eCharacter: Una herramienta libre para la creación de personajes 3D.

Autores:

Sergio de Luis Nieto

David González Ledesma

Alejandro Muñoz del Rey

SISTEMAS INFORMÁTICOS

Directores:

Baltasar Fernández Manjón

Javier Torrente Vigil

Curso 2012/2013

Facultad de Informática

Universidad Complutense de Madrid

SE AUTORIZA A LA UNIVERSIDAD COMPLUTENSE A DIFUNDIR Y UTILIZAR CON FINES ACADÉMICOS, NO COMERCIALES Y MENCIONANDO EXPRESAMENTE A SUS AUTORES, TANTO LA PROPIA MEMORIA, COMO EL CÓDIGO, LOS CONTENIDO AUDIOVISUALES INCLUSO SI INCLUYEN IMÁGENES DE LOS AUTORES, LA DOCUMENTACIÓN Y/O EL PROTOTIPO DESARROLLADO.

Autores:

Sergio de Luis Nieto *David González Ledesma* *Alejandro Muñoz del Rey*

Fecha

Madrid, 21 de Junio de 2013

Agradecimientos

Gracias a nuestras familias, amigos y parejas por apoyarnos en los malos momentos y ayudarnos con sus ideas y opiniones cuando más lo necesitábamos.

Gracias a los directores, Baltasar Fernández Manjón y Javier Torrente Vigil ya que gracias a su dirección y consejos este proyecto ha podido salir adelante.

Y un agradecimiento especial a Iván Martínez Ortiz, por compartir con nosotros sus conocimientos sobre el mundo del modelado 3D y a Ángel Serrano Laguna, por su aportación en el proceso de tratamiento de imágenes.

Índice

Índice de figuras

Resumen

eCharacter es una herramienta de software libre que permite la personalización de modelos 3D para su inclusión en videojuegos. Ofrece al usuario la posibilidad de crear nuevos personajes partiendo de varios modelos base mediante una herramienta sencilla de usar, flexible e intuitiva. El proyecto trata de reducir los costes de creación de recursos artísticos para videojuegos, simplificando de esta manera el proceso de desarrollo. Aunque *eCharacter* puede utilizarse en la creación de personajes para una gran variedad de videojuegos, inicialmente está diseñada y optimizada para prototipado rápido y para su aplicación en proyectos de bajo coste. Por ello se ha diseñado *eCharacter* para su posible integración con otras herramientas de creación de juegos, especialmente aquellas orientadas al ámbito de la educación y dirigidas a usuarios sin conocimientos extensivos en el campo del desarrollo y la programación de videojuegos.

Los videojuegos tienen cada vez más repercusión en la sociedad, y hoy en día, es una industria que está en auge. El uso de los videojuegos no sólo se limita al ocio sino que se está extendiendo a otros campos como el de la salud y la educación en los que los presupuestos de creación de los juegos normalmente son mucho más limitados. Por este motivo, cada vez están surgiendo más herramientas para crear videojuegos, muchas de ellas gratuitas y libres, de una manera sencilla y rápida para el usuario. Este tipo de herramientas, supone un gran ahorro económico para los usuarios que necesitan crear un videojuego simple y que no disponen ni de conocimientos técnicos ni de recursos económicos para realizarlo. Sin embargo, uno de los principales problemas a los que se exponen los usuarios de estas plataformas libres y gratuitas es la obtención de recursos artísticos de alta calidad. Aunque la red proporciona multitud de recursos para obtener modelos 3D, sigue siendo complejo encontrar personajes 3D animados que se ajusten a las necesidades de proyectos no dirigidos al género de acción. Es ahí donde *eCharacter* pretende ser una contribución significativa, proporcionando a la comunidad una aplicación libre que permite crear de una manera sencilla y gratuita diferentes modelos 3D animados, según las necesidades de los usuarios y que sea fácilmente integrable con otras herramientas. Además tanto *eCharacter* como todos los recursos gráficos que incluye son gratuitos, permitiendo al usuario facilitarle la tarea en el proceso de creación de los modelos sin que le suponga coste alguno.

Palabras claves

- eCharacter
- Configurador
- Personaje
- Videojuego
- eAdventure
- Modelo 3D
- Código libre
- jMonkey Engine
- Nifty GUI

Abstract

eCharacter is free software that allows customization of 3D models for video games. It provides the user with a simple and flexible tool to create personalized animated characters for video game development. The project aims to help cut down development costs for creation of art assets for video games, making the whole development process simpler, easier and affordable to more people. Although *eCharacter* can be used to create characters for a wide range of video games, it has been optimized for applications in rapid prototyping and low cost projects. *eCharacter* has been designed for easy integration with other games creation tools, especially those that are oriented to education and target users that are not professionals in videogame development and programming.

Video games are undoubtedly a consolidated sector of the entertainment industry and have gained deep social acceptance. Moreover, the use of video games is not limited to leisure any more. Instead, it is spreading to other fields like education or health. In this kind of fields the budget for video games creation is more limited. Consequently, there are increasingly more free software tools becoming available for creating video games quickly and easily. This kind of tools are extremely useful for users who need to create simple video games and they have neither the experience nor the financial resources to do it. However, it is still problematic to gather high quality art assets, especially for some types of assets. Although the web is full of sources of 3D models and other art designs for games, free 3D animated models are still scant and they are not always compatible with free software technology. In this case is where *eCharacter* aims to contribute back to the community with free application that allows creating different 3D animated models and that can be easily integrated with other tools.

Keywords:

- eCharacter
- Configurator
- Character
- Videogame
- eAdventure
- 3D Model
- jMonkey Engine
- Nifty GUI
- Open source
- Avatar

1. Resumen ejecutivo

1.1. Planteamiento del problema

La industria del videojuego ha experimentado en los últimos años un gran crecimiento, convirtiéndose en uno de los negocios de entretenimiento con mayor facturación. Las empresas desarrolladoras de videojuegos invierten grandes sumas de dinero para conseguir videojuegos cada vez más sofisticados técnicamente, que les permitan competir en un sector que es altamente competitivo y donde el riesgo es elevado. Debido a esta sofisticación los costes y la complejidad de desarrollo de videojuegos comerciales se incrementan con cada nueva generación de consolas, situándose actualmente en el rango de entre 15 y 30 millones de dólares para juegos de Play Station 3 y XBOX 360 (Sharkey, 2010).

Uno de los campos donde las empresas invierten más dinero es en la generación de los modelos 3D. También se ha vuelto práctica común, en muchos casos, proveer al usuario con herramientas dentro del juego para configurar los modelos de los personajes o de su avatar. Este tipo de herramientas de personalización de modelos están normalmente integradas dentro del producto final de modo que suelen ser muy restrictivas y no permiten realizar modificaciones fuera de los límites establecidos por los desarrolladores. Además, son herramientas propias de un producto o juego de modo que reutilizar este tipo de módulos en otros proyectos es imposible debido a las restrictivas licencias que poseen.

De forma paralela, también se ha observado en los últimos años una popularización de la actividad de creación de videojuegos en entornos indie, experimentales, o en entornos ajenos a la industria, como puede ser el sistema educativo. Este tipo de proyectos no disponen de los grandes presupuestos de los que gozan los proyectos estrella de la industria, y por tanto buscan permanentemente nuevas fórmulas para la reducción de costes (F A S, 2006).

A partir de este análisis, surge la idea de este proyecto que no es otra que proporcionar una herramienta de configuración de personajes que sea libre, gratuita, flexible y fácilmente integrable en herramientas de creación de videojuegos.

1.2. Objetivos Generales

El objetivo principal de *eCharacter* es proporcionar una herramienta que facilite y simplifique la creación de personajes animados 3D para videojuegos. La simplificación en la creación de personajes puede tener un impacto significativo en la creación de un videojuego, dado el elevado coste que tiene la creación de recursos gráficos, que en proyectos grandes se sitúa en torno al 25% del coste final del producto (Rogers, 2012).

eCharacter busca ser una herramienta flexible y versátil que además permita al desarrollador adaptarla a sus necesidades. Los usuarios podrán integrar sus propios modelos 3D de forma sencilla y personalizar la interfaz a sus gustos y necesidades.

Además esta herramienta libre de código abierto es fácilmente adaptable, ya que el usuario final no sólo podrá utilizar el configurador de personajes como una herramienta externa para configurar sus modelos y exportarlos sino que también podrá integrarla en su producto final.

1.3. Aportaciones

Las ventajas principales que proporciona este proyecto son las siguientes:

- Contribución a la comunidad de código abierto (open source) y libre: Actualmente existen muchas herramientas gratuitas y de código abierto que facilitan el desarrollo de videojuegos en proyectos de presupuesto limitado. *eCharacter* busca cubrir una necesidad cada vez más acuciante, ya que no nos consta la existencia de configuradores de personajes de código abierto. Gracias a la licencia LGPL del código y a la licencia Creative Commons de los modelos que se incluyen, cualquier usuario podrá utilizar esta herramienta para crear los personajes de su videojuego o incluso integrarla en el propio videojuego reduciendo considerablemente el esfuerzo, tanto en tiempo como en coste, del desarrollo del mismo.

- Flexibilidad: *eCharacter* busca ser una herramienta flexible y por ello se ha desarrollado pensado en posibles modificaciones o ampliaciones futuras por parte de los usuarios finales. También incorpora un sistema de internacionalización de interfaz que permite poder añadir fácilmente nuevos idiomas para dar cobertura al mayor número de usuarios posible. Además se ha desarrollado un esquema subyacente de representación de los modelos, basado en ficheros XML que permite a los usuarios finales poder incorporar nuevos modelos a la herramienta, así como adaptar la interfaz de personalización del personaje según sus necesidades.

- Potencia y versatilidad: Gracias a los modelos proporcionados y a la gran cantidad de texturas incorporadas, se puede lograr un elevado número de combinaciones posibles de modo que se crean personajes muy diversos, pero sin que esto suponga un aumento del coste. Esto, unido a la capacidad de adaptación y flexibilidad comentada en el punto anterior, dota a *eCharacter* de un gran potencial.

- Multi plataforma: *eCharacter* se ha desarrollado y se ha comprobado su adecuado funcionamiento en las plataformas de escritorio más usadas actualmente (Windows, Linux y Mac OS X).

- Integración: Como se ha comentado anteriormente, los usuarios podrán utilizar esta herramienta como un configurador de personajes externo a su videojuego final, y con el cual podrán crear los personajes deseados. Además, y si así lo desean, también podrán integrarla como un módulo de su plataforma dando lugar a un único producto final.

1.4. Conclusiones

eCharacter trata de abordar la necesidad de simplificar y reducir el coste del desarrollo de videojuegos, como es la disponibilidad de una fuente de recursos gráficos que puedan ser personalizados. Usando esta aplicación se puede reducir notablemente los costes de producción de un videojuego, ya que todos los modelos utilizados son gratuitos y por tanto no es necesario realizar inversión económica para utilizar un modelo 3D.

Además gracias al sistema de internacionalización y al sistema de integración de nuevas familias de modelos, el uso de la aplicación no está limitado sólo a los modelos proporcionados. *eCharacter* podrá crecer con la colaboración de usuarios y desarrolladores que deseen crear nuevas familias de modelos o nuevos idiomas e incorporarlos fácilmente al núcleo del sistema.

2. Introducción

eCharacter es una herramienta libre destinada a la creación de personajes 3D para videojuegos, que ofrece al usuario la posibilidad de crear nuevos personajes a partir de un conjunto base que incluye la herramienta de forma sencilla, eficiente y flexible. La herramienta está diseñada para que los personajes creados puedan utilizarse en diversas herramientas de creación de juegos, lo que permite a otros desarrolladores reducir costes en la generaciones de recursos y aumenta el potencial de aplicación de *eCharacter*.

Actualmente los videojuegos tienen una gran relevancia en el mercado del entretenimiento. Se han convertido en una de las principales industrias, y muestra de ello son las elevadas cifras de facturación registradas en los últimos años. Según datos de la Asociación de Empresas de Software de Entretenimiento (*Entertainment Software Association* en inglés), organización que agrupa a las grandes corporaciones del sector como Sony o Activision), durante el año 2011 la industria del videojuego estadounidense generó unos ingresos de 25 mil millones de dólares (ESA, 2012). Ya en nuestro país y según el estudio anual realizado por aDeSe (Asociación Española de Distribuidores y Editores de Software de Entretenimiento) en el año 2012 se generaron ingresos por valor de 822 millones de euros (ADeSe, 2012).

El interés en los videojuegos se ha expandido a otros campos más allá del entretenimiento, como la salud, la concienciación social y, sobre todo, la educación (Hwang & Wu, 2012; Johnson et al., 2013). Este conjunto de nuevas aplicaciones de los videojuegos se conoce como *serious games*. Aunque los juegos destinados al ocio siguen siendo predominantes en cifras de mercado, se observa una clara tendencia al alza en serious games.

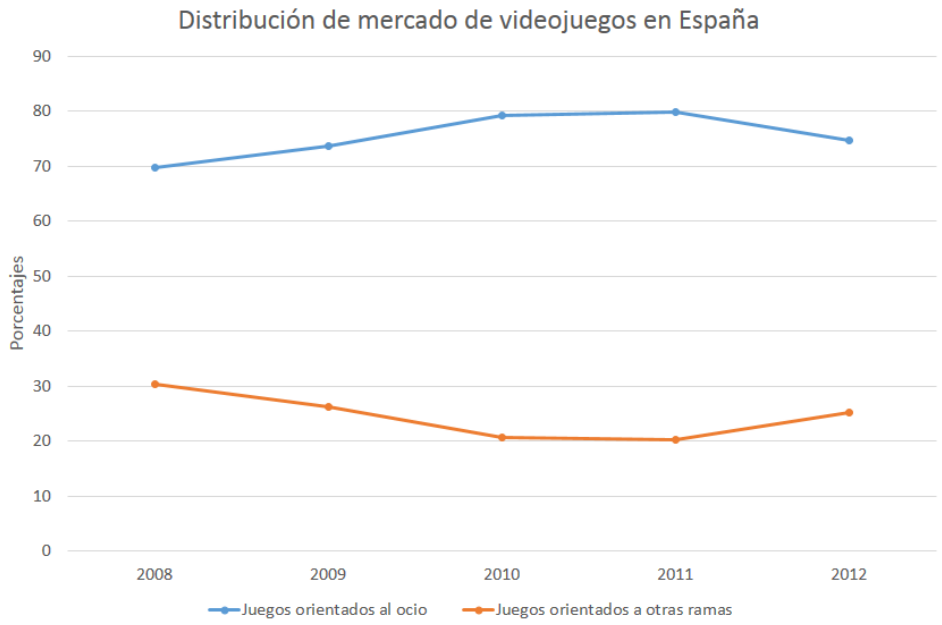

Figura 2.1: Distribución de mercado. Datos obtenidos de (ADeSe, n.d.).

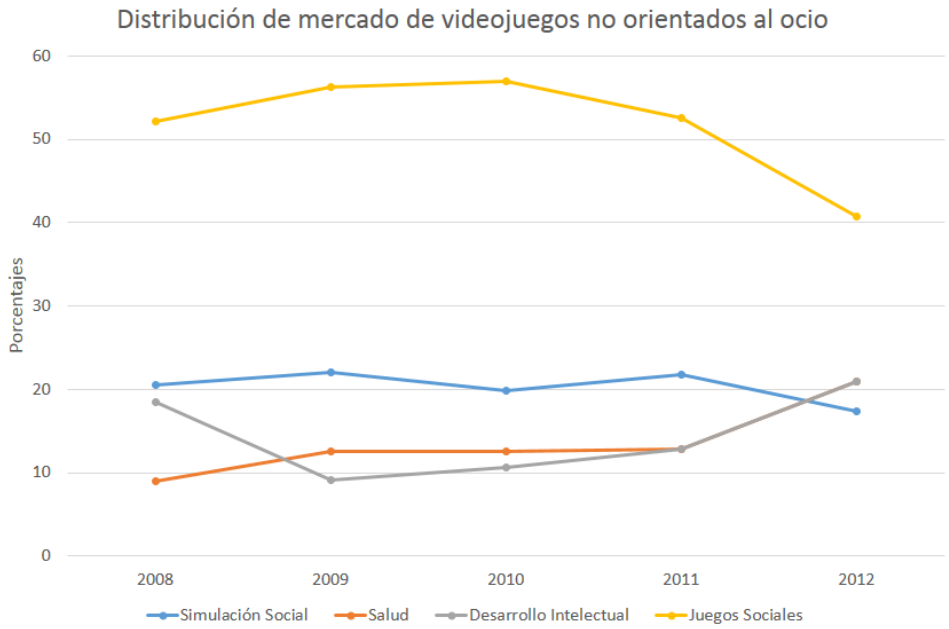

Figura 2.2: Distribución de mercado. Datos obtenidos de (ADeSe, n.d.).

La relevancia de los videojuegos es tal que en los últimos años se ha producido una cierta popularización de la actividad de creación de un videojuego (Overmars, 2004; Resnick et al., 2009). Existen multitud de herramientas libres o gratuitas que permiten desarrollar un videojuego, ya sean muy potentes y complejas que permiten desarrollar proyectos de apariencia profesional, o más simples e intuitivas que permiten un desarrollo rápido y de menor coste (Mayo, 2007). Un claro ejemplo de esta popularización es que con el paso de los años, los profesores empiezan a usar videojuegos didácticos, incluso creados por ellos mismos, para facilitar el proceso de aprendizaje a sus alumnos. Otros profesores han utilizado herramientas como Scratch o Alice para mejorar sus estrategias de enseñanza mediante la actividad de creación de videojuegos.

Uno de los principales problemas que se derivan de esta tendencia es la necesidad de encontrar recursos gráficos adecuados para el desarrollo de un videojuego. A pesar de que existen multitud de recursos gráficos en la red, en muchos casos es difícil adaptarlos para su uso en videojuegos o pueden tener un coste elevado. Además este problema crítico si no se dispone de dinero para las correspondientes licencias o para realizar desarrollos y adaptaciones propias. La primera impresión que un videojuego debe ofrecer es muy importante, y es por esto que los recursos gráficos son muy relevantes en el proceso de creación de un videojuego. Dentro de este tipo de recursos, el más problemático es el de los personajes animados, puesto que los modelos existentes pueden no adecuarse a nuestras necesidades. En este punto es donde entra en juego *eCharacter*.

eCharacter es una herramienta con la cual los desarrolladores de juegos pueden configurar sus propios modelos animados, partiendo de una serie de modelos base. Además incorpora un sistema que permite a los usuarios añadir fácilmente nuevos modelos y configurar sus opciones de personalización. El uso de esta herramienta permite la reducción de costes a la hora de desarrollar un nuevo juego, tanto en tiempo como en dinero.

2.1. Introducción al mundo 3D: términos y conceptos básicos

En este documento se hace referencia a algunos términos relacionados con el mundo 3D que es posible que no todos los lectores conozcan. La finalidad de esta sección es hacer un breve resumen de los conceptos básicos necesarios para la correcta comprensión de este documento, además de incluir un glosario de los términos más referenciados en este documento. Si el lector tiene conocimientos básicos sobre el mundo 3D y el modelado y desarrollo de personajes 3D podría omitir la lectura de esta sección.

2.1.1. Conceptos básicos

- **Escena**

La escena en el mundo 3D contiene los objetos que componen el mundo virtual (personajes, elementos estáticos, etc.). Normalmente la escena se compone utilizando una estructura jerárquica tipo árbol, lo que facilita la construcción de nuevos objetos en la escena mediante composición de otros objetos más simples.

A cada nodo del árbol de la escena se le puede aplicar un conjunto de transformaciones geométricas, incluyendo traslación rotación y escalado, lo que permite alterar la apariencia de un determinado objeto de forma programática (es decir desde código).

Una escena posee un sistema de coordenadas de tres dimensiones gracias a los cuáles se pueden situar los objetos en la posición deseada dentro de la escena. Estos ejes de coordenadas se representan tal y como muestra la siguiente imagen.

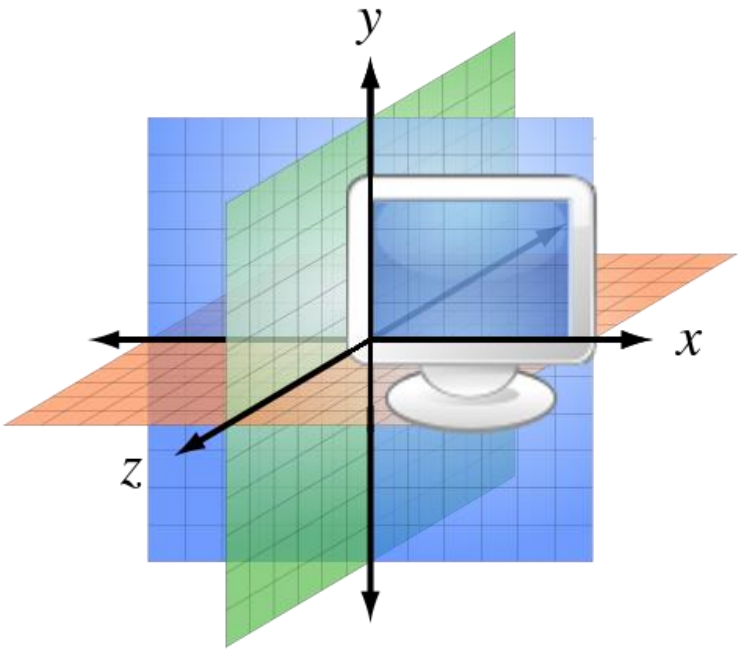

Figura 2.3: Ejes de coordenadas

Una escena posee también una cámara, que determina qué parte concreta de la escena es visible para el usuario. Las cámaras se definen mediante un punto y dos

vectores. El punto indica la posición de la escena en la cual se coloca la cámara. El primero de los vectores es el vector de dirección que indica en qué dirección apunta la cámara y por tanto qué parte de la escena es visible para el usuario. El segundo de los vectores determina la inclinación de la cámara. Además, la escena también incluye un sistema de iluminación que dota de realismo a la escena, pudiendo conseguir efectos de sombras. En el siguiente diagrama se ilustra cómo quedan definidos los elementos dentro de una escena.

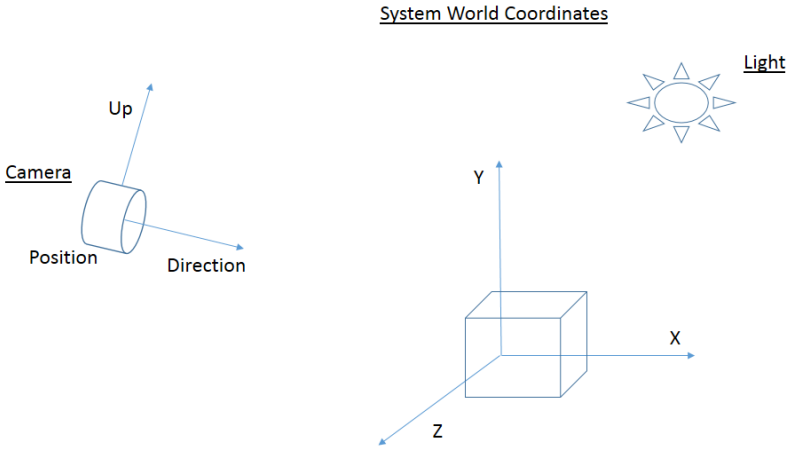

Figura 2.4: Esquema de escena

Una vez que se ha definido los valores de la cámara y se ha establecido la iluminación, se realiza un proceso de presentación gráfica final (habitualmente se usa el anglicismo *renderización*) en el cuál se genera la imagen de la escena que finalmente ve el usuario.

- **Modelos y Mallas**

Los entornos 3D suelen proporcionar funcionalidad para crear objetos de la escena sencillos utilizando geometría básica como, por ejemplo, prismas, cilindros, pirámides o esferas. Sin embargo, este tipo de construcciones resulta insuficiente a la hora de representar formas complejas, como pueden ser un personaje en el que sea posible distinguir los rasgos faciales.

Para suplir esta carencia los entornos de visualización 3D permiten utilizar objetos en la escena generados con una herramienta de modelado visual 3D externa, optimizada para la creación de modelos mucho más complejos, y orientada a diseñadores gráficos más que a programadores. Algunas de las herramientas de modelado 3D más conocidas son Maya, Blender o 3D studio. Es a este concepto de objeto 3D complejo y animado generado con una herramienta específica a lo que nos referiremos a lo largo de este documento como *modelo.*

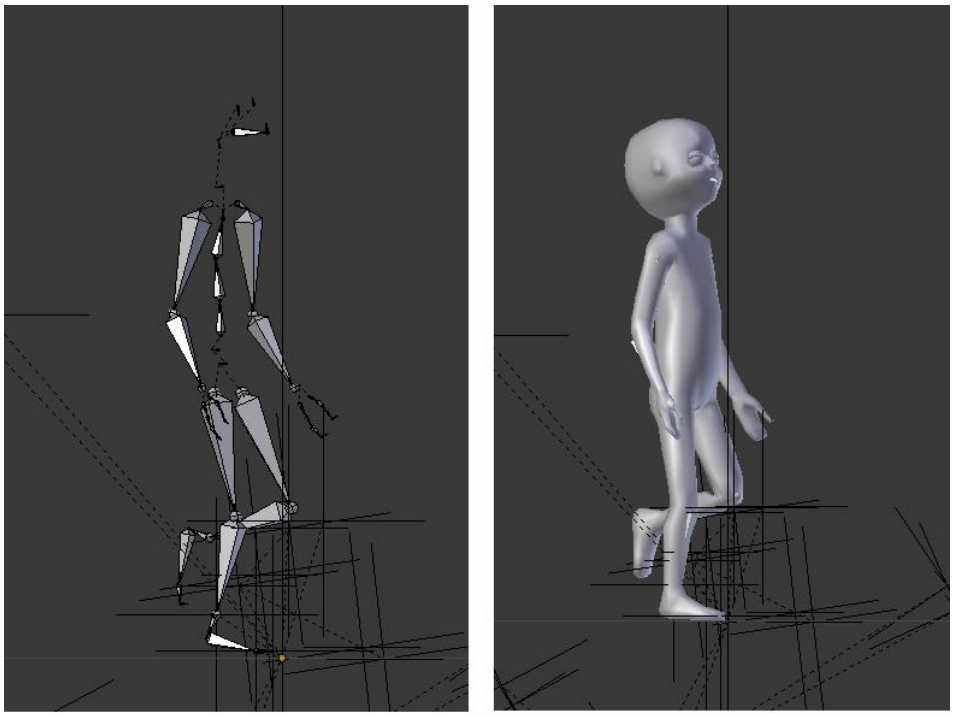

Figura 2.5: Modelo en Blender

Un modelo suele tener tres componentes básicos: un esqueleto, una o varias mallas, y una o varias animaciones.

