



***Facultad
de
Ciencias***

**DESARROLLO DE UN JUEGO SOBRE
GESTIÓN DE PROYECTOS SOFTWARE
(Development of a game about management
of software projects)**

Trabajo de Fin de Grado
para acceder al

GRADO EN INGENIERÍA INFORMÁTICA

Autor: Mario Díaz Santos

Director: Carlos Blanco Bueno

Febrero – 2019

Índice de contenido

Agradecimientos	8
Resumen	9
Abstract	10
1. Introducción	11
1.1 Estado del Arte.....	11
1.2 Motivación	13
1.3 Objetivo	13
2. Herramientas, tecnologías y materiales utilizados	14
2.1 Herramientas.....	14
2.1.1 Unity	14
2.1.1.1 Interfaz de Unity	15
2.1.2 MonoDevelop.....	16
2.1.3 Adobe Illustrator CS6.....	16
2.1.4 Tiled Map Editor.....	16
2.2 Tecnologías.....	17
2.2.1 C#.....	17
2.2.2 Photon Engine	17
2.2.3 XML	19
2.2.4 Librerías de Unity.....	19
2.2.5 Android SDK	20
2.3 Materiales.....	20
3. Metodología	21
3.1 Metodología iterativa incremental.....	21
3.2 Planificación	22
3.2.1 Etapa de formación.....	22
3.2.2 Etapa de desarrollo.....	23
3.2.3 Etapa de integración	24
3.2.4 Funcionalidades por iteración	24
3.2.4.1 Iteración 1	24
3.2.4.2 Iteración 2	24
3.2.4.3 Iteración 3	24
3.2.4.4 Iteración 4	25
3.2.4.5 Iteración 5	25
3.2.4.6 Iteración 6	25
3.2.4.7 Iteración 7	25
3.2.4.8 Iteración 8	25
3.2.4.9 Iteración 9	26
3.2.4.10 Iteración 10.....	26
4. Análisis de requisitos	26
4.1 Requisitos funcionales.....	26
4.2 Requisitos no funcionales.....	27
5. Diseño e Implementación.....	28
5.1 Arquitectura del sistema	28
5.2 Capa de presentación	30
5.2.1 Interfaz de usuario	30
5.2.1.1 Interfaz del juego.....	30
5.2.1.2 Interfaz de menús.....	34
5.2.2 Escenas.....	35
5.3 Capa de negocio	36
5.3.1 Estructura del proyecto	36
5.3.2 Game Manager.....	38
5.3.3 Tablero	40
5.3.4 Servidor Photon	40

5.3.5 Sumario	42
5.4 Capa de datos.....	44
5.4.1 Ficheros XML.....	44
5.4.2 Ficheros csv.....	46
5.4.3 Clasificación.....	47
6. Evaluación y Pruebas	48
6.1 Pruebas unitarias y de integración	48
6.2 Pruebas de sistema	50
6.2.1 Pruebas de portabilidad.....	50
6.2.2 Pruebas de compatibilidad	50
6.2.3 Pruebas de rendimiento	50
6.3 Pruebas de aceptación	52
7. Conclusiones y trabajos futuros	53
7.1 Conclusiones.....	53
7.2 Trabajos futuros	53
8. Referencias.....	54

Índice de figuras

Figura 1 – Dragon Box Elements	12
Figura 2 – Interfaz Unity.....	15
Figura 3 – Servidor Photon	18
Figura 4 – Analizador de rendimiento	19
Figura 5 – Esquema iterativo incremental	21
Figura 6 – Retarded Bird.....	23
Figura 7 – Arquitectura en tres capas	29
Figura 8 – Tablero físico	30
Figura 9 – Interfaz usuario del juego digital.....	31
Figura 10 – Reporte de actuación	32
Figura 11 – Formulario de plan de proyecto.....	32
Figura 12 – Ventana modal de reconfiguración.....	33
Figura 13 – Paleta colores de la interfaz	33
Figura 14 – Mockup del menú principal.....	34
Figura 15 – Mockup lobby.....	35
Figura 16 – Mockup sala.....	35
Figura 17 – Segundo tablero.....	40
Figura 18 – Sumario del juego	43
Figura 19 – Diagrama UML de la herencia de Riesgo.....	46
Figura 20 – Clasificación en dreamlo	47
Figura 21 – Unity Profiler	51
Figura 22 – Información de uso de memoria	51
Figura 23 – Información de uso de CPU	52

Índice de tablas

Tabla 1 – Planning del proyecto.....	22
Tabla 2 – Requisitos funcionales	27
Tabla 3 – Requisitos no funcionales	28
Tabla 4 – Jerarquía de scripts desarrollados	37

Índice de código

Código 1 – Patrón Singleton (GameManager.cs).....	38
Código 2 – Método TirarDado (GameManager.cs)	39
Código 4 – LobbyNetwork.cs	41
Código 5 – Fragmento de PlayerNetwork.cs.....	41
Código 6 – DDOL.cs.....	42
Código 7 – Actualización de puntuaciones.....	43
Código 8 – ContenedorEmpleado.cs	45
Código 9 – Estructura del XML de empleados	45
Código 3 – Fichero .csv del tablero.....	46
Código 10 – Prueba Test Runner.....	49

Agradecimientos

Me gustaría aprovechar estas primeras líneas para agradecer. Agradecer porque realmente estoy agradecido. Agradecer a todas esas personas que han estado ahí tanto en los buenos momentos como en los momentos menos buenos, sobre todo en esos momentos.

Primeramente, quiero darle las gracias a mi familia, sobre todo a mis abuelos, por creer en mí, por apoyarme en todo momento y por animarme al grito de "Tú puedes con todo". Si soy lo que soy se lo debo a ellos más que a nadie. Lo único que les puedo reprochar es que querían un médico en la familia. Lo siento.

Especial mención también a mis hermanas, Paula y Lucía. Lo nuestro es una montaña rusa de emociones, a veces les quiero y a veces les odio pero son lo mejor de este mundo. También quiero darle las gracias a mi padre, allá donde estés, se que estarás orgulloso. Ha pasado mucho tiempo, pero te sigo teniendo presente.

A mis amigos de toda la vida, Kevin y Aaron, por escuchar mis problemas e intentar comprenderme. Por darme los mejores consejos y sacarme siempre una sonrisa. Porque desde pequeños no nos hemos separado y estoy convencido, sin miedo a equivocarme, que estarán siempre que los necesite, lo mismo que yo para ellos.

Gracias a mis compañeros de clase de la universidad, a toda esa gente de bien con los que me he divertido y he pasado estos buenos años. Con algunos incluso he viajado. Quizás con muchos no vuelva a hablar, por distancia o simplemente pérdida de contacto, pero espero que les vaya bien donde quiera que estén.

Por último y no menos importante, muchas gracias a todos los profesores de la carrera porque, además de ser grandes docentes, de todos ellos he aprendido algo, que ya es bastante. Quisiera darle gracias sobre todo a Carlos, al que considero ya un amigo, por sembrar en mi la curiosidad sobre este mundo que es el desarrollo de videojuegos, por ayudarme en todo lo que he necesitado, pero sobre todo por creer en mi al ofrecerme este proyecto tan ambicioso.

Un mensaje también para todas las personas que puedan estar leyendo este documento. Gracias por leerme y espero que podáis encontrar en él algo. Ayuda, información relevante o mera curiosidad.

Resumen

La gestión de proyectos es un área de la informática con enorme importancia. Todo desarrollo de un proyecto conlleva principalmente una gestión de las actividades a realizar, costes, plazos y posibles riesgos. Todo debe estar al día, bien planificado y evaluado para detectar y corregir posibles errores no deseados. En definitiva, es indispensable que un proyecto tenga una buena planificación y seguimiento para que no sea considerado, en el futuro, un auténtico fracaso.

Por otra parte, encontramos la industria de los videojuegos. Un mercado en ferviente crecimiento que parece no tener barreras. Con unas ganancias registradas, el último año, de 108,4 billones de dólares en todo el mundo y millones de jugadores distribuidos por todo el planeta.

Unir la importancia de la gestión de proyectos con la popularidad de los videojuegos es la clave para este proyecto. En él lo que se pretende es darle un formato digital a un conocido juego de mesa llamado *Deliver* sobre la gestión de proyectos software y a la vez, introducir mejoras sobre el mismo.

Palabras Clave: Gestión de proyectos, videojuego, Deliver, Unity.

Abstract

Project management is an area of informatics with huge importance. All development of a software activity carries, mainly, a management of activities to make, costs, deadlines and possible risks. Everything must be up to date, well planned and evaluated for detecting and correcting to possible non-desired errors. In conclusion, it is essential that a project have a good planification and tracking for not being considered, in the future, a real failure.

On the other hand, we find the videogame industry. A market in fervent growing that seems to have no barriers. With registered earnings, last year, of 108.4 billion dollars worldwide and millions of players distributed all over the planet.

Linking the importance of project management with the popularity of video games is the key for this project. In it what is intended is to give a digital format to known board game called Deliver about management of software projects and in the same time, introduce improvements on it.

Keywords: Project management, videogame, Deliver, Unity.

1. Introducción

La gestión de proyectos busca principalmente planificar y controlar de forma adecuada los principales aspectos relacionados con el desarrollo de un proyecto, como el alcance, tiempos, costes, riesgos o calidad. En este sentido encontramos como principal referencia el cuerpo de conocimientos de gestión de proyectos (PMBOK) que propone una serie de procesos considerados como buenas prácticas que buscan maximizar las posibilidades de éxito del proyecto.

En el otro extremo, encontramos la industria de los videojuegos. Un ente imparables que busca ofertar diversión en todo su público. Acompasada a la vertiente tradicional del desarrollo de videojuegos ha surgido otra rama, con menos mercado, aunque mucho más interesante, relacionada con los *serious games* (juegos serios). Los juegos serios son juegos cuyo objetivo principal no es el mero entretenimiento sino el aprendizaje y la práctica de habilidades por parte del usuario. Estos juegos implementan un método de enseñanza que se conoce como *game-based learning*. Se trata de una tendencia que se está expandiendo en escuelas, universidades y grandes empresas.

1.1 Estado del Arte

Actualmente, el uso de los juegos serios ha crecido sobre todo en sectores como la educación, la defensa, la aeronáutica y la salud. Una de sus principales ventajas es que la funcionalidad que enseñan puede ser de lo más variada: desde entrenar pilotos, la capacitación de un equipo de ventas o la enseñanza de idiomas, por poner algunos ejemplos.

El método de aprendizaje que utiliza este tipo de videojuegos es lo que se conoce como *game-based learning*. La clave está en presentar los contenidos que quieren ser enseñados a través del juego y no de una clase, libro o cualquier otro modo de enseñanza.

Realizar un juego serio no es sencillo. La mayoría de los sistemas de este tipo presentan cinco elementos que aseguran un buen funcionamiento y efectividad. Esos elementos son:

- **Una historia:** No se trata de un elemento imprescindible, pero, como es obvio, al tratarse de un videojuego, a menudo suele haber una trama principal. Esta historia facilita que el usuario se sienta mucho más atraído a jugar.
- **Gamificación:** Se trata de las dinámicas de motivación del juego. Entre ellas se incluyen rankings, recompensas o sistemas de puntos que suele motivar a los jugadores a seguir jugando gracias a la competencia o la propia motivación para superarse a uno mismo.

- **Feedback inmediato:** A diferencia de las clases donde hay muchos alumnos y un sólo profesor para resolver dudas, por ejemplo, los juegos serios ofrecen una retroalimentación inmediata. El jugador interactúa directamente con el juego y recibe información, una recompensa o incluso un castigo al instante.
- **Simulación:** En la mayoría de los casos, los juegos serios simulan situaciones que podrían darse en la vida real. Todo esto a través de la recreación de ambientes o la creación de personajes.
- **El aprendizaje como objetivo:** No se puede obviar que el objetivo de los juegos serios es que el usuario aprenda algo. Todos los elementos anteriores se pueden encontrar en un juego comercial. Es en este punto donde realmente se distinguen este tipo de juegos con los videojuegos serios. Además de los elementos anteriores, los juegos serios no tienen una finalidad lúdica sino de enseñanza.

En la figura 1 se puede ver la pantalla de uno de los niveles del juego Dragon Box Elements que es un juego serio cuyo objetivo es la construcción de un ejército para derrotar al malvado dragón Osgard y salvar a la isla de Euclides. Este juego está pensado para niños y lo que busca es que los pequeños aprendan las bases de la geometría euclídea. El juego permite a los jugadores aprender matemáticas mientras se divierten jugando.

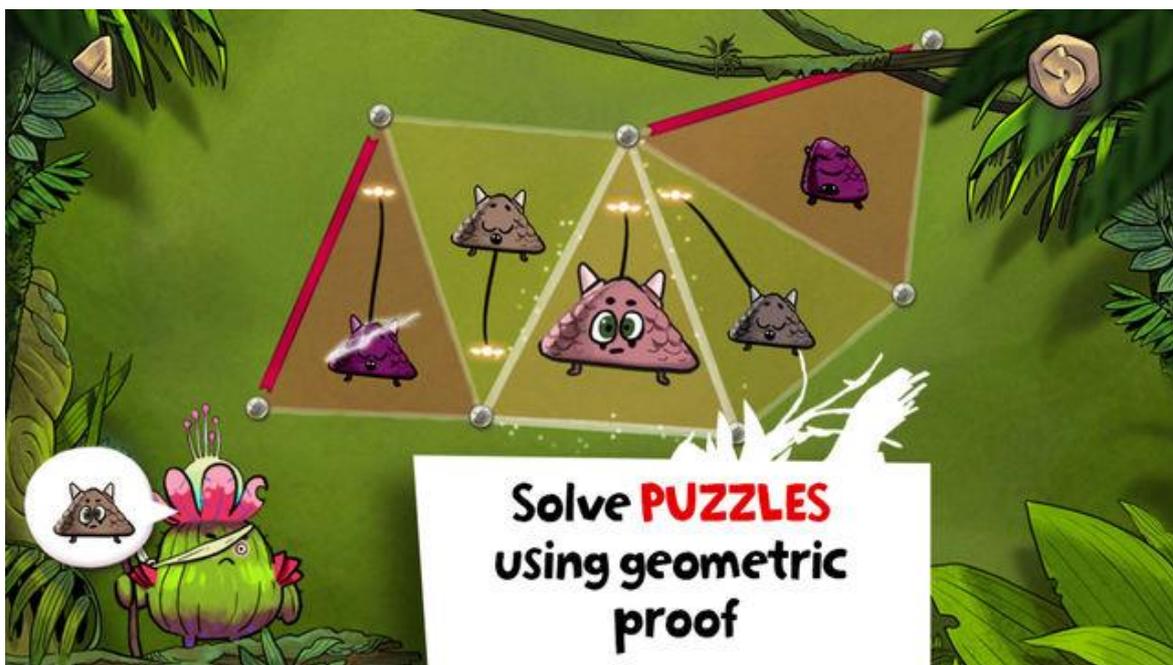


Figura 1 – Dragon Box Elements

1.2 Motivación

Es por todos conocida la importancia de la gestión de proyectos y la cantidad de personas que deben saber sobre ella. Ahí radica la motivación principal de este proyecto que no es otra que la creación de un juego serio que permita tanto a alumnos de asignaturas que traten la gestión de proyectos como a empresas que quieran formar a sus empleados en esta área a aprender a gestionar los recursos, costes y riesgos durante el desarrollo de un producto software.

