UNIVERSIDAD POLITÉCNICA DE VALENCIA

DEPARTAMENTO DE SISTEMAS INFORMÁTICOS Y COMPUTACIÓN

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

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

TESINA DE MÁSTER

Presentado por:
Marcos García Pascual

Dirigido por:
Dr. Ramón Pascual Mollá Vaya

Valencia, 2010

Índice General

Índice de tablas y figuras

Resumen

La industria del ocio interactivo digital ha experimentado, a pesar de su joven existencia, una evolución imparable que la ha llevado a situarse como la industria del entretenimiento más productiva, obteniendo una facturación anual superior a la de las industrias del cine y la música juntas. Esta evolución ha propiciado la aparición de un gran número de empresas dedicadas al desarrollo de videojuegos que pretenden año tras año ofrecer productos más atractivos e innovadores.

Uno de los factores que, probablemente, más ha ayudado a este crecimiento es el alto grado de penetración que poseen los ordenadores personales y el acceso a internet. Esta vía de comunicación facilita a las empresas desarrolladoras un nuevo modo de hacer llegar sus productos a los consumidores. Aparecen por tanto nuevos perfiles de jugadores de videojuegos que se alejan del estereotipo de chico joven con conocimientos tecnológicos y que van desde niños hasta mujeres madres de familia e incluso personas mayores. En todos ellos, las empresas desarrolladoras han conseguido ver un nuevo camino en el mundo del desarrollo de videojuegos: el desarrollo de videojuegos casuales.

A esto se une el carácter social que ha adquirido el acceso a Internet. Hoy en día, gran parte de los internautas dedican su tiempo de ocio a entablar nuevas amistades, conocer gente, hablar con amigos e incluso encontrar pareja por internet. La exitosa irrupción de las redes sociales en Internet plantea por tanto un nuevo modelo de negocio para las empresas desarrolladoras de videojuegos: los juegos casuales online multijugador. Estos juegos suelen alojarse en portales web que permiten al usuario jugar mientras conocen gente nueva y entablan nuevas relaciones sociales.

Es por todo esto por lo que cada vez un mayor número de empresas ven en este modelo de negocio su mayor fuente de ingresos. La competencia existente obliga a las desarrolladoras a buscar modelos de producción más eficientes que le permitan destacar sobre el resto de empresas y convertirse en modelos a seguir. Por ello, el objetivo principal de esta tesina es la creación de una herramienta que mejore la eficiencia en el desarrollo de juegos casuales online multijugador. Para ello, se ha realizado un

trabajo de colaboración con [1]. En primer lugar, en ambos trabajos, se ha realizado un estudio de las distintas herramientas existentes para el desarrollo de este tipo de juegos y un estudio crítico del modelo de producción actual. Tras observar ciertas carencias en este modelo de negocio, que no favorecen la eficiencia en la producción se ha propuesto el desarrollo de una herramienta cuya filosofía de diseño tiene como principal objetivo paliar dichas carencias. La filosofía de diseño se divide en: configuración externa, motor de inicialización y middleware. Esta tesina se centra en el aspecto middleware de la herramienta desarrollada y en ciertos aspectos de su carácter como motor de inicialización. Adicionalmente, esta tesina también trata el desarrollo de cierta funcionalidad adicional que dota a la herramienta de un mayor potencial.

Agradecimientos

Me gustaría expresar mi más sincero agradecimiento a:

Mis padres y mi hermana por apoyarme en todo momento y por ser tan especiales.

A mis compañeros de Exelweiss Ent. por su ayuda en la realización de este trabajo y por hacer tan ameno el día a día.

A Chelo por hacerme sentir tan a gusto a su lado.

A toda mi familia por hacer de cada reunión una fiesta.

A mi director Ramón Mollá por ayudarme a enfocar mis

objetivos. A mis amigos por demostrarme que son los mejores.

Y mi agradecimiento más especial a Clara por hacerme mejorar día a día y por su apoyo incondicional.

Capítulo 1

Introducción

1.1 Motivación

La evolución que ha experimentado la industria del videojuego en los últimos años la ha llevado a situarse como la industria de entretenimiento con mayor facturación, superando incluso a las industrias del cine y la música juntas. Este mercado emergente ha originado la aparición de un gran número de nuevas empresas y estudios de desarrollo dedicados al sector del ocio interactivo digital. El mercado de los videojuegos en España ha experimentado también este crecimiento, llegando a facturar en 2008 el 57% del total facturado por las empresas del ocio digital [2] y consiguiendo que las autoridades pertinentes le otorgaran el carácter de industria cultural [3].

Paralelo a este avance han surgido, en estos últimos años, nuevos perfiles de usuarios de videojuegos y nuevas formas de jugar. Hoy en día el consumidor de videojuegos no se encuentra en un rango de edades determinado o se ciñe solo a hombres o mujeres, sino que existe una gran variedad de videojuegos capaces de satisfacer las necesidades de jugadores bien diferenciados. Se puede decir por tanto que las desarrolladoras han sabido evolucionar y ampliar horizontes enfocando sus juegos a un nuevo público y difundiendo sus productos a través de medios con un mayor nivel de penetración que las consolas, como son el ordenador personal e Internet [4].

La aparición de este nuevo público ha llevado, tanto a las nuevas empresas como a los estudios de desarrollo consagrados, a desarrollar nuevos juegos orientados a este tipo de público; un público que demanda juegos atractivos e interesantes pero a la vez más sencillos y que requieran invertir un menor número de horas: los juegos casuales. Tal ha sido el grado de aceptación de este tipo de juegos por parte de los usuarios que el mercado de los juegos casuales genera anualmente 2,25 billones de dólares [4].

A la vez que la industria del videojuego ha ido evolucionando imparablemente desde su nacimiento, actualmente existe en Internet un fenómeno que mueve cada año cifras económicas muy grandes y del cual el 76% del total de los usuarios de Internet

forman parte [6]: las redes sociales. Hoy en día los internautas ven en la red un lugar nuevo donde desarrollar sus relaciones sociales. Las desarrolladoras de videojuegos no han pasado por alto este aspecto, viendo salida a un nuevo modelo de negocio que fusiona el mundo de las redes sociales y el de los juegos casuales online: portales sociales de juegos multijugador.

Estos portales poseen juegos cuyo principal atractivo es su carácter multijugador. De este modo, el portal se convierte en una red social en la que los usuarios además pueden jugar y competir entre ellos. La continua a parición de empresas que se dedican al desarrollo de este tipo de aplicaciones hace que exista una fuerte competencia contra la cual los estudios de desarrollo deben luchar. Una de las estrategias seguidas por estas empresas es la de desarrollar herramientas que mejoren la eficiencia en el desarrollo de sus juegos para posicionarse favorablemente respecto de su competencia. Desarrollar sus juegos de manera más eficiente permite a las desarrolladoras obtener un mayor beneficio por cada nuevo desarrollo, sacar sus productos al mercado antes que la competencia y poder aceptar un mayor número de ofertas de desarrollo.

El modelo de producción empleado por este tipo de empresas es el principal factor de la falta de eficiencia en el desarrollo de sus videojuegos. La metodología ad-hoc empleada hace que el coste, tanto temporal como económico, de cada nuevo desarrollo no se vea reducido, sino que se mantiene constante. Además, esto plantea la problemática de realizar de manera reiterativa ciertos trabajos que podrían ser desarrollados de forma genérica y reutilizados en desarrollos futuros. Es por esto por lo que, cada vez más, las empresas invierten su tiempo, esfuerzo y dinero en el desarrollo de herramientas que mejoren el proceso de producción de sus videojuegos, teniendo así una visión de futuro.

Dentro de estas herramientas se encuentran los que en el mundo de los videojuegos se conocen como motores. Un motor no es más que un software que implementa de manera genérica ciertas funcionalidades comunes existentes en los diferentes videojuegos. Este carácter genérico permite que posteriormente sus algoritmos puedan ser reutilizados por videojuegos de temáticas bien diferenciadas, evitando así la programación de un mismo código de manera reiterativa y propiciando el consecuente ahorro de costes de producción. Actualmente existe una amplia diversidad de motores que podrían ser clasificados en función de las distintas áreas de la programación de un videojuego (física, red, sonido, render, gráficos, etc.). Además de esto, dos características fundamentales que definen a un motor son su robustez y fiabilidad, es decir, un motor es un software ya testeado capaz de facilitar cierta funcionalidad sin errores y de manera eficiente.

En la empresa Exelweiss Entertainment, desarrolladora del portal de juegos

online multijugador *www.mundijuegos.com*, se lleva trabajando desde hace más de siete años con la metodología de producción ad-hoc expuesta. La problemática existente debida a la falta de eficiencia originada por este modelo de producción ha propiciado la necesidad de estudiar la manera de abordarla, dando como resultado las aportaciones que se demuestran en esta tesina.

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. Realizar un estudio del proceso de producción de un videojuego casual online multijugador y las herramientas utilizadas durante este proceso.

2. En base a dicho estudio y con el objetivo de mejorar la eficiencia de este proceso, identificar qué aspectos son mejorables y qué soluciones se pueden aportar.

3. Desarrollar, en colaboración con el trabajo realizado en [1], una herramienta capaz de solventar las carencias observadas en el estudio realizado.

4. Haciendo uso de dicha herramienta, desarrollar un nuevo juego que permita valorar la efectividad de la misma.

5. Cuantificar la aportación de la herramienta desarrollada mediante un análisis de los costes de producción de tres videojuegos de características similares.

1.3 Contenido

La estructura de esta tesina se resume de la siguiente manera:

El capítulo 2 sirve como introducción al lector al mundo de los videojuegos. En primer lugar se narra una breve historia de los videojuegos y seguidamente se analizan el estado actual y la evolución sufrida tanto por la industria como por los usuarios. A continuación se detallan las distintas herramientas más populares que son empleadas por las empresas desarrolladoras en la producción de juegos casuales online. Por último, el capítulo expone las fases de desarrollo por las que pasa un juego desde su concepción hasta su puesta a la venta y la estructura interna típica que siguen los eventos ejecutados en un videojuego.

El capítulo 3 pretende ser una introducción al concepto de motor en el ámbito de los videojuegos y su uso dentro del proceso de producción de los mismos. Tras tratar el concepto de motor dentro de este ámbito se realiza una breve clasificación de los motores existentes.

El capítulo 4 trata de manera detallada el modelo de producción seguido por las empresas en el desarrollo de juegos online multijugador. Tras exponer la motivación por parte de las desarrolladoras hacia estos juegos como su modelo de negocio, se trata la problemática existente en el desarrollo de este tipo de juegos. Ello plantea la necesidad de elaborar un motor de producción capaz de paliar dichas deficiencias.

El capítulo 5 muestra la filosofía de diseño del motor de producción y detalla sus características y funcionamiento atendiendo a las funcionalidades middleware y de motor de producción que éste posee.

El capítulo 6 describe cómo, con la ayuda de dicha herramienta, se ha creado un nuevo videojuego casual online multijugador y realiza una comparación de los costes de desarrollo entre este juego y otros dos juegos de características muy similares. En uno de estos juegos también se ha hecho uso del motor de producción planteado, mientras que el tercero ha sido desarrollado desde cero.

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

Capítulo 2

Introducción a los videojuegos

2.1 Introducción

Este capítulo pretende servir como introducción al lector al mundo de los videojuegos. En primer lugar, en la sección 2.2 se narra de manera breve la historia de los videojuegos. En la sección 2.3 se realiza una revisión del estado de la industria y de los usuarios de videojuegos. Seguidamente, en la sección 2.4, se describen las distintas plataformas y herramientas empleadas en el desarrollo de videojuegos. Posteriormente, en la sección 2.5 se muestran las fases de desarrollo por la que pasa un videojuego y en la sección 2.6 se detalla la estructura interna típica de un videojuego. Finalmente, en la sección 2.7 se realizan las conclusiones de este capítulo.

2.2 Historia

Fue en 1958 cuando apareció el primer videojuego (que sería comercializado más tarde) de la historia creado por William Higinbotham, un físico estadounidense que diseñó un sistema de juego en el que se simulaba un partido de tenis. En un inicio, este invento no fue creado con ningún interés comercial y Higinbotham nunca patentó su invención. No obstante, poco más de una década más tarde esta idea se convirtió en el juego más relevante de la historia: “PONG”. Atari fue la encargada de su desarrollarlo y comercialización.

Figura 1. Videojuego Pong

La primera consola de la historia fue la "Magnavox Odyssey", que salió al mercado en el año 1972 y que carecía de memoria alguna. En 1975 Atari creó su primera consola doméstica, la "Atari Pong". Esta consola tan sólo permitía jugar al famoso videojuego de la compañía sin ninguna otra posibilidad, lo que ocasionó que no fuera muy bien acogida por el público.

Dos años más tarde Atari consiguió un gran éxito al lanzar al mercado la "Atari2600". Esta consola causó toda una revolución en el mundo de los videojuegos; contaba con la innovación de poder cambiar de juegos mediante el nuevo sistema de cartuchos. Dicha consola fue la primera de 8 bits de la historia y fue la más popular durante más de una década gracias a su amplio catálogo de juegos.

En 1985 apareció la "NES" (Nintendo Entertainment System). Llegó en un momento de crisis dónde parecía que el mundo de los videojuegos empezaba a tocar techo, sin embargo, Nintendo dio un giro de tuerca y se lanzó al mercado consiguiendo pronto ser la videoconsola más vendida del mercado. Por primera vez se introdujo el pad o mando tal y como se conoce hoy en día.

Figura 2. NES

En 1986, con el objetivo de acabar con la exclusividad y el monopolio de Nintendo, Sega lanzó su nueva videoconsola doméstica (“Master System”) a territorio americano y europeo dónde consiguió un notorio éxito llegando incluso a superar las ventas de la NES.

Figura 3. Sega Master System

Tan solo dos años después, Sega apostó por una nueva consola de 16 bits asegurando una mejora en la calidad técnica. La “Sega Mega Drive” consiguió estar a la altura y competir con la “SuperNintendo”; la nueva consola de la empresa japonesa que llegó en 1992 y que también contaba con una potencia de 16 bits.

1Figura 4. Sega Mega Drive

En 1992 Sega lanzó su nueva consola de 32 bits, la “Sega Saturn”, que consiguió acabar tajantemente con las consolas de 8 y 16 bits. La política seguida por la compañía no gustó a los usuarios lo que, unido al estupendo marketing de Sony al lanzar su consola, hizo que la Sega Saturn no proporcionara los beneficios esperados. La “PSX”, la nueva consola de Sony, se convirtió al poco tiempo en la consola número uno del mercado, conviviendo desde 1996 con la “Nintendo 64”. Mientras que la “PSX” había conseguido la atención de jóvenes más mayores ya casi desinteresados por los

videojuegos ofreciendo un tipo de juego más adulto y serio, Nintendo mantenía su política de destinar sus juegos a un público muchísimo más joven e informal.

Figura 5. Sony PlayStation

Durante el reinado indiscutible de Sony, Sega preparó minuciosamente "Dreamcast". De nuevo pionera en la nueva generación de consolas, Sega volvía a arriesgarse una vez más en su intento de convertir a su consola en la número uno. La "Dreamcast" era una consola potente de 128 bits, capaz de trabajar con unos gráficos nunca vistos hasta entonces. Además fue la primera consola en incorporar la posibilidad de jugar online con un módem ya integrado. No obstante, la mala política de la empresa y la inminente llegada de "PSX2" hizo que Dreamcast (la última consola de Sega) tampoco gozara de ningún éxito.

Figura 6. Sega Dreamcast

En el año 2000 Sony lanzó "PSX2", una consola con formato de almacenamiento de datos DVD, capaz de reproducir los mismos, 128 bits de potencia y unas características técnicas debidamente bien aprovechadas. Una gran cantidad de compañías se volcaron y respaldaron a Sony con lo que esta se aseguró de nuevo el liderazgo en esta nueva generación. Las llegadas de "Gamecube" (Nintendo) y de "Xbox" (Microsoft) no fueron suficiente para que la consola de Sony dejara de ser la número uno del mercado de los videojuegos de manera indiscutible.

Figura 7. Xbox de Microsoft

En la actualidad Microsoft ha sido pionera en lanzar su sistema de entretenimiento "XBOX360" de nueva generación. El 02 de diciembre del 2005 llegó a España, obteniendo una aceptación estupenda del público. Esta consola junto con "PlayStation 3" conforman hoy en día una lucha por defender cada una su formato y su mercado.