El esqueleto define la estructura interna del modelo, y supone el componente básico sobre el que crear animaciones. Al igual que el esqueleto humano, el esqueleto de un modelo está compuesto por huesos. Para conseguir esa representación, se hace uso de una estructura de árbol, gracias a la cual se pueden establecer dependencias entre huesos y conseguir una mejor organización de los mismos. Los huesos son las unidades básicas con las que *eCharacter* puede operar, pudiendo realizar sobre ellos operaciones de escalado para alterar la apariencia de los modelos cargados.

Una malla 3D es una colección de triángulos y vértices que aproximan una superficie 3D. En caso de que el modelo sea animado y posea esqueleto, la malla es el componente visual que recubre al esqueleto y que da forma al modelo (algo parecido a los músculos del cuerpo humano). En *eCharacter* también se hace uso de submallas para representar partes del modelo que pueden ser intercambiables, como pueden ser tipos de peinados o faldas. Estas submallas son mallas más pequeñas que no forman parte de la malla principal que recubre al esqueleto, y están asociadas a algún hueso del esqueleto.

Las animaciones de un modelo (por ejemplo, caminar, correr, o saltar) se generan a partir de interpolar la posición y rotación de los huesos del esqueleto en diferentes instantes del tiempo. Algo parecido a lo que sería definir la física de las articulaciones que rigen el movimiento del cuerpo humano.

- **Materiales y texturas**

Cuando se modela un objeto 3D, su superficie queda con un color por defecto uniforme y liso. Con el uso de materiales y texturas se puede dotar al modelo 3D de un

mayor realismo. Siguiendo la analogía con el cuerpo humano, estos elementos serían la piel del modelo.

Las texturas de los modelos suelen definirse como ficheros aparte, normalmente en formato imagen (por ejemplo jpg). Esto permite cambiar la textura que recubre al modelo de forma sencilla desde un programa. *eCharacter* explota esta característica para conseguir una gran variabilidad en la configuración de personajes sin que esto suponga un aumento excesivo del coste.

2.1.2. Glosario de términos

- Escena: La escena en 3D contiene toda la información necesaria para identificar y posicionar todos los modelos, luces y cámaras para su visualización.

- Esqueleto: Estructura de árbol que define la organización y agrupación de los huesos de un modelo. Permite crear modelos animados.

- Hueso: Elemento básico de un esqueleto.

- Malla: Colección de triángulos y vértices que aproximan una superficie 3D.

- Textura: Archivo de imagen que se aplica sobre una malla para conseguir una mayor sensación de realismo.

3. Estado del arte

En esta sección se realiza un análisis de otros sistemas de configuración de personajes ya existentes como sistemas independientes o incluidos en juegos comerciales para identificar qué ideas y funcionalidades aportan y para qué están dirigidos. Pero antes de empezar con el análisis, para clarificar el campo de trabajo, hay que definir qué se entiende por un configurador de personajes y cómo funciona.

3.1. ¿Qué es un configurador de personajes?

Un configurador de personajes es una herramienta que permite la creación de distintos personajes mediante la personalización o elección de características particulares partiendo de un mismo conjunto de modelos base.

Los configuradores de personajes ofrecen una serie de modelos base a partir de los cuáles el usuario toma una serie de decisiones para configurarlo y adecuarlo lo máximo posible a sus necesidades o a su objetivo final.

Los parámetros que suelen ser configurables durante el proceso de personalización del modelo se pueden clasificar en dos grupos. En el primer grupo se pueden incluir todas las decisiones que el usuario tome sobre el modelo relacionado con su aspecto físico. Con este tipo de decisiones el usuario puede lograr, por ejemplo, diferentes complexiones físicas o distintos rasgos étnicos.

El segundo grupo de decisiones son las que están relacionados con la indumentaria (p.e. ropa, complementos) del modelo. Este tipo de opciones suelen ser las más comunes en los configuradores de personajes. Gracias a ellas, normalmente el usuario podrá elegir entre distintos tipos de texturas que se aplican sobre el modelo base, además de poder elegir el color deseado para dicha textura.

El número posible de configuraciones finales que el usuario puede obtener, depende tanto de la cantidad de modelos base que ofrece la aplicación como del número de opciones disponibles a la hora de elegir entre las distintas texturas o complexiones.

3.2. Análisis del estado del arte

Existen un gran número de configuradores de personajes diseñados específicamente para cumplir una serie de tareas, ya sea en videojuegos o en formato web.

En la industria de los videojuegos se fue introduciendo en la década pasada esta característica debido a la creciente demanda de los usuarios. Uno de los usos principales es permitir al usuario configurar el aspecto del personaje que lo representa en el mundo virtual, conocido como *avatar*. La posibilidad de crear un avatar personalizado e introducirlo en el videojuego permite al usuario identificarse más con la historia en la que se desarrolla el videojuego. Estos sistemas han tenido tanto éxito, que hoy en día la mayoría de los videojuegos lo incorporan.

Esta tendencia se fue extendiendo hasta llegar a la web donde también se han desarrollado estos tipos de sistemas de configuración de personajes con distintas

finalidades. En ellos se ofrece la posibilidad de crear avatares personalizados para el uso en chat o desarrollar diferentes personajes y poder incorporarlos, por ejemplo, a comics.

A continuación se va a hacer un estudio más detallado de estos sistemas, agrupados en diferentes categorías. El criterio que se ha seguido a la hora de definir estas categorías es el objetivo final y el uso que se les da a los modelos generados por estos configuradores. La primera de estas categorías son los configuradores integrados en los juegos comerciales, donde el uso del modelo queda restringido al juego en cuestión. La segunda categoría son los configuradores orientados a la creación del contenido multimedia, gracias a los cuáles se pueden configurar modelos para su posterior uso en desarrollo de viñetas animadas. Y por último, la categoría de configuradores de avatares web, que permiten el desarrollo de nuestros avatares para su uso en redes sociales.

3.2.1. Configurador de personajes en juegos comerciales

Uno de los principales objetivos que se buscan cuando se decide incorporar un configurador de personajes en los videojuegos, es buscar una mayor personalización, y que ésta ayude a que la identificación con el juego por parte del usuario sea más plena. Dentro del uso de estos configuradores, se pueden distinguir dos tendencias.

En la primera de ellas, se busca una mayor sensación de realismo, ofreciendo al usuario un control total. Al usuario se le ofrece la oportunidad de configurar multitud de aspectos ya sean físicos, étnicos, de vestimenta o incluso de comportamiento. Ejemplos de este tipo de configuradores son los videojuegos “FIFA” y “Los sims”. En ellos, se parte de un modelo base y se pueden realizar cualquier tipo de modificación posible para que el personaje, que finalmente se desarrolle, sea lo más parecido a los intereses iniciales del usuario. Este tipo de configuradores aportan un gran número de combinaciones posibles, haciendo que la cantidad de personajes distintos que se puedan crear sea muy grande.

Figura 3.1: Configurador Fifa

Figura 3.2: Configurador Los Sims

La segunda tendencia, igualmente versátil, busca que el resultado final de las configuraciones de los usuarios sea más esquemáticos (en vez de muy realista). Un ejemplo claro de esta tendencia es el configurador de personajes de la Wii (Wii Mii).

Figura 3.3: Configurador Wii Mii

En este configurador se ofrece una gran posibilidad de combinaciones. Pero el resultado de todas ellas tiene un patrón y una apariencia gráfica común, que hace que los personajes que han sido configurados con esta herramienta se asocien rápidamente a ella y sean fácilmente reconocibles.

Figura 3.4: Configurador Wii Mii

3.2.2. Configurador de personajes para la creación de contenido multimedia

La finalidad de este tipo de configuradores es proporcionar al usuario una herramienta con la cual puedan crear sus propios contenidos multimedia, como pueden ser comics o viñetas animadas. Principalmente este tipo de configuradores se encuentran en la web.

Este tipo de configuradores no destacan especialmente ni por su realismo ni por su abanico de posibilidades a la hora de personalizar el modelo base. Habitualmente cubren los mismos campos que los anteriores (complexión física, atuendo, rasgos étnicos, etc.) pero ofrecen al usuario menos libertad de personalización. A pesar de ello cubren todas las necesidades más demandadas por los usuarios a la hora de crear sus propios personajes.

Como ya se ha mencionado estos configuradores no destacan por su realismo, puesto que su finalidad es la realización de viñetas animadas o comics. Debido a esto, los personajes generados con este tipo de configuradores tienen un estilo de dibujos animados o "*cartoon*".

Algunos ejemplos de estos tipos de configuradores que se pueden encontrar en la web son "GoAnimate" (http://goanimate.com/) o "Bitstrips" (http://www.bitstrips.com/create/character/)

Figura 3.5: Configurador GoAnimate

3.2.3. Configurador de personajes para la creación de avatares web

Este último bloque de tipo de configuradores está dedicado a aquellos cuya principal finalidad es la creación de avatares personalizados para su posterior uso en redes sociales.

Hoy en día las comunicaciones en línea y el uso de las redes sociales es algo muy común, y debido a esto, los usuarios deben elegir cuál será la imagen que les represente de cara al mundo exterior. Estas herramientas ofrecen a los usuarios la posibilidad de personalizar al máximo la imagen que van a mostrar al público en la red.

Los personajes creados por este tipo de herramientas pueden ser exportados y utilizados posteriormente en redes sociales tales como foros o servicios de mensajería instantánea. Algunos ejemplos de estos tipos de configuradores son “Meez” (http://www.meez.com) o “WeeMee” (http://www.weeworld.com).

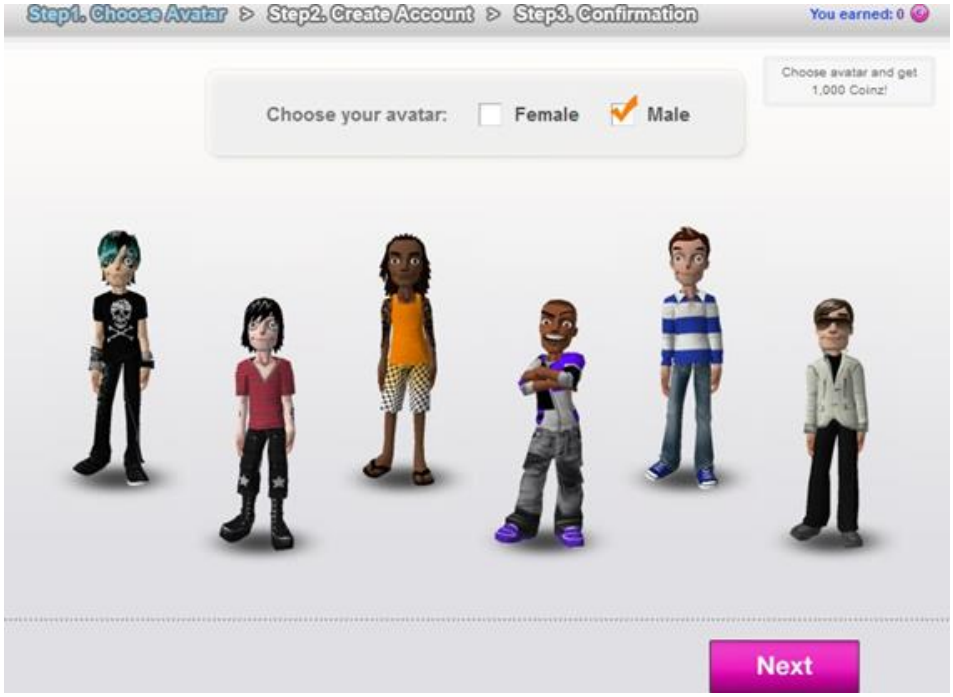

Figura 3.6: Configurador Meez

4. Funcionalidades

Para la elección de las funcionalidades que se ofrecen en *eCharacter* se realizó un estudio previo sobre los sistemas similares ya existentes. El principal objetivo de este estudio fue analizar cuáles eran los puntos fuertes de cada uno de ellos, así como las principales funcionalidades que se ofrecían al usuario. Gracias a ello se pudo recopilar la información necesaria para elegir correctamente las funcionalidades que se iban a incluir en *eCharacter*.

En este capítulo se van a comentar todas las funcionalidades ofrecidas por el sistema. Estas funcionalidades se han agrupado en distintas categorías que se comentarán a continuación.

4.1. Agrupación de los modelos en familias

Una de las funcionalidades que se decidió ofrecer es agrupar todos los modelos en familias. El término familia hace referencia a un conjunto de modelos base que presentan características y estilo comunes.

El motivo de realizar esta agrupación, es que cualquier usuario pueda crear su propia familia y la pueda integrar por medio de un documento XML de una manera muy sencilla. Así se consigue tener cada modelo separado, pudiendo tener, en la aplicación, distintas familias creadas por distintos desarrolladores de modelos.

El sistema de familias se diseñó para que la integración de nuevas familias fuera sencilla, permitiendo a distintos diseñadores poder contribuir al proyecto y aumentar la base de modelos disponibles para crear nuevos personajes.

Cada familia se puede configurar de manera independiente. En el apartado de diseño interno de la aplicación, se hace referencia a qué parámetros son configurables dentro de cada familia y a cómo es el proceso de configuración de una nueva familia. La idea principal es que cada familia tiene varias etapas, y dentro de cada etapa hay distintas subetapas en las que se puede realizar un intercambio de texturas o de mallas (para una definición de estos términos consultar apartado *Introducción al mundo 3D* del capítulo 2 *Introducción*).

4.2. Escalación

Un configurador de modelos 3D es importante que disponga de un sistema para poder escalar determinadas partes del cuerpo.

eCharacter dispone de un sistema de escalación en el que el usuario es capaz de poder escalar varias partes del cuerpo de dos maneras:

- En la primera se puede seleccionar una de las complexiones predefinidas (p. Ej. Alto, delgado o grueso). Estas complexiones vienen definidas en un documento XML propio del modelo. Es decir, dentro de una familia, puede haber dos modelos con distintas complexiones predefinidas. Con esto, se consigue que el usuario pueda elegir de una manera sencilla, distintos aspectos físicos para su modelo.

- La otra manera de escalar es mediante lo que se llama “escalación libre”. Este modo permite al usuario escalar de una manera más fina y exacta para lograr

un modelo que se parezca lo máximo posible a lo que busca. Mediante “sliders”, el usuario puede seleccionar qué parte del cuerpo quiere escalar y cuánto la quiere escalar. También, estos sliders están definidos en el documento XML correspondiente al modelo. Cómo definir las complexiones predefinidas o los valores y huesos a escalar, se explicará en el apartado de diseño interno de la aplicación.

4.3. Texturas y mallas

Otra de las funcionalidades que ofrece *eCharacter* es permitir la personalización del modelo base a través del uso de texturas y mallas. Para ello se ofrecen dos operaciones básicas, a través de las cuáles el usuario podrá configurar su modelo.

La primera de ellas es el intercambio de texturas entre todas las disponibles para ese modelo. Estas texturas están agrupadas en las distintas etapas que componen la familia a la que pertenece el modelo. Durante el proceso de diseño se decidió distinguir cuatro tipos de texturas diferentes que se comentarán en el apartado correspondiente al Diseño interno de la aplicación.

Además de la posibilidad del intercambio de texturas, también se le ofrece al usuario la posibilidad de cambiarle el color a la textura seleccionada. Con esto se consigue aumentar el número de combinaciones posibles que puede configurar el usuario.

Buscando aumentar la potencia de configuración se ofrece también la posibilidad de realizar las mismas operaciones comentadas anteriormente sobre las mallas auxiliares que pueda tener el modelo base (tales como pelo, faldas,...). Así pues se puede intercambiar entre las distintas mallas ofrecidas, además de elegir el color deseado.

Al igual que en las complexiones predefinidas comentadas en el apartado anterior, el usuario también puede decidir cuáles son las texturas y mallas por defecto que aparecerán en el modelo base. Esto se consigue mediante la edición del fichero XML que define el modelo. La estructura de este fichero se comenta en profundidad en el apartado 5.1.2 (*Modelo subyacente*) del capítulo 5 *Diseño*.

4.4. Exportación

eCharacter ofrece la posibilidad de integración de los modelos configurados en videojuegos o herramientas externas. Esta funcionalidad se puede ofrecer gracias a un sistema de exportación que permite generar una secuencia de imágenes del modelo configurado.

En el proceso de exportación se pueden configurar tres parámetros. El primero de ellos es las animaciones del modelo que el usuario desea exportar. El segundo de los parámetros configurables es la calidad. La calidad viene medida en FPS (*Frames por segundo*), que son el número de capturas que hay que realizar por cada segundo de animación. A valores mayores de FPS, se realizan un mayor un número de capturas y por tanto la calidad de la animación exportada es mayor. El tercero de los parámetros que se pueden configurar son las cámaras. La importancia de las cámaras en el proceso de exportación es vital, ya que son las que determinan la vista final con la que se exporta el modelo. Con el uso y la manipulación de las cámaras, *eCharacter* permite realizar exportaciones en 2D o 3D, según cuál sea la vista de cámara

configurada. Esta posibilidad se comenta más detenidamente en el apartado siguiente (*Cámara*).

Gracias a estos tres parámetros el usuario puede exportar su modelo configurado con cualquiera de las posibles combinaciones de estos tres elementos. Todo el proceso de exportación es un proceso automatizado, en cuanto a que el usuario, una vez seleccionados todos los parámetros, no tiene que realizar ninguna acción más. Solo tiene que esperar a que finalice y el resultado de la exportación generará un fichero ZIP con todas las capturas de las animaciones en su interior.

El proceso de exportación permite al usuario dos modos:

- Un modo en el que el usuario simplemente tiene que seleccionar las animaciones para las que quiere realizar la exportación. Este modo exportará todas las animaciones seleccionadas con una calidad por defecto de 6 FPS y con la vista de cámara que esté configurada en ese momento.

- El otro modo es el modo avanzado. En este modo el usuario tiene más libertad para configurar el proceso. Lo que el usuario debe hacer es seleccionar cuales son las animaciones a exportar, en qué calidades quiere realizar la exportación y con qué vistas de cámaras.

4.5. Cámaras

Durante el proceso de configuración del personaje, el usuario puede desear tener un control total de la cámara para ver si el resultado de sus decisiones es el deseado. Para ello, se ofrece un sistema de control de cámara, con dos modos diferentes adecuados según las necesidades del usuario.

En un primer modo más sencillo se ofrecen únicamente los controles de rotación de la cámara sobre la vista original. De esta manera el usuario puede lograr una visión del modelo desde distintos puntos de vista.

Si el usuario quiere ir más allá, existe un segundo modo de control de cámara que permite un movimiento libre de la misma. En este modo, el usuario puede mover la cámara hacia cualquier posición del espacio 3D pudiendo configurar así la vista que desee. Este modo de control de cámara es importante en el proceso de exportación ya que gracias a estos controles, el usuario puede configurar la vista con la cual se exportarán las animaciones deseadas.

Además de estos dos modos de control, el usuario puede definir una serie de cámaras que son propias de la familia de modelos. Estas cámaras aparecen en el menú de exportación para su posible selección tal y como se cuenta en el apartado anterior. Es este control de cámaras por parte del usuario, la que ofrece la posibilidad de configurar cámaras que permitan la exportación del modelo configurado tanto en 2D como en 3D.

4.6. Internacionalización

eCharacter incorpora un sistema de internacionalización que permite configurar el idioma en el cuál se muestran todos los textos en la interfaz. Gracias a este sistema,

el usuario puede añadir nuevos idiomas posibles de la aplicación con sólo editar un par de ficheros y sin tener que tocar código alguno.

La idea principal de este sistema de internacionalización es muy sencilla. Al instalar la aplicación el usuario puede seleccionar el idioma seleccionado entre la lista de idiomas que se ofrecen por defecto (español, inglés y francés). Una vez instalada la aplicación el usuario puede cambiar el idioma de la aplicación en cualquier momento. Este idioma elegido se representa dentro de la aplicación como un identificador único de la forma es_ES o en_EN. Cada cadena de texto que pueda ser mostrada en la interfaz lleva un identificador único. En un fichero externo e independiente para cada idioma se realiza la relación entre la clave, que es el identificador único, y el valor, que es la cadena de texto que se debe mostrar en la interfaz según el idioma elegido por el usuario.

5. Diseño

Una vez que se tenía decidido cuáles iban a ser las funcionalidades clave de la aplicación, comenzó un proceso de diseño de la experiencia de usuario. En este proceso de diseño se trató el flujo de la aplicación así como el estilo y la distribución de la interfaz. En un proceso separado se diseñó la arquitectura interna de la aplicación para dar soporte a todas las funcionalidades ofrecidas.

En esta sección abordamos detenidamente aspectos relevantes sobre el diseño de *eCharacter.* En un primer bloque se analizan todos los aspectos relacionados con la arquitectura interna de la aplicación, como son el modelo subyacente desarrollado para permitir la inclusión de nuevas familias de modelos al sistema, y las estructuras de datos internas utilizadas para proporcionar las funcionalidades comentadas anteriormente. En el segundo bloque se describe el diseño de la interfaz de usuario así como las principales estrategias de diseño seguidas. Y por último se ilustran los casos de usos más relevantes.

5.1. Arquitectura interna de la aplicación

Esta sección se divide en cuatro apartados, que tratan los puntos claves de la arquitectura de *eCharacter.*

El primer punto clave que se trata es el modelo vista-controlador, que es la clave para entender las comunicaciones entre los distintos módulos que forman la aplicación. El segundo punto es el modelo subyacente que se ha desarrollado para permitir que la aplicación sea flexible y configurable por parte del usuario. Gracias a este modelo subyacente, el usuario podrá incorporar fácilmente nuevas familias de modelos así como extender aquellas ya existentes. El tercer concepto clave son las estructuras de datos internas diseñadas para soportar y almacenar todos los cambios que realice el usuario durante el proceso de configuración del modelo. Con la ayuda de estas estructuras, la aplicación es capaz de mantener la consistencia de los datos durante todo el proceso de configuración. Y por último, se trata el sistema de exportación, gracias al cual el usuario puede exportar el modelo configurado para poder utilizarlo de forma externa a la aplicación.

5.1.1. Modelo vista-controlador

El modelo vista-controlador es un patrón de arquitectura de software en el cuál se separan el modelo de datos y su manipulación por un lado y la interfaz de usuario por otro. Además existe un módulo adicional, el controlador, que es el encargado de gestionar las comunicaciones entre los módulos y las interacciones por parte del usuario a través de la interfaz. Este patrón de diseño facilita la reutilización de código así como la separación de conceptos, características que ayudan en el desarrollo de una aplicación y su posterior mantenimiento. En el caso de *eCharacter,* se eligió este modelo de arquitectura porque era el que más se adecuaba a las necesidades, aunque se le han realizado algunas pequeñas modificaciones.

El funcionamiento del modelo vista-controlador es el siguiente. El usuario realiza alguna interacción con la interfaz, como puede ser pulsar un botón. El controlador recibe la notificación de que el usuario ha realizado alguna interacción con la interfaz y la procesa. El proceso de dicha interacción suele implicar una modificación o consulta sobre el modelo de datos existente. Una vez que el controlador ya ha indicado al

modelo de datos que debe actualizarse, hay que mostrar la nueva información al usuario a través de la interfaz. En este caso, es el modelo el que proporciona los datos que tiene que mostrar a la interfaz, actualizando el contenido que se visualiza. De esta forma ya se ha completado una iteración y la interfaz queda a la espera de notificar nuevas interacciones por parte del usuario.

Como ya hemos comentado anteriormente, el modelo de arquitectura desarrollado en *eCharacter* se basa en el modelo vista-controlador pero se han introducido algunas modificaciones, sobre todo en el módulo del controlador debido a su complejidad. A continuación se detalla información acerca de cada uno de los módulos de la aplicación y la funcionalidad que desempeñan.

- Módulo de interfaz: Este módulo es el encargado de proporcionar una visión del estado del modelo de datos. En este caso la interfaz se encarga de mostrar al usuario el estado del modelo que está configurando, conforme con las decisiones que ha ido tomando. Además es el encargado de capturar todas las interacciones por parte del usuario y comunicarlas al controlador para que realice las operaciones correspondientes. Estas interacciones pueden ser, por ejemplo, realizar un cambio de una textura o iniciar el proceso de exportación. En *eCharacter* el componente de interfaz está implementado sobre el motor 3D jMonkey Engine 3.0, del que se hablará más adelante.

- Módulo de datos: Este módulo es el encargado de mantener el modelo de datos acorde con las decisiones tomadas por el usuario. Para ello se ha diseñado una serie de estructuras de datos internas que permiten almacenar y mantener fácilmente todas las decisiones que ha tomado el usuario. Estas estructuras se cuentan y analizan de una forma más detallada en el apartado *Estructura de datos internas* de esta misma sección. Además y como ya se ha contado en secciones anteriores, la distribución de los distintos modelos que contiene la aplicación se ha realizado en dos niveles. En el primer nivel los modelos que comparten una serie de características se agrupan en familias. Ya en segundo nivel es donde se encuentran los modelos seleccionables para su posterior configuración. Cada uno de estos niveles contiene información propia que hay que tratar por separado. Por ejemplo, el nivel de las familias contiene información acerca de lo que la interfaz debe mostrar y en el nivel de los modelos se incluye toda la información relacionada con las texturas o submallas asociadas a dicho modelo en concreto. Como consecuencia de todo esto, el módulo de datos está dividido en tres submódulos, para facilitar así la organización y mantenimiento de la información.

