DEPARTAMENTO DE INFORMÁTICA

PROYECTO DE FIN DE GRADO

MODELADO Y SIMULACIÓN DE TRAYECTORIAS NAVALES Y SU REPRESENTACIÓN EN UNITY

Ángel García-Capelo Blanco

Junio de 2012

Autor: Ángel García-Capelo Blanco

Tutor: Antonio Berlanga de Jesús

Tabla de Contenidos

Tabla de Ilustraciones

1. Introducción

Este primera apartado, se dedica a la descripción explicación de las razones y motivos que han llevado a la elección y realización del proyecto, al igual que una introducción general al problema, los objetivos y una breve descripción del documento.

1.1. Motivación

La simulación y predicción del mundo real usando ordenadores ha sido siempre uno de los principales usos que ha recibido la informática a lo largo de toda su historia, y ha condicionado considerablemente su desarrollo. Desde muy pronto de la aparición de los ordenadores, ya en los años 50, el Ejército de los Estados Unidos desarrolló un sistema semi-automático para controlar y predecir los bombardeos enemigos. Ese sistema, llamado SAGE, se usó durante la Guerra Fría.

Ilustración 1. Interfaz visual del sistema SAGE

Pero este no fue el primer simulador del mundo. Ya en 1947, *Thomas T. Goldsmith Jr.* y *Estle Ray Mann* crearon el primer simulador. Este consistía en la simulación de lanzamiento de misiles sobre un objetivo.

Un gran salto que se dio, fue en 1958, cuando *Willy Higginbotham* creó *Tennis for Two*, que supuso por primera vez la representación visual de una simulación de un videojuego, en este caso, sobre un osciloscopio.

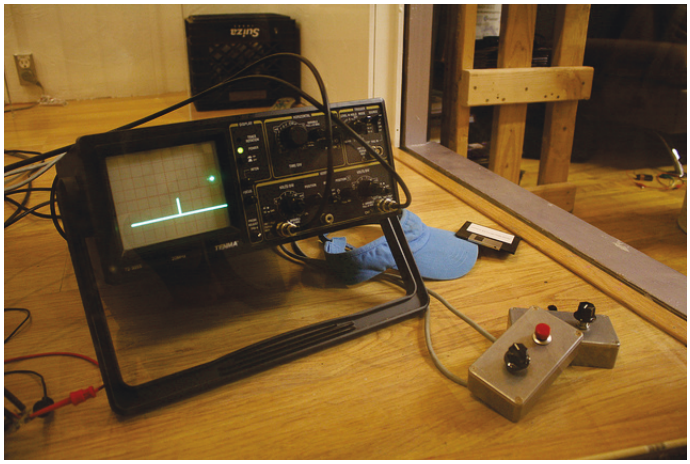

Ilustración 2. Reconstrucción actual de Tennis for Two

Con el entorno de la guerra fría, muchos otros simuladores fueron realizados con el fin de analizar el estado actual del mundo (ya fuese mediante RADAR o sistemas por satélite), y hacer una serie de predicciones, tales como lanzamientos de ICBM, o control del tráfico aéreo entre otros.

No obstante, la guerra fría no solo propició la aparición de estos simuladores, los cuales todos tomaban datos del mundo real, sino que la necesidad de reducción de gasto en el entrenamiento de los pilotos, propiciaron una nueva línea de simuladores que, en vez de tomar datos del mundo real y predecir un posible futuro, estos transcurrían en un mundo completamente virtual. Estos simuladores, permiten a los aprendices (especialmente pilotos de aeronaves), probar o practicar situaciones peligrosas sin que haya peligro de muerte para el aprendiz, ni de perdida de material militar (generalmente, de muy alto coste). No obstante, no solo se hicieron para la aviación, sino que se han realizado simuladores militares de todo tipo de vehículo.

Ilustración 3. Piloto militar en un simulador

Ilustración 4. Simulador para los tanques Leopardo 2E del ejército Español

Esta simulación, siempre está limitada a la potencia actual, por lo que ciertos grados de realismo deben ser eliminados para poder ser ejecutados en tiempo real. En este tipo de simulador militar, suele tener un apartado gráfico más limitado que el que podría tener un videojuego, pero las características de físicas suelen ser muy cercanas a la realidad.

Sin embargo, no solo se han realizado simuladores de tiempo real, sino que otro tipo de simuladores se han usado para la ayuda en el diseño y construcción de artefactos en los que se busca algún tipo de característica concreta. Uno de los casos más famosos el avión de ataque furtivo *Lockheed F-117 Nighthawk*. En el diseño de este avión, la informática y los simuladores tuvieron una gran importancia, ya que gracias a la predicción de los datos del RADAR, se pudieron obtener los ángulos y perfiles necesarios para la producción del primer avión furtivo diseñado para serlo. No obstante, la propia limitación del Hardware existente para los cálculos, obligó a que el avión solo pudiese contar con superficies rectas, lo cual hizo que su aerodinámica fuese muy mala, lo que conllevó que fuese muy difícil de controlar y fuese incapaz de realizar maniobras complicadas. Siguiendo esta línea de desarrollo, se ha llegado a la fabricación del *Lockheed Martin F-22 Raptor*, el cual conserva las capacidades furtivas del anterior modelo, pero con una aerodinámica muy avanzada, lo que le convierte, si no en el mejor, en uno de los mejores cazas hoy en día.

Todos estos desarrollos fueron llevados en el ámbito militar, ya que al disponer de gran cantidad de fondos, permite el uso de tecnologías punteras. No obstante, si bien, los militares fueron los primeros, no han sido los únicos. Con los años, estas técnicas han llegado al mundo civil, y se ha hecho un uso muy similar al que se le ha dado en el ámbito militar. Entrenamiento para la aviación civil, o el diseño de los aviones, entre otros, usan simuladores casi idénticos a los usados en el ámbito militar.

Ilustración 5. Simulador de un avión comercial

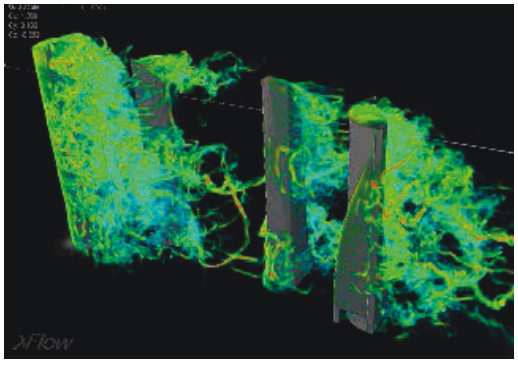

Ilustración 6. Simulación de Viento en las 4 torres de Madrid (X-Flow [Next Limit])

Todo esto llevó a que los simuladores fuesen un paso más allá, y acabasen llegando al ámbito de la industria del entretenimiento. Como ya se vio con *Tennis for Two*, el uso de simuladores y los videojuegos han estado íntimamente ligados. De hecho, es complicado indicar que videojuego es un simulador y cuál no, ya que un simulador se encarga de reproducir un sistema, y todos los videojuegos son un sistema (el cuál puede ser más o menos real). No obstante, se ha venido utilizando la clasificación de simulador a los videojuegos cuyo sistema está basado en el mundo real, y que intentan hacer una aproximación correcta (en comparación con otros, que sacrifican realismo a cambio de jugabilidad o diversión).

Otro uso que han recibido los simuladores en la industria del entretenimiento ha sido su utilización para la recreación en largometrajes y cortometrajes de situaciones, o bien peligrosas, extremadamente caras, o directamente imposibles de llevar a cabo en la realidad. En este campo, existen multitud de simuladores, que hoy en día son capaces de hacer películas en las que es imposible el distinguir que es real que está generado por ordenador.

Ilustración 7. Simulación del movimiento del agua al navegar un barco

La **motivación** para la realización de este proyecto nace con el propósito de poder usar toda la tecnología y conocimientos de los simuladores para el poder realizar predicciones correctas y lo más precisas posibles sobre cuál sería el movimiento y rumbo de un barco, para poder usarlo tanto en el ámbito real como en el ámbito del entretenimiento.

1.2. Objetivos

El objetivo de este proyecto es el desarrollo de un sistema en el motor Unity capaz de predecir y calcular la trayectoria de un barco en un entorno libre de oleaje, mareas y corrientes, lo más realista posible, pero garantizando el funcionamiento del sistema en tiempo real.

El proyecto desarrollado permitirá el manejo de un barco manualmente (utilizando órdenes directas sobre el timón y las hélices), y también un sistema por el cual se introducen una serie de puntos, y el barco navegará hacia ellos.

La configuración del barco es una configuración predeterminada, con los valores ajustados al barco creado para la demostración. Esos valores son fácilmente intercambiables, pero para hacerlo, se requeriría un recompilado de la aplicación.

El entorno consiste únicamente en un plano para el agua y otro para el fondo. En el modelo, se han hecho las animaciones pertinentes para las superficies de control del barco (hélices y timón).

Los controles han de permitir un control fácil de la cámara, al igual que el del propio barco. Para ello, se utiliza las teclas WASDQE para movimiento de la cámara, mientras que para el movimiento del barco se usa IJKL.

El simulador no tiene un objetivo en sí. Únicamente se trata de ver el movimiento del barco. Une pequeña interfaz mostrando el estado del barco ayuda a su control.

Los objetivos adicionales que implican la realización de este proyecto son los siguientes:

- Estudio y aprendizaje del entorno gráfico Unity.
- Estudiar y comprender en profundidad las dinámicas que afectan al movimiento de un barco en el mar.
- Un mínimo de modelado de los objetos a representar, para tener una representación visual comprensible.

1.3. Estructura del documento

En este apartado, se hace un pequeño resumen del contenido del documento, explicando los capítulos y anexos incluidos.

El capítulo 1, **Introducción**, se hace una primera presentación del proyecto, explicando los motivos de su realización, al igual los objetivos que se persiguen. También hay una breve descripción de la estructura del documento.

En el capítulo 2, cuyo título es **Estado del Arte**, se exponen las alternativas de los métodos posibles para la realización del proyecto, indicando sus ventajas, inconvenientes y otros factores que han motivado las elecciones tomadas. Entre ellas se encuentran en primer lugar, alternativas a motores Gráficos, entre los que destacan:

- Unity
- Unreal Engine
- Cry Engine
- Ogre
- Source Engine

También se estudiarán las alternativas a los modelos de representación de la física de los barcos, entre los cuales destacan los siguientes:

- Bla
- Bla Bla

En este punto, también se analizan las compatibilidades o problemas que se producen intentar juntar los distintos modelos.

Para el capítulo 3, titulado **Desarrollo e Implementación**, se exponen todos los aspectos técnicos que se han requerido para la realización del proyecto, incluyendo las fases de análisis, en las cuales se estudiaban los requisitos funcionales y no funcionales, al igual que la elaboración de los diagramas para recrear el sistema. También está incluida la fase de implementación, en la que se explican los métodos y sistemas usados para la realización y puesta en marcha del proyecto.

En el capítulo 4, llamado **Conclusiones y trabajos futuros**, se presentan los resultados y las conclusiones a las que se ha llegado tras la realización del proyecto, al igual que se comentan las posibles mejoras y/o expansiones que puedan llevarse a cabo sobre este proyecto en particular.

En los capítulos 5 y 6, llamados respectivamente **Bibliografía** y **Anexos**, se presentan los materiales adicionales, al igual que las referencias que se han usado para la realización del proyecto. También se incluyen anexos sobre el marco regulador y la planificación del proyecto.

2. Estado del Arte

En esta sección, se presentan las alternativas para la elección tanto del modelo físico del barco, como tanto del motor gráfico. Se presentan tanto los utilizados finalmente como otras propuestas que se han estudiado.

2.1. Sistemas gráficos

En esta primera sección, se describen los sistemas gráficos y sus características, ventajas y desventajas a la hora de aplicarlas en este proyecto.

2.1.1. Unity

Unity es un motor gráfico multiplataforma, creado y desarrollado por "Unity Technologies". Además del motor, se puede obtener una plataforma de desarrollo, para una elaboración más fácil y cómoda. Dicha plataforma está disponible para los sistemas *Microsoft Windows* y *OS X*.

El motor gráfico, actualmente permite la creación de aplicaciones (también llamadas videojuegos, debido a la naturaleza del motor y a que la mayoría de sus productos lo son;) para las siguientes plataformas: *iOS*, *Android*, *Windows Phone 8*, *BlackBerry 10*, *Tizen*, *Microsoft Windows*, *Aplicaciones de la Windows Store*, *Mac*, *Linux & Steam SO*, *Navegador Web* (incluidos *Explorer*, *Firefox*, *Chrome* y *Safari*), *WebGL*, *PS3*, *PS4*, *PSVITA*, *XBOX ONE*, *XBOX 360*, *WiiU*, *AndroidTV*, *Samsung Smart TV*, *Oculus Rift*, *Gear VR* y *Microsoft Hololens*.

El motor está programado en C++ y utiliza Direct3D, OpenGL e interfaces propietarias en las consolas, lo que garantiza un alto rendimiento a la hora de recrear los gráficos. De momento no soporta DirectX 12.

Unity, posee dos tipos de licencias, la primera llamada *Unity Free*, que es gratuita y tiene ciertas limitaciones en tecnologías avanzadas de Render, al igual que una pantalla de carga fija que no se puede ni personalizar ni quitar, y una marca de agua en las versiones de navegador. Esta licencia es gratuita.

La otra licencia, llamada *Unity Pro,* es una licencia de pago y permite el acceso total a todas las características que posee Unity, al igual que quita cualquier tipo de marca de agua o pantalla de carga obligatoria. Permite una modificación total.

Existen otro tipo de licencias, más especiales y que requieren negociación con Unity. Entre estas se encuentran las licencias a entidades educativas, licencias para el uso en consolas, y otras licencias específicas, como por ejemplo, una licencia para organizaciones de juegos de azar.

2.1.1.1. Historia

Unity empezó su desarrollo en el año 2004 de manos de la empresa *Over The Edge Entertainment* (ahora *Unity Technologies*), a manos de Devid Helgason, Nicholas Francis y Joachim Ante, tras un fracaso del primer videojuego que crearon, llamado *GooBall*. El juego, que no tuvo éxito, había requerido de la elaboración de las herramientas necesarias para la creación de un videojuego.

Ilustración 8. Videojuego GooBall, de cuyas herramientas

Estas herramientas, cuando se hacen únicamente para un videojuego, suelen ser muy sencillas y las interfaces no suelen ser trabajadas, ya que es el medio para la creación de un único proyecto.

Sin embargo, para este habían desarrollado unas herramientas que consideraron buenas, y que, dado el trabajo que habían llevado a su creación, y la necesidad de reorientar la empresa tras el primer fracaso, tomaron la decisión de crear un motor de videojuegos, bajo el lema de "democratizar el desarrollo de los videojuegos".

Bajo este lema, iniciaron el desarrollo de un motor para videojuegos que pudiese utilizar cualquier empresa (grande, mediana o pequeña), y que cuyo entorno fuese lo más amigable posible (tanto para programadores, artistas, diseñadores de niveles, etc…).

Las primeras versiones, solo fueron desarrolladas para *Mac* (al igual que lo había sido el videojuego *GooBall*). Ya desde el principio contaba con dos tipos de licencias, aunque por aquel entonces, los nombre eran *Indie* y *Profesional*, y ninguna era gratuita (la primera tenía un coste mínimo de 300 $, mientras que la segunda tenía un precio a partir de 1.500 $).

