PROYECTO FIN DE CARRERA

INGENIERÍA INFORMÁTICA

DISEÑO E IMPLEMENTACIÓN DE UN JUEGO PARA PC MEDIANTE OGRE 3D

Autor: Mario Velázquez Muñoz

Tutor: Juan Peralta Donate

Leganés, Marzo de 2011

Título: Juego 3D con Ogre

Autor: Mario Velázquez Muñoz

Director: Juan Peralta Donante

EL TRIBUNAL

Presidente: _____

Vocal: _____

Secretario: _____

Realizado el acto de defensa y lectura del proyecto fin de carrera el día ___ de _____ de 2011 en Leganés, en la Escuela Politécnica Superior de la Universidad Carlos III de Madrid, acuerda otorgarle la CALIFICACIÓN de

VOCAL

SECRETARIO PRESIDENTE

Agradecimientos

- A mi familia y amigos. Por mantenerme en estos meses de despreocupación y estrés y por apoyarme y animarme en los momentos más difíciles.
- A mi tutor del proyecto, D. Juan Peralta Donate. Por apoyarme en todo momento en la realización del mismo, ayudarme con dudas y estar siempre disponible para darme consejos.
- A toda la comunidad de OGRE y Blender que crean tutoriales a disposición de todo el mundo y ayudan a través de los foros ante cualquier duda planteada.

Resumen

La industria de los videojuegos es hoy en día la actividad más rentable en el sector del ocio en España, y se prevé que aumente considerablemente en los próximos años. El sector se está diversificando en modos y plataformas muy variados, además de la construcción de máquinas cada vez más potentes que hacen que puedan desarrollarse cada vez mejores y complejos juegos.

Este proyecto tiene como objetivo el desarrollo de un juego 3D en tiempo real utilizando la plataforma libre y de código abierto OGRE. Consiste en carreras de descenso en monopatín o longboard en 3D.

El proyecto a su vez es un ejemplo de juego en el entorno de desarrollo con OGRE, utilizando gran cantidad de librerías, para intentar mostrar la mayor parte de las propiedades del entorno y que pueda servir de utilidad para futuros proyectos.

Palabras clave: OGRE, 3D, tiempo real, videojuego, carreras, longboard.

Abstract

Nowadays video game industry is the most profitable activity in the leisure sector in Spain and is expected to increase significantly in coming years. The area is diversifying in many different game modes and platforms, plus the construction of ever more powerful machines that can create better and more complex games.

The aim of this project is to Developer a real-time 3D game using the free platform and open source OGRE. It consists of a 3D downhill Racing in skateboard or longboard.

The project is at the same time an example of game development environment with OGRE, using lots of libraries. This way, we show most of the environment properties that can be useful for future projects.

Keywords: OGRE, 3D, real time, video game, racing, longboard.

Índice General

Índice de Figuras

Índice de Tablas

Índice de Código

Capítulo 1:

Introducción y Objetivos

Este capítulo recoge información introductoria sobre el proyecto, como el por qué surge la idea de la creación del juego y los objetivos principales marcados para el desarrollo del mismo.

También se habla sobre las fases de desarrollo del proyecto y los medios tanto software como hardware empleados.

Al final se comenta un pequeño resumen sobre los contenidos de la memoria.

1.1 Introducción

La industria de los videojuegos es un sector económico de ocio involucrado en el diseño, desarrollo, distribución y venta de videojuegos y del hardware asociado, que engloba decenas de disciplinas de trabajo y emplea a miles de personas. Esta industria, en los últimos años ha experimentando un gran crecimiento económico, siendo el sector más importante de la industria del ocio, superando al cine y la música juntos [35].

En España, la situación es similar a la del resto del mundo, y el consumo, pese a una caída en el sector en torno a un 5,2% este último año 2010, mantiene a España como el cuarto en el mercado Europeo y se espera que recupere y crezca de nuevo a un ritmo alto, que llegue incluso a duplicarse en unos tres años.

Para ver unos datos más concretos [33], se calcula que en 2010 se vendieron en Europa unos 231 millones de juegos y consolas, aportando casi 11.000 millones de Euros y concretamente en España, cuarto país en consumo de la Unión Europea, se alcanzaron los 1.200 millones de Euros en ingresos. Mundialmente la industria del videojuego alcanzó cifras en torno a los 57.600 millones de Euros, contando tanto software como hardware.

Actualmente la situación de crisis ha provocado una caída general en todo el sector en torno a un 6%, en el caso de España, esta caída es menor, en torno al 5,2%, una buena noticia, si se tiene en cuenta que la economía Española es una de la más afectadas por la crisis y que es el principal foco de descargas ilegales a nivel mundial.

Al contrario de lo que parece, es muy probable que el sector comience a recuperarse, ya que cada vez son mayores los sistemas de comercialización, que hacen más sencillo tanto la creación como los medios de difusión, incitando a los usuarios a comprar videojuegos debido a las buenas relaciones calidad/precio, gran variedad de tipos de videojuegos para todo tipo de plataformas y facilidades de adquisición.

Otros sectores muy importantes y en crecimiento actualmente son los videojuegos para móviles y los videojuegos en Internet, que demandan todo tipo de juegos, ya sean meros puzles hasta complicados juegos 3D de acción.

Debido a todos estos datos interesantes para la industria del videojuego, entra en portada este proyecto, ya que se trata de un videojuego con grandes capacidades y que puede ser el inicio para cualquier persona que quiera iniciarse en el mundo del desarrollo de videojuegos.

Además el proyecto se ha realizado con elementos gratuitos y abiertos para poder ser la base de un proyecto comercial sin costos adicionales por uso de software con licencias comerciales. El elemento principal de este proyecto es OGRE [1], un paquete de diseño de software en tiempo real en 3D y Blender [8], un programa de creación de objetos 3D, además de ciertas librerías adicionales para ambos sistemas también con licencias gratuitas y abiertas para la comercialización del futuro software.

Aunque OGRE no es en sí una herramienta de desarrollo de videojuegos, existe a su alrededor una gran comunidad de desarrolladores enfocados al mundo del videojuego y que ayudándose unos a otros y aportando distintas ideas y mecanismos, se consigue que sea una plataforma casi perfecta para el desarrollo de los mismos. Por otra parte en la comunidad,

muchos desarrolladores han creado librerías con elementos específicos para cubrir las carencias de OGRE respecto al desarrollo de juegos, como son librerías físicas, librerías de diseño de escenarios, librerías de diseño de inteligencia artificial, frameworks estructurados para el desarrollo de juegos y otras muchas más que hacen un poco más fácil el desarrollo de software 3D.

El videojuego que se quiere llevar a cabo haciendo uso de OGRE, consistirá en un juego basado en el deporte extremo con gran crecimiento en los últimos años y que consiste en descender una carretera, normalmente de montaña, subido encima de un monopatín. Este deporte es el Downhill skate. En el downhill se utilizan monopatines adaptados para el deporte, algo más grandes que los patines de calle, con ejes de gran angulación para giros cerrados y ruedas blandas para un mayor agarre con el asfalto de las carreteras. Por otra parte los pilotos deben de llevar monos de protección y casco ya que se pueden llegar a alcanzar velocidades de hasta 90 km/h con estos patines sin motor ni frenos.

Figura 1: Fotografía de prueba de Downhill Skate

Los patinadores conducen los monopatines con técnicas de conducción muy depuradas, para alcanzar las mayores velocidades, tener mejor paso por curva y la mejor frenada, incluso derrapando el monopatín. Todo ello se consigue cambiando pesos encima de la tabla y ayudándose de guantes para apoyar las manos en el suelo y el que el monopatín derrape. Para llegar a practicar descenso en monopatín, se necesita mucha experiencia y control sobre la

tabla, ya que una salida en una carretera de montaña, puede tener graves consecuencias, y cualquier precaución es poca.

La mecánica del juego en si es muy similar al típico juego de carreras de coches, sólo que el acelerador es la propia pendiente de la carretera, por lo que es apropiado para el desarrollo de un juego simple 3D como muestra y además es algo innovador, ya que hasta la fecha sólo hay algunos juegos comerciales que hacen un acercamiento al downhill skate, sin llegar a ser este cercano al deporte como se intenta en el proyecto.

1.2 Objetivos

Este proyecto fin de carrera busca varios objetivos principales, de los cuales se pueden destacar los que se enumeran a continuación.

El primer y principal objetivo del proyecto es conseguir crear un pequeño juego utilizando el entorno de OGRE y conseguir demostrar que esta plataforma sirve para el desarrollo de juegos 3D.

El segundo objetivo del proyecto es ayudarse de herramientas libres como Blender para generar los objetos 3D y librerías libres de desarrolladores para acoplar con OGRE y poder así utilizar las propiedades adicionales que estas proporcionan al sistema y mostrar el manejo de las mismas.

El tercer objetivo de este proyecto es servir como un punto de partida para futuros alumnos que quieran desarrollar juegos con este entorno de desarrollo, así como para el propio creador del proyecto.

Como último objetivo del proyecto, se pretende crear una documentación completa y comprensible del proyecto, que pueda utilizarse en futuros proyectos y sirva de ayuda para otros alumnos que comiencen con la herramienta.

1.3 Fases del Desarrollo

Para llevar a cabo los objetivos marcados en el apartado anterior, es necesario dividir el proyecto en fases y seguirlas. Estas fases marcan el ciclo de vida que sigue el proyecto para desarrollarse, desde las fases de documentación previas, hasta las fases de pruebas.

- **Documentación previa sobre OGRE, librerías e implementación de viabilidad**.

Antes de empezar el proyecto, se ha realizado un trabajo dirigido sobre el proyecto que consta de tres documentos: el manual de usuario de OGRE [9], traducido al Español, traducción de algunos de los tutoriales básicos [10] de la Wiki de OGRE y un pequeño manual que explica cómo gestionar un entorno de desarrollo de OGRE [11] con Microsoft Visual Studio C++. Además se creó un pequeño proyecto de OGRE que utilizaba algunas de las librerías que en el futuro se usarán para el proyecto aquí comentado y que sirvió para comprobar la utilidad tanto de OGRE como de las librerías para el desarrollo del juego.

- **Análisis de requisitos.**

Se definen los requisitos para determinar opciones básicas que incluye el juego, como el tipo de controles, las opciones disponibles para un jugador y distintas propiedades generales del juego.

- **Diseño del juego.**

Debe hacerse un pequeño diseño del juego, tal como bocetos de pantallas, modalidades de juego o los componentes que forman la estructura del desarrollo. Que sirven de guía para la implementación del mismo.

- **Implementación del Juego.**

Una vez estudiados todos los elementos del juego, se procede a la implementación. Todo el proceso de implementación se realiza guiándose por el estudio anterior y siguiendo un ciclo de vida funcional, en el que el programa funciona desde las primeras fases de la implementación y se va aumentando su comportamiento, en ciclo incremental.

1.4 Medios Empleados

La realización de este proyecto precisa tanto de elementos software como componentes hardware que se detallan a continuación:

Elementos Hardware:

- Se precisa de un ordenador potente y con una gran tarjeta gráfica. En el caso de este proyecto en concreto se utiliza un iMAC de 27" con 4 GB de RAM y gráfica ATI 4800 de 512 MB.

Elementos Software:

Este proyecto tiene una característica especial y es que casi todos los elementos software utilizados son elementos gratuitos y libres para el desarrollo de aplicaciones, ya sean comerciales o no.

- Windows Visual C++ Studio: Será la herramienta principal del desarrollo del juego. Esta herramienta puede descargarse y actualizarse de forma gratuita desde la página Web de Microsoft.
- Microsoft Windows XP Home Edition: Para este conocido sistema operativo, se encuentran todo tipo de aplicaciones para el desarrollo. Además es un sistema muy estable.
- OGRE 3D y librerías de terceros: En realidad no son software en sí, sino que son paquetes de clases para el desarrollo de aplicaciones. Todas son libres para la creación y comercialización de aplicaciones, incluso hay algunas que son de código abierto.
- Blender: Programa para el diseño de componentes 3D. Con este programa gratuito se pueden hacer modelos 3d, texturizarlos, animarlos y exportarlos al formato aceptado por OGRE.
- Office 2008: Software para la creación de la documentación del proyecto. Como alternativa gratuita se podría haber utilizado OpenOffice.
- Photoshop: Programa de dibujo, utilizado para crear texturas, normales y manipulación de imágenes del proyecto. Como alternativa gratuita se podría haber utilizado Gimp.

1.5 Estructura de la memoria

El proyecto se ha dividido en una serie de capítulos en los que se trata cada uno de los aspectos involucrados en el diseño y desarrollo del videojuego. Estos capítulos son los que siguen a continuación:

En el capítulo 1, **introducción**, se señalan algunos puntos iniciales del proyecto, de qué trata, sus objetivos y se habla sobre los elementos que llevaron a la elección de este proyecto.

En el segundo capítulo, **estado de la cuestión**, se habla sobre la historia de los videojuegos desde su aparición hasta la actualidad. Además se comentan un poco distintos tipos de herramientas existentes para poder realizar el proyecto.

En el tercer capítulo, **análisis, diseño, planificación e implementación**, recoge la parte más importante del proyecto, donde se realiza un análisis de la arquitectura y el modelo del juego. Se muestra la planificación y presupuesto del proyecto y se detalla todo el proceso de implementación del juego.

El cuarto capítulo, **conclusiones**, recoge los resultados obtenidos tras finalizar el proyecto y la consecución de los objetivos marcados al inicio del mismo.

El capítulo cinco, **líneas futuras**, comenta una serie de posibles ideas o procesos para el futuro del juego, para que se continúe el desarrollo del mismo en diferentes aspectos.

El sexto capítulo, **glosario de términos**, contiene palabras y términos utilizados en el proyecto cuyo significado puede ser dudoso.

El séptimo capítulo, **referencias**, recoge las referencias utilizadas en el proyecto, y que pueden servir de apoyo a la información de este documento. Las referencias pueden tratarse tanto de documentos como de direcciones web.

Para finalizar, el documento contiene una serie de **anexos** de utilidad que engloban la planificación final del proyecto, manual de usuario y manual de experto.

Capítulo 2:

Estado de la cuestión

Tras haber realizado una pequeña introducción sobre el proyecto y sus objetivos, en este segundo capítulo, se comenta la evolución de los videojuegos a lo largo de la historia, centrado en niveles generales y en relación con juegos de monopatines. También se comentan y comparan varios sistemas para el desarrollo de juegos 3D existentes en la actualidad y distintos modeladores de gráficos 3D.

2.1 Historia de los videojuegos

Antes de la aparición del primer videojuego, existieron muchas y muy variadas máquinas recreativas que se basaban en principios eléctrico-mecánicos [31]. Es difícil establecer cuál fue la primera máquina automatizada destinada a entretener, por ello, se establece como prehistoria de los videojuegos a aquella época en la que a los autómatas se les incorporó algún ingenio para ganar dinero de forma automática, posiblemente en torno a 1894, con máquinas como *Automated Skill Shooter*, *Erie Digger* o *Play The Derby*.

Figura 2: Automated Skill Shooter.

Figura 3: Erie digger.

Todas estas máquinas, suponen un punto de referencia e inicio para los videojuegos y el entretenimiento electrónico.

2.1.1 Inicios de los videojuegos

En el año **1947** *Thomas T. Goldsmith y Estle Ray Mann* patentaron un sistema electrónico de juego que consistía en simular el **lanzamiento de misiles** contra un objetivo. Este "juego" estaba inspirado en las pantallas de los radares utilizados en la Segunda Guerra Mundial. Para su funcionamiento utilizaba válvulas y una pantalla de rayos catódicos. Las únicas funciones que tenía ese "juego" eran ajustar curva y velocidad del disparo. Nada más. No presentaba movimiento en pantalla, los objetivos estaban sobre impresionados, y por lo tanto no se le puede considerar un videojuego.

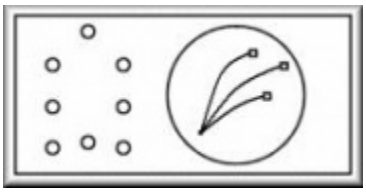

Figura 4: Diagrama de Lanzamiento de misiles.

En **1952**, y como resultado de la tesis doctoral en matemáticas de *Alexander Sandy Douglas*, en la universidad de Cambridge, aparece una versión del tres en raya, también llamado **OXO**. Fue el primero en tener su versión digital y funcionaba en una computadora *EDSAC*. El jugador interactuaba a través de un dial telefónico de rueda. Sin embargo no se puede considerar videojuego (aunque hay opiniones en contra) ya que no presenta movimiento en pantalla, tan sólo imágenes fijas.

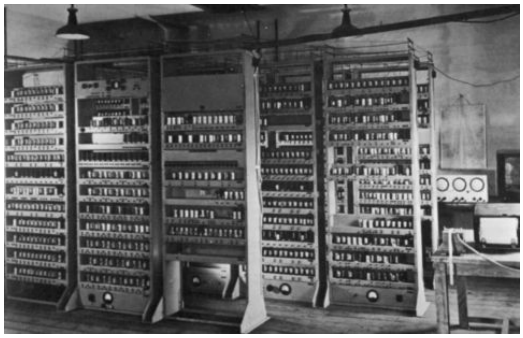

Figura 5: Computadora EDSAC.

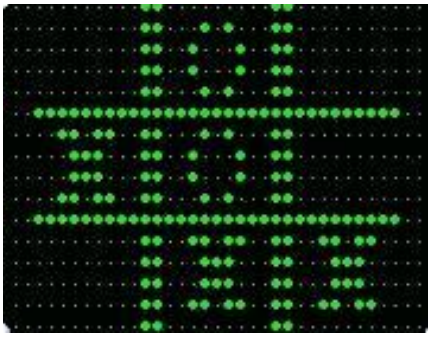

Figura 6: Pantalla de OXO.

En **1958**, aparece **Tennis for two**. Surge de la mente de *William Higinbotham*, físico que trabajaba en los laboratorios Brookhaven National como diseñador de circuitos electrónicos para el Proyecto Manhattan y gran aficionado al Pinball.

Consistía en una línea horizontal que representaba el suelo del campo de Tenis y de una pequeña línea vertical en la mitad del campo que representaba la red. Los jugadores debían elegir el ángulo en que debía salir la bola y golpearla. A diferencia de casi todos sus sucesores no hacía uso de una perspectiva aérea para mostrar el desarrollo del juego. Los dos jugadores que podían participar en cada partida disponían de un controlador compuesto por un pulsador que servía para golpear la pelota virtual y un mando analógico con el que se controlaba la dirección de la bola.

Figura 7: Hihinbotham y el Tennis for two.

Este juego actúa de manera bastante más realista que el conocido **Pong** (publicado 15 años después) y muchos lo consideran el primer videojuego de la historia.

En el verano de **1961**, *Digital Equipment* decide donar uno de sus ordenadores último modelo (PDP-1) al MIT (Instituto Tecnológico de Massachusetts). Ordenador del tamaño de un coche.

Varios miembros (*Wayne Witanen, Martin Graetz y Russell*) empezaron a discutir sobre qué aplicación podrían diseñar para comprobar las posibilidades del nuevo ordenador que tenían en sus manos. Debería ser un juego en el que se pudiera controlar objetos moviéndose por la pantalla. Finalmente decidieron que se trataría de un duelo entre dos naves en el espacio exterior y que se llamaría "**Spacewar**".

Habría dos naves, cada una controlada por un juego de interruptores de la consola. Tendrían un depósito de combustible y algún tipo de arma. Y para según qué situaciones habría un botón para alcanzar el "Hiperespacio".

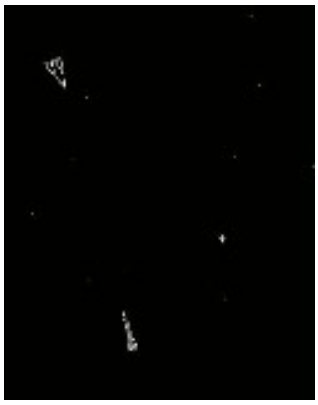

Figura 8: Imagen de Spacewar.

En Abril de **1962** (tras unas 200 horas de trabajo), el juego está completamente listo. Spacewar se puede considerar como el primer juego de ordenador de la historia.

Una década después, en **1971**, apareció el que se puede considerar como primer videojuego comercial de la historia: ***Galaxy Game***. Esto no es una contradicción con respecto a considerar a *Pong* como el primer videojuego, ya que Pong fue comercializado y *Galaxy Game* sólo apareció en recreativas. *Galaxy Game* fue creado por dos estudiantes de la *Universidad de Stamford*, *Bill Pitts* y *Hugh Tuck*, a partir del código original de *Spacewar*. La máquina se instaló dentro del complejo del campus de la universidad y se considera por tanto como la primera máquina de videojuegos en ser instalada en un sitio público. El éxito fue enorme. Hoy día está expuesta en el *Museo de la Computación de Stamford*.

Figura 9: Recreativa Galaxy Game.

Un par de meses después en ese mismo año, **1971**, se produjo el siguiente paso lógico y apareció ***Computer Space***, la primera máquina en ser producida en serie. En realidad el juego no es conocido por la gran mayoría de la gente, pero lo importante no es tanto el juego como su creador: *Nolan Bushnell* quien posteriormente fundó *Atari*. *Bushnell* fue contratado por *Ampex* en **1969** y en sus ratos libres trató de hacer su propia versión de *Spacewar* que no necesitara de un ordenador para mostrarse en pantalla. Esto redujo mucho los costes en componentes para el juego. Tras varios meses de trabajo consiguió terminarlo y se asoció con *Nutting Associates* fabricando alrededor de 1.000 recreativas de *Computer Space* (incluso se llegó a fabricar una versión para dos jugadores). Aunque en un principio funcionó bastante bien, el intento global fue un fracaso, del que *Bushnell* dijo aprender mucho. Según él todo este proceso le dio las directrices básicas para comenzar a gestar su verdadera idea, que más tarde llevaría a cabo fundando *Atari*.

Figura 10: Recreativa Computer Space.

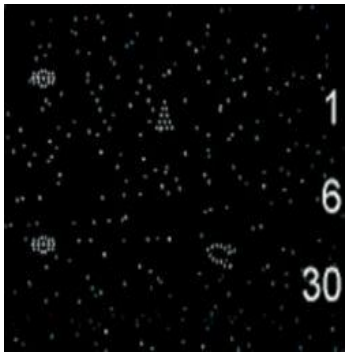

Figura 11: Pantalla de Computer Space.

Aquí termina este punto puesto que los siguientes grandes hitos en la historia de los videojuegos de *Magnavox Odyssey* y *Atari* llegaron de la mano de *Pong* en **1972**. En los puntos siguientes se trata cada una de las décadas desde los 70 hasta nuestros días comentando los hechos más importantes relacionados con la historia de los videojuegos.

2.1.2 Década de los 70

En **1972** y tras muchas dificultades, *Ralph Baer*, consiguió asociarse con *Magnavox*, para crear lo que se considera la primera videoconsola de la historia, la **Magnavox Odyssey**. Era capaz de generar dos puntos por pantalla, una pelota y una línea central. En la caja del juego venían láminas coloreadas y semitransparentes para poner en la televisión, cartas y dados. No tenía sólo un circuito integrado, sino que consistía en 40 transistores y otros tantos diodos, y los cartuchos eran más jumpers que otra cosa. Podía ofrecer 12 juegos diferentes combinando los elementos que traía en la caja y se comercializó únicamente para funcionar en los televisores de *Magnavox*.

Este mismo año, y concretamente el 1 de Junio de **1972**, *Nolan Bushnell* junto a *Ted Dabney* fundaron **ATARI**, palabra que proviene del juego del Go y viene a ser como el "jaque mate" del ajedrez.

Las primeras oficinas de *Atari* se situaron en una zona industrial de Santa Clara, todo muy precario y subsistiendo del poco dinero que habían conseguido hacer con **Computer Space**. El primer cliente de *Atari* fue *Bally*, una empresa dedicada a las máquinas tragaperras y los pinballs. Su encargo era el de hacer máquinas de pinball más anchas de lo normal. Compraron unas cuantas máquinas, las modificaron y las instalaron en los bares de la zona, de forma que pudieran darles un mantenimiento rápido sin que les costase mucho desplazarse.

El segundo empleado de *Atari* fue *Allan Alcorn*, un ingeniero que *Ted Dabney* conocía. Como acababa de entrar en la empresa, *Bushnell* le encargó un primer proyecto, aunque le contó un par de mentiras. Le dijo que acababa de firmar un contrato con *General Electric* para construir un juego de ping pong. La idea de *Bushnell* era la de que *Alcorn* comenzara a manejarse en el campo de diseño de juegos mientras él diseñaba un proyecto más importante, ya que *Bushnell* había hablado con *Bally* para comenzar a crear un juego de carreras.

Alcorn comenzó a eliminar componentes y a abaratar el producto y comenzó a darle la forma que él quería. Sustituyó las palas de ping pong que había sugerido *Bushnell* por unos

rectángulos divididos en ocho secciones diferentes en los cuales la bola rebotaba de diferente manera, haciendo el juego mucho más real. Añadió también aceleración a la pelota.

Alcorn tardó unos tres meses en conseguir crear un prototipo operativo, del cual quedaron "enamorados" *Bushnell* y *Dabney*. *Bushnell* lo llamó **Pong**, y para probarlo lo introdujeron en los bares donde tenían instalados pinballs de *Bally*. Uno de los bares era el *Andy Capp's Tavern*, cuyo propietario llamó a *Bushnell* una noche diciéndole que fuera a cambiar la máquina porque ya no funcionaba. Cuando *Bushnell* fue, se encontró que el cajetín de monedas estaba colapsado, y que al no caber más, no se podía jugar.

Pong fue el mayor éxito jamás visto en la industria de los videojuegos hasta ese momento, cambiando para siempre una industria que dura hasta hoy en día.

Figura 12: Pantalla de juego de Pong.

En **1973**, *Atari* lanza su segundo juego llamado **Space Race**, donde dos cohetes competían por llegar a lo más alto de la pantalla mientras esquivaban asteroides que cruzaban la pantalla de lado a lado.

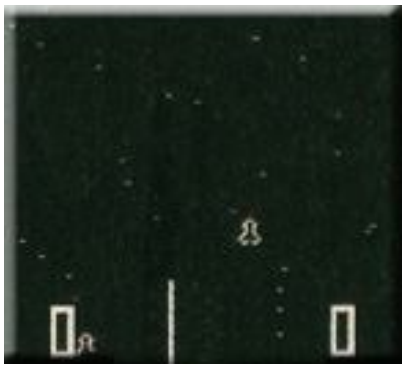

Figura 13: Pantalla de juego de Space Race.

Atari comenzó **1974** lanzando un juego cada mes, pero las nuevas ideas no llegaban y el mercado se colapsó con diferentes versiones de **Pong** como **Pin Pong** o **Quadra Pong**.

En ese mismo año apareció una gran compañía en contra de *Atari*: *Kee Games*. Fundada por *Joe Keenan*, vecino de toda la vida de *Bushnell* y que había formado parte de *Atari*. Desde el primer momento, las relaciones entre las dos compañías fueron muy tensas e incluso *Atari* acusó a *Kee Games* de espionaje industrial. Pero en realidad todo esto era una farsa, pues la compañía era fruto del propio *Bushnell*. La idea de *Bushnell* fue hacer ver al consumidor que había competencia, darse publicidad y de paso, colar un porcentaje mayor de sus máquinas en los salones recreativos.

Con todo esto, *Atari* consiguió consolidarse en el mercado, aunque *Kee Games* resultó ser un poco más peligrosa de lo que *Bushnell* había previsto. *Steve Bristow* llevaba ya tiempo desarrollando un innovador juego llamado **Tank**. En este juego, los jugadores controlaban un tanque blanco y otro negro, y se enfrentaban entre sí en un mapa laberíntico. El juego tenía dos joysticks para controlar el tanque y un botón para disparar en el joystick derecho. Para ganar el juego debías impactar al rival o hacer que pasara sobre unas minas colocadas por el escenario que se representaban con una "X".

Tank representó un avance importante en la industria, al ser el primer videojuego de la historia en utilizar una memoria ROM para almacenar los gráficos. Fue el juego del año sin duda y no era de *Atari* (es lo que pensaba la gente). En ese mismo año apareció **Tank II**, con barreras en el mapeado.

Otra idea revolucionaria llegó de *Mayer and Emmons*, un pequeño estudio que comenzó a desarrollar el primer videojuego de carreras de la historia, llamado **Gran Trak 10**. En este juego el jugador utilizaba un volante como mando, o sea, un control analógico como el de **Pong** pero para girar, con el que conducía un rectángulo blanco en forma de coche en un circuito oval.

Fue muy costoso de desarrollar y aún más de distribuir. A punto de ser lanzado, descubrieron que era prácticamente injugable, por lo que *Alcorn* tuvo que entrar de nuevo en escena para arreglar el problema y dejarlo en un estado óptimo para su distribución. El juego le costó a *Atari* 1095$ y por un error que hubo se vendía a 995$. Tras esas pérdidas iniciales, **Gran Trak 10** fue el juego más vendido de **1974**. Otro juego importante de carreras de ese año fue **Speed Race** de *Taito*.

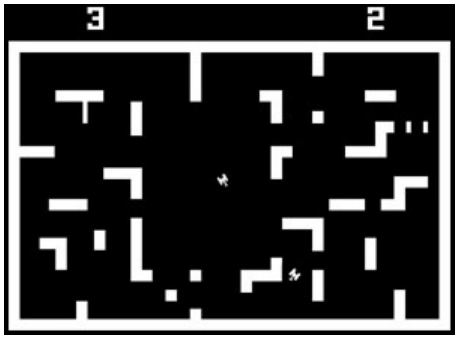

Figura 14: Captura de Tank.

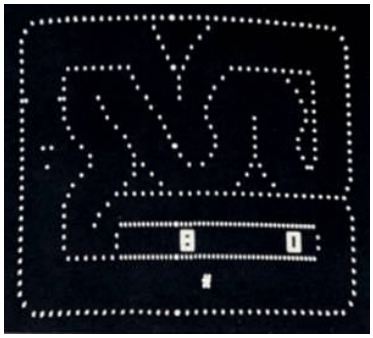

Figura 15: Captura de Gran Trak 10.

A finales de **1974** se produjo uno de los hechos más importantes en la industria del videojuego, el prototipo del primer chip que contenía el **Home Pong**. *Harold Lee*, un ingeniero de *Atari*, fue quien propuso la idea: una máquina que pudiera conectarse al televisor para jugar a **Pong** en casa. A *Bushnell* le encantó la idea.

Figura 16: Home Pong.

En **1975** aparece el primer juego en color, **Indi 800**, de la mano de *Atari*. Otras muchas compañías siguieron desarrollando juegos como **Tank III** (*Kee Games*) o **Wheels II** (*Midway*). Además *Atari* triunfó vendiendo su **Home Pong**.

En **1976** aparece gran cantidad de competencia en el mundo de los videojuegos y grandes novedades. Tras el éxito de ventas de **Home Pong**, cientos de compañías se lanzaron a crear consolas y dispositivos electrónicos para jugar en casa de la misma forma.

En este año, *Atari* contrata a *Steve Jobs* y se empieza a gestar el **Breakout**, que consistía en un nuevo concepto jugable a partir de pong, ya que en esta ocasión sólo había una pala y en el otro extremo una serie de bloques que había que ir destruyendo con una pelota que iba rebotando en ellos. **Breakout** acabó siendo el mejor juego del año y también el primer "clon" de Pong que conseguía ser más adictivo que el original. Otro juego para destacar en ese año fue el **Sprint 2** de *Kee Games*, juego de carreras.

Figura 17: Máquina recreativa de Breakout.

1977, es un año en el que empiezan a aparecer las videoconsolas de cartuchos como la **VCS** de *Atari*. También aparece la primera videoconsola de *Nintendo*, **Nintendo Tv Game 6**, que permitía jugar a 6 versiones diferentes de **Light Tennis**, una versión mejorada y a cuatro colores de Pong. Entre los videojuegos cabe destacar **Cinematronics**, proyecto de la tesis doctoral de *Larry Rosenthal*, que era una versión de Space War y traía como novedad los gráficos vectoriales.

Toshihiro Nishikado era uno de los programadores de *Taito*, y para uno de sus proyectos se le ocurrió la idea de un soldado parapetado detrás de unas trincheras, que disparaba a filas de enemigos acercándose hacia él. Pero la compañía pensó que era demasiado violento, y que podría acarrear problemas, así que decidió cambiar a los humanos por alienígenas y naves espaciales.

De esta manera nació **Space Invaders** en **1978**, la recreativa que dio a conocer los videojuegos al mundo. En este juego, el jugador (detrás de escudos cósmicos) debía vencer a 4 filas de marcianitos que cada vez se aproximaban a él más y más rápido. Pero su principal innovación estuvo en algo que durante años sería muy importante en los videojuegos: El *Hi-Score*. También cabe destacar que antes se jugaba a los juegos por tiempos y el Space Invaders incorporaba tres vidas.

Figura 18: Pantalla de Space Invaders.

Space Invaders fue un éxito absoluto, llegando a provocar que el gobierno nipón cuadriplicara la producción de monedas de yen, por su escasez a causa del juego. Pero no sólo fue éxito en Japón, sino que a través de *Midway* llegó a América con idénticos resultados. El videojuego salió de los bares y salones recreativos y dio el salto a varios negocios. Todo el mundo quería tener una máquina de Space Invaders. Según *Taito*, Space Invaders ha facturado más de 500 millones de dólares a lo largo de su historia.

En **1978**, aparecen numerosas videoconsolas como **Bally Astrocade**, **Magnavox Odyssey 2** y la **Nintendo Tv Game 15**.

En **1978**, también cabe destacar la aparición del primer "huevo de pascua". Los huevos de pascua son comunes hoy en día en el mundo del software y suelen ser elementos ocultos dentro de un producto. En este año, en el juego **Adventure** de *Atari*, había una sala oculta que contenía el nombre del programador del juego, *Warren Robinett*.

Para acabar la década, en **1979** aparecen gran cantidad de juegos míticos, entre ellos **Asteroids** de *Atari*. El objetivo del juego consistía en disparar y destruir los asteroides sin que ningún fragmento se llegase a estrellar contra la nave que se controlaba. Los jugadores contaban con un botón de disparo y con la posibilidad de girar la nave sobre su propio eje. Asteroids fue un superventas.

Por su parte, *Namco* lanza **Pac-Man**, juego diseñado por *Toru Iwantani*. El juego supuso una revolución total en los videojuegos que probablemente no vuelva a suceder. Tuvo tanto éxito que hasta se le dedicó una portada en la revista *Time Magazine*, una serie de dibujos animados y una canción. Programado por *Hideyuki Makajima* y su equipo, el juego se terminó en un año y medio, trabajando en él ocho personas. El equipo se dividió en dos, dedicándose uno de los grupos al hardware y el otro al software. Se considera al **Pac-Man** como el videojuego más influyente de la historia junto al pionero *Pong*.

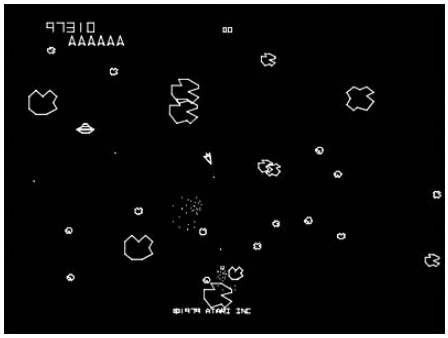

Figura 19: Pantalla de Asteroids.

Figura 20: Pantalla de Pac-Man.

Durante este año, se fundan *Capcom* [17] y *Activision*.

2.1.3 Década de los 80

Al principio de los **80**, *Nintendo* comienza a distribuir las **Game & Watch**, creadas por *Gunpei Yokoi*. Consistían en un solo juego que se podía jugar en la pantalla LCD, además de ser un reloj y alarma. Estas consolas fueron las responsables de la invención del pad de control.

En **1981**, de manos de *Shigeru Miyamoto* para *Nintendo*, nace **Donkey Kong**, donde un fontanero llamado Jumpman (más adelante Mario), debía rescatar a su novia de las garras del gorila Donkey Kong.

Figura 21: Game & Watch.

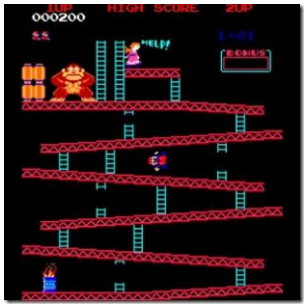

Figura 22: Pantalla de Donkey Kong.

En **1982** aparece **Commodore 64**, esta videoconsola utilizaba unidad de casete y disquetera, existiendo expansiones para cartucho. Tenía aplicación de juegos, gráficos y permitía utilizar el lenguaje de programación BASIC. Contaba con una paleta de 16 colores.

Además de otras videoconsolas, *Atari* rediseñó la **2600**, que incluía el *Pac-Man* y aparece el **ZX Spectrum** de *Sinclair* en el que los juegos venían en cintas de casete, que acabaría siendo el microordenador más popular de la década de los 80 en Europa.

Figura 23: Commodore 64.

Figura 24: Atari 2600.

Figura 25: Spectrum ZX + 2.

También cabe destacar la aparición de la videoconsola **Vectrex**, que basaba sus juegos en vectores, propiedad que le hacía capaz de recrear entornos 3D, algo muy novedoso para la época.

En cuando a desarrollo de software de juegos, en **1982** aparecen a escena las empresas *Electronics Arts* [30] y *Ocean*.

1983 se caracterizó por la crisis que apareció debido a la gran cantidad de videojuegos y videoconsolas que existían. También hay que destacar la aparición de **Mario Bros** de *Nintendo* y el **Bomberman** de *Hudson Soft*. Además del lanzamiento de la primera videoconsola de *Sega*, la **SG-100**.

Figura 26: Pantalla de Mario Bros.

Figura 27: Pantalla de Bomberman.

En este año aparece la primera empresa Española de videojuegos, *Indescomp*, que comenzaría su andanza distribuyendo los primeros **ZX Spectrum** y **Amstrad CPC** del país y comenzando a distribuir videojuegos de creación española, como **La Pulga**, que curiosamente obtuvo más éxito en Inglaterra.

En **1984** continúa la crisis, aunque se puede destacar la aparición de juegos como **1942**, **Bomb Jack** o **Paper Boy**. Cabe destacar que en este año aparece, la que se considera como la primera aventura gráfica de la historia, **King´s Quest I**, desarrollada por *Sierra Online*.

Por otro lado, este año es el considerado como el inicio de la "Edad de oro del Videojuego Español", ya que aparecen empresas como *Dinamic Software* (Dinamic Multimedia a partir de 1993).

En **1985** comienza una recuperación de la industria del videojuego. De la mano de *Nintendo* aparece **Super Mario Bros** para la consola *Famicom*. En este año se crea **Tetris**, que será uno de los juegos más importantes de la historia, fue inventado por *Alexey Pazhintov*.

Figura 28: Pantalla de Super Mario Bros.

Figura 29: Captura de Tetris.

Los juegos más importantes de este año fueron: **Pitfall 2** de *Sega*, **Indiana Jones and the Temple of Doom** y **Empire Strikes Back** de *Atari*, **Commando** y **Ghosts'N Goblins** de *Capcom* y **Dig Dug 2** de *Namco*. Merece especial atención el **Tehkan World Cup** (uno de los primeros juegos de futbol).

En cuando a hardware, aparece la **NES** (Famicom) de *Nintendo*, *Sega* lanza la **Master System** y aparece el **Amiga 1000** de *Commodore*.

En el año **1986** aparecen gran cantidad de juegos destacables como son **The Legend of Zelda** o **Metroid**. *Taito* presenta **Arkanoid**, basado en Breakout y nace **Castlevania**.

Figura 30: Imagen de The Legend of Zelda.

Figura 31: Captura de Arkanoid.

El año **1987**, se puede decir que es el año de los arcades. Este año destacan por la calidad de sus lanzamientos las compañías *Sega*, *Capcom* y *Taito*. Cabe mencionar como juegos destacados **Wonder Boy** de *Sega*, **1943 Battle Of Midway** y **Street Fighter** de *Capcom*.

En este año aparece también el primer **Final Fantasy** de *Squaresoft*, que comenzaría una saga que sigue hasta prácticamente la actualidad. Aparece también **Maniac Mansion** de *Lucasfilm Games*, proyecto de *Ron Gilbert*. Para este juego, se diseñó un motor específico llamado **SCUMM** (*Script Creation Utility for Maniac Mansion*), que supuso una revolución en los interfaces de juego de la época. Por otro lado aparece **MegaMan** y se publica el primer **Metal Gear**, saga que continúa en la actualidad.

Figura 32: Pantalla de Final Fantasy.

Figura 33: Captura de Maniac Mansion.

En **1988** *Sega* lanza la videoconsola **Mega Drive**, entre su catálogo destacan juegos como **Sonic**, **Golden Axe** o **Street Fighter 2**. En este año apareció **La abadía del Crimen**, considerado por muchos como el mejor videojuego de creación española durante años, creado por *Paco Menéndez* y *Juan Delcán*.

Para finalizar esta década, en **1989**, aparece la portátil más famosa de todos los tiempos, la **Game Boy** de *Nintendo*. Además aparecen juegos de la mano de *Capcom* como **Final Fight**,

Cadillacs & Dinosaurs, **Captain Commando** y como más destacable aparece **Pang!** También aparece **Sim City**, diseñado por *Will Wright*, juego que permitía al jugador crear sus propios mapas. Era un simulador de construcción de ciudades que supuso la creación de un subgénero dentro de la estrategia.

Figura 34: Game Boy.

Figura 35: Pantalla de Sim City.

Figura 36: Captura de Pang!

2.1.4 La década de los 90

A principios de los **90**, aparece la nueva videoconsola de *Nintendo*: la **Super Famicom** o **Super NES**. Arrasó por todo el mundo con juegos como **Super Mario World**, **Super Mario Kart**, **The Legend of Zelda: A Link to the Past** o **Super Metroid** entre otros.