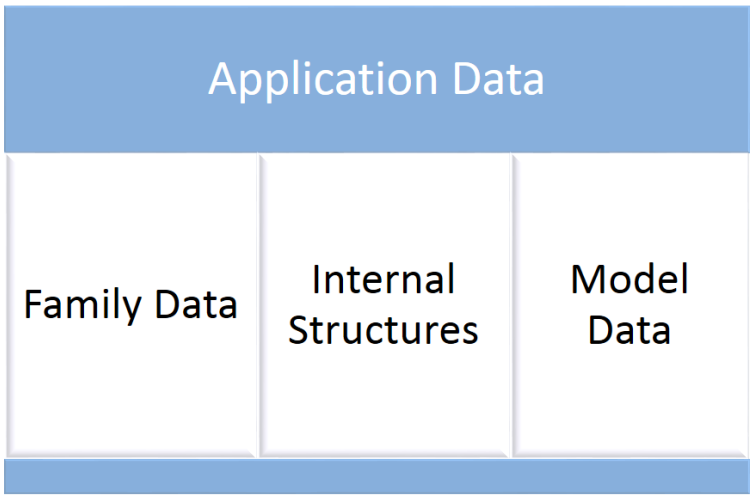

Figura 5.1: Módulo de datos

- Módulo de control: Este módulo es el encargado de gestionar todas las interacciones que realice el usuario a través de la interfaz y notificar al modelo de datos que se actualice en consecuencia con la acción realizada por el usuario. Al igual que en el módulo de los datos, se ha decidido dividir el módulo de control en tres submódulos diferentes que se comunican con los distintos submódulos de datos comentados anteriormente. El primer submódulo es el encargado de interactuar con los datos asociados a la familia de modelos que el usuario haya seleccionado. El segundo de ellos es el encargado de comunicarse y manipular los datos relacionados con el modelo seleccionado por el usuario para su configuración. Y por último, está el submódulo que se encarga de controlar la escena gráfica y todas las decisiones que haya tomado el usuario. Además existe una capa superior que es la encargada de realizar las comunicaciones con el módulo de la interfaz y delegar a cada uno de los submódulos según la operación requerida por el usuario. El siguiente gráfico ilustra esta distribución.

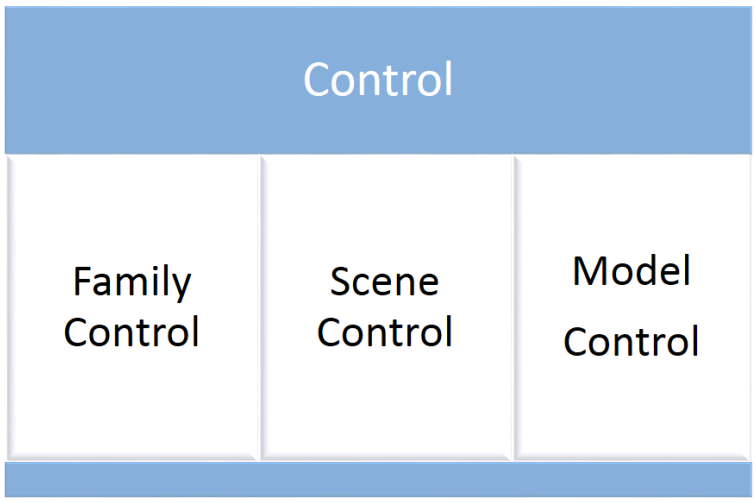

Figura 5.2: Módulo de control

Para finalizar este punto del modelo vista-controlador se muestra una figura en la que se puede observar cómo se realiza la comunicación entre los distintos módulos y submódulos que forman la arquitectura interna de la aplicación.

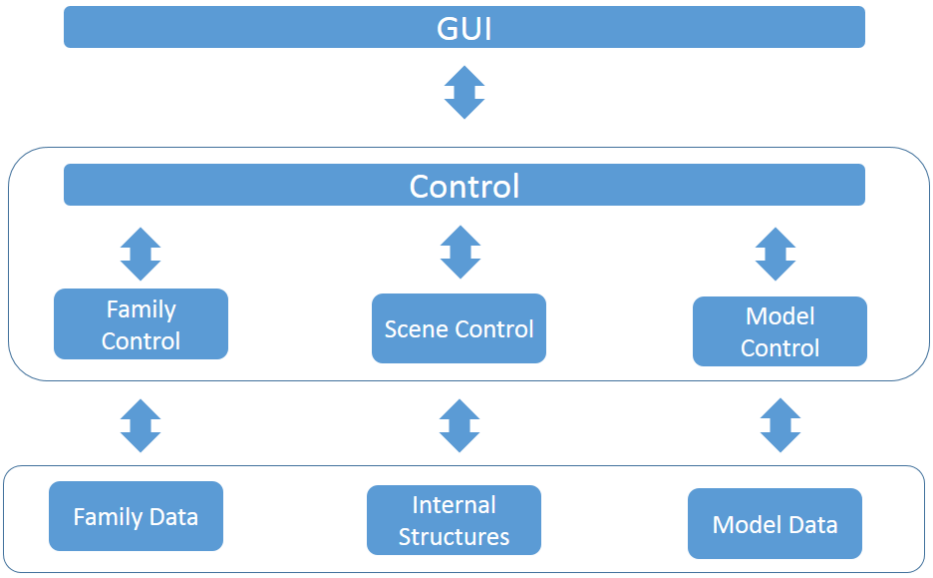

Figura 5.3: Comunicación entre módulos

En la siguiente figura, se muestra el diagrama de secuencia MVC utilizado en la aplicación.

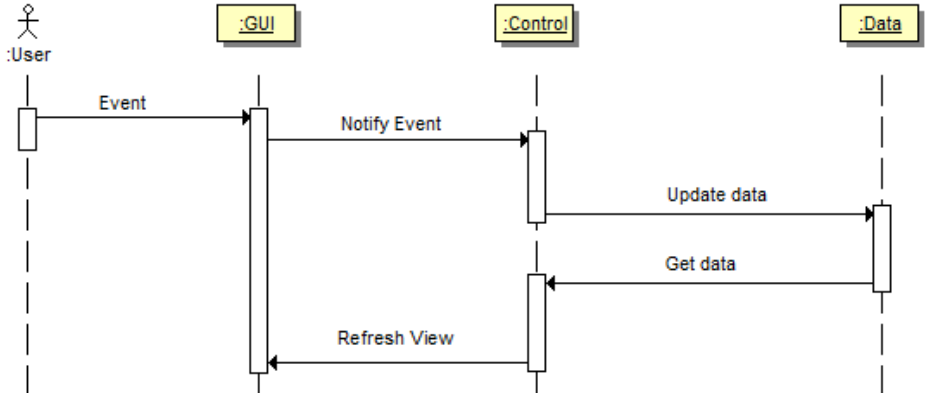

Figura 5.4: Diagrama de secuencia MVC

5.1.2. Modelo subyacente

Como ya se ha comentado en otras secciones, una de las principales características de *eCharacter* es su flexibilidad. Con *eCharacter* los usuarios son capaces de generar sus propias familias de modelos y adecuar la interfaz de configuración a sus necesidades. Para que esto sea posible sea ha diseñado un modelo de datos basándose en dos conceptos clave: familia y modelos. Una familia es un conjunto de modelos que tiene una serie de características comunes tales como el esqueleto, animaciones o texturas disponibles para su intercambio.

Para permitir la fácil integración de nuevas familias de modelos se han desarrollado dos *XML Schema* que definen la estructura que deben tener los ficheros para que puedan ser interpretados por la aplicación. Tal como se define en la página

web del *Worl Wide Web Consortium* (www.w3.org), también conocido como *W3C,* un fichero *XML Schema* define la estructura de bloques que debe tener un documento XML concreto. En este *XML Schema* se definen qué tipos de elementos y atributos pueden aparecer, así como el orden y número de los mismos. Para obtener más información acerca de los *XML Schema* se recomienda visitar la página de *W3Schools* (www.w3schools.com/schema) donde se pueden encontrar varios manuales y tutoriales.

El primero de estos *Schemas* define a la familia y tiene como puntos importantes la información común a todos los modelos, definición de las etapas de configuración y la lista de modelos pertenecientes a dicha familia. El segundo *XML Schema* define los modelos propios de cada familia y contiene información acerca del esqueleto del modelo y las texturas o submallas que se le pueden aplicar. A continuación se explica más detalladamente cada uno de ellos.

5.1.2.1. Familias

La estructura que define a una familia consta de tres partes básicas: información relativa a la familia en sí, también llamados metadatos: secuencia de etapas que forman el proceso de configuración del modelo: y una lista de los modelos que pertenecen a dicha familia. A continuación se explican cada uno de ellos con más detalle.

Los metadatos almacenan toda la información relativa a la familia que no es imprescindible para el funcionamiento de la aplicación, tal como puede ser el nombre de la familia, una breve descripción o el autor de la misma. Además también contiene información acerca de la licencia de distribución y la ruta donde se encuentran los ficheros relacionados con la internacionalización de la familia. La principal funcionalidad de los metadatos, es agrupar la información acerca de la familia en un único lugar. Parte de esta información, como es el nombre de la familia y la descripción, es visible desde la interfaz de selección de familias.

El segundo elemento son las etapas. Esta parte es la más importante a la hora de definir una nueva familia ya que en él se indica el flujo que sigue el proceso de configuración para un modelo de esa familia. Para ello, el usuario debe definir la secuencia de etapas que forman dicho proceso. *eCharacter* ofrece tres tipos diferentes de etapas:

- Etapas de escalado: En este tipo de etapas se pueden realizar todas las operaciones relacionadas con la escalación de los huesos del modelo. Para configurar este tipo de etapas basta con que el usuario defina un identificador para cada controlador, asociado a un hueso en concreto.

- Etapas de intercambio de texturas y submallas: Este tipo de etapas se han definido con el nombre de *MultiStage*. Cada *MultiStage* está compuesta por una secuencia de subetapas, llamadas *SubStage*. Gracias a esta distribución, se ofrece al usuario la posibilidad de tener una mayor libertad a la hora de crear sus propias interfaces ya que puede anidar las etapas según sus necesidades.

- Etapa de exportación: El último tipo de etapas es la etapa de exportación. Esta etapa es única y obligatoria. Además, siempre se encuentra al final del proceso de configuración. En esta etapa el usuario puede definir las cámaras y las calidades, medidas en frames por segundo (*FPS*), con las que desea realizar el proceso de exportación.

Todas las etapas añadidas deben tener asociado un nombre que es el que se muestra en la interfaz cuando los usuarios están configurando algún modelo de esta familia.

El último de los elementos que componen una familia es la lista de modelos que pertenecen a dicha familia. Esta lista contiene los nombres de los modelos y las rutas de los ficheros XML configurables que definen a un modelo.

A continuación se muestra un gráfico para ilustrar toda la estructura comentada anteriormente.

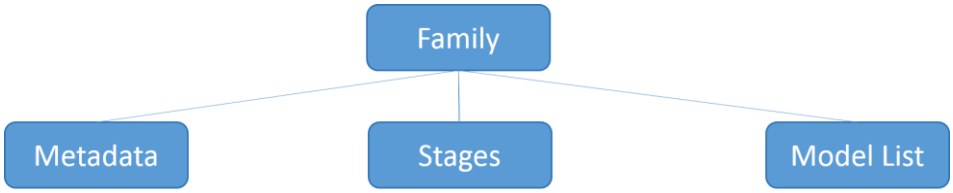

Figura 5.5: Modelo subyacente

5.1.2.2. Modelos

Una vez que se ha definido la estructura de la familia, el usuario puede definir más aspectos propios del modelo. Las características que se han definido en el apartado anterior son comunes para todos los modelos de la familia, mientras que las que se realizan a este nivel son propias y únicas de cada modelo. En este fichero, el usuario puede distribuir entre las distintas etapas definidas anteriormente, todas las texturas y submallas que quiera incluir en el proceso de configuración del modelo. Además puede definir las distintas complexiones predefinidas que se van a mostrar en la interfaz o asignar los identificadores creados en la familia a un hueso en concreto del modelo, para su escalación.

Para la configuración de este fichero existen tres elementos fundamentales: el primero de ellos es donde se encuentra la información relativa al modelo y se definen las complexiones y el escalado de huesos. El segundo, contiene una lista de las texturas intercambiables a la hora de configurar el modelo. Y por último, el tercer elemento incluye una lista de las submallas asociadas a dicho modelo. A continuación se explica con mayor detalle cada una de los partes básicas que componen este fichero.

El primer elemento es donde se encuentra toda la información relativa al aspecto físico del modelo. En un primer lugar tenemos una secuencia de las transformaciones que hay que aplicarle al modelo para situarlo correctamente y que aparezca como el usuario desea. Estas transformaciones pueden ser cualquiera de las transformaciones básicas del mundo 3D: rotación, traslación y escalación. Otra información que se almacena son las complexiones predefinidas que el usuario quiera ofrecer para su modelo. Estas complexiones llevan asignado un identificador único, además del identificador de la etapa en la que deben aparecer. Por último y también relacionado

con las complexiones, se pueden asignar los identificadores creados en la familia para la escalación de huesos concretos. A la hora de asignar estos controladores a cada hueso, el usuario puede decidir el rango de valores que tomará. De esta manera, el usuario tiene total libertad de configuración ya que puede crear tanto controladores como desee y cada uno de ellos con unos valores diferentes.

El segundo elemento contiene la información más importante para el correcto funcionamiento de la aplicación. En él, está representada toda la información relacionada con las texturas que dispone el modelo para su configuración. La información se representa como una lista de texturas, en las que cada elemento de dicha lista posee un identificador único. Además, se incluye también el identificador de la etapa para establecer la relación entre las texturas con la etapa de la interfaz a la que están asociadas. A continuación se detallan cada uno de los tipos diferentes de texturas que puede soportar *eCharacter*.

- Simple Texture o textura simple: En este tipo de textura no se permite el cambio de color y se aplica sobre el modelo tal y como es. De esta manera se respeta la textura creada por el diseñador gráfico y no se realiza modificación alguna sobre la textura original.

- MultiOption Texture o textura multiopción: Este tipo de texturas permite el cambio de color pero de una forma limitada. Para ello se puede modificar el color eligiendo entre un conjunto finito de posibilidades que indique el usuario. De esta manera y siguiendo la filosofía de la textura anterior, se respetan las texturas creadas por el diseñador gráfico y no se aplica ninguna transformación original.

- Base-Shadow Texture o textura base-sombra: En esta textura es donde mayor libertad de configuración puede tener el usuario. Este tipo de texturas está basada en dos capas. La primera capa o capa base es a la que se le podrá cambiar el color eligiendo cualquier combinación posible de los valores RGB. Para ello se le aplicará un algoritmo para tratar el color, que se explica en profundidad en el capítulo de *Implementación.* La segunda capa es una capa de detalles o sombras que se superpone a la primera una vez que se la haya realizado el procesamiento adecuado.

- Double Texture o textura doble: El funcionamiento de este tipo de texturas es similar a la textura anterior con la salvedad de que ahora se le puede aplicar el tratamiento de color a ambas capas. Este tipo de texturas también está compuesta por una capa base y una capa de detalles que va superpuesta sobre la primera.

Por último el tercer elemento básico que define un modelo tiene una estructura similar al anterior. Aquí se incluirá toda la información relativa a las submallas que son intercambiables a la hora de configurar el modelo. Al igual que en las texturas, la información se representa mediante una lista, donde cada submalla tiene un identificador único y el identificador de la etapa al que están asociadas. Como

información extra se incluye el hueso del esqueleto del modelo 3D al que deben ir asociado cada submalla, para que se visualice correctamente a la hora de configurar el modelo.

A continuación se muestra un gráfico para ilustrar toda la estructura comentada anteriormente.

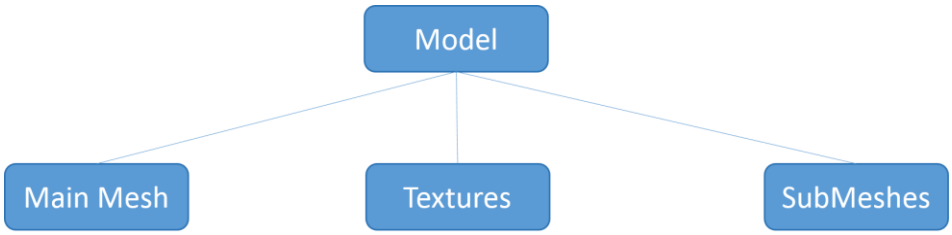

Figura 5.6: Modelo subyacente

5.1.3. Estructuras internas de datos

Para soportar toda la funcionalidad ofrecida por *eCharacter* se han diseñado una serie de estructuras internas. Gracias a estas estructuras la aplicación es capaz de manejar correctamente los datos y también es capaz de mantener la coherencia entre las elecciones del usuario y el resultado que se muestra.

Como ya se comentado en secciones anteriores el proceso de configuración del modelo está dividido en distintas etapas. Éstas son configurables mediante el fichero XML de la familia a la que pertenezca el modelo que se está configurando. Dentro de cada etapa, existen uno o varios paneles donde aparecen todas las posibles texturas o submallas que estén asociadas a dicho panel, además de la posibilidad de cambiar el color a la textura o submallas seleccionada.

Normalmente, el proceso de configuración de un modelo suele ser un proceso guiado y secuencial. Es decir, el usuario va recorriendo en orden las distintas etapas que componen todo el proceso, realizando en ellas los cambios que considere oportunos. Pero esto impone una limitación ya que, puede que alguna de las elecciones que el usuario haya realizado en etapas tempranas, no le parezcan adecuadas según va avanzando el proceso.

Gracias a estas estructuras internas la aplicación permite al usuario alternar entre las distintas etapas del proceso de configuración, sin que sea necesario seguir un orden. Con estas estructuras es posible saber en todo momento que texturas o submallas ha seleccionado el usuario en cada panel y como las ha personalizado. Esto permite poder retomar elecciones que el usuario había realizado pero que no son las que se muestran actualmente.

Todo esto se consigue mediante el uso de dos tablas hash, de la forma <clave, valor>, una para las todas las texturas y otra para todas las submallas asociadas al modelo que se está configurado. La clave de estas tablas es el identificador de cada subpanel y el valor es una lista de las texturas o submallas asociadas a dicho subpanel. Cada elemento de esta lista contiene dos marcas que permiten a la aplicación saber cuál fue el último color que el usuario eligió para esa textura y si esa textura está seleccionado actualmente y forma parte de las elecciones activas que ha realizado el usuario.

Como cada modelo tiene asociado distintas texturas y submallas, estas estructuras se crean en el momento que el usuario elige el modelo que desea configurar.

A continuación se muestra un gráfico que ilustra cómo se han definido estas estructuras, en el ejemplo es para la tabla de texturas.

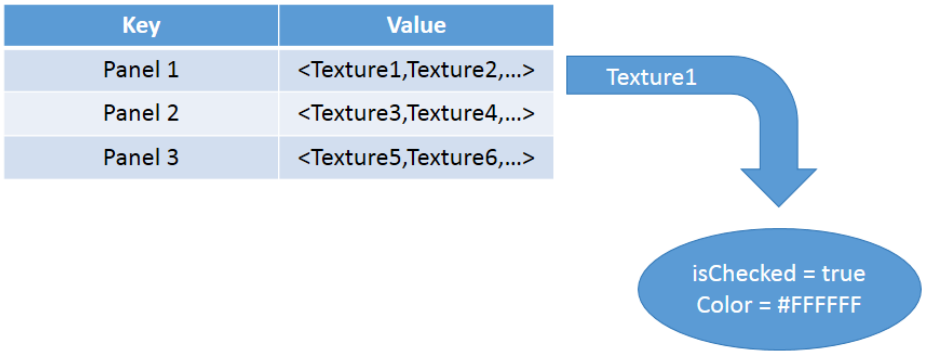

Figura 5.7: Estructura interna

5.1.4. Sistema de exportación

Como se ha contado anteriormente, una funcionalidad necesaria de *eCharacter* es la exportación. En este apartado se va a explicar cómo funciona el proceso de exportación.

Una vez configurado el modelo, el último paso es realizar la exportación. Como se ha contado anteriormente, el proceso de exportación dispone de dos modos de uso: modo básico y modo avanzado.

- En el modo básico, el usuario únicamente debe seleccionar cuáles son las animaciones que desea exportar, y el sistema de manera automática la realizará con una calidad por defecto y para la vista de cámara que haya en ese momento.

- En el modo avanzado, el usuario podrá seleccionar, además de las animaciones, las calidades y las cámaras para las que desea realizar la exportación.

El sistema realiza una exportación basada en *frames*. Esto quiere decir que el sistema va realizando una serie de capturas sobre cada animación según la calidad elegida. Cuanto mayor sea la calidad elegida mayor es el número de *frames* que se capturan. Una vez realizadas todas las capturas, éstas se pueden usar en cualquier otro sistema que las interpole, como por ejemplo eAdventure.

El proceso de exportación es el siguiente:

1) Se realiza el cálculo de cuántos frames se deben generar por animación teniendo en cuenta la calidad con la que se quiere realizar la exportación.

2) Una vez calculado esto, se realiza la exportación propiamente dicha. En el apartado de implementación se indica con más detalles técnicos, cómo se ha implementado este proceso.
3) Por último, se empaquetan todas las capturas en un fichero ZIP, de cara a que sea más manejable para el usuario.

5.2. Diseño de la interfaz gráfica

Hoy en día, cualquier aplicación necesita una interfaz gráfica agradable. Para que una aplicación se convierta en producto, uno de los requisitos indispensables es que disponga de una interfaz trabajada, cuidada, limpia.

Para realizar la interfaz gráfica de *eCharacter*, se ha realizado un estudio minucioso de cuáles son las tendencias en interfaces gráficas hoy en día, para lograr una buena interfaz gráfica.

En el estudio se analizaron diversas aplicaciones que se pueden encontrar en el mercado y que han marcado una tendencia en cuanto a interfaces se refiere. Analizando las aplicaciones de Windows y Apple, se llegó a la conclusión de que la tendencia hoy en día es crear una interfaz gráfica que sea minimalista, con líneas rectas, con uso de iconos representativos y colores que no sean muy llamativos.

Basándose en este estudio, se empezó a trabajar en varios diseños para la interfaz. Estos diseños se mostraron a diversos usuarios ajenos a la realización de este proyecto, y se tuvieron en consideración todos los comentarios aportados.

En un primer diseño, se mostraba toda la funcionalidad que debería tener la interfaz. La intención de este diseño no era tener un diseño bonito, sino tener un diseño de cómo organizar toda la funcionalidad descrita anteriormente. Este primer diseño pasó por varias versiones hasta conseguir la versión definitiva del boceto:

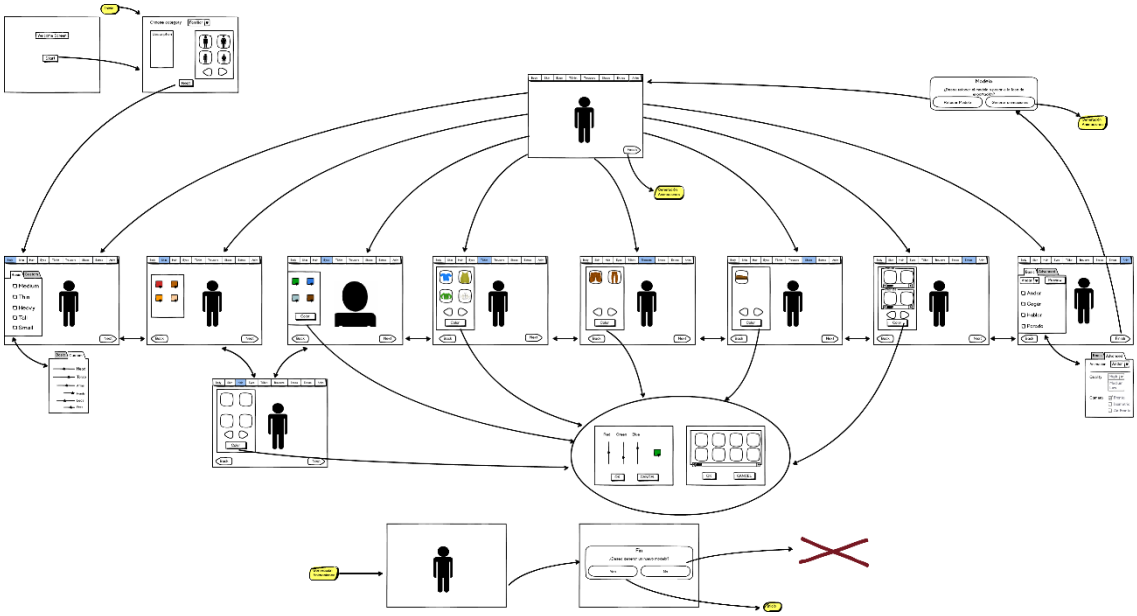

Figura 5.8: Diseño 1

Una vez realizado el boceto, y aprobado por todos los integrantes del grupo y director del proyecto, se llevó a cabo su implementación. La captura de una de las etapas es la siguiente:

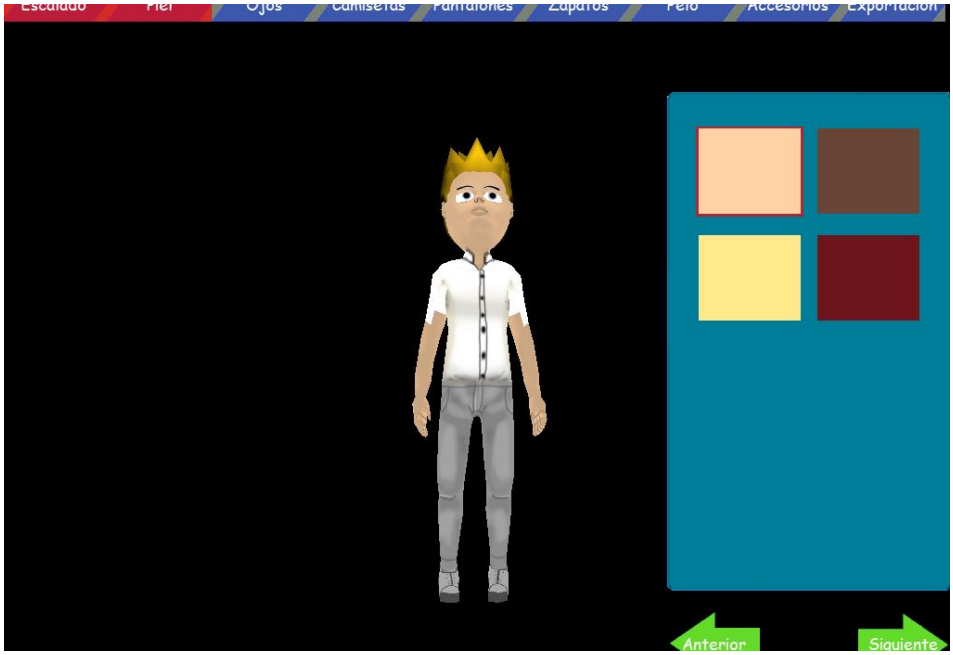

Figura 5.9: Diseño 2

Como se observa en la imagen anterior, esta interfaz no cumplía con los requisitos que hay actualmente sobre interfaces gráficas. Es por ello que se volvieron a diseñar los bocetos anteriores, teniendo en cuenta estos requisitos. En este caso, se tomaron varias decisiones sobre el *look and feel* que debería tener la aplicación. Trabajando sobre esto, se crearon los siguientes diseños:

- Diseño de la pantalla de selección de un modelo.

