FACULTAD DE CIENCIAS

UNIVERSIDAD DE CANTABRIA Facultad de Ciencias

Proyecto Fin de Carrera

Videojuego de tipo Arcade para Dispositivos Móviles sobre motor Unity 3D

(Arcade type video game for mobile devices developed using Unity 3D engine)

Para acceder al Título de

INGENIERO EN INFORMÁTICA

Autor: Javier Telechea Díaz

Director: Jorge Lanza Calderón

Septiembre - 2014

AGRADECIMIENTOS

Hay muchas personas a las que debo agradecer no sólo el apoyo y la ayuda recibida durante la realización de este trabajo, que no es sino el final de un camino de 5 años de carrera, sino también todo lo que me han dado durante dicho camino.

En primer lugar, doy las gracias a la persona que asumió la responsabilidad y se hizo cargo de este trabajo, Jorge Lanza, director del proyecto. Muchas gracias por tu ayuda.

En segundo lugar, pero no por ello menos importante, dar las gracias a mi compañero Adrián Crespo por toda la ayuda que he recibido de él a la hora de pegarme con Unity. No podría haber hecho este trabajo sin ti. Gracias amigo.

También doy las gracias a los otros tres compañeros que empezaron el trabajo conmigo y con Adrián: José Luis Argumosa, Rafael Pereda y Javier Somavilla. Y a Pedro Fernández, por ayudarme con la base de datos. Gracias a todos.

Pero no me puedo olvidar de todas las personas que empezaron siendo compañeros de clase y que ahora son amigos, que en mayor o menor medida me han ayudado durante estos largos 5 años de carrera. Muchas gracias a todos.

Por otro lado, doy las gracias también a todos los profesores que he tenido durante la carrera, que me han ayudado y que me han regalado parte de su conocimiento para que yo pueda decir orgulloso que soy ingeniero informático y que estudié en la Universidad de Cantabria.

Por último, no podría terminar mis agradecimientos sin nombrar a las personas que siempre han estado, están y estarán ahí. Mis padres, abuelos, tíos, primos… Mi hermana Susana. Ángela, mi novia. Y todos mis amigos.

Muchas gracias a todos.

RESUMEN

En la actualidad, una de las industrias en auge y con mayor potencial es la del entretenimiento audiovisual interactivo o, dicho de otra forma, la industria de los videojuegos.

En el mundo de los videojuegos, una mayor inversión en personal de diseño y desarrollo, publicidad o marketing, etc. no es necesariamente sinónimo de éxito, lo que incide directamente en los ingresos que dichos juegos pueden reportar en el medio plazo. En este sentido, juegos simples y sencillos, enfocados a un uso esporádico y altamente adictivos, en muchas ocasiones consiguen generar unos beneficios muy superiores.

La tendencia actual en el desarrollo de este tipo de juegos, conocidos como Arcade, se centra en los dispositivos móviles que un cada vez más numeroso grupo de usuarios emplea en su vida cotidiana. De esta forma, la plataforma de juego está siempre disponible satisfaciendo no solo las necesidades del usuario, sino también generando un mayor beneficio para el desarrollador, pues se incrementan las oportunidades de uso de la aplicación. Esta es una de las razones de la elevada recaudación obtenida por estos juegos.

Este trabajo se centra en el desarrollo de un videojuego de tipo Arcade, explotando las posibilidades que incorporan los terminales móviles como acelerómetro, tecnologías de comunicación, potentes procesadores gráficos, etc. El empleo del motor gráfico Unity 3D (3 Dimensiones) para el desarrollo, busca la interoperabilidad en múltiples plataformas.

Palabras clave: videojuego, 3D, Arcade, Android, Unity 3D, motor gráfico, aplicación móvil, servidor web.

ABSTRACT

Nowadays, one of the most booming and powerful industries is the one based on interactive audiovisual entertainment, or, in other words, the video game industry.

In the world of video games, a higher investment in design and development personnel, advertising and marketing, etc. is not necessarily synonymous with success, which directly affects the revenue that these games can bring in the medium term. In this sense, simple games, focused on occasional use and highly addictive, often generate a much higher benefits.

The current trend in the development of these games, known as Arcade, focuses on mobile devices that an increasingly large number of users employ in their daily lives. Thus, the gaming platform is always available to meet not only the needs of the user, but also generating a higher profit for the developer, because the opportunities of using the application increase. This is one of the reasons of the high revenue obtained from these games.

This work focuses on the development of an Arcade type video game exploiting the possibilities that modern mobile devices have such as accelerometer, communication technologies, powerful graphics processors, etc. By using the Unity 3D game engine for development, we search for interoperability within multiple platforms.

Keywords: video game, 3D, Arcade, Android, Unity 3D, game engine, mobile application, web server.

ÍNDICE

ÍNDICE DE FIGURAS

1 INTRODUCCIÓN

Los videojuegos se han erigido como una de las áreas más importantes dentro del mundo de la informática en general y de la ingeniería de *software* en particular. Son cada vez más las empresas que, atraídas por el gran volumen de dinero que mueve su mercado a nivel global enfocan sus líneas de desarrollo *software* a este tipo de aplicativos.

La historia de los videojuegos se inicia gracias a las contribuciones de los más grandes informáticos de la historia, como Alan Turing, padre de la inteligencia artificial, John Bennett, Alexander Douglas, Arthur Samuel, Ralph Baer, William Higinbotham, Steve Rusell o Nolan Bushnell.

Durante dos décadas, entre 1940 y 1960, éstos hombres empezaron a creer que los computadores, tal y como los conocían, podrían usarse no sólo como instrumento de trabajo, sino también como herramientas para el entretenimiento. Así, mediante programas y algoritmos de inteligencia artificial consiguieron crear juegos de ajedrez o damas en los que el oponente del usuario era la propia máquina [1].

Poco a poco, más informáticos fueron probando sus propios juegos en los laboratorios de las universidades o en sus talleres de ingeniería, hasta que algunos se compartieron para para el público en general. Tal fue el caso de *Tennis for Two* [2], considerado por muchos como el primer videojuego de la historia.

Creado por William Higinbotham [3], un ingeniero norteamericano que había participado en el Proyecto Manhattan, y con la ayuda de su compañero Robert Dvorak, el juego simulaba una partida de tenis entre dos usuarios empleando para ello la pantalla de un osciloscopio y circuitería de transistores.

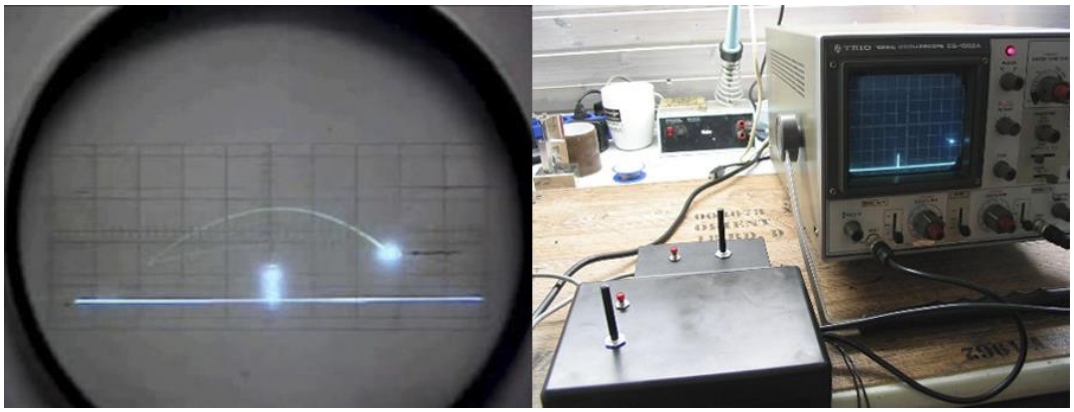

Figura 1.1 Videojuego Tennis For Two

Ya en los años 60, nace otro de los primeros videojuegos de la historia, el *Spacewar!* de Steve Russell, creado por un grupo de estudiantes del MIT (*Massachusetts Institute of Technology*) para probar el primer PDP-1 [4]. El juego causó sensación entre los miembros del MIT y numerosas copias del mismo fueron distribuidas a través de la recién creada red de ARPAnet (*Advanced Research Projects Agency Network*), que más adelante derivó en lo que hoy se conoce como Internet.

También durante esta década se desarrolló la primera videoconsola doméstica tal y como hoy las conocemos. Su creador fue Ralph Baer y la llamó Brown Box. Sin embargo, fue la empresa *Magnavox* la que se hizo cargo del proyecto y empezó a comercializarla con el nombre de *Magnavox Odyssey*.

Durante la década de los 70 se produjo el despegue definitivo de los videojuegos. Nolan Bushnell, que había conocido *Spacewar!* y tenía interés en el negocio del entretenimiento, así como altos conocimientos en el campo de la ingeniería eléctrica, introdujo dicho juego en los salones recreativos del país. Nolan contrató a Steve Russell, creador del juego, y creó la primera empresa que distribuía máquinas recreativas. Dicha empresa terminó por llamarse Atari, y años más tarde comercializó también su versión del *Tennis for Two* de Higinbotham, el clásico *Pong*.

A partir de este momento, se impulsa notablemente el desarrollo de videojuegos en una gran variedad de entornos. La llegada de los videojuegos a los ordenadores personales, la proliferación de videoconsolas domésticas y de máquinas recreativas, y la entrada al mundo de los videojuegos de los creadores japoneses con el *Space Invaders* de Taito, supusieron un punto y aparte en el sector.

La década de los 80 es considerada como la edad de oro de los videojuegos. Los nombres quedan para la historia: ZX Spectrum, Amstrad CPC (*Color Personal Computer*), Commodore 64, Namco, Nintendo, *Pac-Man*, *Donkey Kong*, Miyamoto, Sega, *Game & Watch*, Nintendo Entertainment System (Figura 1.2), *Super Mario Bros.*, *The Legend of Zelda*, *Final Fantasy*, Capcom, Konami, Sega Master System, etc.

Figura 1.2 Videoconsola Nintendo Entertainment System

Desde entonces, el mundo de los videojuegos no ha parado de evolucionar y crecer desde un conjunto de aplicaciones para el entretenimiento hasta convertirse en lo que es hoy, una industria que mueve a millones de usuarios y profesionales en todo el mundo y que alcanza unas recaudaciones que ya superan a otras grandes industrias como el cine o la música.

Además, se han convertido en una parte importante de la sociedad y son una de las formas de ocio preferidas de la población de cualquier edad y sexo. Por otro lado, os videojuegos han adquirido un importante valor como medio de transmisión de información, lo que puede servir como recurso educativo, formativo, concienciador e incluso rehabilitador, y de hecho ya se utiliza para estos objetivos.

1.1 Objetivos y motivación

Tras observar la relevancia de los videojuegos en la sociedad y en la historia de la informática, es difícil no darles la importancia que se merecen.

En este sentido, la principal motivación para crear un videojuego es lograr entretener a los usuarios y que éstos disfruten de él, usándolo durante el mayor tiempo posible. Por este motivo, aspectos como un interfaz gráfico atractivo, jugabilidad, adaptabilidad al usuario, etc. son elementos a tener en consideración a la hora de definir la base de un videojuego.

Adicionalmente, como se ha descrito anteriormente, el videojuego debe estar disponible en el entorno que sea más cercano al usuario. En la actualidad las plataformas *hardware* más empleadas son los dispositivos móviles, teléfonos inteligentes (*smartphones*) y tabletas (*tablets*). Estos aparatos incluyen además un nuevo abanico de posibilidades para el sector del videojuego gracias a las distintas tecnologías que incorporan, no sólo de comunicaciones: Wi-Fi (*Wireless Fidelity*), Bluetooth, NFC (*Near Field Communication*), etc., sino también vinculadas a

la interacción con el usuario (pantallas táctiles, sensores como acelerómetros o giroscopios, micrófonos, etc.).

En los últimos años, se pueden contar multitud de ejemplos de franquicias multimillonarias que han nacido de una idea sencilla y adictiva y se han convertido en verdaderos gigantes del sector. Sirvan como muestra *Angry Birds*, *Plants vs. Zombies*, de los cuales se puede observar una captura de pantalla en la Figura 1.3, *Candy Crush,* etc.

Figura 1.3 Capturas de pantalla de *Angry Birds* y *Plants vs. Zombies*, dos ejemplos de juegos de tipo Arcade

Para evaluar el funcionamiento de alguna de ellas, en este trabajo se llevará a cabo un videojuego de tipo Arcade. Este modelo de juego se caracteriza por ofrecer partidas relativamente rápidas, sencillas, desafiantes y adictivas, donde la historia y la relación entre partidas queda en un segundo plano.

Así, en este proyecto se realiza el seguimiento de todas las etapas en el desarrollo de un juego de tipo Arcade para plataformas móviles haciendo uso de las múltiples capacidades de éstos. El juego, además, deberá permitir la interacción con otros usuarios, así como comparar las capacidades de cada uno a la hora de jugar.

1.2 CONTENIDO DE LA MEMORIA

Para explicar el proceso por el cual se llega a la consecución de los objetivos marcados, la memoria se divide en una serie de capítulos en los que se abordan los distintos aspectos tratados:

- En el capítulo 2 se analiza el estado actual de las herramientas de desarrollo de videojuegos, a partir de lo cual será posible conocer las opciones y seleccionar la tecnología que se empleará en este proyecto.

- En el capítulo 3 se describen todas las tecnologías y herramientas *software* que han sido utilizadas en algún momento del desarrollo del proyecto.

- En el capítulo 4 se muestra una descripción general del juego objeto de este proyecto, se realiza el análisis de requisitos de dicha aplicación *software* y se describe el diseño arquitectónico empleado durante el trabajo, tanto de la parte del servidor como de la aplicación móvil.

- El capítulo 5 explica las diferentes partes del proceso de desarrollo del videojuego y cómo se ha implementado cada una.

- En el capítulo 6 se aborda la fase de pruebas del desarrollo del juego y se explica en detalle cada una de las partes de la que consta.

- En el último capítulo, se incluyen las conclusiones obtenidas del trabajo realizado y se exponen diferentes posibles vías de ampliación o mejora del proyecto.

ESTADO DEL ARTE 2

Este capítulo resume el estado de las diferentes tecnologías existentes para el desarrollo de videojuegos, haciendo una comparativa entre todas ellas, lo que facilita la elección de la tecnología más adecuada para el desarrollo concreto de este videojuego.

2.1 DEFINICIÓN DE MOTOR GRÁFICO

El término "motor gráfico" o "motor de videojuegos" hace referencia a una serie de rutinas de programación que permiten el diseño, la creación, el desarrollo y la representación gráfica de un videojuego. Su funcionalidad básica es servir al desarrollador de un motor de renderizado para los gráficos ya sean en 2D (2 Dimensiones) o en 3D.

Pero además, la gran mayoría de estos motores ofrecen a su vez otras útiles características y funciones que facilitan la construcción del videojuego, como son el motor físico (*software* capaz de realizar simulaciones de ciertos sistemas físicos como la dinámica de un cuerpo rígido, el movimiento de un fluido o la elasticidad) o detector de colisiones, sonidos, *scripting*, animaciones, inteligencia artificial, redes, *streaming*, administración de memoria, etc.

