Proyecto de Fin de Carrera

Herramientas Exportación/Importación de modelos 3D

Interacción 3D Studio MAX - Unity3D

Autor: Fernando Matarrubia García (fernando.matarrubia@um.es)

Director proyecto: Mariano Flores Gutiérrez (ocytsim@um.es)

Convocatoria: Junio 2010

Universidad de Murcia 2009/2010

Departamento: Informática y Sistemas

Abstract

Los videojuegos son considerados por muchos como el octavo arte. Diversas disciplinas artísticas consiguen converger en un mismo producto dando lugar a un medio de entretenimiento que está creciendo de forma exponencial en los últimos años.

Es posible que a simple vista alguien no familiarizado con la industria pueda pensar que realizar un videojuego es una tarea trivial, pero nada más alejado de la realidad. Desde la figura del diseñador principal de la mecánica del juego, pasando por la de los artistas, programadores, modeladores y animadores, todas ellas son esenciales en el proceso de creación de una obra de entretenimiento virtual. Tanto la animación como los videojuegos son dos sectores que a mundialmente cada vez cobran más importancia, abarcando cuotas de mercado que llegan a superar a la del cine, pero que a nivel nacional no han conseguido desplegar todo su potencial.

Todos los elementos de un videojuego en 3D deben ser necesariamente realizados con herramientas especializadas. El uso de dichas herramientas es muy complejo, y normalmente existen profesionales especializados en cada uno de sus campos: modelado, texturizado, animación e iluminación. Por otro lado, en la realización de un videojuego es necesario contar con un *engine* (o motor de juegos) que centralice todos los recursos a utilizar y contenga todas las herramientas necesarias para su desarrollo.

El presente proyecto se sitúa "a caballo" entre ambas industrias. El fin de la "*Herramienta de Importación/Exportación de modelos 3D*" es simplificar el salto existente entre la creación de recursos 3D para un videojuego, y el uso directo que se hace de ellos en el motor, facilitando el flujo de comunicación entre ambos programas. En los siguientes apartados de este documento se dará una breve introducción tanto de los programas de creación de gráficos y animación 3D como de los motores gráficos; se explicarán cuáles son sus principales diferencias, y qué aspectos se deberán tener en cuenta al realizar la migración de recursos entre uno y otro; se detallará la metodología de trabajo llevada a cabo en el proyecto, dentro de un grupo de trabajo real; y se expondrán tanto la especificación como la implementación de una primera solución para el problema planteado. Finalmente, se mostrarán algunos resultados obtenidos gracias a las herramientas realizadas.

Índice

Índice de Ilustraciones

Índice de Pseudocódigos

Índice de Tablas

1. Introducción y referencias históricas

1.1.Contexto

La empresa de producción audiovisual *Neotecno Desarrollos S.L* actualmente se encuentra trabajando en numerosos proyectos relacionados con el campo de la infografía[1]: documentales en 3D, visitas interactivas para museos, mundos virtuales, pequeños videojuegos e incluso una serie de animación.

En colaboración con la Universidad de Murcia, *Neotecno Desarrollos* propone realizar una herramienta que simplifique diversas tareas dentro de su flujo de trabajo, y mejore la productividad de sus empleados.

Figura 1. Neotecno Desarrollos S.L.

En líneas generales, se busca desarrollar una herramienta que cumpla los siguientes objetivos y características:

- Facilitar la comunicación entre los diversos departamentos del equipo, simplificando las tareas de compartición de recursos entre ellos
- Ahorrar tiempo, y por ende costes, en tareas atómicas de planteamiento mecánico que son repetidas numerosas veces
- Presentar un fácil manejo, requiriendo el menor tiempo de aprendizaje posible, y garantizando un máximo aprovechamiento por parte del usuario

Se pretende realizar por tanto una solución software heterogénea que abarca dos programas relacionados con la informática gráfica y los motores de videojuegos: *3D Studio MAX* y *Unity 3D*.

Debido a las diferencias existentes entre ambos programas, y a los lenguajes de desarrollo que soportan, el software final estará compuesto por dos plugins[2], cada uno de los cuales se integrará en uno de los dos entornos de trabajo mencionados.

[1] La **infografía** (información+grafía) es una forma de comunicación que consiste en la presentación visual de información más allá del simple texto escrito

[2] Un ***plugin*** (o plug-in) se define como un pequeño programa que extiende las capacidades de uno de mayor envergadura, integrándose en éste de manera transparente al usuario

1.2.Motivación

La productividad de una empresa es uno de los factores que mejor la definen. En dependencia directa de esta característica se encuentran los beneficios obtenidos, y por tanto, la durabilidad de la organización.

Se hace necesario intentar aprovechar todos los medios posibles para simplificar la cadena de trabajo y agilizar el flujo del mismo, por lo que el desarrollo de herramientas que consigan precisamente esto, es una de las prioridades de toda organización empresarial.

Dentro del campo de la infografía, se parte de una unidad básica que son los recursos. Un recurso es toda aquella unidad de información que puede ser utilizada sóla, o en combinación con otras para dar lugar a un *total* que muestre la información que se desea.

Si tenemos en cuenta el carácter multidisciplinar de la infografía, se puede afirmar que existen gran cantidad de tareas diferentes, cada una de las cuales genera recursos, los cuales deberán ser reutilizados en diferentes contextos para conseguir un resultado final.

La motivación real de este proyecto es agilizar los procesos que impliquen compartir recursos entre diferentes grupos de trabajo, automatizando en la medida de lo posible las tareas de adaptación que se deben de realizar a la hora de enviar la información y de recibirla.

Más concretamente, se pretende optimizar el proceso de exportación de una escena tridimensional desde *3D Studio* e importarla posteriormente en el motor de juegos *Unity 3D*, con la menor pérdida de información posible.

1.3.Informática Gráfica

La Informática Gráfica es una rama de la informática que se encarga de la creación de representaciones gráficas dentro del propio computador. Estas representaciones pueden ser bidimensionales o tridimensionales, y son generadas en su totalidad por la propia máquina.

Al contrario de lo que pueda parecer, la informática gráfica guarda una estrecha relación con la evolución de los computadores en los últimos tiempos, ya que por ejemplo, para realizar las complejas y numerosas operaciones de creación de una escena tridimensional se requiere de potencias de cálculo cada vez mayores conforme ésta va aumentando su complejidad.

Por otro lado, es evidente que la informática gráfica guarda una estrecha relación con la infografía ya que ambas tienen como objetivo final la representación visual de una entidad determinada.

Una de las mayores metas que se persiguen en este campo de la computación es llegar a conseguir el fotorrealismo en una representación ficticia de la realidad.

1.4.Representación 3D

La representación 3D básica por medio de un computador se realiza a partir de la proyección de figuras tridimensionales sobre un plano bidimensional (el monitor).

Una figura tridimensional está formada por polígonos, cada uno de los cuales a su vez está formado por triángulos que a su vez están formados por vértices. El computador almacena la información sobre las posiciones de dichos vértices, y qué relaciones existen entre ellos, en pos de determinar entre qué vértices se deben dibujar los triángulos correspondientes.

De esta manera, a través de información puramente numérica, se puede proyectar sobre la pantalla una figura tridimensional generada completamente por el computador.

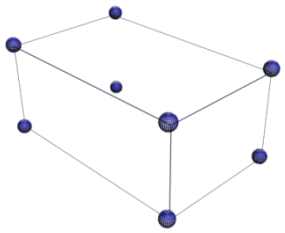

Figura 2. Objeto 3D con 8 vértices

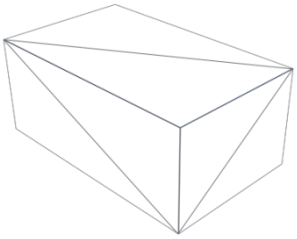

Figura 3. Objeto 3D con 12 triángulos

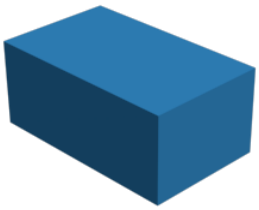

Figura 4. Objeto 3D con 6 caras

Las diferentes interacciones y movimientos de la figura se materializan a ojos de la CPU como traslaciones y rotaciones de los diferentes vértices. Es obvio pensar que el trabajo que conlleva tanto la representación como la variación de las figuras de una escena genera una cantidad importante de operaciones numéricas que sobrecargan la CPU.

Por ello, en los últimos tiempos se han desarrollado las tarjetas gráficas con aceleración 3D. Dichas tarjetas, cuentan con pequeños núcleos de proceso, llamados GPUs (Graphics Process Unit) en los que se delegan las operaciones relacionadas con los gráficos.

El contexto de este trabajo sitúa las escenas tridimensionales como punto de partida. Una escena tridimensional puede estar formada por una serie de objetos indeterminada, y por tanto contener un número de triángulos muy significativo.

Uno de las fases de nuestro software, permitirá encapsular todos los objetos de la escena en un bloque de información portable para posteriormente leer de ese bloque dicha información y representar una escena igual a la original dentro de un espacio tridimensional diferente correspondiente a un software no relacionado con aquel con el que se realizaron los modelos.

1.5.Elementos de una escena 3D

Existen una serie de elementos básicos, comunes a prácticamente todas las escenas 3D, que serán los cuales habrá que tener en cuenta en el presente proyecto:

- *Objetos 3D*: Son la unidad básica de la escena. Básicamente son mallas de puntos que definen una geometría determinada. El término *modelar* significa dar forma a dichas mallas, obteniendo los resultados que se desean.

- *Cámaras*: Como su propio nombre indica, son los objetos desde los cuales se observa la escena. Cuentan con una serie de parámetros configurables. Los más importantes son el campo de visión, y los límites inferiores y superiores del mismo. Una escena 3D puede tener varias cámaras cada una de las cuales otorga un punto de vista diferente.

- *Iluminación*: La iluminación de una escena es esencial. En el mundo de las 3D, una escena sin iluminación es un cuadro totalmente negro. Permite cambiar totalmente la apariencia de una escena dada. Se configura a través de diferentes puntos de luz que a su vez se subdividen en diferentes tipos, los más comunes son:

    - *Point*: Luz de tipo punto. Alumbra radialmente en todas direcciones con una atenuación dada. Un *point* en la vida real puede ser una bombilla

    - *Spot*: Luz en forma cónica. Alumbra a lo largo de su cono, con una atenuación dada. Un *spot* en la vida real puede ser un foco de luz en un escenario.

    - *Omni*: Alumbra radialmente en todas direcciones con una intensidad dada, e igual en todo su alcance. El *omni* más evidente es el Sol

- *Texturas*: Las texturas son las imágenes que se dibujan encima de los objetos. Las texturas se colocan encima de los objetos en base a sus coordenadas UV.

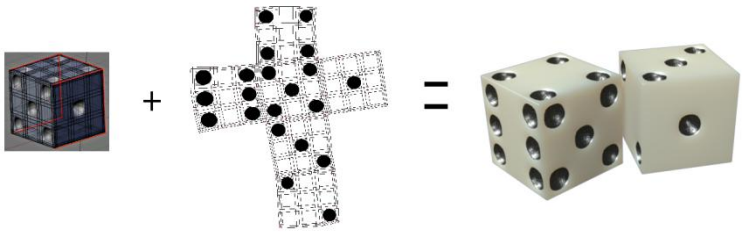

Figura 5. Materiales. Coordenadas UV

- *Animación*: Otra característica base de una escena 3D es la animación que presentan los objetos de la misma. El movimiento de los objetos puede ser simple, es decir, únicamente traslaciones y rotaciones de los mismos, o más avanzado, mediante por ejemplo, sistemas de huesos.

1.6.Videojuegos

A día de hoy, la definición más ajustada de videojuegos podría ser algo como sigue:

"*Todo juego electrónico con objetivos esencialmente lúdicos, que se sirve de la tecnología informática y ermite la interacción a tiempo real del juegador con la máquina, y en el que la acción se desarrolla fundamentalmente sobre un soporte visual (que puede ser la pantalla de una consola, de un ordenador personal, de un televisor, o cualquier otro soporte semejante).*"[1]

Donde podemos distinguir dos aspectos muy importantes sin los cuales un videojuego en sí no tendría sentido: "*Todo juego electrónico se sirve de la tecnología informática y (..) la acción se desarrolla fundamentalmente sobre un soporte visual*."

A lo largo de la historia del videojuego, la evolución de la tecnología utilizada, así como la de los soportes de representación, ha sido realmente impresionante.

Hay gran discrepancia sobre cuál fue el primer videojuego de la historia, existen dos claros referentes, por un lado "*Tennis for two*"[3] creado por William Higinbotham en 1957, y utilizando un osciloscopio para su representación visual, y por el otro "*Spacewar*"[4] desarrollado en 1961 por Steve Russell junto en el Instituto de Tecnología de Massachusetts, y utilizando un computador para permitir su visionado.

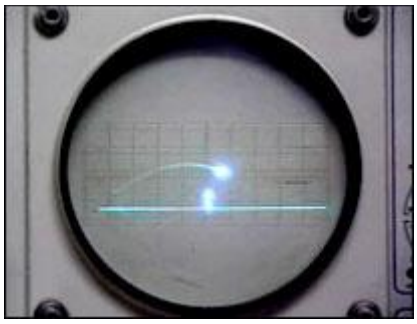

Figura 6. Tennis for Two (1957)
Fuente: Wikipedia

Figura 7. Spacewar (1961)
Fuente: Wikipedia

Con la aparición de las 3D, los videojuegos dieron un enorme salto de calidad hacia un objetivo que aún a día de hoy pretenden conseguir: sumergir al jugador en una atmósfera tridimensional lo más realista posible.

[3] Más información: http://es.wikipedia.org/wiki/Tennis_for_Two

[4] Más información: http://es.wikipedia.org/wiki/Spacewar!

La auténtica expansión comercial de los videojuegos en 3D se puede considerar que comenzó en 1992 gracias al arcade de carreras "*Virtual Racing*"[5] de Mega Driver y continuando en 1993 con el genial shooter "*Star Fox*"[6] de SNES.

Ambos son claros referentes de cómo una nueva dimensión llegó al mundo del entretenimiento electrónico.

Figura 8. Virtual Racing (1992)
Fuente: Elotrolado.net

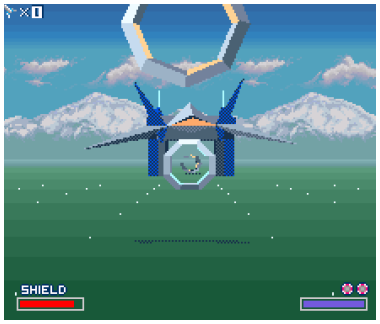

Figura 9. Star Fox (1993)
Fuente: Gamespot.com

Actualmente, es difícil concebir un videojuego sin pensar en ricos y decorados entornos preciosistas, modelos perfectos de personajes y enemigos, o efectos visuales lo más realista posible.

Como ejemplo, si contamos la cantidad de polígonos del personaje principal del videojuego Crysis[7] (actual referente gráfico en el mundo de los videojuegos), éstos ascienden a 67.000[8], comparados con los 3 sprites bidimensionales del ya mencionado *Spacewar*, dejan entrever la inclinada curva de la evolución visual del videojuego en los últimos 40 años.

Esta evolución se lleva a cabo realmente dentro de los motores de videojuegos. Mejoras en el rendimiento y la calidad de las tarjetas gráficas y computadores, permiten cada vez más complejidad a la hora de desarrollar los gráficos y efectos de una obra de entretenimiento electrónico.

Y es aquí donde el presente proyecto se marca uno de sus objetivos. Si bien hemos mencionado anteriormente la idea de encapsular modelos de una escena a un formato portable, ahora hablamos de importar esos recursos en un motor de videojuegos, haciendo la operación lo más atómica posible, y facilitando la tarea tanto al desarrollador que genera los recursos y los envía, como para aquel que los recibe y utiliza.

---

[5] Más información: http://en.wikipedia.org/wiki/Virtua_Racing

[6] Más información: http://es.wikipedia.org/wiki/Star_Fox

[7] Más información: http://es.wikipedia.org/wiki/Crysis

[8] Evolución de la cantidad de polígonos en los videojuegos: http://9esferas.blogspot.com/2008/03/va-de-poligonos.html

1.7.Motores de Juego

La evolución de la informática gráfica en los últimos tiempos ha sido realmente espectacular[9]. Desde las primeras representaciones vectoriales de gráficos simples en pantallas de rayos catódicos (CRT), hasta las complejas representaciones tridimensionales actuales en monitores LED. Evidentemente, todo este desarrollo conlleva una programación gráfica de las entidades visuales donde la mejora en calidad y prestaciones también conlleva un aumento en complejidad.

Desde la década de los 90, se comenzaron a utilizar librerías gráficas que simplificaban la tarea de trabajar con entidades tridimensionales, ofreciendo al usuario una interfaz estable con la que implementar funciones gráficas de manera más sencilla. Dichas librerías convergen básicamente en dos tipos de referencia *OpenGL* (Open Source) y *DirectX* (propietaria de Microsoft).

Parejo a la aparición de dichas librerías, surgen los motores de juego. Formalmente, un motor de juegos podría definirse como una solución software compuesta al menos de un motor gráfico, y una serie de librerías, ideado para la realización de aplicaciones 3D en las que la interacción con el usuario sea parte central de las mismas. Las librerías incluidas en un motor de juegos podrían ser un motor de físicas, que se encargue de otorgar un comportamiento realista a los objetos de la escena, librerías de audio, de Inteligencia Artificial, etc.

Conviene hacer una distinción entre motor de gráficos y motor de juegos. Un motor de gráficos es parte de un motor de juegos, y únicamente se encarga de otorgar al usuario los medios necesarios para el manejo de gráficos tanto bidimensionales como 3D, mientras que un motor de juegos, como se ha dicho anteriormente, integra una serie de librerías que extienden su funcionalidad hasta abarcar el desarrollo completo de un videojuego. Dentro de un motor de juegos, lo más visible es el motor gráfico, ya que es el encargado de la parte visual, y por ello se tiende a asociar ambos conceptos en uno sólo.