Por otro lado, la actualidad de Nintendo pasa por su innovadora consola "Wii" y su propuesta hacia los juegos sociales. Nintendo ha logrado con su revolucionario producto despertar el interés por los videojuegos en otro tipo de jugadores de carácter más "casual".

En cuanto al futuro de las consolas nada es seguro; mientras que por un lado, muchos expertos auguran una nueva generación de consolas con nuevos sistemas de control basados en la captura de movimientos del jugador ("Project Natal" de Microsoft o "Move" de Sony), otros aseguran el fin de las consolas tal y como se conocen hoy en día, dando paso a sistemas de juego a través de streaming de video ("OnLive").

Independientemente del futuro de los videojuegos, a lo largo de su corta historia ha sido posible observar una rápida evolución y una gran capacidad de innovación.

En [7, 8] se realiza una revisión en profundidad de la historia de los videojuegos.

2.3 Industria y usuarios

A pesar de la juventud de la industria del videojuego, con poco más de 50 años, esta ha experimentado una imparable evolución en cuanto a tecnología y capacidad de innovación. El salto cualitativo observado tanto en la tecnología de las consolas, de los ordenadores y de sus periféricos, así como en la variedad y calidad de los juegos es más que evidente y hoy en día todavía continúa en aumento de manera imparable.

Además, el hecho de que la industria del videojuego sea considerada, a efectos políticos, como un bien cultural [3] ha propiciado un mayor impulso para la misma, permitiendo a los desarrolladores disfrutar de los beneficios que esta decisión conlleva.

Balance económico 2009

En el año 2009, la industria del videojuego en España alcanzó el 53% del mercado de entretenimiento audiovisual e interactivo (cine-DVD-música-videojuegos), consiguiendo que sus ventas superasen la suma de lo registrado por el cine, el DVD y la música juntos.

En términos absolutos, de la cifra total de ventas de la industria (1.200m€), el software registró el 53%, mientras que el hardware alcanzó el 47%.

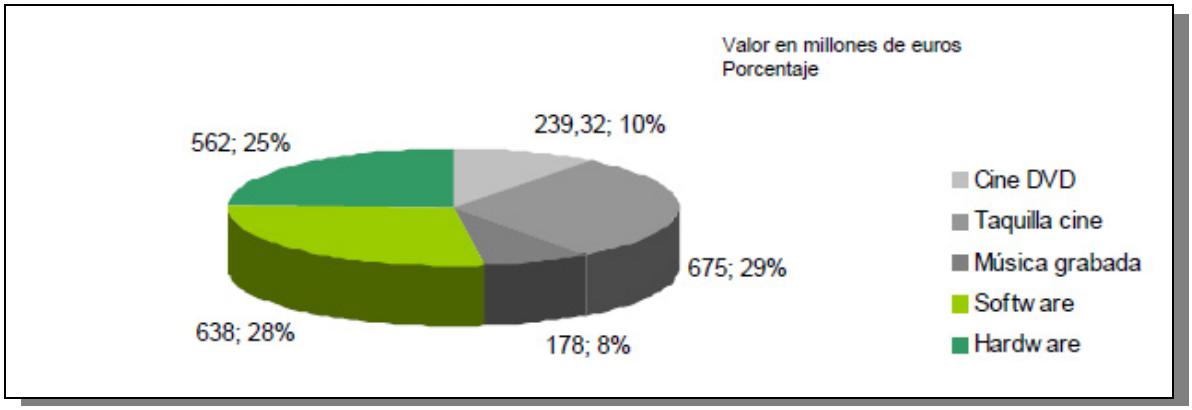

Figura 8. Balance económico 2009

Mercado europeo

El mercado europeo del videojuego se ha visto arrastrado al proceso de desaceleración vivido en todos los sectores económicos. España, a pesar del descenso del 16%, sigue manteniendo el cuarto lugar en el ranking europeo de consumo de videojuegos, sólo por detrás de Reino Unido, Francia y Alemania y continúa con unas cifras de ventas muy buenas.

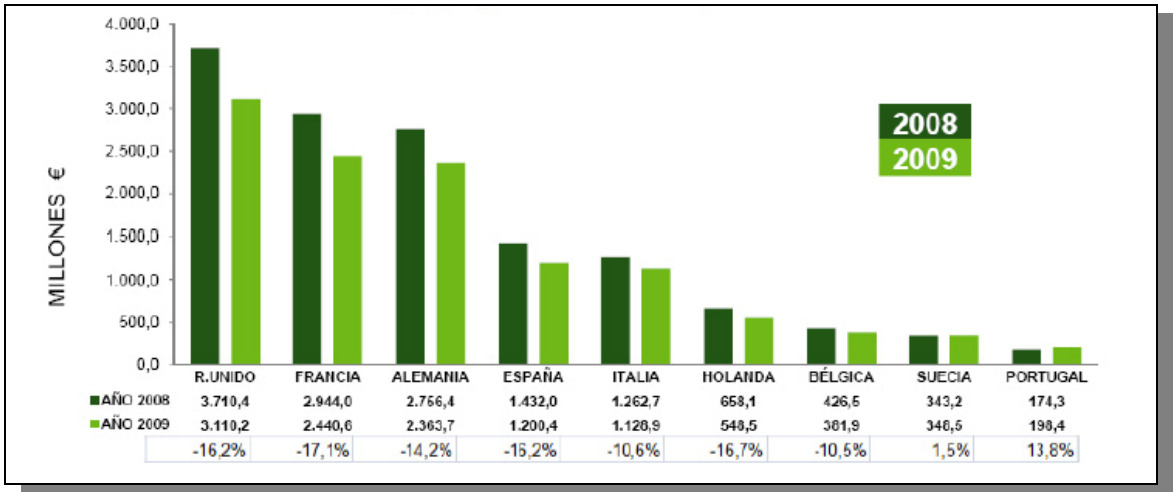

Figura 9. Mercado europeo

Una revisión más detallada de estas estadísticas y gráficas se hace en [2], [9] y [10].

Evolución del número de jugadores de videojuegos en España, 2006-2009:

En España, el 22,5% de la población se declara jugadora de videojuegos; en total, cerca de 10,4 millones de personas; lo que supone cerca de 1.530.000 jugadores más que en 2006.

Jugadores casuales

Hoy en día, el perfil del jugador de videojuegos no solo abarca al chico joven aficionado a los ordenadores y a la tecnología. Mujeres, personas mayores y niños se han convertido en consumidores habituales del software de entretenimiento gracias a que las empresas desarrolladoras han sabido ofrecerles un nuevo catálogo de juegos más adecuado a sus gustos y necesidades: los juegos "casuales".

Los juegos "casuales" a diferencia de los juegos "hardcore" (orientados a jugadores de videojuegos expertos) poseen una curva de aprendizaje muy rápida, lo que permite al jugador inexperto dominarlo en cuestión de muy poco tiempo. El hecho de que este tipo de juegos abarque un abanico tan amplio de potenciales consumidores ha originado en una facturación anual realmente llamativa para las empresas desarrolladoras: según la International Game Developers Association, IGDA, en 2008 se llegó a facturar un total de 2,25 billones de dólares (un 20% más que en 2007) [4].

Penetración de PC y consolas

En 2009, la proporción de hogares españoles con ordenador era del 58,2%, lo que supone un 11% más que en 2006. En total, 9.687.500 hogares tienen PC y en relación al perfil sociodemográfico, destaca que el tamaño del hogar, la presencia de niños y la clase social, son los factores de influencia relacionados con la presencia de ordenador.

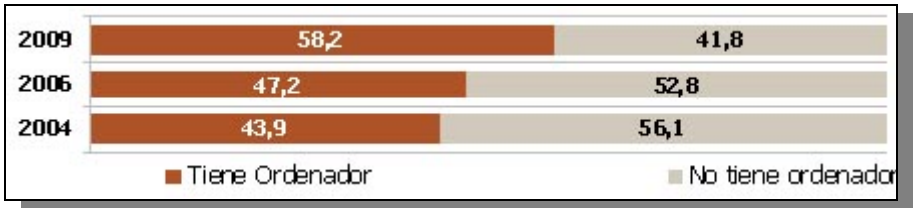

Figura 10. Penetración PC

En relación a las consolas, uno de cada tres hogares españoles cuenta con al menos una. En concreto, en 2009 el 34,7% de los hogares españoles poseía una consola; un 0,7% más que en 2006.

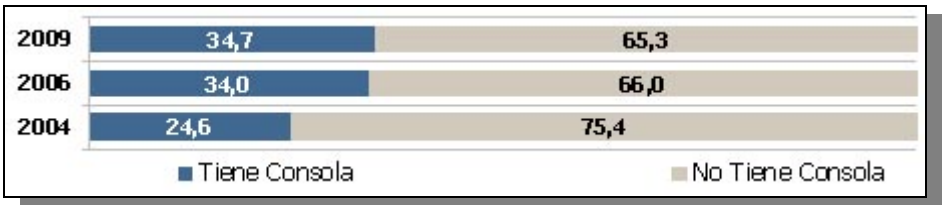

Figura 11. Penetración consolas

En valores absolutos, 5.775.900 hogares tienen videoconsola en España, y al igual que ocurre con los ordenadores, el mayor tamaño del hogar y la presencia de niños son factores que determinan una mayor penetración.

Internet, el juego online y las redes sociales

Hoy en día los usuarios de videojuegos demandan la posibilidad de jugar online como un requisito casi imprescindible. Actualmente un 44,6% de los hogares españoles tiene acceso a Internet, lo que ofrece a los usuarios la posibilidad de jugar de modo cooperativo con otro usuario, jugar de forma masiva formando equipos o jugar uno contra otro a través de la red. Esta práctica ha despertado en los jugadores un gran interés y ha tenido una gran aceptación.

El modo de juego online está directamente relacionado con la reciente aparición y éxito de las redes sociales (según Ocio Network [6] el 76% de los internautas utiliza al menos una red social). Y es en este punto dónde los juegos casuales han tomado una mayor relevancia. Hoy en día un gran número de jugadores disfrutan de este tipo de juegos a través de las redes sociales o los portales de juegos sociales.

2.4 Plataformas y herramientas

Atendiendo a las plataformas sobre las que un juego puede ser ejecutado, se pueden encontrar las siguientes: máquina recreativa o arcade, consola de sobremesa, ordenador, plataforma online y consola portátil. De todas estas plataformas, la plataforma online ejecutada sobre un ordenador es la que permite hacer llegar el juego a un mayor número de usuarios, debido a, como se ha visto en el punto anterior, el alto grado de penetración de los ordenadores y el alto porcentaje de usuarios con conexión a internet.

A la hora de desarrollar videojuegos online se pueden encontrar diversas tecnologías, entre las que se destacan las siguientes:

Tecnología	Propietario	Lenguaje	Penetración	Plugin
Java	Sun MicroSystems	Java	56%	Sí
Flash	Adobe	ActionScript	99%	Sí Flash Player
Shockwave	Adobe	Lingo, JavaScript	50%	Sí Shockwave Player
Unity	Unity Technologies	C#, Ajax, JavaScript	1%	Sí

Tabla 1. Tecnologías de desarrollo

Además de los datos vistos en la tabla cabe señalar que por un lado, tanto Unity como Shockwave permiten gráficos 3D, Java, por otro lado, requiere la instalación de la aplicación en la máquina del usuario para poder ejecutarse y por último, decir que Unity tan solo puede ser desarrollado en MAC.

Una información más detallada acerca del funcionamiento de cada tecnología puede verse en [11, 12, 13 y 14].

2.5 Fases de desarrollo

Durante el desarrollo de un videojuego es necesario mantener un orden cronológico de las

distintas labores a realizar con el fin de mejorar y optimizar lo máximo posible el proceso de realización del mismo. Es por ello por lo que todo desarrollo pasa por una serie de fases, cada una de las cuales se encarga de una tarea concreta. Las distintas fases existentes en el proceso de desarrollo de un videojuego son [8]: concepto, pre-producción, prototipo, producción, alfa, beta, máster y post-producción.

CONCEPTO

Objetivo: Concebir una idea que pueda ser transformada en un videojuego y definir el documento de concepto, en el que se transmite dicha idea y se define el usuario objetivo del juego.

PRE-PRODUCCIÓN

Objetivo: Desarrollar el documento de diseño, especificando la guía de estilo artístico y proceso de producción.

PROTOTIPO

Objetivo: Desarrollar un producto software capaz de representar, de modo general, la idea del juego. El prototipo debe mostrar aquello que hace al juego atractivo así como sus características principales.

PRODUCCIÓN

Objetivo: Desarrollar el juego completamente. Esta fase incluye tareas como la codificación del programa, la creación de gráficos, la grabación de sonidos, etc.

Un estudio exhaustivo de la fase de producción, así como de los diferentes perfiles de programadores involucrados en ella se realiza en [15].

ALFA

Objetivo: Realizar las pruebas necesarias para comprobar que el juego funciona de manera correcta. Es necesario que el juego sea jugable de principio a fin y que cada módulo del mismo funciona correctamente.

BETA

Objetivo: Eliminar todos los errores menores y pulir el producto para dejarlo listo de cara a ser lanzado al mercado.

MASTER

Objetivo: Copiar el juego y empaquetarlo para distribuirlo y proceder a su venta al público.

POST-PRODUCCIÓN

Objetivo: Desarrollar extensiones del juego que permitan alargar su ciclo de vida y desarrollar *patchs* que permitan subsanar posibles errores que fueron pasados por alto en las fases de testeo.

Para un estudio exhaustivo de estas fases, así como del proceso de desarrollo de un videojuego en general pueden consultarse [7, 8, 15].

2.6 Estructura interna

A pesar de que cada juego puede presentar unas cualidades bien diferenciadas del resto, todos poseen una estructura interna típica que resume el orden de ejecución de los diferentes eventos que se llevan a cabo durante la ejecución de los mismos.

El núcleo de esta estructura es el bucle principal, que se ejecuta una vez por cada imagen dibujada sobre la pantalla del dispositivo sobre el que se ejecuta la aplicación. Este bucle alberga todas aquellas funciones que deben realizarse con este periodo y en un orden específico.

La estructura interna y su orden de ejecución puede representarse mediante el siguiente esquema [7]:

- Inicialización del programa principal.
- Bucle principal del juego.
    - Inicialización del interfaz.
    - Bucle de interfaz.
        - Recibir inputs.
        - Renderizar la pantalla.
        - Actualizar el estado del interfaz.
        - Gestionar los cambios de estado del interfaz.

    - Finalización del interfaz.
    - Inicialización del nivel de juego.
    - Bucle de nivel.
        - Recibir inputs.
        - Ejecutar IA
        - Realizar un paso en la simulación física.
        - Actualizar las entidades de juego.
        - Recibir/Mandar mensajes.
        - Actualizar tiempo.
        - Actualizar el estado del juego.
        - Otras acciones posibles.
    - Finalización del nivel actual
- Finalización del juego.

Al observar este esquema puede apreciarse la aparición de eventos de inicialización y finalización de manera emparejada. Es imprescindible para el correcto funcionamiento del juego que todo aquello que se cree en la fase de inicialización sea destruido en orden inverso en la fase de finalización. Este proceso inicialización/finalización evitará posibles errores futuros debidos a pérdidas de memoria u otras causas.

2.7 Conclusiones

A lo largo de este capítulo se han tratado diversos aspectos relacionados con los videojuegos con el objetivo de introducir al lector en el mundo del desarrollo de mismos.

En primer lugar se ha narrado brevemente la historia de los videojuegos desde sus inicios en los años 50 hasta los videojuegos y consolas más actuales de última generación.

Seguidamente se ha realizado un pequeño estudio acerca del estado actual de la industria del videojuego y de su evolución en los últimos años, así como del consumidor de videojuegos y las nuevas tendencias causadas por la aparición de un nuevo tipo de jugador: el jugador casual.

En el punto siguiente se han mostrado las plataformas sobre las que se puede ejecutar un juego y las distintas herramientas que permiten a los desarrolladores crear dichas aplicaciones.

Por último, los dos capítulos finales han servido como introducción a las fases de desarrollo de un videojuego y su estructura interna, ambas presentes en todos los videojuegos independientemente de sus características y especificaciones.

