



# HOME PRISON

Proyecto Desarrollo de Aplicaciones Multiplataforma

CIFP Virgen de Gracia

Curso 2021 - 2022

Daniel Álvarez de Luna Quintana

# ÍNDICE

1. PRESENTACIÓN DEL PROYECTO .....	3
1.1. Explicación resumida.....	3
1.2. Estudio de mercado .....	3
1.3. Valor del producto .....	5
2. PLANIFICACIÓN DE TAREAS Y ESTIMACIÓN DE COSTES .....	6
2.1. Planificación y organización de tareas.....	6
2.1.1. Planificación.....	6
2.1.2. Flujo de trabajo .....	7
2.1.3. Wiki del proyecto.....	8
2.2. Estimación de costes y recursos .....	8
2.2.1. Hardware .....	8
2.2.2. Software.....	9
2.2.3. Humano .....	9
2.3. Herramientas usadas .....	10
2.4. Gestión de riesgos .....	10
3. ANÁLISIS DE LA SOLUCIÓN .....	12
3.1. Requisitos .....	12
3.2. Análisis de escenarios.....	14
4. DISEÑO DE LA SOLUCIÓN.....	15
4.1. Diseño de la interfaz de usuario y prototipos .....	15
4.2. Diagrama de clases.....	21
4.3. Diseño de la persistencia de la información.....	23
4.4. Arquitectura del sistema .....	25
5. IMPLEMENTACIÓN DE LA SOLUCIÓN.....	26
5.1. Análisis tecnológico.....	26
5.2. Elementos a implementar .....	29
6. TESTEO Y PRUEBAS DE LA SOLUCIÓN .....	46
6.1. Plan de pruebas .....	46
6.2. Solución a problemas encontrados.....	58
7. LANZAMIENTO Y PUESTA EN MARCHA .....	59
8. MANUAL DE USO.....	63
9. VALORACIÓN Y CONCLUSIONES .....	63
10. WEBGRAFÍA.....	65

# 1. PRESENTACIÓN DEL PROYECTO

## 1.1. Explicación resumida

El proyecto consiste en la creación un juego de temática Escape Room y puzles, donde el objetivo es investigar una serie de salas cerradas para interaccionar sus elementos en busca de pistas que te permitan escapar. Dichos niveles serán escenas diferentes y estarán ambientados en diversas habitaciones de una casa y se jugarán en un orden determinado y conectados por un argumento ligero.

Este juego será desarrollado en Unity, contará con una estética 2D y se apoyará en descripciones de texto e imágenes para dar mayor detalle a ciertos objetos cuando estos sean inspeccionados.

El juego contará con una base de datos interna para que puedan existir soluciones distintas a ciertos planteamientos en diferentes partidas. Por ejemplo, se distribuirán datos sobre un elemento desconocido en diferentes puntos de la sala, que deben ser recogidos para realizar una búsqueda de la que se obtenga un único resultado que llevará a la solución del puzle.

## 1.2. Estudio de mercado

Los juegos de tipo aventura son un sector abundante en el mercado de los videojuegos y que engloba una gran variedad de títulos. Estos se definen por no priorizar los reflejos del jugador y presentar elementos como la investigación, exploración, deducción y toma de decisiones. A menudo presentan al jugador puzles para resolver.

Suelen ser acompañados por una narrativa que une los retos que presenta el juego y que puede ser de cualquier tipo, como son fantasía, ciencia ficción, suspense, misterio, miedo, o incluso comedia.

Dentro de estos juegos de aventura, se encuentra el subsector de **aventura de rompecabezas**, que son los que ponen mayor énfasis en explorar el entorno para lograr avanzar. Aquí se incluyen los **juegos de Escape Room**, que es el género concreto de nuestro juego.

El objetivo de un juego de Escape Room se deduce por su nombre, “Escapar de la habitación”. Su presentación habitual consiste en un personaje controlable en un entorno cerrado que debe investigar con el objetivo de salir, generalmente encontrando una llave o un código para desbloquear una puerta cerrada. Para encontrarla deberá interaccionar con elementos, encontrar pistas y resolver diversos problemas. También pueden presentar un formato point-and-click, en las que el jugador navega entre pantallas que representan diferentes zonas de la sala e interactúa con sus elementos mediante clicks, es el estilo predominante sobre todo en los inicios de este tipo de juegos.

Sobre los años 70, la popularidad de los videojuegos iba en aumento. Las tecnologías avanzaban y permitían a los desarrolladores utilizar elementos visuales y abandonar los juegos basados en comandos de texto para ofrecer experiencias de juego novedosas. El primer juego de Escape Room fue publicado en 1983, llamado Planet Mephius.

El género fue ganando popularidad, más con el aumento del acceso a internet por parte de la población. La idea de un personaje atrapado en una habitación que es un puzle gigante que esconde la solución para escapar se presentó atractiva hasta el punto de crearse salas de Escape Room reales en todo el mundo.

El impulso generado por contenido diferente a los videojuegos, como películas o las mencionadas salas de juego reales, contribuyen a que estos productos se mantengan relevantes en el mercado para los aficionados de los puzles, con opciones para todo tipo de plataformas, incluyendo móviles.

## **JUEGOS SIMILARES EN EL SECTOR**

999: 9 Hours, 9 Persons, 9 Doors (2009):

En este juego, el jugador controla a un joven que un día es secuestrado y obligado a participar en un macabro juego en el que debe descubrir la manera de escapar de varias habitaciones y descubrir el misterio que le rodea, con normas que ponen en peligro su vida y las de los otros 8 participantes. El videojuego une partes de tipo Escape Room con escenas de tipo novela visual para ofrecer una experiencia intensa con varios finales posibles.

The Witness (2016):

El juego trata de un hombre atrapado solo en una isla. Para poder avanzar, debe ir resolviendo puzles que dan acceso a otros con la misma temática pero sucesivamente más difíciles.

Secret Files: Tunguska (2006):

En este juego controlas a una chica que investiga la misteriosa desaparición de su padre, un científico sumido en una importante investigación. Para ello, deberá ir hablando con gente y encontrando y combinando objetos para resolver los retos que se le presenten.

## **ANÁLISIS DAFO**

Análisis interno:

- Fortalezas: El juego combina la jugabilidad de Escape Room con el siempre añorado estilo 2D predominante en juegos clásicos.
- Debilidades: Puzles sencillos para jugadores avanzados y corta duración.

Análisis externo:

- Oportunidades: El crecimiento de la popularidad de las Escape Room reales hace más visible el género, haciendo más atractivos los videojuegos de dicha temática.
- Amenazas: Mucha competencia de juegos con alto presupuesto.

## **CANALES DE DISTRIBUCIÓN**

En el caso de que el proyecto prospere y se considere válido para llevarlo al público, el canal de distribución elegido será la plataforma de videojuegos Steam.

### **1.3. Valor del producto**

Mi producto busca ofrecer al jugador un juego de tipo Escape Room, con una estética 2D sencilla y a la vez reminiscente de videojuegos clásicos que impulsaron el sector en general. Se busca una experiencia de juego ligera y agradable, que invite

al jugador a resolver los problemas planteados. El juego presentará al jugador soluciones diferentes a los puzzles que se generarán al empezar cada sala, de manera que no se pueda obtener la solución final directamente de fuentes externas.

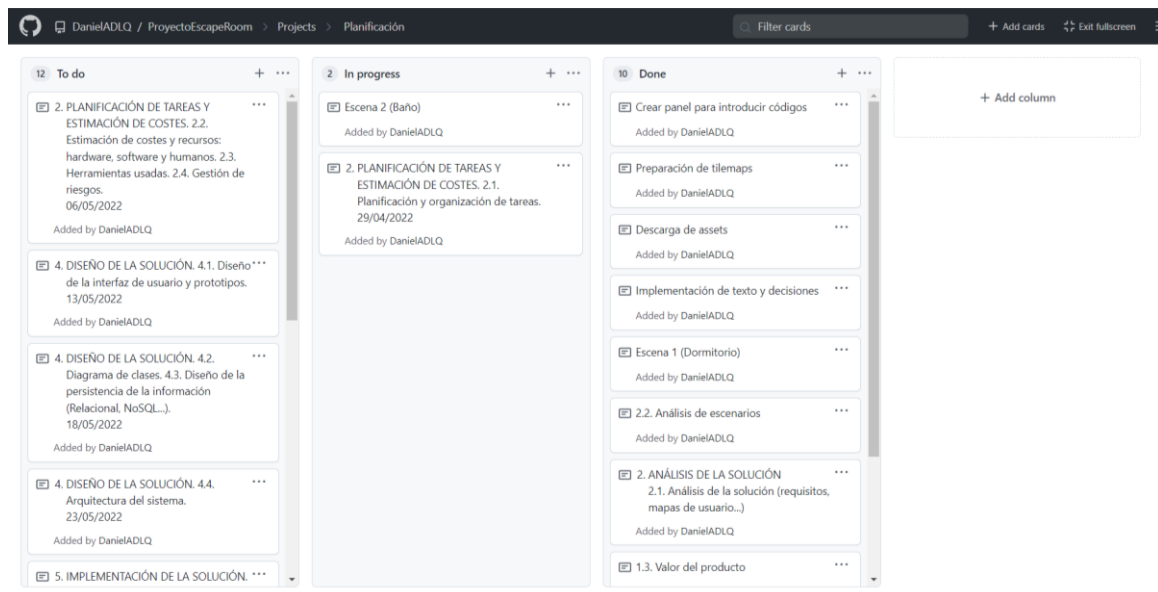
## 2. PLANIFICACIÓN DE TAREAS Y ESTIMACIÓN DE COSTES

### 2.1. Planificación y organización de tareas

#### 2.1.1. Planificación

Para facilitar el seguimiento y progreso del proyecto, es muy importante realizar una planificación de tareas y establecer tiempos límite y estimaciones de duración de cada tarea. Hay que tratar de hacer estimaciones lo más precisas posible para evitar retrasos en las entregas.

Para la planificación de mi proyecto he utilizado la herramienta Projects de Github en la que se crea un kanban con tareas por hacer, en progreso y completadas.



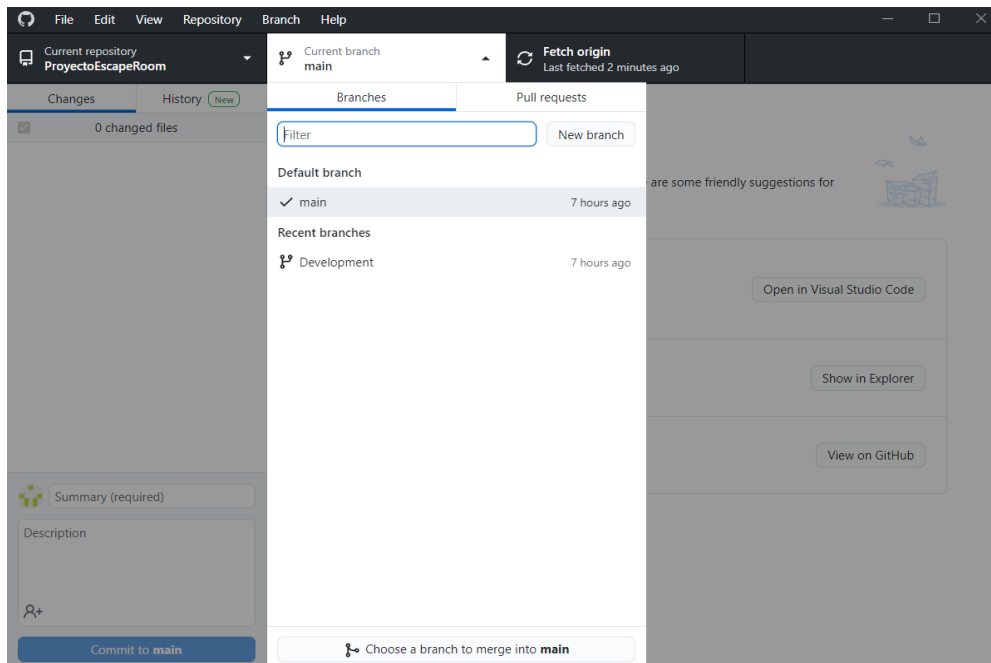
A continuación se detallan los hitos a realizar con sus fechas límite:

- 03/04/2022 → Explicación resumida del proyecto.

- 10/04/2022 → Análisis de la solución.
- 15/04/2022 → Estudio de mercado y valor del producto.
- 22/04/2022 → Análisis de escenarios.
- 29/04/2022 → Planificación y organización de tareas.
- 06/05/2022 → Estimación de costes y recursos: hardware, software y humanos, herramientas usadas y gestión de riesgos.
- 13/05/2022 → Diseño de la interfaz de usuario y prototipos.
- 18/05/2022 → Diagrama de clases y diseño de la persistencia de la información.
- 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

El flujo de trabajo del proyecto se basa en la existencia de dos ramas en el repositorio. Una rama main, que es la principal y tendrá funcionalidades ya probadas. Y una rama development, que es en la que se desarrollan nuevas funcionalidades. Una vez comprobado que funciona correctamente, se pasan esas modificaciones a la rama main.



### 2.1.3. Wiki del proyecto

Para documentar el proyecto, se utiliza la herramienta de wiki de Github, que permite añadir al repositorio información referente al proyecto. Este será el lugar donde se almacenará la documentación ya existente y la que queda por hacer.

## 2.2. Estimación de costes y recursos

La estimación de recursos consiste en realizar una aproximación de las necesidades monetarias para llevar a cabo el proyecto. Una estimación precisa es complicada sobre todo en las primeras fases.

Las estimaciones se refieren tanto a elementos físicos (como hardware, utensilios o instalaciones), software (Programas, licencias...) y costes humanos. También se deben tener en cuenta posibles contingencias.

### 2.2.1. Hardware

Para el desarrollo del proyecto los recursos utilizados se limitan a mi equipo personal. (Portátil con Windows y móvil Android).



Adicionalmente se usará una memoria externa para almacenar copias del proyecto en físico. Dada la escala del proyecto, 32 GB serán suficiente espacio. (12 euros).

