de éste, el icono de amistad y el icono de diálogos pendientes.

(a) Pantalla de inventario.

(b) Pantalla de diálogo con amigo.

Figura 41: Pantallas de introducción y diálogo de múltiples respuestas con amigo.

A continuación en la figura 42 se muestran las pantallas de pause, la primera figura 42a muestra las misiones y la segunda figura 42b muestra las habilidades.

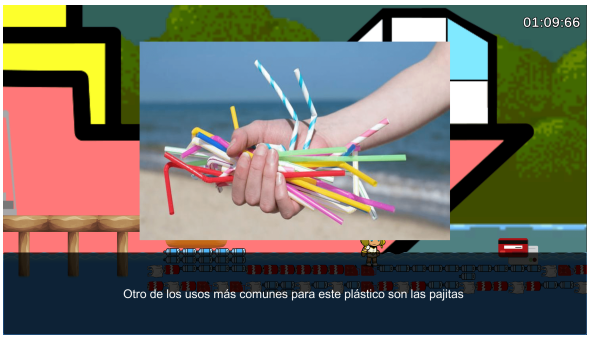
(a) Pantalla de pause con misiones.

(b) Pantalla de pause con habilidades.

Figura 42: Pantallas de pause.

En la figura 43a se muestra una pantalla de diálogo con imágenes del jugador con un NPC. En la figura 43b se muestra una pantalla de diálogo con imágenes al recoger un objeto.

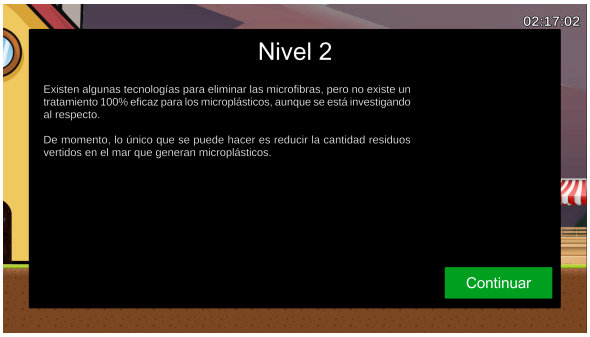
(a) Pantalla de diálogo.



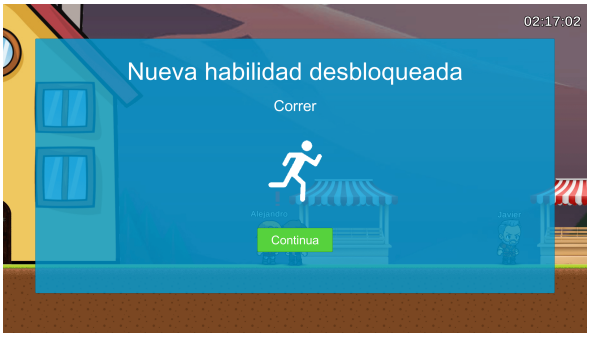
(b) Pantalla de recolección con diálogo.

Figura 43: Pantallas de diálogo y recolección con diálogo.

En la figura 44a se muestra una pantalla con las conclusiones del Nivel 2. En la figura 44b se muestra la pantalla de nueva habilidad desbloqueada, en este caso la habilidad de correr.



(a) Pantalla de conclusión.



(b) Pantalla de nueva habilidad.

Figura 44: Pantallas de conclusión y nueva habilidad.

En la siguiente figura 45 se muestra la pantalla de estadísticas al haber superado un nivel.

Figura 45: Pantalla de estadísticas.

7.2.1. Vídeos del juego

- EcoSea, Demo 1 - Youtube: https://youtu.be/SFiLsmibdVE

8. Conclusiones y trabajo futuro

Conclusiones

En este proyecto se ha desarrollado un juego sobre la ecología marina y sostenibilidad para que a los alumnos de segundo ciclo de educación secundaria (de 14 a 16 años) les sirva de herramienta y para tomar conciencia del daño que se puede ejercer contra la ecología marina. También conseguir que entiendan lo importante que es el reciclaje. Este es un proyecto conjunto desarrollado junto con el estudiante Arnau Rovira.

El juego está dividido en dos partes:

- **Juego Base**: Esta primera parte, que es la que éste proyecto trata, cuenta con:
  - La autenticación o registro de los usuarios a la base de datos, acceso a sus datos y creación de datos nuevos de estos.
  - Navegación base: pantalla principal, pantalla de habilidades, pantalla de ranking, pantalla de selección de nivel.
  - Pantallas en el transcurso de una partida: pause, inventario, intro, conclusiones, recompensas, estadísticas.
  - Niveles básicos para iniciar una partida.
- **Modo editor**: Esta segunda parte está desarrollada por Arnau Rovira, cuenta con:
  - Un modo editor que permite cargar un nivel base.
  - Permite añadir todos los tipos de objetos que se usan en el juego base: puertas, teletransportes, NPCs, decoraciones, objetos recolectables, jugador masculino/femenino, etc.
  - Una interfaz para crear diálogos y asignárselos a objetos o NPCs.
  - Una interfaz para crear misiones y asignárselas a diálogos.
  - Una interfaz para interconectar puertas y teletransportes.
  - Una interfaz en la que se puede teletransportar de una parte del nivel a otra para ir más rápidamente.

Para completar un nivel el jugador tiene que completar las misiones obligatorias y leer los diálogos requeridos de los NPCs. Hay algunos diálogos que se pueden evitar, así como misiones opcionales que no tienen porque hacerse para superar un nivel.

El jugador mientras interactúa con los NPCs y realiza ciertas acciones (descubrir teletransportes invisibles, hacer misiones opcionales, etc) desbloqueará habilidades nuevas al terminar un nivel, que le permitirán facilitar su jugabilidad.

Para comenzar el desarrollo se hicieron varias reuniones seguidas a principios de febrero, después se acordaron reuniones cada dos semanas y se empezó a trabajar con metodología agile, haciendo sprints de dos semanas. Las reuniones siempre se hacían con las dos directoras del trabajo final Anna Puig, Inmaculada Rodríguez y con el compañero Arnau Rovira.

De las ideas planteadas al principio del proyecto, las que no se han podido cumplir por falta de tiempo son:

- Instanciación de NPCs y objetos recolectables proceduralmente según la tipología de jugador.
- Generación de diálogos diferentes según la tipología de jugador.
- Instanciación de zonas secretas extras proceduralmente según el tipo de jugador.

- Banda sonora y más diseños para mejorar el juego.

Todas las anteriores requerían además una adaptación para el modo editor, ya que se usan elementos que en los dos proyectos se comparten.

Del modo editor faltó la parte de serialización junto a la parte del juego base de deserialización de los niveles. La idea era que desde el modo editor se crearían niveles nuevos basados en los ya existentes, se subirían a la base de datos y después cualquier usuario podría bajárselos y jugar.

El resultado final del juego es un resultado positivo en cuanto a las mecánicas de juego planteadas y la jugabilidad, si además se le añaden en algún momento las partes que han quedado en el tintero se puede obtener un resultado mucho mejor, puesto que la parte menos atractiva es la parte gráfica. Aún así ha sido la primera vez que utilizaba C# y Unity, ha servido también para mejorar los conocimientos y experimentar cosas nuevas.

Trabajo futuro

A continuación se detallarán algunos aspectos del juego que no se han podido terminar y que quedan pendientes, así como algunas ideas que se cree que pueden ser interesantes:

- **Añadir una composición musical y SFX**: Aunque hay un par de efectos de sonido, se pueden añadir muchos más. En cuanto a la música, no hay ningún elemento musical y sería interesante añadirle alguno.
- **Niveles adaptados al jugador**: Ahora mismo sólo se personaliza la experiencia de juego con las habilidades especiales de cada tipo de jugador, pero el nivel sigue siendo el mismo. Un gran cambio en el juego podría ser el de implementar un sistema que haga que el nivel también se adaptara a cada tipo de jugador poniendo más retos por los maestros, más situaciones sociales para los socializadores, más rutas alternativas para los disruptores, etcétera.
- **Personalización de avatar**: Ahora mismo el jugador sólo puede escoger si utilizar el modelo de jugador masculino o femenino, pero sería mejor que se pudiera personalizar el avatar para que todo el mundo se pudiera sentir más representado en el juego.
- **Persistencia de niveles**: Tal y como se ha comentado anteriormente, se pueden testear los niveles que se crean en el modo editor pero no se pueden guardar ni subir a ninguna base de datos. Es una de las carencias más importantes de este proyecto, y se cree que poder guardar y cargar los niveles desde cualquier sesión haría de este juego una herramienta mucho más potente.
- **Personalización de elementos ya existentes**: Ahora mismo el usuario sólo puede hacer uso de elementos existentes en el proyecto, pero sería interesante que pudiera subir sus propias imágenes y utilizarlas, ya sea para los diálogos o para los modelos como coleccionables o decoración.
- **Evaluación del juego con usuarios**: haciendo tests de usabilidad y jugabilidad, además de comprobar si el juego cumple con su objetivo principal y de que manera éste profundiza en la conciencia del jugador por tal de cumplir con el reciclaje de plásticos y tomar conciencia por la sostenibilidad.

Bibliografía

[1] U. E. I. Administration. Juegos energy kids. https://www.eia.gov/kids/games-and-activities/. [Online; accessed 01.06.2021].

[2] K. Andersen. How to make a 2d platformer (e04 player script) — unity 2020 tutorial. https://youtu.be/PC5ZK9-zOmI, 2020. [Online; accessed 12.02.2021].

[3] G. Barbosa. Tutorial unity — sistema de diálogo. https://youtu.be/ZGoafs8jiIo, 2017. [Online; accessed 12.02.2021].

[4] R. Bartle. Bartle taxonomy. *mud.co.uk*, 1996. [Online; accessed 01.06.2021].

[5] H. Bayat. Serializer free. https://forum.unity.com/threads/released-free-serializer-free-complete-serialization-solution.463715/. [Online; accessed 05.06.2021].

[6] J. D. N. Cardona. Proyecto26. https://github.com/proyecto26/RestClient, 2017. Last accessed 16.03.2021).

[7] M. creators. Mudlet webpage. https://www.mudlet.org/. [Online; accessed 01.06.2021].

[8] J. Dufault. Fullserializer. https://github.com/jacobdufault/fullserializer, 2014. [Online; accessed 16.03.2021].

[9] Ecoembes. Juegos ecoembes. https://www.ecoembes.com/es/ciudadanos/educa-en-eco/accesible/juegos. [Online; accessed 20.03.2021].

[10] H. Games. Fantasy heroes: Character editor. https://assetstore.unity.com/packages/2d/characters/fantasy-heroes-character-editor-90592. [Online; accessed 02.06.2021].

[11] Google. Identity toolkit for websites. https://developers.google.com/identity/toolkit/web/reference/relyingparty/, 2020. [Online; accessed 16.03.2021].

[12] Google. Api rest de firebase auth. https://firebase.google.com/docs/reference/rest/auth, 2021. [Online; accessed 16.04.2021].

[13] Google. Api rest de firebase database. https://firebase.google.com/docs/reference/rest/database, 2021. [Online; accessed 16.04.2021].

[14] S. Guggiari. *Emergent Personalized Content in Video Games*. Swiss Federal Institute of Technology Zurich, 2019.

[15] T. Hasan and Moodkie. Easy save. https://assetstore.unity.com/packages/tools/utilities/easy-save-the-complete-save-data-serialization-asset-768. [Online; accessed 05.06.2021].

[16] S. IVS. Odin serializer. https://github.com/TeamSirenix/odin-serializer. [Online; accessed 05.06.2021].

[17] A. M. y. l. c. d. G. Juan Linietsky. Using tilemaps. https://docs.godotengine.org/es/stable/tutorials/2d/using_tilemaps.html. [Online; accessed 06.06.2021].

[18] A. Marczewski. *Even Ninja Monkeys like to play*. Bantam, London, 2015. https://www.gamified.uk/user-types/, [Online; accessed 06.06.2021].

[19] Microsoft. Serialización — visual basic. https://docs.microsoft.com/es-es/dotnet/visual-basic/programming-guide/concepts/serialization/. [Online; accessed 06.06.2021].

[20] N. ONU. O nos divorciamos del plástico, o nos olvidamos del planeta. https://news.un.org/es/story/2018/06/1435111/, 2018. [Online; accessed 24.05.2021].

[21] N. ONU. El uso exagerado del plástico durante la pandemia de covid-19 afecta a los más vulnerables. https://www.unicef.org/es/medio-ambiente-cambio-climatico/, 2021. [Online; accessed 22.05.2021].

[22] A. R. Pérez. *EcoSea — Joc 2D de plataformes sobre ecologia marina: Crea el teu propi nivell*. Dipòsit Digital de la Universitat de Barcelona, 2021.

[23] D. Rotolo. Firebase authentication in unity with rest api. https://medium.com/@rotolonico/firebase-authentication-in-unity-with-rest-api-7fc8c1576178, 2019. [Online; accessed 16.03.2021].

[24] D. Rotolo. Firebase database in unity with rest api. https://medium.com/@rotolonico/firebase-database-in-unity-with-rest-api-42f2cf6a2bbf, 2019. [Online; accessed 17.03.2021].

[25] E. Silván. ¿qué es el impacto ambiental y cómo se mide? *Mapfre*, 2020. [Online; accessed 25.05.2021].

[26] B. Studio. Blartenix save and load. https://assetstore.unity.com/packages/tools/integration/blartenix-save-load-system-175048. [Online; accessed 05.06.2021].