Como motivación personal está el demostrarme a mí mismo que puedo hacer un juego completo; sólo y en un tiempo no muy largo. Para trabajar más adelante con otras herramientas de desarrollo de videojuegos disponibles en el mercado como Unreal Engine e incluso con lenguajes en los que no se he sido instruido como C++. También me ayudará, este desarrollo, a fomentar mi interés en aprender más sobre diseño gráfico. En definitiva, lo que busco es desarrollar un primer juego que me permita aprender un motor de videojuegos y que me abra la puerta a aprender muchos más.

1.3 Objetivo

Deliver es un juego muy famoso para la gestión de proyectos. Es de tablero y ayuda a entender procesos en varias dimensiones de la gestión de proyectos como el alcance, los tiempos, los costes o los riesgos, así como el concepto de valor ganado.

Este juego tiene algunos inconvenientes al aplicarse en clase como el rígido uso de los materiales del juego (tarjetas, fichas, dado...), la necesidad de disponer de dichos materiales y un lugar para desplegar el juego o que todos los jugadores tengan que estar alrededor del mismo tablero.

Como objetivo principal se plantea hacer una versión digital del juego *Deliver*, un juego serio, que traslade el juego original de tablero a una versión digital y además introduzca mejoras que permitan un mejor uso en las aulas. Algunos ejemplos de estas mejoras son: variedad en los tableros, variedad de fichas de recursos, distintas configuraciones de las reglas de una partida, multidispositivo y multijugador entre distintos tipos de dispositivos.

El objetivo secundario, para el propio desarrollador, está en la realización de un desarrollo software que permita trabajar con todos los conocimientos aprendidos en la carrera como el empleo de una metodología software y una ejecución clara y bien definida de cada una de las fases del desarrollo.

2. Herramientas, tecnologías y materiales utilizados

Este apartado tiene como objetivo comentar las herramientas, tecnologías y materiales involucrados durante el desarrollo del proyecto, y ofrecer una visión inicial del trabajo realizado.

2.1 Herramientas

En primer lugar, se muestran las herramientas software utilizadas para la consecución de este proyecto. Entre ellas se encuentran principalmente el motor de juegos Unity junto con MonoDevelop como IDE de desarrollo y herramientas adicionales para la creación de mapas basados en casillas (tiles) y el diseño de los aspectos gráficos.

2.1.1 Unity

Unity es un motor para la creación de videojuegos multiplataforma creado por Unity Technologies. Fue lanzado inicialmente en 2005 y la última versión estable es de diciembre de 2018. Inicialmente era un motor para ejecutar sobre MacOS, aunque hoy en día puede ser ejecutado sobre otros sistemas operativos como Microsoft Windows o Linux.

Esta herramienta puede ser empleada tanto para crear juegos en dos dimensiones como en tres dimensiones. También tiene otros usos como la creación de aplicaciones de realidad virtual y realidad aumentada o el desarrollo de aplicaciones móviles para numerosas plataformas.

Actualmente Unity cuenta con cuatro tipos de licencias para aquellas personas que quieran hacer uso del motor: licencia Personal, licencia Plus, licencia Pro y licencia Enterprise. Para la realización del proyecto se ha empleado la licencia Personal porque es la única gratuita. Esta licencia establece varios parámetros como el límite de beneficio económico que se puede recibir por la aplicación (100.000\$) o el número máximo de usuarios concurrentes en un juego multijugador (20 usuarios). Como es obvio, a medida que se usa una licencia de mayor rango, Unity ofrece mejores funcionalidades, aunque también recibe un beneficio económico mayor que deberá abonar el desarrollador.

En el campo de la creación de videojuegos, Unity es uno de los principales motores. Ha sido usado para la creación de videojuegos de éxito como *Hearthstone* o *Assasins's Creed: Identity*. Desde hace años se ha convertido en la opción preferida para nuevos desarrolladores principalmente porque ofrece una intuitiva interfaz, soporte para varios lenguajes de programación (C#, JavaScript y Boo), gran cantidad de funcionalidades y *assets*, muchos de ellos gratuitos, y una gran comunidad de desarrolladores que hace que estén disponibles una enorme cantidad de recursos. Unity además posee un inmejorable soporte y una documentación muy clara.

2.1.1.1 Interfaz de Unity

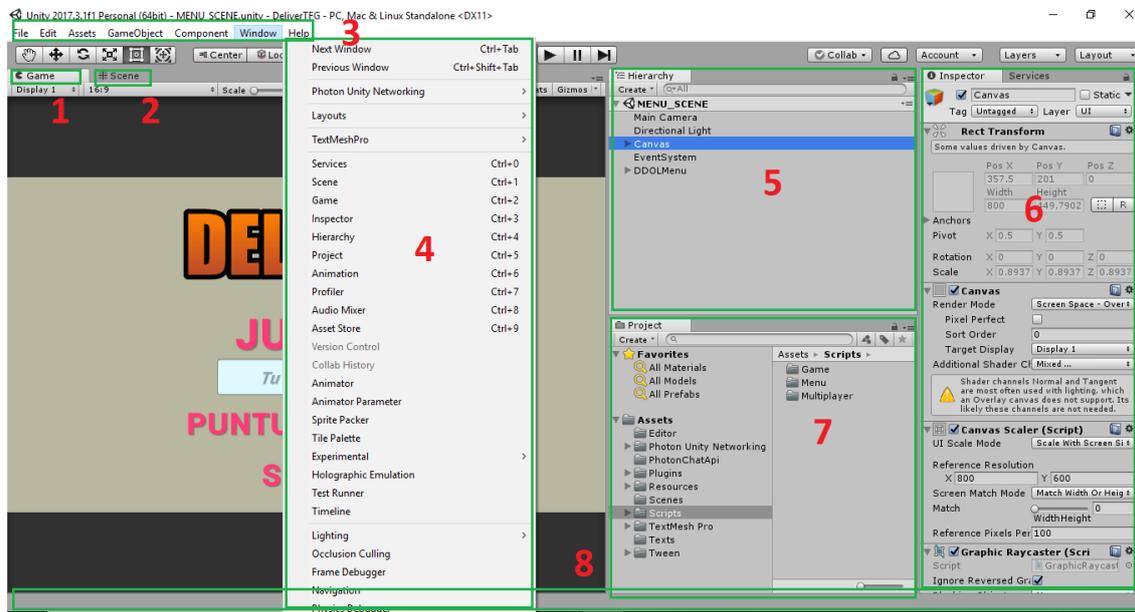


Figura 2 – Interfaz Unity

En la figura 2 se puede observar la interfaz de Unity. Esta interfaz tiene varias peculiaridades y gran cantidad de elementos a considerar. Aun así, es bastante intuitiva. Algunas de las partes más importantes se pueden observar de manera numerada.

El número 1 es la ventana donde se puede observar el juego en ejecución. En la ventana 2 se observa la disposición del juego, es decir, la interfaz (capa de presentación). A diferencia de la ventana del juego, la ventana de escena es manipulable.

Numerado como 3 encontramos el menú de opciones. Y como 4, las diferentes ventanas que se pueden desplegar en la interfaz, algunas de las más importantes, de las que no se encuentran activas en la imagen, son el *asset store* o el *test runner*. Ambas usadas durante el desarrollo, la primera para importar paquetes y la segunda para realizar las pruebas unitarias y de integración.

El número 5 es la jerarquía de la escena. En su interior aparecen los elementos que se contienen en ella. El inspector del GameObject es el número 6. En él se puede, entre otras acciones, configurar parámetros como la posición o añadir distintos componentes al objeto como imágenes, *scripts*, o *colliders*.

Por último, encontramos el elemento número 7 que correspondiente a la jerarquía de archivos del proyecto y el número 8 que se trata de una consola desplegable para observar el correcto funcionamiento del juego.

2.1.2 MonoDevelop

MonoDevelop es un entorno de desarrollo gratuito que viene incluido como editor de código predeterminado en Unity. Integra características similares a otros IDEs como NetBeans o Microsoft Visual Studio y aunque fue diseñado inicialmente para el desarrollo de código escrito en C# también soporta otros lenguajes como Boo, JavaScript, C, C++ o Java.

Al igual que Unity es multiplataforma, funciona sobre Windows, Linux y MacOS. Entre sus características destacan la capacidad de realizar *debug* gráfico, autocompletado de código, una interfaz sencilla y configurable, gestión de soluciones y la posibilidad de incluir *plugins* para programar en otros lenguajes.

2.1.3 Adobe Illustrator CS6

Adobe Illustrator es un editor de gráficos vectoriales desarrollado por Adobe Systems lanzado inicialmente en 1987 con última versión estable de 2017. Junto con Adobe Photoshop forma el servicio básico de diseño gráfico de Adobe llamado *Adobe Creative Cloud*.

Illustrator es una aplicación informática especialmente dedicada al dibujo vectorial y al diseño de elementos gráficos, pudiendo ser usado en proyectos de diseño editorial, dibujo profesional, maquetación web o gráficos para móviles y videojuegos.

Las imágenes vectorizadas con las que trabaja este programa se componen de puntos en un espacio virtual que se van uniendo por medio de trazados, para ser rellenados posteriormente y así obtener imágenes de gran calidad que tienen coherencia a cualquier tamaño.

En la gran calidad de imagen que aporta radica la principal característica de este programa y es por eso por lo que ha sido utilizado para poder diseñar algunos de los elementos de la interfaz del juego.

2.1.4 Tiled Map Editor

Tiled es un editor de nivel 2D que ayuda a desarrollar el contenido de un juego. Su función principal es permitir diseñar y editar mapas de mosaicos, de forma visual, de varias formas. Tiled se enfoca en la flexibilidad general mientras trata de mantenerse intuitivo.

Tiled es gratuito y multiplataforma y en él se puede configurar las dimensiones del nivel (número de casillas y tamaño) y su diseño. Los niveles creados pueden ser exportados en distintos formatos como: txml, csv o json.

En este proyecto se crearon varios ficheros con extensión csv y dimensiones 40x8 para simular distintos tableros empleados en el juego.

2.2 Tecnologías

En este apartado se comentan las tecnologías usadas durante el desarrollo de este proyecto. Entre ellas destacan C# como lenguaje de programación de desarrollo de videojuegos, XML como lenguaje para la persistencia de datos y Photon Engine como servidor para gestionar el modo multijugador.

2.2.1 C#

C# es un lenguaje de programación diseñado y estandarizado por Microsoft como parte de su plataforma .NET. Su sintaxis básica deriva de C/C++ y es muy similar a Java. Como este último, se trata de un lenguaje orientado a objetos.

C# es considerado como una evolución necesaria de C y C++. Algunas de las características de este lenguaje de programación son: sencillez (en comparación con sus antecesores), modernidad, seguridad (un mecanismo muy fuerte para la seguridad de los objetos), extensibilidad (puedes añadir tipos de datos básicos, operadores y modificadores), su versionabilidad (es decir, tiene versiones y se actualiza constantemente) y su compatibilidad, tanto con sus antecesores como con Java.

Hay que mencionar que en la industria de los videojuegos hay dos lenguajes que despuntan sobre el resto, C# y C++. Se usa generalmente el primero cuando lo que se busca es sencillez y el segundo para grandes juegos que requieren una buena gestión de recursos como la memoria.

2.2.2 Photon Engine

Photon Engine es un servicio totalmente administrado (SaaS) de servidores locales de Photon que se ejecutan en muchas regiones del planeta, listos para el juego multijugador de baja latencia en todo el mundo.

Photon proporciona una API que permite a los jugadores coincidir en una sesión de juego compartida llamada "sala", así como transferir mensajes e información en tiempo real, entre los jugadores conectados. Una de las características de Photon es que permite interactuar entre sí a los jugadores sin importar la plataforma desde la que están conectados.

Para hacer uso del servicio se debe crear una cuenta en la página web de Photon y crear una aplicación. Una vez creada esta aplicación, se debe descargar el *asset* de Photon desde Unity e introducir el identificador de la aplicación creada para gestionar la conexión al servidor.

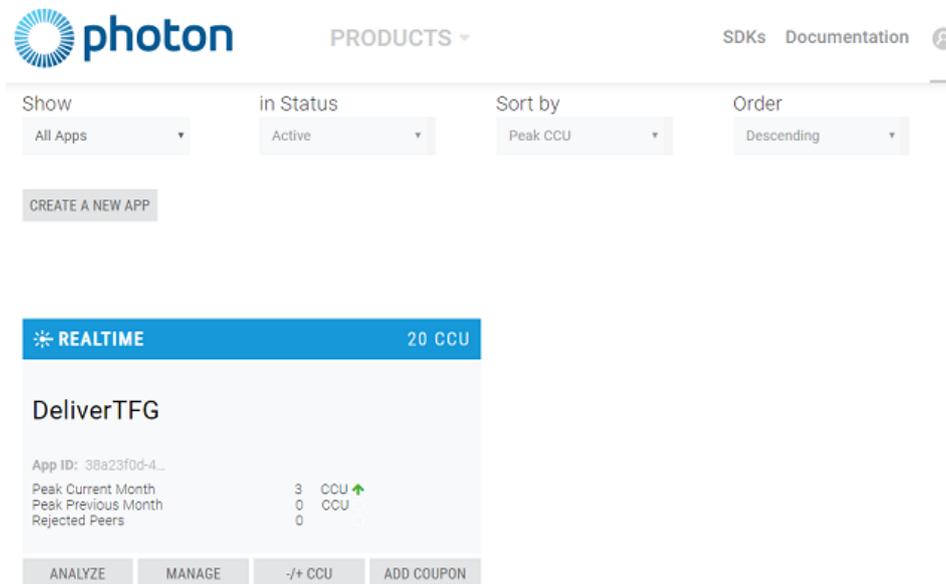


Figura 3 – Servidor Photon

En la figura 3 se puede observar la aplicación creada, la cual servirá como *host* para todos aquellos usuarios que hagan uso del juego. Se puede observar que Photon ofrece, también, información como el pico máximo de usuarios tanto en este mes como en el anterior.

