UNIVERSIDAD POLITÉCNICA DE VALENCIA

DEPARTAMENTO DE SISTEMAS INFORMÁTICOS Y COMPUTACIÓN

MÁSTER UNIVERSITARIO EN INTELIGENCIA ARTIFICIAL,
RECONOCIMIENTO DE FORMAS E IMAGEN DIGITAL

MOTOR DE CONFIGURACIÓN EXTERNA Y DE INICIALIZACIÓN PARA LA MEJORA DE LA EFICIENCIA EN EL DESARROLLO DE JUEGOS CASUALES ONLINE MULTIJUGADOR

TESINA DE MÁSTER

Presentado por:
Isis Garrido Benet

Dirigido por:
Dr. Ramón Pascual Mollá Vaya

Valencia, 2010

Índice General

Índice de tablas y figuras

Resumen

Desde hace poco más de 50 años, la industria de los videojuegos ha experimentado una notable evolución y crecimiento hasta día de hoy. Cada vez son más los distintos medios que se utilizan con el objetivo final de ofrecer distintas formas de entretenimiento a un público cada vez más amplio y de mayor diversidad. Año tras año, estas empresas han sabido innovar y adaptarse a cada época para desarrollar y ofrecer al mundo nuevas tecnologías y formas de ocio cada vez más espectaculares y con un catálogo de juegos cada vez mayor y de mayor calidad.

La revolución de Internet ha sido, probablemente, la principal causa del crecimiento de esta industria. Cada vez son más los hogares que disponen de conexión a la red, donde es posible encontrar un sin fin de nuevas aplicaciones sencillas de utilizar y accesibles por cualquier tipo de usuario, desde el más experimentado al más nuevo o inexperto. Las empresas han sabido ver en los nuevos usuarios de Internet un jugador potencial y, por ello, hoy en día desarrollan videojuegos que se adaptan a sus gustos y necesidades: los juegos casuales.

Además, en los últimos años, poder jugar, competir o incluso conversar con otros usuarios a través de Internet se ha convertido en una de las aficiones preferidas de los internautas. Gran prueba de ello es el implacable éxito que están experimentando las redes sociales. Debido a esto, ciertas empresas del sector, sobre todo los nuevos estudios de desarrollo o empresas más pequeñas con menos recursos, dedican su esfuerzo y tiempo al desarrollo de juegos casuales online multijugador.

Aunque, comparados con los de las grandes producciones, los costes de desarrollo, tanto temporal como económico, de estos videojuegos no son muy elevados, las empresas buscan continuamente la forma de reducirlos. Reducir estos costes permite

a las empresas obtener un mayor margen de beneficios y una mayor ventaja sobre su competencia.

El objetivo principal de esta tesina es la creación de una herramienta que ayude al desarrollo de juegos casuales online multijugador agilizando la producción y reduciendo así los costes de desarrollo. Para llevar a cabo esta tarea, en primer lugar, se ha realizado un estudio crítico del modelo de producción actual y de las distintas herramientas existentes que ha permitido detectar la existencia de ciertas carencias. Por ello, en segundo lugar, se ha planteado una filosofía de diseño que será la base sobre la cual se desarrollará la nueva herramienta y que permitirá paliar dichas carencias.

Agradecimientos

Me gustaría expresar mi agradecimiento a todas aquellas personas que me han acompañado durante la realización de esta tesina.

En primer lugar, a mi familia, que a pesar de la distancia que ahora nos separa, siempre me han apoyado.

A mis compañeros de trabajo por ayudarme, siempre que ha sido necesario, tanto en la realización de esta tesina, como en el ámbito profesional, y por crear un ambiente de trabajo cada día mejor.

A Marcos, por su gran aportación y ayuda durante la realización de este proyecto.

A Ramón Mollá, por sus consejos, que me han ayudado a realizar de la mejor forma posible esta tesina.

A todas mis amigas por el gran grupo que hemos creado y por el apoyo que se me han brindado.

Capítulo 1

Introducción

1.1 Motivación

La industria del videojuego ha experimentado en los últimos años una tasa de crecimiento tan grande que la ha llevado a producir anualmente una facturación mayor que la de las industrias del cine y la música juntas.

En el caso particular de España, la facturación de la industria del videojuego supuso en 2008 un 57% del total de la facturación producida por las industrias del ocio audiovisual [2]. Esto, unido a que el 26 de marzo del 2009 la industria del videojuego fue, a efectos políticos, catalogada de industria cultural [3], ha propiciado el nacimiento de un gran número de nuevos estudios de desarrollo de videojuegos.

Los avances en el desarrollo de hardware gráfico y la generalización en el uso de conexiones de alta velocidad a Internet han hecho que los videojuegos sean cada día más espectaculares y puedan llegar a un público mucho más amplio. Actualmente, el usuario de videojuegos ya no es el chico joven al que le gustan los ordenadores, sino que esta industria está consiguiendo ser aceptada cada vez más por un público mucho más amplio y dispar.

Aunque las grandes producciones permiten a los estudios de desarrollo facturar grandes cantidades de dinero -el videojuego *"Call of Duty:Modern Warfare 2"* facturó cerca de 500 millones de dólares durante las primeras 24 horas tras su lanzamiento- éstas requieren una inversión inicial que no todos los estudios pueden permitirse. Es por ello por lo que las PYMES del sector suelen invertir su dinero y esfuerzo en el desarrollo de videojuegos que requieran una inversión inicial más moderada.

Un alto porcentaje de estos estudios de desarrollo, formados por un número reducido de trabajadores, han visto en los juegos casuales su modelo de negocio. Según [4] el mercado de los juegos casuales genera anualmente 2,25 billones de dólares, motivo más que suficiente para que los estudios vean en éstos una inversión casi segura. Los juegos casuales son juegos orientados a un público muy amplio, que puede ir desde una persona mayor que no acostumbra a consumir videojuegos o una persona joven y apasionada por el software de entretenimiento. Debido a este carácter generalista, estos juegos suelen ser fáciles de aprender y de manejar. Además, no requieren largas horas de juego y tampoco grandes habilidades por parte del usuario.

Aunque a la hora de distribuir este tipo de videojuegos existen diversas plataformas, es la plataforma online sobre PC, debido a la accesibilidad que ofrece, donde este mercado se encuentra más extendido. Actualmente un 60,4% de las viviendas españolas tiene ordenador en casa, ya sea de sobremesa o portátil, cifra que ha crecido tres puntos respecto el año anterior y que sigue en aumento y, por otro lado, un 44,6% de estos hogares tiene acceso a Internet [5].

Dentro del mercado de los juegos casuales online es posible diferenciar un tipo de juegos que van un paso más allá: los videojuegos casuales online multijugador. Estos juegos permiten al usuario jugar contra otros usuarios a través de la red. Poder jugar, competir o incluso conversar con otros usuarios a través de Internet se ha convertido hoy en día en una de las aficiones preferidas de los internautas. Un estudio realizado por Ocio Network [6] afirma que el 76% de los internautas utiliza como mínimo una red social, lo que indica la gran aceptación de este tipo de aplicaciones por parte de los usuarios. Es por esto, por lo que existen empresas que han decido combinar el desarrollo de juegos casuales online con la creación de portales sociales sobre los que los usuarios puedan jugar a los mismos. Es en su carácter multijugador donde estos juegos tienen su valor más preciado. Esta tesina se centrará en el desarrollo de este tipo de videojuegos.

A pesar de que el desarrollo de este tipo de juegos puede ser llevado a cabo en un tiempo muy inferior al de las grandes producciones, las empresas emplean continuamente diferentes estrategias que permitan acelerar, en la medida de lo posible, el proceso de desarrollo, sin que ello suponga ningún tipo de pérdida en la calidad del producto. Los principales motivos por los que las empresas buscan reducir los costes de desarrollo de sus videojuegos son:

- La reducción de costes en general permite a la empresa obtener un mayor margen de beneficios.

- La reducción del coste temporal permite a la empresa aceptar desarrollos que en otras ocasiones debería haber rechazado debido a las restricciones temporales.
- La reducción del coste temporal permite a la empresa sacar al mercado sus productos antes que la competencia.

Uno de los principales motivos que dificulta la reducción de estos costes es el modelo de producción ad-hoc que estas empresas emplean. Este modelo de producción supone tener que crear un nuevo proyecto, desde cero, cada vez que se desea desarrollar un nuevo juego. Desarrollar desde cero cada nuevo proyecto supone realizar repetidamente ciertas tareas que los distintos videojuegos poseen en común. Es por esto, por lo que, cada vez más, los estudios de desarrollo invierten parte de su capital en la creación o adquisición de nuevas herramientas que permitan modularizar el proceso de producción y generar así nuevos videojuegos de manera más eficiente.

El desarrollo o compra de motores de videojuegos es una de las estrategias más comunes que los estudios de desarrollo emplean para reducir los costes de producción. El término motor, dentro del campo de los videojuegos, hace referencia a un conjunto de algoritmos y rutinas de programación que extraen ciertas funcionalidades comunes en los videojuegos y las desarrollan de forma genérica. En otras palabras, un motor evita tener que programar, para cada juego nuevo, las mismas rutinas una y otra vez. Existen distintos motores, cada uno de los cuales atiende a partes bien distintas dentro de la programación de un videojuego (motor de física, motor de render, etc.), pero el objetivo principal de todos ellos es común: conocer de forma general el funcionamiento de un área determinada y desarrollarla de manera genérica, sin incluir ningún tipo de programación específica para un juego concreto.

Aunque el uso de motores permite acelerar el proceso de producción, esta tarea no está exenta de problemas. Por un lado, aparece la necesidad de realizar un estudio previo del motor que permita extraer el máximo provecho del mismo. Aunque esta labor es necesaria, no debería ser realizada una y otra vez para cada nuevo proyecto. Por otro lado, se puede apreciar que existen diferentes aspectos de los videojuegos para los cuales, a pesar de plantear ciertas semejanzas, no se ha desarrollado un motor que permita implementarlos de forma genérica. Concretamente, la estructura interna de un videojuego y el proceso de inicialización del mismo poseen ciertos patrones que se repiten en los diferentes juegos y para los cuales se debería proponer una solución que evite tener que desarrollarlos cada vez.

En la empresa Exelweiss Entertainment se viene trabajando desde hace más de siete años en el desarrollo de un portal de videojuegos casuales online multijugador. El portal *www.mundijuegos.com* es en la actualidad el portal español líder del sector en España y pronto comenzará su expansión internacional. Debido al modelo de producción ad-hoc que se ha seguido durante todos esos años, ha sido necesario abordar la problemática ya expuesta, dando como resultado las aportaciones recogidas en esta tesina y en [1].

1.2 Objetivos

Esta tesina aborda la problemática existente en el proceso de producción de videojuegos casuales online multijugador. Los objetivos que se persiguen son los siguientes:

1. Estudio de la metodología de desarrollo de un videojuego casual online multijugador y de las herramientas que se utilizan durante la fase de producción del mismo.
2. Detección de carencias y aspectos mejorables en el proceso de producción y propuestas para solucionarlas.
3. Desarrollo de una herramienta que resuelva las carencias detectadas e introduzca las mejoras necesarias para agilizar el proceso de producción: motor de producción como solución.
4. Desarrollo de un videojuego utilizando el motor de producción como base en su desarrollo.
5. Valoración del motor desarrollado atendiendo a: el coste de producción de la propia herramienta, el nuevo juego desarrollado y a un segundo juego cuyo desarrollo ha sido previo al motor de producción.

1.3 Contenido

La estructura de esta tesina es la siguiente:

El capítulo 2 tiene como objetivo introducir al lector en el mundo de los videojuegos. Se inicia el capítulo con un breve resumen de la historia de los videojuegos desde sus inicios, hasta llegar al estado actual en el que se encuentra la industria. Seguidamente, se analiza el tipo de usuarios de videojuegos en la actualidad, las

plataformas existentes y algunas de las herramientas utilizadas para su desarrollo. Por otro lado, se expone la estructura interna típica de un videojuego, así como el proceso seguido durante su desarrollo. Para concluir el capítulo, se realiza una introducción al uso de motores en los videojuegos y se realiza una breve clasificación de los distintos tipos.

El capítulo 3 se centra en los juegos casuales online multijugador, un nuevo modelo de negocio que se presenta a los pequeños estudios de desarrollo, y en la problemática existente en el desarrollo de este tipo de juegos. Debido a esta problemática se plantea como posible solución el desarrollo de una nueva herramienta. Dicha herramienta consiste en un motor de producción que facilita la producción de nuevos proyectos y permite reducir los costes de su desarrollo.

El capítulo 4 expone la filosofía de diseño del motor de producción y detalla sus características y funcionamiento en base a la misma.

El capítulo 5 describe cómo, con la ayuda de dicha herramienta, se ha creado un nuevo videojuego casual online multijugador. A lo largo del capítulo se muestra con mayor detalle cómo el motor de producción realiza una serie de tareas y funciones que facilitan el desarrollo del nuevo juego.

El capítulo 6 realiza una comparación de los costes de desarrollo entre este juego y otro juego de características muy similares, el cual ha sido desarrollado completamente desde cero sin la ayuda del motor de producción.

Por último, el capítulo 7 expone las conclusiones de esta tesina y sugiere posibles líneas de trabajo futuro.

En el anexo A aparece detallado un fichero XML como ejemplo práctico de un modelo de aplicación con una posible configuración de un juego.

Capítulo 2

Videojuegos y motores

2.1 Introducción

Durante este capítulo se tratan ciertos aspectos relacionados con los videojuegos y el uso de motores en los mismos. En primer lugar, en la sección 2.2 se realiza una breve introducción a los videojuegos a través de su historia, para posteriormente tratar ciertos temas que permiten conocer el estado actual de los videojuegos desde distintos puntos de vista: estado/evolución de la industria, usuarios en la actualidad y plataformas y herramientas para su desarrollo. En la sección 2.3 se describen, por un lado, las distintas fases por las que pasa un videojuego durante su proceso de desarrollo y, por otro, la estructura interna típica que éste posee. Seguidamente, en la sección 2.4 se describe el concepto "motor" dentro del campo de los videojuegos para, posteriormente, realizar una clasificación de los motores existentes y estudiar su uso. Finalmente, en la sección 2.5 se realizan las conclusiones de este capítulo.

2.2 Videojuegos

Los videojuegos han experimentado una gran evolución a lo largo de la historia desde que, hace poco más de 50 años, se desarrollara el mítico juego "Tenis para dos". El conocido juego fue creado en 1958 por William Higinbotham, un físico estadounidense, con el objetivo de entretener a la gente durante una de sus exposiciones. Unos años más tarde, en 1961, Steve Russell, un estudiante del MIT, creó el "Space War", el primer juego para ordenador.

La primera videoconsola de la historia llegó de la mano de Ralph Baer, en 1972. Se bautizó con el nombre de “Magnavox Odyssey”, salió al mercado con un precio de 100$ y contó con 28 juegos de extrema sencillez, entre ellos el ping-pong. El mismo año Nolan Bushnell junto a su compañero Ted Dabney, fundaron Atari, la cual, después de algunas disputas legales con Ralph Baer, consiguió comercializar el conocido juego “Pong”, creación de Al Alcorn, haciendo su aparición en las primeras máquinas arcade recreativas.

En 1977, llegó de manos de Atari la primera videoconsola de 8 bits que cargaba los juegos mediante un nuevo sistema de cartuchos, la “Atari 2600”. En 1983, cuando la industria parecía empezar a decaer debido a la falta de innovación, una empresa nacida en 1889, que empezaba a interesarse por el mundo del videojuego, lanzó al mercado la “Nintendo Entertainment System”, o “NES”, la primera consola que dio beneficios a sus fabricantes, Nintendo.

A partir de entonces la industria del videojuego ha estado en continua evolución. En muy poco tiempo fueron surgiendo nuevas empresas y otras no tan nuevas, interesadas en aportar sus creaciones e innovaciones a un sector en pleno crecimiento.

Junto con la “NES” aparecieron grandes títulos como “Donkey Kong”, “Mario Bros” o “The Legend of Zelda”. Mientras tanto, Sega, que ya había entrado en el mercado con su consola de 8 bits, la “Sega Master System”, empezó a desarrollar la siguiente generación de consolas. En 1988, lanzó “Sega Mega Drive”, la primera consola de 16 bits, acompañada del famoso personaje “Sonic”. Un año más tarde apareció la “Super Nintendo Entertainment System”, también conocida como “Super Nintendo”, como competencia directa de Sega.

La siguiente generación de consolas fueron las de 32 bits. En 1992, Sega lanzó su consola “Sega Saturn” y, en 1994, Sony lanzó la “Sony PlayStation”. En 1996, Nintendo sacó al mercado la primera consola de 64 bits, la “Nintendo 64”, que consiguió ser líder de ventas, gracias a ciertas mejoras incorporadas como los mandos con vibración, los sticks de un dedo para juegos en entornos tridimensionales y los 4 puertos de mandos para que pudiesen jugar 4 jugadores simultáneamente.

Antes de llegar al estado actual de la industria del videojuego, llegaron las consolas de 128 bits. Fue en 1998 cuando Sega lanzó “Dreamcast”, la primera consola que incorporó un módem para permitir jugar online, y la última consola que lanzaría Sega. Dos años más tarde llegó “PlayStation 2”, la cual, gracias a su predecesora y a la compatibilidad de los juegos antiguos con la nueva consola, eclipsó a “Dreamcast”,

convirtiéndose en la consola con mayor ciclo de vida, compitiendo actualmente con las consolas de la siguiente generación a la suya. Nintendo por su parte, lanzó la "Game Cube" y Microsoft, que aunque había colaborado anteriormente con Sega en el desarrollo de "Dreamcast", se incorporó también a la gran industria del entretenimiento con su "Xbox", la primera consola que incorporaba disco duro.

A lo largo de la corta historia de la industria de los videojuegos se puede observar su rápido crecimiento y capacidad de evolución e innovación llegando hasta nuestros días, donde podemos encontrar en el mercado un amplio catálogo de videojuegos y una nueva generación de consolas cada vez más potentes que ofrecen al público un gran número de posibilidades y nuevas experiencias y que sin duda alguna seguirá creciendo, evolucionando y sorprendiendo a todo el mundo con sus nuevas tecnologías.

Para una revisión más detallada sobre la historia de los videojuegos véanse [7, 8].

2.2.1 Evolución/Estado actual de la industria de los videojuegos

La industria del videojuego lleva creciendo y apostando por las nuevas tecnologías desde sus inicios hasta el día de hoy. Actualmente, se ha incorporado al mercado la nueva generación de videoconsolas, tanto de sobremesa como portátiles. El primero en presentar su nueva consola fue Microsoft con su Xbox 360, que vio la luz a finales del 2005. Un año más tarde llegó la nueva "PlayStation 3" de la mano de Sony y la revolucionaria "Wii" propiedad de Nintendo. Por otro lado, no hay que olvidar a los ordenadores como plataforma de juego. Hoy en día, más la mitad de los hogares españoles tiene un ordenador en casa y aunque se diseñó como herramienta de trabajo, ha sido tal su avance tecnológico que es capaz de competir con las consolas de última generación.

Un gran impulso para la industria del videojuego ha sido su catalogación, a efectos políticos, de industria cultural [3]. Acciones como esta, sumadas al avance tecnológico y a la expansión de Internet, han permitido que, a día de hoy, la industria del ocio interactivo digital supere en facturación a las industrias del cine y la música juntas. Según la Asociación Española de Distribuidores y Editores de Software de Entretenimiento, ADeSe [9], en el año 2008 el consumo de los videojuegos supuso un

57% del total del consumo en ocio audiovisual en España, superando a otros sectores como son el cine, la música o las películas de DVD.

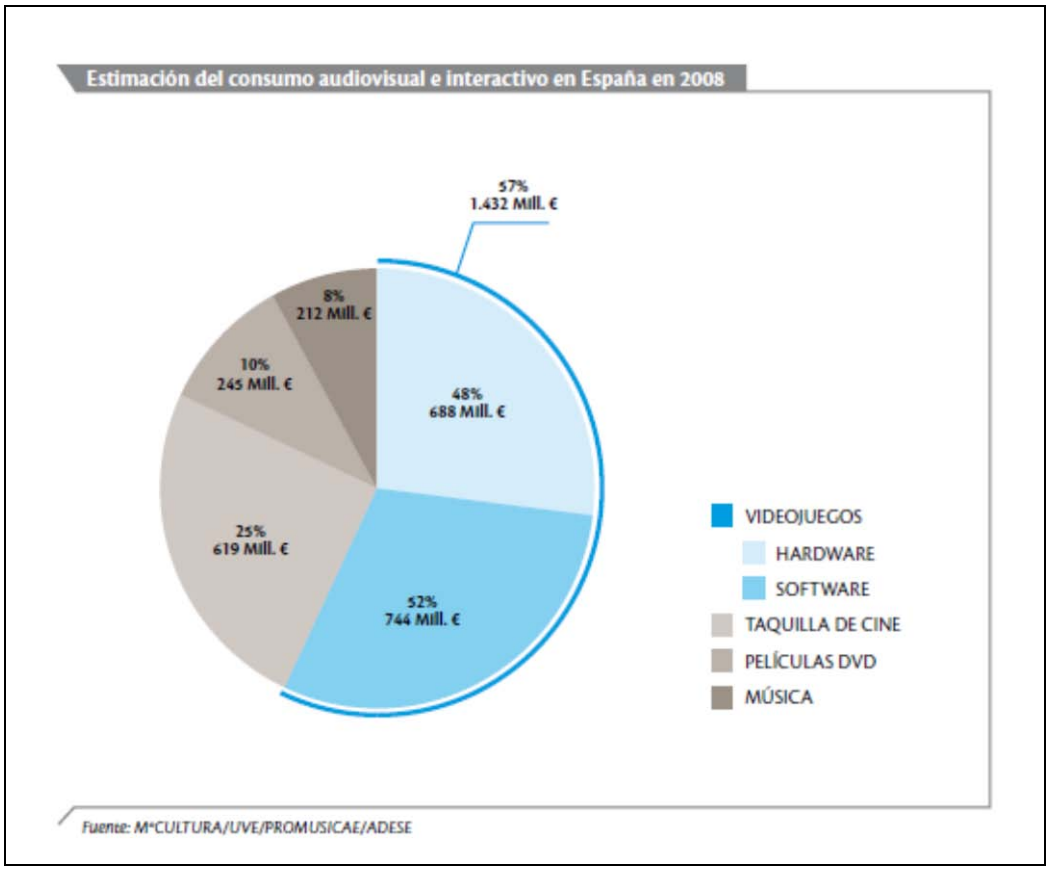

Figura 1. Estimación del consumo audiovisual e interactivo en España en 2008. Fuente: ADeSe

Aunque en 2008 las ventas de consolas fueron algo inferior frente al 2007, un 6,2%, el balance de la facturación global de la industria del videojuego se mantuvo. Esto fue debido, por un lado, a que en 2007 llegaron a España las consolas de sobremesa de nueva generación. Como se puede observar en la siguiente gráfica, las ventas de consolas de sobremesa aumentaron en un 5%, permitiendo contrarrestar, en parte, el descenso en las ventas de consolas portátiles.

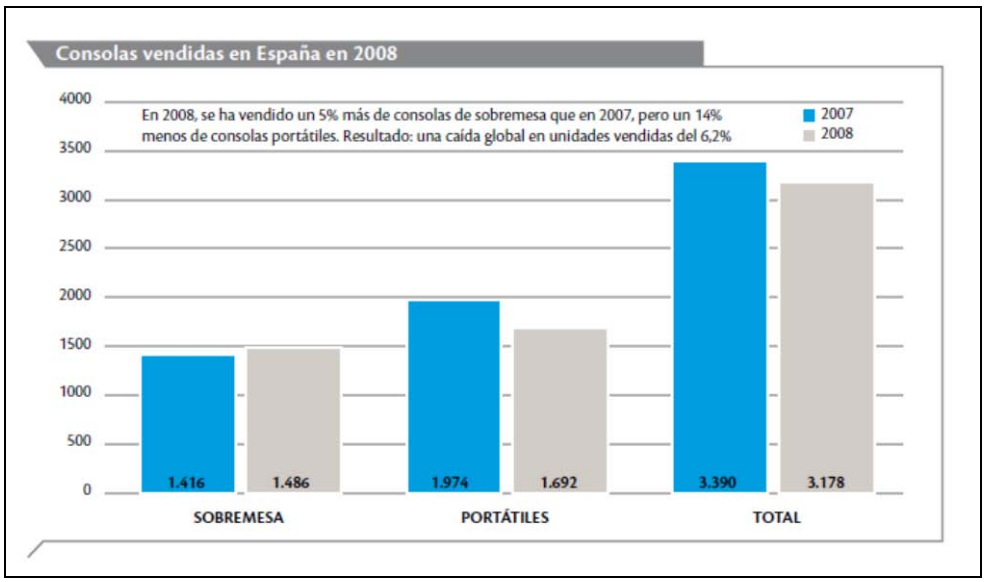

Figura 2. Consolas vendidas en España en 2008. Fuente: ADeSe

Por otro lado, además del incremento en la venta de consolas de sobremesa se registró un incremento de un 3% en la venta de videojuegos, obteniéndose así un balance global del -1.5% respecto el 2007.

