LA FUENTE DE CRISTAL

David Álvarez de Luna Quintana

Proyecto de desarrollo de aplicaciones multiplataforma

CIFP Virgen de Gracia – Curso 2021/2022

ÍNDICE

1. PRESENTACIÓN DEL PROYECTO

1.1. Explicación

Mi proyecto consiste en un juego con gráficos 2d creado en Unity con integración con base de datos.

La temática del juego se centrará en contar una pequeña historia (sobre un joven que acaba en un pueblo por determinadas circunstancias). A lo largo de la historia, se tomarán decisiones que afectarán al desarrollo del juego.

Mi proyecto incluye dos partes principales:

- Juego desarrollado en Unity.
- Base de datos en la que el juego almacenará información sobre el juego que podrá ser consultada. Se usará Firebase.

El flujo del juego es el siguiente:

- Antes de comenzar a jugar, el jugador debe registrarse usando un correo. El registro se hará desde el juego.
- El jugador comienza a jugar. A lo largo de la partida, tendrá que tomar decisiones. Estas decisiones serán almacenadas en variables.
- Cuando el jugador finaliza una escena del juego, se volcarán todas las variables a una base de datos. Esto permite que el jugador pueda guardar y cargar la partida en otro momento.
- El jugador puede consultar las decisiones que un jugador ha tomado a lo largo de la partida. Si el jugador aún no ha terminado el juego, no podrá realizar consultas (porque podría revelarle datos de la historia que no haya visto aún).

Más detalles en Anexo I: Game Design Document

1.2. Estudio de Mercado

Transición de gráficos 2D a 3D

El sector del Videojuego es muy amplio. Con el paso del tiempo, los videojuegos se han ido desarrollando desde los famosos marcianitos a juegos actuales que presentan gráficos cada vez más cercanos a la realidad, y que incluso aprovechan tecnologías como la realidad virtual para ofrecer una experiencia más inmersiva.

Muchos de las sagas de videojuegos más famosas empezaron utilizando una estética 2D, que con el paso del tiempo y el desarrollo de nuevas tecnologías han transicionado al 3D. Sin embargo, estos videojuegos con una estética más empleada en el pasado siguen siendo apreciados por muchos jugadores, de manera que se siguen produciendo videojuegos que aprovechan la tecnología actual pero mantienen la estética 2D para llevarla a su máxima expresión.

Videojuegos de Aventura

Los videojuegos de género aventura son los que ofrecen al jugador exploración, solución de rompecabezas e interacciones con personajes, todo esto enfocándose en contar una historia. Es un género muy variado, ya que sobre dicha base, pueden incluir elementos de acción, y aún ser considerados videojuegos de aventura (Un ejemplo es la saga The Legend of Zelda).

Los videojuegos de aventura presentan similitudes a los de género RPG. La diferencia es que mientras que el RPG se centra en los combates y estadísticas de los personajes, en el género aventura se enfocan más en la resolución de problemas.

Existen en el mercado numerosos videojuegos que dan al jugador la posibilidad de tomar decisiones que afectan al desarrollo de la historia. Uno de los beneficios que tiene este sistema es la rejugabilidad, ya que para ver todo el contenido del juego, será necesario hacer varias partidas.

Ejemplos en el sector

Shin Megami Tensei (1991)

En este videojuego JRPG, el jugador lucha por sobrevivir en un Japón apocalíptico dominado por demonios, que se dividen en dos bandos, ley y caos. Durante el desarrollo del juego, el jugador tomará decisiones que determinarán si sigue el camino de la ley, el caos, o la neutralidad.

Este videojuego inició una saga que siguen publicando productos a día de hoy.

The Oregon Trail (1971)

La premisa de este videojuego basado en comandos de texto, ambientado en 1848, es realizar un trayecto con tu familia desde Independence a Oregon, con una cantidad limitada de recursos que puedes comprar al principio del juego. En el transcurso del juego, se te presentan eventos, por ejemplo, que un familiar ha enfermado, y se te pregunta qué harás. Tus decisiones determinarán en qué estado llegas al final, y tendrás una puntuación que indica tú desempeño.

Life is Strange (2015)

En este juego te metes en la piel de una estudiante que ha vuelto a estudiar a su ciudad natal. A lo largo del juego interactúa con diversos personajes. Las decisiones tomadas a lo largo de la historia se registran al finalizar cada capítulo, algunas son menos influyentes, como elegir la ropa que te pones, y otras pueden tener consecuencias que varían entre que se marchite una planta, la relación con un compañero, o que muera un personaje.

Triangle Strategy (2022)

En este videojuego RPG de estrategia por turnos, con estética denominada HD-2D, tres países están en guerra por la escasez de sal y hierro. El jugador es el patriarca de una pequeña casa noble, y decidirá cómo actuar a lo largo de la guerra intentando convencer a cada uno de sus aliados de que le siga el camino que ha decidido tomar. Sus decisiones afectan al desarrollo de la historia.

Análisis DAFO:

El análisis DAFO define la situación de mi producto desde dos puntos de vista: Interno y externo. Se examinan las fortalezas y debilidades del producto de manera interna, y las oportunidades y amenazas teniendo en cuenta factores externos, como la competencia.

- Fortalezas: Estética 2D atractiva para muchos jugadores, sistema que permite compartir los resultados con otros jugadores.
- Debilidades: Duración corta.
- Oportunidades: Incremento de jugadores interesados en experiencias de juego que se alejan de la acción.

- Amenazas: La competencia es amplia y sus juegos cuentan con mayor presupuesto.

1.3. Valor del Producto

Mi intención con este proyecto es aplicar los conocimientos obtenidos en el ciclo formativo para hacer poder crear un pequeño mundo diseñado por mí, que otras personas puedan experimentar.

Mi producto ofrece al consumidor una alternativa que mezcla el encanto de los gráficos 2D con una historia ligera y un sistema de toma de decisiones que puedes compartir con otros jugadores.

La intención es que los jugadores a los que les guste este tipo de juegos pasen un rato entretenido.

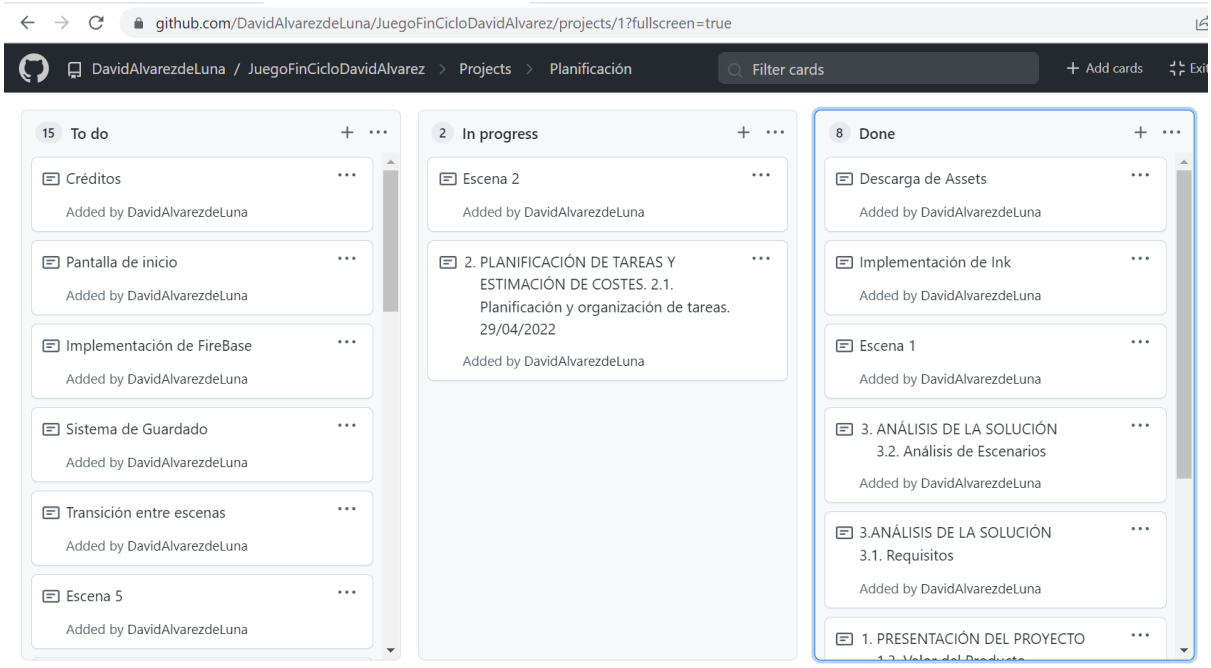
Si la calidad del juego finalizado es suficiente, el proyecto finalizado será distribuido al público mediante plataformas de compra de juegos, como Play Store.

2. PLANIFICACIÓN DE TAREAS Y ESTIMACIÓN DE COSTES

2.1. Planificación de tareas

2.1.1. Planificación

La planificación de tareas se hace desde Github, en su apartado de Projects. La siguiente imagen representa la planificación en una fase intermedia del proyecto, en formato kanban.



Los hitos llevados a cabo a lo largo de la elaboración de este documento son los siguientes:

- 03/04/2022: Explicación resumida del proyecto.
- 10/04/2022: Análisis de la solución (requisitos, mapas de usuario...)
- 15/04/2022: Estudio de mercado, valor del producto.
- 22/04/2022: Análisis de escenarios (casos de uso, mapas de interacción...)
- 06/05/2022: Estimación de costes y recursos: hardware, software y humanos. Herramientas usadas. Gestión de riesgos.
- 13/05/2022: Diseño de la interfaz de usuario y prototipos.
- 18/05/2022: Diagrama de clases. Diseño de la persistencia de la información (Relacional, NoSQL...).
- 23/05/2022: Arquitectura del sistema
- 29/05/2022: Análisis tecnológico.
- 12/06/2022: Elementos a implementar.
- 19/06/2022: Entrega de proyecto, documentación y repositorio.

2.1.2. Flujo de trabajo en Gitflow

El flujo de trabajo del proyecto cuenta con dos ramas: Main y Develop, ambas publicadas en remoto. Se trabajará sobre la rama Develop y el progreso se pasará a la rama Main cuando se consideré que se han realizado cambios estables en la aplicación.

Para gestionar la elevada cantidad de archivos que se generan en un proyecto de Unity, me apoyo de la aplicación GitHub Desktop.

2.1.3. Wiki del proyecto

Para incluir la documentación del proyecto, Github cuenta con un apartado Wiki, donde publicaré el documento con información referente al proyecto que estoy realizando actualmente, el cual se continuará actualizando en sucesivas entregas.

2.2. Estimación de costes y recursos

La estimación de costes variará durante el desarrollo del proyecto, debido a las necesidades que puedan surgir, como pueden ser la publicación del juego o la necesidad de comprar assets para mejorar la calidad del producto final.

2.2.1. Hardware

Las necesidades de Hardware en el proyecto son escasas, debido a que disponía del material utilizado para el desarrollo del proyecto y comprobación de su funcionamiento. Un ordenador y un dispositivo móvil. También se contará con una memoria externa para almacenar una copia del proyecto.

2.2.2. Software

- Unity: Programa de desarrollo de videojuegos. Gratuito
- Assets de Unity: Elementos utilizados en la elaboración del juego, como sprites o efectos de sonido. Se utilizan assets gratuitos, aunque no se descarta la posibilidad de recurrir a pagar por assets si son determinantes para el proyecto y no puedo emularlos con herramientas gratuitas.
- Visual studio Community: IDE para programación en distintos lenguajes, incluido C#. Gratuito.
- Github y Github Desktop: Plataforma de control de versiones en remoto (Github) o en local (Github Desktop). Gratuitos.
- FireBase: Plataforma ubicada en la nube que permite la sincronización de proyectos. Gratuito para mis necesidades actuales en el proyecto (almacenar datos de jugadores y variables generadas durante la partida).

- Software para edición de assets y elaboración de retratos de los personajes (IbisPaint X), gratuito.
- Microsoft Office: Paquete de aplicaciones de escritorio para la creación y edición de documentos. Licencia disponible.
- PlayStore: Plataforma de distribución de aplicaciones de Android. 25€ por aplicación publicada.

2.2.3. Humanos

El proyecto será desarrollado por una persona, que distribuirá su tiempo entre el desarrollo del juego (código, búsqueda y preparación de assets, pruebas), y la elaboración de documentación.

Tiempo dedicado al desarrollo del juego: 300 horas.

Tiempo dedicado a elaborar la documentación: 50 horas.

2.3. Herramientas usadas

El desarrollo del juego se realizará a través de la herramienta especializada Unity. El código del juego se escribe en Visual Studio Community. La preparación de algunos assets personalizados se realiza a través de IbisPaint X. La elaboración de documentación se realiza con Microsoft Office y Google Docs. Para el control de versiones se emplea Github.

2.4. Gestión de riesgos

Planificación:

- Problemas derivados del tiempo de entrega: La prioridad es finalizar una versión estable del proyecto, que demuestre sus funcionalidades y sea capaz de expresar la idea del proyecto. Una vez elaborada esa versión, se buscará mejorar la ambientación del juego y añadir contenido.
- Retrasos respecto a la planificación: Se priorizará el funcionamiento juego sobre la duración y detalles estéticos.

Organización y Gestión:

- Una sola persona a cargo del proyecto: Los esfuerzos se centran en poder hacer una partida completa del juego. Cuando se alcance ese objetivo, se priorizará mejorar el juego y el código.

Herramientas de desarrollo:

- Alguna herramienta no funciona como se esperaba: Se intentará solucionar el problema. Si no es posible, se buscará una alternativa o se intentará que el proyecto funcione sin esa herramienta (por ejemplo, si no es posible implementar un sistema de guardado, se exigirá que se supere el juego en una sesión de juego, y se reducirá la duración si se cree necesario).
- La curva de aprendizaje de las herramientas es elevada: Antes de implementar una herramienta no conocida, se estudiará la funcionalidad principal, para asegurarse de que puede realizar la función deseada de forma sencilla.

Producto:

- El código no está estructurado de forma óptima: Se prioriza que el juego realice las funcionalidades pretendidas. Debido a la naturaleza cambiante del juego y a posibles problemas, el código puede no estar estructurado de la mejor manera posible. Cuando se finalice el juego, se realizará una revisión exhaustiva del código con el objetivo de mejorarlo.
- Falta de calidad: Al priorizar la finalización de una partida, es posible que algunos puntos concretos no queden ambientados como se desearía. Si no suponen un problema a la hora de continuar el proyecto, se tratarán posteriormente (Por ejemplo, que un personaje no se mueva como se desea pero llegue igualmente a su destino).
- Problemas al intentar hacer más accesible el proyecto: En este caso, el proyecto está siendo probado en Windows (durante la elaboración del juego) y Android (la plataforma donde se va a lanzar el producto final). Extender el proyecto a otros sistemas, como Linux o iOS, podría presentar complicaciones que no forman parte del propósito inicial del proyecto.

3. ANÁLISIS DE LA SOLUCIÓN

3.1. Requisitos

3.1.1. Requisitos Funcionales

- RF1: Al iniciar el juego, el usuario se debe registrar o iniciar sesión.
- RF2: El jugador puede cerrar sesión.
- RF3: Pulsando el botón de salir, se cerrará la aplicación.
- RF4: Al iniciar el juego, el jugador tendrá la posibilidad de comenzar una partida, continuar la partida, ver los créditos, consultar los resultados y salir del juego.