En el año 2008, con el lanzamiento de *iPhone*, al ser uno de los pocos motores que lo soportaban y tener un precio muy accesible, Unity tuvo un gran auge, que se incrementó 2010 con el lanzamiento para otras plataformas, además de las de Apple. También, en el año 2009, la licencia *Indie* pasó ya a ser gratuita.

Para la versión 4.0, Unity dio otro gran salto en cuanto a uso y público al que llegaba, pues firmó acuerdos con *Sony*, *Microsoft* y *Nintendo* (las compañías que dominan el mercado de las consolas) para permitir el desarrollo de videojuegos usando Unity para sus respectivas consolas.

Todas estas mejoras y características son las que han permitido a Unity tener una presencia del 45% en todos los videojuegos que usan motores de videojuegos, muy por encima de sus competidores.

2.1.1.2. Características

La versión 5 de Unity, entre otras, tiene estas características principales:

- **Lenguajes:** Admite C#, JavaScript y Boo para la implementación de los scripts del juego.
- **Arquitectura:** Disponible en 32 y 64 bits y para casi todas las plataformas.
- **Flujo de Trabajo:** Un editor de 64 bits, que permite la adición de complementos, y con un pipeline de importación directo.
- **Animación:** Retargeting, Editor de animaciones integrado, cinemática inversa, árboles de mezcla, máquinas de estado y capas sincronizadas y curvas avanzadas.
- **Gráficos:** Shading físico, Sistemade partículas Shuriken, Iluminación Global a tiempo real de Enlighten, acceso al rendering de bajo nivel, renderizado a teturas y efectos de postprocesamiento de pantalla completa.
- **2D:** Físisca 2D avanzada. Animación automática de sprites, Compresión y empaquetado de sprites y otros…
- **Físicas 3D:** Simulación físisca multihilo, simulación de tejidos, física avanzada para vehículos, gran precisión en la detección de colisiones y más.
- **Optimización:** Generación automática de LOD y niveles de detalle, Occlusion CUlling, Profiler, Deferred rendering, stencil buffer Access, GPU Skining para DirectX 11 y OpenGL ES 3.0, Streaming con Asset Bundles y Dynamic Batching.

- **Audio:** Jerarquías de mezcladores, invocación de scripts desde la reproducción de una animación, transición de audios en el entorno y plugins de audio nativos.
- **Scripting:** Depuración del reproductor web, Soporte .NET Socket, Inspector GUI para clases personalizadas, acceso a los datos de la web a través de las funciones WWW, abrir URL en el navegado del cliente, soporte de plugin de código nativo y acceso de scripts al Pipeline de Activos.
- **Otros:** Soporte externo de control de versiones, terrenos avanzados, soporte integrado de SpeedTree, Networking con RakNet para múltiples jugadores, NavMeshes y Path-Finding, Net Streaming, generación de informes de error, reproductor en Linux sin procesador y prueba instantánea de los juegos.

2.1.1.3. Conclusiones

Unity, a pesar de no llevar mucho tiempo en el mercado y compitiendo, ha conseguido alcanzar la cima en muy poco tiempo gracias a su política de precios accesibles para todas las necesidades. Esto hace que sea una opción muy atractiva, pues hay una gran cantidad de documentación en la Web, y el aprendizaje del mismo es bastante rápido e intuitivo.

Su apartado técnico sigue mejorando a un gran ritmo, pero todavía no alcanza a grandes competidores en este aspecto, como por ejemplo, CryEngine 3 o Unreal Engine 4 entre otros.

El hecho de que pueda ser portado a casi cualquier plataforma (aunque esto no es tan rápido e intuitivo como lo venden), lo hace también tener un gran atractivo.

2.1.1.4. Ejemplos

Ilustración 9. Algunos ejemplos de Videojuegos de Unity

2.1.2. Unreal Engine

Unreal Engine es un motor de videojuegos multiplataforma creado y desarrollado por la empresa "Epic Games". Además del motor, se puede obtener una plataforma de desarrollo, para una elaboración más fácil y cómoda. También proporciona editores con capacidades limitadas, para un aprendizaje más rápido y sencillo con el nombre de *UDK*.

El motor, actualmente permite la creación de videojuegos para las plataformas de las siguientes marcas: *Mac*, *Android*, *Web*, *Linux*, *Oculus Rift*, *Play Station*, *Steam OS*, *Windows* y *XBOX*.

El motor está programado en C++ y utiliza Direct3D, OpenGL e interfaces propietarias en las consolas, lo que garantiza un alto rendimiento a la hora de recrear los gráficos. Es de los pocos motores que utilizan DirectX 12.

Unreal Engine ha cambiado recientemente los tipos de licencia, simplificándolos en una sola licencia estándar. Esta licencia permite la adquisición libre y gratuita del motor, y el pago se produce sobre los beneficios, adquiriendo un 5% del precio de la aplicación producida. A diferencia de otros motores, esta licencia "gratuita" da un acceso total al motor, permitiendo utilizar el 100% de su potencia antes de tener que pagar nada por él.

Existen otro tipo de licencias, más especiales y que requieren negociación con Unreal Engine. Ese tipo de licencias suelen estar reservadas para grandes estudios, en

los que se hace una adquisición del programa y se utiliza para gran cantidad de videojuegos.

El cambio en las licencias se debe al auge que ha tenido otros motores, tales como Unity.

2.1.2.1. Historia

De los motores aquí mostrados, Unreal Engine es el que más antigüedad tiene, remontándose a 1.998. No obstante, hay que indicar que cada versión, a diferencia de Unity, por ejemplo, es un motor completamente nuevo, generalmente incompatible con los proyectos de la versión anterior.

En 1.998 los juegos 3D estaban empezando a popularizarse, de hecho, este año es conocido por haber dejado gran cantidad de videojuegos que han pasado a la historia, tales como *Metal Gear Solid* (PS1), *Half-Life* (PC), *The Legend of Zelda: Ocarina of Time* (Nintendo 64), *Starcraft* (PC [este no era 3D]) entre otros. Esto se debe a la salida al Mercado de la *Nintendo 64*, y a la popularización del puerto AGP, que permitía conectar tarjetas gráficas que daban un salto de potencia en los ordenadores.

Entre estos títulos, también salió al mercado un nuevo videojuego llamado *Unreal.* Este videojuego fue el que utilizó el Unreal Engine por primera vez. No obstante, este juego tuvo un éxito moderado, y no sería hasta su secuela, *Unreal Tournament*, que el motor se haría famoso.

Ilustración 10. Logo de Unreal y Unreal Tournament.
Se puede ver la inspiración del logo del motor en esta saga.

El éxito de Unreal Tournament no solo se debió a la calidad del juego y del motor gráfico, pues su competidor directo (Quake III Arena), tenía prestaciones similares. La verdadera clave del éxito fue su facilidad para poder ser modificado y crear nuevos modos de juego, alargando su vida útil, variedad y haciendo que el juego ganase claramente la partida a su rival.

Esta victoria, no solo hizo que el juego fuese un gran éxito comercial, sino que además hizo ver a Epic Games que la facilidad de desarrollo podía animar a la gente a la realización de no solo contenido extra para su propio videojuego, sino desarrollar uno nuevo por completo. Con esta idea en mente, se creó el Unreal Engine 1 en un momento en el que solo había otro motor gráfico 3D (Quake Engine).

En esa época, los videojuegos no tenían tantos elementos visuales como pueden tener hoy en día (sombras reales, iluminación real global, etc…). Eso hacía que la mayor parte del tiempo de desarrollo fuese para el contenido del juego, y no tanto para el motor. Por eso, en esa época todavía era muy común que cada juego tuviese su propio motor, o una copia modificada de raíz de algún antecesor. Este motivo hizo que la cantidad de videojuegos que utilizaban el motor gráfico no fuese demasiado alta, pero sirvió para derrotar a sus competidores.

La versión 2 del motor gráfico fue lanzada en 2.002 con el juego gratuito America's Army. El motor gráfico fue reescrito completamente y tuvo una gran cantidad de mejoras sustancias, como la adición de físiscas más complejas, Ragdolls; y una nueva variedad de avances gráficos.

Esta segunda versión del motor gráfico llegó a muchos más videojuegos que su antecesor, y fue usado en prácticamente todo tipo de videojuegos, desde shooters hasta MMORPG. También este motor se usó numerosas veces en consolas, tales como la *GameCube*, *XBOX* o la *PlayStation 2*. Con su actualización a la versión 2.5 la velocidad de renderizado y la calidad de las físicas se mejoraron considerablemente. Algunos videojuegos, como el Lineage 2 todavía siguen usando este motor.

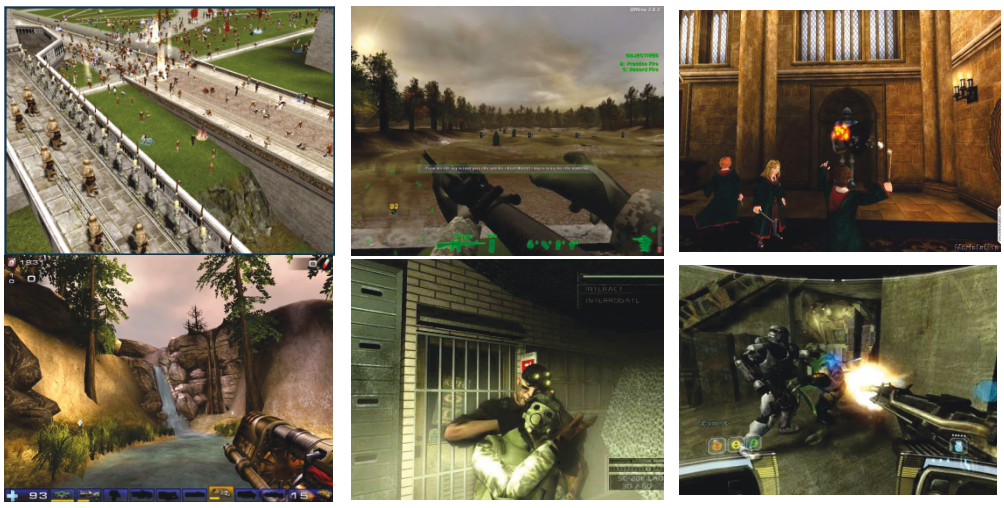

Ilustración 11. Videojuegos que usan Unreal Engine 2. De izquierda a derecha y arriba abajo. Lineage 2, America's Army, Harry Potter y el Prisionero de Azkaban, Unreal Tournament 2004, Tom Clancy's Splinter Cell: Chaos Theory y Star Wars: Republic Commando

Con el éxito que había cosechado Unreal Engine 2, Epic Games empezó el desarrollo de la tercera versión de su motor gráfico en 2003. Esta vez, los cambios con respecto a la versión anterior fueron tan grandes, que más que reescritura del motor, fue un proyecto nuevo.

Los grandes fabricantes de tarjetas gráficas, Intel, Nvidia y ATI (ahora parte de AMD), empezaron a utilizar nuevos shaders en las gráficas, que para poder hacer un uso completo de ellas, había que cambiar radicalmente el enfoque del render. Por eso, esta versión traía nuevas mejoras, como el uso del shader model 3.0, o cálculos de luz realizados por cada pixel, en vez de cada vértice, que era lo habitual hasta entonces. Se empezaron a incluir gran cantidad de efectos de luz, como sombras en tiempo real, reflexiones y refracciones, mapas de normales, etc…

Esta versión del motor gráfico ha sido la que ha tenido vida más larga, y de hecho, hay juegos que han salido al mercado en el año 2.015 que usan este motor gráfico.

Durante la vida de este motor gráfico, nacieron otros muchos, entre los cuales destaca Unity. Viendo el éxito comercial del enfoque realizado por Unity, en 2.009, Epic Games sacó una versión reducida de su motor gráfico, bajo el nombre de UDK (Unreal Development Kit), con unas licencias muy competitivas. La descarga era gratuita, y para su publicación, se había de realizar un pago de 99 $, y una vez alcanzados los 50.000 $, pagar un 25% del precio del producto.

Este método acercó al motor, ya muy conocido entre los estudios, al ámbito de desarrolladores autónomos, e incluso a aficionados.

Ilustración 12. Evolución del motor Gráfico en sus 3 primeras versiones. Juegos: *Unreal Torunament*, *Unreal Torunament 2004* y *Unreal Tournament 3*

Los videojuegos creados en esta versión del motor gráfico son muchos, ya que ahora el estándar de los gráficos de cualquier videojuego (a excepción de los que simulan efectos "*Retro*") es muy alto. Por ese motivo, ahora el desarrollo del motor gráfico puede ser el 80-90% del trabajo de desarrollo de cualquier videojuego. Por ese motivo, ahora solo los grandes estudios se hacen sus propios motores gráficos.

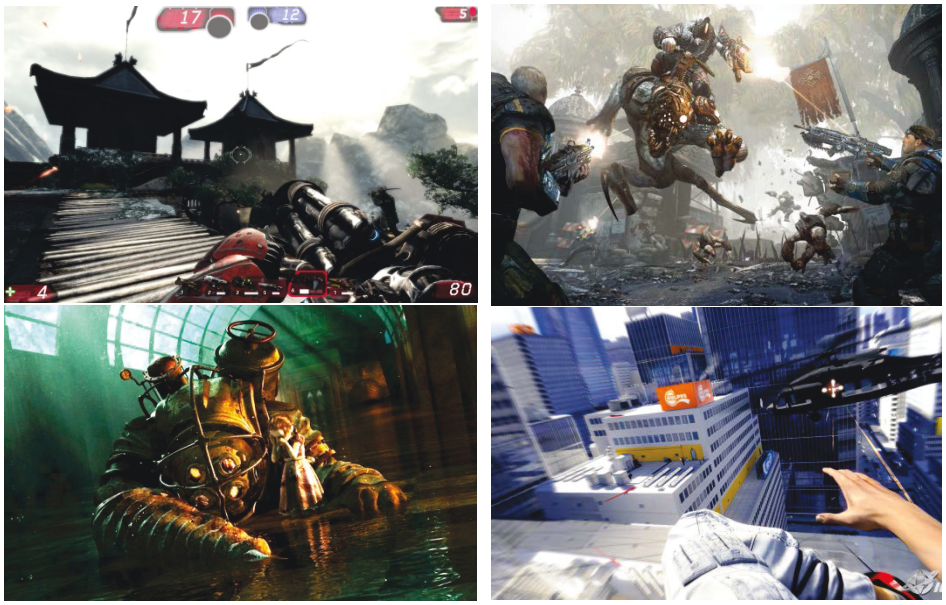

Ilustración 13. Videojuegos que usan Unreal Engine 3. De izquierda a derecha y arriba abajo: Unreal Tournament 3, Gears of War, BioShock y Mirror's Edge

Al igual que con la tercera versión, la cuarta versión comenzó su desarrollo nada más salir al mercado la anterior versión del motor. Por eso, esta versión ha sido la que más tiempo ha estado en desarrollo, desde 2005. No fue hasta mediados de 2008 cuando se incorporó un equipo de verdad, ya que hasta el momento solo había estado trabajando un único ingeniero. Aun así, el desarrollo se prolongó hasta mayo de 2.012, momento en el que se lanzó la primera revisión de la cuarta versión del motor.

Esta versión, al igual que todas las anteriores, ha sido un proyecto nuevo, en el que no se han tenido en cuenta compatibilidades con motores anteriores. Si bien, esto hace que sea uno de los motores más potentes que existen en el mercado (muestra escenas de una calidad máxima sin necesitar un ordenador de 3.000 €), también juega en su contra. El adaptarse al nuevo motor está llevando su tiempo, y ese es el motivo por el cual, a fecha de 2015 la cantidad de juegos que han salido de este motor es mínima.

