

TECNOLOGÍA DE JUEGOS
MÁSTER EN INFORMÁTICA GRÁFICA,
JUEGOS Y REALIDAD VIRTUAL

Postmortem de

Tribes At War

Jaime González
Eder Miguel
Manuel Paleo
Carlos Pérez
David Rosillo

28 de enero de 2010

Índice

I.	Descripción del juego.....	4
II.	Requisitos del sistema	4
III.	Herramientas.....	5
A.	Entorno de desarrollo – Microsoft XNA Studio 3.1	5
B.	Motor de físicas JiglibX	5
C.	Adobe Photoshop CS3	5
D.	Microsoft Xna Framework Net.....	6
E.	GIMP 2.5 y superiores.....	6
F.	Corel VideoStudio 12	6
G.	Autodesk Maya 2008 y 2009.....	7
H.	Koyote Free Video Converter 2.5.....	7
I.	Autodesk 3ds Max 2008	7
J.	XSI Mod Tool	7
IV.	Tareas.....	8
A.	Jaime González.....	8
B.	Eder Miguel	9
C.	Manuel Paleo	10
D.	Carlos Pérez.....	10
E.	David Rosillo	11
F.	Tabla de esfuerzo	13
V.	Problemas y dificultades.....	15
A.	Gestión de eventos y colisiones.....	15
B.	Implementación flexible de las animaciones.....	16
C.	Problemas de colisiones con los elementos “box” de JigLibX	16
D.	Problemas de rendimiento con múltiples mallas de triángulos como mallas de colisión.....	16
E.	Fallos en la detección de colisiones.....	16
F.	Pequeñas bajadas de rendimiento en escenarios grandes.....	17
G.	Creación fondo de pantallas acorde a la tónica del juego	17

H.	Reproducción sonidos	17
I.	Intercambio de pantallas.....	17
J.	Creación función de minimapa	17
K.	Modelo topológico del multiplayer	18
L.	Falta de logs en la librería de red.....	18
M.	Problema de rendimiento por alto número de polígonos.....	18
VI.	Referencias.....	19

I. Descripción del juego

Nuestro videojuego se podría resumir como un juego tipo shooter en el que el jugador controla un tanque y debe completar un objetivo determinado según el escenario en el que se encuentre (sobrevivir el máximo tiempo posible, escapar a una zona segura sin ser destruido, eliminar a todos los enemigos del escenario, etc), todo ello debidamente ambientado.

El juego dispone de dos tanques distintos, uno utilizado por los jugadores y otro controlado por el enemigo, cada uno de ellos con características diferentes. De esta forma, por ejemplo, el tanque controlado por el usuario es más rápido pero algo menos resistente que los enemigos. También existen torretas controladas por el enemigo, lo que añade mayor diversión al juego ya que la estrategia a aplicar depende tanto del número de enemigos con los que tengamos que combatir, como su tipo. En el caso de combatir contra varias torretas y algún tanque enemigo puede resultar más sencillo buscar cobertura y eliminar al tanque sin tener que preocuparse de las torretas.

Existen dos escenarios diferentes, cada uno orientado a un modo de juego (jugador único y multijugador), aunque pueden jugarse en ambos modos de juego. Los escenarios del juego ofrecen total libertad al jugador, ya que puede desplazarse libremente por el mismo, sin "barreras invisibles" que le obliguen a seguir un determinado camino, si bien sí existen obstáculos que deberá rodear ya que no es posible desplazarse sobre ellos.

El juego además, ofrece varias cámaras distintas, desde cámara en tercera persona hasta cámaras en los enemigos y cámaras libres. Cada cámara resulta más útil que otras en determinadas situaciones como pueden ser a la hora de apuntar y disparar, donde el modo de primera persona es el más útil, o en el desplazamiento donde la cámara en tercera persona nos ayuda a orientarnos más fácilmente.

El juego ofrece una interfaz de usuario sencilla y con información suficiente para tener conocimiento de lo que sucede en el escenario, manteniendo en todo momento la atención del usuario en el combate y los enemigos del escenario. Además, se ha cuidado el diseño de los menús para permitir una navegación sencilla entre las diferentes pantallas del juego y se dispone de una pantalla de selección de dificultad, que permite volver jugar los escenarios con distintos niveles de dificultad desafiando la habilidad del jugador.

Gráficamente el juego consta de modelos y texturas con buen acabado y gran nivel de detalle, además de efectos de partículas para las explosiones, los restos de la destrucción de enemigos en forma de humo, etc.

II. Requisitos del sistema

Los requisitos mínimos del sistema para la ejecución del juego vienen dados por XNA, aunque para ejecutar el juego de forma fluida es necesario contar con un equipo bastante potente. El equipo en el que se han realizado las pruebas y las demostraciones es un Intel Core 2 Duo de 2.26 GHz, 3GB DDR2 y

una tarjeta gráfica nVidia GeForce 9600GT. También se han realizado pruebas satisfactorias con un equipo similar pero con tarjeta gráfica de la serie 8 de Nvidia.

