
***Facultad de Ciencias***

# Scape Land: desarrollo de un videojuego 2D basado en la resolución de laberintos

## (Scape Land: development of a 2D video game based on maze resolutions)

**Trabajo de Fin de Grado**
**para acceder al**

**GRADO EN INGENIERÍA INFORMÁTICA**

**Autor: Francisco Alvarez Castilla**

**Director: Carlos Blanco Bueno**

**Febrero - 2022**

# Índice

# ÍNDICE DE ILUSTRACIONES

# ÍNDICE DE TABLAS

# 1. AGRADECIMIENTOS

Llegados a este punto, me gustaría agradecer, en este último trabajo antes de completar los estudios en el grado de ingeniería informática, a todas las personas que me han acompañado durante estos cuatro años y espero que continúen formando parte de mi futuro fuera de la facultad.

He conocido buenos compañeros que me han apoyado y han formado parte de algunos de mis proyectos personales. Hemos vivido juntos la tensión en los exámenes y los hemos superado como un equipo ayudándonos entre nosotros.

Por supuesto, cabe destacar todo el profesorado que ha sabido inspirarnos para afrontar las dificultades. Todos han estado ahí cuando los hemos necesitado y han hecho lo posible por entendernos.

No se puede dejar sin mencionar el apoyo que he recibido por parte de mi familia. Todos me han animado para que continúe en los momentos difíciles y también han estado ahí para ayudarme.

He aprendido mucho en cuanto a conocimientos, pero por supuesto también en el ámbito personal. Me llevo conmigo las herramientas necesarias para afrontar aquellos retos que me depare la vida laboral y la capacidad de afrontar proyectos personales.

Me gustaría agradecer nuevamente el tiempo que hemos compartido en la Universidad de Cantabria. Espero que esto no sea un adiós, sino un hasta luego.

# 2. RESUMEN

Hoy en día, uno de los mayores activos del entretenimiento son los videojuegos. Podemos encontrar todo tipo de videojuegos destinados a usuarios con diferentes gustos y personalidades. Además de entretener, tienen todo un abanico de utilidades como la educación y la medicina.

El proyecto descrito en este documento tiene como objetivo entretener al usuario mientras fortalece la memoria visual y a corto plazo. Con el fin de cubrir estos objetivos, tenemos cómo base principal del videojuego la resolución de laberintos generados automáticamente mediante el uso de un algoritmo.

Por otro lado, el jugador cuenta con distintos puntos de mejora, que incentivan la resolución de los niveles, además de algunos obstáculos que dan cierto dinamismo al juego.

Como herramienta de desarrollo tenemos el motor Unity junto con el lenguaje de programación C#, que facilita las labores de desarrollo. Haciendo uso de esta herramienta, nos centramos en el desarrollo de un videojuego para Android con un requerimiento mínimo en cuanto a las prestaciones del dispositivo utilizado.

## 2.1. PALABRAS CLAVE

1. UNITY
2. VIDEOJUEGO
3. ALGORITMO
4. ANDROID
5. PROGRAMACIÓN

# 3. ABSTRACT

Nowadays, one of the greatest entertainment assets are video games. We can find all kinds of video games aimed at users with different interests and personalities. Besides entertaining, they have a whole range of utilities such as education and medicine.

The project described in this document has the objective of entertaining the user while strengthening visual and short term memory. In order to cover these objectives, we have as the main base of the videogame the resolution of mazes generated automatically through the use of an algorithm.

On the other hand, the player has different improvement points, which motivate the resolution of the levels, as well as some obstacles that give a certain dynamism to the game.

As a development tool we have the Unity engine together with the C# programming language, which facilitates the development work. Using this tool, we focus on the development of a video game for Android with a minimum requirement in terms of the features of the device used.

## 3.1. KEY WORDS

6. UNITY
7. VIDEO GAME
8. ALGORITHM
9. ANDROID
10. PROGRAMMING

# 4. INTRODUCCIÓN

Desde la aparición de los primeros videojuegos en la década de los 50 se ha mantenido una inquietud constante por encontrar los límites alcanzables tanto de la tecnología como de las aportaciones que nos pueden dar estos programas.

La industria de los videojuegos se ha potenciado de tal forma que ha conseguido situarse entre uno de los negocios más lucrativos. Cada vez más usuarios se sienten atraídos por esta forma de entretenimiento debido al amplio abanico de posibilidades que encontramos a la hora de seleccionar un juego. Hoy en día, existen videojuegos destinados para todas las edades en los que se elige minuciosamente la temática, la dificultad, el género e incluso las estrategias que debe seguir el jugador durante la partida. Además, todo esto se ve favorecido por la gran diversidad de dispositivos que son capaces de ejecutar estos programas.

Tal es la aceptación y el entusiasmo de la sociedad por los videojuegos que ha conseguido llevar el entretenimiento ofrecido al mundo profesional. Estamos hablando de que algunos de los grandes referentes de hoy en día son personas que dedican su vida a los videojuegos haciendo uso de plataformas como Twitch o YouTube dónde consiguen conectar con una audiencia. También, encontramos torneos con grandes cantidades de dinero en juego donde jugadores profesionales y amateurs pueden disputarse el premio.

## 4.1. OBJETIVO

El objetivo de este proyecto es desarrollar un videojuego desde cero teniendo en cuenta todas las fases de desarrollo: análisis, diseño, implementación y pruebas. Se trata de un videojuego en 2D donde el jugador se sumerge dentro de laberintos en los cuales debe ir en busca de la salida.

Cada uno de estos laberintos representa un nivel diferente y a medida que se van superando los niveles la complejidad aumenta. Además de incrementar la dificultad con el tamaño, existen otras variantes del laberinto que harán que cada partida suponga un reto mayor y diferente al anterior, tal y como se describe en el apartado *ANÁLISIS DEL VIDEOJUEGO*.

## 4.2. MOTIVACIÓN

La motivación principal para haber tomado la decisión de realizar este proyecto se debe a la curiosidad de cómo se realizan este tipo de aplicaciones a las que he dedicado mucho tiempo a lo largo de mi vida. Desde pequeño he crecido con los videojuegos siendo estos una fuente de diversión y, más importante aún, de aprendizaje. Como bien se ha comentado antes, tenemos todo tipo de videojuegos y algunos de ellos se destinan a plantear retos en los cuales debes poner a prueba tus conocimientos para resolverlos.

Es por ello por lo que el videojuego que se detalla en este documento tiene el objetivo claro de ser un reto para el jugador, poniendo a prueba sus capacidades de la memoria visual y a corto plazo. Además, la experiencia del juego debe ser lo suficientemente inmersiva para qué el usuario deje a un lado la frustración que puedan generar los retos presentados y tenga ganas de superarse una y otra vez.

# 5. TECNOLOGÍAS Y HERRAMIENTAS

## 5.1. UNITY

Unity es uno de los principales motores para el desarrollo de videojuegos multiplataforma. Además, extiende ese potencial a otros sectores gracias a la compatibilidad con otros programas de diseño, cómo puede ser Adobe Photoshop o Adobe Animate.

Una de las principales ventajas que tiene el Unity es la gran comunidad que se ha generado en torno a este motor desde sus inicios en 2005. Destaca como una de las herramientas para iniciarse en el desarrollo de videojuegos, ya que se puede acceder a una gran cantidad de información en su documentación detallada y multitud de foros, en los que se resuelven dudas y aconsejan sobre diferentes metodologías para un correcto desarrollo.

### Unity Remote

Unity Remote es una aplicación diseñada para facilitar los desarrollos en las plataformas móviles. Permite conectar la aplicación instalada en un dispositivo móvil con el editor de Unity facilitando las pruebas.

Haciendo uso de esta herramienta podemos probar el juego directamente en un dispositivo móvil sin necesidad de compilar el proyecto.

## 5.2. C#

C# es un lenguaje de programación orientado a objetos, bastante moderno y fácil de entender. Buena parte de su éxito viene de las similitudes que comparte con otros lenguajes como Java.

Además de esto, incluye mejoras derivadas de otros lenguajes: el control de excepciones, la admisión de valores nulos y las operaciones asíncronas son algunos ejemplos de todas estas mejoras que hacen que este lenguaje sea el adecuado para la creación de aplicaciones duraderas y sólidas.

## 5.3. VISUAL STUDIO CODE

Este editor de texto está entre las herramientas de trabajo de aquellos usuarios que buscan un bloc de notas avanzado compatible con casi los lenguajes de programación.

Cuenta con una gran librería de extensiones destinadas a ayudar y facilitar el uso de cualquier lenguaje.

## 5.4. GIT

GIT es una herramienta muy conocida para el control de versiones permitiendo un desarrollo no lineal gracias a la gestión de ramas. Ofrece la posibilidad de acceder a un historial con las modificaciones realizadas a lo largo del tiempo.

Presenta una arquitectura distribuida que permite que varios desarrolladores tengan una copia del trabajo y puedas actualizarlo a medida que lo vayan haciendo cambios el resto de los miembros del repositorio.

# 6. ANÁLISIS DEL VIDEOJUEGO

El objetivo es desarrollar un videojuego 2D con una temática de aventura y un estilo retro que recuerde las antiguas videoconsolas. La trama principal se basa en la superación de niveles en forma de laberinto en los cuales el jugador controla el movimiento de un personaje principal que se desplaza por los pasillos del laberinto en busca de la salida.

En el camino, se presentan varias dificultades que complican el viaje y añaden ese extra de diversión e intriga que tanto gusta en los juegos.