- RF5: La opción de consultar resultados solo estará disponible para jugadores que hayan superado la partida al menos una vez, lo cual será debidamente comunicado al jugador.
- RF6: En la partida, el jugador tendrá controles para mover a su personaje (palanca direccional), hablar con los personajes no controlables del juego (botón A) y acceder a un menú de pausa (botón PAUSA).
- RF7: En algunas fases del juego el personaje tendrá algún control adicional. Por ejemplo, en un minijuego, disparará con una arma de juguete pulsando el botón B.
- RF8: El menú de pausa ofrecerá información sobre controles del juego y la posibilidad de salir de la partida.
- RF9: El juego se estructura en varias escenas secuenciadas y claramente diferenciadas.
- RF10: En cada escena, el jugador deberá hablar con los personajes no controlables que habitan el pueblo y realizar los minijuegos requeridos antes de poder pasar a la siguiente escena.
- RF11: Diversas acciones realizadas por el jugador serán registradas y tenidas en cuenta a lo largo del desarrollo del juego.
- RF12: La partida se guardará al finalizar cada escena.
- RF13: Al finalizar el juego, el jugador obtendrá acceso a la consulta de resultados.

3.1.2. Requisitos No Funcionales

- RNF1: El registro del jugador y la información recabada a lo largo del juego serán almacenados en Firebase.
- RNF2: El registro se hace con correo.
- RNF3: El juego realizará consultas a Firebase.
- RNF4: El juego será desarrollado en Unity.
- RNF5: El juego tendrá una estética 2D, para lo que requerirá assets. Los assets obtenidos de fuentes externas serán debidamente referenciados en los créditos.
- RNF6: A lo largo del juego, se almacenarán algunos datos en variables en el DecisionManager del juego.
- RNF7: Al finalizar el juego, las variables se almacenan en la base de datos, y estarán disponibles para ser consultadas.

3.1.3. Requisitos de Información

- RI1: Usuario:
  - Correo (Clave primaria)
  - Escena actual
- RI2: Datos de la partida: Registro de las decisiones tomadas a lo largo de la partida, que se podrán consultar desde la aplicación tras superar el juego.

3.2. Análisis de Escenarios

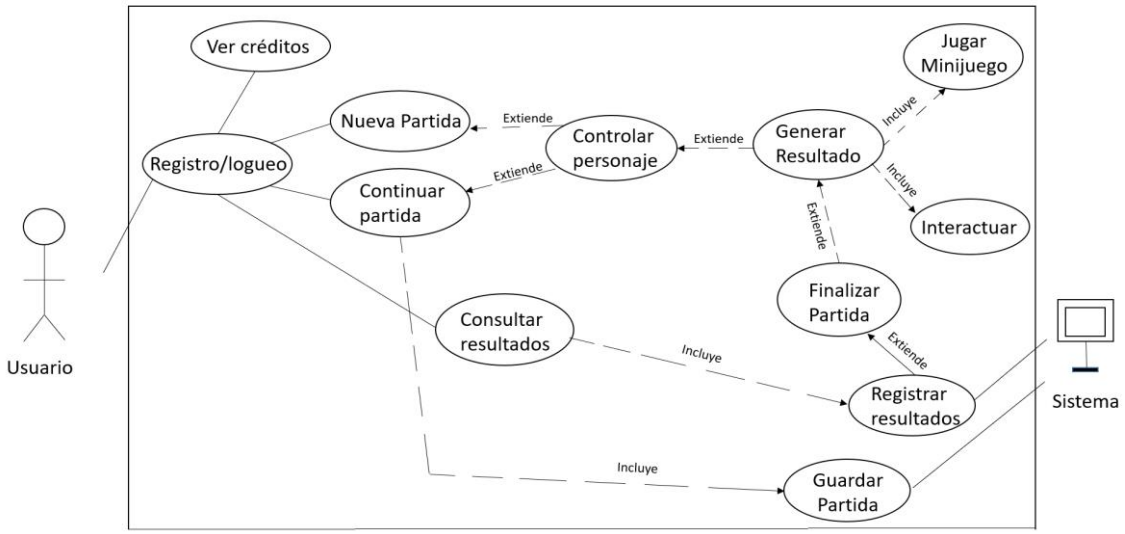
En el proyecto se distinguen dos actores:

- Usuario: Es la persona que juega al videojuego y consulta datos en la aplicación.
- Sistema: Es el encargado de llevar la información del juego a la base de datos, para que se pueda consultar.

El diagrama de casos de uso que aparece a continuación representa el flujo general de acciones que el usuario realizará al utilizar el producto, además de mostrar el momento en el que se vuelca la información a la base de datos.

Consideraciones:

- Guardar partida es una acción del sistema porque se hace automáticamente al finalizar cada escena
- Se considera Finalizar partida como una acción porque es el momento en el que el juego realiza el registro de los datos que los jugadores podrán consultar



4. DISEÑO DE LA SOLUCIÓN

4.1. Diseño de la interfaz de usuario y prototipos

En este apartado se definen las características que marcan la presentación de la aplicación, que tienen como objetivo ofrecer una experiencia agradable e intuitiva al usuario.

Colores

La paleta de colores escogida comprende azules y verdes suaves, que den la sensación de naturaleza que ambienta al pueblo donde se desarrolla la historia del juego.

El azul es 8DFFF8. Transmite seguridad, tranquilidad, protección y salud.

El verde es B1F5D0. Es el color de la esperanza, el optimismo y la buena suerte.

Estética

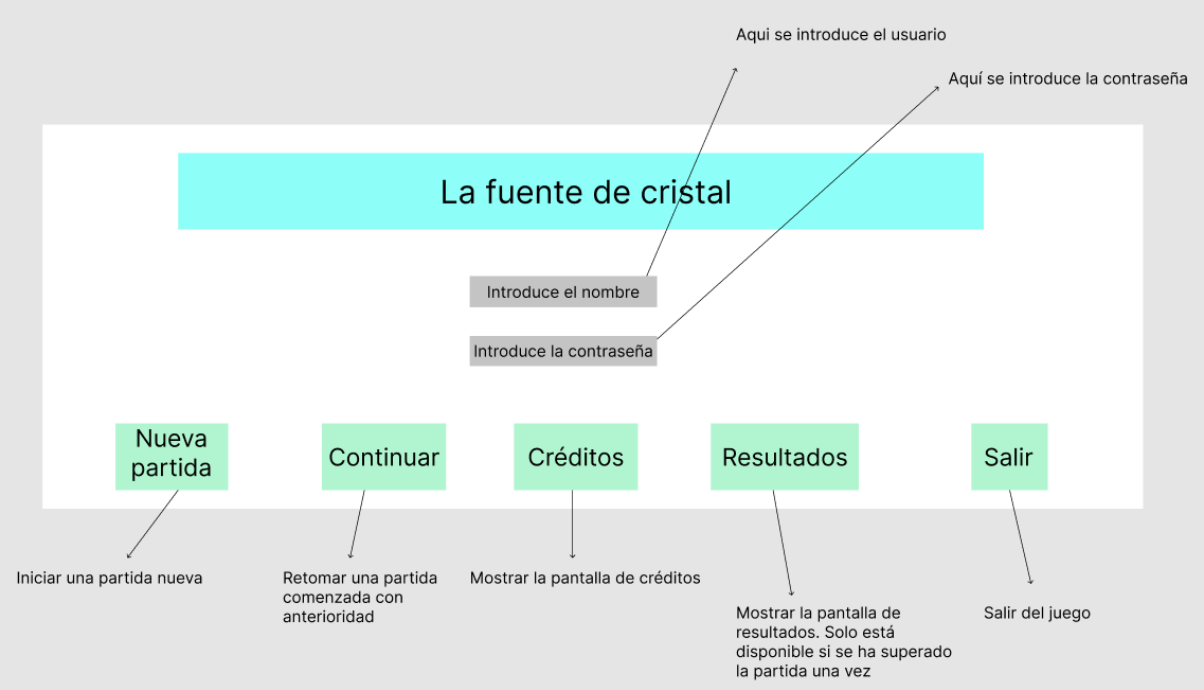
El juego se desarrolla en un escenario principal, el cual presenta una ambientación 2D. Para ello, se emplean assets que representan el entorno del juego ideado por el desarrollador, de forma comprensible para el jugador.

Prototipado

A continuación se muestran de forma conceptual las diferentes pantallas que el jugador verá a lo largo del juego.

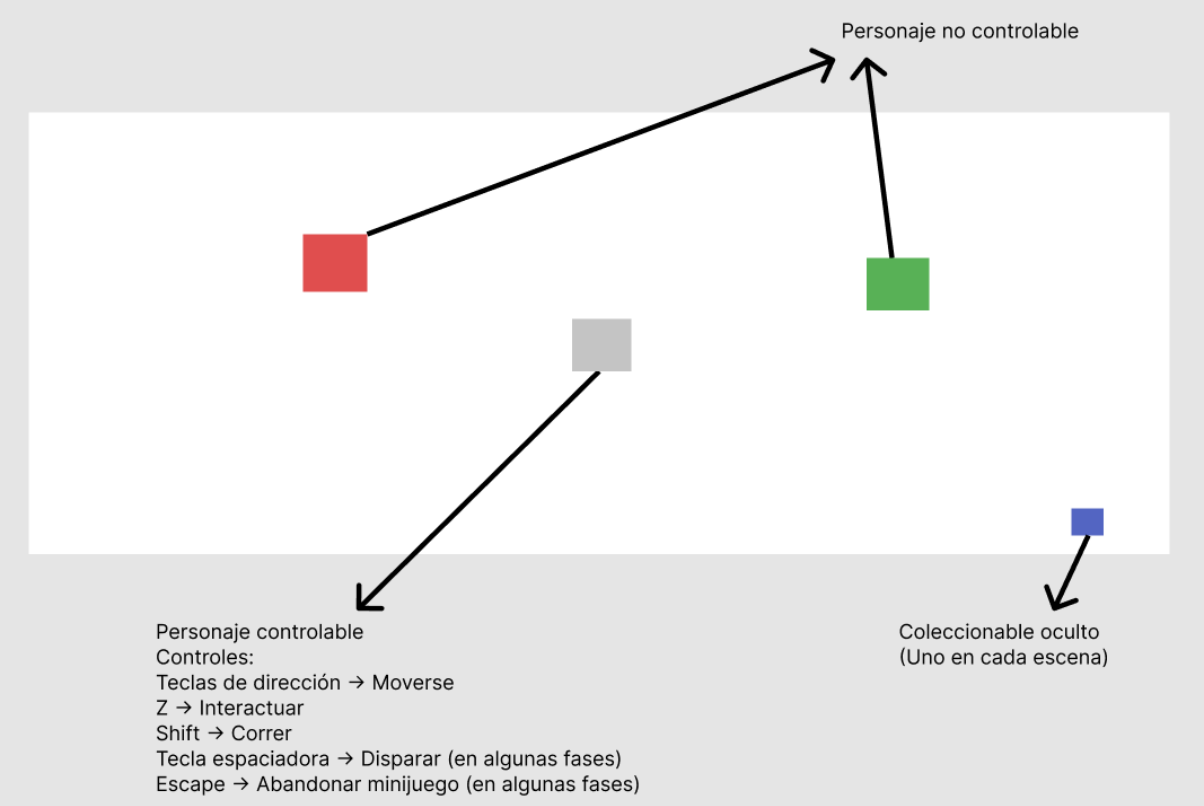
Pantalla inicial

La interfaz que se presenta al jugador al iniciar el juego.



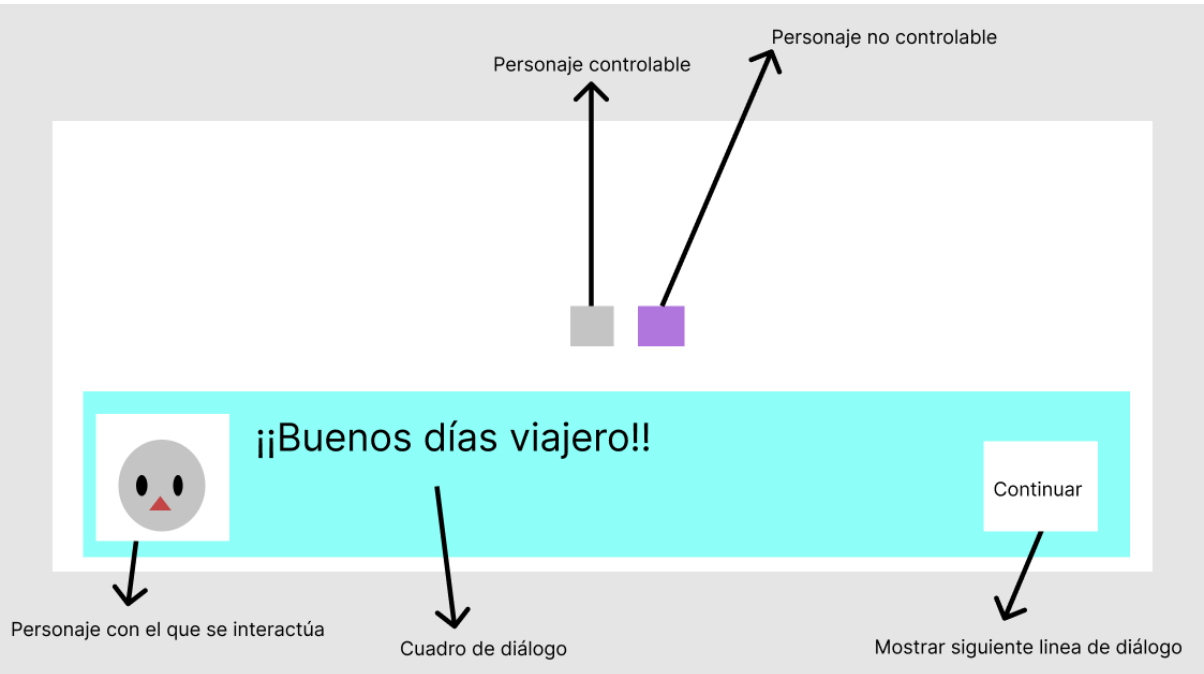
Pantalla de juego

La pantalla en la que el jugador controla a su personaje.



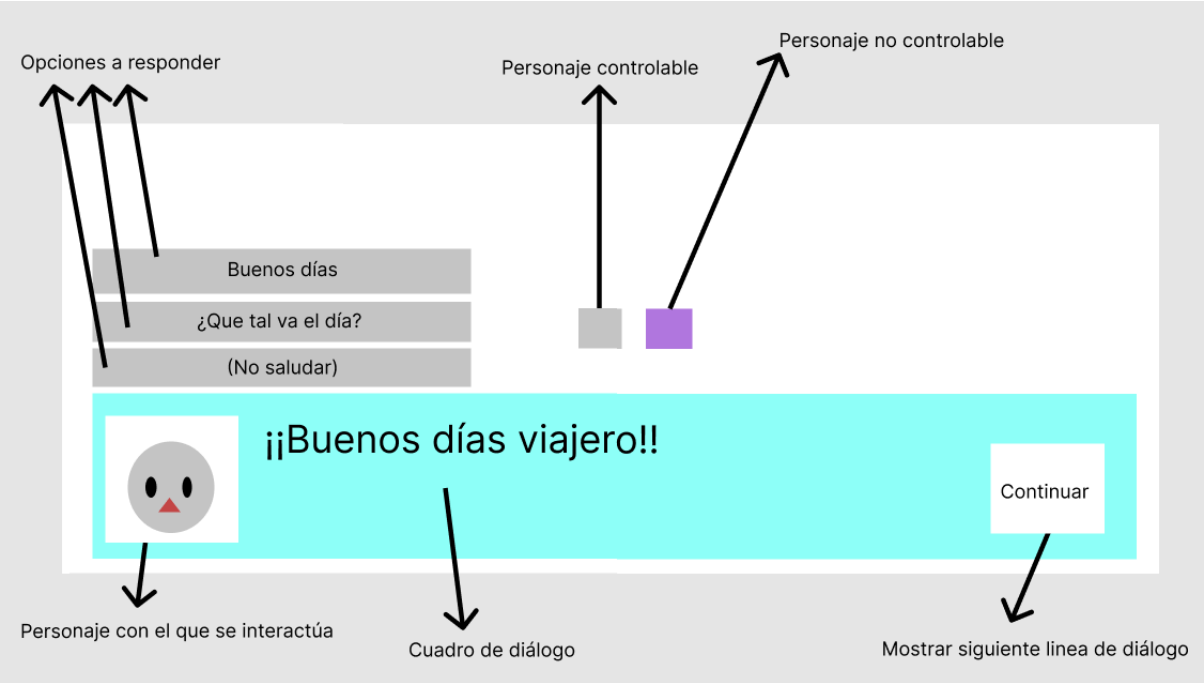
Pantalla de diálogo

La pantalla que se muestra en la parte inferior cuando se interactúa con otros personajes o elementos.