[27] H. Studios. Endling steam. https://store.steampowered.com/app/898890/Endling/. [Online; accessed 29.05.2021].

[28] U. Technologies. Nested prefabs. https://docs.unity3d.com/Manual/NestedPrefabs.html. [Online; accessed 05.06.2021].

[29] UNICEF. Medio ambiente y cambio climático. el cambio climático y la degradación del medio ambiente socavan los derechos de todos los niños. https://www.unicef.org/es/medio-ambiente-cambio-climatico/, 2021. [Online; accessed 22.05.2021].

[30] Unity. Unity — scripting api: Gameobject. https://docs.unity3d.com/es/530/Manual/class-GameObject.html, . [Online; accessed 05.06.2021].

[31] Unity. Unity — manual: Scriptableobject. https://docs.unity3d.com/es/530/Manual/class-ScriptableObject.html, . [Online; accessed 05.06.2021].

[32] L. Voz. Como consumidores, somos los mayores responsables por la generación de residuos / pero ayudar no es tan difícil. https://www.lavoz.com.ar/ambiente/como-empezar-desde-casa-las-4-r/, 2010. [Online; accessed 24.05.2021].

[33] Wikipedia. Factory Method (patrón de diseño) — Wikipedia, the free encyclopedia. Wikipedia, ES, . [Online; accessed 05.06.2021].

[34] Wikipedia. HUD (informática) — Wikipedia, the free encyclopedia. Wikipedia, ES, . Online; accessed 05.06.2021.

[35] Wikipedia. Juego Serio — Wikipedia, the free encyclopedia. https://es.wikipedia.org/wiki/Juego_serio, 2021. [Online; accessed 10.06.2021].

[36] Wikipedia. Información sobre herramientas — Wikipedia, the free encyclopedia. https://es.wikipedia.org/wiki/Información_sobre_herramientas, 2021. [Online; accessed 07.06.2021].

A. Manual técnico

A.1. Guía de instalación y requisitos

A continuación se explicará el proceso de instalación del juego EcoSea. Los requerimientos necesarios son:

- **Sistema operativo Windows**: La build y los tests se han hecho con Windows 10.
- **Espacio en disco de 300MB**: El fichero con el ejecutable y los meta-datos pesan 296MB.
- **Prestaciones estándares**: El juego consume aproximadamente un 10 % de la CPU con una gráfica de 1GB de DDR3 y una RAM de 16GB, por lo que no es nada exigente. Con prestaciones como un intel core i5 a 5GHz o un i3 a 3,7GHz con tarjeta gráfica integrada bastaría.

Dicho esto, los pasos para poder jugar a EcoSea Windows son los siguientes:

1. Descargar archivo del enlace: https://drive.google.com/file/d/1UdmiKB-dJBp9QTk5pSu8gnS33uF8IPIu/view?usp=sharing
2. Descomprimir el archivo zip.
3. Ejecutar EcoSea.exe y empezar a jugar.

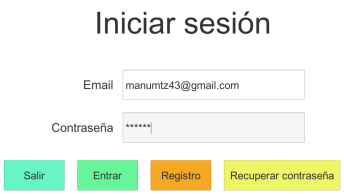
A.2. Guía de usuario

A.2.1. Autenticación

Si durante algún paso hay algún error, se mostrará un mensaje en la parte inferior de color rojo. Si hay algo de lo que informar que no sea un error, será de color verde.

El primer paso es estar autenticado. Para ello se introducirá el email, la contraseña y se pulsará en "**Entrar**".

Si no se tiene cuenta de usuario, pulsando el botón "**Registro**" se puede crear una. El email debe ser válido y no estar en uso, la contraseña debe ser de mínimo 6 caracteres alfanuméricos (ver Figura 46a). Al rellenar los datos se debe pulsar "**Crear cuenta**" (ver Figura 46b).



(a) Formulario de inicio de sesión.

(b) Formulario de registro.

Figura 46: Formularios de autenticación y registro.

Una vez creada la cuenta, esta debe ser activada desde un enlace que llegará al correo electrónico facilitado (ver Figura 47), no se podrá acceder al juego hasta que la cuenta esté activa. Si se pierde dicho correo, al intentar iniciar sesión se mandará otro.

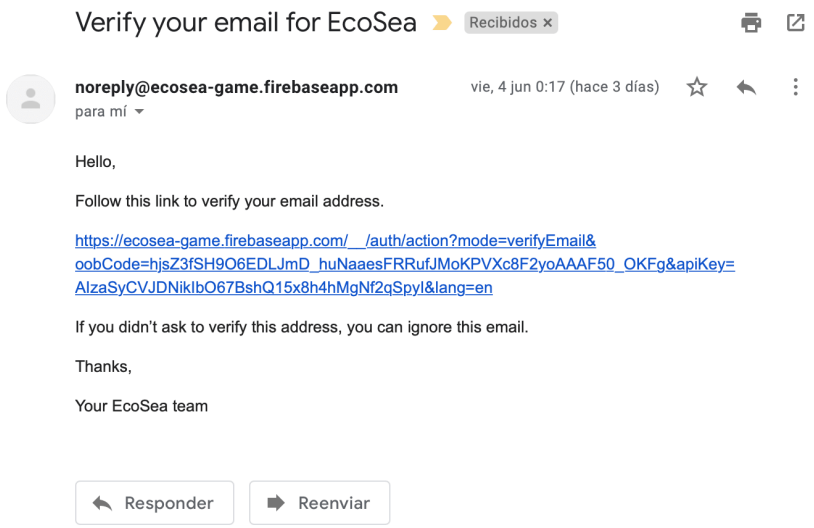


Figura 47: Email de activación de cuenta desde Gmail.

Si existe la cuenta pero se ha perdido la contraseña se puede recuperar pulsando en "**Recuperar contraseña**", llegara un correo electrónico con un enlace para escribir la contraseña nueva (ver Figura 48).

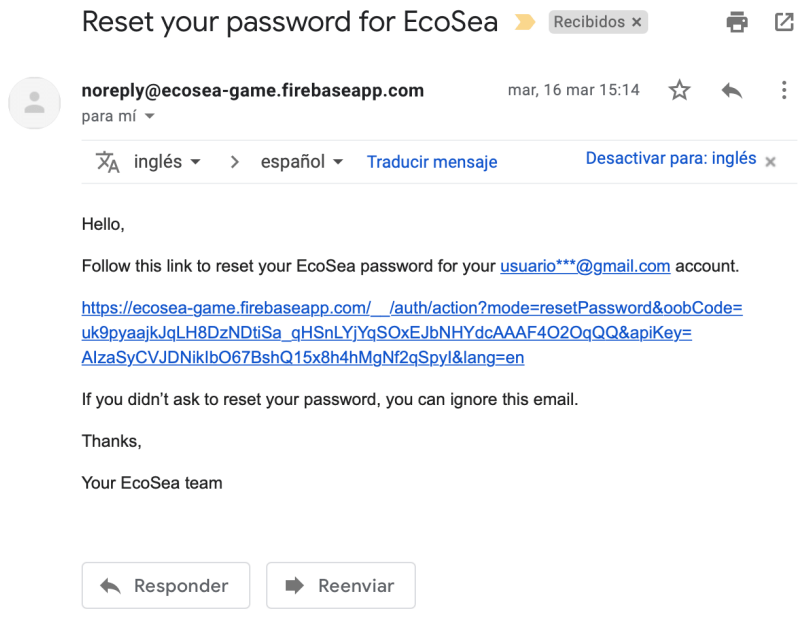


Figura 48: Email de recuperación de cuenta desde Gmail.

Si es la primera vez que se accede al juego, se requerirá un nombre de usuario válido. El nombre de usuario debe ser único. Además se deberá elegir el género de jugador: masculino o femenino. No se podrá cambiar el nombre de usuario ni el género después, la única opción es borrar la cuenta con todo el progreso y volver a crearla, esto es así a propósito. Una vez elegido el nombre de usuario se pulsará en "**Entrar**" (ver Figura 49).

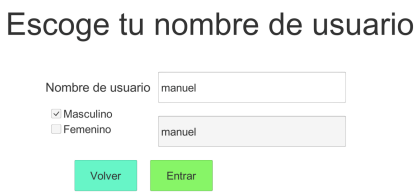


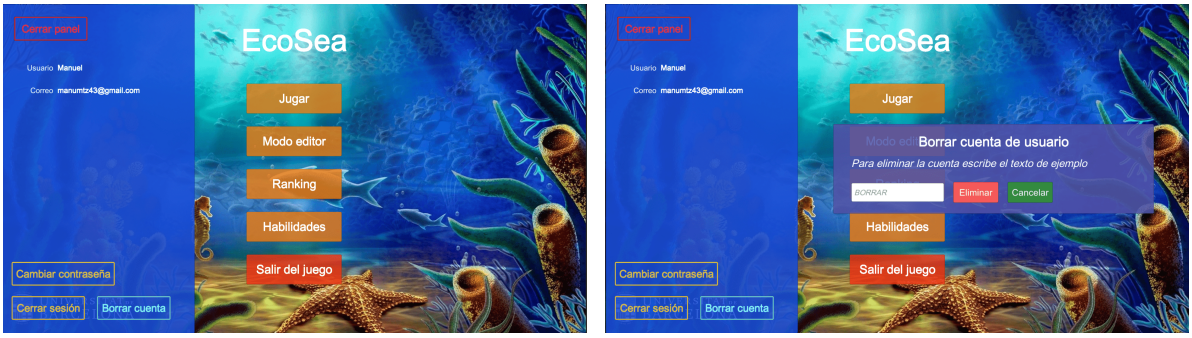
Figura 49: Elección de género y nombre de usuario.

A.2.2. Gestión de la cuenta de usuario

Al abrir el submenú de ver cuenta se mostrará el nombre de usuario, el correo electrónico y además se dispone de varias opciones (ver Figura 50a):

- "**Cambiar la contraseña**": Manda un mail como el de restablecer cuenta (ver Figura 48).
- "**Cerrar sesión**": Se cierra la sesión y se regresa al login (ver Figura 46a).
- "**Borrar cuenta**": abrirá un *PopUp* donde pedirá para confirmar escribir la palabra BORRAR y pulsar "**Eliminar**".
- "**Cerrar panel**": Cierra el submenú de ver cuenta.

Al seleccionar "**Borrar cuenta**" se muestra un *PopUp* de borrar cuenta (ver Figura 50b).



(a) Gestión de la cuenta de usuario.

(b) *PopUp* de borrar cuenta.

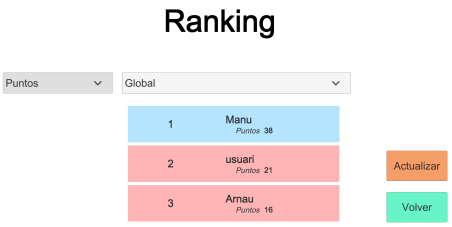
Figura 50: Gestión de cuenta.

A.2.3. Consultar Ranking

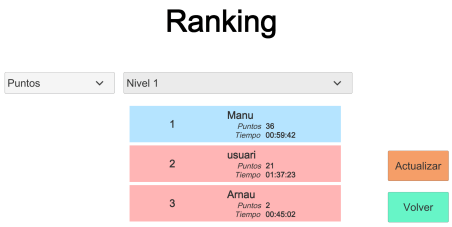
La pantalla de ranking (ver Figura 51a), por defecto muestra el ranking global de partidas. La tarjeta del usuario es azul y las tarjetas de los demás usuarios son de color rojo. El ranking global es el único que no se puede ordenar por tiempo, ya que los distintos niveles no tienen nada que ver entre ellos para contabilizarlos por igual.

Desde el botón "**Actualizar**" se puede refrescar el ranking y desde "**Volver**" se puede regresar al menú principal.

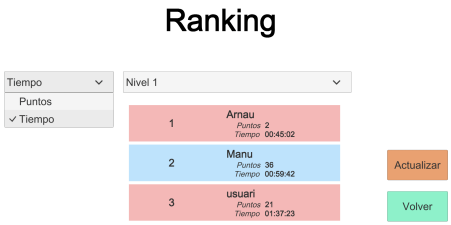
También podemos ver un nivel por ejemplo el nivel 1 ordenado por puntos (ver Figura 51b). Y desde el dropdown podemos cambiar el nivel o por ejemplo la ordenación del ranking por tiempo (ver Figura 51c).



(a) Ranking global.



(b) Ranking del Nivel 1 ordenado por puntos.



(c) Ranking del Nivel 1 ordenado por tiempo.

Figura 51: Rankings.

A.2.4. Elección de habilidades especiales

El jugador podrá tener hasta un máximo de 3 habilidades (ver Figura 52) especiales escogidas. Siempre y cuando tenga las habilidades desbloqueadas (sin candado rojo). Para añadir una habilidad basta con pulsar encima de ella y se activará un check verde.

Al poner el cursor encima de una habilidad aparecerá un *tooltip* con una descripción de la habilidad y de como usarla. Además la pantalla contiene los siguientes botones:

- "**Volver**": Se regresará al menú sin guardar nada.
- "**Resetear**": Desmarcará todas las habilidades.
- "**Guardar**": Mantendrá las habilidades escogidas para las siguientes partidas.