Esto, sumado al hecho de que Unity puso una versión reducida de su motor completamente gratuita, obligó a Epic Games poner un precio bastante más bajo del que tenía anteriormente por la tercera versión del motor. Esta primera

licencia requería un pago mensual de 19,95 $, y un 5% de las ganancias del producto una vez superados los 3.000 $. No obstante, esto no fue suficiente, y en Febrero de 2015 han modificado la licencia, de modo que ahora no hace falta pagar la suscripción mensual, y cualquier persona tiene acceso total al código sin compilar del motor, pudiendo hacer cualquier modificación total sobre cualquier parte del motor antes de pagar nada.

Ilustración 14. Logo de la cuarta versión del Unreal Engine

Al igual que Unity, Unreal Engine 4 proporciona un editor para la creación de contenido, con una característica única, y es la de poder recompilar el código con el juego en marcha y poder ver los resultados al momento (no todas las partes del código pueden ser recompiladas en tiempo de ejecución).

2.1.2.2. Características

- **DirectX 11 y 12:** Soporte para DirectX 11 y 12 y las funcionalidades que conlleva, tales como reflexions HDR en toda la escena, miles de luces dinámicas por escena, teselado programable, desplazamiento basado en físicas de los shading en los materiales entre otros.
- **Cascade Visual Effects:** Es un editor de partículas, las cuales son procesadas por la GPU, permitiendo millones de partículas simultaneas, las cuales tienen interacciones físiscas.
- **Material Pipeline:** Unreal Engine 4 tiene un sistema de materiales basado completamente en la físisca, con una interfaz completamente nueva.
- **Bluprint Visual Scripting:** Interfaz visual para la realización y personalización del juego, sin necesidad de tocar una sola línea de código.
- **Live Bluprint Debbuging:** Se permite la interacción de todos los elementos con el juego pausado, y al reanudar, todos los cambios son aplicados, acelerando el proceso de debug.

- **Content Browser:** Un nuevo sistema para organizar, previsualizar e importar objetos y cualquier tipo de contenido al editor.
- **Persona Animation:** Nuevo sistema de físicas para la animación realista y rápida, especialmente de modelos con esqueletos.
- **Matinee Cinematics:** Nuevo editor para escenas, con el que programar las escenas y secuencias será rápido, fácil, sencillo e intuitivo.
- **Terrain & Folliage:** Posibilidad de crear grandes y bastos mundos gracias al sistema automatico de *LOD* con un uso de la memoria optimizado.
- **Post-Process Effects:** Gran variedad y completamente personalizables e efectos post-renderizado, oclusión ambiental, destellos, corrección de color, profundidad de campo, sobreexposición, antialaising, etc…
- **Full Source Code Access:** Acceso total al código del motor, escrito en C++. De este modo, se puede personalizar cualquier parte del motor, sin límites ni imposciones.
- **C++ Code View:** Acceso directo a las funciones del código, que te llevan directamente a Microsoft Visual Studio con la función que se desea modificar abierta.
- **Hot Reload Function:** Recompila el código y mira como el juego actualiza el código sin reiniciar la partida o la escena.
- **Simulate & Immersive Views:** Una gran variedad de modos de prueba, para poder probar el juego en cualquier estado.
- **Instant Game Preview:** Actualiza el juego y observa su funcionamiento al instante, sin esperar a que se construya ningún archivo.
- **Possess & Eject Features:** Juega el juego desde la vista del jugador, o abandona su cuerpo al momento para comprobar cualquier objeto de la escena.
- **Artificial Intelligence:** Una nueva inteligencia artificial, con la inclusión de navigation meses dinámicas.
- **Audio:** Nuevo editor de audio, para un control total del mismo.

Y otros…

2.1.2.3. Conclusiones

Si Unity se tiene la mayor parte de los proyectos, Unreal Engine los tiene de los proyectos que son grandes. El motor con más juegos vendidos, y con 17 años de experiencia.

Unreal Engine, al permitir el acceso al código, permite un control total. Además, los gráficos de Unreal Engine 4 están todavía un escalón por encima de los de Unity.

Sin embargo, la realización de proyectos es más inmersiva, requiriendo más tiempo para proyectos pequeños. Además, no está en tantas plataformas como Unity (aunque están trabajando para ampliarlas).

En resumen, la curva de aprendizaje es mayor que con Unity, pero el resultado también lo es. También, como lleva poco tiempo, existe poca documentación (aunque está aumentando rápidamente).

2.1.2.4. Ejemplos

Ilustración 15. Algunos ejemplos de videojuegos de Unreal Engine 4

2.1.3. Cry Engine

CRYENGINE®

CryEngine es un motor de videojuegos multiplataforma creado y desarrollado por la empresa "Crytek". Además del motor, se puede obtener una plataforma de desarrollo, para una elaboración más fácil y cómoda.

El motor, actualmente permite la creación de videojuegos para las siguientes plataformas: *Microsoft Windows*, *PlayStation 4*, *PlayStation 3*, *Xbox 360*, *Xbox One*, *WiiU*, *Android* e *iOS*.

El motor está programado en C++ y Lua y utiliza Direct3D, OpenGL e interfaces propietarias en las consolas, lo que garantiza un alto rendimiento a la hora de recrear los gráficos. De momento no soporta DirectX 12.

CryEngine se ha sumado a las licencias que utilizan Unity y Unreal Engine, haciendo una licencia en la que se paga una suscripción mensual de 9,99 $/€, en la que se proporciona un acceso limitado al motor, y otra licencia de pago (pero de negociación individual), que da acceso completo al código.

2.1.3.1. Historia

El motor CryEngine nace en el año 2001 a manos de la empresa Crytek como un motor de demostración para Nvidia. Cuando la compañía vio el potencial se decidió la creación del motor, junto con el videojuego para exponerlo *FarCry*.

Fue el primer motor gráfico en implementar shading pixel y HDR en tiempo real. Como había estado en contactos con Nvidia, también incorporó rápidamente las mejoras que podían sacar partido a los *Pixel Shader 3.0* y *Vertex Shader 3.0*.

Fue de los primeros motores en permitir unos espacios abiertos grandes, como se hizo en el videojuego FarCry, que salió el 23 de Marzo de 2.004, inaugurando un nuevo y potente motor gráfico.

Ilustración 16. *FarCry 1*. Unos gráficos que suponían una revelación para la época

No obstante, debido a una pobre campaña y una compra rápida del motor por parte deUbisoft, que adquirió junto con ello los derechos de FarCry, hicieron que este motor solo se usase para esta saga y un par de juegos más, de los cuales, uno, *Aion*, sufrió fuertes modificaciones para poder adaptarlo a las necesidades de un MMORPG. El otro videojuego fue *Never Ending Dream*.

Ilustración 17. AION, un MMORPG todavía funcional desarrollado con CryEngine 1.

Si el CryEngine 1 fue un gran salto de calidad, CryEngine 2 lo fue todavía más. Ya que el CryEngine 1 había sido vendido a Ubisoft, se creó desde cero. Su

anuncio se realizó enseñando imágenes del que pasaría a ser el primer videojuego de una saga destinada a demostrar la potencia de los motores Gráficos de Crytec.

Saliendo al mercado por primera vez el 13 de Noviembre de 2.007, con unos gráficos que dejaron al mundo impresionado (al igual que los requisitos para poder ejecutarlo), salió el Crysis 1, y con él se inauguró el CryEngine 2. El motor no era multiplataforma, y solo podía ser ejecutado en plataformas Microsoft Windows (de hecho, para portarlo ha sido necesario utilizar la siguiente versión del motor, CryEngine 3). Fue de los primeros videojuegos en incluir un ejecutable compatible con una arquitectura x86_64 que mejoraba el rendimiento un 10% respecto a la x86 normal. También fue el primer en tener como requisitos recomendados un sistema dual de Gráficas (ya fuese CrossFire o SLI).

Ilustración 18. *Crysis 1* a máxima calidad. Un juego que podría pasar por un lanzamiento de 2015

No obstante al igual que con la anterior versión del motor, debido a la poca o nula portabilidad, a los altos requerimientos entre otros factores, se han realizado menos de 10 juegos con este motor, aunque el motor todavía se ofrece y sigue estando vigente.

Con la experiencia de los motores anteriores, Crytek decidió anunciar que en el año 2.009 lanzaría en CryEngine 3. Al igual que su antecesor, usaría un videojuego para demostrar su capacidad técnica (el Crysis 2). Esta vez, el motor si sería multiplataforma, siendo en un principio compatible solo con Microsoft Windows, PlayStation 3 y Xbox 360. Posteriormente se fue anunciando la compatibilidad con otras consolas a medida que estas iban saliendo, tales como la WiiU o la PlayStation 4, entre otras.

El primer videojuego que usó este motor gráfico fue el Crysis 2, que salió el 22 de Marzo de 2.011. Si bien, no supuso ninguna revolución en cuanto a gráficos como lo fue su predecesor, sí que incluía muchas mejoras, entre la cual, la más destacable era la mejora de rendimiento, pues ahora este juego se podía ejecutar en un ordenador más modesto (aunque sigue teniendo unos requerimientos de potencia mayores que los del Unreal Engine o Unity).

Ilustración 19. Crysis 2. Primer videojuego en usar CryEngine 3.

Esta nueva versión del motor, sí está siendo usado por terceras compañías para desarrollar sus videojuegos, y en los 4 años que lleva de vida, lleva más de 20 videojuegos.

Además, esta versión está sujeta a constantes revisiones y actualizaciones, lo que le permite extender su vida útil y poder competir contra otros motores más nuevos, tales como el Unreal Engine 4, por ejemplo.

2.1.3.2. Características

- Actualización instantánea y en tiempo real para cualquier plataforma, dentro del editor (Sandbox).
- Sistema de vegetación y terreno avanzados integrados en el motor.
- Sistemas de partículas en tiempo real.
- Herramientas para la creación rápida y sencilla de ríos, suelos, caminos, etc…
- Creación asistida de vehículos.
- Soporte para múltiples núcleos.
- Iluminación dinámica en tiempo real.

- Iluminación global avanzada y suavizado de sombras dinámicas.
- Niebla volumétrica de alta calidad.
- Texturas normales y avanzadas.
- Oclusión ambiental.
- Tecnología propia “Uber Shader”
- Raytracing
- Renderización de texturas
- Sonidos y música dinámicos
- Herramientas de análisis de rendimiento
- Físicas de alta calidad, incluyendo deformables, destruibles entre otras.
- Físicas con soporte multinúcleo.
- Agua tridimensional de alta calidad.
- Efectos naturales de alta calidad.
- Buscador de caminos para la IA.
- Edición avanzada de la IA.
- Editor de animaciones faciales.
- Motion blur y profundidad de campo.
- HDR.

2.1.3.3. Conclusiones

El CryEngine ha entrado a la escena de los motores gráficos con la última versión, después de dos versiones que se podrían considerar casi de prueba.

Con un rendimiento que está considerado de los más altos (junto con Unreal Engine), es el motor gráfico que está presente en más proyectos importantes después de Unreal Engine.

El motor ya se está acercando al final de su vida, de hecho, Crytek ya hecho anuncios sobre la nueva versión, aunque aún no han trascendido detalles.

Todo esto hace que sea un motor de una gran potencia, pero de una dificultad relativamente elevada.

Tampoco es un motor excesivamente utilizado por la comunidad (es más usado generalmente por los grandes estudios), y las complejidades con las licencias no ayudan a evitarlo, ya que son complicadas de obtener y no están especificadas claramente las condiciones.

No obstante, un equipo profesional con una licencia completa, con la cual se recibe apoyo de Critek, se puede llegar a hacer juegos del más alto nivel, sin nada que se pueda echar de menos.

2.1.3.4. Ejemplos

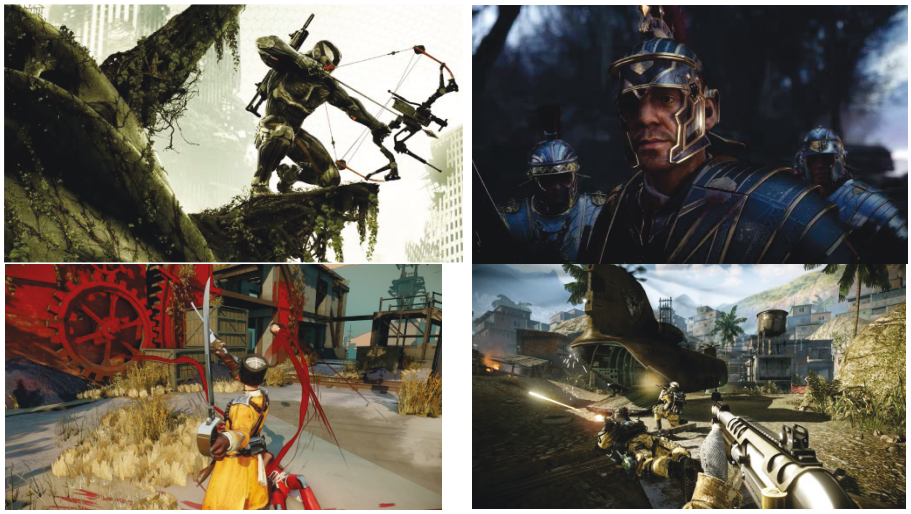

Ilustración 20. Algunos ejemplos de videojuegos de CryEngine

2.1.4. Ogre

Ogre (del inglés *Object-Oriented graphics Rendering Engine*), a diferencia de las anteriores opciones, no es exactamente un motor de videojuegos, sino que es únicamente un motor gráfico. Por lo tanto, cualquier añadido como sonidos, físicas entradas, controles o cualquier otro soporte necesario para un videojuego es necesario implementarlo manualmente.

El motor, al no estar dentro de un conjunto de herramientas, puede ser utilizado en prácticamente cualquier plataforma en la que se ajusten los parámetros adecuadamente. Actualmente tiene soporte apara *Microsoft Windows*, *Linux*, *OS X*, *Windows Phone 8*, *iOS* y *Android*, entre otros.

El motor está programado en C++ y utiliza Direct3D, OpenGL e interfaces propietarias en las consolas, lo que garantiza un alto rendimiento a la hora de recrear los gráficos. De momento no soporta DirectX 12, y DirectX11 parcialmente.

OGRE 3D posee una licencia de tipo MIT, lo cual quiere decir que su uso es completamente libre y legal sin necesidad de ningún tipo de reembolso ni pago previo de ningún tipo.

2.1.4.1. Historia

Ogre nació en 1.999, cuando un motor de renderizado nacido de un proyecto destinado a la creación de una librería gráfica basada en 3D orientada a objetos, llegó a ser completamente independiente y abstracto, que ya no se necesitaba Direct3D. A partir de ese momento, el creador del proyecto, Steve Streeting (cuyo alias era Sinbad), se propuso crear una API más completa e independiente de la plataforma.

A partir de ese momento, se emepezó a trabajar en convertir a Ogre en un verdadero motor gráfico.

En diciembre de 2.000 se añadieron una serie de clases fundamentales, tales como SceneManager o SceneNode, que hacían que el sistema empezase a ser funcional.

La primera versión beta fue creada en Febrero de 2.001, y funcionaba bajo DirecX 7. Ese mes también se hizo posible la carga dinámica completa del motor, permitiéndolo ser usado como librería dinámica.

Finalmente, el 4 de Marzo del mismo año se hicieron las primeras pruebas completas de funcionamiento, mientras que a finales del mismo mes se empezó a crear la extensa documentación que requería un proyecto de estas características.

