Facultad
De
Ciencias

DESARROLLO DE UN VIDEOJUEGO ARCADE CLÁSICO
(Development of a classic arcade video game)

Trabajo de Fin de Grado
para acceder al

Grado en Ingeniería Informática

Autor: Javier García Magaldi

Director: Carlos Blanco Bueno

Marzo-2021

Agradecimientos

En este punto me gustaría agradecer a todas las personas que me han acompañado durante estos años en la Universidad.
Primero, se lo agradezco a toda mi familia por su constante apoyo, comprensión en los momentos bajos y toda la confianza depositada en mi desde que entre el primer día.
Además de mis amigos de toda la vida, los cuales me han animado y soportado como nadie durante este tiempo, así como los amigos nuevos que he hecho durante estos años, con los que he aguantado tanta tensión en muchos momentos y pasado tanto tiempo juntos en la facultad.
Por último, pero no menos importante, a los profesores que tanto me han ayudado a terminar todo este camino y que se han preocupado por todo, sobre todo a mi tutor Carlos, por ayudarme con este último empujón en una situación como la actual.

Índice de contenido

Índice de ilustraciones

Índice de tablas

Resumen

Los videojuegos clásicos son una fuente de horas de entretenimiento, basándose en mecánicas y controles muy sencillos. Aunque actualmente los jugadores busquen videojuegos con unos gráficos lo más realistas posibles e historias con tramas complejas, también los videojuegos arcade clásicos están volviendo a tener fuerza en el mercado para aquellos jugadores ocasionales que buscan un simple entretenimiento. Esto implica adaptar dichos juegos a los motores actuales, y en ciertas ocasiones, implementar algunas nuevas funcionalidades, así como mejorar la parte visual para modernizarlos con el fin de adaptarlos a los juegos de la actualidad.

El proyecto tiene como finalidad conseguir esto con el videojuego Pang, un videojuego clásico con una vista lateral en 2D. La dinámica del juego consiste en que un personaje se encuentra en un escenario con unas bolas que rebotan y pueden matarlo, las cuales tiene que explotar mediante disparos, sobreviviendo así a las mismas. Para ello, el personaje cuenta con la ayuda de ciertos elementos que potencian algunas estadísticas o modifican el comportamiento de las bolas. En este proyecto se respetará la idea básica del videojuego, modernizándolo con mejores gráficos, nuevas mecánicas y una clasificación online, programándolo completamente en un motor actual.

El proyecto se realiza con el motor Unity, ya que proporciona una gran capacidad de desarrollo multiplataforma y herramientas. También se utiliza el servicio ofrecido por *Dreamlo* para almacenar las puntuaciones a nivel global y Photoshop para la creación y modificación de los aspectos.

Palabras clave: Videojuego, clásico, Unity, entretenimiento, funcionalidad, multiplataforma, Dreamlo.

Abstract

Classic videogames are a great source of entertainment, based on very simple mechanics and controls. Although players are currently looking for videogames with graphics as realistic as possible and stories with complex plots, classic arcade videogames are also getting strength in the market for those more casual players looking for simple entertainment. This involves adapting these games to current engines and sometimes implementing some new functionalities as well as improving the visual part to modernize them a little.

This project aims to achieve all these goals with the videogame Pang, a classic videogame with a 2D side view in which a character is on a stage where they need to explode some balls by shooting and survive by getting help from certain elements which enhance some statistics or modify the behavior of the balls. Therefore, the basic idea of the game will be respected, modernizing it by adding better graphics, new mechanics and an online ranking, programming it completely into a current engine.

The project was done with the Unity engine, since it provides a great capacity for cross-platform development and tools, the service offered by *Dreamlo* to store the scores globally and Photoshop for the aspects' creation and modification.

Key words: Videogame, classic, Unity, entertainment, functionality, multiplatform, Dreamlo.

1. Introducción

Dentro de la ingeniería informática existe una rama dedicada al desarrollo de videojuegos. Este es un tipo de desarrollo software similar al de otros proyectos, como puede ser los servicios de planificación de recursos empresariales, los sistemas de gestión de contenidos o los de administración de relaciones con el cliente. En este tipo de proyectos, al igual que en los otros mencionados, es necesaria una gestión de la configuración, análisis de requisitos, planificación, pruebas, modelos, etc.
La industria de los videojuegos tiene un peso muy importante dentro de la economía. En el año 2018, se facturaron 813 millones de euros solamente en España en este sector. También es una buena fuente de empleo, donde se incrementó el número de trabajadores hasta llegar a los 6.900 profesionales y se prevé un crecimiento aún mayor [1]. Todo esto es debido a la alta demanda de entretenimiento de hoy en día, donde, por ejemplo, juegos como el Minecraft han llegado a vender 200 millones de unidades y reúne cada mes alrededor de 125 millones de jugadores [2].
Este impacto en los mercados ha llegado a afectar a otros ámbitos como puede ser el del deporte con los *e-sports*.

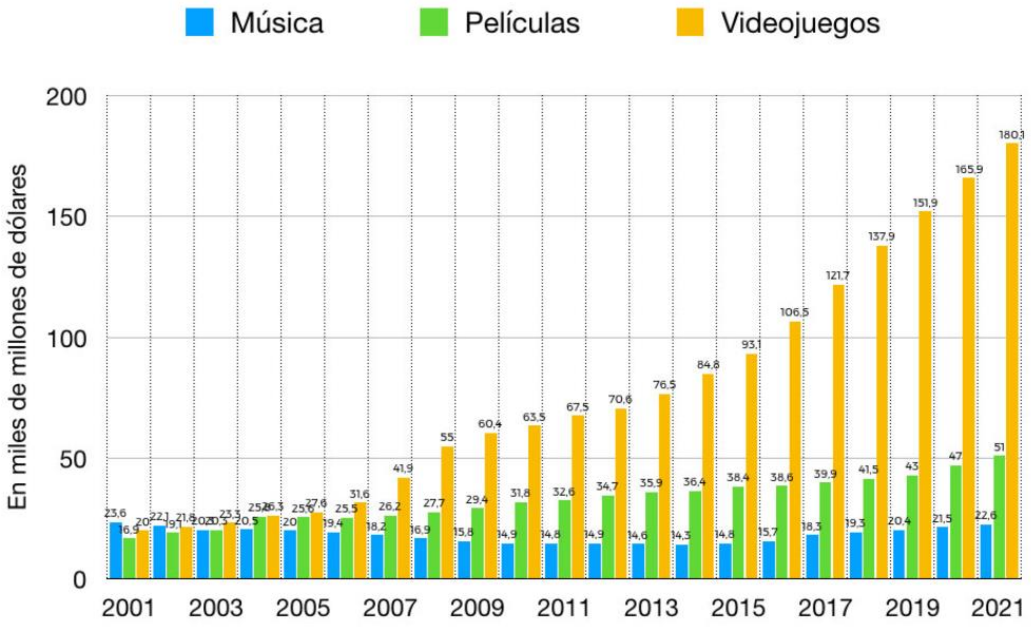

Figura 1 - Comparativa del mercado de los videojuegos frente al de música o cine. Fuente [3]

Los videojuegos arcades clásicos, como pueden ser el Tetris, el Pacman o el Pang, son videojuegos que suelen atraer a jugadores más casuales. Dichos videojuegos están volviendo a tener un gran auge en el mercado, ya sea rehaciendo viejos clásicos o creando nuevos, pero siguiendo una misma estética e idea.

Este proyecto se centrará en el videojuego Pang, ya que es uno de esos videojuegos clásicos que tengo especial cariño, debido a que en mi infancia pasé una gran parte de mi tiempo jugando con él. Por lo tanto, he querido aprovechar esta oportunidad para conocer dicho juego en profundidad, desarrollarlo completamente y a la vez, modernizarlo añadiéndole nuevas funcionalidades.

El videojuego Pang fue creado en 1989 por Mitchell Corporation y publicado por Capcom [4]. A continuación, se muestra una figura del juego Pang original (Figura 2). Brevemente, podemos describirlo como un juego 2d de vista lateral, en el que el jugador maneja a un personaje con acciones de movimiento a la derecha o izquierda y de disparo. Se le presentan escenarios en los que aparecen unas bolas gigantes que botan por la pantalla, siendo el objetivo esquivarlas y destruirlas. El juego clásico así como las mejoras a incorporar, serán descritos en más detalle en el capítulo de análisis de requisitos.

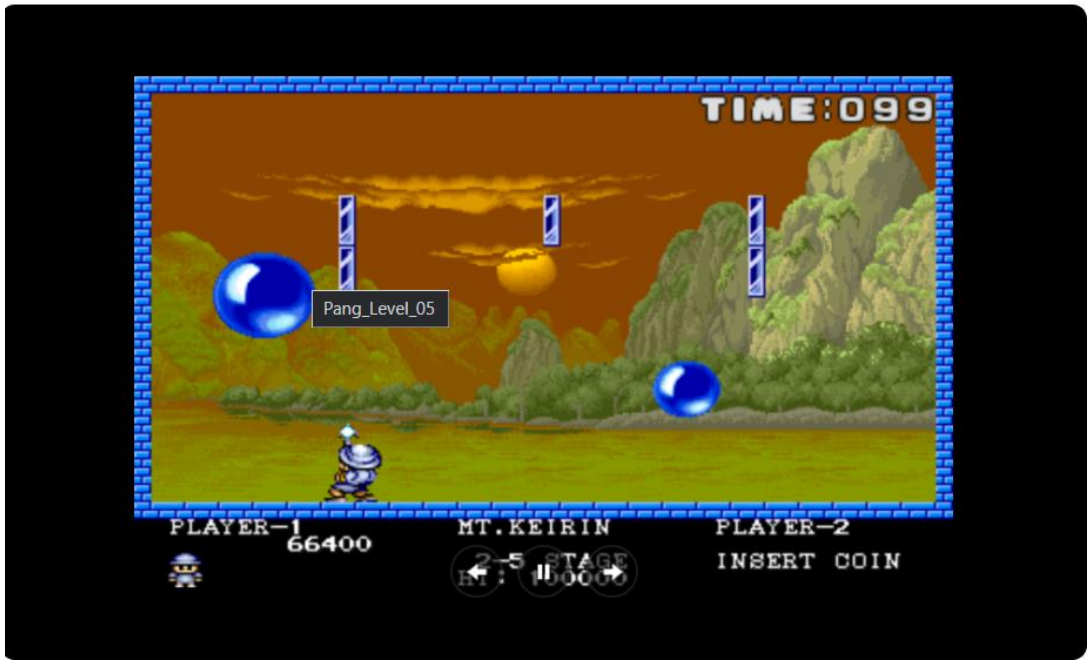

Figura 2 - Imagen de la versión original del videojuego.