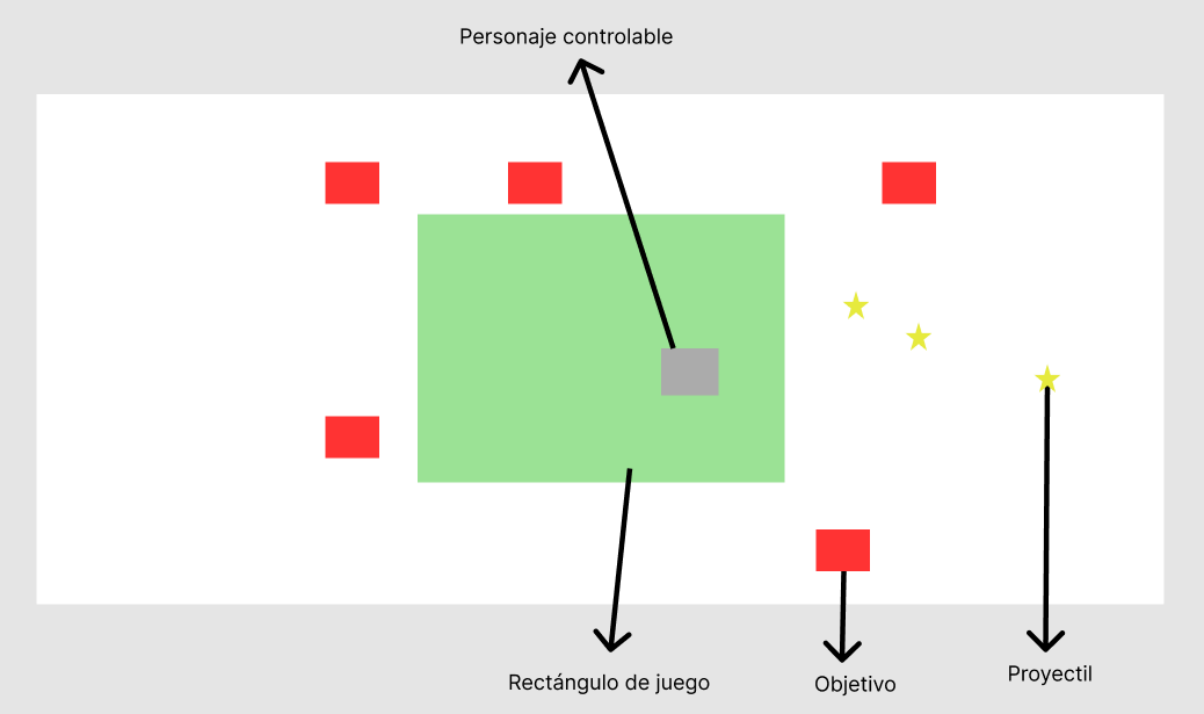
Pantalla de decisión

En algunos diálogos, el jugador deberá decidir entre las opciones propuestas para responder a los personajes. Algunas decisiones pueden afectar al desarrollo de la historia.



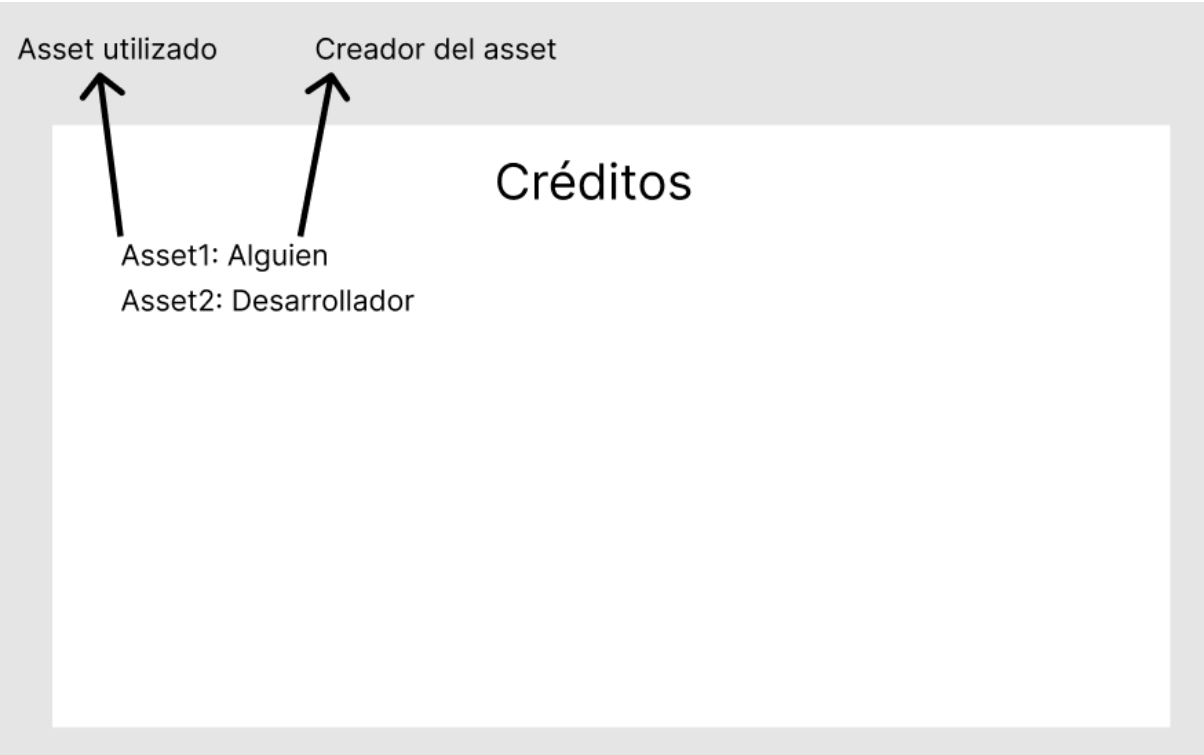
Pantalla de minijuego

En algunas partes de la historia, el jugador deberá poner a prueba su habilidad en minijuegos. Solo se jugarán una vez y su resultado será registrado. La siguiente imagen representa un minijuego donde debes disparar a los objetivos que van apareciendo.



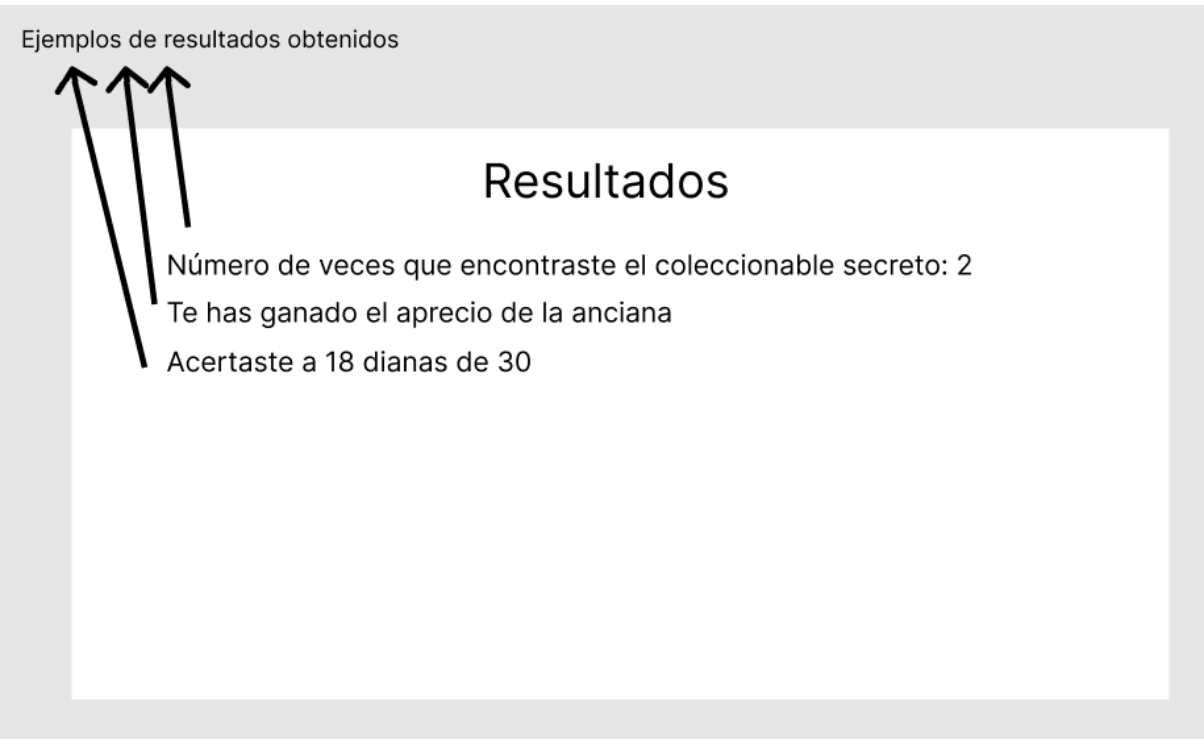
Pantalla de créditos

Desde el menú de inicio, el jugador puede acceder a la pantalla de créditos, donde se acredita debidamente a los creadores de los assets utilizados en el juego.



Pantalla de resultados

Al finalizar la historia, se presentará al jugador una pantalla que le muestra las decisiones y resultados obtenidos a lo largo del juego.



4.2. Diagrama de clases

El objetivo del diagrama de clases es establecer los elementos principales del juego y las relaciones entre ellos.

Protagonista

El personaje que el jugador controla durante el juego. Será el que interactúe con los demás elementos del juego. Contará un un sistema de control y será perseguido por la cámara

Interactuable

Elementos con los que el protagonista interactuará durante el juego. Incluye personajes no controlables y otros objetos que lleven asociado diálogo. El sistema de diálogo se lleva a cabo mediante ficheros JSON generados con el lenguaje Ink, y será gestionado por un InkManager.

Evento

Son diálogos que se activan sin necesidad de interactuar con un personaje u objeto, y sirven para guiar la historia del juego. Comprenden, por ejemplo, pensamientos del protagonista al finalizar cada escena.

Resultado

Durante el transcurso del juego, el jugador irá generando resultados que se almacenan, afectan al desarrollo de la partida y serán mostrados al finalizar el juego. La gestión de resultados se lleva a cabo en un DecisionManager. Los resultados se pueden obtener de diferentes formas como responder a una pregunta, jugar a un minijuego o encontrar un coleccionable.

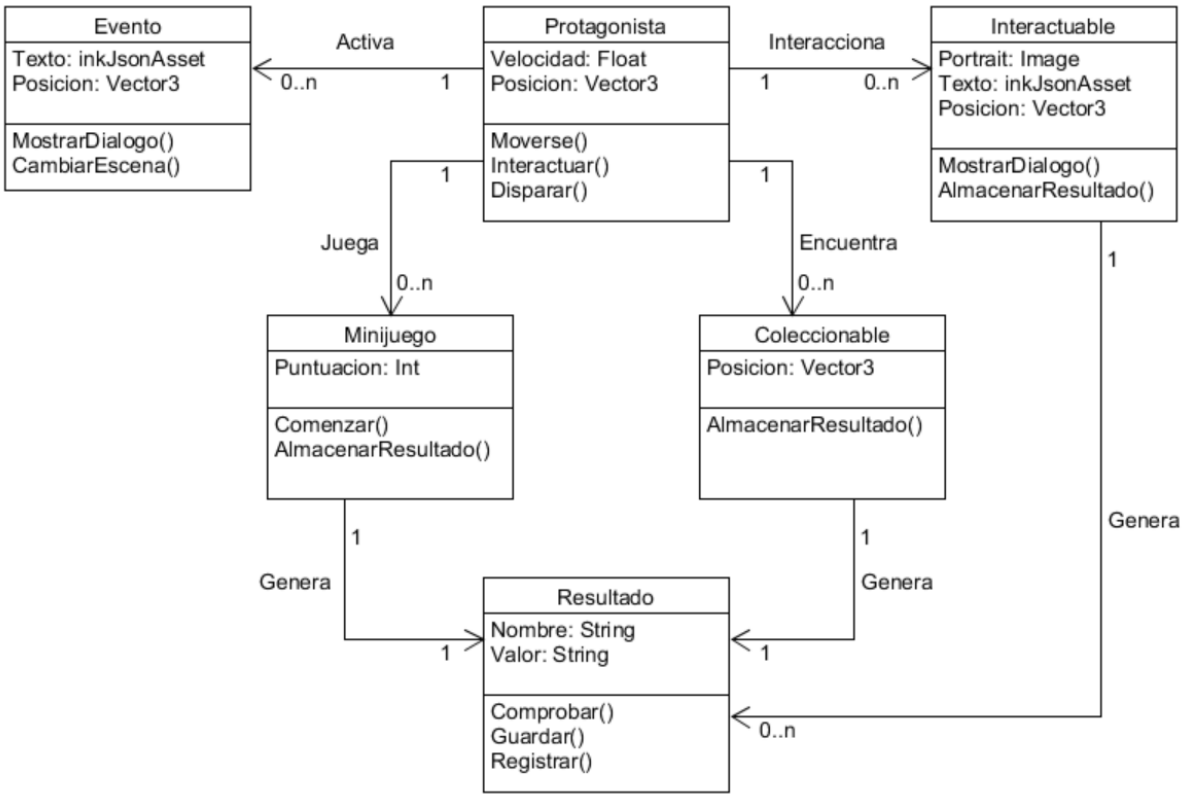
Minijuego

En determinadas partes del juego, el jugador será puesto a prueba en pequeños juegos de distinta naturaleza, que son gestionados individualmente con distintos scripts debido a sus necesidades específicas.

Coleccionable

En todas las escenas del juego habrá un coleccionable que no es necesario encontrar para avanzar la historia, pero será tenido en cuenta.

La siguiente imagen refleja el diagrama de clases:



4.3. Diseño de persistencia de la información

El concepto en el que se basa el juego implica la necesidad de poder almacenar datos de diferentes usuarios para obtener estadísticas basadas en los resultados que se generan en las partidas. Para ello, se utilizará la plataforma Firebase, que utiliza lenguaje NoSQL para realizar consultas.