III. Herramientas

A. Entorno de desarrollo – Microsoft XNA Studio 3.1

Toda la parte de codificación del videojuego se ha desarrollado sobre XNA y Visual Studio. XNA no es ni mucho menos un motor de videojuegos como Torque, Blender Game Engine o Unity, sino un simple entorno de desarrollo con algunas facilidades respecto al desarrollo “desde cero”, como pueden ser el acceso a elementos de entrada y salida, el renderizado de modelos o la carga de shaders.

La mayor ventaja que nos ha aportado el hecho de utilizar este entorno ha sido la libertad de la que hemos disfrutado a la hora de crear nuestra propia arquitectura, enfocándola al tipo de juego en desarrollo. Al contrario que con herramientas cerradas o medianamente prediseñadas como los motores mencionados anteriormente, XNA nos ha permitido establecer nuestras prioridades e ir desarrollando el juego de forma modular pudiendo ver los resultados en iteraciones relativamente cortas.

Finalmente, otra de las grandes ventajas de XNA es la gran cantidad de documentación disponible en la web. Desde videojuegos completos a partir de los cuales desarrollar modificaciones o añadidos, hasta tutoriales concretos y muy específicos con resultados bastante espectaculares.

B. Motor de físicas JiglibX

Debido a que se pretendía ofrecer la posibilidad de recorrer cada escenario libremente en terrenos rugosos y con numerosos obstáculos, con un comportamiento físico medianamente realista, se optó por utilizar un motor de físicas ya existente con lo cual ganamos bastante en tiempo de desarrollo aunque como ocurre en el caso de motores de videojuegos, perdimos en flexibilidad. Uno de los principales inconvenientes que presenta la utilización de herramientas externas dentro del propio código es la dificultad que plantea la modificación de código ajeno, que no conoces, y como en el caso de esta librería, la documentación es escasa y únicamente ayuda con usos muy sencillos de la librería sin explicar detalles necesarios para cualquier funcionalidad avanzada que se quiera implementar.

En cualquier caso, esta librería nos ha proporcionado las funcionalidades necesarias para gestionar colisiones entre tanques, el comportamiento físico en el movimiento y la comprobación de intersecciones entre rayos y objetos del escenario, muy utilizada en las comprobaciones de visibilidad de la inteligencia artificial de los enemigos.

C. Adobe Photoshop CS3

Herramienta muy conocida a nivel mundial debido a su enorme potencial destinado principalmente a la pintura, edición y retoque fotográfico. El uso de dicha herramienta en nuestro juego ha consistido en la creación tanto de los fondos del juego, como de diversos botones.

El fondo principal de las pantallas previas al juego, ha sido creado utilizando y combinando algunas de las numerosas herramientas de Photoshop. Como se menciona antes, en las diversas versiones del juego, se utilizó Photoshop para la creación de botones para cada una de las opciones de cada pantalla. En la versión final del juego, sólo se utiliza un modelo de dichos botones creados con Photoshop.

D. Microsoft Xna Framework Net

Es la librería que ofrece el XNA para el desarrollo de aplicaciones que utilicen la red, se estudio la posibilidad de utilizar otras famosas librerías como lideren-library-network. Finalmente se decidió la utilización de Microsoft Xna Framework Net por: disponer de más documentación, las posibilidades añadidas que ofrece la conexión a Xbox Live, exportación directa de pc a la consola...

La librería facilita tareas como: protocolos de comunicación, manejadores de eventos comunes, posibilidad de simular latencias y pérdidas de datos, etc.

Es tarea del programador crear una estructura de mensajes acorde a su aplicación y ubicar el lugar más propicio para procesar la información.

E. GIMP 2.5 y superiores

GIMP es un editor gráfico al estilo de Photoshop pero con licencia gratuita de uso. Tiene muchas de las funcionalidades que incluyen los programas de pago más conocidos y lo hemos usado ampliamente para edición de texturas, retoques puntuales en las imágenes del juego y para las que se muestran en el video de presentación. Es fácil de usar, potente, completo y gratuito, que siempre es de agradecer cuando se está aprendiendo.

F. Corel VideoStudio 12

Este es un programa de edición de video que permite componer una secuencia a base de imágenes fijas, clips de video y piezas de audio. Podemos editar las propiedades de estos elementos y posicionarlos a lo largo del *timeline*, de forma que vayamos montando la secuencia de video final. Una vez que terminamos de editar la base de la secuencia podemos añadir títulos, video en *overlay*, efectos de transición y un largo etcétera.