Aparte del número máximo de usuarios, Photon ofrece herramientas que permiten, entre otras cosas, analizar el tráfico en el servidor. En el panel, que describe la figura 4, se pueden observar datos sobre el continente desde donde se conectan los usuarios, el número máximo de usuarios concurrentes o el número máximo de salas creadas, entre otras características.

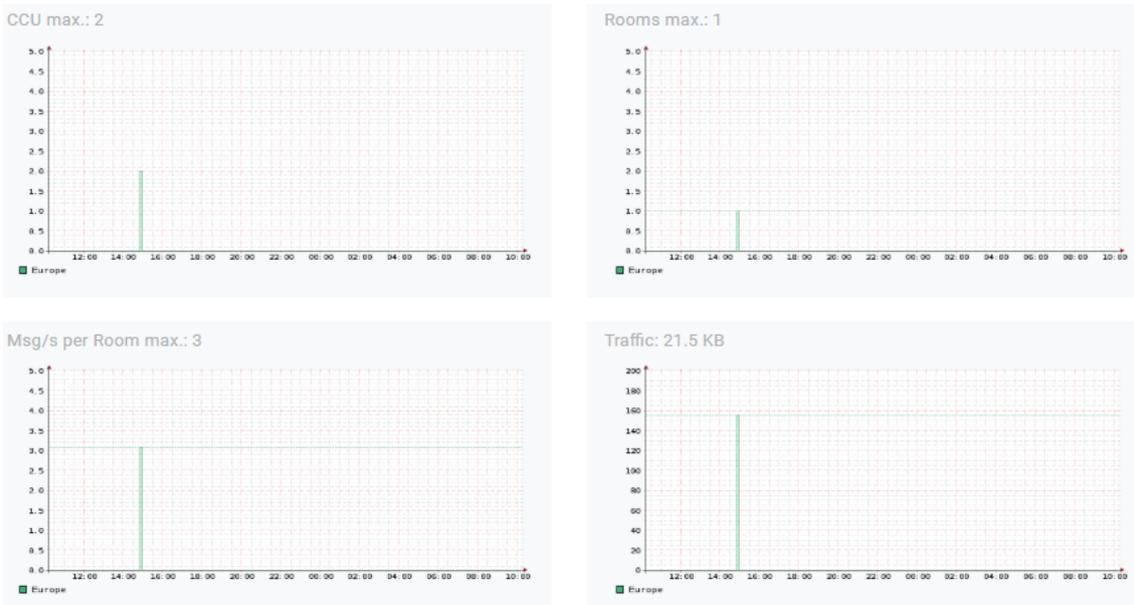


Figura 4 – Analizador de rendimiento

2.2.3 XML

XML es un lenguaje de marcado similar a HTML desarrollado por el World Wide Web Consortium (W3C). Proviene del lenguaje SGML y su primera versión data de 1998. XML sirve para representar información estructurada de modo que pueda ser almacenada, transmitida, procesada, visualizada e impresa por diversos tipos de aplicaciones y dispositivos.

Las ventajas de XML frente a otros tipos de almacenamiento estructurado son que los documentos son fácilmente procesables, se separa radicalmente el contenido y el formato de presentación y está diseñado para cualquier lenguaje o alfabeto.

Este lenguaje se usa principalmente en aplicaciones que requieren publicar e intercambiar contenidos de bases de datos, formatos de mensaje para comunicación entre diversas aplicaciones o descripción de metadatos.

2.2.4 Librerías de Unity

Se emplearon para el desarrollo del proyecto varias librerías que proporciona Unity como parte de su *asset store*. Dichas librerías son las siguientes:

Text Mesh Pro, que permite darles a los juegos una interfaz más vistosa empleando para ello fuentes y formatos de texto más avanzados que los que ofrece la interfaz por defecto del programa. Es fácil de usar y hace uso de técnicas avanzadas de representación de textos.

Tween, un pequeño paquete que incluye un único archivo de código. La gran ventaja del paquete reside en que haciendo uso de él se permite que los objetos que conforman el juego se muevan a través de la escena cambiando sus propiedades (posición y rotación) de manera fluida.

Unity Test Runner era una librería antiguamente externa al propio programa, al igual que las dos anteriores, aunque en las versiones más recientes de Unity se ha incorporado plenamente. El objetivo de esta librería es ayudar al desarrollador a probar el software realizado.

Unity Profiler, al igual que la librería anterior, está incluido en el propio programa y su objetivo es la representación de la gestión de recursos (CPU y memoria entre otros) de los que hace uso el juego durante su ejecución. Se uso esta librería para realizar las pruebas de sistema.

2.2.5 Android SDK

Es un conjunto de librerías para el desarrollo software desarrollado por Google de licencia gratuita cuyo objetivo es la creación de aplicaciones para dispositivos Android. Entre las herramientas de desarrollo que lo componen destacan un depurador de código, un simulador de teléfono y documentación y tutoriales de ayuda. Este paquete ha sido utilizado para la creación de la aplicación móvil y tablet del juego.

2.3 Materiales

En cuanto a la utilización de materiales previamente creados es importante mencionar que se ha hecho uso, principalmente, de los elementos del juego físico *Deliver*, creado por Christiane Gresse von Wangenheim, a los cuales tenía acceso.

A la hora de implementar el juego se han recogido las reglas del juego de tablero y además en la versión digital se han introducido algunas mejoras.

3. Metodología

En este apartado se comenta la metodología usada para la realización del proyecto. Dicha metodología ha sido la iterativa incremental.

3.1 Metodología iterativa incremental

La metodología iterativa incremental es una metodología de desarrollo software que se basa en la realización de varias iteraciones implementando funcionalidades diversas en cada una de ellas (Sommerville, 2012). Lo más importante de esta metodología es que permite realizar una evolución sostenida del proyecto.

Cada iteración funciona como un desarrollo de software independiente con sus fases de análisis, diseño, implementación y pruebas. Las iteraciones no tienen por qué tener la misma duración, ni una cantidad de tareas a realizar similar. Por ello tuvo que planificarse previo a empezar el proyecto el número de iteraciones que se iban a realizar, para el cumplimiento de los plazos, con su duración estimada, así como la funcionalidad que se iba a implementar en cada iteración.

En la figura 5 puede observarse el funcionamiento de la metodología iterativa incremental.

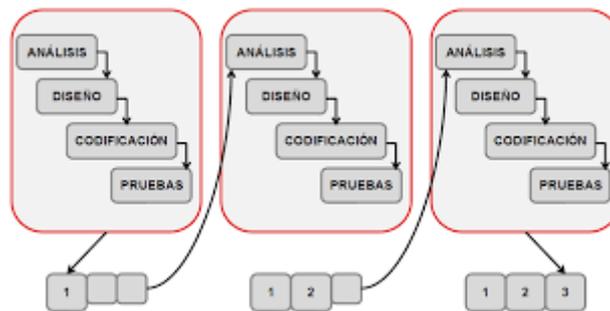


Figura 5 – Esquema iterativo incremental

Las ventajas que ofrece este desarrollo con respecto a otros son que el coste de introducir nuevos requisitos es reducido, es más fácil obtener impresiones sobre cómo va el proyecto y ofrece gran flexibilidad ante posibles cambios.

Se valoró también la utilización de una metodología ágil, como Scrum, pero al final se rechazó la idea debido a que suponía una dificultad mayor su seguimiento, principalmente por tratarse un proyecto desarrollado por una sola persona.

3.2 Planificación

En la tabla 1 se pueden observar el número de iteraciones planificadas, así como las fechas de inicio y fin de cada iteración, y también de las etapas de formación e integración. Más adelante se especificará la funcionalidad implementada en cada iteración.

ETAPAS	FECHA DE INICIO	FECHA DE FIN
ETAPA DE FORMACIÓN	08/10/2018	21/10/2018
ITERACIÓN 1	22/10/2018	04/11/2018
ITERACIÓN 2	05/11/2018	11/11/2018
ITERACIÓN 3	12/11/2018	18/11/2018
ITERACIÓN 4	19/11/2018	25/11/2018
ITERACIÓN 5	26/11/2018	02/12/2018
ITERACIÓN 6	03/12/2018	16/12/2018
ITERACIÓN 7	17/12/2018	30/12/2018
ITERACIÓN 8	31/12/2018	06/01/2019
ITERACIÓN 9	07/01/2019	20/01/2019
ITERACIÓN 10	21/01/2019	27/01/2019
ETAPA DE INTEGRACIÓN	28/01/2019	15/02/2019

Tabla 1 – Planning del proyecto

3.2.1 Etapa de formación

Antes de empezar con el desarrollo del proyecto que describe en el presente documento se llevó a cabo una etapa de formación de dos semanas para instalar en los equipos de trabajo todo el software necesario y familiarizarse con algunas de las herramientas utilizadas.

Durante esta etapa de formación se realizaron algunos bocetos en Adobe Illustrator dada la poca experiencia que tenía con herramientas de diseño gráfico y también se dieron los primeros pasos en Unity.

Siguiendo varios tutoriales (Arrijoa, 2013, Gibson, 2014), y estudiando los elementos del motor creé mi primer juego. Este juego es una copia sencilla del famoso videojuego Flappy Bird, bautizado como Retarded Bird, cuya ejecución se puede observar en la siguiente imagen.



Figura 6 – Retarded Bird

3.2.2 Etapa de desarrollo

La etapa de desarrollo la conforman las diferentes iteraciones especificadas en la tabla 1, las cuales seguían la estructura que define la metodología iterativa incremental anteriormente explicada. Al tratarse de un desarrollo software las fases son las comunes a cualquier desarrollo de este tipo, es decir; análisis, diseño, implementación o codificación y pruebas.

- **Análisis:** Es la primera etapa del desarrollo. Durante cada iteración se determinó en esta fase los hitos que debían alcanzarse para considerar realizado el trabajo correspondiente, las herramientas necesarias para llevarlo a cabo y cómo guiar el proceso de producción de código.
- **Diseño:** En la etapa de diseño se tomaban decisiones enfocadas hacia el cumplimiento de los hitos descritos en la fase anterior. Esas decisiones se tomaban con respecto, principalmente, a la apariencia física del juego (la interfaz), la implementación de código que se iba a llevar a cabo o cómo iban a interactuar los distintos módulos o elementos del software a desarrollar.
- **Implementación:** Fase de generación de la lógica del juego, la capa de presentación y los recursos persistentes.
- **Pruebas:** En la fase de pruebas se comprobaba el correcto funcionamiento de la funcionalidad implementada correspondiente a la iteración. Estas pruebas consistían, a grandes rasgos, en que el juego funcionara como debiera tras varias ejecuciones.

3.2.3 Etapa de integración

En esta última etapa se integraron algunas funcionalidades extra. Se comprobó el correcto funcionamiento del juego en su totalidad, se llevaron a cabo algunas pruebas que quedaron pendientes y se cambió el aspecto de la interfaz para que fuera más intuitiva y completa.

Una vez terminado lo anteriormente mencionado se procedió a redactar de manera formal la documentación del proyecto.

3.2.4 Funcionalidades por iteración

En este apartado se describe de manera resumida el trabajo realizado en cada una de las diez iteraciones del proyecto.

3.2.4.1 Iteración 1

Esta iteración se dedicó a realizar el prototipo básico del juego. Ese prototipo debía tener un ajuste de la interfaz completa del juego, incluir la implementación de las clases de negocio básicas como el *game manager* o el tablero, permitir cargar el tablero desde un fichero csv y ejecutar correctamente la secuencia de partida para un jugador moviendo su ficha a través del tablero según la puntuación del dado.

3.2.4.2 Iteración 2

En esta iteración se realizó la gestión de los hitos. Al llegar a uno se paraba y se pasaba el turno al siguiente jugador (el único que había). Se implementó también el riesgo de parar un turno, la inclusión de un nuevo jugador, así como una primera versión de la gestión por turnos.

3.2.4.3 Iteración 3

La tercera iteración se dedicó a aumentar las clases básicas anteriormente realizadas. Para la clase *GameManager* se buscó implementar el patrón singleton así como definir y almacenar los distintos objetos de los que iba a hacer uso. Se realizó también una primera versión de la clase *Jugador* que poseyera algunos atributos como el array de empleados, la suma de la productividad o la posición actual en el tablero. Además, se creó también la clase *Empleado*.

3.2.4.4 Iteración 4

Se trabajó en su totalidad con los empleados y se realizaron principalmente dos cometidos. Llevar a cabo la selección de empleados iniciales de manera aleatoria entre los empleados disponibles, almacenados en un fichero XML, y la creación de una ventana modal que se lanzara al llegar a un hito para permitir al jugador realizar la reconfiguración de su equipo de trabajo.

3.2.4.5 Iteración 5

El objetivo en la quinta iteración fue lograr un prototipo que implementara la secuencia de una partida completa que permitiera ejecutar acciones como: asignar el turno al jugador correspondiente, cobrar el AC semanal, eliminar al jugador si se ha quedado sin dinero, reconfigurar el equipo y tirar el dado con dos posibles escenarios (mover ficha o riesgo).

3.2.4.6 Iteración 6

En esta iteración se realizó la gestión del valor ganado (al llegar a un hito), su comprobación y la actualización de valores, todo ello a través de un formulario. También se introdujo otro formulario para calcular la nueva planificación (llamado formulario de plan de proyecto) hasta el siguiente hito.