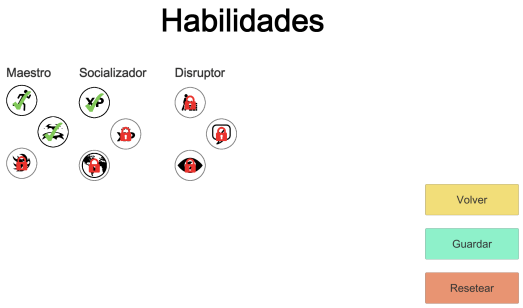


Figura 52: Gestión de habilidades especiales.

A.2.5. Selección de nivel

En la pantalla de selección de nivel y mundo aparecerán las tarjetas de los niveles que al poner el cursor encima de estas aparecerá un *tooltip* con una descripción del nivel. Si se pulsa la **flecha azul** (izquierda o derecha) se alterna de mundo (ver Figuras 53a y 53b).

Al pulsar "**Volver**" se puede regresar al menú principal. Si se pulsa "**Nueva partida**" aparecerá un *PopUp* con un botón de confirmación, si se confirma se borraran todos los datos de partida y la cuenta se quedará como recién creada.

(a) Selección de nivel: Mundo 1.

(b) Selección de nivel: Mundo 2.

Figura 53: Selector de mundo y nivel.

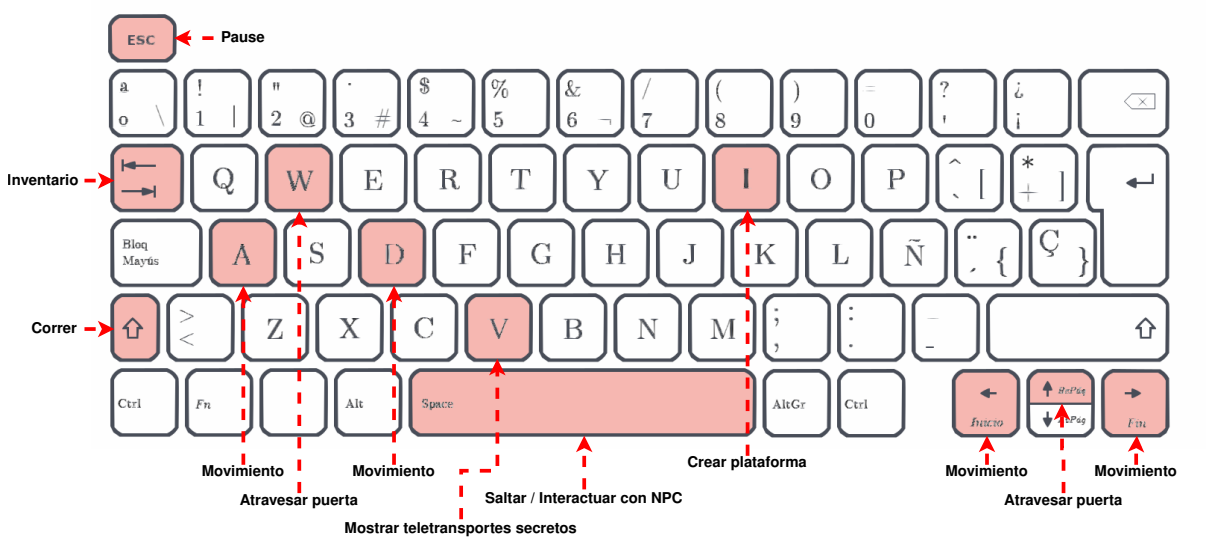
A.2.6. Controles de la partida

El mapa de controles para manejar al personaje del jugador en la partida sigue el estándar de la mayoría de juegos y es el siguiente:

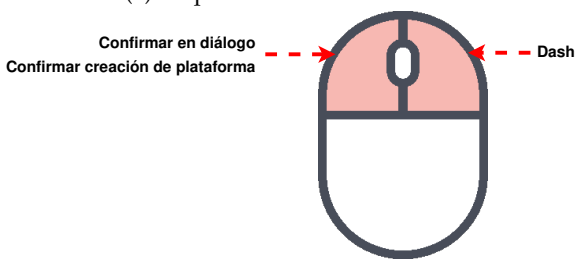
- Las teclas de **movimiento** son: **A, D Flecha izquierda, Flecha derecha**.
- La tecla de **saltar/interactuar con NPC es**: **espacio**.
- La tecla de **pause** es: **escape**.
- La tecla para **mostrar el inventario** es: **tabulador**.
- Las teclas para **entrar por una puerta** es: **W, Flecha arriba**.

Los controles para las habilidades especiales son:

- Tecla de **Correr**: **Tecla shift izquierda**.
- Tecla para hacer **Dash/deslizamiento**: **Botón derecho del ratón**.
- Tecla de **Crear plataforma**: **Tecla I**[14], **Botón izquierdo del ratón** para confirmar.
- Tecla para **Mostrar zonas secretas**: **tecla V**.



(a) Mapa de controles - teclado.



(b) Mapa de controles - ratón.

Figura 54: Controles de la partida.

[14]Si se vuelve a pulsar la misma tecla la plataforma se cancelará.

A.2.7. Partida

Una vez empieza la partida en cualquier momento podemos consultar el inventario y la lista de amigos si mantenemos pulsada la tecla tabulador (ver Figura 55a).

Si se desea pausar la partida se a de presionar la tecla escape, se mostrará un *PopUp* con dos columnas, la columna izquierda contiene botones de navegación y la derecha muestra las misiones activas y completadas así como un *tooltip* informativo de estas si se pasa el cursor por encima del texto (ver Figura 55b).

Respecto a los botones de la parte izquierda:

- "**Ver habilidades**": Si se pulsa el botón ver habilidades se muestra en la columna derecha las habilidades (solo a nivel informativo), si se pasa el cursor por encima de los iconos se mostrará información de cada habilidad y como usarla (ver Figura 55c).
- "**Continuar**": Continuará el transcurso de la partida.
- "**Reiniciar**": Volverá a empezar el mismo nivel desde el principio.
- "**Salir**": Volverá al menú principal.

(a) Inventario del jugador.

(b) Pause con misiones.

(c) Pause con habilidades.

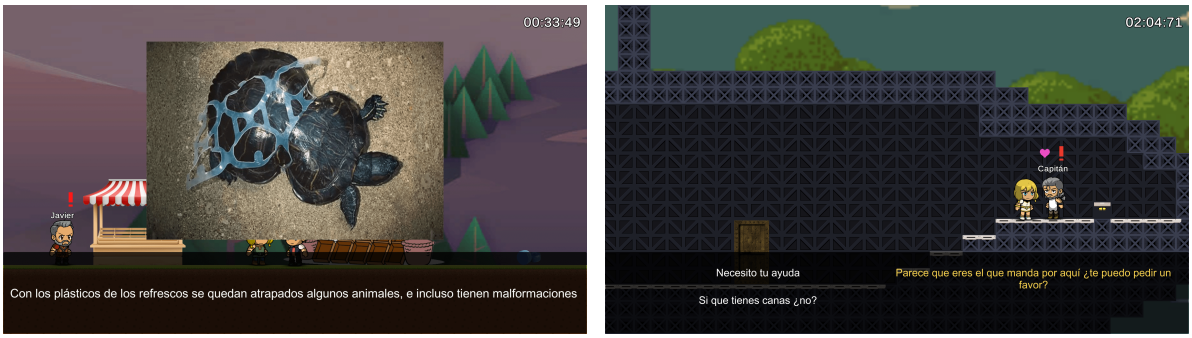
Figura 55: Menús de la partida.

Para que se inicien los diálogos se tienen que cumplir una de las dos situaciones:

- Colisionar con un objeto que se puede recoger y que éste tenga diálogos.
- Interactuar con un NPC utilizando la tecla espacio.

Los diálogos pueden mostrar opcionalmente imágenes (ver Figura 56a). Para pasar los diálogos hay que hacer click con el botón izquierdo del ratón sobre la caja de diálogo gris.

Si el diálogo contiene respuestas para el jugador (ver Figura 56b), se ha de seleccionar una de las posibles respuestas haciendo click con el botón izquierdo del ratón.



(a) Diálogo con imagen.

(b) Diálogo con respuestas.

Figura 56: Diálogos en la partida.

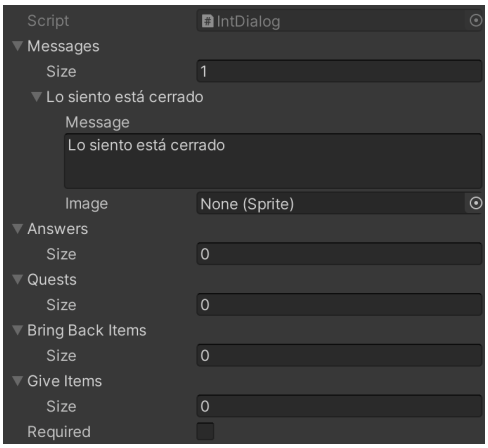
A.3. Manual de desarrollador

La disponibilidad de los mundos y niveles para un usuario está determinada por una lista de enteros, dónde ela posición equivale al mundo y el valor equivale al nivel. Por ejemplo, Juan es un usuario que tiene la disponibilidad de niveles-mundo (**worldLevel** en la base de datos) de su usuario de la siguiente manera: [1,1,3]. Esto se traduce en:

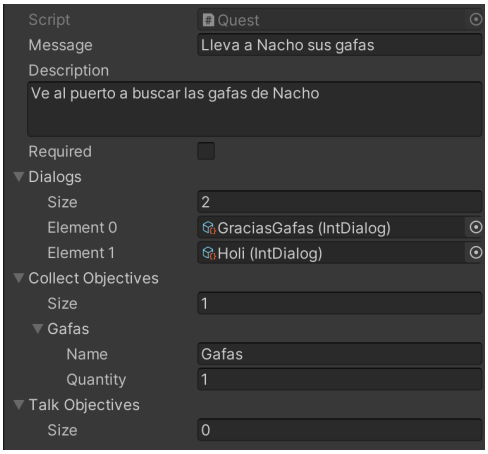
- Del primer mundo el usuario ha superado el segundo nivel.
- Del segundo mundo el usuario ha superado el segundo nivel.
- Del tercer mundo el usuario ha superado el cuarto nivel.

A.3.1. Crear diálogos y misiones

Al igual que para crear una carpeta nueva en el proyecto con click derecho del ratón, se pueden crear los ScriptableObjects de diálogos (ver Figura 57a) y misiones (ver Figura 57b). Una vez desde ahí se pueden rellenar desde el inspector de Unity. Las misiones se asignan a diálogos y los diálogos se asignan a NPCs o a otros diálogos como respuestas.



(a) Ejemplo de diálogo.



(b) Ejemplo de misión.

Figura 57: Ejemplos de diálogo y misión.

A.3.2. Añadir una escena de nivel nueva

Una vez creada una escena nueva y añadida a la sección de "Build" de Unity. Se han de instanciar los Prefabs básicos del nivel que son:

- **GameController**: Controlador de la partida que se encuentra en *Assets/Prefabs*. Es importante indicar el mundo y nivel en el que se encuentra, 0 equivale a 1 (1 a 2, 2 a 3, etc) tanto para nivel como para mundo. LevelName indica el nombre por el que se conocerá el nivel al iniciar. NextLevel indicara al controlador cual es el nivel siguiente al actual (se ha de poner el nombre de la escena), si no se rellena al acabar el nivel no se mostrará botón para jugar al siguiente nivel (ver Figura 58).

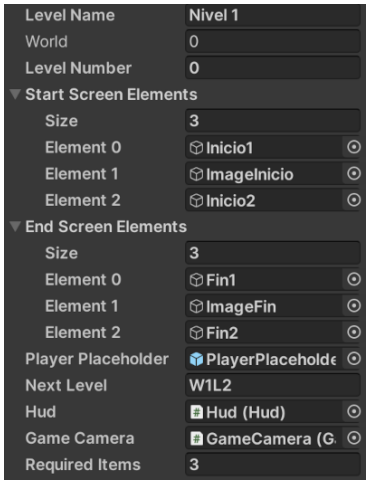


Figura 58: Atributos de GameController para un nuevo nivel.

- **GameCamera**: Cámara que serguirá al jugador *Assets/Prefabs/Camera*.
- **Hud**: Visualización cabeza-arriba de la información del playe en la partida que se encuentra en textitAssets/Prefabs/Ui/Game.
- **PlayerPlaceholder**: prefab que simboliza el player y donde se instanciará en runtime que se encuentra en *Assets/Prefabs*.

La escena debe contener "Empty GameObjects" con los siguientes nombres: PlayerNode, NPCs, Collectables, Doors, Teleports, Walls, Decorations, Backgrounds. En cada GameObject se instanciarán los Prefabs de cada elemento que se quiera asignar. También se ha de crear un GameObject llamado Grid con el componente **Grid**, dentro se crearan tantos objetos GameObject tilemaps como zonas se deseen, a los tilemaps que hagan de escenario y suelo se les ha de asignar el script GridTilemap que sirve para limitar el campo de visión del jugador. El componente **Tilemap Renderer** tendrá un order in layer de 150 si es suelo, 50 si es pared y 105 o valores en el rango de los anteriores si son decoraciones de tiles. A continuación se muestra una imagen de ejemplo de como queda un nivel (ver Figura 59).