Con esta herramienta es con la que se ha compuesto el video demostración de *Tribes at War*. A pesar de las funcionalidades y facilidad de uso de este programa nos hemos encontrado con una desventaja importante a la hora de usarlo y es que, quizá debido a los códecs usados en los clips de video de origen, conforme se iban añadiendo clips de video a la biblioteca de Corel el programa se iba ralentizando más y más, haciendo incomodísima la edición. Además se observó que no estaba preparado para procesadores de doble núcleo con lo que no aprovechaba plenamente la capacidad del ordenador. Aún así era muy rápido compilando los videos y da una calidad de salida muy alta si no se utiliza ningún códec de compresión.

En nuestra opinión deberían buscarse alternativas a este programa para futuros proyectos.

G. Autodesk Maya 2008 y 2009

Maya es uno de los programas más conocidos para modelado y animación en 3D. Se suele relacionar más con proyectos de creación de películas pero también puede usarse para generar contenido para videojuegos. En nuestro caso lo hemos usado como herramienta de modelado y previsualización tanto de escenarios como de personajes y ha servido en varias ocasiones como primera etapa del *workflow* de modelado y texturización. En el sistema operativo XP 64 es frecuente que este software dé pequeños fallos en la actualización de los *viewports* pero aparte de eso no hemos tenido ningún problema. Los plug-ins internos del programa nos permitían exportar fácilmente el contenido generado en Maya a XSI Mod Tool.

H. Koyote Free Video Converter 2.5

Esta herramienta se ha utilizado para convertir el video promocional a distintos formatos y con distintos códecs para buscar una relación tamaño/calidad/compatibilidad adecuada, finalmente se consiguió un video en AVI de unos 2 minutos y 20 segundos de duración de 50 megas de peso con una calidad excelente y una versión en FLV de calidad más que aceptable y un tamaño de 13 megas. Este software puede usarse de manera gratuita con el 99% de sus funcionalidades habilitadas, quedando para la versión de pago solamente las opciones de personalización y los bitrates más altos. Es una herramienta fácil de usar, muy versátil y muy rápida en la conversión de formatos.

I. Autodesk 3ds Max 2008

Autodesk 3ds Max es una aplicación completa de modelado 3D, animación y renderización utilizada principalmente por los desarrolladores de videojuegos, aunque también en proyectos de cine de animación, publicidad en televisión, efectos especiales, etc.

3D Studio lleva implementado un sistema de simulación de luz natural muy realista que permite definir en qué lugar de la Tierra nos encontramos, en qué día y a qué hora e iluminar la escena en consecuencia. Además permite animar la posición del sol con suma facilidad, lo que nos sirvió para elaborar el paso de un día completo que puede verse en una de las escenas del video promocional.

J. XSI Mod Tool

Es una aplicación proporcionada XSI Softimage de forma libre para el modelado, texturizado y animación. Debido a los plugins de conexión al pipeline de contenido de XNA se eligió para ser el editor de niveles en el desarrollo del videojuego.

Un posible defecto de Mod Tool puede ser el que desde el punto de vista de alguno de los integrantes del grupo es un "agujero negro" ya que el contenido 3D que se genera en otros programas se puede exportar a Mod Tool con facilidad pero luego éste presenta muchos problemas para exportarlo a otros programas. Un ejemplo claro es que el plug-in de conversión a DAE-FBX (que podría usarse en Maya y 3D Studio) no conserva los mapas UV y, por tanto, deja los modelos sin texturizar en el programa de destino.

IV. Tareas

A. Jaime González

Durante el desarrollo del proyecto, mis tareas han estado enfocadas principalmente a la conexión entre los modelos y la programación del videojuego.

Inicialmente se comenzó creando un sencillo terreno con algunos obstáculos con una cámara para que los demás miembros del grupo que se dedicasen a la programación pudieran realizar pruebas desde el inicio. Posteriormente se decidió modificar el comportamiento de la cámara para que resultase más suave en sus movimientos.

A partir de ese momento era necesario poder generar niveles de forma que no hubiera que tocar nada en el código, por ello se propuso implementar un exportador de escenarios desde una aplicación de modelado. XSI Mod Tool dispone de una consola de scripting en la que se programó en lenguaje Python el exportador. Como en XNA se permite leer de forma automática ficheros escritos en XML, el fichero de salida desde XSI Mod Tool se genera se realizó en este formato. Previamente se tienen que indicar de forma ordenada todos los campos que tiene que parsear con sus respectivos tipos. Para ello se crearon diferentes clases que representan las distintas estructuras de datos necesarias para representar los objetos del nivel, que serán leídos desde la aplicación.

Cuando se fue avanzando en la creación del videojuego y el exportador ya realizaba las tareas que hasta el momento fueron necesarias, mis tareas se empezaron a desviar al modelado y texturizado. Debido a mis conocimientos más avanzados en 3ds Max, resultaba bastante más eficiente texturizar los objetos en este programa y luego exportarlos a ModTool para su posterior incorporación a los escenarios. Además, comencé a ver la forma de animar los modelos y a mostrar a los compañeros que programaban la forma de hacerlo para que ellos lo incorporaran de la forma necesaria.