3.2.4.7 Iteración 7

En la séptima iteración se trabajó exclusivamente en implementar la funcionalidad multijugador al juego. Se debía permitir la creación de una sala donde pudieran unirse los jugadores y comenzar la partida. Hay que mencionar que el tiempo asignado a esta iteración no fue acorde estrictamente a los plazos establecidos puesto que se tardó un poco más debido a la necesidad de analizar la nueva tecnología Photon y cambiar parte de la lógica ya implementada en iteraciones anteriores para permitir la gestión *online*. Además, se dedicó parte del tiempo a la creación del *lobby* y las salas dentro de la interfaz del juego.

3.2.4.8 Iteración 8

Iteración dedicada a comprobar la portabilidad del juego con versiones como la versión escritorio de Windows, la versión móvil y tablet de Android y la versión WebGL, y resolver posibles problemas de compatibilidad que el juego pudiera generar.

3.2.4.9 Iteración 9

En esta iteración se trabajó en aumentar la variabilidad en las partidas. Para ello se crearon varios tableros csv para permitir jugar en más de uno. Se implementó también la asignación de número de empleados iniciales (1, 2 o 3). Y se permitió que ambas características puedan ser modificadas por el *master client* (creador de la sala) antes de empezar la partida.

La variabilidad en los riesgos también fue un asunto que se tuvo en cuenta, aumentado los tipos de riesgos que pudieran tocar (esperar x turnos, mover la mitad o el doble de la productividad o sumar y restar dinero). Además, se añadió otra ventana modal emergente que aparece cuando toca un riesgo en la que se describen el nombre del riesgo y su efecto.

3.2.4.10 Iteración 10

El objetivo en la décima y última iteración fue cambiar los formularios para resaltar la importancia de los elementos que requieren de una entrada numérica por parte del usuario, la creación de una clasificación *online* por puntos accesible para todos los jugadores y aumentar la fluidez de las fichas para que el juego tenga un aspecto más realista.

4. Análisis de requisitos

En este apartado se detallan los requisitos funcionales y no funcionales del juego realizado.

4.1 Requisitos funcionales

Se detalla en la tabla 2 los requisitos funcionales que debe implementar el juego desarrollado.

#	DESCRIPCIÓN
RF01	El juego deberá ser jugado en modo multijugador.
RF02	El juego permitirá configurar parámetros iniciales como: el número de empleados iniciales o el tablero de entre los disponibles.
RF03	El juego se desarrollará por turnos.
RF04	El juego automatizará el cálculo de valores tales como el SPI, CPI o el coste total de una fase a partir de unos parámetros introducidos por el jugador.
RF05	El juego continuará pese a que un jugador abandone la partida.

RF06	El juego contará con un dado como método para que el jugador avance las casillas correspondientes.
RF07	El juego finalizará cuando un jugador gane, todos se queden sin dinero menos uno o todos, menos uno, abandonen la partida.
RF08	El tablero tendrá una casilla de salida y una de llegada o meta.
RF09	Al finalizar la partida, el sistema mostrará un sumario con la posición de cada jugador en la partida.
RF10	Cuando un jugador llegue a un hito, el sistema mostrará el reporte de actuación de la fase pasada, la ventana de asignación de empleados y el plan de proyecto para la siguiente fase.
RF11	El jugador deberá introducir, en el reporte de actuación de la fase, el valor ganado y el sistema comprobará que el valor es correcto. Después, el sistema mostrará los valores del CPI y el SPI.
RF12	El jugador, utilizando la ventana de asignación de empleados, deberá seleccionar los empleados que desea tener disponibles para la siguiente fase.
RF13	El jugador deberá introducir una nueva planificación para la siguiente fase al inicio y al llegar a un hito.
RF14	El sistema permitirá tener acceso en todo momento a la información relevante del proyecto como el plan de proyecto, el reporte de actuación, los empleados o el dinero disponible.
RF15	Si se lanza el dado y sale un número entre 1 y 4 el sistema avanzará la ficha del jugador normalmente, si sale un 5 o un 6 el sistema asignará un riesgo al jugador.
RF16	El sistema deberá almacenar el conjunto de riesgos y empleados disponibles.
RF17	Al final de cada turno el sistema descontará al jugador el salario correspondiente a la suma del salario semanal de todos sus empleados.
RF18	El sistema eliminará de la partida, automáticamente, al jugador que se quede sin dinero.
RF19	El sistema deberá, ante cualquier cambio en los atributos del jugador (número de empleados, presupuesto...), actualizar la interfaz.

Tabla 2 – Requisitos funcionales

4.2 Requisitos no funcionales

Se detallan en la tabla 3 los requisitos no funcionales del juego, es decir, los requisitos que atañen a las características del funcionamiento y no a la funcionalidad en sí.

#	TIPO	DESCRIPCIÓN	RELEVANCIA
RNF01	Portabilidad	El juego deberá poder ejecutarse en versión escritorio en Microsoft Windows y en móviles y tabletas Android.	Muy alta
RNF02	Usabilidad	La interfaz del juego deberá ser intuitiva.	Alta
RNF03	Disponibilidad	La información relevante para el juego deberá ser almacenada en ficheros XML.	Muy alta
RNF04	Compatibilidad	Los jugadores que se encuentren en la misma partida deberán tener la posibilidad de jugar desde distintos dispositivos (juego cruzado).	Alta
RNF05	Rendimiento	El juego deberá hacer un uso máximo de memoria de 2GB.	Alta
RNF06	Rendimiento	El juego deberá superar en todo momento los 24 fotogramas por segundo.	Alta

Tabla 3 – Requisitos no funcionales

5. Diseño e Implementación

En este apartado se comentará el diseño arquitectónico empleado y los aspectos relevantes sobre la implementación y el diseño de cada una de las partes del sistema.

5.1 Arquitectura del sistema

La arquitectura empleada como modelo fue la arquitectura en tres capas (Taylor, 2009). Se trata de la arquitectura más utilizada en la mayoría de los sistemas software. Esta arquitectura consiste en dividir el sistema en diferentes capas con el fin de facilitar la modificación de módulos, mejorar la flexibilidad y favorecer la reutilización de código.

Otras ventajas que presenta la arquitectura en tres capas con respecto a otros tipos de arquitecturas son: reducir la carga en el lado del cliente, permitir una migración fácil de una tecnología de persistencia a otra, facilidad para escalar el sistema o facilidad para realizar actualizaciones futuras.

En la figura 7 puede observarse un diagrama que muestra el funcionamiento de la arquitectura en tres capas.

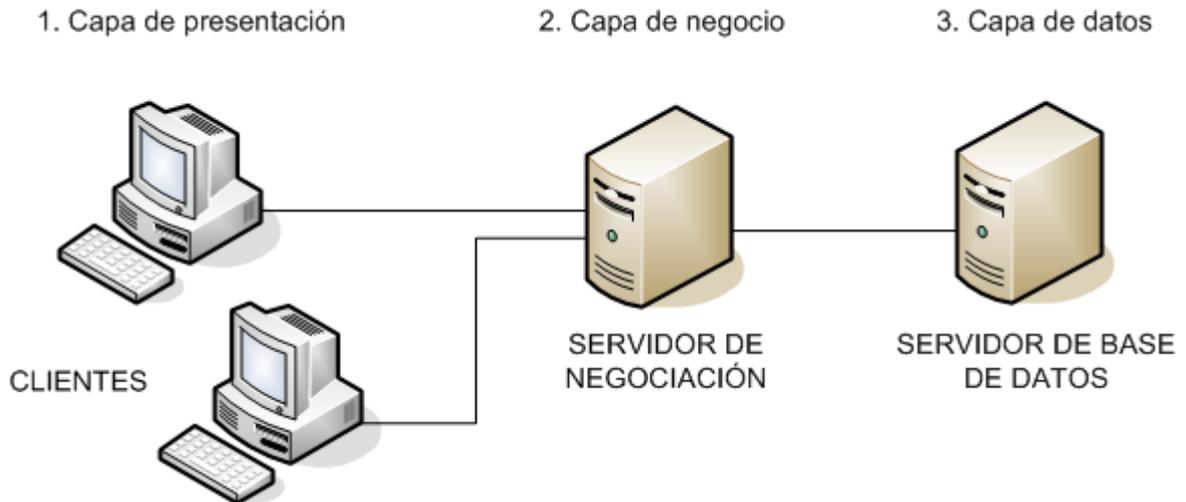


Figura 7 – Arquitectura en tres capas

Las capas en las que se divide el sistema son las siguientes:

- **Capa de presentación:** Capa encargada de llevar a cabo la interacción entre el usuario y el sistema. En esta capa se mostrarán y gestionarán las distintas pantallas del juego. En estas pantallas se podrá gestionar salas, configurar distintos aspectos, así como mostrar el tablero y los elementos gráficos. La función de esta capa, por tanto, es mostrar el sistema al usuario y presentar la información relevante. Comúnmente se conoce a esta capa de presentación como interfaz gráfica del programa. Esta capa debe mantener unos estándares que garanticen que la interfaz sea un elemento intuitivo y fácilmente usable por el usuario final. La capa de presentación se comunica directamente con la capa de negocio.
- **Capa de negocio:** Capa donde reside la lógica del sistema, es decir, las funciones que se ejecutan, el procesado de la información y la gestión de peticiones. En el desarrollo esta capa coincide con la implementación del código realizado. En ella se establecen, por tanto, todas las reglas que se deben cumplir. Esta capa se comunica con la capa de presentación para recibir solicitudes y con la capa de datos o persistencia para llevar a cabo la gestión de los datos. Esta capa de negocio se comunica con dos servicios externos: uno, que gestiona la clasificación (dreamlo) y otro que gestiona las partidas multijugador (Photon).
- **Capa de datos:** Capa encargada de la gestión de los datos del sistema. Puede estar formada por varios sistemas gestores de bases de datos u otro tipo de almacenamiento. Comunica con la capa de negocio y su función principal es almacenar, leer y manipular datos. En este proyecto la capa de datos está representada por los ficheros XML que almacenan los riesgos y los empleados, los ficheros csv que almacenan los distintos tableros y la clasificación almacenada en el servicio comentado.

5.2 Capa de presentación

A la hora definir la capa de presentación del juego se prestó atención a dos aspectos principalmente. Por una parte, a la interfaz del usuario, y por otra a la gestión y navegación por las diferentes escenas que implementan esta interfaz.

5.2.1 Interfaz de usuario

La interfaz de usuario del juego se puede dividir en dos. La interfaz jugable, es decir, la interfaz con la que interactúa el jugador durante la partida. Y la interfaz navegable, es decir, la interfaz por la que navega el jugador hasta que comienza una partida.

Aunque ambas partes formen un conjunto bien formado, el proceso seguido para la realización de los dos componentes fue distinto.

5.2.1.1 Interfaz del juego

La interfaz del juego tuvo como punto de partida e inspiración los elementos físicos del propio tablero. Con esos elementos se intentó hacer una interfaz acorde que se pareciera lo máximo posible al diseño original con algunas peculiaridades. La figura 8 muestra el tablero físico y la figura 9 muestra la interfaz resultante en la versión digital.

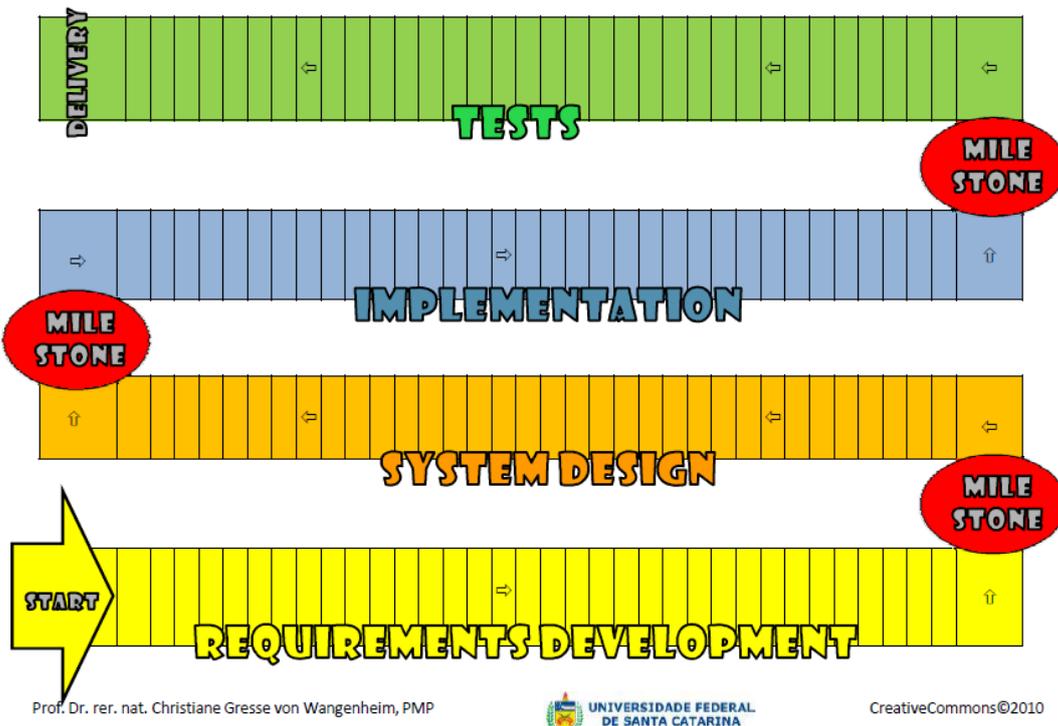


Figura 8 – Tablero físico



Figura 9 – Interfaz usuario del juego digital

La interfaz del juego en formato digital añade nuevos elementos dado que el tablero no es el único activo indispensable para el desarrollo de la partida. En la imagen anterior se puede observar, en la parte superior, información relevante para el jugador como quién tiene el turno, la fase actual del desarrollo del proyecto en la que el jugador está o el número de casillas por fase, dato importante para hacer el cálculo del plan de proyecto para la fase.

En la parte inferior se encuentra, a la izquierda, el presupuesto del jugador, la productividad de su equipo y el salario. En la parte central una lista con los empleados disponibles actualmente; con un formato donde pone el nombre, la productividad y el salario del empleado correspondiente. A la derecha se encuentran los elementos interactivos. Estos elementos interactivos son un dado para realizar el avance de las casillas cuando te toque el turno y dos botones para acceder de manera sencilla a los informes del reporte de actuación y el plan del proyecto.