Seguidamente, en el capítulo 3 se realiza una introducción al uso de motores en el desarrollo de videojuegos y las ventajas que esto plantea. Además, se detallarán aquellos motores que los desarrolladores suelen emplear con mayor frecuencia realizando una clasificación de los mismos.

Capítulo 3

Introducción a los motores

3.1 Introducción

A lo largo de este capítulo se pretende realizar una introducción al uso de motores durante el proceso de desarrollo de un videojuego así como realizar una clasificación de los distintos motores existentes y su posibilidades de uso. Concretamente, en el punto 3.2 se realiza una introducción al concepto de motor en el mundo de los videojuegos, para posteriormente, en el punto 3.3, realizar la clasificación de los mismos. Por último, en el punto 3.4 se realizan las conclusiones pertinentes a este capítulo.

3.2 El concepto "motor" en el desarrollo de videojuegos

En sus orígenes, el desarrollo de un videojuego era una tarea realizada en muchas ocasiones por una o dos personas y su simplicidad permitía que tan sólo fuesen implementadas aquellas funcionalidades imprescindibles para cada juego en concreto. Esta estrategia podía ser útil en estos casos en los que el código implementado para cada juego no era muy extenso, no obstante, a medida que los videojuegos se han ido convirtiendo en aplicaciones más realistas y espectaculares también ha aumentado su complejidad en el desarrollo. Es por ello por lo que surge la idea de desarrollar aquellas funcionalidades comunes en los distintos juegos de manera única, con el fin de reutilizar este desarrollo en futuras aplicaciones. El producto obtenido de este desarrollo selectivo de distintas funcionalidades comunes es lo que se conoce como motor o *engine* de desarrollo de videojuegos.

El concepto de motor de videojuegos conlleva dos características imprescindibles: carácter genérico y fiabilidad. En primer lugar, debido a que un motor debe implementar partes genéricas y comunes en los distintos videojuegos, este debe ser concebido como un software capaz de implementar ciertas funcionalidades de forma genérica pero jamás contener ningún tipo de programación específica que permita llevar

a cabo tareas concretas. En segundo lugar, un motor se caracteriza por ser un software fiable y testeado, capaz de llevar a cabo las funcionalidades para las que ha sido creado sin presentar errores o inconsistencias.

3.3. Clasificación y uso

Atendiendo a la procedencia de un motor se puede realizar una primera clasificación inicial con la que se dividirían los motores en dos tipos: motores propios o motores licenciados.

Mientras que los primeros precisan de un desarrollo propio, previo a poder ser utilizado, los segundos plantean la ventaja de componer un software ya fiable y con la posibilidad de poder utilizar toda la funcionalidad que incorpora desde el momento de su adquisición. Por el contrario, los motores propios poseen la ventaja del ahorro económico que supone la compra de un motor licenciado. Además, un motor propio siempre puede ajustarse en mayor medida a aquellas funcionalidades que posteriormente serán utilizadas, creando así un motor más "personalizado".

Por otro lado, es posible realizar una clasificación de los motores atendiendo al área de la programación para la cual han sido pensados y desarrollados. Debido a la alta complejidad de los juegos actuales han surgido diferentes perfiles de programadores que se especializan en el desarrollo de determinadas funcionalidades concretas (programador de física, programador de red, programador gráfico, etc.). La clasificación siguiente está, por lo tanto, directamente relacionada con estas áreas de la programación de videojuegos.

En primer lugar, el motor de física [16, 17] es el encargado de implementar toda aquella funcionalidad que permita realizar una simulación física realista dentro del videojuego. El motor de físicas debe permitir que la simulación de las propiedades dinámicas de los objetos presentes en la escena sea fiel a como lo sería en la vida real. De este modo el juego adquiere una mayor verosimilitud. No obstante, en los motores de física enfocados al desarrollo de videojuegos es posible encontrar algunos que, a costa de perder un poco de precisión en la simulación, permiten realizar los cálculos con un coste computacional notablemente menor. Esta característica cobra una vital importancia debido al carácter de "tiempo real" presente en los videojuegos.

Del mismo modo que el motor de física trata de simular los comportamientos dinámicos de los objetos de forma realista, el motor de colisiones pretende lo mismo atendiendo a la detección y resolución de las colisiones producidas entre los cuerpos físicos existentes durante la simulación.

Toda simulación física en un videojuego tiene como base los siguientes principios:

1. Los objetos que intervienen en la simulación física están compuestos por el *modelo de colisiones* que representa su superficie geométrica y el *modelo de cuerpo rígido* el cual define sus propiedades dinámicas.

2. Los objetos pueden interactúan entre sí mediante la aplicación de fuerzas.

3. Toda simulación física se lleva a cabo mediante un procedimiento cíclico que calcula el estado de los cuerpos durante intervalos discretos de tiempo. Para cada intervalo se realizan los siguientes pasos:

    a. Cálculo de la fuerza resultante del sumatorio de todas las fuerzas aplicadas a cada objeto.

    b. Estimación, en base a las fuerzas calculadas, de las trayectorias que estos cuerpos realizarán en el siguiente intervalo de tiempo.

    c. Cálculo de los puntos de contacto entre las superficies de los modelos de colisión mediante intersecciones geométricas.

    d. Cálculo de la distancia y eje de separación entre cuerpos, permitiendo así estimar el tiempo de colisión dadas las trayectorias estimadas en el paso 2. Con el dato de tiempo estimado de colisión cada cuerpo reajusta su posición evitando atravesar otros cuerpos rígidos.

    e. Con estos puntos de contacto se definen las restricciones de movimiento que tendrán los cuerpos al colisionar con otros. Se plantea un sistema de ecuaciones diferenciales que pretende satisfacer el estado de los cuerpos cuando se aplican fuerzas de otros simultáneamente (colisiones y fricción), y este sistema se resuelve iterativamente con métodos de penalización cuando los cuerpos no satisfacen las restricciones aplicando fuerzas de reacción durante microintervalos de tiempo. Se usan métodos numéricos como resolución matricial con multiplicadores de Lagrange, o aplicación de impulsos basados en proyecciones Gauss-Seiddel. Este último ofrece soluciones aproximadas pero requieren menos tiempo de cómputo lo que lo hace atractivo para simulación en tiempo real.

4. Adicionalmente, se usa el mismo sistema de penalización para resolver las restricciones de movimiento causadas por las articulaciones mecánicas.

5. Finalmente, con los impulsos finales calculados durante la resolución de restricciones se ajustan los parámetros en las ecuaciones de movimiento y se entregan las posiciones y orientaciones de los cuerpos rígidos al subsistema gráfico.

Es fácil apreciar que el proceso cíclico visto implica un número elevado de cálculos complejos por cada paso realizado en la simulación; consecuencia de que la simulación física cause un alto consumo de recursos y tiempo de CPU. Por ello, se ha de prestar especial atención a la optimización de estos cálculos.

Por su lado, el motor de audio [19] es el que implementa todas aquellas funcionalidades que tienen que ver con el tratamiento y la manipulación de los objetos sonoros de un videojuego incluyendo la carga de los ficheros de audio necesarios.

En cuanto al motor de render [20, 21] se puede decir que es el encargado de implementar todos aquellos algoritmos que permiten proyectar, de manera eficiente, la imagen renderizada del juego sobre la pantalla.

Por otro lado, el motor de inteligencia artificial [18] implementa todos aquellos algoritmos que dotan al juego del poder de tomar decisiones durante su ejecución. Estos algoritmos dotan al juego de cierta “inteligencia”, permitiendo así que las distintas entidades tengan unos comportamientos determinados.

Del mismo modo, existen otros motores como el motor de red o el motor de animación, que al igual que los motores vistos pretenden extraer toda la funcionalidad común presente en el área específica para la cual han sido desarrollados y facilitar al programador un software genérico que las lleve a cabo.

3.4 Conclusiones

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

En primer lugar, se enmarcado el concepto de motor dentro del ámbito del desarrollo de videojuegos, permitiendo de este modo conocer sus características generales y propósitos.

Seguidamente, se ha realizado una diferenciación entre motores propios y motores licenciados mostrando las ventajas e inconvenientes que presentan cada uno de estos tipos.

Por último se han mostrado aquellos motores más importantes que son usados habitualmente en el proceso de producción de un videojuego.

En líneas generales, se puede concluir por tanto que un motor consiste en una pieza de software que implementa parte de la funcionalidad común que se observa en los distintos videojuegos y que éste ha de caracterizarse por su robustez, calidad y carácter genérico.

En el siguiente capítulo se va a mostrar el modelo actual de negocio seguido por muchas empresas desarrolladoras de videojuegos: los videojuegos casuales online multijugador. Por otro se estudiarán los distintos problemas que el desarrollo de este tipo de juegos plantean a las empresas desarrolladoras y se planteará el desarrollo de una herramienta, cuyos principios coinciden con los de un motor, para intentar solventarlos. El desarrollo de la herramienta planteada será analizado en el capítulo 4.

Capítulo 4

Los juegos casuales online multijugador como modelo de negocio y la problemática existente en su desarrollo

4.1 Introducción

En este capítulo se va a exponer la actual tendencia de los pequeños estudios de desarrollo hacia el desarrollo de videojuegos casuales, concretamente, hacia el desarrollo de videojuegos casuales online multijugador. Seguidamente, en la sección 3.3 se va a tratar la problemática existente en el desarrollo de este tipo de juegos debido al modelo de producción ad-hoc empleado en cada nuevo desarrollo. Por último, en la sección 3.4 se realizan las conclusiones de este capítulo y se plantea el desarrollo de un motor de producción como posible solución a la problemática descrita.

4.2 Los juegos online multijugador como modelo de negocio

La actual expansión de la industria del videojuego ha propiciado el nacimiento de un gran número de empresas dedicadas al desarrollo de software de entretenimiento. Un alto porcentaje de estos estudios de desarrollo están formados por un número reducido de trabajadores y su principal fuente de ingresos proviene del desarrollo de juegos casuales. Los datos vistos en la sección 2.2, en el que se concluye que el mercado de los juegos casuales genera 2,25 billones de dólares anuales, justifican el hecho de que gran parte de estas empresas nacientes vean en estos juegos su modelo de negocio.

En particular, existen empresas que, dedicándose también al desarrollo de juegos casuales, han decido combinar esta labor con la de la creación de portales sociales sobre los que los usuarios puedan jugar a sus juegos. Estos juegos tienen en su carácter multijugador su valor más preciado. Algunos ejemplos de este tipo de empresas son: *games.yahoo.com,www.miniclip.com ,www.playray.es, www.gamedesire.com,* **Error!**

Hyperlink reference not valid. España), *www.mundijuegos.com* (España), *www.opqa.com* (España). La idea de este tipo portales surge tras la imparable expansión que en los últimos años están experimentando las redes sociales. Portales como *Facebook* o *Tuenti* junto con otras redes sociales existentes son utilizados por un 76% de los internautas [6].

4.3 La problemática en el desarrollo de juegos casuales online multijugador

Este tipo de empresas suelen emplear un modelo de producción ad-hoc, lo que implica desarrollar de manera específica para cada juego nuevo la mayor parte del código. Esto supone tener que crear un nuevo proyecto, desde cero, cada vez que se desea desarrollar un nuevo juego. Uno de los principales objetivos perseguidos por estas empresas es el de abaratar los costes de producción tanto en recursos como en tiempo para obtener un mayor margen de beneficios, manteniendo en todo momento la calidad del producto, y 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 generar nuevos videojuegos de manera más eficiente evitando la generación de código ad-hoc.

Esta afirmación se basa en la propia experiencia como trabajador de *Exelweiss Ent.*, empresa desarrolladora del portal *www.mundijuegos.com* (portal de juegos online multijugador español líder en España), la de los propios compañeros (profesionales del sector que llevan más de siete años realizando este tipo de desarrollos) y la de otras empresa españolas cuya principal fuente de ingresos proviene del desarrollo de este tipo de juegos y con las que se ha podido tratar el tema en distintas reuniones y congresos, como el CDV, o a través de foros especializados, como *www.stratos-ad.com*.

Gracias al uso de estas herramientas por parte de los estudios de desarrollo, se consiguen los siguientes beneficios:

En primer lugar, debido a que los márgenes de tiempo establecidos en el desarrollo de un videojuego son tan ajustados, reducir los costes de producción permite a una empresa aceptar ciertos desarrollos que en otros casos no podría haber aceptado debido a no poder respetar los plazos solicitados por el cliente.

Por otro lado, aparece el término “oportunidad de negocio”. En determinadas ocasiones puede que un juego necesite ser desarrollado en un momento determinado (aparece una noticia de gran impacto social, se celebra un evento de gran magnitud, etc.). Poder desarrollar un videojuego de unas características determinadas en un espacio de tiempo reducido, permite a un estudio de desarrollo publicar su producto

antes que la competencia. En la industria del videojuego, al igual que en el resto de negocios, ser el primero implica obtener una clara ventaja respecto del resto.

Por último, la reducción en los costes de producción permite trabajar con un equipo de desarrollo más reducido, lo que supone un ahorro económico para la empresa que, viendo así aumentado su margen de beneficios.

Ejemplos claros de este tipo de herramientas son los motores y librerías, vistos en la sección 2.4, que los desarrolladores emplean para la creación de sus videojuegos; beneficiándose así de las ventajas que estas ofrecen.

Por un lado, estas herramientas abstraen las funcionalidades comunes existentes en el desarrollo de videojuegos y las desarrollan de forma genérica. Estas funcionalidades serán potencialmente utilizadas en un gran número de desarrollos futuros. Por ello, todo el esfuerzo empleado en el desarrollo del motor obtiene su correspondiente recompensa cada vez que se desee desarrollar un juego nuevo, ya que para cada juego nuevo en desarrollo, basta con importar el motor pertinente para obtener dicha funcionalidad.

Por otro lado, este tipo de herramientas garantizan calidad y robustez, ya que son programas testeados y cuyas funcionalidades están perfectamente integradas unas con otras.

Con todo esto, se concluye que el uso de motores y librerías en los videojuegos permite reducir los costes de producción de los mismos gracias a que ya poseen implementadas las funcionalidades comunes de una manera robusta y de calidad.

Las herramientas de las que se dispone varían en función de la tecnología empleada para el desarrollo de los videojuegos. Tras la revisión de las tecnologías empleadas para la realización de este tipo de videojuegos, vista en la sección 2.2, se puede realizar un análisis crítico para decantarse por una u otra. En primer lugar, si se pretende emplear una tecnología que permita el mayor abanico posible de tipos de juegos, se pueden descartar aquellas tecnologías que no permitan el desarrollo de juegos 3D. Por otro lado, analizando el público objetivo de los juegos casuales, es conveniente emplear una tecnología que requiera los mínimos conocimientos informáticos posibles. Es por esto por lo que una tecnología que permita el desarrollo de juegos ejecutados directamente sobre el navegador web, sin la necesidad de instalar la aplicación en la propia máquina del usuario, es la más adecuada. Una vez realizada esta criba inicial, Shockwave y Unity parecen ser las candidatas idóneas. Por último, con el objetivo de elegir una de estas dos opciones, se puede prestar atención al nivel de penetración de cada una. Mientras que Shockwave está instalado en un 50% de los navegadores, Unity tan solo lo está en el 1%. Por todo esto, es por lo que Shockwave se presenta como una

tecnología idónea para la creación de juegos casuales online multijugador. El desarrollo de videojuegos que puedan hacer uso de esta tecnología pasa por el empleo de la herramienta de autor Adobe Director. Ésta permite la creación de contenidos multimedia que pueden ser reproducidos por la tecnología Adobe Shockwave Player.

Director incorpora algunas de estas herramientas. En este caso, éstas reciben el nombre de Xtras. A continuación se realiza una exposición de los distintos Xtras que se pueden emplear durante el desarrollo de videojuegos bajo Director:

XML. Director incorpora un Xtra que permite la lectura y carga de ficheros XML. Por un lado, este Xtra comprueba que el fichero XML sea un documento bien formado, es decir, que cumpla con el estándar XML y por otro, procesa dicho fichero y lo convierte en una estructura con la que poder trabajar. Esta estructura consiste en una lista de propiedades para la cual cada uno de sus valores son del tipo propiedad-valor.

3D. Adobe Director ofrece una librería que permite trabajar con ficheros en formato W3D. Estos ficheros pueden ser creados desde diferentes herramientas de autor 3D como Maya o 3D Studio Max. Esta librería incorpora los métodos necesarios para poder importar este tipo de ficheros al proyecto, precargar la escena de modo que todos los elementos 3D estén disponibles para el programador y controlar dichos elementos presentes en el mundo 3D (transformaciones, selecciones de modelos, operaciones vectoriales, etc.).