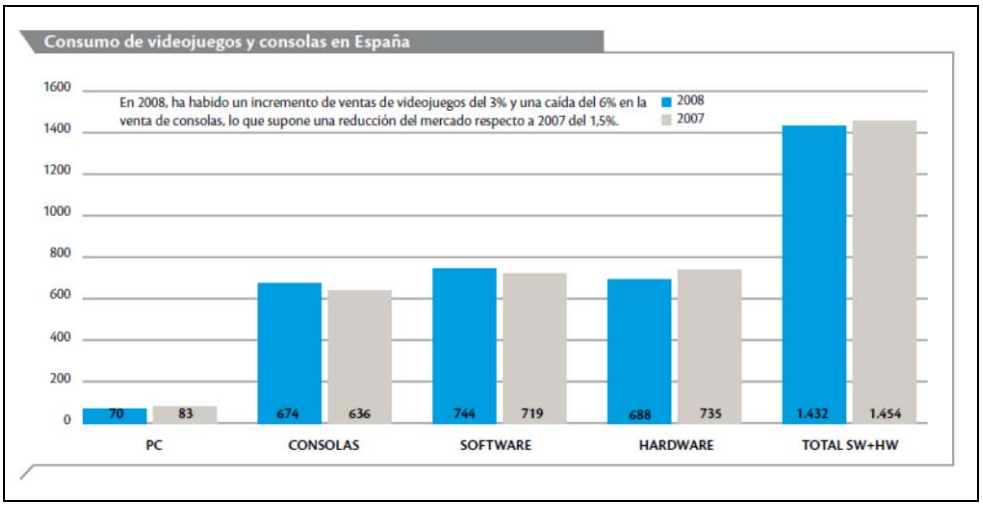

Figura 3. Consumo de videojuegos y consolas en España. Fuente: ADeSe

A pesar de que el 2008 fue un año de crisis para muchos sectores, la industria del videojuego obtuvo un incremento del 42% en sus ventas. A nivel mundial es un sector en pleno crecimiento, como demuestra la gráfica que se presenta a continuación, donde se hace un balance del crecimiento que ha tenido este sector en la última década. Los

datos presentados son el resultado de un estudio realizado por Media Control Gfk International, una consultora que atribuyó un 53% del mercado del entretenimiento casero al sector de los videojuegos.

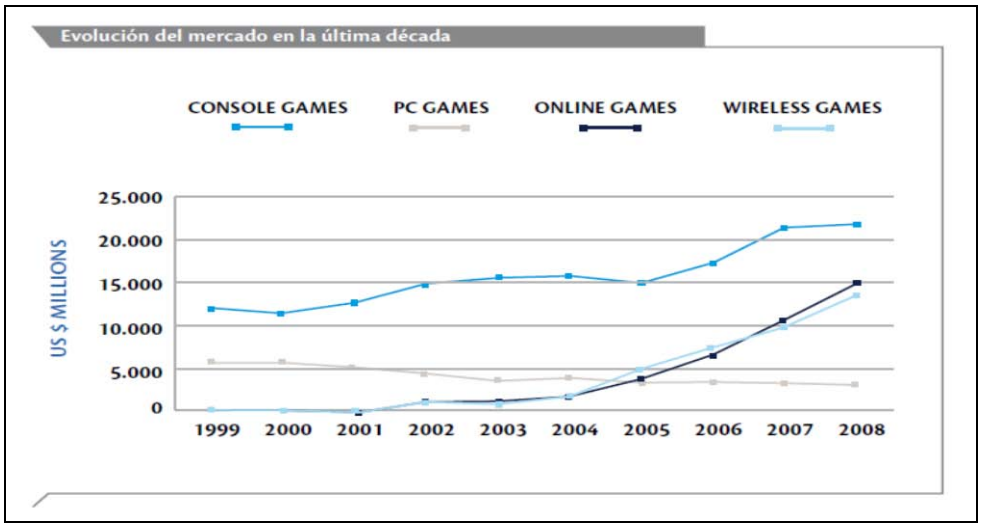

Figura 4. Evolución del mercado de los videojuegos en la última década. Fuente: ADeSe

A nivel Europeo, donde España se encuentra en cuarta posición como consumidora de videojuegos, por detrás de Inglaterra, Francia y Alemania, en 2008 se obtuvo un aumento medio del 15% en ventas respecto al 2007. Estas ventas hacen referencia tanto a la venta de consolas, como a la de videojuegos para todas las plataformas. Esto supuso unas ganancias de 2.000 millones de euros más que en 2007, habiéndose facturado un total de 15.000 millones de euros.

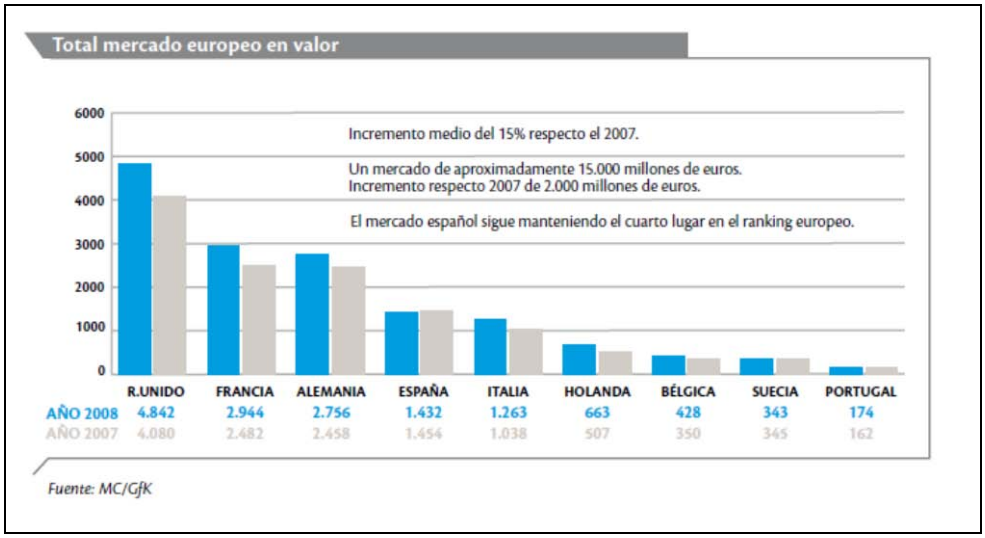

Figura 5. Total mercado europeo de los videojuegos. Fuente ADeSe

Una revisión más detallada de estas estadísticas y gráficas se hace en [2] y [10].

Respecto al 2009, la industria del videojuego ha seguido manteniendo su nivel y ha superado de nuevo las ventas de las industrias del cine, la música y las películas en DVD juntas, alcanzando un 53% del total del consumo en ocio audiovisual en España.

Por otro lado, en el mercado Europeo, se ha producido una disminución de las ventas en general debido a la situación económica actual. A nivel nacional se ha producido un decremento de las ventas de consolas y videojuegos en un 16%. A pesar de estos datos, España sigue manteniendo su cuarta posición como consumidora de videojuegos en Europa llegando a facturar un total de 1.200 millones de euros.

2.2.2 Evolución de los usuarios de videojuegos

Un aspecto clave que ha permitido el crecimiento de la industria ha sido la aparición de un nuevo tipo de usuarios de videojuegos. En los últimos años, las empresas del sector, tanto las multinacionales como las empresas más pequeñas, han intentado acercarse a otro tipo de público más amplio, todavía sin explotar. Hasta ahora se asociaba el mundo de los videojuegos con un perfil de usuario joven y mayoritariamente masculino, pero desde hace unos años se ha ido ampliando este perfil. Gracias a la aparición de nuevos contenidos y diferentes formas de jugar, las empresas ofrecen nuevas posibilidades a gusto de todo tipo de consumidor.

Fue Nintendo la primera en proponer formas de juego diferentes, tanto con su consola de sobremesa, “Wii”, como con su consola portátil de pantalla táctil, “Nintendo DS”. Con sus máquinas en el mercado y un amplio catálogo de juegos para ofrecer, Nintendo ha abierto el mundo de los videojuegos a un sector de usuarios hasta el momento olvidado: niños, mujeres y gente mayor.

Los juegos que han conseguido atraer a este tipo de público son los llamados juegos “casuales”. Se trata de un tipo de juegos fáciles de aprender y de manejar, con una visualización atractiva y llamativa para el consumidor y para los que no es necesario dedicarles muchas horas. Tampoco requieren de grandes habilidades por parte del usuario, lo cual resulta muy atractivo puesto que no requiere un aprendizaje previo. Por otro lado, los costes de desarrollo de estos juegos suelen ser bastante reducidos si se comparan con los de las grandes producciones.

Según un estudio [4] de la International Game Developers Association, IGDA, en los últimos años se ha producido un gran incremento del número de juegos casuales,

llegando a facturar el año pasado hasta un total de 2,25 billones de dólares anuales, un 20% más que en 2007. Por otro lado, un 25% de los usuarios de internet en todo el mundo afirman jugar a juegos online cada semana, lo que supone para la industria un total de 200 millones de usuarios que juegan a este tipo de juegos todas las semanas.

Ligado a este tipo de juegos están Internet, los juegos online y las redes sociales. Actualmente un 60,4% de las viviendas españolas tiene ordenador, ya sea de sobremesa o portátil, cifra que ha crecido tres puntos respecto del año anterior y que sigue en aumento. Por otro lado, un 44,6% de los hogares españoles tiene acceso a Internet, frente al 39,1% del año anterior. Por último, la aparición y el éxito de las redes sociales en estos últimos años y la incorporación de juegos casuales online en ellas han propiciado que un mayor número de internautas jueguen a este tipo de juegos. Un claro ejemplo de este tipo de redes sociales es Facebook, en la cual se han incluido infinidad de aplicaciones, entre ellas juegos casuales para jugar con amigos, contra ellos o con desconocidos.

Según un estudio [6] realizado por Ocio Network sobre los hábitos de Internet en España en 2009, el 76% de los internautas encuestados utiliza como mínimo una red social. Por otro lado, casi la mitad de los usuarios de las redes sociales afirman utilizar más de una, llegando a ser usuarios de hasta cuatro de ellas.

2.2.3 Plataformas y herramientas

A la hora de jugar a los distintos videojuegos que ofrece el mercado, existen diferentes medios sobre los que ejecutarlos. Estos medios son conocidos como plataformas de juego y constan de un conjunto de componentes hardware y software que permiten ejecutar las aplicaciones compatibles.

Las plataformas más comunes son las siguientes:

- **Arcade.** Las máquinas arcade son aquellas que se encuentran normalmente en las salas recreativas. Normalmente, funcionan a partir de la inserción de monedas y solo permiten jugar a un juego por máquina. Los juegos se controlan mediante joysticks y botones y, generalmente, se juega de pie. Según han ido evolucionando se han ido introduciendo otro tipo de controles, como armas, pedales o controles en los pies.

- **Consola de sobremesa.** Las consolas se utilizan generalmente para jugar en casa conectándolas al televisor y suelen soportar hasta cuatro jugadores simultáneos. Una característica importante de esta plataforma es que está creada y diseñada específicamente para ejecutar videojuegos. Las consolas más importantes hoy en día son: "Xbox 360" de Microsoft, "PlayStation 3" de Sony y "Wii" de Nintendo.

- **Ordenador.** Se trata de la plataforma más común en los hogares, dada las múltiples aplicaciones para las que puede ser utilizada. Hoy en día los precios de los ordenadores han descendido mucho y son bastante asequibles.

- **Online.** Aunque normalmente este tipo de plataforma va ligada al ordenador o a las consolas, la tecnología sobre la que se desarrolla y se ejecuta necesita de una conexión a internet y generalmente un servidor donde guardar los datos. La característica principal de esta plataforma es que permite que cientos de jugadores jueguen simultáneamente al mismo juego.

- **Consola portátil.** Se trata de pequeñas consolas que permiten a los usuarios jugar en cualquier lugar. No son tan potentes como las consolas de sobremesa, pero, al igual que éstas, están creadas y diseñadas para ejecutar videojuegos. Hoy en día algunos teléfonos y PDAs se consideran también plataformas de juego portátiles.

Tanto las plataformas arcade como las consolas son plataformas propietarias, lo que significa que el software para estas máquinas se desarrolla exclusivamente para ellas. Una ventaja de desarrollar para este tipo de plataformas es que el hardware no varía, a diferencia de lo que ocurre en el caso de los ordenadores.

Debido al gran número de hogares que actualmente poseen ordenador y una conexión de banda ancha, se puede concluir que la plataforma online ejecutada sobre un ordenador es la más accesible. Por tanto, desarrollar videojuegos para esta plataforma supone que un mayor número de usuarios pueda acceder a ellos.

A la hora de desarrollar un juego online se encuentran las siguientes tecnologías:

- **Flash** es una tecnología de Adobe muy popular que permite el desarrollo de aplicaciones interactivas, como animaciones, presentaciones, juegos, etc. Utiliza imágenes vectoriales, lo que permite redimensionarlas sin que éstas se pixelen.

Actualmente, el lenguaje de programación de Flash, ActionScript 3, ha supuesto un salto cualitativo con respecto a sus predecesores. El plugin necesario para ejecutar aplicaciones Flash sobre un navegador se encuentra instalado actualmente en el 99% de los navegadores [12].

- **Shockwave** es una tecnología también de Adobe que permite crear el mismo tipo de aplicaciones que Flash. Adicionalmente, Shockwave soporta contenido 3D y permite importar y ejecutar archivos Flash. El plugin necesario en este caso se encuentra instalado en un 50% de los navegadores [13]. Como lenguajes de desarrollo se puede emplear tanto Lingo como Javascript.

- **Java** más que una tecnología es un lenguaje de programación desarrollado por Sun MicroSystems. Al igual que las dos tecnologías anteriores es necesario tener instalado un plugin que permita su ejecución en un navegador y además la aplicación debe ser instalada en la máquina del usuario. El plugin necesario en este caso se encuentra instalado actualmente en un 56% de los navegadores [14].

- **Unity** es una tecnología poco conocida pero muy potente en lo que a desarrollo 3D se refiere, ya que incorpora su propio editor de escenas. Soporta como lenguajes de programación Javascript, C# y AJAX. La principal desventaja de esta tecnología es que solo puede ser desarrollada bajo Mac y además el plugin necesario para su ejecución en la web se encuentra instalado tan solo en un 1% de los navegadores [15], a pesar de su fiable y rápida instalación.

Existen muchas más tecnologías para desarrollar juegos online, tanto en dos como en tres dimensiones, pero las mencionadas son las más utilizadas y populares a día de hoy. Otras tecnologías, menos extendidas, que se pueden utilizar son VirTools, PTK, PopCap, Torque 2D y 3D, PaperVision3D o Away 3D entre muchas otras.

2.3 Fases de desarrollo y estructura de un videojuego

Los videojuegos pueden presentar temáticas bien diferenciadas, pueden estar orientados a públicos totalmente distintos e incluso pueden estar pensados para ejecutarse en plataformas totalmente diferentes. A pesar de todo, todos los videojuegos poseen en común un proceso de desarrollo que mantiene una estructura claramente definida así como una estructura interna de ejecución que mantiene siempre un orden específico.

Durante su desarrollo, un videojuego pasa por una serie de etapas que van desde la visualización de una idea que puede convertirse en un juego hasta su posterior distribución y venta.

Del mismo modo, todo videojuego posee una estructura interna que define el orden de ejecución de los eventos que se producen desde el instante en el que el juego arranca hasta que su ejecución finaliza.

2.3.1 Fases de desarrollo de un videojuego

Existen diferentes fases en el proceso de desarrollo de un videojuego [8]: concepto, pre-producción, prototipo, producción, alfa, beta, máster y post-producción.

La fase de concepto comienza cuando se visualiza una posible idea para la creación de un videojuego y termina cuando esta idea se ha madurado y se ha tomado una decisión al respecto. Durante esta fase, el equipo de desarrollo suele estar formado por un número reducido de personas; bastaría con un diseñador, un programador, un artista y un productor. La finalidad de la fase de concepto es definir en qué consiste el juego y transmitir esta idea a los demás de manera escrita. El documento de concepto define el *target* (a quién va destinado el juego) y explica porqué la idea es una oportunidad de mercado.

Una vez definido el documento de concepto, se procede al desarrollo de la propuesta; se entra en la fase de pre-producción del juego. En esta fase se desarrolla un documento adicional, que incluye una guía del estilo artístico, y un plan de producción. Esta fase concluye con la creación del documento de diseño.

El siguiente objetivo es crear un prototipo. La creación del mismo consiste en el desarrollo de un software que permita plasmar en la pantalla la esencia del juego y qué lo hace especial con respecto al resto de juegos. El prototipo permite mostrar la idea del juego al publisher.

Tras la aprobación del prototipo, el equipo de desarrollo se prepara para la fase más extensa; la fase de producción, cuyo objetivo es el desarrollo completo del juego. Esta fase, incluye ciertas tareas como la codificación del programa, la creación de los gráficos 2D o modelos 3D, la creación del interfaz que permitirá al jugador interactuar con el juego y la grabación de las voces o sonidos necesarios.

Dependiendo de la magnitud del juego, la codificación del mismo puede precisar de programadores especializados en ciertas áreas, o que por el contrario, uno o varios programadores lleven a cabo la realización de todas las tareas. Los diferentes programadores especializados en las distintas áreas existentes dentro de la programación de un videojuego son [15]:

- **Programador de físicas:** especialista en el desarrollo de la física existente en el videojuego. Principalmente su función es la de optimizar los cálculos físicos necesarios para la simulación del juego debido al alto coste computacional de los mismos.

- **Programador de interfaz:** especialista en la programación de la interfaz gráfica mediante la cual el usuario interactúa con el juego.

- **Programador de sonido:** especialista en la programación de técnicas avanzadas de sonido como la espacialización de sonido 3D.

- **Programador de inteligencia artificial:** especialista en el desarrollo de la lógica del juego que permitirá al mismo tomar decisiones.

- **Programador gráfico:** especialista en la creación de algoritmos para el desarrollo y optimización de gráficos 2D/3D.

- **Programador de red:** especialista en programar todo lo relacionado con la comunicación, vía LAN o internet, entre los distintos jugadores.

- **Programador de herramientas**: especialista en la creación de herramientas que faciliten el desarrollo del juego y permitan un ahorro en los costes de producción.

- **Programador jefe:** encargado de controlar la programación del juego en su totalidad y comprobar que todos los módulos del juego se están desarrollando de forma correcta.

En el momento en el que el juego se encuentra totalmente codificado es necesario realizar ciertas pruebas que verifiquen su correcto funcionamiento. Durante la fase alfa del proceso de desarrollo se comprueba que el producto es jugable de principio

a fin. Mientras se realizan las pruebas pertinentes, el departamento de testing realiza las siguientes tareas: se asegura de que cada módulo del juego es testeado al menos una vez, crea una base de datos con los errores encontrados, crea un plan de prueba y registra dichos errores. En esta fase, es en la que, por primera vez, alguien externo al equipo de desarrollo puede ver el juego. Para que un juego pase la fase alfa los siguientes elementos deben estar completados:

- Jugable de principio a fin.
- Textos del idioma principal.
- Interfaz básica con documentación preliminar.
- Compatibilidad con las diferentes configuraciones de hardware y software.
- Funcionalidad con los requerimientos mínimos.
- Compatibilidad con interfaces externos.
- Funcionalidad multijugador (si la hubiera).
- Proyecto del manual.

Una vez pasada la fase alfa, el juego entra en fase beta. El objetivo perseguido en esta fase es el de eliminar el mayor número de errores posibles y pulir y estabilizar el producto. Para que un juego pase el testeo beta los siguientes puntos han de estar completados:

- Código.
- Contenido.
- Textos de todos los lenguajes.
- Interfaz de usuario.
- Compatibilidad hardware y software.
- Arte y audio.
- Manual de juego.

Cuando un juego pasa su fase beta es considerado como el máster del juego. En este momento el juego es copiado y empaquetado para su posterior venta.

Una vez el juego está en venta, se entra en la fase de post-producción, en la que se desarrollan extensiones del juego que permiten alargar su ciclo de vida, se crean *patchs* que arreglan posibles *bugs* existentes en la versión definitiva del juego o se realizan actualizaciones que mejoran el juego original en alguno de sus aspectos.

Para un estudio exhaustivo de estas fases, así como del proceso de desarrollo de un videojuego en general pueden consultarse [7, 8, 16].

2.3.2 Estructura de un videojuego

Independientemente de la magnitud de un videojuego y de las técnicas empleadas durante su fase de desarrollo, internamente, éste sigue una estructura típica que lleva siendo utilizada desde los primeros videojuegos hasta los de última generación. El eje central de esta estructura es su bucle principal, el cual se encuentra sincronizado con la tasa de refresco de la pantalla, ejecutándose así una vez por cada imagen generada. La función principal de este bucle es la de realizar una secuencia de tareas siempre en el mismo orden. De forma esquemática, esta estructura se puede resumir en los siguientes puntos [7]:

1. Inicialización del programa principal.
2. Bucle principal del juego.
    a. Inicialización del interfaz.
    b. Bucle de interfaz.
        i. Recibir inputs.
        ii. Renderizar la pantalla.
        iii. Actualizar el estado del interfaz.
        iv. Gestionar los cambios de estado del interfaz.
    c. Finalización del interfaz.
    d. Inicialización del nivel de juego.
    e. Bucle de nivel.
        i. Recibir inputs.
        ii. Ejecutar IA
        iii. Realizar un paso en la simulación física.
        iv. Actualizar las entidades de juego.
        v. Recibir/Mandar mensajes.
        vi. Actualizar tiempo.
        vii. Actualizar el estado del juego.
        viii. Otras acciones posibles.
    f. Finalización del nivel actual
3. Finalización del juego.

Esta lista resume la secuencia de eventos que se producen en un videojuego desde el momento en el que empieza hasta su finalización.

Como se puede observar, existen ciertos patrones en la secuencia de eventos. Los eventos de inicialización y finalización siempre van emparejados, por lo que para cada inicialización se produce la finalización pertinente.

El propósito de la inicialización es el de preparar todo lo que sea necesario para poder continuar la ejecución de un juego.

Por otro lado, cada finalización realiza una secuencia de acciones que se encargan de destruir todo lo que se crea en la fase de inicialización. Este proceso se realiza en el orden inverso al de inicialización, es decir, lo último que se crea en la fase de inicialización es lo primero que se destruye en la fase de finalización.

Es muy importante que la finalización deshaga todo lo que la fase de inicialización hace. El hecho de no realizar esta acción provoca pérdidas de memoria, subestados que nunca son finalizados y aparición de errores.

2.4 Motores

En los orígenes de los videojuegos, cada uno de ellos era desarrollado de forma independiente, de manera que tan solo se implementaban las características propias de cada uno. Si un juego necesitaba conocer, por ejemplo, la trayectoria seguida por un proyectil, dicha funcionalidad era desarrollada. Esto no permitía conocer otro comportamiento del proyectil a excepción de su trayectoria, pero era más que eficiente en el cálculo de la misma. Esta estrategia de desarrollo es útil a la hora de crear juegos simples en los que la cantidad de código puede no exceder de unos cientos de líneas.

No obstante, a medida que la complejidad en el desarrollo del videojuego aumenta se hace complicado desarrollar de manera eficiente cada una de las partes del mismo, ya que desarrollar desde cero cada una de las funcionalidades supone un coste elevado. La dificultad a la hora de desarrollar cada juego de forma independiente, sumado a la existencia de funcionalidades comunes en los mismos, plantea la necesidad de extraer dichas partes comunes y programarlas de forma general, de manera que este código pueda ser reutilizado en futuros desarrollos y se garantice su total funcionalidad. Es aquí donde aparece el término motor o engine.

Debido al carácter genérico que ha de poseer un motor, toda la funcionalidad que este incorpore ha de ser planteada de forma general y nunca se ha de pensar en añadir funcionalidades específicas para un juego en particular. En otras palabras, un motor es una pieza de código que conoce de forma general el funcionamiento de un área determinada dentro de la programación de un videojuego, pero que nunca contiene programación específica para una determinada funcionalidad de un juego en concreto.

2.4.1 Clasificación de los motores existentes y su uso en videojuegos

Antes de emplear un motor en un proyecto se ha de tomar una decisión: emplear uno ya creado o elaborar uno propio.

Adquirir la licencia de un motor puede suponer un desembolso económico inicial que variará en función de la calidad del primero. Por contra, plantea ciertas ventajas como la de conocer de antemano que estamos ante un software fiable y robusto que permite trabajar con toda su funcionalidad desde el primer día.

Por otro lado, la creación desde cero de un motor permite añadir comportamientos innovadores que no pueden encontrarse en los motores licenciados. Claramente, los inconvenientes de un motor propio son su elevado coste de producción, ya que todo ha de ser implementado desde cero, y la necesidad de realizar un testeo exhaustivo que compruebe el correcto funcionamiento del mismo.

Atendiendo a los diferentes tipos de motores con los que se puede trabajar, se puede observar una correlación directa entre estos y las distintas áreas de la programación vistas en la sección 2.3. Cada uno de los estos motores está pensado para implementar las funcionalidades comunes de cada una de estas áreas.

Por un lado, el motor de física [17, 18] es capaz de simular el comportamiento físico de los objetos existentes en la escena en función de los parámetros dinámicos de los mismos (velocidad, rozamiento, masa, etc.). En un videojuego, el comportamiento de los objetos físicos existentes debe asemejarse con la mayor exactitud posible al comportamiento que estos tienen en la vida real. Con ello se dota al producto de una mayor naturalidad y se enriquece la experiencia de juego del usuario. Es por esto por lo que los motores de física cobran tanta importancia en el proceso de producción de un videojuego. A la hora de desarrollar un motor de física existen dos vertientes: motores precisos y motores de tiempo real. En el primero prima la precisión, es decir, el obtener una simulación física que se comporte tal y como se comportan los objetos físicos en el mundo real, mientras que en el segundo, lo más importante es obtener unos resultados suficientemente fieles a la realidad con un menor coste temporal, sin necesidad de que estos sean tan exactos. En el caso de los videojuegos es conveniente el uso de motores que realicen sus cálculos en el menor tiempo posible, aunque esto suponga una perdida en la precisión de los mismos.

Al igual que en el caso del motor de física, el motor de colisiones pretende que el resultado de las colisiones producidas por los distintos cuerpos físicos sean lo más

parecido posible a como lo sería en la vida real. Para conseguir este objetivo y evitar situaciones indeseadas, como la interpenetración entre cuerpos rígidos, el motor de colisiones se hace valer de los *sistemas de detección y de resolución de colisiones.* La detección de colisiones es un problema computacional de tipo geométrico en el que se determina si dos cuerpos rígidos colisionan y en caso de que colisionen indica en qué puntos de los mismos se produce el contacto. Por otro lado, la resolución de colisiones plantea un problema físico, que permite determinar el movimiento de los cuerpos rígidos tras una colisión. Se puede decir por tanto, que los algoritmos de resolución de colisiones determinan el efecto físico que se produce en un cuerpo rígido tras cada colisión detectada por los algoritmos de detección de colisiones. Normalmente, este motor está integrado en el motor de física, englobando este último a ambos.