Estos informes le sirven al usuario para tener disponible en todo momento información relevante sobre su partida. En el informe que se puede observar en la figura 10 se aprecia el reporte de actuación de la última fase recorrida (análisis de requisitos). En este informe, por ejemplo, se muestran datos como el progreso planificado y real de la fase, el coste planificado y real y los valores del EV, CPI y SPI. Por su parte, el otro informe disponible, el informe del plan de proyecto, muestra los costes planificados para la siguiente fase.



Figura 10 – Reporte de actuación

A parte de estos elementos visibles e identificables, la interfaz también dispone de otros elementos inicialmente ocultos que aparecen, en modo de ventana emergente, cuando un jugador llega a un hito. Se les ha decidido dar un nombre genérico de formularios y su principal función consiste en gestionar los recursos y realizar una reconfiguración del proyecto. En la siguiente figura (figura 11) se observa el formulario de plan de proyecto donde el usuario deberá introducir el número de semanas que espera que la fase dure y una cantidad de dinero que asigna para la gestión de la reserva. Los datos del coste de recursos humanos y el coste total se calculan automáticamente, el primero multiplicando en número de semanas por el coste del equipo por semana y el segundo sumando la reserva a esta cantidad.

PLAN DE PROYECTO PARA LA SIGUIENTE FASE	
Nº semanas	4
Gestion de reserva	500
Coste de los recursos humanos	1600
Coste total de la fase	2100
ACTUALIZAR	CONTINUAR

Figura 11 – Formulario de plan de proyecto

Cuando se llega a un hito el jugador deberá realizar también una reconfiguración total de su equipo de trabajo (si así lo desea). La ventana de reconfiguración del equipo se muestra en la figura 12 y en ella aparece una lista de todos los empleados disponibles con una imagen (se ha decidido asignarle una temática relacionada con monstruos), así como su nombre, productividad y salario. El jugador deberá seleccionar los empleados deseados y continuar.

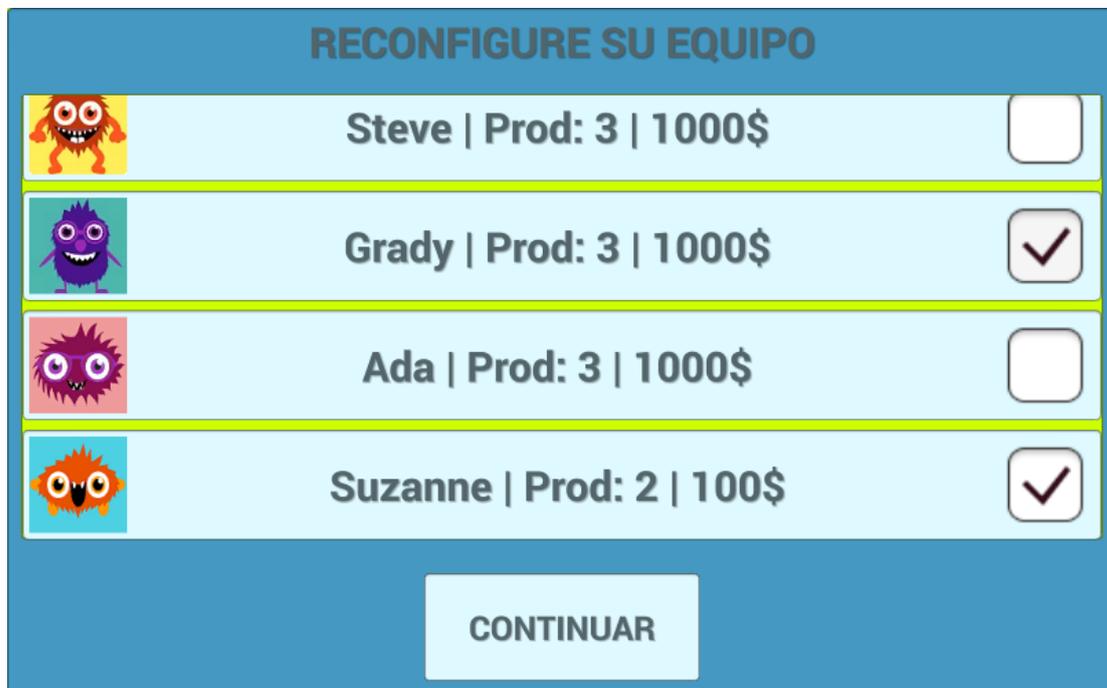


Figura 12 – Ventana modal de reconfiguración

Es importante hablar en este apartado sobre los colores elegidos para la interfaz. Para ello se realizó una selección de posibles paletas de colores utilizando Adobe Color CC. Después de descartar todas las anteriores se llegó a la paleta que le da color al juego. Esta paleta se puede observar en la siguiente figura (figura 13).

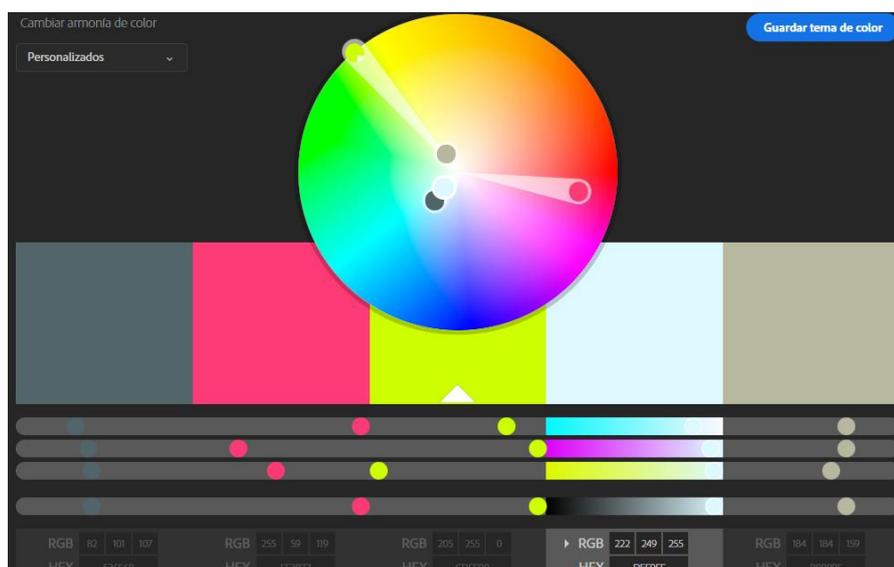


Figura 13 – Paleta colores de la interfaz

La paleta de colores la comprenden cinco colores. Dos colores complementarios y llamativos como el verde y el rosa que resaltarán los elementos más importantes del juego. Dos colores monocromáticos de la misma gama como el blanco y el gris. Y un color neutro como el beige.

De los cinco colores que tiene la paleta se pretendió que, a priori, todos tuvieran su propio cometido, aunque hay algunas salvedades. Se utilizó el gris para los textos de la interfaz, el rosa para resaltar los elementos relevantes (como los hitos o el turno del jugador), el verde para elementos relacionados con la interfaz, el blanco ligeramente azulado para los botones de los formularios, así como para las entradas y salidas de texto y por último el beige como color de fondo del juego.

Estos iban a ser los colores principales inicialmente, pero es importante comentar que se realizaron algunos cambios de última hora. Se hablará más adelante en esta memoria (en el apartado de pruebas de aceptación) sobre ello, pero al acudir a una empresa externa al proyecto para que dieran su valoración sobre el juego propusieron algunos cambios, los cuales algunos fueron adoptados. Uno de ellos fue usar un nuevo color, un azul “no demasiado agresivo”, como color para las diferentes ventanas modales de reconfiguración. Inicialmente la ventana modal que encapsula al formulario tenía color beige pero esto no era correcto dado que producía una ligera confusión con el fondo.

5.2.1.2 Interfaz de menús

Para la creación de la interfaz navegable se crearon unos sencillos *mockups* como base inicial para lo que se pretendía hacer. En las siguientes figuras se pueden dichos *mockups*. Comentar que el resultado final de la aplicación de los *mockups* al juego fue fiel a lo planificado, es por ello que no se incluyen las imágenes del resultado, al ser casi idénticas (no quisiera saturar con imágenes este documento).

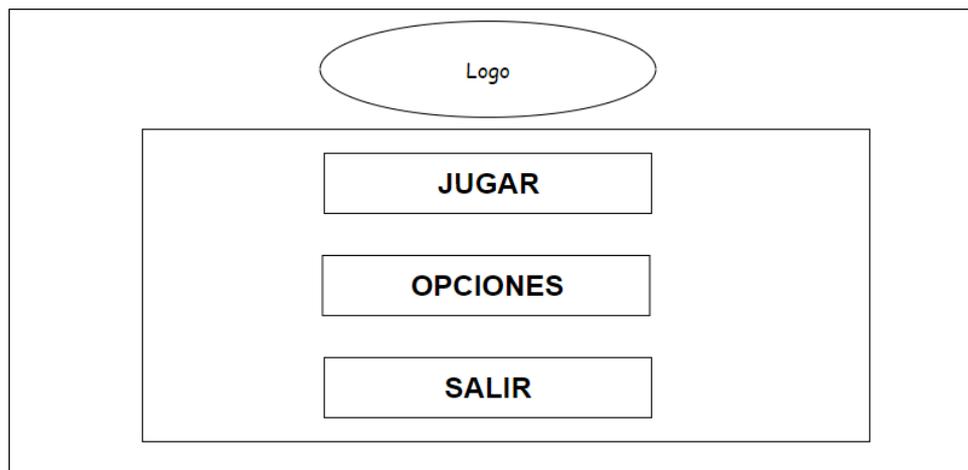


Figura 14 – Mockup del menú principal



Figura 15 – Mockup lobby

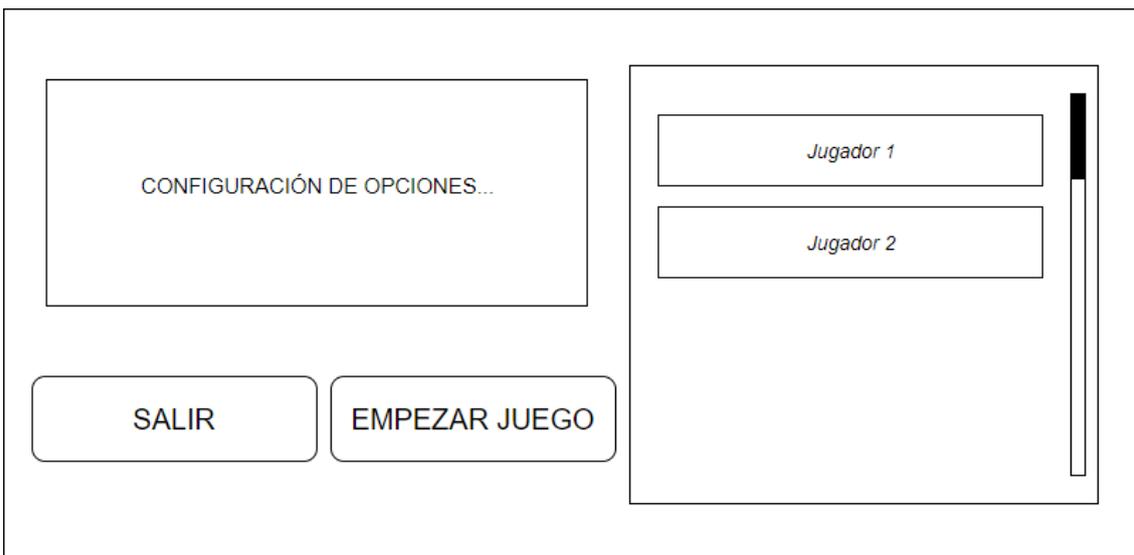


Figura 16 – Mockup sala

5.2.2 Escenas

Las escenas de Unity son elementos que contienen los entornos y menús del juego. Cada escena se considera como un nivel único. El objetivo del programador es colocar los distintos entornos, obstáculos y decoraciones en las escenas. La idea principal del uso de escenas es construir el juego “a pedazos” para facilitar la simplicidad de los distintos módulos.

El sistema tiene cuatro escenas y el contenido de cada una es el siguiente:

- **MENU_SCENE:** Contiene el menú principal del juego. A través de esta escena se puede acceder a la escena de lobby introduciendo un nombre de entre 1 y 16 caracteres y pulsando el botón jugar. También tiene otros dos botones: el botón de puntuaciones para observar la clasificación global del juego y el botón salir para cerrar el juego.
- **LOBBY_SCENE:** Gestiona la conexión al servidor de Photon. Permite crear salas, unirse a los jugadores a esas salas, gestionar las opciones y comenzar la partida cuando hay al menos dos jugadores en la sala.
- **GAME_SCENE:** Esta escena es en la que se desarrolla el juego. Almacena toda la presentación de éste. El contenido de esta escena se ha descrito en el apartado relativo a la interfaz del juego.
- **STATS_SCENE:** Es la última escena del juego y su cometido es mostrar a los jugadores información relativa a la posición en la que han quedado después de celebrarse la partida (un sumario). Posee, en la parte inferior, un botón para abandonar el juego.

5.3 Capa de negocio

5.3.1 Estructura del proyecto

El código del proyecto se subdivide en varios módulos. Por una parte, se encuentra la lógica del juego, por otra la lógica tanto del menú principal como del sumario que aparece al terminar la partida, y por último todo el código relativo a la gestión del modo multijugador. En la siguiente tabla (tabla 4) se observa la jerarquía de *scripts* del sistema.