Se fueron añadiendo soporte a mayas, entidades, conteo de frames, cuaterniones, skboxes y hasta Febrero de 2.015 se estuvo trabajando intensivamente, añadiendo los sucesivos DirectX 8 y 9 hasta lanzar la versión 1.0, con nombre en clave *Azathorth*.

Ilustración 21. Prueba de rendimiento de OGRE, con reflexiones y refracciones.

Desde ese entonces, la comunidad ha ido haciendo progresivas mejores y añadiendo nuevas características al motor, tales como el soporte a iPhone, añadido en Julio de 2.009.

2.1.4.2. Características

- Interfaz sencilla y cómoda orientada a objetos e independiente del sistema de renderizado Diirect3D, OpenGL y similares.
- Una gran cantidad de ejemplos funcionales para probar y aprender rápida y fácilmente.
- Clases claras y precisas, con la documentación completa disponible.
- Al estar completamente orientado a objetos, permite la expansión y creación de nuevas clases que incrementen la funcionales de forma rápida, fácil y sencilla.
- Soporte para Direct3D 9 & 11, OpenGL, incluyendo ES, ES2, ES3 y OGL3+, y WebGL.
- Soportado en las principales versiones de Windows, Linux, Mac, OSX, Android, iOS, Whidows Phone y WinRT.
- Un sistema de materiales que permite a estos ser cargados independientemente del código.
- Soporta el acceso a todos los niveles, al permitir la creación e inclusión de código para el vertex shader.
- Soporte para DirectX 9 HLSL y OpenGL GLSL.
- Soporte para multitexturas, y texturas multicapa, operaciones con canal alfa, etc…
- Soporte para gran cantidad de técnicas de materiales, así OGRE seleccionará la mejor basada en el Hardware disponible.
- LOD para materiales.
- Carga de texturas en los formatos .png, .jpg, .tga, .bmp, .pvrtc y .dds aceptando todas las variedades de estos formatos.
- Soporte para texturas dinámicas, para poder poner vídeo o alguna escena renderizada como textura.
- Soporte para meses personalizadas, índices de vértices y optimización de mayas para un trabajo más rápido.
- Conversores de formato de los Softwares más populares, como 3D Studio Max, Maya, Blender, Wings3D y Milkshape3D.
- Soporte para animación basada en esqueleto y ragdoll.
- Soporte para Superficie curvas basadas en Biquadratic Bezier Patches.
- Soporte para LOD en mayas.
- Sistema de creación de primitivas parecido al de OpenGL.
- Sistema de manejo de escenas personalizable y flexible. Posibilidad de creación de clases personalizadas, derivadas de las principales base para este cometido.
- Plugin de soporte BSP, que admite la incroporación de niveles Quake3 y un renderizado más rápido en escenas de interior.

- Plugin de gestión de superficies terrestres, que permite la creación de terreno a partir de datos de tipo “geo-mipmapped”.
- Gestión de la escena completamente jerarquica. Cada nodo pertenece a otro, permitiendo animaciones con gran facilidad.
- Sistemas de partículas, personalizables vía plugins. Gestión de partículas para evitar la congestión.
- Soporte para Skyboxes, skyplanes y skydomes.
- Sistema de interfaz visual, permitiendo interfaces con elementos 2D y 3D.
- Soporte para niebla.

Y más.

2.1.4.3. Conclusiones

Ogre, a diferencia de las demás alternativas, es la más barata. Al ser software libre, no hay que pagar nada. Cualquier tipo de uso y explotación está permitida. Esto, junto con el hecho de que la versión que se obtiene es la versión más completa, permite la creación de cualquier proyecto sin coste en este apartado.

Por el contrario, al ser solo un motor de renderizado, no tiene ningún sistema de interacción de los que suelen tener los demás motores. Especialmente la interacción física o la gestión de la entrada del usuario (los controles), se hace muy complicada, pues no hay nada diseñado para ello en Ogre3D, y debería realizarse a mano (programado en C++ o mediante alguna otra librería, enlazándola con Ogre3D, tarea no sencilla).

Sobre documentación, Ogre3D ha sido una versión incremental, por lo que salvo algún método obsoleto, solo se ha añadido funciones en cada actualización. Esto hace que la documentación sea muy amplia y que abarque casi todos los aspectos del motor.

Sin embargo, dado que debe usarse en código, el uso del motor puede ser complicado, ya que carece de cualquier tipo de interfaz gráfica para el desarrollo, haciendo que la adición, prueba y el debug de las escenas sea bastante laborioso.

2.1.4.4. Ejemplos

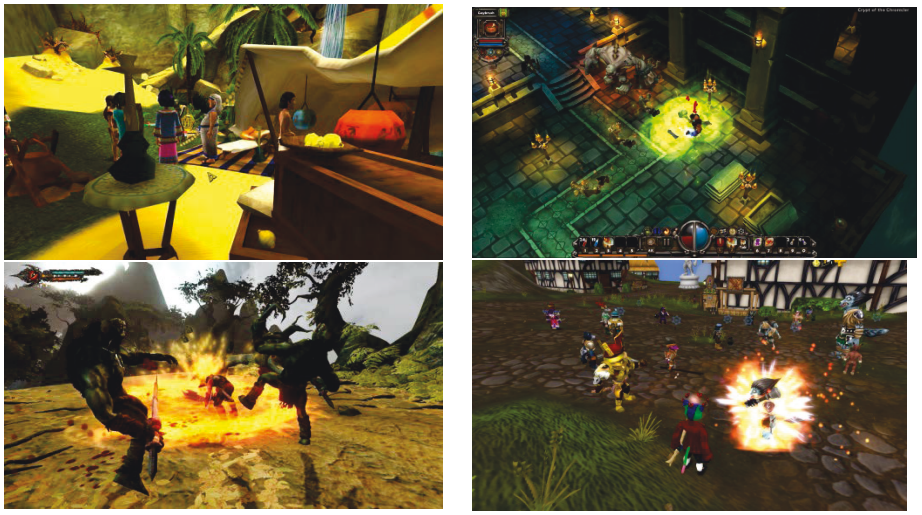

Ilustración 22. Algunos ejemplos de videojuegos de Ogre3D

2.1.5. Source Engine

Source es un motor de videojuegos multiplataforma creado y desarrollado por la empresa "Valve". Además del motor, se puede obtener una plataforma de desarrollo, para una elaboración más fácil y cómoda.

El motor, actualmente permite la creación de videojuegos para plataformas de ordenadores, excluyendo de momento las consolas y móviles. Las plataformas compatibles son: Microsoft Windows, Mac, Linux, SteamOs . No obstante, se han hecho algunos portaciones a Android, concretamente a Shield Portable.

El motor está programado en C++ y utiliza Direct3D, OpenGL, lo que garantiza un alto rendimiento a la hora de recrear los gráficos. No soporta ni DirectX 11 ni 12.

El tema de las licencias con el motor Source es relativamente complicado. Está fuertemente atado a la plataforma de distribución de videojuegos llamada Steam (también perteneciente a Valve). Por ese motivo, salvo alguna contada excepción, todos los juegos han sido distribuidos a través de dicha plataforma.

Ilustración 23. TitanFall. Uno de los pocos videojuegos que usan source no distribuido por Steam

Con estos datos, las licencias de distribución normalmente otorgadas, incluyen también la gestión de la distribución en dicha plataforma.

Normalmente, se requiere de la adquisición del estatus de publicador en Steam, que cuesta 90 €. Una vez realizado el pago, los detalles son discutidos personalmente con Valve, para llegar a un acuerdo personal. No obstante, si el juego hace uso de ciertas características que incluye el Source Engine, pero de las cuales Valve no es propietaria, hay que pagar un extra.

En estos casos se encuentran el uso del motor físico (Havok), que si se usa para un juego no gratuito, requiere un pago de 25.000 $. También están las herramientas RAD, cuyo precio de licencia también se gestiona individualmente con MILES y/o BINK.

El motor ha recibido muchas críticas por no haber recibido modificaciones sustanciales en un tiempo, tener las mismas interfaces en sus editores desde 1.998 y otras críticas similares. No obstante, se ha confirmado el desarrollo de la versión 2 del motor.

Esta segunda versión, ha salido al mercado por primera vez en el videojuego Reborn, una beta de Dota 2, perteneciente a Valve, a fecha de 17 de Junio de 2.015. De momento no se publicado ni se han proporcionado las herramientas de este motor, pero se ha confirmado que tendrá una licencia gratuita y que el pago se hará sobre los beneficios, un método muy similar al que usa Unreal Engine 4. El porcentaje que requiere ser pagado todavía no hay información.

2.1.5.1. Historia

Valve era propietaria del motor gráfico GoldSrc, motor con el que se creó Half-Life 1 y todos los mods relativos a él (por ejemplo, Counter Strike). Este motor se basaba en el motor Quake.

Para el lanzamiento de Half-Life 2, Valve estuvo trabajando en mejoras de su motor, pero los cambios fueron tantos, que finalmente acabaron por eliminar casi todo el código que había en el primer motor. No obstante, todavía quedan trazas de ese primer motor.

Para la mejora del motor, Valve no desarrolló todo lo que iba a usar, sino que también utilizó herramientas de terceros, como por ejemplo, Havok para el motor interno de físiscas. Miles Sound System fue usado para el sonido y Bink Video para el vídeo.

Con el nuevo motor vio la luz junto con el juego Half-Life 2 el 16 de noviembre de 2.004, después de ciertos retrasos por una gran fuga de información que sufrió la compañía, y que decidió cambiar casi todo el contenido filtrado.

Ilustración 24. Half-Life 2. Juego con el que salió el Source Engine

A su lanzamiento, le siguió rápidamente lanzamientos de los videojuegos derivados de Half-Life 1, pero actualizados al nuevo motor y con los gráficos mejorados. Entre estos se encontraban el *Counter Strike: Source* y el *Day of Defeat: Source*. Todos estos juegos estaban disponibles únicamente para la Plataforma Microsoft Windows.

Dos años después, coincidiendo con la salida de *Half-Life 2: Episode One*, los desarrolladores de Valve actualizaron el motor del videojuego, incluyendo una serie de mejoras. Entre estas se encontraban por ejemplo, la introudcción de HDR, corrección de color y un nuevo tipo de shanding basado en Phong.

Un año después, ya en el 2.007, el motor sufrió otra actualización coincidiendo con la salida de *Portal*, otro juego de la compañía Valve. Entre las mejoras, destacaban un nuevo sistema de gestión de partículas, que reemplazaba el anterior, en el que las animaciones debían ser escritas en código base y compiladas.

Otra mejora fue la optimización del motor para hacer uso de multiples núcleos o procesadores. Hay que tener en cuenta, que estas mejoras también llegaban a los juegos ya lanzados, pues al estar distribuidos por Steam, las actualizaciones eran automáticas.

Ese mismo año, se hizo otra mejora, por la cual se podía portar un juego a Xbox 360 recompilando el juego con las características adecuadas. En los siguientes dos años, el motor se fue modificando para poder soportar las características necesarias en una consola, tales como dividir la pantalla para múltiples jugadores, o una nueva interfaz en los menús para facilitar el control en las consolas.

Ya en el año 2.010, y con el objetivo de promocionar Steam en todas las plataformas, Valve empezó a desarrollar el soporte para OS X y Linux. El proceso fue gradual, y se fue implantando en todos los juegos ya salidos en un periodo de dos años. Este proceso culminó con la salida de Portal 2.

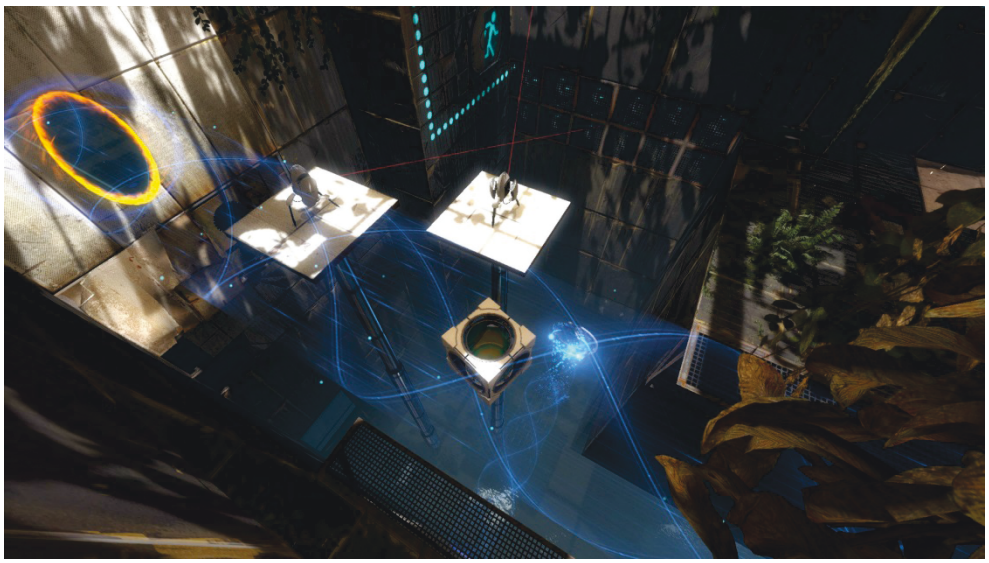

Ilustración 25. Portal 2.
Fue el primer juego Source en ser lanzado en Microsoft, Linux y OS X simultaneamente.

En los últimos años, las mejoras no han sido muchas. Algunas de las más comentadas han sido el soporte para Oculus Rift o la publicación del código de casi todos los juegos un jugador que Valve ha desarrollado hasta la fecha, para ayudar a los desarrolladores.

2.1.5.2. Características

- **Advanced Shader Technology:** Soporte para HLSL shaders, usando Shadel model 3.0. Librería Avanzada de Shaders. LOD en modelos y terrenos.
- **Dyanamic Lighting and Shadows:** Radiosidad. HDR, Transferencia de Radiación. Sombras dinámicas. Renderizado avanzado de materiales.
- **Effects:** Todo tipo de efectos lumínicos y ambientales, como lluvia, niebla, destellos, etc. Efectos con partículas. Editor de partículas. Desenfoque de movimiento. Superficie de agua realista.

- **Materials:** Sistema de Materiales avanzado. Sistema de Relieve avanzado. Texturas de resolución variable, para obtener detalle donde es necesario. Correción del color dinámica.
- **Advances Character Meshes:** Musculatura simulada, que permite animaciones de alta calidad. Ojos basados en esferas. Animaciones vocales dependientes del idioma. Texturas de piel de alta calidad. Plugins para los softwares más conocidos, como Maya, 3DS Max, Gmax, SOFTIMAGE, XSI, Blender, Lightwave Maxon, Cinema 4D, Milkshape 3D y FragMotion. Compiladores de modelos de alta calidad para un renidmiento máximo. Visualizador de modelos.
- **Advances Animation Tools:** Animaciones basadas en esqueletos. Sistema de animación facial. Sistema de creación de animaciones avanzadas. Faceposer, programa para crear animaciones fáciles rápidamente.
- **Environments:** Superficies de terreno con desplazamiento y transición de canales de textura. Objetos dinámicos y estáticos, para acelerar el juego y poder tener más carga y polígonos. Skyboxes. Previsualización de la luz.
- **Physics:** Física para contrucción de maquinaria. Vehículos con física avanzada. Objetos deformables. Cuerdas y cables deformables y rompibles. Cinemática inversa para asegurar la correcta posición de los objetos y/o modelos durante las animaciones.
- **Game Mechanics:** Sistema de navegación y búsqueda de rutas avanzado. Sensores de la IA que emulan a los humanos: Visión, oído e incluso olfato. Relación entre NPC avanzada. Batallas de la IA avanzadas, haciendo formaciones, y comunicándose sus conocimientos. Sistema de I/O que permite configurar la IA sin tocar código.
- **Programación:** Desarrollado en C++. Multiplataforma. Multi-núcleo. Código reutilizable. Control total sobre el motor. Eficiente. Gestor de perfiles para comprobar el rendimiento.
- **Audio:** Sistema multihilo. Manejo óptimo de la memoria precargando o indizando los sonidos. Soporte para sonido envolvente. Creación de ambientes 3D. DSP. Soporte de formatos comprimidos. Sonidos personalizados para cada material.
- **Networking:** Predicción y análisis del lag por parte del servidor. Buscador de servidores. Mensajería instantánea con amigos.
- **Console Support:** Kits de conversión para convertir de PC a Xbox 360. Métodos y funciones específicas de Xbox 360 para agilizar la carga. Interconectividad entre plataformas: Juega entre PC y Xbox 360. Código integrado de Xbox Live.