En el presente proyecto, se realizará un plugin para el motor de juegos *Unity3D*, que importa una serie de elementos tridimensionales en una escena determinada[10] haciendo uso del motor de gráficos.

[9] A Critical History of Computer Graphics and Animation: http://design.osu.edu/carlson/history/lessons.html

[10] Dentro de Unity3D, las aplicaciones gráficas se dividen en escenas o niveles cada uno de las cuales es totalmente independiente de las demás

1.8.Referencias Históricas

La historia de la informática gráfica se remonta a la primera mitad del siglo XX, cuando se comenzaron a representar los primeros gráficos creados por computador en monitores CRT. La evolución hasta nuestros días ha ido dejando por las diferentes épocas muestras en forma de videojuego del resultado de poner en manos de la creatividad las herramientas adecuadas.

El primer motor gráfico data de principios de los años 90, y fue creado por de idSoftware[11], dando lugar a obras de culto como *Doom*, o *Quake*. Desde la aparición de aquel motor, surge una nueva forma de encarar el desarrollo de un videojuego, separando tanto práctica como conceptualmente las fases de creación de contenido y de utilización del mismo. Nuevos motores surgieron años después, hasta día de hoy, donde la oferta es amplia y abarca todo tipo de segmentos de usuarios y mercado.

A continuación se puede observar la evolución a lo largo de la historia de los gráficos del videojuego *Unreal*. La mayoría de las versiones de este juego, han sido publicadas como reclamo a las desarrolladoras como demostración de la potencia del motor *Unreal Engine* (*Epic Games*[12]):

Figura 10. Unreal (1998)
Fuente: 3dJuegos.com

Figura 11. Unreal Tournament (1999)
Fuente: meristation.com

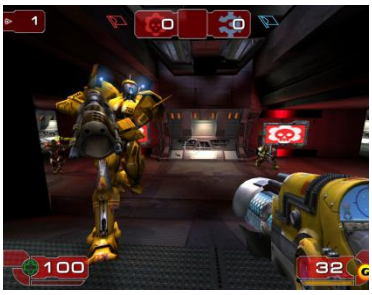

Figura 12. Unreal Tournament 2003 (2002)
Fuente: gamespot.com

Figura 13. Unreal II (2003)
Fuente: meristation.com

Figura 14. Unreal Tournament 2004 (2004)
Fuente: gamespot.com

Figura 15. Unreal Tournament 3 (2007)
Fuente: gamespot.com

[11] Página web oficial: http://www.idsoftware.com/

[12] Página oficial: http://www.epicgames.com/

2. Análisis de Objetivos y Metodología

2.1.Objetivos

El presente proyecto ha sido realizado en un entorno de trabajo real dentro de la empresa *Neotecno Desarrollos S.L*. Es por ello que el objetivo primordial del software desarrollado sea obtener un aumento de productividad en la empresa, simplificando en la medida de lo posible la comunicación entre los diferentes departamentos y grupos de trabajo.

Más concretamente, los objetivos propuestos previos a la realización del proyecto son los siguientes:

- Desarrollo de dos plugins, uno para la plataforma de creación de contenido 3D y otro para el motor de juegos en los que se utilizará dicho contenido. Se busca claridad en la interfaz gráfica de los mismos, facilidad de uso y retroalimentación con el usuario.

- Diseño modular que permita la modificación aislada de las diferentes características de las aplicaciones.

- Carácter individual de cada herramienta, facilitando el uso aislado por diferentes grupos de trabajo.

- Uso de tecnología XML para el guardado de información, consiguiendo una estructura organizada y multiplataforma de la información y evitando la redundancia de datos.

- Obtención de una conversión exacta del contenido, que evite al receptor de los recursos tener que realizar adaptaciones.

- Establecer un control de versiones a lo largo del desarrollo. Se buscará la comunicación con los potenciales usuarios de la aplicación y de esta manera realizar un software "adaptado" a un flujo de trabajo determinado en la medida de lo posible.

2.2.Metodología

La metodología adoptada es una adaptación de la metodología utilizada dentro de la empresa *Neotecno Desarrollos*. En el papel de usuario final (o cliente) se sitúan los empleados de la empresa, y el proyecto a realizar son los Plugins de exportación/importación.

Se adopta una metodología incremental:

"El modelo incremental aplica secuencias lineales de forma escalonada mientras progresa el tiempo en el calendario. Cada secuencia lineal produce un «incremento» del software. El plan afronta la modificación del producto central a fin de cumplir mejor las necesidades del cliente y la entrega de funciones, y características adicionales. Este proceso se repite siguiendo la entrega de cada incremento, hasta que se elabore el producto completo"[2]

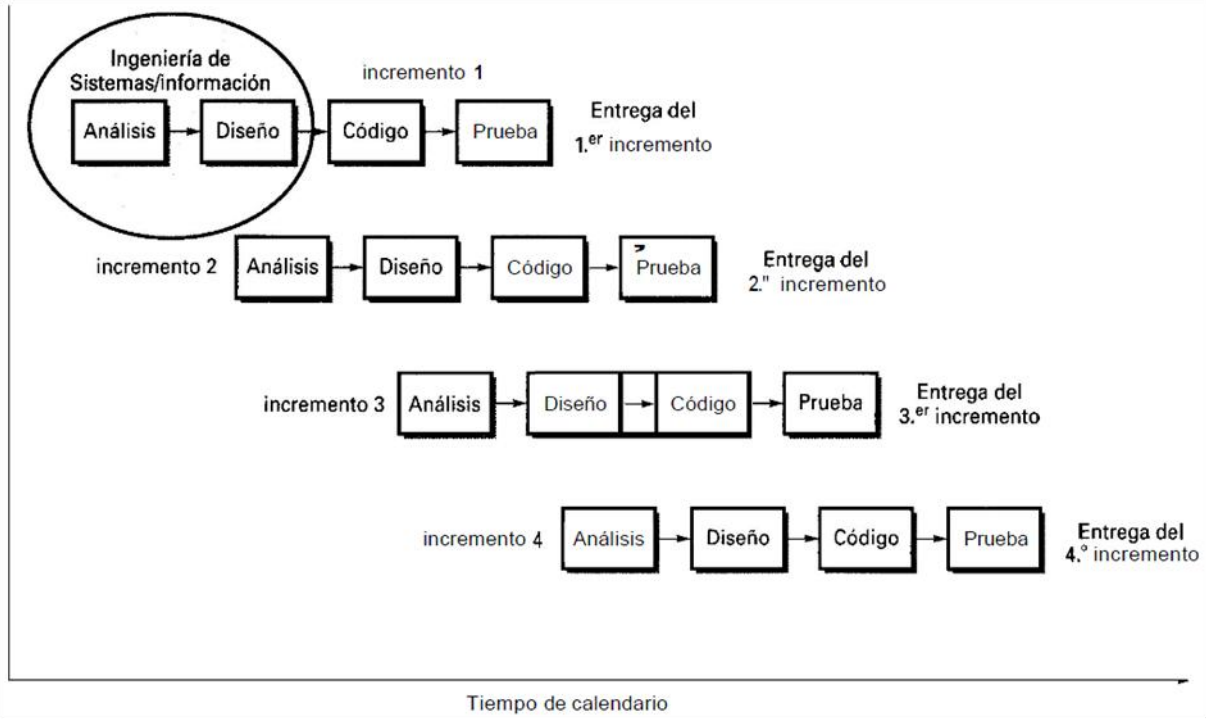

Figura 16. Esquema de metodología incremental

Atendiendo a la definición, se puede afirmar que esta metodología facilita la retroalimentación con los diferentes grupos de trabajo, de manera que, en función de las peticiones de éstos, se puedan modificar diferentes funcionalidades de la aplicación en cada incremento.

La elección de dicho sistema de trabajo se justifica en base a la naturaleza del proyecto. Ideado como un sistema modular en el que existen interacciones con personas externas al mismo, es fácil encuadrar cada tarea en una parte de un determinado incremento. Por otro lado hace la planificación escalable, de manera que para futuras vías de trabajo sea sencillo incorporar nuevas etapas en la planificación.

Para adaptar el sistema incremental a los requerimientos de la empresa y a las solicitudes de los empleados, se ha definido un esquema adaptado para cada incremento, dando lugar a un cronograma detallado sobre las diferentes tareas a realizar.

La estructura de cada incremento atiende a la siguiente configuración:

Figura 17. Esquema de incremento adaptado

El cronograma de tareas, cada una de ellas junto a su descripción, es el siguiente (se puede identificar la subdivisión en incrementos):

Incremento	*Tarea*	*Detalles*
Fase Inicial	Revisión de Documentación. 3DStudio Unity, tutoriales. Pruebas	Puesta a punto del sistema con material anterior. Revisión de documentación sobre 3DStudio y formato FBX. Revisión de tutoriales básicos de Unity. Pruebas de Exportación desde 3DStudio y lectura de datos desde Unity
Exportación/Importación de Objetos Simples	Documentación y análisis	Primeros diseños de plugins básicos, tanto para 3DStudio como para Unity3D. Posibles soluciones y alternativas a la exportación de objetos
	Exportación de objetos	Exportación de objetos desde 3DStudio. Formato XML y revisión de requisitos incluidos en archivos FBX
	Importación de objetos estáticos	Importación de objetos en Unity3D. Desde formato XML combinando con lectura de archivos FBX.
	Ejecución y depuración	Ejecución de prototipo básico (Mallas) de los plugins. 3dStudio -> XML <- Unity3D
	Puesta en común. Requisitos	Puesta en común del trabajo realizado. Solicitud de requisitos/detalles a implementar con los modeladores
	Adaptaciones	Adaptaciones de la aplicación a los requisitos de los empleados. Cambios menores a nivel de interfaz y código para facilitar el uso de la aplicación
Materiales y Texturas	Documentación y análisis	Análisis de la gestión de materiales en 3DStudio, y posibles soluciones para su adaptación a Unity3D
	Exportación de materiales y texturas	Exportación de materiales y texturas de la escena. Extraer archivos y localizarlos en directorios. Escritura de información en XML
	Importación de materiales	Pruebas de importación de materiales y texturas en Unity3D
	Ejecución y depuración	Pruebas de importación y exportación con materiales. Integración en las diferentes plataformas
	Puesta en común. Requisitos	Puesta en común del trabajo realizado. Solicitud de requisitos/detalles a implementar con los modeladores
	Adaptaciones	Adaptaciones de la aplicación a los requisitos de los empleados. Cambios menores a nivel de interfaz y código para facilitar el uso de la aplicación
Luces	Documentación y análisis	Análisis de los diferentes tipos de luces en 3DStudio y de sus correspondencias con las presentes en Unity3D. Configuración de las equivalencias
	Exportación de luces	Exportación de luces en 3DStudio. Formato FBX. Escritura en archivo XML
	Importación de luces	Configuración en Unity3D de las luces importadas. Aplicación de las equivalencias

	Ejecución y depuración	Pruebas de importación y exportación con luces. Integración en las diferentes plataformas
	Puesta en común. Requisitos	Puesta en común del trabajo realizado. Solicitud de requisitos/detalles a implementar con los modeladores
	Adaptaciones	Adaptaciones de la aplicación a los requisitos de los empleados. Cambios menores a nivel de interfaz y código para facilitar el uso de la aplicación
Cámaras	Documentación y análisis	Análisis de los diferentes tipos de cámaras en 3DStudio. Estudio de las equivalencias con Unity3D.
	Exportación de luces	Exportación de cámaras en 3DStudio. Formato FBX. Escritura en archivo XML
	Importación de luces	Configuración en Unity3D de las cámaras importadas. Aplicación de las equivalencias
	Ejecución y depuración	Pruebas de importación y exportación con cámaras. Integración en las diferentes plataformas
	Puesta en común. Requisitos	Puesta en común del trabajo realizado. Solicitud de requisitos/detalles a implementar con los modeladores
	Adaptaciones	Adaptaciones de la aplicación a los requisitos de los empleados. Cambios menores a nivel de interfaz y código para facilitar el uso de la aplicación
Animaciones	Documentación y análisis	Análisis de los diferentes tipos de animación en 3DStudio y Unity3D. Estudio de las adaptaciones
	Exportación de luces	Exportación de animaciones en 3DStudio. Formato FBX.
	Importación de luces	Configuración en Unity3D de las luces importadas. Aplicación de las equivalencias
	Ejecución y depuración	Pruebas de importación y exportación con materiales. Integración en las diferentes plataformas
	Puesta en común. Requisitos	Puesta en común del trabajo realizado. Solicitud de requisitos/detalles a implementar con los modeladores
	Adaptaciones	Adaptaciones de la aplicación a los requisitos de los empleados. Cambios menores a nivel de interfaz y código para facilitar el uso de la aplicación
Configuración de Unity3D – FBX Importer	Documentación y análisis	Análisis del plugin FBX de Unity3D. Lectura de documentación y posible solución para configurarlo vía código
	Plug-in de configuración	Plug-in de configuración general para el importador FBX de Unity3D.
	Ejecución y depuración	Pruebas de importación con diferentes configuraciones. Integración con los menús de Unity3D
	Puesta en común. Requisitos	Puesta en común del trabajo realizado. Solicitud de requisitos/detalles a implementar con los modeladores
	Adaptaciones	Adaptaciones de la aplicación a los requisitos de los empleados. Cambios menores a nivel de interfaz y código para facilitar el uso de la aplicación

Tabla 1. Cronograma de tareas. Incrementos

Como se puede observar, se hace un recorrido iterativo sobre todas las características principales de una escena 3D, realizando cada vez exportaciones e importaciones más completas: Objetos, Materiales, Luces, Cámaras y Animaciones. Por último, se realiza un pequeño plugin para facilitar al usuario el *setup* de algunos aspectos de configuración en Unity3D.

3. Tecnologías y Herramientas

3.1.Entorno de Trabajo: Windows XP, 3D Studio MAX, Unity 3D

Conviene destacar que el entorno de trabajo en el que ha sido desarrollado el presente proyecto se compone de diferentes soluciones software propietarias[13], cuyas licencias han sido proporcionadas por la empresa *Neotecno Desarrollos S.L*. El sistema, enfocado a un uso profesional, cuenta con un procesador de 64 Bits, y una placa madre que soporta nativamente dicha configuración. Como sistema operativo, se ha utilizado *Windows XP x64*, el cual en entornos empresariales es más utilizado. Para el modelado de entornos 3D y su configuración, se ha utilizado *Autodesk 3D Studio MAX 2008 x64*, líder en el mercado junto a *Maya* y *xsi* (todos ellos propiedad de *Autodesk*)[14] y por último, *Unity 3D* 2.61 ha sido el motor de juegos elegido debido a su filosofía basada en la interacción gráfica con los recursos, y a su carácter multiplataforma.

3.1.1. Windows XP x64

Lanzado al mercado en Octubre del año 2001 y traducido a 92 idiomas, *Windows XP* es hasta la fecha la versión de Windows más utilizada de la historia. Actualmente se estima que abarca casi el 60% de la cuota de mercado llegando a existir 400 millones de copias funcionando. Más concretamente, se ha utilizado la versión *XP Professional 64 Bits Service Pack* 2.

Figura 18. Logo de Windows XP

Fuente: microsoft.com

La principal ventaja de utilizar un SO de 64 bits es que soporta mucha más memoria (tanto RAM como virtual) que las versiones de 32 bits. Esto es debido a la cantidad de memoria que se permite direccionar con n Bits. Para un vector de n Bits (BUS de sistema de n Bits), se puede direccionar un máximo de 2^n bytes. El máximo teórico por tanto para 32 Bits es de 4GBytes, cifra que en entornos domésticos es más que suficiente, pero que se queda

[13] Software propietario: http://es.wikipedia.org/wiki/Software_propietario

[14] Autodesk: http://es.wikipedia.org/wiki/Autodesk

escasa en entornos profesionales. Para 64 Bits existe un máximo teórico mucho mayor, pero el efectivo, en la versión que nos ocupa es de 16GBytes[15].

Teóricamente, el funcionamiento de las aplicaciones de 32 Bits es similar al de un computador de 32 Bits, y únicamente se obtendría una mejora en el rendimiento con aplicaciones especialmente diseñadas para funcionar bajo sistemas de 64 Bits.

La utilización de este un entorno Windows viene impuesta por el sector profesional para el que va destinado el proyecto. La producción audiovisual basada en 3D es un campo muy cerrado que cuenta con un número de herramientas muy limitado, donde la mayoría únicamente son compatibles con entornos Windows.

3.1.2. Autodesk 3D Studio MAX 2008 x64

Aparecido en 1990 para sistemas MS-DOS bajo el nombre de *3D Studio, Autodesk 3DS MAX* es uno de los programas de creación de gráficos y animación 3D más utilizado en la actualidad. Se utiliza sobre todo en la creación de contenido para videojuegos, pero también cubre otras disciplinas, como la animación o la arquitectura. Concretamente, se utilizará la versión de 64 Bits la cual otorgará un mayor rendimiento respecto a su homónima de 32. Sus principales ventajas son un gran repertorio de herramientas que otorgan una gran capacidad de edición, y sobre todo, una arquitectura de plugins que lo hacen adaptable a las exigencias de cualquier usuario.

Es precisamente ésta capacidad la cual será aprovechada en el presente proyecto en pos de establecer una interacción directa a bajo nivel que resuelva la problemática planteada.

Figura 19. Logo de Autodesk 3DS MAX

Fuente: autodesk.com

Desde el punto de vista de la programación, *3DS MAX* funciona con *MAXScript* (ver Apartado 3.3.3), un lenguaje de scripting que hace uso de la API proporcionada por Autodesk, la cual da acceso a gran cantidad de controles de usuario, funciones y propiedades de los objetos y la escena 3D. Incluye una herramienta de edición de scripts llamada *MXS SciTE* (basada en *Scintilla* y *SciTE*[16]).