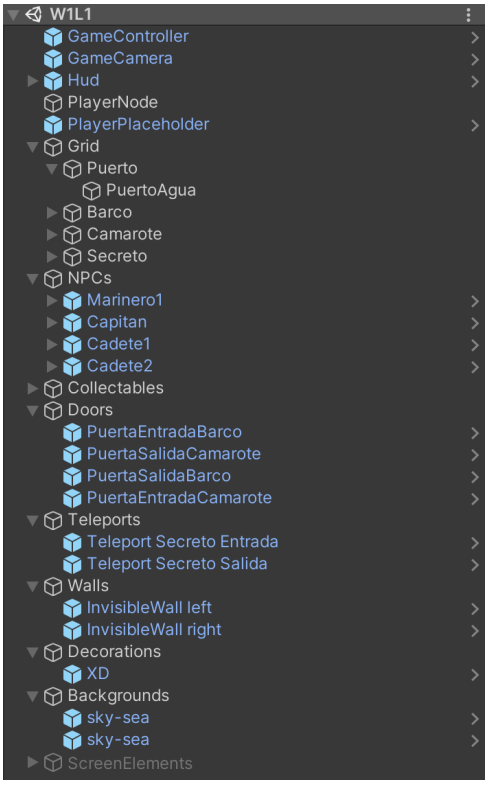


Figura 59: Elementos de una escena.

A.3.3. Añadir mundos al selector de niveles

El Prefab "World" (ver Figura 60) contiene los siguientes elementos:

- **WorldTitle**: Es donde se escribe el nombre del mundo que le será mostrado al jugador.
- **Content**: Es donde se almacenan los niveles que se mostrarán en el selector. El resto de GameObjects son para hacer funcionar el Scroll en la pantalla del mundo.

Figura 60: Prefab de mundo.

Para añadir un mundo nuevo:

1. En primer lugar se ha de abrir la escena "LevelSelector" de *Assets/Scenes/Ui* y desplegar el "Canvas" como en la Figura 61.

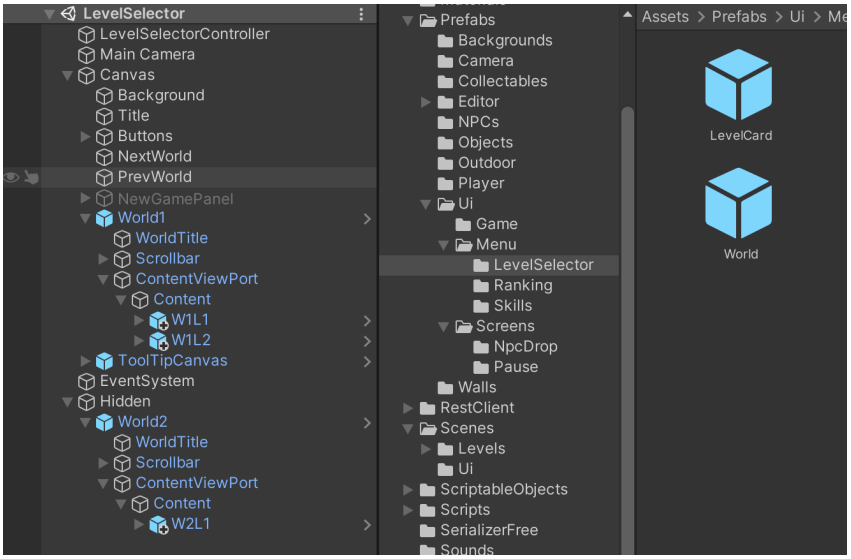


Figura 61: LevelSelector y el Canvas desplegado.

2. Una vez abierta la escena se ha de instanciar el Prefab "World" de *Assets/Prefabs/Ui/Menu/LevelSelector* en el GameObject del "Canvas" de la escena con un nombre con el formato **WorldX**, dónde **X** es el número del mundo, con valor 1 como mínimo. Es importante instanciar lo en el Canvas siempre por primera vez ya que sino se pueden corromper los valores de *position* y *scale* del "Transform". Como ejemplo, se puede observar la Figura 62 que instancia un World como "World4".

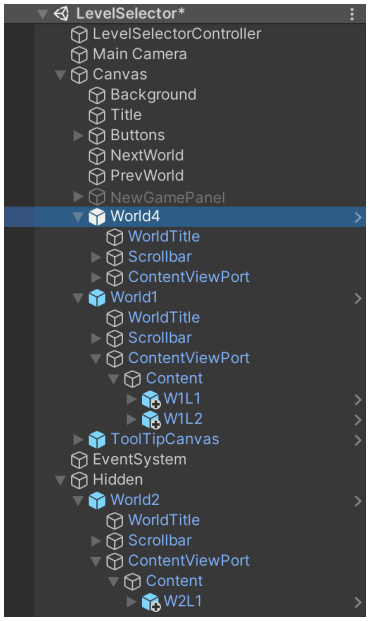


Figura 62: Prefab World instanciado y renombrado a World4.

3. Desde el GameObject "WorldTitle", hijo de World4 se edita el componente **Text** (ver Figura 63) para asignarle nombre del mundo que será mostrado al jugador (ver Figura 64).

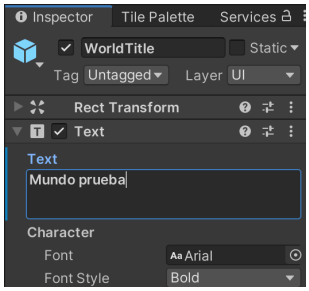


Figura 63: Label del nombre del mundo.

Figura 64: Pantalla de selección del nuevo nivel.

4. A continuación como el nuevo mundo no será el primero habrá que moverlo al GameObject "Hidden" (ver Figura 65), si se quisiera como principal se dejaría en el Canvas y se movería el que ya había.

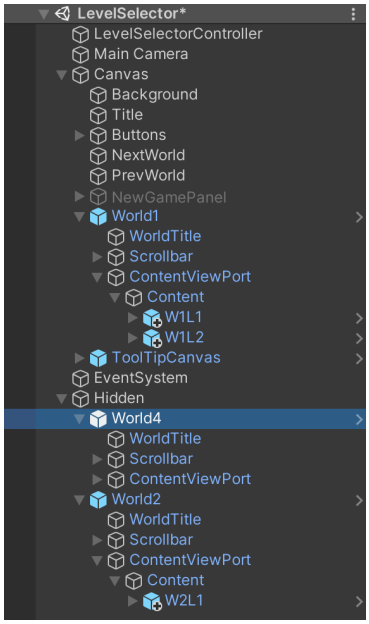


Figura 65: Mundo nuevo trasladado a Hidden.

Una vez hecho esto, habría un mundo nuevo y ya solamente faltaría guardar la escena actual para que no se pierdan los cambios.

A.3.4. Añadir niveles al selector de niveles

El Prefab "World" (ver Figura 66) contiene los siguientes elementos:

- **Thumb**: Imagen representativa del nivel.
- **LevelText**: Nombre del nivel que se mostrará en la tarjeta de nivel en la pantalla de selector de nivel.
- **Lock**: Candado para un nivel bloqueado.

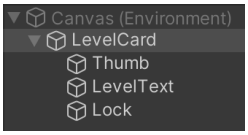


Figura 66: Prefab de nivel para Selector de niveles.

Para añadir un nivel nuevo a un mundo:

1. En primer lugar se arrastra el Prefab "LevelCard" de *Assets/Prefabs/Ui/Menu/LevelSelector* dentro del GameObject "Content" (hijo del GameObject del mundo) tal como aparece en la Figura 67.

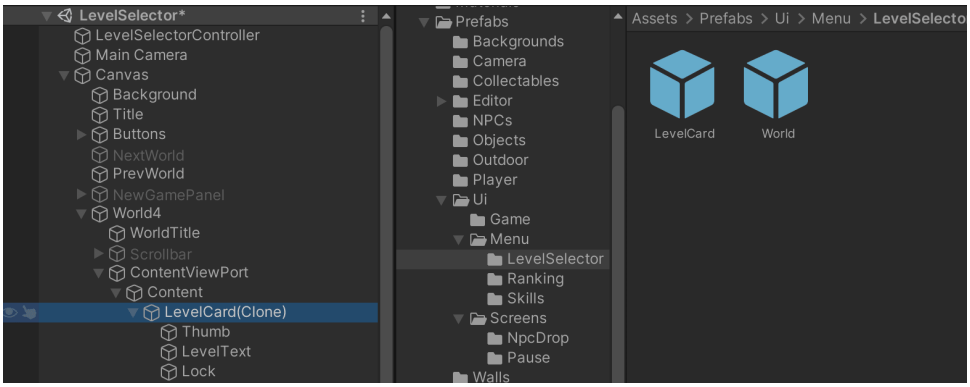


Figura 67: Instancia de un nuevo nivel en un mundo.

2. Una vez añadido, se ha de renombrar siguiendo el formato **WXLY**, dónde **X** es el número de mundo e **Y** es el número de nivel de ese mundo. Ambos empezando como mínimo en 1. En el siguiente ejemplo (ver Figura 68) se ha utilizado el World4 creado en la anterior sección y se ha llamado W4L1 al GameObject del nivel.

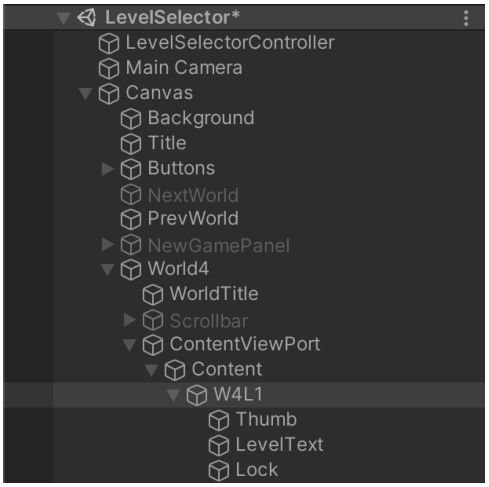


Figura 68: Nuevo nivel con identificador asignado.

3. Desde el inspector se editan las variables del script asociado (LevelCard): LevelScene, DescLevel, Image (ver Figura 69).

   - **LevelScene**: Nombre de la escena que cargará al hacer click en la tarjeta el jugador con el ratón.
   - **DescLevel**: descripción del nivel que se muestra en el tooltipo al pasar el cursor por encima de la tarjeta.
   - **Image**: La imagén que se muestra en la tarjeta del nivel.

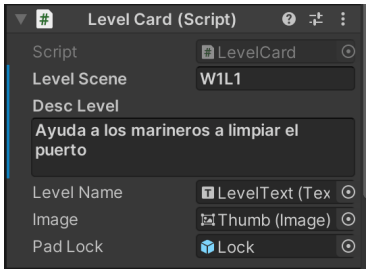


Figura 69: Script de una tarjeta de nivel.

4. Por último se edita el **Text** del GameObject "LevelText" para asignarle el nombre que se le mostrará al jugador en la pantalla.

Una vez hecho esto, habría un nivel nuevo añadido a un mundo existente en la pantalla de selección de nivel y ya solamente faltaría guardar la escena actual para que no se pierdan los cambios.

B. Descripciones textuales de los casos de uso

B.1. Interfaz y Menús

Nombre:	**UC1** - Autenticación / Conectarse
Actor:	Jugador Invitado
Descripción:	Un jugador se conecta al sistema para poder jugar
Precondiciones:	El jugador debe estar conectado a la red y estar en la pantalla de login
Poscondiciones:	Después de una conexión exitosa se accede al menú principal del juego
Flujo principal:	1. El usuario o jugador introduce el email 3. El usuario o jugador introduce la contraseña 4. El usuario o jugador pulsa en Entrar 5. El sistema valida los datos introducidos
Flujo alternativo:	6a. El usuario o jugador introduce el nombre de usuario (si es la primera vez que se conecta) y elije su género 7a. El sistema valida el nombre de usuario 7b. El sistema muestra un mensaje de error de el nombre de usuario esta en uso 5c. El sistema muestra un mensaje de error de el email es inválido 5d. El sistema muestra un mensaje de error de la contraseña es inválida 5e. El sistema muestra un mensaje de error de la cuenta no esta activada 6e. El sistema envía un correo de activación

Tabla 5: UC1 - Autenticación.

Nombre:	**UC2** - Recuperación de contraseña
Actor:	Jugador Invitado
Descripción:	Un jugador restablece su contraseña
Precondiciones:	El jugador debe estar conectado a la red y estar en la pantalla de login
Poscondiciones:	Si el email es válido se recibe un email con link para establecer una contraseña nueva
Flujo principal:	1. El usuario o jugador introduce el email 2. El usuario o jugador pulsa Recuperar contraseña 3. El sistema valida el email 4. El sistema muestra un mensaje de email enviado
Flujo alternativo:	3a. El sistema muestra un mensaje de error de el email es inválido

Tabla 6: UC2 - Recuperación de contraseña.

Nombre:	**UC3** - Registro
Actor:	Jugador Invitado
Descripción:	Un jugador crea una cuenta de jugador
Precondiciones:	El jugador debe estar conectado a la red y estar en la pantalla de login
Poscondiciones:	Si el registro es exitoso se envia email de confirmación
Flujo principal:	1. El usuario o jugador pulsa Registro 2. El usuario o jugador introduce el email 3. El usuario o jugador introduce la confirmación del email 4. El usuario o jugador introduce la contraseña 5. El usuario o jugador introduce la confirmación de la contraseña 6. El usuario o jugador pulsa Crear cuenta 7. El sistema valida los datos introducidos
Flujo alternativo:	7a. Los emails introducidos no coinciden (El sistema muestra un mensaje de error) 7b. El sistema muestra un mensaje de error de l email no puede estar vacío 7c. El sistema muestra un mensaje de error de el email no es válido 7d. El sistema muestra un mensaje de error de las contraseñas introducidas no coinciden 7e. El sistema muestra un mensaje de error de la contraseña no puede estar vacía 7f. El sistema muestra un mensaje de error de la cuenta de usuario ya existe 7g. El usuario o jugador pulsa Volver y regresa a inicio de sesión

Tabla 7: UC3 - Registro.