La creación de estos motores supuso un antes y un después en la historia de los videojuegos. Antes de ellos, los videojuegos se desarrollaban normalmente partiendo de cero, empleando librerías de desarrollo propias de cada programador y con la necesidad de compilar un nuevo código estructural. Los videojuegos se creaban como entidades únicas y singulares, y cada juego tenía su propio motor gráfico ajustado a sus necesidades.

Hoy en día existen numerosos motores, más o menos complejos, con diferentes funcionalidades y características. Estos motores suelen ser reutilizados por las compañías desarrolladoras para la creación de diferentes juegos.

2.2 MOTORES GRÁFICOS ACTUALES

A la hora de elegir qué motor gráfico utilizar para el desarrollo de un videojuego, es fundamental tener varios aspectos en consideración. El primero y más importante es decidir la plataforma en la que correrá el juego, teniendo en cuenta el público objetivo ya que no todos los motores son multiplataforma y sirven tanto para Windows, como para Mac o Linux o, para consolas de sobremesa, para dispositivos móviles o consolas portátiles.

Por otro lado, hay que tener en cuenta que hacer un desarrollo multiplataforma conlleva un coste y una dificultad mayor que enfocarlo a una sola. Además, este tipo de proyectos suele tener el problema de que, en lugar de realizar un videojuego que saque el 100% del rendimiento de la plataforma para la que se desarrolla, al tener que trabajar en todas al mismo tiempo, se intenta llegar a un nivel medio que pueda funcionar o ser más fácil de adaptar a todas a la vez, por lo que el producto final suele perder calidad.

Sin embargo, a la hora de tomar una decisión final aún hay que tener en cuenta otras consideraciones. Además de considerar la comunidad o compañía que soporte el motor también hay que analizar si se adecua al objetivo del juego planteado.

Entre las herramientas que se pueden utilizar hoy en día para crear videojuegos existe gran variedad, lo que es algo muy positivo y muestra el creciente interés. A continuación se describen aquellas con mayor relevancia en el mercado.

2.2.1 UDK (*UNREAL DEVELOPMENT KIT*)

El motor gráfico creado por la empresa Epic Games para su serie de juegos *Unreal Tournament* (Unreal Engine) [5] es considerado como uno de los mejores y más potentes del mundo. Es utilizado por muchas empresas líderes del sector para desarrollo de videojuegos AAA o triple-A (juegos de muy alta calidad con presupuestos muy grandes), pero su licencia es sumamente cara.

Sin embargo, existe una versión gratuita del motor, el Unreal Development Kit, que permite crear juegos con grandes acabados y familiarizarse con el desarrollo profesional de videojuegos. Esta versión permite exportar los resultados a múltiples plataformas PC (*Personal Computer*), Mac Os, iOs, Android, PlayStation 3, Xbox 360, PlayStation Vita, Nintendo Wii, o web, pero está restringida para uso no comercial.

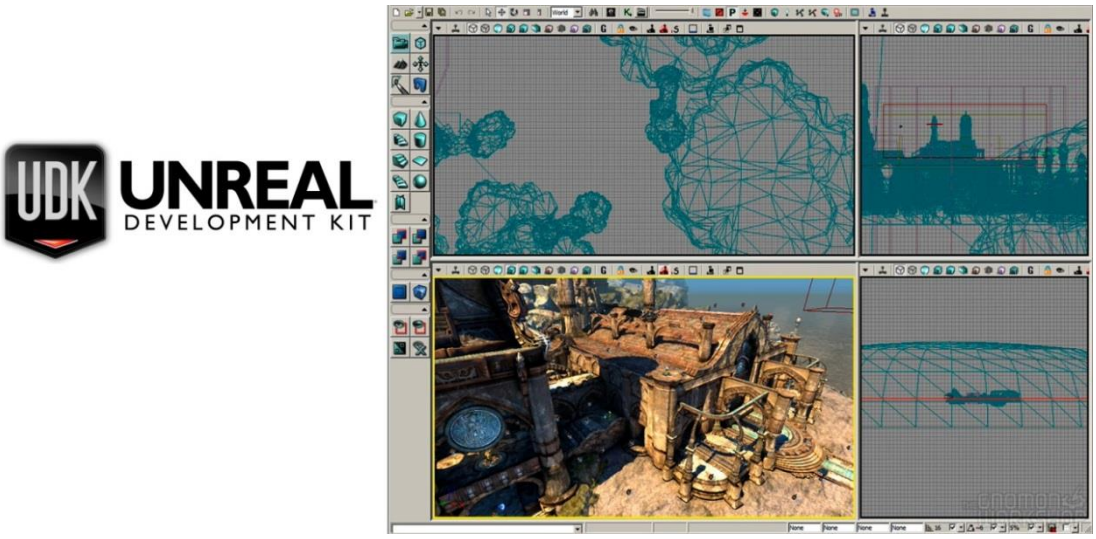

Figura 2.1 Captura de pantalla de un proyecto realizado con el programa UDK

2.2.2 TORQUE

Torque, creado por la empresa Garage Games [6], es uno de los motores gráficos más antiguos del mercado, lo que no le ha impedido evolucionar constantemente para llegar al día de hoy siendo todavía uno de los más importantes. Dispone de una versión gratuita bastante completa para desarrollar juegos menos profesionales. Sus prestaciones son bastante altas y eficientes y, además, su versatilidad y accesibilidad es excelente.

El principal inconveniente de este motor es que ofrece un sistema del tipo WYSIWYG *("What you see is what you get"*) bastante complejo con el que se hace difícil entrar en un desarrollo profundo.

Torque ofrece tres variantes distintas para adaptarse bien a cada proyecto: Torque 3D, el motor clásico mejorado con las últimas tecnologías, Torque 2D, para el desarrollo de juegos en 2D, y iTorque2D, especialmente diseñado para el desarrollo de juegos para plataformas móviles con iOS (*iPhone/iPod/iPad Operating System*).

Figura 2.2 Imagen promocional del motor gráfico Torque 3D

2.2.3 CryEngine

CryEngine es el potentísimo motor gráfico desarrollado por la empresa Crytek [7], creadores de grandes obras artísticas como la serie *Crysis*. Ofrece muchas posibilidades de desarrollo y es considerado, al igual que el Unreal Engine, como uno de los más potentes del mundo.

Al igual que el Unreal Engine, este motor es utilizado principalmente en juegos muy profesionales y presenta una curva de aprendizaje compleja. Para su utilización más amateur existe una versión gratuita del motor, con todas sus funcionalidades, pero restringida también para uso no comercial. Permite exportar a plataformas como Windows, Xbox 360 o PlayStation 3.

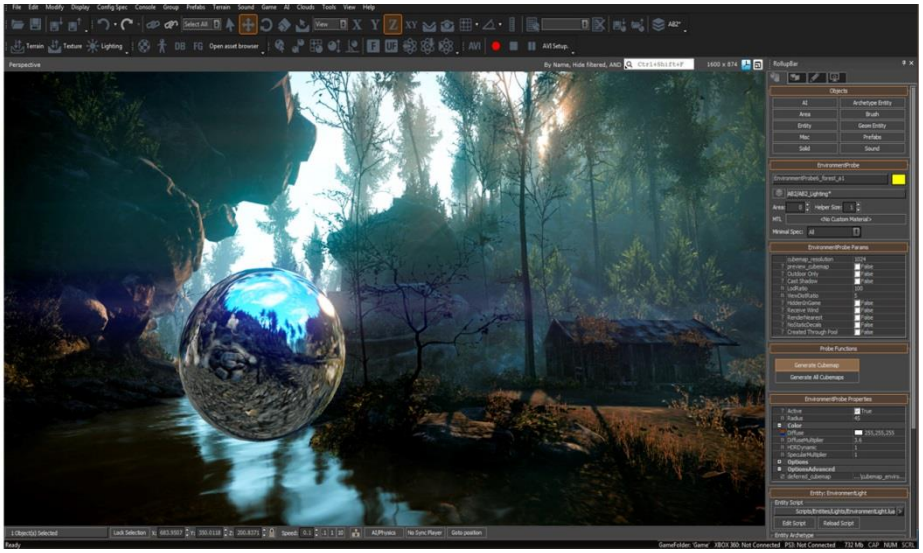

Figura 2.3 Captura de pantalla de un proyecto realizado con el motor gráfico CryEngine

2.2.4 Shiva 3D

Shiva 3D [8] está considerado como uno de los motores más compatibles del mundo, ya que es capaz de exportar a Windows, Mac OS, GNU (*GNU is Not Unix!*)/Linux, iOS, Android y Wii.

Las funciones y elementos de desarrollo que ofrece esta herramienta gratuita son un tanto peculiares, y se alejan de las fórmulas estándar vistas en otros motores, lo que puede resultar en una complejidad adicional si ya se está familiarizado con éstas. Sin embargo, una vez se ha superado el primer contacto y se coge destreza, parece resultar una aplicación cómoda y versátil.

La parte negativa de este motor es que no se ha prodigado mucho en proyectos profesionales, por lo que no tiene una comunidad de desarrolladores tan grande como otras

plataformas, lo que dificulta encontrar apoyos para la resolución de dudas y problemas que surjan de su empleo.

Figura 2.4 Imagen promocional del *software* Shiva 3D

2.2.5 UNITY 3D

Unity 3D [9], es un motor con una versión gratuita bastante potente, fácil de utilizar y de aprender a usar, y con una comunidad de usuarios muy grande, lo que permite encontrar multitud de tutoriales, ayudas, solución de problemas, etc. Además, Unity se ha convertido, de un tiempo a esta parte, en uno de los motores gráficos más famosos y utilizados en todo el mundo, no sólo por estudios independientes, sino también por las grandes empresas del sector.

Unity Technologies fue fundada en 2004 por David Helgason, Nicholas Francis y Joachim Ante con la idea de democratizar el desarrollo de juegos. Su motor gráfico pretendía enfocarse en las necesidades de los desarrolladores independientes que no podían crear sus propios motores ni disponían de dinero suficiente como para comprar las licencias de los motores profesionales disponibles. Unity nació para hacer que el desarrollo de contenidos en 2D y 3D fuera lo más accesible a tantas personas del mundo como fuera posible.

Figura 2.5 Imagen de arranque del *software* Unity 4

La versión gratuita del motor data del año 2009 y, para entonces, Unity ya se había convertido en todo un gigante a nivel mundial, sobre todo para crear juegos destinados a plataformas móviles como iOs o Android.

A día de hoy es, sin duda, el motor gráfico más utilizado a nivel global. Es, sobre todo, utilizado por desarrolladores independientes (en el año 2012 alcanzó la cifra del millón de desarrolladoras registradas), y permite crear juegos para casi todas las plataformas: Windows, Mac OS, Linux, Xbox 360, Xbox One, PlayStation 3, PlayStation 4, PlayStation Vita, Wii, Wii U, Nintendo 3DS, iOs, Android y Windows Phone, además de contar con un *plug-in* web para desarrollar juegos para navegador.

Figura 2.6 Captura de pantalla de un proyecto realizado en Unity 3D

A parte de estas cinco herramientas descritas, como ya se ha dicho, existen muchas más con diferentes características y funcionalidades que las hacen mejores o peores para según qué proyecto. Así, existen aplicaciones más centradas en el desarrollo de videojuegos para plataformas móviles, por ejemplo, como GameMaker, Sparrow o Cocos2D.

2.3 COMPARATIVA

A continuación se muestra una tabla comparativa de estos cinco motores teniendo en cuenta diferentes aspectos, tanto técnicos como empresariales (Figura 2.7).

Motores:	UDK	Torque	CryEngine	Shiva	Unity
Plataformas de distribución	Mac Os, Windows, iOS, Android, PlayStation 3, Xbox 360, PlayStation Vita, Nintendo Wii, Web	Mac OS, Windows, Linux, iOs, Nintendo Wii, Xbox 360, Web	Windows, Xbox 360, PlayStation 3	Mac Os, Windows, Linux, iOS, Android, Nintendo Wii, Web	Mac OS, Windows, Linux, iOS, Android, Windows Phone, Xbox 360, Xbox One, PlayStation 3, PlayStation 4, PlayStation Vita, Wii, Wii U, Nintendo 3DS
2D/3D	2D y 3D	2D y 3D (distintos programas)	3D	2D y 3D	2D y 3D
IDE	Si	Si (Torque 3D sólo para Windows)	Si	Si	Si
Lenguajes de programación	Unreal Script	TorqueScript	VisualScript	C++, LUA	C#, JavaScript, Boo
Licencia	Gratuita. Restringida a uso no comercial	Gratuita	Gratuita. Restringida a uso no comercial	Gratuita	Gratuita
Comunidad	Grande	Mediana	Profesional	Pequeña	Muy grande
Curva de aprendizaje	Compleja	Compleja	Compleja	Normal	Normal

Figura 2.7 Tabla comparativa de los cinco motores gráficos vistos

A la hora de elegir la plataforma más adecuada para este proyecto tuve en cuenta todo lo que aparece recogido en la tabla incluída más arriba. De esta forma, y teniendo en cuenta la naturaleza del videojuego que se quería desarrollar (un juego de tipo Arcade para Android), la herramienta elegida fue el motor gráfico Unity 3D.

Uno de los primeros motivos por los que pensé en elegir ésta como la plataforma para desarrollar el proyecto era el hecho de ser el motor más utilizado a nivel global y el que mayor número de desarrolladores registrados tenía. Supuse, por lo tanto, que aprender a usarlo podría resultar de gran importancia de cara a una posible entrada futura en el mercado laboral relacionado con los videojuegos.

En segundo lugar, tuve en cuenta aspectos como la complejidad de la curva de aprendizaje de cada herramienta, el tipo de licencia y, sobre todo, el tamaño de la comunidad, pues preveía que iba a tener que recurrir a ella en más de una ocasión.

3 TECNOLOGÍAS Y HERRAMIENTAS UTILIZADAS

En este tercer capítulo se verá cuáles han sido las diferentes tecnologías, herramientas y lenguajes de programación utilizados durante el desarrollo del proyecto, identificando para cada una de ellas, cuáles han sido exactamente las funciones y utilidades concretas que han ayudado a la consecución del trabajo.

3.1 TECNOLOGÍAS UTILIZADAS

3.1.1 JAVASCRIPT

JavaScript [10] (abreviado comúnmente como "JS") es un lenguaje de programación interpretado, que se define como orientado a objetos, imperativo, dinámico, basado en prototipos y débilmente tipado. Generalmente, es usado para definir la ejecución de tareas en el equipo cliente dentro de una plataforma web cliente-servidor, permitiendo una mejor gestión de la interfaz de usuario de páginas web dinámicas.

Aunque considerado un lenguaje principalmente orientado a la programación de páginas web, actualmente se emplea también para programación asíncrona en el entorno del servidor bajo plataformas como `node.js`. Además, para el caso de este proyecto, es uno de los lenguajes utilizados para programar los *scripts* que utiliza la herramienta Unity.