[15] Windows de 32 y 64 Bits: Preguntas más frecuentes – Documentación web oficial

[16] **Scintilla** es un componente libre de edición de código fuente. **SciTE** un editor de texto libre con muchas características útiles para hacer programas simples.

3.1.3. Unity3D 2.61

Unity3D es un potente motor gráfico de reciente aparición (la primera versión fue publicada en Junio de 2005).

Figura 20. Logo de Unity3D

Fuente: http://unity3d.com

Sus principales características, las cuales lo diferencian de sus competidores directos (como el *Unreal Engine*[17]) y han impulsado a *Neotecno Desarrollos* a utilizarlo como principal software de creación de contenido 3D interactivo, son su filosofía de trabajo visual con los recursos, prescindiendo de complicadas interfaces y sistemas de menús, su carácter multiplataforma que permite desarrollar juegos para MAC, Windows, y dispositivos móviles de Apple (iPad/iPhone) y su precio, ya que cuenta con una versión gratuita *Indie*, ideal para desarrolladores independientes o amantes del género y una *Pro*, de pago, para uso comercial.

En el lado de la programación, *Unity3D* utiliza *Mono*[18], la implementación open-source del framework .NET, por lo que soporta *Javascript*, *C#* y *Boo*. Todos ellos son lenguajes interpretados de gran potencia, con un rendimiento sobresaliente, y gran facilidad de programación. Unity incluye una herramienta de edición de scripts llamada *Uniscite* (basada en *Scintilla* y *SciTE*), y también permite la integración con Visual Studio.

Otras características de *Unity3D* son:

- Gestor de recursos avanzado, que importa de manera automática los elementos presentes en la jerarquía del proyecto
- Implementación de *OpenGL* para representación de gráficos
- Soporte para *shaders*, e implementación de *ShaderLab* para su creación y configuración
- Exportación a contenido web, que podrá ser reproducido gracias al plugin propietario
- Implementación del motor de físicas *PhysX* de *Nvidia*

[17] Página oficial: http://www.unrealtechnology.com/

[18] Página oficial: http://www.mono-project.com/Main_Page

3.2. Herramientas de desarrollo software: Visual Studio, MXS ProEditor, UniSciTe, BOUML

El proyecto ha sido realizado en diversos lenguajes de programación (*ver Apartado 3.3*), y por tanto se ha hecho necesario utilizar diferentes herramientas de desarrollo para cada uno. A continuación se detallan las soluciones con las que se ha trabajado.

3.2.1. Visual Studio 2008 Professional

Una de las opciones disponibles para desarrollar software en *Unity3D*, es utilizar el lenguaje C#. El núcleo principal del plugin desarrollado para este motor de juegos ha sido programado en dicho lenguaje, utilizando el entorno de programación *Visual Studio 2008 Professional*.

Este IDE[19] es propiedad de Microsoft y permite trabajar nativamente con todos los lenguajes pertenecientes al framework .NET[20], y mediante extensiones con algunos creados por compañías de terceros.

La versión *Professional* es de pago, al contrario que la *Express*, pero es la utilizada en la empresa donde se realiza el proyecto, y por tanto la que ha sido elegida a tal fin.

3.2.2. MAXScript ProEditor

MXS Scite es una versión adaptada de *Scite*, un editor de textos open-source y gratuito.

Con una interfaz rediseñada, permite una funcionalidad inteligente para el manejo de código, ahora con deshacer múltiple, plegamiento, apertura rápida de documentos complejos, buscador de expresiones regulares y muchas otras características[3]

3.2.3. UniSciTe

Al igual que *3DS MAX*, *Unity3D* incorpora su propio editor de textos modificado, y proveniente de *SciTe*: *Uniscite*.

La interfaz y funcionalidad es similar al editor arriba comentado.

3.2.4. BOUML

BOUML es una herramienta CASE[21] destinada a la creación de diagramas UML. Es open-source, y gratuita, y ha sido utilizada en la fase de diseño para representar diferentes diagramas referentes a la solución del problema desarrollada en el presente proyecto.

[19] Integrated Development Enviroment: Entorno de desarrollo Integrado
[20] Framework .NET: http://es.wikipedia.org/wiki/Microsoft_.NET
[21] **C**omputer **A**ided **S**oftware **E**ngineering: Ingeniería del Software Asistida por Ordenador

3.3.Lenguajes interpretados y formatos: MaxScript, C#, Javascript, XML y FBX

3.3.1. Lenguajes Interpretados

En el presente proyecto se han utilizado diversos lenguajes de programación para implementar la solución propuesta al problema planteado.

Dichos lenguajes son en su totalidad lenguajes interpretados. La principal diferencia entre éstos y los lenguajes compilados, es que en los entornos tradicionales un compilador toma el programa fuente y lo traduce al lenguaje que puede ejecutar directamente la máquina. En el caso de *.NET* (o *Mono*) el programa se compilará o traducirá a un lenguaje intermedio llamado *MSIL*. Un intérprete de ese lenguaje intermedio *MSIL* leerá el código y ejecutará ese código intermedio dentro de un sistema controlado para que cumpla ciertas restricciones de seguridad.

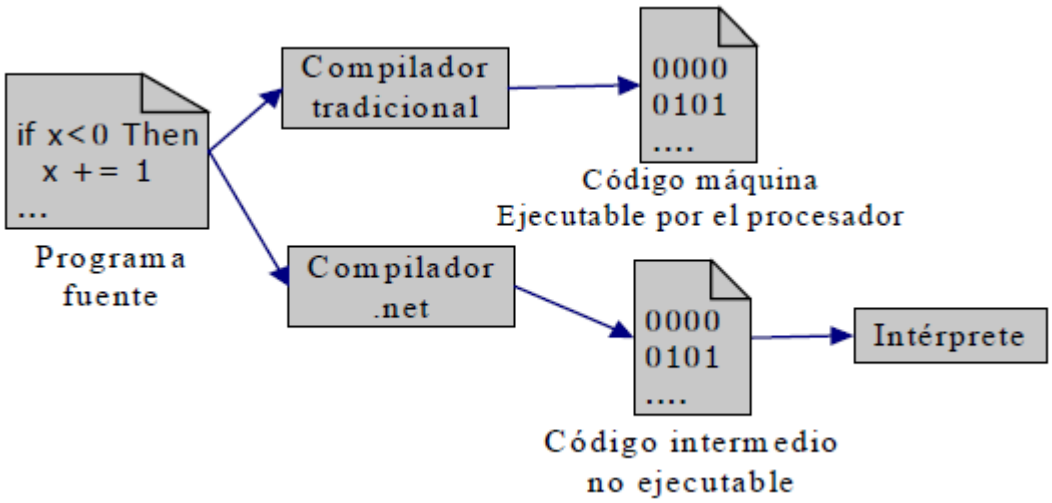

Figura 21. Lenguajes interpretados

Fuente: [4]

3.3.2. C#

C# podría considerarse un lenguaje semi-compilado. Una vez el programador ha escrito la aplicación, el compilador traduce el código fuente a un lenguaje intermedio (MSIL[22]) que luego es interpretado por la maquina virtual de .Net. En nuestro caso, la máquina virtual utilizada por *Unity3D* para interpretar el código intermedio será *Mono* (*ver Apartado 3.1*)

[22] **M**icro**s**oft **I**ntermediate **L**anguage

"Microsoft Visual C# 2005 es un lenguaje de programación diseñado para crear una amplia gama de aplicaciones que se ejecutan en .NET Framework. C# es simple, eficaz, con seguridad de tipos y orientado a objetos. Con sus diversas innovaciones, C# permite desarrollar aplicaciones rápidamente y mantiene la expresividad y elegancia de los lenguajes de tipo C.

Visual Studio admite Visual C# con un editor de código completo, plantillas de proyecto, diseñadores, asistentes para código, un depurador eficaz y fácil de usar, además de otras herramientas. La biblioteca de clases .NET Framework ofrece acceso a una amplia gama de servicios de sistema operativo y a otras clases útiles y adecuadamente diseñadas que aceleran el ciclo de desarrollo de manera significativa".[23]

En el presente proyecto, C# será utilizado para la realización del plugin de importación de objetos en *Unity3D*, conformando así el núcleo del software desarrollado para el motor de juegos.

3.3.3. MaxScript

MaxScript es un lenguaje de scripting o interpretado orientado a objetos, y de sencilla sintaxis. Implementa una API de *3DStudio* para trabajar con todo tipo de variables y objetos relacionados con el programa.

Como ejemplo de programa básico de *MaxScript*, se podría crear un cubo en una escena tridimensional, cambiarlo de color, rotarlo y hacerlo desaparecer, sin ningún tipo de click, únicamente mediante código.

Gracias a su editor visual, permite realizar aplicaciones de interfaz y plugins que se integran con *3DS MAX*, y permiten ampliar la funcionalidad de la suite 3D.

Como detalles sobre este lenguaje, se puede decir que implementa variables sin tipo, y no hay distinción entre mayúsculas y minúsculas. Cuenta con una sintaxis propia, pero similar a otros lenguajes como por ejemplo *Javascript*.

Dentro del presente proyecto, se utilizará *MAXScript* para realizar el plugin de exportación de escenas 3D desde *3DS MAX*.

Conviene destacar que *MAXScript* puede utilizar librerías *.NET*, ampliando así sus posibilidades de forma exponencial. Esta funcionalidad ha sido utilizada en el plugin de exportación, para mostrar un componente que lista los objetos de la escena, y permite al usuario seleccionar un determinado número de ellos realizando exportaciones selectivas

[23] Fuente: Documentación microsoft MSDN

3.3.4. Javascript

Javascript es un lenguaje de script orientado a objetos y no tipado. Se utiliza principalmente integrado en navegadores web para el desarrollo de interfaces de usuario y páginas web dinámicas. Cuenta con una sintaxis similar al lenguaje de programación Java.

Su origen data de 1995 por parte de la empresa *Netscape*. En 1997 se adaptó para implementar el modelo estándar *ECMAScript*[24] y acabó autodefiniéndose como un dialecto de éste.

Unity3D cuenta con una versión adaptada de *Javascript*, basada a su vez en la especificación *ECMAScript* denominada informalmente *UnityScript*[25]. Formalmente, dentro de *Unity3D*, será *Mono* quien a través de una extensión se encargue de la compilación e interpretación del código escrito en este lenguaje. Es por ello que el código intermedio será el mismo que el utilizado en la interpretación de *C#*, es decir *MSIL*.

Algunas de las principales características de esta versión de *Javascript* pueden ser encontradas en el siguiente enlace.

Dentro del presente proyecto, *Javascript* ocupa una posición menor respecto a *MAXScript* y *C#*, siendo utilizado para realizar una pequeña utilidad dentro de *Unity3D* que definirá ciertos parámetros de configuración.

3.3.5. XML

XML es un lenguaje de marcado extensible (*XML* - Extensible Markup Language).

Un lenguaje de marcado es una forma de codificar un documento que, junto con el texto, incorpora etiquetas o marcas que contienen información adicional acerca de la estructura del texto o su presentación.

XML fue desarrollado por el *World Wide Web Consortium (W3C)*, y propuesto como un estándar para el intercambio de información estructurada entre diferentes plataformas, con posibilidad de ser usado en bases de datos, editores de texto, en hojas de cálculo, aplicaciones web, etc. Por otro lado, también se define como un metalenguaje extensible que permite definir la gramática de otros lenguajes específicos.

---

[24] ECMAScript: http://es.wikipedia.org/wiki/ECMAScript

[25] Algunas de las principales características de esta versión de Javascript pueden ser encontradas en el siguiente enlace

En el presente proyecto, *XML* será el lenguaje en el cual se almacene la información de todos los objetos de una escena al ser ésta exportada. De esta forma, y gracias a los parsers[26] incorporados en la plataforma *.NET*, la información podrá ser escrita desde *3DS MAX*, y ser leída desde *Unity3D* de forma totalmente transparente.

3.3.6. FBX

Autodesk FBX es un formato de creación e intercambio de información en 3D, independiente de la plataforma y gratuito, que proporciona acceso al contenido 3D de la mayoría de los proveedores de 3D. El formato de archivo FBX admite los principales elementos de datos 3D, así como elementos 2D, de audio y de medios de vídeo.

Algunas de las principales características de la escena soportadas por FBX[27] son [6]:

- Polígonos, Mallas y NURBS
- Mapping de texturas y materiales
- Normales de vértices
- Constraints de enlace y forma
- Cámaras
- Luces
- Animación FK[28] e IK[29]

Dentro del presente proyecto, *FBX* será utilizado como formato de intercambio de datos entre las dos aplicaciones principales. *3DS MAX* exportará la escena en este formato, mientras que *Unity3D* lo leerá para importar los datos adecuadamente.

[26] Parser - Analizador Sintáctico: http://es.wikipedia.org/wiki/Analizador_sint%C3%A1ctico

[27] FBX - Página Oficial: http://usa.autodesk.com/adsk/servlet/pc/index?id=6837478&siteID=123112

[28] Forward Kinematic: Cinemática Directa

[29] Inverse Kinematic: Cinemática Inversa

3.4.Librerías utilizadas

3.4.1. Integración .NET en MAXScript

MAXScript en su versión 9, implementa integración con librerías .NET.

Formalmente, *MAXScript* expone dentro del código las propiedades, variables y métodos de los diferentes elementos de *.NET* mediante reflexión[30], de manera que puedan ser utilizadas como si de un elemento nativo se tratara.

Los elementos utilizados desde código son los siguientes[31]:

- *dotNetClass*
- *dotNetObject*
- *dotNetControl*
- *dotNetMethod*
- *dotNetMXSValue*

Como ejemplo ilustrativo, sería posible implementar una librería matemática simple en Visual C#, para posteriormente acceder a ella desde una función en *MAXScript*, y aprovechar así su funcionalidad.

En el presente proyecto, se reutilizarán controles de interfaz .NET para hacer más rica la interfaz del plugin en *3DS MAX*.

Para poder utilizar esta funcionalidad, es necesario tener instalado en la máquina un framework .NET compatible que se encargue de la compilación e interpretación del código.

Como consecuencia colateral de la utilización de esta característica, la primera vez que se cargue o ejecute código que haga utilización de la misma, el compilador deberá realizar la conversión del código intermedio a código máquina, causando un impacto negativo en el tiempo de ejecución. En ejecuciones posteriores dentro de la misma sesión, el compilador usará los binarios cacheados y por tanto el usuario no deberá esperar tanto a que se ejecute su aplicación como en la primera ejecución.

---

[30] Definición: Reflexión

[31] Documentación oficial: dotNet in MAXScript

3.4.2. Librerías UnityEngine y Mono

Debido a que *Unity3D* basa su funcionamiento a nivel de código en *Mono*[32] (*ver Apartado 3.1*), las librerías y clases básicas utilizadas son aquellas proporcionadas por éste.

Por otro lado, Unity3D cuenta con dos librerías propias, que aportan toda la funcionalidad añadida a *C#* o *Javascript*: *UnityEngine* y *UnityEditor*:

- *UnityEngine*[33] como su propio nombre indica, contiene toda la funcionalidad relacionada directamente con el motor de juego (engine). Esto es, interacciones con los diferentes objetos, movimiento de cámaras, cambios entre escenas y niveles, vídeo, audio, etc.

- *UnityEditor*[34], complementa al motor añadiendo funcionalidad relacionada con la interfaz; creación de menús, controles visuales, etc.

Mientras que el uso de *Mono* es implícito a nivel de programa, ya que se hace uso de sus recursos obligatoriamente, *UnityEngine* y *UnityEditor* se usan de forma más selectiva.

La librería *UnityEditor* es utilizada en la realización de la interfaz del plugin de importación, así como en la del pequeño programa de configuración. Por su parte lalibrería *UnityEngine* es utilizada para implementar el resto de la funcionalidad. En apartados posteriores (*ver Apartado 5*) se hará hincapié en cómo se recurre a sus métodos para realizar el código correspondiente).

[32] Documentación oficial: *Mono*

[33] Documentación oficial: *UnityEngine*

[34] Documentación oficial: *UnityEditor*

4. Diseño del Sistema

4.1.Problema planteado

En un entorno de trabajo real la productividad es algo que se valora muy positivamente. La capacidad de aprovechar el tiempo de trabajo al máximo, de manera que se puedan ir cumpliendo objetivos conforme a los plazos y estimaciones establecidas es una condición necesaria para que una empresa funcione correctamente.

Dentro de *Neotecno Desarrollos S.L*, y debido a que han comenzado a trabajar con motores de juegos, surgió un problema que afectaba negativamente a la productividad: en cada proceso de paso de información entre un programa de creación de contenido 3D y un motor de juegos, había que realizar engorrosas configuraciones, las equivalencias obtenidas no eran exactas e incluso había pérdida de información.

El problema que se plantea por tanto, es agilizar el flujo de trabajo de manera que se pierda la mínima cantidad de tiempo en estas tareas mejorando el proceso en la medida de lo posible.

El proyecto solución propuesto, consiste en la realización de dos plugins orientados al intercambio de información entre dos suites de diseño gráfico diferentes: *3D Studio MAX*, programa de modelado gráfico, y *Unity3D*, motor de creación de juegos y mundos virtuales.

Por un lado se deberá programar un plugin para *3D Studio MAX 2008* que exporte los datos pertinentes de todos los objetos de la escena guardando la información, tanto en un archivo XML principal, como en diversos archivos secundarios (FBX, imágenes).

Por el otro, se deberá programar un plugin para *Unity3D* el cual lea dicho archivo XML (y demás archivos) e importe en su propia escena el modelo cargado desde 3D Studio.

Se plantean por tanto dos bloques principales bien diferenciados, sobre los que se puede trabajar de manera independiente.