### 2.2.2. Software

Para desarrollar el proyecto se utilizarán los siguientes recursos de software:

- Unity: Plataforma de desarrollo de videojuegos multiplataforma. (Gratuito)
- Visual Studio Community: Para escribir el código del proyecto. (Gratuito)
- GitHub: Plataforma para control de versiones en la nube. Contiene recursos útiles como Wiki y organización del proyecto con modelo kanban (Gratuito)
- GitHub Desktop: Programa para gestionar Git y clonar el repositorio en local. (Gratuito)
- Ibis Paint: Programa de dispositivos móviles para dibujo y diseño. (Gratuito)
- Paquete Office: Se emplean herramientas para crear documentación. (Licencia disponible inicialmente en el equipo de trabajo).
- Google Docs: Para revisión de la documentación en línea.
- Assets de Unity: Elementos del juego, por ejemplo personajes, objetos, música o recursos de funcionamiento. Se recurre a páginas web donde se distribuyen assets. (En principio todo será gratuito, sin embargo es posible realizar algún desembolso para adquirir algún asset que se estime importante y del que no exista una variante gratuita, ni pueda elaborarlo personalmente).
- Steam: Para distribuir el programa se utilizará la plataforma de videojuegos Steam. (100€ para subir un juego, reembolsable si se alcanza 1000€ de beneficio).

### 2.2.3. Humano

Solo habrá un desarrollador, que realizará el código, búsqueda de assets, documentación y las demás tareas.

La duración estimada del desarrollo del juego son 300 horas.

La duración estimada para elaborar la documentación son 60 horas.

Estimando un coste de 12€/hora por la realización del programa y documentación por parte de un único desarrollador, se estiman 4320€.

## 2.3. Herramientas usadas

Desarrollo del juego: se usará la plataforma para juegos multiplataforma Unity.

Creación de código C# de los script: se utilizará Visual Studio Community.

Creación y edición de imágenes: se emplea Paint e Ibis Paint (Aplicación de Android).

Para elaborar la documentación se usará Microsoft Office Word y Google Docs.

Control de versiones: Se usarán Github y Github Desktop.

## 2.4. Gestión de riesgos

A continuación se detallan los riesgos relacionados con el proyecto y las acciones que se realizarán en caso de que ocurran.

Planificación	
Problema	Tratamiento
El tiempo total estimado es insuficiente para finalizar la aplicación.	Priorizar la funcionalidad frente al volumen de contenido. Se preferirá tener menos niveles, pero completos de manera que se pueda entregar un proyecto en estado jugable en su totalidad.
Retraso en una tarea que produce retrasos en cascada.	Evaluar la cantidad de tiempo perdido y posponer partes no esenciales para el funcionamiento.

<b>Organización y gestión</b>	
<b>Problema</b>	<b>Tratamiento</b>
Un solo empleado para realizar todas las tareas relativas al proyecto, dificultando su desempeño.	Se dará prioridad a la funcionalidad básica frente a elementos como la consistencia de los gráficos o el número de niveles.

<b>Ambiente/Infraestructura de desarrollo</b>	
<b>Problema</b>	<b>Tratamiento</b>
Una herramienta no funciona como se esperaba.	Si no se encuentra una solución breve, se optará por buscar una herramienta alternativa.
Curva de aprendizaje elevada para el uso de una herramienta.	Se empleará tiempo para aprender el programa desde la base. Se pospondrán elementos menos importantes del proyecto.
Introducir una funcionalidad es más difícil de lo esperado.	Se evaluará la importancia de dicha funcionalidad. Si se estima esencial, se le dará alta prioridad a costa de otras tareas. En caso contrario, se pospondrá su implementación.

<b>Producto</b>	
<b>Problema</b>	<b>Tratamiento</b>
Tiempo insuficiente para realizar las pruebas pertinentes en partes más complejas del juego.	Se realizarán las pruebas más importantes para lograr que el juego pueda progresar.
Calidad inferior a la requerida.	Se priorizarán los elementos fundamentales para tener una aplicación con funcionalidad completa.

### 3. ANÁLISIS DE LA SOLUCIÓN

#### 3.1. Requisitos

##### Requisitos funcionales

RF1 → El juego tendrá una pantalla inicial que se mostrará al iniciar la aplicación. No requerirá login.

RF2 → El jugador podrá ver los créditos y agradecimientos del juego.

RF3 → El jugador podrá empezar una partida nueva.

RF4 → El jugador podrá continuar una partida guardada anteriormente.

RF5 → El jugador podrá guardar los datos de su partida para poder retomarla posteriormente (Se relaciona con el RF4).

RF6 → El jugador controlará un personaje.

RF7 → El juego contará con niveles que consisten en escapar de un espacio cerrado.

RF8 → Una partida tendrá varios niveles que se presentarán ordenados y de forma sucesiva. Hasta que no termine el nivel 1 no se podrá acceder al nivel 2.

RF9 → El juego tendrá una estructura lineal, por lo que no se podrá volver a un nivel anterior una vez superado en esa partida.

RF10 → El personaje puede moverse con las flechas de dirección.

RF11 → El personaje podrá interactuar con su entorno para realizar acciones como coger un objeto, moverlo u obtener una descripción del mismo.

RF12 → El jugador podrá ver una imagen detallada de algunos objetos con los que interaccione.

RF13 → El jugador podrá decidir entre distintas opciones que se le presenten durante la partida.

RF14 → El jugador podrá introducir claves secretas en un panel y comprobar si ha acertado.

RF15 → El sistema podrá preparar soluciones variables para algunos de los puzles, tanto aleatorias como almacenadas entre varias opciones.

RF16 → El juego debe registrar el tiempo que el jugador ha tardado en completar cada puzle.

RF17 → El usuario podrá ingresar un nombre identificativo de 10 letras como máximo al comenzar una partida. Se permitirá que se repita en distintas partidas.

RF18 → El jugador podrá ver una lista de partidas completadas, con nombre identificativo y tiempo de cada puzle. (Relacionado con RF16)

### Requisitos no funcionales

RNF1 → El juego se desarrollará en Unity con lenguaje C#.

RNF2 → Para seleccionar la solución de algunos puzles variables, se usará una base de datos local. Para este fin se utilizará SQLite.

RNF3 → Algunos puzles requerirán al jugador realizar consultas en la base de datos desde el juego.

### Requisitos de información

RI1 → Los puzles con soluciones variables entre partidas se seleccionarán aleatoriamente al comenzar el nivel. Tendrán información en la base de datos:

- ID (numérico)
- Solución
- Datos que se usarán como pistas para esa solución.

RI2 → Datos de partida:

- ID (numérico)
- Nombre
- Tiempo de cada puzle
- Tiempo total
- Escena Actual

RI3 → Datos de nivel:

- Variables de estado de la partida

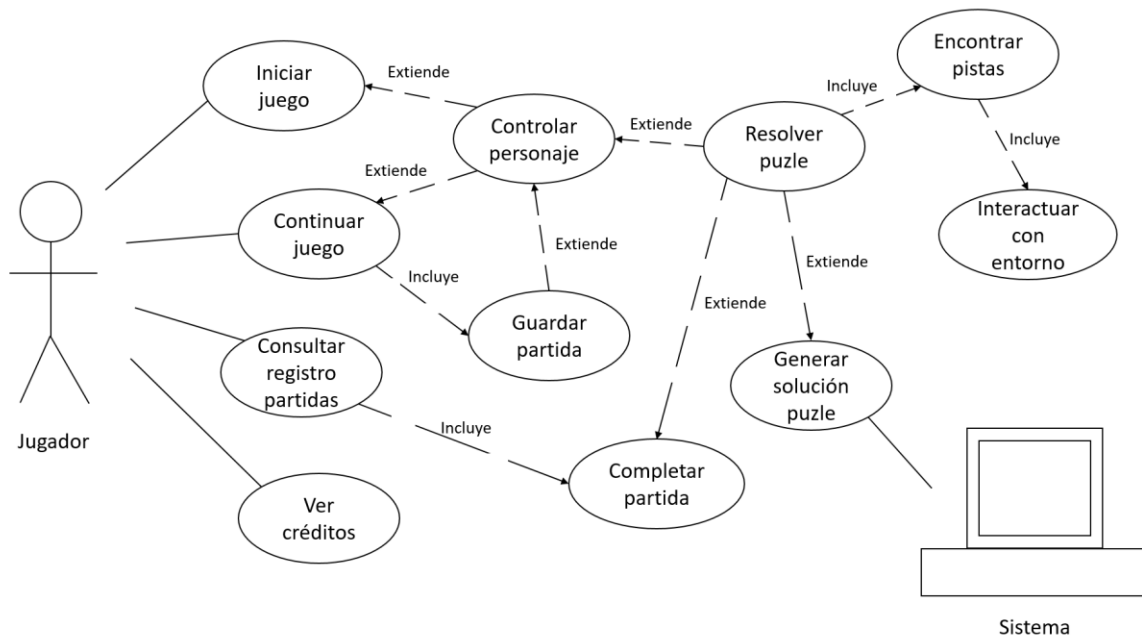
## 3.2. Análisis de escenarios

A continuación se definen los actores y casos de uso del juego. Los actores son las personas o elementos que interaccionan en el programa. El diagrama de casos de uso sirve para identificar las distintas acciones que realizan los actores y la relación que existe entre ellas:

ACTORES:

- Jugador: Persona que realiza acciones en el juego.
- Sistema: Parte del juego que determina ciertos elementos variables del juego.

## CASOS DE USO:



## 4. DISEÑO DE LA SOLUCIÓN

### 4.1. Diseño de la interfaz de usuario y prototipos

La interfaz de usuario es el medio por el que el usuario puede interactuar con la aplicación. Es importante que sea sencilla e intuitiva para que el jugador pueda centrarse en disfrutar el juego sin añadir complicaciones ni frustración por el uso.

### ESTÉTICA

El juego va a utilizar una estética clásica 2D. Para lograrlo, se requiere buscar assets pixelados para representar las escenas que además deben ser de un estilo uniforme.

### COLORES

La paleta de colores elegida para la aplicación consiste en los siguientes:

- Azul celeste (#52BEDE):



El azul en sus tonalidades claras evoca sensaciones cómo:

- Serenidad
- Concentración

Ambas son características importantes para un juego basado en resolver puzles.

- Naranja (#D4A337)



El color naranja en tonalidades claras se relaciona con sensaciones como:

- Creatividad
- Confianza
- Independencia

Estas son cualidades que definen a nuestra protagonista, que debe hacer uso de su ingenio para escapar de salas cerradas sin ayuda.

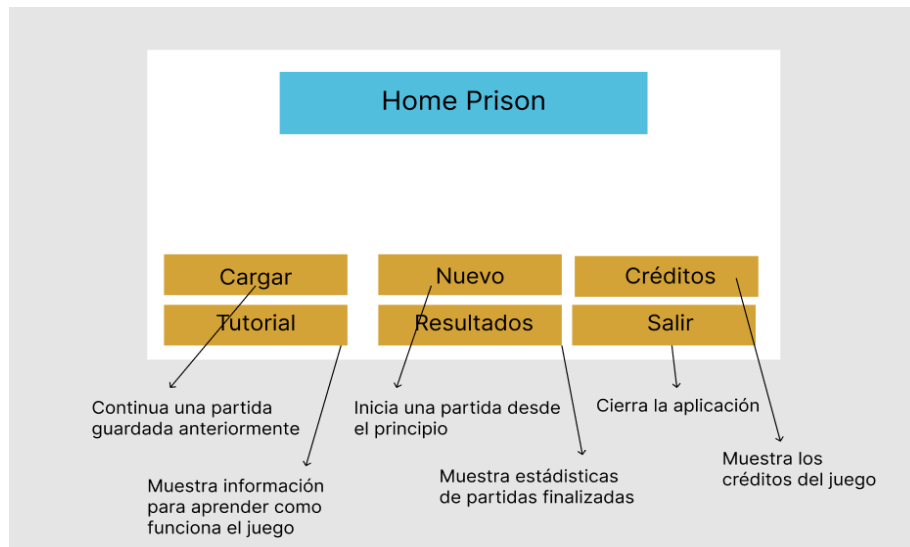
En su vertiente negativa, este color puede representar lo artificial, cualidad que también responda a la temática del juego ya que la protagonista se encuentra de repente atrapada en su propia casa.



# PROTOTIPADO

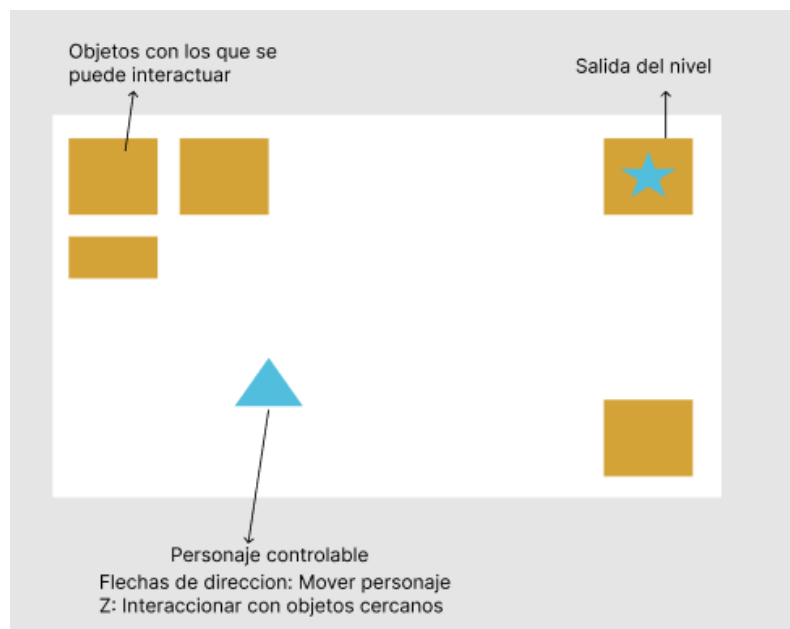
## PANTALLA DE INICIO

Es la pantalla que se muestra al iniciar la aplicación y desde la que se accede a las demás funciones.