Adicionalmente, el desarrollo de este proyecto demuestra que, a pesar de que los videojuegos modernos son realizados con gráficos más realistas y con una mayor complejidad en cuanto a funcionamiento y temática, no significa que los videojuegos más sencillos no sigan siendo parte del entretenimiento diario de millones de jugadores en la actualidad. Con la realización de este proyecto el objetivo ha sido adentrarme en el mundo del desarrollo de videojuegos, un campo muy interesante dentro del desarrollo software.

1.1. Objetivo

El objetivo de este proyecto es recrear el videojuego clásico arcade Pang, versionándolo y actualizándolo a los tiempos actuales. Para satisfacer dicho objetivo se emplearán los nuevos motores de desarrollo como Unity y se adaptará a las nuevas plataformas. Por otra parte, se pretenden realizar ciertos cambios necesarios para los juegos de la actualidad, así como incorporar funcionalidades adicionales

para aportar más contenido para el jugador, buscando los siguientes objetivos secundarios:

- Implementar la funcionalidad esencial del videojuego original, preservando movimientos, estilos, armas e idea del Pang.
- Implementar diferentes estilos de armas, potenciadores y bolas.
- Añadir un sistema de puntuaciones online.
- Añadir un modo de juego infinito.
- Añadir un sistema de logros, ya que en los videojuegos actuales es algo que se tiene muy en cuenta.

2. Herramientas, tecnologías y materiales utilizados

En este apartado se describen las herramientas, las tecnologías y los materiales utilizados para el desarrollo del proyecto. El objetivo del apartado es ofrecer una visión del trabajo realizado durante la implementación del videojuego.

2.1. Herramientas

En primer lugar, se muestran las herramientas software utilizadas para la construcción del proyecto. Principalmente el motor de juegos Unity y herramientas adicionales para el diseño de los aspectos gráficos.

2.1.1. Unity

Unity es el motor diseñado para el desarrollo de videojuegos multiplataforma en 2D o 3D, creado por Unity Technologies. Su inicio data del año 2005 y desde entonces han ido actualizando e implementando nuevas funcionalidades. Aunque en su inicio fue exclusivo para MacOS, hoy en día se puede usar tanto en Linux como en Windows.

Un dato a destacar sobre Unity es que no está solo diseñado para el desarrollo de videojuegos, sino que también se puede usar para crear aplicaciones interactivas como la realidad aumentada o el desarrollo de aplicaciones para móviles.

Unity dispone de cuatro tipos de licencias para aquellas personas que quieran usar dicho motor. Para la realización de este proyecto se ha empleado la licencia personal ya que es la única gratuita. Dicha licencia proporciona todo lo necesario para el desarrollo de este proyecto. Adicionalmente, la licencia fija ciertos parámetros como el límite de beneficio económico que puedes recibir a través de tu aplicación o un número máximo de usuarios concurrentes.
Se ha decidido usar este motor debido a que es uno de los principales empleados dentro de la industria de los videojuegos, siendo el motor usado en grandes éxitos cómo *Rust o Pokemon Go*. Por otra parte, este motor tiene la capacidad

de ofrecer una interfaz intuitiva, soporte para varios lenguajes de programación, una gran cantidad de funcionalidades y una documentación muy completa.[5]

Es importante definir ciertos conceptos sobre Unity, lo que ayudará a una mejor comprensión de todo el desarrollo del sistema que se detalla posteriormente.

La implementación se basa en un tipo de elemento denominado *Game Object,* que es la representación básica de cualquier elemento establecido en el motor. Por sí solos no desarrollan una gran función, su gran capacidad de uso e interés viene dado por los componentes que puede contener. Siempre son creados con el componente *Transform,* el cual sirve para indicar la posición, rotación y escala del elemento dentro del juego. Además, se les puede añadir diversos componentes en función de lo deseado como puede ser el *Collider 2D*, que sirve para crear un borde de interacción del objeto. El buen uso y elección de estos son la clave para crear los diferentes elementos que compondrán el juego final.[6]

Sin lugar a duda, una de las características más importantes y útiles es la de los denominados *prefabs.* Los *prefabs* son elementos similares a las clases en programación, donde una vez empleados, tienen sus valores propios. Estos deben ser generados por el desarrollador según sus requisitos y una vez creado, puede ser empleado tantas veces como se requiera, haciendo posible emplear el mismo objeto en diversos niveles, pero con una sola implementación. Posteriormente, si se requiere un cambio en todo el desarrollo, simplemente habrá que actualizar el *prefab* y este lo actualizará donde se encuentre, además de la posibilidad de crear distintas versiones a partir de un original. De esta manera se fomenta la reutilización de elementos de forma más sencilla y ágil.[7]

Por otro lado, aparece la clase base para todos los scripts generados en Unity, denominada *MonoBehaviour.* De esta clase heredan todos los scripts empleados en el proyecto para acceder a una serie de funciones, propiedades y eventos muy útiles a la hora de programar un videojuego. Algunos ejemplos pueden ser *Awake()* y *Start()*, los cuales son llamadas al crearse el objeto, a *Update()* al cual se le llama en cada fotograma del juego u *OnTriggerEnter(),* el cual se emplea cuando dos elementos interactúan entre sí.[8]

Por último, tenemos las escenas, estructuras empleadas para dividir las partes de un juego. Cada una de ellas puede contener los mismos valores y *game objects*, pero con distinta distribución. Esto hace posible la creación de diversos niveles, menús o pantallas como la de los logros o la clasificación.

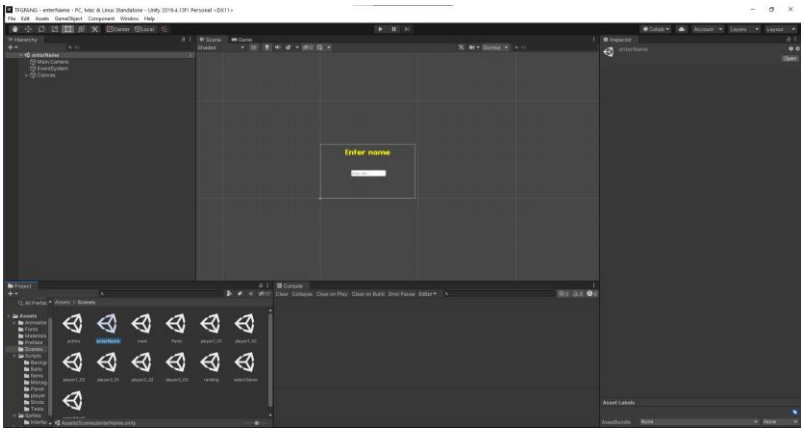

Figura 3 - Interfaz del motor Unity

2.1.2. Visual Studio Code

Visual Studio Code es un entorno de desarrollo en multilenguaje, así como multiplataforma lanzado en 2015 basado en el editor Mónaco, que Microsoft había lanzado al mercado en 2013.
Es un editor de texto basado en extensiones. Estas se sitúan en una barra lateral dando el mismo nivel a la exploración de archivos, la depuración, el control del código fuente y la búsqueda. Consiguiendo que cada usuario gestione sus propias extensiones como quiera de una manera muy visual e intuitiva.
Se ha utilizado esta herramienta ya que es la que uso a diario en el trabajo y me ha facilitado muchas tareas al ser tan cómoda.

2.1.3. Adobe Photoshop

Adobe Photoshop es uno de los softwares de creación y edición de gráficos más conocido y utilizado en su sector. Desarrollado por Adobe Systems Incorporated hace ya 31 años, en 1990, esta herramienta ha sufrido grandes actualizaciones cada poco tiempo, dando un gran paso en cuanto a la edición y el diseño gráfico. Contiene una inmensa cantidad de opciones y herramientas sobre multitud de formatos de archivos. Estas propiedades son el motivo de que se haya utilizado, ya que permite multitud de opciones con los archivos .PNG, archivos usados en los elementos del juego.

2.2. Tecnologías

En este apartado se explicarán las tecnologías utilizadas durante la realización del proyecto.

2.2.1. C#

C# (C Sharp) es un lenguaje de programación desarrollado por Microsoft como parte de su framework .NET, considerado una evolución de los lenguajes C y C++.
Es un lenguaje orientado a objetos, basado en clases y de tipado fuerte.
Es uno de los lenguajes que Unity proporciona de forma nativa para la programación de juegos.

2.3. Materiales

Los materiales utilizados, *sprites* o iconos, han sido obtenidos de un banco de recursos gratuitos además del propio mercado de Unity. A su vez, para el desarrollo de Pang-demia ciertos de estos recursos han modificados con las herramientas empleadas y otros han sido creados de cero.

3. Metodología

En este apartado se describe la metodología empleada en la realización del proyecto. Dicha metodología ha sido la iterativa incremental.

3.1. Metodología iterativa incremental

En el presente proyecto ha sido aplicada la metodología iterativa incremental. Esta es una metodología de desarrollo software que se basa en la realización de diversas iteraciones, en las que en cada una de ellas se implementan ciertas funcionalidades, permitiendo así una evolución continua del proyecto.
Cada iteración es entendida como un bloque completo de un desarrollo de software que cuenta con sus propias fases de análisis, diseño, implementación y pruebas.[9]

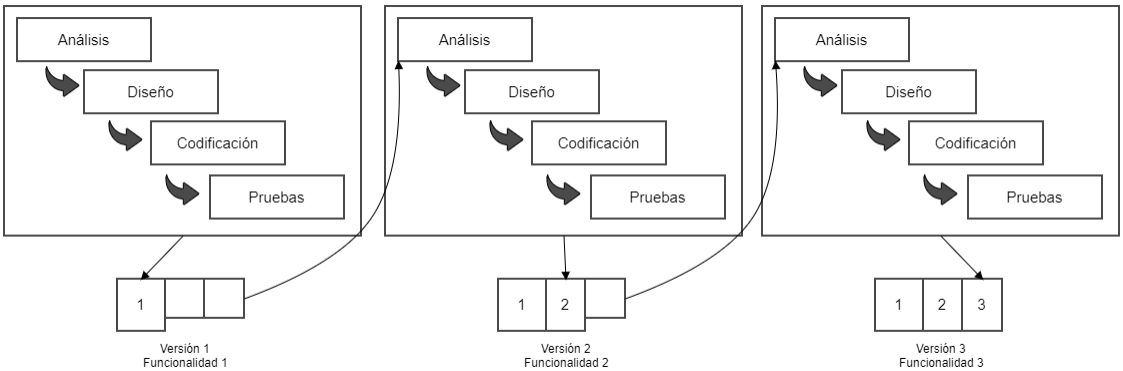

Figura 4 - Modelo iterativo incremental

3.2. Planificación

El proyecto se planificó estableciendo unos plazos bien delimitados a los cuales se ha ceñido todo el desarrollo del mismo. En la Figura 5 se muestra la planificación de tareas en donde puede observarse la planificación del proyecto.