Se contempla también la posibilidad de almacenar la información de las partidas realizadas en un dispositivo, mediante la herramienta phpMyAdmin, que utiliza lenguaje SQL

La base de datos del juego contará con tres tablas:

Usuario

Recoge datos de los usuarios registrados. Sus campos son:

- id_usuario: Clave primaria, autoincremental
- correo: Cuenta de correo del usuario
- password: Clave de la cuenta del usuario
- permisos: Identifica si el usuario es jugador o administrador

PartidaGuardada

Recoge las variables que el juego debe tener en cuenta a lo largo de una partida, con el objetivo de tener un sistema de guardado eficaz. El juego guardará la partida al finalizar cada escena. Sus campos son:

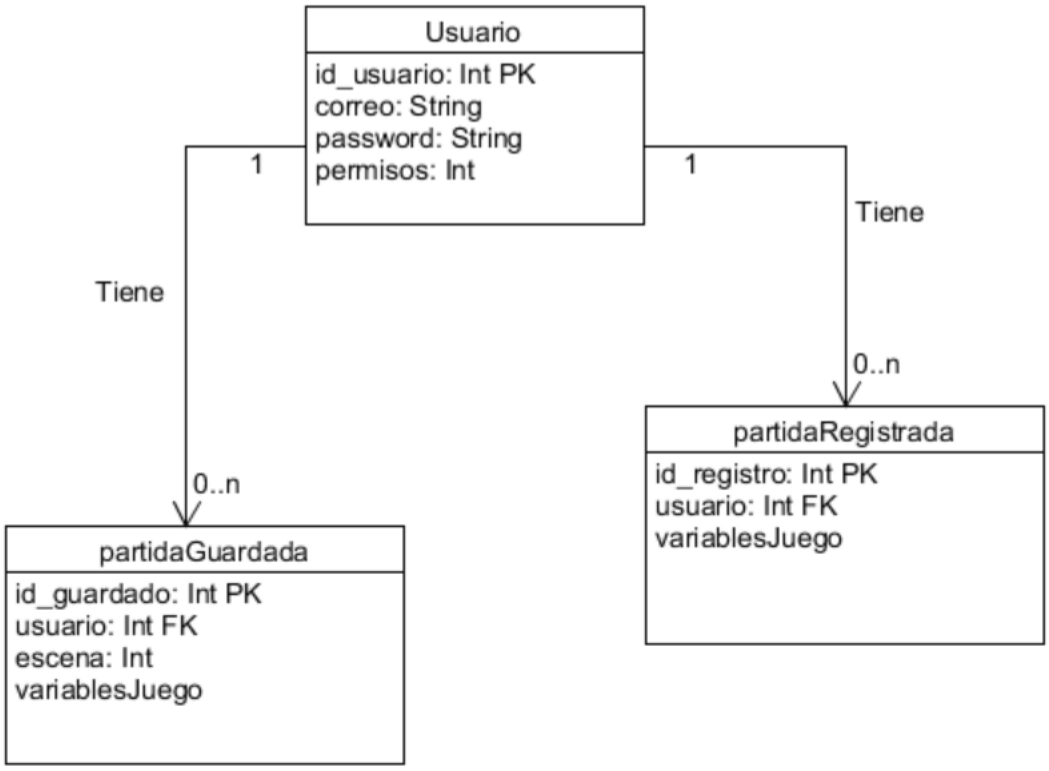
- id_guardado: Clave primaria, autoincremental
- usuario: Clave secundaria. Es el id del usuario
- escena: Escena del juego en la que se encuentra el usuario. Se almacena para que al volver a jugar, el jugador aparezca en la escena que estaba jugando la última vez.
- Variables del juego: Distintos datos que se almacenan a lo largo de la partida, y que es necesario almacenar para que influyan a lo largo de la partida. Un ejemplo es el número de veces que encuentras el coleccionable.

PartidaRegistrada

Recoge las variables que los jugadores han conseguido a lo largo de la partida. Cuando se finaliza el juego, se realiza un registro en esta tabla, que el jugador podrá consultar. Sus campos son:

- id_registro: Clave primaria, autoincremental
- usuario: Clave secundaria. Es el id del usuario
- Variables del juego: Los diferentes resultados finales de la partida.

La siguiente imagen refleja el diseño de persistencia de la información:

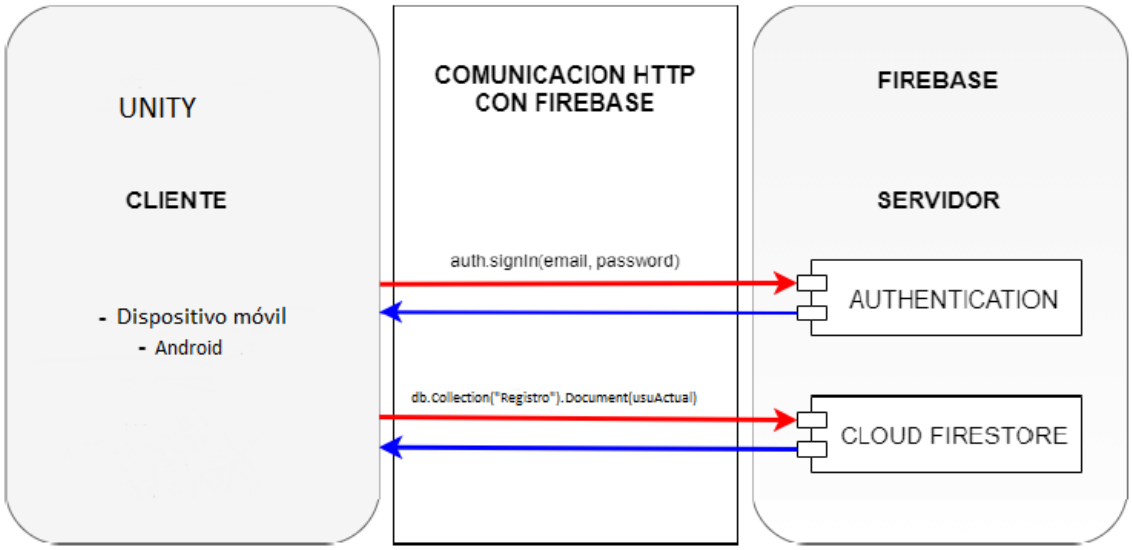


4.4. Arquitectura del sistema

El diagrama de arquitectura representa la estructura general de alto nivel del sistema. A continuación se muestra el diagrama, distinguiendo al cliente, el servidor, y la conexión entre ambos.

El servidor es Firebase, y cuenta con dos servicios que son útiles para el correcto funcionamiento de mi juego:

- Authentication: Permite registrar a los usuarios de manera sencilla. En este caso, el usuario debe ser un correo.
- Firestore Database: Permite almacenar datos de la aplicación en una base de datos NoSQL. En mi aplicación almacena los datos del juego, para que el jugador pueda continuar jugando en otro momento conservando el progreso realizado anteriormente. Además, se consultará la base de datos para ver las estadísticas de usuarios después de finalizar la partida.



La aplicación se desarrolla en Unity, una plataforma que ofrece muchas herramientas para el desarrollo de videojuegos. La aplicación está siendo desarrollada para Android, y probada también en Windows. Sería posible extender la aplicación a iOS, Windows, Linux, Mac o videoconsolas.

5. IMPLEMENTACIÓN DE LA SOLUCIÓN

5.1. Análisis tecnológico

El proyecto está dividido en dos partes fundamentales: El videojuego y la base de datos. En el siguiente apartado se muestran las opciones consideradas para cada parte, y la decisión final.

Videojuego

Las opciones mejor valoradas para desarrollo de videojuegos se presentan a continuación:

- GameMaker Studio: Herramienta intuitiva para desarrollo de juegos 2D con su propio lenguaje de programación GML, basado en C. Facilita la creación de juegos con poco o nada de código, dependiendo de las necesidades del desarrollador. También incluye software integrado de gráficos con herramientas de animación.
- Unity: Herramienta capacitada para el desarrollo de juegos 2D y 3D. Utiliza lenguaje C#. Permite la ejecución de los juegos desarrollados en distintas plataformas como PC, iPhone, Android, o videoconsolas. Presenta un nivel

de dificultad mayor que otras alternativas, pero es usada tanto por desarrolladores que se inician en la programación de videojuegos, como por compañías profesionales que desarrollan juegos famosos, lo que da una idea de su potencial.

- RPG Maker: Herramienta pensada para principiantes, que se diseñó para facilitar la creación de videojuegos del género RPG en 2D de manera sencilla, aunque también se pueden desarrollar juegos de otros géneros. Utiliza su propio lenguaje de programación basado en Ruby, pero presenta muchas opciones de parametrización.
- Unreal Engine 4: Herramienta profesional de desarrollo de videojuegos, que cuenta con opciones avanzadas. Posee un sistema llamado Blueprint que permite desarrollar rápidamente una lógica de juego completa sin necesidad de código, aunque también permite programar en lenguaje C++. Permite la exportación de los juegos creados a dispositivos móviles, PC o videoconsolas.

Base de datos

Las opciones mejor valoradas para desarrollo de la base de datos se presentan a continuación:

- Firebase: Herramienta creada por Google diseñada para facilitar el desarrollo de aplicaciones, incluyendo videojuegos. Facilita la implementación de funcionalidades que hacen la aplicación más segura y manejable. Los servicios ofrecidos de mayor interés son la Autenticación por distintos métodos como correo o Google, y Firestore, que ofrece al desarrollador la posibilidad de crear bases de de datos NoSQL en la nube.
- phpMyAdmin: Aplicación web que facilita la administración de bases de datos MySQL, mediante el uso de una interfaz. Está basado en PHP. Es la herramienta de administración de MySQL más popular. Permite conexión con servidores remotos. La gestión de creación, modificación y borrado de bases de datos o tablas se puede gestionar fácilmente desde la interfaz o mediante código SQL.

Decisión

Finalmente, se decidió utilizar Unity frente a RPG Maker y Firebase frente a phpMyAdmin y MySQL, debido a que RPG Maker es una herramienta menos flexible que no está preparada para la interacción con Firebase, que es necesario para algunas funciones del juego.

En cuanto a la base de datos, se eligió Firebase, debido a que es posible implementarlo con Unity y sus servicios de Firestore y Autenticación se adaptan mejor al almacenamiento de variables y sistema de guardado que requiere el proyecto. Además, el volumen de datos gestionado por el juego no es elevado y permite utilizar toda la funcionalidad de Firebase de manera gratuita.

5.2. Elementos a implementar

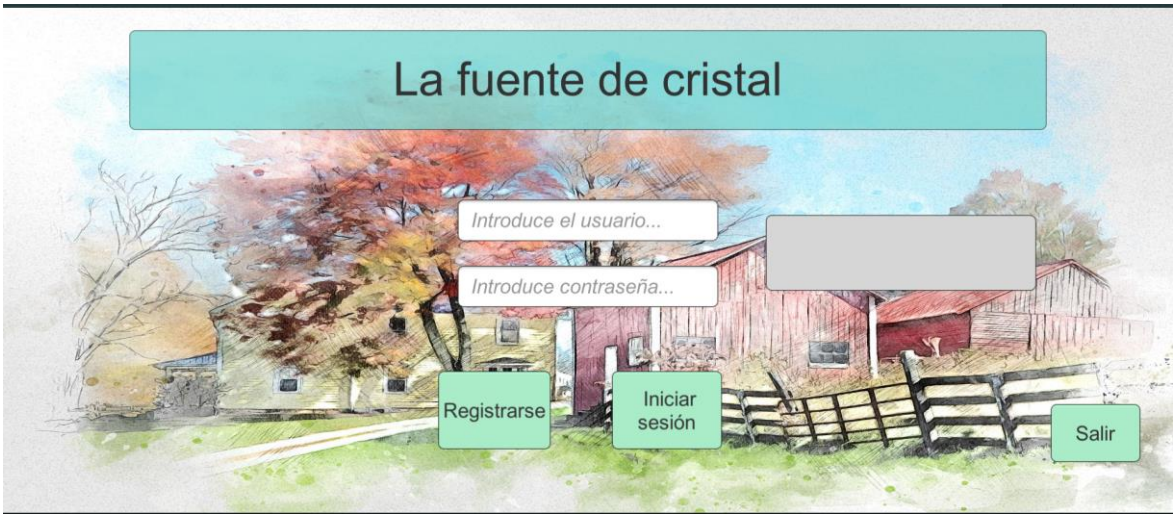
En este apartado se muestra el código relacionado con los aspectos más destacables del juego.

Login y registro

Al iniciar el juego, la aplicación se conecta al servidor de Firebase.

```
using Firebase;
using Firebase.Analytics;
using UnityEngine;

Script de Unity (1 referencia de recurso) | 0 referencias
public class FirebaseInit : MonoBehaviour
{
    public Firebase.FirebaseApp app;
    // Start is called before the first frame update
    Mensaje de Unity | 0 referencias
    void Start()
    {
        Firebase.FirebaseApp.CheckAndFixDependenciesAsync().ContinueWith(task => {
        var dependencyStatus = task.Result;
        if (dependencyStatus == Firebase.DependencyStatus.Available) {
            // Create and hold a reference to your FirebaseApp,
            // where app is a Firebase.FirebaseApp property of your application class.
            app = Firebase.FirebaseApp.DefaultInstance;
            // Set a flag here to indicate whether Firebase is ready to use by your app.
        } else {
            UnityEngine.Debug.LogError(System.String.Format(
            "Could not resolve all Firebase dependencies: {0}", dependencyStatus));
            // Firebase Unity SDK is not safe to use here.
        }
        });
    }

```

En el menú inicial, el jugador deberá registrarse, o iniciar sesión con un usuario registrado, para poder acceder al juego:



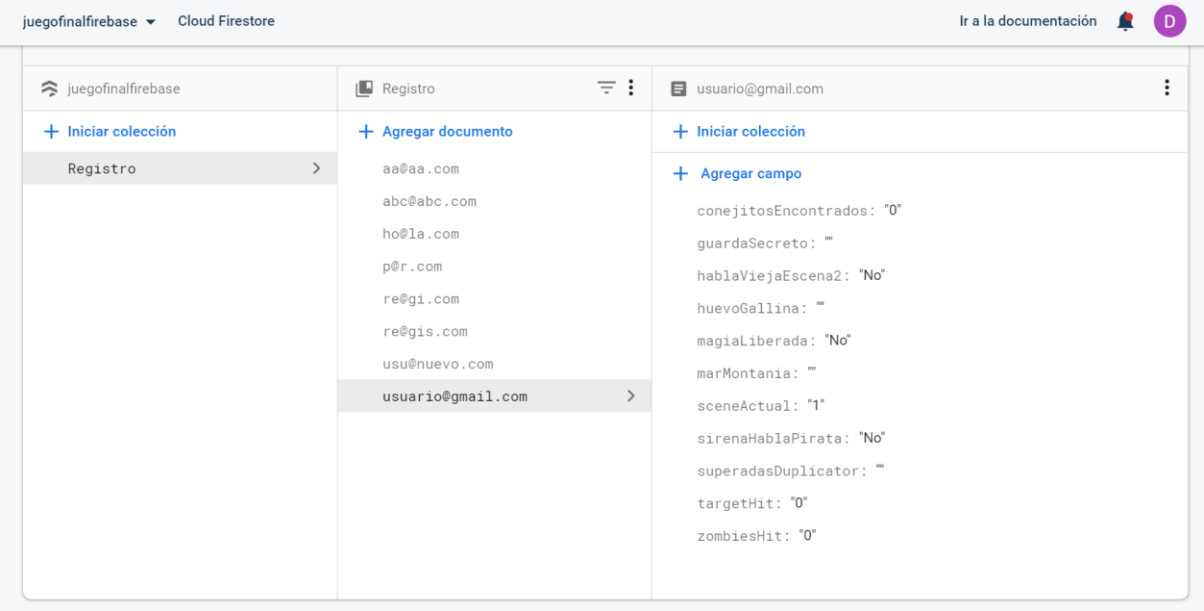
Función de registro:

```
public void registerUser()
{
    mainCamera.GetComponent<clickSound>().playClickSound();

    if (textUsu.GetComponent<Text>().text.Length > 0 && textPass.GetComponent<Text>().text.Length >= 8)
    {
        FirebaseAuth.DefaultInstance.CreateUserWithEmailAndPasswordAsync(textUsu.GetComponent<Text>().text, textPass.GetComponent<Text>().tex
        if (task.IsCanceled) {
            errorRegistro = true;
            Debug.LogError("CreateUserWithEmailAndPasswordAsync was canceled.");
            return;
        }
        if (task.IsFaulted) {
            errorRegistro = true;
            Debug.LogError("CreateUserWithEmailAndPasswordAsync encountered an error: " + task.Exception);
            return;
        }

        if(task.IsCompleted)
        {
            registroUsu = true;
            errMessage.transform.GetChild(0).GetComponent<Text>().text = "Registrado con éxito";
        }

        // Firebase user has been created.
        Firebase.Auth.FirebaseUser newUser = task.Result;
        Debug.LogFormat("Firebase user created successfully: {0} ({1})",
            newUser.DisplayName, newUser.UserId);

        });

    }
    else
    {
        errMessage.transform.GetChild(0).GetComponent<Text>().text = "El usuario debe ser un correo y la contraseña tener al menos 8 caracter
    }
}
```

Al finalizar el registro, se procede a crear un documento para el usuario en firestore que almacenará los datos necesarios para guardar y cargar la partida.

```
public void createDocument()
{
    db = FirebaseFirestore.DefaultInstance;
    DocumentReference docRef = db.Collection("Registro").Document(textUsu.GetComponent<Text>().text);
    Dictionary<string, object> reg = new Dictionary<string, object>
    {
        { "sceneActual", "1" },
        { "huevoGallina", "" },
        { "marMontania", "" },
        { "targetHit", "0" },
        { "guardaSecreto", "" },
        { "conejitosEncontrados", "0" },
        { "hablaViejaEscena2", "No" },
        { "sirenaHablaPirata", "No" },
        { "superadasDuplicator", "" },
        { "zombiesHit", "0" },
        { "magiaLiberada", "No" }

    };
    docRef.SetAsync(reg).ContinueWithOnMainThread(task => {
            Debug.Log("Saved user in database");
    });
}
```

El documento generado se encuentra en Firesetore.