Física. En Director existen dos Xtras que permiten añadir simulación física a un proyecto. Actualmente, Director tiene un acuerdo con la compañía Nvidia para incorporar su motor PhysX. Anteriormente, Director trabajaba con Havok, con el cual ya no existe ningún tipo de acuerdo. No obstante el Xtra sigue estando disponible, pero no recibe ningún tipo de actualización. Tanto un Xtra como otro incorporan toda la funcionalidad necesaria para poder realizar simulaciones físicas y detectar y resolver las colisiones producidas por los elementos físicos del videojuego. Todas estas funcionalidades han sido descritas en la sección 2.4.

Rendering. Director posee su propio motor de render que permite visualizar de manera eficiente en la pantalla tanto el mundo 3D como los gráficos 2D.

Animación. Del mismo modo que Director permite importar ficheros 3D en formato W3D, permite importar y precargar animaciones realizadas bajo este mismo formato. Además, Director incorpora una librería con todos los métodos necesarios para manejar estas animaciones (reproducir, detener, asignar a un modelo, etc.).

Audio. Mediante las librerías de audio de Director es posible importar ficheros de audio en los formatos Mp3 y Wav así como realizar las acciones básicas con este tipo de ficheros. En su última versión, Director ha ampliado esta librería permitiendo realizar efectos de sonido de un nivel más avanzado.

Red. Para permitir a la comunicación entre los distintos jugadores y el servidor, Director ofrece un Xtra para comunicaciones y el servidor Shockwave Multiuser Server 3.Este último es una aplicación que se ejecuta en otra máquina, aunque, para permitir probar la aplicación durante su fase de desarrollo, ofrece la posibilidad de ser ejecutada en el mismo ordenador donde se encuentra Director. Cuando un mensaje procedente desde la aplicación ejecutada en el cliente llega al servidor, este lo reenvía al destinatario pertinente. Por otro lado, el Xtra es el encargado de gestionar los mensajes, buscar posibles errores y prepararlos para ser enviados a la red.
La versión actual del servidor está preparada para ser ejecutada sobre plataformas Windows y Mac, soporta los protocolos TCP y UDP y permite conexiones peer-to-peer además de gestionar hasta un máximo de 2000 usuarios simultáneos, el doble que en su versión anterior. Otras características importantes son: ejecución de scripts en el servidor, ejecución multihilo, lectura y escritura en ficheros, soporte para realizar tareas de depuración de código, soporte para múltiples IP en el mismo ordenador cliente y soporte para extender las funcionalidades del servidor implementado nuevos Xtras.

Por contra a las ventajas anteriormente expuestas, el uso de estos Xtras, así como el de otras herramientas similares empleadas en otras tecnologías, plantea una serie de inconvenientes que de un modo u otro han de ser resueltos. Estos inconvenientes se deben tanto a ciertas características inherentes al propio Xtra como al modo en el que estos son utilizados; son los siguientes:

1. Integración ad-hoc de los Xtras

La primera toma de contacto de un desarrollador con el uso de estos Xtras implica realizar un estudio previo de los mismos. Este estudio es una labor tediosa que debería realizarse el mínimo número de veces posible. No obstante, la metodología ad-hoc empleada a la hora de incorporar los Xtras en cada desarrollo no permite minimizar el número de veces que este estudio ha de llevarse a cabo. Desde un punto de vista práctico, si otro desarrollador ya ha realizado este estudio, no es óptimo que para un nuevo proyecto con un nuevo equipo de desarrollo deba realizarse de nuevo esta tarea.

En el campo de la Realidad Virtual y la Realidad aumentada existen

Middlewares cuya labor es la de integrar en un solo software todas las funcionalidades que implementan las herramientas vistas y por tanto solucionan el problema planteado. Un *Middleware* actúa como una capa de abstracción mayor que encapsula los motores o librerías que pueden ser utilizados para el desarrollo de un tipo de aplicación concreta. De este modo el desarrollador de la aplicación tan solo deberá conocer cómo utilizar el *Middleware,* sin necesidad de realizar un estudio previo de todas las herramientas que este integra. Una buena revisión de los *Middlewares* existentes en el campo de la Realidad Virtual y Aumentada, incluyendo las referencias más importantes, se da en [22] y en [23].

Un ejemplo de este tipo de software es Delta3D. Esta herramienta fue desarrollada para la creación de sistemas de entrenamiento militar del ejército estadounidense. Las librerías sobre las cuales ha sido desarrollado este software son: Open Scene Graph [24] (OSG) utilizado como motor de render, Open Dynamics Engine [25] (ODE) empleado como motor de físicas y de detección y resolución de colisiones, Open Audio Library [26] (OpenAL) utilizado como librería de manejo de audio, Character Animation Library 3D [27] (Cal3D) usado para la animación de modelos 3D y Python [28] usado como lenguaje de scripting. Para obtener una información más detallada acerca de estas librerías pueden consultarse las referencias indicadas.

Otro ejemplo significativo es el de 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 [30] 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. Al igual que Delta3D, SUED encapsula OSG, ODE, Cal3D y OpenAL. Además incluye Xerces [29] para permitir la carga y tratamiento de ficheros XML. En [30] se plantea como punto criticable a Delta3D el hecho de que éste no incorpore un método de configuración externo y accesible para usuarios que carecen de conocimientos técnicos. Ésta es la principal razón por la que SUED incorpora Xerces. Gracias a esta herramienta, toda la configuración de este *framework* se realiza a través de un fichero XML externo.

En [30] y en [31] se analizan con detalle Delta3D y SUED 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 el uso de un *Middleware*, que encapsule en una capa los distintos motores o librerías empleados en el desarrollo de una aplicación, mejora el proceso de producción, ya que por un lado sirve como base de desarrollo y

por otro reduce los costes del mismo. Además, la posibilidad de realizar una configuración de manera externa y a través de un entrono accesible para usuarios sin conocimientos técnicos es un valor añadido para este tipo de herramientas.

2. Presencia de inconsistencias

Aunque una de las ventajas del uso de estos Xtras es su robustez y fiabilidad, en ocasiones es posible observar en los mismos ciertas inconsistencias. Éstas pueden ocasionar que el Xtra no se adapte completamente a las necesidades de una aplicación determinada. El hecho de que éstos Xtras sean de tipo propietario no permite al desarrollador acceder al código para poder subsanar la inconsistencia en cuestión. Por ello, los problemas intrínsecos a cada Xtra pueden dificultar el avance del desarrollo y aunque las empresas desarrolladoras suelen informar de estos errores, es rara la vez en la que estos problemas son resueltos con la suficiente prontitud.

La única solución posible es la de que los programadores inviertan parte de su tiempo y esfuerzo en la creación de "work-arounds" que permitan solventar cada una de las posibles inconsistencias localizadas. No obstante, se puede apreciar con claridad, que no es óptimo el tener que realizar esta tarea para cada proyecto. La solución ideal pasa por que, una vez tratado y resuelto el error, la solución planteada pueda ser usada por los futuros equipos de desarrollo. Es posible que el/los programador/es que en su día dieron con la solución del problema no estén presentes en futuros proyectos, bien porque ya no formen parte de la plantilla de la empresa o bien porque estén trabajando en otro proyecto.

La conclusión a la que se llega tras el análisis de este inconveniente es la de que para los diferentes errores o inconsistencias que puedan ser detectados en los Xtras, se ha de proponer una solución genérica que pueda ser empleada en futuros proyectos, evitando así que los programadores tengan que invertir su tiempo y esfuerzo en solucionar una y otra vez los mismos errores.

3. Ausencia de un motor de inicialización/finalización

Al igual que las distintas áreas de la fase de producción, vistas en la sección 2.3, plantean características comunes y extrapolables a un motor, la estructura típica de un juego y la fase de inicialización del mismo plantean estas mismas semejanzas.

Tal y como se ha visto en el apartado 2.3, tanto los primeros videojuegos de la historia como los juegos actuales, siguen una estructura típica atendiendo a la secuencia de eventos que se producen desde el momento de su inicialización hasta su finalización.

En primer lugar, se realiza la inicialización del programa principal. En este paso se crean las estructuras de datos necesarias para el funcionamiento del juego. Estas estructuras contienen todas y cada una de las entidades del juego ya inicializadas. Seguidamente, se ejecuta el bucle principal del juego hasta la finalización de la partida actual. Una vez finalizada, se realizan las llamadas pertinentes para la finalización del juego y se devuelve al jugador al estado inicial donde puede elegir jugar una nueva partida.

En resumen, la filosofía de diseño de un motor, que es la de extraer funcionalidad común, no está considerada desde el punto de vista de la estructura interna de un juego y del proceso de inicialización del mismo. Por lo tanto, todas las ventajas que ofrece el uso de motores no son aprovechadas desde este punto de vista.

4.4 Conclusiones

En este capítulo se ha tratado la actual tendencia de las empresas desarrolladoras de videojuegos hacia el mercado casual y concretamente hacia el mercado de los juegos online multijugador, así como la problemática existente debida al modelo de producción que estas empresas siguen.

Por otro lado, se ha visto como, cada vez más, las empresas invierten parte de su capital en la creación y adquisición de herramientas que permitan agilizar y abaratar los costes de producción.

Además, tras una valoración de las herramientas existentes para la creación de este tipo de juegos, se ha llegado a la conclusión de la idoneidad de Adobe Director y Adobe Shockwave Player como tecnologías de desarrollo y se han expuesto las utilidades que ofrecen.

El uso de Director como herramienta de desarrollo y de las librerías y motores que este incorpora supone ciertos inconvenientes que no están resueltos:

- La integración ad-hoc de las librerías y motores conlleva el estudio reiterativo de los mismos por parte de los programadores.
- Las inconsistencias o errores que pueden aparecer no son resueltas de una manera genérica, sino que para cada proyecto nuevo, el programador debe enfrentarse a ellos.
- Se observa la inexistencia de un motor que extraiga la funcionalidad común en los distintos videojuegos desde el punto de vista de la estructura interna de los mismos y de su proceso de inicialización.

Estos problemas aquí expuestos serán tratados en los siguientes puntos de la tesina. Concretamente, en el capítulo 5 se presentará el desarrollo de un motor de

producción como una posible solución a estos problemas. La filosofía de diseño del motor es la siguiente:

1. Permitir la configuración externa de los contenidos del videojuego a través de un modelo de aplicación que pueda ser manipulado por personas sin conocimientos técnicos.
2. Realizar las funciones de un motor de inicialización, extrayendo la funcionalidad común existente desde el punto de vista de la estructura interna de un juego y automatizando, en la medida de lo posible, los distintos procesos que se llevan a cabo durante la inicialización de la aplicación.
3. Actuar como *Middleware* que encapsule en una capa de un nivel de abstracción mayor los motores integrados en Director, resolviendo de forma genérica las posibles inconsistencias existentes.

El primer punto de la filosofía de diseño -*"Configuración externa"*- surge tras el estudio del análisis realizado en [30], donde se puede apreciar una notable mejora de la eficiencia en el desarrollo de sistemas virtuales distribuidos gracias a la funcionalidad de configuración externa que incorpora la herramienta SUED. Todo lo relacionado con el proceso de configuración externa del motor de producción desarrollado se trata con detalle en [1].

Posteriormente, en el capítulo 6 se presentará el desarrollo de un videojuego casual 3D online multijugador usando como base de programación el motor de producción. Y seguidamente se realizará una comparativa, atendiendo a los costes de producción, entre el juego desarrollado y otros juegos, que permitirá valorar la aportación del motor de producción.

Capítulo 5

Desarrollo del motor de producción

5.1 Introducción

En este capítulo se va a estudiar el desarrollo del motor de producción planteado como solución a la problemática existente en el desarrollo de juegos casuales online multijugador bajo la herramienta de autor Adobe Director. Este estudio se va a realizar atendiendo a la filosofía de diseño del motor, tratada en la sección 4.2. En concreto, en la sección 4.3 se trata la automatización que el motor de producción realiza de todos los procesos que se realizan durante la inicialización del videojuego. En la sección 4.4, se estudia el carácter *Middleware* que incorpora el motor de producción, debido a su capacidad encapsular en una capa de un nivel de abstracción mayor los distintos motores integrados en Director. Seguidamente, en la sección 4.5 se describen ciertas funcionalidades adicionales que incorpora el motor de producción y que no han podido ser incluidas en las secciones anteriores. Por último, en la sección 4.6 se realizan las conclusiones pertinentes a este capítulo. El tercer pilar sobre el que se sustenta la filosofía de diseño del motor de producción –*"Configuración externa"*- se analiza de manera detallada en [1]. En dicho trabajo se trata el modo en el que el motor de producción permite configurar distintos contenidos del videojuego a través de un modelo de aplicación externo que puede ser manipulado por personas sin conocimientos técnicos.

5.2 Filosofía de diseño del motor de producción

Tras el estudio realizado en el capítulo anterior de los distintos problemas que se plantean durante la fase de desarrollo de un videojuego casual online multijugador, se ha desarrollado, en colaboración con el trabajo realizado en [1], un motor de producción que permite, en la medida de lo posible, paliar estos inconvenientes y permitir que los costes de desarrollo de estos juegos se vean mejorados.

La filosofía de diseño que se ha establecido a la hora de desarrollar el motor de producción tiene como base los siguientes principios:

1. Configuración externa.
2. Motor de inicialización.
3. Middleware.

El primero de estos principios, “Configuración externa”, hace referencia a que el motor de producción debe incorporar la posibilidad de, a través de un fichero externo, poder configurar ciertos parámetros del juego. Este fichero externo ha de ser fácil de manipular y de modificar, incluso por personas sin conocimientos técnicos, de modo que estos parámetros puedan ser modificados sin mayor esfuerzo. El estudio y análisis en profundidad de este primer principio quedan fuera del alcance de esta tesina y se tratan en el trabajo de colaboración realizado en [1].

El segundo, “Motor de inicialización”, determina que el motor de producción debe comportarse como un motor de inicialización del juego. Del mismo modo que otros motores existentes, vistos en la sección 2.4, extraen distintas funcionalidades comunes en los videojuegos dentro de las distintas áreas existentes, el motor de producción debe extraer la funcionalidad común existente desde el punto de vista de la estructura interna de un juego y automatizar, en la medida de lo posible, los distintos procesos que se llevan a cabo durante la inicialización de la aplicación.

El último principio, “Middleware”, indica que el motor de producción debe actuar como una capa de abstracción que encapsule los distintos motores existentes en la herramienta Adobe Director. Esta capa debe servir al programador como medio a través del cual acceder a la funcionalidad de estos motores de una manera más fácil.

5.3 Motor de inicialización

Tal y como se ha visto en la filosofía de diseño del motor de producción, éste debe incorporar como una de sus funcionalidades principales la de actuar como motor de inicialización, extrayendo y automatizando aquellas funciones comunes, en lo que a estructura e inicialización del videojuego se refiere.

Atendiendo a la estructura general de un videojuego, vista en la sección 2.3, las funciones extrapolables que se pueden realizar de forma automatizada por el motor de producción abarcan desde el primer punto, inicialización del programa principal, hasta la finalización del juego, pasando por las llamadas pertinentes al bucle principal del juego. La ejecución propia del bucle principal de juego es independiente del motor, y debe ser creada para cada juego nuevo.

La fase de inicialización del programa principal consta de una serie de funciones que a continuación se enumeran y posteriormente se tratan con mayor detalle:

1. Carga externa de datos.
2. Precarga de datos, incluyendo la inicialización de la estructura de datos final.
3. Inicialización de animaciones.
4. Inicialización de la interfaz.
5. Inicialización de la física.
6. Inicialización de las reglas.
7. Inicialización de colisiones.
8. Inicialización del juego.

La automatización por parte del motor de producción de todos estos procesos conlleva la obtención de una estructura base sobre la que empezar a desarrollar un nuevo proyecto. De este modo, los programadores solo tienen que preocuparse de desarrollar las funcionalidades específicas del juego, como pueden ser, las reglas, las entidades o el bucle principal del juego.

El desarrollo de los puntos 1, 2 y 3 se ha llevado a cabo en el trabajo de colaboración realizado en [1]. Por ello, en esta tesina tan solo se realiza un breve resumen que permite al lector entender el comportamiento del motor de inicialización en su totalidad.