Nombre	Duracion	Inicio	Terminado	Predecesores
Formación	10 days	9/11/20 8:00	20/11/20 17:00	
Desarrollo	**48 days**	**23/11/20 8:00**	**27/01/21 17:00**	
It1	**7 days**	**23/11/20 8:00**	**1/12/20 17:00**	**1**
Análsis	1 day	23/11/20 8:00	23/11/20 17:00	
Diseño	1 day	24/11/20 8:00	24/11/20 17:00	4
Implementación	4 days	25/11/20 8:00	30/11/20 17:00	5
Pruebas	1 day	1/12/20 8:00	1/12/20 17:00	6
It2	**10 days**	**2/12/20 8:00**	**15/12/20 17:00**	**3**
Análsis	1 day	2/12/20 8:00	2/12/20 17:00	
Diseño	1 day	3/12/20 8:00	3/12/20 17:00	9
Implementación	7 days	4/12/20 8:00	14/12/20 17:00	10
Pruebas	1 day	15/12/20 8:00	15/12/20 17:00	11
It3	**10 days**	**16/12/20 8:00**	**29/12/20 17:00**	**8**
It4	**11 days**	**30/12/20 8:00**	**13/01/21 17:00**	**13**
It5	**10 days**	**14/01/21 8:00**	**27/01/21 17:00**	**18**
Integracion	6 days	28/01/21 8:00	4/02/21 17:00	23
Documento	20 days	5/02/21 8:00	4/03/21 17:00	

Figura 5 – Gestión del proyecto.

3.2.1. Formación

Antes de comenzar con el desarrollo del proyecto descrito en este documento, se realizó una fase previa de formación. Dicha etapa se alargó varios días en la que se procedió a la instalación de todo el software necesario para el desarrollo realizado, así como una autoformación para comprender los aspectos principales e iniciales del motor Unity.

3.2.2. Desarrollo e iteraciones

En esta fase se describe cada una de las iteraciones, que han tenido en total una duración aproximada de cuatro meses. En cada una de estas fases se ha ido adaptando el tiempo aplicado dependiendo de la complejidad del proceso desarrollado. Además se desarrollan las distintas etapas de cada iteración.

- Análisis: En esta etapa de la iteración, se definen los puntos a cumplir en la misma. Esta primera fase resultó algo más complicada que en el resto. Esto fue debido a que no se contaba con una visión previa de la situación y es la base de todo el proyecto. En las fases posteriores, ya se tenían, gracias a este análisis, unos conocimientos previos más asentados, además de un código sobre el que trabajar.

- Diseño: Durante esta etapa, se trata de hallar soluciones a los puntos definidos anteriormente. Idealmente sobre cómo organizar la jerarquía de las clases o la parte estética del juego.

- Codificación: En esta tercera etapa, se implementa la funcionalidad del juego con la intención de satisfacer los requisitos establecidos y alcanzar los objetivos definidos previamente.

- Pruebas: Durante la última etapa, una vez terminada la parte de codificación, se prueba el correcto funcionamiento del código generado. Para ello se han probado distintas combinaciones y situaciones comprobando que el funcionamiento era en todos los casos el correcto.

A continuación, se describen las iteraciones realizadas y las funcionalidades que fueron siendo añadidas en cada una de ellas.

3.2.2.1. Iteración 1

Durante la primera iteración se establecieron las bases del juego, así como los puntos clave del desarrollo. El primer punto fue la creación de la pantalla principal, junto con la pantalla de selección del número de jugadores. A continuación, fue creada la pantalla donde el personaje y las bolas interactuarían entre sí, para lo que se realizó la creación del propio personaje, todos sus movimientos e interacciones y las bolas básicas. Por último, fue añadido el disparo inicial del personaje. No hubo un gran avance debido a los pocos conocimientos que en este punto se tenían sobre la compleja realización de todas las interacciones.

3.2.2.2. Iteración 2

Posteriormente, en la segunda iteración, se comenzó el desarrollo del controlador de toda la lógica del juego. Sin embargo, este controlador ha sido modificado a medida que se ha avanzado en los desarrollos ya que es el centro principal del juego. Se añadieron el resto los disparos con los que cuenta actualmente el juego y su controlador. Por último, se crearon objetos que modifican ciertas características del juego como el escudo o el reloj, que pausa el movimiento de las bolas, y las frutas que otorgan puntos.

3.2.2.3. Iteración 3

En este punto, se implementaron ciertos controladores del juego para buscar la correcta interacción entre todos los elementos que lo componen. Por ejemplo, se creó un controlador de los disparos aplicando el patrón *singleton* para que solo exista instancia de este, al igual que en el controlador principal del juego (*GameManager*). Por otro lado, se habilitó la posibilidad de parar la ejecución del juego con un menú de pausa, la aparición de los puntos obtenidos al romper bolas o recoger frutas y un objeto que destruya todas las bolas hasta dejarlas en la última escala.
Finalmente fueron añadidos el resto de los aspectos estéticos como el tiempo de partida, los puntos o las vidas restantes. Además, como punto

relevante a tener en cuenta en esta etapa, se realizó la implementación del manejo del segundo jugador en caso de seleccionar el modo multijugador.

3.2.2.4. Iteración 4

Llegados a este punto, ya habían sido adquiridos unos ciertos conocimientos de todo el funcionamiento del motor y del desarrollo, por lo que se comenzaron a pulir ciertos elementos e implementar pequeños detalles relevantes.

Entre estos detalles podemos encontrar:

- Mostrar un panel con las interacciones tras acabar el nivel.
- Posibilidad de almacenar la puntuación del jugador una vez es finalizada la partida en una tabla de puntuaciones global. Para ello se ha empleado un servicio externo a Unity como es *Dreamlo.*
- Creación de un nuevo modo de juego, el modo Panic descrito anteriormente en el presente documento. Además, fueron añadidos para este nuevo modo la aparición de bolas y su nuevo aspecto, el cual cuenta con una barra de progresión del nivel.

3.2.2.5. Iteración 5

En esta última etapa, se introdujo un nuevo tipo de bola, con un movimiento distinto al establecido en las bolas básicas que no rebotan teniendo en cuenta la gravedad. Por otra parte, se incorporaron logros habilitados a nivel local que el jugador puede ir obteniendo a modo de superación. Por último, se crearon diversos niveles para aumentar la duración de la partida.

4. Análisis de requisitos

En este apartado se recogen todos los requisitos capturados durante las distintas fases de análisis del proyecto. Al ser un proyecto basado en la idea de un videojuego existente al que se le han añadido características nuevas, estos requisitos han sido ideados únicamente por el autor, ya que no dependía de las especificaciones de ninguna otra persona, además de alguna idea sugerida por parte del tutor del proyecto.

4.1. Análisis del juego

Antes de comenzar con las explicaciones más técnicas del proyecto, es conveniente explicar el funcionamiento del juego desarrollado con el fin de describir su mecanismo básico y los diferentes modos que se incluyen en el mismo.

El videojuego Pang basa su idea principal en un personaje que se encuentra en una línea 2D sobre la que se puede desplazar de izquierda a derecha de la pantalla con el objetivo de no ser golpeado por una serie de artefactos que pueden colisionar con el personaje y matarlo. Adicionalmente, en función del nivel del videojuego en el que se esté operando, el personaje también puede desplazarse verticalmente hacia ciertas plataformas existentes en pantalla por medio del uso de escaleras. En cuanto a los artefactos anteriormente mencionados, consisten en una serie de bolas cuya forma puede variar y su función es rebotar contra las estructuras que aparecen en pantalla.

El objetivo del jugador se centra en disparar a dichas bolas, las cuales, cuando son destruidas se subdividen en dos bolas cada vez, pero de menor tamaño. Esto complica la misión del jugador, quien debe conseguir destruir todas las bolas que vayan apareciendo antes de que estas le toquen, ya que esto produce la perdida de una vida dentro del videojuego. Así mismo, mientras el jugador debe esquivar todas las bolas, al irlas destruyendo.

Del videojuego clásico se mantendrán dos tipos de disparos, mejorando su aspecto visual. Dichos disparos se detallan en la siguiente tabla junto a su efecto.

Disparo	Efecto
	Permite disparar dos veces simultáneamente.
	Si al disparar no se colisiona con ninguna bola, se anclará a la superficie durante dos segundos.

Tabla 1 - Disparos del videojuego clásico.

Por otro lado, ocurre lo mismo con otros elementos del videojuego clásico. Se añaden dos bolas, una con forma circular y otra hexagonal, y los potenciadores clásicos del juego que se observan en la Tabla 2.

Potenciador	Efecto
	Se activará un escudo que permitirá al jugador ser golpeado por las bolas una sola vez, sin perder con ello una de las vidas restantes. El efecto desaparece tras avanzar de nivel.
	Le otorga al jugador una vida extra durante la partida.
	Detiene durante tres segundos el movimiento de las bolas.
	Explota las bolas de gran tamaño en pantalla, convirtiéndolas en bolas con el menor tamaño posible que pueden alcanzar.

Tabla 2 - Potenciadores del videojuego clásico.

Además, se plantean dos modos de juego diferentes:

- **Modo Tour**: En este modo el jugador tratará de sobrevivir a las bolas mostradas en pantalla inicialmente mientras va destruyéndolas. Tras lograr acabar con todas las bolas existentes, el usuario accede al siguiente nivel, donde tanto el escenario, como la colocación de inicial y el número de las bolas va variando. Para ello puede beneficiarse de ciertos objetos, generados aleatoriamente al explotar las bolas. Estos objetos le proporcionan modificaciones en el tipo de disparo, su velocidad de movimiento o la velocidad de las bolas. Además, existen otros objetos que proporcionan ayuda al jugador en el desarrollo de su misión: un escudo, una vida extra, puntos adicionales o la explosión de todas las bolas hasta dejarlas con el menor tamaño posible. Este modo cuenta además con la funcionalidad multijugador local. El objetivo final de este modo es ir sobreviviendo de manera que el jugador supere todos los niveles existentes del juego.

- **Modo Panic:** Este modo consiste en ir destruyendo las bolas que vayan apareciendo en la parte superior de la pantalla e ir consiguiendo la máxima puntuación posible. A medida que se aumente de nivel, disminuirá el tiempo de aparición entre bolas. Además, se podrá seguir haciendo uso de los mismos potenciadores descritos anteriormente. El objetivo en este modo es sobrevivir lo máximo posible consiguiendo así la máxima puntuación hasta que el usuario se quede sin vidas, ya que este modo es infinito.