<ul style="list-style-type: none"> ● Multiplayer <ul style="list-style-type: none"> □ <i>MainCanvasManager.cs</i> ● CreateRoom <ul style="list-style-type: none"> □ <i>CreateRoom.cs</i> ● Generic <ul style="list-style-type: none"> □ <i>DDOL.cs</i> ● LobbyHandler <ul style="list-style-type: none"> □ <i>LobbyCanvas.cs</i> □ <i>RoomLayoutGroup.cs</i> □ <i>RoomListing.cs</i> ● Networks <ul style="list-style-type: none"> □ <i>LobbyNetwork.cs</i> □ <i>PlayerNetwork.cs</i> ● RoomHandler <ul style="list-style-type: none"> □ <i>PlayerLayoutGroup.cs</i> □ <i>PlayerListing.cs</i> □ <i>RoomCanvas.cs</i>
--

Tabla 4 – Jerarquía de scripts desarrollados

Es importante mencionar, aunque sea brevemente, la función que realizan, a grandes rasgos, las clases almacenadas en cada uno de los tres módulos:

- **Game:** Esta carpeta contiene toda la lógica del juego. En ella se encuentra el *script* correspondiente al *game manager*. Este *script* gestiona el tablero, gestiona el turno y la lanzada del dado, muestra y actualiza los formularios, y crea los distintos objetos del juego en tiempo de ejecución, como puede ser la información contenida en los informes disponibles del jugador. También se encuentran, en esta carpeta, clases básicas que gestionan los distintos elementos del juego como los riesgos y los empleados, así como sus contenedores. Hay disponible una clase Dado cuya tarea es obtener una puntuación al lanzar el dado y el renderizado de este elemento (con la puntuación que toque) después de lanzarlo. En cuanto a la clase Jugador, en ella se almacenan los datos de cada jugador y la gestión del movimiento de la ficha asignada. Por último, se encuentra la clase Tablero cuyo objetivo es crear las distintas casillas en la escena una vez leído el archivo csv que almacena los datos del tablero.
- **Menu:** Esta carpeta contiene dos clases. Por un lado, la clase MainMenu que gestiona los botones del menú principal, se encarga de cargar la escena del lobby, cerrar el juego si así se desea y mostrar las puntuaciones globales de los jugadores que han jugado al juego. La segunda clase es el Sumario, cuya funcionalidad se basa en la creación de las estadísticas al finalizar la partida, actualizar la tabla de puntuaciones de acuerdo con la posición obtenida y finalizar la ejecución del juego cuando el jugador así lo desee.
- **Multiplayer:** Contiene toda la lógica de la gestión *online*. Las clases anotadas como “Canvas”, “LayoutGroup” o “Listing” tienen como objetivo dotar a la interfaz de distintas funciones y crear los distintos objetos necesarios. Estos objetos son los botones con el nombre de la sala y el nombre de los jugadores dentro de la sala. Cuando otro jugador crea una sala, se crea un botón para todos los jugadores y al pulsarlo te introduce en dicha sala. De esta carpeta hay que prestar especial atención a tres elementos que son: la clase DDOL, la clase LobbyNetwork y la clase PlayerNetwork. Aunque las funcionalidades de estos elementos ya se detallarán, al gozar de gran importancia, en el apartado correspondiente al servidor Photon es importante mencionar que son las clases que hacen posible, a grandes rasgos, la gestión multijugador.

5.3.2 Game Manager

El Game Manager es la clase más importante del juego al ser aquella que controla la mayor parte de la lógica. Es por ello por lo que merece especial atención y una explicación detallada de aquellos aspectos más importantes que en él se contienen.

Se ha de mencionar en este apartado el uso del patrón de diseño Singleton (Gamma, 2002). Este patrón permite restringir la creación de objetos pertenecientes a una clase. Su intención consiste en conseguir garantizar que solo haya una instancia y proporcionar acceso global a ella.

La clase GameManager no es el único ejemplo que se podría poner para el uso del patrón Singleton en este proyecto. En el siguiente fragmento de código (código 1) se muestra el código relativo a la implementación de dicho patrón en la clase mencionada.

```
60     public static GameManager _instance = null;
61
62     void Start()
63     {
64         // chequeamos si existe una instancia de la clase
65         if (_instance == null) {
66             // si no existe esta sera la empleada a partir de ahora
67             _instance = this;
68         }
69         // si existe y es diferente de esta
70         else if (_instance != this) {
71             // destruimos esta instancia
72             Destroy (gameObject);
73         }
74     }
75
```

Código 1 – Patrón Singleton (GameManager.cs)

Es importante desglosar en este apartado, a parte del uso del Singleton, información relativa sobre los métodos más importantes implementados en la clase GameManager. Algunos de estos métodos permiten gestionar las siguientes funcionalidades:

- **Cargar datos:** El objetivo es cargar el tablero seleccionado, los jugadores (sus fichas) y cargar los datos de los ficheros XML que almacenan los recursos necesarios y de los cuales se hablará un poco más adelante.
- **Gestión de la interfaz:** Se trata de un conjunto de dos métodos, uno para activar la interfaz cuando se cargan los datos y otro para actualizar la interfaz (reducir presupuesto, modificar los empleados disponibles...) antes de pasarle el turno al siguiente jugador.
- **Gestión de los formularios:** Se trata de un conjunto de métodos que permite la gestión de los formularios realizando las tareas de cargar los formularios, actualizar los

valores numéricos a partir de unas entradas introducidas y permitir continuar si todos los datos son correctos.

- **Botones:** Son un conjunto de métodos que definen la lógica desencadenada al pulsar los botones de la interfaz. Estos métodos permiten mostrar u ocultar al usuario los informes de plan de proyecto y reporte de actuación.

- **Gestión del dado:** Se trata de un método que permite la lanzada del dado. Este método se puede observar en el siguiente código (código 2). En este método, primero se comprueba si el jugador que pulsa el dado es el jugador que tiene el turno actualmente. Si lo tiene se lanza el dado. En caso de que la puntuación sea un 5 o un 6 se ejecuta un riesgo para el jugador y se avanza. En caso de que sea 4 o menos, no se ejecuta ningún riesgo. Después de llamar al método que permite el movimiento (almacenado en la clase Jugador) se comprueba el valor devuelto. Si el valor devuelto es un -1 se elimina al jugador al haberse quedado sin dinero y se desactiva la interfaz (en caso de que solo quede un jugador en la partida se lanza el sumario dado que ese será el ganador). Si el valor devuelto es un 0 se cambia el turno normalmente, si es un 1 se lanza el sumario de la partida porque el jugador ha llegado al final y si es un 2 se muestran los formularios y la ventana de reconfiguración del equipo.

```
124 public void TirarDado () {
125     int ganador = 0;
126
127     if (PlayerNetwork.instance.jugadores [PlayerNetwork.instance.turno] == jugadorGestionado.GetComponent<PhotonView> ().owner.NickName && permitirTurno) {
128         Dado dScript = GameObject.FindWithTag ("Dado").GetComponent<Dado> ();
129         int puntuacion = dScript.LanzarDado ();
130         Jugador jScript = jugadorGestionado.GetComponent<Jugador> ();
131
132         if (puntuacion >= 5) {
133             Riesgo riesgo = riesgos.riesgos [UnityEngine.Random.Range (0, riesgos.riesgos.Count)];
134             riesgo.EjecutaEfecto(jScript);
135             ganador = jugadorGestionado.GetComponent<Jugador> ().Mover (puntuacion, tablero);
136             if (ganador != 1) { // Si no se ha ganado se muestra el riesgo
137                 cartaRiesgo.SetActive (true);
138                 textRiesgo.GetComponent<Text> ().text = riesgo.nombreRiesgo + "\n" + riesgo.efecto;
139                 permitirTurno = false;
140             }
141         } else {
142             ganador = jugadorGestionado.GetComponent<Jugador> ().Mover (puntuacion, tablero);
143         }
144
145         switch (ganador) {
146             case -1: // JUGADOR ELIMINADO
147                 if (PlayerNetwork.instance.PlayerEliminated ()) {
148                     listaEmpleados.SetActive (false);
149                     datos.SetActive (false);
150                     elementosInteractivos.SetActive (false);
151                     CambiarTurnoJugador ();
152                 } else {
153                     LanzarSumarioPartida ();
154                 }
155                 break;
156             case 0: // TURNO NORMAL
157                 ActualizarInterfaz (false);
158                 CambiarTurnoJugador ();
159                 break;
160             case 1: // GANADOR
161                 ActualizarInterfaz (false);
162                 //PlayerNetwork.instance.ChangeTurn(PhotonNetwork.playerList.Length); // No habra ningun jugador con ID igual al numero de jugadores en la sala,
163                 LanzarSumarioPartida();
164                 break;
165         }
166     }
167 }
```

Código 2 – Método TirarDado (GameManager.cs)

- **Cambiar turno:** Se trata de un método que permite el cambio de turno de jugador. Este método comprueba que el siguiente jugador no tenga que parar ningún turno y que permanezca en el juego. En caso de que no se den las dos condiciones se pasa al siguiente hasta que haya uno que si las cumpla.

- **Lanzar sumario:** Se trata de un método que ordena en una lista a los jugadores en función de las casillas recorridas para posteriormente cargar la escena del sumario de la partida. El ganador aparecerá en primer lugar al haber recorrido más casillas. En

En el código que se presenta a continuación (código 4) se muestra la clase LobbyNetwork. Los tres métodos que en ella se implementan tienen como objetivo conectarse al servidor, unirse al lobby con el nombre de usuario escogido en el menú principal y modificar la interfaz respectivamente.

```
3 public class LobbyNetwork : MonoBehaviour {
4
5     private void Start () {
6         Debug.Log("Start Lobby Network");
7         PhotonNetwork.ConnectUsingSettings ("1.0");
8     }
9
10    private void OnConnectedToMaster() {
11        Debug.Log ("Connecting to master");
12        PhotonNetwork.automaticallySyncScene = true;
13        PhotonNetwork.playerName = PlayerNetwork.instance.pName;
14        PhotonNetwork.JoinLobby (TypedLobby.Default);
15    }
16
17    private void OnJoinedLobby(){
18        Debug.Log ("Joined lobby");
19        if (!PhotonNetwork.inRoom) {
20            MainCanvasManager.instance.LobbyCanvas.transform.SetAsLastSibling ();
21        }
22    }
23 }
24
```

Código 4 – LobbyNetwork.cs

Para la gestión del modo multijugador se creó una nueva clase llamada PlayerNetwork que almacena las variables globales del juego como la lista de los jugadores dentro de la partida, quién tiene el turno y opciones como el tablero o el número de empleados iniciales. Esta clase contiene también unos métodos anotados como “PunRPC” que permiten que todo el conjunto de los jugadores de la partida los ejecute cuándo se produzca un evento que los desencadene. Estos métodos se usan para varias tareas. Las más señaladas son cambiar el turno cuando el jugador que tiene actualmente el turno lo ha acabado o para gestionar el turno si el jugador actual que lo tiene abandona la partida. A continuación, se muestra el código (código 5) del cambio de turno para todos los jugadores.

```
106    public void ChangeTurn(int turno){
107        PhotonView.RPC ("RPC_ChangeTurn", PhotonTargets.All, turno);
108    }
109
110    [PunRPC]
111    private void RPC_ChangeTurn(int turno){
112        this.turno = turno;
113        GameManager._instance.CambiarTextoTurno ();
114    }
115
```

Código 5 – Fragmento de PlayerNetwork.cs

Cuando el jugador que tiene el turno lo finaliza realizará una llamada al método de cambio de turno con el siguiente jugador como parámetro. Este método llama a un

método "PunRPC" que será ejecutado por todos los jugadores de la partida (*PhotonTargets.All*) y que cambiará el turno al siguiente jugador y actualizará el mensaje que informa al usuario de quién tiene el turno actualmente.

Es importante mencionar que la clase *PlayerNetwork*, como es obvio, debe persistir entre escenas, es por ello necesario definirla de manera que no se borre cuando se carga la escena del juego debido a que el borrado es el comportamiento por defecto de todos los objetos del juego en el cambio de escenas.

Para hacer persistente la instancia de esta clase se creó dentro de la escena del lobby un objeto que tiene como componente el *script* llamado DDOL (acrónimo de *DontDestroyOnLoad*) que evita que se borre al cambiar de escena. Dentro de este objeto se encuentra adjunto (como objeto hijo) otro objeto que tiene como componente el *script* *PlayerNetwork*. De esta manera sencilla se consigue que no se elimine la clase que gestiona el modo *online*. A continuación, en el fragmento de código 6 se muestra la sencilla implementación de la clase DDOL.

```
3 public class DDOL : MonoBehaviour {
4
5     private void Awake () {
6         DontDestroyOnLoad (this); // Se usa para que no se destruya el objeto que gestiona el multiplayer
7         // entre escenas
8     }
9 }
10
```

Código 6 – DDOL.cs

5.3.5 Sumario

Se crea este apartado para hablar sobre el sumario y mostrar su apariencia física dado que es un elemento con importancia dentro del propio juego. En la figura 18 se puede apreciar la escena del sumario. En ella se puede ver arriba a la derecha el logo del juego que aparece en todas las escenas disponibles. En la parte central se muestra una pequeña infografía de las estadísticas de los jugadores mostrando su posición después de haber finalizado la partida, una imagen representativa (varias medallas; a saber, de oro, plata, bronce y de color negro para el último) y su nombre. Y por último aparece un botón que permite cerrar el juego.



Figura 18 – Sumario del juego

Se debe mencionar en este apartado también el porqué de la necesidad de incluir un *battletag* (la almohadilla y el número) que es principalmente para evitar problemas que se pudieran producir, relativo a la gestión de los turnos, en el juego al incluir dos jugadores con el mismo nombre. De esta manera si hay dos jugadores con el mismo nombre, aun así, será distinto porque el sistema añade automáticamente un número aleatorio detrás de dicho nombre.

Dentro del código de la clase Sumario quería poner especial atención al código que gestiona la actualización de la clasificación online. Dicho código puede ser observado en el siguiente fragmento (código 7).

```

38 public void AnyadirNuevaPuntuacion(string username, int puntuacion){
39     StartCoroutine (ModificarPuntuacion (username, puntuacion));
40 }
41
42 IEnumerator ModificarPuntuacion(string username, int puntuacion){
43     int valorAnterior = 0;
44
45     if (MainMenu.instancia.puntuaciones.Count > 0) {
46         foreach (PuntuacionJugador pj in MainMenu.instancia.puntuaciones) {
47             if (pj.nombre.Equals (username)) {
48                 valorAnterior = pj.puntuacion;
49                 break;
50             }
51         }
52     }
53
54     WWW www = new WWW (webURL + privateCode + "/add/" + WWW.EscapeURL(username) + "/" + (puntuacion + valorAnterior));
55     yield return www; // para esperar a que se actualice al no ser instantaneo.
56
57 }
58

```

Código 7 – Actualización de puntuaciones

El método que añade una nueva puntuación llama al método de modificar puntuación como una co-rutina pasándole como datos un nombre de usuario y una puntuación. Esta puntuación será de 15 puntos para el primero, 10 para el segundo, 5 para el tercero y ninguno para el cuarto (independientemente del número de jugadores que haya en la partida).