1. Cada nivel cuenta con un tiempo acotado para superarlo. Si el tiempo se agota supone que se debe reiniciar el nivel y volver a intentarlo.
2. Aparecerán rocas en el camino que impedirán el paso y, por tanto, completar el nivel. La forma de superarlas será recoger una bomba, que se generará en algún punto del laberinto, y llevarla hasta la ubicación de la bomba para destruirla y así continuar.
3. Algunos mapas tendrán una dificultad añadida, la temperatura subirá y esto hará que el jugador necesite una botella de agua para su supervivencia. A medida que pasa el tiempo, el agua se gastara y será necesario pasar por puntos de agua, que se generaran aleatoriamente, para poder rellenarla. En caso de que el agua se agote el jugador perderá por deshidratación y tendrá que comenzar el nivel desde el principio.
4. Algunos mapas el jugador se vera sumido en la oscuridad y su visión se limitará a la luz emitida por una linterna.

A medida que el usuario avance en los niveles estos se irán complicando. Por ello, algunas estadísticas y herramientas se pueden mejorar desde una tienda: velocidad, botella, linterna, etc. Cada mejora tendrá un precio, en monedas, que el usuario deberá pagar para adquirirla. Estas monedas aparecerán repartidas aleatoriamente a lo largo de los niveles para que el usuario las recoja a medida que vaya jugando.

En cuanto a los laberintos, se generan mediante un algoritmo que se encarga de crear los pasillos y generar los objetos necesarios en su interior. Este algoritmo recibe unas paramétricas con el tamaño y el entorno del laberinto: normal, desierto o nocturno.

El proyecto se desarrolla con el motor de Unity que utiliza scripts en C# para el control de los elementos del videojuego. También, se hace uso de otras herramientas externas para llevar una clasificación de los jugadores que más niveles hayan completado.

# 7. ANÁLISIS DE REQUISITOS

En las siguientes secciones se detallan los requisitos [2] establecidos tras un primer análisis de la propuesta del proyecto.

## 7.1. REQUISITOS FUNCIONALES

En la tabla que se muestra a continuación están los requisitos funcionales para esta aplicación. Gran parte de ellos hacen referencia a mejorar los menús para que sean más intuitivos y otros a minimizar el número de acciones que debe de realizar el usuario para realizar alguna función.

| **REQUISITO** | **DESCRIPCIÓN** |
|---|---|
| RF01 | Al entrar al videojuego por primera vez se solicitará el nombre del usuario. |
| RF02 | Una vez el usuario está registrado con un nombre, aparecerá un menú inicial con acceso directo a los niveles. |
| RF03 | En el menú principal tendrá un acceso al listado de niveles y a una clasificación online. |
| RF04 | El jugador se desplazará por el mapa haciendo uso de un joystick. |
| RF05 | Mientras el jugador se desplaza, este debe estar contenido siempre en el centro de la pantalla. |
| RF06 | Al comenzar la partida se iniciará una cuenta atrás que limitará el tiempo para superar el nivel. |
| RF07 | En algunos mapas el jugador contará con una botella de agua que se irá vaciando progresivamente y deberá rellenar en puntos delimitados. |
| RF08 | La partida se podrá pausar en cualquier momento |
| RF09 | El progreso del jugador se mantendrá entre sesiones |
| RF10 | El inicio y final de los niveles estará identificado con algún tipo de marca. |
| RF11 | Algunos mapas estarán sumidos en la obscuridad y la versión del jugador se limitará al de un foco de luz. |
| RF12 | Cada uno de los mapas se genera de forma aleatoria teniendo en cuenta unas paramétricas definidas como puede ser el tamaño. |
| RF13 | Una vez la partida ha comenzado, deberá guardarse la generación del laberinto para poder continuarlo en otra sesión. |

| | |
|---|---|
| RF14 | Algunos mapas contarán con un obstáculo que requerirá de algún objeto, que aparecerá de forma aleatoria en el mapa, para atravesarlo. |
| RF15 | En los niveles se generarán monedas de forma aleatoria que podrán ser recogidas por el jugador. |
| RF16 | Existirá una tienda en la que canjear las monedas recogidas por mejoras como aumentar la velocidad de movimiento y aumentar el tamaño de la botella de agua, |
| RF17 | Al superar todos los niveles se informará al jugador mediante un texto o animación. |
| RF18 | Si el usuario pierde se le informará mediante un panel con la posibilidad de reiniciar el progreso. |
| RF19 | El usuario tendrá acceso a un menú donde poder seleccionar un nivel ya superado para volver a jugarlo. |
| RF20 | Habrá algún tipo de configuración editable por el usuario. |
| RF21 | Al completar un nivel, se cargará el siguiente de forma automática. |
| RF22 | Al navegar por los menús se visualizará algún tipo de animación para suavizar el cambio. |

*Tabla 1: Requisitos funcionales*

## 7.2. REQUISITOS NO FUNCIONALES

Los requisitos no funcionales se centran principalmente el mantener unos requerimientos bajos y tener en cuenta el lugar en que se guardan los datos, con el objetivo de evitar trampas.

Actualmente, el público objetivo de este tipo de videojuegos busca una experiencia inmersiva respaldada por un sistema solido que evite bajadas en el rendimiento del juego y no sea susceptible a fallos en la ejecución.

| **REQUISITO** | **TIPO** | **DESCRIPCIÓN** |
|---|---|---|
| RNF01 | Rendimiento | El tamaño total del juego no superara los 50MB. |
| RNF02 | Usabilidad | Interfaz intuitiva y de respuesta rápida. |
| RNF03 | Seguridad | Los datos del jugador se guardarán en un formato no editable. |
| RNF04 | Portabilidad | El juego será ejecutable en cualquier dispositivo con una versión Android 4.4 o superior. |
| RNF05 | Rendimiento | Tendrá una tasa de fotogramas por segundo de 30 estables o superior. |
| RNF06 | Usabilidad | El juego se ejecutará únicamente en el formato vertical. |
| RNF07 | Portabilidad | Las interfaces se limitarán al espacio útil de la pantalla del dispositivo, es decir, se tendrá en cuenta si el dispositivo tiene “notch”. |

*Tabla 2: Requisitos no funcionales*

# 8. METODOLOGÍA ITERATIVA INCREMENTAL

La metodología [9] utilizada para el desarrollo de este proyecto ha sido seleccionada con el objetivo de dividir el desarrollo en diferentes módulos o entregables. Por ello, se ha elegido la metodología iterativa e incremental que se adapta a mi forma de trabajar.

Haciendo uso de esta metodología hemos pasado por varias iteraciones con las fases de análisis, diseño, implementación y pruebas. El objetivo es añadir nuevas funcionalidades en cada iteración y resolver los errores y conflictos generados con iteraciones anteriores.

El trabajo se divide en cinco iteraciones para el desarrollo de este videojuego, explicadas más adelante. Mediante un diagrama de Gantt podemos ver cuál ha sido la planificación del proyecto a lo largo de tiempo.

## 8.1. DIAGRAMA DE GANTT

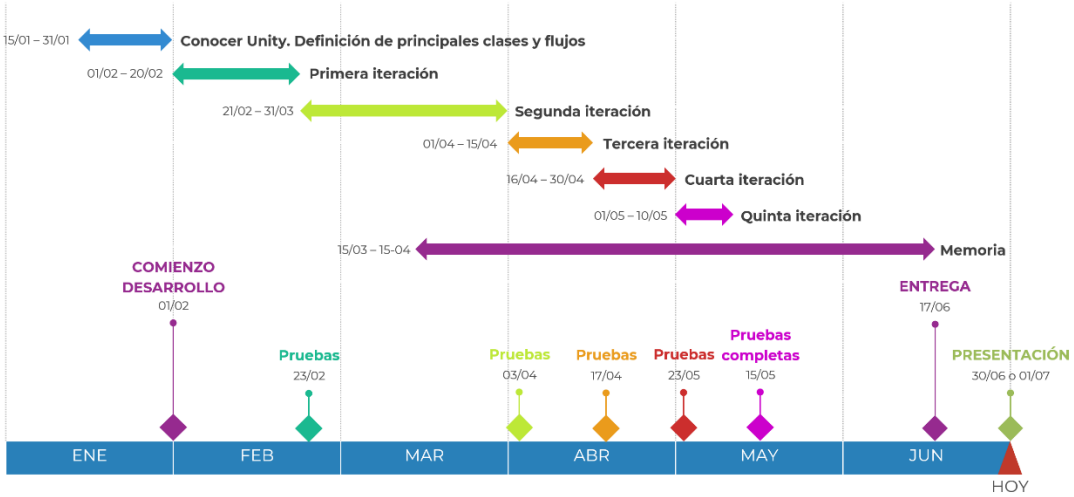


*Ilustración 1: Diagrama de Gantt*

## 8.2. PRIMERA ITERACIÓN (40 horas)

En esta primera iteración se definen las implementaciones básicas del juego a partir de las cuales se desarrollarán el resto de las funcionalidades.

1. El usuario tiene movilidad [4] por el mapa haciendo uso de un joystick.
2. Además, mientras el jugador se desplaza tiene diferentes animaciones en función de la dirección de movimiento, detallado en el apartado *Personaje y objetos*.
3. El jugador detecta colisiones con otros elementos, impidiendo su paso para delimitar el área de juego.
4. Algunos atributos del jugador (velocidad, campo de visión, etc.) se establecen mediante parámetros que puedan modificarse. Con esto podrán ser mejorados en el futuro a través de compras en una tienda.

## 8.3. SEGUNDA ITERACIÓN (110 horas)

En esta segunda etapa, se ha implementado el algoritmo que se encarga de generar los distintos niveles. Se ha comenzado con una versión básica del algoritmo de búsqueda en profundidad, generando niveles como el que podemos ver en la siguiente imagen.