Por otro lado, el motor de inteligencia artificial [19] contiene los algoritmos generales empleados en las tomas de decisión por parte del videojuego. Estos algoritmos permiten a los componentes controlados por el juego (enemigos, cooperantes, etc.) tomar una decisión "similar" a la que tomaría el jugador en aquellos estados del juego en los que existen diferentes opciones de reacción.

El motor de audio [20] permite el tratamiento y la modificación de los elementos sonoros presentes en un videojuego. Algunas funcionalidades existentes en un motor de física son: manipulación básica de sonidos 2D/3D (pitch, volumen, pan, velocidad, play, stop, etc.), carga de ficheros de audio desde diferentes fuentes (CD, disco duro, etc.), espacialización del sonido 3D, etc.

Del motor de render [21, 22] se puede decir que es el encargado de realizar todas las acciones que conlleva el muestreo del videojuego en la pantalla. Esta imagen renderizada está definida en función de ciertas características como la luminosidad, la refracción, las sombras, las transparencias, etc. El motor de render es, por tanto, el encargado de aplicar los algoritmos pertinentes para conseguir esta visualización de manera eficiente.

Por último, el motor de red es el encargado de implementar toda la funcionalidad básica de la comunicación vía LAN o Internet entre los distintos jugadores de un juego y de los mismos con los servidores.

Aunque los distintos motores enunciados son utilizados en la programación de partes bien diferenciadas dentro de la producción de un videojuego, todos persiguen el mismo objetivo: extraer las funcionalidades comunes, dentro del ámbito al que

pertenecen, y desarrollarlas de forma genérica, permitiendo así que en el desarrollo de futuros juegos toda esta funcionalidad no deba ser codificada de nuevo.

2.5 Conclusiones

En este capítulo se han tratado, a modo de introducción, distintos aspectos relacionados con el mundo de los videojuegos.

En primer lugar, se ha contado, de forma breve, la historia de los videojuegos, desde su aparición con el juego “Tenis para dos” hasta los videojuegos actuales de última generación cada vez más complejos y espectaculares.

A continuación, se ha realizado un estudio del estado actual de la industria del videojuego. En este estudio se observa el imparable crecimiento de la industria, la cual actualmente produce una facturación mayor que la de las industrias del cine y la música juntas.

El siguiente punto tratado ha sido el de la evolución de los usuarios de videojuegos. La conclusión a la que se llega tras este estudio es que, cada vez más, los videojuegos se están aproximando a un público más amplio, dejando de un lado el estereotipo de varón joven.

Por otro lado, se han detallado las distintas plataformas bajo las que un videojuego puede ser ejecutado, así como las distintas herramientas existentes para su desarrollo. El estudio de estas herramientas se ha centrado en aquellas que pueden ser empleadas para el desarrollo de juegos online.

Independientemente del tipo de juego creado, existe una serie de fases por las que éste ha de pasar durante su proceso de desarrollo. Tal y como se ha visto, estas fases son: concepto, pre-producción, prototipo, producción, alfa, beta, máster y post-producción.

Además, a pesar del gran salto de calidad que han sufrido los videojuegos, desde sus inicios se ha mantenido una estructura interna típica, que hoy en día sigue siendo utilizada. El eje central de esta estructura es su bucle principal, el cual se ejecuta una vez por cada imagen generada en pantalla.

Tras esta introducción al mundo de los videojuegos, este capítulo se ha centrado en el uso de motores en los mismos. En primer lugar se ha definido el concepto de motor dentro del ámbito de los videojuegos y, posteriormente, se ha realizado una clasificación de los mismos, detallando sus posibilidades de uso durante el proceso de producción.

La conclusión a la que se llega tras el estudio de estos motores, es que permiten la reutilización de un código robusto y de calidad que implementa ciertas funcionalidades, evitando así tener que programar las mismas repetidas veces. Obviamente, el uso de motores permite reducir los costes temporales de desarrollo de un videojuego, ya que para obtener toda su funcionalidad basta con incorporar el mismo al proyecto.

En el siguiente capítulo se va a exponer la tendencia actual de algunos estudios de desarrollo hacia la creación de juegos casuales online multijugador. Además, se estudiará la problemática existente durante la fase de producción de estos videojuegos y se planteará la creación de un motor de producción como una posible solución. El desarrollo de este motor será tratado con detalle en el capítulo 4.

Capítulo 3

Juegos casuales online multijugador

3.1 Introducción

El presente capítulo va a tratar de forma más detallada los juegos casuales online multijugador. En la sección 3.2 se expone cómo una pequeña empresa llega a plantearse este tipo de desarrollos y cómo al final llega a convertirse en un nuevo modelo de negocio en el que centrarse. A continuación, en la sección 3.3 se analiza la metodología que siguen estos pequeños estudios para llevar a cabo el desarrollo de un videojuego de estas características, mostrando cuáles son sus ventajas e inconvenientes. La sección 3.4 se analizan las herramientas y plataformas presentadas en el capítulo anterior para poder empezar con el desarrollo del juego. En la sección 3.5 se analizan los problemas globales que se encuentran durante dicho desarrollo. Por último, en la sección 3.6 se realizan las conclusiones correspondientes a este capítulo.

3.2 Un nuevo modelo de negocio

En los últimos años se ha visto incrementado el número de empresas desarrolladoras de software de entretenimiento. Esto es debido a la actual expansión que está experimentando la industria del videojuego, tal y como se ha podido observar en la sección 2.2.

Este tipo de empresas, como cualquier otra en sus inicios, están compuestas por un número reducido de trabajadores, muchos de ellos siendo programadores y diseñadores. Todas ellas, conocen de ante mano el mercado y la industria en la que

intentan hacerse un hueco, y es por ello que conocen muy bien los riesgos que ello conlleva, aunque también los grandes beneficios que pueden obtener.

Los inicios nunca son fáciles, por lo general, por lo que empezar con grandes proyectos con grandes inversiones no es lo habitual en estos casos. Los recursos de los que disponen son muy limitados, por lo que se suele optar por pequeñas producciones e ir subiendo de nivel según los resultados obtenidos, los beneficios conseguidos y las posibilidades de los propios estudios de desarrollo.

Estos pequeños proyectos abarcan desde los juegos casuales hasta los advergames, incluyendo los juegos por encargo. Son juegos que no necesitan de muchos recursos, la inversión inicial es menor que para las grandes producciones y el tiempo de producción es también menor, con lo que empleando menos tiempo, esfuerzo y dinero se pueden desarrollar muchos más proyectos.

Los datos vistos en la sección 2.2 donde se hacía referencia a la gran expansión del sector, el aumento de los juegos casuales y el alto beneficio que este tipo de juegos genera - 2,25 billones de dólares anuales – muestran por qué este tipo de aplicaciones se están convirtiendo en un nuevo modelo de negocio a seguir para estas pequeñas empresas.

Por otro lado, el impacto social que han tenido las redes sociales en los últimos años, ha propiciado que no solo se desarrollen este tipo de juegos sino que además se integren con las redes sociales ya existentes, llegándose a crear incluso nuevos portales de juegos casuales que han sabido incorporar las funcionalidades necesarias para dotarlos de un carácter social, semejante a las redes sociales ya existentes, creando así nuevas comunidades. Además, para hacer todavía más atractivos estos juegos, y fomentar todavía más el carácter social de estas páginas, se han dotado a los juegos casuales de un modo multijugador online, que permite a los usuarios competir entre ellos.

Gran prueba de ello son algunas páginas como *www.playray.es*, *www.gamedesire.com, www.miniclip.com*, *www.juegatu.com*, *www.shockwave.com, games.yahoo.com* o *www.cyberjuegos.com* mientras que en el ámbito nacional se pueden encontrar otras webs como *www.mundijuegos.com, www.ludoteka.com* o *www.opqa.com*. Todas ellas incluyen juegos casuales online multijugador y permiten a sus usuarios comunicarse entre ellos además de competir, subir fotos, hacer nuevos amigos y un sin fin de posibilidades.

3.3 Metodología de desarrollo

Debido a los pocos recursos de los que disponen este tipo de empresas, a los que ya se hacía referencia en el apartado anterior, a la hora de desarrollar un videojuego nuevo dichas empresas optan por seguir un modelo de producción ad-hoc. Esto significa que cada nuevo proyecto es desarrollado íntegramente desde cero, es decir, que se desarrolla de manera específica la mayor parte del código en función de las características y especificaciones del propio juego.

Este tipo de modelo de producción tiene sus beneficios y sus desventajas. En primer lugar este tipo de desarrollo permite centrarse en las funcionalidades específicas del videojuego, lo que significa que todo aquello que se codifique o desarrolle será utilizado para el proyecto en cuestión de forma única. Esto nos conduce a la obtención de proyectos de menor tamaño, debido a su carácter específico, y permite a los programadores centrarse en optimizar al máximo cada una de las funcionalidades desarrolladas.

Todo ello significa una serie de ventajas en cuanto al producto final, puesto que, para el caso de los juegos online, para el los cuales el tamaño resulta de gran importancia, se consigue no saturar la red y proporcionar así un buen servicio del juego.

Pero no todo son ventajas en este tipo de modelo de producción. El tener que codificar de nuevo cada una de las funcionalidades necesarias para cada uno de los juegos desarrollados puede resultar una tarea repetitiva y poco productiva. Hay una gran cantidad de funciones que pueden resultar de gran utilidad en desarrollos de futuros juegos. Es más, hay una gran cantidad de comportamientos y entidades que pueden ser reutilizados en el desarrollo de videojuegos cuyas características de juego sean similares. Una rutina que se encarga de barajar una baraja de cartas se puede utilizar en un sin fin de juegos de mesa. No resulta productivo tener que rehacer dicha rutina para cada juego de cartas nuevo.

Por otro lado, si surge cualquier tipo de problema en un determinado desarrollo, es posible que éste vuelva a surgir durante el desarrollo de otro proyecto similar, por lo que si un programador ya invirtió su debido tiempo en solucionar dicho problema tener que volver a parar la producción en solucionar el mismo problema para cada nuevo juego no resulta nada beneficioso para la empresa.

Por tanto, para conseguir evitar este tipo de producción y con el objetivo de abaratar los costes de producción tanto en recursos necesarios como en tiempo de

desarrollo, cada vez son más los estudios de desarrollo que invierten parte de su capital en desarrollar o adquirir herramientas que les permita desarrollar sus aplicaciones de una forma más rápida y eficiente, sin perder en ningún momento la calidad de sus productos, evitando así la generación de código siguiendo la metodología ad-hoc.

Estas herramientas a las que se está haciendo referencia son los motores y librerías introducidas en la sección 2.4.

3.3.1 Uso de motores y librerías en el desarrollo de un videojuego

La característica principal de los motores y librerías es su capacidad de abstracción de una serie de funcionalidades que pueden resultar comunes en diversos proyectos que presentan ciertas similitudes en su comportamiento y desarrollo.

Al igual que el modelo de producción ad-hoc, un desarrollo utilizando motores y librerías tiene también sus ventajas e inconvenientes.

En primer lugar, como se expone en el punto 2.4, el primer paso es decidir se va a hacer uso de un motor creado por tercero o si por el contrario se va a desarrollar un motor propio.

Trabajar desde un principio con un motor propietario, tanto si se trata de un motor de pago u *open source*, proporciona una herramienta sobre la que empezar a trabajar de calidad y robusta, la cual ha sido debidamente testeada. Por contra, puesto que se trata de soluciones muy genéricas es posible que muchas de las funcionalidades que el motor ofrece no sean utilizadas. Hay que tener especial cuidado con esto, porque al final puede resultar un proyecto de gran tamaño con una gran cantidad de líneas de código totalmente innecesarias que pueden influir en el funcionamiento final del juego.

Por contra, si se utiliza un motor propio, es posible enfatizar y optimizar al máximo aquellas funcionalidades que van a ser utilizadas, desarrollando solo aquellas que resulten de mayor interés para el desarrollo de una serie de juegos con características similares. Hay que poner especial atención en no desarrollar funciones específicas de cada uno de los juegos. Un motor contiene funciones que posteriormente serán utilizadas en múltiples proyectos. Otro problema que presentan los motores propios es el coste inicial que la empresa debe invertir en su desarrollo.

Una vez se ha obtenido el motor o motores que mejor se adaptan a los proyectos que se desean llevar a cabo, los beneficios que estos proporcionan serán cada vez mayores. El uso de estas herramientas durante la fase de producción de los videojuegos producidos por los estudios de desarrollo aporta los siguientes beneficios:

- *Aumento del margen de beneficio producido por cada uno de los juegos desarrollados.* La reducción de costes que permite el herramienta en cuestión permite a los estudios trabajar con equipos de desarrollo compuestos de un menor número de integrantes, lo que supone un ahorro económico a la empresa y un aumento en sus beneficios.

- *Permite a la empresa aceptar más encargos.* Son muchas las ocasiones en que otras empresas acuden a los estudios de desarrollo de ocio digital con el objetivo de encargar proyectos de todo tipo, que por regla general deben ser desarrollados en unos plazos de tiempo muy limitados. Reduciendo los costes de producción, en este caso, temporales, permite a las empresas aceptar más encargos, puesto que es capaz de realizar más aplicaciones de igual calidad, en un rango inferior de tiempo.

- *Oportunidad de negocio.* En determinadas ocasiones es necesario desarrollar un juego con una serie de características acorde con unos hitos que se van marcando o que ya han sido marcados, como son determinados eventos o noticias que resultan de gran impacto social. Este tipo de juegos deben ser desarrollados en el menor tiempo posible, para no perder la oportunidad que dicho hito proporciona. Poder desarrollar este tipo de proyectos en poco tiempo en el momento en que se produce el evento representa una gran oportunidad de negocio para publicar tus productos antes que la competencia. Ser el primero, como en muchos otros sectores, es un elemento clave para obtener ventaja sobre los competidores.

3.4 Desarrollando un videojuego casual online multijugador

A la hora de adentrarse en un nuevo proyecto es importante tener en cuenta una serie de cuestiones importantes, cuyos resultados se verán reflejados en la finalización de este, así como en futuros proyectos. Las primeras dudas que surgen a las empresas son: qué lenguaje utilizar y sobre qué plataforma van a funcionar sus desarrollos. Otra pregunta que responder es si va a resultar beneficiario el uso de motores y si merece la pena la creación de uno propio.

Como se expuso en la sección 2.2, debido a que gran parte de las plataformas existentes hoy en día son de carácter propietario y al gran número de hogares que poseen ordenadores además de una conexión de banda ancha, se llegaba a la conclusión de que la plataforma online ejecutada sobre un ordenador resulta la más adecuada para que una empresa pequeña empiece con sus proyectos, gracias a la gran accesibilidad de esta plataforma.

Una vez escogida la plataforma, el siguiente paso es la tecnología que se va a emplear. Después del estudio realizado en la sección 2.2 y en función del tipo de desarrollo que se quiera llevar a cabo resulta muy sencillo seleccionar la tecnología que mejor se adapte al proyecto que se desea desarrollar.

En primer lugar si se desea una tecnología que permita el mayor rango de posibilidades, es decir, que pueda abarcar el más amplio campo de tipos de juegos, se debe descartar aquellas tecnologías que no soportan 3D. Una característica que acota bastante el número de tecnologías posibles a utilizar.

En segundo lugar y puesto que el producto final está dirigido a un público inexperto, es conveniente facilitar al máximo a los usuarios destinatarios el uso de dicho producto. Evitar instalaciones, sobre todo si son en la propia máquina del usuario, resulta beneficios, puesto que dichos usuarios no requieren de amplios conocimientos informáticos. Además, muchos usuarios desconfían y son reacios a las instalaciones de programas desconocidos en sus ordenadores. Por tanto, una tecnología que permita ser ejecutada directamente sobre el navegador web resulta la más conveniente.

Llegados a este punto, de las tecnologías vistas en la sección 2.2, las que cumplen con estos dos requisitos son Shockwave y Unity. Para realizar el último descarte se va a analizar el nivel de penetración de ambas tecnologías. Aunque la instalación del plugin de Unity resulte más rápida frente a la instalación del plugin de Shockwave, su nivel de penetración es mucho menor. El plugin de Unity, actualmente, está instalado en apenas el 1% de los navegadores, mientras que el de Shockwave se encuentra en el 50%. Otra posible razón para descartar Unity es que los desarrolladores solo pueden trabajar sobre Mac. Aunque hoy en día, son cada vez más los usuarios de este tipo de ordenadores, resulta una inversión mayor para los inicios de una empresa.

La tecnología que, en este caso, resulta más adecuada para el desarrollo de videojuegos casuales online 3D multijugador es Shockwave. Para su implementación es necesaria la herramienta de autor Adobe Director. Se trata de una herramienta que

permite a los desarrolladores crear un amplio abanico de contenidos multimedia, los cuales utilizan la tecnología Adobe Shockwave Player para su reproducción.

Para facilitar el desarrollo de los diversos contenidos, Adobe Director ofrece una serie de librerías y motores, conocidos en este ámbito como Xtras. Algunos de ellos permiten trabajar con distintas funciones en diversos campos como son: tratado de ficheros XML, 3D y render, animaciones, audio, simulación física y detección de colisiones y comunicación en red. Para mayor información puede consultar [39, 42, 50].

3.5 Problemas en el desarrollo

El uso de estos Xtras pueden resultar de gran ayuda a la hora de desarrollar cualquier proyecto, debido a las ventajas que ofrecen como motores o librerías, pero también se han detectado algunos inconvenientes en el uso e integración de dichos Xtras, así como algunos aspectos mejorables en el uso de motores bajo esta herramienta, que pueden resultar de gran ayuda en proyectos futuros.

3.5.1 Integración de los Xtras en los proyectos

Al inicio de cada nuevo proyecto, antes de empezar con la programación de la aplicación y si las herramientas y tecnologías que se van a utilizar son nuevas para el equipo de desarrollo, es necesario realizar un estudio previo de estas. Como cualquier inicio no es fácil y en muchas ocasiones, sobre todo si las herramientas que se van a utilizar no son de carácter privado, pueden aparecer problemas inesperados.

La integración de los distintos Xtras que proporciona la herramienta de autor Adobe Director se realiza siguiendo una metodología ad-hoc, por lo que es necesario realizar dicha tarea cada vez que se desee utilizar cada uno de estos Xtras. Además, dada la larga duración de los proyectos y a la cantidad de Xtras de los que se dispone es posible que sea necesario un estudio reiterativo de cómo realizar dicha integración. Este estudio es una tarea tediosa y costosa, desde un punto de vista temporal, y debe realizarse el menor número de veces posibles.

Pero no siempre es posible reducir el número de veces que se realiza el estudio de cada Xtra, porque aunque la filosofía de los motores es la de ofrecer robustez y

fiabilidad en su uso, en ocasiones es posible observar una serie de inconsistencias en el propio uso del Xtra, los cuales pueden ocasionar un comportamiento inesperado del propio Xtra y por consiguiente, en el resultado final del proyecto desarrollado.

Aunque la empresa desarrolladora suele informar de estos errores a los desarrolladores de los Xtras o a los de la propia herramienta de desarrollo, es necesaria una solución rápida, la cual ellos no van a proporcionar normalmente a corto plazo, para evitar que el proyecto se pare. Debido al carácter propietario que los Xtras poseen el desarrollador no puede acceder al código interno del propio Xtra para poder solucionar los problemas encontrados. Es por ello que para poder solucionar las inconsistencias que ocasionan los errores, los desarrolladores a menudo deben realizar un estudio más exhaustivo del Xtra que provoca los fallos o incluso deben realizar tareas extras implementando *work-arounds* que ayuden a solucionar los problemas, siempre sin modificar el comportamiento del propio Xtra.

Por tanto, si un desarrollador o equipo de desarrollo ha realizado ya dicho estudio y ha solucionado las posibles inconsistencias que se han presentado durante el desarrollo, desde un punto de vista tanto práctico como económico, no es eficiente ni productivo tener que realizar dichas tareas para todos y cada uno de los nuevos proyectos que se deseen llevar a cabo. Tampoco resulta óptimo si además es otro equipo de desarrollo el encargado de relevar el trabajo del anterior equipo que sí realizó dicho estudio.

Por todo esto, se puede concluir que, es conveniente disponer de un software que integre todas las funcionalidades que implementan las herramientas vistas anteriormente, el cuál haya sido correctamente testeado, y que actúe como motor único, de modo que el desarrollador de la aplicación tan solo deba conocer cómo utilizar dicho software, sin necesidad de realizar un estudio previo de todas las herramientas que este incluye. Además, debe proponerse una solución de forma genérica, de modo que el nuevo código implementado pueda ser reutilizado en futuros proyectos, sin necesidad de que los desarrolladores tengan que solucionar repetidamente aquellos errores a los que ya dedicaron su debido tiempo y esfuerzo.

Dicho software que actúa como una capa de abstracción mayor encapsulando los distintos motores y librerías que pueden ser utilizados en el desarrollo de una aplicación concreta al que se hace mención en el párrafo anterior, es conocido como *Middleware*. Para más información sobre este tipo de problemas consultar [1].

3.5.2 Configuración externa

A lo largo del proceso de producción de un videojuego existen gran cantidad de valores y datos que deben ser configurados, algunos de ellos incluso modificados en más de una ocasión en varias de las fases de las que se compone el ciclo de desarrollo de un juego.

Para cada una de estas modificaciones es necesario que el desarrollador entre en el propio código del juego y busque qué variable es la que contiene dicho valor o valores y modificarlos. Esto puede resultar una tarea tediosa si son muchos los valores que se deben modificar o si son personas distintas las que deben realizar dicha tarea. La posibilidad de modificar estos valores de forma independiente al propio código del juego, e incluso por usuarios que carecen de conocimientos técnicos, resulta de gran utilidad, tanto en la configuración inicial de los datos como en la fase de testeo, reduciendo así el tiempo necesario para realizar los cambios y pruebas pertinentes.

Por otro lado, puede resultar de gran utilidad poder configurar qué tareas, scripts o ficheros han de ser utilizados por la aplicación en función de las necesidades del proyecto en que se esté trabajando. De este modo, se puede trabajar de forma modular, creando distintos scripts o cualquier otro tipo de ficheros que se desee incluir y mediante el entorno de configuración externa decidir cuáles de ellos van a ser utilizados en dicho proyecto. Además de ello, se puede especificar qué tareas se deben llevar a cabo durante la ejecución de la aplicación, y cuáles no.

En el campo de la Realidad Virtual y la Realidad Aumentada se pueden observar algunos ejemplos de este tipo de configuraciones.

Uno de estos ejemplos es SUED (System for de development of Unexpensive and Extensible Distributed Virtual Environment systems). Como su nombre indica, SUED fue creado para agilizar y abaratar el desarrollo de entornos virtuales distribuidos. Según [26] un entorno virtual distribuido, se pude definir como un sistema en tiempo real que permite a múltiples usuarios conectados desde diferentes ordenadores interactuar en un mundo virtual 3D común. SUED ofrece dos características importantes. La primera de ellas es permitir extender las capacidades básicas que ya ofrece el propio *framework*, y por otro lado, permitir incluir algunas tareas que resultan de carácter común en el desarrollo de este tipo de sistemas. De este modo, no solo se reduce el tiempo de desarrollo del DVE, sino que se reduce también la complejidad de la propia programación cuando estos sistemas se utilizan para simulaciones de escenarios más complejos, todo ello gracias al entorno de configuración externa basado en ficheros XML.

En [26] se plantea como punto criticable a Delta3D, otro software de carácter similar, el hecho de que éste no incorpore un método de configuración externa y accesible para usuarios que carecen de conocimientos técnicos. Ésta es la principal razón por la que SUED incorpora Xerces [25]. Gracias a esta herramienta toda la configuración de este *framework* se realiza a través de un fichero XML externo, obteniendo como resultado una reducción de tiempo y de costes, tanto en la fase de desarrollo como en la fase de evaluación del sistema, pudiendo realizar cualquier modificación sin necesidad de recompilar de nuevo el proyecto.

En [26] y en [27] se analizan con detalle SUED y Delta3D respectivamente. En ambos casos se evalúan las correspondientes herramientas a través de diversos experimentos y se concluye que el uso de las mismas permite a los desarrolladores obtener una base sobre la cual desarrollar sus aplicaciones reduciendo de manera notable los costes de producción.

Por todo esto, se puede concluir que, la posibilidad de realizar una configuración de manera externa y a través de un entrono accesible para usuarios sin conocimientos técnicos resulta de gran utilidad a la hora de realizar pequeños cambios y pruebas, sin necesidad de adentrarse en el propio código del juego. Además permite configurar distintas tareas, scripts e incluso cualquier tipo de ficheros que se desee utilizar o incluir en el proyecto final, realizando todas estas tareas de una forma rápida y sencilla.

3.5.3 Abstracción de funciones de inicialización y finalización de un videojuego

Desde que se empezaron a desarrollar los primeros videojuegos de las historia hasta llegar a los juegos de última generación han sido muchos los cambios y evoluciones de los que esta industria ha sido testigo. Sin embargo algunos aspectos tan básicos como la estructura interna seguida durante la ejecución de uno de estos programa sigue siendo la misma.

Como bien se explica en la sección 2.3 todos los videojuegos siguen una estructura de ejecución basada en eventos que se producen de forma secuencial empezando por la inicialización del programa principal y terminado por la finalización de aquellas datos creados e inicializados al principio de la ejecución.

Durante la inicialización del programa principal se crean todas las estructuras de datos internas y se configuran e inicializan en función de una serie de parámetros. Una vez creadas todas las entidades de juego es cuando se procede a la ejecución del bucle principal del juego. Una vez finalizado este, es necesario destruir todas las estructuras de datos creadas al inicio con el fin de liberar recursos.

Esta serie de tareas y ejecuciones se deben realizar para todo juego y siempre en el mismo orden.

Dada la naturaleza de los motores, los cuales se encargan de extraer funcionalidades comunes y encapsularlas de manera que sean reutilizables en distintos proyectos, se puede llegar a la conclusión de que todas estas funciones de inicialización, llamadas al bucle principal del juego y finalización, que resultan siempre las mismas para cada juego nuevo, pueden englobarse en un motor que gestione de la mejor forma posible los eventos que se ejecutan durante estas fases.