Figura 5.10: Diseño 3 – Selección de un modelo

- Diseño de una pantalla de configuración de un modelo.

Figura 5.11: Diseño 3 - Configuración del modelo

Estos diseños ya empezaron a parecerse a la imagen definitiva que queríamos que tuviese nuestra interfaz gráfica. Aun así, se siguió trabajando en estos diseños, teniendo en cuenta las opiniones de los integrantes del grupo, de los directores y de varios usuarios finales.

Con toda esta información y después de varias versiones más, se llegó al diseño final de la interfaz gráfica.

- Diseño final de la pantalla de selección de un modelo.

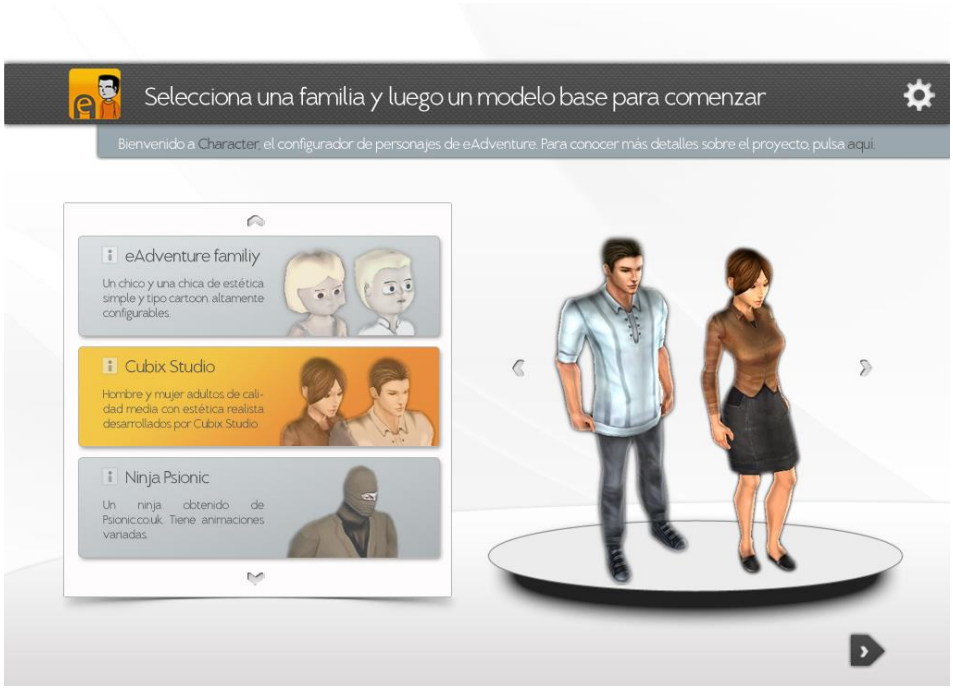

Figura 5.12: Diseño 4 - Selección de un modelo

- Diseño final de una pantalla de configuración de un modelo.

Figura 5.13: Diseño 4 - Configuración del modelo

Una vez alcanzada la calidad deseada de estos diseños finales, se procedió a la implementación de los mismos dando el siguiente resultado que se muestra en el apartado 6.2.1 *Implementación de la interfaz.*

5.3. Casos de uso

5.3.1. Elección del modelo base

Objetivo: Gestiona la selección del modelo base para la configuración del mismo por parte del usuario.

Precondiciones: El usuario debe tener la aplicación en ejecución.

Postcondiciones si éxito: El usuario habrá elegido un modelo base para su personaje.

Postcondiciones si fallo: El usuario deberá volver a elegir el modelo inicial, para que el sistema lo seleccione.

Actores: Usuarios y aplicación de creación de personajes.

Secuencia normal:

P1 --> El usuario elige una familia entre todas las disponibles en el sistema.

P2 --> El usuario elige un modelo base perteneciente a la familia elegida en el paso anterior.

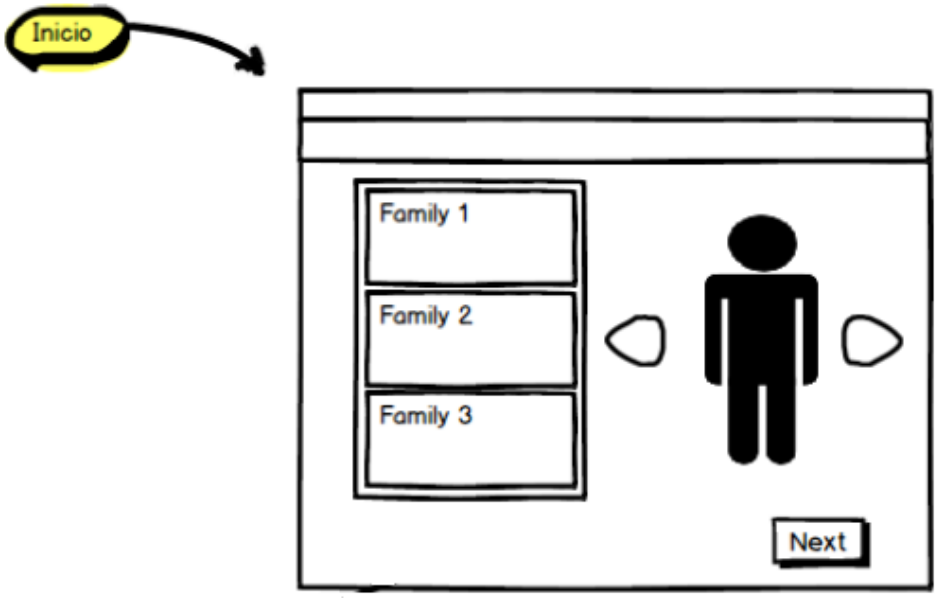

Figura 5.14: Caso de uso: elección del modelo base

5.3.2. Modificación del modelo base

Objetivo: El usuario modificará el modelo base hasta llegar al modelo personalizado deseado para poder realizar la exportación.

Precondiciones: El usuario debe tener la aplicación en ejecución y haber seleccionado un modelo base.

Postcondiciones si éxito: El usuario habrá terminado de editar su personaje poder realizar la exportación.

Postcondiciones si fallo: El usuario deberá volver a modificar el modelo base o elegir uno nuevo.

Actores: Usuarios y aplicación de creación de personajes.

Secuencia normal:

P1 --> El usuario modifica la camiseta del modelo.

P2 --> El usuario modifica los pantalones del modelo.

P3 --> El usuario modifica el tipo de peinado del modelo.

P4 --> El usuario cambia el color de la camiseta del modelo.

P5 --> El usuario modifica el tamaño del modelo.

P6 --> El usuario guarda el personaje para exportarlo.

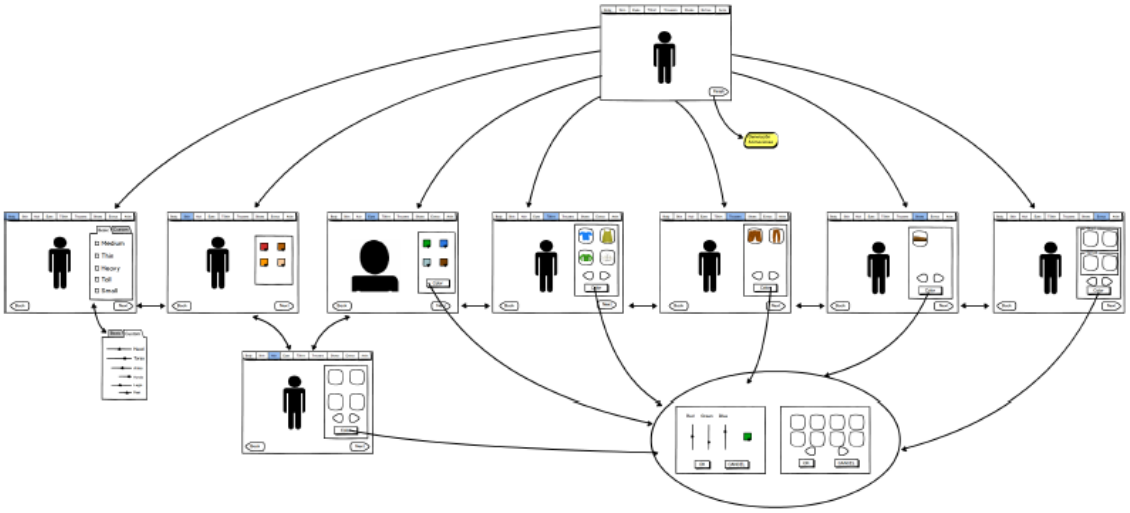

Figura 5.15: Caso de uso: modificación del modelo base

5.3.3. Elección de animaciones

Objetivo: El usuario puede probar las animaciones asociadas al modelos que se está configurando y elegir, de éstas, las que exportará.

Precondiciones: El usuario debe haber completado la fase de modificación del modelo base con éxito.

Postcondiciones si éxito: El usuario habrá elegido las animaciones que desea exportar.

Postcondiciones si fallo: El usuario deberá volver a elegir las animaciones entre las asociadas al modelo que se está configurando.

Actores: Usuarios y aplicación de creación de personajes.

Secuencia normal:

P1 --> El usuario prueba la animación 1.

P2 --> El usuario prueba la animación 2.

P3 --> El usuario selecciona las animaciones que desea incluir en la exportación.

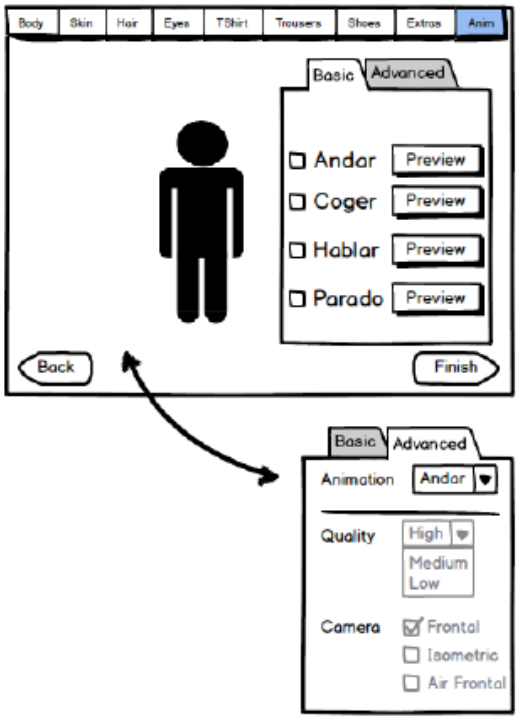

Figura 5.16: Caso de uso: elección de animaciones

5.3.4. Exportación de animaciones

Objetivo: La aplicación ejecuta la exportación.

Precondiciones: El usuario debe haber finalizado la etapa de elección de animaciones.

Postcondiciones: El programa obtendrá los frames necesarios de las animaciones para exportar el modelo.

Actores: Aplicación de creación de personajes.

Secuencia normal:

P1 --> La aplicación muestra al usuario una pantalla en la que se realizará el proceso de exportación.

P2 --> La aplicación captura los frames de las animaciones seleccionadas por el usuario.

P3 --> La aplicación termina el modo de exportación de animaciones y oculta la pantalla de carga al usuario.

P4 --> La aplicación construye un archivo comprimido con todas las animaciones del modelo personalizado con las elecciones del usuario.

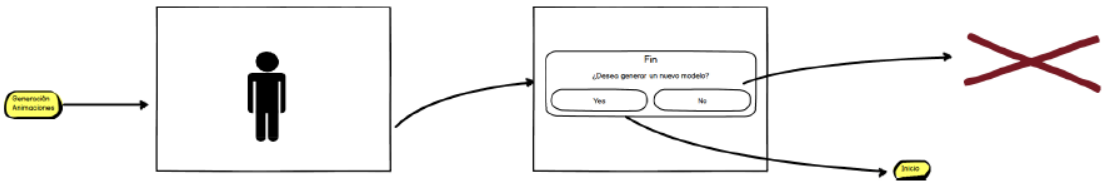

Figura 5.17: Caso de uso: exportación de animaciones

6. Implementación

En este capítulo se analiza los detalles más relevantes sobre la implementación del proyecto. Está formado por dos apartados. En un primer apartado se realiza un repaso sobre las principales tecnologías utilizadas, así como un pequeño análisis de las mismas. El segundo apartado consta de aspectos relacionados con la implementación misma que debido a su complejidad merecen un análisis más profundo.

6.1. Tecnologías usadas

Las principales tecnologías y herramientas que se han usado para desarrollar la aplicación son: jMonkeyEngine, Blender, Nifty GUI, eAdventure, JAXB e izpack.

6.1.1. jMonkeyEngine 3.0

jMonkeyEngine (abreviado jME) es un motor de desarrollo de videojuegos basado en OpenGL e implementado en Java, distribuido con licencia BSD. Posee además su propio entorno de desarrollo integrado (IDE), que recibe el nombre de jME SDK.

La principal característica de jME es el desarrollo basado en árboles de nodos. jME está basado en una arquitectura de escenas de tipo árbol. Esto permite la organización de los datos del juego en un grafo de nodos, que facilita el tratamiento de la información.

Algunos conceptos clave para comprender el funcionamiento de jME son los siguientes:

- Scene Graph (escena gráfica): la escena gráfica es el espacio virtual donde se desarrolla la acción. Todos los objetos que pertenezcan a la escena gráfica se llaman Spatial.

- Spatial: Un spatial es un elemento de la escena gráfica que puede ser transformado mediante la aplicación de cualquiera de las tres transformaciones básicas (escalado, rotación y traslación). Existen dos tipo de spatial:

    - Geometry (geometrías): una geometría representa un objeto visible de la escena en tres dimensiones. Está formado por dos componente principales:

        - Mesh (malla): define la forma que tendrá la geometría.
        - Material: define la apariencia que tendrá la geometría, como puede ser el color, la textura, la opacidad, etc.

    - Node (nodos): son agrupaciones de spatials (tanto de otros nodos como de geometrías). Este conjunto de spatial serán tratados en conjunto, es

decir, cuando se realice una transformación sobre un nodo, esta afectará a todos los hijos de dicho nodo.

La escena gráfica se representa a partir de un rootNode (nodo raíz) a partir del cual se añadirán los nuevos elementos que se incorporen a la escena. Un ejemplo de esta distribución sacado de la propia página de jME (http://jmonkeyengine.org) es el siguiente:

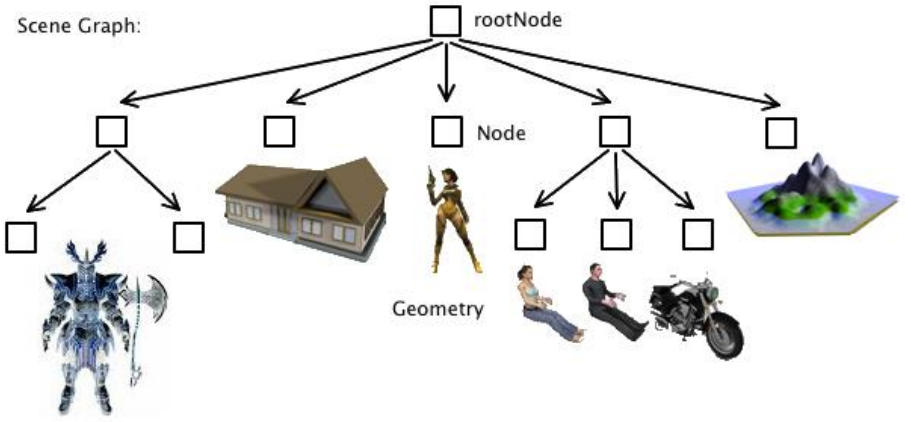

Figura 6.1: Representación del árbol de nodos en una escena gráfica

Esta disposición de la escena gráfica en forma de árbol es muy importante ya que facilita la tarea a la hora de descartar ramas que no queramos procesar.

Una técnica que se simplifica mucho gracias a esta distribución es el culling, que permite indicar las geometrías que no van a ser renderizadas por la tarjeta gráfica y por tanto no aparecerán en la pantalla.

jME ofrece la posibilidad de gestionar el culling de las geometrías presentes en la escena, pero también lo puede realizar automáticamente mediante el uso de cámaras. Todas aquellas geometrías que queden fuera del rango de visión que se haya definido para la cámara, se les aplicará la técnica de culling y por tanto no serán renderizadas.

jME posee una series de formatos de ficheros para el soporte de todos los recursos gráficos necesarios. Estos formatos son los siguientes:

Extensión	Descripción
.j3o	Fichero binario de modelos 3D y escenas.
.j3m	Fichero que permite almacenar la configuración de un material.
.j3md	Fichero que permite la definición propia de materiales

Además de los tipos propios, jME también soporta la carga de formatos externos, bien de forma nativa o mediante el uso de plugins externos. La lista de los formatos de modelo 3D soportados es la siguiente:

Extensión	Descripción
.mesh.xml, .meshxml	Ogre Mesh XML
.scene	Ogre DotScene
.OBJ , .MTL	WaveFront
.blend	Version Blender 2.49
COLLADA	Importado a través de Blender incluido con el SDK
3DS	Importado a través de Blender incluido con el SDK

6.1.2. Blender

Blender es una herramienta de modelado 3D gratuita y distribuida bajo licencia GPL (General Public License). Es uno de los programas de diseño 3D más utilizados y por tanto tiene una gran comunidad de desarrollo detrás. Debido a esto, Blender es capaz de soportar la carga de multitud de formatos gracias al desarrollo de plugins por parte de los usuarios.

En este proyecto se ha utilizado el plugin Ogre XML Exporter que nos permite exportar un modelo animado 3D al formato Ogre XML soportado por jME, como se puede observar en la tabla anterior.

6.1.3. JAXB

Java Architecture for XML Binding (JAXB) proporciona una manera rápida y simple para enlazar XML Schemas y su representación en Java, por lo que es muy útil para los desarrolladores de Java. JAXB proporciona la capacidad de serializar las referencias de objetos Java a XML y viceversa. En otras palabras, JAXB permite almacenar y recuperar datos en memoria en cualquier formato XML, sin la necesidad de implementar un conjunto específico de rutinas de carga y guardado de XML para la estructura de clases del programa.

Se compone de tres procesos básicos:

- Compiling: Convierte un XML Schema en clases Java.
- Marshaling: Genera un documento XML a partir de objetos Java.
- Unmarshaling: Genera objetos Java a partir de un documento XML.

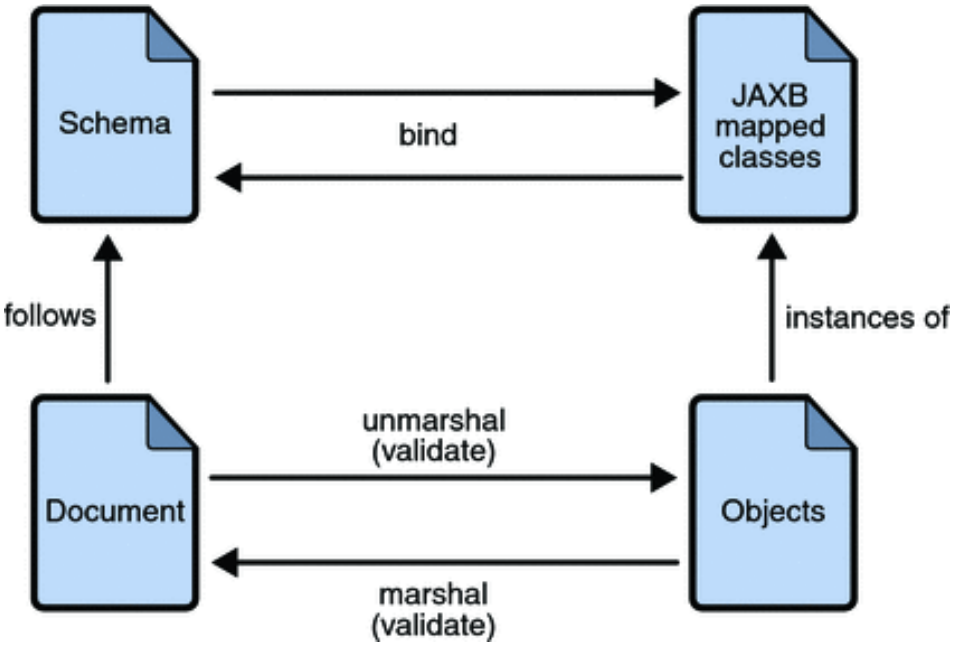

Figura 6.2: Esquema general de funcionamiento de JAXB

JAXB es muy útil cuando la especificación es compleja y cambiante. Por ejemplo, en el caso de tener que cambiar manualmente las definiciones de XML Schema para mantenerlas sincronizadas con las definiciones de Java, se puede convertir en un proceso largo y propenso a errores.

La conversión de tipos que realiza JAXB viene dada en la siguiente tabla:

Tipo de datos en XML Schema	Tipo de datos en Java
xsd:string	String
xsd:positiveInteger	BigInteger
xsd:int	int
xsd:long	long
xsd:short	short
xsd:decimal	BigDecimal
xsd:float	float
xsd:double	double

xsd:boolean	boolean
xsd:byte	byte
xsd:QName	QName
xsd:dateTime	XMLGregorianCalendar
xsd:base64Binary	byte[]
xsd:xsd:hexBinary	byte[]
xsd:unsignedInt	long
xsd:unsignedShort	int
xsd:unsignedByte	short
xsd:unsignedLong	BigDecimal
xsd:time	XMLGregorianCalendar
xsd:date	XMLGregorianCalendar
xsd:g	XMLGregorianCalendar
xsd:anySimpleType (para xsd:element de este tipo)	Object
xsd:anySimpleType (para xsd:attribute de este tipo)	String
xsd:duration	Duration
xsd:NOTATION	QName

6.1.4. IzPack

IzPack es un generador de instaladores de código libre basado en Java. Los instaladores los genera en formato .jar. Esto permite que el instalador se pueda ejecutar en cualquier plataforma que tenga instalado Java.

Aunque en un primer momento, su propósito era crear instaladores multiplataforma (Windows, Linux, Solaris y Mac OS X), con el paso del tiempo se ha ido refinando para permitir realizar acciones determinadas para Windows, como por ejemplo crear accesos directos.

Los instaladores IzPack pueden ser muy personalizados. Cada paso de una instalación se materializa por un panel, con la información del paso que se está dando y los botones de "anterior" y "siguiente".

IzPack usa ficheros XML para describir el proceso de instalación. En estos ficheros se tienen que describir los iconos que se van a utilizar, qué paneles y qué información contienen, directorio de instalación y un sinfín de propiedades que permiten customizar el proceso según las necesidades.

6.1.5. eAdventure

eAdventure es un proyecto de investigación que aspira a facilitar la integración de juegos educativos y simulaciones basadas en juegos en procesos educativos en general y Entornos Virtuales de Aprendizaje (VLE) en particular (Moreno-Ger, Burgos, Sierra, & Fernández-Manjón, 2008; Torrente, Moreno-Ger, Fernández-Manjón, & Sierra, 2008). Está siendo desarrollado por el grupo e-UCM en la Universidad Complutense de Madrid, con tres objetivos principales:

- Reducción de los costes de desarrollo para juegos educativos.
- Incorporación de características educativas específicas en herramientas de desarrollo de juegos.
- Integración de los juegos resultantes con material educativo en el contexto de los Entornos Virtuales de Aprendizaje.

El sistema de animaciones que usa eAdventure se basa en dos conceptos claves: apariencia y animaciones.

El conjunto de animaciones básicas que soporta un personaje son aquellas que representan las acciones básicas que el personaje puede realizar durante el transcurso del juego: permanecer quieto, hablar, caminar e interactuar con objetos. Estas acciones son las más habituales pero no por ello son las únicas que puede realizar un personaje.

Las animaciones de un personaje se agrupan en bloques llamados apariencias.

Además dentro de cada acción que se pueda realizar, hay que tener en cuenta la orientación del personaje en la animación, ya que no es lo mismo hablar mirando hacia la izquierda que hacia la derecha. Es por ello que para cada tipo de animación hay que tenerla definida para cada una de las cuatro orientaciones del personaje (arriba, abajo, izquierda, derecha).

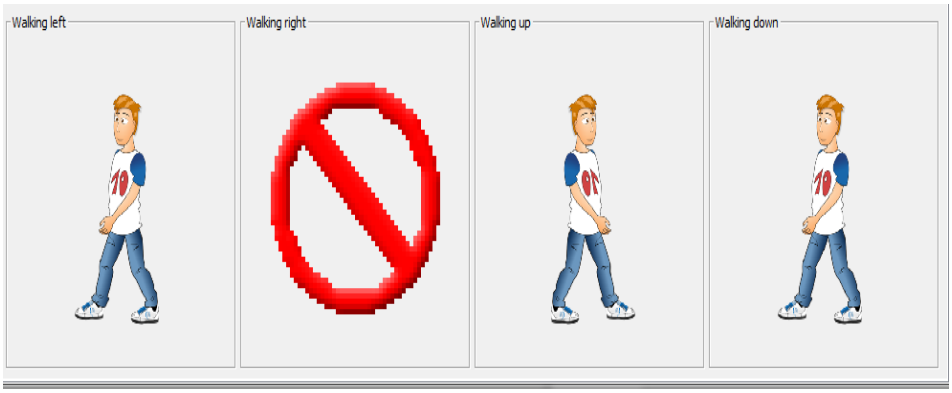

Figura 6.3: Sistema de animaciones en eAdventure

Cabe destacar que puede haber algunas animaciones que no puedan darse en ciertas apariencias, aunque el editor no nos imponga ninguna restricción al respecto.

La señal de prohibido indica que no existe ninguna animación asociada a la apariencia que se esté editando.

- **Formato de las animaciones**

Las animaciones en eAdventure se realizan mediante una secuencia de imágenes en las cuales se va apreciando el movimiento del personaje, por lo que cada animación tendrá asociada una cantidad de imágenes (archivos .PNG, .JPG, .BMP,...) indeterminados.