Sega por su parte lanza la competidora de Game Boy, la **Game Gear** y la competidora de Super Nes, la **Master System II**. Por otra parte se pone a la venta la **Neo Geo** de *SNK*, aunque tanto la videoconsola como sus juegos eran bastante caros y poco accesibles. También aparecen algunas otras videoconsolas de menos consideración.

El juego **Super Mario World**, contiene una gran mejora en gráficos, sonido y jugabilidad. Además y gracias al ya mencionado motor *SCUMM*, aparece **The secret of Monkey Island**, juego que supuso una revolución en las aventuras gráficas, tanto por sus novedosas batallas de insultos, como porque era imposible que te mataran.

Figura 37: Super NES.

Figura 38: The Secret of Monkey Island.

En el año **91**, año en el que aparece el CD, aparece también el competidor de Mario, **Sonic**, el famoso erizo azul de ***Sega***, en un juego de plataformas que llegaría a vender 4 millones de copias.

Este año aparecen también juegos dignos de destacar, como **The Legend of Zelda: A Link to the past**, juego de rol con la novedad de que el protagonista Link, podía viajar a través de dos mundos paralelos. Junto con Zelda aparece **Lemmings**, un juego con un sistema novedoso en el que había de dirigir a un grupo de Lemmings hasta la salida de cada nivel. Este juego estaba disponible para casi todo tipo de plataformas. Y para finalizar el año de novedades aparece también la segunda parte de **Street Fighter** en los que cada luchador tenía unos movimientos y características particulares.

Figura 39: The Legend of Zelda: A Link to the past.

Figura 40: Pantalla de Lemmings.

Figura 41: Captura de Street Fighter II.

En este año **1991**, aparece una crisis en la industria del videojuego en España, por lo que se pone punto y final a la conocida “Edad de Oro del Videojuego Español”.

En el año **1992**, aparece la segunda parte de **Sonic** y un competidor a *Street Fighter II* tras su éxito, el **Mortal Combat** de manos de *Midway* entre otros muchos juegos de lucha. El juego destacaba por ser tremendamente violento, e incorpora los famosos “fatalitys”, que eran matar al rival de una forma muy violenta y gore.

En este año se empiezan a realizar intentos de efectos y juegos en tres dimensiones, es el caso de **Wolfenstein 3D**, que aunque no fue el primer juego con perspectiva tridimensional, si fue el primero en triunfar. No presentaba grandes novedades pero el concepto de juego en primera persona fue revolucionario, y supuso un salto de calidad de los ordenadores sobre las videoconsolas.

En este año también aparece **Survival Horror: Alone in the Dark**, juego en tercera persona y que fue de los primeros en utilizar un entorno 3D tal y como se conoce hoy en día. Gráficamente era una maravilla en la época.

Figura 42: Mortal Combat.

Figura 43: Captura de Wolfenstein 3D.

Figura 44: Survival Horror: Alone in the Dark.

En el año **1993** aparece otra gran remesa de videoconsolas, de entre las cuales puede destacar la **Mega Drive II** de *Sega*.

En este año aparecen dos juegos destacables: el primero de ellos es el **Doom**, similar al anterior Wolfenstein, que obtuvo mucho más éxito y que sirvió de base para todos los **FPS** (*First Person Shooter*, disparador en primera persona) que salieron más adelante a la venta. Técnicamente era muy bueno para la época, con muy buenos modelados y mapas enormes. El motor gráfico fue diseñado por *John Carmack*.

El segundo de los juegos destacables que apareció en este año fue el primer **FIFA**, de *EA* (Electronic Arts). Tenía una novedosa vista isométrica, sistema de repeticiones y una buena inteligencia artificial.

Figura 45: Captura de Doom.

Figura 46: Pantalla de FIFA.

El año **1994** destaca por la aparición de consolas con lector de CD como la **Saturn** de *Sega*, la **Neo Geo CD** de *SNK* y la gran triunfadora de todas, la **PlayStation** de *Sony*.

Entre los juegos, cabe destacar la aparición de **Warcraft** de mano de *Blizzard*. Este juego supuso un punto de inflexión en los juegos de estrategia en tiempo real (**RTS**), pasando a ser un superventas. Su punto fuerte era la inteligencia artificial que poseía.

Figura 47: PlayStation.

Figura 48: Pantalla de Warcraft.

Omitiendo el año **95**, sólo destacable por la gran cantidad de fracasos en la creación de videoconsolas por parte de distintos fabricantes, en el año **1996**, *Nintento* lanza la **Nintendo 64**, que destacó por seguir teniendo los juegos en cartuchos y no pasar al formato CD como el resto de compañías, además aparece la **Game Boy Pocket**, una Game Boy con un tamaño más reducido.

Entre los juegos que aparecen este año se pueden destacar dos: el **Super Mario 64** de *Nintendo*, en el que el gran icono de Nintendo, Mario, se pasa al mundo 3D. El juego se considera como uno de los mejores juegos de la historia, contaba con una gran calidad gráfica para la época y no perdía la jugabilidad respecto a las anteriores versiones en 2D. Junto con Super Mario 64, aparece de las manos de *Capcom* y para *PlayStation* el **Resident Evil**, juego de una de las sagas más conocidas del mundo y que fue una revolución en el género de acción 3D en tercera persona.

1997 es un año de grandes lanzamientos de videojuegos. Uno de los más destacados es el **Final Fantasy VII** de *Squaresoft*, juego de rol de los más famosos de la historia. El primero de la saga con los personajes modelados en 3D y que se consideró una película hecha videojuego. Fue un superventas.

Otro juego destacable es **Gran Theft Auto**, de *ASC Games*, el inicio de la que será una de las sagas más famosas en el mundo de los videojuegos. Desde una vista de pájaro con unos gráficos no muy buenos, triunfó gracias a la libertad de movimiento de ofrecía y las grandes ciudades que tenía.

El último de los juegos destacables de este año, es el **Age of Empires**, desarrollado por *Ensemble Studios*, que consistía en un juego de estrategia en tiempo real, con una destacable y pionera (para este tipo de juegos) vista isométrica.

Figura 49: Final Fantasy VII.

Figura 50: Grand Theft Auto.

Figura 51: Age of Empires.

En el año **97**, empiezan también a tomar importancia los juegos para móviles, como el juego **Snakes** para *Nokia*.

El año **98** trae consigo algunas novedades tanto hardware como software. Como nota, *Atari* es comprada por *Hasbro*, *Sega* lanza su última consola, la **Dreamcast**, *Nintendo* la **Game Boy Color** y aparece la **Neo Geo Pocket** de *SNK*.

Un juego destacable este año es el **Metal Gear Solid** para *PlayStation*, desarrollado por *Konami*. Juego de acción en tercera persona que aprovechaba todo el potencial gráfico de la *PlayStation*.

Aparece, también para *PlayStation* el primer **Gran Turismo**, era de los primeros juegos que introducía al jugador como conductor novel que tenía que ir avanzando y obteniendo licencias para poder conducir con mejores coches y participar en mejores competiciones. Fue el juego más vendido para la consola de *Sony*.

El último juego destacable del año fue **Legend of Zelda: Ocarina of Time**, para *Nintendo 64*. Fue el primer título de la saga en tres dimensiones. Para muchos el mejor juego de la historia.

Figura 52: Metal Gear Solid.

Figura 53: Gran Turismo.

Figura 54: Legend of Zelda: Ocarina of Time.

2.1.5 La era modera

Para finalizar la historia del videojuego, quedan once años, desde el 2000 hasta el 2011, esta era moderna podría dividirse en dos grandes grupos: el de las consolas de penúltima generación (PlayStation 2, Xbox y Game Cube) y el grupo de las consolas de última generación (PlayStation 3, Xbox 360 y Wii), añadiendo a cada grupo el resto de consolas, consolas portátiles y ordenadores. También podría dividirse por ejemplo por el paso de juegos en red o incluso cualquier otra división. Como aún no se ha encontrado mucha información de cómo hacer esta división, se agrupa todo el contenido como era moderna.

En el año **2000** aparece la **PlayStation 2** de *Sony*, con lector de DVD. Para esta consola salen gran cantidad de juegos, de todos los géneros y estilos.

En **2001** *Nintendo* saca **Game Cube**, de nuevo alejada del sistema CD y DVD, usando Mini-DVD. *Nintendo* saca también la evolución de su videoconsola portátil con **Game Boy Advance**. En este mismo año *Microsoft* aparece en el mundo de las videoconsolas con **Xbox**, la primera consola en incluir un disco duro.

En este año aparecieron gran cantidad de juegos, de hecho desde la aparición de *PlayStation I*, la cantidad de juegos que aparecen por año es gigante y es un hecho que sigue en la actualidad, como se ha comentado en la introducción del proyecto. En el **2001** se pueden destacar dos grandes juegos: **Halo** para *Xbox*, otro FPS que se convertirá en una saga importante de este género y **Pro Evolution Soccer** de *Konami*, gran competidor para *FIFA*.

Figura 55: Imagen de Halo.

Figura 56: Captura de Pro Evolution Soccer.

En **2003** destaca la aparición de la consola portátil de *Nintendo* **Game Boy Advance SP**, una versión reducida de la anterior y la **NGage** de *Nokia*.

2004 se caracteriza de nuevo por actualizaciones en el mundo de las portátiles: *Nintendo* saca toda una revolución, la **Nintendo DS**, con dos pantallas, una de ellas táctil, lápiz, reconocimiento de voz y wifi que revoluciona el mundo de las portátiles y consigue atracción para nuevos tipos de jugadores y juegos.

Sony también se adentra en el mundo de las portátiles y en **2004** saca la **PSP**, que usa discos UMD (Universal Media Disk), parecido a un DVD pero más pequeño. La consola permitía ver películas en ella y tenía una potencia muy superior a la Nintendo DS.

En **2005** *Microsoft* lanza su segunda consola al mercado, revolucionándolo, la **Xbox 360**. Es una de las videoconsolas actuales más vendida. Incorpora 3 núcleos, capacidad de jugar online y montones de accesorios.

En **2006** y como réplica a Microsoft, *Sony* saca a la venta **PlayStation 3**, la consola más potente de la actualidad.

Este año y de nuevo sin utilizar grandes prestaciones, *Nintendo* vuelve a triunfar con **Wii**, una consola en la que los mandos tienen sensores y para jugar hay que mover las manos, revolucionando de nuevo el mercado y triunfando con juegos familiares. A su vez *Nintendo* también rediseña su consola portátil con la **Nintendo DS Lite**, que es actualmente la videoconsola más vendida.

Figura 57: Las consolas de última generación: Xbox 360, PlayStation 3, Wii, DS Lite y PSP.

Para finalizar con el hardware, hace escasos meses *Nintendo* ha sacado una nueva versión de su videoconsola portátil, la **Nintendo 3DS**, con capacidad de mostrar 3D sin necesidad de gafas ni lentes y *Microsoft* por su parte ha sacado un revolucionario sistema de detección de movimiento llamado **Kinect** con el que los jugadores pueden interactuar con la consola utilizando su propio cuerpo.

En cuanto al software, la cantidad de juegos que aparecen en la actualidad como se ha comentado es gigante, desde simples puzles para móviles y tabletas hasta complicados juegos 3D con gráficos increíbles, pasando por juegos con periféricos como guitarras o volantes. A continuación se citan algunos juegos de la actualidad, que se han considerado de interés.

Gears of War 3, de *Epic* es uno de los FPS más esperados de la actualidad. *Microsoft* ha anunciado que se podrá jugar a la versión Beta en **Abril de 2011**, por lo que pronto estará disponible la versión definitiva. La calidad gráfica y dinamismo de esta saga ha impresionado en todas sus ediciones y esta no será menos. La saga destaca por el sistema de coberturas de los jugadores, así como los intercambios entre tercera persona (para moverse) y primera persona (para disparar). Gears of War 3 estará disponible únicamente para *Xbox 360*.

Figura 58: Captura de Gears of War.

Figura 59: Captura de Gears of War.

StarCraft apareció de la mano de *Blizzard* en **1998**, es un RTS parecido y evolucionado de Warcraft, con una ambientación futurista. Ha sido uno de los juegos más vendidos de la historia. Tras muchos años de espera, en **2010** apareció **StarCraft II** [27], sorprendiendo de nuevo a todo el mundo, con sus efectos, velocidad y jugabilidad. Este juego está disponible tanto para *Windows* como para *Mac OS X*.

Figura 60: Captura de Starcraft II.

Figura 61: Imagen de Starcraft II.

Entre los juegos musicales como karaokes, destaca la saga de **Guitar Hero**. Un juego en el que imitar a grupos de música míticos tocando sus canciones con guitarra, bajo, batería o cantando. Se incluyen todo tipo de accesorios para poder interactuar con el juego y tocar los instrumentos. También han desarrollado un mini sistema de pinchadiscos, el **DJ Hero**.

Figura 62: Periféricos de Guitar Hero: World Tour.

Figura 63: Pantalla de Guitar Hero: World Tour.

En cuanto a los juegos de deportes, en concreto el fútbol, durante estos años ha habido gran competencia entre **Pro Evolution Soccer** y **Fifa**, y cada año salen nuevas versiones mejoradas y con grandes novedades. Por ejemplo ahora se puede jugar en tercera persona controlando a un único jugador, participando en Internet contra otros muchos usuarios. Este año parece que **Fifa 11** se lleva la palma y ha triunfado más que **Pro Evolution Soccer**.

Figura 64: Captura de Fifa 11.

Figura 65: Imagen de Fifa 11.

Otra saga destacable de los últimos tiempos, es la saga de **Assassin´s Creed**. Un juego con gran ambientación y grandes mapas, en tercera persona y con mucha libertad de movimientos, que incorpora una gran historia a su alrededor. Para **2012** aparecerá la siguiente versión del mismo, el **Assassin´s Creed 3**.

Figura 66: Imagen de Assassin´s Creed.

Figura 67: Captura de Assassin´s Creed II.

La saga de **Gran theft Auto** de *Rockstar Games* [28], sigue su andadura en la actualidad con la versión **IV** del mismo y la más laureada que apareció en **2008**, se espera para este año **2011** la versión **5**. Ciudades enteras que recorrer, donde se puede hacer de todo, con gráficos excepcionales. Aseguran muchas horas de juego formando parte de una red de delincuencia.

Figura 68: Captura de GTA IV.

Figura 69: Imagen de GTA IV.

Uno de los juegos más revolucionarios en la actualidad, que ha servido como referente de otro nuevo modelo de negocio, con ganancias arrolladoras, es el **World of Warcraft**, juego **MMOG** (Massive Multiplayer Online Game – Juego multijugador masivo en línea), en el que gran cantidad de jugadores se dan cita online y compiten unos contra otros en un mundo fantástico con sus personajes 3D. World of Warcraft funciona por suscripción mensual y hoy en día hay unos 12 millones de subscriptores mensuales.

Figura 70: Captura de World of Warcraft.

Figura 71: Imagen de World of Warcraft.

En cuanto a los juegos de coches, a finales de **2010** apareció **Gran Turismo 5**, para *PlayStation 3*. Entre las novedades, además de la más que visible mejora gráfica que roza casi la perfección, ahora se puede jugar en línea con hasta 32 participantes y aparecen coches de la marca Ferrari. Juego imprescindible para todo amante de los simuladores de coches.

Figura 72: Captura de Gran Turismo 5.

Figura 73: Imagen de Gran Turismo 5.

Para señalar un juego de *Wii*, se destaca **Wii Party**, un videojuego familiar que apareció a finales de **2010** y que utiliza todo el potencial de los mandos de Wii y triunfa con la simple idea de un juego lleno de mini juegos en los que toda la familia puede disfrutar y a la vez ser competente. Otro modelo de negocio, enfocado a los más mayores y a su vez hacia los más pequeños que también está dando sus frutos sin grandes complicaciones gráficas, videos como películas o bandas sonoras de ensueño.

Figura 74: Chicos jugando a Wii Party.

Figura 75: Captura de Wii Party.

Para finalizar se destaca la aparición de los móviles de última generación y tabletas, que disponen de gran potencia y han provocado un gran crecimiento del sector de los videojuegos enfocados a estas plataformas, en las que triunfan los juegos simples y puzles, incluso provocando la reedición de juegos míticos y antiguos como muchos de los que se ha hablado en este capítulo. Cabe destacar por ejemplo el juego **Agry Birds**, de *Roxio*, que a un precio más que bajo y con un sistema de juego muy simple: lanzar pájaros contra estructuras en las que se refugian los cerdos que hay destruir para avanzar la pantalla, han conseguido ganancias millonarias y han creado otro modelo de negocio en auge en la actualidad.

Figura 76: Captura de Angry Birds.

Figura 77: Angry Birds en un iPad de Apple.

2.2 Juegos de skate en la historia

Los videojuegos de monopatines han estado presentes a lo largo de toda la historia de los videojuegos, y continúan en la actualidad estando presentes. A continuación se comentan algunos de los videojuegos relacionados con los monopatines más destacables.

En el año **1986**, apareció una máquina arcade diseñada por *Atari* con el juego **720º** [25]. Un juego de monopatines, donde con un joystick y un par de botones, se controlaba a un skater por la ciudad, pudiendo dar saltos y piruetas, además de competiciones más cercanas al mundo del longboard como el slalom. Puede decirse que es el primer juego de deportes extremos de la historia. El juego se pasó posteriormente a diferentes soportes desde la *Commodore 64* en **1987** hasta la *Game Boy Color* en **1999**, pasando por *NES* o *Spectrum*.

Figura 78: Portada de 720º.

Figura 79: Arcade de 720º.

Figura 80: Captura de 720º.

Este primer juego fue la secuela del juego **Skate or Die!** creado por *Electronics Arts* [30] en **1987** para casi todo tipo de plataformas, además de la *NES*. Al igual que su predecesor, se podía participar en un montón de tipos de pruebas como salto, slalom o descenso.

Figura 81: Portada de Skate or Die!

Figura 82: Captura de Skate or Die!

Hay que esperar hasta **1999** para la aparición del siguiente juego de monopatines y que se convertiría en una de las sagas de videojuegos deportivos más famosas de la historia. De la mano de *Activision* junto con *Neversoft* aparece **Tony Hawk Pro Skater** [29].

Este juego incluía skaters famosos, gráficos en 3D, gran cantidad de pantallas y buena música. La saga de Tony Hawk ha continuado hasta la actualidad con los siguientes lanzamientos:

- *Tony Hawk's Pro Skater* (1999)
- *Tony Hawk's Pro Skater 2* (2000)
- *Tony Hawk's Pro Skater 3* (2001)
- *Tony Hawk's Pro Skater 4* (2002)
- *Tony Hawk's Underground* (2003)
- *Tony Hawk's Underground 2* (2004)
- *Tony Hawk's American Wasteland* (2005)
- *Tony Hawk's American Sk8land* (2005)
- *Tony Hawk's Project 8* (2006)
- *Tony Hawk's Downhill Jam* (2006)
- *Tony Hawk's Proving Ground* (2007)
- *Tony Hawk's Ride* (2009)

- *Tony Hawk´s Shred* (2010)

Esta gran saga, entre unos juegos y otros, tiene títulos para todas las plataformas existentes en la actualidad, desde un PC o MAC hasta las videoconsolas de última generación, pasando por videoconsolas portátiles y teléfonos móviles.

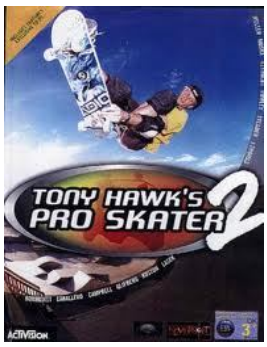

Figura 83: Portada de Tony Hawk Pro Skater 2.

Figura 84: Captura de Tony Hawk Pro Skater 3.

Además, el avance tecnológico queda patente en cada una de las nuevas versiones, con mejores físicos, mejores escenarios y mejores diseños 3D. Incluso en algunos de los títulos, el personaje puede bajarse de su patín y moverse por toda una ciudad andando, en coche, subirse por tejados e incluso hacer grafitis.

Si hay que destacar alguno de ellos, sería el **Tony Hawk´s Pro Skater 2**, ya que para el año en que se desarrolló (**2000**), tenía unos gráficos increíbles y una jugabilidad impresionante, además de una banda sonora con grandes grupos del momento. En la actualidad se sigue jugando al mismo, incluso se ha realizado la portabilidad a plataformas móviles como *iPhone*.

Otros juegos de la saga a destacar son los dos últimos: **Ride** y **Shred**, que incorporan un nuevo periférico con forma de tabla de skate y que el jugador debe controlar con los pies, de la misma forma que un skate real.

Figura 85: Chico jugando a Tony Hawk Shred.

Figura 86: Captura de Tony Hawk Ride.

Durante los años de reinado de la saga *Tony Hawk*, salieron algunos otros juegos de skate, que no llegaron a triunfar ya que esta saga tenía el listón demasiado alto para la competencia. Sólo uno de estos competidores parece haberse hecho un hueco entre los jugadores, y en **2007** aparece el juego **Skate**, creado por *Electronics Arts* [30].

El juego destaca con respecto a Tony Hawk porque se adentra un poco más en la simulación, es decir, el control del skate es mucho más realista y más difícil que en Tony Hawk. Por ejemplo en Tony Hawk, el salto normal de monopatín (Ollie) se hace pulsando un botón, mientras que en Skate, hay que hacer el movimiento del salto con un stick, bajando y subiendo el mismo, imitando el movimiento de piernas que debe de hacer un skater en la realidad para saltar.

La saga **Skate** está actualmente triunfando mucho, sobre todo entre patinadores que juegan a la videoconsola, debido a su acercamiento a la realidad y la dificultad para ejecutar los trucos. El **Skate 2** apareció en **2009**, al igual que su antecesor está disponible para casi todas las plataformas. Finalmente en **2010** apareció el **Skate 3**, para las dos consolas de última generación más potentes: *Xbox 360* y *PlayStation 3*.

Figura 87: Portada de Skate 3.

Figura 88: Imagen de Skate 3.

Para finalizar, cabe destacar que hoy en día y sobre todo para plataformas de móviles están apareciendo algunos títulos de skate, sin llegar a ser tan sorprendentes como la saga Tony Hawk o Skate, destacando por ejemplo **Touchgrind** para *iPhone/iPad* en el que se controla un skate para dedos o fingerboard, con los dedos del jugador en la pantalla multitáctil de estos dispositivos.

Figura 89: Portada de Touchgrind.

Figura 90: Touchgrind en un iPhone de Apple.

Otro detalle a destacar es que pese a haber algunos títulos que hacen referencia al downhill o disponen de modo de juego de descenso, en realidad se centran en descensos incluyendo saltos y piruetas, pero ninguno se centra en el objetivo de este proyecto: una competición de downhill pura, con descenso en longboard.

2.3 Ogre 3D y otros motores gráficos

En la actualidad existen gran cantidad de motores gráficos, algunos especialmente diseñados para el desarrollo de videojuegos y otros pensados para creación multimedia. Es muy común que una empresa cree su propio motor gráfico adaptado a los videojuegos que tiene pensado crear y realizar sobre del mismo varias actualizaciones a lo largo de los años

según los requisitos de la propia empresa. A continuación se recoge información de algunos de los motores gráficos más potentes de la actualidad, así como algunos motores válidos para el desarrollo de juegos, de los que se indicarán sus diferencias.

A continuación se enumeran 10 de los motores gráficos de juegos comerciales mejor considerados en la actualidad [15]:

Source Engine: Deslumbró en su día con juegos como *Half-Life 2* o *Team Fortress 2*, es un uno de los motores más antiguos de la época actual, ya que se comenzó a utilizar en el 2004. Destaca por buena iluminación dinámica, compensación de lag en conexiones, con un motor de físicos muy eficiente en conexiones de Internet y sobre todo, un código fuente de fácil acceso para *mods*.

Figura 91: Imagen del juego Team Fortress 2.

IW Engine: IW o Infinity Ward ha estado haciendo los juegos de la saga *Call Of Duty* desde los inicios de la serie. Con gráficos y animaciones geniales, así como una gran capacidad de desarrollo de la inteligencia artificial. Por otro lado los efectos son bastante buenos, con una gran iluminación y motor físico muy completo. A través de Internet es de los motores con mejor funcionamiento.

Figura 92: Imagen del futuro Call of Duty Modern Warfare 3, actualmente en desarrollo.

Id Tech 5: Motor del que fuera uno de los creadores del famoso *Doom*, utilizado en la continuación de la saga, como *Doom IV*. Se puede decir que esta generación de motores se debe en parte al trabajo de este motor. Con juegos como *Wolfenstein*.

Figura 93: Imagen del Doom IV.

RAGE: Este motor es el creador de uno de los juegos más importantes de la última década, el *GTA IV*. Capaz de manejar grandes mundos abiertos y muchos interiores, con una inteligencia artificial bastante compleja. No dispone de motor de físicos, pero se integra con el espectacular motor de físicos Euphoria [16].

Figura 94: Imagen del GTA V, que aparecerá este año 2011.

MT Framework: Motor creado para todos los juegos de Capcom [17] de esta generación. Geniales gráficos, flexibilidad de plataforma y libertad a la hora de desarrollar para diferentes tipos de juegos. Se han creado con este motor juegos como *Resident Evil 5* o *Lost Planet 2*.

Figura 95: Imagen de Lost Planet 2.

EGO Engine: Es raro ver en una lista de los mejores motores gráficos, un motor para juegos de carreras, pero este tiene unos gráficos increíbles. El motor de daños y el sistema de partículas es genial. Con él se han creado juegos tan dispares como *Race Driver: GRID* (carreras) u *Operation Flashpoint 2* (juego militar).

Figura 96: Imagen del juego Race Driver: GRID.

Geo-Mod Engine: Es un motor con gran equilibrio, que permite buenos gráficos, físicas e inteligencia artificial. Además tiene un sistema de destrucción que permite deforma cualquier estructura del mundo con físicas muy reales. Uno de los juegos más destacados: *Red Faction*.

Figura 97: Juego Red Faction.

Anvil Engine: Creador de juegos como *Assassin´s Creed* o *Prince of Persia*. Además de un motor bastante completo y que se comporta muy bien con escenarios abiertos, fue el primer motor en tener escenarios completamente escalables. En 2012 se espera el *Assassin´s Creed III*.

Figura 98: Imagen de Assassin´s Creed 2.

CryEngine: Este es el motor gráfico utilizado en uno de los juegos que más sorprendieron con su llegada a PC, se trata de *Crysis*. Motor muy avanzado, con grandes físicas, visuales, sonidos y animaciones.

Figura 99: Imagen del juego Crysis II.

Unreal Engine: Sin duda, uno de los motores gráficos que más está dando que hablar en los últimos años. A pesar de no ser el motor más potente, es un motor muy completo y en el cual se han apoyado muchas empresas para dar lugar a juegos tan reconocidos como *Gears of war*, *Mass Effect* o *Bioshock*.

Figura 100: Imagen del juego Gears of war 3.

Además, a principios del año pasado, la empresa encargada del mismo, Epic Games, liberó el programa de kit de desarrollo [18]. Con esto cualquiera puede probar a desarrollar juegos con esta gran plataforma y este gran motor gráfico. El kit de desarrollo puede utilizarse con fines comerciales, aunque en ese caso, es necesario el pago de licencia a Epic Games.

Una vez vistos los motores gráficos más potentes y famosos de la actualidad, se enumeran una serie de motores gráficos en su mayoría gratuitos, y que han estado entre los candidatos para realizar el proyecto junto con la opción de OGRE:

Crystal Space [19]: es un framework para el desarrollo de aplicaciones 3D. Se suele usar como motor de videojuegos, pero el framework es más general y puede ser usado para cualquier tipo de diseño 3D. Diseñado para ser totalmente portable, se ejecuta en Microsoft Windows, Linux y Mac OS X. Crystal Space es gratuito y se distribuye bajo la licencia *LGPL* [24].

Es compatible con *OpenGL, SDL, X11 y SVGALib*. Está programado en *C++* usando un diseño orientado a objetos.

Unreal Engine development kit [18]: Como ya se ha comentado, está disponible este gran motor gráfico para el desarrollo de juegos 3D. Casi todo el proceso de creación del juego se hace mediante un entorno gráfico y la programación queda reducida al máximo. Es gratuito para proyectos libres, pero para la distribución comercial hay que pagar a Epic Games.

Panda3D [20]: es una librería de subrutinas para el renderizado y desarrollo de juegos en 3D. Está basado en los lenguajes *C++ y Python*. Dispone de herramientas de creación de animaciones, tolerancia a errores, es rápido y se pueden crear juegos de propósito comercial. Es de licencia gratuita con algunas restricciones en ciertas librerías, y además dispone de bastante documentación. Hay empresas que utilizan este motor, como Disney.

XNA [21]: Se compone de una serie de librerías proporcionadas por Microsoft que facilita el desarrollo de videojuegos para las plataformas *PC, XBOX 360 y Zune*. Estas librerías están construidas sobre *.NET Framework 2.0* de Microsoft. Además dispone de un *IDE* de desarrollo llamado *Microsoft XNA Game Studio*.

OGRE 3D [1] (*Object-Oriented Graphics Rendering Engine*): Es un motor de renderizado para aplicaciones 3D, escrito en lenguaje *C++* orientado a objetos. El motor es libre bajo licencia *LGPL* [24] y tiene una comunidad muy activa de desarrolladores, sobretodo enfocado al desarrollo de videojuegos, aunque el motor en sí está enfocado a la renderización en general.

Figura 101: Torchlight, juego comercial creado con OGRE.

Muchas empresas han utilizado este motor, existen algunos juegos comerciales creados con este sistema y es multiplataforma.

Figura 102: Captura de Garshasp. Juego comercial creado con Ogre, disponible a partir de Mayo de 2011.

Ogre 3D ha sido el motor gráfico elegido, ya que en niveles generales es de los motores más equilibrados:

- Programar en C++ orientado a objetos es sencillo y a la vez eficiente.
- Pese a no ser un motor enfocado a juegos, existen gran cantidad de librerías enfocadas a este aspecto.
- No dispone de IDE, lo que hace que todo deba ser programado, pero las librerías encapsulan las llamadas más básicas evitando así la programación a bajo nivel, por lo que trabaja a un nivel medio y sencillo.
- Ogre funciona con librerías en su mayoría gratuitas y abiertas, al igual que todo su código fuente. Crear una aplicación comercial no supondrá gasto alguno.
- Es multiplataforma, se puede desarrollar para Windows, MAC o Unix y además se pueden crear aplicaciones tanto para los tres sistemas como para algunos sistemas móviles, por ejemplo iOS de Apple o Android.
- Tiene una gran comunidad de usuarios a su alrededor, por lo que hay gran cantidad de tutoriales, ayudas y los foros están siempre muy activos.

De entre los candidatos, se necesitaba usar una herramienta totalmente libre, para que en el caso de distribución y venta del juego en el futuro, así como en la propia adquisición de la herramienta, no fuera necesario realizar ningún pago a la empresa. Descartando por tanto Unreal Engine Development Kit.

La opción de Panda3D se descarta porque además de que no todas las librerías son libres, usa el lenguaje Python en algunas partes del desarrollo y podría dificultar el avance del proyecto, dado el desconocimiento por el grupo de desarrollo de dicho lenguaje.

XNA tiene gran documentación y es muy sencillo de utilizar, pero sólo permite realizar juegos para sistemas de Microsoft: PC con Windows, XBOX 360 o Zune y se pretende que el juego esté disponible para el mayor número de plataformas posibles.

El último y gran competidor de OGRE es Crystal Space, con el que comparte muchas características, pero que no dispone de tanta documentación y actividad en foros como dispone OGRE.

Figura 103: Logotipo de OGRE.

2.4 Blender y herramientas 3D

Todo juego en 3D precisa de la creación de objetos en tres dimensiones que forman la parte visual del juego. Terrenos, jugadores, elementos del escenario, etc. Para crear elementos 3D y exportarlos a los motores gráficos existen gran cantidad de aplicaciones, incluso algunos motores gráficos incluyen su propia utilidad para crear objetos 3D y usarlos en sus aplicaciones.

A continuación se indican algunas herramientas para el desarrollo 3D y se comentan sus características:

Autodesk Maya [22]: También conocido como *Maya*, es un programa informático dedicado al desarrollo de gráficos 3D, efectos especiales y animación. Se caracteriza por su potencia y capacidades de expansión y personalización de su interfaz y herramientas. *MEL* (Maya Embedded Language) es el código que forma el núcleo y gracias al cual se pueden crear scripts para personalizar el paquete. El software es de pago y válido para todas las plataformas.

Autodesk 3ds Max [23]: Programa de creación de gráficos y animación 3D, es uno de los programas de diseño más utilizados. Dispone de una sólida capacidad de edición y una

omnipresente arquitectura de plugins. Sólo es compatible con plataformas Windows. Además es un programa de pago.

Blender [8]: Es un programa multiplataforma, dedicado al modelado, animación y creación de gráficos tridimensionales. Es un software totalmente gratuito y de código fuente libre, con una interfaz gráfica de usuario muy característica, pero sencilla una vez que el usuario se acostumbra a ella.

Para el desarrollo de este proyecto se ha utilizado Blender, ya que además de ser muy potente, es totalmente gratuito, mientras que el resto de herramientas son de pago. Por otra parte exportar los objetos y animaciones a Ogre desde Blender es muy sencillo y las aplicaciones para esta tarea están muy depuradas, ya que se usan más que las aplicaciones para exportar desde otras herramientas de diseño 3D.

Capítulo 3:

Análisis, diseño, planificación e implementación

Este tercer capítulo, es el más importante de todo el proyecto, ya que se trata en él todo el grueso del proceso de creación del juego, desde la idea inicial hasta su puesta en marcha en un equipo de usuario.

El capítulo está diferenciado en 4 partes. La primera de ellas trata sobre el proceso de análisis realizado en el proyecto, definiendo los primeros pasos realizados para la creación del juego.

La segunda parte del capítulo se centra en el diseño realizado del juego, características sobre el mismo y detalles de cara a la implementación.

La tercera parte habla sobre la planificación y estudio de costes realizado sobre el proyecto, para hacer un estudio de viabilidad del mismo.

La cuarta y última parte del capítulo habla sobre el proceso de implementación del juego, detallando partes interesantes del código que pueden ser de utilidad para futuros desarrollos, así como para comprender mejor el funcionamiento del proyecto.

3.1 Análisis

El análisis es una parte esencial y fundamental en el proceso de desarrollo de un complemento software. Un buen análisis puede optimizar el resto del proceso de desarrollo y hacer que el ciclo de vida del software lleve un buen sentido sin provocar vueltas a fases anteriores del mismo.

El análisis intenta explicar y documentar qué necesita el usuario del juego, centrado en los principales elementos del mismo, una mirada superficial a lo que en el futuro será un complejo juego 3D.

En primer lugar se comentarán un poco las ideas iniciales del proyecto, como el tipo de juego a desarrollar, mecánicas y elementos a tener en cuenta.

A continuación se enumeran los requisitos marcados por los usuarios que debe de tener el juego, seguidos de unos diagramas de casos de uso del juego, que sirven también para dar al desarrollador una idea inicial de lo que el usuario desea.

Para finalizar el análisis se detallarán los requisitos software del juego, obtenidos en base a conocimientos del desarrollador y guiados por los requisitos y casos de uso marcados por los deseos de los usuarios.

3.1.1 Propuesta inicial

Como ya se comentó en el punto 1.2 Objetivos, la primera idea planteada para la realización del proyecto, fue desarrollar un juego de downhill skate en 3D. Este juego es muy similar a los juegos típicos de carreras de coches, sólo que en lugar de controlar un coche, el jugador controla un patinador, con algunas diferencias en los controles, como por ejemplo que el patín no tiene aceleración, sino que la aceleración la provoca la propia gravedad y la inclinación de la carretera que está bajando.

El juego funcionaría en 3D, con una cámara en tercera persona con posibilidad de cambiar a primera persona. Además se planteó la idea de utilizar menús para navegar antes de iniciar y al finalizar una partida.

Otra de las características iniciales del juego sería la posibilidad de tener varios patinadores y circuitos con propiedades diferentes con los que poder jugar, ya que este juego

es de un solo jugador y sin oponentes queda un poco corto, esta propiedad le añade jugabilidad. Además, se precisa la posibilidad de agregar más circuitos y patinadores en el futuro sin necesidad de agregar código y la opción de guardar los mejores tiempos de cada circuito.

Como cualquier juego 3D que se precie, el juego deberá tener sonidos y música de fondo configurable por el usuario desde el menú.

Antes de entrar en un análisis, diseño e implementación profundizado sobre el juego, y aprovechando los manuales de usuario, tutoriales y manuales de puesta en marcha creados anteriormente sobre OGRE [9] [10] [11], se implementó un pequeño proyecto en OGRE, utilizando algunas de las librerías que se requerirán para el proyecto, y que ha servido para poder hacer un análisis, diseño e implementación más directo sobre el proyecto real. Uno de los factores más importantes a la hora de desarrollar un juego con OGRE 3D, es que la plataforma OGRE está implementada en el lenguaje de programación C++ orientado a objetos, por lo que cualquier medio de análisis conocido enfocado a objetos es totalmente compatible con el proyecto.

A lo largo de este documento, se explican algunas características de OGRE, aunque se pueden obviar otras que vienen reflejadas en los documentos mencionados y adjuntos al proyecto [9] [10] [11].

3.1.2 Requisitos de Usuario

Tras recopilar los elementos de la idea inicial y los objetivos del proyecto, se puede realizar una extracción de los requisitos de usuario.

Los requisitos de usuario pretenden recoger qué es lo que quiere el cliente que realice la aplicación, identificando las funcionalidades y restricciones del sistema. A continuación se recogen los requisitos de usuario de capacidad, que describen lo que debe de hacer el sistema para alcanzar los objetivos propuestos y los requisitos de usuario de restricción, que establecen restricciones sobre el sistema.

Los requisitos de usuario para este proyecto están definidos por el siguiente conjunto de atributos:

- **Identificación**: Identifican cada atributo, con el tipo de atributo RUC (Requisito de Usuario de Capacidad) o RUR (Requisito de Usuario de Restricción), seguido de un número de requisito.
- **Necesidad**: Indica la prioridad y lo necesario que es el requisito para el juego. La necesidad puede ser Esencial, deseable u opcional. Los requisitos opcionales en este proyecto se han suprimido, incluyendo únicamente los requisitos esenciales y deseables.
- **Título**: Describe en una pequeña frase en qué consiste el requisito.
- **Comentario**: Describe con algo más de profundidad, si es necesario, el requisito.

A continuación se enumeran los requisitos de usuario de capacidad:

Identificador:	**RUC-01**	Necesidad:	Deseable
Título:	Ejecutable		
Descripción:	El juego debe tener un fichero ejecutable de fácil acceso para arrancar el juego.		

Tabla 1: RUC-01. Ejecutable.

Identificador:	**RUC-02**	Necesidad:	Esencial
Título:	Jugar		
Descripción:	El juego debe permitir jugar a un jugador utilizando el teclado y/o ratón.		

Tabla 2: RUC-02. Jugar.

Identificador:	**RUC-03**	Necesidad:	Esencial
Título:	Configurar opciones y menús		
Descripción:	El juego debe tener varios menús para poder configurar las opciones del juego, como opciones de sonidos. Además debe de tener menús para iniciar y finalidad el juego.		

Tabla 3: RUC-03. Configurar opciones y menús.

Identificador:	**RUC-04**	Necesidad:	Esencial
Título:	Guardar opciones		
Descripción:	Las opciones de configuración de sonido y volumen, deben de poder almacenarse para recuperarlos en futuros accesos al juego.		

Tabla 4: RUC-04. Guardar opciones.

Identificador:	**RUC-05**	Necesidad:	Esencial
Título:	Elegir circuito		
Descripción:	El jugador mediante un menú debe de poder elegir en qué circuito jugar.		

Tabla 5: RUC-05. Elegir circuito.

Identificador:	**RUC-06**	Necesidad:	Esencial
Título:	Elegir patinador		
Descripción:	El jugador mediante un menú debe de poder elegir con qué patinador jugar. Los patinadores deben de tener cualidades diferentes.		

Tabla 6: RUC-06. Elegir patinador.

Identificador:	**RUC-07**	Necesidad:	Deseable
Título:	Pre-visualizar el terreno		
Descripción:	Antes de empezar la partida sería conveniente que el jugador pueda pre-visualizar el circuito.		

Tabla 7: RUC-07. Pre-visualizar el terreno.

Identificador:	**RUC-08**	Necesidad:	Esencial
Título:	Controlar patinador		
Descripción:	Mediante el teclado, el jugador puede controlar al patinador. Los controles pueden ser giros, cambios de posición y derrapes.		

Tabla 8: RUC-08. Controlar patinador.

Identificador:	RUC-09	Necesidad:	Deseable
Título:	Cambiar de cámaras		
Descripción:	Sería deseable que durante el juego, el jugador pudiera cambiar la vista de la cámara entre tercera y primera persona.		

Tabla 9: RUC-09. Cambiar de cámaras.

Identificador:	RUC-10	Necesidad:	Deseable
Título:	Registrar tiempos		
Descripción:	Cuando un jugador inicie la prueba, deben tomarse los tiempos invertidos en la carrera y si supera el mejor tiempo del circuito, este debe guardarse.		

Tabla 10: RUC-10. Registrar tiempos.

Identificador:	RUC-11	Necesidad:	Deseable
Título:	Salir del juego		
Descripción:	Desde los menús se debe de poder salir del juego y volver al sistema.		

Tabla 11: RUC-11. Salir del juego.