De este modo se consigue trasladar las ventajas que proporciona el uso de los motores durante la fase de producción de un videojuego a la inicialización y finalización de este.

3.6 Conclusiones

Este capítulo se ha querido centrar en las pequeñas empresas desarrolladores de videojuegos, las cuales han visto en los juegos casuales online multijugador un nuevo modelo de negocio que las ayude a adentrarse en el sector del ocio interactivo digital.

Por otro lado, se ha tratado el tipo de metodología que usan este tipo de empresas durante el desarrollo de sus proyectos. Es uso de una sistema ad-hoc tiene sus ventajas pero también conlleva una serie de inconvenientes. Es por ello que se plantea otro tipo de metodologías, basándose en el uso de motores y librerías que ayuden en la desarrollo viendo reduciendo así los costes de producción.

Este cambio de metodología pasa por la creación o adquisición de estos motores, por lo antes de invertir en este tipo de soluciones es necesario decir sobre qué plataforma y lenguaje se desea trabajar.

Tras realizar un estudio de las distintas plataformas y tecnologías expuestas en el capítulo 2, se decide empezar a trabajar sobre la plataforma online ejecutada sobre ordenador y con la tecnología Shockwave junto con la herramienta de autor Adobe Director. Pero el uso de esta herramienta y los motores que incorpora presentan también una serie de inconvenientes que deben ser resueltos:

- Debido a la integración ad-hoc de los motores y librerías, el desarrollador debe realizar un estudio reiterativo de estos, lo que conlleva a tener que solucionar de un modo específico los errores o inconsistencias que puedan presentar los propios motores en cada nuevo desarrollo.
- Se ha detectado la carencia de un entorno de configuración externo que agilice las funciones de inicialización, desarrollo y testeo, mediante la configuración de los datos, tareas y scripts o ficheros que van a ser utilizados durante el desarrollo y funcionamiento de la aplicación, ralentiza y aumenta la complejidad del desarrollo del proyecto.
- Atendiendo a la estructura interna de los videojuegos se observa la inexistencia de un motor que realice las pertinentes funciones de inicialización y finalización del programa, así como las sucesivas llamadas al bucle principal del juego de una forma secuencial y estructurada.

Para poder solucionar los problemas descritos se ha propuesto el desarrollo de un motor de producción. Será en el capítulo 4 donde se muestre con detalle la estructura que sigue y las funcionalidades que ofrece.

Posteriormente, en el capítulo 5, con el objetivo de mostrar el potencial del motor de producción desarrollado se presenta un videojuego de tipo casual online multijugador en 3D que ha sido desarrollado íntegramente desde cero utilizando como base en su desarrollo del motor de producción presentado.

Por último, antes de llegar a las conclusiones generales, en el capítulo 6 se realiza un estudio de los costes de producción resultantes del desarrollo de un videojuego haciendo uso del motor de producción frente a otro videojuego de características similares cuyo desarrollo fue anterior al motor, además del coste de producción del propio motor.

Capítulo 4

Desarrollo del motor de producción

4.1 Introducción

En el siguiente capítulo se realiza un estudio del motor de producción desarrollado como solución a la problemática que se encuentra a la hora de llevar a cabo el desarrollo de un videojuego casual online multijugador en 3D sobre la herramienta de autor Adobe Director expuesta en el capítulo anterior.

En la sección 4.2 se describe a grandes rasgos el motor de producción desarrollado. Cada una de sus partes son el resultado de un desarrollo basado en una filosofía de diseño que se compone de tres pilares fundamentales. En las siguientes secciones se trata con mayor detalle cada una de las partes que constituyen el motor de producción.

En la sección 4.3 se muestra como el motor de producción permite realizar la configuración de algunos de los parámetros del juego de un modo externo al propio motor. En la sección 4.4 se centra en la automatización de los procesos de inicialización y finalización de un videojuego mediante el uso del motor de inicialización desarrollado. A continuación, en la sección 4.5 se describen algunas funcionalidades que se han incluido en el motor de producción que resultan de gran interés para el desarrollo de este tipo de juegos y que, debido a sus características, no se han podido incluir en las sección anteriores. Y por último, en la sección 4.6 se realizan las conclusiones correspondientes a este capítulo.

4.2 Filosofía de diseño del motor de producción

A lo largo del anterior capítulo se han podido observar una serie de problemas e inconvenientes que las pequeñas empresas desarrolladoras deben hacer frente a la hora de llevar a cabo un videojuego casual online multijugador durante su fase de producción. Con el objetivo de paliar, en medida de lo posible, dichos problemas y mejorar por tanto costes del desarrollo de este tipo de juegos, se propone como solución el desarrollo de un motor de producción.

El motor de producción ha sido desarrollado siguiendo una filosofía de diseño basada en tres principios clave, dando lugar a cada una de las partes que forman el motor de producción final:

1. Configuración externa. Capacidad de configurar ciertos parámetros que constituyen el juego mediante un fichero externo al motor de producción. Dicho fichero debe ser lo suficientemente sencillo, tanto de entender como de modificar, para que cualquier persona, ajena al código interno del motor o del juego, sea capaz de modificar cada uno de los parámetros permitidos.

2. Motor de inicialización. Capacidad de comportarse como un motor de inicialización de juegos. Debe extraer aquellas funcionalidades que resulten comunes, siempre desde el punto de vista de la estructura de ejecución interna de un videojuego, y automatizar, en medida de lo posible, dichas funcionalidades encapsulándolas en un nuevo motor de creación propia.

3. Middleware. Capacidad de encapsular los distintos motores que ofrece la herramienta de autor Adobe Director creando una capa de abstracción a un nivel mayor que facilite a los programadores el uso de cada uno los motores integrados.

Concretamente esta tesina se va a centrar en aquellos procesos que corresponden a la configuración externa de datos, así como a la precarga y posterior inicialización de los mismos para el correcto funcionamiento del juego. En [1] se encuentra detallada la funcionalidad de *Middleware* que presenta el motor de producción, así como la inicialización de la interfaz de juego, la física general del mundo y la de los objetos, incluyendo el sistema de detección y resolución de colisiones.

4.3 Configuración externa

Una de las claves principales del motor de producción es la posibilidad de configurar de forma externa múltiples características y propiedades de un videojuego, todo ello de una forma fácil y sencilla, a través de un modelo de aplicación.

Por modelo de aplicación se entiende un fichero de texto en el que se define el universo representado (estado de los objetos, propiedades físicas del mundo, etc.) de una forma estructurada. Se ha elegido el estándar XML como formato del modelo de aplicación, debido a que ofrece una característica clave, la estructuración, que permite de una forma muy sencilla crear dicho documento siguiendo unas pautas definidas y que se irán exponiendo poco a poco a lo largo de esta sección.

La creación de una librería, combinada con el Xtra de XML que incorpora Director, permite traducir los datos definidos en el modelo de aplicación a una lista de propiedades, mucho más accesible y fácil de utilizar para los programadores. Este proceso es conocido como *parsear* el modelo de aplicación, y la librería desarrollada en combinación con el Xtra de XML se conoce como *parser*. La estructura de datos que aquí se obtiene es una lista muy sencilla, con todos los datos necesarios para el funcionamiento del videojuego, que deberá ser procesada e interpretada posteriormente durante el proceso de inicialización.

A continuación se va a explicar el proceso seguido durante el *parseado* del modelo de aplicación.

Puesto que el modelo de aplicación se trata de un documento en formato XML, se hace uso de la librería de Director para realizar el primer paso, el cual consiste en comprobar si el contenido del XML cumple los requisitos de estructuración, o lo que es lo mismo, comprobar si dicho documente está bien formado. Esto significa que para toda etiqueta abierta, debe existir una de cierre, además de seguir una estructura en árbol coherente. En caso de encontrar un error, el Xtra lo propaga y el motor de producción lo captura y muestra el mensaje pertinente que indica dónde se ha producido el error léxico.

Si el contenido es correcto, el Xtra devuelve como resultado una lista de propiedades con todos los datos leídos del fichero XML. Esta estructura resulta un tanto tediosa a la hora de trabajar con ella. Además, para que la posterior ejecución del videojuego sea correcta, es necesario comprobar que se han definido una serie de etiquetas concretas en un determinado orden. Esto facilita después el uso de la lista de

propiedades final, puesto que al conservar la estructura definida, se consigue de una forma más eficiente el acceso a los datos de dicha lista.

El proceso empleado para definir la estructura del documento ha sido, definir una lista con todas la etiquetas que pueden ser incluidas en el modelo de aplicación. Cada etiqueta definida puede contener a su vez otra lista con las etiquetas hijas que esta permite. Por tanto, de forma recursiva se accede a la lista proporcionada por el Xtra de Director y se comprueban si los datos resultantes siguen el orden previamente determinado. Para obtener la lista definitiva se hace uso de una lista auxiliar, donde se van guardando la etiqueta procesada y su padre y se comprueba que el parentesco entre ambas etiquetas esté permitido. En la siguiente iteración la etiqueta que se había procesado en la iteración anterior será el padre de la nueva etiqueta que se va a procesar. Al llegar a un nodo hoja, si todo ha sido correcto, el sub-árbol se guarda en la lista de propiedades final.

Este proceso se realiza dos veces, puesto que se han definido dos modelos de aplicación. El primero de ellos trata características generales del juego. Mientras que el segundo define propiedades de los distintos niveles o modalidades que puede incorporar el juego. Por tanto, para cada uno de ellos se debe realizar proceso descrito.

Por lo tanto, el resultado final de todo el proceso son dos listas de propiedades, una para los datos definidos en el modelo de aplicación general y otra para los definidos en el modelo de aplicación de nivel.

Durante este proceso se pueden producir tres tipos de errores. El primero de ellos es que la etiqueta procesada no se permita, es decir, que dicha etiqueta no haya sido definida en la lista de etiquetas válidas. Por otro lado, puede que se detecte que para dos etiquetas analizadas exista una relación de parentesco no permitida. Por último, para evitar fallos posteriores no se admiten etiquetas con valores vacíos, por lo que si se encuentra una definición de este tipo también se producirá un error. Al igual que con los errores léxicos, si se produce un error de alguno de estos tres tipos, se parará la ejecución de la aplicación y se mostrará el correspondiente cuadro de diálogo con la información del error producido, indicando la etiqueta que lo ocasionó.

Algunas de las etiquetas definidas en la lista antes mencionada son de carácter obligatorio, por lo que es necesario incluirlas en el modelo de aplicación. El resto son de carácter opcional y en caso de no incluirlos el funcionamiento de la aplicación seguirá siendo correcto. Por tanto, antes de pasar al siguiente punto, donde se inicializan todos los datos y estructuras necesarios para la posterior ejecución de la aplicación, hay que

realizar una última comprobación que avise en forma de error si alguna de las etiquetas obligatorias no ha sido definida.

Las etiquetas que resultan esenciales para el correcto funcionamiento del juego son aquellas que definen la escena principal sobre la que se va a desarrollar el videojuego, el script de juego que manejará el bucle principal, el motor de física que se va a incorporar y sus scripts correspondientes y, por último, los scripts correspondientes a las entidades definidas.

Tal y como ya se ha expuesto, se definen dos modelos de aplicación, uno de carácter general y el otro de carácter más específico, o de niveles. Como sus nombres bien indican, el modelo de aplicación general definirá aspecto genéricos de la aplicación, mientras que el modelo de aplicación de niveles, estará formado por las etiquetas de contienen aspectos específicos de cada nivel.

Entrando un poco más en detalle, en el modelo de aplicación general se encuentran las siguientes etiquetas:

Nota: para una mayor compresión de las etiquetas que se muestran a continuación, se recomienda al lector consultar el Anexo A.

<*config*>: etiqueta raíz del documento.

<*game*>: etiqueta de nivel superior que contiene algunas características propias del juego. Dentro de esta etiqueta hay otras cuatro, una de ellas de carácter obligatorio. Esta es la etiqueta <*script*> la cual define el script principal del juego. Las otras tres etiquetas son <*name*>, <*initFunctions*> y <*preloadFunctions*>. La primera de ellas contiene el nombre del juego, mientras que las otras dos, permiten definir una lista de funciones que serán ejecutadas sólo para un juego en concreto.

<*dir*>: etiqueta que contiene, mediante un conjunto de etiquetas hijas, la ubicación de las diferentes entidades y de los scripts de juego que deben ser cargados. Por ejemplo, para definir la ubicación de los scripts que van a definir el comportamiento de los objetos del juego, se define la etiqueta <*objects*>, la cual contiene dicha ubicación. Otras etiquetas utilizadas han sido: <*scene*>, <*overlays*>, <*rules*> o <*game*> para definir dónde podrá el motor encontrar, los ficheros w3d, los overlays y sus scripts, las reglas del juego y el propio script de juego, respectivamente. Además se ha incluido una etiqueta <*default*> para

aquellos scripts que no tengan un grupo definido, como son el script de la interfaz o la física. Su ubicación será la indicada por dicha etiqueta.

<*script*>: etiqueta que indica qué scripts, ubicados según se indica en la etiqueta <*dir*>, han de ser cargados para el correcto funcionamiento de la aplicación. Por ejemplo, para definir el script de comportamiento de un objeto, se utilizará la siguiente estructura dentro de esta etiqueta: <*object*> <*script*> nombre_del_script </*script*> </*object*>. Se incluirá una etiqueta script por cada comportamiento que se vaya a utilizar, aunque posteriormente se pueda compartir dicho comportamientos por varias de las entidades definidas.

Todos los scripts de comportamiento que se hayan asignado a las distintas entidades, que se definirán posteriormente en el documento, deben estar aquí incluidos.

<*globals*>: etiqueta que define aspectos globales de la escena. Están incluidas las etiquetas <*physics*> y <*animations*>. En la etiqueta de física se encuentran definidos tanto el motor de física que se va a utilizar y su script correspondiente, así como cualquiera de sus propiedades configurables. De estas son obligatorias las etiquetas correspondientes a la definición del motor y del script, <*engine*> y <*script*> respectivamente. El resto de etiquetas que definen las propiedades del mundo físico vienen definidas con el mismo nombre de cada una de las propiedades y variarán en función del motor utilizado. El *parser* está preparado para soportar tanto Havok como PhysX. La etiqueta <*animations*> indica si existen animaciones que deban ser cargadas y clonadas. Esta acción se llevará a cabo en la fase de inicialización.

<*specific*>: etiqueta que define objetos concretos. Es aquí donde vienen definidas las animaciones y escenas 3D independientes de la escena principal, los objetos que no van a tener una física definida y los que sí la tendrán.

<*w3d*>: etiqueta que contiene las animaciones. Dichas animaciones vienen definidas, cada una de ellas por su nombre, sin la extensión, en una etiqueta <*object*>. Al igual que ocurre con los scripts, se define una etiqueta <*object*> por animación, las cuales podrán ser asignadas posteriormente a uno o varios objetos.

<*objects*>: etiqueta que contiene todos los objetos que no van a tener un cuerpo físico asociado. Existe una etiqueta por grupo de objetos, es decir, si se define una etiqueta *estrella*, aquellas propiedades definidas en dicha

etiqueta se asignarán a todos los modelos de la escena principal que contengan en su nombre la palabra *estrella*, es decir, el nombre de la etiqueta. En esta lista de objetos solo se definirán aquellos modelos que tengan alguna propiedad importante, como un script de comportamiento (definido por la etiqueta <*script*> como hija) o alguna característica especial (definida por la etiqueta <*specialFeatures*>).

<*rigidObjects*>: etiqueta que contiene aquellos objetos que tendrán un cuerpo físico asociado. Este tipo de etiquetas sigue la misma filosofía de las nomenclaturas que la anterior, además de permitir las mismas etiquetas <*script*> y <*specialFeatures*>. Como complemento disponen de dos grupos de etiquetas más. La primera de ellas, <*rigidBody*> contiene las características propias del cuerpo rígido que se va crear para los modelos definidos. Dentro de <*rigidBody*> las etiquetas <*shape*> y <*type*> definen la forma y el tipo del cuerpo rígido respectivamente. Por otro lado, se cuenta con la etiqueta <*physics*>, que contendrá las propiedades físicas del modelo. Puesto que estas propiedades son muchas se ha definido una etiqueta <*default*> que permite asignar a otros objetos las propiedades físicas de uno ya definido.

<*interface*>: etiqueta que define los distintos componentes que componen el interfaz del juego. Estos componentes son principalmente gráficos 2D y textos que pueden o no declararse como *overlays*. Los tipos de *overlays* que se pueden crear son: *overlays* creados a partir de una imagen (etiqueta <*image*>), creados a partir de un conjunto de imágenes (etiqueta <*multImages*>), *overlays* que solo contengan texto estático (etiqueta <*staticText*>) o bien texto dinámico (etiqueta <*dynamicText*>), además de combinaciones de todas ellas, como por ejemplo, *overlays* que estén formados por imágenes y a su vez contengan texto.

Para definir correctamente un *overlay*, es necesario incluir también las etiquetas <*id*>, <*name*> y <*coordinates*>, las cuales definen, respectivamente, el identificador que se utilizará para trabajar con el dentro del motor, el nombre que recibirá el *overlay* y sus coordenadas en la escena.

En caso de tener un comportamiento definido, será necesario indicarlo, al igual que en los objetos, mediante la etiqueta <*script*>.

En cuanto a la definición de gráficos 2D, es necesario definir la etiqueta <*createOverlay*> a falso, en caso contrario se creará el overlay correspondiente

a la definición dada. De este modo el motor importará el gráfico correspondiente al proyecto sin crear ningún *overlay.*

Respecto al modelo de aplicación de nivel, este contiene definidos elementos más específicos en función de cada nivel. En este documento se incluirán todos los niveles que se vayan a incorporar posteriormente en el juego.

Como etiquetas obligatorias contiene <*scene*>, etiqueta que contiene el nombre del fichero w3d que contiene la escena principal del juego y <*rules*>, la cual define el script que contiene las reglas generales del juego.

Cada nivel viene definido por una etiqueta <*level*>. El único requisito que deben cumplir este tipo de etiquetas es contener como hija una etiqueta <*id*>, en la que se especifica el identificador del nivel. Este será utilizado posteriormente para identificar, dentro de la escena 3D, los objetos pertenecientes a cada nivel que aquí se ha definido. Dichos objetos siguen una nomenclatura específica para identificarlos como pertenecientes a un nivel, utilizando como prefijo en su nombre dicho identificador seguido de un guión bajo, *01_*. Aquellos objetos dentro de la escena que no presenten como prefijo ninguno de los niveles definidos en el modelo de aplicación de nivel, se considerarán elementos comunes a todos los niveles.

La filosofía base que sigue este documento es la de redefinir las propiedades ya definidas en el modelo de aplicación general, adaptándolas en función de los requisitos de cada nivel.

Para ello se utiliza, en cada nivel, una etiqueta <*redefinitions*>. En ella se incluye cualquier cambio que se desee realizar. En el modelo de aplicación general, a la hora de definir las propiedades de una entidad, se define una etiqueta con un nombre genérico, por ejemplo *terreno,* de forma que todas las entidades definidas en la escena 3D como *terreno,* tendrán las mismas propiedades físicas. Realizar una redefinición en el modelo de aplicación de nivel puede llevarse a cabo de tres modos diferentes. El primero de ellos consiste en redefinir la misma etiqueta para el nivel o niveles deseados. Siguiendo el ejemplo anterior, bastaría con incluir dentro la etiqueta <*redefinitions*> del nivel la etiqueta que hace referencia a la entidad que se desea modificar, en este caso *terreno,* y a continuación, siguiendo la estructura presentada en el modelo de aplicación general se haría la redefinición. El segundo consiste en redefinir un subgrupo determinado de elementos, en lugar de redefinir todo el conjunto de entidades a que hace referencia una etiqueta, por ejemplo, *terrenoArena*. Por último, el tercer caso consiste en modificar una entidad en concreto. Para llevar a cabo dicha redefinición, el

nombre de la etiqueta debe ser exactamente igual que el de la entidad que se desea redefinir, sin el prefijo del nivel al que pertenece.

Cualquier propiedad definida en el modelo de aplicación general se puede modificar en el modelo de aplicación de nivel siguiendo la misma estructura que se emplea en el primero.

En el modelo de aplicación de nivel se pueden incluir dos tipos de scripts de reglas. El primero de ellos es el script que define las reglas generales del juego. Este viene definido por la etiqueta <*reglas*>. El segundo es el que contiene las reglas específicas para un cierto nivel. Estas reglas pueden variar para cada nivel, tomando las generales como base, o pueden compartirse entre niveles.

Para obtener más información acerca de las propiedades de las entidades que se pueden definir, así como de las propiedades del motor de física, se aconseja consultar [28].

4.4 Motor de inicialización

Siguiendo la estructura principal sobre la que se basa el desarrollo del motor de producción, el motor de inicialización que se plantea como segundo pilar en la filosofía de diseño presentada anteriormente, tiene como objetivo principal extraer y encapsular en un único motor aquellas funcionalidades que resultan comunes atendiendo a la estructura básica de un videojuego y a su fase de inicialización, para su posterior automatización.

Tal y como se explica en la sección 2.3, la estructura de un videojuego ha cambiado poco desde sus inicios. Por otro lado, se observan una serie de funcionalidades que resultan comunes tanto en el desarrollo como en la ejecución de los mismos. Dichas funciones corresponden a la inicialización y finalización del programa principal y a las múltiples llamadas que se realizan al bucle principal del juego. La ejecución de dicho bucle queda al margen del motor de inicialización, puesto que es propio de cada juego y debe ser desarrollado de forma independiente en base a las especificaciones y requerimientos del propio juego.

Por tanto, las funcionalidades que implementa el motor de inicialización, integrado en el motor de producción desarrollado, que corresponden con la fase inicialización del programa principal son las siguientes:

- Carga externa de datos.
- Precarga de datos, incluyendo la inicialización de la estructura de datos final.
- Inicialización de la interfaz.
- Inicialización del motor de física.
- Inicialización de las reglas.
- Inicialización del motor de colisiones.
- Inicialización de animaciones.
- Inicialización del juego.

Los procesos listados se ejecutan de una forma automática y transparente al programador. Una vez finalizado todo el proceso de inicialización se obtiene como resultado un estructura sólida y completamente preparada para su posterior uso. Además todos los scripts han sido correctamente referenciados e inicializados, con lo que se asegura el correcto funcionamiento del posterior bucle del juego, el cual llegados a este punto pasa a ser ejecutado.

Gracias a la automatización realizada por el motor de inicialización los programadores solo deben dedicar su tiempo y esfuerzo al desarrollo de los scripts propios del juego, como son el bucle principal, las reglas del juego o el comportamiento que pueden seguir algunos de los objetos con los que posteriormente interactuará el futuro usuario, dejando el trabajo de la inicialización de todos los elementos y estructuras al propio motor.

Dado el carácter de esta tesina, la cual se centra principalmente en las tareas de cargar y configurar los datos y entidades necesarios para el correcto funcionamiento del juego, se va a hacer hincapié en las funciones de carga externa de datos, precarga de datos e inicialización tanto de las animaciones como del propio juego.

A modo de resumen, la inicialización de la interfaz corresponde con la preparación de aquellos elementos gráficos que corresponden con la interfaz de usuario, es decir, imágenes que aportan información sobre el juego e imágenes con las que el usuario puede interactuar, como pueden ser los botones.

Los motores de física y de detección y resolución de colisiones son los encargados de generar el mundo físico y resolver cualquier actividad física que se produzca de la mejor manera posible. Por ello se deben inicializar todos los parámetros correspondientes a dicho mundo físico y se deben preparar todos los objetos que forman parte del escenario que posteriormente van a poseer propiedades físicas. Además es

necesario registrar qué colisiones resultan de interés al juego y cómo estas van a ser resueltas en el momento en que se produzcan.

Por último, la inicialización de las reglas hace referencia a la inicialización del script que contiene las pautas que van a seguir los jugadores durante el juego. En algunas ocasiones puede ocurrir que existan diferentes scripts de reglas en función de los niveles de juego y sus características. Es en este estado donde se crean las relaciones entre los distintos scripts de reglas que puedan existir y se inicializan según corresponda.

La inicialización de la interfaz, el motor de física, la inicialización de las reglas y la inicialización del motor de detección y resolución de colisiones se describen con mayor profundidad en [1].

4.4.1 Carga externa de datos

Puesto que el motor de producción permite cargar distintos juegos con diferentes características, el primer paso que se tiene que llevar a cabo es la importación de todos los scripts que se utilizan, así como las animaciones, escenas 3D, imágenes y ficheros de audio que se vayan a utilizar en función de cada juego.

En primer lugar, se importan y precargan todas las escenas 3D, así como las animaciones que formarán parte del juego, todas ellas definidas en el formato w3d. Las animaciones por su parte, deben ser exportadas de forma independiente a los modelos, ofreciendo así la posibilidad de asignar una misma animación a varios modelos de la escena. En caso de tratarse de animaciones cuya única funcionalidad sea decorativa, sin llegar a interactuar con elementos importantes de la escena, como son los objetos que maneja el propio jugador, no será necesario separar la animación del modelo.

Para importar los distintos scripts del juego, así como las distintas imágenes y ficheros de audio que forman parte de este, se ha creado una función específica que se encarga de dicha tarea. Ésta crea, en función del tipo de entidad que se desea importar, el elemento necesario dentro del proyecto.

En segundo lugar, se importan todos los scripts definidos en la etiqueta <scripts> del modelo de aplicación. Como se ha visto en la sección 4.3 estos scripts son los correspondientes a los comportamientos de los distintos objetos que van a formar parte del juego, así como el comportamiento de los overlays que forman parte de la interfaz.

Es necesario que todo script definido como comportamiento de cualquier objeto u overlay, esté incluido en esta lista para evitar posteriores errores de ejecución.

Tanto los scripts de los objetos como los de los overlays extienden de una clase padre. Para los scripts correspondientes a los comportamientos de los objetos, dicha clase padre es la clase Object, mientras que para los overlays la clase padre es la clase Overlay. El uso de las mismas permite al programador trabajar con los objetos y los overlays de una manera mucho más cómoda y eficiente.