El sistema de animaciones de eAdventure usa la extensión de fichero .eaa. Este formato de fichero permite especificar una animación como un conjunto de frames o imágenes, así como la duración de cada frame, el intervalo y efectos entre ellos o algún sonido asociado a un frame en concreto. Dentro del editor de aventuras de eAdventure podremos crear animaciones.

6.1.6. Nifty GUI

Es una biblioteca Java que soporta la construcción de interfaces de usuario interactivas para juegos o aplicaciones similares. Utiliza lwjgl para OpenGL.

Nifty fue diseñado pensando en convertirlo en una herramienta y no en un toolkit para la creación de GUIs. No se limita a un conjunto de controles estándar pero pueden construirse interfaces complejas.

El diseño de las interfaces en Nifty puede desarrollarse tanto en XML como en Java. Normalmente se desarrolla la parte estática en lenguaje XML, y se utilizan comandos en Java si es necesario hacer algún cambio en tiempo de ejecución.

Nifty se basa únicamente en tres elementos: imagen, panel y texto. El resto de componentes de la librería están compuestos por una combinación de estos tres elementos básicos. Además se pueden crear nuevos controles combinando los elementos básicos.

La interfaz desarrollada en Nifty está compuesta por una o más pantallas en las que se distribuye la interfaz. Estas pantallas son llamadas screens y sólo una de ellas

puede ser visible en un determinado momento. Cada una de estas screens tendrá el apoyo en lenguaje Java con las que serán controladas todas las acciones que pueda realizar el usuario. Para poder controlar dichas acciones, la clase java debe implementar la interfaz Java Controller class. No hay limitaciones en cuanto al nombre de las screens salvo que la inicial debe llamarse start.

Cada screen contiene una o más capas en las que se insertan elementos llamados layers. Estos layers imponen el alineamiento de su contenido y pueden superponerse unos a otros, pero jamás pueden anidarse. Existe un layer especial que bloquea la interacción con el resto; éste es el que se utiliza para la implementación de los popups.

Todos los layers contienen paneles, y al contrario que los layers, pueden anidarse, pero no superponerse. En su interior pueden contener, con una alineación impuesta por el panel, más paneles, imágenes o texto, o combinaciones de estos tres elementos como son los controles predefinidos, como botones o sliders.

A continuación se incluye un pequeño glosario de términos, donde se definen aquellos que son más referenciados durante la lectura del apartado *Implementación de la interfaz*. El glosario de términos es el siguiente:

- Screen: Es la entidad en la que se divide una interfaz desarrollada en Nifty. En una interfaz se pueden implementar tantas screens como se quiera, pero a la vez se tendrá que implementar un sistema que sea capaz de navegar entre las diferentes screens que forman la aplicación.

- Layer: Es la entidad en la que se dividen las screens. Dentro de estos layers se disponen los diferentes elementos de Nifty, es decir, paneles, imágenes y textos.

- Panel: Es una entidad básica de Nifty al que se atribuye una porción del layer en el que está implementado.

6.2. Detalles de implementación

En esta subsección se analizan aspectos de la implementación que debido a su complejidad o importancia, requieren un análisis y comentario mayor y más profundo. Los puntos principales que se tratan en esta subsección son los siguientes: implementación de la interfaz, algoritmo de coloreado de texturas, implementación del sistema de exportación, algoritmo de recorte de imágenes utilizado en el proceso de exportación y el sistema implementado para la búsqueda de recursos dentro del sistema en el cuál esté ejecutándose la aplicación.

6.2.1. Implementación de la interfaz

En este apartado se detalla cómo se ha realizado el proceso de la implementación que se ha diseñado (para más información consultar la sección 5.2 *Diseño de la interfaz gráfica* en el capítulo *Diseño*). Para ello se han implementado

distintos tipos de pantallas que cumplen un propósito diferente cada una de ellas. A continuación se analizan más detalladamente cada una de ellas.

- **Pantalla de selección del modelo base:**

En esta pantalla inicial se proporciona al usuario la posibilidad de seleccionar un modelo base entre la variedad de modelos disponibles, agrupados en diferentes familias. Está implementada por dos capas, una capa que contiene únicamente la imagen de fondo y otra superponiéndola, llamada foreground, que contiene todos los elementos de interacción con el usuario.

La pantalla foreground la forman tres paneles y una imagen. Inicialmente se dispone un panel con el objetivo de separar la cabecera de la parte superior de la pantalla. Seguidamente nos encontramos la propia imagen de la cabecera. El siguiente elemento se trata de un panel que separa la cabecera del resto del cuerpo de la pantalla, y para finalizar se dispone el panel que contiene el resto de elementos que forman esta pantalla.

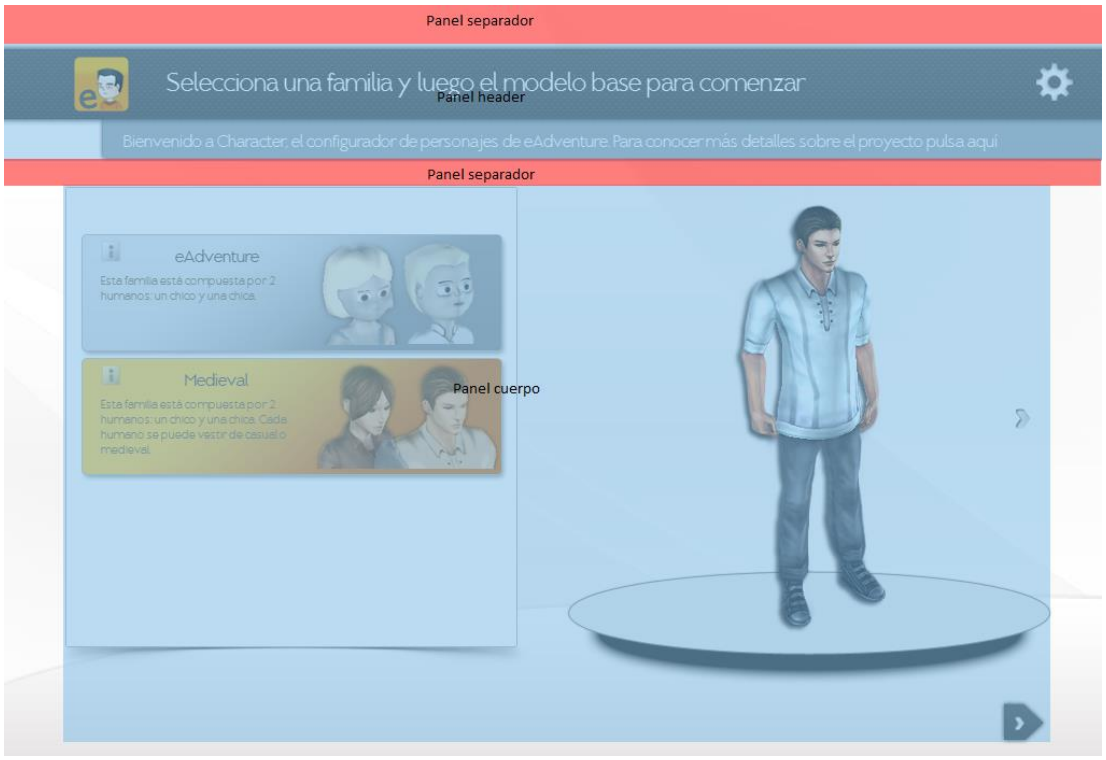

Figura 6.4: Vista de paneles de la pantalla de selección de modelo

La imagen de la cabecera se divide verticalmente en dos mitades. En la mitad superior se introduce un texto como cabecera de la pantalla, y en la parte inferior un texto informativo como subcabecera. El cuerpo de la pantalla está formado por un panel que integra los selectores de familia y modelos y otro panel que contiene el botón para el avance de página.

Figura 6.5: Vista de paneles de la pantalla de selección de modelo

El selector de familias se implementa como una imagen de fondo dividida verticalmente en tres paneles. El primer y último panel están formados por los botones que hacen posible navegar por las diferentes familias que están incluidas en la aplicación. El panel central está formado por tres imágenes dispuestas verticalmente. Cada una de estas imágenes recoge la información de una familia, disponiendo en su interior de una descripción breve, un icono de la familia, y un botón para más información. Si no existen suficientes familias, el mecanismo implementado se encarga de ocultar las imágenes sobrantes, así como los botones que nos permiten la navegación entre las familias.

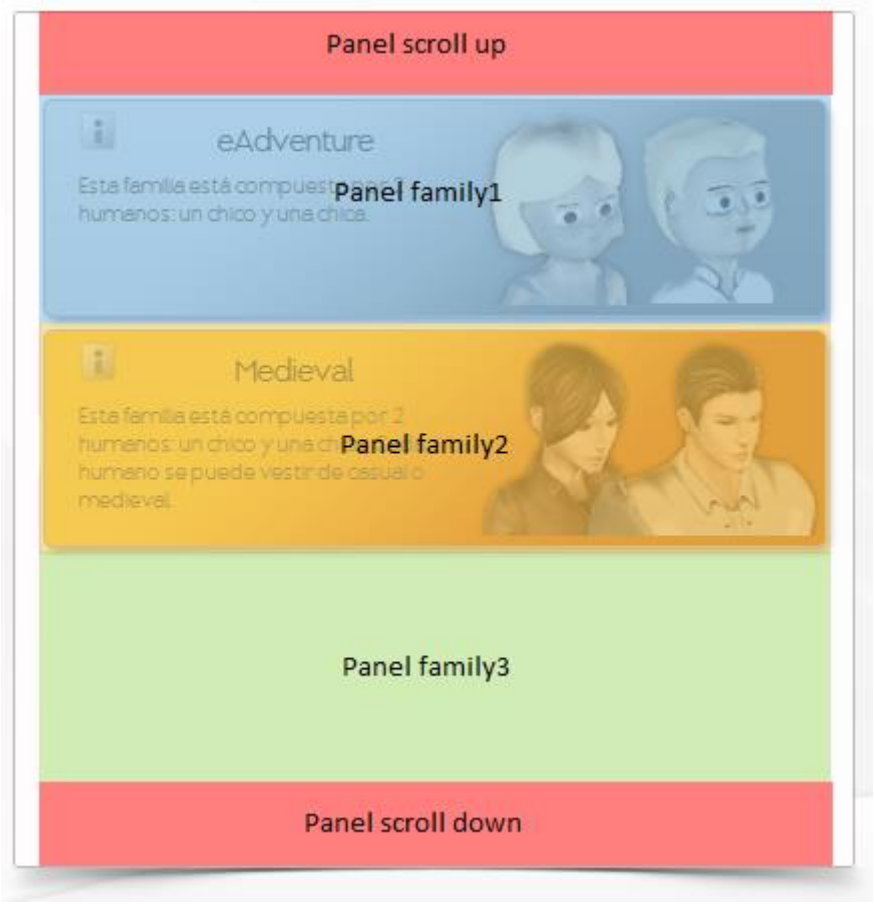

Figura 6.6: Vista de paneles de la pantalla de selección de familia

El selector de modelos está compuesto por una imagen de fondo con forma de peana y tres paneles dispuestos horizontalmente en su interior. Como en el selector de familias, el primer y último panel lo forman los botones de navegación, esta vez entre los modelos de la familia seleccionada. El panel central está formado por la imagen asociada al modelo que está seleccionado. En el selector de modelos también está implementado un mecanismo que esconde los botones de navegación si no dispone de más modelos en la familia seleccionada.

- **Pantallas de personalización del modelo base:**

Está formado por un conjunto de pantallas que se implementan de forma similar. Estas pantallas son la de escalado, la de exportación, y las de selección de texturas o submallas, tanto la que contiene una única subetapa, como la que soporta varias subetapas.

Estas pantallas están formadas por cuatro paneles. El primero se utiliza como separación del menú y la parte superior de la pantalla. El siguiente panel lo compone el menú de etapas. Después se encuentra otro panel de separación, esta vez entre el menú de etapas y el cuerpo de la pantalla. Para finalizar, se dispone el panel que integra el cuerpo de la pantalla.

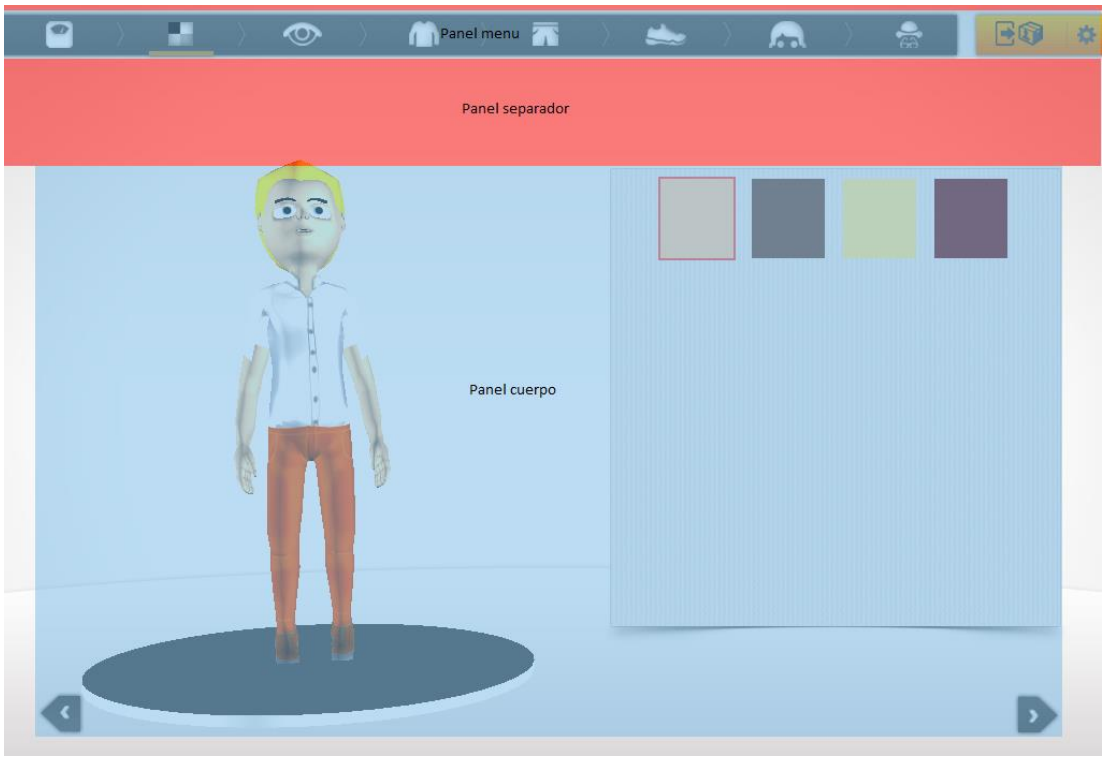

Figura 6.7: Vista de paneles de la pantalla de etapa simple de configuración del modelo

El menú de etapas se compone de dos partes. La primera parte se compone de todas las etapas dependientes del modelo que se ha seleccionado. El número de etapas es variable, por tanto la aplicación se encarga de dividir este espacio en partes iguales entre todas las etapas. Cada botón del menú está formado por una imagen representativa de la etapa y una barra inferior que se activa cuando la etapa está seleccionada. También se ha implementado un tooltip que muestra el nombre que se le ha dado a la subetapa cuando el puntero del ratón se sitúa sobre el botón del menú. La segunda parte se compone por la única etapa fija que deben poseer todos los modelos, la etapa de exportación, y un menú de configuración.

El panel que contiene el cuerpo de las pantallas, se divide en un panel que contiene las opciones de modificación del modelo que proporciona la etapa, y un segundo panel que contiene los botones de navegación entre etapas.

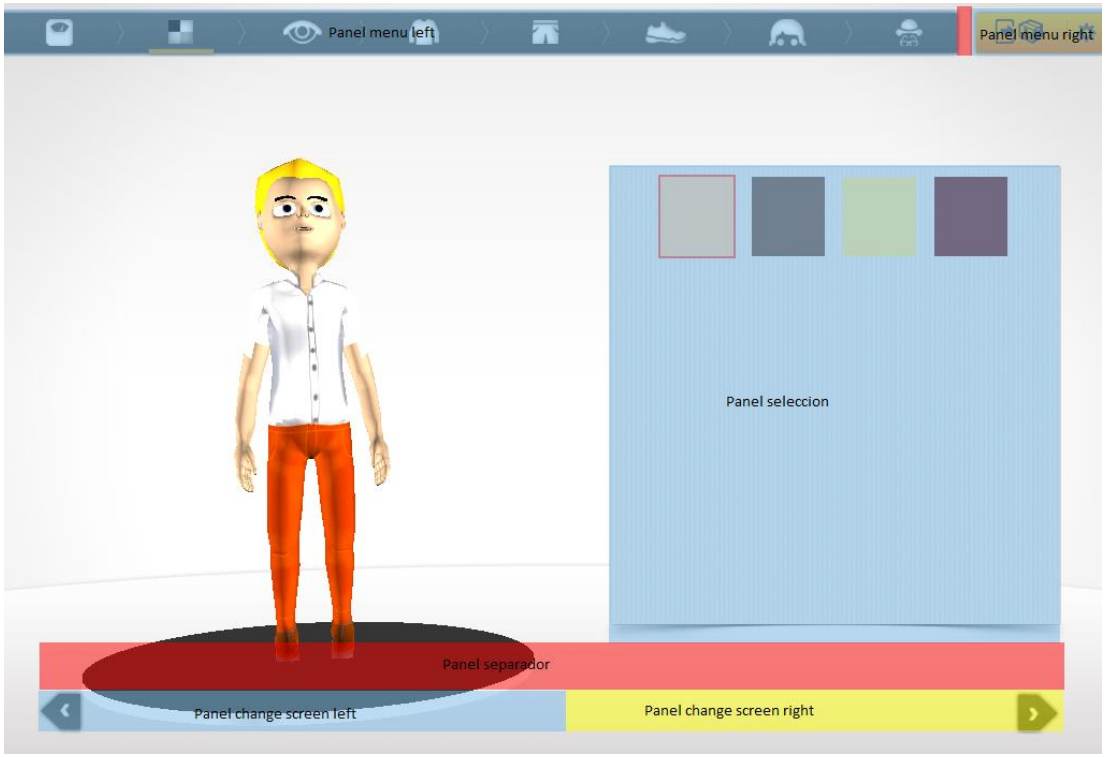

Figura 6.8: Vista de paneles de la pantalla de etapa simple de configuración del modelo

La diferencia entre este conjunto de pantallas radica en la implementación del panel de opciones de modificación del modelo, que se detalla a continuación.

- **Pantalla con una subetapa**

Este panel está compuesto por cuatro paneles dispuestos verticalmente. El primero y el tercero son paneles de separación. El segundo contiene las diferentes texturas o submallas que posee el modelo en la etapa seleccionada, y el cuarto contiene el botón de modificación de color de la textura o submalla seleccionada.

El panel de selección de texturas o submallas se divide horizontalmente en tres paneles. El primer y último panel contienen los botones de navegación entre las diferentes texturas o submallas asociadas a la etapa. En el panel central se disponen cuatro paneles dispuestos verticalmente con paneles separadores entre ellos. Dentro de cada uno de estos paneles se encuentran cuatro imágenes también con paneles separadores entre ellas. Cada una de estas imágenes representa una textura o submalla de la etapa. Estas dieciséis imágenes varían dinámicamente dependiendo de la página de navegación de las texturas o submallas en la que nos encontremos, ocultándose en el caso de no disponer de ninguna textura o submalla asociada.

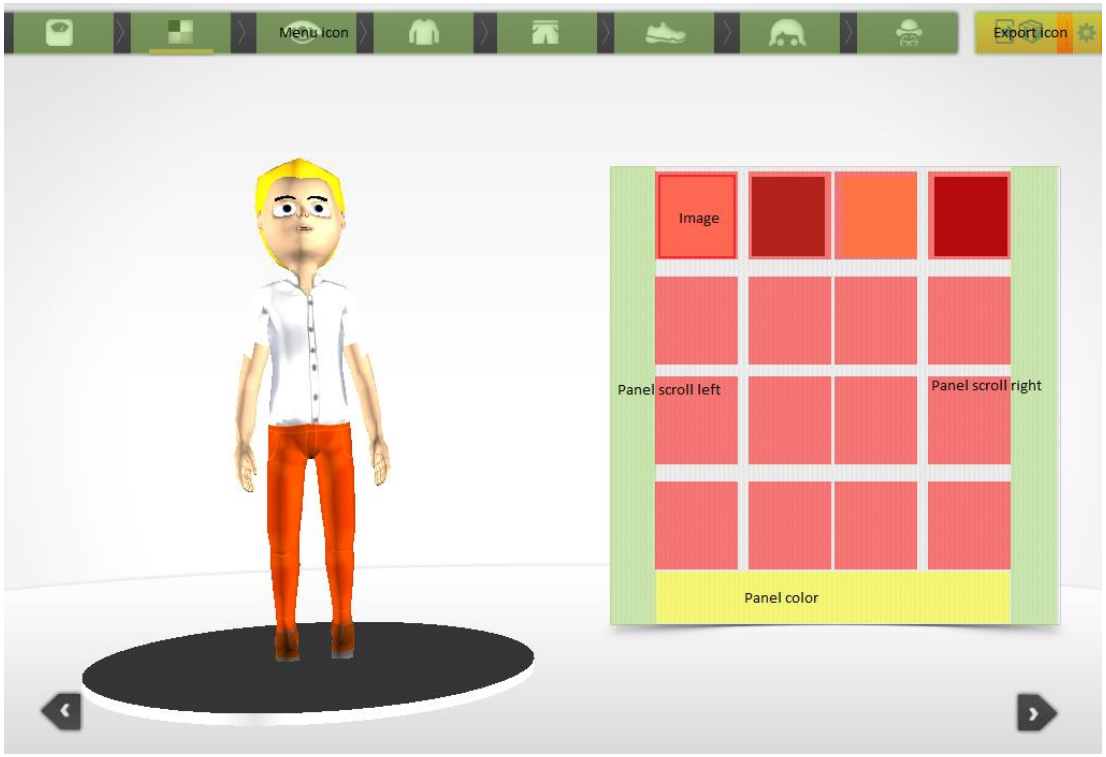

Figura 6.9: Vista de paneles de la pantalla de etapa simple de configuración del modelo

- **Pantalla con varias subetapas**

Este panel se encuentra dividido verticalmente en tres paneles. En el primer y tercer panel se sitúan los botones de navegación entre subetapas. El panel central se divide en dos paneles. Cada uno de estos paneles representa una subetapa.

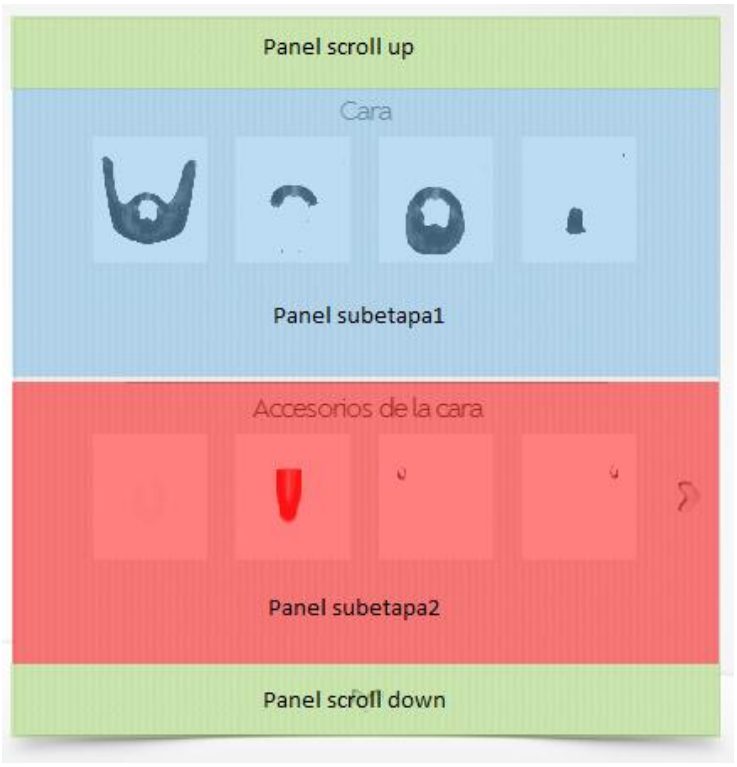

Figura 6.10: Vista de paneles de la pantalla de etapa múltiple de configuración del modelo

Las subetapas poseen la misma estructura que los paneles de selección de las pantallas con una subetapa explicado en la sección anterior. La única diferencia reside en la cantidad de imágenes de texturas o submallas que contienen. Estos paneles constan de un panel con cuatro imágenes dispuestas horizontalmente.

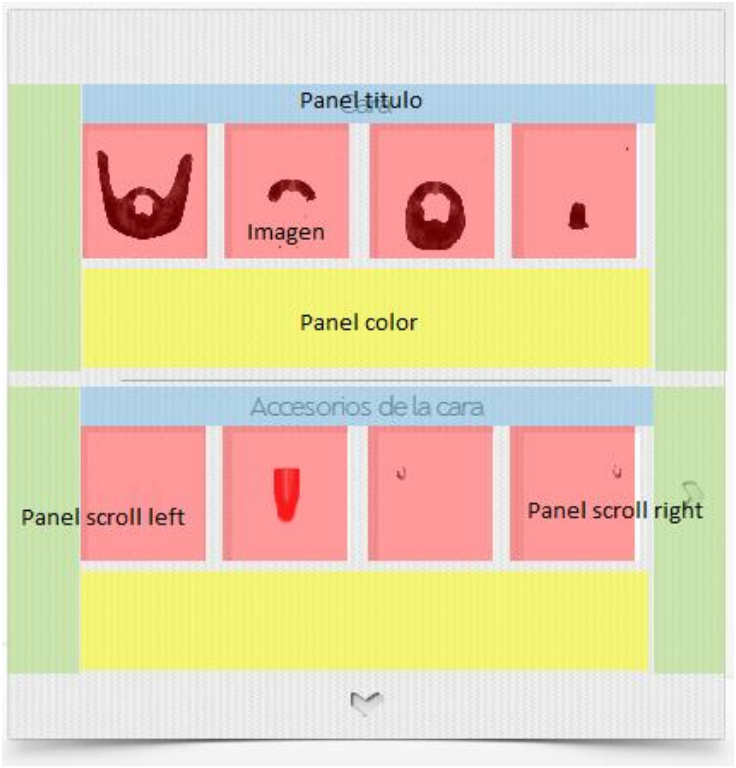

Figura 6.11: Vista de paneles de la pantalla de etapa múltiple de configuración del modelo

- **Pantalla de escalado**

Este panel se encuentra dividido verticalmente en tres paneles. En el primer y tercer panel se sitúan los botones de navegación entre complexiones o sliders de modificación de huesos. El panel central está constituido por diez paneles. Estos diez paneles se asocian a complexiones básicas o a complexiones avanzadas y se muestran u ocultan dependiendo de la pestaña en la que se encuentre el usuario. Los paneles impares se asocian a complexiones básicas y los pares a complexiones avanzadas. Los paneles asociados a complexiones básicas contienen únicamente un texto con el nombre de la complexión. Los paneles asociados a complexiones avanzadas se dividen horizontalmente en dos paneles. El primer panel contiene un texto con el nombre del hueso al que se le asocia el slider, el cual se encuentra en el segundo panel.