Nombre:	**UC4** - Desconexión
Actor:	Jugador Autenticado
Descripción:	Un jugador se desconecta del sistema
Precondiciones:	El jugador debe estar autenticado y en el menú principal
Poscondiciones:	Se regresa a la pantalla principal de inicio de sesión
Flujo principal:	1. El usuario o jugador pulsa Ver cuenta 2. El usuario o jugador pulsa Cerrar sesión 3. El sistema elimina la configuración de inicio
Flujo alternativo:	-

Tabla 8: UC4 - Desconexión.

Nombre:	**UC5** - Borrado de cuenta
Actor:	Jugador Autenticado
Descripción:	El jugador eliminará su cuenta
Precondiciones:	El jugador debe estar autenticado y debe estar en el menú principal
Poscondiciones:	Se regresa a la pantalla principal de inicio de sesión si se ha borrado
Flujo principal:	1. El usuario o jugador pulsa Ver cuenta 2. El usuario o jugador pulsa Borrar cuenta 3. Aparece un Pupup donde el usuario o jugador introduce BORRAR 4. El usuario o jugador pulsa Eliminar 5. El sistema elimina la cuenta del jugador
Flujo alternativo:	3a. El usuario o jugador pulsa Cancelar

Tabla 9: UC5 - Borrado de cuenta.

Nombre:	**UC6** - Cambiar contraseña
Actor:	Jugador Autenticado
Descripción:	Un jugador cambia su contraseña
Precondiciones:	El jugador debe estar conectado a la red y estar en el menú principal
Poscondiciones:	Se recibe un email con un link para cambiar la contraseña
Flujo principal:	1. El usuario o jugador pulsa Ver cuenta 2. El usuario o jugador pulsa Cambiar contraseña
Flujo alternativo:	-

Tabla 10: UC6 - Cambiar contraseña.

Nombre:	**UC7** - Cambio de habilidades
Actor:	Jugador Autenticado
Descripción:	El jugador cambiará la habilidades extras
Precondiciones:	El jugador debe estar autenticado y debe estar en hel menú de habilidades
Poscondiciones:	Los cambios quedan guardados
Flujo principal:	1. El usuario o jugador pulsa sobre la habilidad deseada (hay un límite de 3) 2. El usuario o jugador pulsa Guardar
Flujo alternativo:	1a. El usuario o jugador pulsa Volver 1b. El usuario o jugador pulsa Resetear (se desmarcan todas las habilidades)

Tabla 11: UC7 - Cambio de habilidades.

Nombre:	**UC8** - Selección de nivel
Actor:	Jugador Autenticado
Descripción:	El jugador elige el nivel que jugará
Precondiciones:	El jugador debe estar autenticado y en el menú jugar
Poscondiciones:	Se muestra una pantalla con una introducción (opcionalmente) sobre un tema y después se puede empezar a jugar
Flujo principal:	1. El sistema comprueba la disponibilidad de niveles 2. El sistema muestra los niveles 3. El usuario o jugador escoge el nivel deseado disponible
Flujo alternativo:	3a. El usuario o jugador pulsa la flecha derecha/izquierda y se avanza/-retrocede de mundo 3b. El usuario o jugador pulsa Volver

Tabla 12: UC8 - Selección de nivel.

Nombre:	**UC9** - Inicio de nueva partida
Actor:	Jugador Autenticado
Descripción:	El jugador empezará a jugar desde el principio
Precondiciones:	El jugador debe estar autenticado y en el menú jugar
Poscondiciones:	Se muestra una pantalla con una introducción (opcionalmente) sobre un tema y después se puede empezar a jugar
Flujo principal:	1. Se abre un *PopUp* para confirmar 2. El usuario o jugador pulsa Confirmar 3. El sistema crea unos datos nuevos
Flujo alternativo:	2a. El usuario o jugador pulsa Cancelar

Tabla 13: UC9 - Inicio de nueva partida.

Nombre:	**UC10** - Ver ranking
Actor:	Jugador Autenticado
Descripción:	El jugador consultará la clasificación por niveles del juego
Precondiciones:	El jugador debe estar autenticado y en el menú principal
Poscondiciones:	Se muestra una pantalla con una lista ordenada por puntuación de los usuarios que mejor han superado los niveles en su conjunto
Flujo principal:	1. El usuario o jugador pulsa ranking desde el menú principal 2. El sistema hace una consulta de los usuarios, sus puntuaciones y reconstruye el *leaderboard* global ordenado por puntuación 3. El usuario o jugador pulsa Volver
Flujo alternativo:	3a. El usuario o jugador pulsa Actualizar 4a. El sistema hace una consulta de los usuarios, sus puntuaciones y reconstruye el *leaderboard* global ordenado por puntuación 3b. El usuario o jugador pulsa elige desde el dropdown de ordenar o desde el dropdown de selección de nivel un valor distinto 4b. El sistema hace una consulta de los usuarios, sus puntuaciones y reconstruye el *leaderboard* global ordenado por puntuación

Tabla 14: UC10 - Consulta del ranking.

B.2. Partida

Nombre:	**UC11** - Movimiento
Actor:	Jugador Autenticado
Descripción:	El jugador desplazará a su personaje
Precondiciones:	El jugador debe estar autenticado y debe estar en un nivel
Poscondiciones:	El personaje se desplazará por la pantalla en el eje de las x
Flujo principal:	1. El usuario o jugador utiliza las teclas A, D , flecha izquierda, flecha derecha para mover su personaje 2. El sistema actualiza su posición
Flujo alternativo:	-

Tabla 15: UC11 - Movimiento.

Nombre:	**UC12** - Salto
Actor:	Jugador Autenticado
Descripción:	El jugador hará que su personaje salte
Precondiciones:	El jugador debe estar autenticado y debe estar en un nivel
Poscondiciones:	El personaje aplicará una fuerza en el eje de las Y solo si esta en el suelo o si tiene la habilidad del doble salto
Flujo principal:	1. El usuario o jugador pulsa la tecla espacio 2. El sistema comprueba si esta en el suelo o si tiene más de un salto disponible
Flujo alternativo:	2a. No se aplica la fuerza ya que el personaje no esta tocando el suelo y no le quedan saltos

Tabla 16: UC12 - Salto.

Nombre:	**UC13** - Habilidades especiales (Activas)
Actor:	Jugador Autenticado
Descripción:	El jugador hará que su personaje ejecute una acción especial
Precondiciones:	El jugador debe estar autenticado, debe estar en un nivel y debe tener la habilidad escogida
Poscondiciones:	El personaje ejecutará la habilidad
Flujo principal:	1. El usuario o jugador pulsa la tecla correspondiente a la habilidad especial
Flujo alternativo:	-

Tabla 17: UC13 - Habilidades especiales (Activas).

Nombre:	**UC14** - Acción con NPC
Actor:	Jugador Autenticado
Descripción:	El jugador hará que su personaje interactúe con un NPC
Precondiciones:	El jugador debe estar autenticado y debe estar en un nivel cerca del objeto interactivo
Poscondiciones:	El personaje recibirá información en cajas de texto a veces también con imágenes
Flujo principal:	1. El usuario o jugador pulsa el botón de saltar cuando esta al lado del Npc 2. Se muestra una caja de texto con o sin imagen 3. El usuario o jugador debe elegir una opción para responder 4. Si no hay más conversación se cierra el dialogo
Flujo alternativo:	-

Tabla 18: UC14 - Acción con NPC.

Nombre:	**UC15** - Recolección
Actor:	Jugador Autenticado
Descripción:	El jugador recogerá objetos coleccionables
Precondiciones:	El jugador debe estar autenticado y debe estar en un nivel además de que el objeto debe poderse coger libremente o por misión
Poscondiciones:	Los objetos se añadirán al jugador
Flujo principal:	1. El usuario o jugador colisiona con el objeto 2. El objeto desaparece del escenario
Flujo alternativo:	2a. Aparece un dialogo con o sin fotos interactuable 3a. El objeto desaparece del escenario

Tabla 19: UC15 - Recolección.

Nombre:	**UC16** - *PopUp* de Pausa
Actor:	Jugador Autenticado
Descripción:	El jugador interrumpirá temporalmente el juego
Precondiciones:	El jugador debe estar autenticado y debe estar en un nivel
Poscondiciones:	Todo lo referente al nivel quedará congelado hasta que se le de a continuar
Flujo principal:	1. El usuario o jugador pulsa la tecla Escape 2. Aparece un *PopUp* con diversas opciones: Continuar, Ver habilidades, Reiniciar, Salir. Aparece también un recuadro de objetivos (los completados aparecen tachados) 3. El usuario o jugador pulsa Continuar, se desvanece el *PopUp* y sigue el juego
Flujo alternativo:	3a. El usuario o jugador pulsa Ver habilidades, aparece el árbol de habilidades del personaje en solo lectura 3b. El usuario o jugador pulsa Reiniciar, y el nivel empieza de nuevo 3c. El usuario o jugador pulsa Salir y volverá al menú principal

Tabla 20: UC16 - *PopUp* de Pausa.

Nombre:	**UC17** - *PopUp* de inventario y amigos
Actor:	Jugador Autenticado
Descripción:	Se mostraran dos ventanas de información
Precondiciones:	El jugador debe estar autenticado y debe estar en un nivel
Poscondiciones:	Mientras esté pulsada la tecla se verá la información
Flujo principal:	1. El usuario o jugador mantiene presionada la tecla tabulador 2. Aparece un *PopUp* con información del inventario y de los amigos del jugador
Flujo alternativo:	3. El usuario o jugador suelta el tabulador 4. Desaparece la ventana informativa

Tabla 21: UC17 - *PopUp* de inventario y amigos.

C. Endpoints

En este apéndice se mostrarán los endpoints tanto de la gestión de cuentas/autenticación como los de la base de datos.

C.1. Base de datos

PUT `https://[projectId]-default-rtdb.europe-west1.firebasedatabase.app/scores/[levelName]/[userId].json?auth=[idToken]` — *Añadir / Modificar score*

Body

Atributo	Tipo
time	*float*
points	*int*
completed	*int*

Respuesta

Atributo	Tipo
time	*float*
points	*int*
completed	*int*

Objeto parseado

Score

Figura 70: Añadir o modificar un score.

Body & Response

```
{
  "time" : 30.4,
  "points" : 100,
  "completed" : 2
}
```

PUT	https://[projectId]-default-rtdb.europe-west1.firebasedatabase.app /users/[userId].json?auth=[idToken]	*Añadir / Modificar usuario*

Body

Atributo	Tipo
uname	*String*
socializer	*int*
master	*int*
disruptor	*int*
philantropist	*int*
worldLevel	*Lista<int>*
friends	*Lista<String>*
selectedSkills	*Lista<String>*
playerModel	*int*

Respuesta

Atributo	Tipo
uname	*String*
socializer	*int*
master	*int*
disruptor	*int*
philantropist	*int*
worldLevel	*Lista<int>*
friends	*Lista<String>*
selectedSkills	*Lista<String>*
playerModel	*int*

Objeto parseado

User

Figura 71: Añadir o modificar datos de un usuario.

Body & Response

```
{
  "uname" : "Alejandro",
  "socializer" : 20,
  "master" : 100,
  "disruptor" : 0,
  "philantropist" : 15,
  "worldLevel" : 100,
  "friends" : ["Aldeano Javier", "Marinero"],
  "selectedSkills" : ["Run", "Dash"],
  "playerModel" : 0
}
```

GET	https://[projectId]-default-rtdb.europe-west1.firebasedatabase.app /users/[userId].json?auth=[idToken]	*Obtener usuario*

Body

Atributo	Tipo
-	-

Respuesta

Atributo	Tipo
uname	*String*
socializer	*int*
master	*int*
disruptor	*int*
philantropist	*int*
worldLevel	*Lista<int>*
friends	*Lista<String>*
selectedSkills	*Lista<String>*
playerModel	*int*

Objeto parseado

User

Figura 72: Obtener datos de un usuario.

Response

```
{
  "uname" : "Alejandro",
  "socializer" : 20,
  "master" : 100,
  "disruptor" : 0,
  "philantropist" : 15,
  "worldLevel" : 100,
  "friends" : ["Aldeano Javier", "Marinero"],
  "selectedSkills" : ["Run", "DoubleJump"],
  "playerModel" : 0
}
```

Figura 73: Obtener score para un nivel de un usuario.

Response

```
{
  "time" : 30.4,
  "points" : 100,
  "completed" : 2
}
```

GET	https://[projectId]-default-rtdb.europe-west1.firebasedatabase.app /users.json?auth=[idToken]	*Obtener usuarios*

Body

Atributo	Tipo
-	-

Respuesta

Atributo	Tipo
objeto	*JSON<String>*

Objeto parseado

ResponseHelper – Diccionario<String, User>

Figura 74: Obtener todos los datos de todos los usuarios.