En primer lugar, la carga externa de datos permite importar todos los scripts, animaciones, escenas 3D, imágenes y ficheros de audio que se vayan a utilizar en el juego y que son necesarios para el correcto funcionamiento del mismo.

En cuanto a la precarga de datos, se puede decir que, una vez procesado el modelo de aplicación y obtenida la primera estructura de datos sobre la que trabajar, se procesan dichos datos y se crea una serie de estructuras de datos nuevas, correspondientes a las entidades que van a formar parte del juego.

Por último, atendiendo a la inicialización de las animaciones, decir que en esta fase se importan e integran todas las animaciones necesarias para el correcto funcionamiento del juego. Posteriormente, dichas animaciones se asignan a los objetos pertinentes gracias a un sistema de nomenclatura especial. 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*.

Una vez vistas, a modo de resumen explicativo, las fases del proceso de inicialización que se desarrolladas en [1], en los siguientes puntos se explica con mayor grado de detalle todos aquellos aspectos que dentro del motor de inicialización han sido

elaborados para la realización de este tesina.

5.3.1 Inicialización de la interfaz

Al inicio de la fase de inicialización del programa principal se ha importado a Director el script correspondiente al interfaz del juego. Éste, aunque sigue la misma estructura para todos los juegos, no tiene porque ser igual para todos ellos. Cada interfaz consta de una serie de elementos gráficos en función de las características y requerimientos de cada juego.

El motor de producción automatiza la creación de los *overlays* y la carga de los gráficos necesarios para que el juego proporcione un *feedback* al jugador.

Un overlay consiste en un gráfico que, independientemente de las transformaciones de la cámara, aparece siempre en una coordenada determinada de la pantalla y es siempre visible por encima del resto de elementos presentes en la escena. Algunos ejemplos de overlays presentes en videojuegos pueden ser los indicadores de nivel de vida en un juego de lucha o la cantidad de munición en un juego de acción en primera persona.

Haciendo uso del modelo de aplicación se definen los overlays que desean crearse durante esta fase del juego, de manera que en el momento en el que el programador necesita utilizarlos éstos ya están operativos. El motor de producción permite, a través de los parámetros definidos en el modelo de aplicación, inicializar los overlays en una posición determinada de la pantalla, indicar si aparecen ocultos o visibles, definir su tamaño y otras características que permiten la correcta configuración inicial de los overlays.

La clase Overlay tiene implementadas las funciones necesarias para la creación de los distintos tipos de overlay, así como otras funciones que permiten modificarlos, tanto en su posición como en su visibilidad o apariencia.

Para cargar dichos gráficos en Director, se utiliza la misma función que se ha utilizado para importar las demás entidades. Una vez importados los gráficos, se utilizan estos a modo de texturas para la creación de los overlay de tipo imagen.

Una característica importante de los overlays es que pueden ser dotados de comportamiento. Mientras que algunos de ellos solo son de carácter informativo, pueden existir otros que tengan algún tipo de funcionalidad, como por ejemplo, un overlay que simule un botón que al pulsar sobre él se active una animación. Este comportamiento viene definido en los scripts anteriormente importados.

Por otro lado, el motor permite la carga de gráficos 2D que pueden forman parte del interfaz del juego pero no se comportan necesariamente del mismo modo que los

overlays. El diseñador puede pretender crear un juego más inmersivo empleando un interfaz más integrado en el propio juego. Por ejemplo, el nivel de munición puede aparecer directamente en el arma en lugar de emplear un overlay. La aparición de estos gráficos en la escena debe ser codificada por el programador, no obstante, el motor de producción habrá precargado los mismos previamente.

El motor de producción permite, por tanto, mediante los parámetros definidos en el modelo de aplicación, automatizar totalmente el proceso de carga de gráficos, así como la creación de los overlays pertinentes para la creación del interfaz de usuario encargado de proporcionar el feedback necesario para orientar al jugador en el desarrollo de la trama del juego.

5.3.2 Inicialización de la física

Una vez importado el script de física que va a ser utilizado en la ejecución del juego, el siguiente paso es crear el mundo físico. Para ello es necesario haber definido previamente en el modelo de aplicación el motor de física que se va a utilizar, las propiedades físicas generales y las propiedades dinámicas de cada objeto rígido.

Antes de empezar a trabajar con los objetos es necesario realizar un paso previo, la inicialización de la física. Para ello es necesario hacer, en primer lugar, un *new* del Xtra de física correspondiente. En segundo lugar, se realiza la inicialización como tal del mundo físico. Para ello, se hace uso de la función *init* del motor de producción. Esta función se encarga de realizar todo el proceso de inicialización del mundo físico independientemente del Xtra de física empleado.

Una vez inicializada y configurada la física general del mundo, se da paso a la creación de los distintos cuerpos rígidos también definidos en el modelo de aplicación. Al igual que con la función de inicialización, el motor llamará a la función genérica de creación de cuerpos rígidos, pasándole como parámetros las características definidas para cada objeto, y la función internamente gestionará que pasos debe realizar para la creación de dichos cuerpos rígidos.

Las características generales de un cuerpo rígido hacen referencia al tipo y la forma del mismo. Por un lado, el tipo hace referencia a su movilidad, pudiendo ser estático o dinámico. Por otro, la forma, hace referencia a como el cuerpo rígido envuelve el modelo que representa. Ésta puede ser una caja, una esfera, un modelo cóncavo o un modelo convexo. Un bueno ejemplo es la imagen [33] que se muestra a continuación.

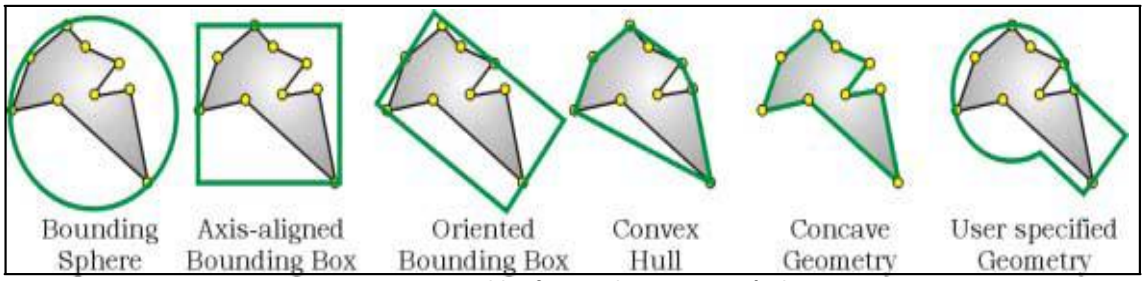

Figura 12. Posibles formas de un cuerpo rígido.

Las demás propiedades definidas para cada cuerpo rígido son las características dinámicas de éste, como son la masa, la fricción, la restitución, etc. Si alguno de ellos no se define, el valor que se asigna a la propiedad no definida, es la definida por defecto en el mundo físico general.

5.3.3 Inicialización de las reglas

En este punto de la inicialización, se aporta la funcionalidad necesaria para automatizar, en la medida de lo posible, el desarrollo de las reglas de juego. El motor permite, gracias a los parámetros definidos en el modelo de aplicación, la asignación de las reglas pertinentes a cada uno de los juegos inicializados.

En la inicialización de las reglas se definen e inicializan todas las variables internas que toman partido en la posterior toma de decisiones, así como ciertas estructuras de datos que permiten el correcto funcionamiento del juego en base a dichas reglas.

Además de lo mencionado, existen juegos en los que las reglas pueden ir variando en función del nivel o del estado de juego en el que el jugador se encuentra (nivel de vida, cantidad de munición, daño provocado por el ataque enemigo, etc.). Estos factores cambiantes han sido previamente definidos en la fase de diseño del juego y por tanto el modelo de aplicación presenta las etiquetas oportunas para su posible modificación.

El uso del motor de producción permite definir, para los distintos niveles de juego, el valor de las variables o estructuras de datos, anteriormente mencionadas, así como asignar diferentes scripts de reglas para cada uno de ellos. Este último punto es interesante a la hora de crear un juego en el que cada nivel suponga realizar una prueba cuyas reglas sean totalmente diferentes.

Por otro lado, atendiendo a la labor que realiza el programador, el motor permite a este realizar una programación modular de sus reglas. La carga externa de los scripts de reglas permite la creación de librerías de reglas que pueden ser empleadas en diversos juegos. De este modo, se puede compartir el código desarrollado en las diferentes librerías sin necesidad de duplicar el mismo.

Por todo esto, aunque el desarrollo de un juego nuevo provoque el correspondiente desarrollo de sus reglas, gracias al motor de producción, es posible modificar las mismas de manera externa al mismo y sin necesidad de codificar nada dentro de los propios scripts del juego. Esto permite separar la labor del programador y del diseñador, ya que el primero se limita a programar cada uno de los scripts de reglas necesarios, mientras que el segundo puede realizar variaciones de las mismas de manera externa, así como decidir que reglas son las asignadas a cada estado o nivel del juego.

5.3.4 Inicialización de las colisiones

Atendiendo a la inicialización de las colisiones, el motor de producción permite habilitar/deshabilitar las colisiones entre los distintos pares de cuerpos rígidos existentes en la escena y asignar *callbacks* a cada uno de los pares de cuerpos rígidos. Un *callback* es una función a la que se llama cada vez que el par de cuerpos rígidos registrado colisiona. Normalmente, este *callback* recibe como parámetro los detalles de la colisión (cuerpos rígidos implicados, normal de la colisión, puntos de contacto, etc.), aunque esto puede variar en función del motor de colisiones utilizado. Si el *callback* no es asignado la colisión puede ser o no resuelta pero, en cualquier caso, es ignorada.

5.3.5 Inicialización del juego

Llegados a este punto, la estructura de datos necesaria para la ejecución del videojuego está ya preparada y se puede pasar a la ejecución del bucle principal de juego. Aun así, todavía hay una serie de funciones de inicialización que no se han podido llevar a cabo durante la precarga, debido a la dependencia de otros datos, como son, las opciones de mesa ofrecidas a los jugadores. Es importante minimizar el número de acciones a realizar en este momento de la ejecución, dejando solo aquellas funciones que no se puedan realizar en la precarga del juego, puesto que de no ser así, las ventajas que la precarga proporciona se perderían. Estas funciones de inicialización corresponden al propio juego y se llevan a cabo al inicio de cada nivel o juego nuevo.

En primer lugar, al inicio de la ejecución del bucle principal del juego, el motor de producción permanece en un estado a la espera de que todos los jugadores acepten el inicio de la partida. Es durante la estancia en este estado donde se realizan las llamadas de inicialización restantes. Este proceso de inicialización final ha de ser desarrollado de manera independiente para cada juego, no obstante, el motor de producción consigue que estas tareas se ejecuten de manera automática y en un orden determinado.

Una vez comienza el juego, el motor de producción cambia su estado. Aquí el motor ejecuta una vez por cada frame el bucle principal del juego que realizará las acciones pertinentes en función del nivel actual y del estado en el que se encuentre. Una vez ejecutado el bucle principal, el motor comprueba si, dadas ciertas condiciones

definidas en las reglas del juego, el nivel actual del juego ha finalizado y de ser así se actualiza a un nuevo estado.

Una vez se produce la finalización del nivel se comprueba si existen nuevos niveles. En el caso de que existan, se procede a la carga e inicialización del nuevo nivel y se actualiza el estado del motor para volver a realizar las llamadas pertinentes al bucle principal del juego. Esta inicialización utilizará los datos procesados y almacenados en la fase de precarga, realizando de nuevo, únicamente, las funciones de inicialización del juego necesarias, reduciendo considerablemente en el tiempo de carga del nuevo nivel.

Al finalizar todos los niveles el motor realiza las llamadas pertinentes para la finalización del juego y se devuelve al jugador al estado inicial donde puede elegir jugar una nueva partida.

5.4 Middleware

El tercer pilar sobre el que se sustenta el motor de producción es el de su capacidad de actuar como *Middleware,* comportándose como una capa de abstracción mayor que encapsula los motores existentes en Director. Los objetivos principales que se persiguen al implementar esta funcionalidad son, por un lado, minimizar la complejidad del desarrollo de videojuegos que precisen del uso de estos motores y , por otro, gracias a esta reducción de la complejidad, reducir el tiempo de desarrollo empleado para la producción de cada juego. En concreto el motor de producción se centra en el encapsulamiento de los motores de física y de detección y resolución de colisiones así como de la herramienta de gestión de sistemas multijugador previamente implementada en la empresa Exelweiss Ent.

5.4.1 Integración de la herramienta de gestión multijugador (MUS)

Previa a la creación del motor de producción la empresa ya poseía implementada una aplicación que permite la gestión del sistema multijugador y que es utilizada en todos los juegos del portal *www.mundijuegos.com*. Esta aplicación es conocida como MUS. La función principal del MUS es la de crear un sistema unificado de gestión de usuarios y elementos multijugador en el que todo jugador se encuentra cada vez que se dispone a jugar. El interfaz del MUS se divide en tres zonas claramente delimitadas: zona de chat, zona de ajustes y zona de mesas, donde los jugadores esperan a los rivales antes de comenzar una partida. El funcionamiento del MUS con mayor detalle, así como los detalles de implementación pertenecen a la propiedad intelectual de Exelweiss Ent. y quedan fuera del ámbito de esta tesina.

Para cada juego nuevo es necesario repetir una serie de acciones de modo que el MUS quede integrado en el proyecto y se pueda hacer uso del mismo. La labor del motor de producción en este punto es la de automatizar la integración del MUS dentro del proyecto.

En primer lugar es necesario crear un Cast de Director y nombrarlo mus_client. Seguidamente, todos los scripts pertenecientes al MUS han de ser importados dentro de este Cast, así como los gráficos necesarios para la creación del interfaz del MUS. Y por último, se han de llevar a la línea de tiempo y situar en el frame correspondiente los comportamientos *"preload"*, *"loopInicio"*, *"loadNet"* y *"loop"*. Una vez realizados todos estos pasos el proyecto ya tiene integrado el MUS.

La automatización de todo este proceso por parte del motor de producción evita tener que realizar los mismos pasos una y otra vez cada vez que se comience con el desarrollo de un juego nuevo. De este modo el desarrollador no debe preocuparse de la integración del MUS en su proyecto y se evitan posibles errores que pudieran surgir al realizar esta tarea.

Además de esto, empleando el modelo de aplicación, se puede definir si se desea que el MUS sea integrado o no en el juego. Esta opción es verdaderamente útil durante la fase inicial de la producción del videojuego, ya que en esta fase todavía no son desarrolladas las funcionalidades multijugador del mismo. Por lo tanto, el programador puede prescindir del uso del sistema MUS hasta que lo necesite, simplemente cambiando una línea del XML de configuración.

Paralelamente, se ha desarrollado nueva funcionalidad que el motor de producción incorpora al MUS una vez este es importado al proyecto. Toda la funcionalidad adicional que se ha implementado persigue el mismo objetivo: evitar las posibles trampas realizadas por los jugadores; la gran lacra de los juegos multijugador.

En primer lugar, se van a describir las trampas más comunes que, según la propia experiencia de la empresa, realizan los jugadores, para posteriormente exponer la nueva funcionalidad que incorpora el motor de producción y que permite evitarlas.

Una de las trampas más comunes que los jugadores realizan es la de cortar la comunicación entre la aplicación Shockwave y el servidor a través de un *firewall*. Al cortar la comunicación la partida queda detenida indefinidamente y nunca se da por finalizada, de modo que el resto de jugadores tan solo tienen como opción abandonar la partida y que esta quede anulada. Los jugadores suelen realizar esta trampa cuando perciben que van a perder la partida.

Otro tipo de artimaña realizada por los jugadores tramposos consiste en hacer uso de programas que permiten capturar los paquetes enviados a través de la red y conocer ciertos datos de la partida a los que realmente no deben tener acceso. Este tipo

de programas son empleados muy a menudo en juegos de cartas con el objetivo de conocer las cartas del rival.

Por último, una trampa algo más sofisticada es la que algunos usuarios realizan haciendo uso de programas como *"Cheat Engine"*. Programas como este permiten conocer las direcciones de memoria donde son almacenadas las variables del juego. Gracias a esto, el jugador puede modificar el valor de estas variables, cambiando totalmente el estado de la partida en su beneficio propio.