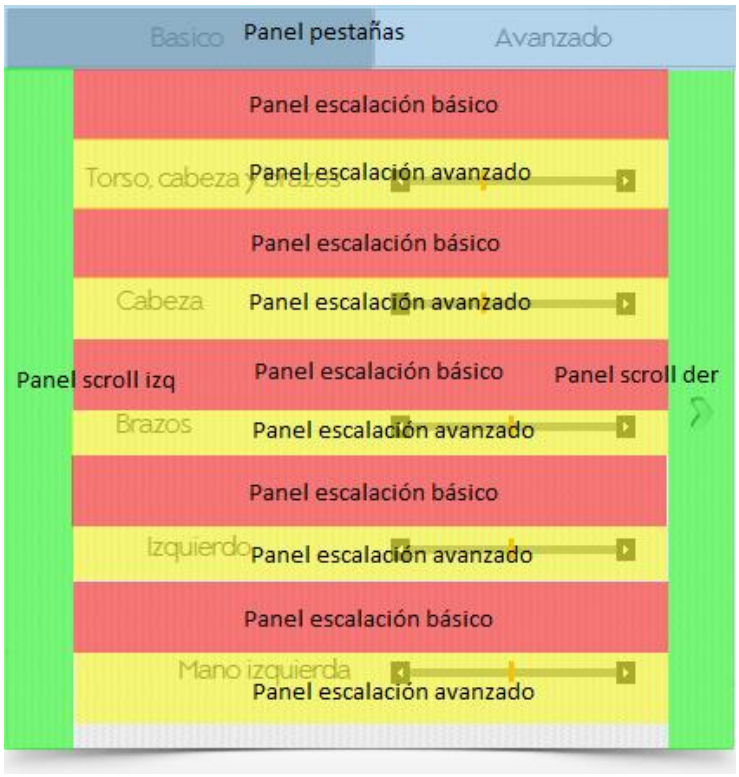

Figura 6.12: Vista de paneles de la pantalla de etapa de escalado del modelo

- **Pantalla de exportación**

Este panel se divide verticalmente en tres paneles con la misma estructura. Cada uno de estos tres paneles se divide verticalmente en cuatro paneles. Los tres primeros paneles contienen en su interior un texto, un checkbox y un botón. El texto puede ser el nombre de una animación del modelo, un tipo de cámara para la exportación o una calidad de exportación. El botón sirve para una pre-visualización de la animación o cámara que se está seleccionando. Las calidades no llevan botón asociado. El último panel contiene cuatro botones. Dos de éstos sirven para la navegación entre las diferentes animaciones, cámaras o calidades. Los dos restantes sirven para marcar todos los checkbox y para desmarcarlos.

Dependiendo de si el usuario ha seleccionado la pestaña básica o avanzada serán visibles o no algunas características de exportación.

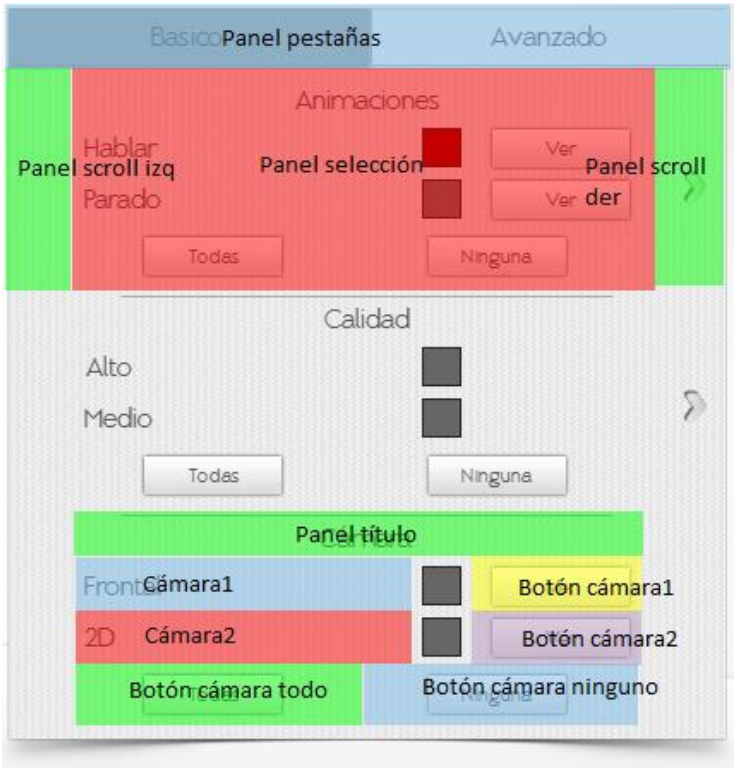

Figura 6.13: Vista de paneles de la pantalla de etapa de exportación del modelo

- **Pantallas adicionales**
  - **Pantalla especial para la realización de la exportación**

Esta pantalla es una pantalla totalmente vacía. Se utiliza en el proceso de exportación del modelo.

  - **Pantalla para los popups finales**

Esta pantalla se compone de un único panel situado en el centro de la pantalla. Este panel está compuesto por un texto y dos botones asociados a las opciones entre las que el usuario puede decidir.

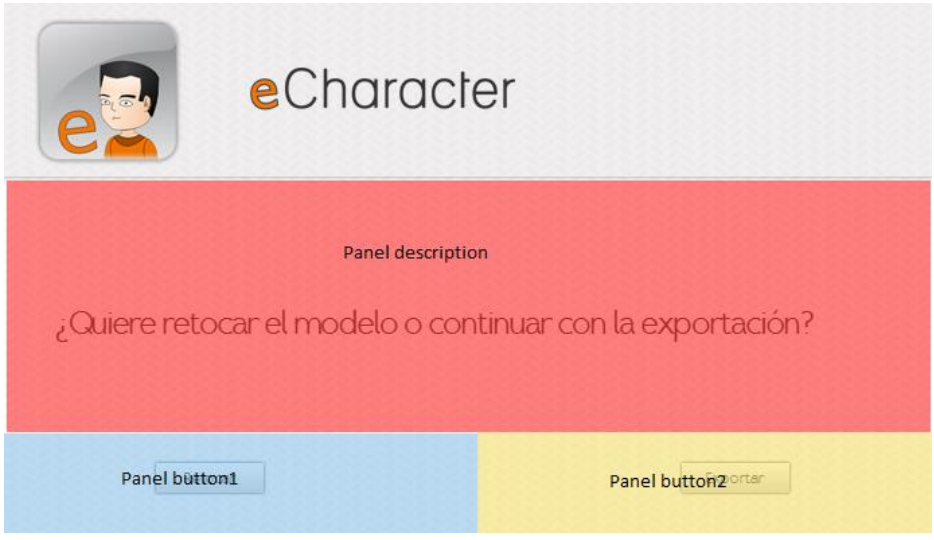

Figura 6.14: Vista de paneles del popup final.

- **Popups para el cambio de color:**
  - **PopUpColorSlider**

Este tipo de popup se divide en dos paneles, uno que contiene una paleta de setenta y dos colores, y otro que contiene tres sliders que representan los colores básicos, rojo, verde y azul, para realizar una selección avanzada del color.

La paleta de colores se compone por setenta y dos paneles con diferentes colores de fondo asociados. Todos los paneles asociados a los colores de la paleta se separan de los otros por paneles vacíos de separación. En la parte inferior de este panel se sitúan tres botones, uno para aplicar el color, otro para cancelar y cerrar el popup sin aplicar cambios en el color, y un tercero que muestra u oculta el panel de selección avanzada del color. En el panel de selección avanzada del color se muestran los tres sliders comentados anteriormente con un panel en el que se muestra el color que se aplicará a la textura o submalla.

Este popup posee un cambio si la textura a la que se la asocia es del tipo doble textura. Este cambio radica en que se muestran dos pestañas, una para la selección del color de la capa base y otra para la selección de color de la capa de detalles.

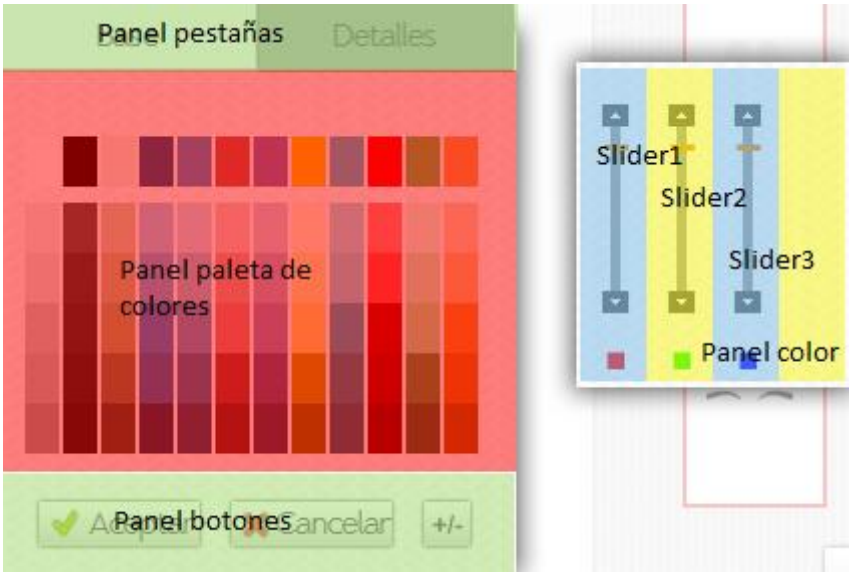

Figura 6.15: Vista de paneles del popup de cambio de color.

  - **PopUpColorDefault**

Este popup está compuesto por cuatro paneles dispuestos verticalmente. El primero y el tercero son paneles de separación. El segundo contiene los diferentes colores que poseen la textura o submalla seleccionada, y el cuarto contiene el botón para cancelar y salir del popup.

El panel de selección del color de la textura se divide horizontalmente en tres paneles. El primer panel y el último panel contienen los botones de navegación entre los diferentes colores asociados a la textura o submalla seleccionada. En el panel central se disponen cuatro paneles dispuestos verticalmente con paneles separadores entre ellos. Dentro de cada uno de estos paneles se encuentran cuatro imágenes también con paneles separadores entre ellas. Cada una de estas imágenes representa

un color asociado a una textura o submalla. Estas dieciséis imágenes varían dinámicamente dependiendo de la página de navegación de colores en la que nos encontremos, ocultándose en el caso de no disponer de ningún color asociado.

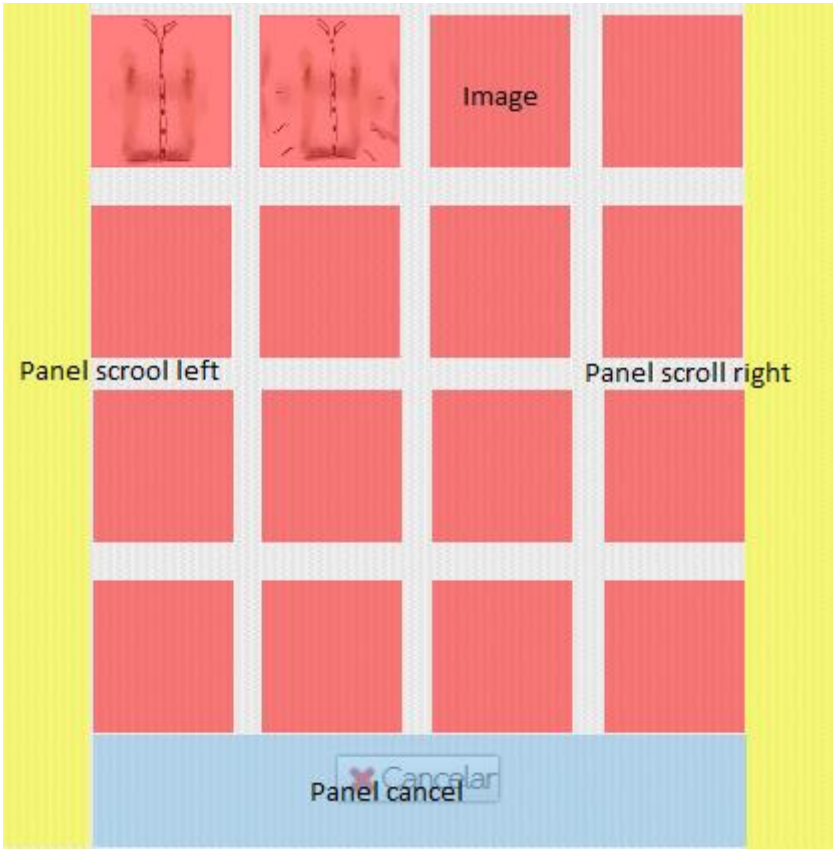

Figura 6.16: Vista de paneles del popup de cambio de color de multiopción.

6.2.2. Algoritmo de coloreado de texturas

Una de las principales características de *eCharacter* es su potencia de configuración, ya que ofrece multitud de texturas posibles que pueden ser personalizadas con cualquier color. Esto es posible gracias a que se ha implementado un algoritmo que realiza un aplica un tratamiento de color sobre la imagen elegida por el usuario, consiguiendo así el color deseado.

Para realizar este cambio de color en las texturas, inicialmente surgió la necesidad de diferenciar las texturas en tres tipos:

- Texturas básicas: Son texturas que poseen un único color en toda la superficie. Éste tipo es el más fácil de modificar. Para modificar su color tan solo es necesario recorrer la matriz asociada a la imagen y cambiar todos los píxeles no transparentes al color seleccionado.

- Dobles texturas: Son texturas compuestas por varias texturas básicas. El cambio de color es igual que en el caso de las texturas básicas.

- Texturas con sombra: Son texturas compuestas por dos imágenes. Una de las imágenes es una textura básica y la otra se trata de una sombra. Este tipo de imágenes son las más costosas a la hora de manipularlas debido al color de la sombra. Para garantizar que el resultado obtenido en el proceso de cambio de color fuera el deseado, también hay que aplicar un tratamiento sobre la sombra para que su color fuese correcto y cambiase el color grisáceo original.

  El algoritmo creado para el coloreado de las sombras consiste en lo siguiente. Se recorre toda la matriz asociada a la imagen de sombra, incluyendo los pixeles con transparencia. En cada pixel se pinta el color seleccionado por el usuario menos la mitad del color que poseía la sombra. En caso de que los píxeles resultantes contengan un valor que no esté comprendido en el rango de color permitido, se truncan con el máximo o mínimo permitido según proceda.

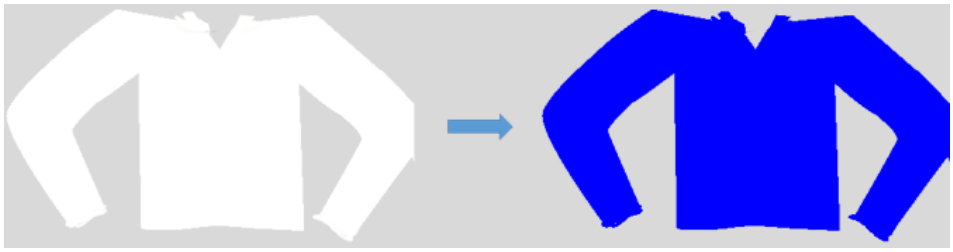

Figura 6.17: Resultado parcial del coloreado de una textura

Una vez finalizado el coloreado de las texturas, se superponen las imágenes que componen a cada textura, esto sólo es necesario en las texturas con sombra y en las dobles texturas. En las texturas con sombra se añade primero la imagen que lleva el color base, y se le superpone la sombra. En las dobles texturas, las imágenes se van superponiendo en el orden en que están definidas en el documento XML.

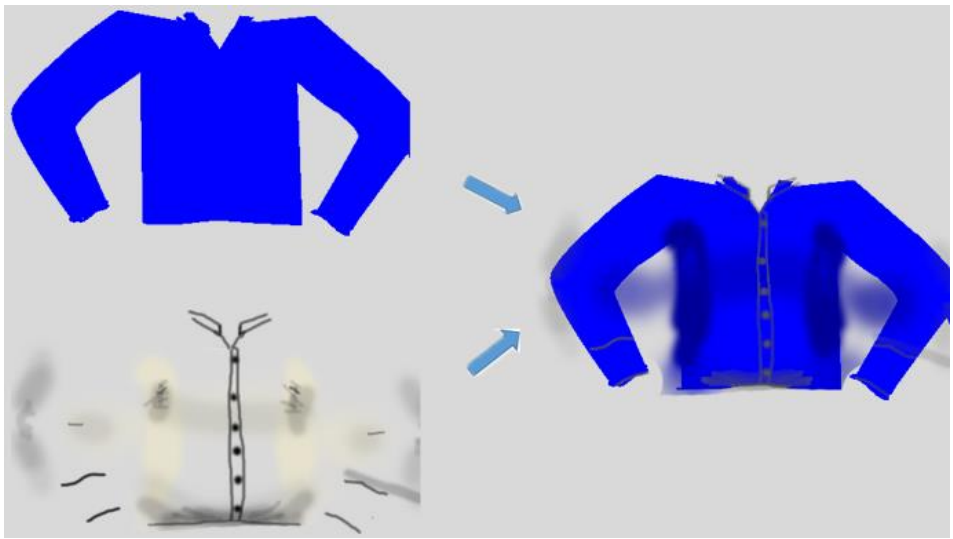

Figura 6.18: Resultado final del coloreado de una textura

El pseudocódigo del algoritmo anterior es el siguiente:

```
ancho = obtenerAncho();

alto = obtener alto();
for (i:=0 to ancho-1){
      for (j:=0 to alto-1){
            //if the pixel isn't transparent
            if (not textura.esTransparente(pixel(i,j)){
                  textura.aplicaNuevoColor(pixel(i,j)nuevoColor.rojo,
                  nuevoColor.verde, nuevoColor.azul, nuevoColor.alfa)
            }
            if(not sombra.esTransparente(pixel(i,j)){
                  colorSombra = colorDeLaSombra(pixel(i,j));

                  if(not (colorSombra.rojo < 85)&&
                  (colorSombra.verde < 85)&&(colorSombra.azul < 85)){
                        rojo:=nuevoColor.rojo-colorSombra.rojo*0.5;
                        verde:=nuevoColor.verde-colorSombra.verde*0.5;
                        azul:=nuevoColor.azul-colorSombra.azul*0.5;
                  }
                  else{
                        rojo:=85;
                        verde:=85;
                        azul:=85;
                  }
                  rojo:=min(255,max(rojo,0));
                  verde:=min(255,max(verde,0));
                  azul:=min(255,max(azul,0));
                  alfa:=transparenciaDeLaSombra(pixel(i,j));
                  sombra.aplicaNuevoColor(pixel(i,j),
                  rojo,verde,azul,alfa);
            }
      }
}
```

6.2.3. Sistema de exportación

El sistema de exportación es una herramienta muy útil en *eCharacter*. Gracias a este sistema, el usuario es capaz de exportar los modelos configurados para poder importarlos en cualquier herramienta de videojuegos que lo soporte.

En esta sección se va a explicar cómo se ha implementado este sistema en *eCharacter*. El proceso que sigue el sistema de exportación es que va realizando todas las capturas para cada animación y éstas se van guardando en el fichero ZIP.

A la hora de implementar el sistema de exportación de *eCharacter* surgieron diversos problemas. El principal problema fue el uso de memoria. Cuando se trabajan con imágenes, el problema de la memoria siempre está presente. Por ello, todos los esfuerzos se centraron en conseguir que el sistema de exportación haga uso del menor espacio de memoria posible, dentro de la relación tiempo-memoria.

Para solucionar este problema se decidió implementar el sistema con el uso de hilos. El problema que había es que el hilo principal, que es el que ejecuta la aplicación, es también el encargado de controlar la animación. Si ese hilo se duerme a la espera de realizar la captura, ésta no se realiza ya que el propio hilo está dormido. Para solucionar esto, se decidió usar dos hilos. El primer hilo es el que ejecutaba la aplicación y el segundo hilo es el que iba ejecutando la animación, pausándola o

reanudándola según las necesidades, realizaba las capturas de la animación y se encargaba de escribir estas imágenes.

Al tener sólo estos dos hilos, surgió otro problema: si el segundo hilo tenía que realizar todo el trabajo de procesamiento de las imágenes además de escribirlas, podría introducir un retardo importante que hiciera que el tiempo entre captura y captura se fuese desajustando. Este desajuste podría provocar que el resultado obtenido de todo este proceso no fuera el esperado. Para solucionar este problema se decidió usar un tercer hilo, que es el encargado de realizar e procesamiento de los frames capturados. El tratamiento de estas imágenes consiste en realizar un recorte de los mismos, centrando de esa forma el modelo y obteniendo unos mejores resultados. Este proceso de recortado de imágenes se ve más detenidamente en el apartado siguiente.

El proceso más detallado es el siguiente:

1) El primer hilo, mediante el bucle de juego, es el que ejecuta la aplicación. En el momento en que el usuario indique a la aplicación que quiere realizar la exportación, este primer hilo lanza un segundo hilo.

2) El segundo hilo, se encarga de comprobar las animaciones seleccionadas por el usuario. Para cada una de ellas, y dependiendo de las calidades y cámaras seleccionadas, realiza un cálculo de cuantos frames tiene que generar y cuánto tiempo tiene que pasar entre captura y captura. También se encarga de ir pausando y reanudando la animación, mientras las capturas se van realizando. Todas las capturas que se van realizando, se van escribiendo en un directorio temporal. Esta decisión se tomó por el problema de la memoria. En un primer momento, las imágenes se iban almacenando en una cola, pero esta idea ocasionaba un gran uso de memoria, el cual era inadmisible. Una vez realizadas todas las capturas de la animación actual, se lanza el tercer hilo. En este momento, el segundo hilo se queda dormido a la espera de que el tercer hilo finalice su ejecución.

3) El tercer hilo es el encargado de realizar el tratamiento de los frames y almacenaje de los mismos en el fichero ZIP. En ese momento, ya se han generado las capturas. Simplemente hay que ir realizando el tratamiento de recorte a cada una y volver a escribirlas. Cuando este hilo finaliza su ejecución, el segundo hilo se reanuda, y se vuelve a repetir el proceso para todas las animaciones restantes.

Aún con esta solución, nos seguimos encontrando con algún pequeño problema de memoria para algunos equipos antiguos. Realizando estudios de la memoria usada, se observó que el uso que se hacía era de 730MB. Por ello, y para evitar posibles problemas, se decidió reservar 1GB para el heap de Java.

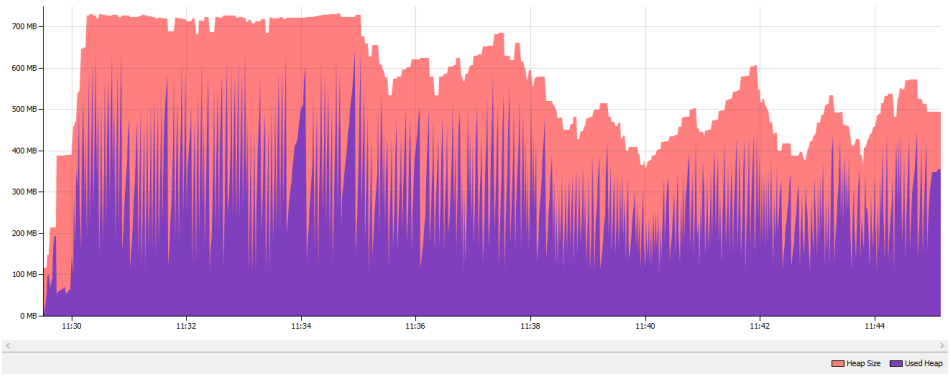

Figura 6.19: Memoria usada

6.2.4. Algoritmo de recorte de los frames de la exportación

Como se ha contado anteriormente, en el proceso de exportación se van realizando una serie de capturas. A estas capturas hay que realizarlas un tratamiento de imagen posterior para que el resultado sea el esperado.

Por este motivo se ha desarrollado un algoritmo que se encarga de realizar un recorte de los frames obtenidos con el sistema de exportación.

Los frames de cada animación tienen que tener el mismo tamaño. Por tanto, el algoritmo se ejecuta para cada animación. El proceso que sigue el algoritmo es el siguiente:

1. Calcular el valor del margen izquierdo del frame.
2. Calcular el valor del margen derecho del frame.
3. Calcular el valor del margen superior del frame.
4. Calcular el valor del margen inferior del frame.

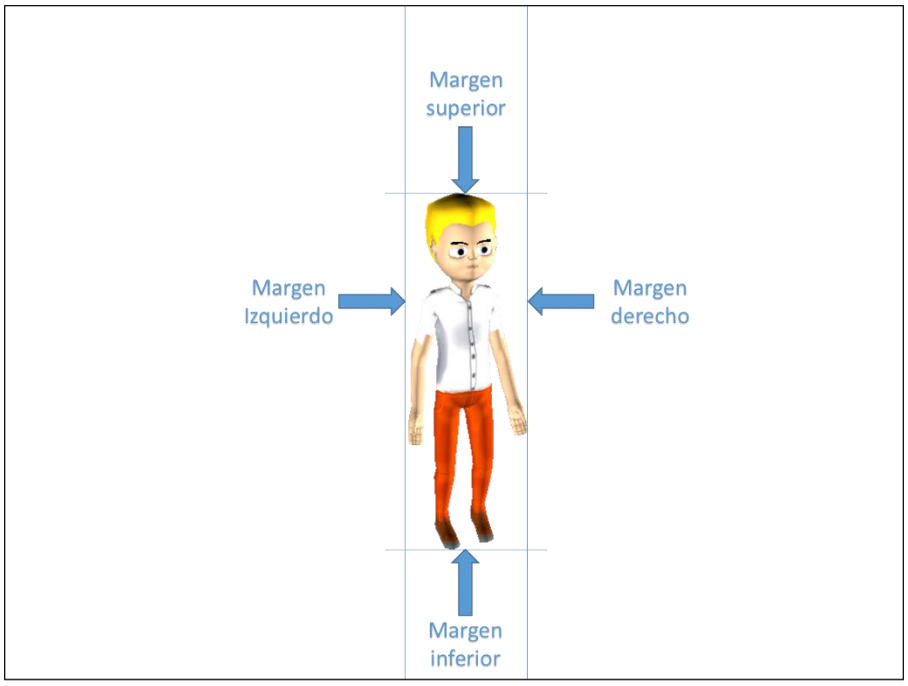

Figura 6.20: Algoritmo de recorte

Este proceso hay que repetirlo con cada frame de la animación. Pero como se ha dicho anteriormente, todos los frames de una animación tienen que tener el mismo tamaño. Por tanto, en todo momento hay que guardar los valores de los márgenes que se van calculando, e irlos actualizando debidamente para cada imagen.