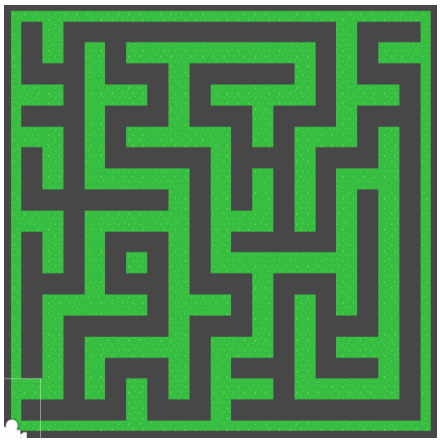

*Ilustración 2: Primera generación de laberintos*

A continuación, se ha adaptado el algoritmo para que genere objetos [5] en su interior, como las monedas, el agua, el elemento que completa el nivel al recogerlo y las piedras y bombas, teniendo en cuenta que la bomba siempre debe generarse antes que la piedra para poder recogerla y destruir la piedra.

Además, se han añadido varias capas de diseño [6] para dar cierta sensación de profundidad y mejorar el aspecto visual.

Tras estas implementaciones podemos generar niveles como el de la siguiente imagen.

*Ilustración 3: Generación final de laberintos*

### 8.4. TERCERA ITERACIÓN (60 horas)

En la tercera iteración se han creado los menús con las distintas funcionalidades que incluye cada uno de ellos, explicado más en detalle en el apartado de *Interfaces.*

En esta fase también se incluye el guardado de los progresos del usuario de forma persistente entre sesiones. Esta funcionalidad almacena información sobre los niveles superados, mejoras compradas en la tienda y lo más importante, la posición del jugador y la generación del laberinto para poder retomar el nivel donde lo dejo en la última sesión.

Además, se añade la clasificación online con el número de niveles que ha superado cada usuario, de la cual se puede ver un pódium con la tres primeras posiciones en el apartado "marcador".

El problema que se encontró en esta interacción es que, en ocasiones, las transiciones entre los menús no se completaban, dejando que algunas interfaces se viesen desplazadas hacia un lateral y mostrando fuera de la visión del usuario (fuera de la pantalla) algunos botones o textos. Tras investigar bastante sobre este tema, la solución fue restablecer la posición de los menús después de cada transición.

### 8.5. CUARTA ITERACIÓN (70 horas)

En el transcurso de esta iteración se han añadido reglas de juego como el temporizador, que define el tiempo restante para superar el nivel, y la botella de agua en algunos niveles, que se vaciara progresivamente hasta vaciarse por completo. Esta información se añade en la interfaz del usuario para que en todo momento sea consciente de estas reglas.

Estas reglas tienen como objetivo que el tiempo para superar cada nivel este acotado. Si se da el caso de que el usuario pierde por alguna de estas reglas, se mostrara una interfaz desde la cual el usuario puedo obtener una segunda oportunidad y continuar donde estaba o reiniciar el nivel por completo y volver a empezar.

El principal problema en esta fase fue que, una vez que el jugador había perdido, podía seguir desplazándose por el mapa mientras se mantenía el dedo presionando la pantalla. La solución fue incluir algunos controles en el script que gestiona el movimiento del jugador deteniéndolo en el momento que pierde.

## 8.6. QUINTA ITERACIÓN (30 horas)

En esta última iteración se ha añadido un nuevo tipo de niveles que están sumergidos en la oscuridad. En estos niveles el jugador se alumbra haciendo uso de una linterna que limita la visión para añadir dificultad.

Esto ha requerido añadir nuevas propiedades a los objetos y elementos del mapa que se han creado en la segunda iteración para que se vean afectados por la fuente de luz (linterna), que porta el jugador.

También se ha añadido en la tienda la linterna como una nueva herramienta que se puede mejorar para aumenta el alcance.

Con esta modificación se pueden generar niveles con el aspecto que se muestra en la siguiente imagen.

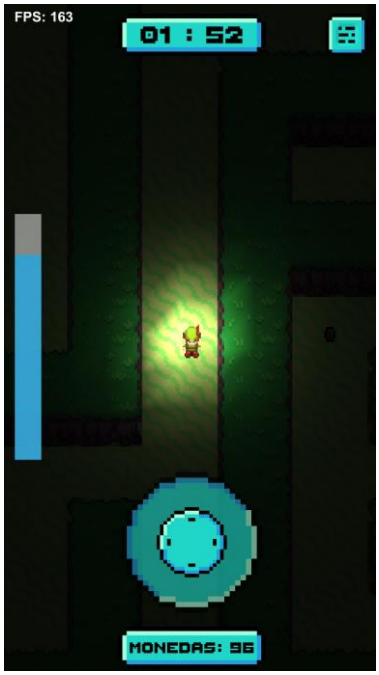


*Ilustración 4: Niveles en la oscuridad*

# 9. ARQUITECTURA, DISEÑO E IMPLEMENTACIÓN

En este proyecto se ha optado por la arquitectura por capas. Comenzando por la capa de presentación más cercana al usuario, seguida de la lógica de juego contenida en la capa de negocio y finalmente la persistencia de la información en la capa de datos.

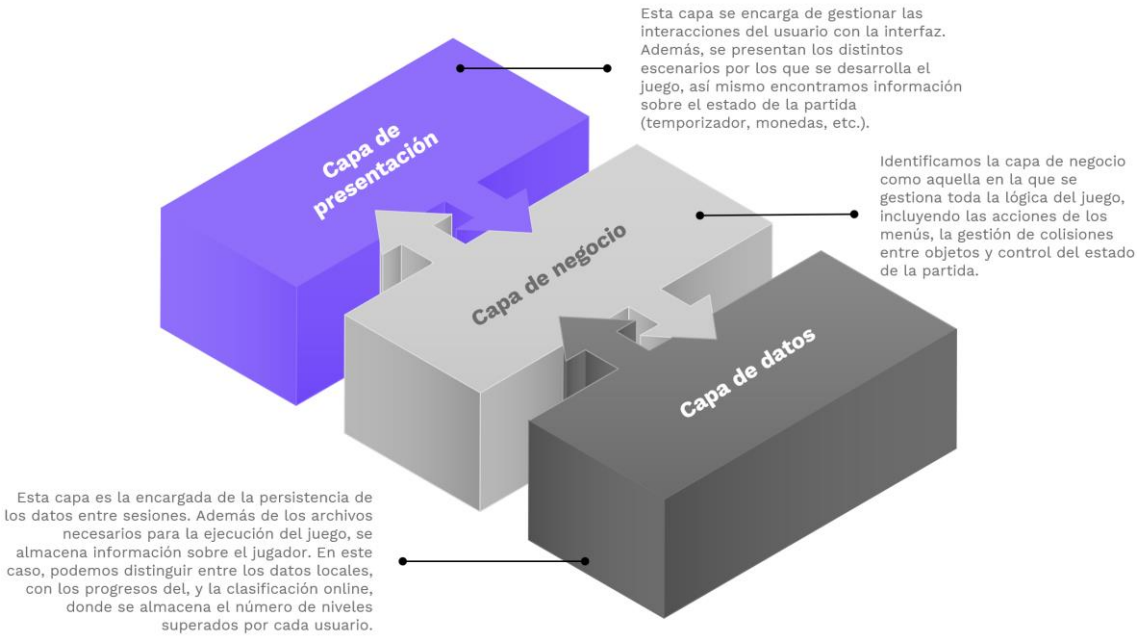


*Ilustración 5: Diagrama de arquitectura*

## 9.1. CAPA DE PRESENTACIÓN

En esta capa tratamos la interfaz gráfica de la aplicación, que intenta ser lo más intuitiva posible, además de tener un diseño de estilo retro que evoque un ambiente de aventura. Para toda la aplicación se ha utilizado una fuente de texto que combina muy bien con los diseños gráficos del juego.

### 9.1.1. Interfaces

En este juego tenemos distintas interfaces para representar de forma distinguida cada una de las funcionalidades. Aun así, se ha mantenido un diseño consistente a través de las pantallas qué hacen que sea intuitivo y agradable para el usuario.

En el siguiente diagrama se puede ver como es el flujo de navegación a través de las diferentes interfaces:

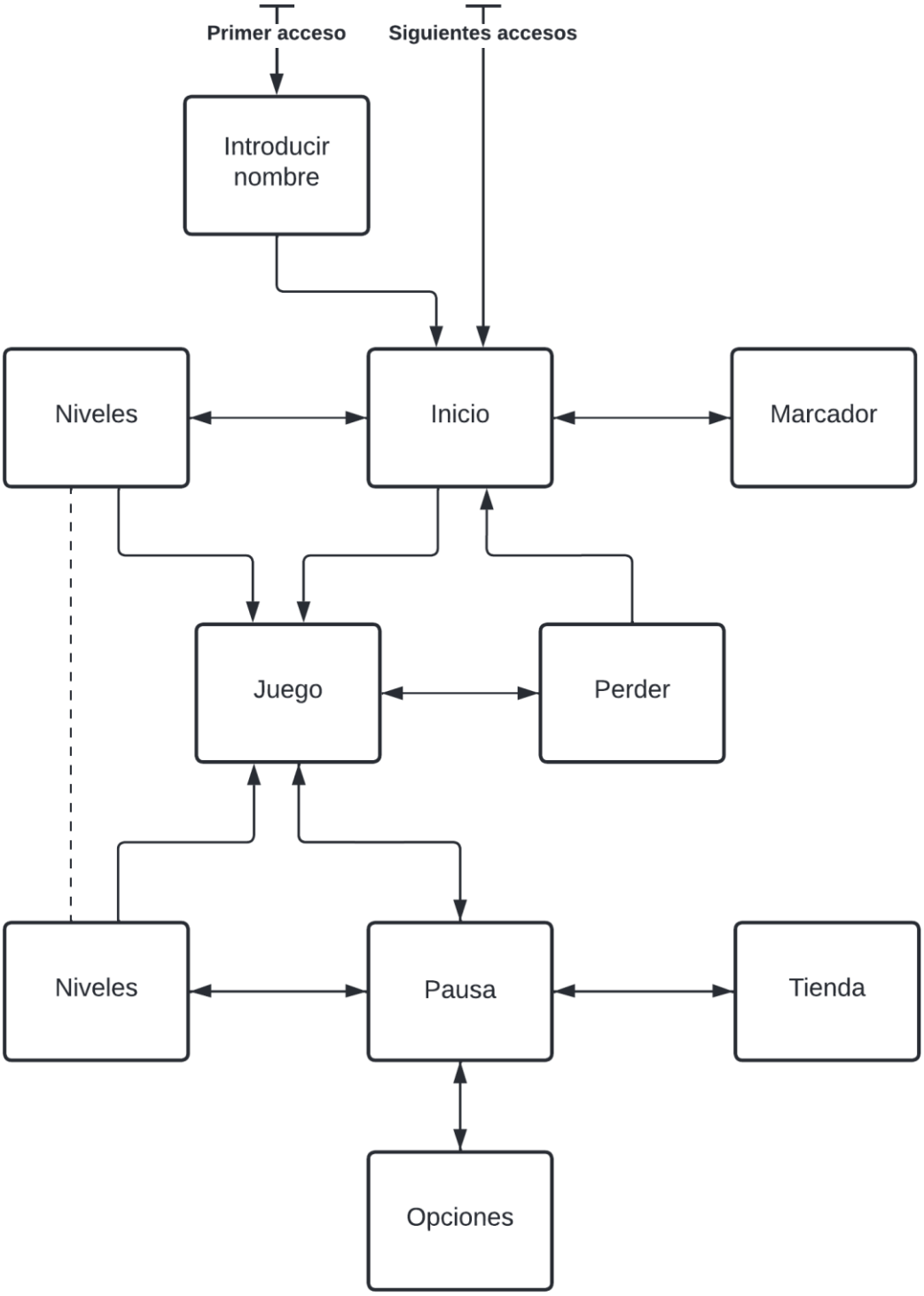


*Ilustración 6: Diagrama de flujo de interfaces*

#### 9.1.1.1. Inicio

Lo primero que encontraremos al ejecutar el videojuego será una pantalla que nos solicitará que ingresemos nuestro nombre. Una vez introducido nos llevará al menú principal, que nos dará acceso a varias funciones.

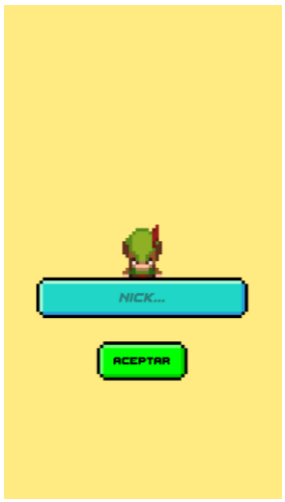



*Ilustración 7: Menú para registrar el usuario*

*Ilustración 8: Menú inicial*

#### 9.1.1.2. Niveles

En el botón “NIVELES”, seremos redirigidos a una pantalla que contiene un listado los niveles. Aparecen resaltados los niveles ya superados y da la opción de volverlos a jugar.


*Ilustración 9: Menú de niveles*

#### 9.1.1.3. Marcador

Desde el botón “MARCADOR”, tendremos acceso a un panel con una clasificación de los 3 jugadores que más niveles han superado.


*Ilustración 10: Menú del marcador*

#### 9.1.1.4. Interfaz de juego

Esta interfaz es la que se muestra mientras el usuario está jugando un nivel. Contiene información sobre las reglas del juego (temporizador, botella) y permite la movilidad del jugador haciendo uso del joystick.

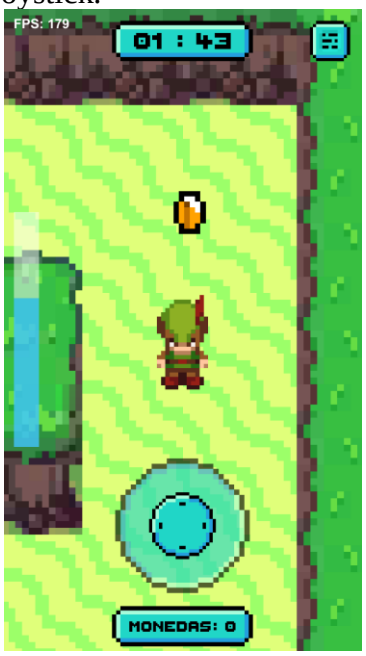


*Ilustración 11: Interfaz de juego*

El reparto de botones es el siguiente: el contador de FPS (esquina superior izquierda), el temporizador (centrado en la parte superior), el botón de pausa (esquina superior derecha), la cantidad de agua restante (lateral izquierdo), el joystick (mitad inferior) y la cantidad de monedas en el inventario (centrado en la parte inferior).

#### 9.1.1.5. Pausa

Desde el menú de pausa tenemos acceso a al menú de opciones, a la tienda de mejoras y, al igual que en el menú principal, al menú de niveles.


*Ilustración 12: Menú de pausa*

#### 9.1.1.6. Opciones

En el menú de opciones podemos activar y desactivar el contador de fotogramas por segundo. Esta configuración habilita y deshabilita un texto informativo en la esquina superior izquierda con el número de FPS.

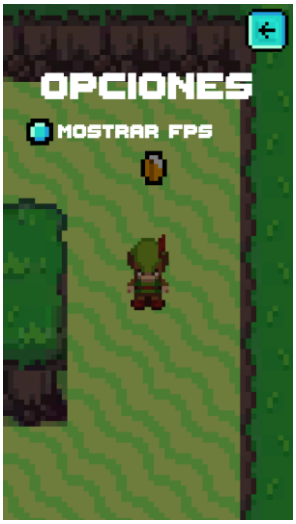


*Ilustración 13: Menú de opciones*

#### 9.1.1.7. Tienda

En este panel tenemos un listado con todos los atributos y herramientas que se pueden mejorar utilizando las monedas que recogemos del mapa. Los textos y botones se actualizan a medida que el usuario compra las mejoras.

Además, si el usuario no dispone de suficientes monedas para realizar una compra, el botón se marca de color rojo por un instante para indicar el error al comprar.


*Ilustración 14: Tienda de mejoras*

#### *9.1.1.8. Panel de perder*

Por último, tenemos una pantalla que se muestra cuando el jugador pierde, ya sea porque se ha agotado el tiempo disponible para superar el nivel o porque se termina el contenido de la botella de agua.


*Ilustración 15: Panel que se muestra al perder*

Presionando el botón azul que se muestra en la imagen, obtenemos una oportunidad extra para completar el nivel, mientras que el botón rojo reinicia el progreso que se haya realizado en el nivel en curso y genera un nuevo laberinto.

### 9.1.2. Animaciones

#### *9.1.2.1. Menús*

Para hacer que el uso de los menús sea más agradable, están configurados para que al cambiar entre las distintas pantallas haya una pequeña transición deslizando los paneles de izquierda a derecha o viceversa.


*Ilustración 16: Secuencia de imágenes de una transición entre menús*

También, hay otra transición diferente para cuando se carga un nivel. Comienza con un fundido hasta dejar la pantalla en negro por completo y, después, vuelve hacer un fundido hasta dejar una visibilidad completa.

#### *9.1.2.2. Personaje y objetos*

Tanto el personaje como los objetos del juego tienen animaciones [7] para que resulte más atractivo y cercano a la realidad.

En el caso del jugador tiene diferentes vistas [8] dependiendo de si está parado o va caminando hacia la derecha, izquierda, abajo o arriba. Además, dentro de cada una de estas vistas se simula el movimiento al andar.

*Ilustración 17: Secuencia de imágenes con el movimiento del personaje*

En el caso de los objetos tenemos diferentes animaciones. Por ejemplo, en el caso de las monedas se simula la rotación.

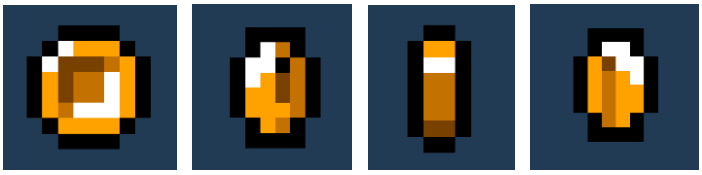

*Ilustración 18: Secuencia de imágenes con la animación de las monedas*

### 9.1.3. Niveles

Como bien hemos comentado anteriormente, se debe transmitir un ambiente de aventura. Por ello, la temática es la naturaleza donde las paredes de tierra limitan el desplazamiento del jugador, el agua está representado mediante charcos y las rocas son los obstáculos que el jugador debe destruir utilizando bombas.

*Ilustración 19: Elementos de los niveles*

El color verde claro que se ve en la imagen es la zona por la que el jugador puede desplazarse. En este terreno se encuentran diferentes objetos como las monedas, explosivos, el elemento que marca el comienzo de la partida y el objeto que identifica el final de cada nivel.

*Ilustración 20: Bomba generada en el nivel*

*Ilustración 21: Moneda generada en el nivel*

*Ilustración 22: Elemento final del nivel*

*Ilustración 23: Elemento del inicio de partida*

## 9.2. CAPA DE NEGOCIO

La capa de negocio engloba toda la lógica necesaria para el correcto funcionamiento de la aplicación. Esta lógica se encuentra dividida en distintos scripts [3], en el lenguaje de programación C#, donde cada uno de ellos se centra en una funcionalidad especifica.