Para evitar que los usuarios detengan la comunicación ente la aplicación Shockwave y el servidor se ha creado un sistema de envío y recibo de mensajes que verifican la actividad de un jugador. Este sistema crea un temporizador que determina el tiempo máximo que puede transcurrir sin que una aplicación ejecutada en el cliente se comunique con el servidor. Si pasado este tiempo no se ha recibido ningún mensaje de un cliente en concreto, se crea otro temporizador de menor tiempo y se envía un mensaje al cliente. Si tras la finalización del segundo temporizador todavía no se ha recibido una respuesta al mensaje enviado el jugador es expulsado del juego por inactividad.

Por otro lado, para evitar la captura de paquetes, se ha creado un sistema de encriptación de mensajes. Este sistema, más que evitar propiamente la captura de los paquetes, evita que una vez capturados estos sean comprensibles por el usuario tramposo. Para el correcto funcionamiento de este sistema es necesario que tanto el cliente como el servidor tengan las funciones apropiadas de encriptación y desencriptación, ya que sin ellas no se puede descifrar el mensaje. El programador puede hacer uso de este sistema siempre que lo desee, pudiendo encriptar tan solo los mensajes que contengan información relevante.

Por último, la forma con la que se logra que los jugadores no puedan obtener las direcciones de memoria de las variables de juego consiste en un sistema que cambia constantemente dichas direcciones, de manera que sea imposible seguir la pista de la variable mediante el uso del tipo de programas visto. Este sistema se ha implementado de una manera muy sencilla; almacenando las variables en listas. Cada vez que se asigna un nuevo valor a una variable se crea una nueva lista en la que el valor es almacenado y posteriormente se asigna como valor de la variable la lista creada. De este modo, cada vez que el valor de la variable se modifica la dirección de memoria a la que apunta cambia, haciendo imposible seguir la pista a la variable.

En resumen, el motor de producción realiza la integración del MUS de manera automática, evitando tener que realizar la misma tarea para cada juego y además incorpora la funcionalidad necesaria para evitar las trampas más comunes.

5.4.2 Integración del motor de física

Tal y como se ha visto en el capítulo 3, al adquirir una licencia de Adobe Director se obtienen las licencias de los motores de física Havok y PhysX.

El motor de producción permite la integración de cualquiera de éstos dos motores en un videojuego de un modo completamente versátil, de tal manera, que la programación propia del juego no deba ser modificada debido a un cambio en el motor físico empleado.

Para lograr esto, el motor de producción actúa como un intérprete de órdenes entre el programador y el motor físico correspondiente. Es decir, cada función empleada por el programador se traduce en la correspondiente secuencia de llamadas a las funciones del motor físico.

Con esto, se consigue que las funciones empleadas por el programador sean las mismas (mismo nombre y mismo tipo y número de parámetros) independientemente del motor físico empleado, lo que le permite familiarizarse rápidamente con ellas. De este modo, se logra que el programador pueda acceder a toda la funcionalidad del motor físico sin la necesidad de realizar un estudio previo del mismo. El conocimiento en profundidad del funcionamiento de los distintos motores es una labor complicada y tediosa, por lo que es óptimo que tan sólo se realice una única vez por cada motor.

Algunas funciones físicas a destacar dentro del motor de producción son:

- Simulate(timeStep,subSteps): realiza un paso de simulación atendiendo al tiempo de simulación y a los subpasos de simulación establecidos.
- ApplyForce(rigidBody,vectorForce,forcePoint): aplica una fuerza a un cuerpo rígido en un punto determinado.
- ApplyTorque(rigidBody,vectorForce,forcePoint):aplica una fuerza de giro a un cuerpo rígido en un punto determinado.

Además de incorporar esta funcionalidad, se ha realizado un estudio de los errores más comunes que se realizan a la hora de emplear los motores PhysX y Havok y se ha realizado un tratamiento de los mismos. Ambos motores poseen ciertas funciones que tienden a generar un error cuando se utilizan en momentos inapropiados. Un ejemplo de este tipo de funciones es *destroy* en el caso de PhysX y *shutdown* en el caso de Havok. Ambas funciones se encargan de destruir los cuerpos rígidos existentes en el mundo físico y de resetear la simulación a su estado inicial. Esta función nunca puede ser llamada si la simulación física no ha sido previamente inicializada. En este caso concreto el motor de producción, comprueba que realmente la simulación ha sido inicializada antes de proceder a su destrucción, evitando así un posible error durante la ejecución del juego. Al igual que en el caso de ejemplo existen otras funciones para las cuales se realizan las comprobaciones pertinentes antes de que estas sean ejecutadas, garantizando así que no generen errores en tiempo de ejecución.

Independientemente del motor de física empleado, en los videojuegos online multijugador un requisito prácticamente imprescindible que éste ha de cumplir es el de ser determinista. El determinismo es la teoría que supone que la evolución de los fenómenos naturales está completamente determinada por las condiciones iniciales. Es decir, en dos o más simulaciones físicas, en las que las condiciones físicas iniciales son las mismas, el resultado definitivo de la simulación ha de ser idéntico. La importancia del determinismo radica principalmente en el ahorro de ancho banda que supone. El uso de un motor físico no determinista supondría saturar el servidor con el envío de mensajes que actualicen, en cada llamada al bucle principal del juego, el estado del mundo físico en cada uno de los clientes. Esto es debido a que la evolución del mundo físico es diferente en cada uno de los clientes y aparece la necesidad de realizar un proceso de sincronización mediante el envío de mensajes cliente-servidor-cliente. Por el contrario, un motor determinista permite que los mensajes se reduzcan tan solo al envío de las fuerzas que actúan sobre los diferentes cuerpos rígidos, ya que el comportamiento de estos es idéntico en todos y cada uno de los clientes sobre los cuales se ejecuta la aplicación.

El motor de producción ha sido desarrollado de manera que se garantiza el determinismo de la simulación física bajo los motores Havok y PhysX. Para ello, es necesario que el motor de inicialización destruya e inicialice el motor de física antes de cada simulación. Si no se sigue esta pauta, la simulación realizada en cada uno de los clientes puede variar, originando los problemas vistos anteriormente. Además el motor sincroniza los instantes de simulación en cada cliente antes de cada simulación, ya que de no ser así el determinismo del motor de física no está garantizado.

Conseguir que el determinismo esté garantizado ha sido una de las labores más complicadas y costosas de todo el desarrollo del motor de producción. Aunque aparentemente la solución es sencilla, llegar a ella ha supuesto un estudio detallado del funcionamiento interno de cada uno de los motores así como la realización de ciertas pruebas que han permitido afirmar el cumplimiento del determinismo. Las pruebas realizadas han consistido en la creación de diferentes mundos físicos sobre los que se han aplicado fuerzas de manera aleatoria. Tras la creación de estos mundos, el sistema se deja en funcionamiento durante más de 48 horas y tras este periodo de tiempo se comprueba que las diferentes simulaciones, que han sido ejecutadas cada una en una máquina distinta, han dado como resultado el mismo estado del mundo físico. Como ya se ha comentado, tanto el estudio en profundidad previo de cada uno de los motores como las distintas pruebas que han tenido que ser realizadas suponen un gran esfuerzo que no debería volver a realizarse para ningún otro juego.

Por todo lo visto, se concluye que el motor de producción integra los motores de física Havok y PhysX permitiendo, por un lado, versatilidad a la hora de cambiar entre uno y otro, por otro lado, evitando la ejecución de ciertas funciones que podrían generar errores en tiempo de ejecución y, por último, garantizando el comportamiento determinista de ambos motores.

5.4.3 Integración del motor de colisiones

En general, un motor de física incorpora su propio sistema de detección y resolución de colisiones, por lo que comúnmente se conoce al motor de física como el motor que engloba estos dos aspectos de la simulación física. Los motores de física Havok y PhysX incorporan sus propios algoritmos de detección y resolución de colisiones, pero ambos plantean ciertas diferencias.

La integración, por parte del motor de producción, del motor de colisiones se realiza de manera idéntica a la realizada para integrar el motor de física. Es decir, por un lado, el motor de producción actúa como un intérprete entre el programador y los diferentes motores de colisiones y, por otro, evita los errores más frecuentes que se producen al usar estos motores.

Por esto, el uso del motor de producción permite hacer transparente al programador las diferencias existentes en ambos motores, evitando así que este deba modificar el código del juego en función del motor de colisiones con el que definitivamente se trabaje. Una de estas diferencias, que puede servir como ejemplo para ilustrar el papel desempeñado por el motor de inicialización es la siguiente: mientras que en el motor Havok, los procesos de registrar una colisión y asignarle su función *callback* se realizan en una única llamada, en el motor PhysX estas dos tareas se realizan de forma independiente.

Paralelamente, se ha desarrollado una librería que permite al programador obtener información acerca del estado de las colisiones de la simulación actual. Esta librería guarda todas las colisiones producidas durante una simulación, de esta manera se pueden realizar las consultas pertinentes. Para potenciar el uso de esta librería se ha realizado un estudio de las consultas más frecuentes realizadas en torno a la colisiones de los cuerpos rígidos y posteriormente se han creado las funciones pertinentes. Mediante el uso de estas funciones el programador puede conocer cosas como: si dos cuerpos rígidos determinados han colisionado, si un determinado cuerpo rígido tiene habilitadas o deshabilitadas sus colisiones, el *callback* registrado para una colisión determinada, etc. Todas las funciones añadidas son potencialmente usadas en el posterior desarrollo del videojuego, por lo que añadir esta funcionalidad al motor de producción permite al desarrollador conocer el estado de las colisiones de una manera rápida y accesible.

En resumen, el motor desarrollado emplea con el motor de colisiones las mismas técnicas que las vistas en el motor de física. Es por ello, por lo que en este caso, el motor también se comporta como un intérprete entre el motor de colisiones y el programador permitiendo así que este último tenga acceso a toda la funcionalidad del motor de colisiones sin la necesidad de realizar un estudio previo del mismo. De forma paralela, gracias a la librería desarrollada, se consigue que el programador tenga un acceso rápido y sencillo al estado de las colisiones ocurridas durante la simulación.

5.5 Funcionalidad adicional

Adicionalmente, el motor de producción posee cierta funcionalidad extra, que no ha podido ser descrita en los apartados anteriores, pero que merece la pena tratar, ya que puede ser de gran utilidad en el desarrollo de juegos casuales online multijugador:

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

Paralelamente a la funcionalidad adicional mencionada, en [1] se tratan en profundidad nuevas funcionalidades adicionales que no son vistas a lo largo de esta tesina. Dichas funcionalidades se resumen en: creación de escenarios abiertos y creación de modelos siempre visibles. Para conocer con detalle estas funcionalidades el lector puede consultar la bibliografía pertinente.

5.5.1 Uso de proxys

El motor de producción permite trabajar con modelos que actúan como *proxy* de los modelos existentes en la escena 3D. Un videojuego puede contener ciertos modelos con un alto número de polígonos y por tanto, los cálculos físicos y de detección y resolución de colisiones que han de realizarse para éstos suponen un alto coste computacional. Un *proxy* es un modelo 3D con un número reducido de polígonos que se utiliza para representar al modelo 3D original presente en la escena. Gracias a que el número de polígonos del *proxy* es sensiblemente menor que el del modelo original los cálculos pueden realizarse con mayor velocidad a costa de perder cierta precisión. Los *proxys* no son visibles para el jugador, ya que su finalidad es la de simular el comportamiento físico del modelo original sin que el juego pierda calidad gráfica.

La asignación del *proxy* a su modelo correspondiente es decisión del diseñador. No obstante, una vez que el diseñador establece que modelos tendrán un *proxy* el programador necesita codificar esta funcionalidad para que el *proxy* funcione como tal. En este punto, la función del motor de producción es la de automatizar esta labor de manera que el programador no tenga que realizarla. Para poder realizar tal automatización es necesario hacer uso de una nomenclatura específica para los modelos *proxy*. En concreto, un modelo *proxy* debe nombrase igual que el modelo al que representa pero con el prefijo proxy. El motor detecta este tipo de modelos y realiza la

asignación pertinente de estos a los modelos 3D originales. A partir de este momento, todos los cálculos físicos y de colisiones que deban hacerse para el modelo 3D original se realizarán a través de su *proxy*.

De este modo, el uso de modelos *proxy* se realiza de manera totalmente transparente al programador, permitiendo que el diseñador decida externamente que modelos precisan del uso de *proxys* empleando la nomenclatura vista. El uso de *proxys* es más que recomendable su uso en aquellos modelos que contengan un número elevado de polígonos.

5.5.2 Scripting desde 3D Studio Max

El motor de producción incorpora la posibilidad de poder realizar scripting desde la herramienta 3D Studio Max. Esta funcionalidad ha sido desarrollada debido a las carencias existentes en esta herramienta a la hora de exportar al formato W3D. Director posee muchas más *features* de las que el exportador permite, por ello, el grafista no puede explotar todo el potencial de la herramienta de manera directa, y es necesario la intervención por parte del programador. Algunos *features* que el exportador no permite son: trabajar con multitexturas, cambiar la calidad de las texturas, cambiar las coordenadas de mapeado de las texturas, modificar ciertos parámetros de los shaders, etc.

El grafista, continuamente, experimenta nuevas maneras de incorporar efectos, ajusta los parámetros de las texturas o los shaders, así como otra serie de acciones que le permiten ajustar la escena 3D a su gusto. El hecho de que para poder ver el efecto de dichos ajustes en el propio juego sea necesaria la intervención del programador supone un alto grado de solapamiento entre la labor de éste y la del grafista, con la consecuente pérdida de eficiencia que ello conlleva.

Para poder solventar estos inconvenientes, se ha creado un pequeño parser que interpreta una serie de órdenes específicas, definidas por el grafista desde la herramienta 3D Studio Max. Esta herramienta permite añadir información adicional dentro de cada modelo presente en la escena. Esta *feature* es comúnmente utilizada para describir, en formato texto, el modelo o añadir ciertas notas, no obstante, en esta ocasión se ha utilizado para que el grafista escriba aquí aquellas órdenes que posteriormente son tratadas por el parser.

Por tanto, con el desarrollo de esta nueva funcionalidad se consigue que, por un lado el programador permanezca ajeno a toda la labor que el grafista realiza en este aspecto, y por otro, que éste último pueda probar el resultado de sus modificaciones en la escena 3D una vez ésta es importada dentro de la herramienta Director.

5.5.3 Tratamiento de objetos aleatorios

Durante la inicialización de un juego, es posible que se desee que ciertos objetos aparezcan en la escena de manera aleatoria, consiguiendo así que el estado inicial del juego o de alguno de los niveles en particular presente ciertas diferencias. La aparición de este tipo de objetos en los juegos multijugador plantea una problemática que ha de ser resuelta, ya que no es factible que la aplicación genere un estado inicial diferente para cada uno de los jugadores.

Debido a que las características iniciales del juego han de ser las mismas para todos los jugadores involucrados, se ha implementado en el motor de producción la sincronización que necesariamente se ha de realizar para que esto sea así. Concretamente, el motor de producción se encarga de delegar en el jugador con identificador 1 (este jugador siempre existe) la responsabilidad de decidir el estado inicial de los distintos objetos aleatorios existentes. Una vez que la aplicación ejecutada en el cliente con identificador 1 inicializa todos los objetos aleatorios, se envía un mensaje al resto de jugadores para que los inicialicen con los mismos valores. De este modo, se consigue, por un lado, la aleatoriedad inicial deseada y, por otro, que todos los objetos del juego se encuentren en el mismo estado inicial para cada jugador.

Por tanto, gracias al modelo de aplicación, es posible asignar a ciertos objetos cierta aleatoriedad que conlleve un cambio en su estado inicial. El cambio que esta aleatoriedad supone en cada objeto debe estar definido en el propio comportamiento de cada juego. El motor de producción realiza la sincronización necesaria para que todos los jugadores obtengan como resultado de este cálculo aleatorio el mismo estado inicial de juego.

5.6 Conclusiones

En el presente capítulo se ha presentado el desarrollo del motor de producción propuesto como solución a la problemática existente en el proceso de producción de videojuegos casuales online multijugador bajo la herramienta Adobe Director. El objetivo principal del motor de producción es el de minimizar, por un lado la complejidad, y por otro, los costes de producción de este tipo de juegos. De este modo, el motor de producción pretende que su uso permita el desarrollo de futuros videojuegos de una manera más eficiente.