Durante las fases finales del proyecto se vio que era necesario reducir la carga que recibía el motor de físicas. Por ello se realizó un modelo de baja resolución del escenario solamente para el tratamiento de colisiones. Con esto se produjeron elevadas mejoras de rendimiento pudiendo incorporar nuevas características al videojuego.

Otra de las tareas fue añadir una iluminación acorde con la temática del videojuego, disminuir la luz ambiental, lo que conllevaba también realizar texturas ligeramente más oscuras.

Una vez que el primer escenario era totalmente funcional, se decidió realizar un segundo nivel, en el que los objetivos fueran destruir todos los enemigos. Para ello se modeló y texturizó un nivel cerrado como el del Coliseo Romano pero con una temática más futurista.

B. Eder Miguel

En general, me he dedicado a programar diferentes funcionalidades dentro del videojuego. En primer lugar, la arquitectura general del juego, donde posteriormente se han ido acoplando diferentes módulos para obtener los comportamientos deseados. Algunas de los componentes más relevantes dentro de esta arquitectura son los gestores de entrada/salida, de cámaras y de objetos del escenario. El gestor de entrada/salida simplemente permitía detectar el estado del teclado, etc, con algunas funciones auxiliares para facilitar tareas comunes dentro de la lógica del juego. El gestor de cámaras es el encargado de permitir el cambio de cámaras, dar acceso a los datos de cada una de las cámaras, principalmente a las matrices de proyección y de cámara. El gestor de objetos del escenario es el encargado de gestionar las distintas listas implementadas (objetos comunes, jugador local, jugador remoto y enemigos) de forma que cualquier otro objeto del escenario tenga acceso a datos relevantes del resto de objetos. Esto ha resultado muy útil por ejemplo en la inteligencia artificial, que requiere que los enemigos conozcan la posición de los jugadores para determinados cambios de estado.

Dentro de la arquitectura general, también destaca la implementación de la jerarquía de objetos del mundo, como tanques, torretas, terrenos y clases derivadas, como tanques enemigos, remotos y locales, cada uno con sus propiedades.

En relación con los puntos que se expondrán a continuación y como enlace desde la arquitectura general, también se han implementado pequeños módulos que permiten la introducción de animaciones e inteligencia artificial en cada objeto. Estos módulos ofrecen una interfaz uniforme, de forma que distintas instancias de una misma clase pueden utilizar distintas animaciones sin tener que programar distintas clases para los primeros. En los siguientes párrafos se explica esto más detalladamente.

Otra de las tareas que he desempeñado es la programación de las animaciones para los tanques. Tomando como base el código que desarrolló Jaime previamente para la animación de las torretas, implementé dos tipos distintos de animaciones para los tanques, uno para el tanque del jugador y otro para el tanque enemigo. Como se ha comentado en el apartado anterior, la arquitectura es bastante flexible ya que la misma animación implementada para el tanque enemigo es válida para cualquier otro tipo de tanque siempre que la estructura jerárquica del modelo sea igual. Por ejemplo, cualquier tanque con cuatro ruedas y dos ejes, podría utilizar la animación del tanque enemigo.

La inteligencia artificial ha sido implementada de forma parecida, con un controlador genérico que especifica la interfaz de los controladores especializados. Se han implementado dos controladores, uno para las torretas enemigas y otro para los tanques, si bien ninguno de los dos utiliza técnicas de inteligencia artificial especialmente avanzadas.

La parte del desarrollo que más tiempo me ha llevado ha sido la integración del motor de físicas. El principal motivo ha sido la falta de documentación, que hace realmente difícil y tedioso implementar cualquier funcionalidad medianamente avanzada sin tener que navegar por el código de la propia librería y deducir ciertas funcionalidades mediante prueba y error. Esta parte de la implementación ha consistido en introducir los parámetros físicos necesarios en la arquitectura presentada anteriormente,

de forma que los distintos tipos de objetos puedan ser controlados correctamente por el motor de físicas.

C. Manuel Paleo

Dentro del proyecto de creación de *Tribes at War* mi contribución ha sido en gran parte el diseño conceptual y el modelado de personajes y escenarios.

Consensuándolo con el resto de miembros del equipo se ha dibujado en líneas generales una ambientación para todo el juego, generando textos que dieran una idea intuitiva del entorno en el que debía desarrollarse el mismo y delimitando ya desde ese momento qué características queríamos que tuviera. En un principio se decidió que los jugadores tuvieran la opción de pertenecer a diferentes tribus del mundo postapocalíptico en el que se desarrolla *Tribes at War* y que cada una de ellas tuviera un modelo de vehículo de combate de distinta geometría y características. En esa línea se diseñaron y modelaron al menos de forma preliminar tres modelos de tanque distintos aunque finalmente no se haya podido implementar más que uno. Se ha intentado que los diseños de estos modelos fueran acordes con la ambientación elegida y cada uno de ellos tiene una pequeña historia detrás.