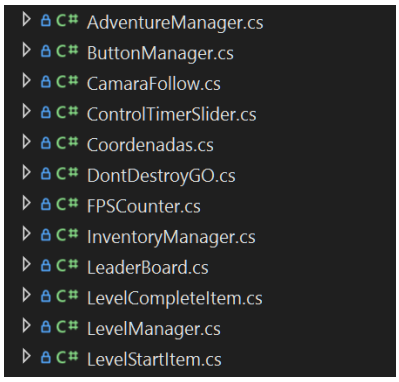


*Ilustración 24: Listado de clases. Parte 1 de 2*

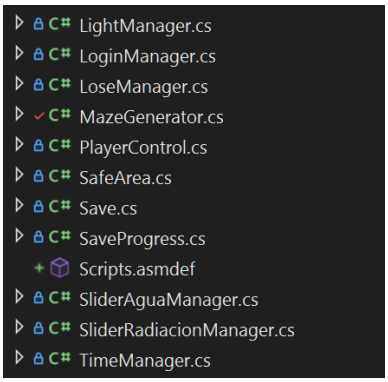


*Ilustración 25: Listado de clases. Parte 2 de 2*

En la siguiente tabla encontramos una breve descripción de la funcionalidad que ocupa cada uno de estos scripts.

### 9.2.1. Control de la partida

| **SCRIPT** | **DESCRIPCIÓN** |
|---|---|
| AdventureManager | Gestiona la carga de niveles y activa los elementos necesarios para el desarrollo de cada nivel. También contiene las dificultades qué habrá en cada nivel (configuración del labrinto, tiempo, etcétera). |
| TimerManager | Controla tiempo restante para superar el nivel. Mantiene actualizado este dato en la interfaz. |
| SliderAguaManager | Controla el gasto de agua del jugador. Mantiene actualizado este dato en la interfaz. |
| LevelStartItem | Al colisionar con el objeto que contenga este script, se activará el temporizador y el consumo de agua (si está activo en el nivel en curso). |
| LevelCompleteItem | Al colisionar con el objeto con este script se comenzará la carga del siguiente nivel. |
| MazeGenerator | Genera los laberintos que representan cada nivel y los dibuja gráficamente. |
| ControlTimerSlider | Pausa y activa el temporizador y el consumo de agua de forma simultánea. Se utiliza al pausar la partida. |

*Tabla 3: Scripts del control de la partida*

### 9.2.2. Control de los menús

| SCRIPT | DESCRIPCIÓN |
|---|---|
| ButtonManager | Mantiene actualizados los textos y botones de la tienda. Con cada compra del jugador se actualiza el coste de la siguiente mejora y el texto del nivel. |
| SafeArea | Establece la zona segura en la que se pueden representar los menús sin verse afectados por "Notch" del dispositivo móvil. |
| LoseManager | Gestiona el menú que se muestra cuando el usuario pierde. |
| FPSCounter | Calcula los fotogramas por segundo y los muestra en pantalla. |
| LevelManager | Gestiona los botones que aparecen en el menú de niveles. Habilita los botones a medida que se van superando los niveles. |

*Tabla 4: Scripts del control de menús*

### 9.2.3. Control del jugador

| SCRIPT | DESCRIPCIÓN |
|---|---|
| PlayerControl | Controla el movimiento y velocidad del jugador. |
| LigthManager | Controla la intensidad y el alcance de la linterna |

*Tabla 5: Scripts del control del jugador*

### 9.2.4. Control de datos

| SCRIPT | DESCRIPCIÓN |
|---|---|
| Save | Unifica todos los elementos que deben de guardar su estado al salir del juego. |
| SaveProgress | Contiene métodos estáticos para guardar la información en el sistema de almacenamiento. |
| LoginManager | Se encarga de solicitar el nombre del usuario si es la primera vez que accede a la aplicación. |
| InventoryManager | Gestiona los objetos recogidos por el jugador. También, gestiona los atributos y herramientas que se mejoran desde la tienda. |
| LeaderBoard | Gestiona la tabla de clasificación online. |
| Coordenadas | Esta propiedad se añade a cada objeto del laberinto para conocer su ubicación. |

*Tabla 6: Scripts del control de datos*

### 9.2.5. Diagrama de clases

A continuación, se muestra un diagrama UML con las clases principales [1]. Se pueden destacar 4 clases como las más importantes:

- AdventureManager
- InventoryManager
- PlayerControl
- MazeGenerator

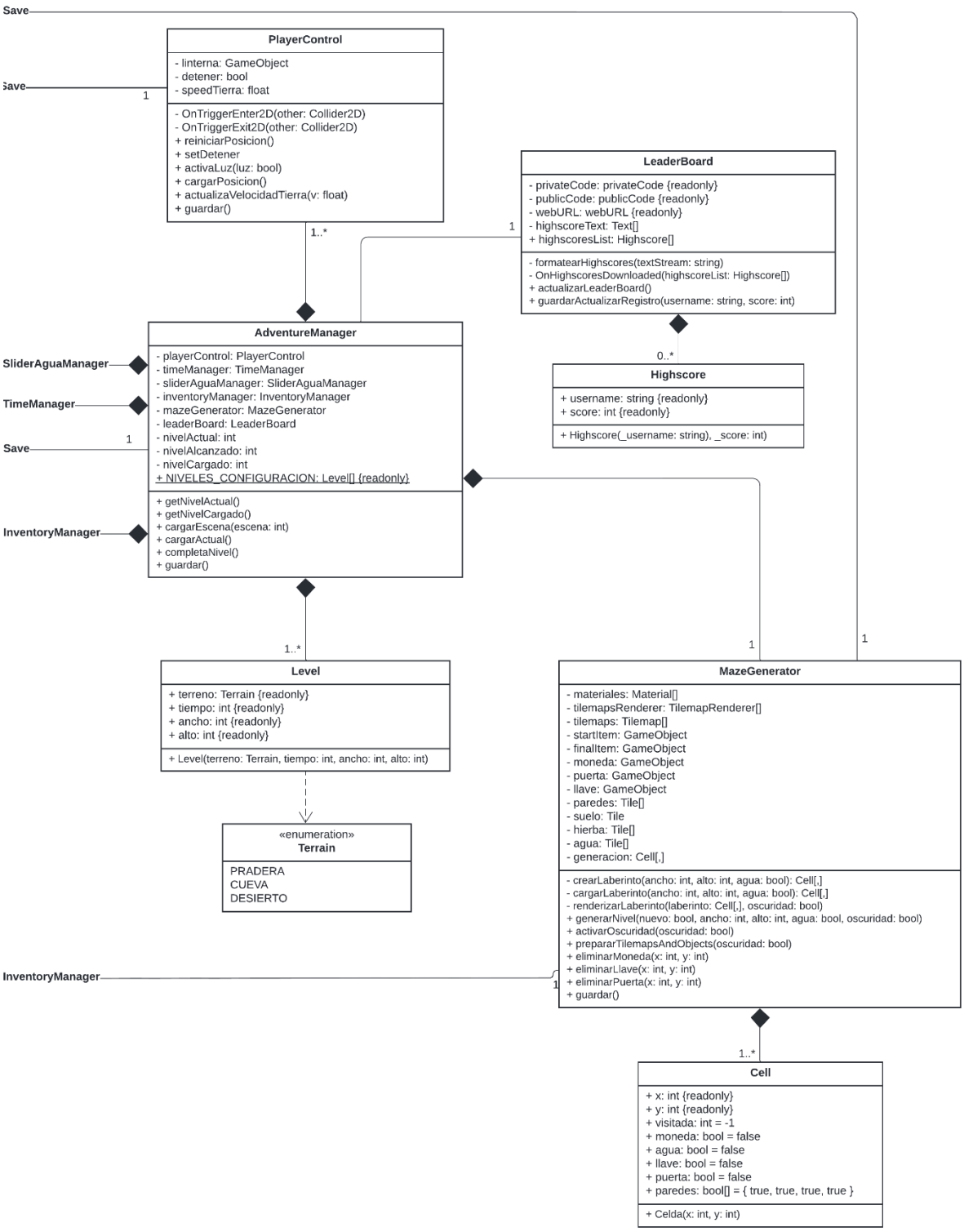


*Ilustración 26: Diagrama de clases. Parte 1 de 2*

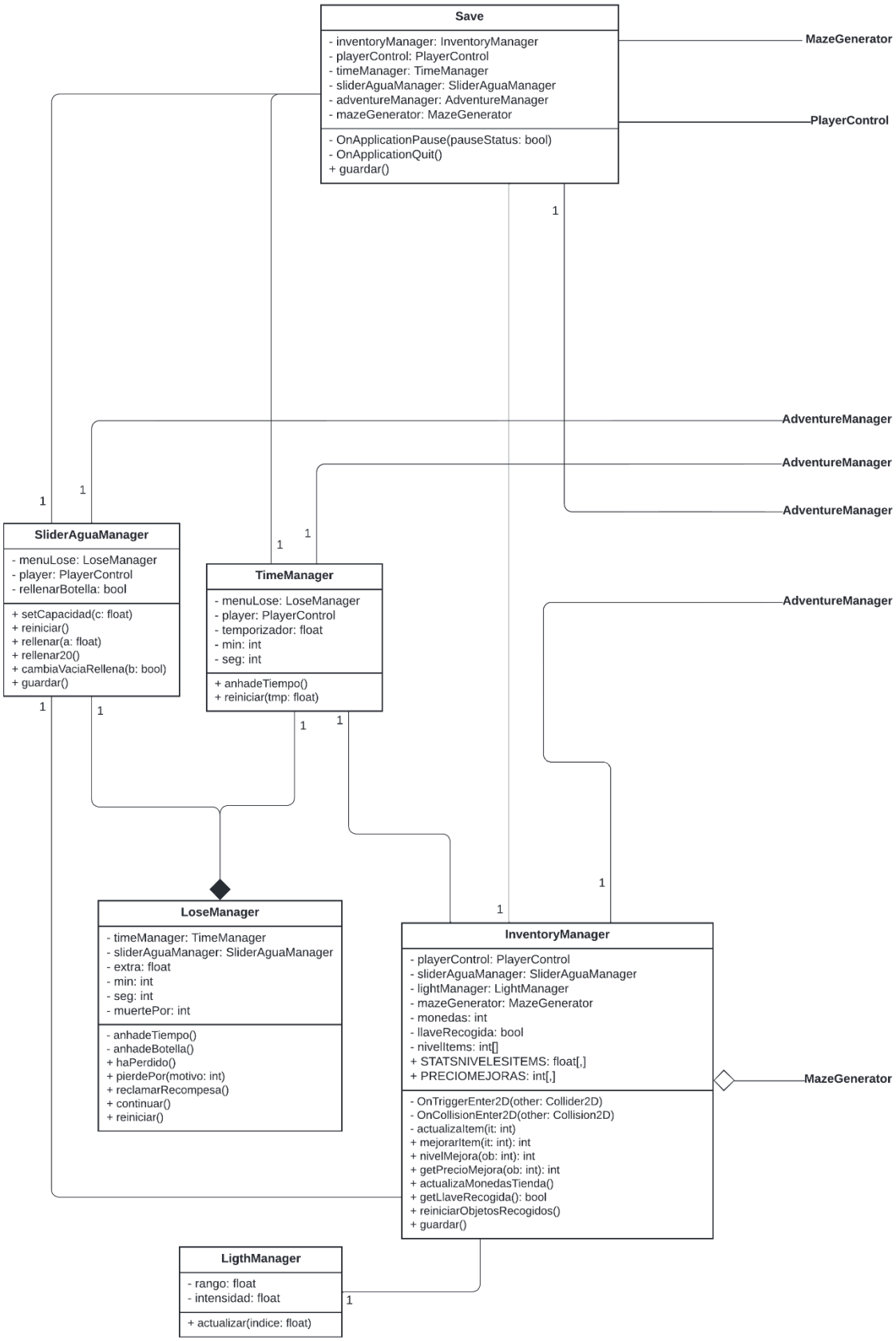


*Ilustración 27: Diagrama de clases. Parte 2 de 2*

## 9.3. CAPA DE DATOS

### 9.3.1. Datos locales

Tenemos cierta información que se modifica a medida que el usuario va jugando. Estos cambios deben preservarse para que jugador pueda continuar desde el punto en que dejó su partida en la última sesión, incluyendo todas las copras que haya realizado en la tienda y objetos que haya recogido.

Haciendo uso de la clase *PlayerPrefs*, proporcionada por Unity, podemos guardar datos de tipo entero, decimal o texto. A continuación, un ejemplo de cómo se lee y guarda información el nombre del usuario sobre la memoria del dispositivo.

```
0 references
public static void guardarNombre(string m) {
    PlayerPrefs.SetString("nombre",m);
    PlayerPrefs.Save();
}

0 references
public static string cargarNombre() {
    return PlayerPrefs.GetString("nombre","");
}
```

*Ilustración 28: Código de ejemplo para guardar datos*

Este es el listado completo con toda la información que se guarda:

- Nombre del usuario.
- Estado de la partida (comenzada o no comenzada).
- Niveles de los atributos y herramientas que se mejoran desde la tienda.
- Llave recogida en el nivel.
- Cantidad de monedas del usuario.
- Tiempo restante del temporizador y tiempo total jugado.
- Cantidad restante en la botella de agua.
- Posición X e Y del jugador.
- Nivel en curso y nivel máximo alcanzado.
- Una cadena de texto que representa la generación del laberinto actual.

### 9.3.2. Clasificación online

Utilizando la plataforma de *Dreamlo*, se guarda una clasificación online con el número de niveles superados por cada jugador. Esta clasificación se comparte con el resto de la comunidad a través del panel que contiene el marcador.

Haciendo uso del módulo "*WWW*" de Unity podemos recuperar contenido de una petición *HTTP*. La URL en cuestión se genera partiendo de una clave privada que se genera en *Dreamlo*.

Para crear o actualizar un registro, se concatena el nombre identificativo y el dato de la clasificación, en este caso, el nombre del usuario y el nivel máximo alcanzado.

```
WWW www = new WWW(webURL + privateCode + "/add/" + WWW.EscapeURL(username) + "/" + score);
yield return www;
```

*Ilustración 29: Código para añadir un dato a la clasificación online*

Por otro lado, tenemos la posibilidad de hacer una descarga completa de la clasificación utilizando la siguiente instrucción.

```
WWW www = new WWW(webURL + publicCode + "/pipe/");
yield return www;
```

*Ilustración 30: Código para descargar la clasificación online*

# 10. ALGORITMO DE GENERACIÓN

La selección del algoritmo de generación tuvo varios candidatos con implementaciones y resultados muy diferentes. Algunos generan salas aisladas conectadas por caminos y otros, como el elegido en este videojuego, generan multitud de pasillos en todas direcciones con distintas longitudes. Estamos hablando del algoritmo de búsqueda en profundidad, uno de los que hemos estudiado en algunas asignaturas de la carrera.

## 10.1. ALGORITMO DE BÚSQUEDA ALEATORIA EN PROFUNDIDAD

El algoritmo de búsqueda en profundidad [10] es un algoritmo de búsqueda no informada utilizado para recorrer los nodos de un grafo de forma ordenada. La especificación de búsqueda no informada nos indica que este algoritmo no distingue si un camino recorrido es mejor o peor que otro para llegar desde un nodo A hasta un nodo B.

Para llevar este algoritmo a la generación de laberintos, representamos el laberinto como si fuesen celdas, y cada una de estas representa un nodo conectado con sus celdas adyacentes (izquierda, derecha, superior e inferior).

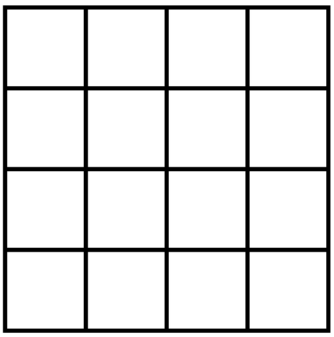

*Ilustración 31: Cuadricula del laberinto*

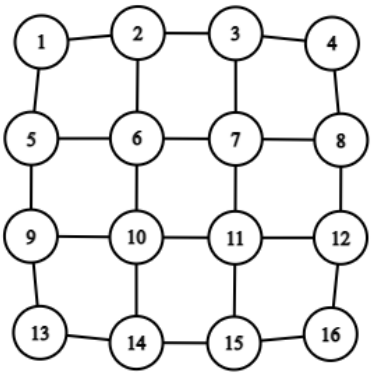


*Ilustración 32: Representación del laberinto en nodos*

Al comenzar la ejecución del algoritmo, se marcan todos los nodos como no visitados y se crea una pila vacía. El primer paso es seleccionar un nodo inicial de forma aleatoria y añadirlo a la pila una vez marcado como visitado. En cada iteración del algoritmo, se comprueba si existe un nodo adyacente sin visitar.

En caso de que exista, se marca como visitado y se añade a la pila para repetir el proceso con este nuevo nodo.

En caso de no haber nodos adyacentes sin visitar, se elimina el nodo actual de la pila y se repite el proceso con el primer elemento de la pila, es decir, el nodo anterior.

Llegará un momento que hayamos marcado como visitados todos los nodos y la pila este vacía. En este punto el algoritmo ha terminado su ejecución y tendremos el laberinto listo.

Todo esto está representado en el siguiente diagrama de flujo, en el cual podemos ver como avanza el algoritmo en su ejecución.