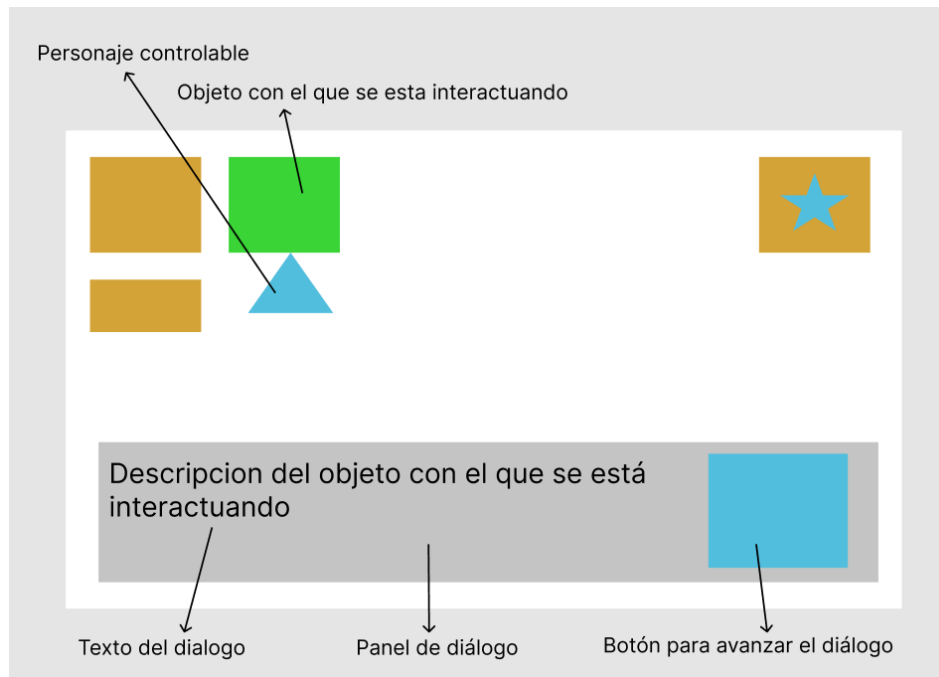
## PANTALLA DE JUEGO

Pantalla en la que transcurre el juego a través de diferentes niveles por medio de un personaje controlable con el teclado. Cada nivel tendrá una puerta cerrada que habrá que abrir para escapar.



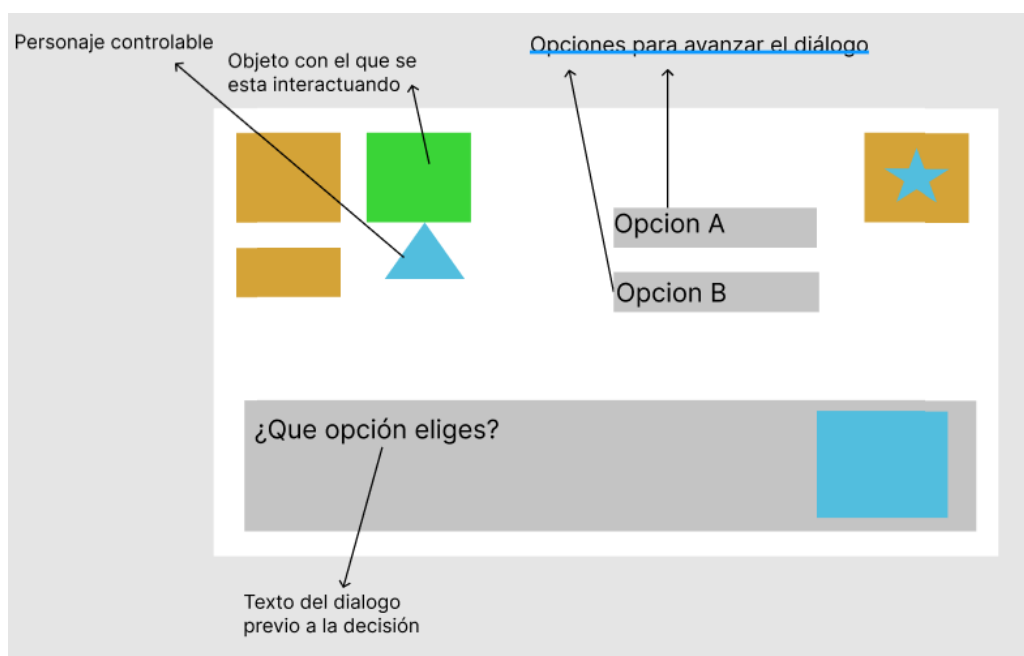
## DIÁLOGOS

Al interactuar con determinados objetos, saldrá un cuadro de diálogo que mostrará descripciones o pistas.



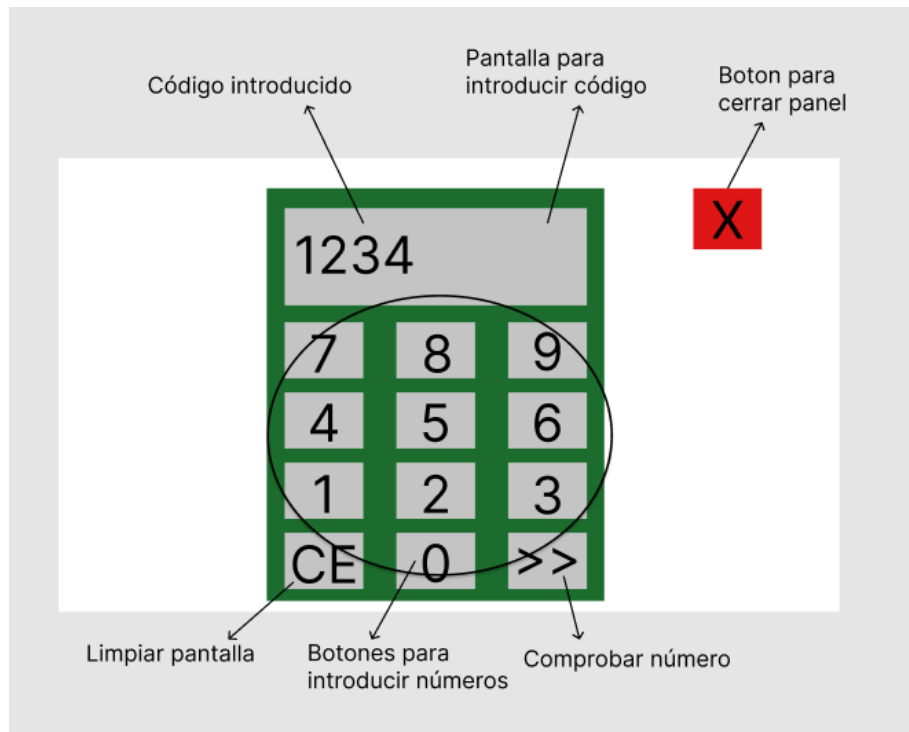
## DIÁLOGOS CON OPCIONES

Determinados diálogos presentarán opciones al jugador que podrá seleccionar para llevar a distintos resultados.



## PANTALLA PARA INTRODUCIR CÓDIGOS

La misión principal de cada nivel es escapar a través de una puerta cerrada. Para ello, se debe introducir un código (mediante clicks de botones en pantalla) que se debe obtener investigando el nivel.



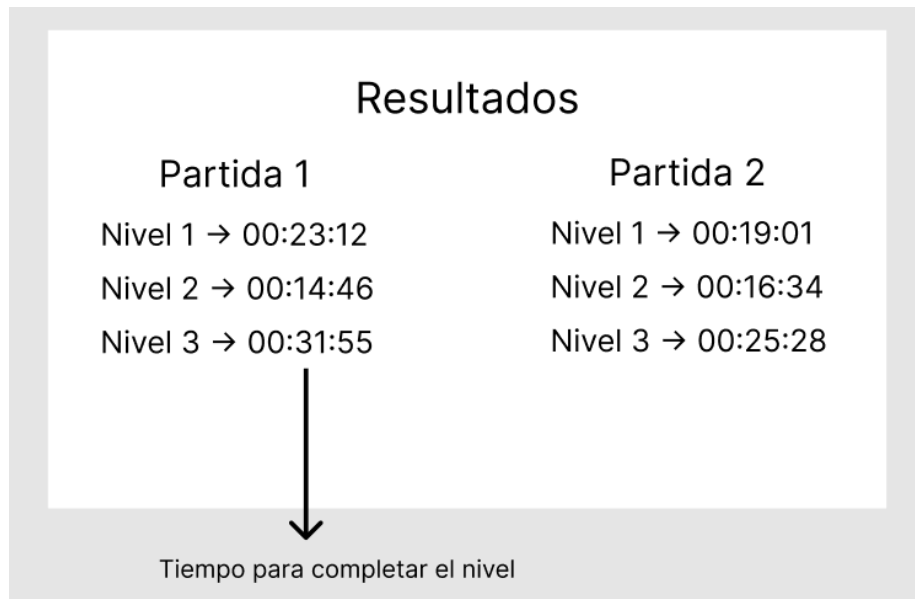
## PANTALLA DE CRÉDITOS

Se habilitará una pantalla para atribuir a las personas que han creado los assets que se han descargado.



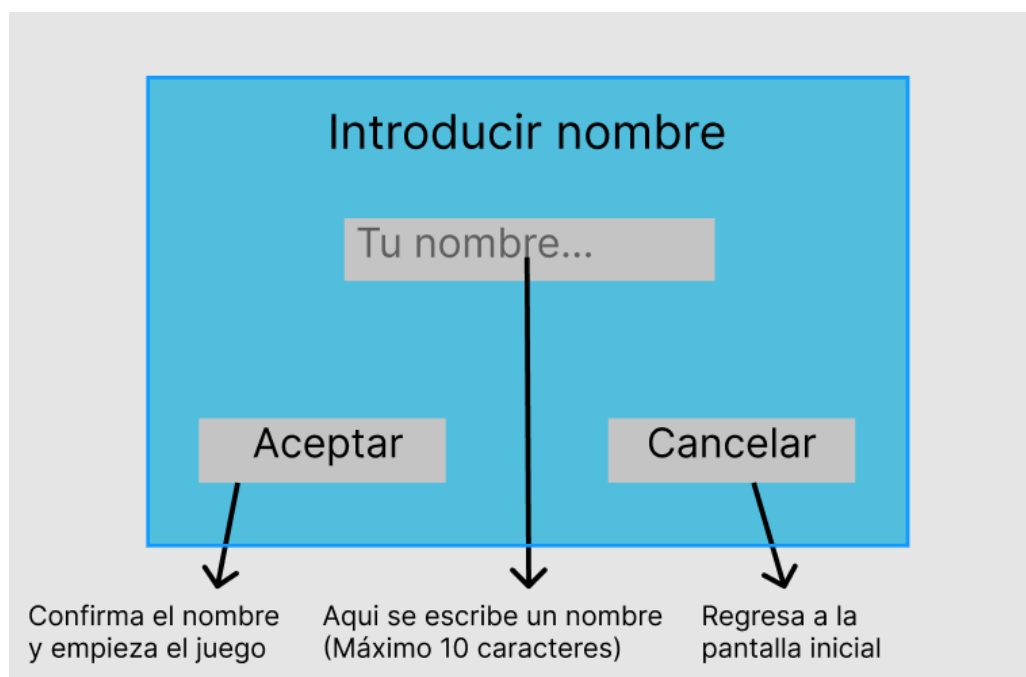
## PANTALLA DE RESULTADOS

Esta pantalla mostrará estadísticas de las partidas finalizadas, como el tiempo que se ha tardado en finalizar cada puzle.



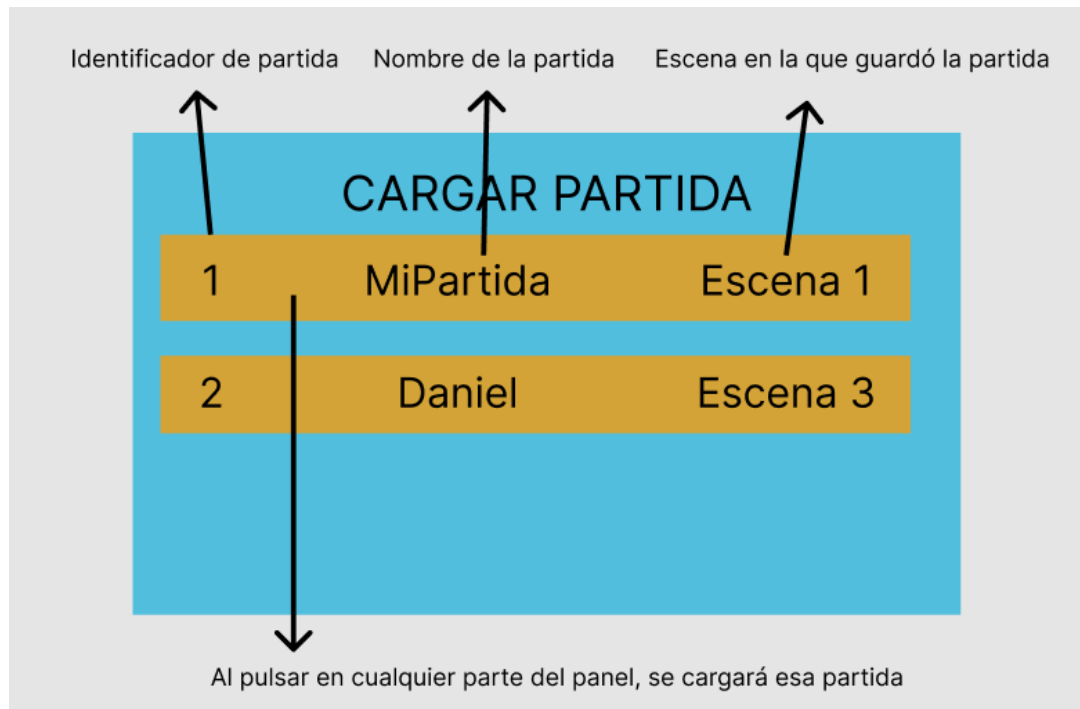
## PANTALLA DE ESCRIBIR NOMBRE

Al principio de una partida, se pedirá al jugador introducir un nombre identificativo para mostrar en otros menús como el de cargar partida o mostrar resultados.



## PANTALLA DE CARGAR PARTIDA

Cuando existan partidas guardadas, se debe seleccionar cual se quiere continuar. Al pulsar en CARGAR en el menú principal, se muestra la pantalla.



## 4.2. Diagrama de clases

Un diagrama de clases en UML(Lenguaje Unificado de Modelado) es un diagrama que sirve para describir la estructura del sistema, mostrando sus clases y las relaciones entre ellos.

Para el proyecto de Unity, toma gran relevancia conocer los tipos de elementos presentes en el juego que serán los que cuenten con los scripts entre sus componentes para hilar el funcionamiento del juego.

## PERSONAJE CONTROLABLE

El elemento a través del cual el jugador puede experimentar el juego. Cuenta con una clase `PlayerController` que define su movimiento. Además, la mayoría de `GameObject` presentes en el juego requieren detectar mediante trigger si el personaje está cerca para llevar a cabo sus correspondientes funciones.