A continuación se muestran los requisitos de usuarios de restricción:

Identificador:	RUR-01	Necesidad:	Esencial
Título:	Sistema operativo Windows		
Descripción:	El juego debe de funcionar en máquinas con sistema operativo Windows XP, sería deseable que también funcionara con Windows 7.		

Tabla 12: RUR-01. Sistema operativo Windows.

Identificador:	**RUR-02**	Necesidad:	Deseable
Título:	Movimientos		
Descripción:	Sería deseable aplicar ciertos modificadores a los movimientos de los patinadores, que añadan dificultad y realismo al juego.		

Tabla 13: RUR-02. Movimientos.

Identificador:	**RUR-03**	Necesidad:	Esencial
Título:	Controlar volumen de efectos		
Descripción:	Los efectos sonoros del juego se deben limitar con niveles de volumen, entre 0 (sin efectos) y 10 (nivel de sonido máximo).		

Tabla 14: RUR-03. Controlar volumen de efectos.

Identificador:	**RUR-04**	Necesidad:	Esencial
Título:	Controlar música de fondo		
Descripción:	La música de fondo del juego podrá habilitarse o deshabilitarse desde algún menú del juego.		

Tabla 15: RUR-04. Controlar la música de fondo.

Identificador:	**RUR-05**	Necesidad:	Deseable
Título:	Guardar tiempo con nombre		
Descripción:	Si se consigue el mejor tiempo de un circuito, debe de almacenarse el nombre del jugador y su tiempo. El nombre podrá tener como máximo 15 caracteres. Del tiempo se guardarán los minutos, segundos y milisegundos.		

Tabla 16: RUR-05. Guardar tiempo con nombre.

Identificador:	RUR-06	Necesidad:	Deseable
Título:	Guardar datos y atributos		
Descripción:	Es deseable poder guardar los datos de forma que se puedan recuperar fácilmente, además de guardar los atributos de circuitos y jugadores.		

Tabla 17: RUR-06. Guardar datos y atributos.

3.1.3 Casos de Uso

Con los requisitos de usuario recogidos y analizados, se realiza un diagrama de casos de uso, para ir progresando en la tarea de analizar y diseñar el sistema software.

Los casos de uso muestran las situaciones que se pueden dar en la aplicación entre usuario y sistema de forma muy generalizada. Donde se muestran los actores (usuarios del sistema) y los usos que realizan sobre el sistema.

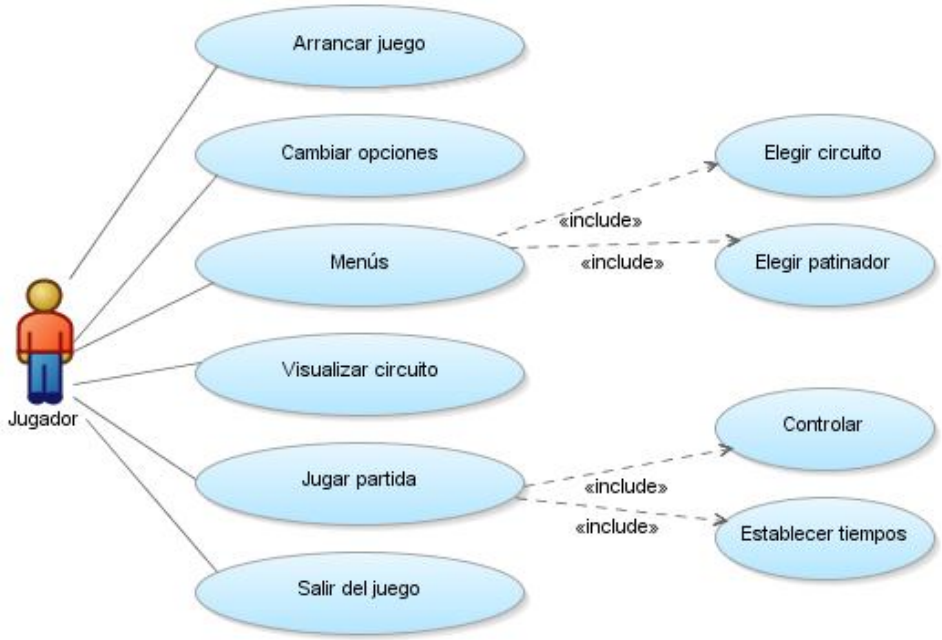

Figura 104: Diagrama de casos de uso.

A continuación se describe detalladamente cada uno de los casos de uso encontrados para el sistema, definiendo las siguientes propiedades para cada uno:

- **Identificador**: Identifica unitariamente cada caso de uso. La nomenclatura utilizada es CU-XX. Donde XX es el número de caso de uso actual.
- **Título**: Un pequeño título del caso de uso.
- **Actores**: Los actores participantes en el caso de uso.
- **Objetivo**: El objetivo a realizar por el caso de uso.
- **Precondiciones**: Condiciones para que pueda darse el caso de uso.
- **Post-condiciones**: Lo que sucederá en el estado del sistema tras aplicar el caso de uso.

Identificador:	**CU-01**
Título:	Arrancar Juego
Actores:	Jugador
Objetivos:	Arrancar el juego
Precondiciones:	Tener instalado el juego en un ordenador compatible y con un sistema operativo compatible
Post-condiciones:	Se carga el menú principal del juego

Tabla 18: CU-01. Arrancar el juego.

Identificador:	**CU-02**
Título:	Cambiar opciones
Actores:	Jugador
Objetivos:	Configurar las opciones de música y efectos a las necesidades del jugador
Precondiciones:	Haber arrancado el juego y pasar del menú principal al menú de opciones
Post-condiciones:	El sistema carga las opciones guardadas. Cambia las opciones y las guarda

Tabla 19: CU-02. Cambiar opciones.

Identificador:	**CU-03**
Título:	Menús
Actores:	Jugador
Objetivos:	Mostrar los distintos menús del juego
Precondiciones:	El juego debe de estar en funcionamiento
Post-condiciones:	Se carga y muestra el menú solicitado

Tabla 20: CU-03. Menús.

Identificador:	**CU-04**
Título:	Elegir circuito
Actores:	Jugador
Objetivos:	Permite al jugador elegir un circuito de entre la lista de disponibles
Precondiciones:	Tener cargado el menú donde se pide al jugador que elija circuito para jugar
Post-condiciones:	Se carga la lista de circuitos disponibles para jugar

Tabla 21: CU-04. Elegir circuito.

Identificador:	**CU-05**
Título:	Elegir patinador
Actores:	Jugador
Objetivos:	Arrancar el juego
Precondiciones:	Permite al jugador elegir un patinador de entre la lista de disponibles
Post-condiciones:	Se carga la lista de patinadores disponibles para utilizar

Tabla 22: CU-05. Elegir patinador.

Identificador:	**CU-06**
Título:	Visualizar circuito
Actores:	Jugador
Objetivos:	Permitir al jugador ver el circuito sobre el que va a jugar la siguiente partida, con una cámara que puede moverse y rotarse por todo el terreno de juego
Precondiciones:	Haber elegido circuito y patinador, y haber pulsado sobre el botón de jugar
Post-condiciones:	Se carga y muestra el circuito en una vista aérea

Tabla 23: CU-06. Visualizar circuito.

Identificador:	**CU-07**
Título:	Jugar partida
Actores:	Jugador
Objetivos:	El grueso de la aplicación, donde el jugador mediante el teclado mueve al skater en el circuito, girando y derrapando para llegar al final del mismo
Precondiciones:	Después de visualizar el circuito, el jugador pulsa una tecla para iniciar el juego
Post-condiciones:	Se carga al skater y el circuito para poder interactuar con ambos

Tabla 24: CU-07. Jugar partida.

Identificador:	**CU-08**
Título:	Controlar
Actores:	Jugador
Objetivos:	En el modo de juego, el jugador puede controlar al skater a través de pulsaciones de teclado, también puede cambiar de cámara
Precondiciones:	Circuito y skater cargados y listos para jugar
Post-condiciones:	El jugador juega al juego

Tabla 25: CU-08. Controlar.

Identificador:	**CU-09**
Título:	Establecer tiempos
Actores:	Jugador
Objetivos:	Cuando el jugador finaliza la partida, se muestra su tiempo y si supera el tiempo de record de la pista, puede poner su nombre, para guardar el tiempo en el juego
Precondiciones:	Finalizar una partida, ya sea por caída, salir del juego, terminar sin record o terminar con record
Post-condiciones:	Se muestra la pantalla de tiempos y la posibilidad de poner el record. Una vez establecido, se vuelve al menú principal del juego

Tabla 26: CU-09. Establecer tiempos.

Identificador:	**CU-10**
Título:	Salir del Juego
Actores:	Jugador
Objetivos:	Cerrar el juego
Precondiciones:	Estar en el menú principal
Post-condiciones:	Se cierra completamente el juego

Tabla 27: CU-10. Salir del juego.

3.1.4 Requisitos del Software

Una vez estudiados los requisitos de usuario y los casos de uso del sistema, la siguiente acción a realizar es estudiar y declarar los requisitos del software.

Los requisitos del software definen requisitos que tendrá el futuro software que se está diseñando. Hay varios tipos de requisitos software, que se comentan a continuación. En este proyecto, no se han recogido todos los tipos de requisitos software que existen, sino que sólo se han definido los necesarios.

Para cada uno de los requisitos se documenta con los siguientes campos:

- **Identificador**: Al igual que en resto de casos, identifica unitariamente al requisito. Tiene unos caracteres que indican que se trata de un requisito software, con uno o más caracteres que identifican el tipo de requisito, seguido de dos dígitos para enumerarlo.
- **Título**: Un pequeño título que resume el requisito.
- **Fuente**: Se comentan los casos de uso y/o requisitos de usuario por los que se ha llegado a este requisito. Si el requisito aparece por consideración del analista, se identifica con la palabra ANA.
- **Necesidad**: Se indica la necesidad de que se aplique el requisito, esta puede ser esencial o deseable.
- **Descripción**: Se describe en qué consiste el requisito software.

Requisitos funcionales. Estos requisitos especifican qué tiene que hacer la aplicación. Suelen obtenerse a partir de los requisitos de usuario de capacidad.

Identificador:	**RSF-01**	Necesidad:	Deseable
Título:	Instalador del juego		
Fuente:	ANA		
Descripción:	Es deseable empaquetar la versión final del juego con un instalador para que el juego se instale de forma sencilla en cualquier sistema		

Tabla 28: RSF-01. Instalador del juego.

Identificador:	**RSF-02**	Necesidad:	Esencial
Título:	Arrancar el juego		
Fuente:	RUC-01, RUR-01, CU-01		
Descripción:	El juego debe de disponer de un fichero ejecutable, para iniciarse		

Tabla 29: RSF-02. Arrancar el juego.

Identificador:	**RSF-03**	Necesidad:	Esencial
Título:	Mostrar menú principal del juego		
Fuente:	RUC-03, CU-03		
Descripción:	Al entrar en el juego, se debe de mostrar un menú principal con las opciones de elegir circuito y jugador, cambiar opciones o salir del juego		

Tabla 30: RSF-03. Mostrar menú principal del juego.

Identificador:	**RSF-04**	Necesidad:	Esencial
Título:	Mostrar menú de opciones		
Fuente:	RUC-03, RUC-04, CU-02, CU-03		
Descripción:	Muestra un menú con las opciones disponibles		

Tabla 31: RSF-04. Mostrar menú de opciones.

Identificador:	**RSF-05**	Necesidad:	Esencial
Título:	Ajustar música		
Fuente:	RUC-03, RUR-04		
Descripción:	El menú de opciones permite activar o desactivar la música de fondo del juego. El volumen de esta música irá acorde con el nivel establecido en el volumen de los efectos. Si los efectos están a nivel 0 y la música está activada, esta sonará a un nivel de intensidad medio		

Tabla 32: RSF-05. Ajustar música.

Identificador:	**RSF-06**	Necesidad:	Esencial
Título:	Ajustar volumen efectos		
Fuente:	RUC-04, RUR-03		
Descripción:	Un deslizador o sistema similar del menú de opciones permitirá establecer el volumen de los efectos del juego, entre 0 y 10		

Tabla 33: RSF-06. Ajustar volumen efectos.

Identificador:	**RSF-07**	Necesidad:	Esencial
Título:	Mostrar menú de circuitos y patinadores		
Fuente:	RUC-03, CU-03		
Descripción:	Tras salir del menú principal para jugar, aparecerá un menú en el que aparecen los distintos circuitos y skaters disponibles para jugar una partida		

Tabla 34: RSF-07. Mostrar menú de circuitos y patinadores.

Identificador:	**RSF-08**	Necesidad:	Deseable
Título:	Elegir patinador		
Fuente:	RUC-06, CU-05		
Descripción:	El menú deberá tener algún tipo de sistema que permita elegir entre los patinadores disponibles en el juego, mostrando algún texto identificativo y descripción		

Tabla 35: RSF-08. Elegir patinador.

Identificador:	**RSF-09**	Necesidad:	Deseable
Título:	Elegir circuito		
Fuente:	RUC-05, CU-06		
Descripción:	El menú deberá tener algún tipo de sistema que permita elegir entre los circuitos disponibles en el juego, mostrando algún texto identificativo y descripción		

Tabla 36: RSF-09. Elegir circuito.

Identificador:	**RSF-10**	Necesidad:	Esencial
Título:	Pre-visualizar el terreno		
Fuente:	RUC-07, CU-06		
Descripción:	Se debe permitir observar mediante una cámara móvil todo el terreno de juego, para que el jugador pueda analizarlo. Esta cámara deberá poder rotarse con el ratón y moverse por todo el terreno con el teclado		

Tabla 37: RSF-10. Pre-visualizar el terreno.

Identificador:	**RSF-11**	Necesidad:	Deseable
Título:	Mostrar el modo de juego		
Fuente:	ANA, RUC-03, CU-07		
Descripción:	Sería deseable que una vez finalizada la pre-visualización del terreno de juego, se pasara a la pantalla de juego, con una pequeña cuenta atrás, antes de que el patinador empiece a moverse por el terreno		

Tabla 38: RSF-11. Mostrar el modo de juego.

Identificador:	**RSF-12**	Necesidad:	Deseable
Título:	Pantalla de juego		
Fuente:	ANA, RUC-02, CU-07		
Descripción:	En la pantalla de juego se debe mostrar al patinador y el terreno. Sería recomendable poner unos sistemas mostrando el tiempo de juego y la velocidad a la que va el skater		

Tabla 39: RSF-12. Pantalla de juego.

Identificador:	**RSF-13**	Necesidad:	Esencial
Título:	Controlar el patinador		
Fuente:	RUC-02, RUC-08, RUR-02, CU-07, CU-08		
Descripción:	El jugador puede controlar al skater con el teclado. Los movimientos de los skaters están definidos por unas propiedades, que hacen que varíe el control entre distintos patinadores, como el ángulo de giro o la aceleración. Un skater puede acelerar al inicio con el pie, estar en pie o agachado, girar y frenar tanto agachado como en pie y realizar un derrape cuando está en pie		

Tabla 40: RSF-13. Controlar el patinador.

Identificador:	**RSF-14**	Necesidad:	Esencial
Título:	Cambio de cámaras		
Fuente:	RUC-09, CU-07, CU-08		
Descripción:	Pulsando un botón del teclado, se puede intercambiar la cámara de juego entre tercera y primera persona		

Tabla 41: RSF-14. Cambio de cámaras.

Identificador:	**RSF-15**	Necesidad:	Deseable
Título:	Pantalla de salida o caída		
Fuente:	RUC-09, RUR-05, CU-09		
Descripción:	Cuando el jugador abandona la partida, bien sea porque quiere salir de la misma, pulsando una tecla de salida o bien porque el skater se ha salido del circuito, se muestra un menú indicando la situación, además se indican el mejor tiempo y jugador del circuito, así como la opción de volver al menú principal		

Tabla 42: RSF-15. Pantalla de salida o caída.

Identificador:	**RSF-16**	Necesidad:	Deseable
Título:	Pantalla de fin sin record		
Fuente:	RUC-09, RUR-05, CU-09		
Descripción:	Si el jugador finaliza la pantalla sin salirse del circuito y no ha conseguido el mejor tiempo del circuito, se muestra una pantalla indicado esta situación, su tiempo logrado, y el tiempo y jugador que poseen el mejor tiempo del circuito. Además tendrá la opción de volver al menú principal		

Tabla 43: RSF-16. Pantalla de fin sin record.

Identificador:	**RSF-17**	Necesidad:	Deseable
Título:	Pantalla de fin con record		
Fuente:	RUC-09, RUR-05, CU-09		
Descripción:	Si el jugador finaliza la pantalla sin salirse del circuito y consigue el record del circuito, se muestra una pantalla indicando esta situación, pidiendo que escriba su nombre para almacenar el nombre y el tiempo como nuevo record del circuito, además dispondrá de una opción para volver al menú principal		

Tabla 44: RSF-1. Pantalla de fin con record.

Identificador:	**RSF-18**	Necesidad:	Esencial
Título:	Salir del juego		
Fuente:	RUC-11, CU-10		
Descripción:	En el menú principal, deberá haber una opción para salir del juego		

Tabla 45: RSF-18. Salir del juego.

Identificador:	**RSF-19**	Necesidad:	Deseable
Título:	Desinstalador del juego		
Fuente:	ANA		
Descripción:	Sería deseable crear una herramienta para poder desinstalar el juego de forma sencilla para el usuario		

Tabla 46: RSF-19. Desinstalador del juego.

Requisitos del software de Rendimiento: Estos requisitos definen valores para variables de rendimiento, como velocidad de procesado y ejecución del sistema software:

Identificador:	**RSR-01**	Necesidad:	Deseable
Título:	Carga del juego		
Fuente:	ANA		
Descripción:	Es deseable que el juego tarde poco en cargar, desde que se lanza la ejecución hasta que aparece la primera pantalla de menú del juego. También es deseable que los tiempos de ejecución dentro del juego no sean muy largos		

Tabla 47: RSR-01. Carga del juego.

Identificador:	**RSR-02**	Necesidad:	Deseable
Título:	Carga de mapas		
Fuente:	ANA		
Descripción:	La carga de los terrenos de juego, suele ser la tarea que más tiempo precisa, por lo que se recomienda bajar los tiempos de carga con sistemas auxiliares como puede ser el uso de cachés		

Tabla 48: RSR-02. Carga de mapas.

Requisitos de Interfaz: Especifican el hardware y software con el que el sistema deberá interactuar:

Identificador:	**RSI-01**	Necesidad:	Esencial
Título:	Imágenes de circuitos y patinadores		
Fuente:	ANA		
Descripción:	Las imágenes para mostrar a los patinadores y circuitos deberán tener extensión jpg y tener unas dimensiones de 300 x 225 píxeles		

Tabla 49: RSI-01. Imágenes de circuitos y patinadores.

Identificador:	**RSI-02**	Necesidad:	Esencial
Título:	Xml		
Fuente:	ANA, RUR-05, RUR-06		
Descripción:	Se definirán unos ficheros xml para las opciones del juego (música y volumen de efectos), circuitos (nombre, descripción, capas, records) y patinadores (objetos 3d, atributos, nombre y descripción). Con estos ficheros se pueden definir de forma simple las propiedades de circuitos y patinadores, así como las opciones generales del juego, para luego cargarlos y guardarlos. Estos xml tendrán la extensión .dhsk		

Tabla 50: RSI-02. xml.

Identificador:	**RSI-03**	Necesidad:	Esencial
Título:	Objetos 3D		
Fuente:	ANA		
Descripción:	Los objetos 3D deberán seguir el formato recomendado por OGRE, definidos como objetos de extensión .mesh, con scripts para las texturas y materiales con extensión .material		

Tabla 51: RSI-03. Objetos 3D.

Identificador:	**RSI-04**	Necesidad:	Esencial
Título:	Formato de mapas		
Fuente:	ANA		
Descripción:	Los formatos de mapas: tanto sus mapas de alturas, como sus máscaras de texturas, se recomiendan que tengan un tamaño de 1024 por 1024 píxeles y extensión .png. Las texturas utilizadas en los circuitos pueden ser de tamaños y extensión variados		

Tabla 52: RSI-04. Formato de mapas.

Requisitos de Recursos: Especifican los recursos físicos necesarios para la ejecución de la aplicación indicando los límites máximos admisibles para cada recurso.

Identificador:	**RSRec-01**	Necesidad:	Deseable
Título:	Entorno de desarrollo		
Fuente:	ANA		
Descripción:	Para el entorno de desarrollo, es necesario un ordenador muy potente, ya que muchas de las tareas en ejecución toman muchos más datos que las mismas en un entorno de producción, para poder obtener más datos sobre el desarrollo del sistema. Se recomienda utilizar un sistema similar al siguiente: • Sistema Operativo: Windows XP con Visual C++ • Memoria RAM: 4 GB • Capacidad: Al menos 20 GB de capacidad en disco duro • Tarjeta gráfica: De al menos 512 MB de memoria		

Tabla 53: RSRec-01. Entorno de desarrollo.

Identificador:	RSRec-02	Necesidad:	Deseable
Título:	Entorno de producción		
Fuente:	ANA		
Descripción:	Para poder ejecutar el juego y jugar cómodamente, se recomienda usar un equipo igual o superior al siguiente: • Sistema operativo: Windows XP o Windows 7 • Memoria RAM: 2 GB • Capacidad: 200 MB de capacidad en el disco duro • Tarjeta gráfica: De al menos 256 MB de memoria dedicada		

Tabla 54: RSRec-02. Entorno de producción.

Requisitos de software de seguridad: Aseguran el sistema contra amenazas, así como la confidencialidad e integridad de los datos almacenados en el mismo.

Identificador:	RSS-01	Necesidad:	Deseable
Título:	Seguridad en los xml		
Fuente:	ANA, RUR-05, RUR-06		
Descripción:	Los ficheros xml con la información de opciones del juego, circuitos y patinadores, pueden localizarse con más o menos facilidad dentro del sistema de ficheros del juego, por ello se precisa un sistema mediante códigos HASH de diseño propio que asegure la inalteración de dichos xml por cualquier usuario		

Tabla 55: RSS-01. Seguridad en los xml.

3.1.5 Diagrama de Actividad

Una vez especificados los requisitos del software y, junto con la información de los requisitos de usuario y con de los casos de uso, se puede crear un diagrama de actividad del sistema, que indica el comportamiento del sistema desde que se inicializa hasta que se cierra. Este diagrama ayudará a comprender mejor los estados por los que debe de pasar el juego, recogiendo las transiciones y eventos del mismo.

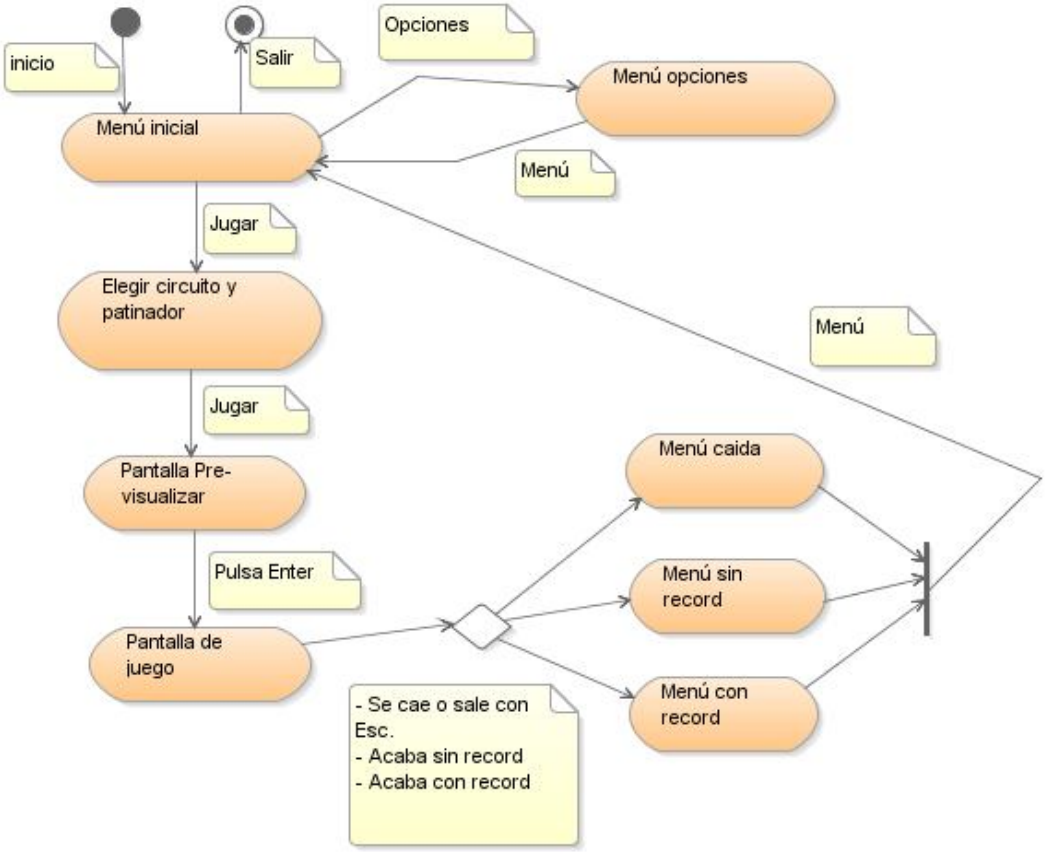

Figura 105: Diagrama de Actividad

3.1.6 Diagramas de Secuencia

Los diagramas de secuencia, permiten representar cómo se comunican entre sí varios objetos para la realización de los casos de uso. Estos diagramas contienen detalles de implementación del escenario y ayudarán mucho a la hora de la definición de las clases a utilizar en el proyecto y la funcionalidad de cada una de ellas.

Para realizar este diagrama, se han tenido en cuenta algunas de las propiedades de OGRE que se detallarán en las fases de diseño e implementación, para obtener así un diagrama acorde con el resultado final.

En un proceso normal, se debe de hacer un diagrama de secuencia por cada uno de los casos de uso existentes en el proyecto, pero en este proyecto y para simplificar el número de diagramas, se han creado varios diagramas de secuencia que pueden representar uno o más casos de uso, que se consideran repetidos o similares. En cada caso, se indicarán los casos de uso cubiertos por cada uno de los diagramas.

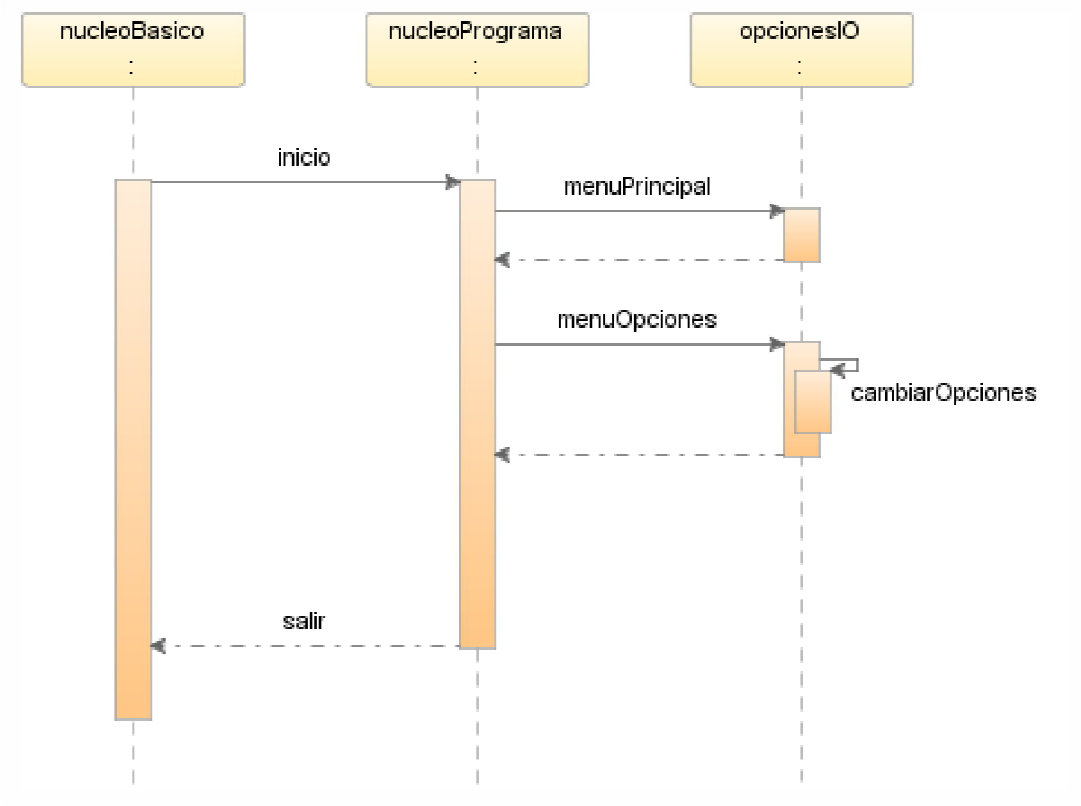

Figura 106: Diagrama de secuencia de opciones

Este diagrama de secuencia define los requisitos CU-01 (Iniciar Juego), CU-02 (Cambiar opciones), CU-03 (Menús) y CU-10 (Salir del Juego).

El juego tiene una clase básica llamada nucleoBasico que inicializa el sistema y librerías auxiliares, crea el escenario principal, las cámaras principales y genera el bucle de renderización de la aplicación, será el encargado de iniciar todo el proceso del puesta en marcha y detección del proyecto.

La clase nucleoPrograma, es una extensión de la primera clase, con opciones menos generales y más cercanas al propio proyecto que se desarrolla. Esta actúa de centro de operaciones llamando al resto de clases, realizando cálculos y dando órdenes, como en este caso la opción de cargar el menú principal y el menú de opciones.

La clase opcionesIO es una clase que gestiona los métodos de entrada/salida del teclado y ratón y se encarga de generar los menús de usuario, así como de controlar las operaciones realizadas por el mismo dentro de la aplicación.

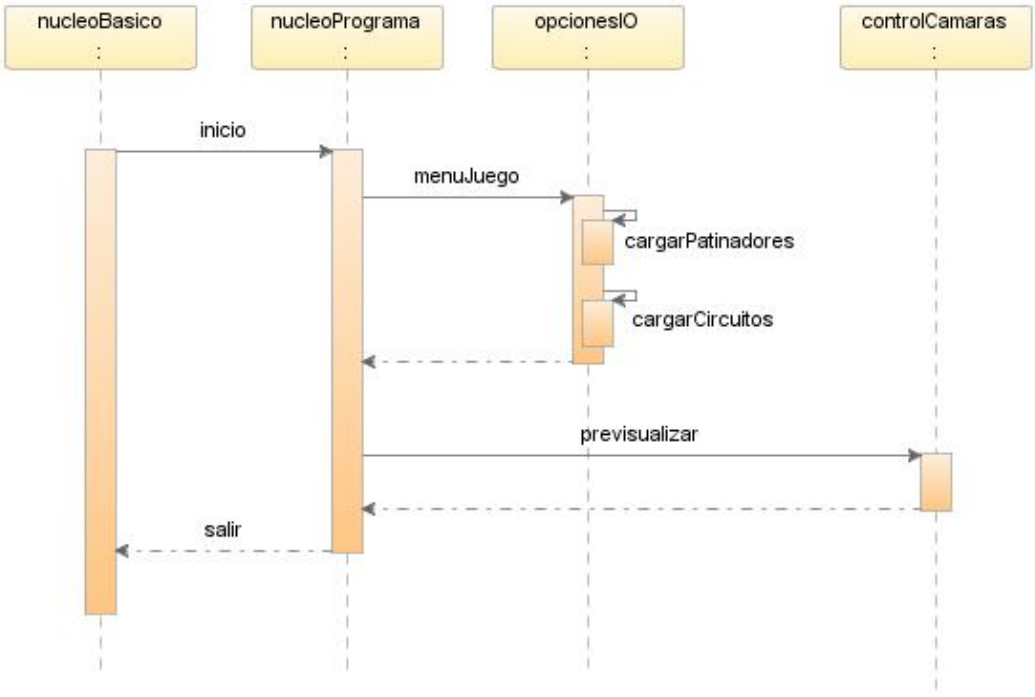

Figura 107: Diagrama de secuencia de carga de jugadores y circuitos y pre-visualización

Este diagrama representa los casos de uso CU-03 (Menús), CU-04 (Elegir circuito), CU-05 (Elegir patinador), CU-06 (Visualizar Circuito).

Igual que en el caso anterior, el programa principal (nucleoPrograma), llama al controlesIO para que cargue la pantalla de menú con las opciones de patinadores y circuitos. Una vez cargados y elegidos, se llama a pre-visualizar con la clase controlCamaras. Esta clase comprueba el estado de las cámaras y lo actualiza, además se recogen los datos de teclado y ratón del usuario para mover la cámara por todo el circuito.

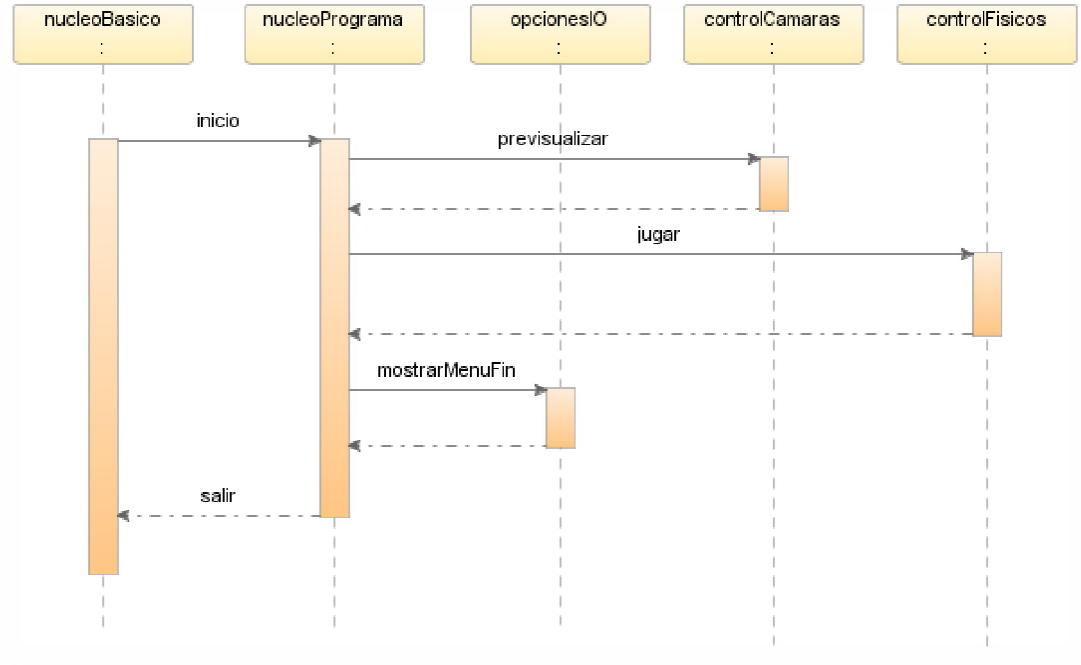

Figura 108: Diagrama de secuencia de Juego

Este diagrama representa los casos de uso CU-06 (Visualizar circuito), CU-07 (Jugar Partida), CU-08 (Controlar), CU-09 (Establecer tiempos) y CU-10 (Salir del juego).

Finalizado el pre-visualizado del juego, desde el componente inicial se llama a jugar, y se pasa a una clase de control de físicos, donde se analizan y actualizan las posiciones del patinador y se captura la entrada de datos de teclado del jugador para controlar al patinador. Una vez que se llega a uno de los tipos de finalización: el jugador abandona por caída o por voluntad propia, el jugador finaliza la pantalla sin record o el jugador finaliza la pantalla con record, se muestra un menú con los datos necesarios y se vuelve el control al componente principal.

3.2 Diseño

Una vez realizado todo el análisis del proyecto, se pasa a la fase de diseño. En esta etapa, se evalúa de forma más exhaustiva todos los datos tomados hasta el momento y se empiezan a detallar características del diseño del proyecto, que aporten toda la información necesaria a la fase de implementación, para que esta quede casi completamente definida, sin dar lugar a dudas que puedan provocar un replanteamiento del análisis, diseño y por tanto un cambio en la planificación del proyecto y su tiempo de producción.

Durante la etapa del diseño se han realizado ciertas tareas que se comentan a continuación, en un primer término se muestra un pequeño modelo de arquitectura del proyecto para organizar las futuras clases que compondrán el proyecto.

Tras el modelo, se indican algunos patrones para el diseño de menús y pantallas de juego, que dan una idea gráfica de lo que será en el futuro el proyecto.

Profundizando un poco en el modelo de arquitectura comentado para las clases, también se muestra una pequeña estructura de cómo almacenar ciertos recursos de componentes del programa, cómo pueden ser las carpetas para guardar los objetos 3D y sus componentes, según se especifica en OGRE.

También se muestra un diagrama de clases de lo que sería un Framework para el desarrollo de juegos en OGRE genérico y un diagrama de clases completo para este proyecto en concreto, incluyendo clases de creación necesaria.

Llegando al final de la etapa de diseño, se comentan características básicas para definir a un skater: los objetos que lo forman, métodos de control y características que lo definen.

De igual forma se habla sobre las características que definen un circuito.

Para finalizar el diseño, se comentará el mecanismo de seguridad empleado en el sistema para proteger los ficheros xml.

Se ha de tener en cuenta que llegados a este punto del análisis, resulta necesario tener ciertos conocimientos en los mecanismos de procesamiento de OGRE 3D, para enfocar el diseño del proyecto a la arquitectura OGRE, así como tener una idea clara de las librerías necesarias y sus métodos de funcionamiento para realizar el diseño con soltura de cara a la implementación final. Un detalle de OGRE 3D y su forma de comportarse, puede encontrarse en los documentos escritos por el autor de este proyecto llamados: Manual de Ogre 3D [9], que contiene un manual de las propiedades de OGRE, Tutoriales básicos [9], que es un documento con algunos pequeños proyectos de OGRE probando distintas características básicas del paquete y por último el documento Entorno de Desarrollo de Ogre 3D [11], que explica cómo montar un entorno de desarrollo para OGRE y cómo añadir librerías del terceros al mismo.

3.2.1 Arquitectura

A continuación se presenta un modelo de la arquitectura del software del proyecto. Este modelo divide al proyecto en distintos componentes y subsistemas. Este diagrama es una aproximación de lo que será el futuro diagrama de clases.

Figura 109: Diagrama de Arquitectura

El sistema se puede englobar en tres componentes principales. El componente principal es el núcleo, donde se gestiona el proceso de inicialización del sistema al completo, incluyendo el resto de componentes. También es el encargado de organizar todo el proceso y ciclo de vida del sistema y llamar al bucle de renderizado de OGRE.

El segundo componente es el componente de Entrada/Salida, encargado de gestionar las iteraciones con el usuario, tales como la entrada por teclado y ratón, y las visualizaciones de los menús. Los objetos que forman parte de este componente suelen ser FrameListeners de OGRE, que se registran en el bucle de renderizado de OGRE y esperan a que ocurran acciones tales como que el usuario pulse una tecla, o se actualizan con el paso de cada **frame** del sistema.

El tercer componente está compuesto por las clases que gestionan el proceso de carga y actualización de los datos del sistema, como puede ser la lectura y actualización de los ficheros xml del juego.

3.2.2 Pantallas

En este apartado se muestran algunos diseños de las pantallas creados, para dar una idea del proceso de una partida en el proyecto, además se definen los menús que se mostrarán en el proyecto, con sus notaciones y componentes, y que servirán de plantilla para los menús definitivos.

Figura 110: Plantilla Menú principal.

Figura 111: Plantilla Menú opciones.

Figura 112: Plantilla Menú selección de patinador y circuito.

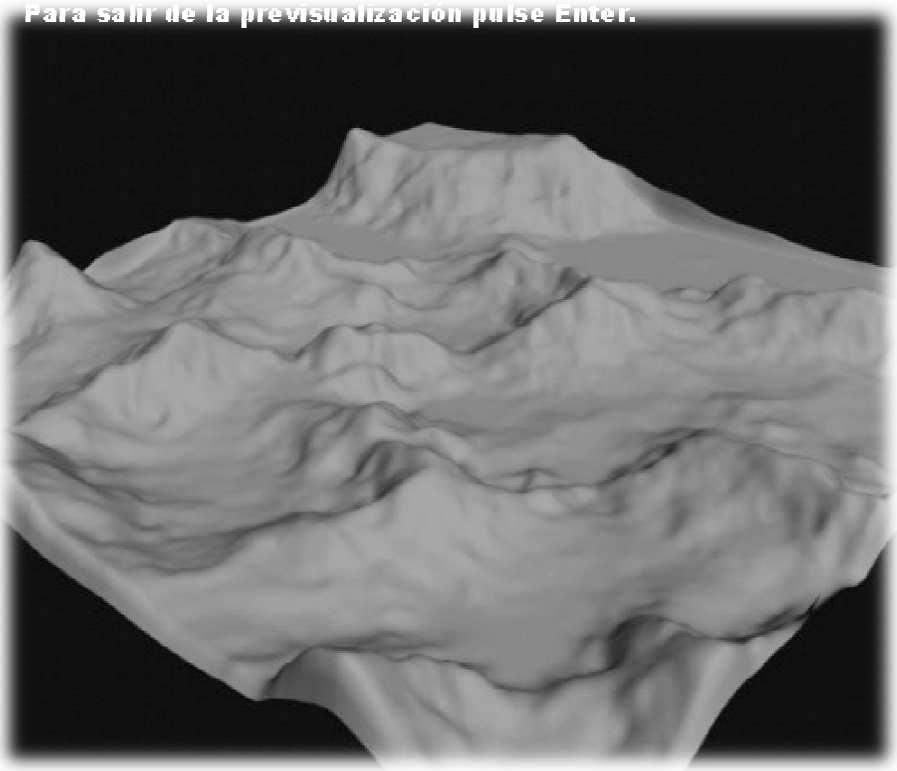

Figura 113: Plantilla de la pantalla de pre-visualización del circuito.