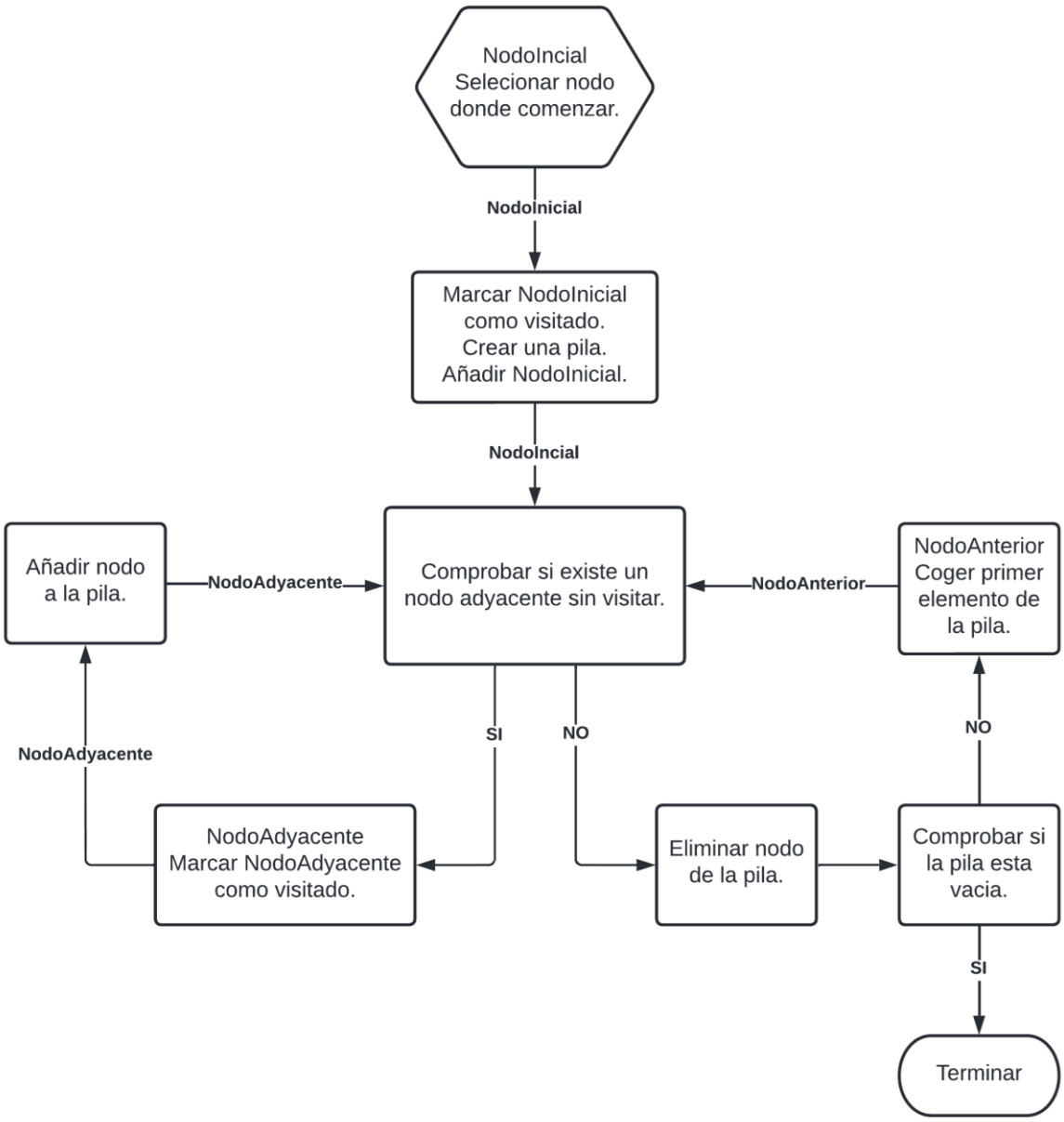


*Ilustración 33: Diagrama de flujo del algoritmo*

## 10.2. ADAPTACIONES

En la implementación del algoritmo se ha determinado que el nodo inicial siempre será el de la esquina inferior izquierda y el final el de la esquina superior derecha.

Además, cada nodo tiene cuatro paredes que se ocultan en función de qué dirección se tome al ir hacia un nodo adyacente. En el caso de que hagamos una transición a la derecha, se ocultará la pared derecha del nodo actual y la pared izquierda del nodo de destino.

### 10.2.1. Generar piedra y bomba

Uno de los requisitos es que algunos mapas contengan un obstáculo entre el inicio y final de cada nivel. Para esto, ejecutamos el algoritmo hasta que alcanzamos el nodo situado en la esquina superior derecha, en este punto de la ejecución guardamos una copia del estado de la pila. En esta copia están almacenados de forma ordenada los nodos que hay que visitar para llegar desde el inicio hasta el final.

Una vez que ha finalizado el algoritmo por completo, generamos el obstáculo, en este caso representado por una roca, en cualquier nodo que esté contenido en la copia que hicimos durante la ejecución.

También ha sido necesario modificar la propiedad de visitado pasando de ser un booleano a un número que se incrementa a medida que se van visitando cada una de las celdas. Con esto, conseguimos poder generar el objeto necesario para superar el obstáculo, en este caso una pequeña bomba, en una posición que se encuentre antes del obstáculo. Simplemente basta con fijarse si se ha situado en una celda cuyo índice de visitada sea menor que el del obstáculo.

A continuación, se muestra un ejemplo donde el punto inicial está marcado de verde, el final de rojo, la ubicación de la bomba de azul y la roca en el amarillo.

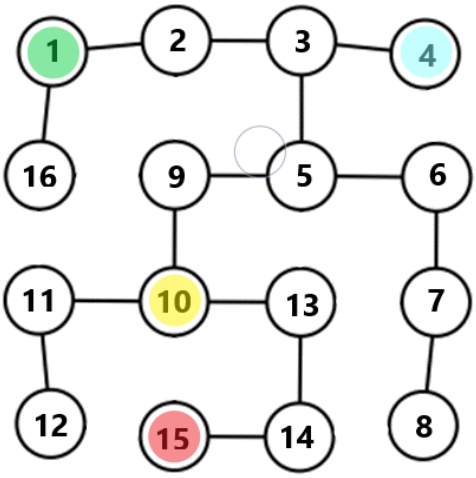


*Ilustración 34: Representación del laberinto en nodos tras la ejecución del algoritmo*

Dentro de cada nodo representado en la imagen está el número que representa en qué orden se ha recorrido. Cómo podemos ver, la bomba está situada en la posición cuatro que es accesible para jugador antes de superar el obstáculo de la posición 10.

### 10.2.2. Generar puntos con agua

Otro de los requisitos es que el usuario tuviese una botella de agua, que se fuese consumiendo, y pudiese rellenarse en algunos puntos situados por el mapa. Estos puntos de agua se generan de manera aleatoria en los nodos a medida que se van visitando.

### 10.2.3. Guardar elementos recogidos

Finalmente, a cada uno de los nodos se le han añadido varias propiedades para identificar si contiene una moneda, una roca, una bomba o agua. Cada vez que se recoge uno de estos elementos, a excepción del agua, el nodo se actualiza para marcar que ya no contiene el objeto recogido.

De esta forma se mantiene actualizado el estado del laberinto y se guarda en memoria en caso de que se cierre la aplicación. Así en la aproxima sesión, encontrara el laberinto en el mismo estado en que lo dejo.

# 11. PRUEBAS

## 11.1. UNITARIAS

El objetivo de estas pruebas es verificar el correcto funcionamiento de fragmentos de código de forma independiente. En Unity, encontramos la herramienta *Test Runner* con el modo de ejecución "*Edit mode*" para las pruebas unitarias.

Por la naturaleza de los videojuegos, es difícil encontrar métodos o componentes que funcionen sin interactuar con otros elementos. Por este motivo, la mayoría de las implementaciones de han probado mostrando resultados por consola durante la ejecución del juego.

La única pieza que funciona de forma independiente es el algoritmo de generación del laberinto. Este algoritmo es el corazón del juego y ha sido muy importante su depuración para obtener los resultados esperados.

La primera prueba lo único que verifica es que el tamaño de la matriz se corresponda con el indicado en los parámetros del método de generación.

```
[Test]
0 referencias
public void MazeSizeTest()
{
    GameObject gameObject = new GameObject();
    MazeGenerator mazeGenerator = gameObject.AddComponent<MazeGenerator>();
    Celda[,] generacion = mazeGenerator.crearLaberinto(DimensionsTest1[0], DimensionsTest1[1], false);

    Assert.AreEqual(DimensionsTest1[0], generacion.GetLength(0));
    Assert.AreEqual(DimensionsTest1[1], generacion.GetLength(1));
}
```

*Ilustración 35: Prueba de tamaño del laberinto*

Esta segunda prueba ha sido una de las más importantes para resolver algunos de los bugs, ya que se encarda de verificar que el personaje pueda recorrer el laberinto sin encontrarse paredes en lugares inesperados impidiendo el paso.

```
[Test]
0 referencias
public void MazeWallsTest()
{
    GameObject gameObject = new GameObject();
    MazeGenerator mazeGenerator = gameObject.AddComponent<MazeGenerator>();
    Celda[,] generacion = mazeGenerator.crearLaberinto(DimensionsTest2[0], DimensionsTest2[1], false);

    for (int x = 0; x < DimensionsTest2[0]; x++)
    {
        for (int y = 0; y < DimensionsTest2[1]; y++)
        {
            if (y < DimensionsTest2[1] - 1) Assert.AreEqual(generacion[x, y].paredes[0], generacion[x, y + 1].paredes[2]);
            if (x < DimensionsTest2[0] - 1) Assert.AreEqual(generacion[x, y].paredes[1], generacion[x + 1, y].paredes[3]);
            if (y > 0) Assert.AreEqual(generacion[x, y].paredes[2], generacion[x, y - 1].paredes[0]);
            if (x > 0) Assert.AreEqual(generacion[x, y].paredes[3], generacion[x - 1, y].paredes[1]);
        }
    }
}
```

*Ilustración 36: Prueba de generación de las paredes del laberinto*

También, podemos introducir, como parámetro del método, la posibilidad de que se generen puntos de agua en ubicaciones aleatorias del mapa. Simplemente se ha probado que en función de este parámetro se generan dichos puntos o no.

```
[Test]
0 referencias
public void MazeWaterTest()
{
    GameObject gameObject = new GameObject();
    MazeGenerator mazeGenerator = gameObject.AddComponent<MazeGenerator>();

    Celda[,] generacion1 = mazeGenerator.crearLaberinto(DimensionsTest3[0], DimensionsTest3[1], true);
    bool aguaEncontrada1 = false;
    foreach (Celda celda in generacion1)
    {
        if (celda.agua) aguaEncontrada1 = true;
    }
    Assert.True(aguaEncontrada1);

    Celda[,] generacion2 = mazeGenerator.crearLaberinto(DimensionsTest3[0], DimensionsTest3[1], false);
    bool aguaEncontrada2 = false;
    foreach (Celda celda in generacion2)
    {
        if (celda.agua) aguaEncontrada2 = true;
    }
    Assert.False(aguaEncontrada2);
}
```

*Ilustración 37: Prueba de aparición de puntos de agua en el laberinto*

Y, por último, se revisa si la ubicación de la bomba es accesible antes de pasar por la piedra, ya que de no ser así sería imposible completar el nivel. Además, se comprueba que se ha generado alguna moneda.

```
[Test]
0 referencias
public void MazeElementsTest()
{
    GameObject gameObject = new GameObject();
    MazeGenerator mazeGenerator = gameObject.AddComponent<MazeGenerator>();
    Celda[,] generacion = mazeGenerator.crearLaberinto(DimensionsTest4[0], DimensionsTest4[1], true);

    int indicePuerta = -1, indiceLlave = -1, monedas = 0;

    foreach (Celda celda in generacion)
    {
        if (celda.moneda) monedas++;
        if (celda.puerta) indicePuerta = celda.visitada;
        if (celda.llave) indiceLlave = celda.visitada;
    }

    Assert.Greater(indicePuerta, indiceLlave);
    Assert.Greater(monedas, 0);
}
```

*Ilustración 38: Prueba de generación de objetos en el laberinto*

## 11.2. DE INTEGRACIÓN

Haciendo uso de las pruebas de integración podemos probar módulos o funcionalidades completas que no necesariamente tienen que interactuar con un único elemento.