Response

```
{
  "IDUSUARIO" : {
    "uname" : "Alejandro",
    "socializer" : 20,
    "master" : 100,
    "disruptor" : 0,
    "philantropist" : 15,
    "worldLevel" : 100,
    "friends" : ["Aldeano Javier", "Marinero"],
    "selectedSkills" : ["Run", "DoubleJump"],
    "playerModel" : 0
  },
  "IDUSUARIO2" : {
    "uname" : "Judit",
    "socializer" : 60,
    "master" : 20,
    "disruptor" : 0,
    "philantropist" : 15,
    "worldLevel" : 100,
    "friends" : ["Marinero"],
    "selectedSkills" : ["Run"],
    "playerModel" : 1
  }
}
```

GET	https://[projectId]-default-rtdb.europe-west1.firebasedatabase.app /scores.json?auth=[idToken]	*Obtener scores*

Body

Atributo	Tipo
-	-

Respuesta

Atributo	Tipo
objeto	*JSON<String>*

Objeto parseado

ResponseHelper – Diccionario<String, Diccionario<String, Score >

Figura 75: Obtener todos los scores de todos los usuarios y niveles.

Response

```
{
  "Nivel 1" : {
    "IDUSUARIO" : {
      "time" : 30.4,
      "points" : 100,
      "completed" : 2
    },
    "IDUSUARIO2" : {
      "time" : 40.4,
      "points" : 80,
      "completed" : 1
    }
  },
  "Nivel 2": {
    "IDUSUARIO" : {
      "time" : 20.4,
      "points" : 120,
      "completed" : 1
    }
  }
}
```

GET	https://[projectId]-default-rtdb.europe-west1.firebasedatabase.app /scores/[levelName].json?auth=[idToken]	*Obtener scores de los usuarios para un nivel*

Body

Atributo	Tipo
-	-

Respuesta

Atributo	Tipo
objeto	*JSON<String>*

Objeto parseado

ResponseHelper – Diccionario<String, Score>

Figura 76: Obtener todos los scores de usuarios para un nivel.

Response

```
{
  "IDUSUARIO" : {
    "time" : 30.4,
    "points" : 100,
    "completed" : 2
  },
  "IDUSUARIO2" : {
    "time" : 40.4,
    "points" : 80,
    "completed" : 1
  }
}
```

DELETE	https://[projectId]-default-rtdb.europe-west1.firebasedatabase.app /users/[userId].json?auth=[idToken]	*Borrar usuario*

Body

Atributo	Tipo
-	-

Respuesta

Atributo	Tipo
uname	*String*
socializer	*int*
master	*int*
disruptor	*int*
philantropist	*int*
worldLevel	*Lista<int>*
friends	*Lista<String>*
selectedSkills	*Lista<String>*
playerModel	*int*

Objeto parseado

User

Figura 77: Borrar los datos de un usuario.

Response

```
{
  "uname" : "Alejandro",
  "socializer" : 20,
  "master" : 100,
  "disruptor" : 0,
  "philantropist" : 15,
  "worldLevel" : 100,
  "friends" : ["Aldeano Javier", "Marinero"],
  "selectedSkills" : ["Run", "DoubleJump"],
  "playerModel" : 0
}
```

DELETE	https://[projectId]-default-rtdb.europe-west1.firebasedatabase.app /scores/[levelName]/[userId].json?auth=[idToken]	*Borrar score*

Body

Atributo	Tipo
-	-

Respuesta

Atributo	Tipo
time	*float*
points	*int*
completed	*int*

Objeto parseado

Score

Figura 78: Borrar los scores para un nivel de un usuario.

Response

```
{
  "time" : 30.4,
  "points" : 100,
  "completed" : 2
}
```

C.2. Autenticación

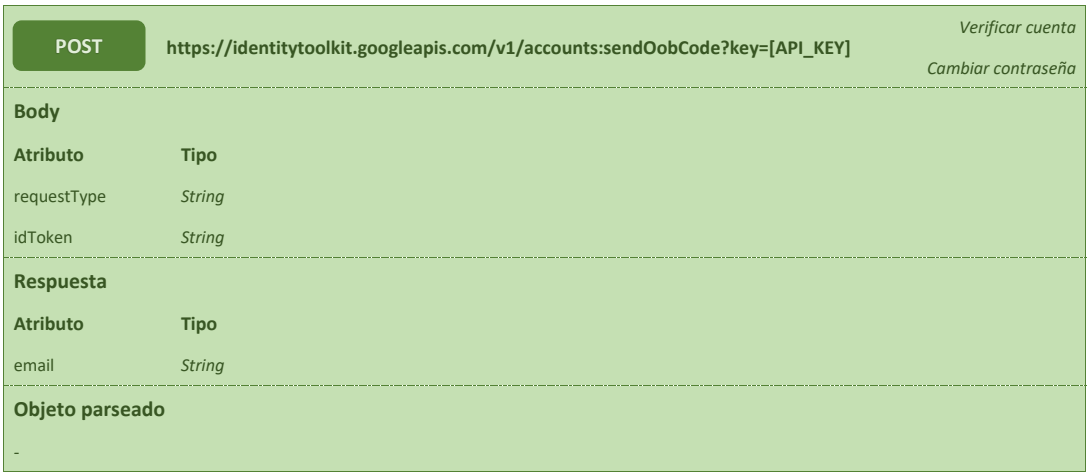


Figura 79: Verificar cuenta o cambiar contraseña.

Body

```
{
  "requestType" : "PASSWORD_RESET",
  "email" : "[user@example.com]"
}
```

Response

```
{
  "email" : "[user@example.com]"
}
```

POST	https://securetoken.googleapis.com/v1/token?key=[API_KEY]	*Refrescar Token*

Body

Atributo	Tipo
grant_type	*String*
refresh_token	*String*

Respuesta

Atributo	Tipo
expires_in	*String*
token_type	*String*
refresh_token	*String*
id_token	*String*
user_id	*String*
project_id	*String*

Objeto parseado

AuthRefreshTokenData

Figura 80: Refrescar token de sesión.

Body

```
{
  "token" : "[CUSTOM_TOKEN]",
  "returnSecureToken" : true
}
```

Response

```
{
  "idToken": "[ID_TOKEN]",
  "refreshToken": "[REFRESH_TOKEN]",
  "expiresIn": "3600"
}
```

POST	https://identitytoolkit.googleapis.com/v1/accounts:signUp?key=[API_KEY]	*Crear Cuenta*

Body

Atributo	Tipo
returnSecureToken	*bool*
email	*String*
password	*String*

Respuesta

Atributo	Tipo
idToken	*String*
email	*String*
refreshToken	*String*
expiresIn	*String*
localID	*String*

Objeto parseado

AuthSignUpData

Figura 81: Crear una cuenta.

Body

```
{
  "email" : "[user@example.com]",
  "password" : "[PASSWORD]",
  "returnSecureToken" : true
}
```

Response

```
{
  "idToken" : "[ID_TOKEN]",
  "email" : "[user@example.com]",
  "refreshToken" : "[REFRESH_TOKEN]",
  "expiresIn" : "3600",
  "localId" : "tRcfmLH7..."
}
```

POST	https://identitytoolkit.googleapis.com/v1/accounts:signInWithPassword?key=[API_KEY]	*Iniciar sesión*

Body

Atributo	Tipo
returnSecureToken	*bool*
email	*String*
password	*String*

Respuesta

Atributo	Tipo
expires_in	*String*
idToken	*String*
email	*String*
refreshToken	*String*
expiresIn	*String*
localId	*String*
registered	*bool*

Objeto parseado

AuthSignInData

Figura 82: Iniciar sesión.

Body

```
{
  "email" : "[user@example.com]",
  "password" : "[PASSWORD]",
  "returnSecureToken" : true
}
```

Response

```
{
  "localId" : "ZY1rJK0eYLg...",
  "email" : "[user@example.com]",
  "displayName" : "",
  "idToken" : "[ID_TOKEN]",
  "registered" : true,
  "refreshToken" : "[REFRESH_TOKEN]",
  "expiresIn" : "3600"
}
```

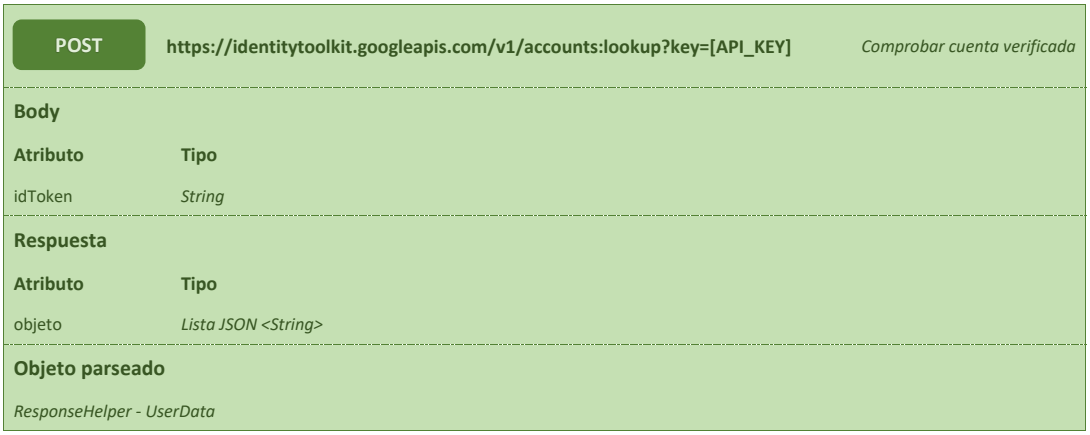


Figura 83: Comprobar si la cuenta está verificada.

Body

```
{
  "idToken":"[FIREBASE_ID_TOKEN]"
}
```

Response

```
{
  "users" : [
    {
      "localId" : "ZY1rJK0...",
      "email" : "user@example.com",
      "emailVerified" : false,
      "displayName" : "John Doe",
      "providerUserInfo" : [
        {
          "providerId" : "password",
          "displayName" : "John Doe",
          "photoUrl" : "http://localhost:8080/img1234567890/photo.png",
          "federatedId" : "user@example.com",
          "email" : "user@example.com",
          "rawId" : "user@example.com",
          "screenName" : "user@example.com"
        }
      ],
      "photoUrl" : "https://lh5.googleusercontent.com/.../photo.jpg",
      "passwordHash" : "...",
      "passwordUpdatedAt" : 1.484124177E12,
      "validSince" : "1484124177",
      "disabled" : false,
      "lastLoginAt" : "1484628946000",
      "createdAt" : "1484124142000",
      "customAuth" : false
    }
```

```
28    ]
29 }
```

POST	https://identitytoolkit.googleapis.com/v1/accounts:delete?key=[API_KEY]	*Borrar cuenta*
Body		
Atributo	**Tipo**	
idToken	*String*	
Respuesta		
Atributo	**Tipo**	
-	-	
Objeto parseado		
ResponseHelper		

Figura 84: Borrar una cuenta.

Body

```
1 {
2   "idToken" : "[FIREBASE_ID_TOKEN]"
3 }
```

D. Diagramas UML

Tal como se muestra (ver Figura 85) podemos ver el diagrama de clases UML con la relación de la cámara y los limites. Ya que queremos que la cámara se quede fija cuando lleguemos a ciertas coordenadas xy de cada sección, cada Grid tendrá unas coordenadas limites.

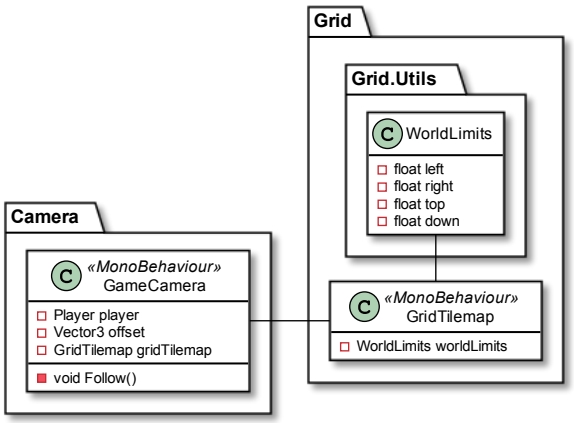


Figura 85: Diagrama UML - Bloque de la cámara y los limites de pantalla.

Las siguientes imágenes (ver Figuras 86, 87) muestran los diagramas de clases de objetos de la interfaz de usuario (Canvas) que actúan dentro de la partida.