Función de inicio de sesión:

```
public void loginUser([SerializeField] GameObject buttonsContainer)
{
    mainCamera.GetComponent<clickSound>().playClickSound();

    FirebaseAuth.DefaultInstance.SignInWithEmailAndPasswordAsync(textUsu.GetComponent<Text>().text, textPass.GetComponent<Text>().text).Contir
    if (task.IsCanceled) {
        errorLogin = true;
        Debug.LogError("SignInWithEmailAndPasswordAsync was canceled.");
        return;
    }
    if (task.IsFaulted) {
        errorLogin = true;
        Debug.LogError("SignInWithEmailAndPasswordAsync encountered an error: " + task.Exception);
        return;
    }

    if(task.IsCompleted)
    {
        Debug.Log(loginUsu);
        loginUsu = true;
        
        Debug.Log(loginUsu);
        errMessage.transform.GetChild(0).GetComponent<Text>().text = "Sesión iniciada con éxito";
    }

    Firebase.Auth.FirebaseUser newUser = task.Result;
    Debug.LogFormat("User signed in successfully: {0} ({1})",
        newUser.DisplayName, newUser.UserId);
    });

}
```

Al finalizar el inicio de sesión, se recogen de Firestore los datos del documento perteneciente al usuario, y se preparan para ser utilizados a lo largo de la partida.

```
public void loadDocument()
{
    db = FirebaseFirestore.DefaultInstance;
    DocumentReference docRef = db.Collection("Registro").Document(decisionManager.GetComponent<DecisionManager>().usuActual);
    docRef.GetSnapshotAsync().ContinueWithOnMainThread(task =>
    {
    DocumentSnapshot snapshot = task.Result;
    if (snapshot.Exists)
    {
        Debug.Log("Document data for "+ snapshot.Id+" document:");
        Dictionary<string, object> reg = snapshot.ToDictionary();
        foreach (KeyValuePair<string, object> pair in reg)
        {
            if(pair.Key != "sceneActual")
            {
                Debug.Log(pair.Key + ": "+ pair.Value);
                decisionManager.GetComponent<DecisionManager>().listaValores[decisionManager.GetComponent<DecisionManager>().listaVariables.In
                Debug.Log(pair.Key + ": "+ pair.Value + " cargado");
                if(pair.Key == "conejitosEncontrados")
                {
                    decisionManager.GetComponent<DecisionManager>().conejitosEncontrados = int.Parse(pair.Value.ToString());
                }
            }
            else
            {
                decisionManager.GetComponent<DecisionManager>().sceneActual = int.Parse(pair.Value.ToString());
            }
        }
        canLoadScene = true;
    }
    else
    {
        Debug.Log("Document "+snapshot.Id+" does not exist!");
    }
    });
}
```

Menú del jugador

Es la pantalla donde el jugador tiene acceso a las opciones que ofrece el juego, que son iniciar partida, continuar partida, ver los créditos, ver los resultados (tras finalizar la partida), cerrar sesión y salir de la aplicación.

Nueva partida carga la primera escena del juego.

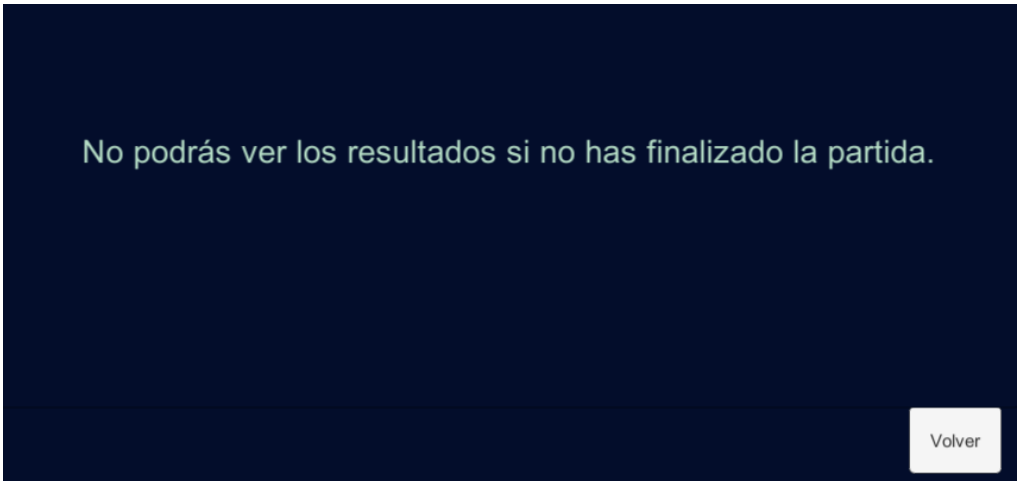
Continuar carga la escena en la que el jugador dejó de jugar por última vez (Los datos se guardan al finalizar una escena)

```
public void loadDocument()
{
    db = FirebaseFirestore.DefaultInstance;
    DocumentReference docRef = db.Collection("Registro").Document(decisionManager.GetComponent<DecisionManager>().usuActual);
    docRef.GetSnapshotAsync().ContinueWithOnMainThread(task =>
    {
    DocumentSnapshot snapshot = task.Result;
    if (snapshot.Exists)
    {
        Debug.Log("Document data for "+ snapshot.Id+" document:");
        Dictionary<string, object> reg = snapshot.ToDictionary();
        foreach (KeyValuePair<string, object> pair in reg)
        {
            if(pair.Key != "sceneActual")
            {
                Debug.Log(pair.Key + ": "+ pair.Value);
                decisionManager.GetComponent<DecisionManager>().listaValores[decisionManager.GetComponent<DecisionManager>().listaVariables.In
                Debug.Log(pair.Key + ": "+ pair.Value + " cargado");
                if(pair.Key == "conejitosEncontrados")
                {
                    decisionManager.GetComponent<DecisionManager>().conejitosEncontrados = int.Parse(pair.Value.ToString());
                }
            }
            else
            {
                decisionManager.GetComponent<DecisionManager>().sceneActual = int.Parse(pair.Value.ToString());
            }
        }
        canLoadScene = true;
    }
    else
    {
        Debug.Log("Document "+snapshot.Id+" does not exist!");
    }
    });
}
```

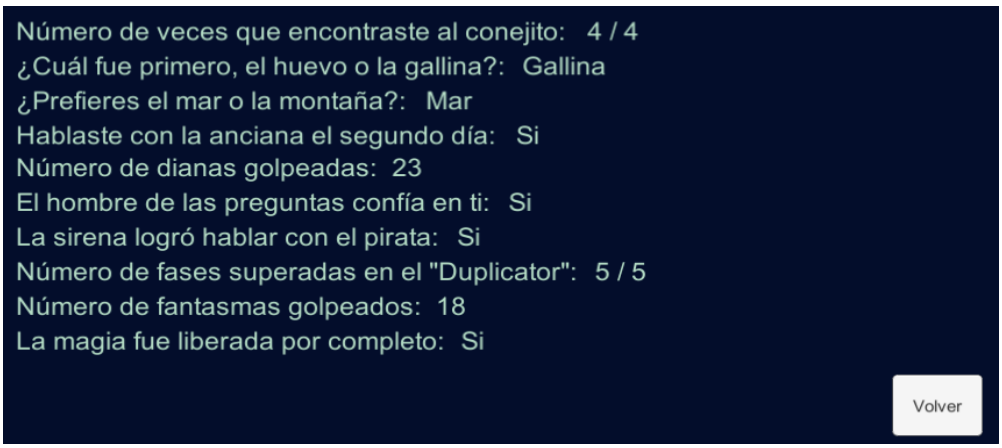
Créditos abre un panel que muestra las atribuciones y agradecimientos.

Resultados muestra un panel con los resultados obtenidos en una partida finalizada, o indica que es necesario finalizar la partida para verlos.

Pantalla de resultados en partida sin completar:



Pantalla de resultados en partida finalizada:



Cerrar sesión vuelve a la pantalla de login y registro.

Salir cierra la aplicación.

Funcionamiento del juego

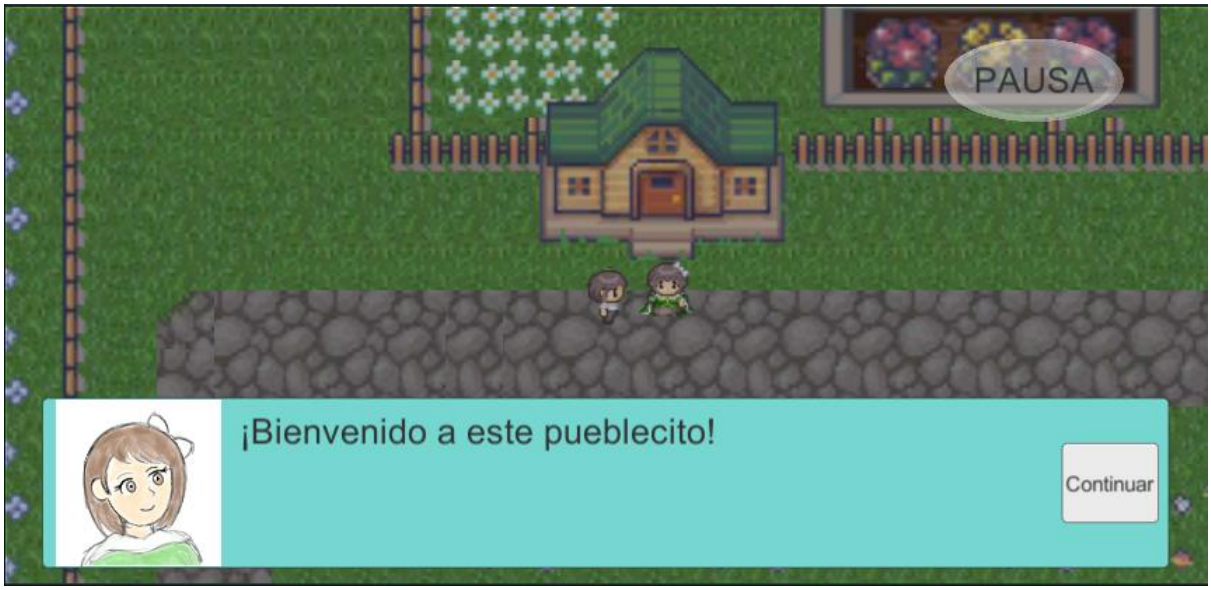
Al iniciar la partida, podemos controlar a un personaje. Se mueve mediante una palanca de movimiento, interactúa con su entorno con el botón A, realiza comandos especiales con el botón B, y accede a un menú de pausa pulsando el botón PAUSA.


```
void Update()
{
    var rigidbody = GetComponent<Rigidbody2D>();
    if(canMove)
    {
        forwardInput = Input.GetAxis("Vertical");
        horizontalInput = Input.GetAxis("Horizontal");
        if(joystick.Horizontal != 0 || joystick.Vertical != 0)
        {
            if(!isWalking)
            {
                isWalking = true;
                anim.SetBool("isWalking",true);
            }
            anim.SetFloat("Horizontal", joystick.Horizontal);
            anim.SetFloat("Vertical", joystick.Vertical);
            this.transform.Translate(Vector3.up * playerSpeed * Time.deltaTime * joystick.Vertical);
            this.transform.Translate(Vector3.right * playerSpeed * Time.deltaTime * joystick.Horizontal);
        }
        else
        {
            isWalking = false;
            anim.SetBool("isWalking",false);
        }

        if (joyButtonB.GetComponent<joystickController>().pressed || Input.GetKey(KeyCode.Space))
        {
            if(canShoot)
            {
                joyButtonB.GetComponent<joystickController>().pressed = false;
                Vector3 v3 = new Vector3(this.transform.position.x,this.transform.position.y,-1);
                Instantiate(star, v3, this.transform.rotation);
                sou.PlayOneShot(starSound, 0.5f);
            }
        }

        if (joyButtonPause.GetComponent<joystickController>().pressed)
        {
            if (!canShoot)
            {
                panelPausa.SetActive(true);
            }
```

El menú de pausa presenta los controles del juego y permite volver a la partida o ir al menú inicial sin guardar


```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;

Script de Unity (2 referencias de recurso) | 0 referencias
public class pauseMenuManager : MonoBehaviour
{
    [SerializeField] private GameObject panelPausa;

    0 referencias
    public void cerrarMenuPausa()
    {
        panelPausa.SetActive(false);
    }

    0 referencias
    public void volverMenuPrincipal()
    {
        SceneManager.LoadSceneAsync("Intro");
    }
}
```

El sistema de diálogos se ha elaborado usando el lenguaje Ink, el cual ofrece la posibilidad de escribir diálogo de forma sencilla y permite la gestión de variables y la toma de decisiones en medio de la conversación. El resultado de escribir código en Ink es un JSON que puede asociarse a los elementos que dialogan. En el juego hay un InkManager encargado de controlar este sistema.