Como se ha indicado anteriormente (*ver Apartado 2.2*) la metodología adoptada será de tipo incremental. Habrá que tener en cuenta que debido a que el software será desarrollado en vistas a ser utilizado en proyectos de *i+D* posteriores dentro de la empresa, existirán puntuales análisis de requisitos realizados de forma conjunta con los empleados que vayan a entrar en contacto directo con la aplicación.

4.2.Diagrama general de la aplicación

La funcionalidad que se quiere implementar tiene un fuerte carácter secuencial, donde en todo caso de uso se ejecutará de la misma manera:

Figura 22. Secuencia de operaciones

En primer lugar, se guarda la información deseada. Dicho guardado se realiza en forma de texto sobre un XML, y también por el plugin en formato FBX.

Una vez haya terminado dicho guardado, se procederá a la importación de dichos datos por parte del usuario dentro del entorno *Unity3D*. El plugin correspondiente a ésta funcionalidad se encargará de leer adecuadamente el archivo XML y acceder a los documentos en forma correcta, de manera que la operación sea lo más transparente posible.

A continuación se muestra un diagrama que ilustra de manera visual el conjunto del trabajo:

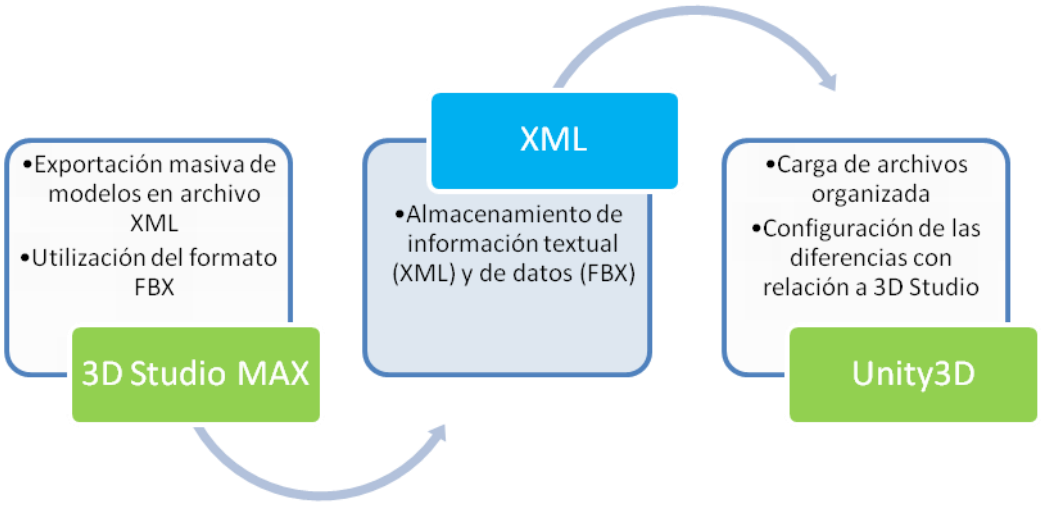

Figura 23. Diagrama general de la aplicación

Se observa claramente que el proyecto tiene una doble dirección, por un lado el plugin desarrollado para la plataforma 3DStudio, y por otro el referente al motor Unity3D.

Cabe destacar en este punto que las características a exportar se pueden dividir en 5 bloques bien diferenciados:

- Objetos
- Luces
- Cámaras
- Materiales y texturas
- Animación

En la metodología incremental adoptada para la implementación de estos 5 bloques se ha optado por desarrollar ambas herramientas a la vez, comenzando por implementar una funcionalidad básica para luego ir evolucionándolas de manera que "crezcan" al mismo tiempo y se vayan probando sus diferentes características de forma escalonada.

Por tanto, se comenzará por el bloque de Objetos, siguiendo por Materiales, Luces, Cámaras y por último, Animación.

4.3.Diagrama Conceptual

A continuación se muestra un diagrama conceptual del proyecto a realizar. Este diagrama es denominado modelo conceptual o del dominio, y se define como la representación visual de los objetos conceptuales del mundo real dentro del dominio de nuestra aplicación.

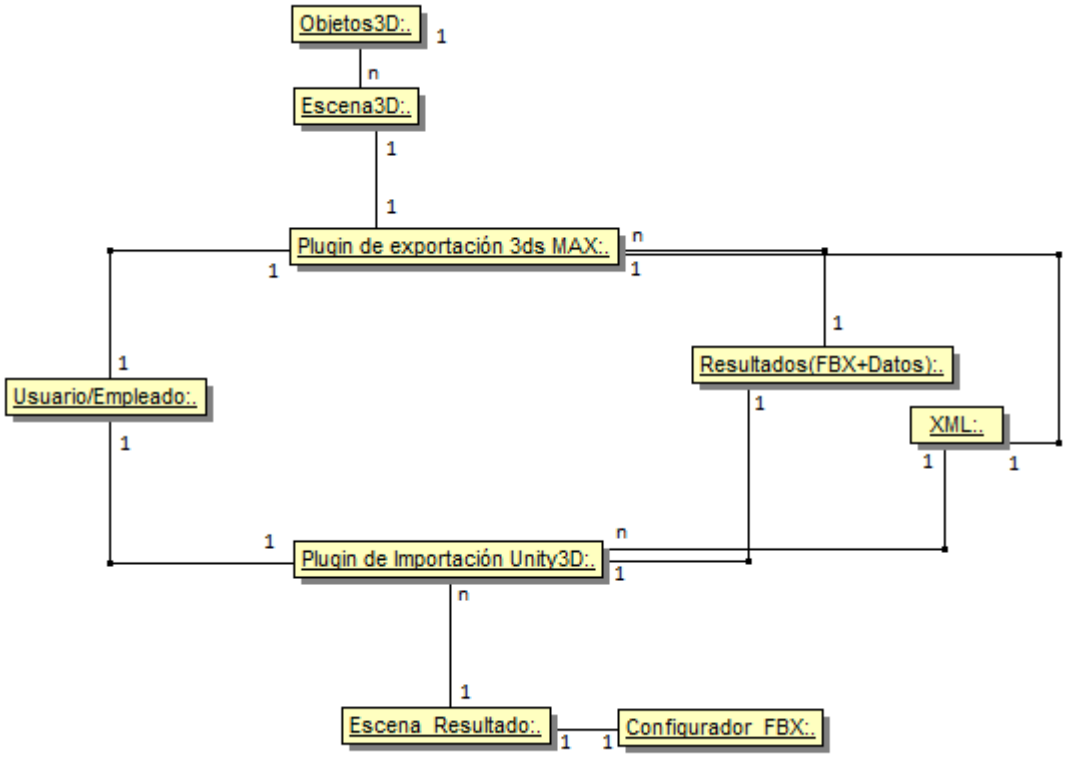

Figura 24. Diagrama conceptual

A nivel general, podemos observar como un usuario puede utilizar una única instancia a la vez de cada plugin. De esta manera, el plugin de *3DS MAX*, podrá exportar una única escena por cada uso, la cual podrá contar con *n* elementos, siendo *n*>=0. En cada exportación se generará un único archivo XML, así como n archivos de datos a importar. Desde el plugin de importación, se analizará toda esa información para crear una única *Escena_Resultado*, la cual teóricamente debería ser igual que la *Escena3D*.

Por otro lado, se puede observar que la *Escena_Resultado* se puede configurar gracias a una instancia única del configurador FBX.

4.4.Diagrama de clases

A continuación se muestra el diagrama de clases general para la aplicación. Hay que tener en cuenta la heterogeneidad del proyecto, por lo que se muestra una versión estándar de cada clase, que no necesariamente guarda una estructura formal respecto a sintaxis. Por otro lado, se han ocultado atributos y métodos secundarios cuya presencia es mínima y por tanto totalmente prescindible en una perspectiva general:

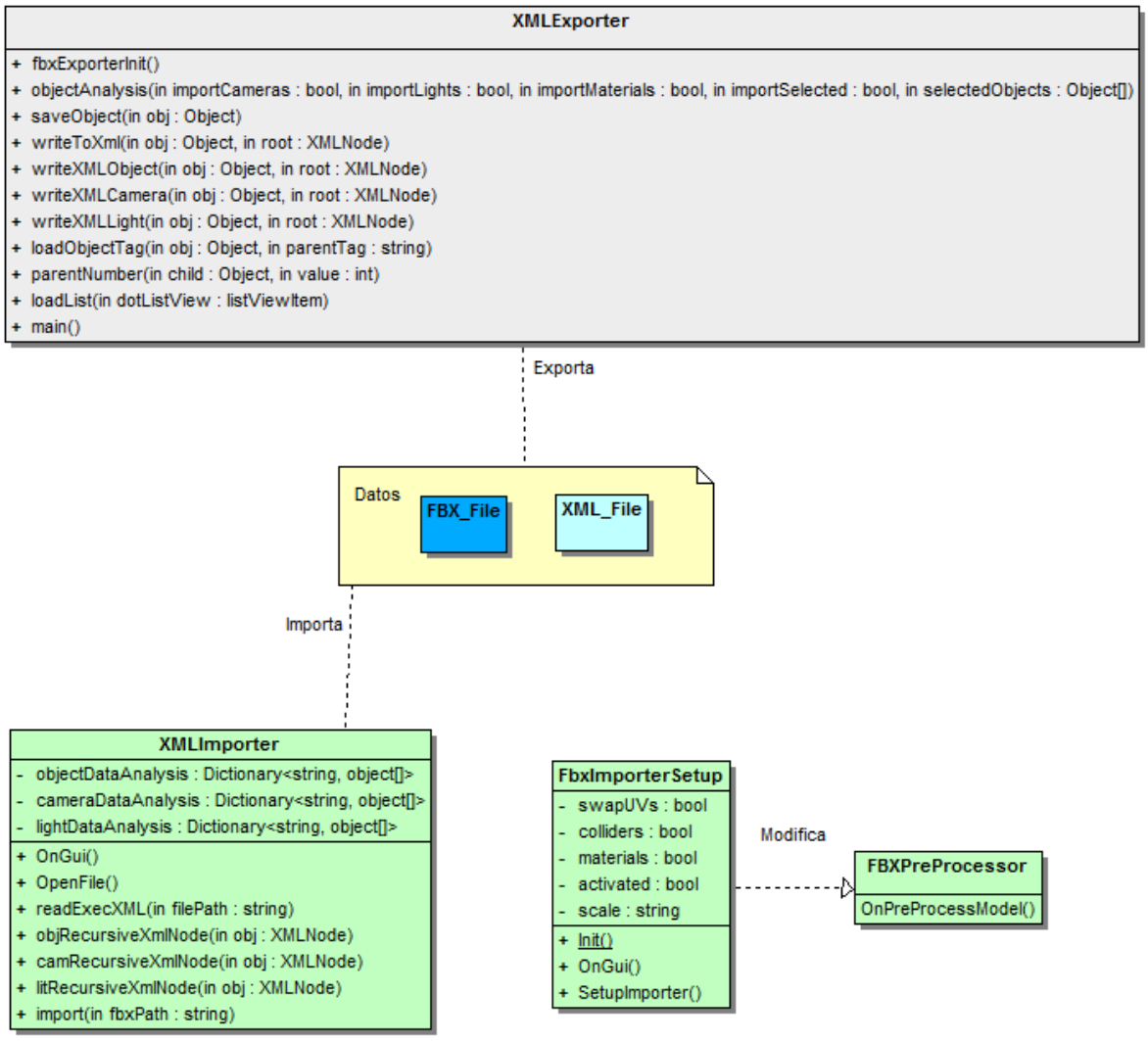

Figura 25. Diagrama de clases

En gris se observa la clase referente al plugin de *3DS MAX XML Exporter*. En verde se representan las clases relacionadas con *Unity3D*, tanto el plugin *XML Importer*, como la utilidad de configuración y su respectivo archivo de parámetros.

4.5.Casos de Uso

4.5.1. Esquema general

Si consideramos la aplicación como un diseño unificado, sin hacer distinción entre los dos plugins que se van a implementar, se pueden distinguir cuatro casos de uso:

- ***Exportación parcial:*** Que permite al usuario exportar parte de los objetos de una escena
- ***Exportación completa:*** Que permite al usuario exportar la escena completa
- ***Importación de datos:*** Que importa, desde un XML con su correspondiente información, la escena dentro del motor de juegos
- ***Configuración del importador FBX:*** Que mediante una interfaz, permite al usuario realizar los ajustes del importador dentro del motor de juegos.

Como actores, contaríamos con el usuario que ejecuta la aplicación, y con la misma aplicación, encargada de llevar a cabo las tareas que se le solicitan.

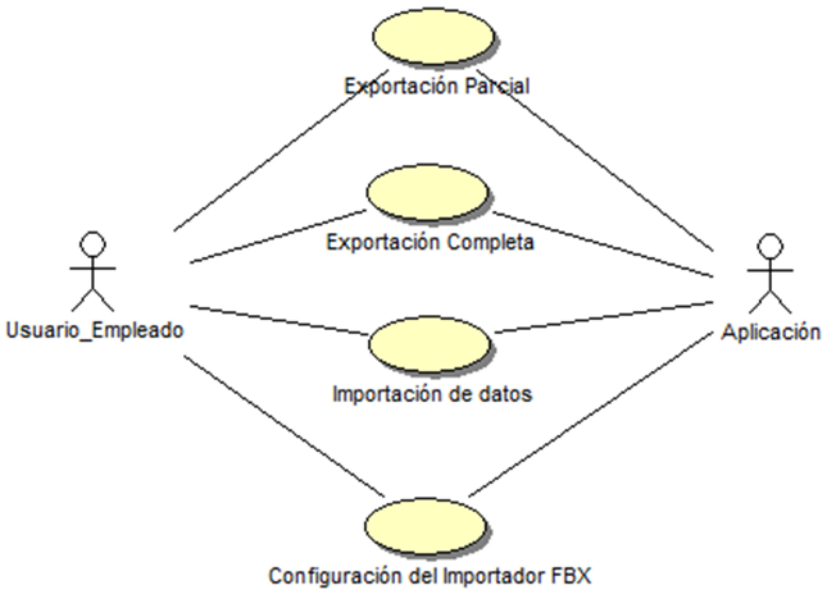

Figura 26. Esquema general de casos de uso (Aplicación unificada)

Por otro lado, podríamos hacer distinción dentro de nuestra aplicación entre los tres módulos que van a conformar la solución final, de manera que como actor primario seguiríamos considerando al usuario, pero encontraríamos 3 actores secundarios: el plugin de *3DS MAX*, el plugin de *Unity3D* y la pequeña utilidad de configuración del importador dentro de *Unity3D*.

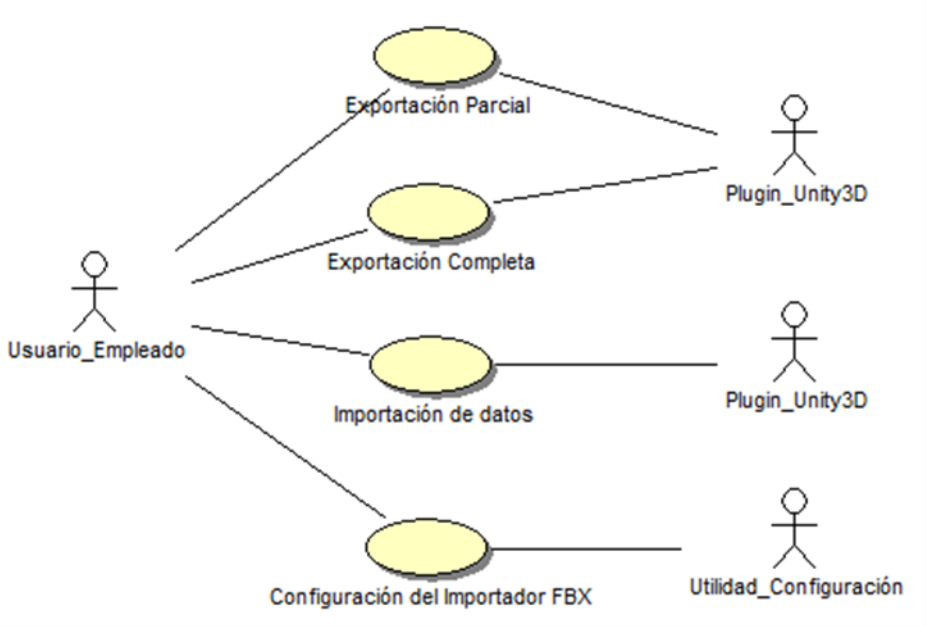

Figura 27. Esquema general de casos de uso (Módulos de la aplicación)

4.5.2. Casos de uso detallados

4.5.2.1. Exportación Parcial

Nombre: Exportación Parcial
Descripción: Permite al usuario exportar parte de una escena a formato XML y FBX desde *3DS MAX*
Actores: Usuario. Normalmente un artista/modelador/programador.
Precondiciones: El usuario debe tener *3DS MAX* abierto, el plugin instalado, y la escena que quiere exportar en el viewport
Escenario base: 1.- El usuario abre el plugin 2.- El usuario selecciona de una lista los objetos que quiere exportar 3.- El usuario elige una ruta para el archivo XML y selecciona los parámetros de la exportación 4.- El usuario marca la casilla "Selected Only" para indicar que únicamente quiere exportar aquellos objetos que ha seleccionado 5.- El usuario elige una ruta para el archivo XML 6.- El usuario pulsa el botón de "Exportar" 7.- El sistema realiza las operaciones solicitadas y exporta la información indicada
Variantes: 5.- Es posible que la ruta seleccionada no exista, o el usuario de un nombre no válido de archivo. En ese caso la ejecución dará error 5.- Es posible que el archivo ya exista, por lo que el sistema pedirá al usuario que confirme que quiere sobreescribir el archivo 6.- Si el usuario intenta realizar una exportación selectiva, sin haber seleccionado ningún objeto de la lista, no se exportará nada, y se le mostrará un XML vacío 7.- Si el usuario selecciona los objetos a exportar, pero no marca la casilla "Selected Only" se exportarán todos los objetos de la escena
Postcondición: El sistema crea los archivos para la importación de la información en el motor de juegos