A pesar de que Unity permite estas pruebas con la herramienta *Test Runner* en el modo "*Play mode*", su implementación es algo compleja. Por ello, se ha optado por la comprobación a través de la ejecución de flujos definidos y la comprobación de resultado a través de mensajes por consola.

Como ejemplo tenemos el diagrama para las pruebas de la tienda de mejoras. Su ejecución comienza en la interfaz de control del personaje., que pasa al menú de pausa y desde ahí a la tienda.

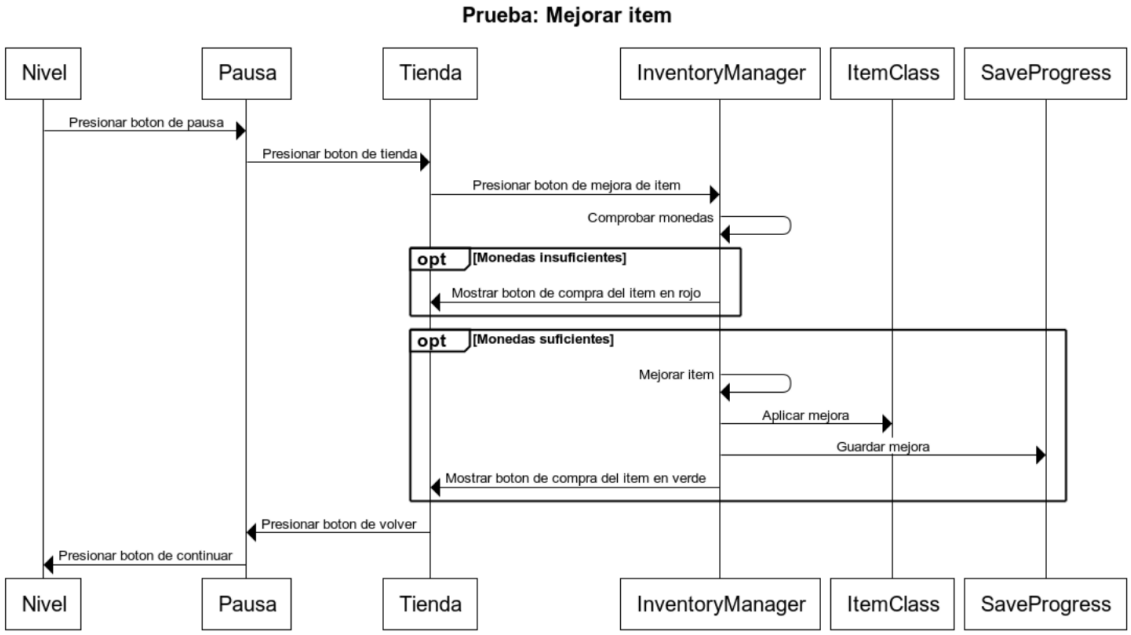


*Ilustración 39: Flujo de pruebas de la tienda*

En la tienda tenemos un botón diferente por cada herramienta que se puede mejorar, con un coste asociado. Al presionar el botón se comprueba en la clase "*InventoryManager*" si dispone de monedas suficientes para comprarlo.

En caso de no tener monedas suficientes, el botón se muestra en rojo durante un segundo para indicar el error en la compra. Una vez hay, se puede reintentar o volver a la pantalla de juego.

En caso de tener monedas, se mejora la herramienta, se aplica la mejora sobre la clase que controla dicha herramienta y se guarda en memoria el progreso. Por último, el botón de la compra se muestra en verde para indicar la compra. En este punto, se pude comprar otra mejora o volver al juego.

## 11.3. USABILIDAD

En cuanto a usabilidad tenemos como objetivo principal tener una interfaz intuitiva y de respuesta rápida. Para conseguir una interfaz intuitiva se ha utilizado un diseño uniforme a lo largo de todos los menús con mensajes claros de cuál es la función que desempeña cada botón o menú. También, se ha mantenido una estructura en cuanto a la disposición de los botones que hacen que el usuario sepa de manera intuitiva dónde deberían de estar dependiendo de la funcionalidad que busque.

Además, esto se ha combinado con unas transiciones entre menús con una duración de 1/4 de segundo que dan cierta fluidez a la navegación. Además, el tiempo de carga de un nivel se demora escasos 3 segundos, que es un tiempo más que aceptable teniendo en cuenta que los niveles se generan y renderizan al presionar el botón sin necesidad de una precarga previa.

### 11.4. PORTABILIDAD

Tenemos dos requisitos haciendo referencia a la portabilidad de la aplicación. La primera contempla que el juego será ejecutable en cualquier dispositivo con una versión de Android 4.4 o superior y la segunda tiene en cuenta si el dispositivo cuenta con "notch" para evitar renderizado contenido relevante en ese espacio.

Estas pruebas se han realizado utilizando dispositivos con diferentes versiones de Android, algunos sin "notch" y otros con él. El resultado ha sido el esperado dando por superados estos requisitos.

### 11.5. RENDIMIENTO

El requisito principal en esta sección es que el juego tenga una tasa de refresco de al menos 30 fotogramas por segundo.

Tras el desarrollo estuve haciendo algunas pruebas y aunque duplicaban este requisito, los fotogramas por segundo no eran estables. Investigando un poco descubrí que se puede modificar los métodos de compresión cuando se compila el proyecto consiguiendo que la aplicación tuviese una tasa de 60 fps estables.

### 11.6. ACEPTACIÓN

Una de las partes más importantes siempre es recibir feedback del usuario final. Cómo no podía ser de otra forma, he utilizado a mis amigos para probar la aplicación y den una crítica sobre su experiencia.

En general, ha tenido bastante éxito el aspecto visual de la aplicación y la jugabilidad de los niveles.

Por otro lado, han reclamado la falta de una trama principal que genere un ambiente aventurero y de un objetivo por el que merezca la pena jugar. Además, por falta de tiempo no se han incluido sonidos que te envuelvan en lo que le está sucediendo al personaje.

# 12. CONCLUSIONES Y FUTUROS TRABAJOS

## 12.1. CONCLUSIONES

El objetivo del proyecto era desarrollar un videojuego teniendo en cuenta unos requisitos detallados y gestionar las tareas a ejecutar. También, se debía de tener en cuenta el aspecto visual para que al usuario le resultase interesante e intuitivo.

Llegados a este punto en el que me toca valorar el trabajo descrito en este documento estoy muy contento con el resultado obtenido, ya que he gestionado bastante bien el desarrollo y el resultado final es una versión jugable.

Me gustaría destacar la capacidad de resolución de los problemas que se han planteado a lo largo del desarrollo, especialmente en el algoritmo que genera los niveles. Este algoritmo ha sido adaptado desde una versión en que simplemente se iban visitando nodos de una matriz hasta una versión que se traslada al entorno gráfico y tiene en cuenta la generación de obstáculos y otros objetos.

He mejorado muchos aspectos en el ámbito personal como la disciplina, ya que ha sido un proyecto de largo recorrido, y la gestión de mi tiempo para cumplir en el entorno laboral y desarrollar este proyecto. También, he mejorado mucho mis conocimientos de programación, gestión de proyectos, resolución de problemas y metodologías de desarrollo.

## 12.2. FUTUROS TRABAJOS

Una de mis metas desde el comienzo de la carrera siempre ha sido desarrollar un videojuego por mí mismo, que con este proyecto queda zanjada. Por supuesto que hay muchas cosas que deben de mejorarse y algunas ideas que se han descartado por falta de tiempo.

Mi próximo reto será terminar este videojuego añadiéndole sonidos y animaciones que hagan el juegos mucho más estimulante y atractivo para el usuario. Una vez se completen todos los evolutivos, se contempla la publicación en la plataforma de Play Store para poder recibir feedback de usuarios y mejorar aquellas partes que reclamen.

Desde luego que esto no queda ahí, tengo un pequeño montón de papeles con varias ideas de sobre diferentes temas.

El tiempo dirá sí es mi vocación desarrollar por mi cuenta o continuar con el trabajo y llegar lo más lejos posible.

# 13. REFERENCIAS


[1] Arlow, J., & Neustadt, I. (2005). *UML 2 And The Unified Process: Practical Object-Oriented Analysis And Design* (2.ª ed.). Addison-Wesley Professional.

[2] Jalote, P. (2012). *An Integrated Approach to Software Engineering*. Springer Publishing.

[3] Technologies, U. (s. f.). *Unity - Manual: Scripting*. Unity Documentation. https://docs.unity3d.com/Manual/ScriptingSection.html

[4] Technologies, U. (s. f.). *Unity - Scripting API: MonoBehaviour*. Unity Documentation. https://docs.unity3d.com/ScriptReference/MonoBehaviour.html

[5] Technologies, U. (2017). *Unity - Manual: GameObjects*. Unity Documentation. https://docs.unity3d.com/Manual/GameObjects.html

[6] Technologies, U. (2017). *Unity - Manual: Tilemap*. Unity Documentation. https://docs.unity3d.com/Manual/class-Tilemap.html

[7] Technologies, U. (201). *Unity - Manual: Animation System Overview*. Unity Documentation. https://docs.unity3d.com/Manual/AnimationOverview.html

[8] Technologies, U. (2018). Unity - Manual: Sprites. Unity Documentation. https://docs.unity3d.com/Manual/Sprites.html

[9] Watkins, R. (2016). Procedural Content Generation for Unity Game Development. Van Haren Publishing.

[10] Gabrovšek, P. (2019). Analysis of maze generating algorithms. IPSI Transactions on Internet Research, 15(1), 23-30.