## OBJETO INTERACTUABLE

Todo aquel elemento que responde a acciones realizadas por el jugador. Contienen texto en ficheros JSON generado mediante lenguaje Ink que se mostrará cuando el jugador interactúe con ellos gracias a un script que gestiona el avance del diálogo línea por línea, con variantes que correspondan a distintas situaciones del juego y preguntas con varias respuestas posibles.

Algunos afectarán a variables importantes para el progreso del juego, modificando datos de scripts propios o de un script de variables globales.

## PUZLE

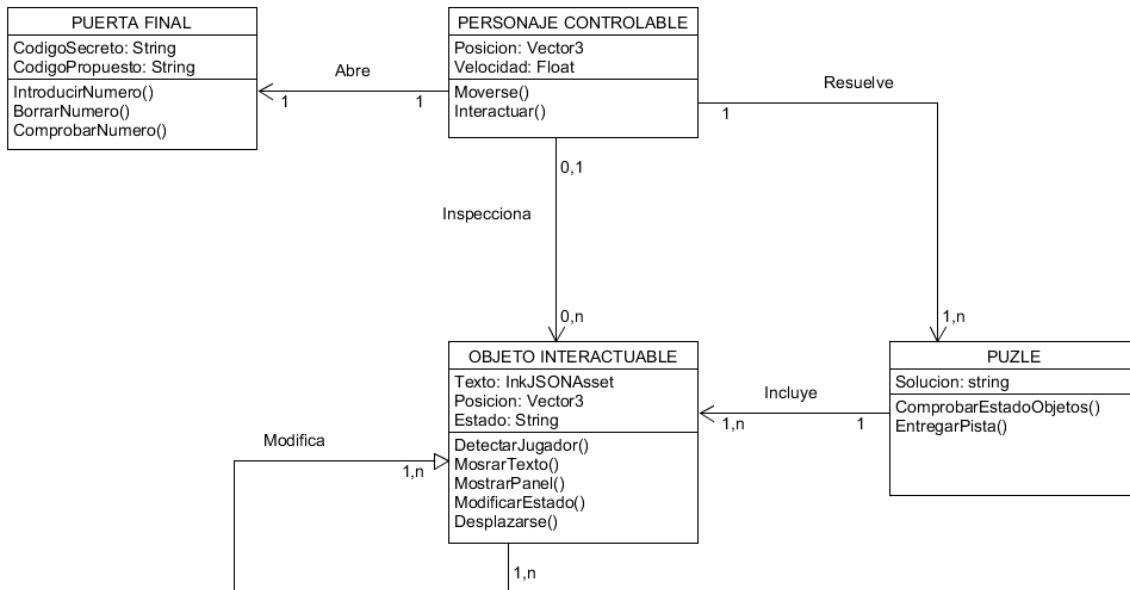
Algunos objetos interactivables estarán relacionados entre sí mediante uno o varios scripts que determinan qué acciones se deben llevar a cabo entre ellos para llegar a una determinada solución.

Hay puzzles que se desarrollan en la vista normal del juego y otros que mostrarán una imagen con sus propias normal apoyándose en botones y otros elementos visuales.

## PUERTA FINAL

Todos los niveles finalizan con una puerta cerrada que se debe desbloquear ingresando el código correcto en un panel numérico. Contará con funciones para comprobar si el número introducido es correcto o no. Además cuenta con un script que genera números aleatorios para que la solución sea distinta cada vez que se juegue.

Para representar el flujo del juego en cada nivel, se realiza un diagrama que presenta la relación entre los elementos del juego.



### 4.3. Diseño de la persistencia de la información

Es importante determinar cuál es modo en que se almacenará la información de la aplicación. Fijándome en las características de mí juego, que se corresponden con aquellas de los juegos clásicos, como son:

- Juego offline
- Sin registro previo
- Posibilidad de guardar y continuar partida
- Estadísticas de la partida dentro del juego, como tiempo de finalización de cada nivel.

He decidido gestionar la información del juego en una base de datos SQL interna mediante un fichero SQLite. En este caso, esos resultados finales quedarán contenidos en cada juego (al mismo estilo de una máquina de arcade) y no se dispondrá de ranking globales, pues realmente este tipo de información es más relevante en otros tipos de juego como los de acción o shooter.

A continuación se indican las tablas que se almacenarán:

#### PARTIDA

Contiene información de la partida. Es lo que se mostrará en la pantalla de resultados:

- IDPartida (PK)
- Nombre
- TiempoN (N Para cada nivel, almacenado en segundos como int)
- TiempoTotal (Suma de los tiempos de cada nivel)

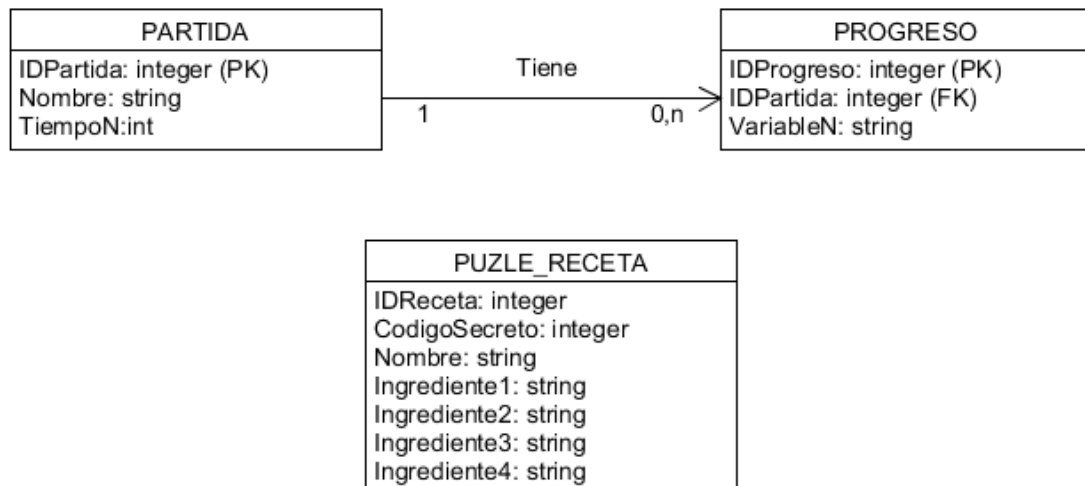
#### PUZLE\_MENU

Hay un puzle que consiste en realizar consultas desde el juego a la base de datos. Se seleccionará una fila al azar al iniciar el nivel, y se repartirá su información por la sala. La temática es de alimentos.

- IDMenu (PK)
- CodigoSecreto
- Nombre
- Ingrediente1
- Ingrediente2
- Ingrediente3
- Ingrediente4



El diagrama de clases se establece de la siguiente forma:



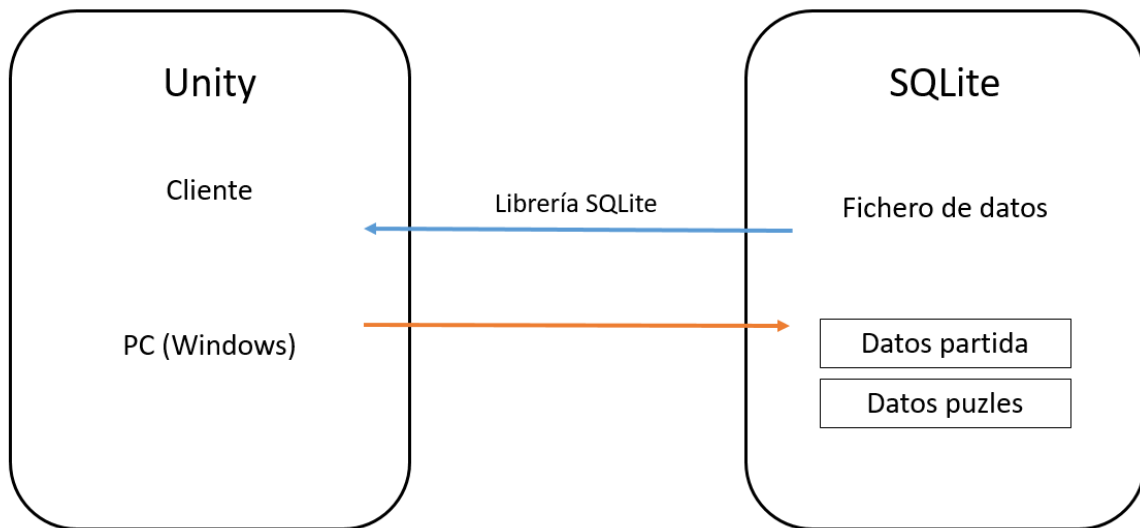
#### 4.4. Arquitectura del sistema

Al no contar con funcionalidades online y no requerir operaciones concurrentes, el juego puede cubrir sus necesidades de almacenamiento y consulta con una base de datos autocontenida y sin servidores, en un único fichero. Para esto, el motor de base de datos SQLite se muestra como una opción ligera y eficaz.



GameInfo.db

A continuación se muestra un esquema de la arquitectura del sistema:



La aplicación se desarrolla en Unity, y se elaborará el producto final para Windows. Esta herramienta ofrece muchos recursos para el desarrollo de videojuegos. También posibilita la migración a otros entornos como Linux, Mac, IOs, Android e incluso videoconsolas.

## 5. IMPLEMENTACIÓN DE LA SOLUCIÓN

### 5.1. Análisis tecnológico

Para llevar a cabo la implementación del proyecto, se estudian y analizan diferentes herramientas que nos pueden servir para el desarrollo.

Comenzando por el propio juego, se observaron las siguientes opciones:

#### **GameMaker Studio**

Esta herramienta es una potente suite de desarrollo que se usa habitualmente para juegos 2D. Cuenta con una interfaz intuitiva y un lenguaje propio basado en C llamado GML.

Aporta muchas opciones básicas para crear juegos sin necesidad de escribir código, por lo que es un buen punto de partida para principiantes, y cuenta con software integrado para gráficos y animación.

## **Unity**

Unity es una herramienta de desarrollo de videojuegos potente y con muchas alternativas. Permite crear juegos en plataformas muy variadas entre las que se incluyen PC, iPhone, Android y videoconsolas. Utiliza el lenguaje C# para creación de scripts.

Aunque inicialmente se presenta como un recurso para crear juegos 3D, su versatilidad lo hace apto para juegos 2D. Es utilizado tanto por desarrolladores independientes como por grandes editores.

## **Adventure Game Studio**

Esta herramienta se especializa en la creación de juegos de aventura, para juegos que busquen mostrar historias, diálogos y rompecabezas.

Cuenta con opciones para crear juegos sin código, aunque también permite crear tus propios scripts con un lenguaje propio similar a C.

Sin embargo, está limitado a creación de juegos para Windows.

## **RPG Maker**

Como bien indica su nombre, esta herramienta está pensada para crear juegos de rol de estilo japonés (Role Playing Game). Trae muchos recursos propios y opciones para crear tus juegos de forma sencilla y sin código. Aunque se ha partido de esta base para crear otros tipos de juego como shooters.

Aunque está pensado para principiantes, con opciones para crear diálogo, combates, exploración y cinemáticas de forma sencilla, permite crear tu propio código utilizando el lenguaje Ruby.

## **Unreal Engine 4**

Esta herramienta tiene una curva de aprendizaje muy elevada debido a que cuenta con herramientas de diseño avanzadas. Por tanto, es una herramienta más profesional.

Cuenta con un sistema Blueprint para crear la lógica de juego e incluso crear juegos sin código. Para crear tus propios scripts, se utiliza el lenguaje C++.

Unreal Engine permite distribuir tus juegos en múltiples plataformas, incluyendo móviles, videoconsolas y sistemas de realidad virtual.

Para la elección del sistema de almacenamiento de datos del sistema, se valoran las siguientes opciones.

### **MySQL**

MySQL es un SGBD relacional que pertenece a Oracle. Es un software de código abierto que emplea el lenguaje SQL. Proporciona opciones para manipular la información basada en tablas.

Para gestionar este SGBD, la opción más popular es usar la aplicación web basada en php llamada phpMyAdmin. Permite conexión a servidores remotos.

### **SQLite**

Es una herramienta de software libre, que permite almacenar datos de forma local en un dispositivo. Permite un acceso rápido a la información mediante consultas con lenguaje SQL.

Su principal ventaja radica en su sencillez, potencia y eficacia, siendo muy popular para dispositivos con pocas capacidades de hardware, como móviles.

## **ELECCIÓN**

Para el programa que voy a desarrollar, he decidido utilizar Unity. Comparada con otras de las opciones analizadas, no tiene tantas posibilidades para crear sin escribir código. Sin embargo, Unity aporta mucha flexibilidad para crear tus propios scripts, vincular otras herramientas y lanzar el producto final en diferentes plataformas. Lo cual lo hace preferible para el desarrollo actual e incluso es más válido en caso de que en el futuro se decida actualizar o añadir nuevos elementos o despliegues para nuevas plataformas.

Respecto a la opción restante, Unreal Engine, la he descartado finalmente por ser menos intuitiva y no requerir los mejores gráficos disponibles al ser un juego 2D hecho con sprites.

Para almacenar y tratar datos del programa, he elegido SQLite ya que cumple con las necesidades del programa al no requerir servidores. Por tanto, se aprovecha su sencillez y agilidad sin sufrir limitaciones en el desarrollo.

## **5.2. Elementos a implementar**

En este apartado se muestran los elementos básicos de la implementación del juego.

### **CREACIÓN DE LA BASE DE DATOS Y CONSULTA DE INFORMACIÓN**

Nada más iniciar el juego, nos encontramos con un menú en el que tenemos varios botones para realizar diferentes funciones.