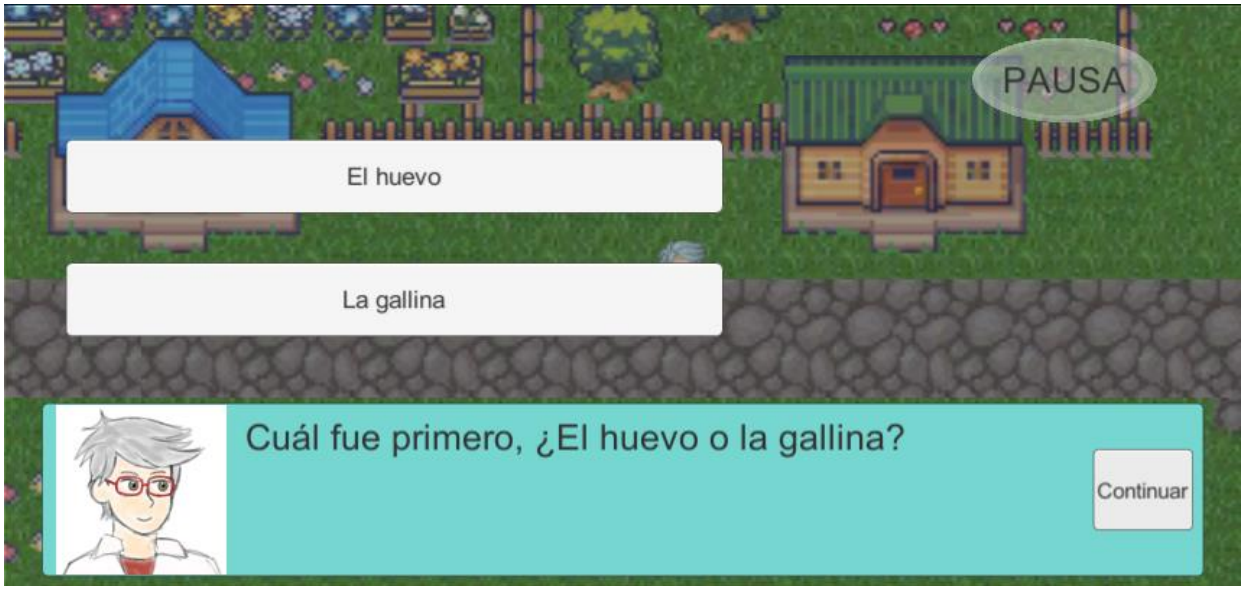


Los habitantes del pueblo comienzan a conversar con tu personaje mediante el siguiente código:

```
void Update()
{
    if(isPlayerInRange && joyButtonA.GetComponent<joystickController>().pressed)
    {
        if(player.GetComponent<PlayerController>().canMove)
        {
            //didDialogueStart = true;
            dialogueImage.sprite = characterPortrait;
            player.GetComponent<PlayerController>().canMove = false;
            inkManager.GetComponent<InkManager>().StartStory(inkJsonAsset, nomVariable, visited, personTag);
            //player.GetComponent<PlayerController>().canMove = true;
            visited = true;
        }
```

El diálogo se avanza pulsando el botón continuar. En algunas ocasiones, se presentan al jugador diferentes opciones, de las que se debe elegir una para seguir dialogando.

```
private void DisplayChoices()
{
    // checks if choices are already being displayed
    if (_choiceButtonContainer.GetComponentsInChildren<Button>().Length > 0) return;

    for (int i = 0; i < _story.currentChoices.Count; i++) // iterates through all choices
    {

        var choice = _story.currentChoices[i];
        var button = CreateChoiceButton(choice.text); // creates a choice button

        button.onClick.AddListener(() => OnClickChoiceButton(choice));
    }
}
```



Algunas decisiones son almacenadas en un DecisionManager, para poder recogerlas al guardar y cargar datos, y tenerlas en cuenta en futuros eventos del juego. Este elemento es un singleton para no perder su información entre escenas.

```
public static DecisionManager Instance;

Mensaje de Unity | 0 referencias
private void Awake()
{
    if(Instance != null)
    {
        Destroy(gameObject);
        return;
    }

    Instance = this;
    DontDestroyOnLoad(gameObject);
}
```

InkManager llama a DecisionManager si al final de un diálogo hay una variable que almacenar

```
if(nomVariable != "")
{
    //solo ocurre la primera vez que se habla con el personaje
    if(!visited)
    {
        if(_story.variablesState[nomVariable].ToString() != "Null")
        {
            Debug.Log(_story.variablesState[nomVariable]);
            valorVariable = _story.variablesState[nomVariable].ToString();
            //registro de la variable en DecisionManager.cs
            decisionManager.GetComponent<DecisionManager>().actualizarValor(nomVariable,valorVariable);
        }
    }
}
```

```
public void actualizarValor(string nom, string valor)
{
    this.listaValores[listaVariables.IndexOf(nom)] = valor;
}
```

Si una conversación depende de una variable, esta se recoge del DecisionManager con la siguiente función:

```
public string getValue(string nom)
{
    return this.listaValores[listaVariables.IndexOf(nom)].ToString();
}
```

Al finalizar una escena, se registran en la colección del usuario en Firestore los datos que se han almacenado a lo largo de la escena.

```
public void updateDatabase()
{
    DocumentReference usuRef = db.Collection("Registro").Document(usuActual);
    db.RunTransactionAsync(transaction =>
        {
            return transaction.GetSnapshotAsync(usuRef).ContinueWith((snapshotTask) =>
            {
                DocumentSnapshot snapshot = snapshotTask.Result;
                Dictionary<string, object> updates = new Dictionary<string, object>
                {
                    { "sceneActual", sceneActual.ToString() },
                    { "huevoGallina", this.listaValores[listaVariables.IndexOf("huevoGallina")].ToString() },
                    { "marMontania", this.listaValores[listaVariables.IndexOf("marMontania")].ToString() },
                    { "targetHit", this.listaValores[listaVariables.IndexOf("targetHit")].ToString() },
                    { "guardaSecreto", this.listaValores[listaVariables.IndexOf("guardaSecreto")].ToString() },
                    { "conejitosEncontrados", this.listaValores[listaVariables.IndexOf("conejitosEncontrados")].ToString() },
                    { "hablaViejaEscena2", this.listaValores[listaVariables.IndexOf("hablaViejaEscena2")].ToString() },
                    { "sirenaHablaPirata", this.listaValores[listaVariables.IndexOf("sirenaHablaPirata")].ToString() },
                    { "superadasDuplicator", this.listaValores[listaVariables.IndexOf("superadasDuplicator")].ToString() },
                    { "zombiesHit", this.listaValores[listaVariables.IndexOf("zombiesHit")].ToString() },
                    { "magiaLiberada", this.listaValores[listaVariables.IndexOf("magiaLiberada")].ToString() }
                };
                transaction.Update(usuRef, updates);
            });
        });
}
```

Al finalizar una escena, se comprueba cual es la escena actual, y se procede a cargarla. Si la escena es la última del juego, se carga la escena del menú inicial, y se habilita la pantalla de resultados.

```
//CAMBIO DE ESCENA
if(SceneManager.GetActiveScene().name == "Scene5")
{
    decisionManager.GetComponent<DecisionManager>().sceneActual++;
    decisionManager.GetComponent<DecisionManager>().updateDatabase();
    SceneManager.LoadSceneAsync("Intro");
}
else
{
    decisionManager.GetComponent<DecisionManager>().sceneActual++;
    decisionManager.GetComponent<DecisionManager>().updateDatabase();
    SceneManager.LoadSceneAsync("Scene" + decisionManager.GetComponent<DecisionManager>().sceneActual.ToString());
}
```

A lo largo del juego hay algunos minijuegos que se pueden jugar una vez y cuyo resultado es almacenado.

El minijuego de disparos permite al jugador disparar proyectiles con forma de estrella que deben golpear la mayor cantidad de objetivos posibles.

Las estrellas se instancian al pulsar el botón B durante los minijuegos en los que se permite disparar.

```
if (joyButtonB.GetComponent<joystickController>().pressed || Input.GetKey(KeyCode.Space))
{
    if(canShoot)
    {
        joyButtonB.GetComponent<joystickController>().pressed = false;
        Vector3 v3 = new Vector3(this.transform.position.x,this.transform.position.y,-1);
        Instantiate(star, v3, this.transform.rotation);
        sou.PlayOneShot(starSound, 0.5f);
    }
}
```

Las estrellas se mueven en línea recta, en la dirección que mira el protagonista al dispararlas. Se destruyen si se alejan demasiado o si golpean algún objetivo, el cual también será destruido.

```
void Update()
{
    if(direccion == "der")
    {
        this.transform.Translate(Vector3.right * Time.deltaTime * speed);
    }
    if(direccion == "izq")
    {
        this.transform.Translate(Vector3.left * Time.deltaTime * speed);
    }
    if(direccion == "arr")
    {
        this.transform.Translate(Vector3.up * Time.deltaTime * speed);
    }
    if(direccion == "aba")
    {
        this.transform.Translate(Vector3.down * Time.deltaTime * speed);
    }


    float comparex = Math.Abs(this.transform.position.x - player.transform.position.x);
    float comparey = Math.Abs(this.transform.position.y - player.transform.position.y);

    if (comparex > 15 || comparey > 5)
    {
        Destroy(gameObject);
    }
}
```

Las dianas del minijuego se instancian con un SpawnManager. Se lanzan 30 en total, desde 4 posiciones al azar.

```
void spawnRandomTarget(){

    if(contTargets < numberTargets)
    {
        targetDirection = Random.Range(0,4);

        Vector3 spawnPos;

        switch (targetDirection)
        {
            case 0:
                spawnPos = new Vector3(120, 17 , -1);
                Instantiate(target, spawnPos, target.transform.rotation);
                break;

            case 1:
                spawnPos = new Vector3(130, 2 , -1);
                Instantiate(target, spawnPos, target.transform.rotation);
                break;

            case 2:
                spawnPos = new Vector3(109, -3, -1);
                Instantiate(target, spawnPos, target.transform.rotation);
                break;

            case 3:
                spawnPos = new Vector3(99, 11 , -1);
                Instantiate(target, spawnPos, target.transform.rotation);
                break;
        }
        contTargets++;
    }
}
```

Otro minijuego disponible es el llamado Duplicator. Consiste en mover al personaje y una copia de este, que se mueven al mismo tiempo, y conseguir que los dos pisen dos tocones al mismo tiempo. Hay 5 fases de dificultad creciente. Durante este minijuego, puedes pulsar el botón B para abandonar.

Los tocones registran si el jugador o su duplicado están en su rango

```
void Start()
{
    player = GameObject.FindWithTag("Player");
    duplicate = GameObject.FindWithTag("Duplicate");
    isPlayerInRange = false;
}

// Update is called once per frame
Mensaje de Unity | 0 referencias
void Update()
{

}

Mensaje de Unity | 0 referencias
private void OnTriggerEnter2D(Collider2D collision)
{
    if(collision.gameObject.CompareTag("Player") || collision.gameObject.CompareTag("Duplicate"))
    {
        isPlayerInRange = true;
    }
}

Mensaje de Unity | 0 referencias
private void OnTriggerExit2D(Collider2D collision)
{
    if(collision.gameObject.CompareTag("Player") || collision.gameObject.CompareTag("Duplicate"))
    {
        isPlayerInRange = false;
    }
}
```

Este minijuego se gestiona en un script llamado DuplicatorGameManager, que controla la posición de los tocones en cada fase

del juego. Se almacena el resultado del juego al terminar la última fase o al abandonar pulsando el botón B.

```
if (Time.frameCount % interval == 0)
{
    if(log1.GetComponent<LogBehavior>().isPlayerInRange && log2.GetComponent<LogBehavior>().isPlayerInRange && !phaseDone)
    {
        player.GetComponent<PlayerController>().playStarSound();
        phaseDone = true;
        switch (phase)
        {
            case 0:
                phase=1;
                translateLog(log1, 121, 68);
                translateLog(log2, 125, 63);
                phaseDone = false;
                break;

            case 1:
                phase=2;
                translateLog(log1, 110, 74);
                translateLog(log2, 122, 78);
                phaseDone = false;
                break;

            case 2:
                phase=3;
                translateLog(log1, 167, 63);
                translateLog(log2, 167, 70);
                phaseDone = false;
                break;

            case 3:
                phase=4;
                translateLog(log1, 142, 88);
                translateLog(log2, 135, 88);
                phaseDone = false;
                break;

            case 4:
                phase=5;
                phaseDone = false;
                decisionManager.GetComponent<DecisionManager>().actualizarValor("superadasDuplicator", phase.ToString());
```

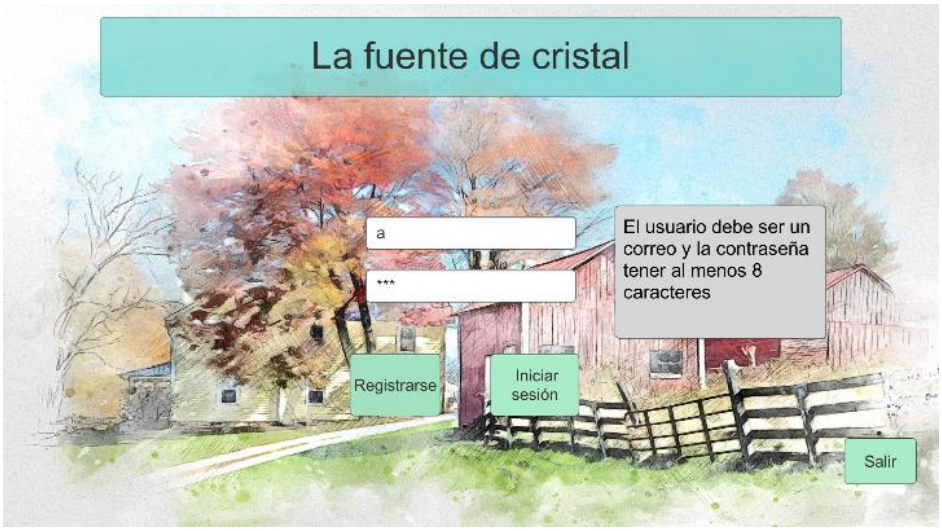
6. TESTEO Y PRUEBAS DE LA SOLUCIÓN

6.1. Plan de pruebas

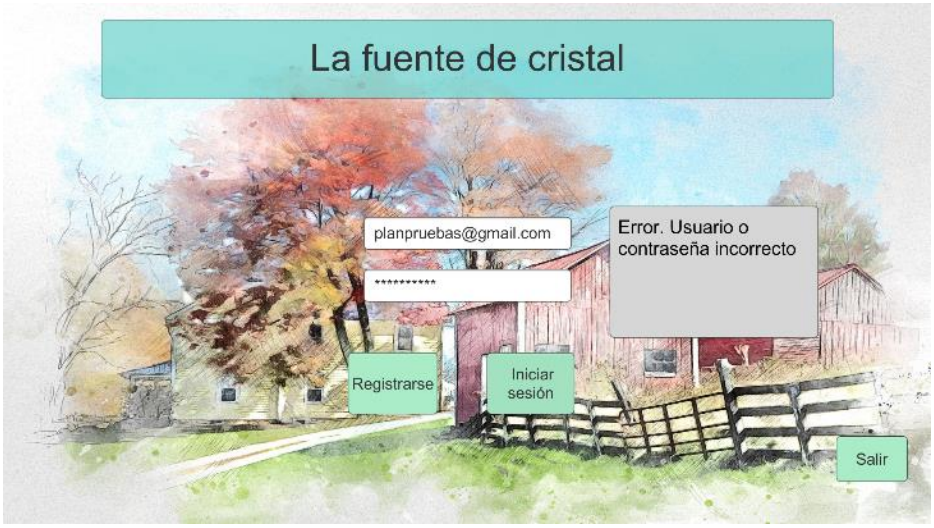
Para comprobar el correcto funcionamiento del juego, se ha elaborado un extenso plan de pruebas desde el registro hasta finalizar la partida.

Pruebas de login/registro

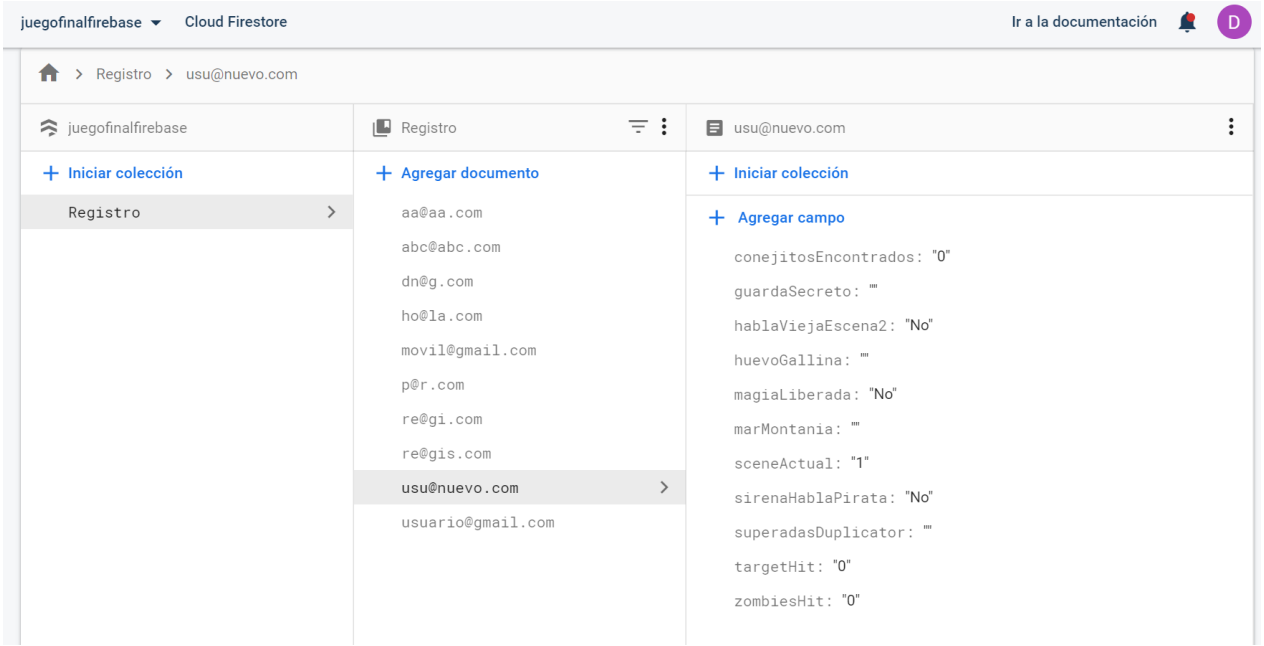
- El usuario debe tener estructura de correo (por ejemplo, ejemplo@gmail.com). La contraseña debe tener al menos 8 caracteres.