3.1.2 C#

C# [11] es un lenguaje de programación orientado a objetos desarrollado y estandarizado por Microsoft como parte de su plataforma .NET, cuya licencia se ha abierto recientemente en casi su totalidad para permitir su uso en otras plataformas de forma optimizada. En este sentido, se trata de uno de los lenguajes de programación diseñados para la infraestructura de lenguaje común y mantiene numerosas similitudes con Java.

Al igual que en el caso anterior, este lenguaje se ha utilizado también en este proyecto dentro del *scripting* de Unity, con el fin de orientar su versatilidad a la programación de videojuegos.

3.1.3 PHP (*PHP HYPERTEXT PREPROCESSOR*)

PHP [12] es otro lenguaje de programación de uso general principalmente utilizado para el desarrollo web de contenido dinámico. Al contrario que JavaScript, este lenguaje es usado en su forma del lado del servidor (*server-side*). El código es interpretado por un servidor web con un módulo de procesador de PHP que genera la página web resultante.

Así, el uso que se la ha dado a este lenguaje en este proyecto es el de generar las páginas web que acceden a las bases de datos en las que se encuentran los usuarios y los tiempos asociados a ellos dependiendo del nivel del juego. Dicho de otro modo, los códigos PHP de este proyecto, son llamados desde el juego a través de la URL (*Uniform Resource Locator*) del servidor en el que se encuentran y se encargan de hacer las consultas a las bases de datos para generar el ránking que aparece en el juego o para insertar un nuevo usuario con su tiempo asociado a las mismas.

3.1.4 MYSQL (*MY STRUCTURED QUERY LANGUAGE*)

MySQL [13] es un sistema de gestión de bases de datos relacional, multihilo y multiusuario que se encuentra desarrollado en la mayoría de servidores y plataformas web. Es fácil de instalar y configurar y destaca por su escalabilidad, robustez y conectividad.

En este proyecto, su uso ha consistido en ser el sistema de gestión de las bases de datos en las que se guarda la información de los usuarios con sus tiempos.

3.1.5 ANDROID SDK (*SOFTWARE DEVELOPMENT KIT*)

El SDK de Android [14] es el conjunto de herramientas de desarrollo *software* que permite programar aplicaciones para dicha plataforma. Este SDK incluye, además de las herramientas para desarrollar y compilar aplicaciones para esta plataforma móvil, un depurador de código y un simulador de teléfono basado en QEMU (*Quick Emulator*).

La herramienta Unity permite exportar los juegos a cualquier plataforma, pero para poder generar el ejecutable necesita el SDK correspondiente. Así pues, para la creación del juego, la función de este SDK ha sido la de poder usarlo con Unity para poder crear el ejecutable del juego en plataformas Android.

3.1.6 APACHE

Apache [15] es un servidor web HTTP (*HyperText Transfer Protocol*) de código abierto para plataformas Unix (BSD [*Berkeley Software Distribution*], GNU/Linux, etc.), Microsoft Windows, Mac OS y otras. Se trata de uno de los servidores web más utilizados en la actualidad debido a su gran variedad de características altamente configurables, bases de datos de autenticación y negociado de contenido.

El uso del servidor Apache es necesario en este proyecto para la gestión del servidor físico en modo local, que se utiliza para conectar con los programas PHP que acceden a las bases de datos.

3.2 HERRAMIENTAS

3.2.1 UNITY 4

Unity es un ecosistema de desarrollo de juegos, un poderoso motor de renderizado totalmente integrado con un conjunto completo de herramientas intuitivas y flujos de trabajo rápido para crear contenido 3D interactivo en multitud de distintas plataformas. Es uno de los motores gráficos más importantes del mundo, gracias a su versatilidad, adaptabilidad, portabilidad, accesibilidad, rendimiento y rapidez. Constituye, además, una de las mejores herramientas *software* para empezar a crear videojuegos de gran calidad para cualquier plataforma tanto en 3D como en 2D, gracias también a la enorme comunidad de usuarios que lo utilizan. Anteriormente ya se ha abordado este motor gráfico, pero en este capítulo se verán algunas de sus características más técnicas.

Unity soporta la integración con los siguientes programas de diseño 3D: 3D Studio Max, Maya, Blender, Softimage, ZBrush, Modo, Cheetah 3D, Cinema 4D, Adobe Photoshop, Adobe Fireworks y Allegorithmic Substance.

El motor gráfico utiliza Direct3D (Windows), OpenGL (*Open Graphics Library*, Mac y Linux), OpenGL ES (*Open Graphics Library for Embedded Systems*, Android y iOS) y APIs (*Application Programming Interface*), propietarias como las de la Nintendo Wii. Además, soporta mapeado de relieve, reflexión de mapeado, mapeado por paralaje, sombras dinámicas usando mapas de sombras, pantalla de espacio de oclusión ambiental, etc.

El *scripting* de Unity mencionado anteriormente, se realiza a través de MonoDevelop, herramienta de la que se tratará más adelante.

Unity integra de forma nativa, su propio sistema de animación, singularmente potente y flexible llamado Mecanim animation [16], desarrollado por la propia empresa Unity Technologies. Esta poderosa herramienta de Unity incluye el uso de *retargeting* en sus animaciones, así como árboles de mezcla y máquinas de estado para construir animaciones.

Unity 3D es, como se ha explicado en el segundo capítulo, el motor gráfico con el que se ha desarrollado el juego. Se convierte así en la herramienta más importante de todo el proyecto. Todo el trabajo ha girado en torno a él.

3.2.2 MonoDevelop

MonoDevelop [17] es un entorno de desarrollo integrado (IDE, *Integrated Development Environment*) libre y gratuito diseñado principalmente para lenguajes .NET como C#, Boo o Java, entre otros. Se trata de uno de los IDEs más importantes del mundo con gran cantidad de herramientas y funcionalidades, y con soporte completo para GNU/Linux, Microsoft Windows y Mac OS.

MonoDevelop tiene una versión personalizada para el entorno de Unity, y es la herramienta que se ha utilizado durante el desarrollo del *scripting* en JavaScript y C# de este proyecto.

Figura 3.1 Logo de la versión de Monodevelop para Unity

3.2.3 Adobe Photoshop CS 6

Adobe Photoshop [18] es, probablemente, la aplicación más famosa y utilizada a nivel global para la edición y el retoque fotográfico, así como para todo tipo de diseño gráfico en general.

El uso que se le ha dado a esta herramienta durante el desarrollo del videojuego ha sido importante para el diseño de toda la interfaz gráfica. Todo, desde los botones hasta los fondos, pasando por las instrucciones del juego, los corazones de las vidas, etc. ha sido diseñado con Photoshop.

3.2.4 Audacity

Audacity [19] es un programa de grabación y edición de audio multipista, multiplataforma y libre, distribuido bajo la licencia GPL (*GNU General Public License*). Se trata del editor y grabador de audio más difundido en los sistemas GNU/Linux, y uno de los más famosos en Microsoft Windows.

Durante el desarrollo de este juego su uso se ha centrado en la edición de audios y efectos sonoros.

3.2.5 Servidor web: LAMP

El servidor web es necesario para poder implementar la comunicación del juego con la base de datos a través del servidor Apache en modo local desplegado con LAMP [20], un conjunto de programas utilizados para desarrollar sitios web dinámicos con este tipo de servidores sobre sistemas operativos GNU/Linux.

Sus siglas significan Linux + Apache (ver apartado 3.1.6) + MySQL (ver apartado 3.1.4) + PHP (ver apartado 3.1.3).

En la Figura 3.2 se puede observar una representación gráfica de la estructura de este tipo de servidor web.

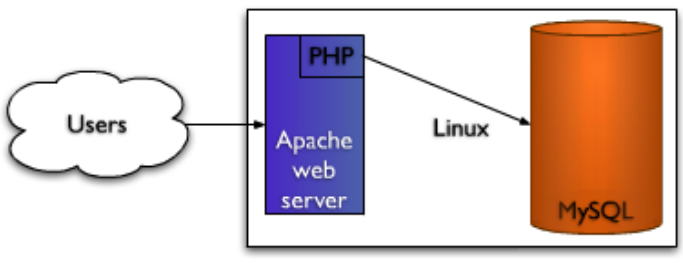

Figura 3.2 Esquema gráfico del funcionamiento de un servidor web LAMP

4 DESCRIPCIÓN DEL TRABAJO Y ANÁLISIS DE REQUISITOS

En este capítulo se realiza una descripción general de las funcionalidades y el comportamiento del juego. Se parte de un análisis de los requerimientos que debe cumplir el juego para lograr satisfacer al usuario final y generar una aplicación robusta. A continuación se plantea la arquitectura funcional con la que dar respuesta a estas necesidades y se desarrolla la metodología aplicada para hacerla realidad.

4.1 DESCRIPCIÓN GENERAL DEL JUEGO

La aplicación consiste en un videojuego de tipo Arcade al que se ha llamado *Humpty Dumpty's Rolling Crush*. Se basa en la idea del clásico juego de laberinto en el que el jugador debe guiar una canica por un circuito predeterminado hasta llegar a la meta, evitando los agujeros que se encuentre en su camino (Figura 4.1).

Figura 4.1 Juego de tablero *Labyrinth* en el que se basa nuestro videojuego

Para este proyecto se han creado tres tipos de obstáculos distintos -no sólo los clásicos agujeros- que, durante el recorrido, el jugador se irá encontrando de forma aleatoria: badén piramidal, badén redondeado y agujero.

Con el fin de diferenciar este juego de otros de la misma temática (laberinto) y dotarlo de mayor originalidad, se define que el objeto que el jugador debe llevar a la meta es un huevo, el popular personaje de la literatura británica *Humpty Dumpty* [21], en lugar del usual objeto esférico (pelota, bola, etc.).

Este aspecto aporta una nueva dimensión al juego, ya que un huevo no gira igual que una canica o una pelota, pues su movimiento es mucho más errático y caótico, lo cual provoca que el jugador no sepa cómo va a reaccionar o hacia donde girará en cada momento. Además, un huevo es un objeto bastante frágil, lo que significa que choques a demasiada velocidad contra una pared u obstáculo provocarán la rotura de éste.

El juego está desarrollado para ejecutarse y controlarse en dispositivos móviles. El movimiento y aceleración del huevo a lo largo del recorrido se realiza mediante los sensores de los acelerómetros. Por lo tanto, el movimiento y la aceleración del huevo dependen de hacia dónde y cuánto incline el jugador su *smartphone* o *Tablet* (Figura 4.2).

Figura 4.2 Ejemplo de uso de la aplicación

Adicionalmente, el juego define tres niveles con dificultad creciente. Esta dificultad viene dada no solo por la longitud del laberinto y por la cantidad de obstáculos en el camino, sino que también está directamente relacionada con la fragilidad del huevo. El jugador tiene tres intentos o vidas para superar cada nivel, que pierde cada vez que el huevo se rompe o cae por un abismo o agujero. Al ser cada nivel más largo y con más obstáculos, las probabilidades de perder más vidas aumentan.

Por otro lado, el juego es también un desafío contrarreloj, por lo que se muestra un cronómetro que va contando lo que tarda el jugador en completar el laberinto y llegar a la meta sin perder todas las vidas ni caer en un agujero.

Figura 4.3 Pantalla de inicio del juego

4.2 Análisis de Requisitos

Tras realizar la descripción general del funcionamiento del juego, a continuación se incluyen los requisitos identificados a partir del uso planteado y que deben cumplirse al final del desarrollo.

4.2.1 Requisitos Funcionales

Id	Descripción del Requisito
RF01	El usuario deberá poder elegir jugar tres distintos niveles con dificultad creciente.
RF02	El usuario deberá poder leer las instrucciones del juego antes de empezar la partida.
RF03	El usuario deberá poder salir de la aplicación en cualquier momento.
RF04	El usuario deberá poder pausar la partida en cualquier momento.
RF05	El usuario deberá poder reanudar siempre una partida en pausa
RF06	El usuario deberá poder cambiar de nivel en cualquier momento durante el transcurso de una partida.
RF07	El usuario deberá poder reiniciar el nivel en el que esté jugando en cualquier

	momento.
RF08	El usuario deberá poder comprobar la situación del ránking online al final de cada partida.
RF09	El usuario deberá poder ver en todo momento durante una partida el número de vidas restantes.
RF10	El usuario deberá poder ver en todo momento durante una partida el tiempo que lleva en dicho nivel.
RF11	La aplicación deberá habilitar un recuadro para que el usuario escriba su nombre al final de una partida victoriosa y deberá agregar dicho usuario junto con el tiempo realizado, a la base da datos del ránking.

Figura 4.4 Tabla de requisitos funcionales de la aplicación

4.2.2 Requisitos No Funcionales

Id	Descripción del Requisito	Categoría
RNF01	El sistema deberá garantizar que el nombre de usuario entrante en la base de datos no supere los 10 caracteres de longitud.	Seguridad, Escalabilidad, Rendimiento.
RNF02	El sistema garantizará que el ránking online es veraz y está actualizado siempre a su última versión al comienzo de cada partida.	Rendimiento, Eficiencia.
RNF03	La aplicación deberá impedir que la pantalla del dispositivo se oscurezca durante la partida aunque el usuario no la toque.	Seguridad, Rendimiento.
RNF04	El sistema garantizará la óptima fluidez del juego en todo momento y la máxima precisión de la lectura de los sensores (siempre y cuando disponga del *hardware* necesario).	Rendimiento.
RNF05	El juego podrá ser jugado en cualquier dispositivo móvil siempre y cuando disponga del *hardware* necesario.	Accesibilidad, Portabilidad.
RNF06	El juego podrá ser jugado en cualquier sistema operativo móvil.	Accesibilidad, Portabilidad.
RNF07	La aplicación garantizará que el movimiento del huevo sea fiel a la realidad.	Rendimiento.
RNF08	La aplicación garantizará que la dureza y fragilidad del huevo	Rendimiento.

	al chocarse contra la pared o los obstáculos sea fiel a la realidad.	
RNF09	La aplicación garantizará que los gráficos del juego se vean correctamente en cualquier dispositivo que disponga del *hardware* necesario.	Rendimiento, Portabilidad.

Figura 4.5 Tabla de requisitos no funcionales de la aplicación

4.3 METODOLOGÍA

En todo proyecto *software* suele existir una metodología de trabajo inherente al mismo en la que se definen las tareas a realizar y las etapas de las que consta el proceso de desarrollo y en las cuales se llevan a cabo dichas tareas. El objetivo de fijar una metodología de este tipo es mejorar la productividad durante el desarrollo del *software* y, por consiguiente, la calidad final del producto.

En este proyecto concreto, la metodología utilizada ha sido la clásica en el desarrollo de videojuegos: la incremental o iterativa. En cualquier desarrollo de cualquier videojuego, al igual que en muchos otros tipos de proyecto *software*, se empieza con una versión muy primaria de la aplicación (en la primera iteración), a la que se le van añadiendo funcionalidades nuevas y/o mejorando apartados ya existentes en cada una de las iteraciones posteriores, consiguiendo así diferentes versiones del juego final con cada nueva iteración.