Figura 114: Plantilla similar al juego (Imagen de Tony Hawk 2).

Figura 115: Plantilla Menú de finalización por caída.

Figura 116: Plantilla Menú de finalización normal.

Figura 117: Plantilla Menú de finalización con record.

3.2.3 Ogre y librerías

En este punto del diseño, es importante conocer las capacidades de OGRE y sus necesidades. Estas necesidades, hay que saber cómo cubrirlas en base a librerías de terceros.

Como se ha comentado en la introducción, OGRE no es una librería específica para el desarrollo de juegos, sino que simplemente se centra en el renderizado en tiempo real de objetos y mundos 3D. Pero gracias a la gran comunidad de desarrolladores y sobretodo la gran cantidad de ellos centrados en el uso de OGRE para el desarrollo de juegos, es bastante sencillo encontrar soluciones propias del desarrollo para juegos en foros y tutoriales.

Además, muchos de estos usuarios han empaquetado sus soluciones específicas en librerías disponibles para todo el mundo, por lo que se consigue que estén disponibles un gran número de librerías que cubren por completo las necesidades de cualquier desarrollador de juegos.

Para este proyecto se han utilizado las siguientes librerías:

- **MyGUI** [5]: Es una librería que sirve para hacer interfaces gráficos de usuarios como menús, botones, etc. de una forma sencilla y eficaz. Además dispone de un método, mediante el cual por xml se pueden definir capas o pantallas con datos, y cargarlos de rápidamente en el programa. Este método es muy eficaz para la creación de menús.

Figura 118: Juego comercial Zero Gear, con OGRE y MyGUI.

- **cAudio** [6]: Esta librería es la encargada de reproducir sonidos. Tiene gran cantidad de formas de reproducción, como bucles y varios sonidos a la vez.

Dispone de sonidos en 3D, que hacen que se pueda variar el origen del sonido y su fuerza, para que la sensación de realidad en el juego sea mayor.

- **Camera Control System** [4]: También conocida como CCS, es una librería que implementa cámaras especiales y muy útiles para juegos. Se pueden crear varias cámaras e intercambiarlas de forma sencilla, además existen modos de cámaras creados como cámaras que siguen a un objeto 3D a una determinada distancia, cámaras libres o cámaras adjuntas a un objeto 3D en primera persona.
- **Newton** [3]: Es una librería en C++ de control de físicos, junto con ella hay que usar la librería OgreNewton, que sirve de enlace entre Ogre y Newton. Esta librería crea un mundo físico y cuerpos físicos sobre los que pueden actuar fuerzas y colisionar con otros cuerpos del mundo.

Figura 119: Disc Drivn´, juego comercial para iPhone con Newton.

3.2.4 Carpetas del proyecto

En este apartado se habla sobre la organización de las carpetas de recursos que tiene el proyecto, algunas de ellas siguiendo la estructura recomendada por OGRE y otras propias. Es importante saber que las carpetas con elementos que el sistema puede usar se deben registrar en un fichero de configuración de OGRE llamado *resources.cfg*.

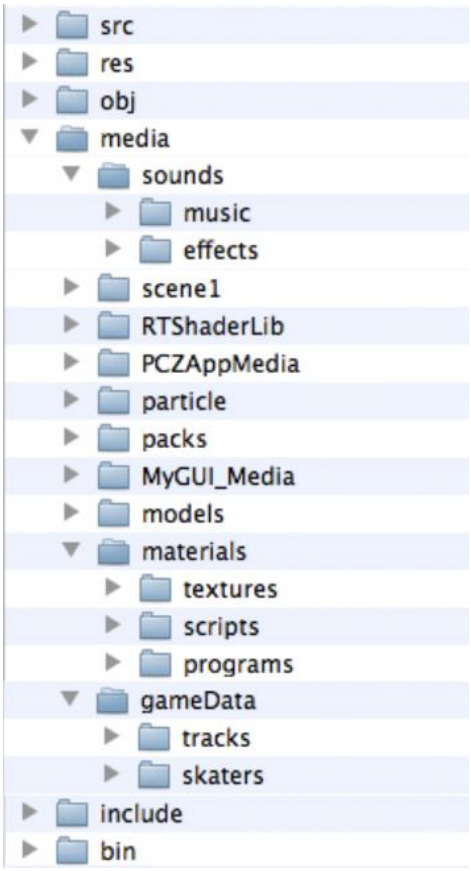

Figura 120: Carpetas del proyecto.

- En la **carpeta de proyecto** se han incluido los ficheros de configuración de OGRE, logs del sistema y los ficheros de implementación de las clases.
- En la carpeta **include**, se han incluido los ficheros de definición de clases del proyecto.
- La carpeta **media**, contiene todos los recursos del proyecto, a los que accederá la aplicación. En la carpeta **gameData** junto con sus subcarpetas **tracks** y **skaters**, se encuentran los ficheros xml de definición de opciones de juego, definición de circuitos y definición de patinadores.
- En la carpeta **materials** se puede encontrar lo referente a los materiales y texturas utilizados en el juego, clasificados según su tipo en las carpetas disponibles: **programs**, **scripts** o **textures**.
- En la carpeta **models**, se encuentran todos los modelos 3D que usa la aplicación.
- La carpeta **MyGUI_media** contiene toda la información relativa a los menús del juego, incluyendo imágenes de fondo y ficheros de definición de capa.

- En la carpeta **packs**, se localizan ficheros empaquetados con información de texturas.
- La carpeta **scene1** es una carpeta con información general y particular del juego, como son imágenes de terrenos y capas.
- Finalmente la carpeta **sounds**, con las subcarpetas **music** y **effects** tienen los ficheros de música y efectos utilizados por el juego.
- El resto de carpetas no mencionadas, no se utilizan en el proyecto.

Esta ha sido la estructura utilizada en el proyecto.

3.2.5 Clases (Framework)

Como se ha comentado anteriormente, en una primera aproximación a un programa realizado con OGRE y como ejemplo para la documentación generada, se creó un pequeño proyecto que utilizaba varias librerías del proyecto final y que se basaba en clases de ayuda de OGRE para realizar ejemplos de forma sencilla y rápida.

Se comprobó que estas clases son útiles, pero quedan un poco cortas y demasiado abstractas como para obtener un control pleno de la aplicación. Es por ello que surge la necesidad de implementarse un núcleo del programa propio. Tras esta idea, se persigue hacer un núcleo propio y generalizado, del que puedan heredar los núcleos personalizados de cada proyecto y pueda ser reutilizado para futuros juegos, o al menos ser un punto de partida estable para los mismos.

Con este propósito surge el siguiente Framework de clases, que aunque como se demostrará en el apartado de implementación, no es tan independiente del proyecto como en un principio se pretendía, sí puede usarse modificándolo ligeramente como punto de partida para nuevos juegos.

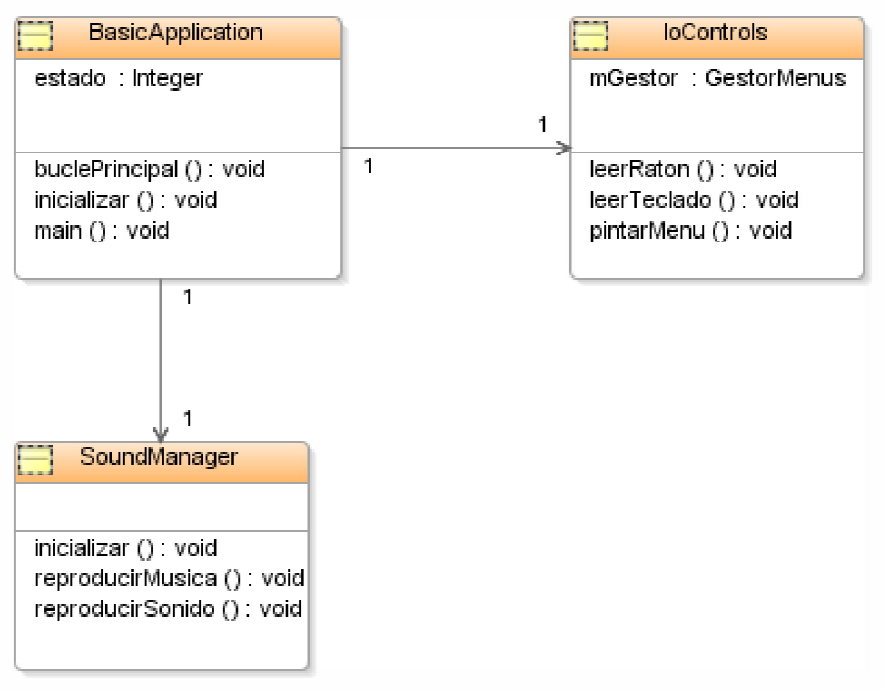

Figura 121: Diagrama de clases del Framework.

La clase **BasicApplication** es el núcleo del Framework y contiene al resto de clases, inicializa los recursos de OGRE, el depurador y llama al bucle general de renderización que se ejecuta hasta que se cierra la aplicación. Esta clase se referencia en los diagramas de secuencia como *nucleoBasico* y pertenece al componente *núcleo* del diagrama de Arquitectura.

La clase **IOControls**, que se representa como *controlesIO* en los diagramas de secuencia, es la encargada de gestionar el sistema de menús, que se trata de una librería externa de OGRE. Por otra parte gestiona la entrada y salida de usuario por teclado y ratón. Es una clase en continua comunicación con la clase principal, ya que además actúa como *FrameListener* asociado al núcleo de ejecución del programa.

La clase **SoundManager**, es la encargada de inicializar el sistema de sonido y será invocada cada vez que se desee reproducir un sonido o música de fondo del juego.

Los métodos de las tres clases, son acercamientos a la solución final, ya que quedan bastantes funciones sin definir completamente con este diagrama.

3.2.6 Clases (Proyecto)

Las clases del proyecto son una extensión del Framework anterior, el diagrama, al igual que en el caso anterior, representa objetos y métodos ficticios para dar una idea del proceso y función de cada clase, a su vez, en la explicación clase por clase, se indican algunos de los métodos reales más importantes, así como variables de la clase y elementos de la clase a destacar.

En el diagrama de clases, sólo se representan las clases necesarias de crear o extender, y se omiten clases procedentes de librerías y de OGRE, aunque se comenta su existencia y función.

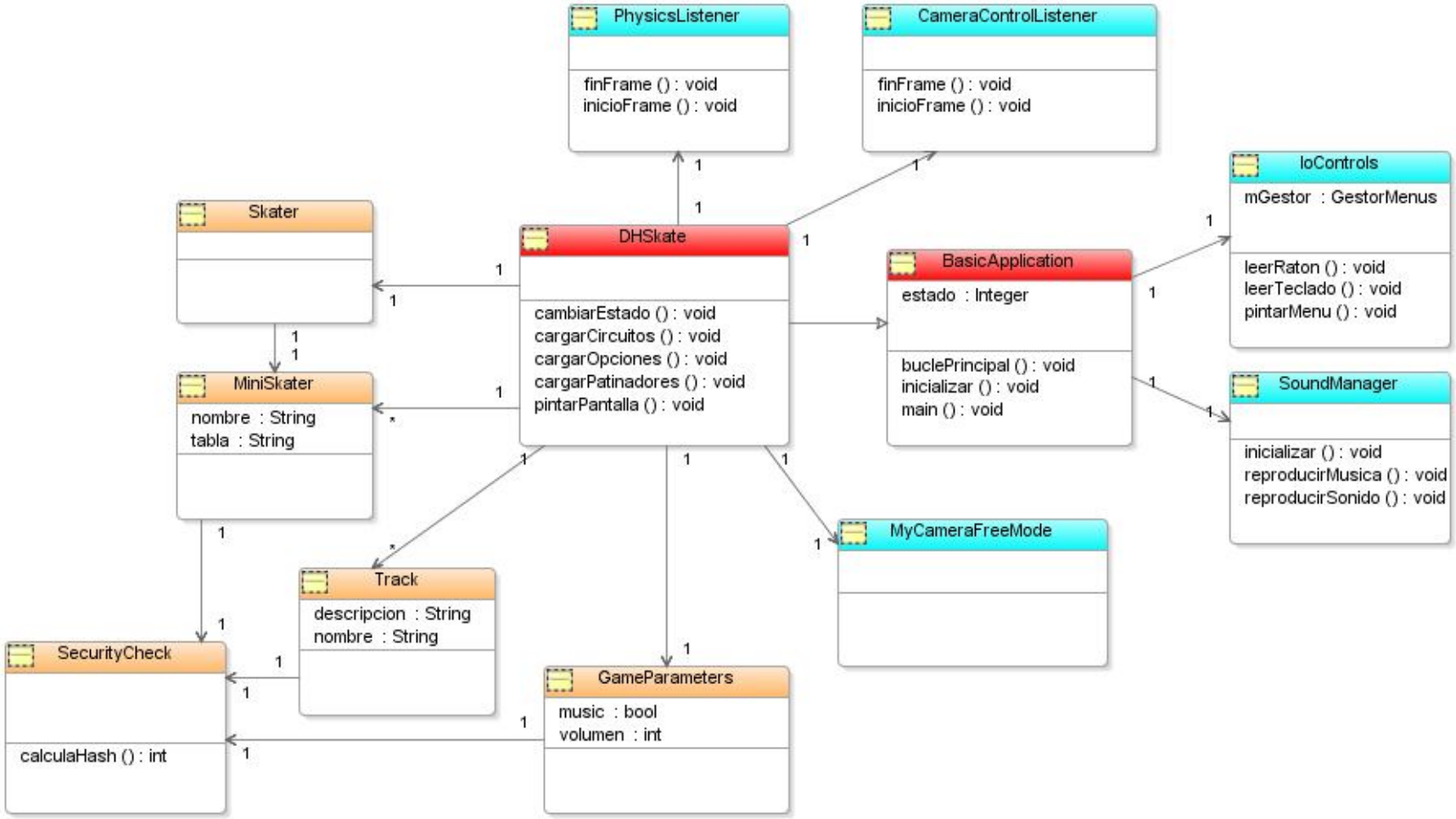

Figura 122: Diagrama de clases.

A continuación se comenta cada una de las clases, organizadas por los componentes obtenidos en la arquitectura.

Componente Núcleo.

BasicApplication

Clase:	**BasicApplication**	Herencia:	
Miembros:	- OgreNewt::World *mWorld; - Ogre::Root *mRoot; - OIS::Keyboard *mKeyboard; - OIS::Mouse *mMouse; - Ogre::Terrain *mTerrain; - IoControls *mIoControls; - SoundManager *mSoundManager; - CCS:CameraControlSystem *mCameraCS; - MyGUI::Gui *mGui; - int state;		
Métodos:	- void createRoot(); - void defineResources(); - void createRenderWindow(); - void setupScene(); - void setupGUI(); - void startRenderLoop();		

Tabla 56: Clase BasicApplication.

La clase **BasicApplication** como se ha comentado en el apartado anterior, es la clase más importante del proyecto y de la cual hereda la segunda clase más importante del juego, que es la clase *DHSkate*, que además de utilizar los métodos y datos de BasicApplication, complementa a la primera para satisfacer las necesidades del juego. La clase BasicApplication inicializa todo el sistema, incluyendo las partes básicas de cualquier aplicación OGRE, como son la pantalla, el escenario general o el objeto principal Root. Finalmente llama al bucle de renderización general.

De entre los miembros se pueden observar los objetos de las clases principales de muchas de las librerías que se han incluido en el proyecto.

La clase posee también un miembro que aunque todos los proyectos pueden usarlo, sus valores dependen del proyecto en concreto. Este miembro es state, que representa numéricamente el estado en el que está el juego y que puede ser muy útil en los juegos. Para este proyecto, el estado puede tener cuatro valores:

0 ->Juego parado. El usuario está navegando por los menús, no hay que renderizar nada ni capturar entradas de teclado, sólo las entradas de ratón en los menús.

1 ->Jugando. El usuario está jugando la partida, hay que tener en cuenta la renderización de la pantalla y el skater y la entrada por teclado de los comandos del jugador. El mundo físico ha de actualizarse.

2 ->Pre-visualización del circuito. Hay que renderizar sólo el terreno y estar al corriente de las entradas de ratón y teclado para mover la cámara que navega por el terreno.

3->Cuenta atrás. Se renderiza el terreno y el skater, pero aún no se puede jugar, porque está iniciada la cuenta atrás. Si el jugador pulsa algún botón, no ocurre nada en el skater.

La siguiente máquina de estados muestra la evolución del estado a lo largo del juego y lo relaciona con los casos de uso del análisis.

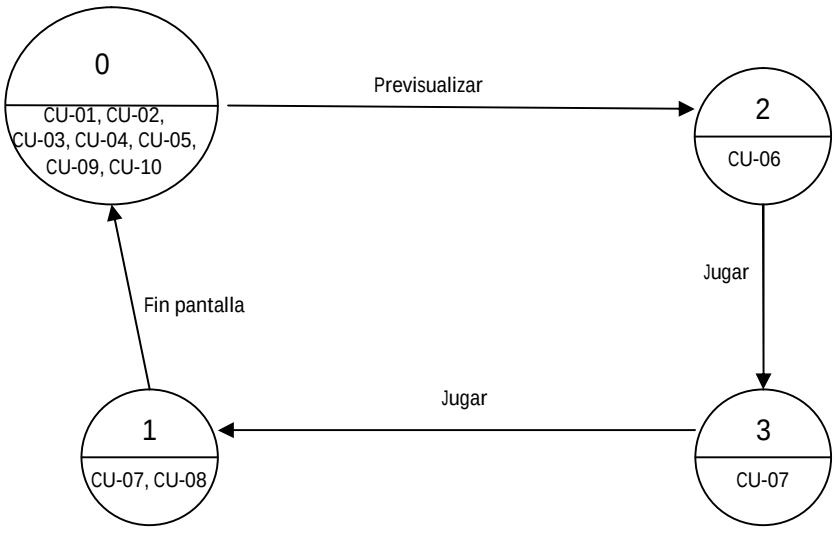

Figura 123: Máquina de estados del estado del juego.

DHSkate

Clase:	**DHSkate**	Herencia:	BasicApplication
Miembros:	- std::vector <MiniSakater*> skaters; - std::vector <Track*> tracks; - GameParameters *gameParameters; - Skater *skater; - PhysicsListener *mPhysicsListener; - CameraControlListener *mCameraControlListener;		
Métodos:	- void setupScene(); - void loadSkaters(); - void loadTracks(); - void setState();		

Tabla 57: Clase DHSkate.

Esta clase es el centro de todo el juego y desde ella (y su clase padre) se tiene el control de todo el sistema. En esta clase reside el método main() del programa, que crea una instancia estática de la clase.

En la clase se carga la parte de datos: circuitos, skaters y opciones del juego y la parte de entrada/salida que estaba sin cargar: cámaras y físicos. Además se construye el circuito generado y se crea la clase que representa al skater. Esta clase actúa de intermediaría entre el resto de clases del juego.

Componente Datos

SecurityCheck

Clase:	SecurityCheck	Herencia:	
Miembros:	- static const int X;		
Métodos:	- bool checkData(Ogre::String data1, int key); - bool checkData(Ogre::String data1, data2, int key); - bool checkData(Ogre::String data1, data2, data3, int key); - bool checkData(Ogre::String data1, data2, data3, data4, int key);		

Tabla 58: Clase SecurityCheck.

Esta clase implementa el sistema de seguridad de los ficheros xml del juego. Como se ha comentado, los ficheros xml pueden modificarse, y tienen datos sensibles como records de los circuitos o parámetros de los skaters que deben de estar protegidos.

Para ello se ha ideado un sistema simple para proteger a los ficheros, basándose en un estilo de código hash. Cada línea con datos de los ficheros xml tiene un atributo llamado “key” con un número. Este número es el resultante de aplicar una fórmula hash al resto de datos.

Para obtener el hash de una línea hay que hacer los siguientes pasos:

- Por cada valor de la línea, se recoge su primer y último carácter (sólo el valor del atributo).
- Cada carácter, se traduce a número según la posición de dicho carácter en este listado:

```
char mkey[] = "0123456789abcdefghijklmnopqrstuvwxyz.-_/";
```

Código 1: Listado de caracteres.

- Si el carácter no aparece, automáticamente se le asigna el valor 40.
- El resultado del parseo de los dos caracteres del valor, se suman y se multiplican por la constante de control X, en el proyecto se usa el 3.
- Este proceso se hace para cada valor de la línea y al final se suman todos estos valores, dando el número “key” resultante para la línea.

Cuando se lee un fichero de datos, se pasan los datos de cada línea del fichero a esta clase, que realiza el proceso y comprueba que los valores dan la misma "key" que el fichero tiene marcado.

Track

Clase:	Track	Herencia:	
Miembros:	- Ogre::String name; - Ogre::String description; - Ogre::String heightImg; - int worldSize; - int inputScale; - Ogre::String layer1Img; - Ogre::String layer1Normal; - Ogre::String layer1Mask; - ... - Ogre::String layer5Img; - Ogre::String layer5Normal; - Ogre::String layer5Mask; - Ogre::Vector3 startPoint; - Ogre::Vector3 endPoint; - Ogre::String id; - Ogre::String recordTime; - ...		
Métodos:	- get() y set() para todos los datos. - setNewRecord(Ogre::String newTime, Ogre::String newPlayer);		

Tabla 59: Clase Track.

La clase Track representa un circuito del juego. Cuando se inicializa la clase, esta lee el fichero xml correspondiente al circuito y establece los datos de los miembros. Para cada grupo de datos, se realiza una comprobación de seguridad con la clase SecurityCheck.

El xml de un circuito podría ser el siguiente:

```
<track>

<data name="Insul" description="Prueba en Alemania del campeonato del
mundo" key="375" />
<values img="IN_heightmap.png" worldsize="3000" inputScale="400" key="189">
<layer img="alphalt3.jpg" normal="asphalt_normal_MH.jpg"
mask="IN_asphalt_mask.png" key="324" />
```

```
<layer img="capa_tierra_MH.jpg" normal="capa_tierra_normal_MH.jpg"
mask="IN_dirt_mask.png" key="336" />
<layer img="grass.jpg" normal="grass.jpg" mask="IN_tree_mask.png" key="360"
/>
<layer img="" normal="" mask="IN_trees_mask.png" key="168" />
<layer img="" normal="" mask="IN_straw_mask.png" key="168" />
</values>
<points start="815 0 78" end="439 0 615" rot="0 -40" key="75" />
<record r="0:48:29" player="" key="27" />
</track>
```

Código 2: Xml de circuito.

Las características de todo circuito son las siguientes:

- **Data (Datos)**: name (nombre) y description (descripción).
- **Values (Valores)**:
  - img, que representa en escala de grises el mapa de alturas.

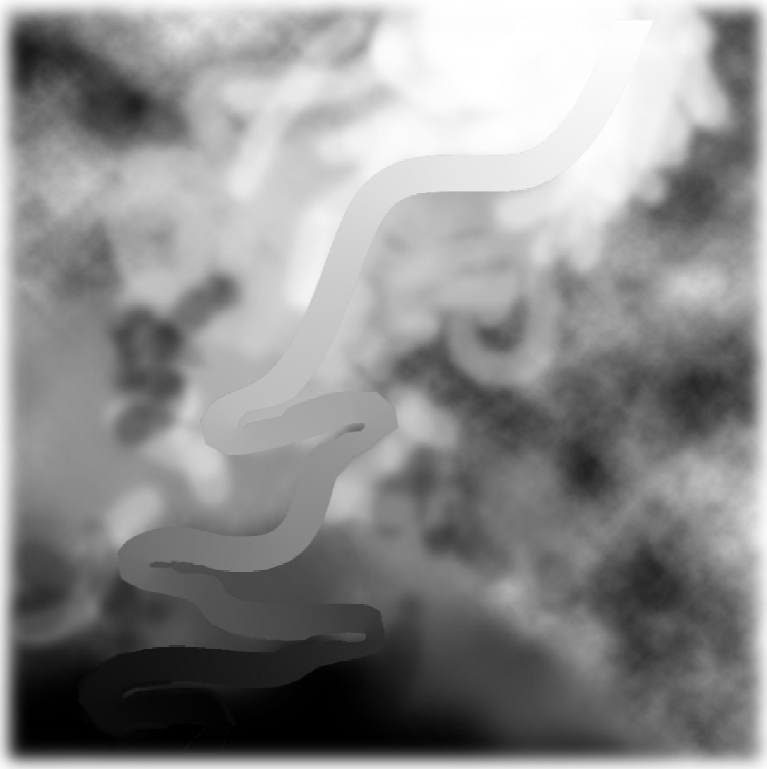

Figura 124: Imagen de ejemplo de mapa de alturas.

  - worldSize (tamaño del mundo), que es 4 veces el tamaño en metros deseado.
  - inputScale (Altura del mundo), que es 4 veces el tamaño en metros deseado.

- **Layers (Capas)**: Las capas sirven para texturizar el terreno, para ello se necesita un fichero con la textura (img), un fichero con la normal de la textura (normal) y un fichero con los lugares donde se aplica la textura y la intensidad (mask). Los circuitos tienen tres capas. Siendo la primera de ellas la utilizada para pintar el asfalto de la zona de juego. La segunda y tercera capa representan las partes exteriores del circuito, como césped, o rocas. La cuarta capa sólo utiliza la máscara (mask) en blanco y negro que dice los puntos donde habrá árboles. Y la quinta y última capa, igual que la anterior informará de los puntos donde se pondrán alpacas.
- **Points (Puntos)**: start (inicio) marca el punto de inicio del recorrido, en unidades del mapa de alturas. End (Fin) marca el punto final del recorrido en unidades del mapa de alturas. Rot indica la inclinación y rotación en grados que tiene el skater en el punto inicial.
- En **record** se guarda el record (r) en minutos:segundos:milisengundos y el player (nombre) del jugador con el record.

GameParameters

Clase:	**GameParameters**	Herencia:	
Miembros:	- `int volume;` - `bool music;`		
Métodos:	- `get() y set() para todos los datos.` - `void update();`		

Tabla 60: Clase GameParameters.

Esta clase recoge la información general del juego de opciones. Estas opciones son el volumen de los efectos y si habrá o no música de fondo. Los parámetros se leen al construir la clase de un xml y se comprueba su validez con el *SecurityCheck*.

```
<game>
<data volume="0" music="no" key="141" />
</game>
```

Código 3: xml de GameParameters.

MiniSkater

Clase:	**MiniSkater**	Herencia:	
Miembros:	- Ogre::String riderMesh; - Ogre::String riderName; - Ogre::String riderDescription; - Ogre::String longboardName; - Ogre::String longboardMesh; - Ogre::String longboardDescription; - Ogre::String truckMesh; - Ogre::String tireMesh; - Ogre::Real turnFirst; - ...		
Métodos:	- get() y set() para todos los datos.		

Tabla 61: Clase MiniSkater.

La clase *MiniSkater* representa a un skater del juego, al igual que el resto de las clases del componente de datos, al crearse lee un xml con los datos del jugador, verifica estos datos con la clase *SecurityCheck* y los almacena para usos del programa.

```
<skater>
<boy mesh="skaterScoot.mesh" name="Scoth" description="Varias veces campeón
mundial sobre la tabla" key="456"/>

<longboard mesh="malla_tabla_scoot.mesh" name="Landyachtz"
description="Tabla de diseño Americano, famosa por los grandes triunfos en
competiciones internacionales" key="546">
<truck mesh="malla_truck.mesh" key="117" />
<tire mesh="malla_rueda_scoot.mesh" key="117" />
<turn first="0.11" second="0.13" key="12" />
<velocity tfirst="0.18" tsecond="0.12" bfirst="0.10" bsecond="0.13"
key="39" />
</longboard>
</skater>
```

Código 4: xml de skater.

Los atributos del xml son los siguientes:

- **Boy**: que representa al skater.
  - Mesh: el nombre del fichero del objeto 3D que representa al jugador.
  - Name: Nombre del skater
  - Description: Descripción del skater.

- **Longboard**: Representa la tabla
    - Mesh: el nombre del fichero del objeto 3D que representa a la tabla.
    - Name: Nombre de la tabla.
    - Description: Descripción de la tabla.
    - Truck. Representa un eje de la tabla y se define con el nombre del fichero con el objeto 3D del eje.
    - Tire. Representa una rueda de la tabla y se define con el nombre del fichero con el objeto 3D de la rueda.
- **Turn**: son factores de giro de la tabla, hay dos: primero y segundo.
- **Velocity**: son factores de velocidad de la tabla y hay cuatro.

Además de que visualmente cada skater es diferente, incluso hasta por su tabla y ruedas. Cada jugador tiene unas propiedades que le hacen distinto del resto según su control, para ello sirven los factores que se incluyen en el fichero xml. Estos factores se explican con detalle más adelante.

Skater

Clase:	**Skater**	Herencia:	
Miembros:	- `MiniSkater *miniSkater;` - `OgreNewt::Body *body;` - `Ogre::SceneManager* mMgr;`		
Métodos:	- `static Skater* create(MiniSkater* mini, Ogre::SceneManager* mgr, OgreNewt::World* world, Ogre::Quaternion iniRot, Ogre::Vector3 iniPos);`		

La clase *Skater* representa al patinador que se utiliza en el juego, con el que interactúa el jugador.

El Skater está definido por un *MiniSkater* donde obtener todos los datos.

Cuando se construye el Skater, se monta el sistema de objetos para el mundo físico y el sistema de objetos 3D para el mundo visual, además se le posiciona con una rotación y posición determinada.

Para la iteración física, el Skater queda definido mediante un *Body* de Newton con forma de caja, invisible en el juego, pero sobre la que actuarán las fuerzas físicas. Esta caja se ha acoplado a Nodos del escenario OGRE (*SceneNode*) que representan objetos 3D mediante entidades de OGRE (*Entity*), y se ha creado un sistema enlazado de entidades y nodos como sigue a continuación:

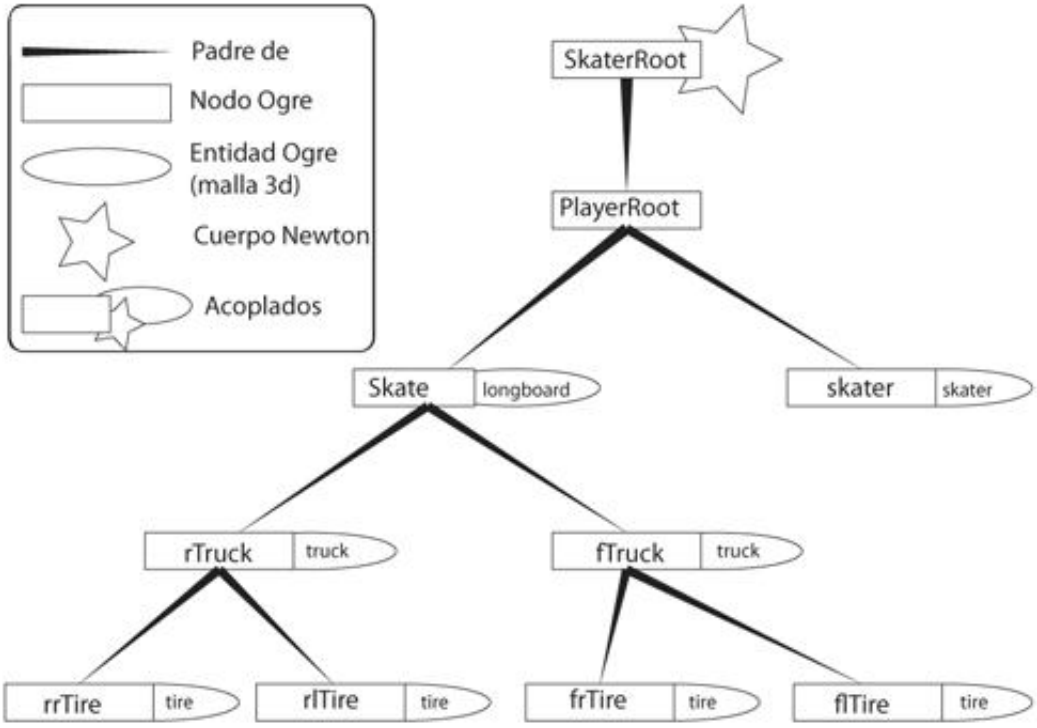

Figura 125: Diagrama de Objetos del patinador.

Con esto queda definido el patinador, y se puede acceder a cualquiera de sus elementos, tanto visuales 3D, como el objeto físico.

Componente Entrada/Salida

MyFreeCameraMode

Clase:	**MyFreeCameraMode**	Herencia:	CCS::FreeCameraMode
Miembros:			
Métodos:			

Tabla 62: Clase MyFreeCameraMode.

Como se ha comentado anteriormente el sistema de cámaras utilizado es el *Camera Control System*. Para el proyecto se utilizan 3 modos de cámara: uno libre para mover por todo el terreno, uno de tercera persona siguiendo al skater y uno en primera persona.

El modo de cámara libre de la librería CCS, tiene un pequeño defecto, que no permite establecer su posición inicial. Con esta clase, se consigue poder tener un constructor de la clase en el que establecer la posición inicial de la cámara y evitar así el error.

CameraControlListener

Clase:	**CameraControlListener**	Herencia:	Ogre::FrameListener
Miembros:	- `CCS::CameraControlSystem* mCameraCS;` - `int vertical;` - `int horizontal;`		
Métodos:	- `bool keyPressed(const OIS::KeyEvent &evt);` - `bool mouseMoved(const OIS::MouseEvent &evt);` - `...`		

Tabla 63: Clase CameraControlListener.

Esta clase es un *FrameListener* para las cámaras, en concreto para cuando está habilitado el modo de cámara libre y pre-visualización. El gestor general de entrada/salida *IoControls* manda los eventos de pulsación de teclado y ratón a esta clase, que gestiona para que se actualice la posición de la cámara libre y mueva o rote la misma según movimientos de teclado y ratón del usuario.

PhysicsListener

Clase:	**PhysicsListener**	Herencia:	Ogre::FrameListener
Miembros:	- `Skater *mPlayer;` - `...`		
Métodos:	- `bool frameStarted(const Ogre::FrameEvent &evt);` - `...` - `void skateCallback(OgreNewt::Body *body, float timeStep, int threadIndex);` - `void makeAnimation(Ogre::String animationName, bool loop);`		

Tabla 64: Clase PhysicsListener.

Esta clase es otro *FrameListener* del proyecto, y se encarga de actualizar los movimientos físicos y visuales del skater en la pantalla. El control general de entrada/salida recoge las pulsaciones de tecla del usuario y se lo indica a esta clase, que por otra parte, tras el paso de cada frame, actualiza el skater según las distintas fuerzas físicas por las que se ve afectado el mismo. Al final de la actualización realiza un control de posición del jugador, para detectar si ha llegado al final o si se ha salido del circuito.

Además en esta clase, se llama a las animaciones 3D creadas en Blender, para reflejar los movimientos del skater encima de la tabla según el estado del mismo y las teclas que esté pulsando el jugador. Las teclas y su efecto se resumen a continuación:

Tecla	Efecto
I	Con esta tecla el skater acelera, empujando con el pie contra el suelo, sirve para iniciar el movimiento de la tabla en una prueba.
K	Freno, el skater frena, bien con el pie contra el suelo si está en pie o perdiendo efecto aerodinámico si está agachado.
J, L	Giros a izquierda y derecha respectivamente. Tanto en pie como agachado puede girar.
A	Por cada pulsación, el skater cambia de posición, en pie o agachado. Cada uno con sus propiedades particulares.
C	Cambio de cámara. Cambia la cámara a tercera o primera persona.

Espacio	Realiza un derrape. El jugador debe de estar en pie para realizarlo, es una forma muy efectiva de bajar la velocidad.
Escape	Con esta tecla se abandona la partida para volver al menú principal.

Tabla 65: Teclas y efectos en el modo de juego.

A continuación se explican los controles que se tienen en cuenta en esta clase y la forma sobre la que actúan las fuerzas sobre el skater para simular el movimiento del skater en la pantalla.

Como se ha comentado, la parte física se controla mediante un cubo de *Newton* sobre el que se aplican ciertas fuerzas. Estas fuerzas se ven modificadas por las propiedades de cada jugador, cada fuerza sigue la siguiente fórmula: **FuerzaResultante = FuerzaPorDefecto + FuerzaPorDefecto * parametroJugador**.

Los parámetros que posee un jugador y los efectos que producen son los siguientes:

- **Turn first (T1)** = Giros cuando está agachado.
- **Turn second (T2)** = Giros cuando está en pie.
- **Velocity tFirst (VT1)** = Aumento de velocidad cuando está agachado.
- **Velocity tSecond (VT2)** = Aumento de velocidad cuando está en pie.
- **Velocity bFirst (VB1)** = Aumento de la frenada cuando está agachado.
- **Velocity bSecond (VB2)** = Aumento de la frenada cuando está en pie.

Las fuerzas que actúan sobre un skater son las siguientes:

- **Fuerza de gravedad (FGv, negro).** Esta fuerza siempre se aplica y es equivalente a 9,8 * masa (100) = 980.
- **Fuerza de empuje (FE, rojo).** Es la fuerza que hace acelerarse al skater en la dirección que lleva. Esta fuerza en la realidad es inexistente, ya que la aceleración del un patinador se produce por la propia gravedad sobre la pendiente del terreno. En un entorno simulado, ha sido necesaria la inclusión de esta fuerza, que simula la aceleración del propio patín. Establecida por defecto a 1500.

- **Fuerza de freno (FF, azul).** Fuerza producida por el skater contraria a la dirección del monopatín cuando se está frenando. Por defecto establecida a 1500.
- **Fuerza de giro (FG, verde).** Conjunto de fuerzas aplicadas en distintos puntos del skater para hacer girar el cuerpo del mismo. Por defecto tiene un valor de 1500. Se aplica en tres puntos: en el centro del cuerpo para desplazarlo y reducir inercias. En la parte frontal del cuerpo con un valor del doble del resultante, para rotar el cuerpo y ayudar en el giro. Y finalmente se aplica una fuerza 8 veces inferior a la normal, en sentido contrario y en la parte posterior del skater, para ayudar en la rotación.
- **Fuerza de derrape (FD).** Es una fuerza pequeña y aleatoria en cuanto al sentido, que actúa cuando el jugador está derrapando, para simular la dificultad de controlar los derrapes.

El siguiente diagrama muestra visualmente el lugar dónde se aplican las fuerzas, según el sentido del avance:

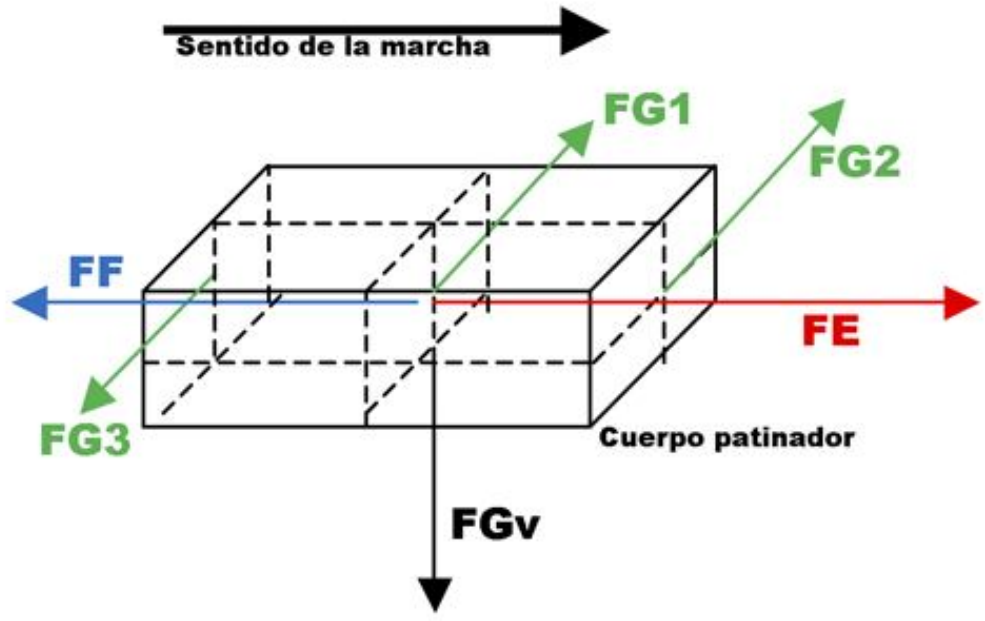

Figura 126: Diagrama de fuerzas.

En la siguiente tabla, se pueden observar cómo se ven afectadas las distintas fuerzas que recibe el skater en el juego, en la primera tabla se muestra cuando el jugador realiza un control normal sin derrapar ni acelerar con el pie:

NOTA: La notación FE + FE(VT2) es equivalente a FR = FE + FE * VT2, que cumple la condición comentada anteriormente: **FuerzaResultante = FuerzaPorDefecto + FuerzaPorDefecto * parametroJugador**

	Siempre	Sin pulsar nada	Girando	Freno
Agachado	FE + FE(VT2)	FE += FE(VT2) /2	FG1 = FG + FG(T2) FG2 = 2 * FG1 FG3 = -1(FG1 / 8)	FF + 2 * FF(VB2)
En pie	FE+ FE(VT1)	FE += FE(VT1) /2	FG1 = FG + FG(T1) FG2 = 2 * FG1 FG3 = -1(FG1 / 8)	FF + 2 * FF(VB1)

Tabla 66: Fuerzas en situación normal.

Hay una fuerza de empuje (FE), que siempre actúa, igual que ocurre con la gravedad (FGv). La fuerza de empuje (FE) adicionalmente se ve incrementada si el jugador va totalmente recto y sin frenar, simulando que ha adoptado una postura aerodinámicamente beneficiosa. Cuando gira se le aplican las 3 fuerzas que se han indicado anteriormente (FG1, FG2 y FG3). Finalmente si frena, se añade fuerza de freno (FF).