El método de modificación busca en las puntuaciones almacenadas en la instancia de la clase MainMenu (al abrir el juego se cargan estas puntuaciones del servicio dreamlo) y comprueba si hay actualmente alguna puntuación del usuario pasado como parámetro. Si lo hay se modifica el atributo del valor anterior a esa puntuación. Esto se tiene que realizar de esta manera porque, a la hora de realizar la llamada a la API REST que gestiona las puntuaciones, si ya hay un jugador con ese nombre se sobrescribe la puntuación. Procediendo así se consigue que se sobrescriba la puntuación que tenía antes con la suma de esta más la nueva puntuación correspondiente a la partida recién jugada.

Es importante comentar también que no todos los jugadores modifican todas las puntuaciones, sino que es el propio jugador el único que modifica su puntuación y no las de los demás. En cuanto a lo relativo a la asignación de la puntuación con valores 15, 10, 5 y 0 se decidió que fuera así debido a que no hay ningún registro de usuarios del sistema, por tanto, si un jugador decide jugar una partida con un nombre que ya había sido utilizado por otro usuario, esto no perjudicará a la puntuación de ese usuario puesto que lo peor que puede pasar es que quede cuarto en una partida y, al no sumar ningún punto, la puntuación se mantenga como estaba hasta ahora.

5.4 Capa de datos

Los datos que se usan en el juego están almacenados en tres sitios diferentes. Por un lado, están los datos de la interfaz definida en Unity. Estos datos se almacenan por el motor en ficheros de configuración que se gestionan automáticamente a medida que se van creando. Por otro lado, se encuentran los recursos que se usan dentro del propio juego (empleados y riesgos) que están contenidos en ficheros XML, los tableros contenidos en ficheros csv y por último se encuentra la clasificación *online* que se gestiona a través de la página dreamlo y el servicio que ésta proporciona.

Los datos que almacena Unity no requieren de explicación, sin embargo, los otros tres tipos de datos almacenados sí. Es por ello por lo que se especifican en los siguientes puntos de manera más detallada.

5.4.1 Ficheros XML

Para almacenar los datos del juego se decidió emplear ficheros XML dado que se trata de un método sencillo y una tecnología de la cual ya tenía conocimientos previamente (Rusty, 2004). Los datos que se necesitaban almacenar eran los empleados y los riesgos.

Para los empleados se creó una clase Empleado y otra ContenedorEmpleado que contiene una lista de empleados y un método para cargar los empleados del XML. En el código 8 se puede observar esta última clase mencionada.

```

7 [XmlRoot("ColeccionEmpleado")]
8 public class ContenedorEmpleado {
9
10     [XmlArray("Empleados")]
11     [XmlArrayItem("Empleado")]
12     public List<Empleado> empleados = new List<Empleado>();
13
14     public static string xml_path = "xmls/empleados";
15
16     public static ContenedorEmpleado CargarEmpleados(){
17         TextAsset _xml = Resources.Load<TextAsset> (xml_path);
18
19         XmlSerializer serializer = new XmlSerializer (typeof(ContenedorEmpleado));
20
21         StringReader reader = new StringReader (_xml.text);
22
23         ContenedorEmpleado _contenedor = serializer.Deserialize (reader) as ContenedorEmpleado;
24
25         reader.Close();
26
27         return _contenedor;
28     }
29 }

```

Código 8 – ContenedorEmpleado.cs

En cuanto al propio empleado, éste debe tener un nombre, un salario y una productividad. En el código 9 se puede apreciar la estructura del fichero de datos XML para los empleados. En ella aparecen dos empleados, aunque en el juego oficial haya doce.

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <ColeccionEmpleado>
3     <Empleados>
4         <Empleado Nombre="Bill">
5             <Productividad>3</Productividad>
6             <Salario>1000</Salario>
7         </Empleado>
8         <Empleado Nombre="Steve">
9             <Productividad>3</Productividad>
10            <Salario>1000</Salario>
11        </Empleado>
12    </Empleados>
13 </ColeccionEmpleado>

```

Código 9 – Estructura del XML de empleados

En cuanto a los riesgos se realizó el mismo almacenado con una peculiaridad. Al tratarse de diferentes riesgos a implementar y con diferentes efectos en el jugador se implementó el código de negocio como una herencia de la clase Riesgo, de tal manera que ésta definiera unos atributos comunes a todos los riesgos como el nombre y el efecto e implementara un método abstracto para ejecutar el efecto correspondiente en el jugador afectado. El diagrama UML de la herencia desarrollada se puede observar en la figura 19.

5.4.3 Clasificación

Para realizar la gestión de la clasificación y las puntuaciones se realizó un estudio inicialmente de las posibilidades que había para implementar esa funcionalidad. Se valoró la posibilidad de usar una base de datos llamada Firebase gestionada por Google incluyendo en ella una entidad que almacenar el nombre del usuario y la puntuación. También se estudió otro servicio proporcionado por Google llamado Google Play Games Services que permite, entre otras cosas, la gestión de una clasificación y la creación de logros para tus juegos. Ambas posibilidades se desecharon, la primera por lo tedioso que resultaba la tecnología para realizar una tarea tan simple y la segunda por el coste monetario que tenía.

A parte de las desventajas que tienen las tecnologías revisadas se encontró una página web que permite la creación de una clasificación en la red. Una especie de API REST que almacena la tabla de puntuaciones en un *host* llamado dreamlo y al que se puede acceder a través de enlaces web.

Here is your **private** url. Copy and paste this somewhere.
Do not tell anyone about this link.

The screenshot shows the Dreamlo website interface. At the top, there's a URL input field with a private URL. Below it, there are instructions and a warning: "You copy and pasted that somewhere and are never going to tell anyone right?" and "You can not have an asterisk * character in your URL, scores, usernames, etc." The main content is divided into two columns. The left column, titled "Your Leaderboard", shows a table of the highest 1000 scores. The right column, titled "Adding and deleting scores", shows various API endpoints for adding, deleting, and clearing scores, as well as saving a trip. At the bottom, there's a section for "Codes for Unity Example" and "Getting your scores" with input fields for different data formats like XML, JSON, and pipe delimited.

Name	Score	
mario	30	delete
player2	30	delete
player1	30	delete
carlos	15	delete
mario2	15	delete
aaaaaaa	15	delete
Ludi	15	delete
JUGADOR1	15	delete
David+el+Nohomo	15	delete
pebe	15	delete
LuDisan	15	delete
osle	15	delete
Mario2	15	delete
Mario	10	delete
eeey	10	delete

Figura 20 – Clasificación en dreamlo

En la figura anterior (figura 20) se muestra la página de la clasificación. En ella se pueden observar, en un recuadro verde, las clasificaciones actuales que tiene

disponibles el sistema. Este servicio permite un máximo de 1000 puntuaciones ampliable si se contacta con el gestor del servicio. Para la problemática que me ocupa se supusieron suficientes este número de entradas. A la derecha se encuentran los diferentes enlaces que se deben ejecutar para realizar acciones sobre los datos de la tabla. En cuanto a estos enlaces se usan principalmente dos: el primer enlace que gestiona la creación/actualización de una puntuación y el último que devuelve todos los valores de la tabla.

Por razones de seguridad, los enlaces de modificación no son visibles, la clave privada de la clasificación tampoco ni, por supuesto, el enlace a la misma (que aparece arriba).

6. Evaluación y Pruebas

Este apartado tiene como objetivo comentar las pruebas realizadas en el juego desarrollado. Las pruebas son un elemento fundamental en un desarrollo software y deben realizarse siempre para evitar fallos no deseados.

Existen los siguientes tipos de pruebas:

- **Pruebas unitarias:** Estas pruebas tienen como objetivo verificar la funcionalidad y estructura de cada componente individual de código realizado. Son pruebas a nivel de métodos codificados.
- **Pruebas de integración:** Su objetivo es comprobar el correcto ensamblaje entre los distintos componentes del código. Cubren la funcionalidad establecida y se ajustan, en mayor medida, a los requisitos funcionales.
- **Pruebas de sistema:** Están destinadas a probar el sistema como un todo y se centran en los requisitos no funcionales como por ejemplo la gestión de recursos, compatibilidad, usabilidad. Suelen estar enfocadas a la gestión de los recursos como el procesador o la memoria.
- **Pruebas de aceptación:** Se usan para validar que el sistema cumple con el funcionamiento esperado y permitir al usuario del sistema que determine su aceptación sobre la funcionalidad y el rendimiento.

6.1 Pruebas unitarias y de integración

Se ha decidido crear un sólo epígrafe para las pruebas tanto unitarias como de integración dado que, por la naturaleza del propio programa, Unity, y de los juegos como software desarrollado es casi imposible encontrar elementos aislados de código que funcionen sin depender de otros.

Es importante mencionar que en este apartado aparte de las pruebas realizadas también se han considerado como pruebas los logs generados durante el desarrollo del proyecto, así como las ejecuciones en el editor como parte de la integración porque al tratarse de un juego, los elementos debían coordinarse a la perfección para

no obtener bugs indeseados. De esta manera se subsanaron errores que aparecían al producirse excepciones en el código.

Para llevar a cabo estas pruebas se ha hecho uso de una funcionalidad implementada por Unity llamada Test Runner que permite su configuración y ejecución. El Unity Test Runner hace uso de la librería NUnit que es una librería de pruebas de software libre para lenguajes .Net.

El Test Runner permite ejecutar los test tanto en modo editor (*edit mode*) como en modo de juego (*play mode*). Por la dificultad que tiene programar las pruebas en modo de juego se optó por no hacer uso de esta funcionalidad sustituyéndola por, como he mencionado anteriormente, ejecuciones del propio juego y comprobación de que todo funcione como debe.

En cuanto a las pruebas de modo editor, si se ha hecho uso de ellas. Estas pruebas consistían en la creación de una nueva clase que almacena las pruebas como propias funciones al igual que ocurre con otras librerías de pruebas con JUnit en Java. A continuación, se muestra el código (código 10) de una de las pruebas realizadas.