Tras cada versión, ésta es evaluada, probada y estudiada, y se anotan posibles mejoras o actualizaciones que el juego pudiera implementar en sus siguientes versiones, de forma que el producto final cumpla tanto con todos los objetivos y requerimientos que se plantearon al comienzo del desarrollo, como con aquellos que han sido generándose como consecuencia de la evaluación de alguna versión intermedia.

En la Figura 4.6 se puede observar un esquema gráfico de la metodología utilizada.

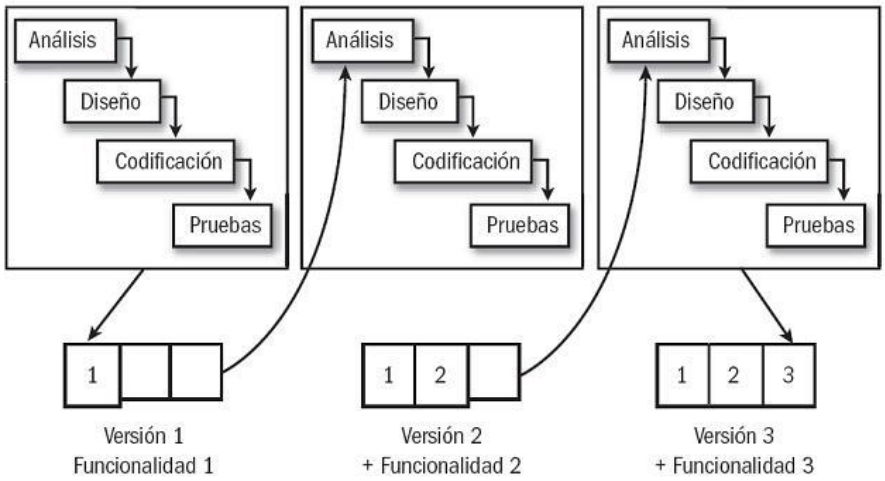

Figura 4.6 Representación gráfica del tipo de metodología iterativa o incremental

Utilizando la metodología descrita, durante este proceso de desarrollo se han realizado las siguientes iteraciones:

1. Diseño primario de la interfaz y de los niveles.
2. Creación de la aplicación y de la interfaz y menús del juego.
3. Creación de los niveles y del objeto del huevo.
4. Programación del comportamiento del movimiento huevo y reacción a los sensores.
5. Creación de la base de datos online para la realización del ránking.
6. Programación de la parte de comunicación entre el juego y la base de datos.
7. Retoques finales de interfaz y gráficos.

Como se ha mencionado anteriormente, en cada una de las iteraciones también se repasaba cada una de las anteriores y se iban haciendo cambios y modificaciones desde la primera hasta la última, que afectaban a las superiores.

4.4 Arquitectura de la Aplicación

El diseño arquitectónico de un proyecto *software* representa el nivel más alto de abstracción en la estructura del sistema. El diseño escogido es el modelo cliente-servidor en el que existe una comunicación entre un programa que actúa de servidor y se dedica a cumplir con las peticiones que recibe del cliente: una aplicación móvil de usuario.

De esta forma se puede diferenciar entre el diseño del propio servidor encargado de realizar dichas peticiones y el diseño de la aplicación móvil que envía las peticiones al servidor.

4.4.1 Servidor

El servidor implementa internamente una arquitectura de tres capas que son: capa de presentación de datos, capa de negocio y capa de acceso a datos. Este diseño genera una separación lógica entre las diferentes capas de tal forma que cada una es independiente de las demás, lo que permite realizar cambios internos en cualquiera de ellas sin afectar a las otras dos.

Capa de presentación de datos: esta capa se comunica únicamente con la capa de negocio, y está formada por todos aquellos elementos que conforman la interfaz de usuario, ya que ésta es la capa que ve el usuario y que se encarga de representar el sistema al mismo. También realiza la captura de la información que el usuario pueda introducir. En este trabajo, esta capa consistiría en la propia interfaz gráfica de la app móvil del juego.

Capa de negocio: en esta capa se realizan la mayoría y las más importantes operaciones con los datos. Es la capa encargada de comunicarse tanto con la capa de presentación como con la de acceso a datos. Su función consiste tanto en comunicarse con esta última para facilitar u obtener toda la información necesaria que se precise a través del gestor de la base de datos (en este caso: número de nivel, nombre de usuario y tiempo tardado), como en hacerlo con la primera para recibir y gestionar las peticiones que se invoquen desde la misma, así como para devolver las posibles respuestas a dichas peticiones.

Capa de acceso a datos: aquí se representa el manejo y almacenamiento de los datos de la base de datos. Esta capa está compuesta de un sistema gestor de base de datos, en nuestro caso MySQL, que contiene información precisa y estática. Sólo se comunica con la capa de negocio, al igual que la de presentación.

En la Figura 4.7 se puede comprobar cómo están organizadas estas tres capas en este tipo de arquitectura mediante un esquema.

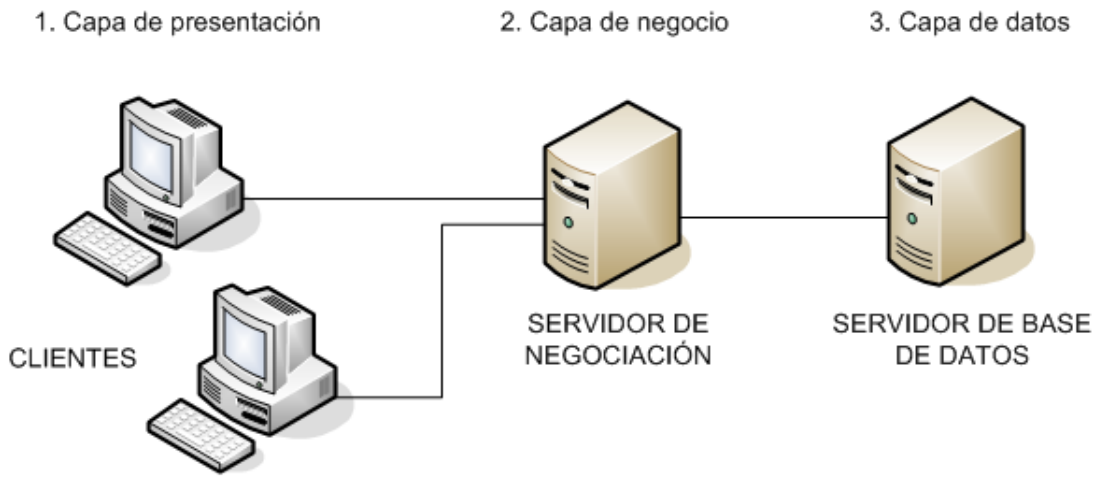

Figura 4.7 Representación gráfica de la arquitectura de tres capas

4.4.2 Aplicación Móvil

La aplicación móvil está diseñada basándose en el clásico patrón de arquitectura *software* Modelo-Vista-Controlador, que separa los datos y la lógica de negocio de la aplicación, de la interfaz de usuario y el módulo encargado de gestionar los eventos y las comunicaciones. Así, define por separado los componentes para la representación de la información y los componentes para la interacción con el usuario.

Modelo: se trata de la representación de la información y los datos con los que opera la aplicación y gestiona todos los accesos a dichos datos; es decir: las peticiones de acceso o manipulación que vienen del controlador. Además, envía a la vista aquella información que ésta le solicita y que ha de ser mostrada al usuario.

En este apartado estarían incluidas todas las clases y que se han implementado y que constituyen el conglomerado del trabajo.

Vista: presenta la información que recoge del modelo en un formato adecuado para el usuario a través de una interfaz gráfica. Los eventos que se generan en la vista y que responden a la interacción del usuario con la interfaz gráfica son manejados por el controlador.

La vista de este proyecto software está compuesta por las diferentes pantallas del juego que se han ido creando y que forman parte de la interfaz gráfica.

Controlador: como se acaba de decir, el controlador se encarga de responder a los eventos generados por el usuario a través de la vista, e invoca peticiones al modelo cuando se realizan solicitudes de información o actualizaciones. Su función podría resumirse diciendo que actúa de intermediario entre la vista y el modelo.

En este caso, el controlador forma parte del propio motor de Unity que se encarga, por ejemplo, de controlar las físicas del movimiento y los choques.

Tanto la implementación de las clases que forman el modelo, como la creación de las diferentes pantallas mencionadas en la vista, como la explicación de esta última parte del controlador, se tratarán en más profundidad en el siguiente capítulo.

En la Figura 4.8 se puede ver un esquema gráfico del funcionamiento de este patrón modelo-vista-controlador y de las relaciones entre sus tres componentes.

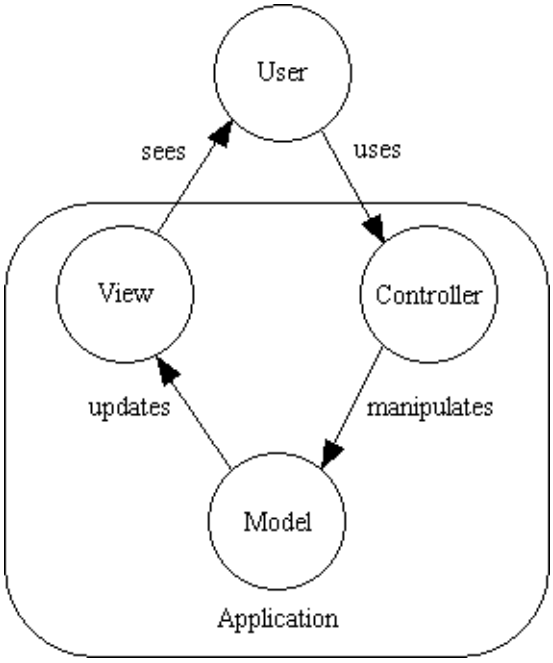

Figura 4.8 Representación gráfica del patrón modelo-vista-controlador

5 DISEÑO, DESARROLLO Y PROGRAMACIÓN DEL VIDEOJUEGO

En este capítulo se describe cómo se ha ido construyendo el videojuego diferenciando las cuatro partes fundamentales del mismo: los escenarios, el huevo, la interfaz gráfica y la parte de comunicación con la base de datos.

5.1 ESCENARIOS

Los escenarios se corresponden con cada uno de los niveles o laberintos diseñados. La primera tarea realizada en este proyecto, y que sirvió para familiarizarse con la plataforma Unity, fue la construcción del conjunto de laberintos por los que circulará el huevo.

En Unity, todos los objetos del escenario engloban en lo que se denomina `GameObject`, la clase básica para todas las entidades de las escenas [22], y el primero que se creó para comenzar a modelar la superficie fue el terreno. A éste se le dio el color verde para simular la hierba, con una textura de superficie lisa para el buen deslizamiento del huevo. También se añadió una cúpula celeste como otro `GameObject` en el que se integró el lightmapper Beast [23] que ofrece Unity [24], para crear unas luces naturales muy logradas.

En Unity las texturas reciben el nombre de materiales y se pueden crear de diferentes formas. Unity proporciona texturas básicas que simulan una serie de materiales (por ejemplo, tierra, hierba o madera), pero también permite importarlas automáticamente desde un paquete de modelado 3D o crear nuevas a partir de una imagen. Esta última es la opción empleada para crear la textura de ladrillos, así como otras texturas del juego que se verán más adelante. Además, Unity ofrece una Tienda de Activos [25] en la que se pueden comprar cientos de texturas y materiales además de multitud de objetos 3D ya diseñados.

El segundo paso en la creación del escenario fue la inclusión de las paredes de ladrillo que delimitan el recorrido o circuito del laberinto. Estas paredes se crearon deformando un

`GameObject` con forma de cubo, a las que se les dio también una textura de ladrillo (Figura 5.1).

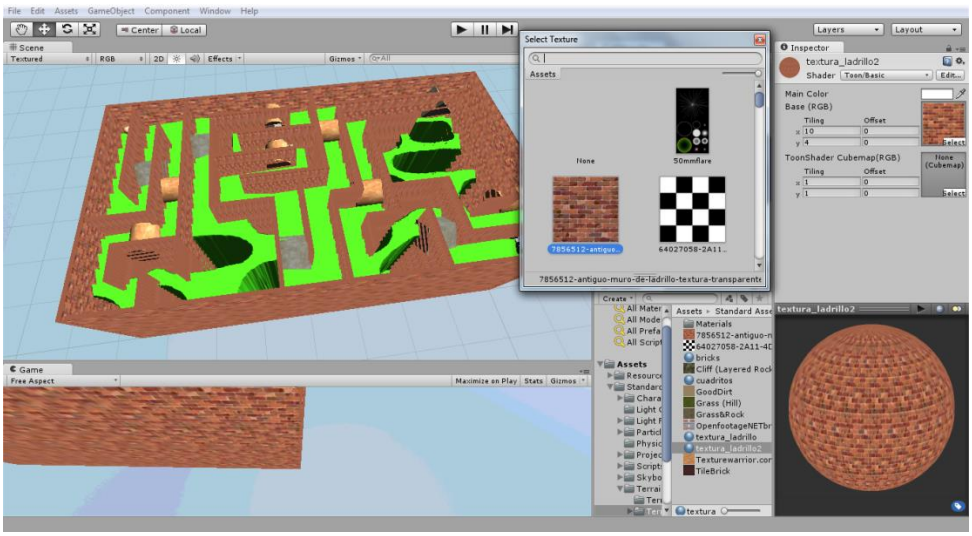

Figura 5.1 Captura de pantalla de Unity, seleccionando la textura de las paredes de ladrillo

Después de crear las paredes y con el circuito del laberinto definido, se crearon una serie de agujeros al terreno con una herramienta del editor de terrenos de Unity con la que se pueden generar elevaciones o depresiones del mismo, además de muchas otras transformaciones.

A continuación se añadieron los obstáculos y la meta. Como se había diseñado desde un primer momento, en este juego habría dos tipos de obstáculos a parte de las propias paredes del laberinto y los agujeros. Éstos serían unos badenes de dos tipos: uno más redondeado, que se asemejara a un puente de madera, y otro más piramidal con textura de asfalto. Ambos se crearon deformando dos `GameObject`, uno con forma de cilindro (Figura 5.2), y el otro con forma cúbica. Las texturas se realizaron de la misma forma que la de los ladrillos que se ha explicado anteriormente.