Para los modelos de jugador del enemigo, NPC's, se intentó realizar diseños fríos y futuristas que fueran acorde con las corporaciones esbozadas en la ambientación y finalmente se modelaron una torreta y un tanque centinela. Quedó únicamente sobre el papel un diseño de plataforma de disparo enemiga.

En la misma línea, se han diseñado dos niveles para el juego de los cuales sólo uno pasó de la etapa de modelado preliminar, Hehlmagast, y que fue luego complementado con el diseño de un tercero realizado por Jaime. En Hehlmagast corrieron de mi cuenta el definir los objetivos del escenario y realizar el modelado de la malla de triángulos que compone el suelo y de los edificios que pueblan el nivel, así como la distribución de estos elementos. Contribuí junto con Jaime a la optimización a nivel geométrico de este escenario pues en un principio se sospechaba que era la causa del bajo rendimiento del juego. El segundo nivel que realicé y que no llegó a la fase de texturizado se llamaba Cold Volcano y estaba orientado al modo de juego Deathmatch cooperativo que es la marca de fábrica de *Tribes at War*, se desarrollaba sobre una plataforma basáltica poblada de columnas hexagonales que surgían del suelo.

Finalmente tuve como tarea la creación del video promocional del juego en el que usaba los escenarios creados para el juego y los animaba y renderizaba con Maya y 3D Studio para conseguir mayor definición o efectos más realistas que en el propio juego. En el video se enfatizaba la ambientación del juego y se intentaba dar una visión global de las características de *Tribes at War*.

D. Carlos Pérez

Mi labor ha consistido en dar apoyo en todos los frentes del proyecto, pero centrándome en la implementación del multiplayer y el interfaz del juego.

El principal inconveniente presentado ha sido la necesidad de utilizar siempre dos pcs, tanto para los desarrollos como para la realización de las pruebas.

En el desarrollo de una aplicación multiplayer con XNA se subdividen varias etapas:

- **Login en Xbox Live**
- **Configuración de la partida:** tenemos dos parámetros principales, el número máximo de jugadores que podrá haber en la partida y el número de jugadores locales a un host. En nuestro caso el número máximo de jugadores es cuatro y, debido a los problemas de rendimiento presentados, el número máximo de jugadores por host es uno.
- **Publicar la partida**
- **Unión a la partida**
- **Finalización de la partida**

He seguido un modelo incremental, comenzando con una aplicación básica en la que se presentaban dos objetos simples interactuando entre ellos, para posteriormente adaptarme a los modelos y escenarios de nuestra aplicación.

Posteriormente se valoraron las posibilidades ofrecidas por XNA a la hora de crear las partidas multijugador, tenemos dos posibilidades de juego:

- **SystemLink:** partidas sobre red local o cable cruzado
- **Signed in live:** partidas en internet. Tenemos dos parámetros principales, el número máximo de jugadores que podrá haber en la partida y el número de jugadores locales a un host. En nuestro caso el número máximo de jugadores es cuatro y, debido a los problemas de rendimiento presentados, el número máximo de jugadores por host es uno.

Decidimos conseguir el funcionamiento con SystemLink y no tener limitaciones con el rendimiento de la conexión a Internet.

El siguiente paso es integrar la aplicación multiplayer en el proyecto global.

En este punto se interactúa con todos los componentes del grupo de trabajo para:

- **Introducir las pantallas necesarias (lobbyScreen)**
- **Nombrar las restricciones implícitas de multiplayer**, como la carga dinámica de jugadores y sincronización de elementos entre otros
- **Estudiar las duplicaciones** o desdoblamiento de códigos para que sea posible el multiplayer
- Finalmente se introducen mejoras como la **comunicación por voz**

E. David Rosillo

En líneas generales se ha intentado trabajar sobre el mayor número de funcionalidades del juego pero, mi trabajo ha consistido principalmente en la creación de los menús y distintas pantallas del juego, así como la creación del PlayGUI, la inserción de los sonidos y la definición de los estados de fin de partida.

En primer lugar es importante resumir la creación de los distintos menús del juego. En un primer momento se crea una única pantalla extremadamente sencilla que no permite nada más que dos opciones, de "juego" o "salida", limitando en exceso el desarrollo y ampliación de los menús y las posibilidades de opciones de juego. Es por eso que se decide buscar una mejora u otra forma de implementar el sistema de menús y pantallas para permitir cómodamente crear nuevas pantallas y

ampliar las opciones disponibles de juego. Se cambia la estructura de forma que existe una clase controladora de todas las pantallas, en la cual se decide qué pantallas están activas y cuáles no, llamando esta misma a los métodos de actualización y renderizado de cada una de las pantallas. Se implementa también una nueva forma de opciones de menú en la que todas heredan la misma estructura, simplemente siendo necesario modificar tanto el método de renderizado, como modificando las llamadas que devuelve la selección de dicha opción, permitiendo todo esto una mayor agilidad en la creación y uso de las entradas de menú.