Cuando el jugador decide derrapar, se produce la suma de fuerzas siguientes:

	Siempre	Frenada
Derrape	FE + FE(VT2)	FF = FF + 8FF(VB2) FD = FD en sentido aleatorio y que cambia cada cierto tiempo.

Tabla 67: Fuerzas en derrapes.

La fuerza de empuje (FE), al igual que la de la gravedad (FGv) siempre está presente, por otra parte se suma la fuerza de frenada del derrape (FF) y se añade una fuerza en sentido aleatorio que cambia de sentido cada pocos milisegundos (FD).

Finalmente se muestra la tabla que contiene la forma en que varía la fuerza de empuje del jugador (FE), cuando debe de acelerar con el pie o cuando está acelerando con el pie y ya no es necesario. Cuando el skater va a una velocidad por debajo de 10 kilómetros por hora, indica que el jugador debe de acelerar:

	Tecla de acelerar pulsada	Tecla de acelerar sin pulsarse
Debe acelerar	FE = 0.8 * FE	FE = 0.5 * FE
NO debe acelerar	FE = 0.7 * FE	FE = FE (situación normal)

Tabla 68: Fuerzas con aceleraciones de pie.

Esto refleja que cuando la velocidad es baja, es recomendable acelerar al patinador y cuando la velocidad es alta, acelerar con el pie es contra productivo.

El último control que añade la clase es que si hay dos ruedas fuera del circuito, la fuerza de empuje se reduce a la mitad, y si hay tres o las cuatro ruedas fuera del circuito, entonces se sale del modo de juego, ya que se considera que el skater se ha caído.

IoControls

Clase:	**IoControls**	Herencia:	Ogre::FrameListener, Ogre::WindowsEventListener, OIS::KeyListener, OIS::MouseListener
Miembros:	- OIS::InputManager *mInputManager; - OIS::Keyboard *mKeyboard; - OIS::Mouse *mMouse; - MyGUI::Gui *mGui; - Ogre::Timer *tGame; - ...		
Métodos:	- bool frameStarted(const Ogre::FrameEvent &evt); - ... - void nextGuiMenu(Ogre::String filename); - void initMenu1(void); - ...		

Tabla 69: Clase IoControls.

La clase *IoControls* es un *FrameListener* de OGRE que gestiona toda la entrada/salida de teclado y ratón, y controla los menús y visuales de interfaces gráficas de usuario. Además gestiona algunos contadores de tiempo de utilidad para el juego.

Al recoger toda la entrada/salida del juego, es la clase encargada de pasar esa información al resto de FrameListeners del juego, en cada caso y según el estado del juego.

Se encarga de intercambiar menús de la librería *MyGUI*.

La última tarea de la clase es la de llevar contadores de tiempos para la duración de la partida y para la cuenta atrás de la misma. Además si el jugador pulsa alguna tecla durante la cuenta atrás, el temporizador del juego añadirá un segundo como penalización.

SoundManager

Clase:	SoundManager	Herencia:	
Miembros:	- cAudio::IAudioManager* manager; - int mVolume; - bool mMusic;		
Métodos:	- void playMusic(); - void stopMusic(); - void playEffect(Ogre::String effectName, bool loop); - void stopEffect(Ogre::String effectName); - void playRunEffect(int turn, int speed); - ...		

Tabla 70: Clase SoundManager.

Esta clase gestiona todos los sonidos del juego, tanto efectos sonoros como música de fondo. Utiliza la librería *cAudio* y puede controlar muchos sonidos al mismo tiempo.

Además los efectos pueden "moverse" para simular que la fuente del sonido proviene de distintas zonas del juego según los movimientos del patinador.

3.3 Planificación y presupuesto

Un proyecto no debe empezarse sin haberse realizado una planificación previa. Uno de los fines de la planificación es el de estructurar el proyecto en etapas ordenadas, para facilitar el cálculo de recursos, costes asociados al proyecto y el tiempo estimado de trabajo en el mismo.

Al presentar una planificación ordenada, se puede hacer un mejor control del seguimiento y proceso del proyecto, de forma que se pueda comprobar si los plazos se están cumpliendo.

En este apartado se comentan los ciclos de vida que suelen seguirse en la realización de un videojuego, además del ciclo de vida seguido en el proyecto de *DHSkate*. Finalmente se muestra el diagrama de planificación inicial del juego.

3.3.1 Ciclo de vida de un juego

A continuación se detallan las fases típicas en el ciclo de vida de un videojuego. Estas no han sido las seguidas en el proyecto, ni tampoco significan un método cerrado y definitivo para planificar y desarrollar un videojuego, pero sí son muy genéricas y pueden servir de ayuda o guía para cualquier tipo de planificación y diseño de videojuego.

Concepto: En esta fase, se define el concepto del juego, a partir de una idea de partida o una tormenta de ideas. En ella se crea una propuesta de juego y el arte conceptual. Se definen características del juego como ambientación, características o género. Además de realiza una planificación previa con costes de recursos, tiempo y una estimación del presupuesto.

Pre-Producción: En esta etapa se estudia la viabilidad del videojuego y se determina si el proyecto puede realizarse por el equipo de trabajo. Ha de definirse un estudio del juego completo, desde el análisis, incluyendo detalles del mismo como personajes, ambientaciones, interfaces de usuario, etc. Finalmente y con el análisis profundo del videojuego realizado, se vuelve a hacer una planificación detallada del mismo.

Producción: En esta fase se construye el juego. Con todas las partes principales del mismo. Cuando el juego está a nivel suficiente como para poder probarse, los testeadores se encargan de realizar este trabajo y junto con el equipo de desarrollo, se arreglan errores, se realizan avances y se añaden nuevas funcionalidades.

Alfa: Cuando el juego está en fase alfa, se puede jugar completamente al mismo. El motor, la interfaz de usuario y otros subsistemas están completos, pero pueden quedar detalles de agregar o arreglar.

Beta: En este punto todas las funcionalidades están desarrolladas. Por lo que el aumento de características queda finalizado, y en esta situación sólo se realizarán correcciones de errores. En esta etapa se persigue dejar el proyecto en una situación estable: completamente jugable y sin fallos.

Liberación: Una vez que el programa está completo, se congela la versión del código para impedir cambios y se aprueba. Finalmente se libera a los medios de distribución determinados para que el juego pase al público.

Parches: En algunas situaciones extrañas, aunque en actualidad cada vez más frecuentes, aparecen errores encontrados por los usuarios que deben ser reparados. Hoy en día resulta sencillo realizar la distribución de un parche, a través de la red.

Actualizaciones: Al igual que los parches, se distribuyen a través de la red. Las actualizaciones son mejoras para el videojuego, que pueden incluir desde pequeños elementos, hasta mundos o niveles completamente nuevos para el juego.

3.3.2 Ciclo de vida del proyecto

Antes de definir las etapas del proyecto, se debe de elegir un modelo de ciclo de vida a seguir durante el proyecto. Tras evaluar las distintas posibilidades y ver las intenciones de desarrollo, se ha elegido el Modelo de desarrollo evolutivo o por prototipado evolutivo.

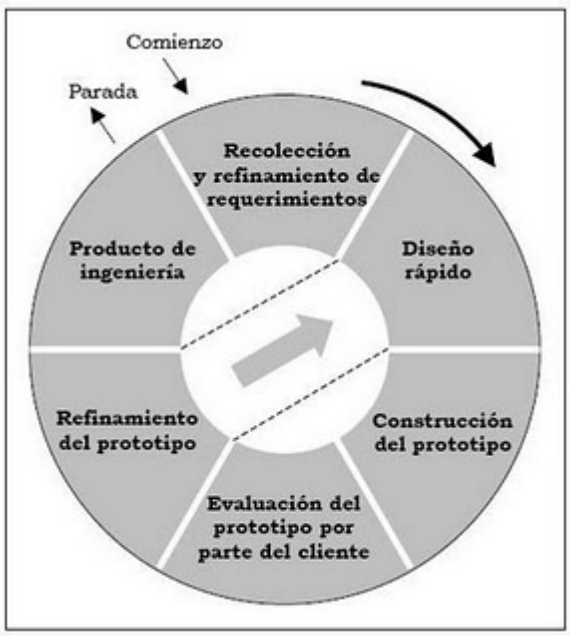

Figura 127: Diagrama del Modelo de desarrollo evolutivo.

La idea del desarrollo del proyecto es ir centrándose en ciertos módulos del proyecto y completar cada uno de ellos, además de añadirlo al principal, por lo que este modelo se ajusta a estas necesidades (como se verá en la planificación). De esta forma desde el primer momento se obtiene una versión ejecutable y testeable del juego, a la que poco a poco se le van añadiendo funcionalidades como físicos, sonidos, interfaces de usuario, etc.

Con esta adopción, se ha dividido al proyecto en varias partes o prototipos, y para cada uno de ellos se han realizado las tareas que comprenden este modelo: un diseño y estudio

previo, la implementación del prototipo y su evaluación. Teniendo en cuenta que puede darse un refinamiento de los requisitos y volver de nuevo a diseñar y construir, tal y como muestra la figura anterior.

El proyecto se compone de los siguientes componentes o prototipos:

- **Formación previa**: Consiste en el estudio de los elementos a utilizar en el proyecto. Para este caso concreto, un estudio sobre OGRE y Blender, las dos herramientas a utilizar. En caso de disponer de los conocimientos necesarios, en un proyecto normal, este paso no sería necesario.
- **Documentación del proyecto**: Incluye este documento general sobre el proyecto. Este componente se irá evolucionando a lo largo del tiempo del proyecto, recogiendo la documentación de cada uno de los prototipos realizados. Incluyendo su análisis, diseño, implementación, evaluación y problemas encontrados. Las partes generales se realizan al inicio del proyecto y sin prototipo asociado, como es el caso del análisis general y algunas partes del diseño.
- **Framework inicial**: Se compone del núcleo del proyecto. Que genera un proyecto con la funcionalidad mínima y suficiente para que el juego arranque, además de servir de base en la que apoyar el resto de prototipos.
- **Terreno**: Prototipo en el que se crean los terrenos de juego y se añaden al proyecto.
- **Cámaras y sonidos**: En esta sección, se crea el sistema de cámaras del proyecto y el sistema de sonidos y efectos.
- **Patinador**: En este componente, el desarrollo se centra en la creación del patinador, desde sus bocetos y diseños en 3D hasta poder incluirlo en el proyecto general.
- **Menús y temporizadores**: Con este prototipo se diseñan los menús del juego y el sistema de gestión de los mismos, para intercambiarlos en los distintos momentos del juego. También se implementan los temporizadores auxiliares del juego.
- **Motor físico**: En este prototipo, se crea el sistema de gestión de fuerzas físicas y colisiones que interactuarán entre el patinador y el terreno.

- **Controles patinador**: En este prototipo se diseña e implementa el sistema de control del patinador, para interactuar con el jugador y pasar los comandos de teclado y ratón a movimientos del patinador.
- **Elementos adicionales**: En esta parte del desarrollo, se crean nuevos circuitos y patinadores y se añaden al sistema.
- **Evaluación y pruebas generales**: Con todos los prototipos anexados, se realizan unas pruebas de evaluación y aprobación generales para comprobar el correcto funcionamiento de todo el juego.

Todos estos prototipos quedan definidos en detalle en los diagramas de planificación, contando una estimación de tiempos y recursos implicados en cada una de las tareas más importantes de cada una de los mismos.

3.3.3 Planificación inicial

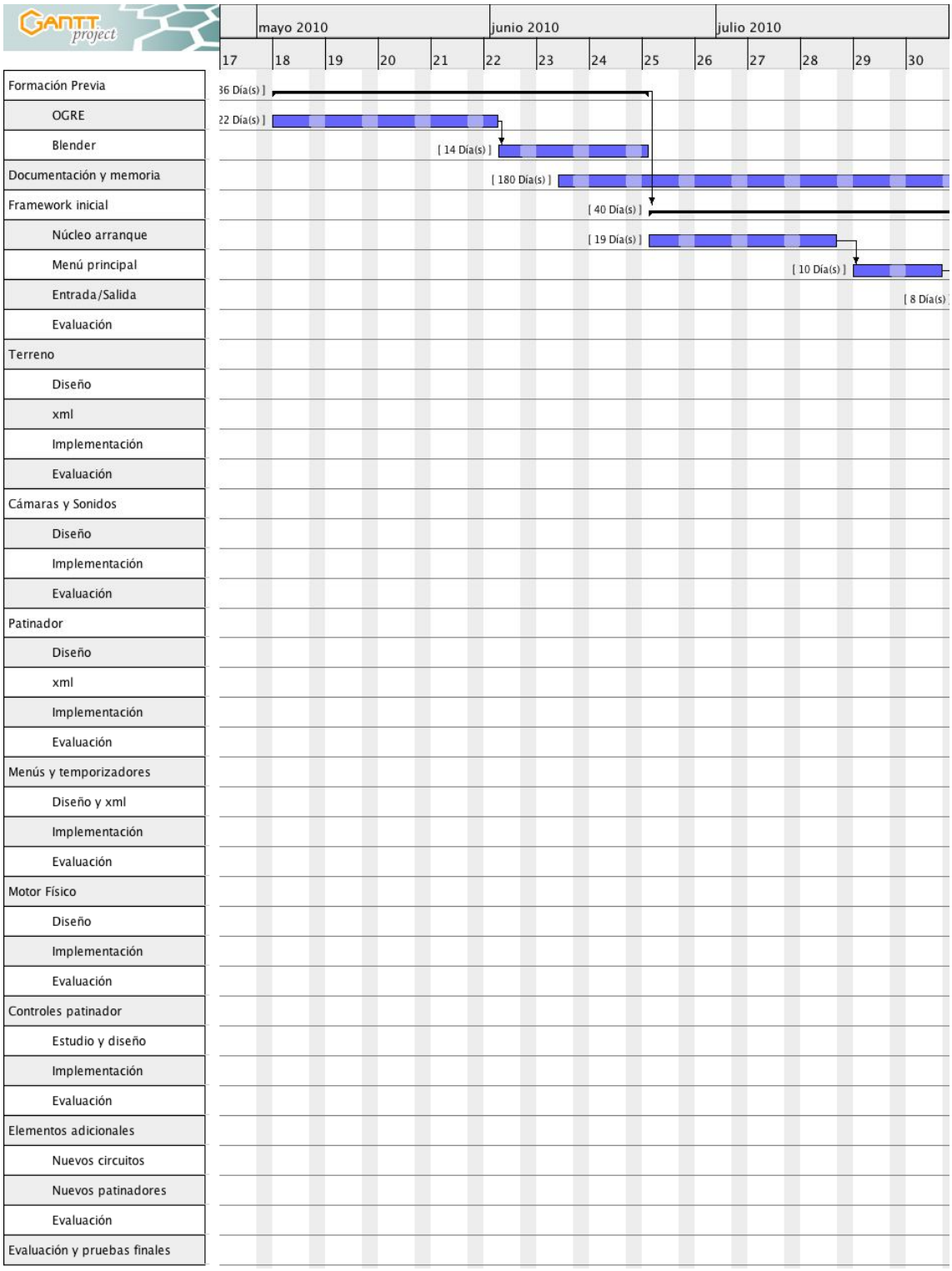

Figura 128: Primera parte de la planificación inicial.

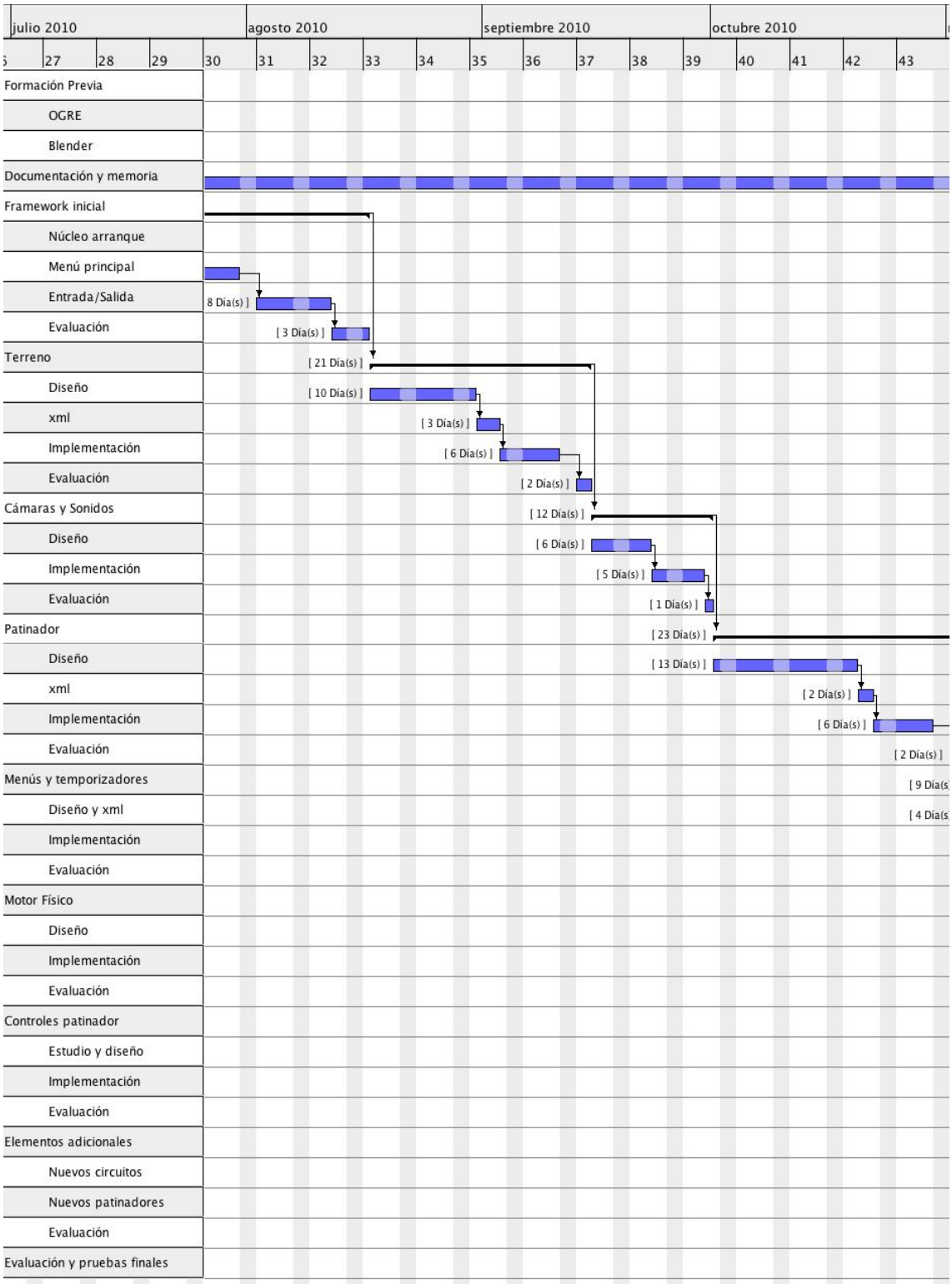

Figura 129: Segunda parte de la planificación inicial.

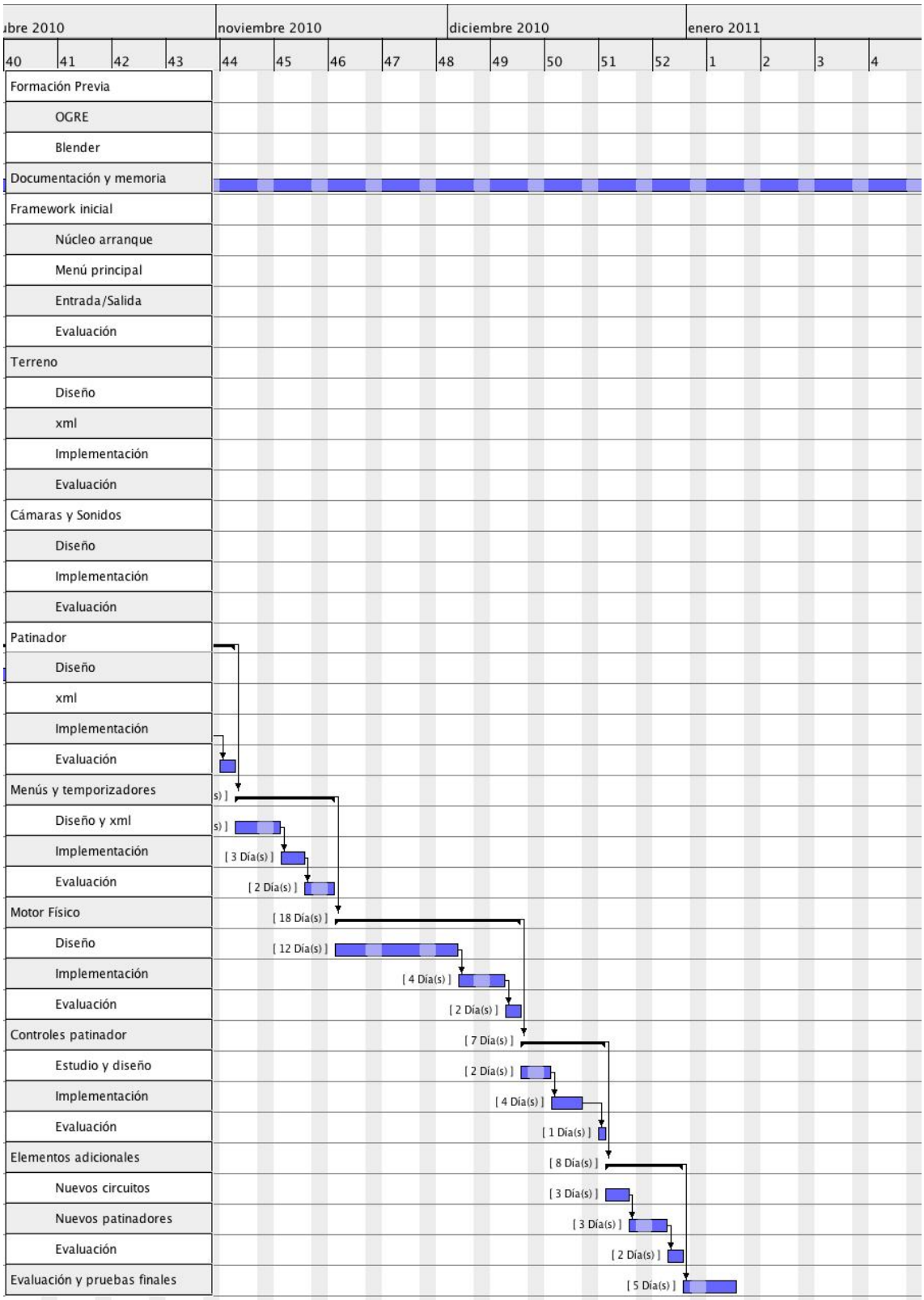

Figura 130: Tercera parte de la planificación inicial.

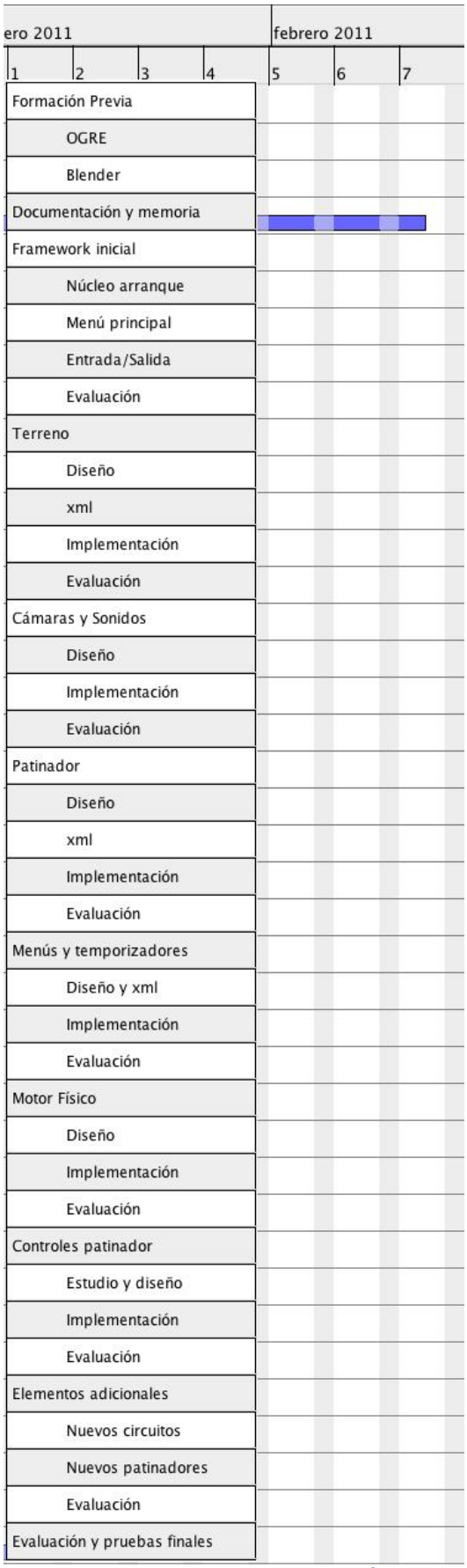

Figura 131: Cuarta parte de la planificación inicial.

La planificación inicial del proyecto se ha establecido con una duración de 10 meses, desde Mayo del año 2010 hasta Febrero del 2011. En esta planificación se ha contado con un único recurso, el autor del proyecto. Además los Sábados y Domingos no se han contabilizado. El resumen de la planificación queda reflejado en la siguiente tabla, donde se resumen las tareas generales y su duración sin desglosar en subtareas:

Tarea	Inicio	Fin	Duración
Formación previa	03/05/2010	22/06/2010	36 días
Documentación y memoria	10/06/2010	07/02/2011	180 días
Framework inicial	22/06/2010	07/08/2010	40 días
Terreno	07/08/2010	15/09/2010	21 días
Cámaras y sonidos	15/09/2010	01/10/2010	12 días
Patinador	01/10/2010	03/11/2010	23 días
Menús y temporizadores	03/11/2010	16/11/2010	9 días
Motor físico	16/11/2010	10/12/2010	18 días
Controles patinador	10/12/2010	21/12/2010	7 días
Elementos adicionales	21/12/2010	31/12/2010	8 días
Evaluación y pruebas finales	31/12/2010	07/01/2011	5 días

Tabla 71: Planificación inicial de tareas.

3.3.4 Presupuesto

A continuación se muestra un cálculo de costes asociados al proyecto, desde los costes de recursos del personal asociado al mismo, hasta gastos de terceros y gastos de material o equipos.

PRESUPUESTO DE PROYECTO

1.- Autor:

Mario Velázquez Muñoz

2.- Departamento:

Departamento de Informática

3.- Descripción del Proyecto:

- Titulo: **Diseño e implementación de un juego para PC mediante OGRE 3D**
- Duración (meses): **10**
- Tasa de costes Indirectos: **20%**

4.- Presupuesto total del Proyecto (valores en Euros):

38.052,00 Euros

5.- Desglose presupuestario (costes directos)

PERSONAL

Apellidos y nombre	N.I.F. (no rellenar - solo a titulo informativo)	Categoría	Dedicación (hombres mes) [a)]	Coste hombre mes	Coste (Euro)
Peralta Donate, Juan		Jefe de Proyecto	1	4.289,54	4.289,54
Velázquez Muñoz, Mario		Ingeniero	10	2.694,39	26.943,90
					0,00
					0,00
					0,00
		Hombres mes	**11**	**Total**	31.233,44

[a)] 1 Hombre mes = 131,25 horas. Máximo anual de dedicación de 12 hombres mes (1575 horas)

Máximo anual para PDI de la Universidad Carlos III de Madrid de 8,8 hombres mes (1.155 horas)

Figura 132: Primera parte del cálculo del presupuesto

EQUIPOS

Descripción	Coste (Euro)	% Uso dedicado proyecto	Dedicación (meses)	Periodo de depreciación	Coste imputable [d)]
Ordenador diseño gráfico y programación	1.200,00	100	10	60	200,00
Disco duro externo	158,00	100	10	60	31,60
Impresora	100,00	30	10	60	6,00
				Total	237,60

[d)] Fórmula de cálculo de la Amortización:

$$\frac{A}{B} x C x D$$

A = nº de meses desde la fecha de facturación en que el equipo es utilizado
B = periodo de depreciación (60 meses)
C = coste del equipo (sin IVA)
D = % del uso que se dedica al proyecto (habitualmente 100%)

SUBCONTRATACIÓN DE TAREAS

Descripción	Empresa	Coste imputable
	Total	0,00

Figura 133: Segunda parte del cálculo del presupuesto.

OTROS COSTES DIRECTOS DEL PROYECTO[e)]

Descripción	Empresa	Costes imputable
Fungible	Ofistore	60,00
Viajes	-	40,00
Licencia Microsoft Office Hogar y Estudiantes	Microsoft Store	139,00
		0,00
		0,00
		0,00
		0,00
	Total	239,00

[e)] Este capítulo de gastos incluye todos los gastos no contemplados en los conceptos anteriores, por ejemplo: fungible, viajes y dietas, otros,...

6.- Resumen de costes

Presupuesto Costes Totales	**Presupuesto Costes Totales**
Personal	31.233
Amortización	238
Subcontratación de tareas	0
Costes de funcionamiento	239
Costes Indirectos	6.342
Total	38.052

Figura 134: Tercera parte del cálculo del presupuesto.

El presupuesto total del proyecto ha ascendido a un total de 38.052 Euros, incluyendo el coste de trabajo durante 10 meses de un Ingeniero y el trabajo de 1 mes de un jefe de proyecto. Además de gastos por equipos informáticos como ordenadores e impresoras, y sumando costes de viajes, fungibles y licencias, todo ello teniendo en cuenta unas tasas indirectas del 20%.

3.3.5 Comercialización del producto

Una vez realizados la planificación y los cálculos del presupuesto del proyecto, se debe de hacer un estudio de mercado, donde estimar las ventas a realizar para que el juego sea rentable.

Para realizar cálculos de comercialización, hay que conocer los medios de difusión del juego y sus costes. Para este proyecto, la plataforma de difusión será Steam [12].

El juego incorporará una cantidad grande de circuitos y patinadores, para que aporte mucha jugabilidad.

En cuando a aspectos de publicidad, no se realizará ningún mecanismo para publicitar el juego, salvo el propio uso de Steam.

En cuanto a temas económicos, Steam tiende a quedarse un 60% (negociables) del valor final del juego. En este caso se ha elegido un precio final de 5 Euros por juego, que puede variar a lo largo de los años a la baja y alza para atraer nuevos compradores.

Otro aspecto a realizar, es la inclusión dentro del juego de publicidad, poniendo los logotipos de las empresas publicitadas en los menús y en componentes del juego, como puede ser en el mono y casco del patinador. Se puede considerar cobrar 100 Euros a cada empresa publicitada y se estima una cantidad de empresas entre 3 y 10. Las empresas pueden ser:

- Tiendas de Internet relacionadas con el deporte
- Tiendas físicas relacionadas con el deporte
- Marcas de monopatines
- Marcas de ruedas y ejes
- Marcas de protecciones

- Marcas de ropa

A continuación se muestran los cálculos para recuperar los gastos calculados en el presupuesto durante el periodo de un año.

- Presupuesto: 38.052 Euros.
- Aporte publicitario, contando unas 6 empresas publicitadas: 6 * 100 = 600 Euros.
- Cantidad recaudada por cada juego vendido en Steam: 40% de 5 Euros = 2 Euros.
- Cantidad a recuperar por ventas (presupuesto – publicidad) = 38.052 – 600 = 37.452 Euros.
- Cantidad de copias a vender para recuperar gastos y empezar a obtener beneficios (Cantidad a recuperar / valor del juego) = 37.452 / 2 = 18.726 copias.
- Cantidad de copias que hay que vender al mes durante un año para empezar a tener beneficios: 18.726 / 12 = 1561 copias por mes.

Vender 1561 copias al mes a través de Steam es una tarea dura, aunque no imposible. El juego cuenta con un precio muy atractivo para el usuario final y con pequeños desarrollos, de coste mínimo, se pueden atraer nuevos compradores y nuevas empresas a publicitarse.

La estrategia a seguir podría ser la siguiente:

- Realizar pequeños desarrollos a coste mínimo como los comentados en el capítulo Líneas futuras: modo fantasma, records online, nuevos circuitos y jugadores, algunos de los cuales ocultos, etc.
- Buscar nuevos patrocinadores publicitarios, incluso en algún cambio de versión renovar algunos de los sponsors antiguos.
- Trabajar en desarrollos más complejos y atrayentes al usuario como modo multijugador o modo con adversarios con inteligencia artificial. Estas modificaciones no tendrán un costo muy elevado porque el sistema permite el acople de nuevos componente de forma sencilla. Estos desarrollos y actualizaciones podrían incorporarse al cabo de un año o año y medio con el juego en comercialización, para impulsar de nuevo las ventas.

- Ampliar el número de plataformas de difusión, como por ejemplo crear una web propia sin intermediarios o incluir el juego en sistemas similares a Steam.
- Cuando el nivel de pérdidas no sea muy elevado, se puede pensar también en publicitar el juego a través de sistemas web como las redes sociales o buscadores de contenidos.

3.4 Implementación

Una vez realizado un estudio completo, queda el trabajo de implementar el proyecto. Normalmente es la tarea más larga de todo un proyecto y requiere muchas horas de trabajo. Además, aunque las tareas de análisis y diseño hayan sido buenas, siempre pueden aparecer problemas a nivel de programación que provoquen cambios en el ciclo de vida del software y retrasos en los tiempos estimados.

En este apartado se comentan algunos de los detalles de implementación más significativos del proyecto, y los que más han dificultado el desarrollo del proyecto, omitiendo las partes menos significativas.

Se comienza explicando una introducción, sobre la que se comentan las acciones realizadas al principio del desarrollo, así como algunos de los problemas que aparecieron durante la implementación y que obligaron a realizar ciertos cambios en el diseño y desarrollo del proyecto.

Seguidamente se comenta algo de información sobre los diseños 3D realizados con la herramienta Blender y algunas de las circunstancias a tener en cuenta a la hora de diseñar para OGRE.

Para finalizar se comentan los aspectos principales del juego, como son el núcleo, los mapas del juego, la lectura de ficheros xml, los físicos, el sistema de sonidos o los interfaces gráficos de usuario.

En el apartado de implementación, como sólo se muestran partes del código, para ayudar a localizarlo en los fuentes del proyecto, en el título adjunto a cada trozo de código, se indica la clase y método donde localizar la parte mostrada, y así poder ver el código completo.

3.4.1 Implementación previa y cambios

En algunos apartados del análisis y del diseño se ha hablado sobre la realización de un pequeño proyecto de OGRE, para seguir los tutoriales y obtener algo de soltura con el manejo de la aplicación Microsoft Visual C++. Es muy recomendable crearse un pequeño proyecto de OGRE siguiendo el Manual de Entorno de OGRE [11] y seguir algunos de los tutoriales básicos [10] de OGRE, para familiarizarse con el entorno del motor gráfico.

Finalmente y tras realizar los primeros pasos indicados, se aconseja ver qué librerías serían necesarias para el proyecto, cuál se adapta más a las necesidades del mismo, acoplarlas a estos pequeños proyectos de prueba de OGRE e intentar trabajar un poco con ellas para ver si se adecuan a las necesidades que tendrá el proyecto, así como para las necesidades de programación.

En los tutoriales básicos [10] y en el entorno de trabajo de OGRE [11] se puede ver con detalle cómo realizar todo este proceso, por lo que no se entrará en detalles del mismo. Para la realización de este proyecto, también se creó un mini proyecto, donde probar ideas y librerías, antes de entrar en la implementación general del mismo para encontrar el menor número de errores posibles y enfocar un poco mejor el análisis y diseño del mismo.

Pero como puede ocurrir en cualquier proyecto, este proyecto ha sufrido un contratiempo bastante grande que aquí se detalla. El problema radica en la librería física para la detección de colisiones y físicos *Newton*. En un principio, se pensó en la idea de tratar al monopatín como un coche, con sus ruedas y efectos sobre las mismas, pero no pudo ser así. La librería dispone de algunas clases creadas por usuarios que implementan coches, en base a conectores. Estos coches tienen suspensión, ruedas y cuerpo, por lo que se realizó un sistema que acoplaba el monopatín a estas clases. Después de varias pruebas, se comprobó que este sistema no era válido, ya que el monopatín deslizaba demasiado. Se hicieron varios intentos con varios tipos de vehículo sin buenos resultados, con lo que se llegaba a dos posibles acciones a realizar: crear un sistema de vehículo propio y a mano, o aplicar fuerzas directamente sobre el objeto.

Al final se decidió la segunda opción porque se sabía que funcionaría bien y era la solución más rápida. La otra solución podría acarrear problemas de deslizamiento de nuevo y el periodo de implementación sería grande, por lo que quedó descartada.

Este cambio provocó pérdida de tiempo de implementación y por tanto un cambio en la planificación, además supuso cambios en el diseño y algunas partes que interactuaban con el skater.

Otro problema del juego y que no se ha conseguido solucionar del todo, es el tema de los escenarios. Al ser estos escenarios de carreteras reales [38], han aparecido bastantes problemas a la hora de pasar estos escenarios a mapa de alturas de Ogre. La primera solución adoptada fue la de buscar en Google Maps [37] la carretera a representar, recoger los mapas de relieve y con un editor gráfico como Photoshop editarlo para hacer el mapa de alturas a partir de estos mapas. Además había que dibujar el asfalto a mano, para que estuviera recto.

Figura 135: Creación de mapa de alturas. Versión 1.

El resultado valió para las primeras pruebas, pero el asfalto no quedaba muy uniforme y en el terreno en general daba unos saltos grandes, provocando que el terreno quedara no muy vistoso.

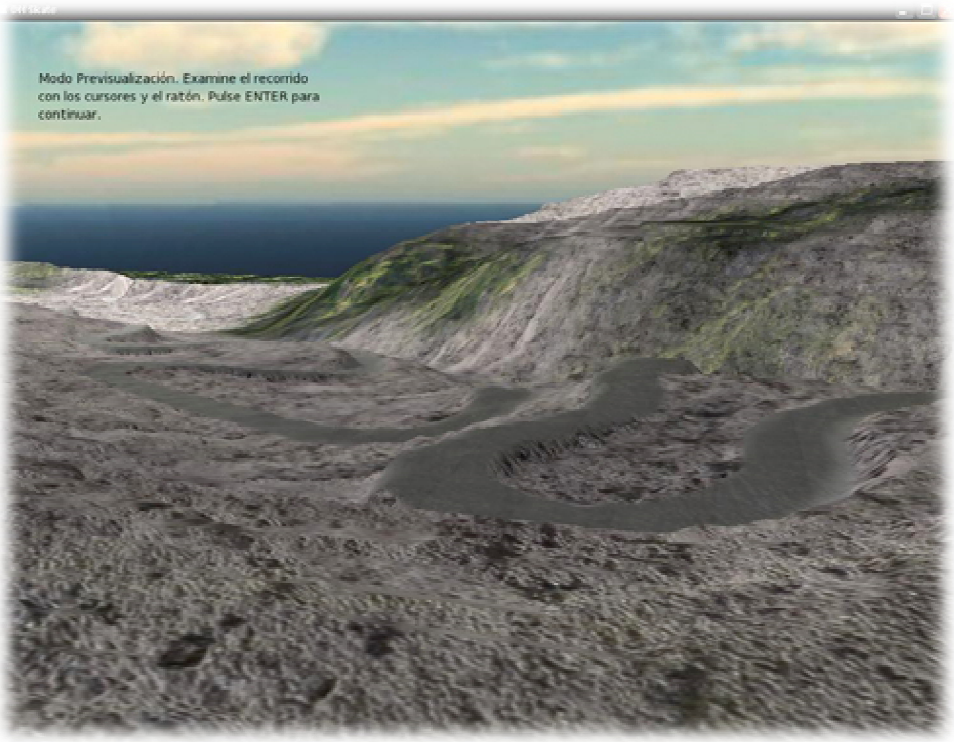

Figura 136: Captura mapas versión 1.

Al final se ha probado con otro sistema, que aunque el asfalto sigue sin estar muy uniforme, arregla en parte este problema. El sistema ha sido dibujar con Blender la carretera siguiendo el mapa de Google, darle altura con operaciones de Blender y exportarlo como mapa de alturas. Luego se han añadido los alrededores de la carretera pintando a mano. Con esto el problema se ha resuelto un poco, aunque queda pendiente para versiones futuras una solución mejor.

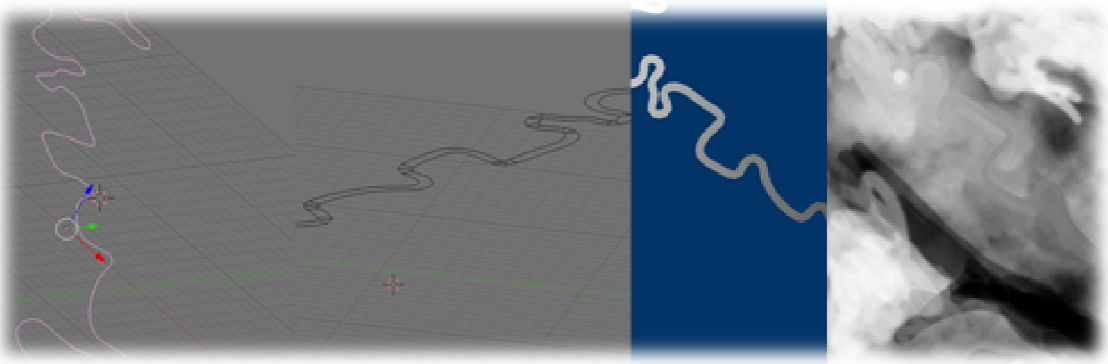

Figura 137: Creación de mapa de alturas. Versión 2.