A continuación, los siguientes ficheros en ser importados son el script encargado de manejar el interfaz de juego, el propio script de juego, el cual contiene el bucle principal y las reglas del juego, tanto las de carácter genérico como las específicas de cada nivel.

Por último, es necesario incluir en la escena 3D principal las animaciones previamente importadas. Existen dos métodos en función del tipo de animación que se desee incorporar a la escena principal. Si se trata de una animación completa, es decir, aquella que se corresponde con el decorado del juego y que se compone de un modelo y de su correspondiente animación, se utiliza la función *cloneModelFromCastMember*. Por otro lado, si se trata de un fichero que contiene únicamente la animación sin el modelo correspondiente, se utiliza la función *cloneMotionFromCastmember*. La labor de dichas funciones es clonar, en el primer caso, el modelo y su correspondiente animación dentro de la escena indicada, mientras que la segunda únicamente será clonada la animación.

Llegados a este punto, todavía no se ha creado ninguna instancia de ninguno de los scripts importados, ni se ha inicializado ninguna entidad. Por el momento, la función realizada ha sido la de incorporar a Director los scripts necesarios para continuar con la ejecución del motor de producción. A continuación, poco a poco se irán creando las entidades y realizando las inicializaciones pertinentes según sea necesario.

4.4.2 Precarga de datos

Una vez procesado el modelo de aplicación y obtenida la primera estructura de datos sobre la que trabajar, el siguiente paso es procesar dichos datos, creando una serie de estructuras de datos nuevas, correspondientes a las entidades que van a formar parte del juego.

El objetivo de la fase de carga es generar dichas estructuras de datos internas necesarias, para el correcto funcionamiento de un videojuego. Para ello es necesario realizar una serie de acciones previas. Esto es conocido como precarga. Estas acciones van desde cargar ficheros externos a precalcular las distintas posiciones de los objetos en cada uno de los niveles de juego. Por tanto, se puede decir que la carga externa de datos vista en el punto anterior forma parte de la propia precarga.

A diferencia del punto anterior, donde se procesa un modelo de aplicación y se guardan los datos obtenidos en una estructura de datos tipo árbol, esta fase crea a partir dicha estructura junto con el grafo de escena obtenido al cargar el escenario 3D una nueva estructura de datos necesaria para empezar la ejecución del videojuego. Estos datos hacen referencia a las entidades del juego.

No es imprescindible tener implementada una fase de precarga para el correcto funcionamiento de un videojuego, pero es muy recomendable tener los datos preprocesados y cargados con antelación para no tener que repetir el proceso en cada nivel de juego o partida nueva. De este modo se evita que el usuario tenga que sufrir cargas innecesarias.

Cada modelo definido en el mundo 3D tiene una textura y un shader, pero estos pueden ser compartidos por varios modelos para conseguir un ahorro de espacio en los archivos w3d. Esto es muy importante a la hora de enviar los datos a los jugadores. Es necesario reducir cuanto sea posible el tamaño de los ficheros w3d al exportarlos desde la herramienta de creación 3D, para evitar saturar el ancho de banda. En Director, para conseguir un efecto de transparencia sobre un modelo, se aplica un factor alfa a su shader correspondiente, por lo que si varios modelos comparten un mismo shader, al aplicar dicho factor, todos ellos se hacen transparentes por igual. Para evitar que esto ocurra, sin que sea necesario duplicar los shaders desde la herramienta de creación 3D, se duplican por código los shaders de aquellos objetos que pueden hacerse transparentes en algún momento del juego. A la hora de aplicar el factor alfa al modelo deseado, se cambia su shader original por el duplicado y se aplica a este último el factor alfa deseado. Cuando se quiera volver al aspecto que presentaba el modelo inicialmente bastará con volver a poner el shader original.

Hasta el momento no se ha tenido en cuenta ni se ha mencionado la estructura de datos final que se genera en esta fase. Es a partir de este punto donde dicha estructura empieza a tener forma.

Como ya se ha mencionado, algunos de los ficheros importados en el paso anterior son scripts que definen propiedades y funcionalidades de los objetos que se encuentran en la escena. Es en este momento cuando se crean las instancias correspondientes a dichos scripts y se les asigna el modelo de la escena al que hacen referencia.

En el modelo de aplicación se pueden definir dos tipo de objetos, los que posteriormente actuarán como cuerpos rígidos y los que no. Como ayuda a los desarrolladores se han creado dos clases base que facilitan el manejo de estos objetos, incluyendo las funcionalidades básicas de los mismos. La primera de ellas, la clase Object, es la clase base de todos los objetos. En ella se encuentran funciones para asignar un modelo y su sombra, distintas funcionalidades que modifican la visibilidad de los objetos (blend, shader, etc.), funciones para manejar su posición y rotación, así como funciones para controlar la sombra del modelo si este se encuentra en movimiento. La segunda clase, RigidObject, extiende de Object. Esta clase añade las funcionalidades básicas para el manejo de los cuerpos rígidos, permitiendo, entre otras cosas, la asignación del cuerpo rígido en cuestión, realizar trasformaciones sobre el objeto y la asignación y modificación de sus propiedades físicas.

Puesto que pueden existir múltiples niveles de juego, para cada uno de ellos se crea una lista con las instancias de los modelos que pertenecen a dicho nivel, obteniendo así una lista con todos los niveles definidos y sus objetos correspondientes. Además se incluye un nivel adicional, el nivel 00, que contiene aquellos objetos comunes en todos los niveles, como puede ser el decorado de la escena.

En el documento de configuración general, se definen una serie de propiedades que se aplican a los objetos del mundo 3D, de un modo genérico, pero en el documento de configuración de nivel, se ofrece la posibilidad de redefinir alguna de estas propiedades, según el nivel del juego. Por ello, una vez se tenga la estructura anterior completa, con todas las instancias de los objetos creadas, y antes de pasar a la creación de sus correspondientes cuerpos rígidos, se realizan las redefiniciones correspondientes a cada objeto. El objetivo es, para cada objeto, tener correctamente definidas tanto las propiedades del modelo 3D como las del cuerpo rígido, antes de pasar a la creación del mundo físico y la posterior ejecución del videojuego.

A la hora de crear las sombras de la escena 3D, se pueden utilizar lightmaps, para objetos estáticos, o modelos que simulan ser la sombra de otros modelos. El efecto del lightmap se consigue mediante el ajuste, por código, de las propiedades gráficas definidas en la herramienta de creación 3D. Para el caso de los modelos que simulan la

sombra de otros modelos, se asigna a cada objeto el modelo correspondiente de la escena que simula su sombra. Esta asignación se consigue mediante las funciones definidas en las clases base de cada objeto.

Por último, antes de finalizar la fase de precarga, se inicializa la cámara. En caso de existir múltiples cámaras se inicializa además un manejador de cámaras. Dicho manejador, controla la cámara por defecto que ofrece Director, además de las cámaras definidas por el grafista en la escena 3D. La cámara por defecto es la cámara principal, para la cual son creados los overlays. Las demás cámaras definidas en el mundo son cámaras auxiliares que sirven como puntos de referencia a la hora de desplazar y modificar las propiedades de la cámara principal.

En caso de existir más de una cámara, será el manejador el que se encargue de inicializarlas correctamente, según las propiedades definidas por el grafista. Dicho manejador ofrece la posibilidad de realizar todos los cálculos necesarios para obtener las posiciones y rotaciones iniciales de todas las cámaras definidas para cada nivel, en caso de ser necesario. Al final de la inicialización se habrá creado una lista de fácil acceso con las posiciones y rotaciones correspondientes a cada nivel, evitando así su posterior cálculo al inicio de cada nuevo nivel.

Tanto el script que controla la cámara como el manejador, ofrecen diferentes funciones para permitir el movimiento de las cámaras. Las funciones más importantes definidas son: la rotación respecto a un objeto haciendo uso del ratón, la interpolación de movimiento, y el zoom.

Por tanto, al final de la precarga se habrá creado una estructura en forma de lista, que albergará una nueva lista por cada nivel de juego existente, con todos los datos necesarios para el correcto funcionamiento del videojuego. Una vez acabada esta fase no será necesario volver a realizar ninguna de las tareas ya realizadas en ella, consiguiendo así una mejora en los tiempos de carga entre distintos niveles o partidas.

4.4.3 Inicialización de las animaciones

Antes de adentrarse en el propio juego y después de haber cargado e inicializado correctamente las instancias de los objetos con sus correspondientes características gráficas y físicas, el último paso es la inicialización de las animaciones. Hay que recordar que estas ya se encuentran debidamente incluidas en la escena 3D principal, tal y como se ha visto en el primer punto de esta sección.

Una vez importadas e integradas todas las animaciones, el método seguido es buscar a que objetos se les ha de asignar las correspondientes animaciones. Una vez realizada esta asignación, se inicializa cada una de ellas situándola en el instante cero de su reproducción hasta el inicio del juego. Una vez iniciado el juego, el motor de producción realiza, por cada llamada al bucle principal del juego, un *update* de todas las animaciones existentes avanzando, para aquellas que sea necesario, su reproducción en función del intervalo de tiempo transcurrido desde el último *update.*

La automatización del proceso de asignación de las animaciones a sus modelos y de la reproducción de las mismas, facilita al animador el poder testear sus animaciones dentro del juego sin necesidad de la colaboración del programador.

En segundo lugar, es necesario hacer una diferenciación entre los modelos animados que forman parte del decorado y que, por tanto, no intervienen en el transcurso de una partida y los modelos que poseen su correspondiente cuerpo rígido y pueden interactuar con el resto de cuerpos rígidos de la escena, ya que el *update* de cada uno de ellos se realiza de forma diferente.

Por un lado, cualquier modelo animado que no tenga asignado un cuerpo rígido simplemente debe actualizar su nueva posición por cada llamada a la función *update.* En cambio un modelo físico animado deberá también actualizar el estado de las propiedades dinámicas de su cuerpo rígido.

La animación por computador se realiza dentro de un espacio/tiempo discreto, por lo que el desplazando de los modelos animados no se lleva a cabo de forma continúa al igual que sucede en el mundo real. Es decir, se conoce las posiciones inicial y final del movimiento que un objeto debe hacer y el tiempo que debe durar dicho desplazamiento y se realiza la interpolación correspondiente para conocer el estado del modelo en cada frame. Esto implica que los cuerpos rígidos no puedan ser actualizados del mismo modo que los modelos animados, ya que necesitan una frecuencia de muestreo mayor. Si no se cumple esta condición puede producirse submuestreo, pudiendo así penetrar los cuerpos rígidos unos dentro de otros.

Para conseguir actualizar el cuerpo físico el motor de producción realiza los siguientes pasos:

1. En primer lugar, calcula la posición y rotación actual del modelo.

2. Seguidamente, calcula la diferencia entra las posiciones actual y anterior y las rotaciones actual y anterior.

3. En función del tiempo transcurrido entre el *update* anterior y el actual el motor calcula las velocidades lineal y angular que el cuerpo rígido debe tener en el instante actual. Es decir, las velocidades se calculan en función del incremento de tiempo y el espacio recorrido por el modelo.

4. El motor desplaza el cuerpo rígido de manera que adquiera la posición y rotación calculadas en el punto 1. Este desplazamiento se lleva a cabo a través de pequeñas fuerzas que permiten al motor de física modificar el estado del cuerpo rígido con una tasa de muestreo suficiente para evitar la penetración entre éste y los demás cuerpos rígidos presentes en la escena.

5. En caso de producirse una colisión, esta es resuelta.

6. Una vez que el cuerpo rígido está situado en el punto deseado se asignan las velocidades calculadas en el punto 3 y se continúa con la ejecución de la aplicación.

Siguiendo este proceso se facilita que una animación pueda tratarse del mismo modo independientemente de que intervenga o no en la simulación física, gracias al tratamiento especializado que el motor realiza con los cuerpos físicos animados y que además, todas ellas, puedan ser testeadas en el propio juego por parte del animador.

4.4.4 Inicialización del juego

El último paso antes de empezar con la ejecución propia del juego es realizar la inicialización del mismo. Esta no sería posible sin haber realizado previamente la precarga de datos, puesto que es en ese punto donde todos los datos han sido debidamente inicializados y preparados para la posterior ejecución del juego.

Para poder realizar la inicialización del juego es necesario llevar a cabo la ejecución de una serie de funciones que dependen, además de la estructura de datos ya preparada, de algunos otros datos externos. Estos datos son aquellos que proporciona el jugador antes de empezar la partida, como por ejemplo, el número de jugadores que van a participar o el número de rondas que compone la partida.

Puesto que dichas funciones de inicialización dependen de las características propias del juego, han de ser implementadas de forma independiente al motor, es decir, en el script principal del juego. Es importante realizar el menor número de acciones en

este punto de la ejecución puesto que dichas funciones se ejecutan al inicio de cada partida nueva y al inicio de cada una de las fases. Por tanto, puesto que el objetivo de la precarga de datos es minimizar el tiempo de espera de carga, si incluimos demasiadas acciones en esta fase se perderían los beneficios que la precarga proporciona. Es por ello que esta inicialización debe ser mínima y debe incluir únicamente aquellas acciones que no puedan realizarse en la precarga del juego.

Una vez han sido seleccionadas las opciones de juego y todos los jugadores implicados han aceptado las condiciones de juego, dichas funciones de inicialización son ejecutadas (INIT GAME) mientras que el motor de producción permanece en un estado de espera hasta la finalización de las mismas (WAIT GAME).

Inicializado el juego completamente el motor de producción pasa al siguiente estado (PLAY GAME), donde se ejecuta la primera llamada al bucle principal del juego (UPDATE GAME). Dicha llamada se realiza una vez cada frame hasta la finalización del nivel o del juego. Mientras tanto el bucle principal del juego se encarga de llevar a cabo las acciones pertinentes en función de la evolución del juego y las reglas definidas.

El siguiente paso, antes de ejecutar el siguiente frame, es comprobar si el nivel o el juego ha finalizado (END LEVEL?). A medida que transcurre el juego, este va actualizando su estado en función de las acciones realizadas por los jugadores y por las reglas definidas. Será el motor el encargado de preguntar al juego si ha finalizado o bien el nivel actual o bien el juego para pasar al siguiente estado (NEXT LEVEL?) o bien para realizar una nueva llamada al bucle principal del juego (PLAY GAME).

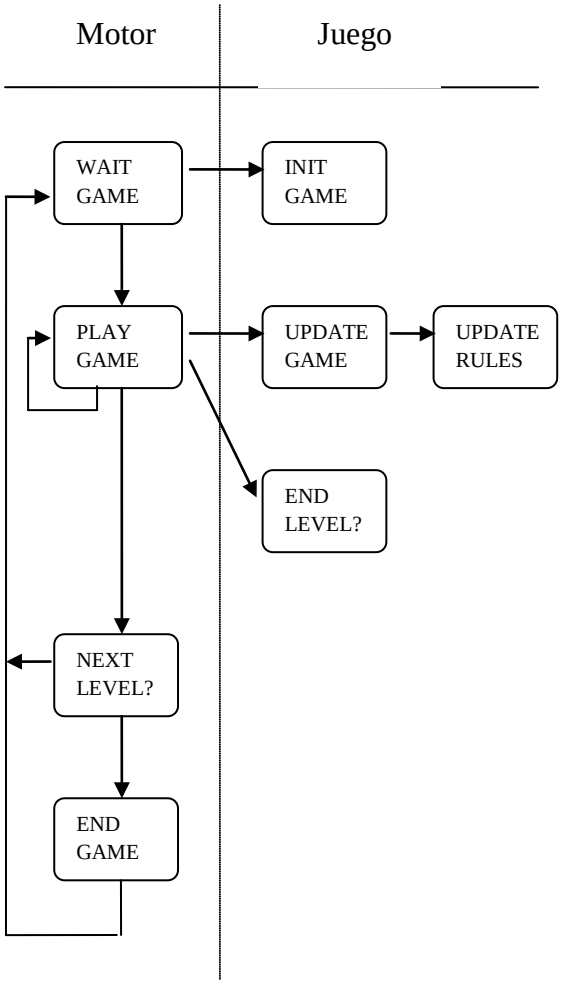

Figura 6. Flujo de estados. Inicialización del juego.

Al finalizar cada uno de los niveles se debe comprobar si todavía quedan niveles que jugar o si por el contrario ha sido el último. Mientras queden niveles, se finaliza correctamente el nivel actual y se carga e inicializa el siguiente (vuelve a WAIT GAME). Para ello se hace uso de los datos precargados al inicio del juego, por lo que la espera que sufre el jugador es mínima. A su vez, el estado del motor se actualiza para que, una vez cargado y listo el nuevo nivel, vuelva a realizar las llamadas pertinentes al bucle principal del juego (PLAY GAME).

Una vez finalizados todos los niveles de juego el motor se encarga de la ejecución de las pertinentes funciones de finalización del juego y devuelve a los jugadores al estado inicial donde se les ofrece la posibilidad de volver a jugar una nueva partida.

Aunque muchas de las funciones aquí descritas se ejecuten en la parte del propio juego, el motor de inicialización es el encargado de realizar las llamadas necesarias al script del juego de forma automática, sin importar qué acciones realice el juego en si, sino el estado en que se encuentre. En resumen, el motor inicializa el juego y realiza las pertinentes llamadas al bucle principal del juego hasta su finalización. En caso de existir más de un nivel, se repite el proceso.

4.5 Funcionalidad adicional

Por otra parte, de forma complementaria a las funciones principales del motor de producción, han sido desarrolladas una serie de funcionalidades extra. Dichas funciones proporcionan a los programadores una herramienta adicional sobre la que trabajar y dotar a los nuevos desarrollos de ciertas características de una forma más rápida y sencilla.

Dada las características del tipo de juego que se va a desarrollar, es decir, videojuegos casuales online multijugador en 3D, algunas de estas funciones resultan de gran utilidad mientras que otras aportan variedad y vistosidad a los juegos, obteniendo unos grandes resultados, tanto gráficos como de jugabilidad.

Dichas funcionalidades son las siguientes:

- Creación de escenarios abiertos.
- Creación de modelos siempre visibles.

- Tratamiento de objetos aleatorios.
- Uso de proxys.
- Scripting desde 3D Studio Max.

Las dos últimas de las funcionalidades mencionadas se describen con mayor detalle en el trabajo realizado en [1].

4.5.1 Creación de escenarios abiertos

La creación de escenarios abiertos es llevada a cabo haciendo uso de la técnica de creación de fondos skydome. Con esta técnica se consigue, mediante el uso de unos pocos polígonos y una sola textura repetida múltiples veces, simular un escenario que aparentemente es más grande de lo que en realidad es. El resultado obteniendo es muy realista, ya que se consigue que los objetos que están en la lejanía, como pueden ser el cielo, montañas o edificios, parecen estáticos, mientras que los elementos más cercanos al jugador se desplazan, tal y como ocurre en la vida real.

Para que esta técnica sea efectiva es necesario que el skydome esté suficientemente alejado al punto de vista del jugador, ya que si este se aproxima puede apreciar el "truco" y se pierde el efecto deseado.

Por esto, es necesario que, tras la carga del mundo 3D, se sitúen todos los objetos, tanto los lejanos como los cercanos al punto de vista del jugador, en unas coordenadas válidas para que el skydome consiga el efecto buscado. Este proceso es realizado de manera automática por el motor de producción para cada uno de los niveles del juego durante la fase de precarga. En concreto, el motor de producción traslada la escena correspondiente a cada nivel al centro de coordenadas del mundo, coincidiendo así con el centro del decorado.

Para diferenciar que objetos forman parte del decorado y se comportan como elementos lejanos al jugador se ha utilizado una nomenclatura especial que los identifica. Concretamente, todos estos modelos deben contener el prefijo "*dec_*".

El uso de esta técnica permite, por tanto, la creación de escenarios abiertos bastante realistas con un número muy reducido de polígonos y texturas. Emplear el mínimo número de texturas y polígonos es una constante en el desarrollo de

videojuegos, pero es en los juegos online donde este aspecto cobra una mayor importancia.

4.5.2 Creación de modelos siempre visibles

En ciertos juegos aparece la necesidad de que ciertos modelos de la escena permanezcan siempre visibles al jugador, principalmente debido a que estos modelos son claves para alcanzar un determinado objetivo. La creación de este tipo de modelos, requiere un tratamiento especial que es realizado de forma automática por el motor de producción. Además, gracias al modelo de aplicación es posible definir de manera externa que modelos de la escena presentan esta característica así como el orden de prioridad de cada uno de estos modelos. Concretamente, esto es definido dentro de la etiqueta *<specialFeatures>*.

Desde la herramienta Director no es posible modificar el orden de aparición de los modelos en la escena, ya que esto se realiza de forma automática en función de su lejanía a la cámara. Internamente, los modelos del grafo de escena son ordenados bajo un índice que está directamente relacionado con como dichos modelos han sido creados dentro de la herramienta de creación 3D. Es fácil observar el gran inconveniente que esto conlleva, ya que aunque en primera instancia resulta muy cómodo este modo de trabajo, si aparece la necesidad de tener que variar estos índices la tarea se vuelve intratable. Tener que rehacer la escena 3D dependiendo del orden en el que se desea que aparezcan los objetos es totalmente antiproductivo. En resumen, si se desea desarrollar un juego en el que ninguno de los modelos deba aparecer siempre visible al jugador, la forma en la que Director trata la aparición de los modelos en la pantalla es óptima, pero si por el contrario se desea que existan ciertos objetos que siempre permanezcan visibles por encima de todos los demás, Director plantea una serie de impedimentos que dificultan mucho esta tarea.

La forma en la que el motor de producción facilita la existencia de este tipo de modelos consiste, principalmente, en la clonación de los mismos. Para cada uno de los modelos que presentan esta característica, el motor de producción crea un modelo idéntico al original, gracias a la clonación de modelos que ofrece Director. De este modo, el orden en el que se sitúa el nuevo objeto dentro del grafo de escena está siempre por detrás del resto de modelos, es decir, tiene asignado un índice mayor. El modelo clonado se sitúa en las mimas coordenadas que el original, pero, para facilitar el posterior tratamiento de los dos objetos, el original y el clonado, se crea una relación

padre-hijo entre ambos, logrando así que todas las transformaciones que sufra el modelo original se vean reflejadas en el modelo clonado. Una vez se realiza este proceso, al modelo clonado se le asigna una propiedad de visibilidad específica, #front_depth. Esta propiedad permite al objeto clonado ser visible por encima de todos los objetos que aparecen por delante de este en el grafo de escena, o lo que es lo mismo, el objeto clonado será visible por encima de aquellos modelos cuyos índices sean menores al índice del modelo clonado. Cualquier otro objeto definido por detrás de este lo tapará.

Cada vez que un objeto es clonado, se crea a su vez un clon de cada uno de los shaders que el modelo tiene asignado de tal manera que el modelo clonado posee un shader independiente del modelo original, pudiendo así modificar los valores de cada uno sin que el otro se vea afectado. El objetivo principal de esta distinción en los shaders de cada uno de los modelos es el de poder variar el nivel de transparencia del modelos clonados. De este modo, el modelo puede aparecer siempre visible al jugador, pero diferenciando cuando este está tapado por otro modelo y cuando no; los polígonos del modelo que estén detrás de otro modelo pueden aparecer semitransparentes, mientras que los polígonos totalmente visibles pueden mostrarse con un nivel de alfa 0.

En resumen, la creación de modelos siempre visibles es realizada de forma automática por el motor de producción durante la fase de inicialización mediante la técnica de clonación de modelos. Para ello, basta con definir en el modelo de aplicación aquellos modelos que presenten esta característica y el orden de prioridad de cada uno de ellos. Además, es posible realizar ciertas variaciones en los shaders del modelo original y el modelo clonado para conseguir un efecto más realista y poder diferenciar entra aquellos polígonos ocultos y aquellos totalmente visibles.

4.5.3 Tratamiento de objetos aleatorios

Una característica que puede resultar interesante para el usuario del juego que se desea desarrollar es dotar a algunos de los objetos de cierta aleatoriedad. Esto significa que, se puede conseguir en los juegos una mayor jugabilidad con el simple hecho de que algunos de los objetos pertenecientes a la escena donde se desarrolla el juego no aparezcan al inicio de la fase siempre en el mismo lugar o bien estén aparentemente en el mismo sitio pero que en realidad han sido desplazados un par de milímetros, siendo imperceptible por el jugador.

Dicha característica de los objetos ofrece a los futuros jugadores mayor variedad y jugabilidad, evitando que se aprendan las fases e incluso conseguir diferentes efectos si además introducimos física a los objetos.

A la hora de trasladar esta característica al código del juego, puesto que de lo que se está hablando es de juegos online multijugador, es necesario que todos los jugadores presentes durante la partida jueguen en igualdad de condiciones y que al inicio de cada una de las fases los objetos se encuentren en el mismo lugar para cada uno de ellos en sus respectivas pantallas. Por tanto, la siguiente partida que jueguen, los objetos aparecerán en lugares distintos a los de la partida anterior, pero todos los jugadores verán dichos objetos en la misma posición que sus rivales.

Para ello, el jugador con identificador número uno, el cual se trata de un jugador que siempre va a estar presente, durante la fase de inicialización de los objetos, donde se crean sus correspondientes instancias y asignaciones de modelos y texturas, calcula además la posición donde se mostrará en cada una de las fases de las que se compone el juego que se está cargando. Una vez calculadas todas las posiciones, el jugador uno informa a los demás jugadores mediante un mensaje de las posiciones calculadas que corresponden a cada uno de los objetos. Los demás jugadores, al recibir el mensaje, asignan la información indicada al objeto que corresponda. De este modo, una vez finalizada la inicialización, se obtiene un estado inicial del juego diferente entre partida y partida, pero igual entre todos los jugadores.

El factor de aleatoriedad de cada uno de los objetos depende del propio objeto, es decir, es independiente del script del juego y del motor de producción, y debe estar definido en el script que define el comportamiento del propio objeto. Por otro lado, no tiene por qué tratarse de un objeto aislado, sino que dicho comportamiento y a su vez su factor de aleatoriedad puede asignarse a un grupo de objetos, gracias a la incorporación del modelo de aplicación basado en ficheros XML, expuesto en la sección 4.3, donde se pueden definir tipos de objetos con su correspondiente script de comportamiento.