En segundo lugar y pese a no ser una parte primordial y esencial de un videojuego, la PlayGUI ayuda siempre al aumento de realismo y otorga toda la información necesaria al usuario del juego. La interfaz o PlayGUI consta de ciertos elementos como son: la barra de vida, tiempo transcurrido, nombre de jugador, puntuación y Minimapa. Cada uno de estos elementos, por simples que parezcan conlleva sus complicaciones. En el caso de la vida del tanque del usuario, conlleva ciertas complicaciones debido a la forma de la arquitectura del videojuego que restringe mucho el acceso a ciertos datos. El tiempo transcurrido conlleva la dificultad del cálculo del tiempo transcurrido desde la primera ejecución y la partida actual, así como los momentos de pausa de juego. Y sin duda, el de mayor complicación es la función de MiniMapa, la cual ha resultado bastante tediosa debido al número de cálculos necesarios para situar un objeto de una cámara y de un mapa tridimensional en un minimapa de dimensiones reducidas y en 2D. Todo esa dificultad multiplicada por dos debido al número de mapas existentes en la última versión.

Importante también mencionar también el trabajo realizado para la conseguir la reproducción de sonidos. Para permitir la reproducción de audio en XNA es necesario crear alguna clase que tenga ciertos métodos que son necesarios, en nuestro caso se concentran en la clase AudioManager. Debido a la forma de trabajo de XNA, es necesario crear archivos de sonido en programas distintos de XNA para conseguir su reproducción en el videojuego. Algunos de los sonidos que se han incluido para los eventos más importantes del juego, ya sea de la parte de los menús, así como para todo lo correspondiente a los eventos propios del juego son los siguientes: explosiones, sonido motor, disparos, victoria y derrota, movimiento por los distintos menús, selección de menús y salida de juego. Un problema grave que se encontró a la hora de añadir una banda sonora al juego, aparte de los muy conocidos derechos de autor, un problema persistente en el método de actualización de todas las pantallas que provocaba la reproducción multiplicada de dicha banda sonora a medida que se avanzaba en las distintas pantallas del juego.

Por último comentar el trabajo realizado para definir métodos que controlen los estados de la partida, y determinen los distintos modos de finalización de partida, ya sea por eliminación de usuario o bien porque el usuario ha alcanzado el final del escenario y por tanto ha alcanzado la victoria. Existen varios métodos que incluidos dentro de la lógica de funcionamiento del juego permiten cómodamente determinar las variables que deciden dicha finalización.

Lo más costoso para mí ha sido añadir los sonidos y hacerlos funcionar correctamente por las especificaciones que se han comentado y se comentan más abajo con más detalle, así como añadir la funcionalidad del minimapa de la PlayGUI que ha requerido más trabajo del pensado en un principio.

F. Tabla de esfuerzo

	J. González	E. Miguel	M. Paleo	C. Pérez	D. Rosillo	Subtotal
Programación	46	165	0	13	65	289
• Arquitectura inicial	5	20	-	2	10	37
• Implementar módulos en función de necesidades	10	30	-	2		42
• Integración de físicas	-	25	-	3		28
• Adaptar físicas al juego	-	35	-	0		35
• Animaciones/cámaras	15	20	-	1		36
• Inteligencia artificial	5	25	-	0		30
• Interfaz	-	-	-	2	30	32
• Audio	1	-	-	3	20	24
• Sist. de partículas	-	10	-	0	5	15
• Iluminación	10	0	0	0	0	10
Multiplayer	-	-	0	129	5	134
• Configuración e inicio de la partida	-	-	-	15	-	15
• Finalización y liberar sesión	-	-	-	12	-	12
• Estructuración de	-	-	-	20	-	20

mensajes						
• Adaptación del código single player	-	-	-	35	-	35
• Topología de red	-	-	-	20	-	20
• Sincronización	-	-	-	27	5	3
Herramientas	40	0	0	0	0	40
• Exportador de propiedades	5	-	-	-	-	5
• Lectura de datos de escenarios	20	-	-	-	-	20
• Exportar datos del escenario	15	-	-	-	-	15
Modelado	30	-	80	-	-	110
• Geometría de objetos	5		25			30
• Edición de los niveles	10		20			30
• Escenarios baja resolución	5					5
• Elementos de decoración	5		5			10
• Vehículos	5	-	30	-		35
Texturizado	35	0	15	5	40	95
• Escenarios	25	-	5	-	-	30

• Vehículos	10	-	10	-	-	20
• Menús	-	-		5	40	45
Audio	0	0	0	5	30	35
• Efectos	-	-		5	30	35
Vídeo	0	0	17	0	0	17
• Generación de contenido	-	-	10	-	0	0
• Composición	-	-	5	-	0	0
• Compresión y formatos	-	-	2	-	0	0
TOTAL	151	165	112	152	140	720