2.1.5.3. Conclusiones

Source engine es un motor ligeramente anticuado que fue sacado, sobre todo, con la intención de que los jugadores pudiesen crear contenido para los videojuegos ya creados, más que desarrollar nuevos.

Por este motivo, la creación de un videojuego es muy complicada y enrevesada en este motor gráfico. En cambio, la creación de contenido es muy fácil, sencilla e intuitiva.

Existe mucha documentación respecto al motor, y especialmente sobre la importación de contenido, aunque también hay bastante sobre creación de código y modos de juego.

Un gran problema es que la licencia está prácticamente atada a la plataforma de distribución Steam, por lo que hacer proyectos independientes de ella es imposible si no se adquiere la licencia completa (precio negociado en cada caso).

2.1.5.4. Ejemplos

Ilustración 26. Algunos ejemplos de videojuegos de Source Engine

2.2. Modelos navales

La predicción de trayectorias navales el un problema que no tiene una solución exacta. Si bien las fuerzas implicadas en el movimiento de un barco son muy conocidas, el hecho de los tamaños, masas y cantidad de fuerzas y pequeñas anomalías, hacen que sea un problema imposible de computar. Por este motivo, se utilizan aproximaciones. El número de aproximaciones es muy elevado, en esta sección se van a ver las princpiales

2.2.1. Simulación de partículas

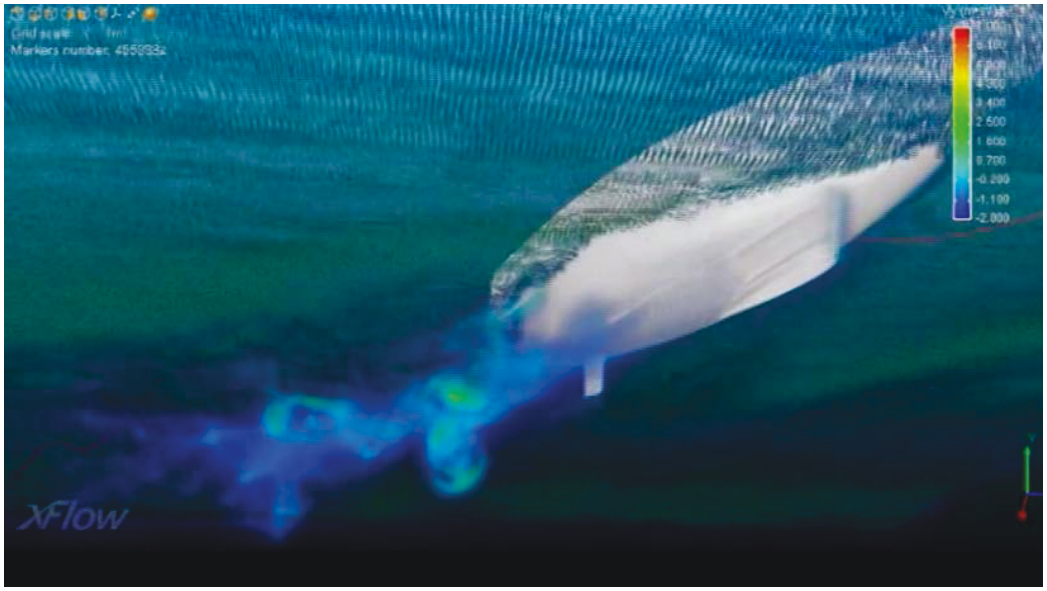

Ilustración 27. Simulación del movimiento de un barco y de las partículas colindantes

Esta es la aproximación más realista, y que menos misterios tiene. Esta técnica requiere software que simule túneles de viento, o en este caso, túneles de "agua". Se recrea el barco físicamente con un modelo 3D muy preciso. En algunos casos también se subdivide el barco en secciones, ya que la zona donde se sitúan las máquinas pesa más que aquellas que no tienen maquinaria pesada.

Una vez modelado el barco, se ajustan las partículas. Las partículas, deben estar configuradas para reaccionar lo más parecido al agua. Si es necesario, se les puede dar una fuerza incial para simular corrientes.

Para las olas, hay dos métodos de creación. El primero es un algoritmo propio para producirlas. Este método es el más usado, pues aunque el coste computacional es alto, es mucho menos que el otro método.

El segundo método, es hacer un segundo sistema de partículas en interacción con las partículas del agua. Estas segundas partículas serían el aire, y se configurarían como viento. Para este segundo método conviene configurar muy bien las temperaturas, presiones y el fondo del mar, para que las ondas se reflejen y

distribuyan correctamente. Este método solo se ha utilizado experimentalmente, y los resultados han sido inciertos.

Una vez que el mar tiene su fuerza y el movimiento deseado, no queda más que configurar las fuerzas y superficies de control del barco. Normalmente, se suele configurar las revoluciones de las hélices, pero también se puede configurar un sistema simulado de energía para las hélices.

Este sistema, permite el uso de hélices de cualquier tipo, ya sean convencionales, hélices de proa o pods.

Ilustración 28. Los tres tipos actuales de propulsión marítima

Este sistema también se puede utilizar para veleros, en el que la simulación de la vela suele ser calculado primero aisladamente, y posteriormente se extrae la fuerza que recoge la vela, y se aplica al sistema marítimo.

Este tipo de sistemas suelen ser computacionalmente muy costosos, por lo que su implementación en tiempo real es completamente inviable, o al menos por el momento.

Su uso principal suele ser para la ayuda al diseño de los cascos de los barcos, hélices, etc…

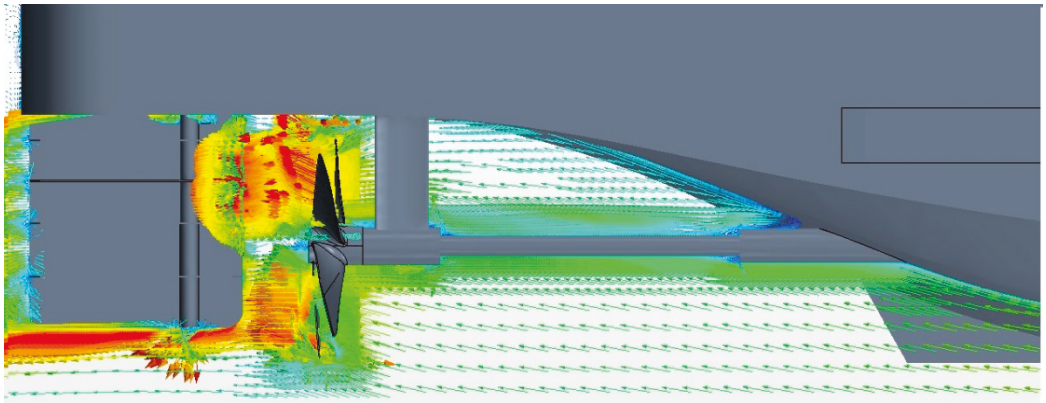

Ilustración 29. Visualización de las fuerzas que recibe el timón y que son generadas por la hélice

2.2.2. Modelo de Shyh-Kuang Ueng

El modelo desarrollado por *Gatis Barauskis* y *Peter Friis-Hansen*, y mejorado por *Ueng*, presenta unas características que lo hacen más idóneo para ser tratado en tiempo real.

Se basa en una premisa, llamada Seis Grados de Libertad. Con esto se consigue indicar los 6 tipos de movimientos que puede realizar un barco (3 movimientos son de desplazamiento y otros 3 de rotación).

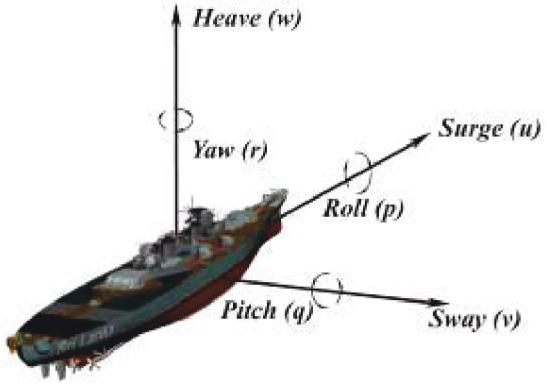

Ilustración 30. Los seis Grados de Libertad con sus nombres en inglés

Estos modelos están basados un conjunto de dos conjuntos de ecuaciones no lineales. Cada conjunto se encarga de 3 de los posibles movimientos.

El primer conjunto, está formado por las ecuaciones que controlan el Cabeceo (*Pitch*), la altura (*Heave*), y Alabeo (*Roll*). Estos movimientos son causados por el oleaje, y por ese motivo, en este documento no se explicará mucho sobre ellos.

El segundo conjunto está formado por las ecuaciones que controlan el desplazamiento frontal (*Surge*), el desplazamiento lateral (o derrape) (*Sway*) y la rotación (*Yaw*).

Según lo obtenido, los movimientos del segundo conjunto se pueden indicar con las siguientes ecuaciones:

Grados de Libertad	Momentos de Fuerza	Velocidad Angular/Lineal	Posición en Ángulos de Euler
Surge	X	u	X
Sway	Y	v	Y
Heave	Z	w	Z
Yaw	N	r	Ψ
Pitch	M	q	Θ
Roll	K	p	Φ

- Movimiento Frontal (*Surge*):

  El movimiento frontal viene dado por la siguiente ecuación:

  $$M\dot{u} = T_e - X_{u|u|}u|u| - (M + X_{vr})vr$$

  Donde M representa la masa del barco; T_e representa la fuerza de propulsión efectiva que es transmitida desde las hélices; $X_{u|u|}u|u|$representa la resistencia cuadrática al avance a la velocidad u; $(M + X_{vr})vr$ representa la pérdida de fuerza en los movimientos de Yaw y Sway combinados.

  Se pueden deducir ciertas relaciones:

  $(M + X_{vr})vr = 0$ si $u =$ cte. y $T_e - X_{u|u|}u|u| = 0$

  En estas condiciones, si se conoce T_e y u se puede calcular $X_{u|u|}$.

  La fuerza proporcionada por las hélices está relacionada con las revoluciones que hacen n, el diámetro D y el coeficiente de eficacia K_t, y viene dado por las fórmulas:

  $$T_e = K_t n^2 D^4 \text{ y } X_{vr} = 0{,}33\, M$$

- El movimiento de Rotación (*Yaw*):

  El movimiento de rotación viene dado por la siguiente ecuación:

  $$\tau\dot{r} + r = K\delta$$

  Donde δ es el ángulo del timón; K y τ son constantes que deben ser halladas mediante pruebas (imposible en este caso). Este modelo es efectivo para timones de hasta 35º de giro,

- El movimiento Lateral (*Sway*):

  El movimiento lateral viene dado por la siguiente fórmula:

  $$\tau_v\dot{v} + v = K_v(\tau\dot{r} + r)$$

  Donde K_v y τ_v son constantes. En este caso pueden asumirse los siguientes valores:

  $$K_v = 0{,}5 \quad \text{y} \quad \tau_v = 0{,}3\tau$$

De este modo se llega a la conclusión de que para poder calcular estas ecuaciones, se necesitan las siguientes constantes, independientes para cada tipo de barco:

M, D, K_t, K, τ, K_v

Y los siguientes parámetros, que dependen de la situación inmediata:

u, n, δ

No obstante, quedan dos problemas. El primero es la resolución de la fuerza de resistencia al avanza ($X_{u|u|}u|u|$), que no hay otra forma de obtenerla más que de datos experimentales. Dado que no se posee ningún barco, se ha tomado la aproximación de una resistencia cuadrática, por lo que dicha fuerza será la velocidad elevado a 4 (más un parámetro de ajuste).

El segundo es X_{vr}, que representa la fuerza perdida en la rotación y en el desplazamiento lateral (derrape). Para este problema también se ha tomado una aproximación, ya que este se debe obtener de datos reales del barco. La simplificación ha sido tomar la velocidad en el punto anterior y descomponerla en las nuevas direcciones. Entonces se toma la componente que tenía hacia el exterior.

Con estas aproximaciones, se tratará de crear el simulador para el puerto.

2.3.Simulación naval

La simulación naval en los ordenadores en tiempo real existe desde hace mucho tiempo. El grado de realismo ha ido cambiando a lo largo de los años, al igual que su representación.

En la mayoría de los casos, han sido siempre juegos de PC considerados simuladores. Uno de los primeros simuladores fue *GATO: WWII GATO-class Submarine Simulator*, en el que se tomaba el rol de un submarino estadounidense durante la segunda guerra mundial en el Océano Pacífico.

De gráficos vectoriales, la simulación del movimiento de los barcos era mínima.

Durante los siguientes años fueron apareciendo más juegos similares, tales como *Silent Service*, *PHM Pegasus*, *688 Attack Sub*, *Advances Destroyer Simulator (ADS)*, *Wolf Pack* o *Jutland*, entre otros.

Estos juegos eran casi más de estratégia que simuladores.

Pero en 1.994 llegó Aces of the Deep, que ya tenía un primer entorno 3D basado en polígonos, lo que le hizo dar un gran salto adelante.

Ilustración 31. Aces of the Deep. Primer simulador naval con graficos poligonales

Mención merece, que durante esta época, los simuladores de submarinos eran más populares y en general tenían una calidad mayor.

En 1.996 apareción un videojuego que, durante años, ha sido la saga más puntera en cuanto a simulación naval. No es otro que Silent Hunter. En este juego, no solo la trayectoria y velocidades se mejoraron, sino que la velocidad de inmersión y emersión, los ángulos profundidades y el sistema del sonar alcanzaron muy buenos niveles.

Ilustración 32. SIlent Hunter 1. Inció la época de simulación más realista

A medida que salía este juego, fueron saliendo más que mejoraban, sobre todo, el apartado visual. Por ejemplo, en 2.001 salió *Silent Hunter II*, y en 2.002 salió *Command Destroyer*, haciendo posible por primera vez que dos jugadores jugasen en ordenadores distintos, uno manejando un submarino y el otro uno o más barcos.

Ilustración 33. BattleStations: Pacific. Uno de los últimos simuladores navales

En cuanto a los juegos de superficie, a partir de 2005 empezaron a mezclar el simulador naval con la simulación aérea, requisito indispensable para los portaaviones. También en los simuladores de submarinos se colaron los aviones, aunque en menor medida.

Uno de los últimos simuladores en salir fue el Silent Hunter V, que tiene un gran apartado técnico y visual (aunque recibió críticas por ser en primera persona).