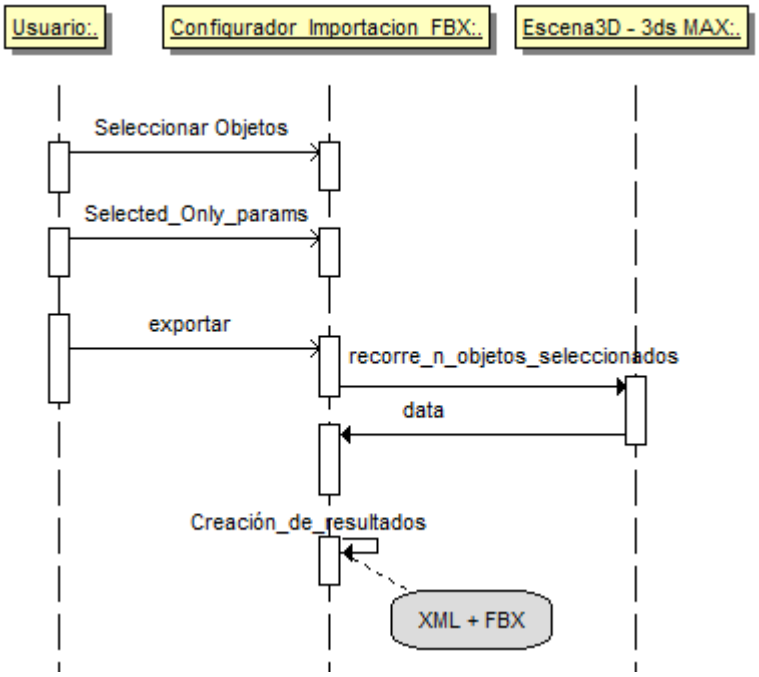

Figura 28. Diagrama de secuencia. Exportación Parcial

4.5.2.2. Exportación Completa

Nombre: Exportación Completa
Descripción: Permite al usuario exportar una escena completa en formato XML y FBX desde *3DS MAX*
Actores: Usuario. Normalmente un artista/modelador/programador.
Precondiciones: El usuario debe tener *3DS MAX* abierto, el plugin instalado, y la escena que quiere exportar en el viewport
Escenario base: 1.- El usuario abre el plugin 2.- El usuario elige una ruta para el archivo XML y selecciona los parámetros de la exportación 3.- El usuario pulsa el botón de "Exportar" 4.- El sistema realiza las operaciones solicitadas y exporta la información indicada
Variantes: 2.- Es posible que la ruta seleccionada no exista, o el usuario de un nombre no válido de archivo. En ese caso la ejecución dará error 2.- Es posible que el archivo ya exista, por lo que el sistema pedirá al usuario que confirme que quiere sobreescribir el archivo 3.- Si el usuario intenta exportar una escena vacía, no se exportará nada, y se le mostrará un XML en blanco 4.- Si el usuario marca la casilla "Selected Only", pero no ha seleccionado ningún objeto, no se exportará nada, y se mostrará un XML en blanco
Postcondición: El sistema crea los archivos respectivos a la escena completa para la correcta importación de la información en el motor de juegos

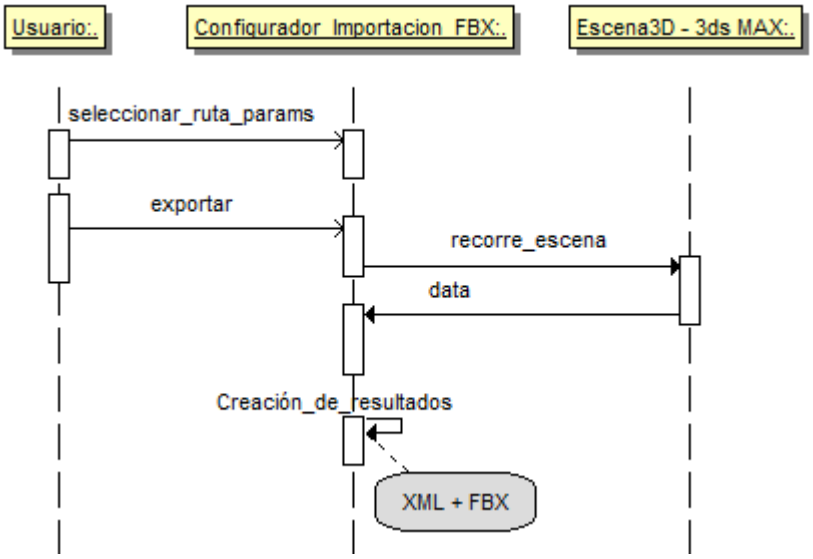

Figura 29. Diagrama de secuencia. Exportación completa

4.5.2.3. Importación de datos

Nombre: Importación de datos
Descripción: Permite al usuario importar una escena en *Unity3D*
Actores: Usuario. Normalmente un artista/modelador/programador.
Precondiciones: El usuario debe tener *Unity3D* abierto, el plugin instalado, y los archivos a importar deben estar disponibles
Escenario base: 1.- El usuario abre el plugin 2.- El usuario selecciona la ruta del archivo XML que contiene la información a importar 3.- El usuario marca las características que desea importar 4.- El usuario pulsa el botón de "Importar" 5.- El sistema realiza las operaciones solicitadas y configura la escena adecuada
Variantes: 2.- Es posible que la ruta seleccionada no exista, o el usuario de un nombre no válido de archivo. En ese caso la ejecución dará error 2.- Es posible que el archivo XML no tenga formato correcto, con lo cual saltará una excepción y el importador no hará nada
Postcondición: El sistema crea y configura la escena importada mostrándola en la vista de edición tal y como se encontraba en *3DS MAX*

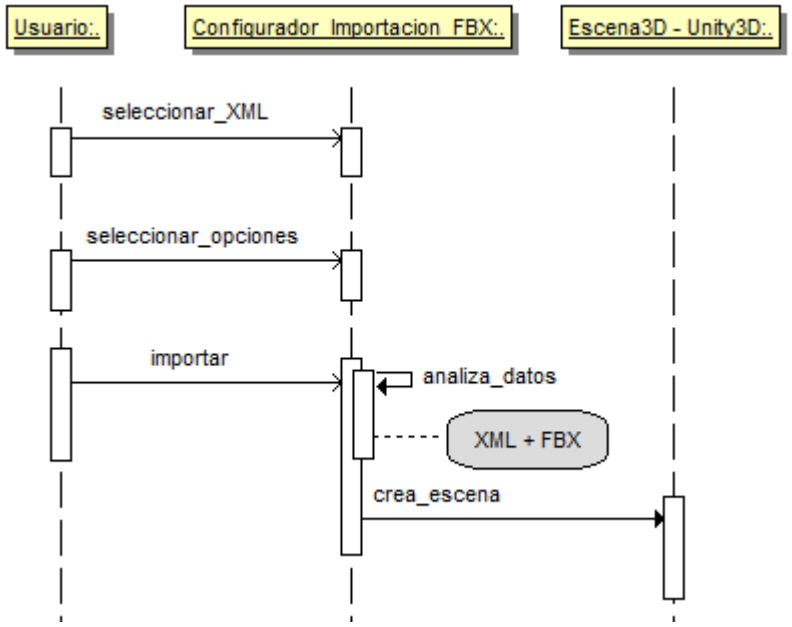

Figura 30. Diagrama de secuencia. Importación

4.5.2.4. Configuración del importador FBX

Nombre: Configuración del importador FBX
Descripción: Permite al usuario configurar los parámetros del importador FBX
Actores: Usuario. Normalmente un artista/modelador/programador.
Precondiciones: El usuario debe tener *Unity3D* abierto, y la aplicación del importador dentro de la jerarquía de directorios de su proyecto
Escenario base: 1.- El usuario abre el configurador 2.- El usuario selecciona los valores deseados para cada característica 3.- El usuario pulsa el botón de "Aplicar"
Variantes: 2.- Es posible que el usuario introduza datos en un formato incorrecto, con lo que se mostrará un mensaje de excepción, y no se realizará la configuración
Postcondición: El sistema modificará el archivo de configuración del importador FBX con los valores introducidos por el usuario

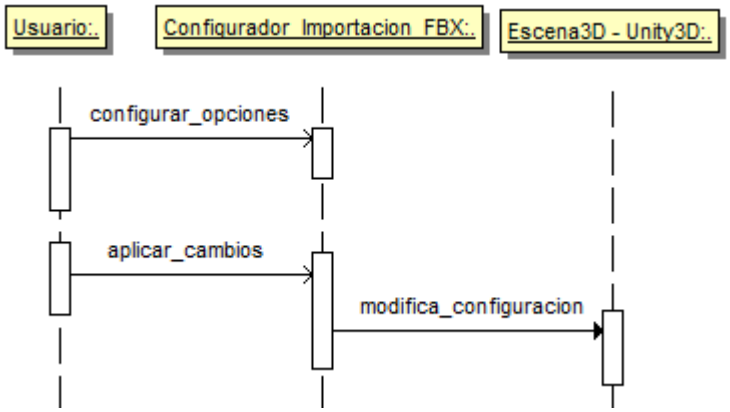

Figura 31. Diagrama de secuencia. Configuración del importador FBX

5. Resolución

A lo largo de esta sección de la documentación se hará un recorrido horizontal por la solución realizada al problema planteado. En dicha explicación se omitirán las diferentes fases (incrementos) en las que se ha dividido el proceso de trabajo, mostrando únicamente los resultados finales obtenidos.

5.1. Plugin de exportación. XML Exporter. 3DS MAX

El plugin de exportación o *XML Exporter* (nombre que se dará de ahora en adelante al plugin de *3DS MAX* realizado en el presente proyecto) conforma uno de los módulos principales del proyecto final. Está realizado completamente en *MAXScript*, y únicamente puede ser ejecutado desde *3DS MAX*.

5.1.1. Configuración del exportador: FBX

El formato FBX se ha utilizado para encapsular la información de una escena *3DS MAX* en un único archivo, y así poder ser importado éste en *Unity3D* con la mínima pérdida de información posible.

Conviene destacar que el exportador *FBX* y el *XML Exporter* son herramientas diferentes. La primera es utilizada como una parte de la segunda y de esta forma, el *XML Exporter* "contiene" al *FBX Exporter*. Su interacción se realiza vía código, y como resultado de la misma se generan parte de los archivos necesarios para la importación en *Unity3D*.

El plugin oficial de exportación en FBX, presenta una interfaz algo engorrosa, con multitud de opciones y parámetros de configuración que en la mayoría de los casos no es necesario modificar.

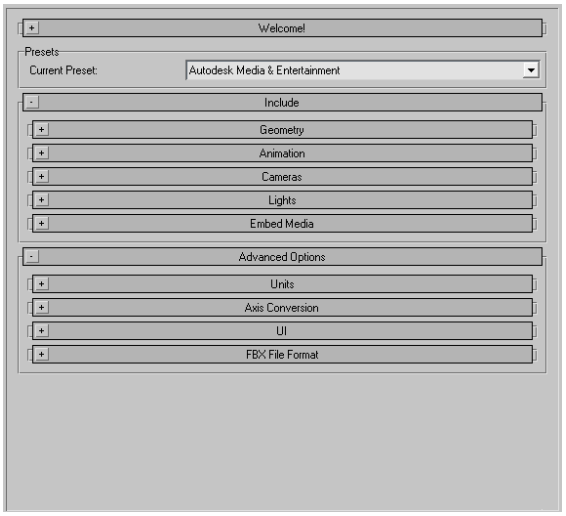

Figura 32. FBX Exporter oficial v.2010

Como solución, y debido a que muchos de estos parámetros están expuestos en el código, dentro del plugin del *XML Exporter* se han configurado algunos parámetros como fijos, mientras que otros han sido enlazados a elementos de interfaz. De esta manera se simplifica considerablemente el proceso de exportar una escena determinada, tanto si es completa, como si se selecciona un número de objetos determinado.

Los parámetros fijos que han sido configurados son los siguientes (formato **"Nombre = [Valor]"**):

- ***"FileVersion = [FBX201000]"***: Establece qué versión del exportador FBX se va a utilizar. En nuestro caso, la versión 2010 (*201000*).

- ***"Binary = [true]"***: Indica que el archivo FBX deberá ser almacenado en formato binario. La otra opción es guardarlo en formato texto, de forma que se pueda leer la información que almacena, pero esta opción es incompatible con la exportación de texturas.

- ***"Shape = [false]"***: Exporta también las formas bidimensionales de la escena

- ***"Skin = [true]"***: Activa o desactiva la exportación de deformaciones en skins

- ***"ShowWarnings = [true]"***: Indica al exportador que muestre los warnings surgidos al codificar la información

- ***"UpAxis = [Y]"***: Indica que se adopta el eje Y como vertical, de manera que la importación de los datos en un programa de 3D con otro sistema de ejes sea equivalente (ver Apartado 5.2.3).

- ***"Convert2Tiff = [false]"***: Establece que las texturas importadas sean almacenadas en su formato nativo, sin efectuar cambios en su formato, de manera que conserven todas sus propiedades.

- ***"ColladaTriangulate = [true]"***: Transforma la malla poligonal en triángulos al exportar los objetos. Esto evita problemas de compatibilidad (ver Apartado 5.2.3).

Los parámetros enlazados con elementos de interfaz son los siguientes:

- ***"Cameras = [true/false]":*** Dependiendo de su valor, indica al exportador si debe almacenar información sobre las cámaras o no

- ***"Lights = [true/false]":*** En función de su valor, se indica al exportador si debe almacenar información sobre las luces o no

- ***"EmbedTextures = [true/false]":*** En función de su valor, se indica al exportador si debe almacenar información sobre las texturas o no

- ***"Animation = [true/false]":*** En función de su valor, se indica al exportador si se quieren exportar animaciones

- ***"BakeAnimation = [true/false]":*** En función de su valor, se indica al exportador si se quieren guardar el resultado final de las animaciones, sin todo el proceso intermedio (utilizada siempre junto a `Animation`).

De esta forma, dentro del plugin de exportación de *3DS MAX*, se puede apreciar un pequeño cuadro de configuración con algunos checkbox que permiten dar valor o no a los atributos variables:

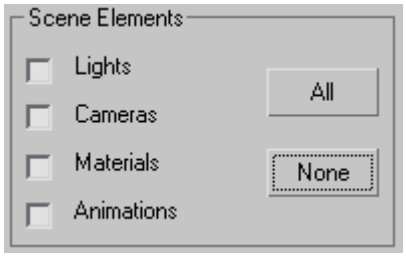

Figura 33. Cuadro de configuración del XML Exporter

Una relación completa de todos los parámetros del *FBX Exporter* modificables desde código se puede encontrar en [7].

5.1.2. Análisis de la escena y recorrido de objetos

Como ya se ha mencionado anteriormente, existen dos casos de uso para el *XML Exporter*. Por un lado, la exportación parcial, la cual permite al usuario elegir qué elementos de la escena quiere exportar. Por el otro, la exportación completa, la cual asume que será la escena completa la información que se quiere migrar a *Unity3D*.

5.1.2.1. Exportación Parcial

En este primer caso, se hace uso de un componente .NET de tipo *listViewItem*[35] en el cual aparecen listados los objetos presentes en la escena en el momento en el que el usuario ejecutó el plugin.

De esta forma, dicho componente por defecto tiene el siguiente aspecto:

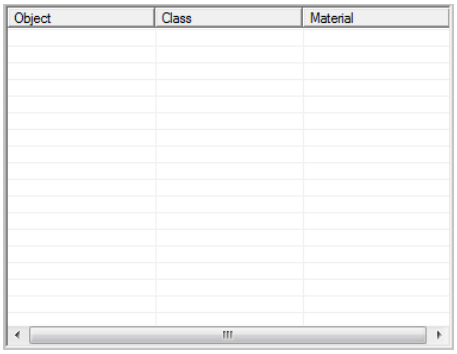

Figura 34. Componente .NET listView vacío

En caso de haber algún elemento en la escena, se mostrará el nombre del mismo, su clase, y el material que tiene aplicado.

Así, para una escena con cuatro figuras (dos cajas y dos esferas) el componente presentaría el siguiente aspecto:

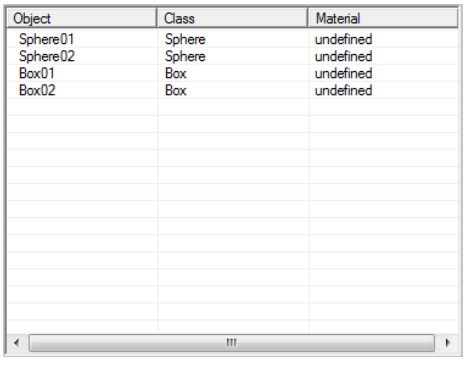

Figura 35. Componente .NET listView con elementos

[35] Documentación oficial MSDN: listViewItem

En el caso de querer realizar una exportación selectiva, el usuario deberá seleccionar en la lista aquellos elementos que quiera guardar, y marcar la opción "*Only Selected*" en la interfaz.

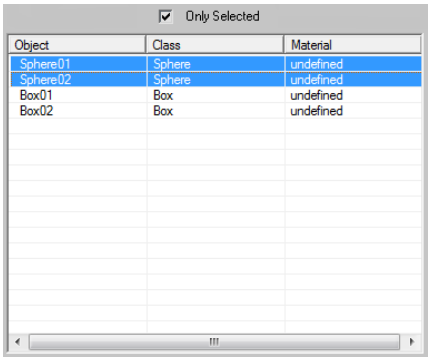

Figura 36. Componente .NET listView con elementos seleccionados. Checkbox activado

Internamente cuando esto es realizado, el código va almacenando en una estructura de tipo array los nombres de los objetos seleccionados de la lista. Posteriormente, se recorre dicho array, añadiendo en otra estructura similar todos los objetos cuyo nombre se encontraba en la lista. En una etapa posterior, se realiza una selección dentro de *3DS MAX* del último array configurado (dicha selección se puede observar en el *viewport*[36]). Finalmente al utilizar el *FBX Exporter* se le indica que exporte únicamente los objetos seleccionados, consiguiendo así el resultado que se buscaba.