V. Problemas y dificultades

A. Gestión de eventos y colisiones

La gestión de eventos de colisiones es fundamental en el correcto funcionamiento del juego ya que se encarga de todas las operaciones necesarias ante una colisión. Este tipo de eventos son muy frecuentes: colisiones entre los tanques y el terreno, colisiones entre tanques, impactos de proyectiles sobre cualquier otro objeto, etc. El tratamiento que hay que dar a cada evento es distinto, por lo que ha sido necesario implementar una máquina de estados capaz de identificar cada tipo de colisión para operar de forma adecuada. Así, por ejemplo, el tratamiento para colisiones entre terreno y tanques permite establecer una determinada propiedad en el tanque para simular la tracción del mismo. Por ejemplo, en el caso de perder contacto con el terreno tras pasar a gran velocidad por una elevación, el vehículo deja de ser controlable. Así, mediante este mecanismo de gestión eventos se ha podido clasificar cada tipo de colisión y tratarla correctamente.

B. Implementación flexible de las animaciones

Con el objetivo de poder añadir futuras animaciones de forma sencilla, se planteó la posibilidad de implementar un mecanismo lo más flexible posible que lo facilitase. La principal dificultad que planteaba era cómo implementarlo sin requerir una jerarquía de clases muy extensa. Por ejemplo, una misma animación debería valer tanto para tanques remotos, enemigos o locales, por lo que si se implementaba como un método dentro de las clases sería necesario duplicar el código y mantener un cierto control sobre cada uno de ellos. Por ello, se optó por la creación de una clase genérica que implementase la interfaz básica de cualquier animación, y en la creación de cada objeto, asignar la animación correspondiente, con una jerarquía de animaciones totalmente independiente de la jerarquía de objetos del mundo. La única limitación que se impone es que la animación debe adaptarse a la jerarquía del modelo sobre el que se aplica, por lo que hay que tener cuidado con eso.

C. Problemas de colisiones con los elementos "box" de JigLibX

En la integración de la librería de físicas se han utilizado cajas envolventes como malla de colisión para los tanques y las torretas. Sin embargo, cuando se ha tratado de utilizar con elementos como los edificios del escenario han surgido problemas que no hemos sabido resolver en un tiempo razonable, y por ello y debido a la falta de tiempo para investigar a fondo el funcionamiento de este tipo de objetos en la librería hemos tomado la decisión de utilizar mallas de triángulos para las colisiones de los vehículos y proyectiles con los elementos comunes (terreno y elementos no jugables). Como se verá posteriormente esto también ha generado algunos problemas.

D. Problemas de rendimiento con múltiples mallas de triángulos como mallas de colisión

Tras el paso del punto C, se empezaron a observar bajadas importantes en el rendimiento. El problema era que la librería de físicas procesa las colisiones de todos los elementos del escenario, de forma que incluso los objetos de tipo "terreno" que tienen activa la propiedad de "inmóvil" eran procesados unos con otros. En el caso de objetos con mallas de triángulos como mallas de colisión resulta obvio que la cantidad de procesamiento es considerable, por lo que tras analizar las soluciones ofrecidas por la librería de físicas, se optó por establecer una serie de comprobaciones en la gestión de colisiones para no procesar aquellas entre elementos inmóviles. Como precaución para futuras implementaciones, también se revisó el mallado de los modelos para reducir en la medida de lo posible el número de polígonos de cada uno.

E. Fallos en la detección de colisiones

Tras algunos avances en la carga de escenarios a partir de ficheros XML, comenzamos a tener problemas con las colisiones. En muchos casos no eran detectadas correctamente, observándose interpenetraciones entre objetos del escenario. A pesar de que se buscó la raíz del problema, probablemente en las rotaciones y traslaciones aplicadas en el fichero XML generado desde la herramienta de modelado, no se logró localizar el error, ya que el número de etapas a analizar

(modelado a XML, XML a procesador de contenido de XNA y de ahí al juego) y el tiempo disponible lo hacían bastante difícil. Por ello, finalmente se decidió generar los escenarios como un único modelo compuesto por diferentes mallas, que en el procesado genera una única malla de colisiones dentro del motor de físicas. De esta forma se logró obtener un buen comportamiento en la detección de colisiones.

F. Pequeñas bajadas de rendimiento en escenarios grandes

Con la modificación del punto E se observaron pequeñas bajadas de rendimiento en escenarios grandes, debidas en gran medida a la densidad de la malla de colisiones empleada con el terreno. Para resolver esto se ha optado por la utilización de dos mallas diferentes en los objetos que lo requieran. Una de alta resolución para el renderizado, y una de baja resolución para la detección de colisiones, lo que ha hecho que el rendimiento sea notablemente mejor.

G. Creación fondo de pantallas acorde a la tónica del juego

En las primeras versiones del juego es un factor totalmente irrelevante pero a medida que avanza el juego y se va diseñando una interfaz, historia e iluminación se hace obligatorio crear un fondo de pantalla acorde. Para ellos se realizan numerosos bocetos utilizando la herramienta photoshop, que durante las diversas entregas obligatorias, se va corrigiendo y cambiando de diseño hasta alcanzar el de la entrega final.