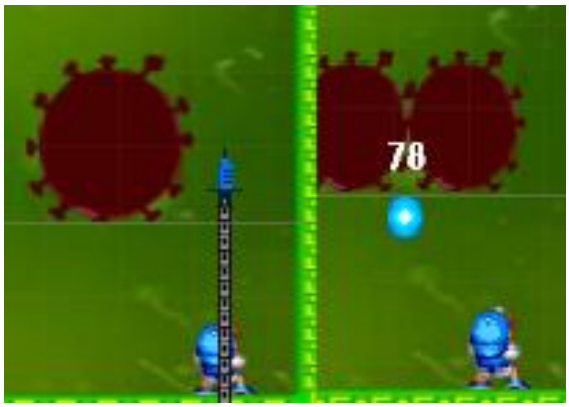

Figura 6 - Bola dividiéndose al ser disparada.

En la versión del videojuego Pang de este proyecto, las bolas que aparecen en pantalla simulan el virus COVID-19. Existe otro tipo de bola adicional con forma hexagonal que hace de cura para el virus al simular una pastilla. Esta adaptación ha sido creada debido a que el desarrollo del proyecto ha sido ambientado en la situación actual en la que nos encontramos. Por este motivo, el videojuego se llama Pang-demia.

Para esta nueva versión, se han añadido dos tipos extra de disparos que se muestran en la Tabla 3.

Disparo	Efecto
	Dispara tres láseres en formación triangular y con un límite de quince simultáneamente.
	Coloca minas en la superficie para explotar las bolas.

Tabla 3 - Disparos añadidos al videojuego.

También se ha incorporado un nuevo potenciador que modifica la velocidad del jugador.

Potenciador	Efecto
	Aumenta la velocidad de movimiento del jugador hasta colisionar con una pared o finalizar el nivel.

Tabla 4 - Potenciadores nuevos del videojuego.

A parte de los elementos más visuales dentro del videojuego, se ha añadido una funcionalidad nueva. Dicha funcionalidad, consta de una clasificación a nivel mundial, donde el jugador podrá comparar su puntuación con la de otros jugadores. Al finalizar la partida, aparecerá un panel donde el jugador insertará el nombre que desea que se muestre en la clasificación.

4.2. Requisitos funcionales

Los requisitos funcionales son aquellos que definen las funcionalidades que el sistema debe tener tras el desarrollo.
Los requisitos funcionales de este proyecto son los expuestos en la Tabla 5, identificados por un código y su breve descripción sobre las acciones que se pueden llevar a cabo por los actores que utilicen el sistema.[10]

ID	Descripción
RF01	El jugador no podrá moverse, disparar o romper las bolas hasta que estas comiencen a moverse.
RF02	Cada nivel tendrá una duración máxima de 100 segundos.
RF04	El usuario podrá seleccionar el modo de juego entre el modo Tour o el modo Panic.
RF05	Cuando se complete un nivel en el modo Tour, se mostrará información resumida del nivel: bolas destruidas, frutas recogidas y tiempo sobrante.
RF06	El juego tendrá un sistema de clasificación online.
RF07	El jugador solo dispondrá de un tipo de disparo habilitado.
RF08	El jugador podrá elegir si cambiar su disparo o no cuando aparezca uno nuevo en pantalla.

RF09	A medida que avance niveles en el modo Panic, la dificultad aumentará.
RF10	El jugador finalizará el nivel del modo Tour cuando no quede ninguna bola en pantalla.
RF11	El jugador perderá cuando se quede sin vidas.
RF12	El usuario podrá mover al jugador de izquierda a derecha.
RF13	El usuario podrá mover al jugador de abajo a arriba y subir escaleras.
RF14	El usuario podrá disparar
RF15	El usuario podrá finalizar la ejecución del juego.
RF16	El jugador perderá cuando se agote el tiempo.
RF17	Al finalizar, el jugador podrá introducir un nombre para la clasificación.
RF18	Se mostrará la clasificación de las 10 mejores puntuaciones.
RF19	El jugador podrá ver el progreso de los logros.
RF20	El sistema deberá actualizar la interfaz en todo momento con la puntuación actual del jugador y el número de vidas restantes.
RF21	El usuario podrá parar la ejecución del nivel cuando lo desee.
RF22	El sistema recargará el nivel si el usuario pierde una vida.
RF23	Al terminar, el usuario podrá volver a jugar si lo desea.
RF24	EL juego dispondrá de un modo multijugador local.

Tabla 5 - Requisitos funcionales

4.3. Requisitos no funcionales

Los requisitos no funcionales son aquellos que no hacen referencia directa al funcionamiento del sistema, sino a ciertas propiedades con las que dicho sistema debe contar.
Los requisitos no funcionales de este proyecto son los expuestos en la Tabla 6, diferenciados por su código, breve descripción, tipo e importancia para el sistema.

ID	Descripción	Tipo	Importancia
RNF01	El juego podrá ejecutarse en Windows, Linux y navegador web.	Portabilidad	Muy alta
RNF02	Las puntuaciones se almacenarán en un servicio externo al juego.	Disponibilidad	Muy alta
RNF03	El sistema hará un uso de menos de 2GB de RAM.	Rendimiento	Alta
RNF04	El juego será apto para todos los públicos.	Usabilidad	Alta
RNF05	El juego no bajará de 30 FPS mínimos.	Rendimiento	Media

Tabla 6 - Requisitos no funcionales

5. Diseño e implementación

En este capítulo se detalla primero el diseño arquitectónico del sistema para después entrar en detalle con el diseño y la implementación de cada una de las partes.

Para una mejor comprensión de la explicación, hay que recordar ciertos conceptos mencionados con anterioridad sobre el motor Unity como son los *game object*, escena o *prefabs*.

5.1. Arquitectura del sistema

El modelo arquitectónico aplicado en el proyecto es la arquitectura basada en tres capas. Es la arquitectura más empleada en el desarrollo de sistemas software y trata de dividir el sistema en tres capas diferenciadas para facilitar la administración de cada una independientemente de las otras. De esta manera es posible alterar el comportamiento de cada una de ellas sin que el resto se vean afectadas. Estas tres capas son la capa de presentación, la capa de negocio y la capa de datos.

La capa más superficial es la capa de presentación, la cual es la parte visual de todo el sistema y con la que el usuario interacciona. Dicha capa se ha tratado de mejorar, pero manteniendo una estética arcade clásica. La cual debe ser atractiva e intuitiva para el usuario.

La capa intermedia es la capa de negocio, la cual contiene todo el control del sistema y sirve de conexión entre las otras dos. Esta capa ha sido programada empleando el lenguaje C#, al ser el común empleado en Unity.

La capa de datos en este caso es algo compleja de explicar. Esto se debe a que, si bien es la empleada para la persistencia de los datos, a la hora de identificar los elementos persistentes encontramos ciertos componentes que la propia herramienta de Unity almacena automáticamente. Estos elementos son: las escenas, la posición inicial de las bolas y la máxima puntuación local. Por lo que no es necesario crear un almacenamiento ajeno de datos para ello.

Aunque con respecto a las puntuaciones para la clasificación sí se hace uso de un sistema ajeno a Unity y, por lo tanto, tiene sentido el uso de la arquitectura.

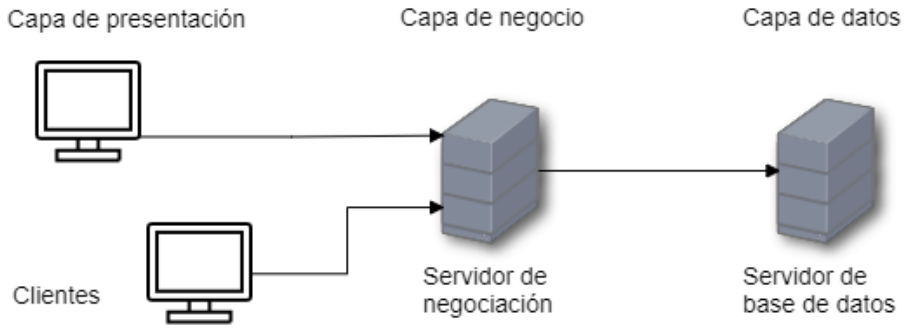

Figura 7 - Arquitectura en 3 capas

5.2. Capa de presentación

La capa de presentación se encarga de llevar a cabo la interacción entre el usuario y el sistema. Por tanto, aquí se presentan y administran las diferentes pantallas del juego. Con esta capa el usuario puede visualizar el escenario por el que va a moverse, calcular donde disparar y ver la puntuación actual, el tiempo restante de partida, el número de vidas disponibles y la puntuación local máxima.

Dentro de esta capa se distinguen dos componentes distintos: la interfaz de usuario y las escenas o niveles del juego.

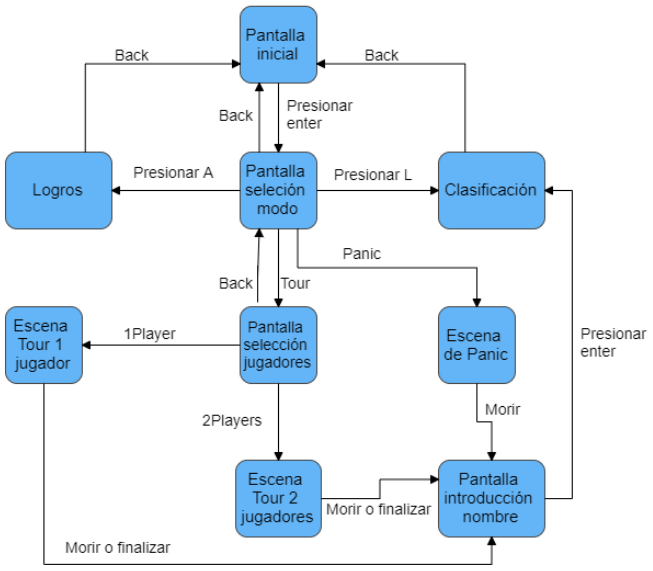

Figura 8 - Diagrama de navegación.

5.2.1. Interfaz de usuario

La idea principal de la interfaz del usuario es que todo sea intuitivo y sencillo para el jugador. Para ello, se ha simplificado lo máximo posible, dejando las posibles opciones claras. Este juego no contiene una gran variedad de menús o pantallas donde seleccionar elementos debido a que no está enfocado en ello. Aun así, dentro del juego podemos encontrar las siguientes pantallas:

Figura 9 - Pantalla inicial del juego.

Lo primero que aparece al iniciar el juego, es la pantalla inicial de este, mostrada en la Figura 9. Esta sesión contiene el título del juego y un letrero que parpadea indicando que se debe presionar la tecla *intro* para comenzar.

Figura 10 - Pantalla de selección de modo de juego.

Una vez se pasa a la siguiente pantalla, podemos encontrar lo mostrado en la Figura 10 donde se muestra la posible selección entre ambos modos de juego. El modo seleccionado se verá con un color claro y una opacidad del 100% mientras que el modo que no se selecciona, se muestra con una opacidad del 50%, haciendo que este se vea más oscuro o sombreado. Para intercambiar el modo de juego es necesario utilizar las flechas del teclado, izquierda o derecha correspondientemente. Además, el usuario puede acceder a la visualización de los logros presionando la tecla "A", a la clasificación mundial presionando la tecla "L" e incluso volver atrás con la tecla de borrado.