Para conseguir este objetivo el motor de producción se basa en tres puntos claves que definen su filosofía:

En primer lugar, la configuración externa mediante un fichero XML permite que cualquier persona sin necesidad de tener conocimientos técnicos, pueda realizar de forma rápida y sencilla ciertos cambios en el juego que permitan realizar pruebas, sin necesidad de adentrarse en el código.

Por su parte, el motor de inicialización integrado en el motor de producción, se encarga de realizar todas las tareas comunes en la estructura típica de un videojuego. Estas tareas consisten en integrar los distintos scripts y entidades que se utilizan en el propio juego, inicializar todas las estructuras de datos necesarias para el correcto funcionamiento de un videojuego, inicializar los distintos motores de los que se dispone, ejecutar el bucle principal del juego y finalizar, tanto los diferentes niveles que puedan existir como el propio juego. Todas estas tareas se tienen que realizar en cada desarrollo de un juego nuevo, por lo que gracias al motor de producción los programadores no tienen que desarrollarlas una y otra vez, pudiéndose dedicar a tareas más concretas, como la programación de los scripts propios del juego. Esto permite el acelerar el proceso de producción de cada juego y reducir así los costes de desarrollo.

Por otro lado, el motor de producción integra de manera transparente al desarrollador los motores de física y de detección y resolución de colisiones y el sistema MUS. La integración por parte del motor de producción, actuando como *Middleware,* reduce los costes de desarrollo del producto gracias a las ventajas que ofrece:

- Evita el estudio previo que ha de realizarse para los diferentes motores.
- Permite versatilidad en el cambio de motor sin que esto suponga la modificación del código de juego.
- Evita los errores más comunes que suelen producirse al emplear estos motores.

- Garantiza el determinismo de la simulación física.
- Añade funcionalidad extra que puede ser de gran utilidad para el programador a la hora de evitar las posibles trampas o conocer el estado de las colisiones de una manera rápida y sencilla.

Además de la funcionalidad básica, el motor de producción incorpora cierta funcionalidad que permite acelerar la producción de un videojuego.

En primer lugar el motor de producción automatiza el uso de *proxys*, de manera que todo el proceso se realiza de manera transparente al programador. El uso de modelos *proxy* permite disminuir el coste computacional producido por los cálculos realizados durante la simulación física.

Por otro lado, el motor de producción 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. Esto permite que el grafista pueda probar los resultados de su trabajo en la escena 3D importada a Director y que el programador permanezca al margen de esta tarea, pudiendo dedicarse así a la programación del juego.

Por último, el motor de producción trata la problemática de la aleatoriedad de ciertos objetos del juego. Al tratarse de juegos multijugador, el estado inicial de la partida ha de estar sincronizado en todos y cada uno de los clientes en los que se ejecuta la aplicación. Este proceso de sincronización es llevado a cabo por el motor de producción haciendo uso del envío de mensajes.

Además de las funcionalidades descritas, en [1] se detalla como el motor de producción también permite la creación de escenarios abiertos haciendo uso de la técnica *skydome* y la automatización del proceso de creación de modelos siempre visibles empleando la técnica de clonación de modelos.

En el próximo capítulo se expondrá un breve resumen del desarrollo de un videojuego para el cual se ha empleado como base de desarrollo el motor de producción visto en este capítulo y, a continuación, se realizará una evaluación de la mejora introducida por el motor de producción en base a los costes de producción de dicho juego comparados con los de otros dos juegos de características similares. Posteriormente, en el capítulo 6 se expondrán las conclusiones generales de esta tesina y las posibles líneas de trabajo futuro.

Capítulo 6

Análisis crítico del motor de producción

6.1 Introducción

En el presente capítulo presenta el desarrollo de un videojuego para el cual se ha empleado el motor de producción desarrollado. Posteriormente, se realiza un estudio de los costes de desarrollo de éste y de otros dos videojuegos con el objetivo de comparar los resultados obtenidos.

Tras exponer en la sección 5.2 el juego desarrollado y describir brevemente en la sección 5.3 los otros dos proyectos, en la sección 5.4 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 5.5 se exponen las conclusiones de este capítulo.

6.2 Descripción del proyecto desarrollado

Con el objetivo de comprobar la eficiencia y usabilidad del motor de producción desarrollado en el capítulo 4 se ha llevado a cabo la producción de un videojuego 3D casual online multijugador que permita valorar dichos parámetros.

A lo largo de la siguiente sección se va a realizar un breve resumen del videojuego desarrollado quedando el tratamiento en detalle de la programación del juego fuera del alcance de esta tesina.

El juego que se ha desarrollado empleando como base la tecnología desarrollada en el motor de producción consiste en un juego de billar 3D online multijugador. Este

juego dispone de las modalidades de juego “bola 8” y “bola 9” y, al igual que el motor de producción, ha sido desarrollado bajo la herramienta de autor Adobe Director y programado en el lenguaje Lingo.

Los motivos por los que se ha decidido desarrollar este juego son: por una lado, la petición expresa de los usuarios y por otro, y más importante, el poder utilizar y aprovechar gran parte de la funcionalidad ofrecida por el motor de producción.

Tras un estudio realizado por la empresa *Exelweiss Ent.*, en el que se preguntaba a los usuarios por sus preferencias a la hora de incorporar nuevos juegos al portal *www.mundijuegos.com*, el juego del *Billar* obtuvo un 27.2% de los votos, quedando en segundo lugar tras el *Poker Texas Hold'em* que obtuvo un 32.9%.

Por otro lado, este tipo de juego permite explotar gran parte del potencial del motor de producción, ya que es necesario hacer uso de la mayor parte de la funcionalidad que dicho motor incorpora.

Por un lado, este juego emplea el modelo de aplicación XML para configurar el estado inicial del juego, las propiedades físicas de los objetos, los componentes necesarios para crear el interfaz, las reglas del juego, etc.

La posibilidad de definir diferentes scripts para cada una de las modalidades del billar desarrolladas es de gran utilidad a la hora de definir distintos scripts de reglas para cada modalidad. Poder definir distintos scripts de reglas ha permitido desarrollar dichos scripts de manera modular, evitando así duplicar el código común de los mismos. Por un lado se ha desarrollado una librería que incorpora las funciones comunes a ambas reglas de modo que, independientemente de la modalidad a la que se juegue, los scripts correspondientes a la librería son cargados. Este script incorpora, por tanto toda la funcionalidad común, permitiendo por ejemplo: conocer las bolas que un jugador puede tocar, conocer que bolas ha de meter cada jugador, saber que bolas continúan en la mesa, cuales se han metido y cuales han salido fuera, etc. Por otro lado, para cada modalidad de juego se ha desarrollado un script que sigue la estructura compuesta por sentencias condicionales típica de un script de reglas. Cada uno de estos scripts es el que define realmente las reglas de cada una de las modalidades.

Por otro lado, la ejecución de este juego precisa una simulación física que permita reproducir fielmente el movimiento de las bolas tal y como ocurre en la realidad, así como un sistema de detección de colisiones que permita saber con qué bolas contacta la bola blanca, los rebotes en los cojines o que bolas han entrado en los agujeros. Poder definir las propiedades físicas del mundo, de las bolas y del resto de

cuerpos rígidos de manera externa a través del modelo de aplicación ha permitido probar el resultado de los ajustes de los parámetros físicos de una manera más eficiente. Conseguir que las bolas se comportasen de una manera similar a las reales ha supuesto tener que modificar los parámetros dinámicos de los objetos en multitud de ocasiones. El acceso rápido y estructurado a todas y cada una de las propiedades físicas de los distintos cuerpos rígidos de la escena ha permitido acelerar el proceso que se ha llevado a cabo para ajustar el comportamiento físico a la realidad. Para llevar a cabo un ajuste de los valores de las propiedades físicas que permita al juego comportarse fiel a la realidad el primer paso realizado ha sido el de investigar cuáles son estos valores en un billar de verdad. Un estudio a fondo de las propiedades dinámicas de un billar se hace en [62]. Inicializar las propiedades dinámicas a los valores reales y posteriormente ajustarlas a mano, permite definir el comportamiento físico de una manera más eficiente que intentando ajustar todos los valores desde cero. Además de esto, la librería de colisiones que incorpora el motor de producción ha permitido acceder al estado de las mismas de una manera fácil y rápida. Las funciones genéricas existentes en esta librería han sido utilizadas un número elevado de veces durante la creación de este videojuego, sobre todo a la hora de decidir el resultado de las reglas tras una tirada. Conocer con qué bola ha impactado en primer lugar la bola blanca, saber si alguna bola ha tocado el suelo porque ha salido de la mesa o conocer que bolas han colisionado con el hoyo al haberse colado son ejemplos de la gran variedad de consultas que se atienden mediante el acceso a dicha librería. Por último, el hecho de que el motor de producción garantice el determinismo de la simulación física ha permitido reducir la complejidad en la producción del juego. Gracias a que la simulación física se sincroniza para cada tiro basta con que el jugador que tira mande al resto de jugadores la dirección del tiro y la fuerza para que en el resto clientes la simulación de cómo resultado el mismo estado del mundo. Esta reducción en la complejidad está directamente relacionada con un importante ahorro en el envío de mensajes a través del servidor.

Por otro lado, debido al carácter multijugador del juego, es preciso el envío mensajes entre el servidor y los jugadores para poder comunicar el estado de la partida. El hecho de que el motor de producción configure automáticamente la aplicación MUS evita tener que realizar esta tarea para este juego en concreto. Además se han empleado ciertas funcionalidades que permiten evitar las trampas en el juego, en concreto no se permite la modificación de la variable que almacena la fuerza del tiro a través del acceso su dirección de memoria y tampoco se permite bloquear la conexión con el servidor a través de un firewall.

Además de esto, también se ha hecho uso de algunas de las funcionalidades adicionales que incorpora el motor de producción. En primer lugar, se ha utilizado el

tratamiento de objetos aleatorios que incorpora el motor para definir el estado inicial de las bolas sobre la mesa. Añadir cierta aleatoriedad aquí permite que los usuarios expertos no conozcan de antemano un tiro inicial ganador. En segundo lugar, se ha hecho uso del parser de scripting de Max. Con ello se ha conseguido que el grafista pueda definir a su antojo, y sin la colaboración del programador, las multitexturas de las bolas, que permiten brillos y sombras, los brillos de la mesa y así otros muchos parámetros

Finalmente, cabe decir que programar el juego "Billares" teniendo ya una base de programación que permite centrarse en la programación propia de los scripts de 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.

6.3 Descripción de los demás proyectos

Además del juego visto, los otros dos videojuegos que se van a ver en este capítulo son: "Minigolf" y "Superbuteo".

El segundo juego, "Minigolf", es un juego para el cual también se ha hecho uso del motor de producción visto en el capítulo 4 y que se ha desarrollado paralelamente al juego "Billares". Este juego consiste principalmente en una versión arcade del famoso deporte del minigolf. El objetivo principal del juego en meter la bola en el hoyo realizando el mínimo número de golpes y de tiempo. La característica más llamativa de este juego es que, a diferencia de otros juegos similares, no se juega por turnos. Todos los jugadores juegan a la vez en una carrera por llegar primero al hoyo.

El tercer videojuego, “Superbuteo”, para el cual no se hizo uso del motor de producción, posee unas características muy similares a los otros dos juegos. Este juego está basado en el conocido juego de mesa “Subbuteo”, el cual consiste en disputar un partido de fútbol mediante una mecánica de juego muy similar a la del popular juego de “Las chapas”. Este juego había sido previamente desarrollado por la empresa Exelweiss Ent.

Las características principales que tienen en común estos tres juegos, y que son la base de su desarrollo son: por un lado, su carácter online multijugador, por otro, que son juegos 3D y, por último, que precisan de la realización de una simulación física haciendo uso de un motor de física.

6.4 Estudio de los costes de producción

Con el objetivo de estudiar el aporte real del motor de producción al proceso de desarrollo de un videojuego se ha realizado una comparativa en base a los costes de producción derivados del desarrollo de cada uno de los tres juegos expuestos.

En los tres casos el equipo de desarrollo estaba formado por un diseñador-grafista y un programador junior, este último dedicado la jornada completa al desarrollo del juego.

Para el desarrollo del juego “Superbuteo” se siguió un modelo de producción had-hoc, visto en el capítulo 3, mientras que para el desarrollo de los juegos “Billares” y “Minigolf” se ha hecho uso del motor de producción desarrollado.

Tal y como era de esperar, el resultado visual y la funcionalidad de los tres videojuegos son muy similares. Desde el punto de vista del rendimiento de la aplicación, medida en frames por segundo (FPS), los resultados observados en cada uno de los juegos no muestra diferencias suficientemente relevantes. En concreto, los tres juegos oscilan entre una tasa de 23 y 30 FPS. Esta oscilación depende, entre otros factores, del número de modelos que han de ser dibujados en pantalla, los polígonos de los mismos, si se está realizando la simulación física o no, etc. La Figura 7 muestra una captura de cada uno de los juegos en ejecución.

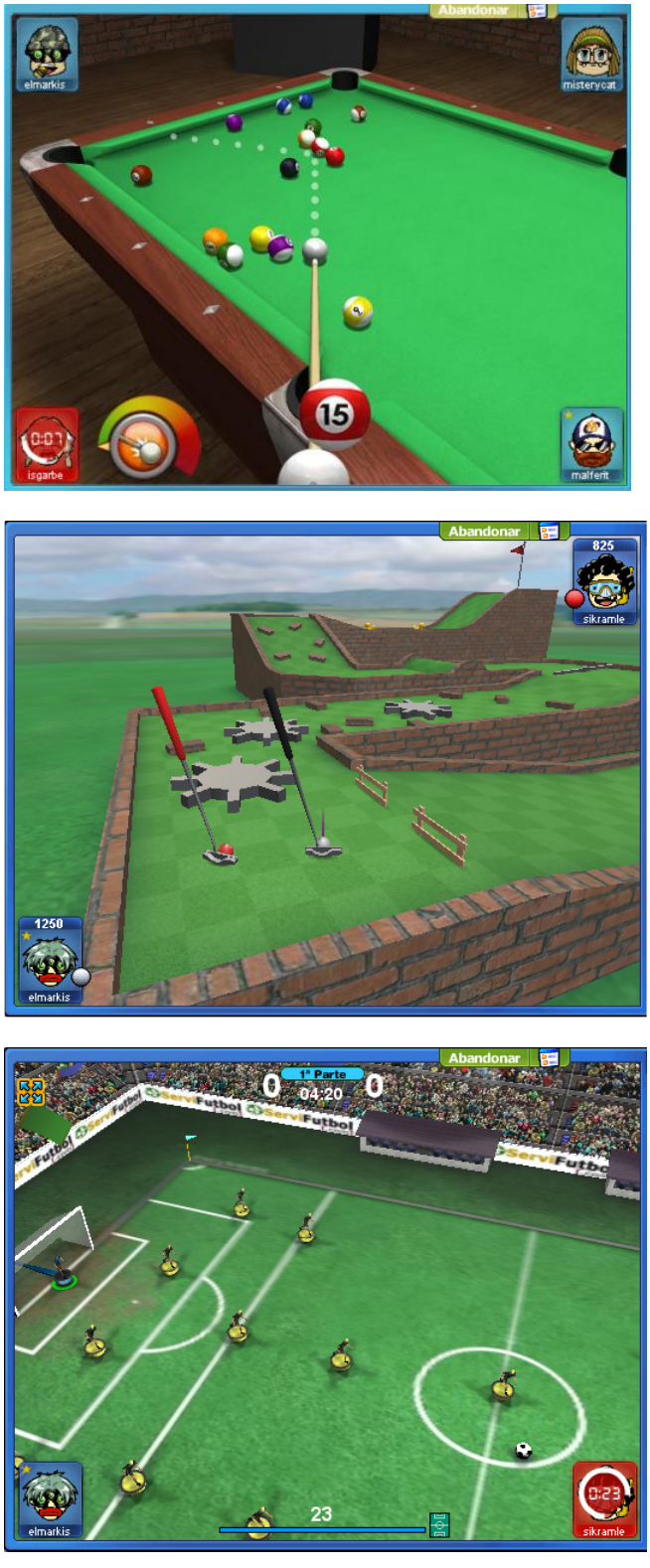

Figura 13. Videojuegos en ejecución.