Además es importante destacar que esta característica no solo es aplicable a la posición inicial de los objetos, sino que puede trasladarse a cualquier propiedad que posea el objeto, como puede ser aleatoriedad en su rotación o visibilidad. El motor de producción proporciona los mecanismos de comunicación y sincronización necesarios para que todos los jugadores obtengan un estado inicial común. La característica propia de aleatoriedad la proporciona cada uno de los tipos de objetos definidos.

4.6 Conclusiones

Tras localizar y analizar una serie de problemas y carencias en el proceso de producción seguido actualmente para el desarrollo de los videojuegos casuales online multijugador, y centrándose en aquellos desarrollos que utilizan la herramienta de autor Adobe Director, en el presente capítulo se ha mostrado con mayor detalle y precisión las características del motor de producción desarrollado como solución a dichos problemas.

Mediante el uso del motor de producción se consigue, no solo minimizar los costes de producción minimizando el tiempo dedicado al desarrollo y a la fase de testeo de los videojuegos desarrollados, sino además, la complejidad del propio desarrollo. Esto es posible gracias a que el motor de producción desarrollado proporciona una base sobre la que empezar a trabajar, permitiendo al programador centrarse en las cuestiones claves del juego, consiguiendo un desarrollo mucho más eficiente y modular.

Gracias a la filosofía sobre la que se ha basado el desarrollo del motor de producción se han conseguido los objetivos descritos.

Configuración externa. La integración en el motor de producción de esta característica proporciona, gracias al uso de documentos en formato XML, una herramienta que permite que cualquier persona del equipo de desarrollo, aun careciendo de conocimientos técnicos, sea capaz de configurar cualquier parámetro del juego, siempre dentro de unos límites establecidos. Además esto permite que se puedan realizar distintas pruebas sin necesidad de modificar el código interno del juego, de una forma rápida y sencilla, como por ejemplo probar diferentes escenarios, comportamientos de los objetos, animaciones o propiedades físicas.

Motor de inicialización. Se trata de un nuevo motor que ha sido desarrollado específicamente para que realice aquellas tareas de inicialización y finalización comunes en la estructura de ejecución de los videojuegos. Gracias al papel que desempeña dentro del motor de producción, una vez finalizado este proceso, el juego se encuentra totalmente preparado para su ejecución. Todos los datos han sido correctamente instanciados e inicializados, además de modificados en caso de ser necesario, y todos los scripts y demás ficheros han sido incorporados la juego creando las entidades pertinentes para su posterior uso. De este modo, el programador únicamente debe centrarse en el desarrollo propio del juego, sin necesidad de implementar en repetidas ocasiones la fase de carga e inicialización y finalización del juego, incluyendo la inicialización y finalización de cada uno de los niveles que pueda integrar.

Middleware. Por último, la capacidad de actuación del motor de producción como sistema *Middleware* proporciona la programador la integración de forma transparente de varios motores de gran utilidad, como son los motores de física y de detección y resolución de colisiones, además de la integración del sistema multiusuario MUS. Gracias a esta funcionalidad se consigue reducir los costes de producción de un videojuego, evitando al desarrollador estudios e integraciones costosas. Si se desea obtener una mayor información sobre el aspecto *Middleware* del motor de producción consultar [1].

Además, se han incorporado una serie de funciones adicionales al motor de producción que permite al desarrollador implementar una serie de características con mayor agilidad y menor coste.

En primer lugar, el motor permite la creación de escenarios abiertos haciendo uso de la técnica *skydome.* Del mismo modo que en el caso anterior, todo el proceso necesario para la utilización de esta técnica es realizado de forma automática por el motor.

En segundo lugar, el motor automatiza el proceso de creación de modelos siempre visibles empleando la técnica de clonación de modelos. Desde Director no es posible decidir el orden en que los objetos son dibujados por el motor de render. La técnica de clonación de modelos permite la creación de modelos siempre visibles de una manera fácil y automática, sin que el programador deba desarrollar ninguna funcionalidad adicional.

Y en tercer lugar, el motor proporciona la herramienta necesaria para la inclusión de elementos con características aleatorias. Aunque es el propio script del objeto el encargado de proporcionar el tipo de aleatoriedad que este tiene, es el motor el encargado de sincronizar los estados iniciales de los jugadores una vez calculados los factores de aleatoriedad para cada uno de los objetos con dicha propiedad. Gracias al uso de mensajes y al proceso de inicialización de objetos es posible introducir este tipo de elementos de una forma sencilla y automática.

Además, el motor de producción automatiza el uso de *proxys* y permite realizar scripting desde la herramienta 3D Studio Max, con el objetivo de paliar las limitaciones que posee el exportador de ficheros W3D de dicha herramienta. Todo ello se encuentra debidamente documentado en [1].

En el siguiente capítulo se muestra como ha sido desarrollado un nuevo juego casual online multijugador utilizando el motor de producción como base de su

desarrollo y como han sido aprovechadas las funciones que este proporciona. En el capítulo 6, se realiza una comparación del nuevo juego desarrollado con otro juego de características similares desarrollado con anterioridad al motor de producción. Es en este capítulo donde se muestra, mediante una evaluación de los costes de producción, la eficacia del motor desarrollado. Por último, en el capítulo 7 se realiza un balance general de la tesina con las conclusiones obtenidas y se concluye con una serie de propuestas como posibles líneas de trabajo futuro.

Capítulo 5

Desarrollo de un videojuego: Minigolf

5.1 Introducción

A lo largo de la siguiente sección se presenta un videojuego que se ha desarrollado utilizando como base para su programación el motor de producción que se ha presentado en la sección 4, con el objetivo de comprobar la eficiencia y usabilidad de dicho motor en la fase de desarrollo de un videojuego de tipo casual 3D online multijugador, quedando el tratamiento en detalle de la programación del juego fuera del alcance de esta tesina.

El juego desarrollado ha sido un juego de minigolf 3D online multijugador. Se trata de un juego conocido, popular y fácil de aprender, en el que el objetivo final del juego es introducir la bola dentro del hoyo con el menor número de golpes posibles a lo largo de las diferentes pistas.

Después de realizar un pequeño estudio sobre los distintos juegos de minigolf existentes en diversos portales de juego, se obtuvo como resultado que: los juegos de minigolf en 3D son casi inexistentes, y que los del tipo online multijugador, aunque aparecen en un porcentaje un tanto mayor, siguen siendo de número reducido. La mayoría de estos juegos son juegos en dos dimensiones y por turnos, es decir, en cada turno tira uno de los jugadores.

Por tanto, debido a los resultados obtenidos en dicho estudio, a la sencillez que presenta el juego y a las características propias de este, el juego del “Minigolf” resulta idóneo para explotar al máximo las funcionalidades ofrecidas por el motor de producción, obteniendo como resultado final un juego de minigolf en 3D, online

multijugador, en el que el juego transcurre en tiempo real, es decir, con todos los jugadores jugando al mismo tiempo sobre las diferentes pistas.

Para este juego se han desarrollado dos modalidades de juego distintas. La primera de ellas permite colisiones entre las bolas de los jugadores deshabilitándolas con el resto de bolas únicamente en el momento en que el jugador se dispone a realizar un tiro, mientras que la segunda modalidad de juego no permite ningún tipo de colisión entre las bolas de los jugadores a lo largo de la partida, pero si contra los distintos obstáculos que se encuentran en las pistas. Las dos modalidades, al igual que el motor de producción, han sido desarrolladas bajo la herramienta de autor Adobe Director, utilizando como lenguaje de programación Lingo.

A continuación se expone cómo se ha llevado a cabo el desarrollo del juego utilizando el motor de producción, haciendo hincapié en las partes de la producción del juego donde se han aprovechado las funcionalidades que ofrece el motor y que abarcan esta tesina, sin entrar en detalle en la programación interna del propio juego. En la sección 5.2 se hace un pequeño análisis de cómo se ha configurado el modelo de aplicación y qué elementos han sido incluidos para la realización de este juego. En la sección 5.3 se expone a grandes rasgos la aportación ofrecida por el motor de inicialización en el desarrollo del juego del “Minigolf”. Por último, antes de las conclusiones, en la sección 5.4 se hace referencia a las funcionalidades adicionales incluidas en el motor de producción, haciendo una descripción de su uso dentro del juego desarrollado.

5.2 Configuración externa. XML

Para el desarrollo de este juego se ha utilizado la configuración externa, principalmente para definir los objetos que constituyen la escena principal del juego y los distintos niveles de los que consta.

Mediante los modelos de aplicación, se ha definido todos los parámetros necesarios para el correcto funcionamiento del juego. Algunos de ellos son: la física global del juego, la escena principal 3D, el script propio del juego o los directorios donde se han ubicado las entidades utilizadas en el juego y sus correspondientes scripts. Todo ello y los demás parámetros obligatorios descritos en la sección 4.3, han sido debidamente incluidos según la definición de los modelos de aplicación que soporta el motor de producción.

Como elementos destacables, se incluye una serie objetos especiales, los cuales vienen definidos con un script de comportamiento particular previamente desarrollado. Además se incluyen distintas animaciones y la asignación a los correspondientes modelos de la escena que posteriormente serán animados en función de dicha definición. Dichos objetos especiales, se exponen con más detalle en la sección 5.4.

Tanto las animaciones, los objetos y cualquier tipo de comportamiento definidos en el modelo de aplicación permite que estos se puedan probar, modificar y/o eliminar con solo modificar el fichero de configuración. Además se puede aplicar una misma animación a múltiples objetos con solo incluir su correspondiente etiqueta en el modelo de aplicación indicando los objetos a los que se desea asignar dicha animación o por el contrario eliminar dichas animaciones sin necesidad de entrar en el propio código. Esta característica es también aplicable a los comportamientos definidos para los objetos.

Otra característica que se ha configurado haciendo uso del modelo de aplicación general son un par de objetos que utilizan la propiedad de visibilidad permanente expuesta en la sección 4.5. Dichos objetos son las bolas y los palos propios de los jugadores. Gracias a la configuración externa y al motor de inicialización la propiedad de visibilidad necesaria para crear dicho efecto, se aplica de forma automática sobre los objetos que definen en el modelo de aplicación la etiqueta correspondiente <*specialFeatures*>.

El interfaz que se utiliza en este juego es muy sencillo y contiene escasos overlays. Solo contiene un marcador temporal que representa una cuenta atrás, en forma de reloj digital, que se muestra a los jugadores cuando alguno de ellos ha superado el nivel de juego actual o lo que es lo mismo, ha conseguido introducir su bola dentro del hoyo antes que sus contrincantes. Este reloj indica a los demás jugadores el tiempo restante de que disponen para conseguir introducir su bola dentro del hoyo. El modelo de aplicación permite definir las imágenes necesarias para la creación de dicho overlay de una forma rápida y sencilla, permitiendo además poder cambiar las imágenes utilizadas en cualquier momento, sin necesidad de entrar en el propio código.

Por otro lado, en el modelo de aplicación de nivel, se incluye, la escena principal 3D, las reglas generales del juego y las definiciones de todos los niveles existentes en la escena. En lugar de definir niveles de dificultad, se ha tratado cada nivel como una pista de juego distinta. Para este juego no se utilizan ningunas reglas específicas de nivel, todas las pistas se rigen por las mismas reglas generales. No obstante se incluyen redefiniciones de algunos de los modelos incluidos tanto en la escena como en el modelo de aplicación general de la aplicación.

El uso del XML en el modelo de aplicación, tanto general como de nivel, resulta de un gran utilidad tanto a los desarrolladores como a los diseñadores, los cuales pueden añadir, modificar o eliminar pistas y objetos, además de modificar las propiedades físicas o las reglas del propio juego de una forma rápida, sencilla y totalmente independiente del código del motor de producción, observando los cambios al instante.

5.3 Motor de inicialización. Inicialización de estructuras

Puesto que el juego consta de varias pistas para jugar, es necesario crear en la fase de precarga las estructuras de datos necesarias para el correcto funcionamiento del juego. Mediante el uso del motor del inicialización incluido en el motor de producción, se consigue realizar dicha tarea de forma automática para cada uno de los niveles definidos en el modelo de aplicación de nivel. Es muy importante, dada la magnitud de datos que se va a manejar, tener todos los datos procesados e inicializados correctamente, para evitar a los jugadores esperas innecesarias mientras se cargan los distintos niveles o pistas de juego.

Al inicio de esta fase se importarán todos los script y ficheros, previamente definidos en los modelos de aplicación, los cuales son necesarios para poder empezar con la inicialización de las estructuras de datos pertinentes, todo ello de una forma automática y transparente para los desarrolladores.

Dichos scripts y ficheros no son más que los scripts de comportamiento de juego, las reglas, el manejador de física y colisiones y el propio script de juego, además de los ficheros w3d que contienen la escena principal y las animaciones, en caso de ser necesarias. A todo ello se suma las imágenes que serán utilizadas para la creación de los overlays definidos en la interfaz de juego, las cuales deben ser también importadas.

Siguiendo los pasos descritos en la sección 4.4, una vez realizada la importación de ficheros, es el momento de empezar a dar forma a la estructura de datos principal, aquella en la que se almacenan todos los datos referentes a cada nivel. Para ello se analiza cada nivel descrito en los modelos de aplicación y se crean las instancias correspondientes a los objetos pertenecientes a cada nivel.

Paso a paso, cada objeto de la escena será inicializado y preparado con todas las características descritas en los modelos de aplicación. Esto incluye desde la asignación de modelos y propiedades que afectan a la visualización final del objeto, así como la

creación de los cuerpos rígidos que se utilizarán para la simulación física, con sus respectivas propiedades físicas, pasando por la asignación de sombras, duplicación de shaders, posicionamiento dentro de la escena, creación de clones y asignación de animaciones.

Una vez finalizado dicho proceso, se ha obtenido una lista con todos los objetos que forman parte del juego, los cuales se encuentran totalmente preparados para ser utilizados en el momento en que se requiera a lo largo de la ejecución del juego.

Por otro lado, es necesario también inicializar correctamente las cámaras que mostrarán a los jugadores la escena de juego. Para ello se va a hacer uso del manejador de cámaras que se ha incluido en el motor de producción, ya que para este juego se ha decido hacer uso de distintas cámaras. La primera de ellas se trata de una cámara situada en la parte superior de la escena, que muestra a los jugadores la pista que se va a jugar des del cielo, ofreciendo una vista general al inicio de cada pista. Esta cámara se utiliza también como transición entre pistas, mostrando el cielo mientras se cambia de una pista a otra. La segunda cámara es la cámara principal del juego, la cual realiza un seguimiento de la bola de cada jugador en función de su desplazamiento por el terreno de juego, es decir, es la cámara que se utiliza por defecto para jugar. Esta permite, rotaciones respecto la bola, para definir mejor el tiro, además de realizar dicho seguimiento de la trayectoria que sigue la bola, después de haber realizado el tiro. Por tanto el manejador de cámaras inicializa las cámaras definidas en la escena y calcula sus posiciones iniciales para cada pista. Posteriormente, durante el juego, se encarga de realizar los movimientos pertinentes según el estado en que se encuentre el juego.

A continuación, es necesario inicializar las reglas, el motor de física del juego y el motor de detección de colisiones, además de la inicialización de la interfaz de juego, pero dichas tareas queda fuera del alcance de esta tesina y no se entrará en más detalle. Para más información sobre este tema consultar [1].

Por último, antes de empezar con la ejecución propia del juego, es decir, antes de adentrarse en el bucle principal del juego, el último paso es la inicialización de las animaciones. Hasta el momento, se han importado dentro de la escena principal y se ha asignado dichas animaciones a sus correspondientes modelos, pero, como se expuso en el punto 4.4 donde se hace referencia a las animaciones, para su correcta inicialización y funcionamiento, se requiere de los cuerpos rígidos de los modelos animados, en caso de no ser estos elementos decorativos. Por tanto, en este punto de la ejecución del motor de inicialización se inicializan las animaciones.

Llegados a este punto los datos y objetos están preparados para la ejecución del bucle principal del juego. Previamente el script correspondiente al juego ha sido correctamente importado y se ha creado su correspondiente instancia al inicio de esta fase.

Por tanto, recapitulando, es en esta fase donde se inicializan todas las entidades que forman parte del juego, es decir, los objetos, las animaciones y las cámaras propias del juego. Además se crean las instancias de los scripts previamente importados y que se van a utilizar durante la ejecución del juego. Estos scripts corresponden a los comportamientos de los distintos objetos y overlays que aparecen en el juego, a las reglas propias del juego, al intérprete de física necesario para el correcto funcionamiento del motor de física escogido para la ocasión y el script principal del juego.

Tal y como se expone en la sección 4.3 son muchas las tareas internas que realiza el motor de inicialización. Dichas tareas, han sido utilizadas y aprovechadas al máximo durante el desarrollo de este juego, para minimizar a medida de lo posible tanto el tiempo como el coste de desarrollo, automatizando todos los procesos que lo permitiesen.

5.4 Funcionalidades adicionales

A continuación se describe de qué modo han sido utilizadas las funcionalidades adicionales que han sido desarrolladas en el motor de producción. Estos son solo algunos ejemplos de su uso, mostrando la aportación que ofrecen al juego desarrollado para la ocasión:

Creación de escenarios abiertos.

El juego del minigolf ha sido ambientado en un espacio verde, rodeado de montañas, con un diseño realista tanto de las pistas de juego como de los objetos que podemos encontrar en ellas.

Una de las funcionalidades adicionales que ofrece el motor de producción y que se ha utilizado en el desarrollo de este espacio de juego es la creación de escenarios abiertos mediante la técnica de creación de fondos *skydome*. Para conseguir el efecto

deseado, es decir, la sensación de encontrarse en un espacio abierto, que proporciona dicha técnica, se ha creado el escenario sobre una semiesfera a la que se ha aplicado una textura con la imagen de un paisaje con montañas.

Para facilitar el trabajo a los diseñadores, cada una de las pistas se encuentra situada en lugares separados dentro del propio espacio creado para la utilización de la técnica del *skydome*, es por ello por lo que se debe trasladar cada una de las pistas junto con sus objetos al centro de coordenadas de dicha semiesfera. Los objetos pertenecientes al decorado vienen definidos por el prefijo "dec_". Estos objetos son fijos, y son los que ayudan a recrear el efecto deseado tal y como se ha expuesto en la sección 4.5.

Por tanto, durante la fase de precarga del juego realizada por el motor de inicialización, donde se crean e inicializan todas las estructuras de datos anteriormente mencionadas, se realiza dicha translación de las diferentes pistas y objetos que componen el juego al centro de coordenadas de la escena. De este modo se consigue el efecto deseado, un escenario aparentemente muy grande, con una apariencia realista, utilizando pocos polígonos y texturas.

Creación de modelos siempre visibles.

Las pistas diseñadas para este juego vienen dotadas de objetos y diseños creativos que aportan una imagen y jugabilidad que resulta muy atractiva de cara al usuario final. Hay que recordar que es un juego en tiempo real, donde los jugadores se encuentran en la pista jugando todos al mismo tiempo sorteando los distintos obstáculos que puedan encontrar en cada pista, por lo que los objetos y el diseño de las pistas deben estar muy cuidados para no entorpecer la visibilidad de los jugadores.

Para resolver este problema se ha hecho uso de los objetos siempre visibles, una de las funcionalidades adicionales que se encuentra desarrollada en el motor de producción y que se expuso en la sección 4.5.

Los objetos que en este caso aplican dicha propiedad son las bolas y los palos que utiliza cada uno de los jugadores. De este modo, durante el transcurso del juego todas las bolas y palos de los participantes en el juego serán siempre visibles para el propio usuario, pudiendo tener a sus contrincantes o compañero de juego siempre localizado con el mínimo esfuerzo.

Para ello es necesario definir en el modelo de aplicación general en los objetos correspondientes, en este caso *bola* y *palo*, la etiqueta anteriormente mencionada <*specialFeatures*> indicando que se desea aplicar dicha propiedad y en qué orden de aparición. Puesto que son dos los objetos a los que se va a aplicar la propiedad de visibilidad permanente es necesario definir cuál de los dos objetos será visible por encima del otro, puesto que, como se expuso en la sección 4.5, la visibilidad del objeto depende de su índice dentro del grafo de escena, por lo que los dos objetos no pueden tener el mismo índice.

Utilizando esta función adicional incluida en el motor de producción, el programador solo necesita añadir un par de líneas en el modelo de aplicación general para indicar qué objetos requieren hacer uso de esta técnica, sin necesidad de tener que preocuparse cada vez por su implementación con todo lo que ello supone.

No obstante, para aquellos objetos más grandes que se encuentran en la pista, y que de algún modo pueden entorpecer la visibilidad de los jugadores a la hora de realizar su tiro, se ha aplicado la técnica descrita en la sección 4.4, donde se expone como se aplica la transparencia a los objetos mediante la duplicación de sus shaders. Para realizar este efecto, existe una serie de shaders definidos con el prefijo “trans_”. En la fase de precarga, al encontrar este tipo de shader se está indicando que durante la ejecución del juego puede darse el caso en que el objeto al que dicho shader pertenece deba hacerse transparente para facilitar la visión al jugador.

Es necesario recordar que, el mismo shader puede ser compartido por más de un objeto. Es por ello que estos shaders se diferencian con dicho prefijo. Estos son los que se clonan durante la fase de precarga y se les aplica un valor de alfa distinto al que ya tienen definido para dotarlos de cierta transparencia. En el momento sea necesario, se intercambia el shader original del modelo que se deba hacer transparente por el clonado. Una vez ya no se sea necesaria dicha transparencia, se vuelve a asignar su shader original.

Tratamiento de objetos aleatorios.

Para evitar que acabe siendo un juego elitista, en el que solo los mejores sean admitidos en el juego, y que los nuevos usuarios no se vean frustrados, tanto por los niveles del juego, como por los jugadores antes mencionados, también conocidos como hardcore, se ha añadido al juego un factor de aleatoriedad. Para ello, se ha llevado a cabo una serie de acciones.

La primera de ellas consiste en la creación de múltiples pistas, muy variadas, y con distintos elementos y formas, las cuales se jugarán de forma aleatoria, evitando que el jugador más avanzado pueda escoger solo aquellas pistas que domine.

La segunda acción llevada a cabo está relacionada con los distintos elementos estáticos que forman parte de las distintas pistas que componen el juego, los cuales vienen dotados de ciertas características que les proporciona cierta aleatoriedad. Algunos de ellos aparecen o desaparecen aleatoriamente entre juego y juego, es decir, cada vez que se juega la pista, los elementos aparecen o no, según un factor de aleatoriedad. Otro tipo de estos elementos son aquellos que se ven desplazados en función de una trayectoria previamente definida por el diseñador, la cual no resulta conocida por el usuario final. Y por último, existe otro tipo de objetos definidos en grupo, los cuales, siempre aparecen en la pista, pero cada vez que se juega, solo aparece uno de los elementos que pertenece a dicho grupo, seleccionado también de forma aleatoria. Este tipo de elementos ofrecen mucho juego; a los diseñadores les permite crear los distintos niveles con mayor facilidad y rapidez, facilitan el trabajo de los programadores, puesto que solo tienen que desarrollar el comportamiento del tipo de elemento una sola vez, y a los jugadores les ofrece una mayor jugabilidad.

Las dos últimas acciones son, la introducción de elementos móviles, es decir, animaciones dentro de las propias pistas y la asignación aleatoria de la posición de salida de cada jugador en cada una de las pistas. Para cada pista, siempre hay cuatro posiciones iniciales predefinidas, que marcan el lugar de comienzo de cada uno de los jugadores. Al inicio de cada nivel y juego, la posición se asignará de forma aleatoria a cada uno de los participantes, siendo esta distinta respecto al nivel y juego anterior.

Esto ha sido posible gracias a la configuración de datos y carga de ficheros externos que permite el motor de producción. Como se vio en el punto 4.2, donde se explica con detalle el parser y el modelo de aplicación, los objetos pueden dotarse de comportamientos a través de scripts externos. Estos comportamientos pueden ser, cálculos para determinar la ubicación de los objetos, como es el caso que se presenta a continuación, acciones que deben realizar los objetos en determinados estados de ejecución o cualquier otro comportamiento que se desee conseguir.

Los comportamientos que aquí se presentan son ubicaciones aleatorias. Existen tres tipos de objetos con esta característica.

Los primeros, son los objetos llamados *random*. Vienen definidos con el prefijo *ale* seguidos de un número que indica la probabilidad de que este aparezca en la pista.

Por ejemplo, un objeto que se llame *ale50* tendrá un 50% de posibilidades de estar presente. Por tanto, se trata de objetos únicos, que solo aparecen o desaparecen según un factor de aleatoriedad.

Los segundos, son los objetos llamados *move*. Estos objetos son un tanto más complejos que los anteriores, y se caracterizan por ser siempre visibles, pero en una localización distinta siguiendo una trayectoria lineal. Cada uno de estos objetos viene acompañado de un dummy que indica cuál será la posición final en la que puede aparecer dicho objeto. Con su posición inicial y la posición marcada por dicho dummy, se traza una línea recta y se calcula una posición aleatoria dentro de la trayectoria calculada. El resultado, será la nueva posición del objeto *move*.

Por último, se encuentran los objetos llamados *group*. Este tipo de objetos se caracteriza por ser un grupo de objetos de los cuales solo será visible uno. Es decir, se define, por ejemplo, un grupo de triángulos, tantos como se quieran, y se selecciona uno aleatoriamente, mostrando el escogido y ocultando los demás componentes del grupo. Es un tipo de objeto bastante similar a los objetos de tipo *random*, con la diferencia que se muestra uno entre varios.

En caso de que el diseñador del juego quisiese crear nuevos comportamientos o nuevos grupos de objetos, bastaría con definirlos en el modelo de aplicación y crear su script correspondiente. El motor de producción se encargará de crear el tipo de objeto definido.