Ilustración 34. Silent Hunter V. Último videojuego de la saga Silent Hunter

Sin embargo, todos estos simuladores, sus sistemas de simulación son secretos, al haber sido desarrollados por empresas para sus propios videojuegos.

Por otro lado, los simuladores civiles han sido mucho más escasos. Motivo por el cual, solo merece destacar la saga Virtual Sailor, que ha sido la que más ha destacado en este campo.

Ilustración 35. Virtual Sailor es el referente en cuanto a simulación de barcos civiles

En resumen, se puede comprobar que la simulación naval en los ordenadores es algo que se ha usado en una cantidad muy alta de ocasiones. La cantidad de simuladores navales, tanto militares como civiles que están o han estado a la venta, supera en gran medida las cien unidades.

No obstante, aunque no se puede afirmar con rotundidad, puesto que sus motores de navegación y simulación no están publicados, la mayoría tiene unos sistemas simplificados de movimiento, no siendo tan exhaustivos ni llegando al sistema de los seis grados de libertad.

3. Análisis y Diseño

El análisis y diseño del proyecto han sido etapas consecutivas la hora de desarrollo. Es así en cualquier proyecto de Software.

La primera de estas dos fases, el **análisis**, es aquella etapa en la que los requisitos son definidos para ir recogiendo y almacenando la información relativa al problema que se desea resolver, y poder definir las funcionalidades del software que se desea desarrollar.

Por otro lado, el **diseño** es la parte donde se da forma a todos los requisitos que se han obtenido en la fase anterior.

En el apartado, se incluye la planificación usada durante la realización del proyecto, al igual que las metodologías, los requisitos, a partir de los cuales se ha desarrollado el proyecto y las características técnicas de las funciones utilizadas durante su implementación.

La última de las fases aquí expuestas es la **implementación**, en la que se describe el código generado durante la elaboración de este proyecto.

3.1. Planificación

Para la correcta realización de este proyecto, se establecieron una serie de episodios o etapas, en las que se debía ir realizando los distintos trabajos.

La primera etapa y la más importante, es la de búsqueda de la información. Esta a su vez está dividida en dos etapas.

La primera de estas sub-etapas es la búsqueda, elección y aprendizaje acerca de los diferentes motores gráficos posibles.

La segunda sub-etapa de la primera parte, consiste en buscar información sobre el movimiento de los barcos, y como estos reaccionan a las fuerzas marítimas. Esta es la parte más complicada, y la que requerirá más tiempo.

La segunda de las etapas generales, es la de búsqueda, mejora y adaptación, o creación de un modelo para usar de ejemplo. Esta fase, debido a la experiencia personal, no requerirá tanto tiempo como podría llegar a tomar.

La tercer y última parte, es la creación e integración en Unity de todo lo creado y desarrollado hasta el momento.

Una última parte es la toma de apuntes, y redacción de la memoria, que irá en paralelo a todo el proceso, aunque comenzará y terminará ligeramente después.

	Nombre de la Tarea	Duración	Comienzo	Fin
1	Estudio Motores Gráficos	31 días	01/03/2015	01/04/2015
2	Estudio Movimiento Marítimo	44 días	01/04/2015	15/05/2015
3	Modelado del barco	6 días	15/05/2015	21/05/2015
4	Integración y Codificación Unity	24 días	21/05/2015	14/06/2015
5	Documentación	92 días	21/03/2015	21/06/2015

Ilustración 36. Planificación

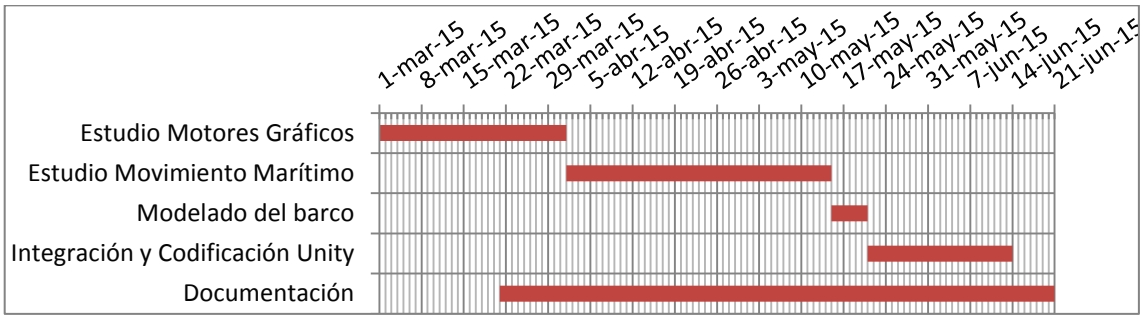

Ilustración 37. Diagrama de Gantt

3.2. Metodología de desarrollo

La metodología escogida para la realización de este proyecto ha sido la de modelo en cascada, ya que para el desarrollo de este proyecto se ha considerado la adecuada. Acto seguido, se explican un poco las principales características de esta metodología:

3.2.1. Desarrollo en cascada

El modelo en cascada es un tipo de metodología de desarrollo en el que el proyecto se divide en diferentes partes y/o secciones y cada una se realiza una única vez, y el hilo conductor es siempre hacia adelante. No puede haber retrocesos en el método.

En este método, se hace especial hincapié en el cumplimiento de plazos, y que cada fase pueda ser validada por sí misma, de tal modo que no requiera una revisión sobre una fase ya realizada.

Las fases suelen estar supeditadas al final de cada una con algún tipo de control, para comprobar que dicha fase está superada y finalizada. Dichas revisiones suelen implicar todo tipo de controles posibles. También se comprueba la documentación existente, para comprobar si es suficiente.

La documentación también es primordial, ya que algo hecho en una fase anterior, si es necesario para una siguiente, no se puede tocar, ni rehacer. Tampoco conviene perder tiempo revisando una parte que ya funciona, por eso en esta metodología se le da mucho peso a la documentación.

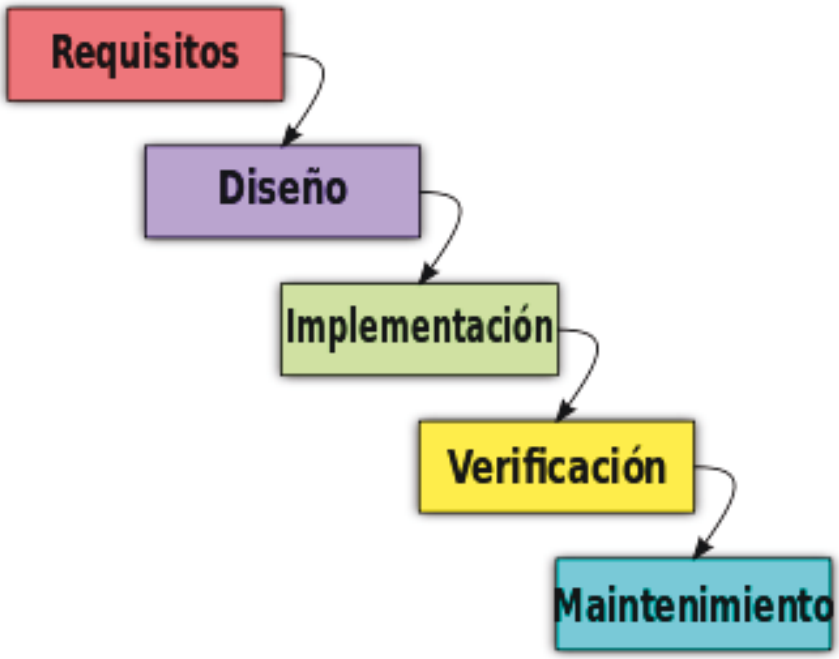

Ilustración 38. Ejemplo de modelo en cascada

El gráfico anterior muestra cómo funciona un proyecto de este tipo. Cada fase es independiente de la anterior, y suelen estar estructuradas en contenidos específicos, para evitar confusiones y crear un modelo consistente.

3.3. Requisitos

Para que la recogida de requisitos esté clara y bien documentada, se expondrán con las siguientes características.

Identificador	SR-F-00
Nombre	…
Descripción	…
Necesidad	…
Prioridad	…
Prerrequisito	…

Dónde las siguientes siglas significan:

- **Identificador:** Un identificador para que cada requisito no pueda ser confundido con ningún otro. La nomenclatura sigue la siguiente estructura:

  SR-Tipo-Número

  - **Tipo:** Indica el tipo de requisito que es. Puede tomar los siguientes valores:
    - **U:** Requisito de Usuario.
    - **F:** Requisito Funcional.
    - **NF:** Requisito no funcional.
    - **US:** Requisito de Usabilidad.
  - **Número:** Es un número de dos cifras que comienza por el 01 en base decimal, y que a cada requisito aumenta su valor en 1.
- **Nombre:** Expresa en un formato muy breve el requisito.
- **Descripción:** Una breve descripción del requisito.
- **Necesidad:** Este valor expresa el nivel de ncesisdad que tiene el requisito. Puede toma los siguiente valores:
  - **Básico:** Significa que el programa no puede funcionar sin él.
  - **Conveniente:** Significa que su aparición ayudará mucho en el funcionamiento de programa.
  - **Opcional:** Significa que su aparición es deseable, pero no es necesaria.
- **Prioridad:** Indica el la prioridad del requisito. Puede tomar los siguientes valores:
  - **Alto:** Requisito completamente prioritario.
  - **Bajo:** Requisito no prioritario.
- **Prerequisito:** Indica si es necesario haber completado algún otro requisito antes para poder realizar este.

3.3.1. Funcionales

Esta sección contendrá los requisitos funcionales necesarios para el sistema, con los servicios que debe otorgar.

Identificador	SR-F-01
Nombre	Creación de la escena
Descripción	El mundo debe crearse y tener todos los elementos necesarios para su visualización.
Necesidad	Básico
Prioridad	Alto
Prerrequisito	Ninguno

Identificador	SR-F-02
Nombre	Movimiento realista del barco
Descripción	El barco debe ser capaz de moverse adecuadamente y conforme a la física
Necesidad	Básico
Prioridad	Alto
Prerrequisito	SR-F-01

Identificador	SR-F-03
Nombre	Control de la aceleración
Descripción	El control de la aceleración del barco debe estar disponible para el usuario
Necesidad	Básico
Prioridad	Alto
Prerrequisito	SR-F-02

Identificador	SR-F-04
Nombre	Control del timón
Descripción	El control del timón del barco debe estar disponible para el usuario.
Necesidad	Básico
Prioridad	Alto
Prerrequisito	SR-F-02

Identificador	SR-F-05
Nombre	Visualización de la aceleración
Descripción	El usuario debe poder ver la potencia que tiene el barco en el momento concreto.
Necesidad	Básico
Prioridad	Alto
Prerrequisito	SR-F-03

Identificador	SR-F-06
Nombre	Visualización del timón
Descripción	El usuario debe poder ver en todo momento el estado del timón.
Necesidad	Básico
Prioridad	Alto
Prerrequisito	SR-F-04

Identificador	SR-F-07
Nombre	Timón a 0
Descripción	El usuario será capaz de poner el timón a 0º en cualquier momento.
Necesidad	Conveniente
Prioridad	Bajo
Prerrequisito	SR-F-04

Identificador	SR-F-08
Nombre	Parar máquinas
Descripción	El usuario será capaz de parar las máquinas del barco en cualquier momento.
Necesidad	Conveniente
Prioridad	Bajo
Prerrequisito	SR-F-03

Identificador	SR-F-09
Nombre	Visualización de las hélices
Descripción	El usuario podrá ver las animaciones de las hélices, con la potencia actual del barco
Necesidad	Conveniente
Prioridad	Bajo
Prerrequisito	SR-F-03

Identificador	SR-F-10
Nombre	Visualización del timón
Descripción	El usuario podrá ver las animaciones del timón, con el radio de giro actual del barco.
Necesidad	Conveniente
Prioridad	bajo
Prerrequisito	SR-F-04

Identificador	SR-F-11
Nombre	Visualización de la velocidad del barco
Descripción	El usuario podrá ver la velocidad instantánea del barco en Nudos.
Necesidad	Conveniente
Prioridad	Alto
Prerrequisito	SR-F-02

Identificador	SR-F-12
Nombre	Visualización del rumbo del barco
Descripción	El usuario podrá ver la dirección actual del barco en grados sexagesimales.
Necesidad	Conveniente
Prioridad	Alto
Prerrequisito	SR-F-02

Identificador	SR-F-13
Nombre	Creación de puntos de navegación
Descripción	El usuario podrá crear puntos de navegación e introducirlos en una lista
Necesidad	Básico
Prioridad	Alto
Prerrequisito	SR-F-02

Identificador	SR-F-14
Nombre	Carga desde archivo
Descripción	El usuario podrá cargar un archivo con una serie de puntos para ser introducidos en una lista.
Necesidad	Conveniente
Prioridad	Alto
Prerrequisito	SR-F-02

Identificador	SR-F-15
Nombre	Activación del piloto automático
Descripción	El usuario podrá activar el piloto automático para que recorra la lista de puntos automáticamente
Necesidad	Conveniente
Prioridad	Alto
Prerrequisito	SR-F-13, SR-F-14

3.3.2. Requisitos no funcionales

En esta sección se definen los requisitos no funcionales, es decir, las restricciones del sistema.

Identificador	SR-NF-16
Nombre	Compatibilidad
Descripción	La aplicación debe ser compatible con todas lasplataformas en las que Unity pueda ejecutarse.
Necesidad	Basico
Prioridad	Alto
Prerrequisito	Ninguno

Identificador	SR-NF-17
Nombre	Requisitos mínimos
Descripción	La aplicación debe poder ejecutarse con fluidez en los entornos desarrollados.
Necesidad	Basico
Prioridad	Alto
Prerrequisito	Ninguno

Identificador	SR-NF-18
Nombre	Accesos
Descripción	La aplicación debe proporcionar la mayor cantidad de accesos posibles a las funciones del barco.
Necesidad	Conveniente
Prioridad	Bajo
Prerrequisito	SR-F-03, SR-F-04

3.4. Diseño de clases

Para la realización de este proyecto, se ha considerado necesario la creación de 4 clases extras de Unity.

- **ExtendedFlycam:** Esta clase es la encargada del manejo de la cámara. Desde esta clase se realiza la gestión y se maneja la lógica de la cámara, para poder tener una cámara de fácil uso e independiente del resto de la escena.

- **Boat:** Esta es la clase principal, que gestiona el barco. Además de guardar toda la información relativa al mismo, es la clase encargada de actualizar su posición y calcular las físicas que afectan al barco, y cómo le afectan.

- **Route:** Esta clase es la encargada de Guardar y capturar los puntos para almacenarlos en la lista de coordenadas. También realiza la carga del archivo, y decodifica su contenido.

- **AutoPilot:** Esta clase es la encargada de dirigir la nave automáticamente si hay puntos. Se encarga de comprobar las distancias y direcciones, y da las ordenes apropiadas la clase Boat, para que esta lo ejecute.

Hay otros elementos de control, que son los que se encuentran en la interfaz. Los distintos botones tienen llamadas para las distintas funciones de las clases anteriormente comentadas. Entre estos se encuentran “Piloto Automático”, “Ver Ruta”, “Timón a 0º”, “Parar”, y Cargar Ruta”.

Además, están los dos sliders que envían su valor al actualizarse a lasmismas clases anteiormente descritas.

3.5. Implementación

Para la implementación del programa, lo primero era necesario tener el modelo de un barco, para poder ver el resultado.