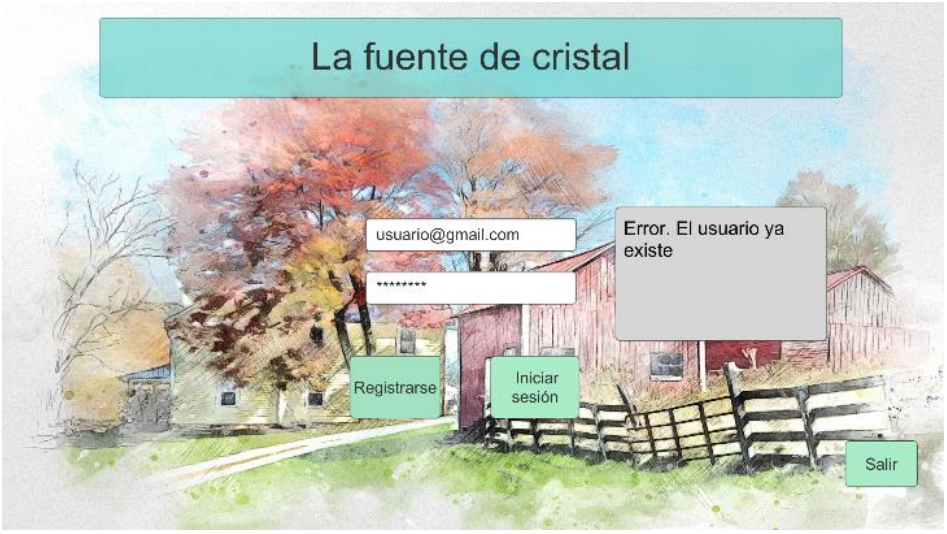
- Un usuario no registrado puede registrarse, pero no iniciar sesión.



- Cuando un usuario se registra, Firebase almacena el usuario y la contraseña, y se crea un documento en Firestore con su nombre.



- Un usuario registrado no puede volver a registrarse, debe iniciar sesión.

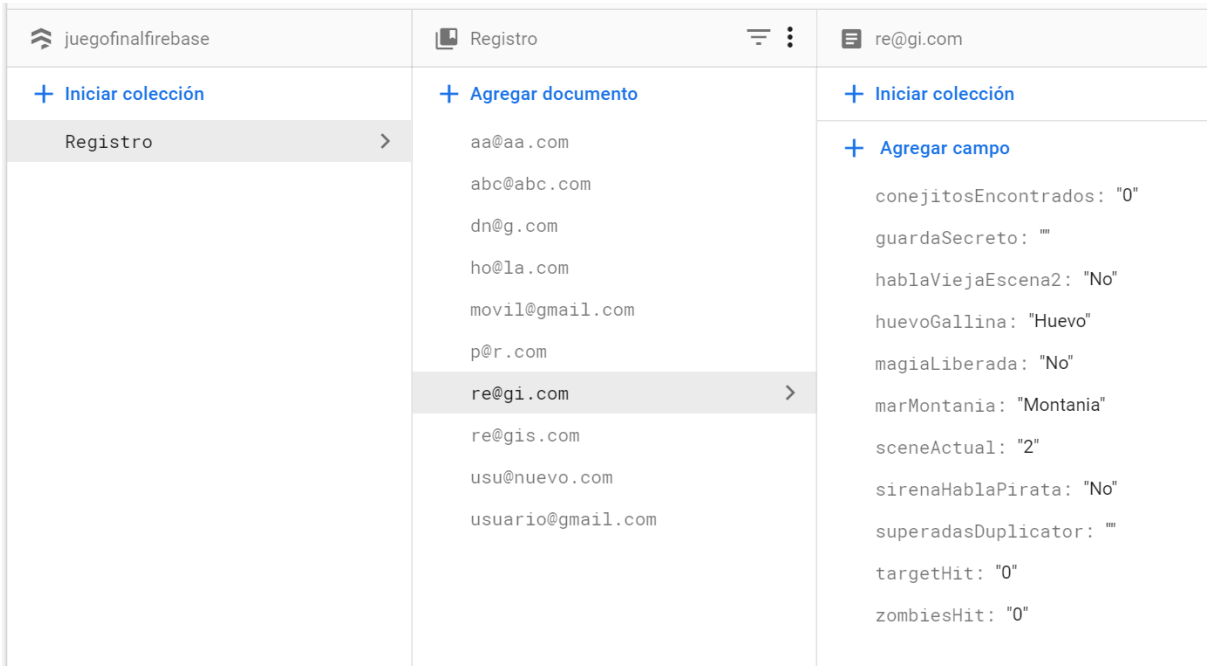


- Salir cierra la aplicación

Pruebas del menú inicial

- Nueva Partida carga la primera escena del juego, independientemente del progreso del jugador.

El usuario re@gi.com tiene el siguiente documento en Firestore



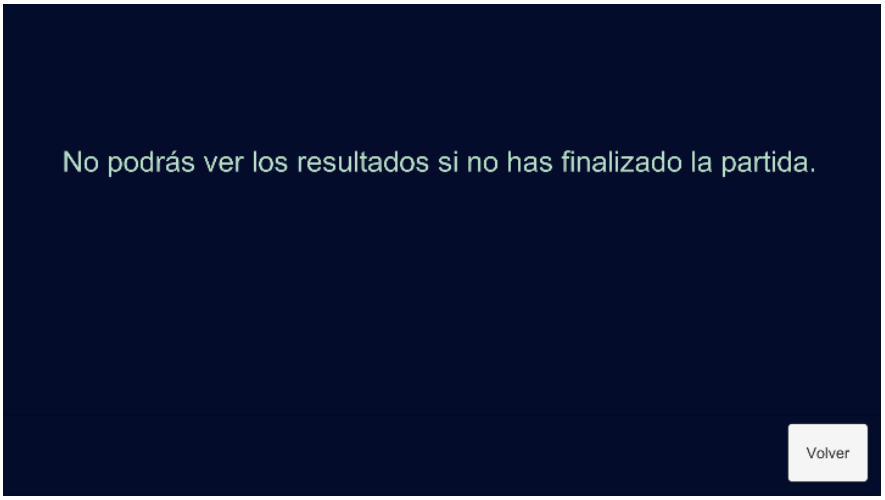
El valor de la variable sceneActual es 2, lo cual indica que se encuentra en la escena 2.
Al pulsar Partida Nueva, se carga la escena 1.

- Continuar carga la escena donde el usuario dejó de jugar por última vez

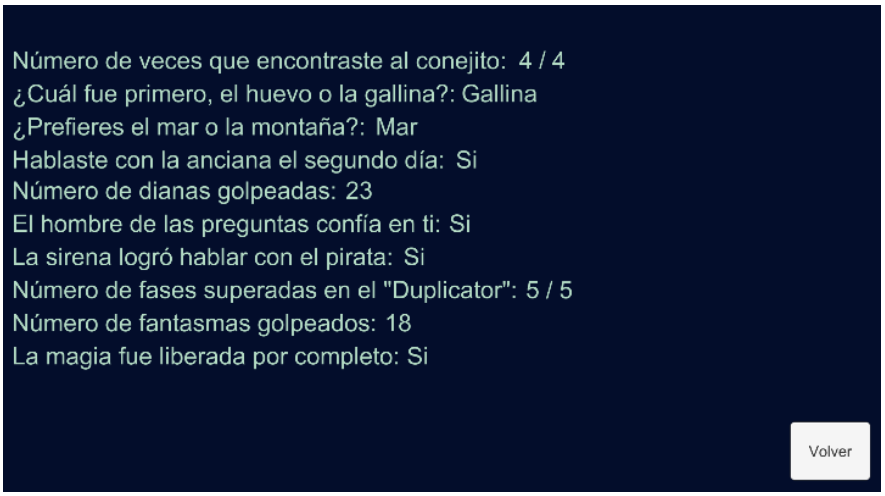
  Si el usuario re@gi.com mencionado anteriormente selecciona Continuar, se carga la escena 2.

- Si un jugador con la partida finalizada pulsa Resultados, verá los datos obtenidos en su partida. Si aún no se ha completado la partida, no podrá ver sus resultados, y le aparecerá un mensaje indicando que es necesario finalizar la partida para ver los resultados.

  Un usuario que no ha finalizado la partida y pulsa Resultados ve la siguiente pantalla:



Un usuario que ha terminado la partida y pulsa resultados obtiene el siguiente resultado, variable según sus logros:



- Cerrar sesión vuelve a la pantalla de login

  Funciona correctamente.

Guardado de datos

- Al finalizar una escena, se actualiza en Firestore el documento del jugador.

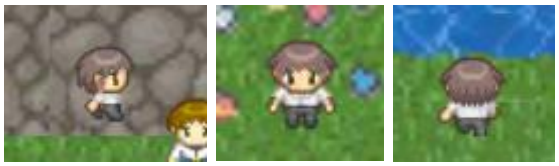
Se empieza una partida nueva. Se supera la primera escena, y en Firestore se almacenan correctamente los datos de las variables que han sido modificadas a lo largo de la escena.

- Solo se puede finalizar una escena tras cumplir ciertos requisitos, como hablar con todos los vecinos y jugar a los minijuegos.

  En cada escena se ha comprobado que los requisitos para finalizar escena son los deseados y que no se puede finalizar la escena antes de cumplirlos.

Controles del personaje

- Deslizar la palanca direccional hace que el personaje se mueva, con la animación correspondiente a la dirección de movimiento



- Pulsar A cerca de un elemento interactuable comienza el dialogo correspondiente

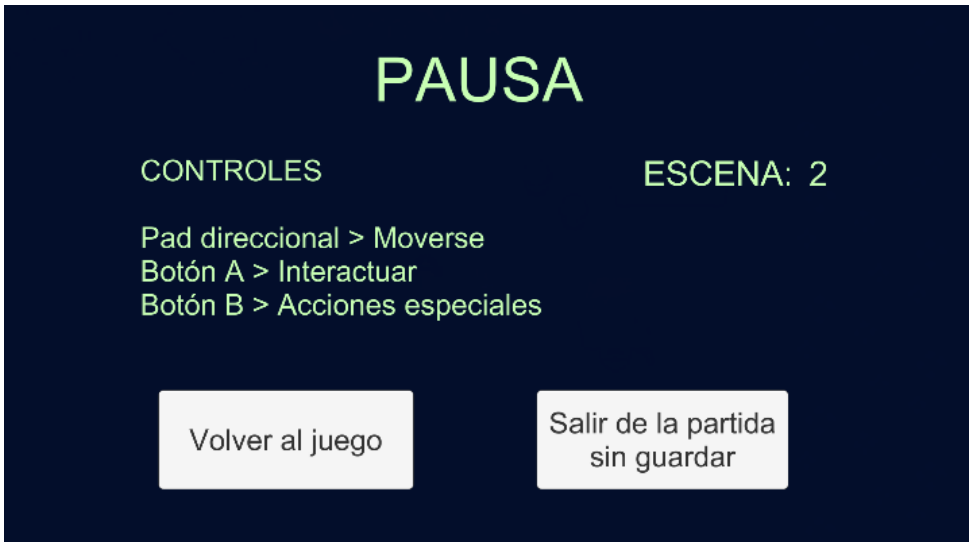
- Durante una conversación, el personaje no puede moverse ni se abre el menú de pausa.

  Funciona correctamente. En una escena del juego hay un duplicado que comparte el código de controles con el jugador,

pero se le ha restringido la función de pausar el juego porque podía hacerlo mientras el jugador esta interactuando.

- Al pulsar pausa, se abre una pantalla y el personaje no puede moverse

  Funciona correctamente. Cuando aparece el menú de pausa, se deshabilitan las opciones de movimiento.



- En el menú de pausa, pulsar Volver al juego cierra la pantalla de pausa y pulsar Salir de la partida sin guardar cierra la escena actual y carga la pantalla de login/registro

  Ambos botones cumplen su función.

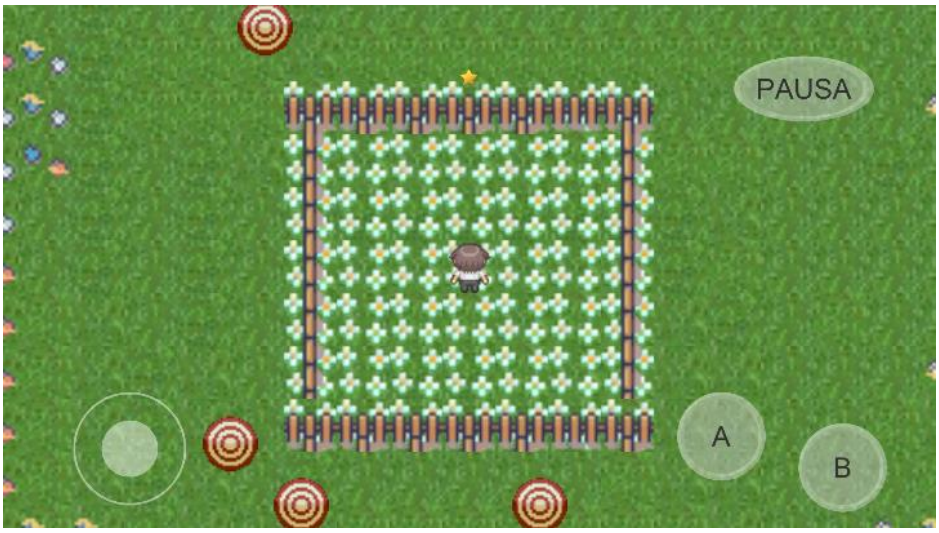
Minijuegos

- Los minijuegos solo se pueden jugar una vez

  Después de hablar con la chica de la tienda en las escenas 2 y 3, el jugador tiene acceso a los minijuegos, a los que solo puede jugar una vez.

- En las fases de disparos, al pulsar el botón B se dispara una estrella en la dirección a la que mira el jugador

  Las estrellas se instancian al pulsar el botón B



- En las fases de disparos, no se permite pausar la partida

  El botón de pausa esta deshabilitado en las fases de disparos, para evitar conflictos con el movimiento de las estrellas y los objetivos.

- En las fases de disparos, si una estrella golpea una diana o un fantasma, se destruyen ambos y se suma un acierto al contador correspondiente

Funciona correctamente. El resultado acumulado se guarda al final de la escena.

- En las fases de disparos, si un fantasma golpea al personaje, desaparece el fantasma

  El fantasma desaparece y no se suma al contador de fantasmas derrotados.

- En el minijuego “Duplicator”, pulsar el botón B hace que se abandone la partida

  Ocurre correctamente. Cuando se abandona el juego, se registra el número de fases superadas, que se guarda al final de la escena. Se ha decidido dar la opción de abandonar este minijuego por si es demasiado complicado para algunos jugadores.