Esto proporciona mucha versatilidad en lo que a tipos de objetos se refiere, puesto que resulta muy sencillo crear distintos tipos de objetos o reutilizarlos en otros proyectos con solo incluir los datos pertinentes en el fichero de configuración.

La inclusión de este tipo de objetos, al igual que la aleatoriedad de las pistas y las posiciones de salida de los jugadores, son posibles gracias a la funcionalidad adicional incluida en el motor de producción que trata objetos aleatorios y su sincronización entre los distintos jugadores, visto en la sección 4.5.

5.5 Conclusiones

A lo largo del capítulo se ha mostrado como se ha desarrollado de forma sencilla un juego popular, sencillo y divertido utilizando todas las funcionalidades que ofrece el motor de producción desarrollado.

Cabe decir que programar el juego del “Minigolf” teniendo ya una base de programación que permite centrarse en la programación propia de los scripts del juego y sus reglas ha propiciado una importante reducción en los costes de producción del mismo. En primer lugar, el motor de producción ha servido como guía a la hora de definir la estructura interna de un juego. Debido a que el motor ya se encuentra estructurado conforme a la estructura típica de un juego, la posibilidad de cometer errores en la estructuración del juego se ve claramente reducida. No obstante, es en el proceso de automatización de los procesos de inicialización donde el motor de producción ha realizado uno de sus mayores aportes al proceso de producción del juego. Gracias a este proceso automatizado, un gran número de tareas han requerido una colaboración mínima por parte del programador o incluso no la han requerido. En el caso de que esta funcionalidad no hubiera estado implementada en el motor de producción, gran parte de la misma debería haber sido programada ad-hoc para este juego en concreto.

Por otro lado, la posibilidad de definir diferentes scripts de comportamiento para cada uno de los distintos objetos dentro de la escena, además de la posibilidad que ofrece de definir múltiples animaciones, las cuales pueden ser asignadas a distintos modelos, resulta de gran utilidad. Poder definir este tipo de modelos permite dotar al juego de una mayor jugabilidad, aportando al juego un importante factor de aleatoriedad en los distintos niveles, sin necesidad de tener que modificar el código interno del motor de producción.

Gracias a ello, no solo se ha conseguido realizar el desarrollo del juego en poco tiempo, sino que además se ha conseguido incluir una serie características que resultan innovadoras y que aportan más jugabilidad al usuario final. Además se han desarrollado una serie de comportamientos, scripts y animaciones que pueden ser reutilizables en otros proyectos de carácter similar.

Como resultado final se ha obtenido un juego de minigolf sobre un escenario 3D con una simulación física real. Un juego online multijugador divertido, el cual trascurre en tiempo real, con una gran variedad de pistas y objetos que cambian de lugar de un

juego a otro, además de sugerentes animaciones, que animan a todo tipo de jugador a pasar un buen rato frente la pantalla de su ordenador.

Todo ello sin olvidar ofrecer a todos los jugadores la posibilidad de configurar a su gusto cuántas pistas desea jugar, el tiempo extra que se aplica a cada pista cuando un jugador consigue introducir su bola en el hoyo y la posibilidad de anular las colisiones durante el juego.

En el próximo capítulo se van a presentar, mediante una breve descripción, otros dos juegos de carácter similar al juego descrito en esta sección para posteriormente realizar una comparación, en base al coste de desarrollo de cada uno de ellos, y realizar así un estudio de la aportación del motor de producción durante la fase de desarrollo de este tipo de juegos.

En el capítulo 7 se desarrollan las conclusiones finales de esta tesina y se presentan algunas líneas de trabajo futuro en base a todo el estudio realizado.

Capítulo 6

Análisis crítico del motor de producción

6.1 Introducción

Una vez presentado el juego del "Minigolf" desarrollado empleando el motor de producción como base en su programación, en el presente capítulo se va a presentar, mediante una breve descripción otro juego de carácter similar a éste para, posteriormente realizar un estudio de los costes de desarrollo de los dos videojuegos con el objetivo de comparar los resultados obtenidos.

Tras dar en la sección 6.2 una breve descripción del otro juego mencionado, en la sección 6.3 se realiza un estudio de los costes de producción de cada uno de ellos. Los costes de producción han sido medidos en base al tiempo de desarrollo, las líneas de código y el coste económico de cada proyecto. Por último, en la sección 6.4 se exponen las conclusiones de este capítulo.

6.2 Descripción del otro proyecto

Además del juego visto, el otro juego que se va a ver y con el que se van a realizar las comparaciones es: "Superbuteo".

El juego "Superbuteo" está basado en el conocido juego de mesa "Subbuteo" o fútbol de mesa y consiste en disputar un partido de fútbol mediante una mecánica de juego muy similar a la del popular juego de "Las chapas". El objetivo del juego es

conseguir marcar más goles a tu oponente. Para ello se dispone de once piezas que simulan los diez jugadores y el portero.

Se trata de un juego por turnos, en el que cada usuario dispone de un tiempo limitado para realizar sus movimientos. Además dispone de una barra de movimientos que decrece con cada movimiento ejecutado. Durante un turno, el usuario podrá realizar tantos movimientos como la barra de movimientos permita, durante el tiempo establecido para cada turno. Finalizará su turno si acaba agota la barra de movimientos o si finaliza el tiempo de turno.

Este juego fue desarrollado por la empresa Exeleweiss Ent. antes de realizarse el desarrollo del motor de producción. Su desarrollo siguió un modelo de producción ad-hoc, desarrollando cada función y módulo de forma individual y centrándose en las características y necesidades del propio juego. Por ejemplo, a diferencia del juego del "Minigolf", el motor de física fue desarrollado íntegramente desde cero para este juego en particular.

Al igual que el juego del "Minigolf", el "Superbuteo" es un juego de carácter online multijugador, desarrollado sobre un escenario 3D y dotado de una simulación física real gracias al uso de un motor de física.

6.3 Estudio de los costes de producción

Una vez presentados los dos juegos, los cuales, como se ha podido observar, presentan una serie de características de carácter muy similar, es el momento de analizar, entrando un poco más en detalle, las posibilidades que ofrece el motor de producción.

El objetivo de esta sección es estudiar la aportación del motor de producción que se ha presentado en la sección 4, durante la fase de desarrollo de un videojuego de las características anteriormente presentadas, es decir, para juegos del tipo online multijugador, que se desarrollan sobre un escenario en tres dimensiones y que hacen uso de un motor de física, ya sea un propio o uno de los que ya integra Director. Para ello se va a realizar un estudio de los costes de producción de los dos juegos presentados.

Para la realización de ambos juegos ha sido necesario un equipo de desarrollo compuesto por un diseñador-grafista y un programador junior, este último dedicado a jornada completa al desarrollo propio del juego. Por otro lado, para el desarrollo del

motor de producción, se ha contado con un equipo de dos programadores juniors también dedicados a jornada completa al desarrollo del mismo.

Por último, y antes de empezar con las comparaciones de ambos juegos, hay que recordar que el juego del “Superbueto” ha sido desarrollado con anterioridad al desarrollo del motor de producción, por lo que no hace uso de este, mientras que el juego del “Minigolf” se ha desarrollado utilizando como base para su desarrollo el motor de producción.

A continuación, en la Figura 6, se muestra una imagen de cada uno de los juegos durante su ejecución. Como se puede observar, a nivel gráfico las diferencias son insignificantes. Respecto el rendimiento de los juegos en funcionamiento, la cual se mide en frames por segundo (FPS), se puede observar que, en los resultados obtenidos tanto para el juegos del “Superbueto” como para el “Minigolf”, la tasa de frames por segundo oscilan entre 23 y 30 FPS. Esta variación se debe principalmente al número de polígonos dibujados en pantalla, a la simulación física del juego y a las animaciones existentes en los juegos. Es muy importante tener en cuenta estos datos y evitar tasas de frames menores a 15 FPS, porque a menor número de frames registrado, peor es el rendimiento del juego pudiendo crear una gran desventaja durante el juego.

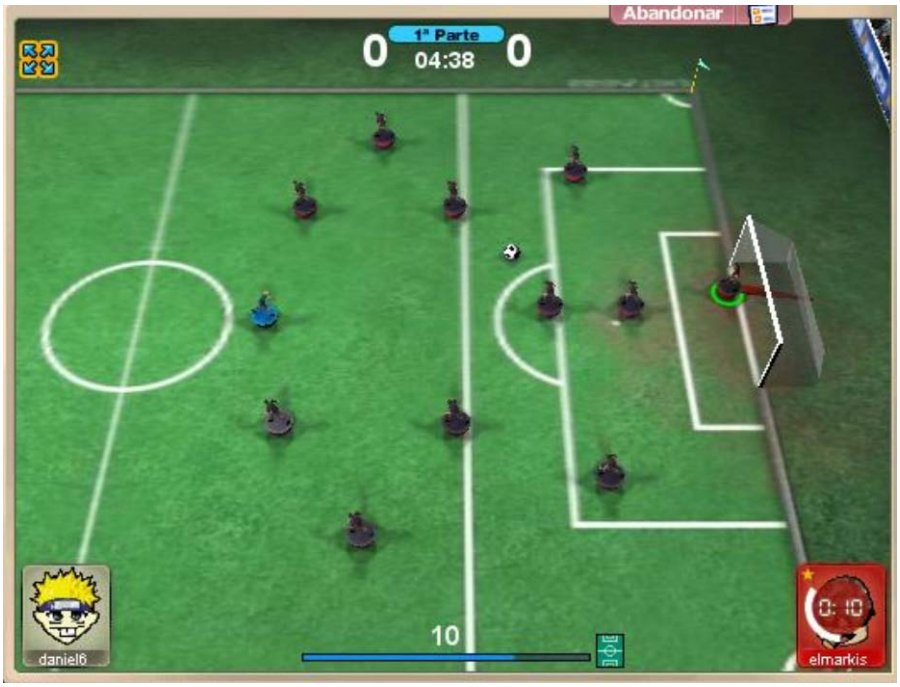

Figura 7. Videojuegos en ejecución.

Para poder analizar mejor el coste de desarrollo de ambos juegos se han realizado una serie de tablas atendiendo a las líneas de código escritas para cada juego, el tiempo que ha sido necesario para llevar a cabo cada desarrollo, incluyendo la fase de testeo, y por último se ha hecho una estimación del coste económico resultante. Dicha estimación económica se ha calculado en base a los datos proporcionados por la empresa Exelweiss Ent. atendiendo a los costes que supone para ellos mensualmente la realización del desarrollo de un videojuego de estas características.

Para realizar este análisis y debido a que esta tesina se centra en la parte de codificación de los proyectos incluida en la fase de producción, solo se ha cuantificado el trabajo realizado por el programador del equipo de desarrollo.

Coste medido en líneas de código escritas

En primer lugar se analiza el número de líneas de código que han sido escritas por el programador para el desarrollo de cada uno de los tres proyectos. Para ello se ha hecho uso de los siguientes valores:

- LC Específicas: número de líneas que corresponden específicamente a cada uno de los distintos proyectos. Estas líneas de código son aquellas que el programador ha escrito desde cero para llevar a cabo el desarrollo de cada aplicación.

- LC Total: número total de líneas en los diferentes proyectos. Gracias a este valor se puede cuantificar de algún modo el tamaño de cada proyecto.

Nº DE LÍNEAS DE CÓDIGO			
	Superbuteo	**Minigolf**	**Motor de producción**
LC Específicas	11640	4660	0
LC Total	11640	15040	11380

Tabla 1. Medida de los costes de producción medido en líneas de código

Los datos que se observan al analizar la Tabla 1 es que en número de líneas total codificadas por un programador (hay que recordar que el motor de producción ha sido desarrollado por dos programadores junior) son casi las mismas para el juego del "Superbueto" y para el motor de producción. Si comparamos solo los juegos, el juego del "Minigolf", haciendo uso del motor de producción, tiene un número de líneas total mayor. Con estos resultados se puede concluir que el uso del motor de producción genera proyectos de mayor tamaño a los proyectos desarrollados siguiendo una metodología ad-hoc.

La similitud en número de líneas entre el juego del "Superbueto" y el motor de producción se debe al carácter generalista este último y al uso de una metodología ad-hoc para el desarrollo del juego. En el caso del motor, puesto que debe ser capaz de integrar distintos motores y ser lo más flexible posible requiere de una programación más genérica, para abarcar el más amplio abanico de juegos posible. En el caso del juego, puesto que se ha desarrollado todo el código de forma específica a las características y demandas del juego, se ha podido reducir sus líneas de código y optimizarlas. Por tanto, es necesario tener especial cuidado con el uso que se da al motor de producción y aprovechar de la mejor forma posible, todas la funcionalidades que ofrece, puesto que de no ser así, se obtendrá como resultado un juego que ocupe más espacio del debido y posiblemente se codificarán más líneas de las necesarias.

Un ejemplo del caso contrario al que se hacía mención en las líneas anteriores es el juego del "Minigolf", el cual, como ya se pudo observar en la sección 5, aprovecha al máximo las funcionalidades ofrecidas por el motor de producción. Buena prueba de ello son las 4660 líneas de código nuevas introducidas por el programador para el desarrollo

específico del juego, frente a las 11640 del juego del “Superbuteo”, reduciendo en un 59.97% el número de líneas de código nuevas escritas por el programador, es decir, para cada desarrollo nuevo, se ahorra más de un 50% del código nuevo escrito. Como ya se ha mencionado en varias ocasiones, el motor de producción proporciona una buena base sobre la que empezar un nuevo proyecto, además de integrar un gran número de funcionalidades muy útiles, las cuales, no necesitan ser programadas de nuevo para cada proyecto, reduciendo así el número de líneas nuevas que han de ser escritas.

Coste temporal

El estudio del coste temporal de los tres proyectos se ha basado en los siguientes tres parámetros:

- FP: coste temporal, medido en semanas, que corresponde a la fase de producción. Tiempo que ha necesitado un programador junior para la codificación de cada proyecto en su fase de producción.

- FT: coste temporal, medido en semanas, que corresponde con la fase de testeo, necesaria para comprobar que las aplicaciones funcionan correctamente.

- TTotal: coste temporal, medido en semanas, de producción. Es la suma de los dos parámetros anteriores, coste temporal de la fase de producción más coste temporal de la fase de test.

COSTE TEMPORAL			
	Superbuteo	**Minigolf**	**Motor de producción**
FP	40 semanas	19 semanas	54 semanas (27*2 semanas)
FT	2 semanas	1 semanas	2 semanas (1*2 semanas)
TTotal	42 semanas	20 semanas	56 semanas (28*2 semanas)

Tabla 2. Medida de los costes temporales de las implementaciones desarrolladas

Comparando únicamente los dos juegos, se puede observar una notable reducción del coste temporal necesario para el desarrollo del juego del “Minigolf”, el cual ha reducido sus costes en un 52.38%. Debido a las múltiples funcionalidades desempeñadas por el motor, y que han sido utilizadas durante la fase de desarrollo del

juego del “Minigolf” ha sido posible realizar este proyecto en un tiempo inferior al juego del “Superbueto”.

Por otro lado, si se tiene en cuenta los datos obtenidos en la Tabla 1 donde se hacía referencia al número de líneas de código escritas por el programador, se puede observar que aun teniendo casi el mismo número de líneas el juego del “Superbueto” (11640 LC Total) y el motor de producción (11380 LC Total), este ha necesitado de un mayor tiempo para su desarrollo completo, siendo un 25% mayor el coste temporal respecto al juego. Esto es debido, al igual que el número de líneas, a su carácter genérico, ya que este tipo de proyectos resultan ser más costosos.

Llegados a este punto, se ha observado una reducción en el juego que ha utilizado el motor de producción como base para su desarrollo, tanto en tamaño como en tiempo necesario para su desarrollo. Además esta reducción a resultado ser directamente proporcional, lo que lleva a la conclusión que, el correcto uso del motor resulta realmente beneficioso para este tipo de desarrollos.

Coste económico

El coste económico se ha calculado en función de los gastos que ha supuesto a la empresa Exelweiss Ent. llevar a cabo los tres proyectos. Este parámetro ha sido calculado en base al tiempo total necesario para realizar cada uno de los desarrollos (TTotal), y el coste que supone mensualmente a la empresa realizar este tipo de aplicaciones con el equipo mencionado al inicio de esta sección.

COSTE ECONÓMICO			
	Superbuteo	**Minigolf**	**Motor de producción**
Euros	84.000	40.000	112.000

Tabla 3. Medida del coste económico de las implementaciones desarrolladas

Después de ver los datos mostrados en la Tabla 3, puede llevar a pensar que, dado el alto coste económico que resulta solo el desarrollo del motor de producción, no resulte una buena inversión. En cambio, si analizamos únicamente el coste de los dos juegos, se puede observar como sigue la tendencia a la baja en el juego del “Minigolf”. En este caso la reducción es del 47.62%, es decir, se ha producido un ahorro de 44.000€ para la empresa. Si observamos la siguiente tabla se puede observar cómo, con el

tiempo, y realizando más proyectos similares al juego del “Minigolf”, con el desarrollo del tercer juego, el motor de producción se amortiza.

COSTE ECONÓMICO			
	Primer juego	**Segundo juego**	**Tercer juego**
Sin motor de producción	84.000€	168.000€	252.000€
Con motor de producción	152.000€ (112.000€ + 40.000 €)	192.000€	232.000€

COSTE TEMPORAL			
	Primer juego	**Segundo juego**	**Tercer juego**
Sin motor de producción	42 semanas	84 semanas	126 semanas
Con motor de producción	76 semanas (56+20 semanas)	96 semanas	116 semanas

Tabla 4. Evolución del coste económico y temporal

En 116 semanas aproximadamente, la empresa empezará a ver los beneficios que aporta, económicamente, el motor de producción en el que ha invertido su tiempo y su dinero.

6.4 Conclusiones

Después de haber mostrado como un juego en su fase de desarrollo, hacía uso del motor de producción desarrollado en el capítulo 4, en el presente capítulo se ha presentado otro juego de características muy similares desarrollado con anterioridad al desarrollo del motor de producción, para poder realizar un estudio de la aportación del motor de producción en el desarrollo de juegos casuales online multijugador en 3D con simulación física. El primero de ellos se trata del juego “Minigolf” el cual se ha apoyado en el motor de producción para su desarrollo, y el segundo juego es el “Superbueto”, juego desarrollado íntegramente de cero utilizando un modelo de producción ad-hoc.

Después de analizar los datos obtenidos se llega a la conclusión que aquellos juegos que utilizan el motor de producción como base en su desarrollo, aunque su número de líneas es un tanto mayor que para los juegos desarrollados desde cero,

reducen el número de líneas nuevas codificadas por el programador, en este caso, un 59,97%.

Por otro lado, puesto que la base del juego ya se ha sido correctamente testeada, la fase de test se ve también reducida, siendo testeada únicamente estas líneas de código nuevas. Todo ello supone una importante reducción en el coste temporal del desarrollo de un videojuego utilizando el motor de producción. Para el caso del juego del "Minigolf" se ha reducido en un 52.38%.

Todas estas mejoras y reducciones de costes suponen al final a la empresa desarrolladora un ahorro del 47.62% del coste económico que este tipo de desarrollos supone, una cifra que asciende, para el caso del juego del "Minigolf" a los 44.000€ de ahorro.

Además, se ha demostrado que la inversión realizada por la empresa se amortiza con el tercer juego desarrollado, cubriendo así los objetivos presentados en el capítulo 3 que persiguen las empresas, es decir, reducen costes, consiguen desarrollar juegos de calidad en poco tiempo pudiendo sacar al mercado juegos antes que la competencia y aceptan desarrollos que en otras ocasiones, por restricciones temporales, no hubiesen podido aceptar, lo que supone en conjunto, una oportunidad de negocio para estas y mejor márgenes de beneficio.

Por todo ello, se concluye que el uso del motor de producción desarrollado permite desarrollar juegos casuales online multijugador en 3D con simulación física en un corto periodo de tiempo y a un bajo coste para las empresas desarrolladoras.

Capítulo 7

Conclusiones y líneas de trabajo futuro

7.1 Conclusiones generales

La industria de los videojuegos es un campo muy atractivo para mucha gente y desde hace unos años puedo afirmar que formo parte de ella. Mi propia experiencia como desarrolladora en la empresa *Exelweiss Ent.* me ha llevado a plantearme una serie de cuestiones. Esta pequeña empresa cuenta con un producto estrella, el portal *www.mundijuegos.com*, un portal de juegos online multijugador en español, que pronto estará disponible en otros idiomas, que actualmente es líder en España y que cuenta con profesionales cuya experiencia en el sector y en este tipo de desarrollos asciende a más siete años.

Durante la fase de producción nos encontramos con una serie de tareas repetitivas y problemas que sabemos no solo nos ocurre a nosotros. Prueba de ello es la experiencia de otras empresas similares cuyos ingresos provienen también de los juegos casuales, con los cuales se ha tenido la ocasión de poder tratar algunos de estos problemas a través de foros especializados (*www.stratos-ad.com*) y distintas reuniones y congresos de gran interés, como es el CDV.

Como conclusión a todo ello se han planteado una serie de objetivos con el fin de poder encontrar la mejor solución a estos problemas. A continuación se hace una pequeña clasificación para poder observar con mayor claridad cómo ha sido tratado cada uno de ellos a lo largo de este trabajo.

OBJETIVO 1. Estudio de la metodología de desarrollo de un videojuego casual online multijugador y de las herramientas utilizadas.

Situándose des del inicio de esta tesina en el ámbito de la pequeña empresa desarrolladora de videojuegos, se ha analizado en el capítulo 3 como dichas empresas debido a la escasez de recursos de que disponen la gran mayoría acaba desarrollando sus proyectos siguiendo una metodología ad-hoc.

Este tipo de metodología conlleva un desarrollo desde cero en todos los proyectos siendo desarrollos muy específicos y muy bien optimizados, pero a largo plazo resultan costosos, puesto que cada proyecto necesita más tiempo tanto en su fase de producción y como en el testeo. Los desarrolladores se encuentran ante tareas repetitivas, tanto a la hora de desarrollar las funcionalidades necesarias en cada videojuego como a la hora de detectar y resolver errores.

Pero dicha metodología no es la única existente, el uso de motores y librerías puede resultar muy beneficioso a la hora de llevar a cabo sus proyectos, siempre y cuando dichos motores se utilicen adecuadamente en función de cada desarrollo. Los motores y librerías encapsulan funcionalidades que resultan comunes en determinados proyectos de características similares, y gracias a ellos dichas funciones permiten ser reutilizadas en cada proyecto sin necesidad de entrar en los detalles de la función. A su vez, dichas funciones están correctamente testeadas, por lo que los errores ya han sido detectados y solucionados. De este modo, los desarrolladores solo se centran en el código nuevo escrito, reduciendo considerablemente su tiempo y esfuerzo.

No solo es importante el modo de desarrollar un videojuego, hay que tener en cuenta sobre que plataforma va a ser ejecutado y sobre que lenguaje será desarrollado. Dada las condiciones del tipo de videojuego que se desea desarrollar en esta tesina y después de realizar un análisis de las distintas plataformas y tecnologías existentes visto en el capítulo 2, se decide realizar videojuegos para la plataforma online ejecutada sobre ordenador, dado su amplia accesibilidad hoy en día, y utilizar como tecnología para su desarrollo Shockwave, el cual proporciona la herramienta de autor Adobe Director para llevar a cabo los distintos contenidos multimedia.

Utilizando Shockwave se ofrece al desarrollador el más amplio abanico de posibilidades, gracias a que dicha tecnología soporta y permite trabajar con elementos en 3D. Por otro lado, se trata de una tecnología que se ejecuta directamente sobre el navegador, evitando que el usuario tenga que instalar ningún programa en su ordenador, y además tiene un nivel de penetración de un 50%, lo que significa que la mitad de los

navegadores tiene ya instalado el plugin necesario para el funcionamiento de aplicaciones desarrolladas bajo Shockwave.

OBJETIVO 2. Detección de carencias y aspectos mejorables en el proceso de producción y propuestas para solucionarlas.

La herramienta Adobe Director proporciona a los desarrolladores una serie de motores para el desarrollo de distintos proyectos multimedia. Para el desarrollo de un videojuego son muchos los motores que pueden ser utilizados para ayudar a los desarrolladores en la fase de producción.

Como se ha analizado en el capítulo 3 la integración de dichos motores en los distintos proyectos puede resultar una tarea a veces complicada debido a la integración ad-hoc de dichos motores en cada videojuego nuevo. Esto conlleva que el desarrollador, no solo tenga que estudiar cómo funciona cada motor que desea integrar sino, como debe integrarlo.

Por otro lado, aunque los motores se caracterizan por su fiabilidad y robustez es posible encontrar algún tipo de inconsistencia, ya sea a la hora de integrarlo con otros motores o con el propio proyecto que se esté desarrollando. Todo ello ocasiona errores que deben ser resueltos de forma específica para cada nuevo desarrollo.

Por último, aunque son muchos los motores que ofrece Adobe Directos para el desarrollo de proyectos multimedia, se ha detectado la inexistencia de un motor cuya funcionalidad sea la de abstraer aquellas funciones que resultan comunes en la propia estructura y ejecución de un videojuego, así como la inicialización del mismo.

Para poder paliar dichos inconvenientes y carencias se ha propuesto el desarrollo de un motor de producción que proporcione la mejor solución posible. Dicho motor deberá realizar las funciones pertinentes de configuración e inicialización de un videojuego, permitiendo configurar, en medida de lo posible y de una manera externa al propio motor, todos aquellos datos necesarios para la posterior inicialización y ejecución del videojuego. Además deberá encapsular aquellos motores y librerías que puedan resultar de mayor utilidad para futuros desarrollos, creando una capa de un nivel de abstracción mayor que los englobe. Dicha capa es conocida también como *Middleware.*

OBJETIVO 3. Desarrollo del motor de producción como solución.

Una vez expuesta la filosofía de diseño seguida para el desarrollo del motor de producción propuesto como solución, es en el capítulo 4 donde se analiza y explica con detalle cada una de las partes de dicho motor.