```
1 using UnityEngine;
2 using UnityEditor;
3 using UnityEngine.TestTools;
4 using NUnit.Framework;
5 using System.Collections;
6
7 public class MainMenuTest {
8
9     [UnityTest]
10    public IEnumerator TestLobbyJugar() {
11        GameObject objeto = new GameObject();
12        objeto.AddComponent<MainMenu> ();
13
14        Assert.IsNull (objeto.GetComponent<MainMenu>().nombre);
15
16        objeto.GetComponent<MainMenu> ().LobbyJugar ();
17
18        Assert.IsNotNull (objeto.GetComponent<MainMenu> ().nombre);
19        yield return null;
20    }
21
```

Código 10 – Prueba Test Runner

Esta prueba, propuesta como ejemplo, consistía en comprobar que en la clase MainMenu se le asignara un nombre al usuario al ejecutar el método *LobbyJugar()*. Este método será ejecutado cuando se pulse el botón jugar del menú principal. Lo que se buscaba al hacer esta prueba era comprobar que el nombre del jugador, almacenado en la variable nombre de la clase mencionada, no fuera nulo una vez desencadenado el evento.

Como se puede observar, el Test Runner usa, como elementos para comprobar el correcto funcionamiento de los métodos, aserciones que comprueban numerosas condiciones como: que un número sea mayor, menor o igual que otro, o el contenido de una cadena de texto, entre otras. En el método comprobado se puede ver como se comprueba, primero que el nombre es nulo y después que, tras la ejecución del método, el nombre no sea nulo.

6.2 Pruebas de sistema

Se dividieron las pruebas de sistema en tres bloques diferenciados atendiendo a los requisitos no funcionalidades establecidos previamente en este documento.

6.2.1 Pruebas de portabilidad

En cuanto a la portabilidad se había pedido que el juego pudiera ser ejecutado de manera multiplataforma. Se seleccionaron entre las plataformas disponibles las más usadas en el mundo de la informática hoy en día. Estas plataformas son ordenador y los dispositivos móviles como teléfonos o tabletas.

De entre esta gran variedad de plataformas se decidió que, en cuanto al ordenador, el juego pudiera ser ejecutado en entornos Windows y que en móviles pudiera ser ejecutado en entornos Android, a pesar de que hay otras alternativas como MacOS para sobremesa o IOS en móviles.

Para garantizar el cumplimiento de estos requisitos no funcionales se estableció que la prueba consistiría en que se pudiera ejecutar correctamente el juego en la plataforma deseada. Después de poder ejecutar el juego en las distintas plataformas sin ningún problema se concluyó que la prueba estaba superada.

6.2.2 Pruebas de compatibilidad

En cuanto a la compatibilidad lo que se buscaba era que, al tratarse de un juego multijugador, éste permitiera que se jugara desde distintos dispositivos la misma partida. A esto se le denomina, en el argot de los videojuegos, juego cruzado.

Para realizar estas pruebas se procedió a crear una partida con dos jugadores. Uno accedió al juego vía ordenador y el segundo desde su dispositivo móvil Android. Se jugó la partida completa y se concluyó entonces que la partida podía ser jugada sin ningún problema, puesto que no se produjo ningún error. Para el juego es indistinto que una persona se conecte desde un dispositivo u otro.

6.2.3 Pruebas de rendimiento

Para realizar y analizar las pruebas de rendimiento se hizo uso de la herramienta Profiler de Unity. El Profiler ayuda a optimizar el juego. Se trata de un reporte en ejecución que calcula el tiempo empleado en la ejecución distintas áreas del juego o los recursos consumidos. En la figura 21 se puede observar las gráficas que se generan atendiendo a los diferentes recursos que se pueden monitorizar. En este caso, las gráficas se corresponden con el uso de la CPU, el uso de la GPU, el renderizado y la memoria.

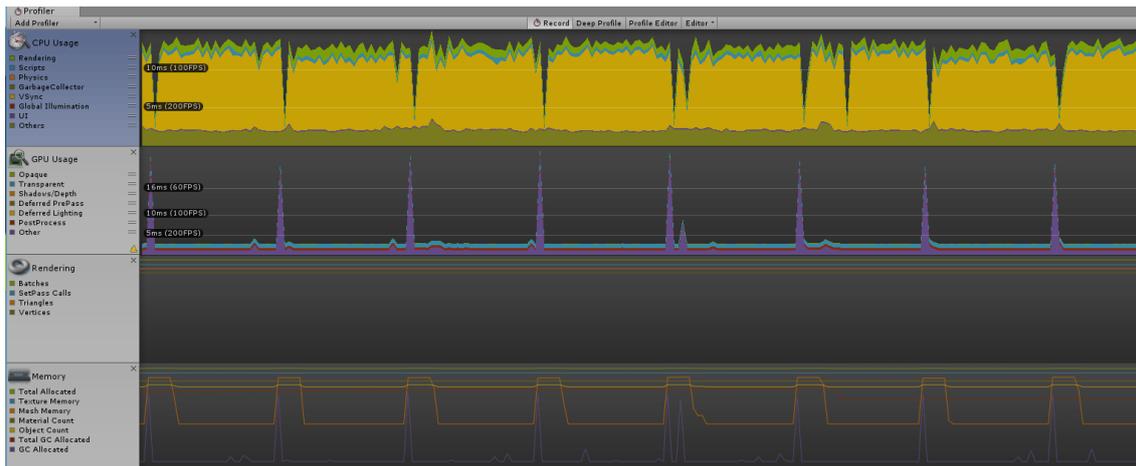


Figura 21 – Unity Profiler

A parte de las gráficas que se generan también se puede observar en detalle cada uno de los aspectos que se contemplan con información mucho más detallada. Para este proyecto se habían establecido como requisitos no funcionales de rendimiento que el juego no consumiera más de 2GB de memoria y que no bajase de los 24 fotogramas por segundo que es el mínimo necesario para que el ojo humano tenga una sensación de fluidez y velocidad.

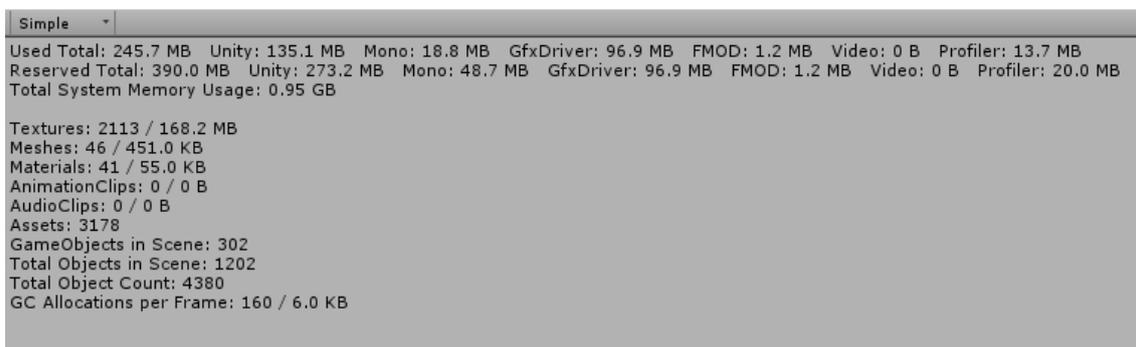


Figura 22 – Información de uso de memoria

En la figura anterior (figura 22) se puede observar con mayor detalle el uso de memoria del sistema. En la primera línea aparece el total usado (245,7 MB) y el desglose de esa cantidad por elementos. Observamos como Unity es el elemento que más consume (135,1 MB), mono, que es la memoria usada por el código para guardar variables, consume (18,8 MB), GfxDriver, la memoria que el controlador usa para gestionar texturas y renderización, (96,9 MB) o el propio Profiler que al activar su ejecución también consume recursos (13,7 MB).

En la segunda línea de la imagen se observa una porción de memoria que el propio juego reserva (390,0 MB) y en la tercera el uso total de memoria del que se hace uso. Vemos como este uso total es de 0,95 GB luego determinamos que cumplimos que se use menos de 2GB de memoria. Atendiendo a los datos, se puede observar que este último valor es mayor que la suma tanto de la memoria usada como de la memoria reservada y esto es porque el juego también almacena en memoria, una vez ejecutado, otros elementos como texturas o assets que suponen un aumento leve en la capacidad requerida.

Para comprobar que el juego superara los 24 fotogramas por segundo se observó, al igual que en la imagen anterior la información detallada de la memoria, la información en detalle del uso de la CPU. Esta información está representada en la figura 23. En ella se puede observar que la tasa de refresco de la CPU es de 16,16 ms. Con esta cantidad se aplicó la fórmula $FPS = 1/T(s)$ con un resultado de aproximadamente 61 fotogramas por segundo. Luego se concluyó también que se cumplía el requisito no funcional establecido.

Hierarchy		CPU:16.16ms GPU:2.00ms		GC		
Overview	Total	Self	Calls	GC Alloc	Time ms	
Initialization.PlayerUpdateTime	59.0%	0.1%	1	0 B	9.54	
EditorOverhead	18.6%	18.6%	2	0 B	3.01	
Profiler.CollectGlobalStats	9.6%	9.6%	1	0 B	1.56	
Camera.Render	5.6%	0.4%	1	0 B	0.91	
Update.ScriptRunBehaviourUpdate	1.0%	0.0%	1	0 B	0.17	
BehaviourUpdate	1.0%	0.0%	1	0 B	0.17	
PreLateUpdate.ScriptRunBehaviourLateUpdate	0.7%	0.0%	1	0 B	0.12	
PostLateUpdate.PlayerUpdateCanvas	0.6%	0.0%	1	0 B	0.11	
PostLateUpdate.UpdateAudio	0.4%	0.0%	1	0 B	0.07	
Profiler.CollectStats	0.4%	0.4%	1	0 B	0.06	
UGUI.Rendering.EmitWorldScreenspaceCameraGeometry	0.3%	0.3%	1	0 B	0.09	
EarlyUpdate.UpdateMainGameViewRect	0.3%	0.0%	1	0 B	0.04	

Figura 23 – Información de uso de CPU

A parte de esta tasa de refresco también se pueden observar otras tareas realizadas y el uso en porcentaje y tiempo que hacen de la CPU. En la imagen, por ejemplo, se puede ver que un 59% del uso de la CPU está dedicado a tareas de inicialización, o lo que es lo mismo 9,54 ms, un 18.1% corresponde a *overhead* del editor en tiempo de ejecución (o 3,01 ms) y un 5,6% (0,91 ms), por comentar otro aspecto interesante, a tareas de renderizado de la cámara. El desglose es mucho más amplio pero el resto de los procesos usan un porcentaje despreciable, al menos en este juego.

6.3 Pruebas de aceptación

Una vez finalizado el juego se realizaron, también, las pruebas de aceptación. Para realizar dichas pruebas se acudió a dos fuentes, a parte de la propia aceptación del director del proyecto, Carlos. Por una parte, nos pusimos en contacto con la empresa de desarrollo de videojuegos Concano Games, a los que le agradezco su ayuda puesto que no tenían ninguna responsabilidad sobre el proyecto, con el objetivo de que probaran el juego y dieran sus impresiones sobre la apariencia física del mismo.

Tras jugar un par de partidas comentaron el buen trabajo realizado y aprobaron los elementos gráficos. Como es obvio, al tratarse de una empresa seria y profesional propusieron algunas mejoras en el terreno gráfico que fueron tenidas en cuenta y otras que a buen seguro serán estudiadas y evaluadas para proceder con su implementación en el futuro.

Para completar las pruebas de aceptación se acudió también a compañeros de clase, un conjunto de cuatro personas, para que probaran el juego; dado que ya lo habían probado en formato físico en la asignatura de gestión de proyectos. Para extraer unas conclusiones lo más completas posibles se les pidió un reporte de opinión después de jugar. Todos concluyeron que se trataba de una versión digital intuitiva, divertida y fiel al juego real. Por lo que se decidió dar por concluido el desarrollo de la primera versión jugable del juego al haber sido ampliamente aceptada por los usuarios.

Hay que mencionar que este juego será, en un futuro cercano, usado por los alumnos del grado. Esto proporcionará nuevos reportes y añadirá una nueva dimensión a las pruebas de aceptación. Una nueva dimensión que será tenida en cuenta para trabajos próximos.

7. Conclusiones y trabajos futuros

Nunca está de más hacer una recapitulación de aquello que se ha hecho anteriormente. Y ahora estoy en este punto. Después de finalizar un proyecto que he afrontado con ilusión y ganas durante algo más de tres meses. Me encuentro feliz porque creo que mi objetivo está cumplido con creces.

7.1 Conclusiones

El proyecto desarrollado cumple con el objetivo planeado ya que ha trasladado de forma satisfactoria el juego de tablero Deliver a un formato digital más usable en aulas. Además, se ha mejorado en diversos aspectos como la inclusión de variedad en cuanto a los tableros y la aleatoriedad de los empleados asignados, la inclusión de una clasificación *online*, la posibilidad del juego multijugador y juego cruzado entre dispositivos de distinto tipo (pc y dispositivo móvil).

Al principio de este camino acepté un reto ambicioso por el mero hecho de probarme a mí mismo que podía hacer algo que no estuviera estrictamente ligado al conjunto de conocimientos adquiridos en la carrera. Espero que se me entienda, no es que no haya realizado proyectos software antes, sino que ninguno de ellos había estado enfocado en el campo de los videojuegos que tanto me apasiona.

Creo que mi trabajo y esfuerzo han valido la pena. Primero porque en este proceso, que no siempre ha sido sencillo, me he encontrado con obstáculos que hábilmente he solventado, de los cuales he aprendido mucho. Y segundo por haber creado un juego, un proyecto, que estoy seguro no quedará “abandonado en el fondo un cajón”. Habrá, digo convencido de ello, gente a los que servirá de ayuda.

7.2 Trabajos futuros

Por supuesto la historia no queda aquí. Si bien es cierto que el juego está en una versión cien por cien jugable, sería de necio frenar este espíritu imparible que crece en mi interior. Hay aspectos variados que por falta de tiempo no han podido ser implementados pero que en un futuro lo serán.

Al tratarse de un juego que se pretende que sea utilizado por cualquier persona que quiera aprender sobre la gestión de proyectos, lo primero que quiero hacer es implementar una traducción de este juego al inglés, al tratarse de la lengua de uso más extendido por todo el mundo (obviando el chino y el español). De esta manera el juego llegará a un público más amplio.

Otra de las cosas que nos hacía especial ilusión, tanto a Carlos como a mí era incluir sonidos. Cualquier juego que se precie debe incluir sonidos. Los sonidos hacen de la experiencia de juego más rica y por tanto aumenta el interés por parte del usuario final. Una música calmada o un sonido esporádico pueden hacer que un juego bueno, como es éste, se convierta en algo fabuloso. En caso de que no guste el componente audible, el usuario siempre podrá silenciar el volumen.

En cuanto al aspecto gráfico se desea mejorar teniendo en cuenta la retroalimentación obtenida. En este sentido no sólo se plantea un cambio de colores sino la inclusión de pieles que aporten distintas temáticas al juego. Esto permitirá acercar el juego a distintos públicos, por ejemplo, a niños a través de una temática infantil o a profesionales de distintos ámbitos con temáticas relacionadas con su profesión.

Como última mejora, había pensado en incluir una base de datos en la nube que gestione el registro de los usuarios. Si bien es cierto que el mundo de desarrollo de videojuegos me llama especialmente la atención, lo que más me gusta de la informática es todo lo relacionado con la gestión de datos y veo en este proyecto una clara oportunidad para abrirme a nuevos horizontes que no voy a desaprovechar.

8. Referencias

- Arlow, J. & Neustadt, I. (2005). *UML 2 and the Unified Process: Practical Object-Oriented Analysis and Design*. Addison-Wesley.
- Arriola Landa, N. (2013). *Unity: diseño y programación de videojuegos*. RedUSERS.
- Gamma, E. (2002). *Patrones de diseño*. Addison-Wesley.
- Gibson Bond, J. (2014). *Introduction to Game Design, Prototyping, and Development*. Pearson.
- Photon Engine. (2018). *Class List*. Recuperado en febrero de 2019, de Photon Engine Networking: <https://doc-api.photonengine.com/en/pun/current/annotated.html>
- Photon Engine. (2018). *Public API*. Recuperado en febrero de 2019, de Photon Engine Networking: https://doc-api.photonengine.com/en/pun/current/group_public_api.html
- Rusty Harold, E & Scott Means, W. (2004). *XML in a Nutshell*. O' Reilly.
- Sommerville, I. (2012). *Ingeniería del software*. 9a Edición, Addison-Wesley.
- Taylor, R. N., Medvidovic, N. & Dashofy, E. M. (2009). *Software Architecture: Foundations, Theory, and Practice*. Wiley John + Sons.
- Unity Technologies. (2018). *Game Objects*. Recuperado en febrero de 2019, de Unity Manual: <https://docs.unity3d.com/Manual/GameObjects.html>
- Unity Technologies. (2018). *Platform-specific*. Recuperado en febrero de 2019, de Unity Manual: <https://docs.unity3d.com/Manual/PlatformSpecific.html>
- Unity Technologies. (2018). *Prefabs*. Recuperado en febrero de 2019, de Unity Manual: <https://docs.unity3d.com/es/current/Manual/Prefabs.html>
- Unity Technologies. (2018). *Profiler*. Recuperado en febrero de 2019, de Unity Manual: <https://docs.unity3d.com/Manual/Profiler.html>
- Unity Technologies. (2018). *Scene*. Recuperado en febrero de 2019, de Unity Manual: <https://docs.unity3d.com/ScriptReference/SceneManagement.Scene.html>
- Unity Technologies. (2018). *Scripting Tools*. Recuperado en febrero de 2019, de Unity Manual: <https://docs.unity3d.com/Manual/ScriptingTools.html>

- Unity Technologies. (2018). *Sprites*. Recuperado en febrero de 2019, de Unity Manual: <https://docs.unity3d.com/Manual/Sprites.html>
- Unity Technologies. (2018). *Test Runner*. Recuperado en febrero de 2019, de Unity Manual: <https://docs.unity3d.com/Manual/testing-editor-test-runner.html>
- Unity Technologies. (2018). *UI System*. Recuperado en febrero de 2019, de Unity Manual: <https://docs.unity3d.com/Manual/UISystem.html>