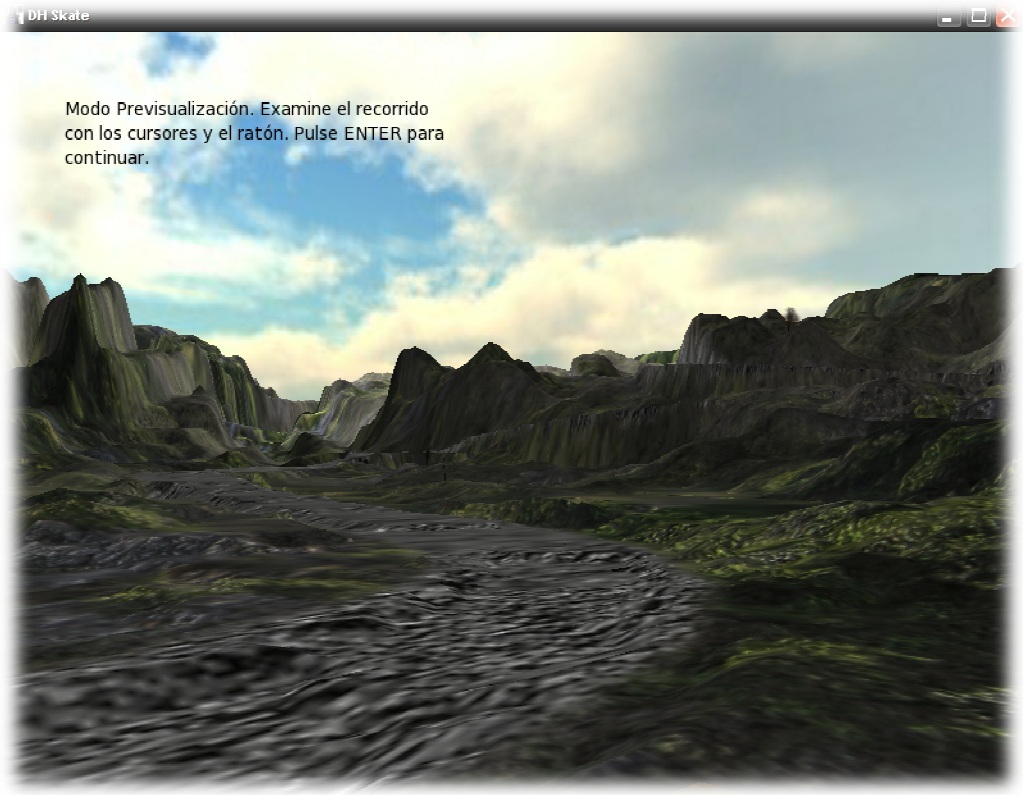

Figura 138: Captura mapa versión 2.

3.4.2 Objetos 3D con Blender

Blender es un programa de diseño 3D, que difiere bastante en cuanto al manejo del programa respecto a otros programas de diseño 3D, aunque una vez que se adquiere práctica, se pueden hacer grandes trabajos con él. Al ser software libre, existen multitud de páginas donde encontrar manuales y tutoriales, que pueden ayudar a crear objetos sencillos [7].

Para este proyecto se han tenido que desarrollar varios objetos 3D de Blender, algunos otros simplemente se han adquirido desde Internet y se han modificado o se han transformado en objetos válidos para OGRE.

Objetos

En el proyecto se han creado o modificado un total de 5 objetos: skater, tabla, eje, ruedas y alpacas. Además se han utilizado una serie de modelos de árboles de Blender para el terreno.

Una forma sencilla de modelación es importar imágenes del objeto a desarrollar que muestran su perfil, planta y alzado. Se pueden cargar en distintos planos y sirven como patrón de diseño. Usando esta técnica se han creado las partes de la tabla: tabla, ejes y ruedas.

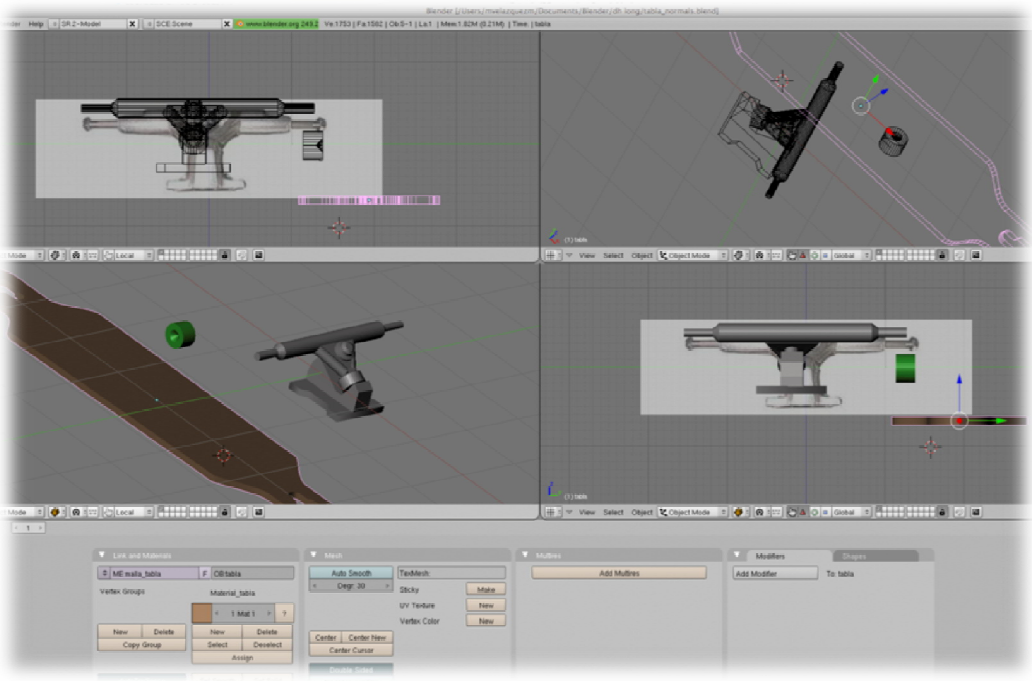

Figura 139: Captura de Blender con objetos 3D creados desde 0.

Otra método de creación de objetos 3D es partir de un cubo e ir añadiendo más cubos y cambiando su forma. Con esta técnica se ha modelado el casco del skater. El cuerpo del skater se importó por completo y se han hecho algunas modificaciones, como añadir el casco y modificar el cuerpo para dar sensación de que lleva un "mono" de motorista.

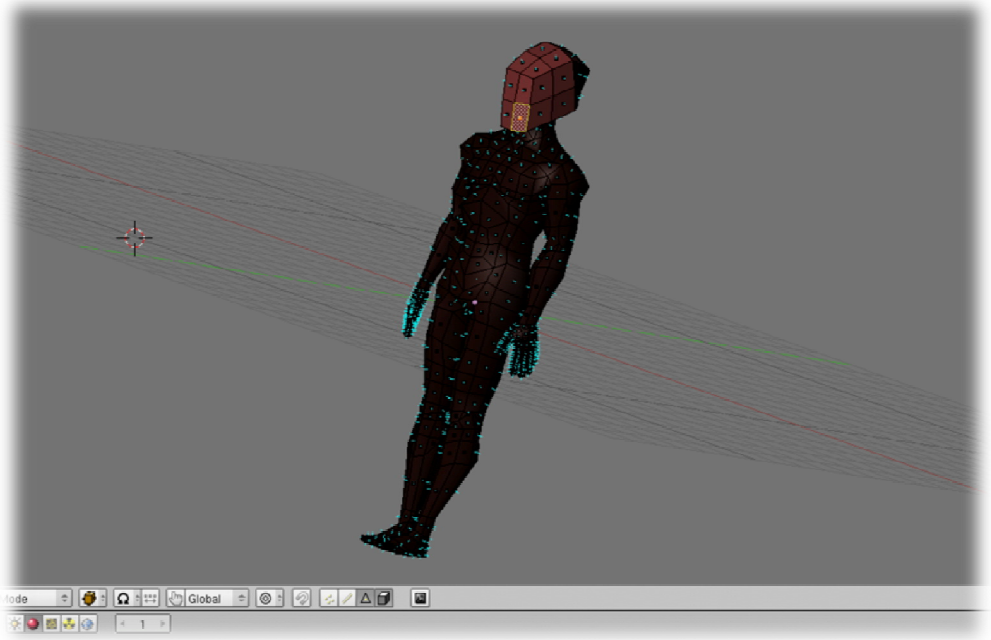

Figura 140: Captura con el patinador en Blender.

Las alpacas simplemente son un cubo, al que se le ha añadido una textura de alpaca.

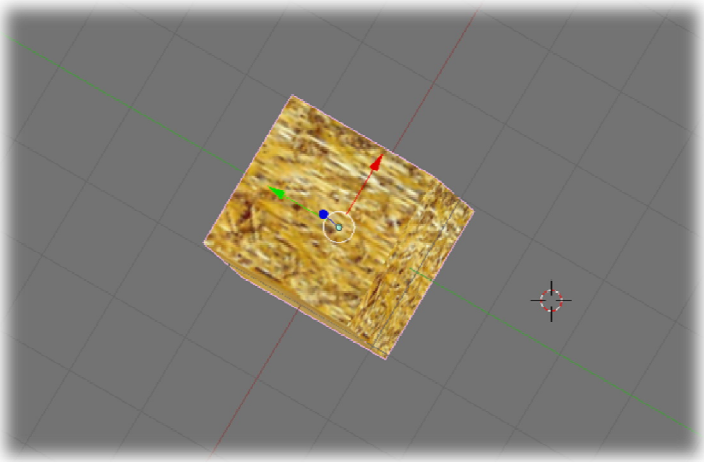

Figura 141: Alpaca con textura en Blender.

Armadura y animaciones

Una propiedad muy útil de los editores 3D para juegos, es poder modificar el aspecto del objeto en tiempo real en el juego. Para ello se utilizan las armaduras.

Las armaduras definen un “esqueleto” del objeto y los movimientos que puede tener. Los vértices del objeto se asocian a puntos de la armadura. Con lo que si se mueve un elemento de la armadura, se moverán todos los vértices asociados al mismo.

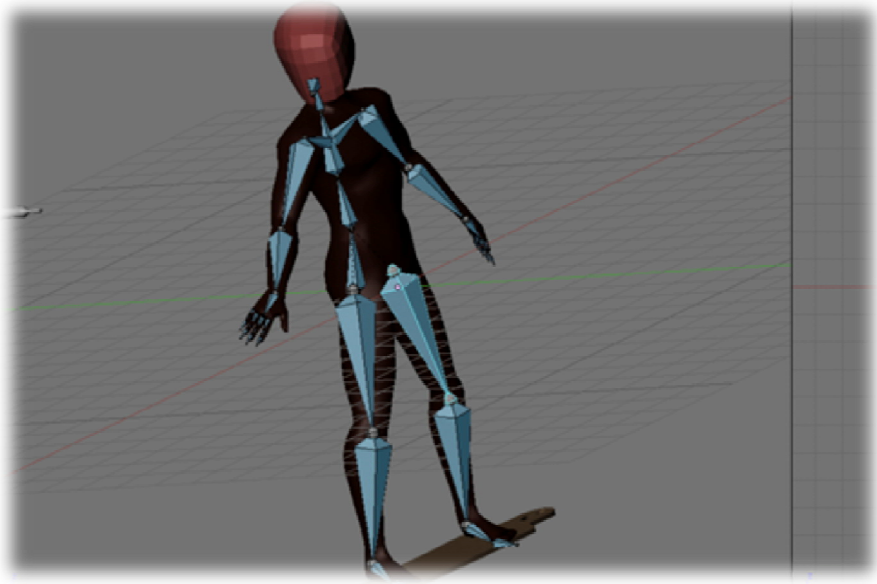

Figura 142: Patinador y su armadura (en azul).

En el proyecto, el jugador traía la armadura, por lo que sólo se ha tenido que redefinir y agregar los vértices del casco a la misma. La redefinición se debe a que OGRE sólo permite 4 “huesos” hijos por cada “hueso” padre dentro de las armaduras, y el objeto importado tenía algunos huesos con más de 4 huesos hijos. Es muy útil para no perderse en este punto, utilizar un sistema de nombrado muy descriptivo, tanto para los grupos de vértices, como para cada uno de los huesos.

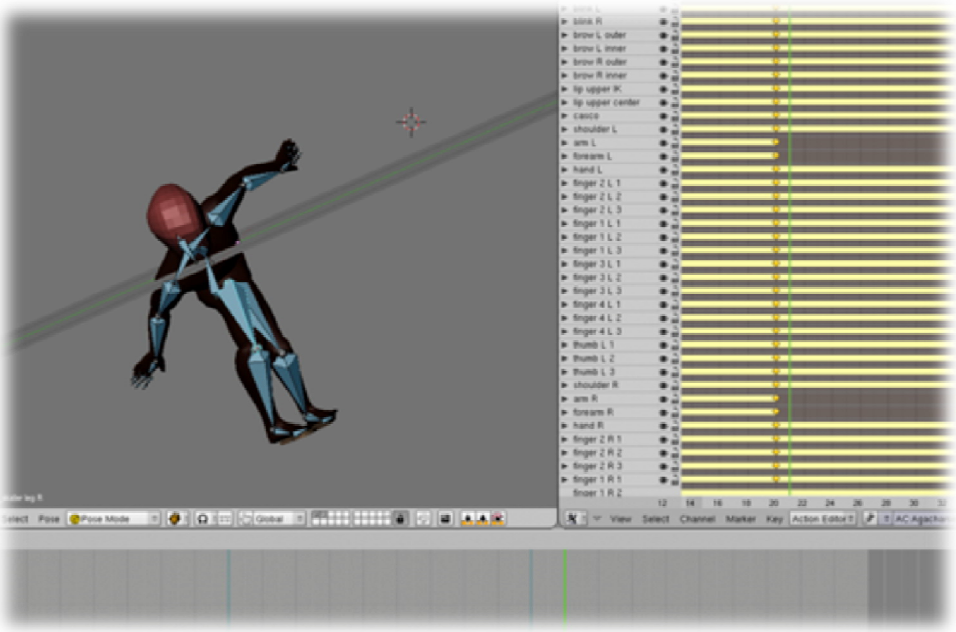

Figura 143: Animación de patinador.

Una vez que está el esqueleto del jugador bien definido, se han creado unas animaciones. Las animaciones muestran movimientos del esqueleto del objeto en el tiempo, se pueden definir muchas e importar a OGRE. Esto hace que el control del esqueleto no sea al 100% controlado por el motor gráfico, sino que el motor simplemente llama a la animación y la ejecuta. Muy útil para hacer animaciones como andar o correr. Para este jugador se han definido 11 animaciones como son agacharse, levantarse, frenar agachado, frenar en pie, derrapar, girar a izquierda o girar a derecha.

Materiales y texturas

Además de los objetos y sus animaciones, en Blender se pueden definir los materiales y/o texturas que utilizan estos. Simplemente hay que entrar en las opciones de materiales y texturas que Blender proporciona y ponerlos a gusto del usuario.

Cuando todo el objeto está terminado, hay que pasarlo a Ogre, aunque primero hay que tener varias cosas en cuenta:

- **Poner las normales hacia fuera del objeto**: Blender tiene un botón para cambiar todas las normales hacia fuera del objeto, si no se hace esto, el objeto puede parecer como transparente en OGRE.

- **Centrar el eje de coordenadas en el centro del objeto**: Blender también tiene una opción para establecer esta opción. Así será más sencillo colocar el objeto y manejarlo dentro de OGRE.
- **Convertir los objetos a mallas**: Blender exporta mallas, por lo que se tendrán que unir los objetos que se desean agrupar como una única malla y transformarlos en mallas.

Una vez realizado este proceso, con la herramienta de exportación de mallas de Blender a OGRE, se exportan las mallas y los ficheros de materiales y texturas. Con los ficheros de mallas y usando la herramienta de *OgreXMLConverter*, se convierten los objetos 3D xml obtenidos en objetos mesh de OGRE. Los objetos *mesh* son las mallas con las que trabaja OGRE.

3.4.3 Núcleo del proyecto

El núcleo del proyecto se ha construido en torno a la clase del Framework *BasicApplication*, y la clase *DHSkate* que hereda del primero y es el núcleo de este proyecto concreto.

En el núcleo del proyecto genérico (*BasicApplication*) deben estar las clases principales de las librerías utilizadas y las clases principales del proyecto, intentando ser lo más genérico posible, para poder exportar este núcleo a otros proyectos.

La segunda clase que forma el núcleo es la clase *DHSkate*, que hereda de la primera y contiene los enlaces con todos los elementos particulares del proyecto.

El proceso que se ha realizado en la clase *BasicApplication* es sencillo: se declaran los elementos principales del programa, se declaran los métodos de inicialización de los mismos, se implementan algunos de estos métodos y otros se dejan para que las clases que hereden de esta los implementen y finalmente se implementa el método que contiene las llamadas a todas las inicializaciones y la llamada al bucle de renderización de OGRE.

```
void BasicApplication::go(void) {
     createRoot();
     defineResources();

     setupRenderSystem();
     createRenderWindow();
     initializeResourceGroups();
     setupAudio();
     setupScene();
```

```
     setupGUI();
     setupInputSystem();
     createFrameLPhysics();
     startRenderLoop();
}

//Bucle de renderización
void BasicApplication::startRenderLoop() {
     mRoot->startRendering();
}
```

Código 5: Métodos principales donde inicializa recursos y comienza el bucle de renderizado. BasicApplication.

Otro de los aspectos a destacar es que en esta clase se inicializan las rutas de los recursos, y se lanza la pantalla de configuración inicial de usuario, donde el usuario puede elegir drivers de renderización, resolución del juego y si se ejecutará en pantalla completa o en modo ventada. También se puede crear un sistema de logs para comprobar estados de variables y otros datos.

```
void BasicApplication::setupRenderSystem() {
     mRoot->showConfigDialog();
}

void BasicApplication::defineResources() {
     Ogre::String secName, typeName, archName;
     Ogre::ConfigFile cf;
     cf.load("resources.cfg");

       Ogre::ConfigFile::SectionIterator seci = cf.getSectionIterator();
       while (seci.hasMoreElements()) {
          secName = seci.peekNextKey();
          Ogre::ConfigFile::SettingsMultiMap *settings = seci.getNext();
         Ogre::ConfigFile::SettingsMultiMap::iterator i;

          for (i = settings->begin(); i != settings->end(); ++i) {
             typeName = i->first;
            archName = i->second;
Ogre::ResourceGroupManager::getSingleton().addResourceLocation(archName,
typeName, secName);
          }
    }
}

void BasicApplication::initializeResourceGroups() {
     Ogre::TextureManager::getSingleton().setDefaultNumMipmaps(5);
     Ogre::ResourceGroupManager::getSingleton().initialiseAllResourceGroups
();
     #ifdef _DEBUG
mDebugLog = Ogre::LogManager::getSingleton().createLog("DebugLog.txt");
//Inicializamos el log ....
     #endif // _DEBUG
```

Código 6: Algunos métodos del núcleo. BasicApplication.

3.4.4 Mapas del juego

Los mapas del juego se han creado usando las propiedades de OGRE de Terreno y construcción mediante heightmap (mapa de alturas).

Los heightmap son mapas de alturas representados por imágenes en escala de grises que representan la altura máxima blanca y la altura mínima en negro.

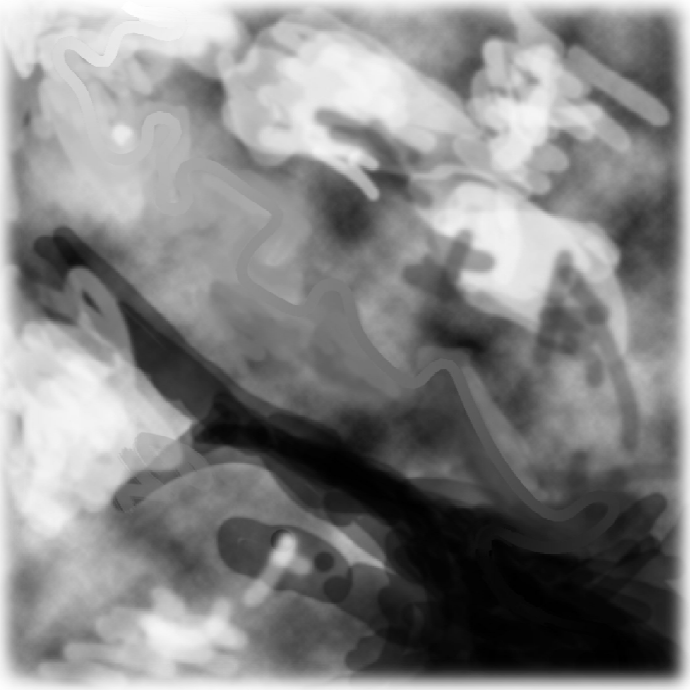

Figura 144: Heightmap de ejemplo.

OGRE contiene un sistema de creación de escenario muy sencillo y útil. Se le puede asignar un heightmap, junto con el tamaño del mundo en unidades de OGRE y la variación de la altura del mundo, con lo que quedaría definido todo el terreno.

```
/* Creamos el terreno de juego */

void DHSkate::scene(void) {
      mGlobals = OGRE_NEW Ogre::TerrainGlobalOptions();
      mTerrain = OGRE_NEW Ogre::Terrain(mSceneMgr);

     std::ifstream inp;
     std::ofstream out;
     std::string myCache;

     myCache = "media/terrainCache" + track->getId();
      inp.open(myCache.c_str(), std::ifstream::in);
     inp.close();

     if (inp.fail()) {
           //El terreno no está en cache, se crea de nuevo
```

```
            inp.clear(std::ios::failbit);

            Ogre::Image img;
            img.load(track->getHeightImg(),
Ogre::ResourceGroupManager::DEFAULT_RESOURCE_GROUP_NAME);

              Ogre::Terrain::ImportData imp;
            imp.inputImage = &img;
            imp.terrainSize = 513;
            imp.worldSize = track->getWorldSize();
            imp.inputScale = track->getInputScale();
            imp.inputBias = 0;
            imp.minBatchSize = 33;
            imp.maxBatchSize = 65;
```

Código 7: Creación del terreno con el heightmap. DHSkate::scene.

OGRE aporta un método para añadir texturas al terreno de forma muy sencilla. Una textura estará definida por una pequeña imagen que representa la textura, una imagen de la normal de la textura, para dar relieve a la misma y dotarla de sensación de rugosidad, una imagen en escala de grises que actúa de máscara en el terreno: blanco se aplica la textura al 100%, negro no se aplica la textura en ese punto del terreno, además se proporciona el tamaño que tendrá la textura en el mundo. Si es un tamaño pequeño, la textura se repite por todo el mundo.

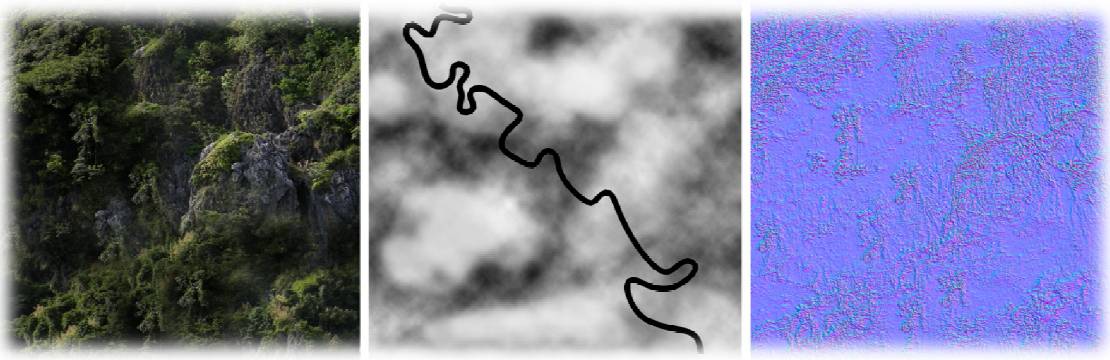

Figura 145: Textura, máscara y normal.

Combinando varias texturas se pueden conseguir efectos muy realistas sobre el terreno a desarrollar, en el proyecto se usan tres texturas, más una básica por defecto.

```
       imp.layerList[0].worldSize = 800;
       imp.layerList[0].textureNames.push_back("default.jpg");
       imp.layerList[0].textureNames.push_back("default_normal.jpg");

       imp.layerList[1].worldSize = 400;
     imp.layerList[1].textureNames.push_back(track->getLayer1Img());
      imp.layerList[1].textureNames.push_back(track->getLayer1Normal());
```

Código 8: Carga de dos capas de texturas. DHSkate::scene.

Además Ogre permite guardar una caché del terreno en un fichero físico, para ahorrar tiempos de carga. Bien por el método normal, o por el método de caché, el terreno queda cargado en el sistema.

```
mTerrain->prepare(imp);
mTerrain->load();

myCache = "terrainCache" + track->getId();
mTerrain->save(Ogre::String(myCache));
```

Código 9: Carga y guardado en caché del terreno. DHSkate::scene.

Aunque en este punto el terreno y sus capas están definidos, hay que utilizar unas clases de mezcla de capas, para realizar la mezcla de las texturas en los distintos puntos del terreno. Para poder aplicar las texturas de la forma indicada en las imágenes de las máscaras, hay que usar unas clases adicionales de mezcla de textura, así como métodos adicionales de OGRE para realizar un parseo de la posición en la imagen de máscara a la posición real en el terreno.

```
Ogre::TerrainLayerBlendMap *blendMap1 = mTerrain->getLayerBlendMap(1);
Ogre::TerrainLayerBlendMap *blendMap2 = mTerrain->getLayerBlendMap(2);

Ogre::Image image1;
image1.load(track->getLayer1Mask(),
Ogre::ResourceGroupManager::DEFAULT_RESOURCE_GROUP_NAME);
Ogre::Image image2;
image2.load(track->getLayer2Mask(),
Ogre::ResourceGroupManager::DEFAULT_RESOURCE_GROUP_NAME);

image1.resize(blendMapSize, blendMapSize);
image2.resize(blendMapSize, blendMapSize);

float* pBlend1 = blendMap1->getBlendPointer();
float* pBlend2 = blendMap2->getBlendPointer();

Ogre::uint16 id = 0;

for (Ogre::uint16 y = 0; y < mTerrain->getLayerBlendMapSize(); ++y) {
       for (Ogre::uint16 x = 0; x < mTerrain->getLayerBlendMapSize(); ++x) {
       Ogre::Real val1;
          Ogre::Real val2;

         Ogre::ColourValue col1 = image1.getColourAt(x, y, 0); //color en
mask
          Ogre::ColourValue col2 = image2.getColourAt(x, y, 0); //color en
mask
          val1 = col1.r;
          val2 = col2.r;

        *pBlend1++ = val1; //indica la cantidad de textura 1 a aplicar
          *pBlend2++ = val2; //indica la cantidad de textura 2 a aplicar
     }
}

blendMap1->dirty(); //actualiza
blendMap1->update();

blendMap2->dirty();
blendMap2->update();
```

Código 10: Mezclado de texturas según máscaras. DHSkate::scene.

Con estos métodos, se puede además definir la posición inicial y final del jugador en el terreno y colocar elementos adicionales como alpacas o árboles, usando también imágenes de máscaras.

```
if (track->getLayer5Mask() != "") {
       Ogre::ColourValue colo = imageo.getColourAt(x,y,0);
       if (colo.r == 1) { //Si en la máscara es blanco, pone el árbol.
          Ogre::Real txo, tyo;
          Ogre::Vector3 pointo;
          blendMap1->convertImageToTerrainSpace(x, y, &txo, &tyo); //parseo

         //Obtiene la altura del terreno en el punto localizado
     Ogre::Real heighto = mTerrain->getHeightAtTerrainPosition(txo, tyo);
          mTerrain->getPosition(txo, tyo, heighto, &pointo);

     //Coloca en el escenario el arbol como un SceneNode y Entity.
          putForestObstacule(pointo, 1);
       }
}
```

Código 11: Colocación de árboles según máscara. DHSkate::scene.

En el proyecto, hay varios tipos de árboles, y cuando se llama a crear el objeto 3D, se elige uno de forma aleatoria, para que el terreno acabe con distintos tipos de árboles sobre él. Los árboles y alpacas forman parte de geometría estática de OGRE, una forma de definir objetos 3D que quedarán estáticos durante toda la renderización. El uso de geometría estática para estos objetos en lugar de nodos y entidades comunes, conlleva una mejora en el rendimiento del sistema substancial.

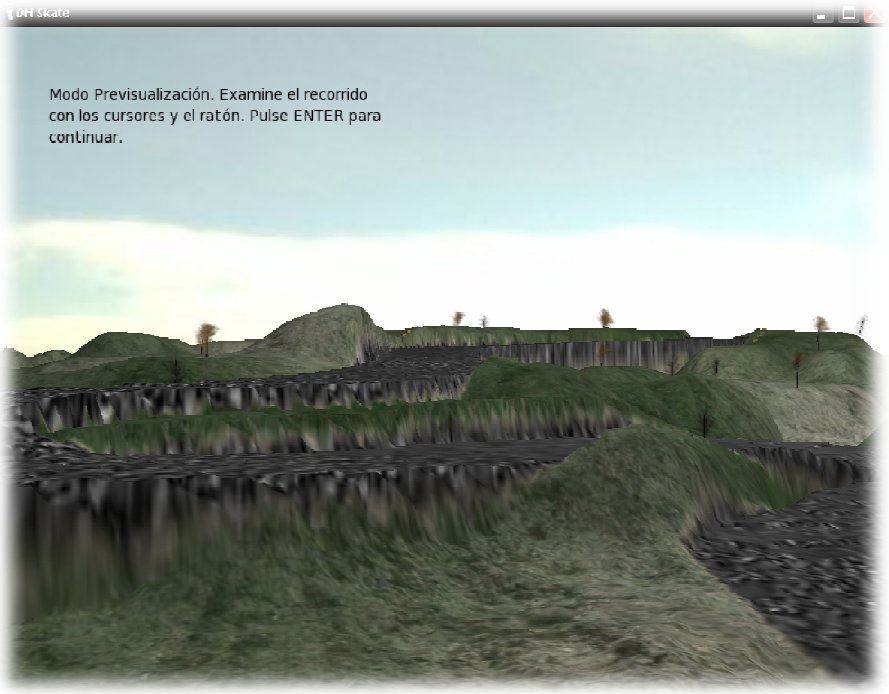

Figura 146: Muestra de un circuito.

Una vez creado todo el terreno, se le pueden añadir luces y sombras según las propiedades de Ogre, además se puede añadir una imagen para el cielo y los alrededores del terreno con funciones básicas, como se ha hecho en este proyecto, obteniendo un resultado como el siguiente:

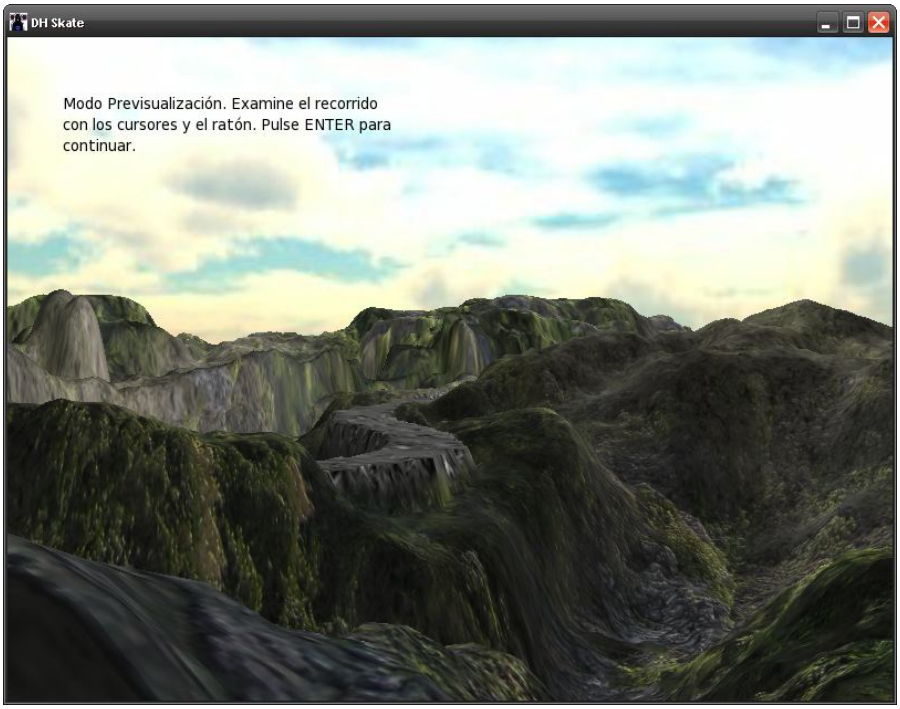

Figura 147: Otra captura de terreno.

3.4.5 Lectura y escritura de ficheros

En el proyecto se ha desarrollado un sistema de almacenaje de datos mediante xml, en unos archivos de extensión .dhsk. En estos ficheros se guarda la información de opciones del juego, la información de patinadores y la información de circuitos.

En referencia al rendimiento, el uso de ficheros xml no es muy apropiado, ya que para leer un valor determinado, hay que buscar la etiqueta apropiada. Además hace que datos sensibles, como en este caso los records de los circuitos, queden muy a la vista de los usuarios, ya que el etiquetado hace referencia a los datos.

Sin embargo hay una causa por la que el uso de xml se hace apropiado, y es para poder permitir a los usuarios avanzados o incluso a los propios desarrolladores incorporar nuevos circuitos y jugadores al juego de una forma muy sencilla.

En este proyecto y con el fin de evitar las situaciones de cambio de datos sensibles, se ha introducido un sistema de seguridad, basado en un algoritmo propio similar a un código hash. El proceso es sencillo, para cada línea de un xml con datos sencillos se genera este código hash. Cuando se abre el xml con el proyecto, se lee la línea y se calcula el hash. Si el hash obtenido y el hash que contiene el xml para esa línea son iguales, entonces los datos contenidos en la línea son correctos. En caso contrario se desecha el xml y se toma por no válido.

Para el proyecto se han diseñado 3 tipos de xml: uno para los parámetros generales del juego, uno para circuitos y uno para patinadores. Estos están asociados a sus respectivas clases en el proyecto: *GameParameters*, *Track* y *MiniSkater* respectivamente.

Cuando se construye una de estas clases, se lee el fichero xml con una librería adicional llamada *tinyxml*. Se va recorriendo el fichero y para cada conjunto de datos que representa una línea, se manda al sistema de seguridad.

```
Track::Track(Ogre::String file) {
     valid = false;
     int key = 0;
     mFile = file;

     SecurityCheck *seguridad = new SecurityCheck(); //Carga seguridad
     doc.LoadFile(("media/gameData/tracks/" + file).c_str()); //carga
fichero

       Ogre::StringVector sv = Ogre::StringUtil::split(file,".");
     if (sv.size() == 2)
```

```
            id = sv[0]; //Obtiene identificador del circuito (nombre
fichero)

     //Empieza la lectura de propiedades del xml
     TiXmlElement* track = doc.FirstChildElement("track");
     if (!track) return;

     TiXmlElement* data = track->FirstChildElement("data");
     if (!data) return;

     name = (Ogre::String)data->Attribute("name"); //campo nombre
     description = (Ogre::String)data->Attribute("description"); //campo
des
     data->QueryIntAttribute("key",&key); //Obtiene el hash de la línea

     //llama al chequeo de seguridad con los datos de la línea y el hash
     if (!seguridad->checkData(name,description,key)) return;

     ...
```

Código 12: Lectura de fichero xml. Track::Track.

El sistema de seguridad se ha implementado en la clase SecurityCheck, siguiendo el proceso especificado en el diseño del sistema.

```
//Chequeo de seguridad para línea con dos valores.
bool SecurityCheck::checkData(Ogre::String data1, Ogre::String data2, int
key){
      return ((getValorString((std::string)data1) +
getValorString((std::string)data2)) == key); //si la suma == hash, bien
}

//Obtiene el valor hash de un carácter.
int SecurityCheck::getValorKey(char c) {
  char mkey[] = "0123456789abcdefghijklmnopqrstuvwxyz.-_/";
  int i = 0;
  char * pch;
  pch = strchr(mkey,c);
  if (pch == 0) return 40;
  else return pch-mkey;
}

//Obtiene el valor de dos caracteres (primero y último de un valor)
int SecurityCheck::getValorData(char c0, char c1){
  return (getValorKey(c0) + getValorKey(c1)) * X;
}

//Obtiene el valor hash de un valor, usando si primer y último carácter.
int SecurityCheck::getValorString(std::string s){
     if (s.length() == 0) return 0;
     return getValorData(s.at(0), s.at(s.length()-1));
}
```

Código 13: Proceso de comprobación de seguridad. SecurityCheck.

3.4.6 Físicos

Los sistemas físicos suelen tener clases que representan cuerpos, y se asocian a los nodos de OGRE. *Newton* funciona de la misma forma: se define un mundo de físicos, y se crean objetos físicos que se asocian a los nodos de OGRE para definir las colisiones entre ellos y las fuerzas que reciben.

Para el proyecto, el mundo de Newton se crea en la clase del núcleo *DHSkate*. El mundo debe de ocupar lo que ocupa el escenario más unos valores adicionales, para quedar seguros que todo objeto en el escenario forma parte del mundo físico.

A la hora de definir las dimensiones del Mundo hay que tener en cuenta que el centro de coordenadas del mundo físico se sitúa en (0,0,0) y que se expande a su alrededor, por lo que para definir los valores mínimos y máximos de los ejes hay que hacer algunos cálculos con el tamaño del terreno.

```
mWorld = new OgreNewt::World();
Ogre::Real minMaxWidth = (track->getWorldSize() + 20) / 2;
Ogre::Real minMaxHeight = (track->getInputScale() + 20);

mWorld->setWorldSize(Ogre::Vector3(-minMaxWidth,-minMaxHeight,-
minMaxWidth),Ogre::Vector3(minMaxWidth,minMaxHeight,minMaxWidth));
```

Código 14: Creación del mundo físico y redimensionado. DHSkate::applyPhysics.

Una vez creado el mundo físico, hay que asignar un FrameListener al núcleo de OGRE, para que se actualicen los objetos del mundo físico.

```
Ogre::RenderWindow *win = mRoot->getAutoCreatedWindow();
mNewtonListener = new OgreNewt::BasicFrameListener(win, mWorld, 300);
mRoot->addFrameListener(mNewtonListener);
```

Código 15: Definición del FrameListener de los físicos. DHSkate::applyPhysics.

Cuando está el mundo en general establecido, hay que crear los sistemas de objetos físicos, llamados cuerpos en Newton. El proceso normal de Newton para crear un objeto físico es:

- Crear un puntero de colisiones usando un objeto primitivo como una caja o esfera, o un objeto no primitivo a partir de una Entidad o malla 3D.
- Con el puntero de colisiones se crea el cuerpo, de la clase OgreNewt::Body.
- Para finalizar se asocia el cuerpo con un Nodo de OGRE.

```
OgreNewt::Body* skateBody;

OgreNewt::ConvexCollisionPtr col = OgreNewt::ConvexCollisionPtr(new
OgreNewt::CollisionPrimitives::Box(world, Ogre::Vector3(1.8f, 1.0f, 2.8f),
1, Ogre::Quaternion::IDENTITY)); //Crea una caja

skateBody = new OgreNewt::Body (world, col); //Cuerpo
skateBody->setPositionOrientation(iniPos, iniRot);

body->attachNode(skaterRoot);
```

Código 16: Creación de un cuerpo del mundo físico. Skater::create.

En el proyecto se ha utilizado la creación de un cuerpo con forma de caja para representar al jugador y se ha asociado a un nodo sin entidad, por lo que sobre la caja actúan los efectos físicos, pero esta no es visible en el mundo de OGRE. También se ha creado otro cuerpo con la forma del terreno de juego, para que interactúe con el cuerpo del jugador.

```
OgreNewt::CollisionPtr hf = OgreNewt::CollisionPtr(new
OgreNewt::CollisionPrimitives::HeightField(mWorld, mTerrain, 0));

OgreNewt::Body* hfBody = new OgreNewt::Body(mWorld,hf);

Ogre::SceneNode* hfNode = mSceneMgr->getRootSceneNode()->
createChildSceneNode();

hfBody->attachNode(hfNode);
hfBody->setPositionOrientation(Ogre::Vector3(-minMaxWidth + 10,0,-
minMaxWidth + 10), Ogre::Quaternion::IDENTITY);
```

Código 17: Creación del cuerpo del terreno. DHSKate::applyPhysics.

Sobre los objetos físicos, se pueden realizar gran cantidad de operaciones como cálculos de masa, matriz de inercia o posicionamiento del cuerpo. Esta última sólo se ha de usar cuando se crea el cuerpo, y luego moverle gracias a la aplicación de fuerzas, rotaciones o colisiones.

Además *Newton* tiene otras muchas características, como definir anclajes (*OgreNewt::Joint*) para hacer enlaces entre cuerpos como bisagras o ruedas, entre otros. Otra opción es establecer pares de materiales y el modo de colisión entre los mismos, para definir las colisiones por ejemplo entre cuerpos gelatinosos y cuerpos rígidos. Estas propiedades se utilizaron en las primeras versiones físicas del juego, pero al no conseguir el efecto deseado se descartaron.