Lo primero que hace el programa es crear un fichero de base de datos SQLite y sus tablas, si no existían previamente. Esto es importante para tener datos de partida y la información necesaria para la lógica de algún puzle.

```
1 referencia
public void CreateDB()
{
    using(var connection = new SqlConnection(dbName))
    {
        connection.Open();

        using(var command = connection.CreateCommand())
        {
            //Se usa el campo oculto ROWID como identificador y clave primaria
            command.CommandText = "CREATE TABLE IF NOT EXISTS PARTIDA(Nombre VARCHAR2(10), EscenaActual VARCHAR2(10))";
            command.ExecuteNonQuery();
            //command.CommandText = "CREATE TABLE IF NOT EXISTS PROGRESO(IDPartida VARCHAR2(10), FOREIGN KEY(IDP:
            //command.ExecuteNonQuery();
            command.CommandText = "CREATE TABLE IF NOT EXISTS PUZLE_MENU(CodigoSecreto VARCHAR2(4) PRIMARY KEY, 1
            command.ExecuteNonQuery();
        }

        connection.Close();
        fillMenuTable();
    }
}
```

```

public void fillMenuTable()
{
    using (var connection = new SqlConnection(dbName))
    {
        connection.Open();

        using (var command = connection.CreateCommand())
        {
            //Se usa el campo oculto ROWID como identificador y clave primaria
            command.CommandText = "SELECT COUNT(*) FROM PUZLE_MENU";

            using (IDataReader reader = command.ExecuteReader())
            {
                reader.Read();

                if(reader.GetInt32(0) == 0)
                {
                    connection.Close();
                    cargarMenus();
                }
            }
        }
        connection.Close();
    }
}

```

Para empezar a jugar, se debe seleccionar Nuevo. Aquí podrás introducir un nombre en un InputText y al confirmar se inserta una nueva fila en la tabla de Partidas asignándole un código único como clave primaria.

```

public void AddPartida()
{
    using(var connection = new SqlConnection(dbName))
    {
        connection.Open();

        using(var command = connection.CreateCommand())
        {
            command.CommandText = "INSERT INTO PARTIDA(Nombre, EscenaActual) VALUES('"+enterName.text.Trim()+"', 'Inicio')";
            command.ExecuteNonQuery();

            command.CommandText = "SELECT last_insert_rowid()";
            command.ExecuteNonQuery();

            using(IDataReader reader = command.ExecuteReader())
            {
                while(reader.Read())
                {
                    saveVariables.GetComponent<SaveVariables>().cod = reader.GetInt32(0);
                }
            }
        }
        connection.Close();
    }
}

```

En la opción Cargar se hace una búsqueda de partidas que aún no están finalizadas, y se muestran en un botón dentro de un ScrollView indicando el código

identificador, nombre de la partida y por qué nivel va. Las partidas se guardan mientras juegas.

Cargar partida			X
1	lola	Scene2	
3	animtry	Scene1	

```

public void cargarListaPartidas()
{
    using(var connection = new SqliteConnection(dbName))
    {
        connection.Open();

        using(var command = connection.CreateCommand())
        {
            command.CommandText = "SELECT ROWID, Nombre, EscenaActual FROM PARTIDA WHERE EscenaActual NOT LIKE 'Start'";

            using(IDataReader reader = command.ExecuteReader())
            {
                //Limpiar resultados anteriores
                foreach(Transform child in contentBox.transform)
                {
                    GameObject.Destroy(child.gameObject);
                }

                //Leer resultados de la select
                while(reader.Read())
                {
                    Debug.Log("Carga");
                    //Crear boton por cada resultado
                    var button = Instantiate<Button>(btnSaveSlot);

                    button.transform.GetChild(0).GetComponent<Text>().text = reader.GetInt32(0).ToString();
                    button.transform.GetChild(1).GetComponent<Text>().text = reader.GetString(1);
                    button.transform.GetChild(2).GetComponent<Text>().text = reader.GetString(2);
                    //Coloca el boton en el content del scrollView
                    button.transform.SetParent(contentBox.transform, false);
                }
            }
        }
    }
}

```

Al pulsar en una de estas partidas, recoge su id y te lleva a la escena indicada en tu progreso. El id es necesario para los guardados posteriores.



```

public void cargarPartidaSeleccionada()
{
    saveVariables.GetComponent<SaveVariables>().cod = int.Parse(gameObject.transform.GetChild(0).GetComponent<Text>().text);
    try
    {
        SceneManager.LoadSceneAsync(gameObject.transform.GetChild(2).GetComponent<Text>().text);
    }
    catch (Exception ex)
    {
        Debug.Log("No se puede cargar la escena");
        Debug.Log(ex.Message);
    }
}

```

El menú Resultados también hace consultas en base de datos, pero en este caso localiza las partidas que ya han acabado, generando un panel para cada uno que informa de datos como el tiempo de juego total y de cada nivel. Están ordenados por menor tiempo, a modo de ranking.

Resultados			
yorr	Scene 1 :	00:19:334	Tiempo total
	Scene 2 :	00:21:783	00:00:54.739
	Scene 3 :	00:13:622	
pru	Scene 1 :	00:36:532	Tiempo total
	Scene 2 :	00:15:899	00:01:06.189
	Scene 3 :	00:13:758	

```

public void cargarListaResultados()
{
    using (var connection = new SqliteConnection(dbName))
    {
        connection.Open();

        using (var command = connection.CreateCommand())
        {
            command.CommandText = "SELECT ROWID, Nombre, Tiempo1, Tiempo2, Tiempo3, TiempoTotal FROM PARTIDA "+
                "WHERE EscenaActual LIKE 'Start' ORDER BY TiempoTotal";

            using (IDataReader reader = command.ExecuteReader())
            {
                //Limpiar resultados anteriores
                foreach (Transform child in contentBox.transform)
                {
                    GameObject.Destroy(child.gameObject);
                }
                //Leer resultados de la select
                while (reader.Read())
                {
                    //Crear boton por cada resultado
                    var panelResults = Instantiate<GameObject>(panelPlayerResults);
                    //Asignar texto a los botones

                    panelResults.transform.GetChild(0).GetComponent<Text>().text = reader.GetString(1);
                    panelResults.transform.GetChild(1).GetComponent<Text>().text = reader.GetString(2);
                    panelResults.transform.GetChild(2).GetComponent<Text>().text = reader.GetString(3);
                    panelResults.transform.GetChild(3).GetComponent<Text>().text = reader.GetString(4);
                    panelResults.transform.GetChild(4).GetComponent<Text>().text = reader.GetString(5).Substring(0, reader.GetString(5).Length-4);
                    //Coloca el boton en el content del scrollView
                    panelResults.transform.SetParent(contentBox.transform, false);
                }
            }
        }
        connection.Close();
    }
}

```

## FUNCIONAMIENTO DEL JUEGO

Una vez creamos o cargamos una partida, podemos jugar. Para esto, controlamos un personaje que tiene asignado un script de movimiento. Puedes caminar con las teclas de dirección.

```
void Update()
{
    if(canMove)
    {
        verticalInput = Input.GetAxis("Vertical");
        horizontalInput = Input.GetAxis("Horizontal");

        if(horizontalInput!=0 || verticalInput!=0){

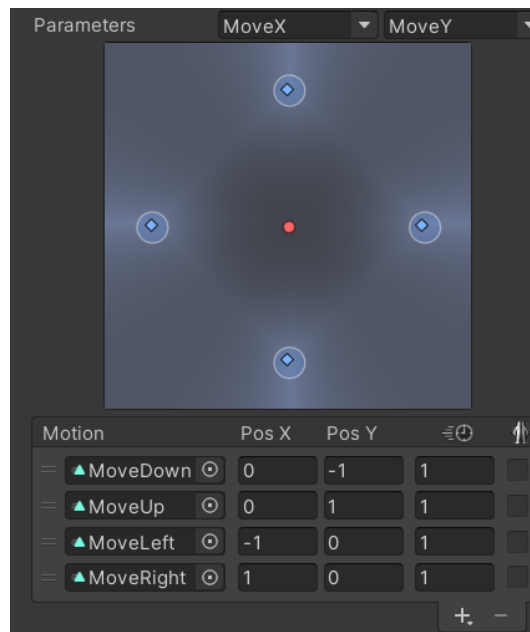
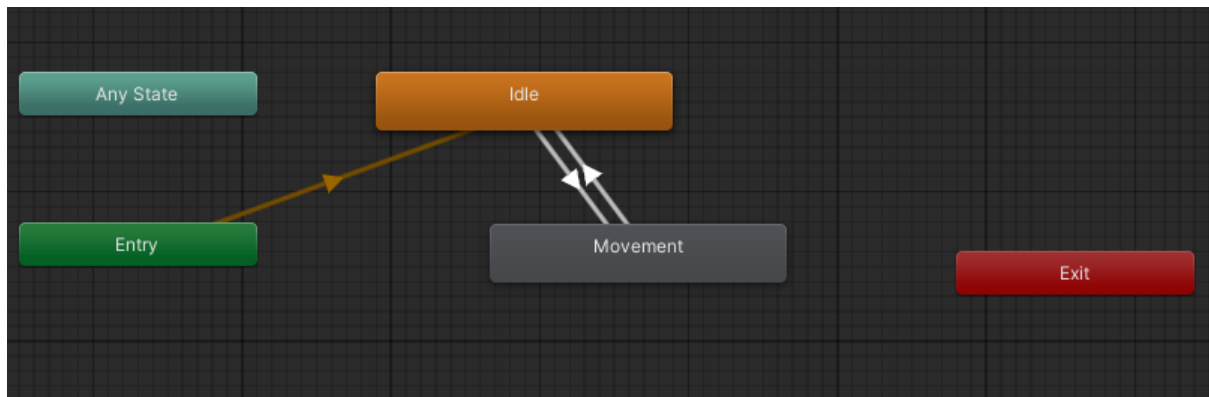
            if(!isWalking){
                isWalking = true;
                anim.SetBool("isWalking",isWalking);
            }

            anim.SetFloat("MoveX",horizontalInput);
            anim.SetFloat("MoveY",verticalInput);

            this.transform.Translate(Vector3.up * playerSpeed * Time.deltaTime * verticalInput);
            this.transform.Translate(Vector3.right * playerSpeed * Time.deltaTime * horizontalInput);

        }
        else
        {
            if(isWalking)
            {
                isWalking = false;
                anim.SetBool("isWalking",isWalking);
            }
        }
    }
}
```

El personaje tiene un set de animaciones que reaccionan a la dirección en la que se está moviendo, simulando sus pasos en dicha dirección o quedándose quieto mirando en la dirección que finalizó.



Al acercarse a ciertos objetos y pulsar la tecla Z se generará una interacción. Para detectar si el jugador está en dicho rango se usa un trigger:

```

private void OnTriggerEnter2D(Collider2D collision)
{
    if (collision.gameObject.CompareTag("Player"))
    {
        isPlayerInRange = true;
    }
}

private void OnTriggerExit2D(Collider2D collision)
{
    if (collision.gameObject.CompareTag("Player"))
    {
        isPlayerInRange = false;
    }
}
  
```

La interacción incluye en muchas ocasiones la aparición de texto para dar descripciones o pistas. Se muestra un panel con diálogo y un botón para avanzar a la siguiente línea. El texto se ha generado con lenguaje Ink, que proporciona un JSON que se pasa al objeto que vaya a mostrarlo.



El diálogo se gestiona desde un objeto llamado InkManager.

```
void Update()
{
    if(isPlayerInRange && Input.GetKeyDown(KeyCode.Z))
    {
        if(player.GetComponent<PlayerController>().canMove)
        {
            player.GetComponent<PlayerController>().canMove = false;
            player.GetComponent<Animator>().SetBool("isWalking",false);
            inkManager.GetComponent<InkManager>().StartStory(inkJsonAsset, variableInk, itemTag);
        }
    }
}
```

```

public void DisplayNextLine()
{
    if (_story.canContinue)
    {
        string text = _story.Continue(); // gets next line
        text = text?.Trim(); // removes white space from t
        _textField.text = text; // displays new text
    }
    else if (_story.currentChoices.Count > 0)
    {
        DisplayChoices();
    }
    else if (!_story.canContinue)
    {
        EndStory();
    }
}

```

1 referencia

```

private void EndStory()
{

```

Algunas veces se muestran varias opciones posibles para avanzar el diálogo. Cada opción posible instancia un botón de selección.

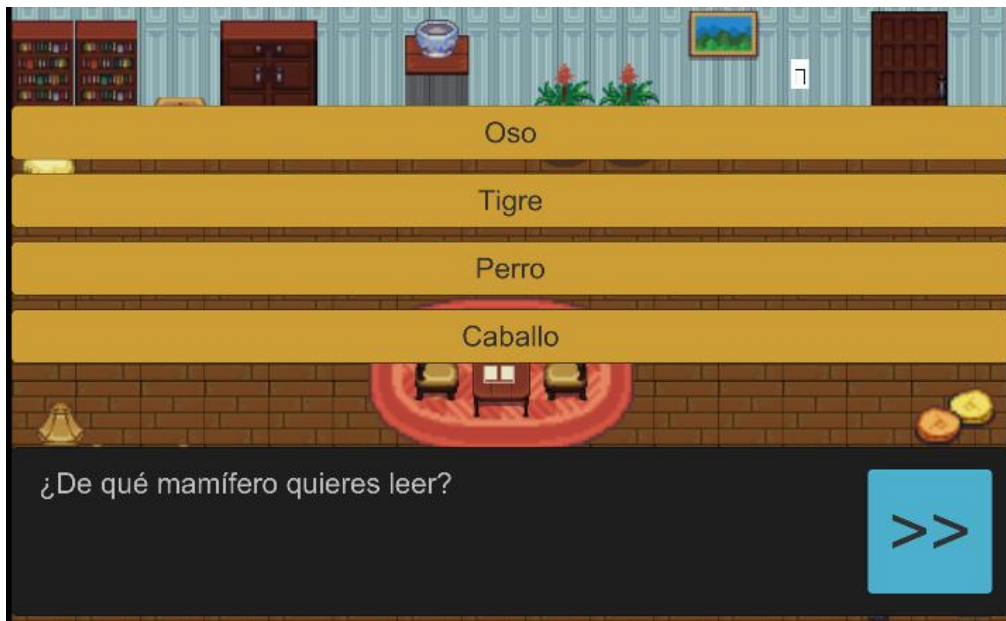
```

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 interacciones se deben registrar para cambiar el comportamiento de ese objeto u otros. Para esto se dispone de una clase GlobalVariables.

```
public class GlobalVariables : MonoBehaviour
{
    ArrayList variables = new ArrayList();
    ArrayList values = new ArrayList();

    // Start is called before the first frame update
    📄 Mensaje de Unity | 0 referencias
    void Start()
    {
        //0 false
        //1 true
        variables.Add("gotKey1");
        values.Add("0");
        variables.Add("gotKey2");
        values.Add("0");
        variables.Add("taponEncontrado");
        values.Add("0");
    }

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

    3 referencias
    public void changeVariable(string nomVar, string valueVar)
    {
        this.values[variables.IndexOf(nomVar)] = valueVar;
    }

    2 referencias
    public string getValue(string nomVar)
    {
        return this.values[variables.IndexOf(nomVar)].ToString();
    }
}
```

Para superar el nivel, es necesario introducir un código secreto en un panel numérico. El panel aparece al inspeccionar la puerta final. Pulsando los botones puedes introducir números, borrar lo que has escrito y probar si el código es correcto.

También hay un botón rojo que cierra el panel. Es un elemento recurrente en varios paneles del juego.