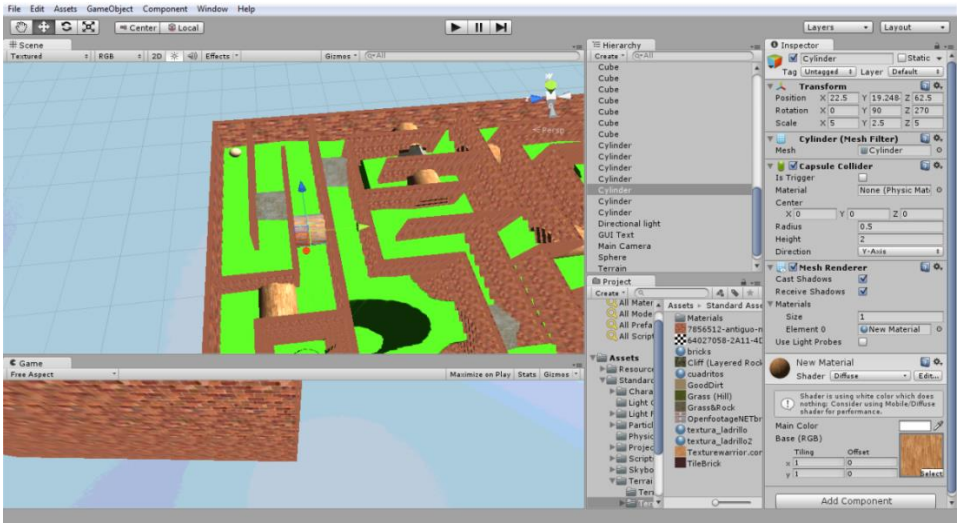

Figura 5.2 Captura de pantalla de Unity, instalando uno de los obstáculos con forma cilíndrica

Para finalizar, se realizó la meta, nuevamente a partir de un `GameObject` de forma cúbica con una textura de damero clásica. Sin embargo, lo importante de este `GameObject` es que incluye un manejador por el cual ante una colisión del huevo se da por terminada la partida de forma ganadora; es decir: el tiempo se detiene y se guarda para poder mandarlo a la base de datos, en caso de que el usuario lo demande, tras lo cual aparece la pantalla del final del nivel junto con el audio de enhorabuena.

En la siguiente Figura 5.3 se puede ver un recuadro rojo que señala la adición de la fuente de audio al `GameObject Meta` y el *script* `SonidoTerrain2.js`, que gestiona todo lo mencionado en el párrafo anterior.

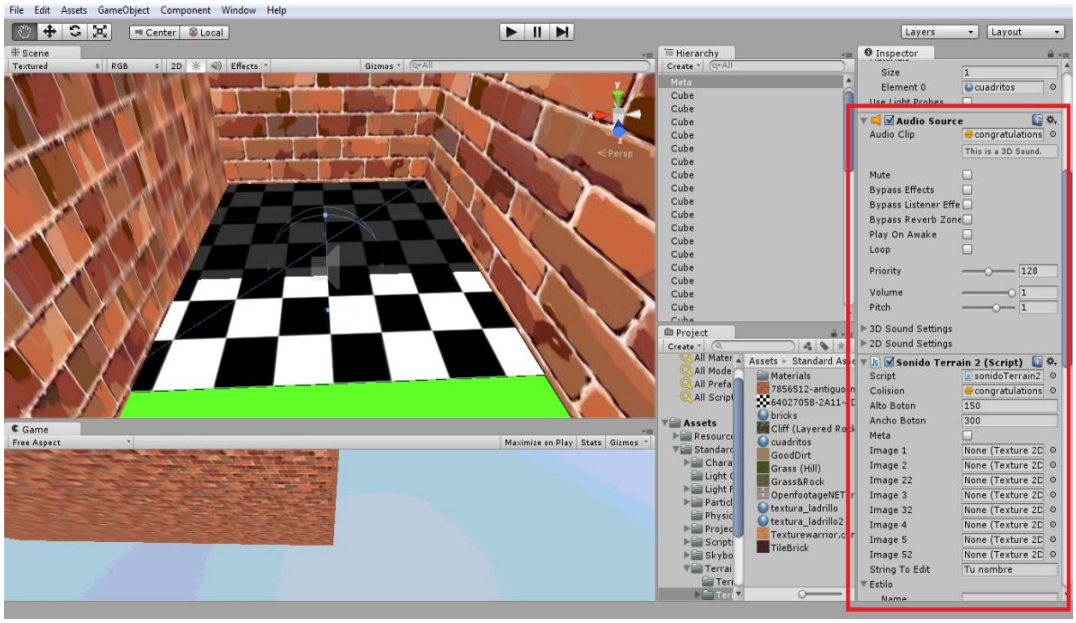

Figura 5.3 Captura de pantalla de Unity, creando la meta

Es importante reseñar que antes de proceder con la implementación del laberinto, se hizo un diseño preliminar incluido en la Figura 5.4 con el que apoyarse a modo de *storyboard* a la hora a de definir posteriormente el escenario de cada nivel del laberinto.

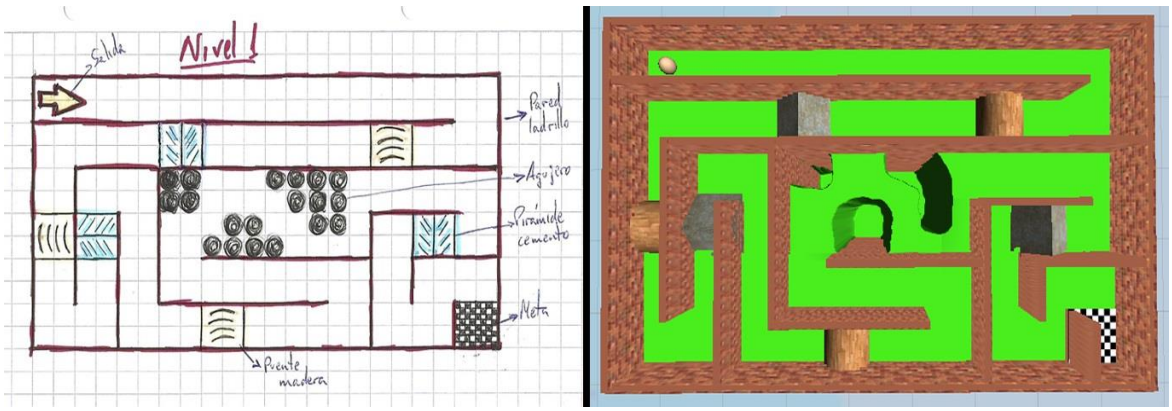

Figura 5.4 Imagen escaneada del diseño preliminar en papel del nivel 1 a la izquierda, y modelo final en 3D del mismo nivel en Unity, a la derecha

Es importante señalar también que todos los `GameObject` tienen este tipo de comportamiento que responde a las colisiones con el huevo. Seguidamente se profundiza en el modelado del huevo y las colisiones.

```
var colision:AudioClip;
var altoBoton : int = 150;
var anchoBoton : int = 300;
var meta : boolean = false;

var image1 : Texture2D;
var image2 : Texture2D;
var image22 : Texture2D;
var image3 : Texture2D;
var image32 : Texture2D;
var image4 : Texture2D;
var image5 : Texture2D;
var image52 : Texture2D;
var stringToEdit : String = "Tu nombre";
var estilo : GUIStyle;
var tiempos : ControladorTiempos;
var tiempos2 : ControladorTiempos;

function Start()
{
    image1 = Resources.Load("congratulations");
    image2 = Resources.Load("CambiarNivel");
    image22 = Resources.Load("CambiarNivelActivo");
    image3 = Resources.Load("RepetirNivel");
    image32 = Resources.Load("RepetirNivelActivo");
    image4 = Resources.Load("gana");
    image5 = Resources.Load("leaders");
    image52 = Resources.Load("leadersActivo");
    estilo.fontSize = 80;
}


function OnCollisionEnter(collision:Collision)
{
    if (collision.collider.material.staticFriction == 0.25f) {
        audio.PlayOneShot(colision);
        Time.timeScale=0.0;
        meta=true;


    }else{
        audio.PlayOneShot(colision);
        Time.timeScale=0.0;
        meta=true;
        }
}

function OnGUI()
{
    if(meta)
    {
        stringToEdit = GUI.TextField (Rect (10, 110, 400, 100), stringToEdit, 10, estilo);
        if (GUI.changed == true)
        {
            Variables.nombre = stringToEdit;
            tiempos.postScore(Variables.nombre, Variables.mins, Variables.secs, Variables.cents);
            if (Variables.error)
            {
                stringToEdit = "Error";
            }
            else
            {
                stringToEdit = "Hecho";
            }
        }

        GUI.skin.button.normal.background = image4;
        GUI.skin.button.hover.background = image4;
        GUI.skin.button.active.background = image4;
        if (GUI.Button(Rect(500,300,900,550), ""){}

        GUI.skin.button.normal.background = image1;
        GUI.skin.button.hover.background = image1;
        GUI.skin.button.active.background = image1;
        if (GUI.Button(Rect(500,20,900,400), ""){}

        GUI.skin.button.normal.background = image2;
        GUI.skin.button.hover.background = image2;
        GUI.skin.button.active.background = image22;
        if (GUI.Button(Rect(600,860,anchoBoton,altoBoton), ""))
        {
            Application.LoadLevel("MenuNivel");
            Time.timeScale=1.0;
        }

        GUI.skin.button.normal.background = image3;
        GUI.skin.button.hover.background = image3;
        GUI.skin.button.active.background = image32;
        if (GUI.Button(Rect(1000,860,anchoBoton,altoBoton), ""))
        {
            Application.LoadLevel("nivel0");
            Time.timeScale=1.0;
        }

        GUI.skin.button.normal.background = image5;
        GUI.skin.button.hover.background = image5;
        GUI.skin.button.active.background = image52;

        if (GUI.Button(Rect(1700,110,180,180), ""))
        {
            Time.timeScale = 1.0;
            tiempos2.getScores();
            Application.LoadLevel("LeaderboardN0");
        }
    }
}
```

Figura 5.5 Código en JavaScript del *script* SonidoTerrain.js, similar al Sonido Terrain 2

Como se puede observar en el código de la Figura 5.5, la meta reacciona a la colisión con el huevo haciendo sonar su fuente de audio y desplegando el menú de final de la partida, realizando todas las tareas tratadas más arriba. En la ésta figura aparece el *script* `SonidoTerrain.js`, similar al `SonidoTerran2.js` mencionado anteriormente. Cada uno de estos *scripts* está relacionado con la meta de cada nivel (1, 2 y 3).

5.2 El Huevo y su Movimiento

El huevo, o Humpty Dumpty, es el `GameObject` fundamental del juego, ya que es el propio protagonista y el personaje que maneja el jugador. En él están presentes todas las físicas de colisiones, movimiento, etc. y es él el que dispara todos los acontecimientos, tales como perder vidas, morir o llegar a la meta.

El objeto en sí es un `GameObject` en forma de esfera que se ha deformado para darle forma de huevo, y al que se le ha aplicado una textura que se asemeja a la superficie de un huevo real, creada de la misma forma que las vistas anteriormente en este mismo capítulo.

A partir de ahí se le han ido añadiendo *scripts*, físicas de Unity y diferentes configuraciones, que lo han dotado de todas las funciones que realiza. En primer lugar, al igual que a los obstáculos, se le añadió el `collider` [26], para garantizar que el huevo se chocase contra ellos en vez de atravesarlos.

Tras esto se implementó lo que se puede considerar la parte más importante y compleja del juego: modelar el desplazamiento del huevo y lograr que su movimiento se parezca lo más posible al de uno verdadero, en respuesta a los sensores de los acelerómetros de los dispositivos móviles.

Para ello, se dotó al objeto de un `rigidbody` de Unity, componente encargado de dotar de movimiento a un objeto [27], logrando así que el huevo girara como una esfera perfecta. Seguidamente se le ajustó una malla a la superficie deformada del objeto para conseguir que el huevo se moviera como uno real. Esto se consiguió recurriendo a la inclusión de un `collider` en el huevo al que se le ajustaron los atributos de 'x', 'y' y 'z', de forma que la malla se ajustara a los mismos atributos del objeto (Figura 5.6).

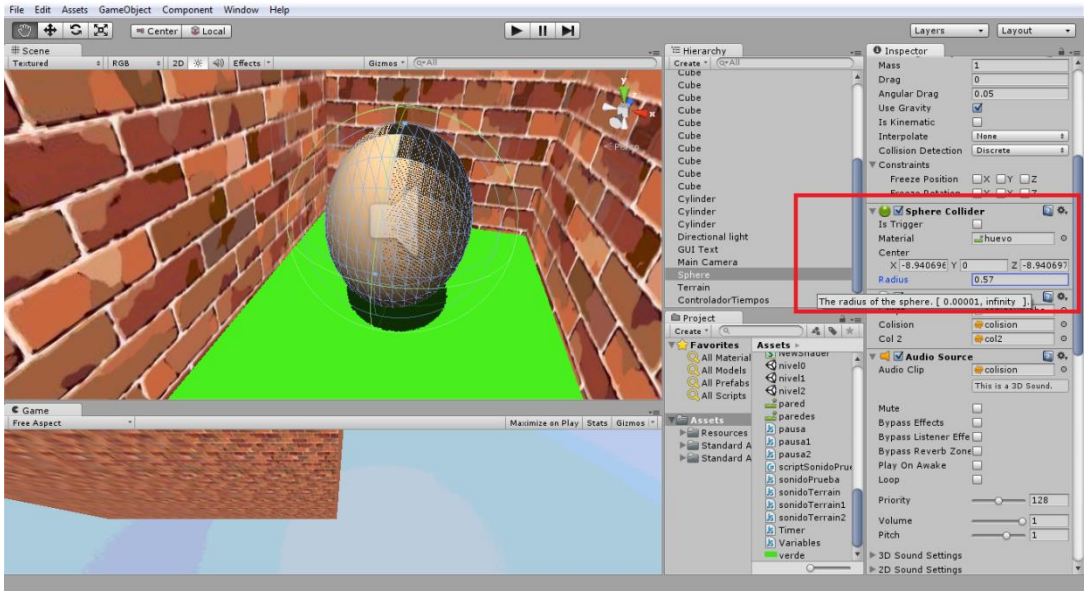

Figura 5.6 Captura de pantalla de Unity, ajustando la malla al huevo (recuadro rojo)

Tras esto se desarrolló el programa que emula el movimiento real en respuesta a los desplazamientos obtenidos de los sensores. De esta forma se determina la dirección, el sentido y la velocidad y aceleración del huevo respecto a las coordenadas 'x', 'y' y 'z' leídas del dispositivo móvil. Además de esto, también se añadió el sonido básico de colisión para los efectos de rebote en las superficies y el sonido de colisión fuerte, aplicado ante la rotura de la cáscara del huevo. Los sonidos empleados, descargados de librería de sonidos con licencia

gratuita, se ajustaron en duración y calidad empleando la herramienta Audacity, tras lo cual se añadieron al juego en Unity mediante su herramienta de gestión de audio FMOD.

Una vez terminada la programación del movimiento del huevo (Figura 5.7), se hizo necesario crear otro *script* de movimiento, pero esta vez para la cámara del juego. La cámara es otro `GameObject` llamado `Main Camera`, que tiene asociado el *script* `ControlCamara.cs` que se ocupa de que el jugador vea, en todo el momento, al huevo en el centro de la pantalla (Figura 5.8).