Se muestra a continuación el pseudocódigo donde se puede observar este procedimiento:

```
si (only_selected = verdadero) entonces (

        --En caso de que la opción de exportar los objetos seleccionados sea marcada tendremos que tener
        en cuenta las filas marcadas en el listViewItem

        objetos = listView.SelectedItems

        objetosLim = objetos.count

        objetosLim-=1

        selectionArray = items del listView seleccionados por el usuario

        objectArray= array vacío donde se irán guardando los objetos de la
        escena seleccionados

        --Para cada fila marcada del listViewItem
```

[36] Se denomina ***viewport*** a la región de la pantalla dentro de una aplicación 3D donde se representan las figuras en 3D como si fueran vistas desde la posición de una cámara virtual

```
      para cada m en selectionArray hacer(
            --Buscamos un objeto en la escena con el mismo nombre
            obj = buscarObjetoConNombre m.name
            --Y cada objeto lo metemos en un array
            añadir en objectArray -> obj
      )
      --Seleccionamos todos los objetos contenidos en el array
      seleccionar objectArray
      --Y finalmente se exporta la selección
      exportar selección
)sino(…)
```

Pseudocódigo 1. Exportación selectiva

5.1.2.2. Exportación Completa

La exportación completa de una escena se podría considerar una versión ampliada de la exportación parcial. Lo más normal teniendo en cuenta la explicación anterior sería pensar que el usuario tendría que marcar en el *listView* todos los objetos, y realizar una exportación selectiva. Esta configuración es perfectamente funcional, pero existe una forma mucho más simple de realizar una exportación completa de la escena.

XML Exporter ha sido implementado de manera que asume por defecto que se quiere exportar la escena completa y se ejecuta en consonancia. Esto es posible gracias a las herramientas disponibles en *MAXScript*, y a la implementación del plugin *FBX Exporter*. A nivel de interfaz no es necesario que el usuario realice ningún cambio, más allá de seleccionar qué parámetros quiere que se exporten y verificar que el *checkBox* "*Only Selected*" no está marcado.

La implementación a bajo nivel de es mucho más sencilla, ya que es el propio código internamente el encargado de seleccionar todos los objetos de la escena y realizar la exportación. Así, el pseudocódigo que representa esta situación, sería el siguiente:

```
si (only_selected = verdadero) entonces (
      (…)
)sino(
      exportar escena completa
)
```

Pseudocódigo 2. Exportación completa

5.1.3. XML: Guardado de información

En la exportación de una escena 3D desde *3DS MAX*, *XML Exporter* genera un documento XML que contiene información detallada sobre cada objeto de la escena.

La ruta de dicho archivo es elegida por el usuario desde un botón de la interfaz (que al ser pulsado muestra una ventana *getSaveFileName*[37]):

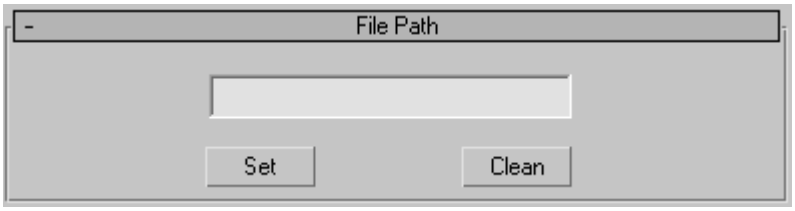

Figura 37. Control de interfaz para seleccionar la ruta del archivo XML

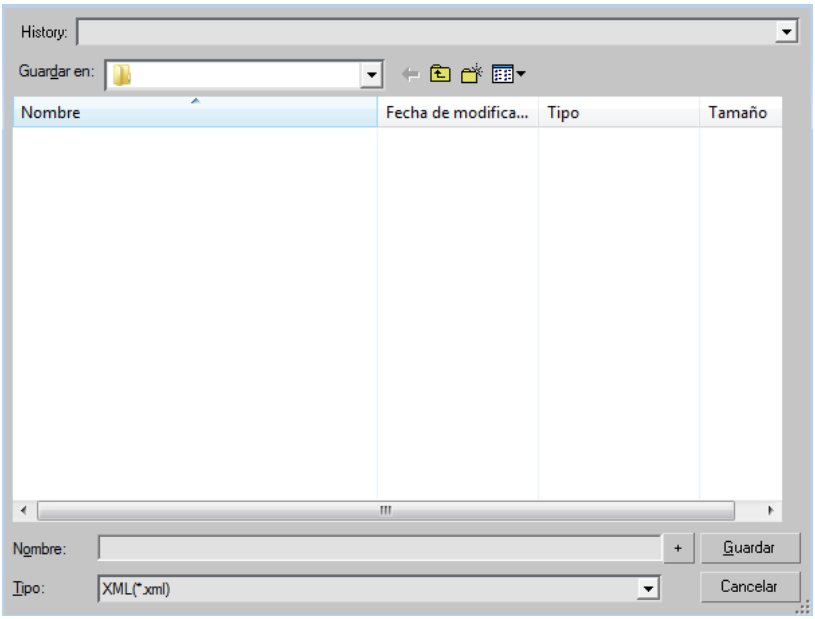

Figura 38. 3DS MAX: getSaveFileName

El formato del archivo XML es el siguiente:

```
<Scene release=[] API=[] SDK=[] FBXPath=[]>
  <Objects/>
     [0..n objects]
  <Cameras/>
     [0..n Cameras]
  <Lights />
     [0..n Lights]
</Scene>
```

Pseudocódigo 3. Estructura del archivo XML

[37] Documentación oficial: getSaveFileName

Donde el nodo raíz almacena información general sobre el API utilizado, el SDK y el *path* del FBX exportado. Dicho nodo cuenta también con tres hijos:

- `Objects`: Almacena información sobre todos los objetos de la escena. El formato de cada hijo de este nodo será el siguiente:

```
<Object Name=[] Parent=[] Color=[] Position=[] Scale=[] ClassOf=[]
SuperClassOf=[] Material=[] />
```

Pseudocódigo 4. Estructura XML - Nodo Object

- `Cameras`: Almacena información sobre todas las cámaras presentes en la escena. El formato de cada hijo de este nodo será el siguiente:

```
<Camera Name=[] Parent=[] ClassOf=[] Position=[] Scale=[] Rotation=[] FOV=[] />
```

Pseudocódigo 5. Estructura XML - Nodo Camera

- `Lights`: Almacena información sobre todas las luces presentes en la escena. El formato de cada hijo de este nodo será el siguiente:

```
<Light Name=[] Parent=[] ClassOf=[] Position=[] Scale=[] Rotation=[] FallOf=[]
Shadows=[] Distance=[] Multiplier=[] Attenuation=[] />
```

Pseudocódigo 6. Estructura XML - Nodo Light

A continuación se muestra un glosario de los diferentes atributos de cada nodo dentro del archivo XML:

Glosario de atributos del XML	
Name	Nombre del objeto
Parent	Nombre del padre del objeto
Color	Código RGB del color del objeto
Position	Coordenadas de posición del objeto
Scale	Escalado que se aplica en el objeto
ClassOf	Clase del objeto
SuperClassOf	Superclase del objeto
Material	Nombre del material que está aplicado en el objeto
Rotation	Rotación de la cámara
FOV	Campo de visión (Field of View) de la cámara
FallOf	FallOF de la luz
Shadows	Indica si la luz genera o no genera sombras
Distance	Distancia de alcance de la luz
Multiplier	Intensidad de la luz
Attenuation	Parámetro que indica si la luz tiene atenuación cercana y/o lejana

Tabla 2. Glosario de atributos en las entradas XML

La implementación de esta funcionalidad ha sido realizada gracias a un assembly .NET (ensamblado .NET[38]): ***System.XML***[39]

[38] Un ***esamblado .NET*** se define como una librería de código compilado para ser utilizado en otras aplicaciones. Enlace

[39] Documentación oficial: System.xml

De nuevo se considera un pseudocódigo la forma más directa de explicar la implementación de la escritura en XML:

```
--Se llama a la función writeToXML pasando como parámetros el objeto que se quiere analizar, y el nodo raíz del XML de donde deben colgar todos los hijos

función writeToXML objeto raíz (
	superclase = superClase de objeto
	si (superClase=="camera") entonces (
		si (objeto no tiene padre) entonces (
			writeXMLCamera objeto raíz
		)
	)
	sino si (superClase=="light") entonces (
		si (objeto no tiene padre) entonces (
			writeXMLLight objeto raíz
		)
	)
	sino (
		si (objeto no tiene padre) entonces (
			writeXMLObject objeto raíz
		)
	)
)
```

Pseudocódigo 7. Función principal, escritura en XML

Existe una función principal ***writeToXML*** que recibe como parámetros un objeto y un nodo raíz. El objeto es una entidad presente en la escena 3D y el nodo raíz el nodo padre del que tiene que colgar la información analizada dentro del archivo XML. Dentro de esta función, se comprueba de qué tipo es el objeto (geometría, cámara o luz) y se llama en cada caso a un método ***writeXML[Camera|Light|Object].*** Se puede observar que únicamente llamamos a dicha función pasando como parámetro aquellos objetos que no tengan padre ya que dentro de la función ***writeXML[Camera|Light|Object]*** se recorren todos los hijos del objeto. Si no se llevara a cabo esta comprobación, obtendríamos datos duplicados dentro del XML, ya que se realizan llamadas recursivas al método de escritura que como resultan en un avance descendente en la jerarquía de objetos hasta llegar al final.

Las funciones ***writeXML[Camera|Light|Object]*** cuentan con una estructura similar, y únicamente se diferencian en las propiedades del objeto que escriben en el archivo XML. Por ejemplo, buscar la intensidad de luz que emite una cámara no tiene sentido,

Se muestra el pseudocódigo de una función genérica que representa la estructura básica de dichas funciones (en negrita se resaltan las funciones del ensamblado XML):

```
función writeXML[…] objeto raíz (

      --Nodo principal

      Creación del nodo correspondiente node = xmlFile.CreateElement
      "Object/Cameras/Lights"

      Para cada propiedad hacer(

            Escritura de las propiedades = node.SetAttribute
            "NombreDePropiedad" valor

      )

      --Se añade el nodo como hijo del nodo raíz

      raíz.AppendChild node

      --Llamada recursiva que explorará todos los hijos del objeto, colgándolos de él mismo

      Para cada hijo de objeto hacer(

            writeXML[] hijo node

      )

)
```

Pseudocódigo 8. Función específica de cada Nodo. Escritura en XML

(*sigue en página siguiente*)

Como ejemplo, se muestra el archivo XML resultado de exportar la escena anteriormente utilizada en los ejemplos (dos esferas y dos cajas) junto con una cámara y una luz y sin tener en cuenta los materiales aplicados en los objetos:

```
<Scene release="10000" API="23" SDK="0" FBXPath="\prueba1.xml-Data\prueba1.FBX">

  <Objects>

    <Object Name="Box01" Parent="undefined" Color="[177.0,27.0,88.0]"
Position="[50.602,4.67522,0]" Scale="[1,1,1]" ClassOf="Box" SuperClassOf="GeometryClass"
/>

    <Object Name="Box02" Parent="undefined" Color="[6.0,135.0,6.0]" Position="[-
28.7497,-8.47736,0]" Scale="[1,1,1]" ClassOf="Box" SuperClassOf="GeometryClass" />

    <Object Name="Sphere01" Parent="undefined" Color="[213.0,154.0,229.0]"
Position="[17.9388,-39.7771,0]" Scale="[1,1,1]" ClassOf="Sphere"
SuperClassOf="GeometryClass" />

    <Object Name="Sphere02" Parent="undefined" Color="[8.0,61.0,138.0]" Position="[-
1.20679,57.4617,0]" Scale="[1,1,1]" ClassOf="Sphere" SuperClassOf="GeometryClass" />

  </Objects>

  <Cameras>

    <Camera Name="Camera01" Parent="undefined" ClassOf="Freecamera" Position="[4.03227,-
12.7882,0]" Scale="[1,1,1]" Rotation="[0.0,0.0,0.0]" FOV="45.0" />

  </Cameras>

  <Lights>

    <Light Name="Omni01" Parent="undefined" ClassOf="Omnilight"
Position="[64.6214,75.0504,0]" Scale="[1,1,1]" Rotation="[0.0,0.0,0.0]" FallOf="0"
Shadows="true" Distance="0" Multiplier="1.0" Attenuation="false-false" />

  </Lights>

</Scene>
```

Figura 39. XML resultado de una exportación sencilla

La implementación del plugin permite seleccionar si se quieren o no exportar materiales. De esta forma, y como se ha observado en el ejemplo inmediatamente anterior, en caso de seleccionar la no exportación de los mismos, su información no es añadida en el archivo XML.

A continuación se muestra el archivo XML generado para una escena con dos cajas, dos esferas, una cámara y una luz, con la opción de exportar materiales activada (en negrita se marca la información que antes no aparecía):

```
<Scene release="10000" API="23" SDK="0" FBXPath="\prueba2.xml-Data\prueba2.FBX">

  <Objects>

    <Object Name="Box01" Parent="undefined" Color="[177.0,27.0,88.0]"
Position="[50.602,4.67522,0]" Scale="[1,1,1]" ClassOf="Box" SuperClassOf="GeometryClass"
Material="undefined" />

    <Object Name="Box02" Parent="undefined" Color="[6.0,135.0,6.0]" Position="[-
28.7497,-8.47736,0]" Scale="[1,1,1]" ClassOf="Box" SuperClassOf="GeometryClass"
Material="undefined" />

    <Object Name="Sphere01" Parent="undefined" Color="[213.0,154.0,229.0]"
Position="[17.9388,-39.7771,0]" Scale="[1,1,1]" ClassOf="Sphere"
SuperClassOf="GeometryClass" Material="undefined" />

    <Object Name="Sphere02" Parent="undefined" Color="[8.0,61.0,138.0]" Position="[-
1.20679,57.4617,0]" Scale="[1,1,1]" ClassOf="Sphere" SuperClassOf="GeometryClass"
Material="undefined" />

  </Objects>

  <Cameras>

    <Camera Name="Camera01" Parent="undefined" ClassOf="Freecamera" Position="[4.03227,-
12.7882,0]" Scale="[1,1,1]" Rotation="[0.0,0.0,0.0]" FOV="45.0" />

  </Cameras>

  <Lights>

    <Light Name="Omni01" Parent="undefined" ClassOf="Omnilight"
Position="[64.6214,75.0504,0]" Scale="[1,1,1]" Rotation="[0.0,0.0,0.0]" FallOf="0"
Shadows="true" Distance="0" Multiplier="1.0" Attenuation="false-false" />

  </Lights>

</Scene>
```

Figura 40. XML resultado de una exportación con materiales

5.1.4. Exportación de datos

Hasta ahora se ha hablado sobre los dos conjuntos de información que se generan cuando se realiza una exportación de datos: el archivo XML y un archivo FBX.

Dentro del *XML Exporter*, como se ha explicado anteriormente, se exporta la información gracias al *FBX Exporter*, con la siguiente instrucción:

```
ExportFile (ruta) #noPrompt using:FBXEXPORTER
```

Pseudocódigo 9. Instrucción de exportación FBX

Donde la opción *#noPrompt* sirve para que no salgan mensajes molestos al usuario y el proceso de exportación se haga en segundo plano y de forma automática.

La *ruta* está determinada por la ruta que el usuario haya elegido para guardar el archivo XML de su elección.

Con cada exportación se generan una serie de archivos complementarios distribuidos en una jerarquía determinada, los cuales tienen que estar "visibles" dentro del proyecto de Unity3D para poder ser cargados (se muestran capturas relativas a un supuesto ejemplo de nombre "*principal*"):

En el directorio raíz, es decir, aquel donde el usuario haya decidido crear su archivo XML, se creará lo siguiente:

- Archivo XML general
- Directorio con el archivo fbx y texturas

Figura 41. Archivos y jerarquía de directorios (I)

Dentro del directorio ***[nombreXML]-FBXObjects*** se puede encontrar la siguiente disposición:

Figura 42. Archivos y jerarquía de directorios (II)

Donde, en el directorio ***Materials***, se encuentran los materiales, y en el ***[nombreXML].fbm***, se encuentran las imágenes usadas como texturas:

Figura 43. Archivos y jerarquía de directorios (Materials)

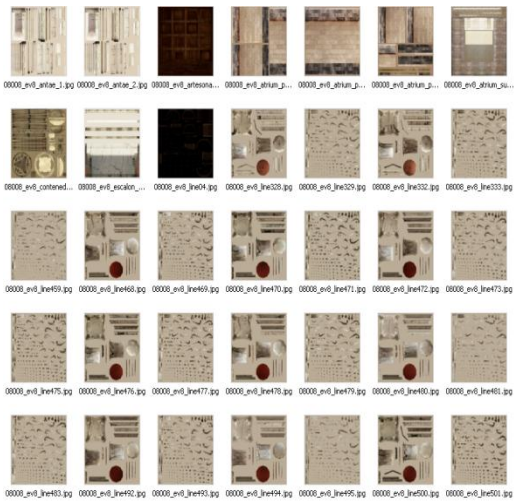

Figura 44. Archivos y jerarquía de directorios (folder.fbm)

5.2. Plugin de importación. Unity3D

Cuando se dispone de aquellos datos que se quieren importar, es necesario hacer uso del segundo módulo principal de la solución, el *XML Importer*. Se trata de un plugin integrado en *Unity3D* que permite de forma sencilla replicar escenas creadas con *3DS MAX* en el viewport de trabajo propio. En otras palabras, crea automáticamente una escena similar a aquella creada en *3DS MAX* y exportada con el *XML Exporter*.