```

public void inputNumber()
{
    if(screen.text.Length < codeLength)
    {
        //screenNum = screen.text;
        inputNum = gameObject.GetComponentInChildren<Text>().text;
        Debug.Log(inputNum);
        screen.text += inputNum;
        Debug.Log(screen.text);
    }
}
0 referencias
public void checkCode()
{
    if(screen.text.Length == codeLength)
    {
        //acertado = true;
        if(screen.text.Equals(secretNum))
        {
            closePanelCode();
            door.SetActive(false);
        }
        else
        {
            screen.text = "";
        }
    }
}
1 referencia
public void clearScreen()
{
    screen.text = "";
}

```

El código que hay que introducir cambia en cada partida. Para las escenas 1 y 2 se crea un número aleatorio entre 0000 y 9999. Se crea cifra a cifra para mostrar los números por separado repartidos por el nivel.



```

public class SecretCodeGenerator : MonoBehaviour
{
    public string n1;
    public string n2;
    public string n3;
    public string n4;
    public string numCompleto;

    // Start is called before the first frame update
    🔔 Mensaje de Unity | 0 referencias
    void Start()
    {
        n1 = Random.Range(0,10).ToString();
        n2 = Random.Range(0,10).ToString();
        n3 = Random.Range(0,10).ToString();
        n4 = Random.Range(0,10).ToString();
        //Se genera un numero de 4 cifras con las obtenidas
        numCompleto = n1+n2+n3+n4;
    }
}

```

El código secreto del nivel 3 depende de una tabla de la base de datos. Al empezar el nivel se eligen al azar varios criterios (en este caso, alimentos) y se localiza una fila única. Este registro tiene asignado un número de 4 cifras que es la solución del puzle, y las pistas son los alimentos que se deben seleccionar en unos desplegables para descubrir el número.

```

public string getRandomMenuCode()
{
    string[] listaIng1 = { "Merluza", "Atún", "Salmón", "Dorada", "Lubina" };
    string[] listaIng2 = { "Cerdo", "Pollo", "Buey", "Tertera", "Cordero" };
    string[] listaIng3 = { "Manzana", "Plátano", "Naranja", "Melocotón", "Kiwi" };
    string[] listaIng4 = { "Natillas", "Tiramisú", "Flan", "Helado", "Tarta" };

    string ingMenu1 = listaIng1[UnityEngine.Random.Range(0, 5)];
    string ingMenu2 = listaIng2[UnityEngine.Random.Range(0, 5)];
    string ingMenu3 = listaIng3[UnityEngine.Random.Range(0, 5)];
    string ingMenu4 = listaIng4[UnityEngine.Random.Range(0, 5)];

    laptopPuzzle.GetComponent<Laptop>().ing1 = ingMenu1;
    laptopPuzzle.GetComponent<Laptop>().ing2 = ingMenu2;
    laptopPuzzle.GetComponent<Laptop>().ing3 = ingMenu3;
    laptopPuzzle.GetComponent<Laptop>().ing4 = ingMenu4;

    string menuCode = "";

    using (var connection = new SqlConnection(dbName))
    {
        connection.Open();

        using (var command = connection.CreateCommand())
        {
            command.CommandText = "SELECT CodigoSecreto FROM PUZLE_MENU WHERE ING1 LIKE '" + ingMenu1 + "' AND ING2 LIKE '" + ingMenu2 + "'";

            using (IDataReader reader = command.ExecuteReader())
            {
                //Leer resultados de la select
                reader.Read();

                Debug.Log(reader.GetString(0));
                menuCode = reader.GetString(0);
            }
        }

        connection.Close();
        return menuCode;
    }
}

```

Cuando resuelves el código, la puerta se abre, y al avanzar se lanza mediante trigger un código para cambiar de escena.

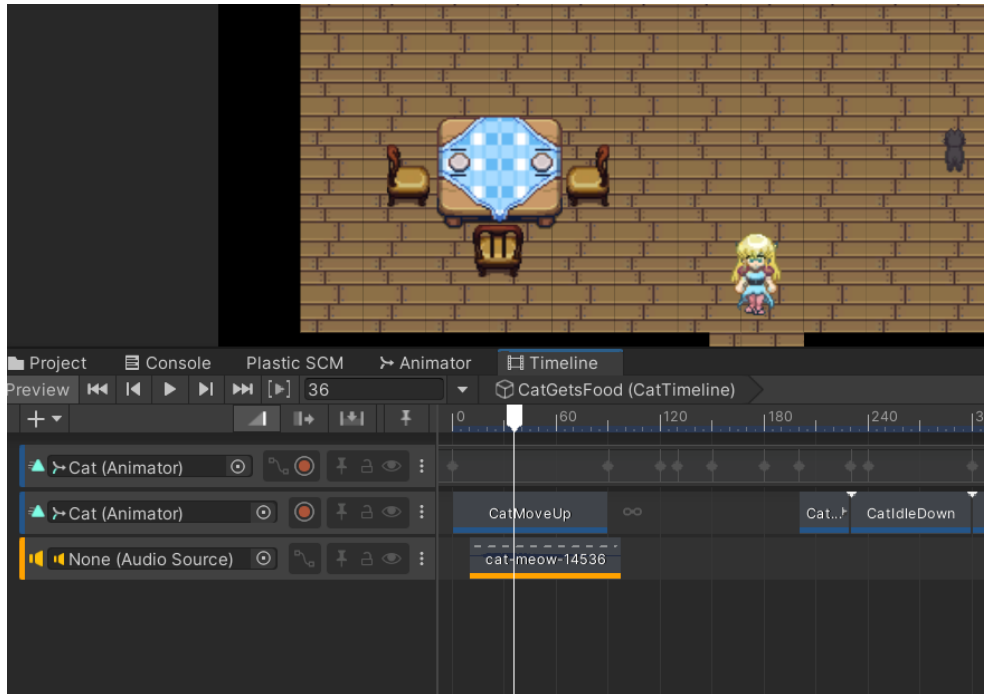
```

void Update()
{
    if(isPlayerInRange)
    {
        cam.GetComponent<DBManager>().guardarTiempo(saveVariables.GetComponent<SaveVariables>().cod, actualSceneNum, c);
        loadNewScene(newSceneName);
    }
}

1 referencia
public void loadNewScene(string sceneN)
{
    SceneManager.LoadSceneAsync(sceneN);
}

```

En ciertas ocasiones se lanza una secuencia animada creada con la herramienta Timeline. Sirven para dotar al juego de mayor personalidad o indicar visualmente que se debe hacer a continuación. Implican movimiento, cambio de animaciones y audio.



```

public class TimelinePlayer : MonoBehaviour
{
    private PlayableDirector director;
    public GameObject controlPanel;
    public bool isplaying;
    private GameObject player;
    Mensaje de Unity | 0 referencias
    private void Awake()
    {
        player = GameObject.FindWithTag("Player");
        director = GetComponent<PlayableDirector>();
        director.played += Director_Played;
        director.stopped += Director_Stopped;
        //isplaying = true;

        if (gameObject.tag == "InitialCutscene")
        {
            director.Play();
        }
    }
    1 referencia
    private void Director_Stopped(PlayableDirector obj)
    {

```

## REGISTRO DE PROGRESO DEL JUEGO

Como se ha mostrado anteriormente, la pantalla de Cargar del menú principal muestra partidas que están a medias. El progreso de la partida se guarda automáticamente al inicio de cada escena mediante el script AutoSave.

```

public class AutoSave : MonoBehaviour
{
    private GameObject saveVariables;
    [SerializeField] private GameObject panel;
    // Start is called before the first frame update
    🔔 Mensaje de Unity | 0 referencias
    void Start()
    {
        saveVariables = GameObject.FindWithTag("SaveVariables");

        if(saveVariables.GetComponent<SaveVariables>().cod != 0) //Llegas al terminar el juego
        {
            gameObject.GetComponent<DBManager>().guardarPartida(saveVariables.GetComponent<SaveVariables>().cod);
            if(panel != null)
            {
                panel.SetActive(true);
            }
        }
    }
}

```

Para recoger el tiempo utilizado para superar el nivel se usa el script Timer, con el que se obtiene el tiempo en un formato minutos:segundos:milisegundos.

```

    void Start()
    {
        currentTime = 0;
        StartCrono();
    }

    // Update is called once per frame
    🔔 Mensaje de Unity | 0 referencias
    void Update()
    {
        if(cronoActive)
        {
            if(currentTime >= 3600) //Fuerza un tiempo máximo
            {
                StopCrono();
                currentTime = 3599.999f;
            }
            else
            {
                currentTime = currentTime + Time.deltaTime;
            }

            TimeSpan time = TimeSpan.FromSeconds(currentTime);
            timeStr = time.ToString(@"mm\:ss\.fff");
        }
    }

    1 referencia
    public void StartCrono()
    {
        cronoActive = true;
    }

    1 referencia
    public void StopCrono()
    {
        cronoActive = false;
    }
}

```

Para que las funciones anteriores se apliquen a la partida correcta, se recoge su código en un script singleton que viaja entre escenas.

```

public class SaveVariables : MonoBehaviour
{
    public static SaveVariables instance;

    public int cod;

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

        instance = this;
        DontDestroyOnLoad(gameObject);
    }
}

```

Para abandonar la partida, se puede pulsar la tecla Escape, que muestra un panel que te permite cerrar la aplicación o volver al menú inicial. Al cargar esa partida de nuevo estarás al principio de la escena donde abandonaste.

```

void Update()
{
    if (Input.GetKeyDown(KeyCode.Escape))
    {
        if (gameObject.tag == "ClosePanel")
        {
            if (player.GetComponent<PlayerController>().canMove)
            {
                player.GetComponent<PlayerController>().canMove = false;
                player.GetComponent<Animator>().SetBool("isWalking", false);

                panelCloseGame.SetActive(true);
            }
        }
    }
}

0 referencias
public void returnToMainMenu()
{
    saveVariables.GetComponent<SaveVariables>().cod = 0;
    SceneManager.LoadSceneAsync("Start");
}