Finalizadas las fases de diseño, creación del huevo, gestión de su movimiento a partir de los sensores del terminal móvil y la dureza y respuesta antes colisiones contra los obstáculos se implementaron las caídas por los agujeros y se realizó una primera prueba de usabilidad con unos resultados satisfactorios.

```
1 using UnityEngine;
2 using System.Collections;
3
4 public class Move : MonoBehaviour {
5
6     public float speed;
7     public AudioClip gameover;
8
9     // Update is called once per frame
10    void FixedUpdate ()
11    {
12        float movehorizontal = -Input.acceleration.x;
13        float moveVertical = Input.acceleration.y;
14
15        Vector3 movimiento = new Vector3 (moveVertical, 0.0f, movehorizontal);
16
17        rigidbody.AddForce (movimiento * speed * 3 * Time.deltaTime);
18    }
19
20    void OnCollisionEnter(Collision obj)
21    {
22        GameObject vida = GameObject.Find ("CorazonesVida");
23        if (obj.relativeVelocity.magnitude > 15)
24        {
25            vida.SendMessage("reducirVida");
26        }
27    }
28 }
```

Figura 5.7 Código del *script* que da al huevo su movimiento, escrito en C#

Es importante apreciar en este código cómo se pasa al `GameObject vida` el mensaje de `reducirVida`. Este mensaje se envía cuando se produce una colisión entre el huevo y otro objeto y la velocidad supera la magnitud 15. Más adelante se tratará la parte de `CorazonesVida`.

```
1 using UnityEngine;
2 using System.Collections;
3
4 public class ControlCamara : MonoBehaviour {
5
6     public GameObject huevo;
7     private Vector3 offset;
8     // Use this for initialization
9     void Start () {
10
11         offset = transform.position;
12
13     }
14
15     // Update is called once per frame
16     void LateUpdate () {
17
18         transform.position = huevo.transform.position + offset;
19     }
20 }
```

Figura 5.8 Código del *script* que da movimiento a la camara, escrito en C#

Para desarrollar los programas que se encargan de las vidas del huevo y del fin de la partida se tuvo en cuenta la velocidad con la que el huevo chocaba contra los obstáculos. Así se creó un `GameObject` abstracto relacionado de forma directa con el `GameObject` del huevo, que contabilizaría la vida del mismo con un valor de 0 a 200. De esta forma, cada vez que el huevo se chocaba a más velocidad que la permitida, la vida se reducía un 33,33%. La idea de realizar así el objeto de las vidas está relacionada directamente con el punto siguiente: la interfaz gráfica.

```
1 using UnityEngine;
2 using System.Collections;
3
4 public class CorazonesVida : MonoBehaviour
5 {
6     public GUIStyle corazonesVida;
7     public Texture2D corazonesFondo;
8     public Texture2D corazonesFrente;
9
10    public float energia = 200f;
11
12    void OnGUI ()
13    {
14        GUI.BeginGroup (new Rect (10, 10, 200, 64));
15        GUI.Box (new Rect(0, 0, 200, 64), corazonesFondo, corazonesVida);
16
17        GUI.BeginGroup (new Rect (0, 0, energia, 64));
18        GUI.Box (new Rect(0, 0, 200, 64), corazonesFrente, corazonesVida);
19
20        GUI.EndGroup();
21        GUI.EndGroup();
22    }
23
24    void reducirVida()
25    {
26        energia -= 66.66f;
27    }
28
29    public bool muerto()
30    {
31        return (energia < 1f);
32    }
33
34    public void restart()
35    {
36        energia = 200f;
37    }
38 }
```

Figura 5.9 Código referente al *script* CorazonesVida.cs, parte que se encarga de contabilizar las vidas del huevo, escrito en C#

Como se puede apreciar en el código de la Figura 5.9, el mensaje que este `GameObject` recibe del huevo cuando éste colisiona a velocidad alta visto anteriormente, se trata de una llamada a una función que realiza esta clase para restar una vida. Se podrá ver el significado que tienen los atributos `corazonesFondo` y `corazonesFrente` en el siguiente capítulo.

5.3 La Interfaz Gráfica

La interfaz gráfica de gestión del juego ha sido el tercer gran bloque desarrollado. Consiste en una serie de pantallas en las que, por medio de botones diseñados con Adobe Photoshop, se accede a diferentes menús de opciones y a los propios niveles del juego.

Además, durante la propia partida se ha superpuesto al escenario de juego una interfaz que indica las vidas que tiene el huevo en un momento dado, el tiempo que lleva jugando el nivel en concreto y un botón de pausa. El botón de pausa genera una nueva interfaz, por encima del juego, con diferentes botones para reanudar la partida en curso, reiniciar el nivel en cuestión o salir del nivel (Figura 5.10).

Figura 5.10 Captura de pantalla del juego en pausa

Para acabar, también se han creado dos pantallas de final del juego, una ganadora (si el huevo ha llegado a la meta con al menos una vida) y otra perdedora (si el huevo ha perdido sus tres vidas o si se ha caído por un agujero).

Cada uno de los menús externos de la partida se define como un nivel de Unity y está enlazado con los otros mediante los botones. En el gráfico de la Figura 5.11 se puede apreciar el flujo de acción entre las diferentes pantallas (niveles) del juego. De este modo se puede ver cómo, desde la pantalla de inicio, el usuario puede acceder a la pantalla de instrucciones del juego o a la de selección de nivel. También se puede comprobar cómo una vez en la pantalla del juego éste se puede poner en pausa e incluso salir del nivel o reiniciarlo, antes de terminar la partida -bien con una derrota o una victoria-. Momento en el que el usuario puede acceder a la pantalla del ránking.

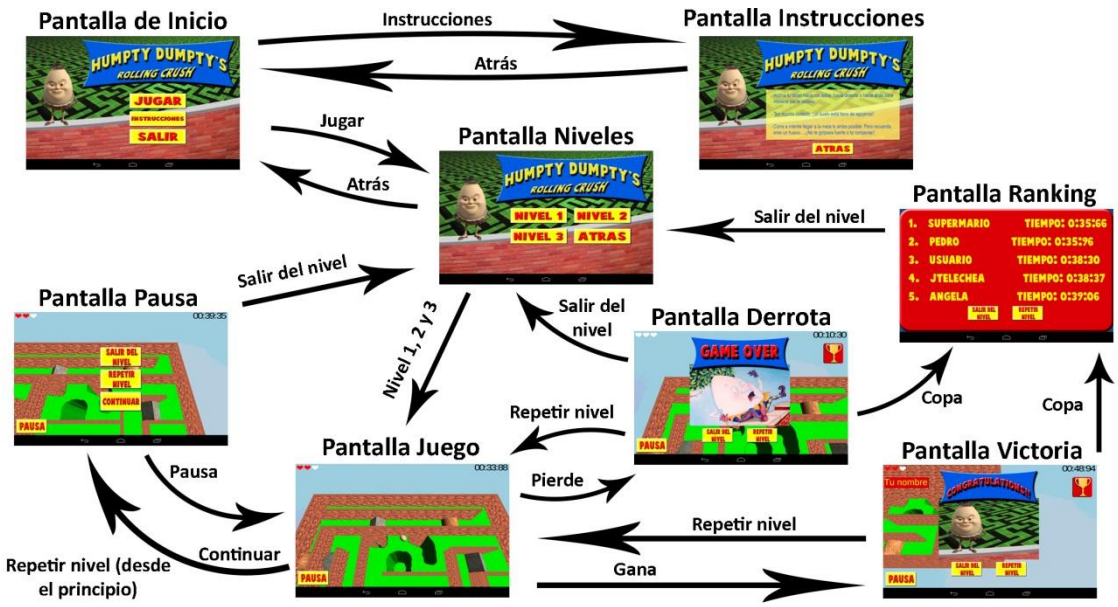

Figura 5.11 Esquema gráfico del flujo entre pantallas

El juego ha sido diseñado e implementado para concebir una interfaz sencilla e intuitiva, que permita al usuario familiarizarse con ella rápida y eficazmente. La clase que Unity facilita para el posicionamiento manual en la interfaz de objetos como botones, imágenes, etc. recibe el nombre de `GUI` (*Graphical User Interface*) [28], y es la que se ha usado para la creación de los diferentes menús.

Como se ha explicado en el punto anterior, el *scripting* de las funciones principales del juego, como el manejo del tiempo y el de las vidas por ejemplo, está, obviamente, muy relacionado con la parte de la interfaz gráfica.

El objeto que se encarga de contabilizar las vidas del huevo fue creado siguiendo el patrón fijado durante la fase de diseño de forma que pudiera representarse como tres corazones. Por este motivo se programó un número del 0 al 200 que se reduce un 33,33% cada vez que el huevo choca fuertemente contra un obstáculo. En la interfaz, existen tres corazones blancos (`corazonesFondo`) y otros tres corazones rojos, que se encuentran por encima de los primeros, pero por debajo del escenario (`corazonesFrente`) (Figura 5.12). Éstos últimos se conciben como una barra que mide de 0 a 200. Así, a la vez que la vida numérica se reduce, estos corazones lo hacen a la vez, dando la visión al jugador de que lo que desaparece es un corazón.

Figura 5.12 Esquema de las dos capas que forman los corazones que indican las vidas del huevo en el juego

El texto del tiempo se ha realizado usando un objeto `GUIText` de Unity (Figura 5.13), que va unido a un *script* que maneja el tiempo (Figura 5.14) implementado usando la interfaz `Time` de Unity [29].

Figura 5.13 Captura de pantalla del juego que muestra el cronómetro de la interfaz

```
#pragma strict

private var startTime : float;
var textTime : String;

function Awake ()
{
    startTime = Time.time;
}

function OnGUI ()
{
    var guiTime = Time.time - startTime;
    var minutes : int = guiTime/60;
    var seconds : int = guiTime%60;
    var fraction : int = (guiTime*100)%100;

    textTime = String.Format("{0:00}:{1:00}:{2:00}", minutes, seconds, fraction);
    GetComponent(GUIText).text = textTime;

    Variables.mins = minutes.ToString();
    if (seconds < 10)
    {
        Variables.secs = "0"+seconds.ToString();
    }
    else
    {
        Variables.secs = seconds.ToString();
    }
    if (fraction < 10)
    {
        Variables.cents = "0"+fraction.ToString();
    }
    else
    {
        Variables.cents = fraction.ToString();
    }
}
```

Figura 5.14 Código referente al *script* Timer.js que se encarga del control del tiempo, escrito en JavaScript

Por último, también son menús de la interfaz gráfica los encargados de mostrar el ránking online. A estos menús, que al igual que los demás son niveles de Unity, se accede mediante un botón que muestra una copa y que aparece en la esquina superior derecha de la pantalla, debajo del tiempo, en las pantallas de final de la partida.

Además, en la pantalla de final ganador también aparece, a la misma altura que el botón de la copa pero al lado izquierdo, un recuadro para escribir el nombre de usuario para ser guardado en la base de datos. En la Figura 5.15 se puede ver la colocación de ambos elementos, con las flechas (que no aparecen en el juego) indicando su posición.

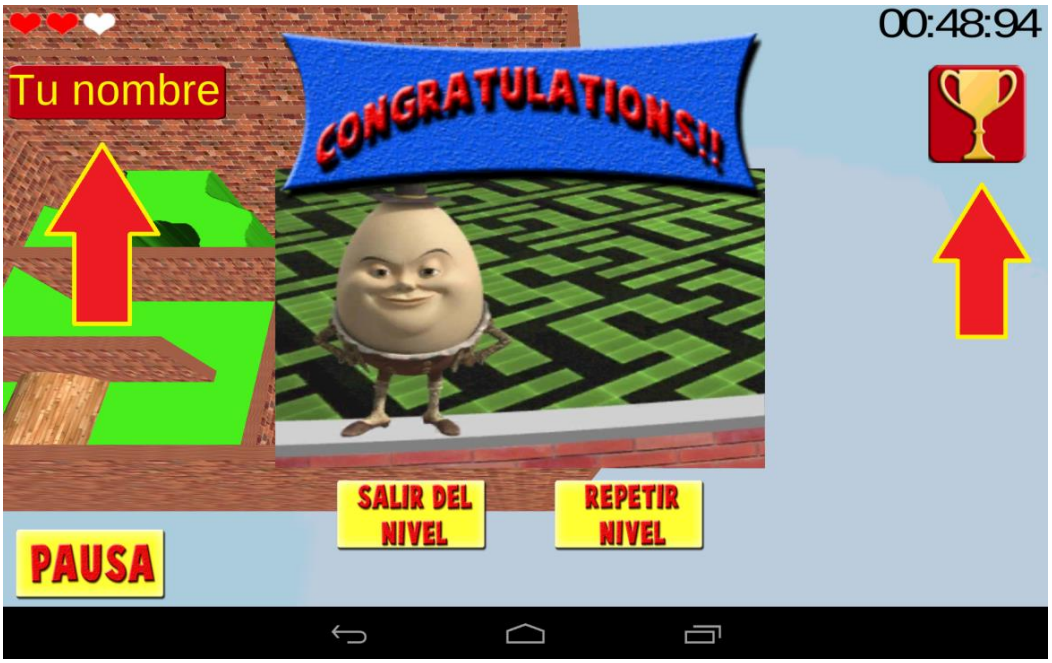

Figura 5.15 Captura de pantalla del final de la partida en victoria

Todo esto ha sido implementado usando la clase `GUI` de Unity, al igual que lo descrito anteriormente. La programación de la parte de comunicación con la base de datos y el ránking se explicará en el siguiente apartado.

5.4 La Comunicación con la Base de Datos y el Ránking Online

Este ha sido el último bloque en ser definido en el juego. Consta en sí de dos partes diferenciadas dentro de su propio desarrollo: la propia creación de la base de datos, y la programación de la comunicación del juego con la misma a través del servidor local.

5.4.1 La Creación de la Base de Datos

La base de datos ha sido construida en un sistema operativo GNU/Linux, Ubuntu, por su facilidad a la hora de la instalación de las aplicaciones y herramientas necesarias para el desarrollo de esta parte como son MySQL Server, Apache, etc.

Una vez instalado todo, se debe acceder a MySQL server a través de la terminal de Linux para crear la base de datos y las tres distintas tablas (una para cada nivel) con los campos: identificador, nombre de usuario, minutos, segundos y centésimas. En la Figura 5.16 se puede ver el resultado de una consulta a una de las tablas de la base de datos, la referente al nivel 1.

```
mysql> select * from times;
+----+------------+------+------+-------+
| id | name       | mins | secs | cents |
+----+------------+------+------+-------+
|  1 | javier     |    1 |   20 |    87 |
|  2 | angela     |    0 |   58 |    92 |
|  3 | pedro      |    1 |   32 |    14 |
|  4 | ciguas     |    2 |    0 |    25 |
|  5 | grillin    |    1 |   42 |     4 |
|  6 | c_trospido |    1 |   14 |    42 |
|  7 | lua        |    1 |   14 |    30 |
|  8 | txingurri  |    1 |   26 |    42 |
|  9 | jtelechea  |    0 |   42 |    37 |
| 10 | usuario    |    0 |   38 |    30 |
| 18 | pedro      |    0 |   35 |    96 |
| 12 | supermario |    0 |   35 |    66 |
| 19 | angela     |    0 |   39 |     6 |
| 17 | francisco  |    0 |   39 |    47 |
| 15 | gervasio   |    0 |   49 |    37 |
| 16 | jtelechea  |    0 |   38 |    37 |
| 20 | abelardo   |    1 |    2 |    22 |
+----+------------+------+------+-------+
```

Figura 5.16 Captura de la una consulta a una tabla de la base de datos

Este formato de tiempo en unidades separadas ha sido elegido debido a la facilidad de manejo de los datos tanto en MySQL, como en PHP, como en los *scripts* de Unity.