5.2.1. Configuración del importador: Plugin y FBX

De forma homóloga al *3DS MAX*, *Unity3D* cuenta con un importador FBX propio, el cual se encarga de adaptar los recursos FBX para poder ser utilizados adecuadamente. El objetivo del plugin *XML Importer*, es utilizar dicho importador y realizar una serie de adaptaciones y correcciones sobre los resultados generados con el mismo, de forma que la escena resultado sea lo más fiel posible al original.

Por otro lado, se ha realizado también un pequeño programa visual, que permite al usuario configurar ciertos parámetros del importador FBX en pro del objetivo ya mencionado (ver Apartado 5.3).

La interfaz del *XML Importer* se ha realizado lo más simple posible para facilitar las tareas de importación de datos:

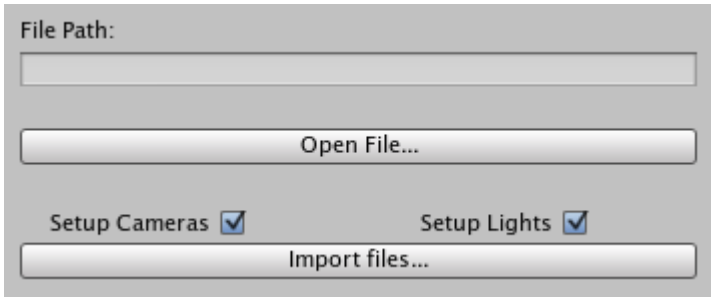

Figura 45. Interfaz del XML Importer

Como se puede observar en la imagen, únicamente existe un botón para elegir desde qué archivo XML queremos leer la información, así como dos *checkboxes* que permiten indicar si queremos importar las cámaras o las luces. Finalmente, un botón principal permite al usuario importar los ficheros deseados.

La configuración de estos parámetros por parte del usuario es algo totalmente trivial, y no requiere de aprendizaje. Se ha realizado una interfaz intuitiva que permita prescindir de explicaciones o tutoriales por parte de terceros, de manera que el uso por parte de los empleados de *Neotecno Desarrollos* sea inmediato y se consiga así el aumento en productividad que se pretende.

5.2.2. Configuración del importador: FBX

Como ya se ha mencionado anteriormente, se ha realizado una pequeña aplicación que gestiona la configuración del importador FBX incluido en *Unity3D*.

Su explicación, así como las diferentes opciones que permite configurar, se han detallado en el Apartado 5.3.

5.2.3. Lectura de XML e importación de modelos

El proceso de lectura de XML se realiza de forma similar al de creación de los mismos.

Se explora la jerarquía de nodos, seleccionando en cada uno aquella información útil que posteriormente servirá para configurar la escena debidamente.

Al pulsar el botón "*Open File…*" en la interfaz, se llama a una función especial **`OpenFile()`** que muestra un dialog **`OpenFilePanel`**[40], el cual permite al usuario seleccionar el archivo que desee:

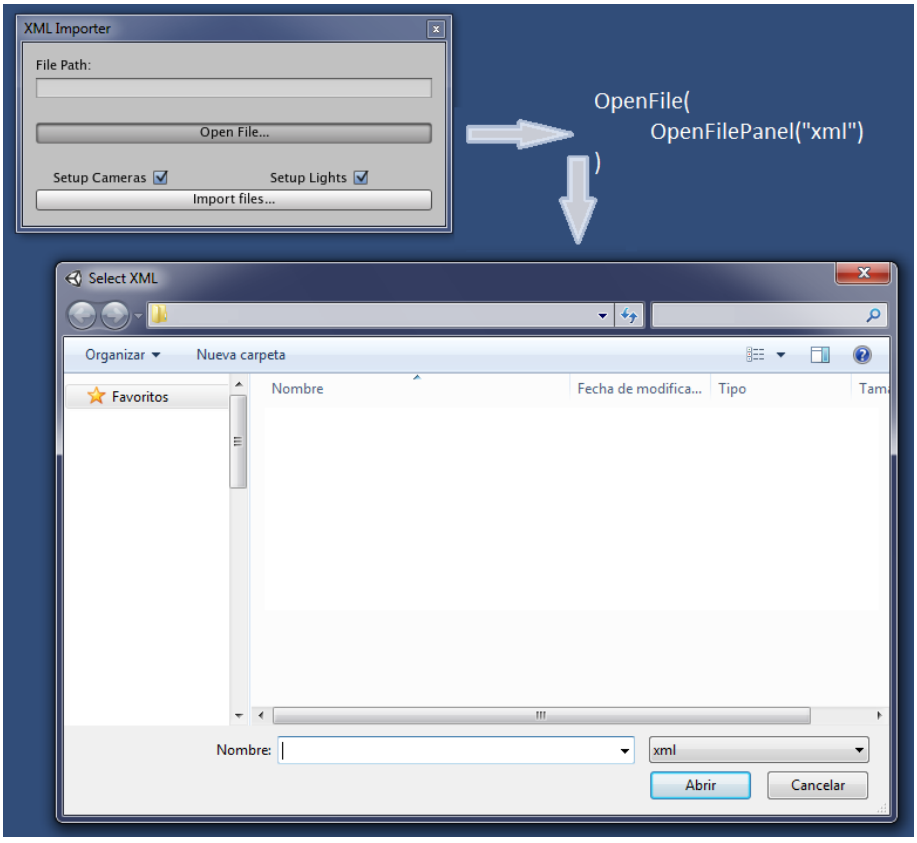

Figura 46. OpenFilePanel

[40] Documentación oficial: OpenFilePanel

Una vez ha sido configurada la ruta del archivo XML que contiene la información sobre la escena a importar, se seleccionan las opciones deseadas (***Setup Cameras*** y/o ***Setup Lights***) y se pulsa el botón "*Import files*…". Con la pulsación de dicho botón se hace una llamada a una función ***readExecXML(filePath)*** la cual se encarga de la lectura completa del archivo XML.

El pseudocódigo de dicha función se muestra a continuación:

```
función readExecXML (filePath : texto) {

            //Se crea el documento XML

            XmlDocument studioInf = nuevo XmlDocument

            //Se localiza el path del FBX correspondiente, el cual se almacena en la raíz

            texto fbxPath = sceneRoot.Attributes["FBXPath"].Value;

            //Se obtienen todos los demás elementos

            XmlNodeList objetos = studioInf.getElementos(Objetos)

            XmlNodeList camaras = studioInf.getElementos(Camaras)

            XmlNodeList luces = studioInf.getElementos(Luces)

            //En caso de que en el XML no haya cámaras o luces, establecemos el booleano a falso

            Si (no hay cámaras ) entonces loadCamaras = false

            Si (no hay luces) entonces loadLuces = false

            //Para cada hijo inmediato de la raíz... OBJETOS

            Para cada (XmlNode obj en objetos) hacer{

                  //Extraemos su nombre

                  texto nombreO = obj.Atributos[“Nombre”]

                  //Y si no existe en el diccionario todavía, lo añadimos

                  si (!existe en diccionarioObjetos) entonces
                        diccionarioObjetos.añadir(nombreO)

                  //Y llamamos a un método recursivo el cual se encargará de leer todos los hijos
                  del nodo en análisis

                     objRecursiveXmlNode(obj);

                }

            }

            //Para cada hijo inmediato de la raíz... CÁMARAS

            Para cada (XmlNode obj en camaras) hacer{
```

```
                    //Extraemos su nombre

                    texto nombreO = obj.Atributos["Nombre"]

                    //Y si no existe en el diccionario todavía, lo añadimos

                    si (!existe en diccionarioCamaras) entonces
                          diccionarioCamaras.añadir(nombreO)

                    //Y llamamos a un método recursivo el cual se encargará de leer todos los hijos
                    del nodo en análisis

                       camRecursiveXmlNode(obj);

                   }

               }

               //Para cada hijo inmediato de la raíz... LUCES

               Para cada (XmlNode obj en luces) hacer{

                    //Extraemos su nombre

                    texto nombreO = obj.Atributos["Nombre"]

                    //Y si no existe en el diccionario todavía, lo añadimos

                    si (!existe en diccionarioLuces) entonces
                          diccionarioLuces.añadir(nombreO)

                    //Y llamamos a un método recursivo el cual se encargará de leer todos los hijos
                    del nodo en análisis

                       litRecursiveXmlNode(obj);

                   }

               }

               //Una vez hemos leído toda la información del XML, realizamos la importación

               import(fbxPath);

}
```

Pseudocódigo 10. Función principal. Lectura del XML

El pseudocódigo de dicha función únicamente recorre todos los nodos del archivo XML, y va añadiendo los nombres de dichos nodos en tres diccionarios diferentes, evitando repeticiones. De esta manera, cuando esta función ha finalizado, se tiene una relación completa de todos los objetos de la escena original que han de ser importados en *Unity3D*.

Posteriormente, para cada nodo leído, se llama a una función ***[obj|cam|lit]RecursiveXmlNode*** la cual es encargada de leer la información de todos los atributos de cada objeto y de ir añadiéndola en la entrada del diccionario correspondiente.

Estas funciones tienen una estructura similar, que se detalla en el siguiente pseudocódigo:

```
función [obj|cam|lit]RecursiveXmlNode(XmlNode obj){
        //Array donde se irán almacenando los diferentes datos leídos del archivo XML para cada objeto
        object[] datos = new object[n];
        //Nombre del objeto que se está analizando
        texto nombre = (texto)obj.Atributos["Nombre"]
        //Valores escritos en el XML
        //Para cada nodo n hijos, cada uno de los cuales contiene la diferentes información
        Para cada atributo en obj hacer {
            Leer atributo
            Guardar atributo en datos[i]
        }
        //Dentro del diccionario, en la entrada cuya key es el nombre del objeto, añadimos el array data
        diccionario[Objetos|Camaras|Luces] [nombre] = datos;
        //Se obtienen los hijos del nodo
        XmlNodeList hijos = obj.ChildNodes;
        //Si existen, se hace un recorrido por todos ellos, llamando al correspondiente método de lectura e importación
        //De esta manera, se exploran todos hijos de cada nodo, de manera recursiva
        si (hijos.Count > 0) entonces {
            para cada (XmlNode child in hijos) hacer {
                texto nombreO = child.Atributos["Nombre"]
            si (!existe en diccionario[Objetos|Camaras|Luces]) entonces {
                  diccionario[Objetos|Camaras|Luces].añadir(nombreO)
            }
            }
      }
}
```

Pseudocódigo 11. Función específica a cada Nodo. RecursiveXMLNode

Se puede observar en el pseudocódigo que para cada nodo pasado como parámetro, se recorren todos sus atributos (ver Apartado 5.1.3) leyendo su información y guardándola en

un array ***datos***. Dicho array es guardado como valor en la entrada del diccionario cuya clave es el nombre del objeto. Posteriormente, se recorren todos sus hijos realizando la misma operación. De esta manera, se realiza un recorrido descendente por toda la jerarquía (al igual que cuando se escribía el archivo) analizando todos los objetos y guardando su información en el diccionario correspondiente.

Al final de la función ***readExecXML(filePath)*** se observa una llamada a ***import(fbxPath)*** método encargado de realizar la importación de los datos dentro de la escena (ver Apartado 5.2.5)

5.2.4. Análisis de información y equivalencias entre sistemas

El primer paso antes de crear la escena solución dentro de Unity3D, es recorrer los recursos importados para verificar que hayan sido correctamente creados, y añadir la información pertinente.

Para tal efecto, se sigue el siguiente pseudocódigo:

```
función import(texto fbxPath){
      //Creación del recurso
      GameObject prefab = LoadAssetAtPath("Assets" + fbxPath)
      //Creación de un array en el que se almacenarán los objetos no presentes en el XML
      ArrayList noName = nuevo ArrayList();

       //Para cada hijo inmediato de la escena importada
      para cada hijo en prefab hacer {
            //Obtenemos el gameObject asociado
             GameObject obj = hijo.gameObject;
            //En caso de que el diccionario contenga información sobre este hijo (que es lo previsible)
            //Se configuran sus atributos correctamente
       si(diccionarioObjetos contiene información sobre obj) entonces {
             #Se obtienen los datos correspondientes del diccionario#
             #Se configuran los parámetros del objeto correctamente#
       }
      //Se realiza el configurado de las cámaras
       sino si (diccionarioCamaras contiene información sobre obj &&
      setupCameras) entonces {
```

```
            #Se obtienen los datos correspondientes del diccionario#
            #Se configuran los parámetros de la cámara correctamente#
            }
            //Se realiza el configurado de las luces
      sino si(diccionarioLuces contiene información sobre obj &&
      setupLuces) entonces {
            //Se obtienen del diccionario los datos correspondientes
            #Se obtienen los datos correspondientes del diccionario#
            #Se configuran los parámetros de la luz correctamente#
      }sino{
            noName.añadir(obj);
      }
      //Finalmente, y una vez se ha realizado toda la pre-configuración, se instancia el objeto en la escena
     Instantiate(prefab);


//Realizamos un tratado después de la instanciación, que elimine todos aquellos objetos no presentes en el
XML (elimina objetos introducidos por el FBX, y que a priori son innecesarios)
     GameObject instantiate = GameObject.Find(prefab);
     //Durante la lectura del XML se han ido almacenando en un arrayList los nombres de aquellos
       objetos que no se encontraron
     para cada elemento en noName hacer {
           GameObject.DestroyImmediate(elemento);
     }
}
```

Pseudocódigo 12. Configuración de la escena en Unity3D. Análisis de datos XML

En el pseudocódigo se puede observar el proceso general de configuración de la escena en *Unity3D* llevado a cabo por el *XML Importer*.

En primer lugar se carga toda la información del FBX en bruto sobre una variable propia de *Unity3D* de tipo *GameObject*[41] llamada *prefab*. Aparte, se crea un array *noName* donde se

[41] En Unity3D un GameObject se podría considerar la estándar básica de información. Descripción oficial

irán almacenando aquellos objetos presentes en el prefab, pero no presentes en el archivo XML.

El prefab contiene información sobre todos los elementos que el *XML Exporter* junto al *FBX Exporter* crearon dentro del archivo FBX correspondiente a la escena. Estos elementos son susceptibles de estar mal configurados, o no ser necesarios, por lo que se realizan las comprobaciones posteriores.

Para cada uno de ellos, se comprueba en qué diccionario de datos se ha almacenado su información (ver *Apartado 5.2.3*) y en caso de no estar disponible en ninguno, se añade al array *noName*.

Si el elemento era un objeto se realizan comprobaciones y configuración sobre su posición, rotación y material.

En ambos programas se utilizan sistemas de ejes diferentes[42]. Aparte, se nota que el eje Z en *3DS MAX* pasa a ser el Y en *Unity3D*.

Para *3DS MAX* el eje de coordenadas utilizado es el siguiente (*Right-Handed*):

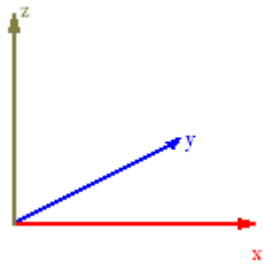

Figura 47. Ejes en 3DS MAX

Para *Unity3D* se utiliza el siguiente sistema de ejes (*Left-Handed*):

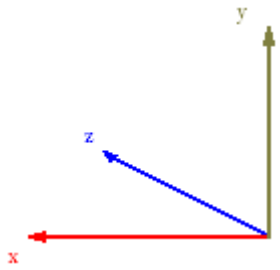

Figura 48. Ejes en Unity3D

En caso de ser una cámara, se comprueba la posición, la rotación, el FOV y el tipo.

Las equivalencias entre los tipos de cámara disponibles en *3DS MAX* y *Unity3D* son directas, ya que todos los tipos disponibles en *3DS MAX* se mapean en una única cámara estándar en *Unity3D*.

3D Studio MAX	***Unity3D***
Free Camera	Camera
Target Camera	Camera

Tabla 3. Equivalencias entre cámaras

[42] Right-Handed vs Left-Handed: Enlace

Para una luz, se realiza una comprobación de su posición, rotación, tipo, alcance, generación se sombras, y atenuación.

Las equivalencias entre los tipos de luces disponibles en *3DS MAX* y *Unity3D* se muestran en la siguiente tabla:

3D Studio MAX	*Unity3D*
targetSpot	Spot Light
freeSpot	Spot Light
directionalLight	Directional Light
targetDirectionalLight	Directional Light
Omnilight	Point Light
Skylight	Spot Light
miAreaLight	Point Light
miAreaLightomni	Point Light
Target_Point	Point Light
Free_Point	Point Light
Target_Linear	Point Light
Free_Linear	Point Light
Free_Area	Point Light
Target_Area	Point Light

Tabla 4. Equivalencias entre tipos de luces

Como aspecto final, hay que tener en cuenta que la interacción entre ambos plugins no es compatible con los nombres de objeto que contengan acentos. De esta forma, cuando se guarda información acerca de un objeto cuyo nombre contenía un acento en un archivo XML existe incompatibilidad de caracteres, y dicho nombre se guarda en forma errónea. Por otro lado, en el archivo FBX se guarda correctamente. Así, cuando se realiza la lectura de datos desde *Unity3D* el nombre almacenado en el diccionario correspondiente (proveniente del XML) y el nombre propio del objeto (proveniente del FBX) no coinciden, por lo que el objeto será eliminado de la escena.

5.2.5. Creación de escena solución

En el Pseudocódigo 12 se puede observar la instrucción **`Instantiate(prefab).`**

Dicha instrucción crea en la escena de Unity3D, una entidad 3D equivalente a la representada en el archivo FBX.