0 referencias
public void closeGame()
{
    try
    {
        UnityEditor.EditorApplication.isPlaying = false; //Para simular la funcionalidad
    }
    catch
    {
        Application.Quit(); //Para la version final
    }
}

```

## 6. TESTEO Y PRUEBAS DE LA SOLUCION

### 6.1. Plan de pruebas

Una parte muy importante del desarrollo es realizar pruebas para comprobar el correcto funcionamiento de los diversos elementos y sistemas que componen el juego.

- **Primera ejecución del programa:**
  - Solo la primera vez que se ejecute el juego debe crearse un fichero con extensión .db en el que se crean las tablas que contienen la información necesaria para el juego.

Nombre	Fecha de modificación	Tipo	Tamaño
Home Prison_Data	13/06/2022 0:34	Carpeta de archivos	
MonoBleedingEdge	12/06/2022 20:02	Carpeta de archivos	
GameInfo.db	15/06/2022 0:10	Data Base File	52 KB
Home Prison.exe	13/06/2022 0:34	Aplicación	639 KB
UnityCrashHandler64.exe	27/01/2022 0:45	Aplicación	1.205 KB
UnityPlayer.dll	27/01/2022 0:45	Extensión de la ap...	27.574 KB

- **Creación de partida:**

- No se puede confirmar si el texto está vacío o son espacios en blanco.

Al probar un nombre compuesto solo por espacios, se observa que el botón confirmar esta desactivado.

Escribe el nombre con el que se registrará tu partida:  
(Máximo 10 caracteres)

Confirmar Cancelar

- Permite repetir nombre, pera asigna distinto código identificador.

Cargar partida			X
1	daniel	Scene1	
2	daniel	Scene1	

- **Abandonar una partida a medias:**
  - Hay un registro en la pantalla cargar.
  - Indica la escena en la que se dejó de jugar
  - Al pulsarlo inicia esa escena.

Se crea la partida prueba2, y se interrumpe al empezar la escena 2.

Cargar partida			X
1	daniel	Scene1	
2	daniel	Scene1	
3	prueba2	Scene2	

La partida empieza en la escena 2 al pulsar en el botón de prueba2.

- **Completar una partida:**
  - Desaparece su registro de la pantalla cargar.
  - Se crea un registro en la pantalla resultados.
  - Hay datos de tiempo de cada puzle y el total que es la suma de los anteriores.



Se prueba a completar una partida con el usuario prueba2. Desde el menú principal se pulsa el botón Resultados y tiene su registro con tiempos parciales y total.

Resultados				X
prueba2	Scene 1 :	00:17:144	Tiempo total	
	Scene 2 :	00:20:794	00:00:48.856	
	Scene 3 :	00:10:918		

- **Control del personaje:**

- Mira en la dirección de movimiento.
- Realiza animación de caminar mientras se mueve.

Se dirige al personaje en las 4 direcciones. También admite movimiento en otras direcciones, pero sus posiciones se determinan por el eje y sentido más predominante, ajustándose a los sprites disponibles, como es habitual en juegos 2D. Al detenerse se queda mirando en la dirección que se estaba moviendo.



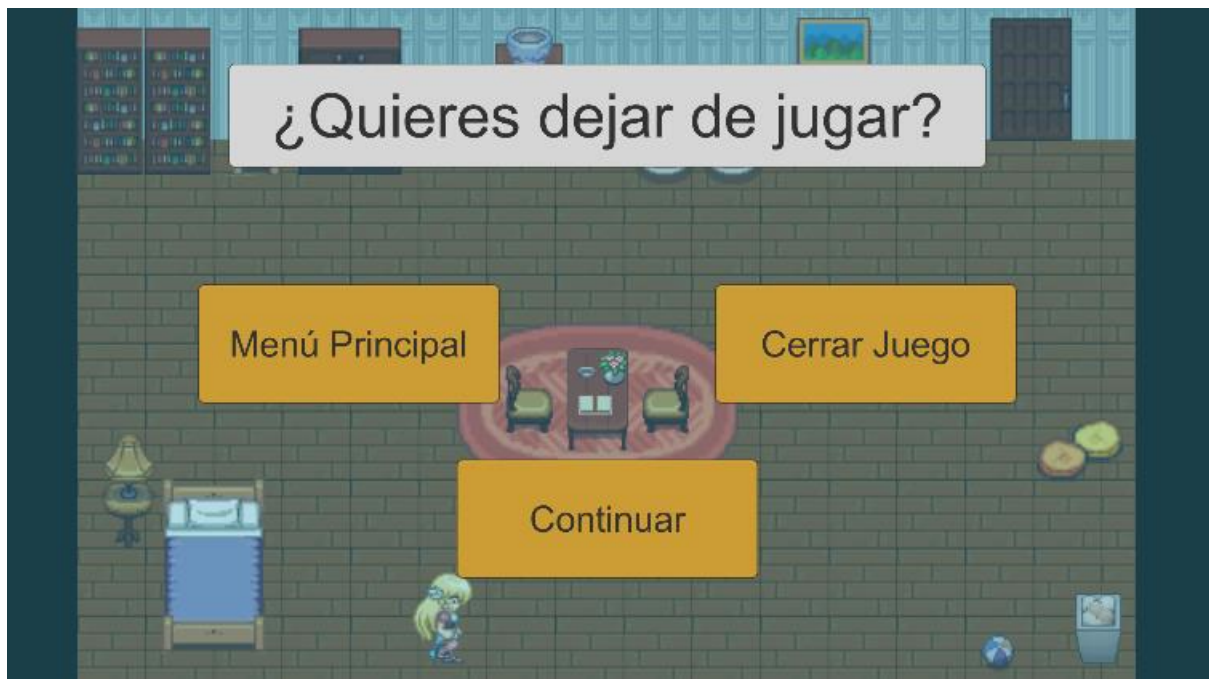
- No puede moverse al iniciar una interacción (diálogo o abrir panel) y se vuelve a poder mover al terminarla.

Se utiliza un booleano para prohibir el movimiento al interactuar, y habilitarlo de nuevo al terminar. Se controla cada escenario, particularmente el que tiene un Timeline seguido de dialogo. En el instante de transición entre ambas, se podían realizar acciones no pretendidas como abrir el menú de pausa con la tecla Escape, solucionado tratando esas dos partes como un único bloque.

- **Menú de salir de la partida:**

- Volver al menú regresa a la pantalla inicial.
- Cerrar juego cierra por completo la aplicación.

Se prueban en mitad de una partida, y realizan sus funciones pretendidas.



- **Panel numérico:**

- Escribe el número pulsado
- Borra lo escrito con el botón CE
- Solo permite 4 números
- Comprobar numero con el botón >>. Si falla borra el número y si acierta cierra el panel y desaparece la puerta.

Al marcar el quinto número, no se escribe nada. Si se quiere modificar la cifra, se debe borrar el número entero con CE.



El botón de confirmar funciona como se pretende. La puerta desaparece al acertar, dejando un espacio al que el jugador se debe dirigir para ganar la escena.

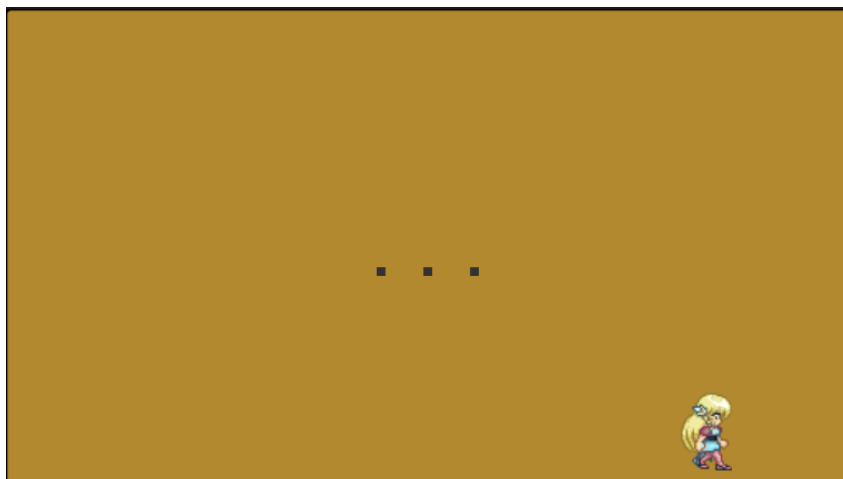


#### - Terminar nivel

- Aparece pantalla intermedia para suavizar la transición entre escenas.
- Al abrir siguiente sala, se actualiza el progreso de la partida.

Al acabar cada nivel cruzando la puerta final, se visualiza el siguiente panel durante un tiempo determinado. Después se abre la siguiente

escena y entonces se actualiza la partida como se comprobó en una prueba anterior.



## **PRUEBAS DE LÓGICA DE LOS NIVELES DEL JUEGO**

Además de las pruebas ya expuestas, los videojuegos implican numerosas interacciones entre objetos y ejecución simultanea de múltiples procesos que en muchas ocasiones dan resultados inesperados y difíciles de predecir, conocidos como bugs. Algunos pueden ser inofensivos y otros ser críticos para el progreso del juego.

El desarrollo ha incluido muchas pruebas para reducir en la medida posible el número de fallos. A continuación se presenta una muestra de pruebas importantes para poder completar el juego exitosamente:

- **Nivel 1**
  - Para obtener uno de los números, es necesario “mirar al cielo estrellado”. Esto implica apagar la luz de la sala y encender la lámpara de la mesita. Solo al cumplir estas condiciones, el personaje se puede tumbar en la cama y descubrir la pista en el techo. Se prueban las combinaciones posibles y que solo se obtenga en la correcta.



- Una pista está en el jarrón del estante. Para alcanzarla se debe empujar una silla a una posición cercana y entonces interactuar con ella para ver cómo se sube y obtiene la pista. Se prueban las opciones:



## - Nivel 2

- Ambientado en un cuarto de baño, todas las acciones conducen a conseguir que se empañe el espejo para ver los números dibujados en él. Por ejemplo, necesitas encontrar un tapón para ponerlo en la bañera. Solo al conseguirlo debe preguntar si quieres ponerlo:





- El dialogo generado con Ink admite variables que deben reaccionar al estado de la bañera, según las decisiones del jugador:

bathtink

File Edit View Story Ink Window Help

No issues.