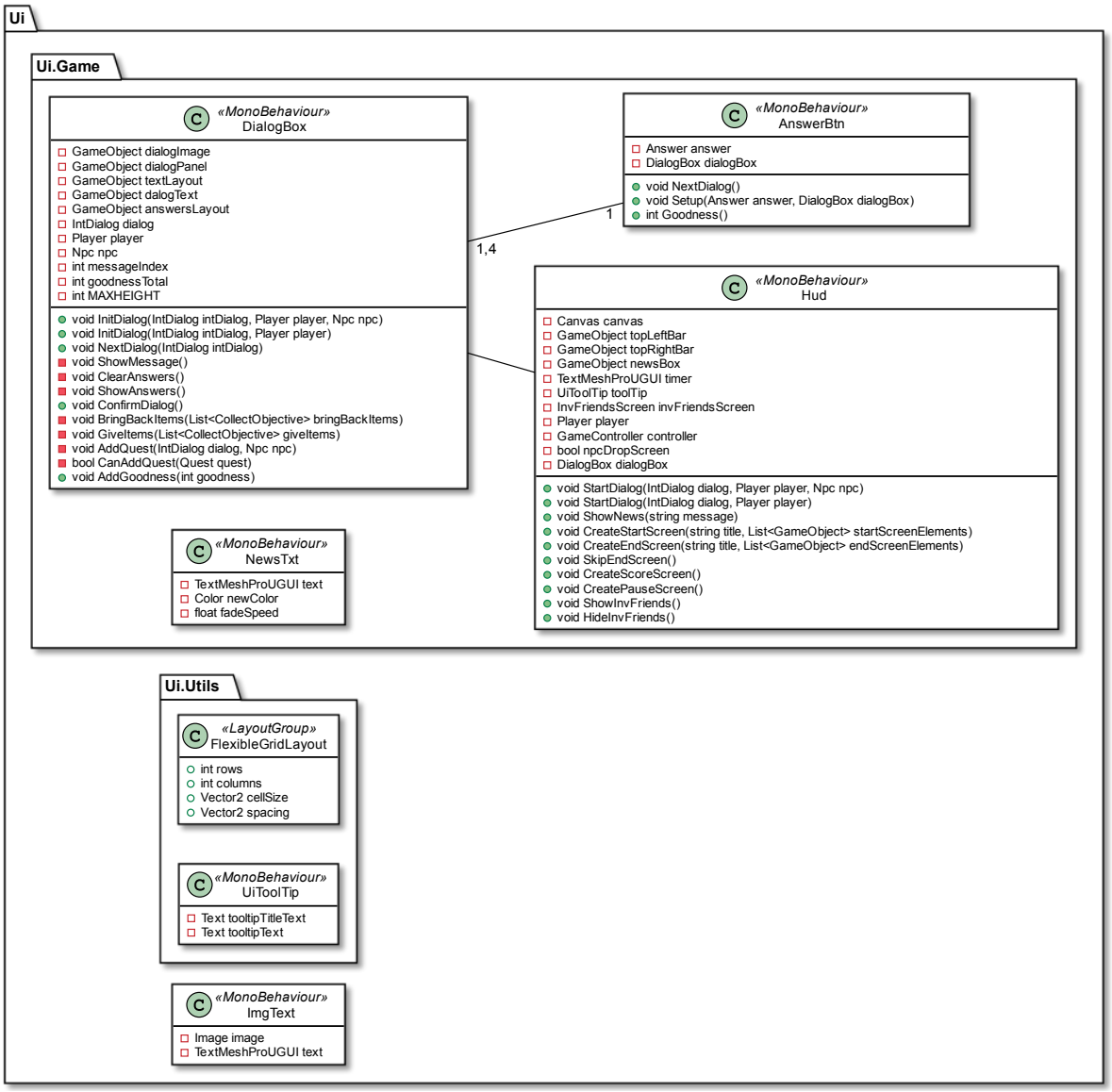


Figura 86: Diagrama UML - Bloque de la UI en la Partida y Utils.

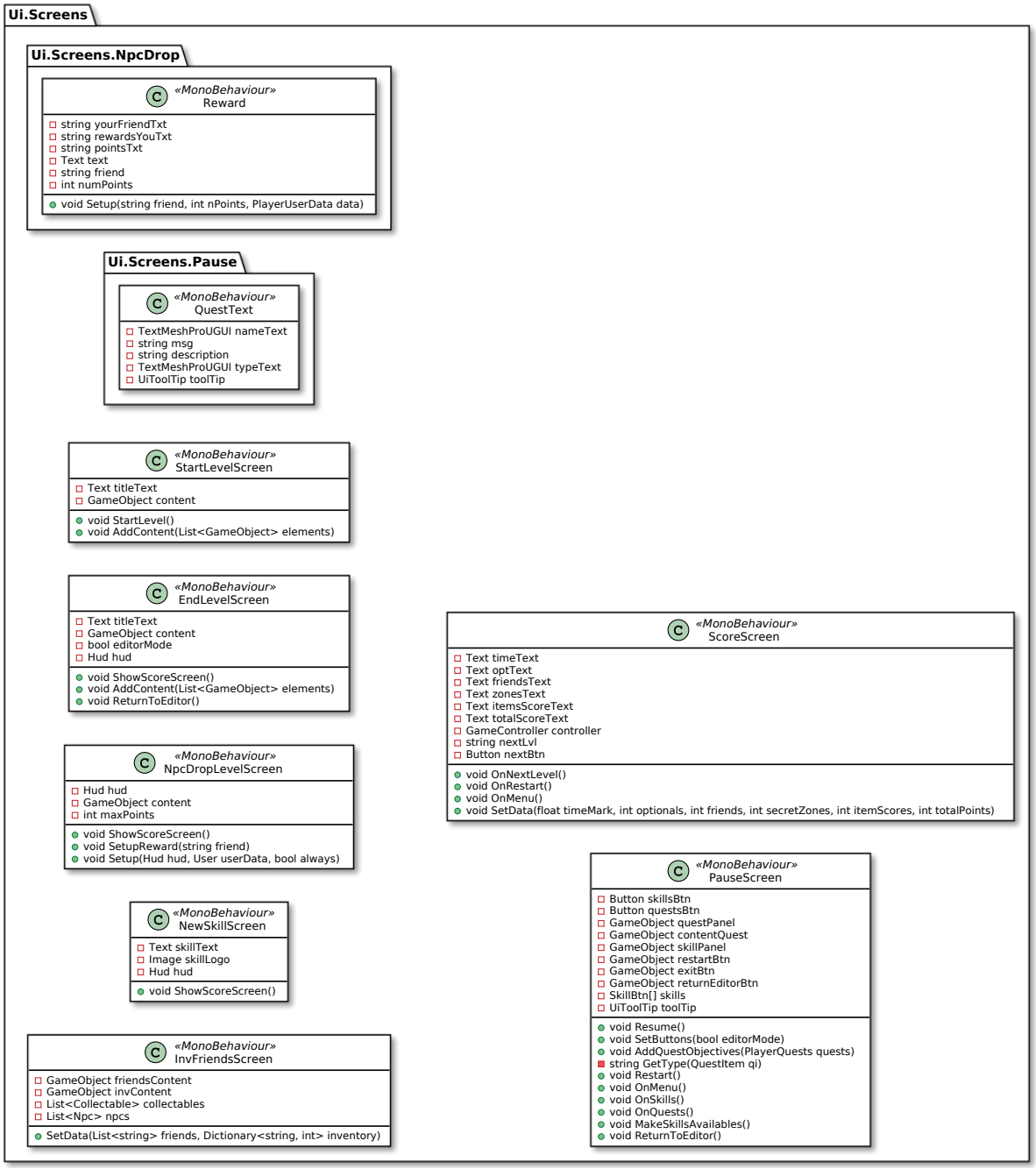


Figura 87: Diagrama UML - Bloque de la UI en Screens.

A continuación (ver Figura 88) se puede ver el diagrama de clases UML del jugador.

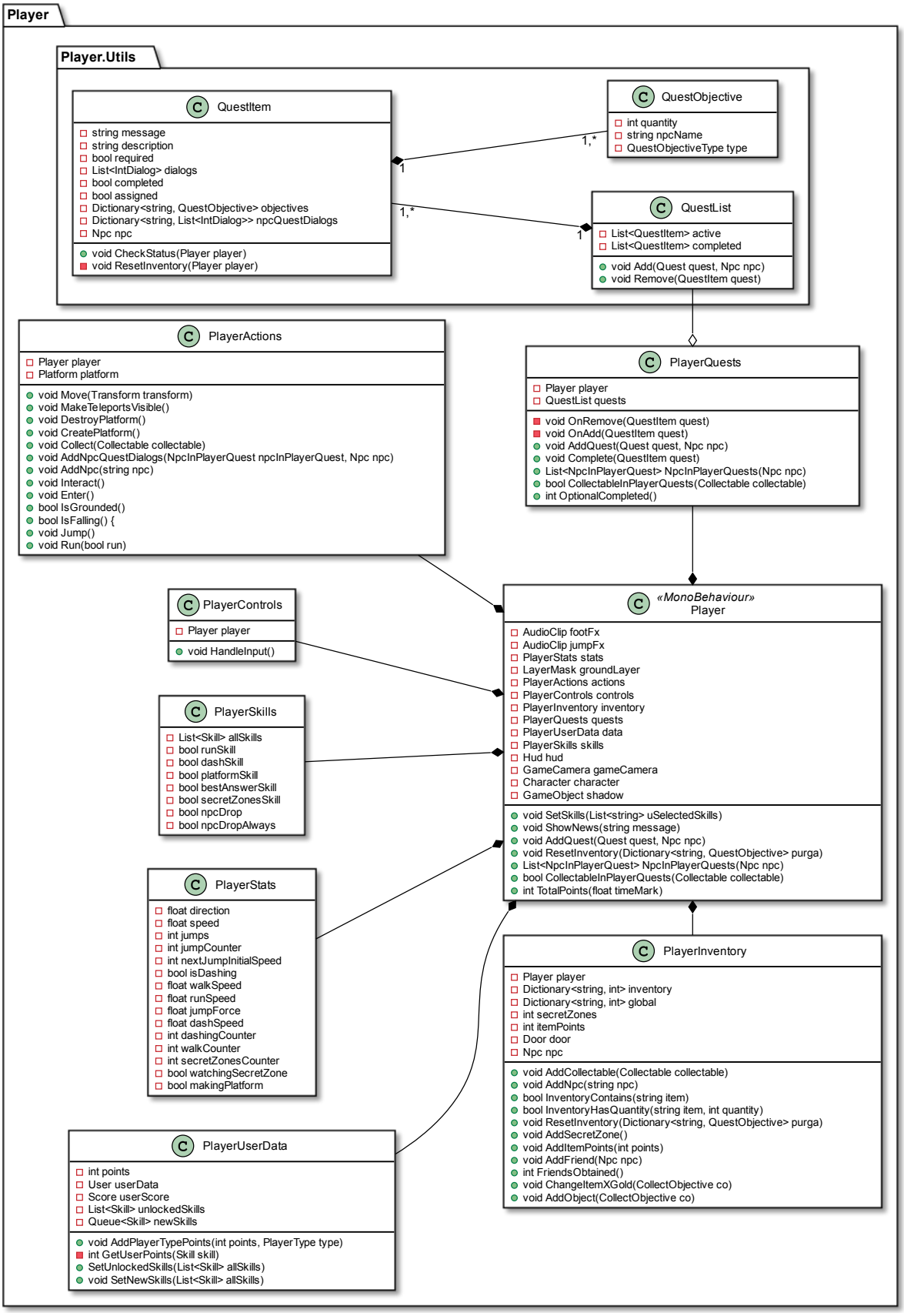


Figura 88: Diagrama UML - Bloque del Jugador.

El jugador tiene una relación de 1:1 con el Hud y GameCamera, además de que en diversas clases hay relación con NPC como el PlayerInventory (queremos saber el Npc que esta en contacto con el jugador para interactuar) y lo mismo pasa con Door.

QuestItem tiene relación 1:1 con el Npc también.

PlayerSkills y PlayerUserData tienen relacion con Skills, además que PlayerUserData se relaciona con los modelos de base de datos User y Score.

La idea de separar al jugador en diversas clases nos ayuda a tener el código más entendible y más fácil de modificar o escalar [2].

La siguiente imagen (ver Figura 89) representa el bloque de las clases que se usan en el menú.

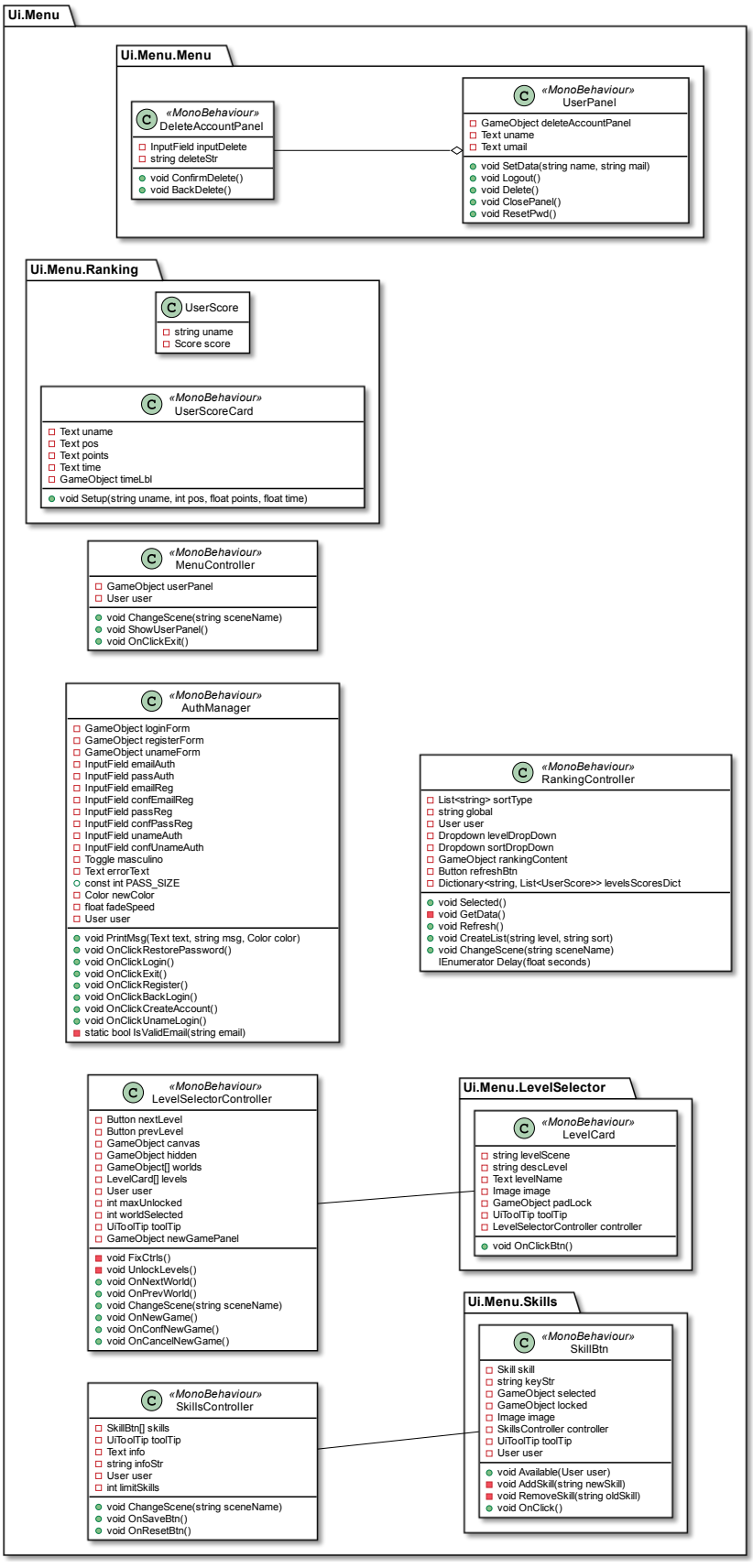


Figura 89: Diagrama UML - Bloque de los menús.