La Tabla 2 muestra los resultados obtenidos tras las mediciones de los costes de producción de cada juego. Estas medidas se han realizado en base a las líneas de código, el tiempo de desarrollo y el coste económico de cada proyecto. La última de estas tres mediciones, el coste económico, se ha realizado en base al coste mensual que para la empresa Exelweiss Ent. supone el desarrollo de un videojuego de estas características.

Por un lado, la tabla muestra el número total de líneas de los diferentes proyectos (LT). Dentro de este total, se ha hecho una nueva medición que se corresponde con las líneas que el programador ha introducido de manera específica para cada uno de los proyectos. Este último parámetro se corresponde con LJ.

Por otro lado, la tabla muestra el coste temporal de producción (medido en semanas) de cada proyecto. El coste total de producción se corresponde con el parámetro T, el cual ha sido dividido en dos periodos correspondientes con la fase de producción (TP) y la fase de test (TT).

El último parámetro (CE) se corresponde con el coste económico que ha supuesto a la empresa Exelweiss Ent. el desarrollo de cada uno de los proyectos. Este valor ha sido calculado en base al coste T y el coste que supone mensualmente para la empresa el desarrollo de un juego de estas características con el equipo de desarrollo antes expuesto. Debido a que dentro de la fase de producción, esta tesina se centra en la codificación de la aplicación, tan solo se ha tenido en cuenta el trabajo realizado por el programador.

Parámetro	Superbuteo	Billar	Minigolf
LT	11640	16930	15040
LJ	11640	5550	4660
TP	40 (semanas)	21 (semanas)	19 (semanas)
TT	2 (semanas)	1 (semanas)	1 (semanas)
T	42 (semanas)	22 (semanas)	20 (semanas)
CE	84.000 €	44.000 €	40.000 €

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

Los resultados vistos en la Tabla 1 parecen indicar que el uso del motor de producción implica la necesidad de codificar un mayor número de líneas de código y que, por tanto, el modelo de producción ad-hoc permite proyectos de tamaños más

reducidos. No obstante, el parámetro al que realmente hay que prestar atención es al que mide el número de líneas reales que el programador ha tenido que introducir para cada proyecto (LJ). Es fácil observar el importante decremento de este valor cuando el motor de producción es utilizado.

El hecho de que el número de líneas totales sea mayor en aquellos proyectos en los que se ha hecho uso del motor de producción es debido a que la programación genérica del mismo requiere una codificación más extensa. Aunque en la tabla no aparezcan estos datos, es interesante indicar que el número de líneas totales del motor de producción es de 11380, mientras que el coste de producción del mismo ha supuesto el trabajo, a jornada completa, de dos programadores junior durante 7 meses. Como se puede observar, la magnitud del motor es prácticamente la misma que la del juego "Superbuteo" completo. Además, el coste temporal de desarrollo del motor de producción ha sido un 25% mayor que el del juego "Superbuteo" completo, a pesar de que el número de líneas de código en ambos casos es muy similar. Esto también es debido al carácter genérico del motor, ya que desarrollar herramientas genéricas siempre es más costoso y se requiere una fase de testeo más extensa.

Mientras que el modelo de producción ad-hoc seguido en el "Superbuteo" supone que todas las líneas de código hayan sido introducidas de manera específica para dicho juego, el uso del motor de producción ha supuesto una reducción del 52.32% y del 59.97% del número de líneas introducidas realmente por el programador en los juegos "Billares" y "Minigolf", respectivamente.

Aunque estas cifras hacen pensar que el uso del motor de producción supone un gran beneficio, es importante estudiar el coste temporal de cada proyecto. Aunque el número de líneas reales introducidas por el programador sea menor en los casos en los que se hace uso del motor de producción, hay que plantear la posibilidad de que esto no se vea reflejado en el coste temporal, debido a una mala usabilidad por parte del motor.

No obstante, en el caso del motor de producción desarrollado la relación entre la reducción en el número de líneas realmente codificadas y el coste temporal del desarrollo del proyecto es directamente proporcional. En el caso del juego "Billares", el coste temporal de producción ha sido un 47.62% menor que en el caso del "Superbuteo", mientras que en el juego "Minigolf" esta reducción ha sido del 52.38%. De estos datos se puede concluir que el coste de producción de aquellos juegos que han hecho uso del motor de producción ha sido reducido a la mitad.

Tal y como se ha visto en el capítulo 3, esta importante reducción en el coste temporal del desarrollo de un juego supone para la empresa la oportunidad de aceptar desarrollos que anteriormente debería haber rechazado debido a las restricciones temporales, así como la posibilidad de sacar al mercado un juego determinado antes que la competencia, lo que conlleva las ventajas y beneficios ya vistos.

El último parámetro, el coste económico, permite valorar si la inversión económica por parte de la empresa en el desarrollo del motor de producción realmente resultará productiva. Tal y como se puede observar en la tabla, el coste medio de producción de cada juego nuevo ha sido reducido aproximadamente en un 50% gracias al uso del motor de producción. Este porcentaje supone un ahorro medio de 42.000€ por cada juego desarrollado, una cantidad que permitirá, en un corto periodo de tiempo, recuperar la inversión económica desembolsada en el desarrollo del motor de producción.

6.5 Conclusiones

En el presente capítulo se han estudiado los costes de producción de los videojuegos "Superbuteo", "Billares" y "Minigolf". El primero de estos tres siguió un modelo de producción ad-hoc, mientras que en los otros dos, se hizo uso del motor de producción desarrollado en el capítulo 4.

Aunque el número de líneas totales de los videojuegos "Billares" y "Minigolf" son mayores que en el caso del videojuego "Superbuteo", el número real de líneas que el programador ha tenido que codificar para el desarrollo de los dos primeros ha sido reducido en más de un 50%.

Además, esta reducción, se corresponde de manera directa con el ahorro temporal en el coste de producción de los videojuegos "Billares" y "Minigolf", ya que éstos han podido ser desarrollados en la mitad de tiempo que el juego "Superbuteo".

Esta mejora en los costes de producción supone un ahora medio de 42.000€ por cada nuevo juego desarrollado, cantidad que permitirá, en un periodo corto de tiempo, recuperar el dinero invertido por parte de la empresa en el desarrollo del motor de producción y comenzar a beneficiarse del desarrollo del mismo.

Por todo esto se concluye que el motor de producción desarrollado es una herramienta que permite la creación de videojuegos casuales 3D online multijugador bajo la herramienta Director a un bajo coste de producción.

Capítulo 7

Conclusiones y líneas de trabajo futuro

7.1 Conclusiones generales

En la sección 1.2 se plantearon los objetivos de esta tesina: realizar un estudio del proceso de producción de un videojuego casual online multijugador y de las herramientas utilizadas, identificar qué aspectos de éste son mejorables y plantear las posibles mejoras, desarrollar una herramienta capaz de solventar la problemática existente y desarrollar un nuevo juego que permita valorar la efectividad de la misma y, finalmente, cuantificar la aportación de la herramienta desarrollada mediante un análisis de los costes de producción de tres videojuegos. En esta sección se revisarán las aportaciones de esta tesina en función de dichos objetivos.

Estudio del proceso de producción de videojuegos casuales online multijugador

A lo largo del capítulo 4 se ha descrito el proceso de producción ad-hoc que los estudios de desarrollo emplean a la hora de desarrollar sus videojuegos. El hecho de que los juegos casuales online multijugador sean producciones mucho menos costosas que las grandes producciones de videojuegos lleva a las empresas a crear sus proyectos desde cero para cada nuevo desarrollo.

Esto ha permitido concluir en la problemática existente en este tipo de modelo de producción: los programadores realizan las mismas tareas una y otra vez para cada nuevo juego creado. Realizar estas tareas repetidamente impide que los costes de desarrollo se reduzcan a medida que se desarrollan más juegos.

Estudio de las herramientas usadas en el proceso de producción de videojuegos casuales online multijugador.

En el capítulo 2 se ha realizado un estudio de las distintas tecnologías sobre las que este tipo de juegos pueden ser desarrollados. Con ello se ha podido, en el capítulo 3, realizar un estudio crítico de las mismas con el objetivo de seleccionar la herramienta más completa. La conclusión a la que se ha llegado tras este estudio es que la herramienta Adobe Director, debido a su soporte 3D, su ejecución directa sobre un navegador y su alto porcentaje de penetración, es una herramienta ideal para este tipo de desarrollos.

Identificación de los aspectos mejorables en el proceso de producción de videojuegos casuales online multijugador.

Tras la elección de Director como herramienta de autor de desarrollo, en el capítulo 4 se han detallado aquellos aspectos que podrían ser mejorables dentro del proceso de producción. En concreto, se ha visto: como la integración ad-hoc de las distintas librerías y motores utilizados conlleva el estudio reiterativo de los mismos por parte de los programadores; como las posibles inconsistencias o errores que pueden aparecer en dichos motores no son resueltos de una manera genérica, lo que conlleva que para cada proyecto nuevo el programador debe enfrentarse a ellos; y como la inexistencia de un motor que extraiga la funcionalidad común en los distintos videojuegos desde el punto de vista de la estructura interna de los mismos y de su fase e inicialización conlleva la programación reiterativa de ciertas funcionalidades.

Planteamiento de las posibles mejoras.

Con el fin de evitar la problemática vista y mejorar los aspectos mejorables identificados, al final del capítulo 4 se ha planteado la creación del motor de producción cuya filosofía de diseño es la de permitir la configuración externa de los contenidos del videojuego, realizar las funciones de un motor de inicialización y actuar como *Middleware* que encapsule en una capa de un nivel de abstracción mayor los motores integrados en Director.

Desarrollo una herramienta capaz de solventar la problemática existente.

El desarrollo del motor de producción propuesto en el capítulo 4 ha sido totalmente explicado a lo largo del capítulo 5. La estrategia seguida para explicar el funcionamiento del motor ha sido la de detallar como éste consigue llevar a cabo cada uno de los principios vistos en la filosofía de diseño.

El primer principio, "Configuración externa", ha sido desarrollado íntegramente en el trabajo de colaboración visto en [1] y se ha llevado a cabo mediante el uso de un fichero XML y la creación de un parser que ha permitido transformar este modelo de aplicación en unas estructuras de datos comprensibles por la aplicación.

El segundo, "Motor de inicialización", desarrollado en colaboración con [1], se ha llevado a cabo extrayendo la funcionalidad común existente desde el punto de vista de la estructura interna de un juego y automatizando, en la medida de lo posible, los distintos procesos que se llevan a cabo durante la inicialización del videojuego.

El último principio, "Middleware", se ha llevado a cabo mediante el desarrollo de una capa de abstracción que encapsula los motores existentes en la herramienta Adobe Director, así como la herramienta MUS, previamente desarrollada en la empresa Exelweiss Ent. Esta capa permite al programador usarla como medio a través del cual acceder a la funcionalidad de estos motores de una manera más fácil y rápida, permitiendo además cierta versatilidad a la hora de cambiar de motor.

Además de las funcionalidades que implementa el motor de producción en base a sus principios, al final del capítulo 5 se ha detallado como el motor implementa cierta funcionalidad adicional que es de gran utilidad. Esta funcionalidad adicional se resume en: automatización en el uso de proxys, posibilidad de realizar scripting desde la herramienta 3D Studio Max y tratamiento de objetos aleatorios durante la fase de inicialización para sincronizar su estado inicial en todos los clientes en los que se ejecute la aplicación. Paralelamente, en [1] se detallan otras funcionalidades adicionales como la creación de escenarios abiertos de manera automática mediante la técnica skydome y la automatización en la creación de modelos siempre visibles mediante la técnica de clonación de modelos

Desarrollo de un videojuego que permita valorar la efectividad de la herramienta desarrollada.

Con el objetivo de comprobar la efectividad del motor de producción, a lo largo del capítulo 6 se ha explicado como se ha llevado a cabo el desarrollo de un videojuego que ha empleado dicho motor como base de producción. Durante este capítulo no se ha entrado en detalle en cómo se ha llevado a cabo la programación de los scripts propios del juego, sino que se ha hecho un mayor hincapié en aquellos procesos que han podido ser minimizados, acelerados e incluso evitados gracias al uso del motor de producción.

Cuantificación de la aportación de la herramienta desarrollada mediante un análisis de los costes de producción de tres videojuegos.

Aunque tras la lectura del capítulo 5 ya era posible apreciar la aportación del motor de producción al desarrollo del videojuego, en el capítulo 6 se ha realizado un experimento que ha permitido cuantificar dicha aportación en base a tres criterios: líneas de código que el programador ha tenido que codificar, coste temporal de producción y coste económico de desarrollo. Frente a las 42 semanas que se tardó en producir el videojuego "Superbuteo", los videojuegos "Billares" y "Minigolf" fueron desarrollados en 22 y 20 semanas respectivamente. Esto permite concluir que el coste temporal de futuros videojuegos, de características similares a los expuestos, será, aproximadamente, un 50% inferior si se hace uso del motor de producción. Por otro lado, de manera directamente proporcional a esta reducción en el coste temporal, el coste económico de los juegos "Billares" y "Minigolf" ha sido aproximadamente un 50% menor que el del juego "Superbuteo". Por todo esto, se puede llegar a la conclusión final de que, aunque el tamaño final del proyecto sea mayor cuando se hace uso del motor de producción, la labor que realmente realiza el programador se ve beneficiosamente disminuida y los costes de producción de los videojuegos se reducen aproximadamente un 50%.

7.2 Líneas de trabajo futuro

A partir del trabajo realizado en esta tesina se han dejado abiertas algunas líneas de trabajo que pueden realizarse en un futuro.

Migración a otras plataformas y herramientas de desarrollo

Tal y como se expuso en el punto 2, son muchas las plataformas para las que se desarrollan videojuegos. Esta tesina se ha centrado en los juegos casuales online multijugador, pero debido a la estructura propia de los juegos, así como al proceso de desarrollo de los mismos, los problemas expuestos en este documento son fácilmente trasladables a otras plataformas existentes. En concreto, realizar un nuevo motor de producción ejecutado bajo dispositivos móviles puede ser un trabajo futuro interesante.

Siguiendo la misma lógica que en las plataformas, otra línea de trabajo posible es realizar el motor de producción para otras herramientas de desarrollo, como puede ser Flash.

Ampliación de las capacidades de middleware que ofrece el motor de producción actual

El motor de producción desarrollado, además de permitir la configuración externa de ciertos parámetros e incluir un motor de inicialización propio, proporcionaba la posibilidad de actuar como un middleware, integrando varios motores en uno solo. Como se ha mostrado, no solo se pueden integrar motores comerciales, sino que es posible desarrollar motores propios en función de las necesidades propias de los juegos que se desean desarrollar e integrarlos todos en el propio motor de producción. Es por ello que se propone como línea de trabajo futuro desarrollar nuevos motores para facilitar el desarrollo de otro tipo de juegos, como pueden ser, juegos de cartas, de mesa o juegos en dos dimensiones, e integrar dichos motores en el motor de producción.

Bibliografía

[1] Garrido Benet, I. *Motor de configuración externa e inicialización para la mejora de la eficiencia en el desarrollo de juegos casuales online multijugador.* Tesina final de master IARFID. UPV. Febrero 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.com

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

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

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

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

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

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

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

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

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

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

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

[22] 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.

[23] 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.

[24] Página web de Open Scene Graph (OSG): http://www.openscenegraph.org/

[25] Página web de Open Dynamics Engine (ODE): http://www.ode.org/

[26] Página web de Open Audio Library (OpenA)l: http://www.openal.org/

[27] Página web de Character Animation Library 3D (Cal3D): http://gna.org/projects/cal3d/

[28] Página web de Phyton Programming Language: http://www.python.org/

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

[30] 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.

[31] 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.

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

[33] Manual *Havok Physics Primer Version 1.0*. Página web de Director-online*:* http://www.director-online.com/havok/

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

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

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

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

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

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

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

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

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

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

[44] 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. (xtra red, server y multiuser)

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

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

[47] Phil Gross and Mike Gross. *Macromedia Director 8.5 Shockwave Studio for 3D: Training from the source*. Macromedia Inc., 2002. (director en general, + 3d, relación padre-hijo, funciones vectores, luces, shaders y texturas, texto 3d, clonación, animación, mayas y optimización)

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

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

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

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

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

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

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

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

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

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

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

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

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

[61] 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/

[62] Inhwan Han. *Dynamics in Carom and Three Cushion Billiards.* Departament of Mechano-Informatics & Design Engineering. Hongik University.