```

1  VAR situacionBath = -1
2  Es la bañera.
3  -> Inicio
4
5  === Inicio ===
6  {situacionBath == -1: -> noTapon}
7  {situacionBath == 0: -> tienesTapon}
8  {situacionBath >= 1: -> taponPuesto}
9  ->END
10
11 === noTapon ===
12 ¿Quieres abrir el grifo?
13 ***Si
14   -> echarAgua
15 ***No
16   ->END
17
18 === echarAgua ===
19 ¿A qué temperatura?
20 ***Fría
21   Abres el grifo y dejas el agua correr.
22   Para no desperdiciar agua, cierras el grifo.
23   ->END
24 ***Templada
25   Abres el grifo y dejas el agua correr.
26   Para no desperdiciar agua, cierras el grifo.
27   ->END
28 ***Caliente
29   Abres el grifo y dejas el agua correr.
30   Para no desperdiciar agua, cierras el grifo.
31   ->END
32
33 === tienesTapon ===
34 ¿Quieres poner el tapón?
35 ***Poner
36   -> ponerTapon
37 ***No poner
38   -> END
39
40
41 === taponPuesto ===
42 ¿Que quieres hacer?

```

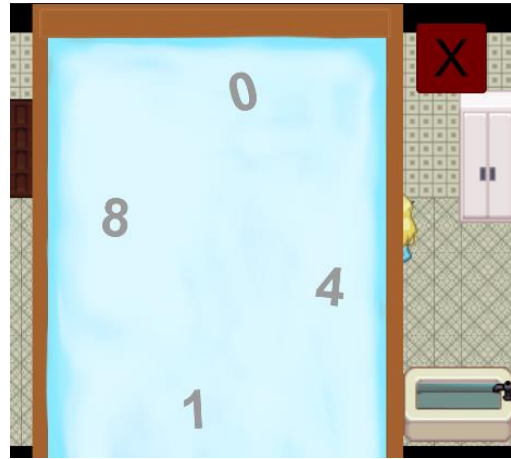
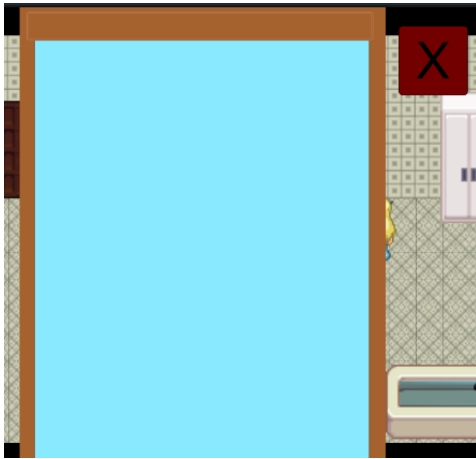
Es la bañera.

¿Quieres abrir el grifo?

Si

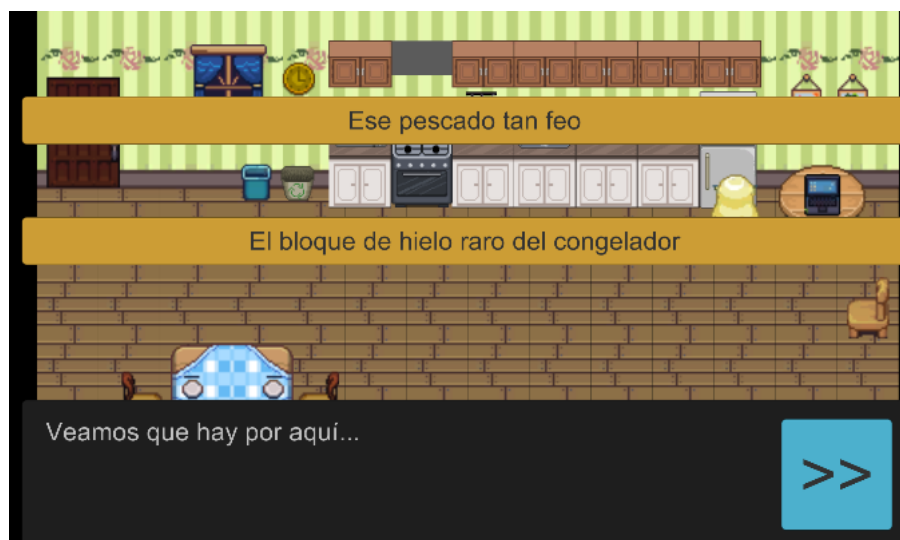
No

- Se prueban todas las combinaciones. A destacar que llenar la bañera de agua se refleja en la imagen y que solo el agua caliente empaña el espejo.



### - Nivel 3

- Hay una nevera con 2 objetos posibles por coger. Solo al tenerlos se puede interaccionar con otros. Al usarlos se pierden, y no se puede tener los dos a la vez. Se comprueba que el diálogo te de la opción de dejar el objeto donde estaba:







- El portátil de la mesa es una base de datos de menús, donde usas listas desplegables y buscas resultados. Se comprueba que hace búsquedas en la tabla de menús de la base de datos al pulsar en Buscar.



## 6.2. Solución a problemas encontrados

Aunque el desarrollo del proyecto ha sido suficientemente fluido, me he encontrado con algunos problemas que requirieron investigación y pruebas:

### **SISTEMA DE DIÁLOGO**

En un principio fui capaz de diseñar un sistema sencillo basado en ir cargando en un elemento text una lista de String. Demostró ser efectivo para diálogos sencillos. Sin embargo, a la hora de hacer texto más elaborado y decisiones se quedó corto y hubo que recurrir a otras alternativas.

Finalmente descubrí la herramienta Ink con su editor Inky, que permite construir diálogo con decisiones de forma sencilla, y viene acompañado de tutoriales para crear la base del código. El sistema de diálogo se gestiona desde un gameObject con un script llamado InkManager, que recoge el texto elaborado traducido automáticamente a JSON que se puede leer línea por línea.

También hay que destacar que en algunos casos ha sido necesario modificar una variable de Ink desde el código del juego de Unity y viceversa.

## CREACIÓN DEL MUNDO DEL JUEGO

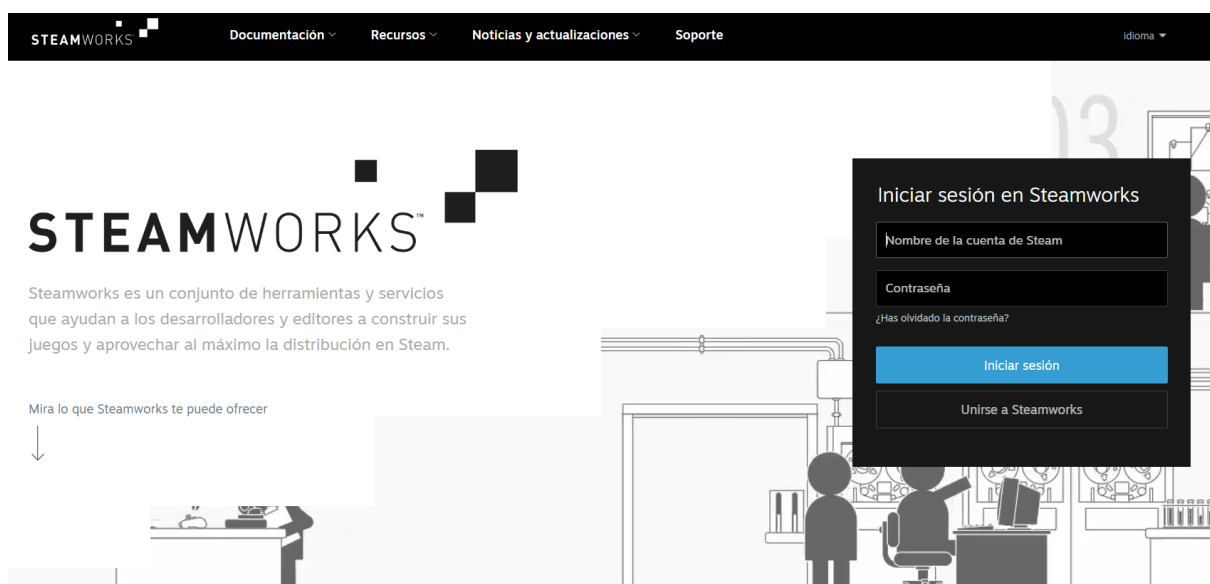
El primer reto con el que me encontré en mi proyecto fue traducir el aprendizaje de clase con juegos 3D a un mundo 2D, donde todo está sujeto a normas más rígidas. He aprendido a buscar y recortar tilesets con las herramientas de Unity para usarlos como suelo, paredes u objetos del juego. Además, para que estéticamente el juego sea uniforme, hay que encontrar assets con estilos similares lo que limita bastante las opciones si no empleas bastante tiempo en aprender y crear los tuyos propios o pagas a alguien para que los cree para tu proyecto.

## 7. LANZAMIENTO Y PUESTA EN MARCHA

La plataforma de destino elegida para el producto es Steam. En lo que respecta al almacenamiento de información del juego, se utiliza un fichero .db de SQLite que se genera automáticamente en la primera ejecución, sin funciones online.

Para publicar el juego en Steam, se debe seguir una serie de pasos. Para cada juego que se quiera subir, se debe pagar 100 \$.

Primero es necesario registrarse en steamworks como desarrollador. Puede hacerse usando una cuenta de Steam existente.

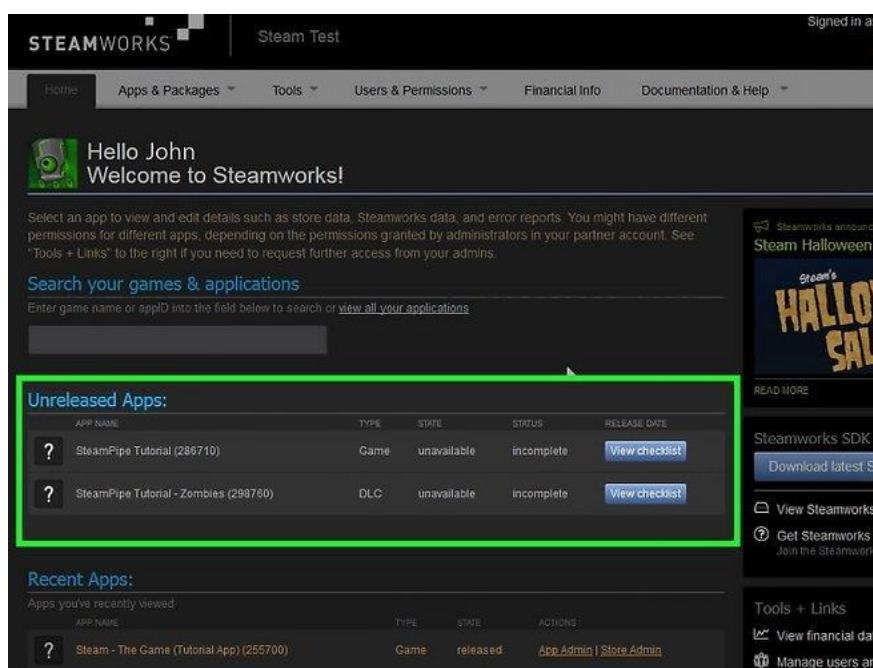


Una vez dentro hay que rellenar una serie de formularios para dar información sobre datos como nombre o nombre de la compañía, cuanta bancaria o dirección.

También es necesario crear una página que será tu tienda de juegos en Steam.

En ésta página se pueden ver los juegos tus juegos sin publicar. Aquí se puede establecer la fecha de lanzamiento, siempre que el juego supere un proceso de evaluación. Para ser aprobado, se debe aportar previamente información:

- Datos básicos como nombre, desarrollador, redes sociales...
- Una descripción larga y otra breve que definan el juego.
- Rating: En función del contenido se debe establecer la edad mínima para jugar. Esto se puede obtener de una empresa oficial o esperar a que lo valoren.
- Opciones para acceso anticipado.
- Imágenes y trailers. Al menos uno debe ser sobre el sistema de juego en sí, es decir, no enfocado en escenas cinemáticas no jugables directamente.
- Contenido extra como Google Analytics, contenido descargable (DLC) o demos.



The screenshot shows the Steamworks dashboard for a user named John. The dashboard includes a navigation bar with links like Home, Apps & Packages, Tools, Users & Permissions, Financial Info, and Documentation & Help. A welcome message is displayed, followed by a search bar for games and applications. A table titled 'Unreleased Apps' is highlighted with a green box, showing two entries: 'SteamPipe Tutorial (286710)' and 'SteamPipe Tutorial - Zombies (298760)'. Both are marked as 'unavailable' and 'incomplete'. Below this, there is a 'Recent Apps' section showing 'Steam - The Game (Tutorial App) (255700)' as 'released'.

APP NAME	TYPE	STATE	STATUS	RELEASE DATE
? SteamPipe Tutorial (286710)	Game	unavailable	incomplete	<a href="#">View checklist</a>
? SteamPipe Tutorial - Zombies (298760)	DLC	unavailable	incomplete	<a href="#">View checklist</a>

APP NAME	TYPE	STATE	ACTIONS
? Steam - The Game (Tutorial App) (255700)	Game	released	<a href="#">App Admin</a>   <a href="#">Store Admin</a>

Cuándo revises todo el contenido, lo puedes publicar para que se muestren los cambios de tu página.

También debes establecer precio del juego e iconos. En muchos casos se hace una rebaja de lanzamiento para atraer jugadores.

Para la propia subida del juego, es necesario los SO para los que estará disponible y datos como el directorio de instalación o nombre del ejecutable.

The screenshot shows the 'App Data Admin' interface for a game named 'SteamPipe Tutorial'. The interface has a top navigation bar with 'Home', 'Apps & Packages', and 'Documentation & Help'. Below this, the breadcrumb trail is 'Steamworks > App Admin > SteamPipe Tutorial'. The main title is 'App Data Admin: SteamPipe Tutorial'. There are several tabs: 'Basic Info', 'Builds', 'Installation', 'DRM', 'Stats + Achievements', 'Community', 'Misc', and 'Publish'. The 'Basic Info' tab is selected. Below the tabs, there is a section for editing basic metadata. It includes fields for 'Name' (set to 'SteamPipe Tutorial'), 'Type' (set to 'Game'), and 'For AppID'. There are 'Save' buttons for each. A red box highlights the 'Supported Platforms' section, which shows checkboxes for 'Windows', 'OSX', and 'Linux', all of which are checked. Below this, there is a note: 'Your game is marked as unreleased. The game name and icons will not show up publicly in the Steam Community. Please contact us if you wish to change this.' Further down, there is a 'Community Coupon Program' section with a link to 'See Community Coupon FAQ' and a checkbox for 'Enable coupons giveaways for this game'. Below that is a 'Community Display' section with a heading 'General community presence' and three radio button options: 'Default, game is shown in community once set to release state 'Coming Soon'', 'Always hide game from community', and 'Show game on community'. At the bottom, there is a section for 'Capsules' with a note: 'These capsules are used in the Steam Community when showing recent playtime on a user's profile and when listing out games their'.

Para crear el build de subida a Steam, es necesario utilizar el SDK de Steamworks. Indicando datos de ID que te proporciona Steam, y guardando los archivos a subir en la carpeta de contenido.

Cuando finalmente hayas realizado todos los pasos previos para tu juego sin publicar, debes pulsar en el botón de Listo para Revisión. Un equipo de Valve analiza tu juego y da un veredicto sobre si es apto para su publicación o no, dando información de que necesitarías cambiar.

The screenshot shows the Steamworks dashboard for appID 286710. The top navigation bar includes links for 'Tools', 'Users & Permissions', 'Financial Info', and 'Documentation & Help'. The main content area is titled 'SteamPipe Tutorial - appID 286710' and features a section for 'Prices, pricing, & release dates'. Below this, there is a table with columns for 'RELEASE DATE', 'STATE', 'PRICE', and 'ACTIVE DISCOUNT'. The table contains one row with the following data: 'Not set', 'Hidden', '\$19.99 USD', and an empty 'ACTIVE DISCOUNT' cell. To the right of the table, there is a sidebar with various settings and a 'Mark as ready for review' button highlighted with a red box. The sidebar also displays the 'Date to Release on Steam' as '11/20/14 15:00 PDT' and the 'App Release State' as 'Unavailable (?)'. Below these, there is a 'Launch Discount' of '10%' and a 'Ready for release?' checkbox. The 'Mark as ready for review' button is a red button with white text. Below the button, there is a paragraph of text explaining the review process: 'Now that you've completed the release checklist, you can select the box above to indicate that you would like Valve to review your application. Once approved, you will be able to release your application yourself.'

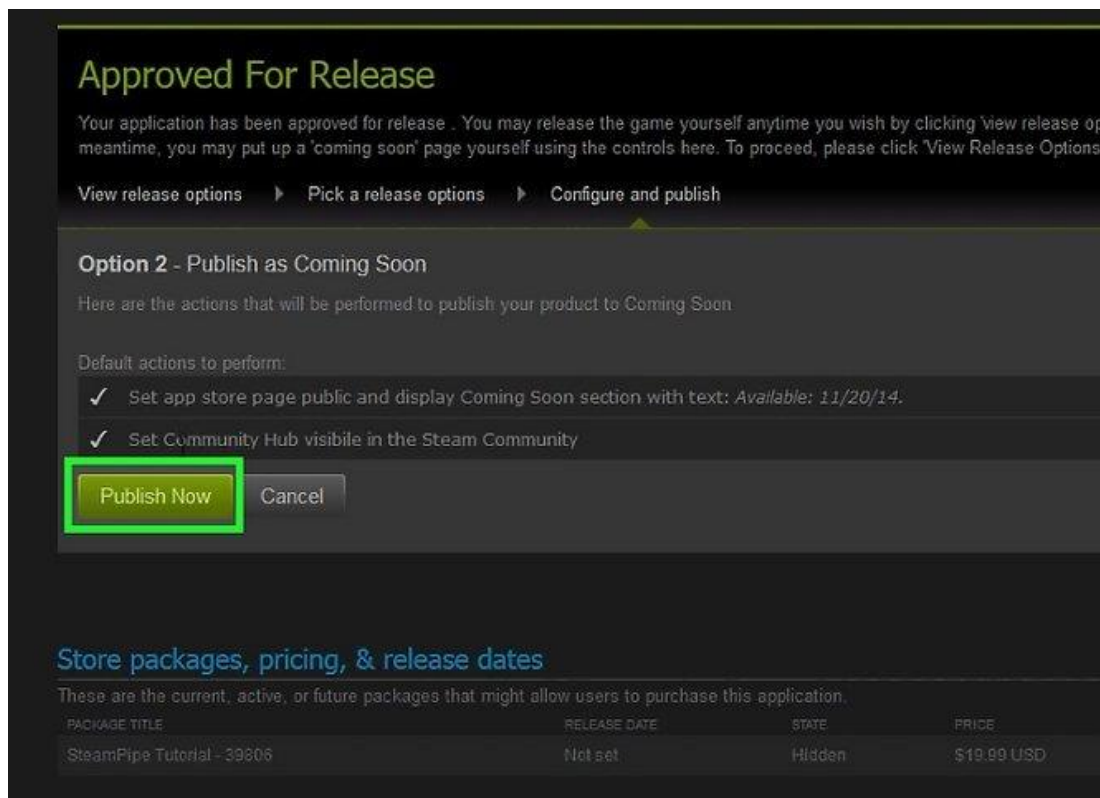
Prices, pricing, & release dates

RELEASE DATE	STATE	PRICE	ACTIVE DISCOUNT
Not set	Hidden	\$19.99 USD	

Mark as ready for review

Now that you've completed the release checklist, you can select the box above to indicate that you would like Valve to review your application. Once approved, you will be able to release your application yourself.

Una vez aprobado, puedes definir algunas opciones de publicación y confirmar el lanzamiento de tu juego en la plataforma.



## 8. MANUAL DE USO

El manual creado está disponible en el siguiente enlace:

[https://github.com/DanielADLQ/ProyectoEscapeRoom/blob/main/HOME\\_PRI  
SON\\_Manual\\_de\\_uso.pdf](https://github.com/DanielADLQ/ProyectoEscapeRoom/blob/main/HOME_PRI<br/>SON_Manual_de_uso.pdf)

## 9. VALORACIÓN Y CONCLUSIONES

La realización de este proyecto ha supuesto un reto personal que considero valioso y productivo. Cuando se me indicó que podía pensar el tema libremente, sopesé varias opciones, decantándome finalmente por un videojuego creado con Unity. Siempre había sentido curiosidad por cómo se realizan, y en éste proyecto he visto la oportunidad de crear el mío propio, lo que me ha ayudado a motivarme para avanzar y mejorar progresivamente.

Cómo bien he aprendido durante el ciclo, crear una buena aplicación requiere planificación, tiempo y esfuerzo. En el caso de un juego en 2D, también es muy importante que la aplicación sea vistosa y atractiva para el consumidor, como las que hemos creado en Desarrollo de Interfaces y Programación Multimedia y Dispositivos Móviles. Coordinar los assets a mi disposición para obtener un aspecto cohesivo ha supuesto un reto en sí mismo más allá de escribir el código del programa.

Mención aparte merece la creación de la documentación. Sobre todo al principio parecía una tarea larga y tediosa, pero he podido comprender la gran importancia de disponer de esta información para un proyecto serio. La organización de las entregas por apartados ha ayudado a hacer el proceso más llevadero.

Aunque no se ha establecido de forma clara los elementos del proyecto que deben presentarse, he procurado hacer la aplicación más completa posible en el tiempo dado. Al realizar las prácticas a la vez, he priorizado la funcionalidad y aunque considero el resultado satisfactorio, le faltaría algo de refinado en el aspecto visual. En definitiva, he disfrutado de y aprendido mucho de este desarrollo, hasta el punto de considerarlo a la vez un proyecto pasional que puede seguir creciendo en el futuro.



## 10. WEBGRAFÍA

[https://es.wikipedia.org/wiki/Videojuego\\_de\\_aventura](https://es.wikipedia.org/wiki/Videojuego_de_aventura)

<https://theescapegame.com/blog/the-history-of-escape-rooms/>

<https://theescapeeffect.com/5-escape-room-video-games>

<https://www.dynamicgc.es/dafo-analisis-modelo-negocio/>

[https://www.youtube.com/watch?v=wj\\_4KV3r1GI](https://www.youtube.com/watch?v=wj_4KV3r1GI)

<https://xsolla.com/blog/self-publish-on-steam-the-ultimate-guide#:~:text=Steam%20Publishing%20Costs,Store%20and%20in%2Dapp%20purchases.>

<https://assetstore.unity.com/>

<https://itch.io/game-assets>

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

<https://creatly.com/blog/es/diagramas/tutorial-diagrama-caso-de-uso/>

<https://unity3dtutorial.wordpress.com/2015/11/12/diagrama-de-clases/>

<https://www.oberlo.es/blog/psicologia-del-color-significados-del-color>

<https://www.sqlite.org/index.html>

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

<https://www.wikihow.com/Get-Your-Game-on-Steam>