H. Reproducción sonidos

Debido a la estructura de XNA, se obliga a que los archivos de sonido estén incluidos en un fichero de tipo .xap que incluye un Wave Bank y un Sound Bank donde se añaden los archivos de sonido en formato wav. Una vez completados todos estos pasos previos, es necesario crear métodos en el proyecto que te permitan controlar el archivo de sonido. El mayor problema fue la reproducción múltiple de sonidos en la pantalla de juego, que consistía en un bucle infinito por ejemplo, de explosiones al intentar recrear un disparo. El problema se solucionó después de una búsqueda tediosa, llamando a los métodos de reproducción de cada uno de los eventos desde métodos distintos del Update.

I. Intercambio de pantallas

Un efecto muy desagradable, visualmente hablando, que se producía en el momento de selección de algún menú y por tanto de cambio de pantalla, era la permanencia de dicho menú perteneciente a la pantalla anterior en la siguiente pantalla, durante un periodo de tiempo lo suficientemente largo para desagradar al usuario. Este efecto permanecía más tiempo, si las pantallas incluían algún tipo de imagen o textura. La solución consistió en obligar al sistema a esperar un determinado tiempo en los cambios entre pantallas, ampliando dicho tiempo de espera en el caso de incluir imágenes.

J. Creación función de minimapa

Uno de los trabajos más simples a priori de la PlayGUI, pero que esconde la necesidad de realizar algunos cálculos y dar un par de vueltas para conseguir el resultado deseado. El problema en sí consiste en cómo pasar las coordenadas de representación del tanque en 3D en un sistema o mapa en 3D

también a una captura de dicho mapa en 2D de dimensiones muchísimo más pequeñas y lo que es más importante, renderizado en dos dimensiones. La solución pasaba por observar las coordenadas máximas del mapa en 3D y calcular la traslación y escalado de dichas dimensiones a la textura de la captura de dicho mapa. Además era necesario calcular una proporción del desplazamiento del tanque, lo suficientemente ajustada como para permitir al usuario desenvolverse en el mapa de 3D, siguiendo el mapa que aparece en la PlayGUI de su pantalla de juego.

K. Modelo topológico del multiplayer

En los comienzos del desarrollo del multiplayer se utiliza un modelo peer to peer, en el cuál la conviven en cada host su propio motor de físicas.

En cada fase de update se transmite la información al resto de elementos. La información enviada es el Force y Torque generado en su motor.

Este sistema produce variaciones de las imágenes presentadas en cada host debido al procesamiento independiente de cada máquina (diferente rendimiento, diferente precisión, etc).

Se valora la posibilidad de una topología cliente/servidor, finalmente descartada por la sobrecarga que añade en el equipo servidor.

Para solventar la desincronización, se presenta la siguiente solución:

Asignar a uno de los equipos como master, teniendo la responsabilidad de elegir el escenario de la partida y ejecutar una función de sincronización.

Cada jugador envía su posición y orientación al resto de los participantes los cuales actualizarán sus valores.

El equipo master envía la posición y orientación en broadcast de todos los jugadores para mantener la coherencia.

L. Falta de logs en la librería de red.

La librería tiene algunas peculiaridades en cuanto a la recepción de mensajes; solo puede haber una llamada de recepción para un funcionamiento correcto, si tenemos varias el compilador no nos muestra ningún tipo de aviso y no sabemos con certeza donde se recibe la información.

El envío de información tiene la siguiente restricción: los mensajes son siempre enviados en broadcast.

Estas y otras peculiaridades son adquiridas durante el desarrollo, por lo que sería de gran ayuda contar con trazas para verificar las recepciones.

M. Problema de rendimiento por alto número de polígonos

En un principio se sospechaba que el número de polígonos era demasiado elevado y reducía el *framerate* del juego de forma sensible. Para solucionar esto se optimizaron los modelos de dos formas: la primera totalmente manual que consistía en eliminar vértices en el programa de modelado que se estuviera usando en base al criterio del modelador y con especial atención a las caras planas con vértices que no formaran parte de sus bordes, y una segunda pasada en Mod Tool con la herramienta de

reducción de polígonos. Habitualmente esta segunda pasada volvía a retocarse a mano y se corregían los fallos generados por el algoritmo, especialmente visibles en los bordes de las mallas. Con estas medidas se consiguió reducir el escenario Hehlmagast, el de mayor número de modelos, por debajo de los 25.000 vértices. Finalmente se comprobó que el problema principal residía en las físicas y se palió como se detalla en el punto D.

VI. Referencias

- http://www.gamasutra.com/resource_guide/20030714/hao_01.shtml
- http://www.gamasutra.com/view/feature/3774/postmortem_2k_boston2k_.php
- http://www.gamasutra.com/view/feature/3809/postmortem_naughty_dogs_.php