La actualización de los márgenes es sencilla:

- Si el valor que hay guardado del margen izquierdo es mayor que el nuevo valor calculado, se actualiza el valor antiguo con el nuevo.
- Si el valor que hay guardado del margen derecho es menor que el nuevo valor calculado, se actualiza el valor antiguo con el nuevo.
- Si el valor que hay guardado del margen superior es mayor que el nuevo valor calculado, se actualiza el valor antiguo con el nuevo.
- Si el valor que hay guardado del margen inferior es menor que el nuevo valor calculado, se actualiza el valor antiguo con el nuevo.

El pseudocódigo del algoritmo anterior es el siguiente:

```
ancho = obtenerAncho();
alto = obtener alto();

for( i:=0 to ancho-1 ){
      for( j:=0 to alto-1 ){
            if( not esTransparente(pixel(i,j)) ){
                  margenIzquierdoActual:=i;
                  ir_a finBucle1
            }
      }
}
finBucle1
for( i:=ancho-1 to 0 ){
      for( j:=alto-1 to 0 ){
            if( not esTransparente(pixel(i,j)) ){
                  margenDerechoActual:=i;
                  ir_a finBucle2
            }
      }
}
finBucle2
for( i:=0 to alto-1 ){
      for( j:=0 to ancho-1 ){
            if( not esTransparente(pixel(i,j)) ){
                  margenSuperiorActual:=i;
                  ir_a finBucle3
            }
      }
}
finBucle3
for( i:=alto-1 to 0 ){
      for( j:=ancho-1 to 0 ){
            if( not esTransparente(pixel(i,j)) ){
                  margenInferiorActual:=i;
                  ir_a finBucle4
            }
      }
}
finBucle4
```

```
margenIzquierdo:=min(margenIzquierdo, margenIzquierdoActual)
margenDerecho:=max(margenDerecho,margenDerechoActual)
margenSuperior:=min(margenSuperior, margenSuperiorActual)
margenInferior:=max(margenInferior, margenInferiorActual)
```

Una vez que se han procesado todos los frames para hallar los valores de los márgenes, únicamente hay que procesar estos frames, indicando los márgenes que hay que recortar. Para esto, hay que indicar el punto del que partir y el ancho y el alto desde ese punto.

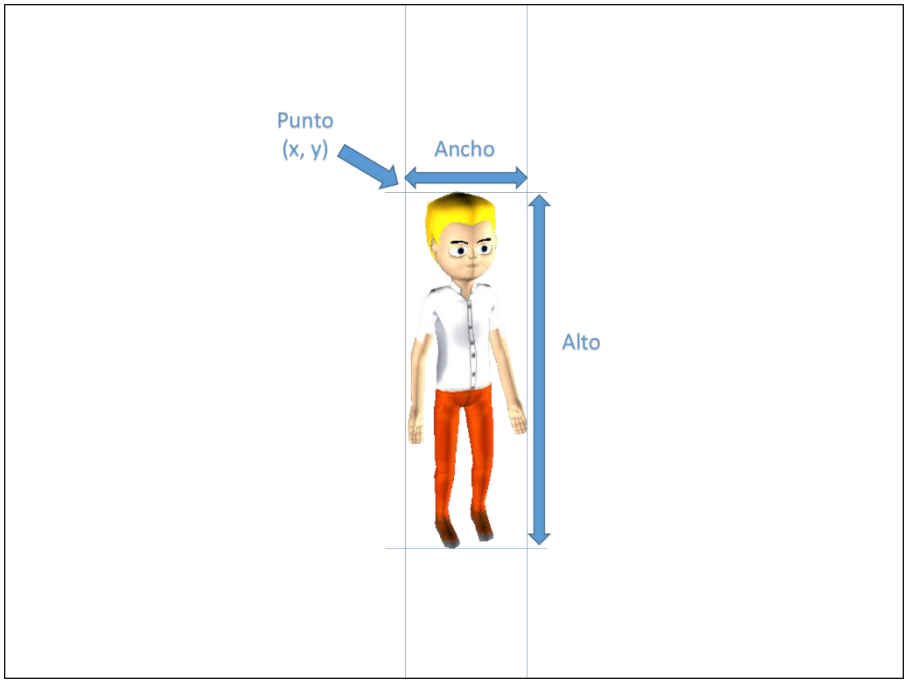

Figura 6.21: Algoritmo de recorte

6.2.5. Búsqueda de recursos

Como ya se apuntó anteriormente (ver el capítulo 1, *Resumen Ejecutivo*) una de las principales aportaciones de *eCharacter* es su carácter multiplataforma. Debido al gran número de recursos que debe cargar la aplicación, era necesario desarrollar un sistema de carga de recursos que fuera independiente de la plataforma donde se estuviera ejecutando.

Durante la ejecución de la aplicación se hacen referencias a dos tipos de recursos diferentes. El primer tipo de recursos son aquellos que son cargados directamente a través del sistema de localización propio de jMonkeyEngine. Este tipo de recursos son todos los relacionados con los modelos 3D y las texturas y submallas que se les pueden aplicar. jMonkeyEngine posee un sistema propio de carga de recursos, pero además ofrece una interfaz que posibilita la implementación de un localizador de recursos propio. Este interfaz se llama *AssetLocator* y posee dos métodos:

- setRootPath: Este método permite fijar cuál es el directorio raíz de nuestro árbol de búsqueda de recursos.

- locale: Este método permite buscar un recurso a partir de un nombre dado y buscando desde la raíz fijado con el método anterior.

Para implementar esta interfaz se ha creado una clase llamada *ResourceLocator,* que es la clase encargada de gestionar toda la carga de recursos.

El segundo tipo de recursos son aquellos que no son cargados a través del sistema propio de jMonkeyEngine. Este tipo de recursos son por ejemplo, todas las imágenes necesarias para la creación de la interfaz. Siguiendo el estilo que impone jMonkeyEngine en su gestor de localización de recursos, se han creado dos métodos, también en la clase *ResourceLocator,* para gestionar la carga de recursos. El primer método permite establecer cuál es la raíz del árbol de búsqueda. El segundo método es el que permite localizar el recurso. Para garantizar que se pueda cargar correctamente los recursos independientemente de la plataforma se hace uso de *InputStream* en lugar de *String* que contenga la ruta del recurso que se quiera cargar.

El funcionamiento para localizar un recurso usando *InputStream* es el siguiente: El método recibe un *InputStream* que representa la localización exacta del recurso que se desea cargar. Lo primero que realiza el método es intentar cargar directamente el recurso indicado. En caso de que esta carga directa no diera resultado, porque el recurso no se encuentra en la ruta exacta que indicaba el *InputStream*, empieza una búsqueda recursiva por todo el árbol de directorios partiendo del nodo raíz que se haya establecido. Si la búsqueda por todo el árbol de directorio no tuviera éxito, se devuelve un error indicado que el recurso solicitado no existe.

7. Métricas

Para realizar una medición del trabajo realizado en *eCharacter*, se ha utilizado la métrica de número de líneas de código.

Se ha utilizado una división en los módulos básicos de la aplicación. Los resultados obtenidos son los siguientes:

Módulo	Nº de líneas
Módulo de inicialización y launcher	367
Módulo de carga de recursos	750
Módulo de control	2228
Módulo de internacionalización	1419
Módulo de datos	6318
Módulo de exportación	1163
Módulo de la interfaz	5522
Módulo de procesado de imágenes	946
Ficheros de familias y modelos	2381

El número total de líneas de código es 21094.

La distribución de estas líneas de código en *eCharacter* viene dado en el siguiente gráfico.

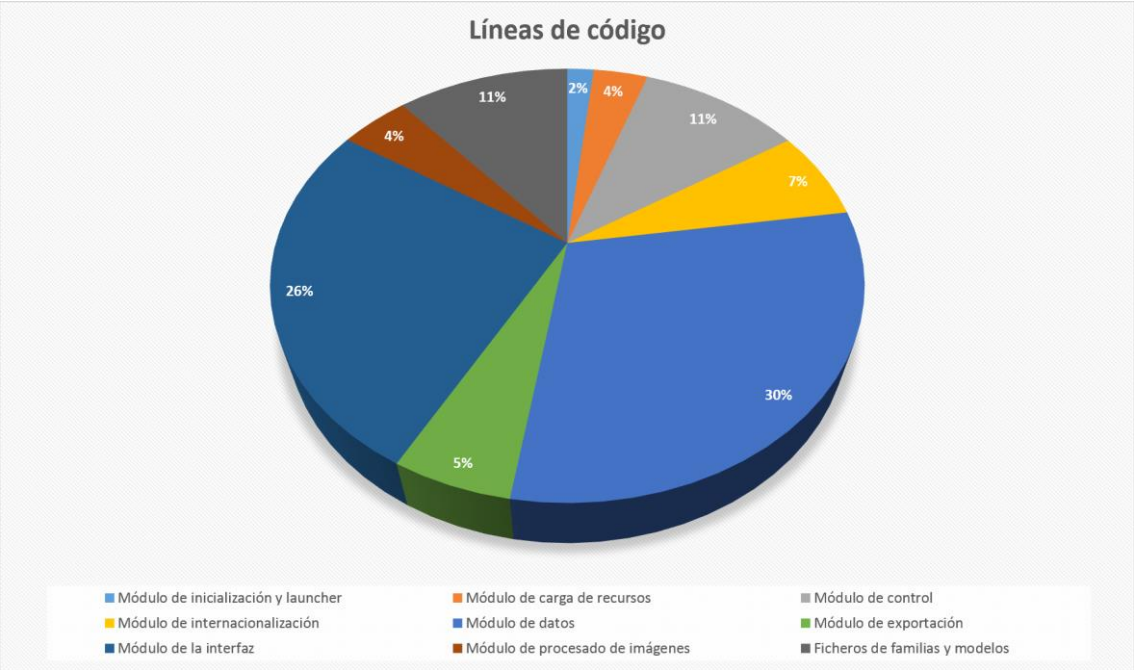

Figura 7.1: Distribución de líneas de código por módulo

8. Pruebas de Usuario

Para realizar un proceso de mejoras y búsqueda de errores en *eCharacter*, están programadas una serie de pruebas con varios usuarios (10-15 personas). Los usuarios que se han buscado son de dos tipos: usuarios finales y expertos en creación de juegos educativos. Cada sesión de evaluación tiene una duración estimada de 30-60 minutos por usuario.

8.1. Objetivos

Los objetivos de las pruebas son los siguientes:

- Evaluar la usabilidad de la herramienta con personas de características similares a los usuarios finales de la herramienta (personas sin elevados conocimientos técnicos, principalmente dedicados a la educación).

- Evaluar la percepción de utilidad de la herramienta a la hora de simplificar el proceso de creación de un videojuego. Para este segundo objetivo se plantea trabajar con expertos en creación de juegos educativos. Esto nos permitirá estimar la probabilidad de obtener una alta aceptación en el campo.

8.2. Métodos

Durante las sesiones de evaluación los asistentes (usuarios finales o expertos) tendrán que realizar una o dos tareas siguiendo los métodos propios de la evaluación de usabilidad de aplicaciones. Se pedirá a los participantes que anoten o verbalicen todos los problemas encontrados y se medirá los tiempos invertidos en la resolución de cada tarea. Así mismo se utilizarán instrumentos cualitativos (cuestionarios) para medir la usabilidad de la aplicación, así como la percepción de utilidad.

El flujo de actividades que se propone para cada sesión de evaluación es la siguiente:

1. Recepción de los participantes. Presentación del proyecto e instrucciones referentes al desarrollo de las pruebas (5 minutos).
2. Presentación de la primera tarea a realizar: crear tres personajes que encajen en las descripciones proporcionadas por los evaluadores (5 minutos).
3. Realización de la primera tarea por parte de los usuarios. Máximo 20 minutos.
4. Distribución del instrumento de evaluación SUS sobre la usabilidad de la aplicación (5 minutos).

(Únicamente para expertos):

5. Presentación de la segunda tarea a realizar: integrar los tres personajes en un juego ya montado con la herramienta eAdventure (5 minutos).
6. Realización de la segunda tarea por parte de los expertos (Máximo 10 minutos).
7. Distribución del instrumento de evaluación sobre la aceptación de la aplicación (5 minutos).
8. Despedida de los asistentes.

Después de realizar todas las pruebas, se hará una recopilación de todos los datos obtenidos y se tendrá en cuenta las opiniones de los usuarios a la hora de realizar las mejoras. De igual manera se procederá a solucionar todos los errores encontrados.

8.3. Instrumentos

Para la evaluación de la usabilidad utilizaremos una traducción del cuestionario SUS (System Usability Scale) (Brooke, 1996). La utilización de este instrumento de medición en detrimento de otros se debe a su sencillez, la facilidad y rapidez en rellenarlo, facilidad en su interpretación y a que proporciona resultados fiables con un pequeño número de usuarios.

El cuestionario SUS consta de diez ítems y cinco opciones de respuesta. Su formato es el siguiente:

1. Perfil del Usuario

Nombre y apellidos _____

Edad _____

Sexo: Masculino Femenino

¿Trabajas en la educación? Sí No

Si es que sí, define tu profesión (por ejemplo, profesor de primaria)

_____

Valora, en una escala de 1 a 7, tu nivel de conocimiento en el ámbito de la creación de videojuegos educativos

1	2	3	4	5	6	7

No tengo ni idea — Soy un experto

2. Evaluación de usabilidad

Por favor, valora de 1 a 5 cómo de acuerdo o en desacuerdo estás con las siguientes afirmaciones referentes al sistema eCharacter:

2.1 Considero que me gustaría usar este sistema frecuentemente.

1(totalmente en desacuerdo)	2	3	4	5(totalmente de acuerdo)

2.2 Tuve la impresión de que el sistema es innecesariamente complejo

1(totalmente en desacuerdo)	2	3	4	5(totalmente de acuerdo)

2.3 Me pareció que el sistema fue fácil de utilizar

1(totalmente en desacuerdo)	2	3	4	5(totalmente de acuerdo)

2.4 Creo que necesitaría la ayuda de personal técnico para ser capaz de usar el sistema.

1(totalmente en desacuerdo)	2	3	4	5(totalmente de acuerdo)

2.5 Tuve la impresión de que las distintas funcionalidades del sistema estaban bien integradas

1(totalmente en desacuerdo)	2	3	4	5(totalmente de acuerdo)

2.6 Me pareció que hay demasiadas inconsistencias en el sistema

1(totalmente en desacuerdo)	2	3	4	5(totalmente de acuerdo)

2.7 Me imagino que la mayoría de la gente aprendería a utilizar este sistema muy rápido.

1(totalmente en desacuerdo)	2	3	4	5(totalmente de acuerdo)

2.8 Me pareció que el sistema era muy difícil de usar.

1(totalmente en desacuerdo)	2	3	4	5(totalmente de acuerdo)

2.9 Me sentí muy seguro y confiado utilizando el sistema.

1(totalmente en desacuerdo)	2	3	4	5(totalmente de acuerdo)

2.10 Necesité aprender muchas cosas antes de empezar a manejar el sistema con soltura.

1(totalmente en desacuerdo)	2	3	4	5(totalmente de acuerdo)

Nota: cuestionario adaptado de:
http://www.measuringusability.com/sus.php

8.4. Cómputo de resultados e interpretación

Una vez que se han realizado las pruebas se deben computar los resultados y obtener un valor significativo que nos indique el nivel de usabilidad. Para computar los resultados se siguen las siguientes indicaciones:

1. Para los ítems impares se resta uno de la respuesta del usuario.

2. Para los ítems pares se restan a cinco el valor de las respuestas.

3. Esto escala los valores con un intervalo de cero a cuatro. Después se suman todos los valores de las respuestas de los usuarios y se multiplica por 2.5, dejando la escala en un rango de 0 a 100.

Según como se indica en la página web (www.measuringusability.com/sus.php) y tras el análisis de los resultados de aplicar este test en al menos 500 proyectos diferentes, el valor que se ha establecido para que una aplicación tenga un buen nivel de usabilidad debe ser de al menos 68 puntos.

9. Manuales

En este capítulo se incluyen dos manuales que explican desde dos puntos de vista diferentes cómo ha de usarse la aplicación. Por un lado, se proporciona un breve manual de usuario, que contiene indicaciones sobre cómo instalar y utilizar la aplicación. No obstante, la lectura de este documento no es imprescindible pues la aplicación tiene una interfaz intuitiva y su aprendizaje es sencillo, incluso de manera autodidacta. Por otro lado se proporciona un manual orientado al desarrollador que profundiza en la estructura y contenido de los ficheros XML que configuran la aplicación. Conocer esta estructura permitirá a otros desarrolladores aumentar el conjunto de modelos y recursos base que utiliza la aplicación.

9.1. Manual del usuario

Este manual está orientado al usuario final. Se trata de un manual básico sobre cuál es el proceso que debe seguir el usuario para realizar la configuración de un modelo. En caso de querer introducir nuevas familias, nuevos modelos o configurar algún aspecto de la aplicación, deberá remitirse al *Manual del desarrollador*, dentro de esta misma sección.

9.1.1. Instalación

La aplicación *eCharacter* se distribuye teniendo en cuenta el mercado actual. Se ofrecen tres instaladores para las tres principales plataformas (Windows, Linux y Mac OSX) incluyendo todos los recursos necesarios para que la aplicación funcione en estos sistemas, y una versión multiplataforma en la que se incluye el fichero JAR, para ejecutar la aplicación en cualquier sistema que soporte Java, y todos los recursos necesarios para su ejecución.

En el caso de usar el instalador, el sistema es totalmente automático y únicamente se deberá seleccionar el directorio en el que se quiere instalar la aplicación.

9.1.2. Ejecución y Uso

Tras ejecutar la aplicación, lo primero que se obtiene es una pantalla en la que debemos seleccionar con qué familia se desea trabajar. Dentro de esta familia, en la parte derecha de la pantalla, se debe seleccionar el modelo deseado. Con las flechas situadas al lado del modelo, se pueden previsualizar todos los modelos disponibles, dentro de la familia seleccionada.

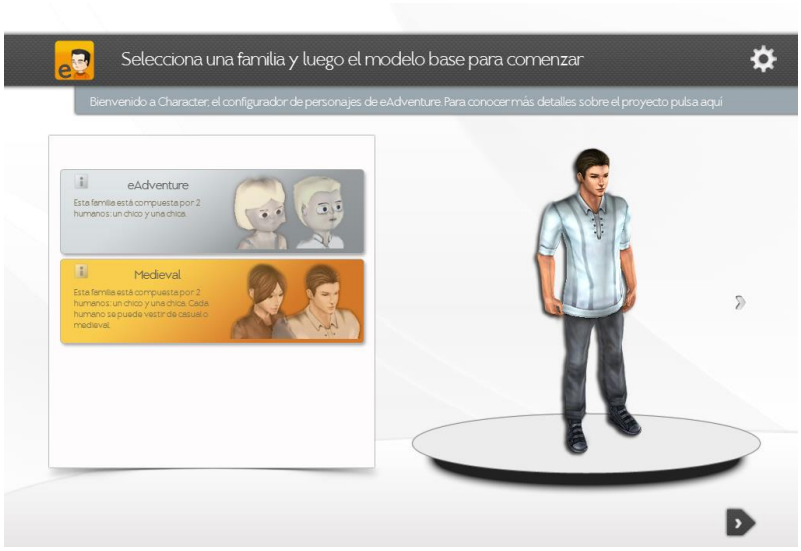

Figura 9.1: Imagen de selección de modelo

Cuando se haya elegido el modelo con el que se quiere trabajar, hay que hacer click en la flecha situada en la parte inferior derecha de la pantalla, para poder iniciar el proceso de configuración. Cabe destacar que el flujo que se va a seguir en este manual es lineal, es decir, se van a ir realizando todas las etapas en orden. Aun así, el usuario tiene la libertad de ir realizando cada etapa en desorden, sin necesidad de tener que pasar por todas las etapas.

Una vez iniciado el proceso de configuración, y dependiendo de las etapas que se hayan definido en esa familia, en primer lugar nos encontraremos con la(s) etapa(s) de escalación. En esta etapa, se pueden seleccionar dos modos: modo básico y modo avanzado.

En el modo básico, se ofrecen una serie de complexiones predefinidas y el usuario puede elegir entre cualquiera de ellas. En caso de querer tener un control más fino, el usuario puede seleccionar el modo avanzado. En este modo, el usuario puede seleccionar cuáles son las partes del cuerpo que quiere escalar. Según se vayan modificando los slider, se podrá observar el resultado final.

Figura 9.2: Imagen de escalación del modelo en modo básico

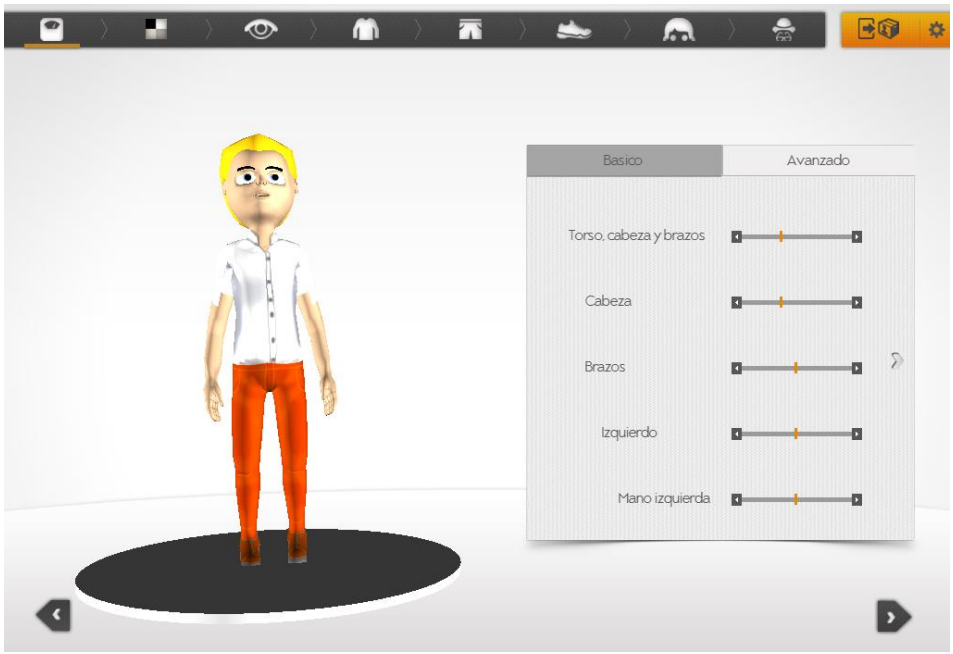

Figura 9.3: Imagen de escalación del modelo en modo avanzado

Una vez realizadas todas las escalaciones deseadas, se procederá a realizar todos los cambios relacionados con texturas o submallas. En estas etapas, se podrán cambiar las texturas haciendo click en el icono correspondiente, o en caso de que se pueda cambiar de color, aparecerá un botón en la parte inferior. Si pulsamos en ese botón, aparecerá un popup en el que se puede seleccionar el color dentro de un conjunto de colores dado, o haciendo click en el botón +/-, podremos seleccionar el color mediante sliders. Cada slider se corresponde con un color del sistema RGB. La R es el color Rojo, la G es el color Verde y la B es el color azul.

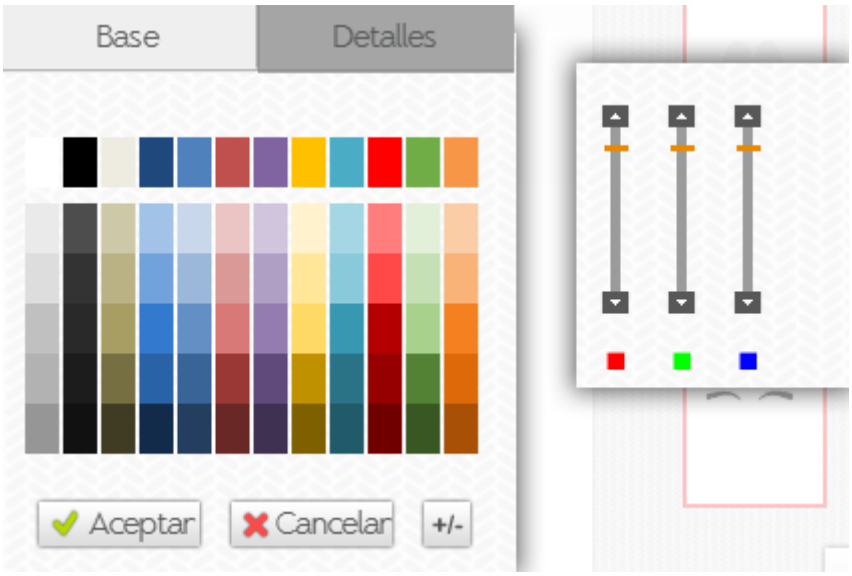

Figura 9.4: Imagen de selección del color

Cuando ya se hayan configurado todas las texturas y submallas, procederemos a realizar la exportación. Para ello hay que dirigirse al botón  y así accederemos a dicha etapa.

En esta etapa, nos encontramos con dos modos de uso: el modo básico y el modo avanzado.

En el modo básico, se pueden previsualizar las animaciones haciendo click en el botón “Ver”. En el caso de querer seleccionar una animación, hay que hacer click en el cuadro que hay a la izquierda del nombre de la animación. También se dispone de un botón para seleccionar todas las animaciones o deseleccionarlas.