5.4.2 La Programación de la Comunicación

Tras crear la base de datos, el siguiente paso es definir los códigos en PHP que conectan con dicha base de datos a través del servidor MySQL, ya sea para leer los datos de una tabla concreta, dependiendo del nivel como se ha dicho anteriormente, o para añadir un nuevo elemento.

La configuración que se ha realizado aloja a estos programas PHP dentro del directorio "/var/www" para poder acceder a ellos mediante una URL a través del servidor local de Linux, a través de su IP (*Internet Protocol*) pública.

En la Figura 5.17 se puede ver el código referente al programa `addtime.php`, que recibe el nombre de usuario, el tiempo separado por unidades y el número del nivel al que pertenece dicho tiempo. El programa realiza la inserción de los datos dentro de la tabla correspondiente de la base de datos.

addtime.php

```php
<?php
        $db = mysql_connect('127.0.0.1', 'root', [redacted]) or die('Could not connect: ' . mysql_error());
        mysql_select_db(hdrc_tiempos) or die('Could not select database');

        // Strings must be escaped to prevent SQL injection attack.
        $level = mysql_real_escape_string($_GET[level], $db);
        $name = mysql_real_escape_string($_GET[name], $db);
        $mins = mysql_real_escape_string($_GET[mins], $db);
        $secs = mysql_real_escape_string($_GET[secs], $db);
        $cents = mysql_real_escape_string($_GET[cents], $db);

        // Send variables for the MySQL database class.
        $query = "insert into times$level values (NULL, '$name',$mins,$secs,$cents);";
        $result = mysql_query($query) or die('Query failed: ' . mysql_error());
?>
```

Figura 5.17 Código de addtime.php

En la Figura 5.18 se ve el código referente al programa `showtimes.php`. Este programa es el encargado de realizar la consulta a la tabla correspondiente y devolver el texto en el formato adecuado (se adecúa el tiempo de forma que siempre haya dos cifras en cada unidad: 01:20:08) de los cinco mejores tiempos de la tabla. Con este texto que contiene de forma enumerada, a los cinco nombres de usuario asociados a dichos cinco mejores tiempos, se forma el ránking que luego aparecerá en la pantalla del juego.

showtimes.php

```php
<?php
    // Send variables for the MySQL database class.
    $db = mysql_connect('127.0.0.1', 'root', [redacted]) or die('Could not connect: ' . mysql_error());
    mysql_select_db(hdrc_tiempos) or die('Could not select database');

    $level = mysql_real_escape_string($_GET[level], $db);

    $query = "select * from times$level order by mins, secs, cents limit 5";
    $result = mysql_query($query) or die('Query failed: ' . mysql_error());

    $num_results = mysql_num_rows($result);
    $i = 1;

    while ($row = mysql_fetch_array($result, MYSQL_ASSOC))
    {
        if (strlen($row[secs]) == 1 && strlen($row[cents]) == 1)
        {
                printf ("%d.   %-20stiempo: %d:0%d:0%d \n", $i, $row[name], $row[mins], $row[secs], $row[cents]);
        }
        else if (strlen($row[secs]) == 1)
        {
                printf ("%d.   %-20stiempo: %d:0%d:%d \n", $i, $row[name], $row[mins], $row[secs], $row[cents]);
        }
        else if (strlen($row[cents]) == 1)
        {
                printf ("%d.   %-20stiempo: %d:%d:0%d \n", $i, $row[name], $row[mins], $row[secs], $row[cents]);
        }
        else
        {
                printf ("%d.   %-20stiempo: %d:%d:%d \n", $i, $row[name], $row[mins], $row[secs], $row[cents]);
        }
        $i++;
    }
?>
```

Figura 5.18 Código de showtimes.php

De esta forma, la parte elaborada en Linux estaría terminada. A continuación, restaría la parte de Unity que conecta con el servidor local y recoge los datos del ránking que genera el programa PHP o recoge el nombre y el tiempo del usuario para añadirlo a la base de datos.

Con este fin se ha creado en Unity un `GameObject` abstracto que recibe el nombre de `ControladorTiempos`, y que tiene asociado un *script* que se encarga de realizar dicha comunicación. Este *script* tiene dos funciones: `postScore` y `getScores` (Figura 5.19). La primera recoge el nombre introducido por el usuario a través de la interfaz de la que se ha hablado anteriormente y el tiempo realizado en la partida en cuestión, y lo envía a la URL que enlaza con el programa `addtime.php`, que añade el elemento a la base de datos. La segunda es la encargada de acceder a la URL en la que el programa `showtimes.php` imprime el texto formateado leído de la base de datos y que contiene el ránking de los mejores tiempos, recoger dicho ránking, y pasárselo a la interfaz para poder mostrarlo por pantalla.

```
1 private var secretKey : String = "mySecretKey";
2 var addScoreUrl : String = "http://192.168.1.45/addtime.php?"; //be sure to add a ? to your url
3 var highscoreUrl : String = "http://192.168.1.45/showtimes.php";
4
5 function Start()
6 {
7     getScores();
8 }
9
10 function postScore(name : String, mins : String, secs : String, cents : String)
11 {
12     //This connects to a server side php script that will add the name and time to a MySQL DB.
13     var highscore_url = addScoreUrl + "name=" + WWW.EscapeURL(name) + "&mins=" + mins + "&secs=" + secs + "&cents=" + cents;
14
15     // Post the URL to the site and create a download object to get the result.
16     var hs_post : WWW = WWW(highscore_url);
17     yield hs_post; // Wait until the download is done
18     Variables.error = hs_post.error;
19     if(hs_post.error)
20     {
21         print("There was an error posting the high score: " + hs_post.error);
22     }
23 }
24
25 // Get the scores from the MySQL DB to display in a GUIText.
26 function getScores()
27 {
28     var hs_get : WWW = WWW(highscoreUrl);
29     yield hs_get;
30
31     if(hs_get.error)
32     {
33         print("There was an error getting the high score: " + hs_get.error);
34     }
35     else
36     {
37         Variables.texto = hs_get.text; // this is a GUIText that will display the scores in game.
38     }
39 }
```

Figura 5.19 Código referente al *script* ControladorTiempos.js, que representa la parte de Unity en la comunicación con el servidor de la base de datos

Estas funciones son llamadas desde la interfaz cuando el usuario ingresa su nombre o pulsa el botón de la copa que permite ver el ránking. En el momento en el que el jugador realiza esta última acción, la aplicación abre una nueva pantalla como se ha visto en la Figura 5.11 que muestra el ránking online. Este ránking se genera con el texto recibido a través de la función `getScores`.

Se hace evidente que para que la comunicación se realice correctamente, el dispositivo móvil debe estar en la misma red local que el servidor, y éste debe estar activado y en funcionamiento.

Durante todo el capítulo se ha podido observar en las figuras de los códigos, que existe una especie de clase llamada `Variables` que contiene unas variables estáticas que sirven para el paso de datos entre un *script* y otro (Figura 5.20). La solución de crear un *script* con variables estáticas con dicho fin, se ideó porque no existía otra manera de comunicar los diferentes *scripts* entre sí para que se pasaran, por ejemplo, los valores de minutos, segundos y centésimas entre el *script* `Timer.js` y el del `ControladorTiempos.js`, o el texto del ránking que recibe la función `getScores` y pasa a la interfaz que, en forma de `GUIText` lo muestra en pantalla. Esta solución fue encontrada dentro de los foros de la comunidad de Unity, como otras muchas soluciones a problemas que fueron surgiendo a lo largo del desarrollo del proyecto.

```
1 #pragma strict
2 static var nombre : String;
3 static var mins : String;
4 static var secs : String;
5 static var cents : String;
6 static var texto : String;
7 static var error : String;
```

Figura 5.20 Código referente al *script* Variables.js que contiene las variables estáticas para el paso de datos entre *scripts*

6 PRUEBAS

Durante el proceso de desarrollo de cualquier herramienta *software* se hace necesario añadir una fase de validación y pruebas que permita garantizar el correcto funcionamiento de la aplicación, así como su seguridad e integridad.

Como se ha mencionado en el capítulo 4, durante todo el proceso de desarrollo se ha utilizado la metodología incremental o iterativa, que obliga a realizar, tras cada iteración del trabajo, una serie de verificaciones y pruebas que sirvan para garantizar el correcto desarrollo de la aplicación.

El objetivo fundamental de estas pruebas es comprobar que todo lo implementado funciona correctamente en cada iteración. Así, se hace necesario probar tanto las nuevas actualizaciones realizadas en la última iteración, como todas las anteriores, y la interacción directa entre todas ellas.

De esta forma, y como es costumbre a la hora de realizar este tipo de pruebas, éstas se han realizado dividiéndose en cuatro tipos:

- Pruebas unitarias
- Pruebas de integración
- Pruebas de sistema
- Pruebas de aceptación

6.1 PRUEBAS UNITARIAS

Las pruebas unitarias se basan en la verificación independiente de cada módulo del sistema. En cada iteración se han ido probado las nuevas actualizaciones concretas que se han

ido añadiendo, ya sean parte de la interfaz, del movimiento o comportamiento del huevo, o de la parte del servidor y la base de datos.

En estas pruebas se ha hecho especial hincapié en los puntos críticos de cada apartado, realizando pruebas concretas buscando el error.

Así, se han hecho verificaciones de botones concretos, de la sensibilidad de los acelerómetros, de la adición de nuevas entradas a la base de datos, de la representación gráfica de elementos como el cronómetro o las vidas, de la representación del ránking, etc.

6.2 Pruebas de Integración

Con las pruebas de integración se pretende, una vez verificados los módulos unitarios por separado, ampliar el rango de alcance de las pruebas de forma que se compruebe el comportamiento de dichos módulos a la vez que se van integrando en el sistema con cada iteración.

Así, se ha ido probando que tras cada iteración, todo lo introducido en el sistema funcionaba correctamente. Estas pruebas han consistido en probar el correcto flujo entre diferentes pantallas y niveles de la aplicación móvil (Figura 5.11); la correcta consecución de los choques, la pérdida de vidas y el lanzamiento de las pantallas de *game over*; la correcta comunicación con el servidor y la base de datos desde la aplicación móvil, etc.

Como ejemplo, sirva la forma de comprobar cuál sería el valor más adecuado de velocidad con la que el huevo choca contra un obstáculo y se rompe, de forma que la sensación del jugador sea lo más real posible. Esto se realizó dando a la velocidad de choque diferentes magnitudes, primero en un rango entre 8 y 10, luego entre 11 y 14, y después entre 15 y 18. De esta forma se comprobó que la sensación más realista de choque y rotura del huevo se producía con un valor entre el segundo y el tercer rango de valores. Finalmente, se fijó el valor de la magnitud en 15.

6.3 Pruebas de Sistema

Estas pruebas, finalmente, son las que se encargan de verificar que todo el sistema funciona correctamente en su conjunto, con todos los módulos y funciones instaladas, como parte final de todas las iteraciones realizadas.

Durante esta última fase de pruebas se comprueba el cumplimiento de todos los requisitos planteados, tanto funcionales como no funcionales. Así, se verifican también aquellos requisitos más relacionados con aspectos de seguridad, rendimiento o accesibilidad.

De esta forma se han hecho pruebas portando el juego a diferentes dispositivos como PC (cambiando los controles del acelerómetro por el teclado), *smartphones* y *tablets* de

diferentes tamaños y *hardware*. De esta forma se ha podido comprobar que el juego funciona de manera correcta en todos ellos, por lo que no necesita *hardware* muy potente. Sin embargo, al cambiar de tamaño de pantalla entre dispositivos, se ha hecho necesario un reajuste de la resolución de diferentes aspectos de la interfaz.

6.4 Pruebas de Aceptación

Las pruebas de aceptación se conciben como la última parte del proceso de verificación, ya que se refieren a la validación del funcionamiento del sistema según los propios usuarios o jugadores, en este caso.

Debido al hecho de que esta aplicación consiste en un videojuego tiene especial interés, no sólo el grado de corrección y fluidez en el funcionamiento general de la aplicación en sí, sino también el grado de satisfacción que produce el juego como entretenimiento, así como el grado de desafío que ofrece.

En la realización de estas pruebas se ha empleado a amigos y familiares que han ido probando el juego a lo largo de todo el proceso de desarrollo y han dado su veredicto. Gracias a sus opiniones, críticas y propuestas, se ha podido desarrollar nuevos elementos o añadir nuevas funciones con las que quizá no había contado en un primer momento, así como corregir errores que se me podían haber pasado en las anteriores fases de pruebas.

De cara al futuro, si el juego pudiera estar disponible en alguna tienda de aplicaciones a nivel mundial, podrían realizarse unas pruebas de aceptación mucho más grandes y satisfactorias recibiendo *feedback* de muchos más jugadores, que ayudarían a pulir más un mayor número de aspectos del juego.

7 CONCLUSIONES Y TRABAJOS FUTUROS

En este último capítulo de la memoria del proyecto se incluyen las conclusiones obtenidas de su realización y se plantean nuevas líneas de trabajo a través de mejoras o adaptaciones sobre el juego desarrollado, así como nuevas opciones de entretenimiento que han surgido durante el proyecto.

7.1 CONCLUSIONES

El desarrollo de este proyecto tenía como objetivo realizar un videojuego de tipo Arcade pensado para plataformas móviles y que fuera desarrollado mediante el motor gráfico Unity 3D. Para su consecución se han ido realizando cada una de las fases del ciclo de vida *software*: el análisis de requisitos, la fase de diseño, la de implementación y la de pruebas, cumpliéndose satisfactoriamente los objetivos marcados al inicio del proyecto.

Las metas alcanzadas satisfactoriamente han sido:

- El diseño y creación de un juego ágil y sencillo que supone un reto en cada partida, en el cual haya diferentes niveles de dificultad, que funcione en dispositivos móviles y que su jugabilidad dependa de la tecnología de los acelerómetros que éstos llevan instalados. Esto se ha desarrollado con la plataforma Unity 3D y se ha podido exportar a Android mediante su SDK propio.

- El diseño y la programación de una interfaz gráfica fácil e intuitiva que ayude al juego a ser accesible para la mayoría de usuarios y que pueda funcionar bien con la tecnología que ofrecen los dispositivos móviles para los cuales el juego ha sido pensado. Esto se ha llevado a cabo usando las posibilidades del motor gráfico Unity.

- La creación de un servicio de comunicación entre la aplicación móvil y una base de datos con el fin de que el juego pueda disponer de un ránking online actualizado de forma que los jugadores puedan tener cierto nivel de desafío no sólo de manera individual, sino con el resto de jugadores. Esto se ha desarrollado mediante la implementación de un servidor LAMP.

Como conclusión final, cabe destacar la satisfacción personal que supone haber conseguido completar todos los objetivos propuestos al comienzo del proyecto, sobre todo al haber sido enfocado hacia una temática que me inspira tanto interés como los videojuegos, ya que la idea con la que empecé mis estudios de informática fue, precisamente, dedicarme a su diseño y creación en el futuro.