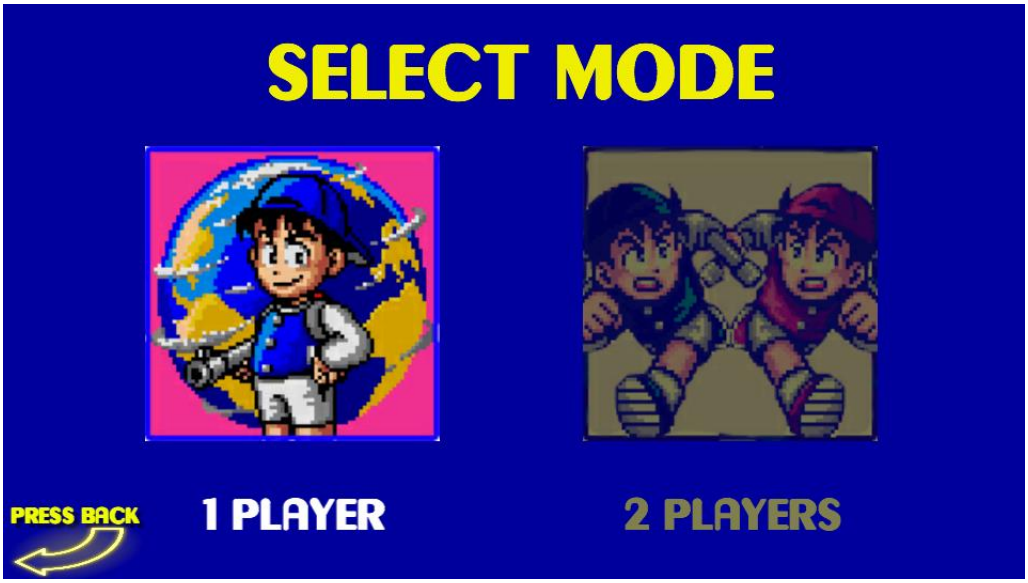

Figura 11 - Pantalla de selección de jugadores.

Si el jugador selecciona el modo Tour, accederá a la pantalla de la Figura 11. Esta pantalla sigue el mismo comportamiento que la anterior, dando a escoger entre jugar utilizando el modo de un único jugador o multijugador, además de la opción para volver atrás.

```
public void UpdateScore(int pts){
    points+=pts;
    scoreText.text =points.ToString();

    if (points>hiScore)
    {
        hiScore=points;
        hiScoreText.text="HS-"+hiScore.ToString();
        PlayerPrefs.SetInt("HiScore",hiScore);

    }
    PlayerPrefs.SetInt("Puntos",points);
}
/**
*Cargae el fichero con el valor del hiscore almacenado, pa tenr los datos
*/
public void UpdateHiScore(){
    hiScore=PlayerPrefs.GetInt("HiScore");
    hiScoreText.text="HS-"+hiScore.ToString();

}
```

Figura 12 - Ejemplo de uso de PlayerPrefs.

Al utilizar una llamada a la herramienta *PlayerPrefs,* el sistema almacena los datos requeridos en un fichero. Sin embargo, estos artefactos solo permiten almacenar datos de forma local.

Figura 13 - Pantalla que muestra la clasificación global.

En la Figura 13 se muestra las posiciones que mantienen las 10 mejores puntuaciones establecidas hasta el momento. Estas puntuaciones son las almacenadas en el servicio externo *Dreamlo,* el cual se detalla más adelante.

Figura 14 - Pantalla de introducción de nombre.

En esta pantalla (Figura 14) se muestra el campo donde se introduce el nombre del usuario una vez se ha finalizado la partida, lo que sucede tras perder todas las vidas o haber completado todos los niveles del modo Tour. El nombre introducido será enviado al servicio externo empleado para las puntuaciones.

Figura 15 - Panel de pausa de la partida.

En la Figura 15, se representa la situación en la que el jugador a decido parar momentáneamente la ejecución del juego, pudiendo volver a su ejecución en el mismo instante donde se para.

Figura 16 - Panel de recuento al finalizar un nivel.

En la figura 16 se muestra el panel que aparece cuando se finaliza un nivel del modo Tour. Esta sesión contiene el recuento de bolas destruidas, la cantidad de frutas recogidas y el tiempo restante. Los tres valores que se muestran sumarán una puntuación extra al total de la partida cuando se finaliza cada nivel.

5.2.1. Escenarios

Se puede decir que las escenas son las partes más visuales de la interfaz ya que es en estas pantallas donde se desarrolla la acción del juego. Cada una de estas pantallas es independiente de las demás y hace de contenedor para los menús, paneles y objetos.

Dentro del juego, encontramos las siguientes escenas:

5.2.1.1. Escena del modo Tour

Figura 17 - Pantalla del modo Tour con un jugador.

En la Figura 17 se muestra el primer nivel del modo Tour con un solo jugador. En ella se puede ver el avatar en el centro de la línea inferior horizontal, así como los dos tipos de bola desarrollados en Pang-demia.

En la subinterfaz de abajo se encuentra de izquierda a derecha: puntuación actual, tipo de disparo, tiempo restante, mejor puntuación local y número de vidas restantes.

Todos los elementos mencionados son objetos de tipo *game object.* El personaje y las bolas, además de *game object* son *prefabs*, lo que facilita su utilización en otras escenas del videojuego en las que aparecen, ya que son los elementos principales del juego. Estos objetos no son utilizados en todos los niveles, sino que cada avatar es distinto en cada uno de ellos a pesar de mantener el mismo aspecto.

En lo referente a los datos a ser almacenados entre niveles, únicamente es necesario guardar la puntuación actual y el número de vidas. Para almacenar esta información se utiliza un objeto de tipo *DontDestroy,* objeto proporcionado por la herramienta Unity para preservar datos entre cada una de las escenas.

Si se utiliza el modo de juego multijugador, la diferencia en esta escena es la aparición de un segundo avatar del personaje en color rojo junto al ya conocido personaje vestido de azul.

5.2.1.2. Escena del modo Panic

Figura 18 - Pantalla del modo de juego Panic

Por último, tal y como se muestra en la Figura 18, se encuentra la pantalla del juego en el modo Panic. En esta escena la diferencia más notoria es la sustitución del tiempo restante que aparece en el modo Tour, por la barra de progreso del nivel de dificultad. Dicha barra se va rellenando de color de manera progresiva a medida que el jugador dispara a las bolas y estas se van dividiendo.

En el modo Panic las bolas no aparecen siempre en la misma posición cuando se inicia una partida. Al contrario, estas bolas se generan de manera aleatoria a lo largo del eje horizontal y van cayendo desde la parte superior de la escena. La aparición de estas bolas se realiza en función del tiempo que haya transcurrido dependiendo del nivel en el que se encuentre el jugador o bien cuando ya haya sido destruida la bola anterior que aparecía en escena.

5.3. Capa de negocio

La capa de negocio es la encargada de almacenar toda la lógica del sistema, incluyendo tanto las funcionalidades que debe de implementar el sistema, como las interacciones y reglas entre elementos.

Para no detallar todo el código se describen los puntos más relevantes de este. En la Figura 19, se muestra la organización de las clases que componen el proyecto.

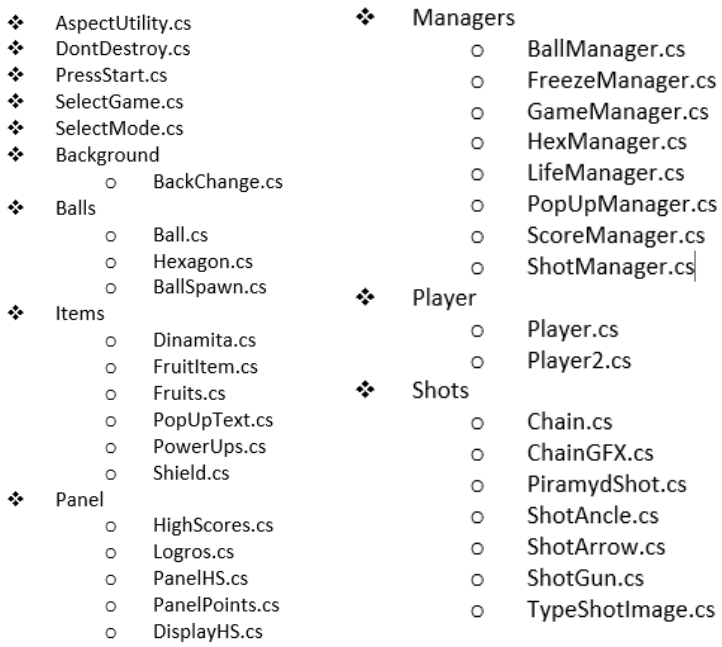

Figura 19 - Clases del videojuego.

5.3.1. Gestor del juego (Game Manager)

Los juegos están formados por diversos módulos, que realizan funciones diferentes e interactúan entre sí, por lo que hace falta un gestor para llevar a cabo esta acción. El *GameManager* es la clase fundamental y básica del juego. Por esta clase pasa toda la gestión del sistema, desde el comienzo de la partida hasta su fin ya que se encarga de comunicar a los elementos entre sí y organizar todos los elementos utilizados dentro del videojuego.

Esta clase aplica el patrón *Singleton*, por lo que solo existe una instancia del objeto. Esto es debido a que así se proporciona un único punto de conexión entre todos los elementos y se puede acceder globalmente a dicha instancia de la clase *GameManager*.

```
public static GameManager gm;
private void Awake(){

    if(gm==null){
        gm=this;
    }else if(gm!=this){
        Destroy(gameObject);
    }
    player= FindObjectOfType<Player>();
     if (SceneManager.GetActiveScene().name.Contains("player2_"))
    {
        player2=FindObjectOfType<Player2>();
    }
    lm= FindObjectOfType<LifeManager>();
    fruits= FindObjectOfType<Fruits>();

    if (SceneManager.GetActiveScene().name.Equals("Panic"))
    {
        gameMode=GameMode.PANIC;
    }else{
        gameMode=GameMode.TOUR;
    }

}
```

Figura 20 - Patrón Singleton en GameManager.cs

Como se observa en la Figura 20, el *GameManager* se encarga de la gestión de los componentes principales del juego. En esta figura, se muestra el método *Awake(),* el cual se inicia al empezar la ejecución y asigna tanto los jugadores como el modo de juego.

Al ser de acceso público, la clase se encarga de aspectos de alto funcionamiento dentro del juego, pudiendo ser solicitados desde cualquier punto. Esto es útil para almacenar en la clase información que afecta a gran parte del juego y que debe ser accedida por varios módulos.