- *Configuración externa*: el objetivo de esta sección del motor es poder configurar una serie de parámetros del videojuego de una manera externa al juego en sí, de manera que cualquier persona del equipo, aun careciendo de conocimientos técnicos, sea capaz de cambiar algunas de las características del juego. Hay una gran cantidad de parámetros que pueden ser configurables, tanto de los motores que se van a utilizar como características del propio juego. Para ello se ha hecho uso del estándar XML para definir un modelo de aplicación que resulte fácil de entender y rápido de manejar, además de un parser que permite transformar los datos introducidos en el fichero XML en datos válidos comprensibles por la aplicación.

- *Motor de inicialización*: se encarga de inicializar correctamente el juego antes de proceder a su ejecución. Dicha inicialización corresponde con una serie de funciones que resultan comunes en la estructura de ejecución interna del propio videojuego. Gracias al motor de inicialización el proceso de inicialización de los datos, entidades de juego y motores necesarios para el correcto funcionamiento del videojuego, así como el bucle principal del juego, son realizados de una forma automática y transparente al desarrollador, pudiendo conseguir además, con la ayuda de algunos de los parámetros definidos durante la fase de configuración, distintas inicializaciones.

- *Middleware*: se trata de una capa de nivel de abstracción mayor que encapsula una serie de motores proporcionados por la herramienta de autor Adobe Director y que resultan de utilidad para el desarrollo de videojuegos. Además de ha integrado la herramienta MUS, la cual ha sido desarrollada por la empresa Exelweiss Ent. durante sus inicios y que proporciona algunas funcionalidades de comunicación entre usuarios. El Middleware actúa como intermediario entre los distintos motores que puede integrar, facilitando una serie de funciones que resultan fáciles de entender y rápidas de utilizar, independientemente del motor que se utilice en la capa inferior. Además durante su fase de creación han sido resueltos algunos problemas de inconsistencia entre los distintos motores para evitar que los desarrolladores tengan que preocuparse por ellos en un futuro [1].

Aunque no forma parte de los principios definidos en la filosofía de diseño que sigue el motor de producción, durante su desarrollo se han encontrado una serie de

funcionalidades que se ha considerado que pueden resultar de gran utilidad en muchos proyectos. Dichas funcionalidades adicionales son:

- Automatización en el uso de proxys.
- Skydome, creación de escenarios abiertos.
- Scripting desde la herramienta 3D Studio Max.
- Creación de modelos siempre visibles.
- Tratamiento de objetos aleatorios.

OBJETIVO 4. Desarrollo de un videojuego utilizando el motor de producción como base en su desarrollo.

Una vez finalizado el desarrollo del motor de producción, la mejor forma de valorar su efectividad es desarrollar un videojuego utilizando el propio motor como base de su producción. Es importante la elección del juego, el cual debe tener un serie de características que permita aprovechar al máximo las funcionalidades que ofrece el motor de producción, de lo contrario, puede no resultar beneficioso su uso.

Para sacar el mayor rendimiento al motor de producción desarrollado es conveniente que el juego sea en 3D, online y que esté dotado de una simulación física real, debido a los motores y funcionalidades que han sido integrados e implementados.

A lo largo del capítulo 5 se muestra con detalle como el desarrollo de un conocido juego como es el “Minigolf” aprovecha de la mejor manera las funcionalidades ofrecidas por el motor de producción. Durante el capítulo no se ha entrado en detalle en el propio código del juego, sino que se ha intentado exponer, de la mejor manera posible y siguiendo la misma estructura que en el capítulo 4, aquellos aspectos importantes durante el desarrollo del juego referentes al uso del motor de producción.

Debido al carácter y contenido de esta tesina, la cual comparte desarrollo con [1] como se ha podido observar durante todo el trabajo, en el capítulo 5 se ha hecho hincapié únicamente en las partes del motor de producción desarrolladas correspondientes a esta tesina, teniendo que dejar a un lado las ventajas ofrecidas por algunas de las otras funcionalidades que el motor ofrece o por el propio Middleware.

OBJETIVO 5. Valoración del motor desarrollado.

Antes de adentrase en el capítulo 6, el lector puede hacerse una idea de los beneficios que aporta el uso del motor de producción desarrollado durante la fase de producción de un videojuego, pero a veces es necesario algunos datos más para verificarlo con más certeza.

En el capítulo 6 se realiza un estudio más exhaustivo del uso del motor de producción en términos de costes de producción. Para ello se ha presentado un nuevo juego, el "Superbueto", el cual fue desarrollado con anterioridad al motor de producción por la empresa Exelweiss Ent. y que además presenta una características similares al juego de "Minigolf" desarrollado con el motor de producción.

El análisis ha sido cuantificado en base a el coste temporal de ambos proyectos, el número de líneas codificadas por el programador y el coste económico que supuesto a la empresa.

Respecto el coste temporal se puede observar que, mientras que el juego "Superbueto" ha necesitado 42 semanas para su desarrollo, el juego de "Minigolf" se ha sido desarrollado empleando únicamente 20 semanas, lo que supone un ahorro de tiempo del 52.38%.

Por otro lado, aunque el número de líneas totales del juego "Minigolf" sea un poco mayor que para el juego de "Superbuteo" si se analiza el trabajo realizado por el programador se observa una reducción de líneas de código nuevas escritas de un 59.97%.

Por último y de una forma directamente proporcional a las reducciones ya comentadas, la reducción del coste económico asciende a la una cantidad de 44.000€, es decir, desarrollar el juego de "Minigolf" ha supuesto un ahorro del 47.62% frente al coste de desarrollo del juego "Superbuteo".

Además de estos análisis entre juegos se ha mostrado además el coste de desarrollo del propio motor de producción en base a los mismos parámetros y cuánto costaría a la empresa empezar a obtener beneficios de la inversión realizada en el desarrollo del motor, quedando amortizado con el desarrollo del tercer juego y cubriendo así los objetivos expuestos en el capítulo 3 que persiguen las empresas:

- Reducción de costes.
- Desarrollo de juegos de calidad en un menor espacio de tiempo.

- Aceptar más desarrollos. Nuevas oportunidades de negocios.
- Mayor margen de beneficios.

7.2 Líneas de trabajo futuro

A lo largo de esta tesina se ha presentado brevemente aspectos básicos referentes a la industria del ocio digital, la cual, como se ha podido comprobar se encuentra en pleno auge.

El resultado final ha sido el desarrollo de un motor de producción que ayuda a paliar algunos aspectos clave a la hora adentrarse en la producción de un nuevo producto, como es el videojuego casual online multijugador.

A partir de los datos y resultados mostrados en esta tesina se pueden observar una serie de líneas de trabajo que en un futuro podrían llegar a ser una realidad.

1. Desarrollo sobre plataformas y herramientas alternativas

El mundo de los videojuegos es un campo muy amplio y diverso. Como se ha podido observar en el capítulo 2 existen una gran cantidad de plataformas sobre la que poder ejecutar este tipo de proyectos, y un sin fin de lenguajes con sus correspondientes herramientas para su desarrollo en función de la plataforma elegida.

Esta tesina se ha centrado en aquellas empresas del sector que empiezan a hacerse un hueco en esta gran industria, las cuales disponen de un presupuesto reducido y un pequeño equipo de desarrollo, llevándolas a centrarse en pequeños proyectos a los que poder hacer frente, como son los juegos casuales online multijugador.

Una posible línea de trabajo futuro es migrar el motor presentado a otras plataformas o herramientas. Esto es posible debido a la estructura interna que presentan los videojuegos. Para poder realizar dicha migración bastaría con escoger una nueva plataforma con su correspondiente tecnología con sus respectivos motores y librerías, extraer aquellos parámetros configurables y trasladarlos al correspondiente modelo de aplicación. El proceso de desarrollo sigue la misma estructura que se ha seguido para desarrollar el motor de producción presentado.

Por otro lado, podría realizarse la migración únicamente cambiando de tecnología y utilizando la misma plataforma. Por ejemplo, puede realizarse un motor de características similares empleando cualquiera de las tecnologías vistas en el capítulo 2.

2. Extender las características del entorno de configuración externa incluido en el motor de producción desarrollado

El motor de producción desarrollado, además de incluir un motor de inicialización propio y de proporcionar la posibilidad de actuar como *Middleware*, integrando varios motores en uno solo, permite la configuración externa de ciertos parámetros. Como se ha mostrado, se pueden integrar una gran variedad de datos, desde parámetros propios de la física del juego hasta cualquier tipo de fichero que se ha necesitado para este tipo de proyecto, como son los distintos scripts que se integran o los ficheros w3d.

Por tanto, se propone como línea de trabajo futuro ampliar el abanico de posibilidades que ya ofrece el motor de producción, pudiendo incluir otro tipo de ficheros, o permitir la configuración de cualquier otro valor que pueda resultar de utilidad para futuros proyectos.

Anexo A

Modelo de aplicación. Fichero XML

En este anexo se presentan un par de ejemplos de modelos de aplicación. El primero de ellos es un ejemplo de un XML de configuración general, donde se puede apreciar con más detalle la estructura seguida a la hora de configurar los parámetros generales del juego. Se divide en tres grandes grupos principalmente, encapsulados en una etiqueta general <*config*>.

La primera parte define aquellos aspectos más generales, como son, datos generales del juego, los directorios donde se encuentran las entidades y los scripts y las propiedades de la física generales.

```
<?xml version="1.0" encoding="utf-8"?>
<config>
	<game>
		<name>minigolf</name>
		<script>MiniGolfGame</script>
		<initFunctions>[1]</initFunctions>
	</game>

	<use_mus>true</use_mus>

	<dir>
		<objects>Scripts\Objects\</objects>
		<scene>Scripts\W3d\</scene>
		<overlays>Scripts\Overlays\</overlays>
		<cameras>Scripts\Cameras\</cameras>
		<rules>Scripts\Rules\</rules>
```

```
        <game>Scripts\Game\</game>
        <animation>Scripts\W3d\</animation>
        <sounds> Scripts\Sounds\</sounds>
        <default>Scripts\</default>
</dir>

<scripts>
        <objects>
                <script>Ball</script>
                <script>Group</script>
                <script>Move</script>
                <script>Random</script>
                <script>Club</script>
                <script>Arrow</script>
                <script>DefaultObject</script>
                <script>DefaultRigidObject</script>
        </objects>
        <overlays>
                <script>OvShot</script>
                <script>OvScore</script>
        </overlays>
</scripts>

<globals>
        <physics>
                <engine>ageia</engine>
                <script>Physics</script>
                <linearDamping>0.75</linearDamping>
                <angularDamping>1.5</angularDamping>
                <contactTolerance>0.1</contactTolerance>
                <friction>0.2</friction>
                <gravity>vector(0,0,-980)</gravity>
                <restitution>0.2</restitution>
                <sleepMode>#energy</sleepMode>
                <sleepThreshold>0.5</sleepThreshold>
                <subSteps>20</subSteps>
                <timeStep>0.033</timeStep>
                <timeStepMode>#equal</timeStepMode>
                <scalingFactor>vector(0.01,0.01,0.01)</scalingFactor>
        </physics>
        <animation>true</animation>
```

```
</globals>
```

La segunda parte define algunos de los distintos objetos que se pueden encontrar en el juego. En este caso se muestran animaciones y objetos con y sin cuerpo rígido.

```
<specific>
	<w3d>
		<object>anm_circular</object>
		<object>anm_ocho</object>
		<object>bandera</object>
	</w3d>
	<objects>
		<club>
			<script>Club</script>
			<specialFeatures>
				<type>clonePlayer</type>
				<order>2</order>
			</specialFeatures>
		</club>
		<bandera>
			<script>DefaultObject</script>
		</bandera>
		<flecha>
			<script>Arrow</script>
		</flecha>
	</objects>
	<rigidObjects>
		<default>
			<rigidBody>
				<shape>#concaveShape</shape>
				<type>#static</type>
			</rigidBody>
			<physics>
				<contactTolerance>0.1</contactTolerance>
				<friction>0.1</friction>
				<isPinned>true</isPinned>
				<isSleeping>true</isSleeping>
				<mass>1.0</mass>
				<orientation>[vector(1,0,0),0]</orientation>
				<restitution>0.6</restitution>
				<axisAffinity>false</axisAffinity>
```

```
        </physics>
        <script>DefaultRigidObject</script>
</default>
<ball>
        <rigidBody>
                <shape>#sphere</shape>
                <type>#dynamic</type>
        </rigidBody>
        <physics>
                <angularDamping>1.2</angularDamping>
                <angularMomentum>vector(0,0,0)</angularMomentum>
                <angularVelocity>vector(0,0,0)</angularVelocity>
                <centerOfMass>vector(0,0,0)</centerOfMass>
                <contactTolerance>0.01</contactTolerance>
                <friction>0.13</friction>
                <linearDamping>0.5</linearDamping>
                <linearMomentum>vector(0,0,0)</linearMomentum>
                <linearVelocity>vector(0,0,0)</linearVelocity>
                <mass>0.0459</mass>
                <orientation>[vector(1,0,0),0]</orientation>
                <restitution>0.5</restitution>
                <sleepMode>#energy</sleepMode>
                <sleepThreshold>1.05</sleepThreshold>
                <isPinned>false</isPinned>
                <isSleeping>true</isSleeping>
                <axisAffinity>false</axisAffinity>
        </physics>
        <script>Ball</script>
        <specialFeatures>
                <type>clonePlayer</type>
                <order>1</order>
        </specialFeatures>
</ball>

<lane>
        <rigidBody>
                <shape>#concaveShape</shape>
                <type>#static</type>
        </rigidBody>
        <physics>
                <contactTolerance>0.05</contactTolerance>
```

```
                <friction>0.85</friction>
                <isPinned>true</isPinned>
                <isSleeping>true</isSleeping>
                <mass>1.0</mass>
                <orientation>[vector(1,0,0),0]</orientation>
                <restitution>0.65</restitution>
        </physics>
        <script>DefaultRigidObject</script>
</lane>

<grupo>
        <rigidBody>
                <shape>#concaveShape</shape>
                <type>#static</type>
        </rigidBody>
        <physics>
                <other>default</other>
        </physics>
        <script>Group</script>
        <specialFeatures>
                <type>gru01</type>
        </specialFeatures>
</grupo>
<mov_gironormal>
        <rigidBody>
                <shape>#convexShape</shape>
                 <type>#dynamic</type>
        </rigidBody>
        <physics>
                 <contactTolerance>0.01</contactTolerance>
                 <friction>0.0</friction>
                 <isPinned>false</isPinned>
                 <isSleeping>false</isSleeping>
                <mass>1.0</mass>
                <restitution>0.6</restitution>
                 <angularDamping>0.0</angularDamping>
                <linearDamping>0.0</linearDamping>
                 <sleepThreshold>0.0</sleepThreshold>
                <axisAffinity>false</axisAffinity>
        </physics>
                <script>Move</script>
```

```
        <specialFeatures>
            <type>mov_gironormal</type>
        </specialFeatures>
 </mov_gironormal>

<move>
        <rigidBody>
            <shape>#concaveShape</shape>
            <type>#static</type>
        </rigidBody>
        <physics>
            <other>default</other>
        </physics>
        <script>Move</script>
        <specialFeatures>
            <type>mov01</type>
        </specialFeatures>
</move>

<random>
        <rigidBody>
            <shape>#concaveShape</shape>
            <type>#static</type>
        </rigidBody>
        <physics>
            <other>default</other>
        </physics>
        <script>Random</script>
        <specialFeatures>
            <type>ale50</type>
        </specialFeatures>
</random>

<ocho>
        <rigidBody>
            <shape>#convexShape</shape>
            <type>#dynamic</type>
        </rigidBody>
        <physics>
            <contactTolerance>0.1</contactTolerance>
            <friction>0.1</friction>
```

```
                        <isPinned>false</isPinned>
                        <isSleeping>false</isSleeping>
                        <mass>1.0</mass>
                        <restitution>0.0</restitution>
                    </physics>
                    <script>DefaultRigidObject</script>
                    <specialFeatures>
                        <proxy>false</proxy>
                        <animation>ocho</animation>
                    </specialFeatures>
                </ocho>

        </rigidObjects>
</specific>
```

La última sección define los objetos utilizados en la interfaz de juego.

```
<interface>
        <script>Interface_minigolf</script>
        <score>
                <id>Score</id>
                <createOverlay>true</createOverlay>
                <coordinates>point(0,0)</coordinates>
                <staticText>
                        <name>score</name>
                        <text>value</text>
                        <coordinates>point(0,0)</coordinates>
                        <properties>
                                <member>
                                        <color>color(255,255,255)</color>
                                        <fontStyle>[#bold]</fontStyle>
                                        <fontSize>12</fontSize>
                                </member>
                                <overlay>
                                        <blend>0</blend>
                                </overlay>
                        </properties>
                </staticText>
                <script>OvScore</script>
         </score>
        <timerUser>
```

```
                <id>timerUser</id>
                <createOverlay>true</createOverlay>
                <coordinates>point(246,390)</coordinates>
                <image>
                        <name>timerUser</name>
                        <coordinates>point(246,390)</coordinates>
                        <properties>
                                <member>
                                        <useAlpha>true</useAlpha>
                                        <alphaThreshold>100</alphaThreshold>
                                </member>
                                <texture>
                                        <width>64</width>
                                </texture>
                        </properties>
                 </image>
                <script>Overlay</script>
        </timerUser>
        <num0>
                 <id>num0</id>
                <createOverlay>false</createOverlay>
                <image>
                         <name>d0</name>
                </image>
        </num0>
        <num1>
                <id>num1</id>
                 <createOverlay>false</createOverlay>
                <image>
                        <name>d1</name>
                </image>
        </num1>

 </interface>
</config>
```

El segundo modelo de aplicación trata la configuración específica de niveles. Se trata de un documento menos extenso, en este caso, en el que se pueden redefinir ciertas propiedades ya definidas en el modelo de aplicación general.

```
<?xml version="1.0" encoding="utf-8" ?>
```

```
<levels>
	<scene>minigolf</scene>
	<rules>generalRules</rules>
	<level01>
		<general>
			<id>01</id>
		 </general>
	</level01>
	<level09>
		<general>
			 <id>09</id>
		</general>
		<redefinitions>
			<mov_gironormal01>
				<rigidBody>
					<shape>#concaveShape</shape>
					 <type>#dynamic</type>
				</rigidBody>
			 </mov_gironormal01>
		 </redefinitions>
	</level09>
 </levels>
```

Como se puede observar se definen dos niveles distintos. El primero de ellos utilizará todos los parámetros generales definidos, mientras que en el segundo se redefine las propiedades del cuerpo rígido correspondiente al modelo *mov_gironormal01*.

En el siguiente ejemplo se define unas reglas de nivel o modo de juego, utilizando las generales como base, además de redefinir algunos parámetros de la física. Este fragmento corresponde al juego “Billares”.

```
<levelb8>
	<general>
		<id>b8</id>
		<rules>pool8</rules>
	</general>
	<physics>
		<subSteps>30</subSteps>
		<timeStep>0.033</timeStep>
	</physics>
</levelb8>
```

Bibliografía

[1] Marcos García Pascual. *Middleware y Motor de inicialización para la mejora de la eficiencia en el desarrollo de juegos casuales online multijugador*. Universidad Politécnica de Valencia, 2010.

[2] Resultados Anuales 2008. Página web de la Asociación Española de Distribuidores y Editores de Software de Entretenimiento: http://www.adese.es/

[3] Página web del Boletín Oficial del Estado, BOE: http://www.boe.es/

[4] 2008-2009 Casual Games White Paper. Página web de la International Game Developers Association: http://www.igda.org/

[5] Encuesta sobre Equipamiento y Uso de Tecnologías de la Información y Comunicación de los hogares 2009. Página web del Instituto Nacional de Estadística (INE): http://www.ine.es/

[6] Estudio de Hábitos de Internet. Información, consumo de medios y redes sociales. Octubre 2008. Página web de Red de Blogs, Ocio Network S.L.: http://www.ocio.net/

[7] Steve Rabin. *Introduction to Game Development*. Charles River Media.

[8] Jeannie Novak. *Game Development Essentials: an introduction, Second Edition*. Thomson Delmar Learning.

[9] Página web oficial de la Asociación Española de Distribuidores y Editores de Software de Entretenimiento (ADeSe): http://www.adese.es/

[10] Anuario - Memoria 2008. Página web de la Asociación Española de Distribuidores y Editores de Software de Entretenimiento: http://www.adese.es/

[11] Usuarios de videojuegos en Europa 2008. Página web de la Asociación Española de Distribuidores y Editores de Software de Entretenimiento: http://www.adese.es/

[12] Página web de Adobe, nivel de penetración de Flash: http://www.adobe.com/products/player_census/flashplayer/version_penetration.html

[13] Página web de Adobe, nivel de penetración de Shockwave: http://www.adobe.com/products/player_census/shockwaveplayer/version_penetration.html

[14] Página web de StatOwl, nivel de penetración de Java: http://www.statowl.com/java.php

[15] Página web de Unity, nivel de penetración de Unity: http://blogs.unity3d.com/2008/03/31/thoughts-on-browser-plugin-penetration/

[16] Erik Bethke. *Game Development and Production*. Wordware Publishing, Inc., 2003.

[17] Ian Millington. *Game Physics Engine Development*. Academic Press, 2007.

[18] David M. Bourg. *Physics for Game Developers*. O'reilly, January 2002.

[19] Brian Schwab. *AI Game Engine Programming*. Charles River Media.

[20] James R. Boer. *Game Audio Programming*. Cengage Learning, 2003.

[21] David H. Eberly. *3D Game Engine Architecture: Engineering Real-time Applications with Wild Magic*. Morgan Kaufmann, 2005.

[22] David H. Eberly. *3D Game Engine Design: A Practical Approach to Real-time Computer Graphics*. Gulf Professional Publishing, 2007.

[23] C. Endres, A. Butz, and A. MacWilliams. A Survey of Software Infrastructures and Frameworks for Ubiquitous Computing. *Mobile Information Systems Journal*, *IOSPress*, 1:41-80, January 2005.

[24] T. Reicher, A. MacWilliams, B. Bruegge, and G. Klinker. Results of a Study on Software Architecture for Augmented Reality Systems. In *Proceedings of the 2nd IEEE and ACM International Symposium on Mixed and Augmented Reality (STARS)*, page 274, Tokio, Japan, October 2003. IEE Computer Society.

[25] Página web de Xerces Java Parser: http://xerces.apache.org/xerces-j/

[26] S. Casas, P. Morillo, J. Gimeno, and M. Fernández. *SUED. An Extensible Framework for the Development of Low-cost DVE Systems.* Instituto de Robótica, Universidad de Valencia.

[27] Perry McDowell, Rudolph Darken, Joe Sullivan, and Erik Johnson. *Delta3D: A Complete Open Source Game and Simulation Engine for Building Military Training Systems.* MOVES Institute, Naval Postgraduate School.

[28] Ayuda de Adobe Director 11.5: http://help.adobe.com/en_US/Director/11.5/UsingScripting/index.html

[29] Rick Hall, Jeannie Novak. *Game Development Essentials: Online Game Development*. Delmar Cengage Learning.

[30] William Muehl, Jeannie Novak. *Game Development Essentials: Game Simulation Development*. Thomson Delmar Learning.

[31] Kevin Saunders, Jeannie Novak. *Game Development Essentials: Game Interface Design*. Thomson Delmar Learning.

[32] John B.Ahlquist, Jr., Jeannie Novak. *Game Development Essentials: Game Artificial Intelligence*. Thomson Delmar Learning.

[33] Troy Dunniway, Jeannie Novak. *Game Development Essentials: Gameplay Mechanics*. Delmar Cengage Learning.

[34] Thor Alexander. *Massively Multiplayer Game Development 2*. Charles River Media.

[35] Andrew Rollings, Dave Morris. *Game Architecture and Design: A New Edition*. New Riders, November 2003.

[36] Gary Rosenzweig. *Advanced Lingo for Games*. Hayden Books. April 2000.

[37] Paul Catanese. *Director's Third Dimension. Fundamentals of 3D Programmin in Director 8.5*. Pearson Education, 2002.

[38] Jason Roberts. *Curso oficial de Lingo. Actualizado a la versión 6 de Director*. Anaya Multimedia, 1997.

[39] Jay Armstrong, Greg Barnett, Stephanie Gowin, Tom Higgins, Marcelle Taylor, and Frank Welsch. *Macromedia Director 8.5 Shockwave Studio: What's New in Director Shockwave Studio*. Macromedia, Inc., March 2001.

[40] Jay Armstrong. *Macromedia Director 8.5 Shockwave Studio: Lingo Dictionary*. Macromedia Inc., February 2000.

[41] Jay Armstrong, Barbara Herbert, and Stephanie Gowin. *Macromedia Director 8.5 Shockwave Studio: Using Director Shockwave Studio*. Macromedia Inc., February 2000.

[42] Phil Gross and Mike Gross. *Macromedia Director 8.5 Shockwave Studio for 3D: Training from the source*. Macromedia Inc., 2002.

[43] Jaume Duran Castells, Lidia Sánchez Gomez. *Industrias de la comunicación audiovisual*. Edicions Universitat Barcelona, 2008.

[44] Adriana Gil Juárez, Tere Vida Mombiela. *Los videojuegos*. Editorial UOC, 2007.

[45] David H. Eberly, Ken Shoemake. *Game Physics*. Morgan Kaufmann, 2004.

[46] Kenneth C. Finney. *3D Game Programming All In One*. Cengage Learning, 2004.

[47] Eric Lengyel. *Mathematics for 3D Game Programming and Computer Graphics*. Cengage Learning, 2004.

[48] Colección completa de *Game Programming Gems*. Volumen 1 al 7. Charles River Media.

[49] Página web de Unity: http://unity3d.com/

[50] Página web de Adobe: http://www.adobe.com/

[51] Página web de Java: http://www.java.com/es/

[52] Página web oficial de foros de Director: http://www.directorforum.com/

[53] Página web de Adobe Director: http://www.adobedirectoronline.com/

[54] Página web de la Interactive Software Federation of Europe: http://www.isfe-eu.org/

[55] DigiWorld YearBook 2009 España. Los retos del mundo digital. Página web del Centro de IE Business School para el Análisis de la Sociedad de la Información y las Telecomunicaciones ENTER-IE: http://www.enter.ie.edu/