Como se puede observar en la sección anterior, antes de crear la escena solución se han ido recorriendo todos los elementos almacenados en el archivo FBX, verificando sus atributos, y configurando aquellos susceptibles de generar problemas de compatibilidad.

5.3.Utilidad de configuración FBX. Unity3D

La utilidad de configuración FBX consta de dos clases principales (ver *Apartado 4.4*).

- El archivo básico de configuración: ***FbxPreProcessor***
- La utilidad de configuración: ***FBXImporterSetup***

Para el primero de ellos, el pseudocódigo se basa en la función ***OnPreprocessModel***[43] implementada en la librería *UnityEngine* y que se ejecuta antes de cualquier cambio llevado a cabo por el monitor de recursos.

De esta manera, simplemente con ubicar este archivo dentro de nuestro proyecto, cada cambio realizado dentro de esta función es aplicado a todos los recursos incluidos en nuestro proyecto de *Unity3D* antes de cada operación de proceso de archivos llevada a cabo por el monitor de recursos. Teniendo en cuenta que cada vez que queremos utilizar un recurso que hayamos exportado anteriormente debemos incluirlo en nuestra jerarquía, como resultado final de esta estrategia obtenemos que los cambios realizados dentro de la clase ***FbxPreProcessor*** son aplicados a cada recurso siempre que éste es incorporado a nuestro proyecto.

Dentro de la clase se modifican las siguientes propiedades del *FBX Importer* que utiliza *Unity3D* para configurar los recursos importados:

- `generateMaterials`: Define si se deben o no crear los materiales en los objetos importados (por defecto, siempre se pone a true)
- `addCollider`: Define si se crean controladores de colisión para los objetos importados[44]
- `globalScale`: Define la escala global de los objetos importados (*Unity3D* establece un valor por defecto de 0.01, el valor correcto para una conversión 1:1 es 1)
- `swapUVChannels`: Define si se deben "cambiar" los canales UV del material[45]

[43] Documentación oficial: OnPreProcessModel

[44] Un ***controlador de colisión*** permite a otros objetos saber cuando están en contacto con aquel que lo tiene. Entre otras muchas aplicaciones, es utilizado para que los objetos de la escena puedan chocar entre sí

[45] Documentación oficial: SwapUVChannels

- `generateAnimations`: Define en qué manera deben importarse las animaciones. Existen cuatro posibilidades[46]:
    - `InNodes`: Cada animación se almacena en el nodo correspondiente
    - `InRoot`: La animación completa se almacena como una unidad en la raíz del objeto (proporciona un buen rendimiento, pero resta control sobre la información importada)
    - `InOriginalRoots`: Cada animación en su raíz original
    - `None`: No importar animaciones

Para hacer visibles estas variables al usuario, y evitar que tenga que estar editando el archivo ***FbxPreProcessor*** cada vez que quiera realizar un cambio en la configuración del exportador, se ha realizado una utilidad de interfaz que permite modificarlas de forma fácil e intuitiva mediante campos de texto y botones.

Básicamente, esta utilidad modifica directamente el propio archivo ***FbxPreProcessor*** reescribiendo el código correspondiente a las modificaciones realizadas por el usuario.

Simplemente localiza cadenas de texto fijas que identifica con las asignaciones de valores correspondientes a cada propiedad del *FBX Importer* y las reescribe en función de los settings establecidos por el usuario en la interfaz.

[46] Documentación oficial: Character Animation

5.4.Resultados

A continuación se muestran capturas de las herramientas realizadas, así como algunos ejemplos de resultados obtenidos con las mismas:

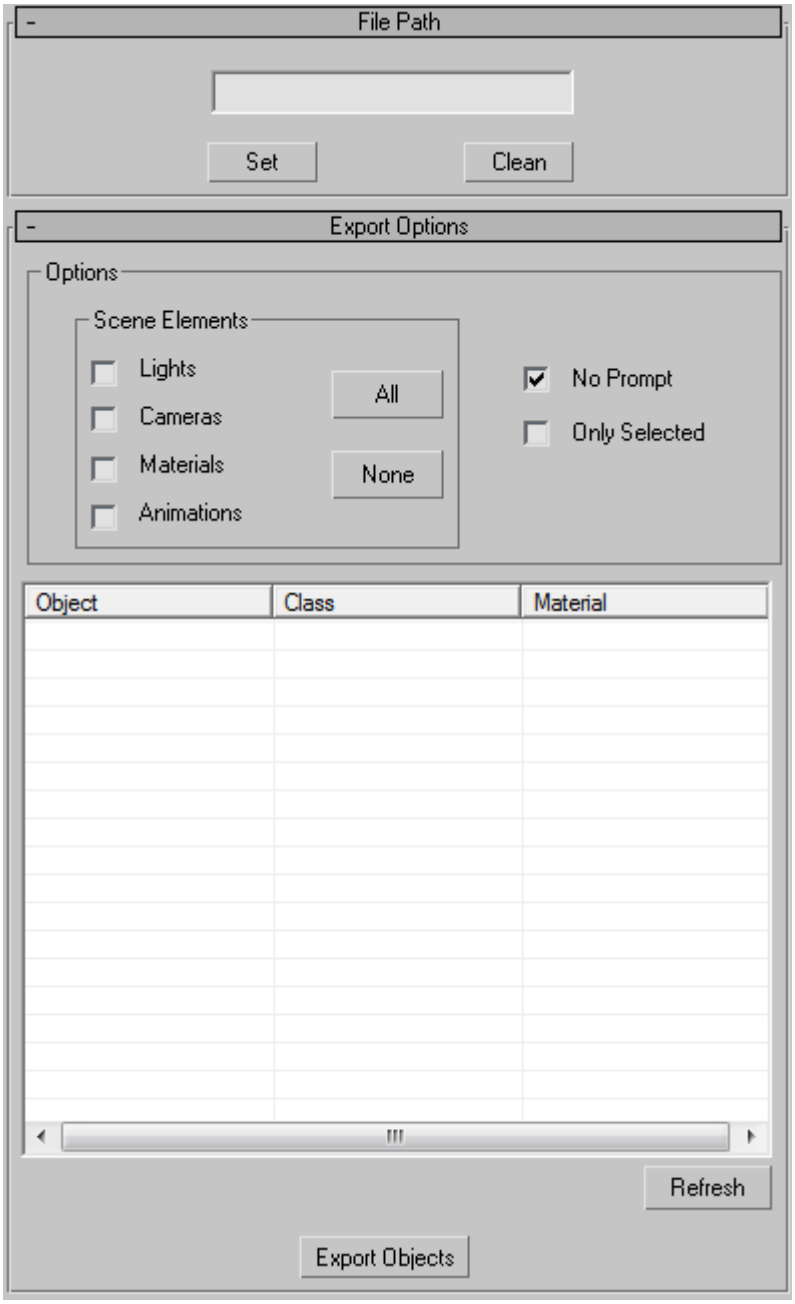

Figura 49. XML Exporter

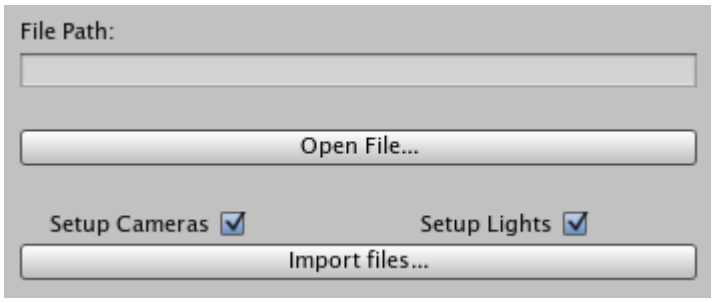

Figura 50. XML Importer. Unity3D

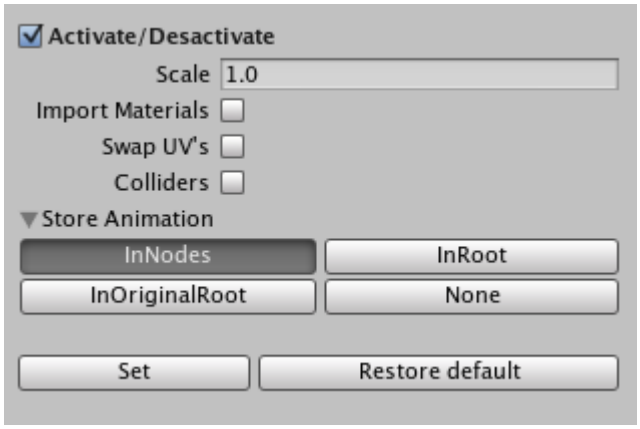

Figura 51. FBXImporterSetup. Utilidad de configuración de FBX Importer. Unity3D

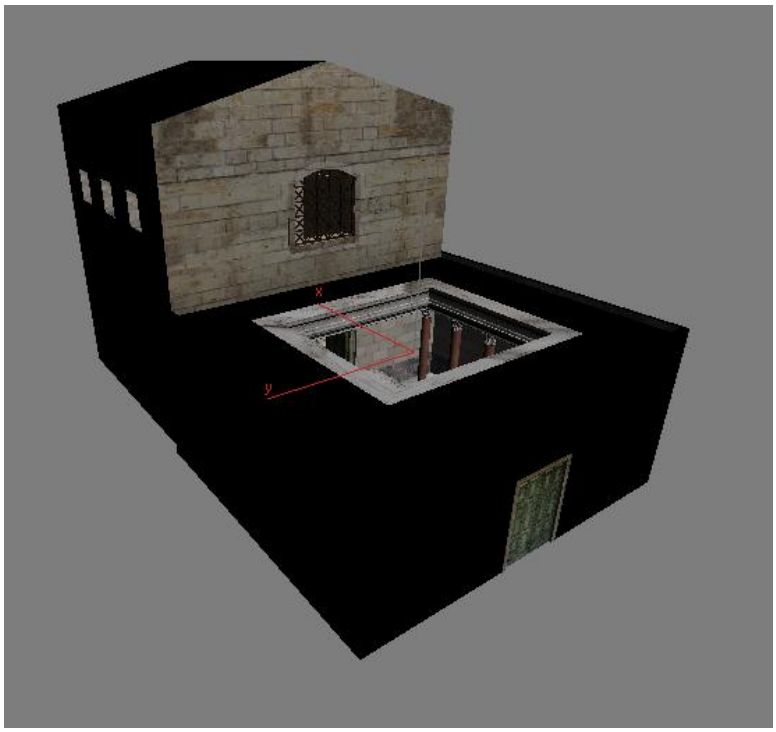

Figura 52. Modelo en 3DS MAX

Figura 53. Modelo resultado en Unity3D

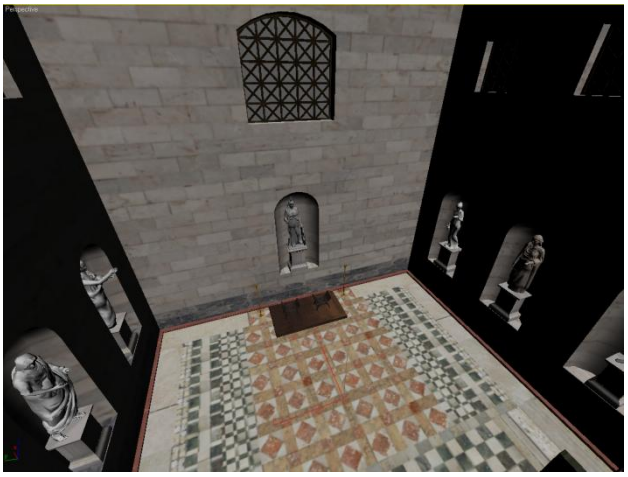

Figura 54. Detalle del interior. 3DS MAX

Figura 55. Detalle del interior Unity3D (retocado)

6. Conclusiones y vías futuras.

6.1.Conclusiones

El presente proyecto ha realizado un recorrido horizontal sobre todas las fases de desarrollo de software dentro de un grupo real de trabajo.

En un primer momento, se realizó un análisis de requisitos junto al cliente (en este caso los empleados de la empresa) con el que se acotaron sus necesidades y se consiguió una idea general de la aplicación. Posteriormente, se realizó un diseño de la solución que mediante una metodología incremental se fue completando en base a diferentes fases. Entre cada una de ellas, existía comunicación y retroalimentación con los diferentes empleados de forma que la aplicación final ha respetado un compromiso entre prestaciones proyectadas y añadidos solicitados por el personal, dando lugar a una solución adaptada lo máximo posible a las necesidades de la empresa.

El proceso ha sido complicado, sobre todo debido a que la solución se integraba en dos plataformas diferentes y ha sido necesario llegar a entender profundamente el funcionamiento de ambas. Como experiencia base, era necesario tener conocimientos medios de programación, así como de modelado en 3D y creación de juegos, ya que la herramienta realizada guarda relación con ambos mundos. A lo largo del proyecto, se han aprendido nociones sobre el lenguaje *MAXScript*, así como sobre las librerías principales de *Unity3D*; *UnityEngine* y *UnityEditor*. Ha sido también necesario realizar diferentes interfaces en cada sistema, con lo que también se han aprendido a manejar herramientas visuales de creación de aplicaciones visuales.

De esta forma, si actualmente se presentara un proyecto de similares características, dentro de la metodología adoptada sería posible eliminar fases relativas a la documentación y revisión de tutoriales, disminuyendo en gran medida el tiempo proyectado de realización.

El principal problema y punto de inflexión se produjo casi al inicio del proyecto, ya que en un principio se pretendía exportar la información como texto plano puro, prescindiendo del formato FBX. Al poco tiempo de trabajar sobre esta idea, se concluyó que era un trabajo mucho más complicado de lo que parecía a priori, y se escapaba de la planificación temporal estimada, así como del carácter general de un proyecto de fin de carrera. Por tanto, de entre todas las alternativas se optó por utilizar el formato FBX.

Los objetivos iniciales marcados al inicio han sido satisfechos ya que:

- Se han desarrollado dos plugins y una herramienta de configuración que realizan la funcionalidad para la que fueron diseñados, y se ajustan a las necesidades de los empleados de *Neotecno Desarrollos*, presentando una interfaz clara y sencilla.

- Se ha llevado a cabo una metodología incremental, que favorecía el diseño y desarrollo de la estructura modular de la aplicación. De esta manera ha sido más sencillo añadir mejoras o realizar cambios sobre la misma.

- Ambas herramientas pueden ser utilizadas de forma individual. Su funcionalidad está bien acotada, y no dependen del resto del software para llevarla a cabo.

- Se utiliza XML como lenguaje para transportar información entre ambas aplicaciones. Únicamente se genera un fichero de información que centraliza los datos sobre la escena exportada.

- Se realiza una conversión exacta del contenido, respetando escala, posiciones y rotaciones de los objetos. Aún así, tanto el sistema de importación de cámaras como el de luces puede ser mejorado. Esto se ha propuesto en trabajos futuros

- Ha habido comunicación constante con todo el equipo durante todo el desarrollo del proyecto. La retroalimentación con los empleados ha sido constructiva, de manera que sus peticiones se han podido satisfacer en su totalidad.

Por tanto, podemos concluir que el proyecto ha sido finalizado satisfactoriamente y que por tanto se está utilizando dentro de la empresa *Neotecno Desarrollos S.L*.

6.2.Vías Futuras

Para futuras versiones del proyecto, y debido a su carácter incremental, se presentan diversas tareas que pueden ser integradas como nuevos incrementos:

- Redefinir las equivalencias *3DS MAX – Unity3D* entre cámaras y luces, de forma que se consiga una relación prácticamente directa.
- Integrar el *XML Importer* y el *FBXImporterSetup* como una única herramienta, ampliando la funcionalidad del segundo, y unificando así el módulo referente a *Unity3D* en una única interfaz.
- Añadir funcionalidades dentro del plugin de *3DS MAX* que permitan realizar una configuración pre-exportación que evite posteriores adaptaciones dentro de *Unity3D*, tales como posiciones incorrectas, rotaciones imprecisas o materiales mal configurados.

7. Bibliografía

[1].*La Psicología de los Videojuegos: Un modelo de investigación.* Ricardo Tejeiro Salguero, Manuel Pelegrina del Río (2008).

[2].*Ingeniería del Software: Un enfoque práctico 5ª Edición.* Roger S. Pressman (Trad. Darrel Ince) (2002).

[3].*Novedades 3DS MAX 2008.* Comgraph (2007).. Enlace

[4].Apuntes de la asignatura: *Implementación de una tienda web sencilla mediante plataforma .net*. Universidad de Zaragoza

[5].*Alias FBX Initiative Web Paper.* Michel Besner (2005). p.4 Enlace

[6].Documentación Oficial *MAXScript 9*. Enlace

[7].*Autodesk 3DS MAX FBX Plugin Help (2010)*. Autodesk. Enlace

[8].*Autodesk 3DS MAXScript Essentials (2007).* Autodesk.

[9].Página Oficial de *Autodesk*. Enlace

[10]. Página Oficial de *Unity3D*. Enlace

[11]. Página Oficial de *Microsoft*. Enlace

[12]. *Wikipedia* en inglés. Enlace

[13]. *Wikipedia* en Castellano. Enlace

8. Anexos

8.1.Acuerdo de confidencialidad

El presente proyecto ha sido realizado bajo un acuerdo de confidencialidad firmado junto a la empresa *Neotecno Desarrollos S.L*. Es por ello que aunque la autoría se atribuya al autor del presente documento (*Fernando Matarrubia García*), los derechos de uso y PI son propiedad de *Neotecno Desarrollos S.L*.

Por tanto no es posible, ya que estaría fuera de la legalidad, entregar junto al proyecto los resultados obtenidos.

Cualquier aplicación desarrollada en base a las directrices del presente documento deberá contar con la aprobación de la empresa *Neotecno Desarrollos S.L*. En caso contrario, la actuación se considerará ilegal, y podrá ser sometida a acciones legales por parte de cualquiera de las partes involucradas en la realización del proyecto.