Al ser la clase organizadora principal, se encarga tanto de la iniciación del juego como la finalización, además de la gestión de otros controladores, como se puede ver en la Figura 21 que se muestra a continuación.

```
public void StartGameOver(){
    if(gm!=null){
        StartCoroutine(GameOver());
    }
}
public IEnumerator GameStart(){

    yield return new WaitForSeconds(2);
    ready.SetActive(false);

    if (gameMode==GameMode.TOUR){
        BallManager.bm.StartGame();
    }else{
        BallSpawn.bs.NextBall();
    }
    inGame=true;

}
public IEnumerator GameOver(){

    gameOver.SetActive(true);
    yield return new WaitForSeconds(5);
    SceneManager.LoadScene("enterName");

}
```

Figura 21 - Código de inicio y fin de la partida.

5.3.2. Jugador

La clase *Player.cs* existente dentro del proyecto, es la encargada de toda la gestión y acciones realizadas por el jugador. En ella se engloba el movimiento del avatar, evitando que salga de la escena con su movimiento y sus animaciones al ganar, morir o moverse.

Para la programación del movimiento del jugador, se emplea una herramienta propia de Unity donde se establece el nombre de los ejes y se le asignan las teclas. Para ello es necesario hacer una llamada a dicha herramienta desde el código de esta clase *Player* pasando como parámetro el nombre de los ejes. En la Figura 22, se puede observar cómo se asignan estos ejes, además de la inversión de la imagen del avatar a la hora de caminar hacía un lado u otro.

```
void Update()
{
    if (GameManager.inGame)
    {
        movementX=Input.GetAxisRaw("Horizontal")* speedX;
        movementY=Input.GetAxisRaw("Vertical")* speedY;
        animator.SetInteger("velX", Mathf.RoundToInt(movementX));
        animator.SetInteger("velY", Mathf.RoundToInt(movementY));
        if(movementX<0){
            sr.flipX=true;
        }else{
            sr.flipX = false;
        }
    }
}
```

Figura 22 - Parte del código encargada del movimiento.

```
private void OnBecameInvisible()
{
    if (lm.lifes<=0)
    {
        GameManager.gm.StartGameOver();
    }else{
        Invoke("ReloadLevel", 0.5f);
    }

}
```

Figura 23 - Fragmento del código encargado de recargar el nivel.

En la Figura 23, se observa el método empleado para recargar el nivel. Una vez el avatar es golpeado por una de las bolas existentes en la escena, el personaje realiza un salto con el cual sale de la pantalla. Una vez que el sistema detecta que no es visible, realiza la recarga del nivel.

Para el uso del avatar, se creó uno de los denominados *prefabs,* pudiendo de esta manera ser empleado en cualquier nivel de la misma forma. Apoyándose en la creación de los *prefabs* se ha podido realizar otra versión del avatar para el segundo jugador utilizado en el modo multijugador. Para ello se han realizado unas pequeñas modificaciones como: la asignación de las teclas de movimiento o disparo, los diseños del avatar y la propia clase que se usa para controlarlo.

5.3.3. Bola

En este apartado es relevante mencionar cuatro clases que conforman el proyecto, ya que todas ellas son imprescindibles para su correcto funcionamiento. Estas clases son *BallManager*.cs, *Ball*.cs, *Hexagon*.cs y

SpawnBall.cs. Además, junto con el uso de estas clases es necesaria la creación de *prefabs* con el fin de habilitar tantas bolas como se requiera. La creación de la bola de tipo hexagonal sigue un mismo procedimiento que la bola normal, pero con ciertos matices que se detallan a continuación.

BallManager.cs es la clase encargada de llevar la cuenta de las bolas en pantalla y avisar al gestor del juego de su situación. Por otra parte, tiene la capacidad de otorgar la fuerza inicial a las bolas con las que se comienza un nivel y la gestión del juego una vez se recoge el objeto dinamita.

Por otro lado, se encuentra la clase *Ball*.cs que puede definirse como la clase principal de las bolas. Esta clase se encarga de que una vez sean disparadas las bolas, se destruya el objeto para que no aparezca y se creen las dos bolas siguientes, proporcionándolas una fuerza opuesta la una de la otra. Como se puede observar en la Figura 24, el método *Split* se encarga de instanciar un potenciador y generar unos puntos aleatorios, que pueden ir desde 10 hasta 300, que se le sumarán al jugador.

```
public void Split(){

    GameManager.gm.PanicProgress();
    int n= PlayerPrefs.GetInt("Bolas");
    n++;
    PlayerPrefs.SetInt("Bolas",n);
    if (nextBall!=null)
    {
        CreaPremio();
        GameObject b1=Instantiate(nextBall, rb.position+Vector2.right/4, Quaternion.identity);
        b1.GetComponent<Ball>().derch=true;
        GameObject b2=Instantiate(nextBall, rb.position+Vector2.left/4, Quaternion.identity);
        b2.GetComponent<Ball>().derch=false;
        if (!FreezeManager.fm.freeze)
        {
            b1.GetComponent<Rigidbody2D>().isKinematic=false;
            b1.GetComponent<Rigidbody2D>().AddForce(new Vector2(2,5), ForceMode2D.Impulse);
            b2.GetComponent<Rigidbody2D>().isKinematic=false;
            b2.GetComponent<Rigidbody2D>().AddForce(new Vector2(-2,5), ForceMode2D.Impulse);

        }else{
            b1.GetComponent<Ball>().saveSpeed=new Vector2(2,5);
            b2.GetComponent<Ball>().saveSpeed=new Vector2(-2,5);
        }
        if (!BallManager.bm.destruyendo)
        {
            BallManager.bm. DestroyBall( gameObject, b1, b2);
        }
    }else
    {
        BallManager.bm.LastBall(gameObject);
    }
    int score = Random.Range(10,301);
    PopUpManager.pm.creaPopUpText(gameObject.transform.position, score);
    ScoreManager.sm.UpdateScore(score);
    GameManager.gm.UpdateBalls();
}
```

Figura 24 - Fragmento de código encargado de la división de las bolas.

La clase *Hexagon*.cs, tiene un comportamiento muy similar, aunque el movimiento de este tipo de bola es ligeramente diferente, despreciando completamente la gravedad y añadiendo rotación sobre su movimiento.

```
void Update()
{
    if (GameManager.inGame){
        if (!FreezeManager.fm.freeze){
            rotation=250*Time.deltaTime;
            transform.Rotate(0,0,rotation);
            rb.velocity=new Vector2(forceX,forceY);
        }
    }
}
```

Figura 25 - Movimiento del hexágono.

Dicha clase también se encarga de congelar el movimiento de las bolas, como se observa en la figura 26. En caso de que el jugador recoja el objeto reloj, se detiene el movimiento de las bolas, proporcionándole una ayuda al jugador a la hora de dispararlas.

```
public void FreezeBall(params GameObject[] balls){
    foreach (GameObject item in balls)
    {
        if (item!=null)
        {
            saveSpeed=item.GetComponent<Rigidbody2D>().velocity;
            item.GetComponent<Rigidbody2D>().isKinematic=true;
            item.GetComponent<Rigidbody2D>().velocity=Vector2.zero;
        }
    }
}

public void UnFreezeBall(params GameObject[] balls){
    foreach (GameObject item in balls)
    {
        if (item!=null)
        {
            item.GetComponent<Rigidbody2D>().isKinematic=false;
            item.GetComponent<Rigidbody2D>().AddForce(saveSpeed,ForceMode2D.Impulse);

        }
    }
}
```

Figura 26 - Detención y Arranque del movimiento de la bola.

Por último, se muestra en la Figura 27 la clase *BallSpawn*.cs que es utilizada únicamente en el modo de juego Panic. En ella, se realiza la generación de las bolas en la parte superior de la pantalla una vez se ha destruido la anterior o haya transcurrido cierto tiempo. Además, una vez el jugador vaya progresando y aumentando los niveles, este aumentara la dificultad reduciendo el tiempo de aparición de la siguiente bola.

```
//Instaciar la bola a lo largo del eje X de -7.25 a 7.25
public void NextBall(){
    if (!FreezeManager.fm.freeze && ball == null )
    {
        ball=Instantiate(ballsPrefab[Random.Range(0,ballsPrefab.Length)], new Vector2(AleatoryPosition(),transform.position.y),Quaternion.identity);

        ball.gameObject.tag="Untagged";
        StartCoroutine(MoveDown());
    }
}

float AleatoryPosition(){
    return (Random.Range(-7.25f,7.25f));
}

public void IncreaseDificulty(){
    dificulty++;
    if (dificulty>=5 && dificulty<10)
    {
        timeSpawn=15;
    }else if (dificulty >=10 && dificulty<50)
    {
        timeSpawn=10;
    }else  if(dificulty >=50 ){
        timeSpawn=5;
    }
}
```

Figura 27 - Fragmento de la clase BallSpawn.cs

5.3.4. Disparo

El correcto funcionamiento de los disparos, así como el tener una variedad de tipos de disparo, es algo fundamental para el desarrollo del juego, ya que son necesarios para hacer desaparecer las bolas.

Para conseguir esto, se ha empleado la creación de *prefabs*, generando uno por cada tipo de disparo. Además, se genera otro *prefab* para la consiguiente cadena que dejan como rastro los disparos de la flecha o el ancla.

Al igual que pasa con las bolas u otros elementos del juego, los disparos también hacen uso de un gestor, en este caso *ShotManager*.cs. Este gestor utiliza una lista con los distintos disparos, la posición de los jugadores y un valor que simula una máquina de estado, para facilitar el cambio de tipo de disparo.

Esta clase se encarga de generar el disparo cuando el jugador presiona la tecla indicada. Generalmente los disparos salen de la posición del avatar y ascienden en línea recta en sentido vertical. Sin embargo, existe un disparo que crea dos réplicas de sí mismo y ascienden en sentido vertical, pero con una inclinación (Figura 28).

```
void Shot(){
    if (typeOfShot!=3)
    {
        Instantiate(Shots[typeOfShot], player.position, Quaternion.identity);

    }else{
        Instantiate(Shots[3], new Vector2(player.position.x + 0.5f, player.position.y+1), Quaternion.Euler(new Vector3(0,0,-5)));
        Instantiate(Shots[3], new Vector2(player.position.x, player.position.y+1), Quaternion.identity);
        Instantiate(Shots[3], new Vector2(player.position.x - 0.5f, player.position.y+1), Quaternion.Euler(new Vector3(0,0,5)));
    }
    numShots++;
}
```

Figura 28 - Creación de los disparos.