El hecho de haber desarrollado este proyecto como una idea personal y desde cero, me ha ayudado a adquirir muchos conocimientos sobre el funcionamiento de algunas de las plataformas que se utilizan para la creación de videojuegos y con la que es posible que trabaje en el futuro cercano, pues pretendo continuar mi formación especializándome en dicho ámbito.

Sin embargo, nada de lo que he desarrollado podría haberlo hecho sin los conocimientos y enseñanzas adquiridas a lo largo de toda la carrera, ya que mis nociones previas a comenzar el proyecto sobre las herramientas de creación de videojuegos eran mínimas, en el mejor de los casos, o directamente nulas. Y aunque no haya utilizado dichas herramientas durante los estudios, sí he encontrado los conocimientos necesarios para asimilar, de forma rápida, toda la información relacionada con ellas y su documentación, hasta llegar a poder diseñar y crear un videojuego como éste, que podría llegar a ser comercial, y enfrentarme sin ayuda a los continuos problemas que un proyecto de este tipo puede generar durante todo su proceso de desarrollo.

A la hora de trabajar con el motor gráfico Unity 3D, como antes he mencionado, partiendo de un conocimiento nulo de la herramienta, me he encontrado con una herramienta intuitiva y fácil de manejar con la que el aprendizaje se hace paso por paso y a buena velocidad. Sin embargo, no está exenta de problemas y errores que he ido solucionando, eso sí, gracias a su gran comunidad online, recursos como APIs o documentación, tutoriales, etc.

7.2 TRABAJOS FUTUROS

Si una cosa puede ser cierta en cuanto a los videojuegos de tipo Arcade, es que están en continuo cambio y actualización. Ya sea por medio de nuevas ediciones o por medio de actualizaciones o mejoras, estos juegos cambian continuamente añadiendo nuevos niveles, modos de juego, personajes o funciones. Así pues, también se han pensado algunas actualizaciones y mejoras que podrían implementarse en este *"Humpty Dumty's Rolling Crush"*.

7.2.1 El Modo Televisor

Esta fue una de las primeras ideas que surgió durante el desarrollo del juego pero que, finalmente, no pudo ser desarrollada por falta de tiempo. La idea consiste en construir un sistema con el que juego pudiera verse en un monitor de forma que el dispositivo móvil funcionase como mando controlador del juego. Así, el jugador seguiría utilizando los acelerómetros del dispositivo móvil como la forma de mover el huevo, pero la acción se seguiría mediante la pantalla del monitor.

Esto podría implementarse de diferentes maneras: con la ayuda del *streaming* de vídeo, enviando los datos a un receptor tipo *Chromecast* o similar que se conectaría al monitor; usando un PC como receptor de los datos del dispositivo móvil y que funcionase a modo de videoconsola, corriendo el juego en su interior y conectándolo a un monitor externo, etc.

Este nuevo modo dotaría al juego de una nueva dimensión, haciendo que pudieran crease a partir de ahí diversos modos y funcionalidades nuevas, como el modo multijugador.

7.2.2 El Modo Multijugador

En el mundo de los videojuegos, hay una corriente que lleva ganando adeptos de forma rápida y exponencial durante la última década: la de los juegos multijugador online. Estos juegos definen un sistema en el que varios jugadores participan simultáneamente en una misma partida con diferentes roles. Este modo multijugador puede ser cooperativo o competitivo, es decir, los diferentes jugadores pueden ayudarse mutuamente con el fin de lograr objetivos conjuntos, o pueden enfrentarse entre ellos hasta que uno sólo consigue el suyo.

En este sentido, añadir algún modo multijugador sería una buena manera de actualizar y mejorar el juego. Esto podría concebirse de muchas maneras distintas, pero por la temática y el tipo de juego, la manera más lógica sería la de implementar el modo competitivo, que podría ser de diversas formas: dos o más jugadores, con sus respectivos dispositivos móviles, echando una carrera en el mismo nivel, lo que podría diseñarse con la idea del modo televisor con pantalla compartida o partida; algún tipo de desafío en el que varios jugadores jugasen en el mismo nivel y, a la vez, de forma que varios huevos apareciesen en la pantalla de cada jugador, siendo sólo uno de ellos el controlado por cada uno, lo que aumentaría las formas de chocar y perder vidas, así como el riesgo de caer por los agujeros y la diversión del juego, etc.

7.2.3 La Conexión Social

Es algo cada vez más frecuente el ver incluido en juegos de este tipo, un servicio con las redes sociales de forma que los jugadores puedan publicar sus récords o hazañas en ellas de forma automática, además de poder retar a sus amigos, etc.

Este tipo de funciones no han sido añadidas en esta versión del juego porque se ha valorado el hecho de que, al carecer de un carácter comercial, los usuarios finales del juego no existen a nivel global, y no tiene sentido incluir dicha función, que es puramente publicitaria y de marketing para el propio juego.

Los usuarios que han probado el juego durante la fase de pruebas de aceptación, a quienes se les hizo la pregunta de si creían que podría añadirse dicha funcionalidad, manifestaron que, efectivamente, hasta que el juego no adquiriera un carácter comercial y fuera usado masivamente, esta función no tenía sentido. Sin embargo, en la eventualidad de que el juego pudiese comercializarse, sería una actualización que debería ser añadida.

ACRÓNIMOS

AAA	Este acrónimo se utiliza en el mundo de los videojuegos para referirse a títulos de muy alta calidad y muy alto presupuesto. Las letras vienen de las calificaciones que reciben los juegos en sus tres apartados generales (Jugabilidad, Gráficos, Sonido), y que, al ser AAA, serían sobresalientes.
API	Acrónimo del inglés que significa *Application Programming Interface* ("Interfaz de Programación de Aplicaciones").
ARPAnet	Acrónimo del inglés que significa *Advanced Research Projects Agency Network* ("Red de la Agencia de Proyectos de Investigación Avanzada").
BSD	Acrónimo del inglés que significa *Berkeley Software Distribution* ("Distribución de *Software* de Berkeley").
CPC	Acrónimo del inglés que significa *Color Personal Computer* ("Computador Personal en Color").
GNU	Acrónimo recursivo del inglés que significa *GNU is Not Unix!* ("¡GNU No es Unix!").
GPL	Acrónimo del inglés que significa *GNU General Public License* ("Licencia Pública General de GNU").
GUI	Acrónimo del inglés que significa *Graphical User Interface* ("Interfaz Gráfica de Usuario").
HTTP	Acrónimo del inglés que significa *HyperText Transfer Protocol* ("Protocolo de Transferencia de HiperTexto").
IDE	Acrónimo del inglés que significa *Integrated Development Environment* ("Entorno de Desarrollo Integrado").
iOS	Acrónimo del inglés que significa *iPhone/iPod/iPad Operating System* ("Sistema Operativo de iPhone/iPod/iPad").
IP	Acrónimo del inglés que significa *Internet Protocol* ("Protocolo de Internet").
JS	Acrónimo que referencia al lenguaje de programación JavaScript.

LAMP	Acrónimo que significa: Linux + Apache (ver apartado 3.1.6) + MySQL (ver apartado 3.1.4) + PHP (ver apartado 3.1.3).
MIT	Acrónimo del inglés que significa *Massachusetts Institute of Technology* ("Instituto de Tecnología de Massachusetts").
MySQL	Acrónimo del inglés que significa *My Structured Query Language* ("Mi Lenguaje de Consulta Estructurado").
.NET	Es un *framework* (infraestructura digital) creada por Microsoft que hace énfasis en la transferencia de redes, independientemente de la plataforma *hardware* y que permite un rápido desarrollo de aplicaciones.
NFC	Acrónimo del inglés que significa *Near Field Communication* ("Comunicación de Campo Cercano").
OpenGL	Acrónimo del inglés que significa *Open Graphics Library* ("Biblioteca de Gráficos Abierta").
OpenGL ES	Acrónimo del inglés que significa *Open Graphics Library for Embedded Systems* ("Biblioteca de Gráficos Abierta para Sistemas Embebidos").
PC	Acrónimo del inglés que significa *Personal Computer* ("Computador Personal").
PHP	En su origen, fue el acrónimo regular del inglés que significaba *Personal Home Page* ("Página Principal Personal"), pero después pasó a ser el acrónimo recursivo del inglés que significa *PHP Hypertext Preprocessor* ("Preprocesador de Hypertexto PHP").
QEMU	Acrónimo del inglés que significa *Quick Emulator* ("Emulador Rápido").
SDK	Acrónimo del inglés que significa *Software Development Kit* ("Kit de Desarrollo de *Software*").
URL	Acrónimo del inglés que significa *Uniform Resource Locator* ("Localizador Uniforme de Recursos").
Wi-Fi	Acrónimo del inglés que significa *Wireless Fidelity* ("Fidelidad Inalámbrica").
3D	Acrónimo que significa 3 Dimensiones.
2D	Acrónimo que significa 2 Dimensiones.

BIBLIOGRAFÍA Y REFERENCIAS

[1] Anderson, J. (1983) *"Who really invented the video game? There was Bell, there was Edison, there was Fermi. And then there was Higinbotham" Creative computing and video games*, n. 1, vol. 1, tomado de http://www.atarimagazines.com/cva/v1n1/inventedgames.php (consultado el 11-08-2014)

[2] Nosowitz, D. (2008) *"Retromodo: Tennis for Two, the World's First Graphical Videogame"* tomado de http://gizmodo.com/5080541/retromodo-tennis-for-two-the-worlds-first-graphical-videogame (última consulta el 11-08-2014)

[3] Luscombe, A. *"Great men you've never heard of: Alan Turing, inventor of the computer program"* tomado de http://www.primermagazine.com/2010/learn/great-men-youve-never-heard-of-alan-turing-inventor-of-the-computer-program (última consulta el 11-08-2014)

[4] Graetz, J.M. (1981). "*The Origin of Spacewar*" *Creative Computing,* pp. 56-67

[5] Epic Games. "*Unreal Engine*" tomado de https://www.unrealengine.com/ (última consulta el 19-08-2014)

[6] Garage Games. "*Torque 3D, Torque 2D e iTorque 2D*" tomado de http://www.garagegames.com/ (última consulta el 19-08-2014)

[7] Crytek. "*CryEngine*" tomado de http://cryengine.com/ (última consulta el 19-08-2014)

[8] Shiva Technologies. "*Shiva 3D*" tomado de http://www.stonetrip.com/ (última consulta el 19-08-2014)

[9] Unity Technologies. "*Unity Engine*" tomado de http://unity3d.com/es (última consulta el 19-08-2014)

[10] Flanagan, D. *JavaScript: The Definitive Guide.* (6th Edition). Editorial O`Reilly. Tomado de ftp://91.193.236.10/pub/docs/linux-support/programming/JavaScript/%5BO%60Reilly%5D%20-%20JavaScript.%20The%20Definitive%20Guide,%206th%20ed.%20-%20%5BFlanagan%5D.pdf (última consulta el 20-08-2014)

[11] Microsoft. "Guía de programación de C#" tomado de http://msdn.microsoft.com/es-es/library/67ef8sbd.aspx (última consulta el 20-08-2014)

[12] Varios autores. "*Manual de PHP*" tomado de http://php.net/manual/es/index.php (última consulta el 20-08-2014)

[13] Oracle Corporation. "*MySQL*" tomado de http://www.mysql.com/ (última consulta el 20-08-2014)

[14] Burnette, E. "*Hello Android, Introducing Google's Mobile Development Platform*" Editorial Pragmatic Bookshelf. Colección Pragmatic Programmers. Tomado de http://es.slideshare.net/housain80/hello-android-introducing-googles-mobile-development-platform-ed-burnette (última consulta el 20-08-2014)

[15] The Apache Project Foundation. "*Apache HTTP Server Project*" tomado de https://httpd.apache.org/ (última consulta el 20-08-2014)

[16] Unity Techonologies. "*Unity Manual: Mecanim Animation System*" tomado de http://docs.unity3d.com/Manual/MecanimAnimationSystem.html (última consulta el 20-08-2014)

[17] MonoDevelop. "*MonoDevelop*" tomado de http://monodevelop.com/ (última consulta el 20-08-2014)

[18] Adobe Systems Software. "*Adobe Photoshop*" tomado de http://www.adobe.com/es/products/photoshop.html (última consulta el 20-08-2014)

[19] Audacity. "*Audacity*" tomado de http://audacity.sourceforge.net/about/ (última consulta el 20-08-2014)

[20] Sverdlov, E. "*How To Install Linux, Apache, MySQL, PHP (LAMP) Stack on Ubuntu*" tomado de https://www.digitalocean.com/community/tutorials/how-to-install-linux-apache-mysql-php-lamp-stack-on-ubuntu (última consulta el 20-08-2014)

[21] Mother Goose. "*Humpty Dumpty Sat On A Wall*" tomado de http://www.poetryfoundation.org/poem/176327 (última consulta el 21-08-2014)

[22] Unity Technologies. "*Scripting API: GameObject*" tomado de http://docs.unity3d.com/ScriptReference/GameObject.html (última consulta el 16-07-2014)

[23] Autodesk Inc. "*Beast*" tomado de http://gameware.autodesk.com/beast (última consulta el 22-07-2014)

[24] Unity Technologies. "*Unity Manual: Lightmapping Quickstart*" tomado de http://docs.unity3d.com/Manual/Lightmapping.html (última consulta el 22-07-2014)

[25] Unity Technologies. "*Asset Store*" tomado de https://www.assetstore.unity3d.com/en/ (última consulta el 22-07-2014)

[26] Unity Technologies. "*Scripting API: Collider*" tomado de http://docs.unity3d.com/ScriptReference/Collider.html (última consulta el 16-07-2014)

[27] Unity Technologies. "*Scripting API: Rigidbody*" tomado de http://docs.unity3d.com/ScriptReference/Rigidbody.html (última consulta el 16-07-2014)

[28] Unity Tecnologies. "*Unity Manual: GUI Scripting Guide*" tomado de http://docs.unity3d.com/Manual/GUIScriptingGuide.html (última consulta el 16-07-2014)

[29] Unity Tecnologies. "*Scripting API: Time*" tomado de http://docs.unity3d.com/ScriptReference/Time.html (última consulta el 16-07-2014)

[30] Unity Technologies. *"Unity Documentation"* tomado de http://unity3d.com/learn/documentation (última consulta el 16-07-2014)

[31] Unity Technologies. *"Unity Tutorials"* tomado de http://unity3d.com/learn/tutorials/modules (última consulta el 14-03-2014)

[32] Unity Technologies. *"Unity Community"* tomado de http://unity3d.com/community (última consulta el 17-07-2014)

[33] Lindeman, K. (2013) "*Server Side Highsocores"* tomado de http://wiki.unity3d.com/index.php?title=Server_Side_Highscores (última consulta 22-07-2014)

[34] Llinares, J. Martínes, J.V. Candela, A. (2012) "*Gráficos a la máxima potencia: una comparativa entre motores de juegos*" *Revista 3 Ciencias*. Editada por Área de Innovación y Desarrollo, S.L. tomado de http://www.3ciencias.com/wp-content/uploads/2012/09/graficos-max-potencia.pdf (última consulta 19-08-2014)