Con los cuerpos físicos definidos y creados, sólo hay que aplicar fuerzas sobre el cuerpo que representa al patinador en un *FrameListener*. El FrameListener utilizado para esta causa ha sido *PhysicsListener*, que define un método de refresco del patinador al que se le aplican diferentes fuerzas según las teclas pulsadas por el usuario. El método de refresco en muchas

ocasiones se trata simplemente de la aplicación de la fuerza de la gravedad. En el proyecto, se ha definido e implementado un método propio, para poder aplicar distintas fuerzas.

```
void PhysicsListener::startPhysics() {
     mPlayer->getBody()->setLinearDamping(0.01f);
     mPlayer->getBody()->
setCustomForceAndTorqueCallback<PhysicsListener>(&PhysicsListener::skateCal
lback, this); //Método personalizado de actualización del cuerpo Newton
      mPlayer->getBody()-
>setAngularDamping(Ogre::Vector3(0.02f,0.02f,0.02f));
}
```

Código 18: Establece el método de actualización del cuerpo. PhysicsListener::startPhysics.

La forma en que se aplican las fuerzas en el método personalizado, se ha explicado en el apartado de diseño. Para aplicar la fuerza, hay que declararla y aplicarla en algún punto del cuerpo. Destacan dos formas de aplicar fuerzas a un objeto, una es una fuerza directa, que actúa sobre todos los puntos del cuerpo, como puede ser la gravedad. El otro tipo de fuerzas se añaden indicando el punto sobre el que se ejerce la misma, y puede provocar rotaciones sobre el objeto, como puede ser una fuerza de empuje.

```
Ogre::Real mass;
Ogre::Vector3 inertia;

body->getMassMatrix(mass,inertia);
Ogre::Vector3 gForce(0, -9.8, 0);
gForce *= (mass); //Fuerza de la gravedad
Ogre::Vector3 fEmpuje(0,0,1500); //Fuerza de empuje.

Ogre::Vector3 gravityPoint = body->getCenterOfMass(); //Punto del centro de
masas
Ogre::Vector3 tailTurnPoint = gravityPoint + Ogre::Vector3(0,0,-3); //Otro
punto

body->addForce(gForce); //Fuerza global sin rotación.
body->addLocalForce(fEmpuje, gravityPoint); //Fuerza local, con posible
rotación.
body->addLocalForce(fEmpuje, tailTurnPoint); //Código añadido como ejemplo.
```

Código 19: Uso de fuerzas sobre el patidador. PhysicsListener::skateCallback.

El código anterior, sólo muestra cómo declarar y crear una fuerza, si se comprueba el método en los fuentes del proyecto, se puede ver cómo se implementa la tabla de fuerzas según la posición del patinador y las teclas pulsadas por el jugador que se ha declarado en el diseño y cambia el valor de las fuerzas aplicadas, así como el número de ellas que afectan al cuerpo.

3.4.7 Entrada de teclado y ratón

El paquete IOS que viene junto con OGRE es un gran paquete para la gestión de entrada y salida de dispositivos de control como pueden ser teclado, ratón o joystick. Normalmente la entrada/salida debe de acoplarse a un *FrameListener* de OGRE y sólo puede haber un registro de captura de eventos de teclado, ratón y joystick. Es por esto que en el proyecto, se han asociado las entradas de ratón y teclado a la clase del Framework general *IoControls*.

```
//Recoge algunos parámetros de la ventana de renderizado
win->getCustomAttribute("WINDOW", &windowHnd);
windowHndStr << windowHnd;
pl.insert(std::make_pair(std::string("WINDOW"), windowHndStr.str()));

//Crea el manejador de entrada de IOS
mInputManager = OIS::InputManager::createInputSystem(pl);

//Crea los objetos de entrada de teclado y ratón
mKeyboard = static_cast<OIS::Keyboard*>(mInputManager->
createInputObject(OIS::OISKeyboard, true));

mMouse = static_cast<OIS::Mouse*>(mInputManager->
createInputObject(OIS::OISMouse, true));

//Establece la clase actual, como Listener de eventos de teclado y ratón
mKeyboard->setEventCallback(this);
mMouse->setEventCallback(this);

const OIS::MouseState &mouseState = mMouse->getMouseState();
mouseState.width = 1024; // Anchura del área de renderizado
mouseState.height = 768; // Altura del área de renderizado

//Añade esta clase como Listener de eventos de la ventana de renderizado
Ogre::WindowEventUtilities::addWindowEventListener(win,this);
```

Código 20: Declaración del sistema de entrada. IoControls::preparaConfiguracion.

Los eventos de entrada recogidos por esta clase, se traspasan a las clases competentes, para diversificar el código, según el estado en el que se encuentra el juego. El proceso es simple, hay que capturar eventos de teclado y ratón en los métodos de captura por frame.

```
bool IoControls::frameRenderingQueued(const Ogre::FrameEvent& evt){
     mMouse->capture(); //Captura eventos de ratón y teclado
     mKeyboard->capture();
```

Código 21: Captura de teclado y ratón. IoControls::frameRenderingQueued.

Y declarar los métodos en los que capturar los eventos y pasarlos a las clases competentes. Hay muchos métodos de evento de teclado y ratón, como pulsación de tecla, suelte de tecla, moviendo de ratón, etc.

```
//Control de pulsación de tecla
bool IoControls::keyPressed(const OIS::KeyEvent &arg) {
//Si estado == MENUS, pasa la pulsación al MyGUI, control en menús
```

```
if (DHSkate::getSingleton().getState() == 0)
      mGui->injectKeyPress(MyGUI::KeyCode::Enum(arg.key), arg.text);

//Si estado == JUGANDO ó estado == CUENTA ATRAS y no es la tecla de salida,
   //se manda al listener de físicos.
if ((DHSkate::getSingleton().getState() == 1 ||
DHSkate::getSingleton().getState() == 3) && arg.key != OIS::KC_ESCAPE) {
      DHSkate::getSingleton().mPhysicsListener->keyPressed(arg);

if (DHSkate::getSingleton().getState() == 3) {
...
```

Código 22: Control de pulsación de tecla. IoControls::keyPressed.

3.4.8 Interfaz gráfica de usuario

Como se comentó en el diseño, la interfaz gráfica de usuario para menús e información del juego se ha realizado con una librería adicional, *MyGUI*. OGRE dispone de métodos y clases para la definición de interfaces, pero MyGUI es un poco más completa y permite realizar mejores diseños.

Una de las características más valoradas de *MyGUI* es que se pueden crear composiciones de objetos mediante ficheros de texto xml, por lo que resulta muy sencillo diseñar y manipular menús mediante ese método.

```
<?xml version="1.0" encoding="UTF-8"?>
<MyGUI type="Layout">
<Widget type="StaticImage" skin="StaticImage" position="112 64 616 536"
layer="Back">
      <Property key="Image_Texture" value="fondo.png"/>
     <Widget type="Button" skin="Button" position="250 200 96 40"
align="Center" name="bJugar">

<Property key="Text_TextAlign" value="Center"/>
     <Property key="Widget_Caption" value="JUGAR"/>
<Property key="Widget_Visible" value="true"/>
</Widget>

<Widget type="Button" skin="Button" position="250 250 96 40" align="Center"
name="bOpciones">
     <Property key="Text_TextAlign" value="Center"/>
     <Property key="Widget_Caption" value="OPCIONES"/>
     <Property key="Widget_Visible" value="true"/>
</Widget>

<Widget type="Button" skin="Button" position="250 300 96 40" align="Center"
name="bSalir">
     <Property key="Widget_Caption" value="SALIR"/>
     <Property key="Widget_Visible" value="true"/>
     <Property key="Text_TextAlign" value="Center"/>
</Widget>
</Widget>

</MyGUI>
```

Código 23: Composición MyGUI en texto. Menu1_DHSkate.layout.

Además MyGUI dispone de un proyecto llamado *LayoutEditor*, en que se pueden componer estos ficheros de forma gráfica, por lo que se ha utilizado para diseñar todos los menús del juego.

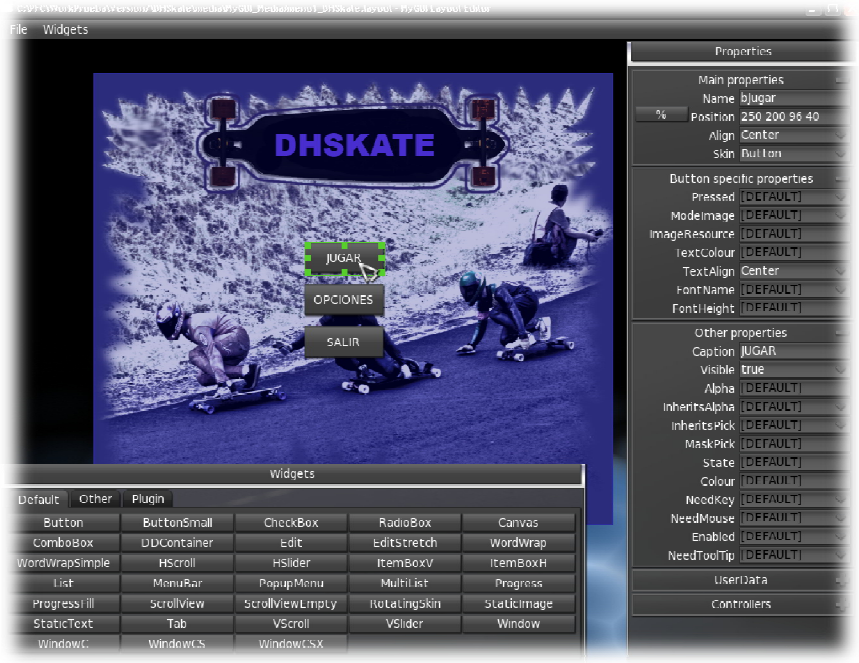

Figura 148: Captura del LayoutEditor.

Los menús creados se graban en ficheros *.layout* en la carpeta de *MyGUI_Media* y se cargan con una simple línea de código en el juego. Todo el control de MyGUI se ha realizado desde la clase general del Framework *IoControls*. Aunque de forma excepcional, la carga del menú principal y la creación de la clase general de MyGUI se han realizado desde la clase del núcleo del Framework BasicApplication.

```
void BasicApplication::setupGUI() {
     //Inicializa MyGUI
     mPlatform = new MyGUI::OgrePlatform();
     mPlatform->initialise(mRoot->getAutoCreatedWindow(),mSceneMgr);

     mGui = new MyGUI::Gui();

     Ogre::Viewport *port = mRoot->getAutoCreatedWindow()->getViewport(0);
     mGui->resizeWindow( MyGUI::IntSize(port->getActualWidth(), port-
>getActualHeight()) );

     mGui->initialise();

     //Carga el menú principal usando el xml menu1_DHSkate.layout
     MyGUI::LayoutManager::getInstance().load("menu1_DHSkate.layout");
}
```

Código 24: Inicialización de MyGUI y el menú principal. BasicApplication::setupGUI.

Cuando se crea una composición desde un xml, MyGUI crea los objetos de los elementos (Widgets) que incluye la composición, con lo que permite un control posterior de los mismos. Una vez que se ha creado el menú, hay que definir los métodos controladores de eventos de ciertos elementos, por ejemplo la pulsación con el ratón sobre un elemento botón. Por una parte se asocian los métodos a los eventos del Widget.

```
//Busca el botón salir (bSalir)
MyGUI::ButtonPtr button = mGui->findWidget<MyGUI::Button>("bSalir");
//Se asocia a la pulsación del botón el método IoControls::botonSalir
button->eventMouseButtonClick = MyGUI::newDelegate(mIoControls,
&IoControls::botonSalir);

//Se hace el mismo proceso con el botón jugar y el botón opciones.
MyGUI::ButtonPtr bJugar = mGui->findWidget<MyGUI::Button>("bJugar");
bJugar->eventMouseButtonClick = MyGUI::newDelegate(mIoControls,
&IoControls::botonJugar);

MyGUI::ButtonPtr bOpciones = mGui->findWidget<MyGUI::Button>("bOpciones");
 bOpciones->eventMouseButtonClick = MyGUI::newDelegate(mIoControls,
&IoControls::botonOpciones);
```

Código 25: Asociación de métodos a las pulsaciones de botones. BasicApplication::setupInputSystem.

Y por otra se declaran e implementan los métodos controladores del evento.

```
//Método de pulsación del botón salir del menú principal
void IoControls::botonSalir(MyGUI::WidgetPtr _sender) {
     mContinue = false; //Finalizará el bucle de renderizado y se cierra el
programa
}

//Método de pulsación del botón de opciones del menú principal
void IoControls::botonOpciones(MyGUI::WidgetPtr _sender) {
     initMenuOptions(); //Inicializa el menú de opciones
}

//Método de pulsación del botón de jugar del menú principal
void IoControls::botonJugar(MyGUI::WidgetPtr _sender) {
     initMenu2(); //Inicializa el menu 2.
}
```

Código 26: Implementación de métodos para botones del menú principal. IoControls

Cuando se cambia de un menú a otro, hay que tener el cuidado de destruir todos los elementos que el menú anterior creó, para que no haya problemas en la visualización y en la declaración de nombres de los elementos, para evitar duplicidades.

```
void IoControls::nextGuiMenu(Ogre::String fileName) {
  MyGUI::Gui::getInstance().destroyAllChildWidget();      //destruye los
elementos
  MyGUI::LayoutManager::getInstance().load(fileName); //Carga Nuevo layout

  showGui(); //Muestra el Nuevo layout
}
```

Código 27: Cambio de menú. IoControls::nextGuiMenu.

Además de gestionar menús creados en xml, con *MyGUI* se pueden crear elementos aislados en el código. En el proyecto se ha utilizado esta característica para mostrar la cuenta atrás antes de empezar la partida, mostrar mensajes de alerta cuando el usuario pulsa una tecla y está aún la cuenta atrás, mostrar la velocidad del patinador y el tiempo de juego, además de mostrar el texto descriptivo en la pre-visualización del terreno.

```
void IoControls::muestraCartelCorr(int posInit) {
  //Activamos MyGUI con una imagen según el contador actual
MyGUI::StaticImagePtr initCount = mGui->
findWidget<MyGUI::StaticImage>("initCount");

//Establece la imagen
if (posInit == 0)
     initCount->setImageTexture("0.png");
  if (posInit == 1)
     initCount->setImageTexture("1.png");
  if (posInit == 2)
     initCount->setImageTexture("2.png");
  if (posInit == 3)
     initCount->setImageTexture("3.png");

...
  initCount->setImageCoord(MyGUI::IntCoord(0, 0, 60, 85)); //Posición
initCount->setVisible(true); //Lo hace visible
}
```

Código 28: Creación de elementos con código. IoControls::muestraCartelCorr.

En el proyecto se han utilizado imágenes, botones, deslizadores, cajas de chequeo, cajas de combo y carteles de texto.

Figura 149: Menú de fin de prueba.

Figura 150: Menú de selección de patinador y circuito.

3.4.9 Sonidos

El control de los sonidos en el juego se ha desarrollado en la clase del Framework general *SoundManager*. Esta utiliza la librería externa *cAudio*, que permite la reproducción de varios sonidos a la vez y aplicar sobre los mismos muchos efectos como looping, o efectos de sonido 3D, todo ello sin que el desarrollador tenga que controlar los **threads** de reproducción.

En el momento de construir la clase de sonidos, se han cargado todos los sonidos posibles: la canción de fondo, el sonido de las ruedas, el bip de la cuenta atrás y el sonido de los derrapes.

```
manager = cAudio::createAudioManager(true);

//Inicializamos los sonidos y efectos
cAudio::IAudioSource* mySound = manager-
>create("music","media\\sounds\\music\\rarae.ogg",true);
```

Código 29: Inicialización de cAudio y carga de sonido. SoundManager::SoundManager.

En la clase se han implementado varios métodos públicos para reproducir y parar sonidos. Uno básico para reproducir la música de fondo y pararla, y otro similar para reproducir y parar un efecto.

```
//Reproduce un efecto de sonido
void SoundManager::playEffect(Ogre::String effectName, bool loop) {
       if (mVolume != 0) {
         cAudio::IAudioSource* myEffect = manager->
getSoundByName(((std::string)effectName).c_str());

                   myEffect->setVolume(float(mVolume / 2));
                   myEffect->play2d(loop);
     }
}

//Detiene la reproducción de un efecto
void SoundManager::stopEffect(Ogre::String effectName) {
     cAudio::IAudioSource* myEffect = manager->
getSoundByName(((std::string)effectName).c_str());
     myEffect->stop();
}
```

Código 30: Reproducción y parada de un efecto. SoundManager.

La clase también tiene unos métodos algo más complejos para reproducir el sonido de las ruedas con un efecto 3D. Si el patinador va a cierta velocidad o gira hacia un lado u otro, la posición del origen del sonido cambiará, para hacer un efecto de movimiento en el juego.

```
//Reproduce un sonido en movimiento
void SoundManager::playRunEffect(int turn, int speed) {
     int MOVEVAL = 80; //Cantidad de movimiento del sonido
     if (mVolume != 0) {
```

```
            cAudio::IAudioSource* runEffect = manager-
>getSoundByName("run");
            if (!runEffect->isPlaying()) {
                  runEffect->setVolume(float(mVolume / 2));
                  runEffect->play3d(cAudio::cVector3(0,0,0), 1.0f, true);
            }

            if (turn == 0)
                  movez = float(-1 * movez);
      ... //Cálculos de los desplazamientos del sonido

             cAudio::cVector3 pos = (movex,0.0, movez);
             runEffect->move(pos); //Desplaza la posición del efecto.
```

Código 31: Reproducción de un efecto de movimiento. SoundManager::playRunEffect.

El cálculo del desplazamiento ha sido algo difícil, aunque utilizando algunos comandos, se consigue que en los giros, aceleraciones y frenos, la posición del sonido cambie.

3.4.10 Luces y cámaras

Las cámaras y las luces son parte muy importante en cualquier juego y en OGRE sin ambos, no se produciría renderización alguna, con lo que no se vería nada en el juego.

En el proyecto *DHSkate* se ha utilizado una luz ambiental para iluminar toda la pantalla sin tener en cuenta el terreno generado y una luz direccional. Con esto las pantallas y el patinador han quedado bastante bien iluminados.

```
//Establece una luz ambiente
mSceneMgr->setAmbientLight(Ogre::ColourValue(0.5, 0.5, 0.5));

//Crea un foco de luz
Ogre::Light* l = mSceneMgr->createLight("MainLight");
l->setType(Ogre::Light::LT_DIRECTIONAL); //Luz direccional
l->setDirection(Ogre::Vector3(1,-1,1));
```

Código 32: Creación de luces. DHSkate::createGame.

En cuanto a cámaras, en el proyecto se ha utilizado una librería externa llamada *Camera Control System* (CCS). Esta librería crea modos de cámaras muy útiles para juegos, ya que pueden crearse cámaras que siguen a un elemento, cámaras acopladas a un nodo, y con distintos comportamientos.

Al igual que ocurre con las luces, en OGRE si no existen cámaras, no se renderiza nada de la escena, es por ello que ha de crearse una cámara al inicio del proceso. Además la librería externa utilizará esta cámara como base de sus modelos.

```
mCamera = mSceneMgr->createCamera("Camera");
Ogre::Viewport *vp = mRoot->getAutoCreatedWindow()->addViewport(mCamera);
```

Código 33: Se crea la cámara principal y se asocia a un Viewport. DHSkate::setupScene.

Una vez creada la cámara, se ha inicializado la librería de cámaras CCS utilizando la cámara del juego.

```
mCameraCS = new CCS::CameraControlSystem(mSceneMgr, "CameraControlSystem",
mCamera);
```

Código 34: Inicialización del sistema de control de cámaras. DHSkate::createGame.

Con el sistema creado, se han definido los tipos de modos de cámara a utilizar, que han sido: una cámara libre, con la que el usuario usando el teclado puede desplazarla a su gusto y con el ratón puede rotarla a su gusto, una cámara de tercera persona y una cámara de primera persona.

Al crear estos modos se pueden definir una serie de valores: como la orientación de la cámara y la distancia al objetivo. Cada modo de cámara tiene sus propias propiedades.

```
//Cámara en tercera persona
CCS::ChaseCameraMode* camMode1 = new CCS::ChaseCameraMode(mCameraCS,
Ogre::Vector3(0,10,-20), Ogre::Vector3::UNIT_Y);

//Cámara de primera persona
CCS::FirstPersonCameraMode* camMode3 = new
CCS::FirstPersonCameraMode(mCameraCS,Ogre::Vector3(0,5,0),
Ogre::Quaternion(0,0,1,0));

camMode3->setCharacterVisible(false);//Primera persona sin mostrar el
elemento

//Cámara libre
MyFreeCameraMode* camMode2 = new MyFreeCameraMode(mCameraCS,
Ogre::Vector3(0,800,0), Ogre::Quaternion(1,0,-1,0));
camMode2->setMoveFactor(15); //Establece la velocidad de la cámara al
moverla

//Registra los modos de cámara
mCameraCS->registerCameraMode("Free",camMode2);
mCameraCS->registerCameraMode("TerceraPersona",camMode1);
mCameraCS->registerCameraMode("PrimeraPersona",camMode3);
```

Código 35: Declaración de modos de cámaras. DHSkate::createGame.

Una vez declarados los modos de cámara, simplemente hay que decir qué modo de cámara utilizar y a qué nodo del juego hay que acoplarlo. En el caso de *DHSkate* se ha acoplado al nodo principal del patinador. Con las mismas instrucciones con las que se establece el modo de la cámara y el objetivo, se cambia el modo en mitad del juego.

```
mCameraCS->setCameraTarget(mSceneMgr->getSceneNode("skaterRoot"));
mCameraCS->setCurrentCameraMode(mCameraCS-
>getCameraMode("TerceraPersona"));
```

Código 36: Estableciendo el objetivo y el modo de la cámara. DHSkate::startGame.

En este paso aún falta un pequeño detalle para que estos modos de cámara funcionen correctamente. Este paso es el de actualizar la posición de la cámara en cada frame. Para ello simplemente en alguno de los *FramesListeners* acoplados al bucle, o declarando y creando uno nuevo, hay que añadir una instrucción de actualización de las cámaras.

```
mCameraCS->update(evt.timeSinceLastFrame);
```

Código 37: Actualización de la cámara. PhysicsListener::frameRenderingQueued.

Además, para el caso de la cámara libre que se mueve con pulsaciones de ratón y teclado, hay que definir unos métodos para controlarla. Para fragmentar el código, en el proyecto se ha creado un *FrameListener* de cámara (*CameraControlListener*), para actualizar la cámara libre.

```
bool CameraControlListener::frameRenderingQueued(const Ogre::FrameEvent&
evt) {
       mCameraCS->update(evt.timeSinceLastFrame); //Actualiza posición
     if(mCameraCS->getCameraModeName(mCameraCS->getCurrentCameraMode()) ==
"Free") {
     //Si es modo libre, la cogemos y la movemos
CCS::FreeCameraMode* freeCameraMode = (CCS::FreeCameraMode*)mCameraCS-
>getCameraMode("Free");

        if (vertical == 1) freeCameraMode->goForward();
          if (vertical == -1) freeCameraMode->goBackward();
          if (horizontal == 1) freeCameraMode->goRight();
          if (horizontal == -1) freeCameraMode->goLeft();
       }
       return true;
}
```

Código 38: Moviendo la cámara libre. CameraControlListener::frameRenderingQueued.

Finalmente y para terminar con las cámaras, se ha de notar que el modo cámara libre utiliza una clase perteneciente al proyecto y no a la librería CCS (*MyFreeCameraMode*), esto se ha debido a que la cámara libre de la librería establece su punto inicial en la posición (0, 0, 0) del escenario sin posibilidad de cambiarla de posición. Este pequeño detalle hace en el caso del proyecto, que en algunos terrenos, la cámara apareciera debajo del mismo, y hacía que quedase un poco mal a la vista del usuario. Por eso se ha definido una clase que hereda de la clase de cámara libre de la librería, añadiendo la posición y rotación inicial. Esta modificación sólo ha supuesto el cambio de la definición de la clase, sin tener que tocar la implementación.

```
#ifndef _MyFreeCameraMode_H_
#define _MyFreeCameraMode_H_

class MyFreeCameraMode : public CCS::FreeCameraMode {
public:
     MyFreeCameraMode(CCS::CameraControlSystem* cam, Ogre::Vector3
position,  Ogre::Quaternion orientation) : FreeCameraMode(cam)
       {
     mCameraPosition = position;
     mCameraOrientation = orientation;
     }
     ~MyFreeCameraMode(){};
};
#endif
```

Código 39: Definición de la clase de cámara libre. MyFreeCameraMode.

3.4.11 Renderización y temporizadores

Con el terreno, las luces y cámaras, los objetos físicos y los objetos visuales implementados, junto con el ciclo de renderización iniciado, todo el proceso del juego y la renderización de los objetos en la escena quedan cubiertos. Pero en este punto se explica cómo se han implementado algunos detalles para la renderización de objetos como puede ser el sistema de renderización y sombras y las animaciones del personaje.

Todo sistema general de OGRE debe de ser de un tipo, adaptado para escenas de niveles internos, escenarios con niveles externos o escenarios genéricos entre otros. En este proyecto se ha utilizado el tipo de escenario exterior. Además se puede establecer el tipo de sombras que habrá en el juego.

```
//Crea el escenario general como Genérico. (DHSkate::setupScene)
mSceneMgr = mRoot->createSceneManager(Ogre::ST_EXTERIOR_CLOSE, "Default
SceneManager");

//--

//Establece las características del sombreado (DHSkate::scene)
mSceneMgr->setShadowTechnique(Ogre::SHADOWTYPE_STENCIL_MODULATIVE);
mSceneMgr->setShadowFarDistance(Ogre::Real(50));
mSceneMgr->setShadowDirLightTextureOffset(0.75);
mSceneMgr->setShadowTextureSize(1024);
```

Código 40: Algunas propiedades generales del renderizado. DHSkate.

OGRE controla de una forma muy trivial las animaciones creadas en programas de diseño 3D, además de permitir manejar el esqueleto del objeto.

En este proyecto, se han utilizado animaciones para dar algo de realismo y movimiento al patinador en el terreno de juego. Manejar animaciones es sencillo, simplemente hay que cargar la animación y actualizarla con el paso de los frames en un *FrameListener*.

```
void PhysicsListener::makeAnimation(Ogre::String animationName, bool loop)
{
     mAnimationState->setEnabled(false);
     mAnimationState = mSceneMgr->getEntity("skaterBody")->
getAnimationState(animationName);
     mAnimationState->setTimePosition(0);
     mAnimationState->setLoop(loop);
     mAnimationState->setEnabled(true);
}
```

Código 41: Método para invocar una animación. PhysicsListener::makeAnimation.

```
//Actualiza la animación
mAnimationState->addTime(evt.timeSinceLastFrame);
```

Código 42: Actualiza la animación al procesar el frame. PhysicsListener::frameRenderingQueued.

Además de las animaciones y el movimiento generados por las fuerzas físicas sobre el objeto físico asociado a las mallas 3D del patinador, en el proyecto también se ha hecho uso de la rotación de nodos simples de OGRE para perfeccionar algunas animaciones, como es el caso de los derrapes, en los que se rota el monopatín hasta 90 grados.

Figura 151: Patinador derrapando en el juego.

Otro punto clave en el desarrollo del proyecto ha sido el uso de temporizadores para obtener distintos controles. OGRE dispone de un mecanismo muy sencillo de temporizador con una clase *Ogre::Timer*. En el proyecto se han utilizado temporizadores para realizar la cuenta atrás antes de empezar la partida, para controlar el tiempo que lleva jugando el jugador en la pantalla y un controlador para la gestión de los derrapes.

Como se ha comentado en la parte de los físicos, existen unas fuerzas aleatorias que se aplican al patinador cuando está derrapando, estas fuerzas representan la dificultad que conlleva realizar este tipo de derrapes. Para hacer este efecto poco previsible, cada cierto tiempo habilitado con un temporizador se aplica la fuerza en una dirección concreta, cuando este tiempo pasa, se cambia la dirección de la fuerza y así sucesivamente hasta que el patinador deja de derrapar.

3.4.12 Control de localización

Para finalizar los detalles de la implementación del proyecto, hay unas características del juego muy importantes para la jugabilidad del mismo y que no se han comentado. Estas son el control del patinador en el circuito, es decir, un control de si se sale de la pista o de si ha llegado a la meta.

En el juego se ha desarrollado de la siguiente forma. En cada actualización física, se llama a un método que comprueba si se ha llegado al final del circuito y otro método que indica el número de ruedas que están fuera del asfalto del terreno de juego. Si el jugador está en el final, se anota su tiempo y se muestra la pantalla de finalización con o sin record, según el tiempo invertido. En el segundo método, si el patín tiene una o dos ruedas fuera de la calzada, se disminuye la fuerza de empuje y si tiene 3 o 4 ruedas fuera de la pista. Se pasa a la pantalla de finalización por caída.

Para estas detecciones se hace uso de la transformación de posiciones reales y posiciones en el mapa de máscara del asfalto que se puede conseguir con algunas instrucciones de OGRE.

```
bool DHSkate::inTheEnd() {
  Ogre::SceneNode* skater = mSceneMgr->getSceneNode("skater");//nodo
patinador
Ogre::Vector3 posSkater = skater->_getDerivedPosition(); //Posición real

  int difX = posSkater.x - track->getRealEndPoint().x; //distancia al final
  int difZ = posSkater.z - track->getRealEndPoint().z;

  if (difX < 0) difX *= -1;
  if (difZ < 0) difZ *= -1;

if (difX <= 30 && difZ <= 30 ) //Si está a menos de 30 puntos
return true;
  else return false;
}
```

Código 43: Comprueba si está en el final. DHSkate::inTheEnd.

Capítulo 4:

Conclusiones

Una vez finalizado el proyecto, se puede afirmar que el trabajo realizado ha cumplido la mayor parte de los objetivos planteados de forma satisfactoria. En primer lugar se ha logrado crear un sistema OGRE completo, con buen funcionamiento desde el arranque hasta el final.

Como segundo objetivo marcado y cumplido, se han aprendido a utilizar gran cantidad de librerías de código abierto y programas de gran utilidad como Blender. Todo esto ha sido posible gracias a la gran cantidad de personas que apoyan y contribuyen con estas iniciativas libres, en las que cada cual aporta su granito de arena, en mayor o menor medida, con el único objetivo de ayudar a los demás, aportando documentación, código, ejemplos y respuestas a consultas tanto en foros como por correo.

Toda esta ayuda hace que al final y tras el objetivo conseguido, uno mismo considere casi necesario el hecho de seguir viendo estos medios de comunicación, y aportar lo mismo que se ha obtenido, ayudando en lo posible a nuevos usuarios y desarrolladores y contribuyendo a la comunidad.

Por otra parte, debido a lo simple del juego, se puede considerar como un proyecto ideal para utilizarse como referencia para nuevos desarrolladores de juegos en OGRE, un punto del que partir, del que tomar ideas o en que buscar solución a problemas típicos que pueden aparecer. Con esto puede verse cumplido el tercer objetivo marcado en el proyecto.

El cuarto y último objetivo que se marcó al inicio del proyecto, fue la idea de crear una documentación completa sobre OGRE. Todo este trabajo de documentación, junto con los fuentes del proyecto, aportan gran cantidad de información sobre un software de OGRE, además de los documentos creados como trabajo dirigido de este proyecto: Manual de Ogre 3D [9], Tutoriales Básicos de Ogre 3d [10] y Documento de Entorno [11], el resultado es el obtener una gran cantidad de información referente a OGRE y en Español, que puede y pretende ser muy útil para futuros estudiantes y desarrolladores que se animen a realizar juegos o sistemas basados en este motor gráfico.

Al margen de los objetivos marcados, se puede llegar a varias conclusiones adicionales y personales. En primer lugar, se observa que el motor gráfico da unos resultados muy buenos, con pocos recursos, aunque como este motor es pionero como trabajo en esta universidad para el desarrollo de un juego y a la vista del contenido encontrado, es también uno de los pocos creados por un Español, cada paso del proyecto, pese a su sencillez final, ha costado mucho, encontrando muchos problemas en cada uno de los movimientos realizados, aunque pasando los mismos poco a poco.

Por otra parte, el ver finalizado un proyecto de este estilo, aporta una gran satisfacción personal, ya que los resultados son muy gratificantes y siempre están muy bien valorados por todo tipo de personas, ya sean cercanas o ajenas al mundo de la informática y del desarrollo de videojuegos.

Además, la gran cantidad de conocimientos adquiridos podrán utilizarse para crear en el futuro, desarrollos más complejos y ambiciosos.

Capítulo 5:

Líneas futuras

Dados los límites en tiempo establecidos en el proyecto y gran cantidad de opciones y mejoras que surgen prácticamente cada día, hay muchas variaciones que se hubiera deseado incluir en el proyecto y que no ha sido posible. Por lo tanto se han anotado una gran cantidad de acciones futuras a realizar sobre el proyecto:

Como primera acción a realizar y más importante, es arreglar los pequeños problemas que el sistema tiene actualmente y en los que más tiempo se ha invertido para corregir. Es el caso de arreglar los controles físicos, ya que aunque se acercan un poco a la realidad, en cuanto a derrapes, aún falta bastante para quedar realista al 100%. Además y junto con este arreglo iría la corrección de circuitos, para suavizar las carreteras y evitar esos baches que se forman y que afectan a los mecanismos de físicas.

Como segunda acción, se buscaría la publicación del juego en alguna plataforma como por ejemplo Steam [12], esta primera versión se vendería a un precio muy bajo y se intentaría buscar el apoyo mediante publicidad incorporada en el juego, como banners publicitarios y mediante sponsors de tiendas o marcas relacionadas con el deporte del Downhill skate, por ejemplo poniendo los logotipos de las marcas en los menús, tablas, monos del patinador, el casco o incluso decorando las alpacas. Además y con el sistema actual sería muy sencillo añadir nuevos circuitos y patinadores al repertorio del juego, incluyendo por ejemplo circuitos y patinadores bloqueados hasta que el jugador consiga una serie de buenos resultados con los patinadores y circuitos disponibles al principio.

La tercera acción sería crear una segunda versión incorporando la posibilidad de incluir los records de forma online, ampliando el número de patinadores y circuitos, añadiendo modo competición contra el propio ordenador con adversarios con inteligencia artificial y añadiendo un modo de juego online con otros jugadores. Se podría pasar a guardar los records de los circuitos y usuarios de forma online para incitar a la competitividad, así como incluir versiones de juego con fantasma: Guardar los movimientos realizados en una partida y competir contra ti mismo viendo los movimientos realizados en la partida anterior.

Finalmente y como última acción sería la posibilidad de portar el juego a otras plataformas, como pueden ser las plataformas móviles. Por ejemplo, OGRE incorpora unos paquetes para crear aplicaciones para MAC y para plataformas móviles de Apple como son el iPad y el iPhone. La conversión de aplicaciones de PC a MAC y móviles resulta bastante sencilla con estos paquetes, salvo algunos pequeños detalles.

Capítulo 6:

Glosario de términos

En este capítulo se muestra la descripción de algunos términos utilizados en el proyecto y cuyo significado puede resultar difícil de conocer, por ser un tema técnico.

- **Downhill**: Descenso de montañas.
- **FPS**: *First Person Shooter (Game).* (Juego) de disparos en primera persona.
- **Frame**: Imagen particular dentro de una sucesión de imágenes que componen una animación.
- **Framework**: Conjunto estandarizado de conceptos, prácticas y criterios para enfocar un tipo de problemática particular, que sirve como referencia para enfrentar y resolver nuevos problemas de índole similar.
- **Hash**: Función o método para general claves, los algoritmos hash, también llamados función resumen, no tienen vuelta atrás, no se puede pasar del código hash al código sin codificar.
- **IDE**: Entorno de Desarrollo Integrado. Programa informático compuesto por un conjunto de herramientas de programación, con interfaz gráfica, compilador, etc.
- **Longboard**: Patinete algo más grande y ancho que los patinetes de uso común, que aportan estabilidad y control.
- **MMOG**: *Massive Multiplayer Online Game.* Juego multijugador masivo en línea.

- **Mod**: Es una extensión que modifica un juego original, proporcionando nuevas posibilidades, ambientaciones, personajes, etc.
- **Parseo**: Un cambio de un formato a otro en un sistema informático.
- **Plugin**: Es un complemento informático de una aplicación para añadirle cierta funcionalidad.
- **Renderizar**: Generar una imagen (3D o animación 3D) a partir de un modelo usando una aplicación computadora.
- **RTS**: *Real-Time Strategic (Game).* (Juego) de estrategia en tiempo real.
- **Skate**: Monopatín.
- **Skater**: Patinador.
- **Thread**: Un hilo de ejecución dentro de un programa informático.
- **Xml**: Lenguaje de marcas extensible. Lenguaje mediante etiquetado que permite representar casi cualquier elemento de forma organizada.

Capítulo 7:

Referencias

En este capítulo se incluyen todas las referencias a documentos y páginas webs que se han utilizado tanto para el desarrollo del proyecto, como para la creación de este manual.

- [1] *Ogre.* Disponible [Internet]:<http://www.ogre3d.org> [10 de Abril de 2011]
- [2] *IberOgre*. Disponible [Internet]: <http://osl2.uca.es/iberogre/index.php/P%C3%A1gina_Principal> [4 de Marzo de 2011]
- [3] *Newton*. Disponible [Internet]:<http://newtondynamics.com> [15 de Febrero de 2011]
- [4] *CCS*. Disponible [Internet]:<http://ogre-ccs.sourceforge.net/> [10 de Noviembre de 2010]
- [5] *MyGUI*. Disponible [Internet]:<http://mygui.info/> [30 de Diciembre de 2010]
- [6] *cAudio*. Disponible [Internet]:<http://caudio.deathtouchstudios.com/> [10 de Noviembre de 2010]
- [7] González Morcillo, Carlos: '*Aprende en 24 horas Blender y Yafray. Diseño Gráfico 3D con Software Libre'*. Escuela Superior de Informática – Universidad de Castilla-La Mancha. Julio de 2006. Disponible [Internet]: <http://www.inf-cr.uclm.es/www/cglez/> [20 de Diciembre de 2010]

- [8] *Blender*. Disponible [Internet]: <http://www.blender.org/> [20 de Diciembre de 2010]
- [9] Velázquez Muñoz, Mario: '*Manual de Ogre 3D'* (Traducción). Universidad Carlos III de Madrid. Mayo de 2010. Versión de Ogre 1.7.1. Disponible [Internet]: <http://www.ogre3d.org/docs/manual/> [25 de Febrero de 2011]
- [10] Velázquez Muñoz, Mario: '*Tutoriales Básicos'* (Traducción). Universidad Carlos III de Madrid. Mayo de 2010. Versión de Ogre 1.7.1. Disponible.
- [11] Velázquez Muñoz, Mario: '*Entorno de desarrollo de Ogre 3D'*. Universidad Carlos III de Madrid. Mayo de 2010. Versión de Ogre 1.7.1. Disponible.
- [12] *Steam*. Disponible [Internet]: <http://store.steampowered.com/> [8 de Marzo de 2011]
- [13] *Wikipedia*. Disponible [Internet]: <http://es.wikipedia.org> [10 de Abril de 2011]
- [14] Gregory, Janson: '*Game Engine Architecture'*. A K Peters, Ltd. 2009. Disponible
- [15] *Neo-Teo*. Disponible [Internet]: <http://www.neoteo.com/top-10-los-motores-graficos-mas-importantes> [1 de Abril de 2011]
- [16] *Euphoria*. Disponible [Internet]: <http://www.naturalmotion.com/euphoria> [1 de Abril de 2011]
- [17] *Capcom*. Disponible [Internet]: <http://www.capcom.com/> [1 de Abril de 2011]
- [18] *Unreal Development Kit*. Disponible [Internet]: <http://www.udk.com/> [2 de Abril de 2011]
- [19] *Crystal Space*. Disponible [Internet]: <http://www.crystalspace3d.org> [2 de Abril de 2011]
- [20] *Panda 3d*. Disponible [Internet]: <http://www.panda3d.org/> [2 de Abril de 2011]
- [21] *XNA*. Disponible [Internet]: <http://create.msdn.com> [2 de Abril de 2011]
- [22] *Autodesk Maya*. Disponible [Internet]: <http://usa.autodesk.com/maya/> [2 de Abril de 2011]

- [23] *Autodesk 3d Studio Max*. Disponible [Internet]: <http://usa.autodesk.com/3ds-max/> [2 de Abril de 2011]
- [24] *LGPL*. Licencia pública general reducida de GNU. Disponible [Internet]: <http://www.viti.es/gnu/licenses/lgpl.html> [2 de Abril de 2011]
- [25] *720 degree*. Disponible [Internet]: <http://www.720zone.com/> [10 de Abril de 2011]
- [26] *UltimoNivel.* Disponible [Internet]: <http://www.ultimonivel.net> [10 de Abril de 2011]
- [27] *StarCraft II.* Disponible [Internet]: <http://us.battle.net/sc2/es> [10 de Abril de 2011]
- [28] *Rockstar Games*. Disponible [Internet]: <http://www.rockstargames.com> [10 de Abril de 2011]
- [29] *Tony Hawk*. Disponible [Internet]: <http://www.tonyhawk.com> [10 de Abril de 2011]
- [30] *EA*. Disponible [Internet]: <http://www.ea.com/es> [10 de Abril de 2011]
- [31] *El otro lado.* Disponible [Internet]: <http://www.elotrolado.net/wiki/Historia_de_los_videojuegos> [10 de Abril de 2011]
- [32] *DirectX 9c*. Disponible [Internet]: < http://www.microsoft.com/downloads/es-es/details.aspx?FamilyID=2da43d38-db71-4c1b-bc6a-9b6652cd92a3> [10 de Abril de 2011]
- [33] *ADeSe*. Asociación Española de Distribuidores y Editores de Software de Entretenimiento. Disponible [Internet]: <http://www.adese.es> [10 de Abril de 2011]
- [34] *Ogrees*. Ogre en Español. Disponible [Internet]: <https://sites.google.com/site/ogreesp/Home> [1 de Abril de 2011]
- [35] *Wikipedia*. *Industria de los videojuegos*. Disponible [Internet]: <http://es.wikipedia.org/wiki/Industria_de_los_videojuegos> [1 de Abril de 2011]

- [36] *Niel*. Foros de Blender en Español. Disponible [Internet]: <http://niel.seyanim.com/> [20 de Diciembre de 2010]
- [37] *Google Maps*. Disponible [Internet]: <http://maps.google.es/> [10 de Abril de 2011]
- [38] *IGSA*. International Gravity Sports Association (Asociación Internacional de derpotes de gravedad). Disponible [Internet]: <http://www.igsaworldcup.com> [10 de Abril de 2011]

Anexo A:

Planificación final y conclusiones

A continuación se muestra el diagrama de GANTT real con la planificación final del proyecto. De esta forma se puede ver el tiempo invertido en el proyecto y junto con el diagrama de la planificación inicial hacer un estudio sobre las variaciones obtenidas entre lo planificado y lo obtenido.