Ilustración 39. Modelo de muestra importado y funcionando en Unity

Posteriormente, hubo que diseñar la cámara. La cámara es una cámara libre, que utiliza el botón W para ir hacia el frente, el S para retroceder, el A para desplazarse lateralmente hacia la izquierda, el D para desplazarse lateralmente hacia la derecha, la Q para ir hacia arriba y la E para ir hacia abajo.

Además, tiene tres velocidades distintas, las cuales se pueden usar presionando Crtl Izquierdo, Shift Izquierdo o sin pulsar ninguna tecla extra. Esto hay que hacerlo mientas se navega con las teclas ya descritas en el párrafo anterior.

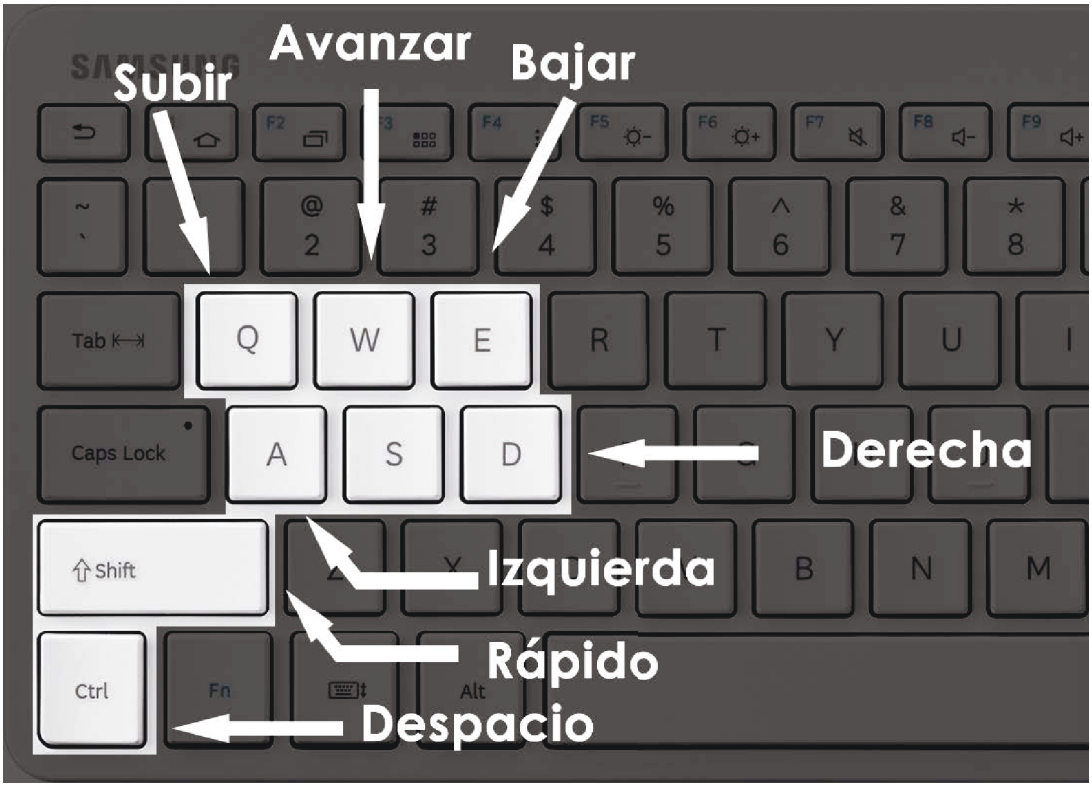

Ilustración 40. Controles de la cámara

Además, para rotar la cámara, es necesario pulsar el botón secundario de ratón.

Todo esto se consigue gracias al Método especial Update(), que se llama cada vez que un fotograma es terminado, por lo que entre cada fotograma, se comprobarán que teclas han sido pulsadas durante el transcurso del fotograma anterior, y se moverá la cámara con las correspondientes órdenes recividas.

Posteriormente, se realizó el trabajo sobre la clase Boat, teniendo esta la mayor carga de trabajo.

Esta clase también tiene un método Update(), por lo que al terminar el fotograma, se comprueban las teclas que han sido pulsadas, y se actualizan las ordenes, y se prepara para ejecutar la función updateMotion().

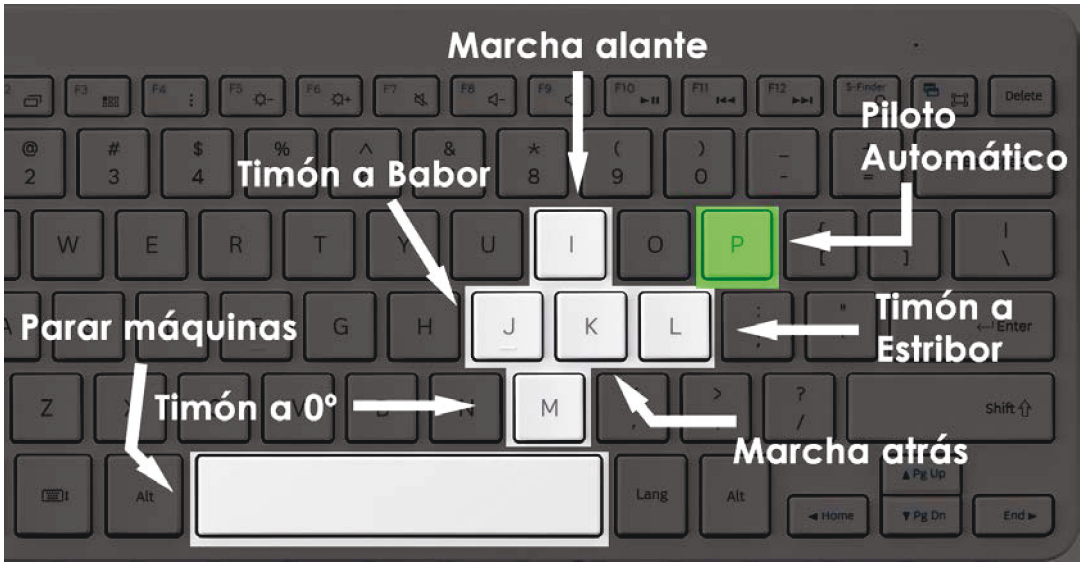

Ilustración 41. Controles del barco

La tecla P, es la única que no se desactiva si el piloto automático está en marcha.

Una vez recibidas las ordenes, el método updateMotion(). Con las ordenes recividas, y con el estado del barco, esta clase comienza a calcular el movimiento del mismo a lo largo del plano.

En esta función se calculan los tres movimientos que se decidieron hacer: Surge, Sway y Yaw.

Junto a este método, hay otro método igual de importante, que se llama updateInterface(). Este método actualiza todos los datos mostrados en la interfaz, y si es necesario, pone los sliders de potencia y timón en su posición justa.

Ilustración 42. Interfaz de la Pantalla

4. Conclusiones y Futuro

En este apartado se exponen las conclusiones a las que se ha llegado después de la elaboración del proyecto, mientras que se comentan las dificultades que se han encontrado y se comentan las posibles expansiones o añadidos que se puedan hacer.

4.1. Conlusiones generales

Tras la realización del proyecto, se ha conseguido un modelo básico de barco que se mueve muy correctamente por la superficie del agua. No obstante, este agua ha sido diseñado desde el principio como agua estática. Por ello, las conclusiones son las siguientes:

- Se ha creado una aplicación que recrea las trayectorias navales de un barco concreto en un entorno de Unity. El objetivo principal no especificaba que tipo de barco, o si solo debía ser u barco; pero en las dificultades encontradas se encuentra una explicación a este hecho. No obstante, hay un barco, que se mueve correctamente en el entorno de Unity. Por este motivo, se puede considerar que el principal objetivo está complido.
- Se ha estudiado a fondo Unity, viendo las capacidades que tiene este Software. Dado que en anteriores proyectos, el motor utilizado había sido Unreal Engine 3, el cambio a Unity, en un principio fue complicado, pero tras haber usado Unity, ha quedado patente su facilidad de uso, rapidez y comodidad a la hora de implementar. Unity queda grabado como una alternativa más que factible para ser usada en el futuro. El objetivo secundario también ha sido cumplido.
- Para la realización del modelo usado en el programa, si bien se ha partido de un modelo ya existente [Dirección: http://www.cadnav.com/3d-models/model-3940.html], la conversión, mejora y añadido de elementos vitales tales como las hélices o el timón, han sido satisfactorios. Para esto se ha usado el Software 3D Studio Max, que no es gratuito, pero se puede obtener una licencia de estudiante con todas las características (simplemente, limita su uso comercial).
- El movimiento del barco ha sido probablemente la parte más complicada de realizar de todo el proyecto. Especialmente su estudio. Ya que las ecuaciones resultantes, muchas no resultaban ser lineales, y otras no podían ser resueltas, llevó más tiempo del esperado la obtención de un sistema consistente para dar movimiento al barco. Este punto, el objetivo se califica como cumplido a medias.

4.2.Dificultades encontradas

A lo largo de la realización del proyecto, han surgido varios problemas y dificultades, que han entorpecido y retrasado el avance del proyecto considerablemente. Las dificultades más influyentes en el desarrollo han sido las siguientes.

- **Comprensión de las fórmulas:** La compresión de las fórmulas de navegación del barco ha sido sin duda alguna el problema más grande que ha tenido que superar este proyecto. En cierto modo, más que ser solucionadas, han sido salvadas, pues algunas ecuaciones daban problemas sin importar el punto de vista tomado.

  Las fuentes tomadas para la lectura y compresión de los movimientos del barco, estaban el inglés, lo que ha añadido dificultad a la hora de su compresión, pues si bien, el inglés hablado y cotidiano no suele representar un problema, matemáticas avanzadas con notación y términos ingleses es algo a lo que no se suele estar acostumbrado.

  Posteriormente, una vez estudiadas las fórmulas y llegado a un compromiso de elaboración, en el momento de implementar, surgían errores, que si inesperados, una vez estudiados, tenían su lógica.

  Entre estos, destacan la fórmula de giro (*Yaw*), en la cual no interviene para nada la velocidad actual del barco. Si bien, en la hora del estudio, descomposición y adaptación de las fórmulas a ecuaciones lineales, no se vio; cuando empezaron las primeras pruebas, resultaba que el barco giraba sobre sí mismo sin potencia ninguna.

  Este problema y otros, han llevado a la necesidad de combinar diferentes soluciones, algunas casi realizadas por experiencia náutica más que por modelos matemáticos precisos.

  En el ejemplo anterior, un timón necesita velocidad para poder Girar. Si se le quita velocidad, el timón pierde efectividad (como ocurrió en el caso del Titanic). Si bien, este detalle no ha podido ser implementado completamente, si se ha eliminado la posibilidad de que el barco gire sobre sí mismo sin ningún tipo de velocidad.

- **Realización de la interfaz:** Si bien Unity ha sido bastante más facil y sencillo de utilizar de lo que en un principio se esperaba, la parte de la interfaz, aparte de tener más dificultad que el resto de los componentes, requería un diseño avanzado que no era parte del estudio de este proyecto.

  Por ese motivo, la interfaz de control del barco no tiene el nivel de acabado que sería deseable, y que facilitaría mucho el control.

- **Elección del modelo:** Mientras que de barcos militares, la información existente es de lo más amplia y variada, cuando se trata de barcos civiles, encontrar la información detallada se vuelve mucho más complicado.

  El modelado de un barco con el nivel de detalle que presenta el modelo escogido, puede llevar más de un mes a tiempo completo, solo el modelado y el texturizado. Y aun así, para poder llevarlo a cabo, se requieren los planos (al menos simplificados) del barco, para poder diseñar correctamente las cuadernas y las curvas del casco.

  Dado que el modelado del barco no era el trabajo principal, se optó por buscar un modelo base, que aunque estuviese falto de detalles, las principales líneas fuesen correctas. Y así se hizo. De esta forma, la adaptación, detallado y añadido de piezas faltantes (como hélices y timón), no se demoró tanto y pudo hacerse en una semana.

  Sin embargo, con un modelo escogido, los datos tenían que adaptarse a dicho modelo. Y como ya se ha mencionado anteriormente, la falta de información acerca de los barcos civiles, ha hecho que losdatos usados sean una aproximación y no 100% reales.

4.3. Posibles procesos futuros

Como ya se ha comentado anteriormente, la cantidad de trabajo ampliable es bastante elevada. Este proyecto ha servido como lanzadera, y sobre todo con el principal objetivo de realizar movimientos correctos.

Una primera mejora, más que obvia, es una revisión de los movimientos, utilizando modelos más precisos, aunque se puede perder la capacidad de ser portado a plataformas de potencia más limitada.

Una segunda mejora, que estaría enlazada con la anterior, sería el añadir oleaje al mar. No solo las olas generadas por el barco al navegar, sino también olas, mareas y demás fuerzas que están presentes en el mar.

Una tercera mejora posible sería la realización de un filtro acuático para cuando la cámara se introduzca bajo agua. Este efecto ganaría mucho en cuanto a aspecto del programa. Dicho filtro debería consistir en un filtro de colores (tirando hacia el azul), un efecto de distorsión de lo visible y una pérdida de visión más elevada que en el aire a causa de la distancia.

Una cuarta mejora, que se complementaría muy bien con la anterior, es una textura de la superficie del agua más avanzada,con transparencias, reflexiones y refracciones. Esta mejora limitaría mucho el ámbito donde se podría ejecutar, pues ese tipo de texturas suelen consumir muchos recursos, y necesitan de una tarjeta gráfica.

Una quinta mejora sería el añadir sonido. Una aplicación gráfica, por muy brillante que sea, pierde mucho si el sonido es malo, por no hablar de si no existe. En este caso, los ruidos que se tendrían que tener en cuenta, son: giros de las hélices, oleaje, agua al chocar con el barco, motores del barco y viento. Estos serían los principales, aunque se podrían añadir otros como voces en las zonas de la tripulación, sonidos de gaviotas, etc. Si se realiza la tercera mejora, esta también debería aplicar el filtro, en este caso acústico, pues el sonido es muy diferente si se oye dentro del agua o fuera.

Una sexta mejora sería el asignar al barco regiones en las que la cámara no pueda acceder, de esta forma se evitaría que la cámara pueda atravesar el barco como si este fuese una proyección holográfica.

Una séptima mejora sería la de rediseñar toda la interfaz, haciendo un estudio en condiciones de las acciones necesarias para el controlador humano, y poder mostrar esas acciones de una forma amigable, sencilla e intuitiva.

La octava mejora podría tratarse de una gestión más avanzada de la ruta, con opciones como poder ver los puntos, con líneas interconectadas entre ellos, y el primero unido al barco.

La novena mejora podría ser de incluir una librería de barcos, con los modelos ya realizados y con la configuración correcta para su funcionamiento. Un menú para poder seleccionar el barco sería necesario junto con esta mejora.

La décima mejora debería tratarse de la capacidad de importar barcos de una forma sencilla y cómoda, del tal forma que los parámetros se puedan introducir fácilmente, al igual que el modelo, o incluso el desarrollo de un programa anexo que analizase el barco, realizando algunas preguntas al usuario para poder analizar el barco mediante técnicas de partículas y físicas que no fuesen en tiempo real, calcular sus características y poder importarlos posteriormente.

Si se implementasen todas estas mejoras, el aspecto final del programa no tendría nada que envidiar al de otros programas, tales como Virtual Sailor o Silent Hunter V.