A la hora de cambiar el tipo de disparo asignado al jugador, se utiliza un *switch* y un valor. Además, se indica el número máximo de disparos posibles por cada tipo. El disparo básico tiene asignado el valor cero por defecto, lo que se traduce en que cada jugador solo puede ejecutar un único disparo y esperar para volver a disparar a que el anterior se destruya, ya sea por medio de la colisión con una bola o una plataforma.

```
public void ChangeShot(int type){

       if (typeOfShot!=type)
       {
           switch (type)
           {
               case 0:
                   if(player2!=null){
                       maxShots=2;
                   }else{
                       maxShots=1;
                   }

                   shotImage.TypeShot("");
                   break;
               case 1:
                    if(player2!=null){
                       maxShots=4;
                   }else{
                       maxShots=2;
                   }
                   shotImage.TypeShot("Arrow");
                   break;
               case 2:
                    if(player2!=null){
                       maxShots=2;
                   }else{
                       maxShots=1;
                   }
                    shotImage.TypeShot("Ancle");
                   break;
               case 3:
                   if(player2!=null){
                       maxShots=16;
                   }else{
                       maxShots=8;
                   }

                    shotImage.TypeShot("Laser");
                   break;
               case 4:
                if(player2!=null){
                       maxShots=16;
                   }else{
                       maxShots=8;
                   }

                    shotImage.TypeShot("Piramyd");
                   break;
           }
           typeOfShot=type;
           numShots=0;
       }

   }
```

Figura 29 - Método para el intercambio de disparo.

A continuación, encontramos los tipos de disparo: *ShotArrow*.cs, empleado tanto para el disparo inicial, como para el disparo doble, *ShotAncle*.cs, *ShotGun*.cs y *PiramydShot*.cs.

El primer tipo de disparo consta de una cabeza o puntero, la cual asciende de forma vertical en línea recta hasta toparse con algún elemento en pantalla y así

detonarse. En caso de colisionar contra una bola, esta explotará convirtiéndose en dos bolas de menor tamaño siempre y cuando las bolas no sean ya del menor tamaño existente. Si fuese el caso y las bolas tuvieran el tamaño mínimo desaparecen por completo de escena. En caso de que el disparo colisione con un potenciador o fruta, estos desaparecen y se activan sus efectos a favor del jugador. Si, por el contrario, el disparo golpea contra el techo o una plataforma simplemente desaparecerá de escena.

```
void Start()
{
    startPos=transform.position;
    GameObject chain=Instantiate(chainGFX, transform.position,Quaternion.identity);
    chain.transform.parent=transform;
    startPos=transform.position;
}

void Update()
{
    transform.position+=Vector3.up*speed*Time.deltaTime;
    if ((transform.position.y-startPos.y)>=0.2f)
    {
        GameObject chain=Instantiate(chainGFX, transform.position,Quaternion.identity);
        chain.transform.parent=transform;
        startPos=transform.position;
    }

}
private void OnTriggerEnter2D(Collider2D collider){

    if (collider.gameObject.tag=="Ball")
    {
        collider.gameObject.GetComponent<Ball>().Split();
    }
     if (collider.gameObject.tag=="Hexagon")
    {
        collider.gameObject.GetComponent<Hexagon>().Split();
    }
    if (collider.gameObject.tag!="Player" && collider.gameObject.tag!="Ladder")
    {
        Destroy(gameObject);
        ShotManager.shm.DestroyShot();
    }
```

Figura 30 - Código del disparo básico.

En el caso del disparo de tipo ancla, este tiene un comportamiento similar. Sin embargo, si este tipo de disparo llega a golpear el techo o una plataforma, se pegará a la superficie permaneciendo ahí durante unos segundos. Esto proporciona la posibilidad a las bolas de golpear la cadena generada con su rastro y así explotar, como se observa en la Figura 31.

```
IEnumerator DestroyAncla(){

    speed=0;
    yield return new WaitForSeconds(1);
    GetComponentInParent<SpriteRenderer>().color=Color.red;
    foreach (GameObject item in chains){
        item.GetComponent<SpriteRenderer>().color=Color.red;
    }
    yield return new WaitForSeconds(1);
    Destroy(gameObject);
    ShotManager.shm.DestroyShot();
}
```

Figura 31 - Fragmento de código de la clase ShotAncle.cs.

El tercer disparo se corresponde a un láser, el cual, como se ha explicado antes, creará dos copias de sí mismo y ascenderán las tres en formación de cono. Este tipo de disparo permite una mayor cantidad de ejecuciones de disparo en pantalla simultáneamente y no dejan un rastro tras de sí. Además, se mueve con mayor velocidad que los disparos descritos con anterioridad.

```
public class ShotGun : MonoBehaviour
{
    float speed=8;

    void Update()
    {
        if (transform.rotation.z==0)
        {
            transform.position += Vector3.up*Time.deltaTime*speed;
        }else if (transform.rotation.z<0)
        {
            transform.position += new Vector3(0.1f,1,0)*Time.deltaTime*speed;
        }else{
            transform.position += new Vector3(-0.1f,1,0)*Time.deltaTime*speed;
        }

    }
    void OnTriggerEnter2D(Collider2D collider)
    {
        if (collider.gameObject.tag=="Ball")
        {
            collider.gameObject.GetComponent<Ball>().Split();
        }
        if (collider.gameObject.tag=="Hexagon")
        {
            collider.gameObject.GetComponent<Hexagon>().Split();
        }
        if (collider.gameObject.tag!="Player" || collider.gameObject.tag!="Ladder")
        {
            Destroy(gameObject);
            ShotManager.shm.DestroyShot();
        }
    }
}
```

Figura 32 - Código de la clase ShotGun.cs.

Por último, existe un tipo de disparo que no se mueve de manera ascendente y vertical en la escena, sino que tiene forma de bola y se posiciona en el suelo. Su funcionamiento es semejante a una mina explosiva pero su finalidad es la misma que la del resto de disparos.

```
public class PiramydShot : MonoBehaviour
{

    void OnTriggerEnter2D(Collider2D collider)
    {
        if (collider.gameObject.tag=="Ball")
        {
            collider.gameObject.GetComponent<Ball>().Split();
        }
        if (collider.gameObject.tag=="Hexagon")
        {
            collider.gameObject.GetComponent<Hexagon>().Split();
        }
        if (collider.gameObject.tag!="Player" || collider.gameObject.tag!="Ladder")
        {
            Destroy(gameObject);
            ShotManager.shm.DestroyShot();
        }
    }
}
```

Figura 33 - Clase PiramydShot.cs

5.4. Capa de datos

La capa de datos es la capa que permite la persistencia de los datos en el sistema, almacenando y proveyendo a este de cualquier valor que se desee guardar en todo momento. En el caso de este videojuego, interesa almacenar el alias del jugador junto con la puntuación obtenida al finalizar la partida. Por otro lado, también es necesario almacenar otros valores para la comprobación del estado de los logros del juego.

En la pantalla mostrada en la Figura 34 se muestra el estado de los logros a nivel local alcanzados por el jugador. Para almacenar continuamente los datos, se emplea una herramienta de Unity llamada *PlayerPrefs*, la cual consiste en un sistema clave-valor, en el que la clave es el nombre del fichero donde almacenar el valor. El fichero que permite la persistencia de los datos será almacenado en el sistema local del dispositivo donde se ejecuta.

Figura 34 - Pantalla de logros.

Para el sistema de puntuaciones se ha hecho uso de un servicio externo, empleando una clasificación online a nivel mundial, ya que era uno de los requisitos iniciales del proyecto. Para ello se realizó una comparativa de diversas posibilidades, y finalmente se decidió emplear *Dreamlo* debido a su buena documentación. En ella se detalla claramente el uso de dicha herramienta y la facilidad de uso, así como la posibilidad de visualizar vía web dicha clasificación, como se observa en la Figura 35.

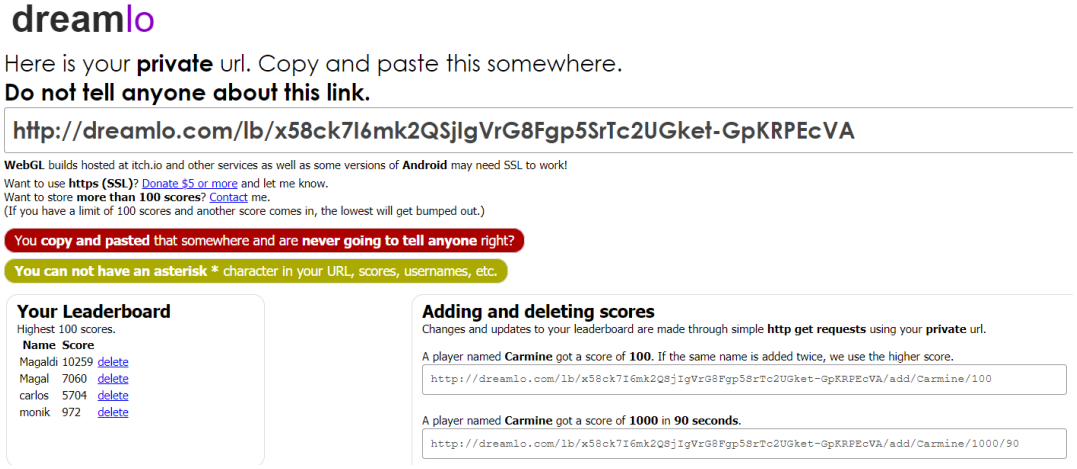

Figura 35 - Vista web de la clasificación.

Para el uso de esta herramienta, al obtener el enlace de la clasificación inicial, se generan dos claves: una publica y una privada. La clave privada permitirá hacer llamadas al servicio, enviándole la información que se desea almacenar, es decir, la puntuación y el nombre. Por otro lado, la clave pública permite obtener la información almacenada en el sistema.

La información obtenida de su servicio tiene un formato especial, por lo que es necesario la modificación de este para la correcta presentación en el videojuego.

En la Figura 36, se muestra cómo se realiza la llamada a su servicio, haciendo una petición de envió de los datos en un determinado orden. Este orden, tal y como se ha descrito anteriormente, viene perfectamente detallado en su documentación. Para ello se emplea, además, la clase *UnityWebRequest*, propia de Unity, la cual realiza la petición y retorna un error si la petición no ha sido procesada correctamente.[11]

```
public static void AddNewHS(string user, int score){
    instance.StartCoroutine(instance.UploadNewHS(user,score));
}

IEnumerator UploadNewHS(string user, int score){
    UnityWebRequest  www = new UnityWebRequest(webURL + privateCode +"/add/" + user+"/"+score);
     print(user);
      print(score);
    yield return www.SendWebRequest();
    if(www.isNetworkError) {
        Debug.Log(www.error);
    }
    else {
        Debug.Log("Upload complete!");
        DownloadHS();
    }

}
```

Figura 36 - Fragmento de código encargado del envío.