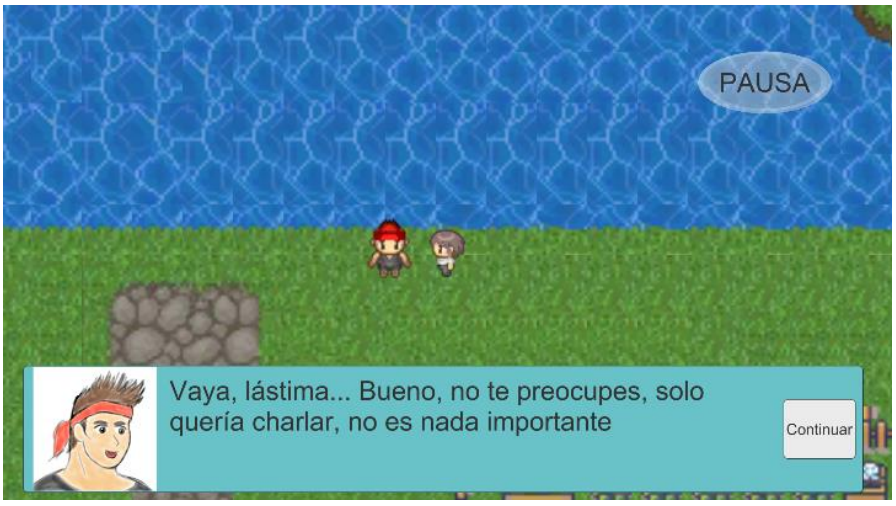
- En el minijuego “Duplicator”, si el personaje y su duplicado pisan dos tocones al mismo tiempo, se suma una fase completada a un contador, y se mueven los tocones, salvo si es la última fase, en la que acaba el minijuego.

  Funciona correctamente

Específicos de escenas

- En la escena 3, hablar con la sirena antes que con el pirata, hace que la sirena se desplace. Si se habla con el pirata antes que con la sirena, esta desaparece durante el resto de la escena.

  Funciona correctamente. En la escena 3 no es obligatorio hablar con la sirena, pero sí con el pirata, de forma que los dos escenarios contemplados están controlados.



- En la escena 4, golpear la fuente con una estrella desplaza la fuente hacia arriba

  Posición de la fuente antes de ser golpeada por una estrella:

Posición de la fuente después de ser golpeada por una estrella:

6.2. Solución a problemas encontrados

A lo largo del desarrollo del proyecto, surgieron algunos problemas que requirieron investigación para ser solucionados. Los más destacados son los siguientes:

- Implementación del texto con decisiones: Supuso un desafío explorar las diferentes alternativas sugeridas para preparar el sistema de texto del juego. Finalmente, me decanté por Ink para escribir el diálogo por su sencillez y su capacidad de gestionar variables.

- Implementación de Firebase: Aunque lo había usado en Android, implementar Firebase correctamente resultó complicado porque los videos que explicaban eran demasiado específicos. Finalmente, la solución fue seguir las instrucciones de la página oficial con criterio.

- Build para Android: En cierto punto del desarrollo, mi proyecto superó el número de métodos que puede tener para generar el gradle, debido a la implementación de Firebase. La solución fue utilizar un gradle personalizado con multiindex.

- Sistema gráfico 2D: Fue un cambio respecto al 3D que vimos en clase. La preparación de Tilesets para poder dibujar los escenarios requirió buscar tutoriales. A lo largo del desarrollo proyecto ha sido importante buscar assets con estilos similares para poder crear un mundo coherente.

7. LANZAMIENTO Y PUESTA EN MARCHA

De cara al lanzamiento del producto, tengo que considerar dos aspectos relevantes: La integración de Firebase y la publicacion en Play Store.

Integración de Firebase

Para gestionar los usuarios y sus datos, es necesario integrar el juego con Firebase. Para ello, se crea un proyecto de Firebase, que se asocia al juego pulsando el haz clic en el ícono de Unity.

Se siguen los pasos indicados y se obtiene un archivo google-services.json, que debe incluirse en la carpeta Assets del juego.

Tras seguir estos pasos, es necesario instalar en Unity los paquetes realcionados con Firebase que se van a utilizar. En mi caso, se instalaron FirebaseAnalytics, FirebaseAuth y FirebaseFirestore.

Finalmente, se agregó el código indicado en la página oficial para asegurar la conexión entre el juego y Firebase.

```
using Firebase;
using Firebase.Analytics;
using UnityEngine;

Script de Unity (1 referencia de recurso) | 0 referencias
public class FirebaseInit : MonoBehaviour
{
    public Firebase.FirebaseApp app;
    // Start is called before the first frame update
    Mensaje de Unity | 0 referencias
    void Start()
    {
        Firebase.FirebaseApp.CheckAndFixDependenciesAsync().ContinueWith(task => {
        var dependencyStatus = task.Result;
        if (dependencyStatus == Firebase.DependencyStatus.Available) {
            // Create and hold a reference to your FirebaseApp,
            // where app is a Firebase.FirebaseApp property of your application class.
            app = Firebase.FirebaseApp.DefaultInstance;
            // Set a flag here to indicate whether Firebase is ready to use by your app.
        } else {
            //UnityEngine.Debug.LogError(System.String.Format(
            //"Could not resolve all Firebase dependencies: {0}", dependencyStatus));
            // Firebase Unity SDK is not safe to use here.
        }
        });
    }

    // Update is called once per frame
    Mensaje de Unity | 0 referencias
    void Update()
    {

    }
}
```

Publicación en Play Store

Para poder publicar el juego en Play Store, se deben seguir una serie de pasos:

- Crear una cuenta de desarrollador: Se proporciona la información solicitada y se pagan 25 euros como cuota de registro. Los usuarios verán el nombre de desarrollador indicado. En caso de incluir apps con pagos, también se debe configurar un perfil de pagos.
- Crear la app: Se selecciona crear aplicación y se indica el idioma y el nombre que se verá en la tienda.

- Completar la ficha del producto: Incluye el nombre de la app, descripción breve, descripción completa, idioma, imágenes, género del juego, datos de contacto y política de privacidad.
- Subir la APK: Se sube una versión del código fuente. Se puede elegir entre lanzar una prueba interna, una versión cerrada para testear, una versión abierta o una versión de producción para todos los usuarios.
- Clasificación del contenido: Debe completarse un documento sobre clasificación del contenido, que incluye información como la edad recomendada. Si no se cumplimenta la aplicación será eliminada de la tienda.
- Establecer precio y países de distribución: Una aplicación de pago puede cambiar a gratuita, pero no al revés.
- Enviar la app a revisión: Se revisa la app. Puede tardar hasta dos días dias. Si se supera la revisión, la app se sube a Play Store.

8. MANUAL DE USO

El manual de uso del juego se encuentra en el siguiente enlace:

https://github.com/DavidAlvarezdeLuna/JuegoFinCicloDavidAlvarez/blob/main/manual_fuente_cristal.pdf

9. VALORACIÓN Y CONCLUSIONES

Este proyecto ha supuesto una experiencia muy interesante para mí. La primera razón es que ver como la idea que tuve al principio ha ido cogiendo forma, enfrentándome a los problemas que han ido surgiendo, hace que ver el resultado final me haga sentir que el esfuerzo ha valido la pena. Por otro lado, tenía curiosidad sobre lo que implica crear un videojuego, pero suponía mucho tiempo y esfuerzo, y este proyecto ha supuesto mi oportunidad para crear uno seriamente.

He podido aplicar los conocimientos adquiridos a lo largo del ciclo, especialmente los relativos a Unity. También los conocimientos generales que he aprendido me han ayudado a saber proceder frente a errores y a comprender las nuevas herramientas que he utilizado en el desarrollo del proyecto.

Sobre la documentación, no me había parado a pensar en la cantidad de información que puede llegar a tener. Me ha venido bien la estructuración en entregas semanales, ya que de esta forma me he obligado a no abandonarla a favor del desarrollo del proyecto.

Finalmente, la mayor complicación que ha tenido el proyecto en general es coincidir con el desarrollo de las prácticas de empresa. El proyecto se ha desarrollado en días libres o después de la jornada de trabajo, dejándome menos tiempo de desarrollo del que me hubiera gustado para perfilar detalles del juego, aunque estoy satisfecho con el resultado actual.

He aprendido mucho realizando el proyecto, y me he sentido motivado a lo largo del desarrollo debido a que pude elegir la temática que deseara. No descarto realizar en el futuro algunas mejoras que no me ha dado tiempo a implementar.

10. BIBLIOGRAFÍA

https://assetstore.unity.com/

https://itch.io/

https://opengameart.org/content/lpc-collection

https://www.toulouselautrec.edu.pe/blogs/herramientas-crear-videojuego

https://firebase.google.com/docs/unity/setup?hl=es-419

https://klaudiabronowicka.com/blog/making-a-visual-novel-with-unity-1/

https://videlais.com/2019/07/08/unity-ink-part-1-importing-and-testing/

https://www.arsys.es/blog/phpmyadmin

https://es.wikipedia.org/wiki/Videojuego_de_aventura

https://www.eleconomista.es/diccionario-de-economia/analisis-dafo

https://www.youtube.com/watch?v=eTQ7-Y9w3Yk

https://elcomercio.pe/respuestas/que-significa-el-color-rojo-amarillo-blanco-rosa-azul-y-verde-psicologia-de-los-colores-tdex-revtli-noticia/

https://www.webyempresas.com/ejemplos-de-diagramas-de-casos-de-uso/

https://www.yeeply.com/blog/guia-subir-app-google-play-store/

https://forum.unity.com/threads/enable-multidex-support-for-unity-2020.990440/

https://www.noveltech.dev/import-tilemap-unity/

11. ANEXOS

ANEXO I: GAME DESIGN DOCUMENT

1. DISEÑO CONCEPTUAL

1.1. Título

Mi juego se titula “La fuente de cristal”. Pretende comunicar al jugador que el juego se desarrolla en un ambiente de fantasía, al estilo de juegos clásicos de aventura.

1.2. Concepto

Un joven periodista está investigando una extraña explosión de gran calibre que no parece haber causado estragos. De camino al lugar del suceso, escucha una voz misteriosa y se ve obligado a deambular por un bosque hasta hallar un pueblo escondido en el que pasará unos días, conociendo a sus habitantes e intentando descifrar la causa real del suceso.

1.3. Género

El juego pertenece al género aventura, ya que consiste en explorar un pueblo durante varios días buscando resolver un misterio. Además, la interacción con los habitantes del pueblo y otros elementos pueden causar cambios en el desarrollo de la historia.

1.4. Público objetivo

El público objetivo incluye a los aficionados al estilo gráfico 2D y jugadores que buscan una experiencia de juego alejada de la acción.

1.5. Aspectos de interés

El juego ofrece una entretenida experiencia que combina los gráficos 2D que gustan a muchos a día de hoy, y un sistema de decisiones con el que forjar tu propio camino, el cual podrás compartir con otros jugadores.

2. DISEÑO DEL PRODUCTO

2.1. Experiencia de juego

El jugador controla a un joven periodista que deberá comunicarse con los habitantes del pueblo donde ha ido a parar. Se pretende que el jugador tenga una experiencia tranquila que mantenga su interés gracias a su ligera historia y personajes. Para hacer la experiencia más dinámica, en algunas fases del juego se realizarán pequeños minijuegos en los que no se exige gran habilidad, pero los resultados obtenidos serán tenidos en cuenta.

2.2. Estilo visual y audio

El juego cuenta con estética 2D que es agradable de ver y es valorada por muchos jugadores por su encanto clásico. La música utilizada a lo largo del juego es tranquila y pretende evocar la sensación de tranquilidad que transmite la aldea donde transcurre el juego.

2.3. Mundo donde se desarrolla el juego

El mundo del juego está basado en la realidad, donde no hay elementos de fantasía. Sin embargo, un punto clave de la historia del juego es cambiar esta creencia, a partir de la llegada del protagonista a un pueblecito misterioso donde no todo es lo que parece.

2.4. Plataformas, tecnologías y alcance

El juego se ha desarrollado en Unity y será lanzado para dispositivos Android. Ha sido elaborado durante tres meses por una persona. La duración estimada de una primera partida ronda entre 30 minutos y 1 hora, siendo más próxima a 30 minutos en partidas sucesivas.

3. DISEÑO DE SISTEMAS DEL JUEGO

3.1. Flujo de juego

El jugador comienza la partida. A lo largo del juego, tendrá que tomar decisiones. Estas decisiones serán almacenadas en variables. Cuando el jugador finaliza una escena del juego, se volcarán todas las variables a una base de datos. Esto permite que el jugador pueda guardar y cargar la partida en otro momento. Cuando se supera la última escena, el jugador puede consultar las decisiones que ha tomado a lo largo de la partida.

3.2. Objetivos y progresión

El jugador comienza introduciendo su usuario en la pantalla inicial. Tras eso, puede empezar una partida nueva o continuar en caso de que tuviera alguna iniciada. En la partida, el objetivo es hablar con todos los habitantes del pueblo para descubrir sus propias historias, y ayudarles con distintas peticiones que te puedan pedir. También se exigirá jugar a algún minijuego de vez en cuando. Una vez se ha hablado con todos los vecinos requeridos en la escena, el jugador vuelve a la posada donde se hospeda, y se pasa a la siguiente escena.

3.3. Sistemas de juego

- Implementación de los gráficos 2D: Para diseñar un mundo cohesivo para el desarrollo del juego, se han empleado assets. Unity facilita la creación de los mapas con las herramientas Sprite Editor y Tile Palette.
- Sistema de diálogos con decisiones: Para implementar el sistema de diálogos en el juego se emplea el lenguaje Ink, que se edita en el programa Inky, generando archivos JSON que lee Unity para generar los textos escritos. Ink permite generar decisiones y gestionar variables según las decisiones tomadas, las cuales se pueden pasar a Unity.

ANEXO II: REQUISITOS DEL CLIENTE

En la siguiente tabla se recoge la información de las reuniones que se han tenido con el cliente a lo largo del desarrollo de la aplicación:

Fecha	Asunto	Reunidos	Funcionalidades a implementar	Plazos
14/03/2022	Reunión inicial	Cliente, desarrollador	Estilo gráfico 2D Preparación de assets. Preparación de sistema de diálogo. Preparación de audio.	Dos semanas
28/03/2022	Revisión de assets	Cliente, desarrollador	Creación de un pueblo según las instrucciones indicadas Diseño de personajes Preparar pantalla de menú inicial Primera escena	Tres semanas
18/04/2022	Punto de control	Cliente, desarrollador	Realización de correcciones	Una semana
25/04/2022	Siguientes pasos	Cliente, desarrollador	Creación de las escenas dos, tres y cuatro según el guión Sistema de login y registro Sistema de guardado	Cuatro semanas
23/05/2022	Muestra del progreso realizado hasta el momento	Cliente, desarrollador	Revisión de sugerencias aportadas por el cliente Escena cinco según el guión Mejoras en la presentación	Dos semanas

			Preparación de la documentación	
06/06/2022	Muestra del juego completo y de la documentación	Cliente, desarrollador	Retoques finales al juego Completar la documentación según las indicaciones aportadas Realización extensa de pruebas	Dos semanas
20/06/2022	Presentación del producto final, demostración completa del juego	Cliente, desarrollador	Entrega del proyecto completo al cliente	

ANEXO III: FORMULARIO DE INCIDENCIAS

En la página web del juego se incluye un formulario donde los jugadores pueden hacer llegar los problemas que hayan podido encontrar utilizando la aplicación. Los apartados que se les pide rellenar son:

- Nombre del usuario (para referirse a él en la respuesta)
- Usuario del juego (correo)
- Dispositivo móvil
- Tipo de incidencia
- Explicación de la incidencia

En la siguiente imagen se muestra una sugerencia de presentación del formulario de incidencias.