Ilustración 43. Barcos pertenecientes al Silent Hunter V

5. Bibliografía

Información para la documentación:

- Información sobre el uso, historia y características de Unity (Junio 2015)

  https://unity3d.com/es/public-relations

  https://store.unity3d.com/es/

  https://unity3d.com/es/unity

  https://unity3d.com/es/unity/multiplatform

  http://docs.unity3d.com/es/current/Manual/UnityManualRestructured.html

  https://unity3d.com/es/unity/engine-features

- Información sobre el uso, historia y características de Unreal Engine (Junio 2015)

  https://www.unrealengine.com/unreal-engine-4%20

  https://www.unrealengine.com/what-is-unreal-engine-4

  https://www.unrealengine.com/blog/unreal-engine-48-released

- Información sobre el uso, historia y características de CryEngine (Junio 2015)

  http://www.crytek.com/cryengine/cryengine1/overview

  http://www.crytek.com/cryengine/cryengine2/overview

  http://www.crytek.com/cryengine/cryengine3/overview

- Información sobre el uso, historia y características de OGRE 3D (Junio 2015)

  http://www.ogre3d.org/

  http://www.ogre3d.org/tikiwiki/tiki-index.php?page=Brief+history+of+OGRE

  http://www.ogre3d.org/tikiwiki/tiki-index.php?page=Current+Ogre+Features

- Información sobre el uso, historia y características de Source Engine (Junio 2015)

  https://partner.steamgames.com/documentation/source_games

  https://developer.valvesoftware.com/wiki/Source_Engine_Features

  https://developer.valvesoftware.com/wiki/Source_SDK_2013

https://developer.valvesoftware.com/wiki/Open_Source_Tools

https://developer.valvesoftware.com/wiki/Category:Programming

- Información sobre la simulación de barcos usando partículas (Junio 2015)

  http://www.xflowcfd.com/industries/view/marine

  http://www.yourepeat.com/g/XFlowCFD/

- Información sobre aplicaciones de simulación naval a tiempo real (Junio 2015)

  http://www.listal.com/list/damn-torpedoes-armchair-admiral-naval

  http://www.listal.com/list/dive-dive-history-submarine-sims

- Información sobre el modelo de Shyh-Kuang Ueng (Junio 2015)

  http://jmst.ntou.edu.tw/marine/un_paper/JMST-2012-003-DOI.pdf

  http://icter.sljol.info/article/10.4038/icter.v3i2.2847/galley/3771/download/

  http://www.iau.dtu.dk/secretary/pdf/wavesII.pdf

  https://tubdok.tub.tuhh.de/bitstream/11420/916/1/Bericht_Nr.498_H.Bttcher_Simulation_of_Ship_Motions_in_a_Seaway.pdf

  http://www.marinecontrol.org/References/papers/MIC-2007-1-3.pdf

  http://ittc.sname.org/CD%202011/pdf%20Procedures%202011/7.5-02-06-03.pdf

  http://www.scs-europe.net/dlib/2014/ecms14papers/ese_ECMS2014_0074.pdf

  http://www.diva-portal.org/smash/get/diva2:618571/fulltext01.pdf

  http://eprints.soton.ac.uk/46182/1/138.pdf

  http://www.researchgate.net/publication/238677929_Mathematical_models_for_ship_path_prediction_in_manoeuvring_simulation_systems

  http://eprints.soton.ac.uk/46182/1/138.pdf

  http://www.iau.dtu.dk/secretary/pdf/TP-MB-shipmod.pdf

6. Anexos

6.1. ANEXO A: Utilización del Software

El software desarrollado se encuentra compilado en un ejecutable .exe. Para proceder a su ejecución, basta con hacer doble clic y abrir el programa. Cuando el programa se haya abierto, aparecerá un cuestionario preguntando sobre las opciones gráficas del programa y la configuración que se desea tener.

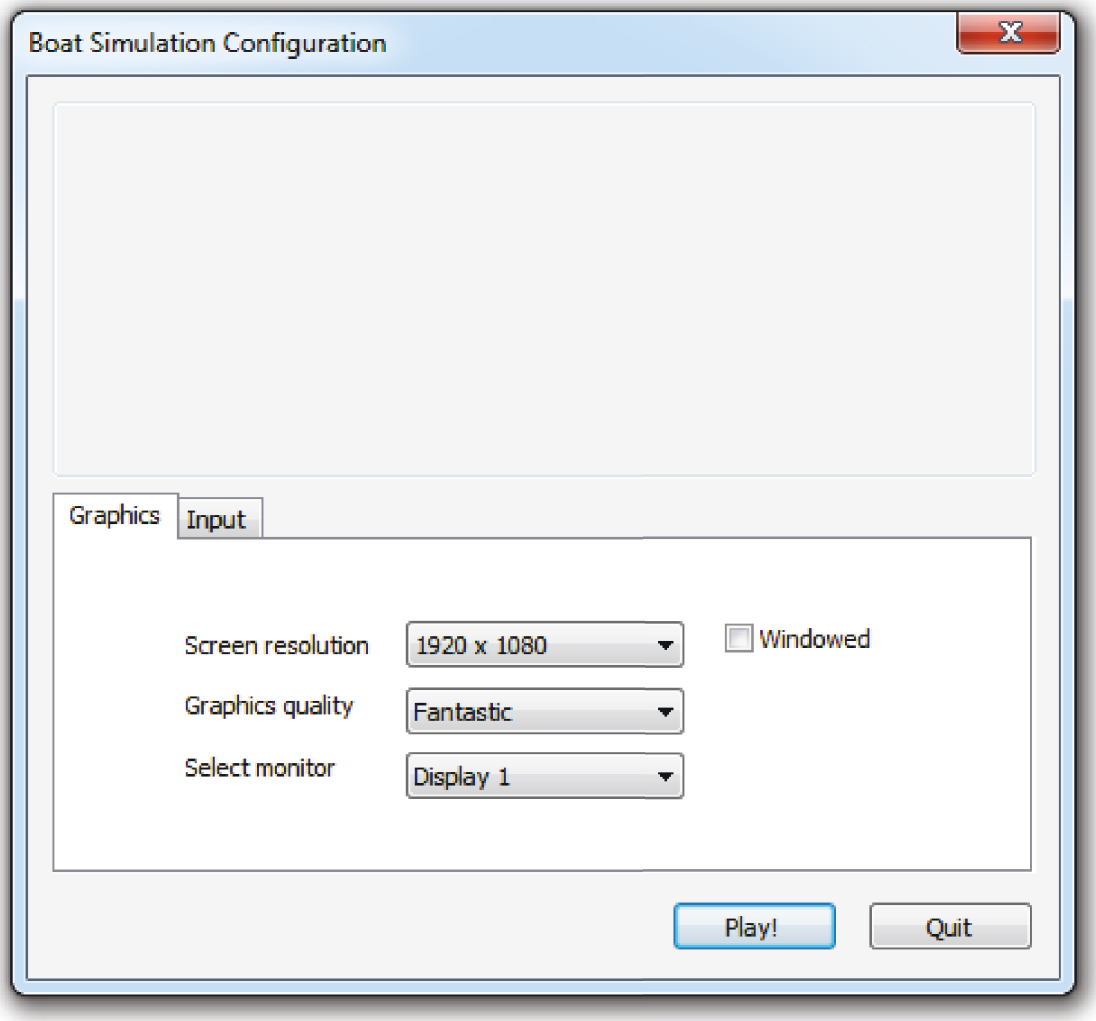

Ilustración 44. Configuración Gráfica del programa

Una vez terminada y escogida la configuración deseada, presionar en el botón Play! Para procedes a lanzar la aplicación. Tras una breve pantalla de carga con el logo de Unity, aparecerá el programa abierto y funcionando.

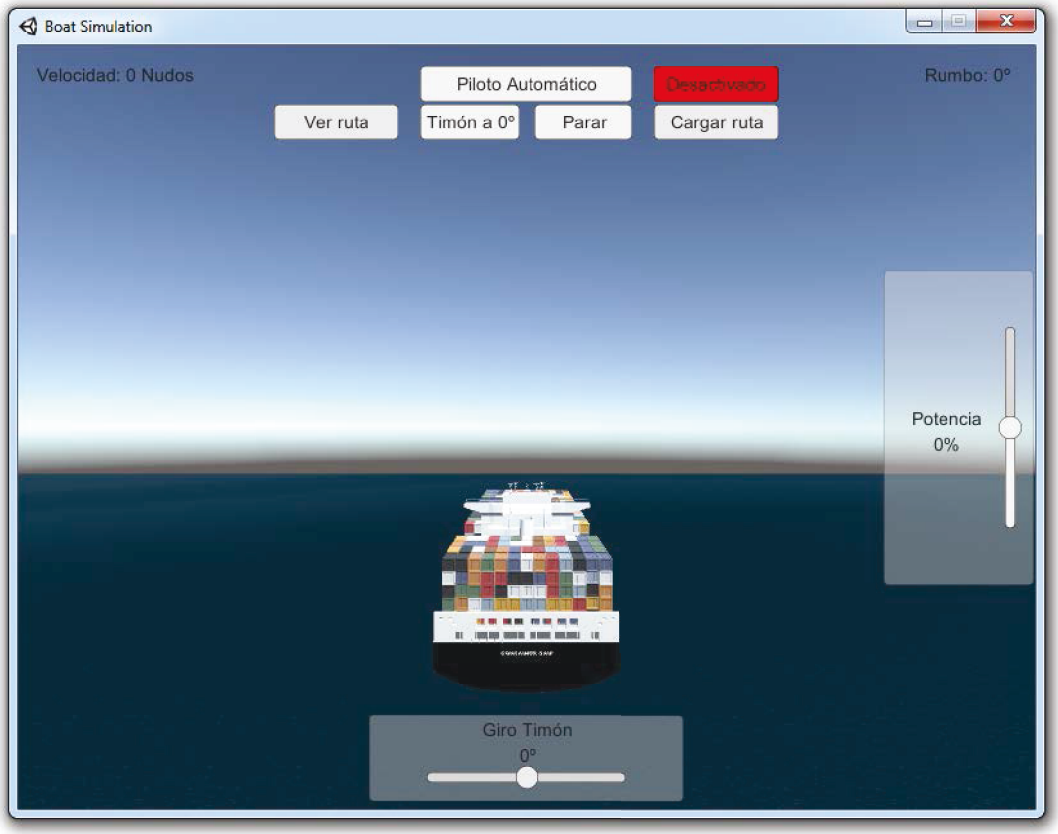

Ilustración 45. Ventana del Programa

En la ventana se pueden ver las principales opciones que tiene el programa.

- Muestra de velocidad.
- Muestra de Rumbo.
- Slider de potencia con indicador de potencia.
- Slider de timón con indicador de timón.
- Botón para alternar el piloto automático.
- Panel de indicación del piloto automático.
- Botón que muestra los puntos de la ruta.
- Botón para poner el timón a 0º
- Botón para parar las maquinas.
- Botón para cargar de un archivo de texto una serie de puntos que tomará como ruta.

En adición, el programa se puede manejar con botones. Estos están dividios en dos categorías:

- Botones de la cámara. Estos botones son WASDQE. Si se presionan estos botones, la cámara se mueve en los tres ejes. Para rotar la cámara, es necesario pulsar y mantener el botón secundario del ratón. Si se mantiene Ctrl pulsado, el movimiento es más lento. Si por el contrario se presiona Shift, el movimiento es más rápido.

- Botones de control del barco. Estos botones son I para más potencia y K para menos. J gira el timón hacia babor, mientras que L lo lleva a estribor. El botón P activa el piloto automático. El botón espacio para las máquinas y el botón M pone el timón a 0º. Todas estas acciones, excepto la de activar y desactivar el piloto automático se deshabilitan con el piloto automático en marcha.

6.2.ANEXO B: Presupuesto

1.- Autor:

Ángel García-Capelo Blanco

2.- Departamento

Informática

3.- Descripción del Proyecto:

- Título: Cálculo de trayectorias navales en Unity
- Duración (meses): 4
- Tasa de costes indirectos: 20%

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

€

5.- Desglose presupuestario (costes directos)

PERSONAL

Apellidos y nombre	N.I.F.	Categoría	Dedicación (hombres mes) 1)	Coste hombre mes	Coste	Frima de conformidad
Ángel García-Capelo Blanco		Ingeniero	4	2.794,39	0,00 0,00 11.177,56 0,00 0,00	
		Hombres mes	**4**	**Total**	11.177,56	

1) 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)

EQUIPOS					
Descripción	Coste	% Uso dedicado proyecto	Dedicación (meses)	Perioso de depreciación	Costa inmutable 2)
MSI GE70 2PE-684XES Intel i7-4720HQ/8GB/1TB/GTX860M/17.3" - Portátil	852,62	100	4	60	55,04
				Total	55,04

2) Fórmula de cálculo de la Amortización:

$$\frac{A}{B}CD$$

Donde :

A es: Nº de meses desde la fecha de facturación en que el equipo es utilizado.
B es: Periodo de Depreciación
C es: Coste del equipo sin IVA.
D es: Portentaje de uso dedicado al proyecto.

SUBCONTRATACIÓN DE TAREAS

Descripción	Empresa	Coste imputable
	Total	0,00

OTROS COSTES DIRECTOS DEL PROYECTO 3)

Descripción	Empresa	Coste imputable
	Total	0,00

3) 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	11.177,56
Amortización	55,04
Subcontratación de tareas	0,00
Costes de funcionamiento	0,00
Costes indirectos	0,00
Total	**11.232,60**

6.3.ANEXO C: Marco legal regulador:

Sobre este tema no existe ningún marco legal regulador ni ley que afecte al proyecto.

6.4.Glosario de Términos:

- **AGP:** Puerto de Gráficos Acelerados (del inglés *Accelerated Graphics Port*). Puerto de expansión desarrollado por Intel en 1.996 como método de interconexión entre el procesador y las gráficas, ya que el puerto anterior (PCI), se quedaba obsoleto.
- **DirectX:** Colección de API de Microsoft Windows para acceder a las funcionalidades de la gráfica, sin tener que usar el lenguaje estricto de la misma.
- **ICBM:** Misil balístico interncontinental (del Inglés *Inter-Continental Ballistic Missile*). Es un misil de largo alcanze (entre 5.000 y 12.000 km) con capacidad de ataque nuclear y de alcanzar sus objetivos rápidamente.
- **LOD:** Nivel de Detalle (del Inglés Lvel of Detail). Son simplificaciones de los modelos y/o texturas, para objetos que están muy lejos, o que las características de la plataforma no permite tener a más detalle.
- **MMORPG:** Juego RPG multijugador masivo. (del inglés *massively multiplares online RPG*). Es un videojuego parecido a los juegos de Rol, pero la diferencia suele radicar en que se suelen desarrollar en mundos permanentes, y que puede llegar a haber más de 1.000 jugadores conectados al mismo “mapa”.
- **OpenGL:** Colección de API multiplataforma que sirve para poder acceder a la funcionalidad avanzada de la gráfica sin necesidad de acceder a su modelo concreto.
- **Render:** Proceso de creación de una imagen digital a partir de datos.
- **RPG:** Videojuego de rol (del inglés *Role-Playing Game*). Es un juego en el queel personaje va ganando experiencia a través de características numéricas que se van mejorando a lo largo del juego, haciendo que en este tipo de juegos sea mejor contar con una buena estrategia que con buenas habilidades (del jugador).
- **SAGE:** Control Semi-Automático de Tierra (Del Inglés *Semi-Automatic Ground Enviroment*), fue un sistema que permitía ver en tiempo real la información obtenida por una serie de radares interconectados, y realizar sencillas simulaciones de su objetivo, tomando en cuenta su rumbo y velocidad entre otros factores.