Por otro lado, en la Figura 37 se detalla la situación contraria, es decir, la petición para obtener la información almacenada, donde se establece que solamente se obtendrán las 10 mejores puntuaciones de la lista, tal y como se indicó en los requisitos funcionales.

```
public void DownloadHS(){
    StartCoroutine(DownloadHSFromDB());
}

IEnumerator DownloadHSFromDB(){

    UnityWebRequest www = new UnityWebRequest(webURL + publicCode +"/pipe/0/10");
    www.downloadHandler = new DownloadHandlerBuffer();
    yield return www.SendWebRequest();

    if(www.isNetworkError) {
        Debug.Log(www.error);
    }
    else {

        FormatHS(www.downloadHandler.text);
        hsDisplay.HSDownloaded(hsList);
    }
}
```

Figura 37 - Fragmento de código encargado de la obtención.

6. Evaluación y pruebas

6.1. Pruebas unitarias y de integración

Las pruebas unitarias, tratan de técnicas empleadas para la comprobación del correcto funcionamiento de fragmentos de código.

Durante el desarrollo de este videojuego, se han realizado una serie de simulaciones de dichas pruebas mediante mensajes en la consola. Esto es debido a la dependencia y arquitectura empleada en la generación del código, donde la única manera de comprobar el correcto funcionamiento de lo implementado es mediante una breve ejecución del juego y su comprobación manual. Por otra parte, el propio motor de Unity ya genera una serie de mensajes cuando algo no funciona correctamente.

Por otra parte, existen también las pruebas de integración, estas tratan de comprobar la correcta interacción de dos módulos cuando estos son ejecutados simultáneamente en la misma ejecución, siendo esta una situación muy cerca a la ejecución real del videojuego.
Dada la funcionalidad de Unity, podemos asemejar dichas pruebas a la ejecución del juego usando la previsualización que el motor nos proporciona ya que este permite la utilización de una determinada escena rápidamente. De esta manera, se pueden comprobar todos los elementos desarrollados a medida que se van integrando, y así detectar cualquier comportamiento erróneo de manera ágil.

Tras finalizar la total implementación del juego, para realizar la comprobación general de la correcta integración de todas las partes, se ejecutó el juego en el mismo editor, recorriendo cada escena y utilizando los componentes de estas, para asegurar el correcto funcionamiento.

6.2. Pruebas de sistema

Para verificar el comportamiento del sistema en su totalidad y así verificar los requisitos no funcionales establecidos con anterioridad, se ha hecho uso de una herramienta propia de Unity como es Unity Profiler. Dicha herramienta realiza un análisis del juego durante su ejecución en el mismo motor, calculando así los tiempos y uso de los recursos. En la Figura 38 se observan las gráficas generadas durante la ejecución del juego.

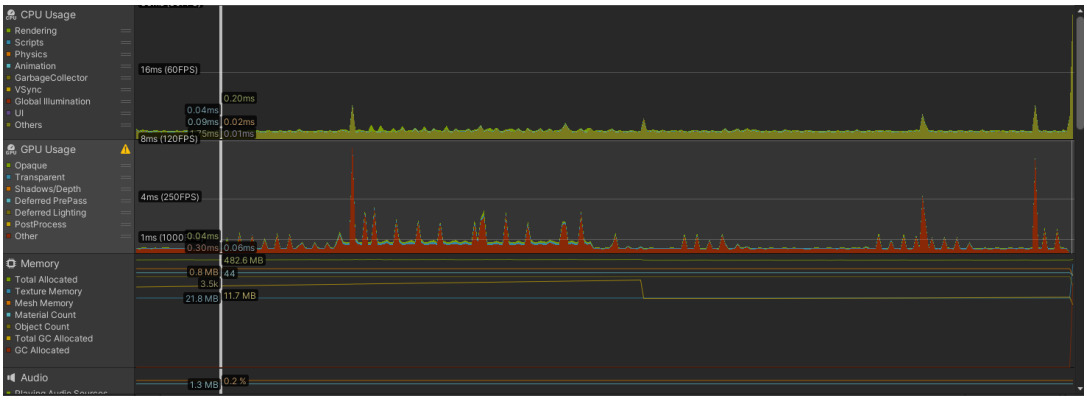

Figura 38 - Gráficas generadas por el Profiler tras ejecutar el juego.

En este proyecto se había establecido como requisito que el sistema no bajaría de los 30 FPS durante su ejecución y como se observa en la gráfica de la Figura 38, solamente hay un punto donde la ejecución descienda de los 60 FPS, llegando a un mínimo de 30 FPS, por lo que dicho requisito estaría satisfecho. Por otro lado, se

decretó que no haría uso de más de 2GB durante su ejecución. Para ello, volvemos a hacer uso de la herramienta, y como se observa en la Figura 39, donde se detalla la parte de consumo de memoria, se observa que el máximo de memoria empleado es de 1.22 GB.

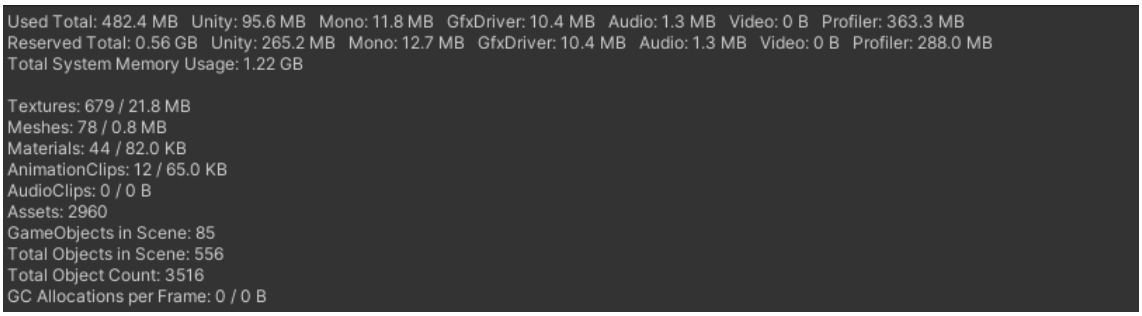

Figura 39 - Uso de la memoria durante la ejecución.

También se estableció como requisito que el juego sería apto para todos los públicos. Aunque el proceso de aprobación de este requisito sería competencia del marco europeo o americano, basándonos en los requisitos de los sistemas PEGI para Europa o ESRB para norte América, podemos determinar que el juego cumpliría las condiciones necesarias para dicha etiqueta. [12] [1]

Por último, siguiendo lo establecido, se decidió que el juego sería ejecutable tanto en Windows, como MacOS y Linux, además de en navegadores Web. Para ello se crean los archivos ejecutables en cada plataforma y se comprueba el correcto funcionamiento de estos.

6.3. Pruebas de aceptación

Una vez el que sistema estuvo completo y finalizado, se realizaron las pruebas de aceptación. El objetivo de estas pruebas no es ya buscar fallos del funcionamiento, sino que se busca una validación del sistema por parte de los usuarios.
Para realizar dichas pruebas, tras generar una versión estable del juego, fue compartido con un grupo de tres personas, para que interactuasen con el videojuego. Tras dejar varios días de uso, se pidió de vuelta las opiniones respecto al conjunto del juego y evaluación de su experiencia, para así conocer la existencia de algún defecto que no se hubiera detectado antes. Al no haber incidencias de gravedad, se determinó la finalización del videojuego.

7. Conclusiones y trabajos futuros

7.1. Conclusiones

Para hablar de las conclusiones es necesario recordar que el objetivo inicial del proyecto era la creación de una versión actual del videojuego clásico arcade Pang.

A la vista del resultado obtenido, se considera cumplido el objetivo con gran satisfacción, tras haber solventado todos los requisitos con éxito. Se ha conseguido una nueva versión del videojuego con nuevas y renovadas características, convirtiéndolo en un juego bastante completo.

Ha resultado un proyecto muy interesante, ya que se partía de una base escasa acerca de la creación de videojuegos. He sido capaz de llevar a cabo el desarrollo de un proyecto completo por mi cuenta, algo que durante el transcurso del grado no había llevado a cabo. Por otra parte, he aplicado diversas técnicas adquiridas a lo largo del grado, además, de haber aprendido muchas otras cosas a lo largo del desarrollo en lo relacionado con los videojuegos y su creación.

Por último, recalcar la satisfacción de ver un videojuego creado por uno mismo, habiendo sido amante de estos desde pequeño, todo ello gracias a lo aprendido en los últimos años.

7.2. Trabajos futuros

Respecto a los trabajos futuros, si bien es cierto que la satisfacción tras finalizar es grande, existe una infinidad de posibilidades de expandir el sistema en todos los ámbitos.

Uno de los aspectos más llamativos para futuras ampliaciones sería la posibilidad de jugar en modo multijugador de manera remota además de local. Por otra parte, sería interesante la creación de poder establecer niveles por parte del jugador, ya que es una funcionalidad que hoy en día está ganando popularidad. Así mismo, con más tiempo podría quedar más moderna y adaptada a los videojuegos actuales la opción de personalización de los avatares para los jugadores o las bolas.

Por último, debido a la arquitectura del juego, existe la posibilidad de añadir de manera relativamente intuitiva más tipos de disparos y potenciadores, proporcionando así al juego una mayor variedad de elementos y posibilidades en el desarrollo de la partida.

8. Referencias

[1] DEV. (2019). *Libro Blanco del Desarrollo Español de Videojuegos.*

[2] *García, A. (2020). La Vanguardia. Obtenido de https://www.lavanguardia.com/videojuegos/20200518/481256142459/minecraft-200-millones-ventas-millones-usuarios.html*

[3] *Digital. (2018). En Digital. Obtenido de https://en.digital/blog/videojuegos-industria-mobile-crecimiento*

[4] *GAMINGPICKS. (2014). gamingpicks.wordpress. Obtenido de https://gamingpicks.wordpress.com/author/vgamesvideos/*

[5] *Rebollo, P. (s.f.). Akademus. Obtenido de https://www.akademus.es/blog/tecnologia/desarrollo-programacion/juegos-hechos-con-unity/*

[6] *Librería Unity, GameObject. https://docs.unity3d.com/Manual/class-GameObject.html*

[7] *Librería Unity, Prefabs. https://docs.unity3d.com/Manual/Prefabs.html*

[8] *Librería Unity, MonoBehaviour. https://docs.unity3d.com/ScriptReference/MonoBehaviour.html*

[9] *Martínez, S. (2014). MundoERP. Obtenido de https://www.mundoerp.com/blog/metodologia-iterativa-o-incremental-gestion-proyectos/*

[10] Sommerville, I. (2012). *Ingeniería del Software* (Vol. 9). Addison-Wesley.

[11] *Librería Unity, UnityWebRequest. https://docs.unity3d.com/es/530/Manual/UnityWebRequest.html*

[12] *coalition, I. a. (2021). esrb. Obtenido de https://www.esrb.org/ratings-guide/*

[13] *Europea, U. (s.f.). pegi.info. Obtenido de https://pegi.info/es*