Además se incluye un nuevo cálculo del presupuesto del proyecto, que añade dos meses más de trabajo de un ingeniero y un mes adicional para el jefe del proyecto.

A.1 Planificación final

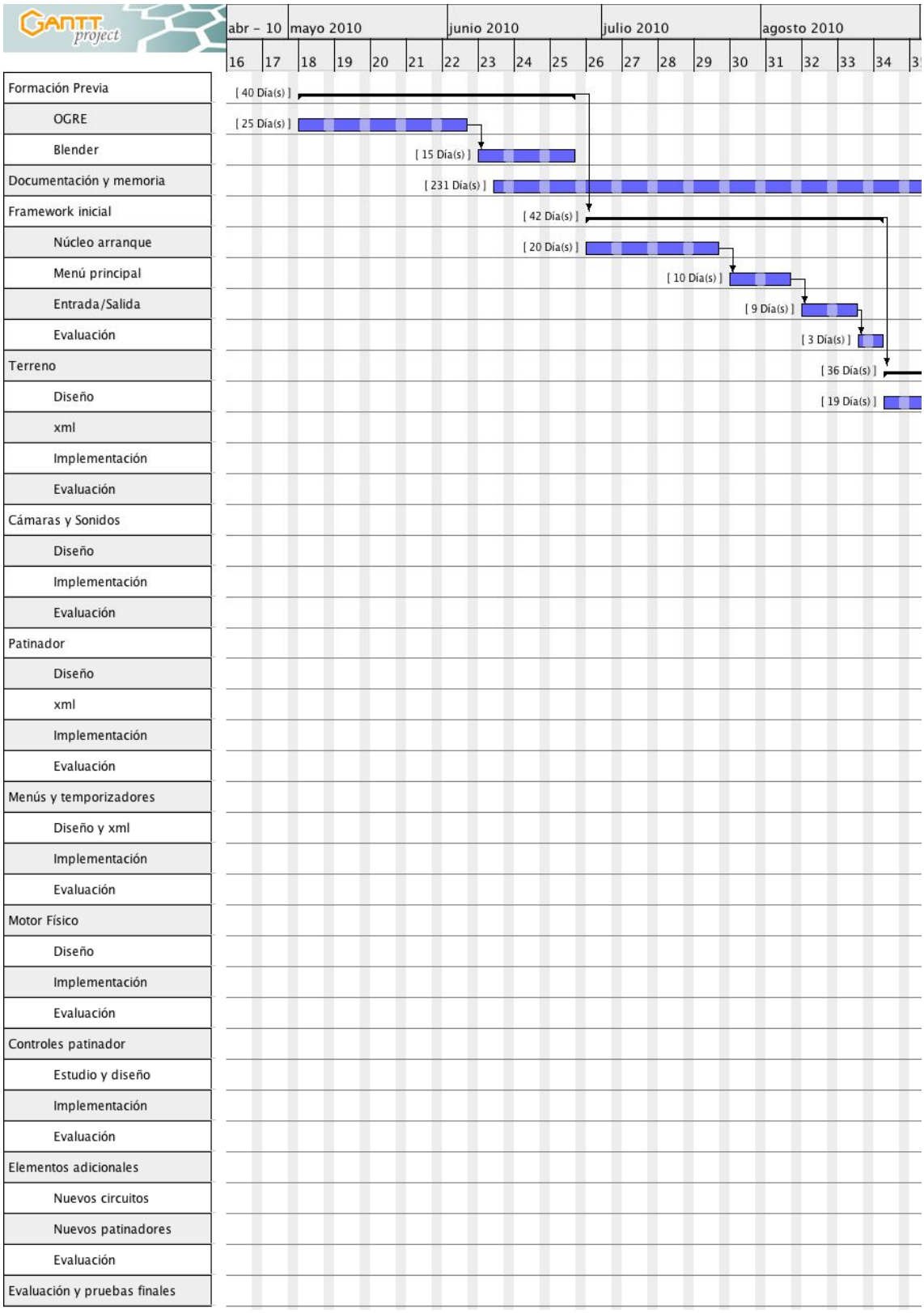

Figura 152: Primera parte del diagrama de GANTT final.

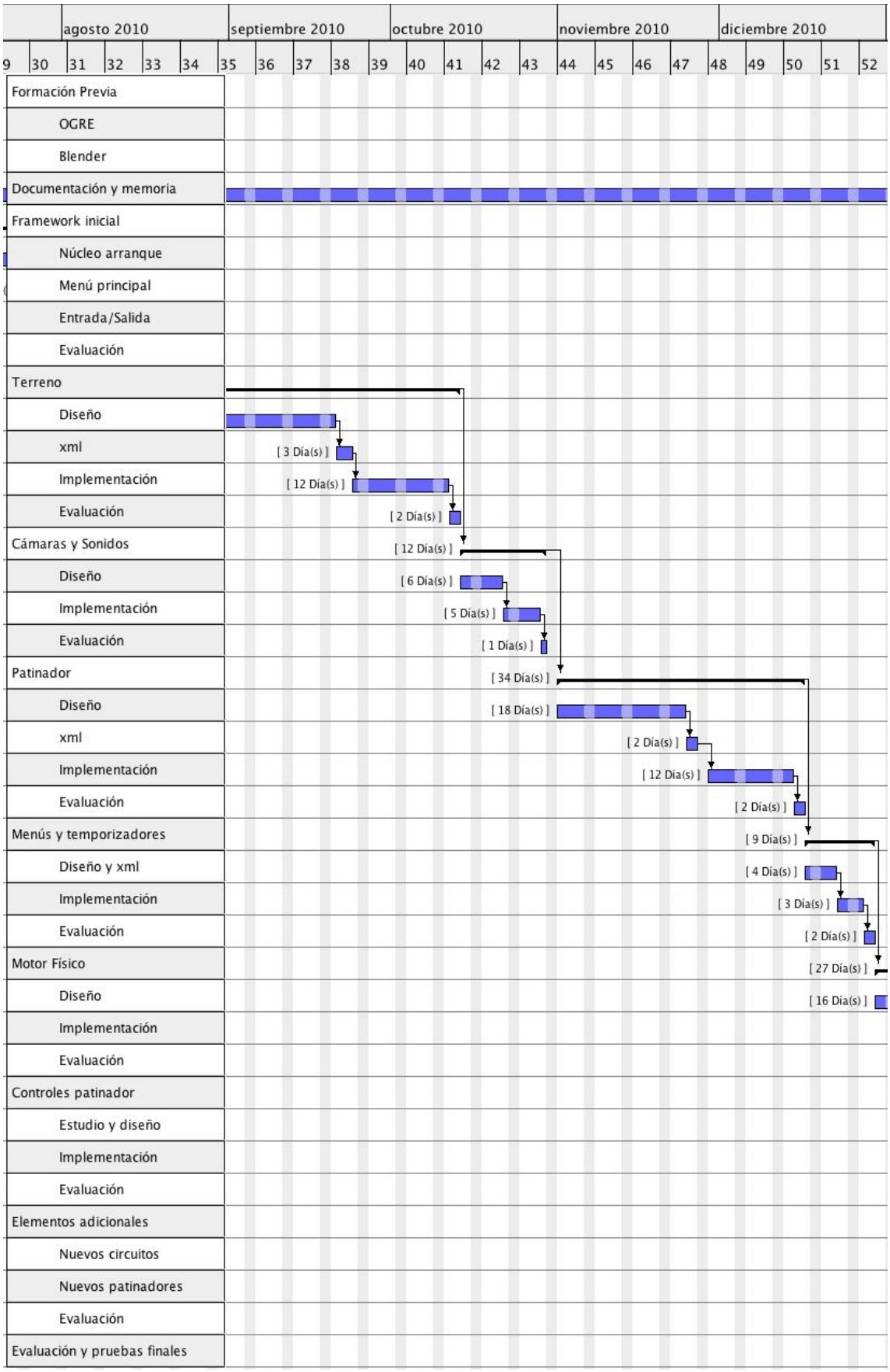

Figura 153: Segunda parte del diagrama de GANTT final.

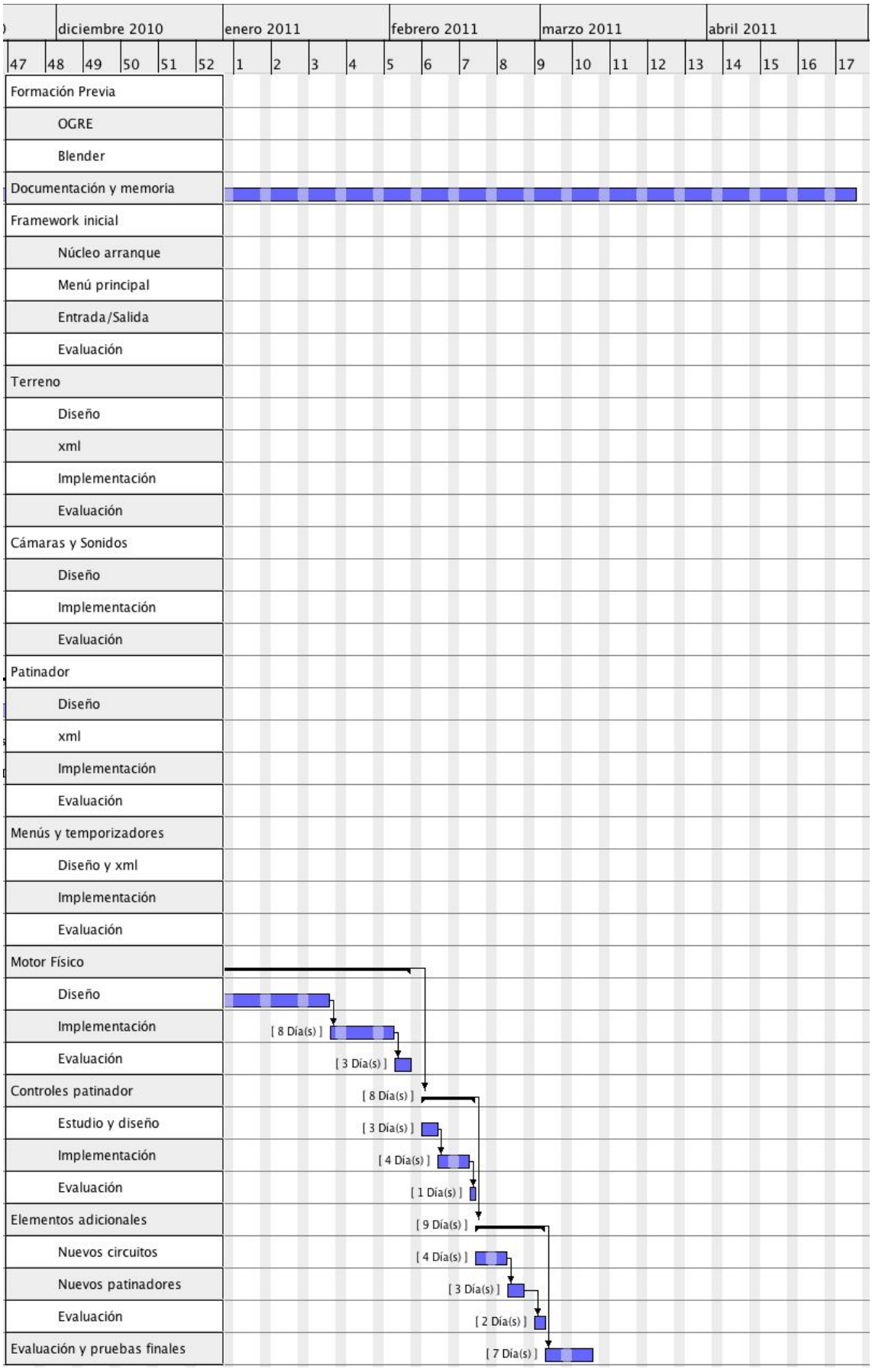

Figura 154: Tercera parte del diagrama de GANTT final.

La planificación final obtenida ha sido de 12 meses de trabajo, Desde Mayo de 2010 hasta Abril de 2011. Al igual que en la planificación inicial, en esta planificación se ha contado con un único recurso, el autor del proyecto. Además los Sábados y Domingos no se han contabilizado. El resumen de la planificación queda reflejado en la siguiente tabla, donde se resumen las tareas generales y su duración sin desglosar en subtareas:

Tarea	Inicio	Fin	Duración
Formación previa	03/05/2010	26/06/2010	40 días
Documentación y memoria	10/06/2010	29/04/2011	231 días
Framework inicial	28/06/2010	25/08/2010	42 días
Terreno	25/08/2010	14/10/2010	36 días
Cámaras y sonidos	14/10/2010	30/10/2010	12 días
Patinador	01/11/2010	17/12/2010	34 días
Menús y temporizadores	17/12/2010	30/12/2010	9 días
Motor físico	30/12/2010	05/02/2011	27 días
Controles patinador	07/02/2011	17/02/2011	8 días
Elementos adicionales	17/02/2011	02/03/2011	9 días
Evaluación y pruebas finales	02/03/2011	11/03/2011	7 días

Tabla 72: Planificación final de tareas.

A.2 Conclusiones planificación

Comparando la planificación inicial y la planificación real resultante, se puede ver que el proyecto ha sufrido un desvío de unos dos meses respecto al plazo marcado. A continuación se muestra un gráfico que puede ayudar a comprender mejor el desplazamiento en las fechas ocurrido:

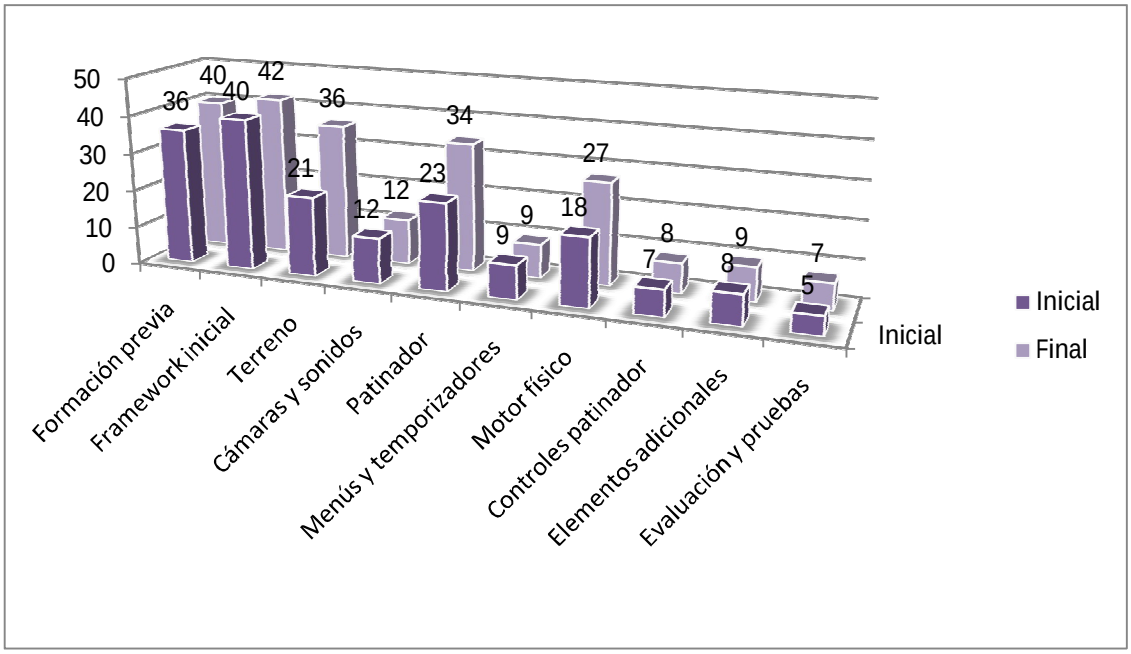

Figura 155: Diagrama de comparación de planificación inicial y final.

Como se puede observar en la gráfica, la mayor parte de las tareas están dentro de plazos normales, con desvíos o retrasos insignificantes. El mayor desplazamiento viene en los dos puntos que se han destacado en la documentación del proyecto, como los problemas más grandes encontrados: la generación del terreno, que ha provocado realizar y buscar varias alternativas para que los terrenos quedaran algo más refinados y el problema que se encontró con la librería física y que arrastró a su vez a la tarea del patinador.

Como se ha comentado en el documento, estos errores se han debido un poco a la propia inexperiencia con el motor de OGRE y el motor físico, que pese ha haber realizado algunas pruebas previas satisfactorias, en el proyecto final resultaron en errores, al entrar en juego escenarios creados a partir de carreteras reales y la búsqueda de efectos físicos más realistas.

A.3 Presupuesto final

Con los cambios aparecidos en la planificación, los costes del proyecto también sufren un pequeño cambio, que quedan reflejados en los siguientes diagramas.

PRESUPUESTO DE PROYECTO

1.- Autor:

Mario Velázquez Muñoz

2.- Departamento:

Departamento de Informática

3.- Descripción del Proyecto:

- Titulo	**Diseño e implementación de un juego para PC mediante OGRE 3D**
- Duración (meses)	**12**
Tasa de costes Indirectos:	**20%**

4.- Presupuesto total del Proyecto (valores en Euros):

49.714,00 Euros

5.- Desglose presupuestario (costes directos)

PERSONAL

Apellidos y nombre	N.I.F. (no rellenar - solo a titulo informativo)	Categoría	Dedicación (hombres mes) [a)]	Coste hombre mes	Coste (Euro)
Peralta Donate, Juan		Jefe de Proyecto	2	4.289,54	8.579,08
Velázquez Muñoz, Mario		Ingeniero	12	2.694,39	32.332,68
					0,00
					0,00
					0,00
		Hombres mes	**14**	**Total**	40.911,76

[a)] 1 Hombre mes = 131,25 horas. Máximo anual de dedicación de 12 hombres mes (1575 horas)

Máximo anual para PDI de la Universidad Carlos III de Madrid de 8,8 hombres mes (1.155 horas)

Figura 156: Primera parte del presupuesto final del proyecto.

EQUIPOS

Descripción	Coste (Euro)	% Uso dedicado proyecto	Dedicación (meses)	Periodo de depreciación	Coste imputable [d)]
Ordenador diseño gráfico y programación	1.200,00	100	12	60	240,00
Disco duro externo	158,00	100	12	60	31,60
Impresora	100,00	30	12	60	6,00
				Total	277,60

[d)] Fórmula de cálculo de la Amortización:

$$\frac{A}{B} x C x D$$

A = nº de meses desde la fecha de facturación en que el equipo es utilizado
B = periodo de depreciación (60 meses)
C = coste del equipo (sin IVA)
D = % del uso que se dedica al proyecto (habitualmente 100%)

SUBCONTRATACIÓN DE TAREAS

Descripción	Empresa	Coste imputable
	Total	0,00

Figura 157: Segunda parte del presupuesto final del proyecto.

OTROS COSTES DIRECTOS DEL PROYECTO[e)]

Descripción	Empresa	Costes imputable
Fungible	Ofistore	60,00
Viajes	-	40,00
Licencia Microsoft Office Hogar y Estudiantes	Microsoft Store	139,00
		0,00
		0,00
		0,00
		0,00
	Total	239,00

[e)] Este capítulo de gastos incluye todos los gastos no contemplados en los conceptos anteriores, por ejemplo: fungible, viajes y dietas, otros,...

6.- Resumen de costes

Presupuesto Costes Totales	**Presupuesto Costes Totales**
Personal	40.912
Amortización	278
Subcontratación de tareas	0
Costes de funcionamiento	239
Costes Indirectos	8.286
Total	49.714

Figura 158: Tercera parte del presupuesto final del proyecto.

A la vista del presupesto final obtenido, con un total de 49.714 Euros, el proyecto sufre un aumento con respecto a los 38.058 Euros del presupuesto inicial de 11.656 Euros.

Este aumento en los costes, provoca que el número de ventas que deben realizarse al mes para que tras un año de la puesta en el mercado del juego este sea rentable en 486 juegos más al mes. Tarea de nuevo difícil pero no imposible.

Anexo B:

Manual de usuario

A continuación se indican todas las opciones de debe realizar un usuario para poder jugar al juego, así como una explicación de las acciones que puede realizar dentro del mismo para jugar.

B.1 Requisitos del sistema

El juego, al ser un juego en 3D y con cierto nivel gráfico precisa de una máquina potente, con unos valores mínimos a los siguientes:

- Disco Duro de 200 MB.
- Memoria RAM de 2 GB.
- Tarjeta gráfica dedicada de 256 MB.
- Sistema Operativo Windows XP.
- Directx 9c

Estos son los requisitos mínimos para que el juego pueda funcionar, además el juego también funciona con Windows 7, pero los requisitos mínimos de RAM se verían aumentados hasta 3 GB.

En caso de no disponer de las DirectX 9c, se recomienda descargarla desde el siguiente enlace [32].

Si el ordenador destino no cumple los requisitos mínimos, el juego no funcionará o puede que funcione pero sin llegar a un rendimiento óptimo del mismo.

A continuación se citan un par de notas a tener en cuenta en determinados sistema:

- Windows 7: En algunos sistemas con Windows 7, la ruta de instalación por defecto del programa, puede provocar que el juego no arranque. En caso de ocurrir esto, se recomienda instalar el juego en carpetas más accesibles, como por ejemplo: C:\DHSkate.
- Ordenadores potentes: En ordenadores muy potentes, es posible que el juego funcione demasiado rápido usando los drivers DirectX 9.0 c, por lo que se recomienda instalar los drivers de open GL y ejecutar el juego usando este sistema de renderizado.

B.2 Instalación

El juego dispone de un instalador, por lo que lo único que hay que hacer es pulsar sobre el icono incluido en la carpeta del proyecto llamado *setup.exe*. Este instalador pedirá algunos datos como ruta de instalación y si se desea o no crear un icono de acceso directo en el escritorio, así como un acceso desde el menú inicio.

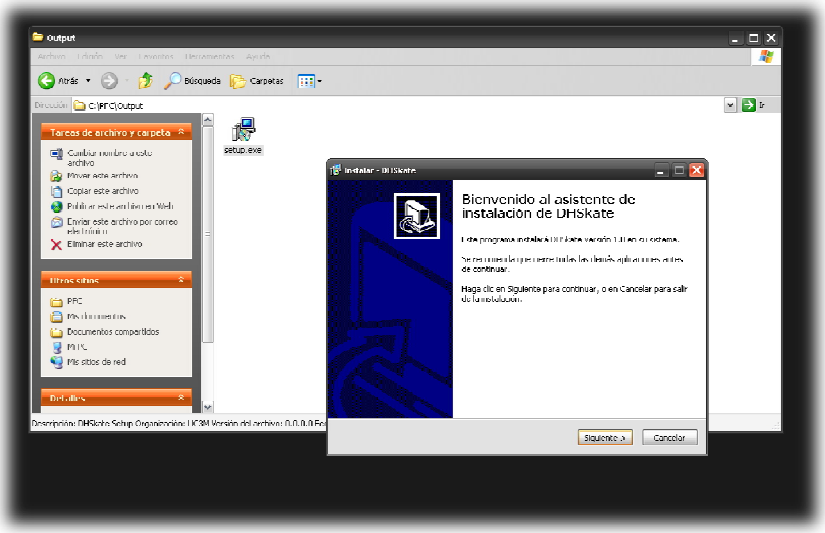

Figura 159: Instalación de DHSkate.

A los usuarios inexpertos se les recomienda no cambiar los valores que el juego trae por defecto y por tanto ir pulsando sobre “siguiente” en cada una de las pantallas que aparecen.

B.3 Juego

Una vez instalado el juego, se puede comenzar a jugar al mismo pulsando sobre el ejecutable en la carpeta de instalación, el acceso directo del escritorio (si se creó durante la instalación) o bien desde menú inicio (si se habilitó en la instalación).

Pantalla de renderización

La primera pantalla en aparecer es la de configuración del sistema de renderizado de OGRE, a los usuarios básicos se les recomienda no tocar estos parámetros. Los parámetros pueden bajarse en caso de que el juego no funcione fluido con los parámetros establecidos por defecto.

Figura 160: Pantalla de inicio de programa OGRE.

Menú principal

Después de la pantalla de configuración del sistema de renderizado aparece la pantalla del menú principal con tres opciones: Jugar, Opciones y Salir.

Figura 161: Pantalla principal de DHSkate.

Con el botón “jugar” se pasa a la ventana de selección de patinador y circuito, con el botón de “opciones” se pasa a la pantalla de configuración y con el botón “salir” se sale del juego para volver a Windows.

Menú opciones

En el menú opciones se pueden configurar los sonidos del juego.

Se podrá habilitar o no música de fondo, una canción que sonará mientras se juega.

Figura 162: Pantalla menú opciones.

También se puede establecer el volumen de los efectos sonoros. Este sonido también marca el volumen de la música de fondo. Para deshabilitar los efectos hay que poner el deslizador en la posición inicial, si la música de fondo está habilitada, esta seguirá sonando, aunque el volumen esté a 0. Para deshabilitar todos los sonidos, hay que bajar el volumen de los efectos a 0 y quitar el check de música.

El menú también muestra los controles del juego:

- Con la **tecla I**, el patinador se ayuda de los pies para acelerar el patín, sirve para acelerar el patín al principio del juego y si en algún momento de la bajada, el patinador se quedara parado. Hay que tener cuidado con el uso de este botón, pues si el patinador lleva suficiente velocidad, el uso de este botón ralentizará al jugador. Esta acción sólo puede realizarse si el patinador está en pie.
- Con las **teclas J y L** el patinador gira para izquierda y derecha respectivamente. El patinador gira tanto en pie como agachado, aunque hay que tener en cuenta que en pie se gira mejor y que dependiendo el tipo de patinador, algunos patinadores giran mejor que otros.

- Con la **tecla K** el patinador frena, tanto en pie como agachado, en pie frena mucho más que agachado, aunque este freno en sí no es muy potente. Dependiendo del patinador elegido, frenará mejor o peor.
- Con la **tecla A** se cambia la postura de agachado y en pie. Hay que tener en cuenta que agachado la resistencia aerodinámica es menor y por tanto el jugador correrá más. Según el estilo del patinador, alcanzará más o menos velocidad en las distintas posturas.
- Con la **tecla ESPACIO**, se realiza un derrape o slide. Estos derrapes sólo pueden hacerse con el patinador en pie. Son frenazos algo más bruscos que los frenos normales, pero controlar un derrape es difícil y puede que el patinador se desvíe de la trayectoria deseada.
- Con la **tecla C**, se cambia la cámara entre tercera y primera persona. En tercera persona es más controlable, aunque la sensación de velocidad será mucho mayor en la cámara de primera persona. Controlar un derrape con la cámara de primera persona será todo un reto.
- Finalmente con la **tecla ESC**, el jugador se sale del juego, por lo que no se le tomará tiempo y se considera que se ha caído de su longboard.

Tras elegir las opciones de audio deseables, se pulsa sobre “menú” para volver al menú principal. Hay que tener en cuenta que las opciones de audio se guardan, por lo que en la siguiente ocasión en que se arranque el juego, no será necesario volver a ajustar si se desea jugar con las propiedades de la entrada anterior.

Menú selección de patinador y circuito

Si desde el menú principal se pulsa sobre el botón “Jugar”, se pasa al menú de selección de patinador y circuito. Usando los combos se puede ver una imagen del patinador y una descripción, al igual que en el caso del circuito.

Figura 163: Pantalla de selección de patinador y circuito.

Una vez determinado el patinador y el circuito se pulsa el botón “jugar” para pasar al modo de pre-visualización.

Modo de pre-visualización

En este modo, el jugador puede recorrer de forma virtual todo el circuito, para estudiar las curvas y ver los puntos complicados del mismo.

Figura 164: Pantalla de pre-visualización.

Para moverse por el circuito se usan los cursores del teclado y para rotar la vista se utiliza el movimiento del ratón. Si se pulsa ESC se vuelve al menú principal. Pulsando ENTER en el teclado se empezará a jugar con una cuenta atrás.

Jugando

La partida comienza con una cuenta atrás de 3 segundos. Si en este periodo el jugador pulsa alguna de las teclas de juego, salvo ESC para salir, se penalizará el tiempo de partida con un segundo, por lo que hay que esperar a que acabe la cuenta atrás para acelerar al jugador con la tecla I.

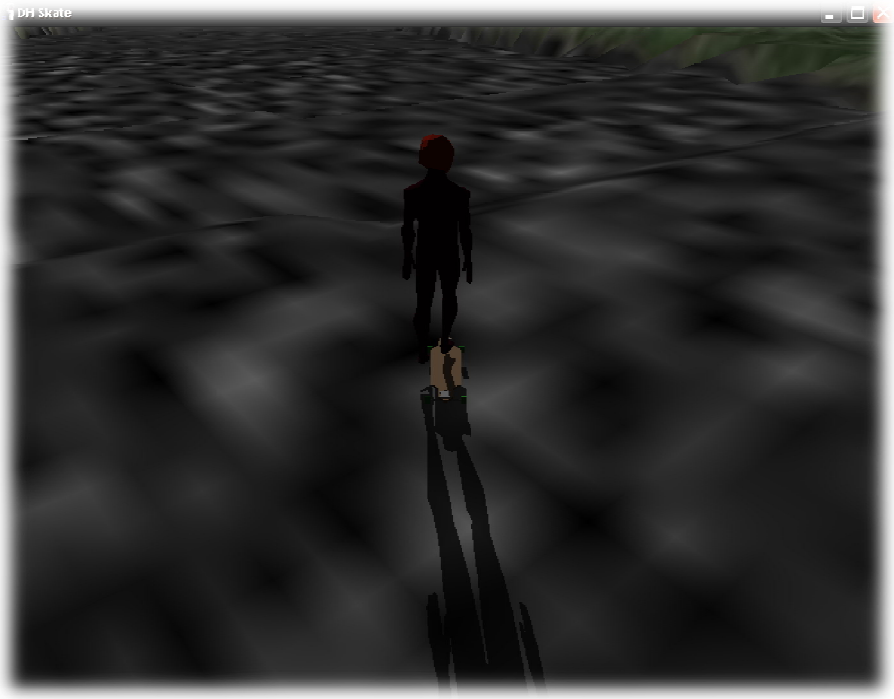

Figura 165: Pantalla de juego, iniciando la prueba.

Durante el juego se controla al jugador con las teclas indicadas en el apartado de menú de opciones. Se recomienda no abusar de la tecla de aceleración con pie y agachar al jugador y no tocar teclas en recta, ya que si el patinador está recto, su posición sobre la tabla es perfecta y aumenta al máximo la aerodinámica, aumentando por tanto su velocidad.

Es importante también saber el punto en el que ponerse en pie y usar el derrape para afrontar una curva muy cerrada, por ejemplo justo antes de las curvas de casi 180 grados, tan características en puertos de montaña y también en circuitos de longboard.

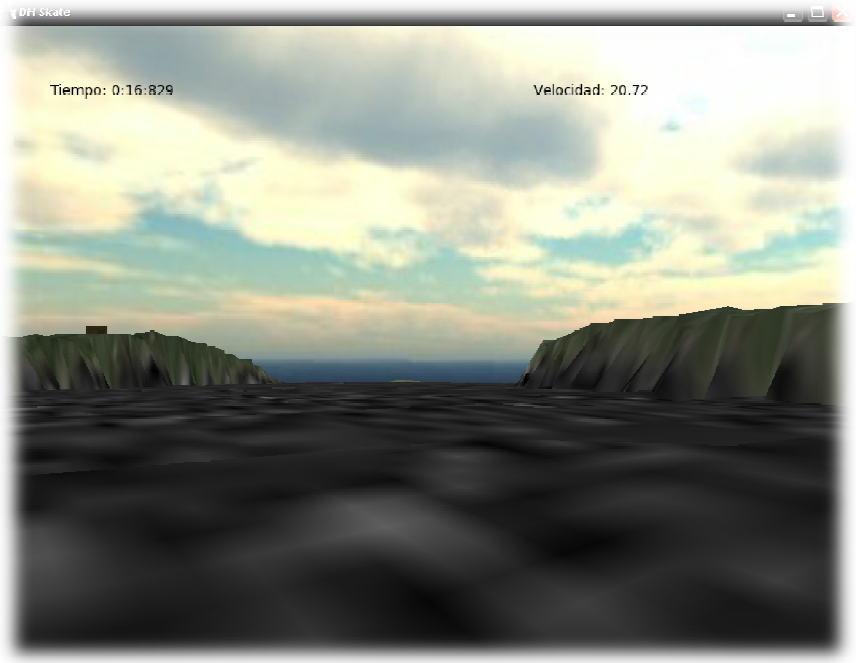

Figura 166: Pantalla de juego en primera persona.

Durante el juego si el jugador pulsa ESC, se considera que se ha caído, mostrando el menú de fin por caída. También aparecerá este menú cuando el jugador se salga del asfalto, aunque para esto tienen que salirse 3 o más ruedas del longboard fuera del circuito.

Figura 167: Pantalla de juego en tercera persona.

El juego acaba al llegar al final de la cuesta, donde se pasará automáticamente al menú de fin. Este menú será diferente si el jugador ha conseguido el mejor tiempo del circuito, donde podrá poner su nombre y se guarda el tiempo para futuras partidas, o un menú normal indicando su tiempo y el tiempo de record si no ha conseguido el mejor tiempo.

Menús de fin

Los menús de fin tienen características similares. En todos ellos pulsando sobre el botón "MENÚ" se vuelve al menú principal del juego y todos muestran el mejor tiempo del circuito y el jugador que lo alcanzó.

Figura 168: Pantalla de menú de fin con record.

Además según el tipo de carrera, cada menú tiene unas opciones especiales si el jugador se ha salido del juego o se ha caído, si ha terminado sin record y si ha terminado con record.

- Si cae o sale pulsando ESC. Simplemente se indica su caída del juego.
- Si finaliza la prueba sin obtener el mejor tiempo, se indica el tiempo que ha obtenido.
- Si finaliza superando el mejor tiempo, aparece un cuadro de texto, para que el jugador escriba su nombre y se quede guardado para el futuro.

B.4 Desinstalación

Como el juego guarda los mejores tiempos, es una buena práctica intentar superar el mejor tiempo de cada circuito probando distintos jugadores, por lo que puede entretener durante muchos días, y además, como se puede poner el nombre del jugador, puede ser un medio para competiciones entre amigos para ver quién obtiene el mejor tiempo y supera el record actual.

Pero si no se desea seguir disfrutando del juego, se puede eliminar completamente del ordenador utilizando la utilidad de desinstalación.

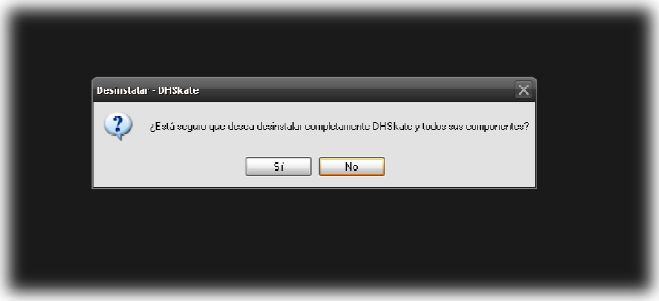

Figura 169: Desinstalador del juego.

Para desinstalar el juego basta con pulsar en desinstalar que está en la carpeta del juego o que aparece en el menú de inicio de Windows (si se instaló con esta opción). Sólo hay que seguir las indicaciones que aparecen en pantalla para eliminar al completo el juego del sistema.

Anexo C:

Manual de experto

A continuación se indica cómo incluir nuevos circuitos y patinadores al juego. Esta opción sólo debe de realizarse por expertos o desarrolladores, ya que en caso de realizar mal alguna de las operaciones, puede que el juego no funcione correctamente.

C.1 Agregar un patinador

Agregar un patinador puede involucrar desde crear un simple perfil utilizando objetos 3D existentes, hasta crear un patinador desde 0 completamente.

Mallas 3d

Para crear un nuevo patinador, longboard, eje o rueda, se puede utilizar cualquier programa de diseño 3D que tenga posibilidad de importación en OGRE, aunque se recomienda que se use Blender y sobre todo, que se usen como plantilla los que se incorporan en el proyecto como plantillas, ya que una variación de tamaño o eje de coordenadas, puede provocar que no se vea apropiadamente en el juego.

Una vez modificadas las mallas de plantilla, usando la herramienta de exportación de Blender para Ogre y la herramienta de OGRE para conversión de mallas, se obtienen los ficheros .mesh de cada malla objeto y sus .material relacionados. Las mallas deben de situarse

en la carpeta media/models/ del juego y los scripts de textura en la carpeta media/materials/scripts/ del juego.

Foto

Hay que importar una foto del jugador de 300 x 225 píxeles en formato .jpg para mostrar en el menú de selección de patinador. Esta debe de guardarse en la carpeta media/MyGUI_Media/ del juego, con nombre igual al nombre del fichero descriptor de patinador xml (con extensión .dhsk), el programa utiliza el nombre para cargar el xml y la foto, por lo que deben de ser exactamente iguales.

Xml

Para finalizar hay que crear un fichero Xml con extensión .dhsk en la carpeta media/gameData/skaters/ con el nombre del patinador, igual que el nombre puesto en la foto. Se recomienda utilizar una copia de un patinador existente.

Dentro del xml hay que poner todos los datos del patinador, los nombres de los objetos 3D que utilizará y las propiedades de giros y velocidad, siguiendo las tablas que se muestran en el Diseño.

Para establecer el código de seguridad hay que calcular este para cada una de las líneas del xml siguiendo las indicaciones del Diseño.

Con todo este proceso el nuevo jugador estará disponible la próxima vez que el juego arranque.

C.2 Agregar un circuito

Crear un circuito tiene un proceso muy similar al de crear un patinador, aunque es un poco más tedioso en cuestión de configuración y creación de mapas.

Mapa de alturas

El mapa de alturas o heightmap como se comentó en el diseño, es un mapa en escala de grises que representa las alturas de cada punto del terreno. El mapa de alturas debe de ser de 1024 x 1024 píxeles con extensión .png y debe de guardarse con el nombre que se quiera en la

carpeta media/scene1 del juego. Para construir el mapa se puede utilizar cualquier medio de los comentados, como por ejemplo usando Blender y un editor gráfico como GIMP.

Capas del terreno

El terreno puede tener hasta 5 capas distintas: la primera queda reservada para definir la textura y situación del asfalto de la carretera, la segunda y tercera definen capas exteriores al asfalto como vegetación o piedras, la cuarta indica lugares donde se situarán los árboles y la quinta lugares donde se situarán las alpacas. Las dos últimas son opcionales.

Cada capa excepto las dos últimas se define por una textura de cualquier tamaño y extensión, una normal de la textura también de cualquier tamaño y extensión y una máscara de tamaño 1024 x 1024 en escala de grises y extensión .png que indica los lugares donde se aplicará la textura y con qué intensidad.

Para el caso del asfalto, la intensidad será siempre la máxima en el circuito (blanco) y la mínima en el resto (negro).

En las dos últimas capas sólo se necesita la máscara, y tendrá todo negro excepto los puntos en los que existirá un árbol o alpaca, según si es la capa 4 o 5.

Todos los elementos creados para las capas deben de almacenarse en la carpeta del juego media/scene1/.

Foto

Al igual que para el caso del patinador, hay que incluir una foto del circuito de 300 x 225 píxeles con extensión .jpg en la carpeta media/MyGUI_Media/, esta debe de tener el mismo nombre que el futuro fichero xml del circuito para que el programa pueda indexarlo.

Xml

El circuito necesita un xml con nombre igual al nombre del circuito y extensión .dhsk. Este debe de guardarse en la carpeta del proyecto media/gameData/tracks/. Se recomienda copiar uno existente a modo de guía.

En el xml hay que incluir el nombre, que debe de ser igual que el nombre del fichero, y la descripción del circuito. También se debe de incluir el fichero que representa el mapa de

alturas, con el tamaño que tendrá el circuito en el juego (aproximadamente 4 veces el tamaño real) y la diferencia de altura entre el punto más alto y más bajo (aproximadamente 4 veces la diferencia real).

En las siguientes líneas se incluyen las capas del juego, siempre primero la capa de asfalto. Si la cuarta y quinta capa no se incluyen, se borran del fichero.

Finalmente se indica el punto inicial y final de la partida tomando el mapa de alturas como referencia con el punto x = 0 en la parte de más a la izquierda, el punto y = 0 siempre y el punto z = 0 en la parte superior de la imagen.

Además hay que indicar la rotación inicial del patinador el primer campo indica la inclinación, normalmente a 0 y el segundo la rotación en grados. En 0 indica que el patinador mira hacia la parte superior del mapa. El número de grados se cuenta rotando al patinador hacia la derecha.

La última línea indica el record y el jugador con el mejor tiempo. Como record inicial se recomienda poner un tiempo de 23:23:999 y como jugador “nobody”.

Para cada línea hay que crear el código se seguridad siguiendo el esquema marcado en el diseño.