La siguiente imagen (ver Figura 90) representa el bloque de las clases que se usan de modelos para la base de datos y autenticación así como sus handlers.

DatabaseHandler, GameSession y AuthHandler son clases estáticas.

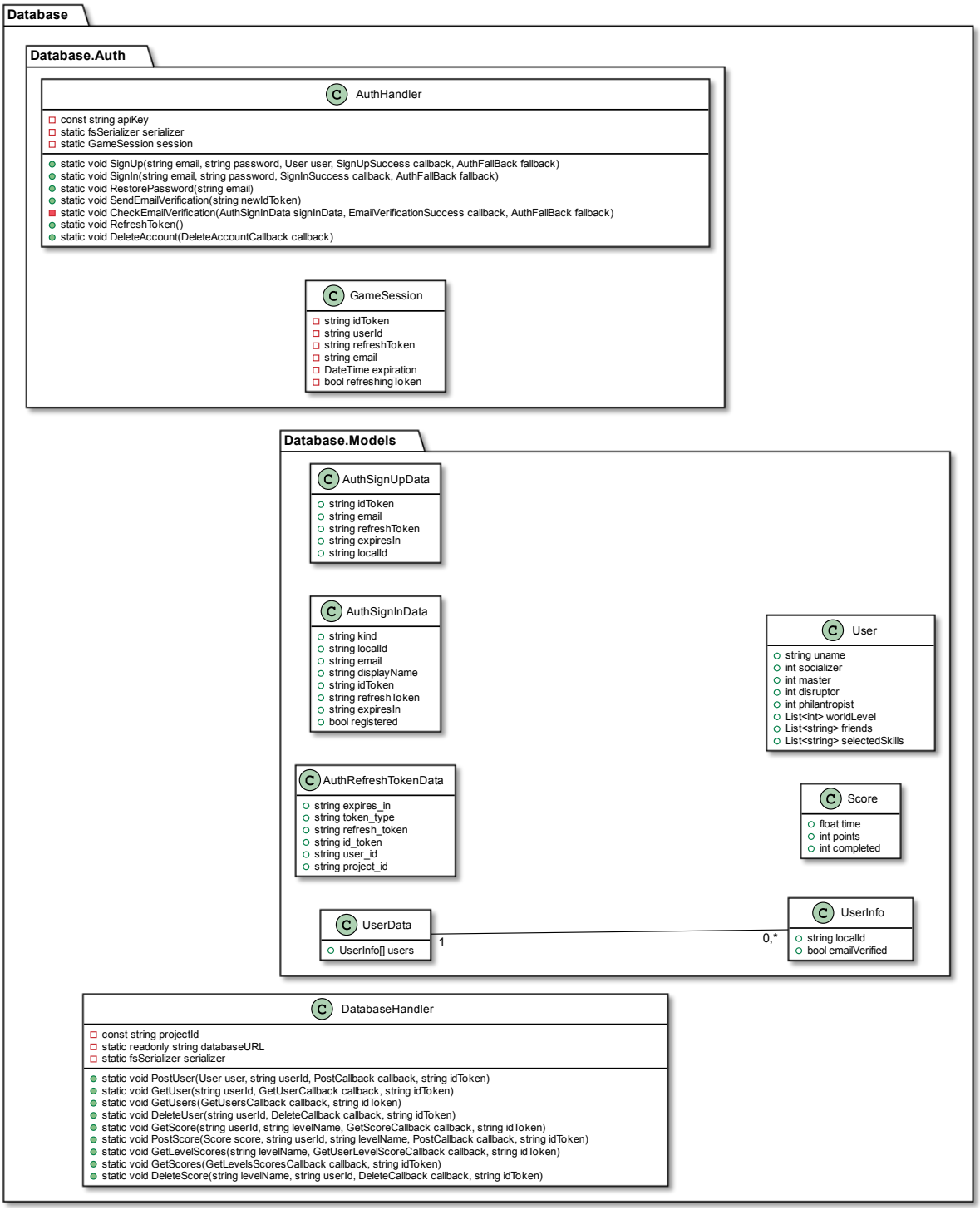


Figura 90: Diagrama UML - Bloque de los modelos para la persistencia de datos.

La siguiente imagen (ver Figura 91) representa las clases que usan ScriptableObject: Skill, Quest e IntDialog

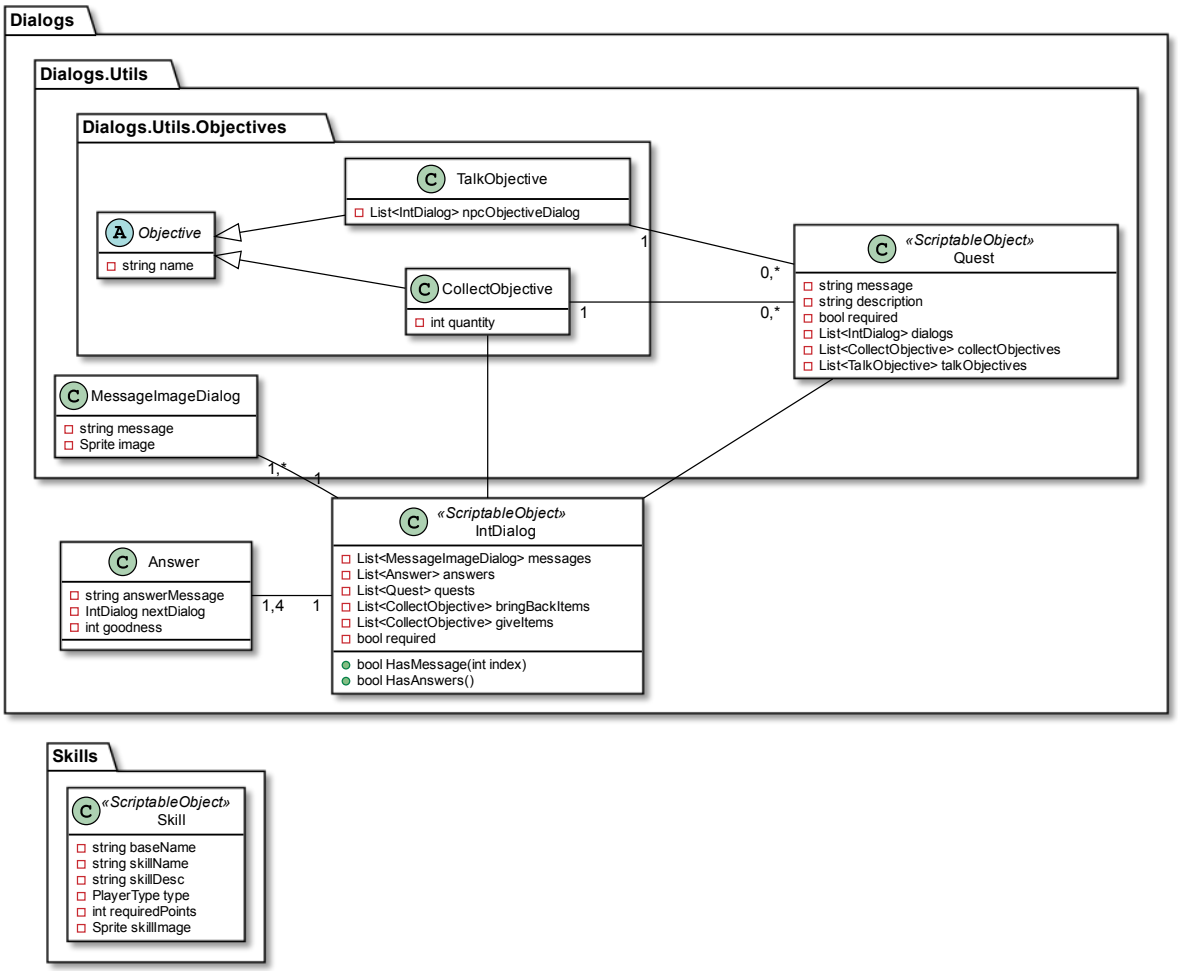


Figura 91: Diagrama UML - Bloque de las clases que usan ScriptableObject.

La siguiente imagen (ver Figura 92) representa el resto de clases que existen en una partida: Coleccionables, Npcs, Controller, Puertas...

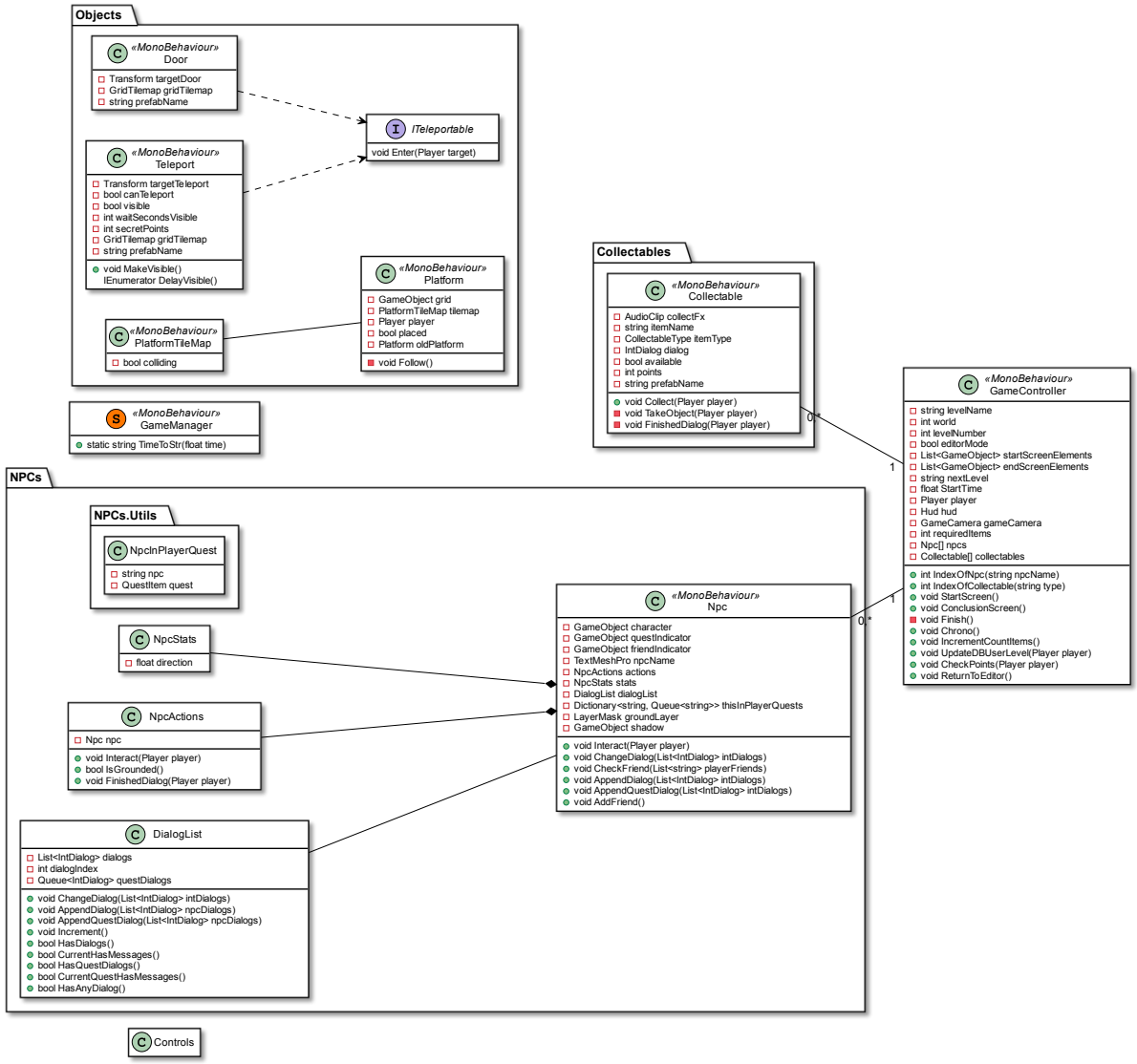


Figura 92: Diagrama UML - Bloque del resto de clases de la partida.

La clase Controls básicamente es estática y tiene atributos estáticos con el mapeado de controles y nombres de estos.