En el modo avanzado, aparte de la funcionalidad que hay en el modo básico, también se pueden elegir cuáles son las calidades con las que se desea realizar la exportación. Para ello, hay que hacer click en el cuadro que se encuentra a la izquierda del nombre de la calidad. Por último, se pueden elegir las cámaras con las que realizar la exportación. En el caso de querer previsualizar una cámara, hay que hacer click en el botón “Ver”. En el caso de querer seleccionar una cámara, hay que hacer click en el cuadro que hay a la izquierda del nombre de la cámara.

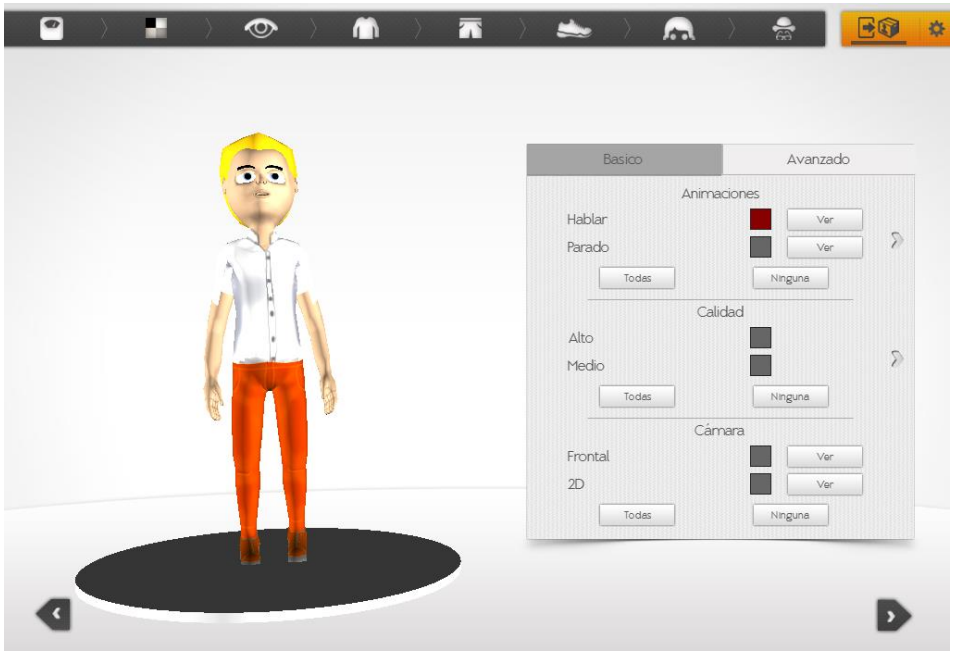

Figura 9.5: Imagen de la etapa de exportación

Por último, hay que hacer click en la flecha situada en la parte inferior derecha para realizar el proceso de exportación.

El fichero ZIP resultante de realizar la exportación, se encuentra en un directorio llamado “eCharacter.zip”, dentro del directorio correspondiente del usuario.

9.2. Manual del desarrollador

Una de las principales funcionalidades que ofrece *eCharacter*, y que se ha comentado en el capítulo *Funcionalidades*, es permitir la incorporación de nuevas familias y modelos. Para ello hay que crear y editar una serie de ficheros siguiendo unas pautas básicas. Este manual está orientado al segundo perfil de usuario, aquel que desea crear sus propias familias de modelos.

Para crear una nueva familia de modelos hay que rellenar dos ficheros XML siguiendo unas XSD que definen la estructura básica de estos ficheros y que el sistema es capaz de interpretar. El manual está dividido en dos secciones, en las cuáles se analiza en profundidad cada una de las XSD para que el usuario desarrollador pueda crear fácilmente sus nuevas familias siguiendo estas instrucciones.

9.2.1. Fichero XML de la familia.

En este XML es donde se permite al desarrollador tener una cierta libertad a la hora de crear la interfaz de configuración del modelo. El desarrollador puede definir las etapas que tiene el proceso de configuración de sus modelos así como las opciones que se van a mostrar en cada etapa del proceso. La estructura que define a una familia tiene tres nodos principales:

- Metadatos: En este nodo se incluye toda la información relativa a la familia que se está creando así como a su autor. Deben aparecer los siguientes campos:

    - Nombre de la familia.
    - Breve descripción de la familia y que se muestra en la interfaz de selección de familias de la aplicación.
    - Autor de la familia de modelos.
    - Ruta del fichero donde se encuentre el icono que representa a la familia.
    - Directorio donde se encuentran todos los ficheros necesarios para la internacionalización de la familia. En este directorio se debe incluir un fichero por cada idioma disponible.

  Opcionalmente, también se puede incluir la licencia bajo la cual se puede distribuir la familia y todos sus recursos y una URL que el autor quiera incorporar para obtener información adicional acerca de la familia desarrollada o sobre él mismo.

- Etapas: Éste es el nodo principal que define a una familia. En él se puede definir la secuencia de etapas que forman el proceso de configuración para alguno de los modelos pertenecientes a esta familia. La secuencia de etapas se define en el XML siguiendo una **secuencia, de tres o más etapas, que determina el orden en el que aparecen en la interfaz.** En *eCharacter* se ofrecen tres tipos diferentes de etapas:

    - Etapa de escalación: Una etapa de escalado es aquella en la que se puede realizar un escalado parcial de algunas de las partes del modelo. En este tipo de etapas el usuario puede definir **un conjunto de controladores** que luego puede asignar a los huesos del modelo que desee. Los campos de información que contiene cada controlador son:

        - Etiqueta de controlador: Nombre asociado al controlador que aparece en la interfaz
        - **Identificador único** del controlador para posteriormente poder asociar unívocamente hueso y controlador.
        - Nivel: Este atributo permite establecer una estructura en niveles entre los distintos controladores que se han definido. De esta manera se puede crear controladores cuyo ámbito de acción sea mayor, si tienen un nivel menor, que otros que tengan un nivel mayor. Por ejemplo se puede definir un controlador de nivel 0 que escale todo el torso, y dos controladores de nivel 1 que escalen únicamente el brazo izquierdo o el derecho.

      Además del conjunto de controladores deben aparecer **obligatoriamente**

- **Identificador único** de la etapa.
- Icono de la etapa que se muestra en el banner superior de la interfaz durante el proceso de configuración.
- Nombre de la etapa. Este texto se usa como información complementaria al icono y se muestra mediante una ventana emergente si el usuario lo solicita.

Este tipo de etapas deben **aparecer al menos una vez** y siempre al principio del proceso de configuración del modelo.

- Etapas multietapas: Una multietapa es una única etapa dentro del proceso de configuración, la cual **está formada por una secuencia de una o más subetapas** que comparten alguna característica común. Las subetapas pueden ser de dos tipos:

  - Mesh Substage (Subetapa de malla): En estas etapas se pueden añadir, eliminar o modificar distintas tipos de mallas que vayan asociados a los modelos, como pueden ser por ejemplo, distintos tipos de peinados.

  - Texture Substage (Subetapa de textura): El funcionamiento es el mismo que en la subetapa anterior con la salvedad de que en esta ocasión los elementos intercambiables son texturas.

  Además ambas subetapas contienen los mismos campos de información:

  - Icono de la etapa que se muestra en el banner superior de la interfaz durante el proceso de configuración.
  - Nombre de la etapa: Este texto se usa como información complementaria al icono y se muestra mediante una ventana emergente si el usuario lo solicita.
  - **Un identificador único** para poder asociar las texturas o mallas cada subetapa.
  - Un atributo booleano que permite indicar la posibilidad de la multiselección de elementos dentro de una misma subetapa.

  Este tipo de etapas deben **aparecer al menos una vez** y siempre después de la secuencia de etapas de escalado.

- Etapa de exportación: En esta etapa es donde el usuario puede configurar el proceso de exportación seleccionando las animaciones a exportar. Esta etapa es **obligatoria y debe aparecer siempre al final** del proceso de configuración de los modelos. En esta etapa se definen los valores de las calidades de muestreo con los que se generan las animaciones, así como los distintos tipos de vista. Los elementos que tiene esta etapa son:

    - FPS (Frames por segundo): Lista de pares de la forma <nombre calidad, valor de FPS>. Esta lista **debe contener al menos un elemento**.

    - Cámaras: Aquí se pueden definir los distintos tipos de cámaras. Una cámara queda unívocamente definida por dos puntos y un vector de orientación (para más información acerca de cómo se definen las cámaras ver el apartado *Introducción al mundo 3D* del capítulo *Introducción*):
        - Localización: punto donde está situada la cámara.
        - Dirección: punto hacia el que está mirando la cámara.
        - Vector Up: vector que indica la orientación de la cámara

      Igual que con las calidades, en esta lista debe aparecer al menos una cámara definida.

- Modelos: El último de los nodos principales que definen a una familia es una lista de los modelos que están asociados a dicha familia. Un modelo dentro de su familia queda definido por la siguiente información:

    - Nombre del modelo que aparece en la interfaz de selección.
    - Ruta del icono del modelo que se muestra en la interfaz de selección.
    - Ruta del fichero .xml que describe al modelo. Este fichero .xml debe estar creado siguiendo el *XML Schema* de modelos que se comenta en el siguiente apartado.

A continuación se incorpora el código del fichero XSD que define la estructura que deben seguir los ficheros que se creen para las nuevas familias. Este código esta comentado para facilitar su compresión a los desarrolladores.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified" attributeFormDefault="unqualified">
	<xs:element name="family">
		<xs:complexType>
			<xs:sequence>
			<!--This element contains all information about
			the family. License and URL are optional
			elements.-->
				<xs:element name="metadata">
					<xs:complexType>
						<xs:sequence>
							<xs:element name="license"
							type="licenseType" minOccurs="0"
							maxOccurs="1"/>
```

```
                                        <xs:element name="name"
                                        type="xs:string"/>
                                        <xs:element name="description"
                                        type="xs:string"/>
                                        <xs:element name="author"
                                        type="xs:string"/>
                                        <xs:element name="URL"
                                        type="xs:string" minOccurs="0"
                                        maxOccurs="1"/>
                                        <!--Internationalization files
                                        should be in this path-->
                                        <xs:element name="languagesPath"
                                        type="xs:string"/>
                                        <xs:element name="iconPath"
                                        type="xs:string"/>
                                </xs:sequence>
                        </xs:complexType>
                </xs:element>
                <!-- This element contains information about the GUI.
                        The order in which stages are defined, defines
                        the order in which show it -->
                <xs:element name="stages">
                        <xs:complexType>
                                <xs:sequence>
                                        <xs:element name="scaleStage"
                                        type="scaleStageType" minOccurs="1"
                                        maxOccurs="10"/>
                                        <xs:element name="multiStage"
                                        type="multiStageType" minOccurs="1"
                                        maxOccurs="10"/>
                                        <!-- Animation stage is
                                        obligatory.-->
                                        <xs:element name="animationStage"
                                        type="animationStageType"/>
                                </xs:sequence>
                        </xs:complexType>
                </xs:element>
                <xs:element name="modelsRef">
                        <xs:complexType>
                                <xs:sequence>
                                        <xs:element name="modelRef"
                                        type="modelRefType" minOccurs="1"
                                        maxOccurs="unbounded"/>
                                </xs:sequence>
                        </xs:complexType>
                </xs:element>
        </xs:sequence>
    </xs:complexType>
</xs:element>

<xs:complexType name="licenseType">
        <xs:sequence>
                <xs:element name="title" type="xs:string"/>
                <xs:element name="terms" type="xs:string"/>
                <xs:element name="URL" type="xs:string" minOccurs="0"
                maxOccurs="1"/>
        </xs:sequence>
</xs:complexType>

<xs:complexType name="multiStageType">
        <xs:sequence>
                <!-- If iconPath exists, it will be displayed in the top
                banner instead of stageLabel. -->
                <xs:element name="iconPath" type="xs:string"/>
                <xs:element name="subStage" type="subStageType"
                minOccurs="1" maxOccurs="unbounded"/>
        </xs:sequence>
        <xs:attribute name="stageLabel" type="xs:string" use="required"/>
</xs:complexType>

<xs:complexType name="subStageType">
        <xs:attribute name="subStageType" use="required">
                <xs:simpleType>
                        <xs:restriction base="xs:string">
                                <xs:enumeration value="meshSubStage"/>
                                <xs:enumeration value="textureSubStage"/>
                        </xs:restriction>
```

```
                </xs:simpleType>
            </xs:attribute>
            <xs:attribute name="subStageLabel" type="xs:string"
            use="required"/>
            <xs:attribute name="idPanel" type="xs:ID" use="required"/>
            <!--If multiselection is true, users would be able to select many
            textures or submeshes in the same substage.-->
            <xs:attribute name="multiselection" type="xs:boolean"
            use="optional"/>
        </xs:complexType>

        <xs:complexType name="scaleStageType">
            <xs:sequence>
                <!-- If iconPath exists, it will be displayed in the top
                banner instead of stageLabel. -->
                <xs:element name="iconPath" type="xs:string"/>
                <xs:element name="boneController" minOccurs="1"
                maxOccurs="15">
                    <xs:complexType>
                        <xs:attribute name="controllerLabel"
                        type="xs:string" use="required"/>
                        <xs:attribute name="idController" type="xs:ID"
                        use="required"/>
                        <!-- Level allow us make a tree structure to
                        organize the controllers. The highest level is
                        0 and it increase in lower levels.-->
                        <xs:attribute name="level" type="xs:integer"
                        use="required"/>
                    </xs:complexType>
                </xs:element>
            </xs:sequence>
            <xs:attribute name="stageLabel" type="xs:string" use="required"/>
            <xs:attribute name="idPanel" type="xs:ID" use="required"/>
        </xs:complexType>

        <xs:complexType name="animationStageType">
            <xs:sequence>
                <xs:element name="fps" type="fpsType" minOccurs="1"
                maxOccurs="unbounded"/>
                <xs:element name="camera" type="cameraType" minOccurs="1"
                maxOccurs="unbounded"/>
            </xs:sequence>
            <xs:attribute name="stageLabel" type="xs:string" use="required"/>
        </xs:complexType>

        <xs:complexType name="fpsType">
            <xs:attribute name="qualityLabel" type="xs:string"
            use="required"/>
            <xs:attribute name="fpsValue" type="xs:integer" use="required"/>
        </xs:complexType>

        <xs:complexType name="cameraType">
            <xs:sequence>
                <xs:element name="upAxis" type="vectorType"/>
                <xs:element name="location" type="vectorType"/>
                <xs:element name="direction" type="vectorType"/>
            </xs:sequence>
            <xs:attribute name="cameraLabel" type="xs:string"
            use="required"/>
        </xs:complexType>

        <xs:complexType name="vectorType">
            <xs:attribute name="valueX" type="xs:float" use="required"/>
            <xs:attribute name="valueY" type="xs:float" use="required"/>
            <xs:attribute name="valueZ" type="xs:float" use="required"/>
        </xs:complexType>

        <xs:complexType name="modelRefType">
            <xs:sequence>
                <xs:element name="modelLabel" type="xs:ID"/>
                <xs:element name="iconPath" type="xs:string"/>
                <xs:element name="modelPath" type="xs:string"/>
            </xs:sequence>
        </xs:complexType>
</xs:schema>
```

9.2.2. Fichero XML del modelo.

Este XML es el que permite al desarrollador definir un modelo junto con los conjuntos de texturas y submallas que se le pueden aplicar, según la interfaz diseñada en el XML de familia. Además se pueden personalizar los distintos tipos de complexión física que se ofrecen, así como las opciones de escalaciones individuales de huesos, todas las submallas asociadas al modelo, la ruta dónde se encuentra el modelo y sus ficheros de internacionalización.

Este XML dispone de tres nodos principales:

- Malla principal: Este es el nodo en el que se tiene que definir todo lo relacionado con el modelo propiamente dicho. **Debe aparecer obligatoriamente una única vez**. A su vez dispone de varios nodos:

    - Path: En este nodo es dónde hay que poner la ruta en la cual se encuentra el modelo. Esta ruta es la que usada por la aplicación para cargar el modelo. La aplicación soporta la carga de modelos en formato .mesh.xml y .j3o.

    - Transformaciones: Este nodo es una lista de todas las transformaciones iniciales que hay que aplicar al modelo para situarlo en su posición inicial. Contiene la siguiente información:

        - Tipo de transformación: En este campo se indica que tipo de transformación hay que aplicar:
            - Scale
            - Rotate
            - Translate
        - Valor X: Es el valor que tiene que tomar la transformación en el eje X.
        - Valor Y: Es el valor que tiene que tomar la transformación en el eje Y.
        - Valor Z: Es el valor que tiene que tomar la transformación en el eje Z.

    - Texturas de la malla principal: En este nodo contiene una lista de todas las texturas que tiene el modelo. El sistema soporta cuatro tipos distintos de texturas. Todas las texturas tienen una información común:

        - Path del icono: Es la ruta donde se encuentra el icono para mostrar en la interfaz. Si no existiera un icono definido, el sistema le asigna uno por defecto
        - idPanelRef: Identificador de la etapa al que irá asociada dicha textura.
        - Layer: Prioridad a la hora de aplicar la secuencia de las distintas texturas seleccionadas sobre el modelo principal y garantizar

que todas ellas sean visibles. Menores valores indicarán mayor prioridad (elemento más prioritario = 0).

- idTexture: Es el identificador de la textura. **Debe ser único**.
- Text: Es el texto que se mostrará en el tooltip de esta textura.

Los tipos de texturas que soporta el sistema son las siguientes:

- Simple Texture (Textura simple): En este tipo de texturas no se permite el cambio de color, es decir, la textura se aplica sobre el modelo tal y como es. La información que contiene es la siguiente:
  - Path: Es la ruta donde se encuentra el fichero de la textura.
  - Default: Este atributo indica si la textura ha de ser cargada inicialmente o no.
  - Text: Es el texto que se mostrará en el tooltip de esta textura.

- MultiOption Texture (Textura multiopción): En este tipo de texturas se podrán modificar el color, pero eligiendo entre un conjunto finito de posibilidades que proporcione el usuario. Así pues, se definirá una secuencia de texturas entre las cuales se podrá elegir e intercambiarlas.

  Contiene una lista con todas las texturas que tienen que aparecer a la hora de cambiar el color:

  - Option Texture: Este elemento es **obligatorio** y puede haber varios. Contiene la siguiente información:
    - Path: Es la ruta donde se encuentra el fichero de la textura.
    - Path del icono: Es la ruta donde se encuentra el icono para mostrar en la interfaz. Si no existiera un icono definido, el sistema le asigna uno por defecto
    - idSubTexture: Es el identificador de la sub textura. **Debe ser único**.
    - Default: Este atributo indica si la textura ha de ser cargada inicialmente o no.
    - Text: Es el texto que se mostrará en el tooltip de esta textura.

- Base-Shadow Texture (Textura Base-Sombra): Este tipo de texturas está basado en dos capas. La primera capa o capa base, a la que se le puede cambiar el color eligiendo cualquier combinación posible de los valores RGB, y una segunda capa que contiene las sombras para dar un mayor realismo a la textura. Esta segunda capa, a la que no se le puede aplicar

tratamiento de color, se superpone a la primera, una vez que se le haya realizado el procesamiento para cambiarla al color deseado.
Contiene la siguiente información:

- Path: Es la ruta donde se encuentra el fichero de la textura.
- Path de la sombra: Es la ruta donde se encuentra el fichero de la sombra. Es un elemento opcional, ya que puede no ser necesario que haya esta capa.
- Color default: Es el color por defecto con el que tiene que ser cargada la textura. Contiene el código RGB necesario para definir el color. Este color viene dado mediante el valor para el rojo (R), para el verde (G) y para el azul (B).
- Default: Este atributo indica si la textura ha de ser cargada inicialmente o no.
- Text: Es el texto que se mostrará en el tooltip de esta textura.

■ Double Texture (Textura doble): El funcionamiento es similar a la textura anterior con la salvedad de que en este caso se le puede aplicar el tratamiento de color a ambas capas. Tendremos una capa base y una capa de detalles que irá sobre la primera. La información que contiene es la siguiente:

- Path base: Es la ruta donde se encuentra el fichero de la textura de la capa base.
- Path detalles: Es la ruta donde se encuentra el fichero de la capa de detalles.
- Text: Es el texto que se mostrará en el tooltip de esta textura.

○ Huesos: Contiene una secuencia de los huesos del modelo a los que se les puede aplicar un escalado parcial mediante los sliders definidos previamente en el fichero XML de la familia a la que pertenece el modelo. La información que contiene cada elemento de la secuencia es la siguiente:

■ BoneName: Nombre del hueso del esqueleto del modelo.
■ idControllerRef: Es una referencia al identificador único de slider definido en el XML de la familia y que permite realizar la asociación hueso-controlador. Un mismo hueso puede estar asociado a varios sliders.
■ MaxValue, MinValue y Default Value: definen el rango de valores posibles de escalación que se asignan al slider del controlador, así como un valor por defecto para el mismo.

- Complexiones físicas: Contiene una secuencia de complexiones físicas predefinidas que se ofrecen al usuario como “complexiones base” a la hora de personalizar su modelo. La información que contiene cada elemento de la secuencia es la siguiente:
  - Label: Es el nombre de la complexión. Este nombre es el que se mostrará en la interfaz.
  - idPanelRef: Es una referencia al identificador único del panel en el que tiene que aparecer la complexión.
  - idPhysicalBuild: Es un **identificador único** de la complexión.
  - **Secuencia de escalaciones** que indicarán el orden en el cuál hay que realizarlas sobre el modelo original para conseguir la complexión deseada. Cada escalación tiene una valor para cada componente (x, y, z) así como el nombre del hueso sobre el cual hay que aplicar la transformación. **Existe un carácter especial para el nombre del hueso que el sistema lo interpretará como un escalado global.** Este carácter es “ALL”.

- Lista de submallas: Es una **secuencia opcional** de todas aquellas submallas intercambiables que puedan ser utilizadas para personalizar el modelo principal, por ejemplo distintos tipos de peinados o una falda. De cara a la funcionalidad de la aplicación estas submallas son tratadas como elementos que van asociados a ciertas pantallas de la interfaz definidas en el XML de familia y no como el modelo principal sobre el que se está trabajando. Cada elemento de esta secuencia contiene la siguiente información:

  - Path: Es la ruta del fichero que define a la submalla. Los tipos de ficheros admitidos son: .mesh.xml y .j3o.
  - Transformaciones: Este nodo es una lista de todas las transformaciones iniciales que hay que aplicar al modelo para situarlo en su posición inicial. Contiene la siguiente información:

    - Tipo de transformación: En este campo se indica que tipo de transformación hay que aplicar:
      - Scale
      - Rotate
      - Translate
    - Valor X: Es el valor que tiene que tomar la transformación en el eje X.
    - Valor Y: Es el valor que tiene que tomar la transformación en el eje Y.
    - Valor Z: Es el valor que tiene que tomar la transformación en el eje Z.

  - Textura de la submalla: Este campo define la textura de la submalla. Contiene uno de cuatro tipos de texturas definidos anteriormente (Simple Texture, Multioption Texture, Base-Shadow Texture y Double Texture). En este caso, sólo puede aparecer una de las cuatro. El

tratamiento y la manera de definirla es igual que en el caso de las texturas del modelo. En este caso, las texturas sólo contienen en común los siguientes atributos:

- Layer: Prioridad a la hora de aplicar la secuencia de las distintas texturas seleccionadas sobre el modelo principal y garantizar que todas ellas sean visibles. Menores valores indicarán mayor prioridad (elemento más prioritario = 0).
- idTexture: Es el identificador de la textura. **Debe ser único**.

- Path del icono: Es la ruta del icono de la submalla que se mostrará en la etapa correspondiente de la interfaz. Si no estuviera definido el sistema le asignará uno por defecto.
- idPanelRef: Es una referencia al identificador único del panel en el que tiene que aparecer la submalla.
- associatedBone: Es el nombre del hueso del esqueleto del modelo principal al que va asociada la malla.
- idSubMesh: Es el identificador de la submalla. **Debe ser único**.
- Default: Este atributo indica si la submalla ha de ser cargada inicialmente o no.
- Text: Es el texto que se mostrará en el tooltip de esta submalla.

A continuación se incorpora el código del fichero XSD que define la estructura que deben seguir los ficheros que se creen para los nuevos modelos. Este código esta comentado para facilitar su compresión a los desarrolladores.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified" attributeFormDefault="unqualified">
	<xs:element name="model">
		<xs:complexType>
			<xs:sequence>
				<!-- This node defines all the textures, physical
				builds,
				initial transformations and all the bones to scale.
				-->
				<xs:element name="mainMesh" type="mainMeshType"/>
				<!-- This node defines all the submeshes of the
				model -->
				<xs:element name="subMesh" type="subMeshType"
				minOccurs="0"
				maxOccurs="unbounded"/>
				<!-- This node define the languages path of the
				model -->
				<xs:element name="languagesPath" type="xs:string"/>
			</xs:sequence>
		</xs:complexType>
	</xs:element>

	<xs:complexType name="meshType">
		<xs:sequence>
			<!-- This node define the mesh path -->
			<xs:element name="path" type="xs:string"/>
```

```
                <!-- This node define the list of initial transformations
                to make to the model -->
                <xs:element name="transformation" type="transformationType"
                minOccurs="0" maxOccurs="unbounded"/>
        </xs:sequence>
</xs:complexType>

<xs:complexType name="mainMeshType">
        <xs:complexContent>
                <xs:extension base="meshType">
                        <xs:sequence>
                                <!-- This node define the list of the
                                textures of the model -->
                                <xs:element name="mainMeshTextures"
                                minOccurs="0" maxOccurs="1">
                                        <xs:complexType>
                                                <xs:choice minOccurs="1"
                                                maxOccurs="unbounded">
                                                        <xs:element
                                                        name="baseShadowTexture"
                                                type="baseShadowTextureType"/>
                                                        <xs:element
                                                        name="simpleTexture"
                                                        type="simpleTextureType"/>
                                                        <xs:element
                                                        name="doubleTexture"
                                                        type="doubleTextureType"/>
                                                 <xs:element
                                                        name="multiOptionTexture"
                                                type="multiOptionTextureType"/>
                                                </xs:choice>
                                        </xs:complexType>
                                </xs:element>
                                <!-- This node define the list of the bones
                                to scale -->
                                <xs:element name="bones" type="bonesType"/>
                                <!-- This node define the list of the physical
                                builds of the model -->
                                <xs:element name="physicalBuilds"
                                type="physicalBuildsType"/>
                        </xs:sequence>
                </xs:extension>
        </xs:complexContent>
</xs:complexType>

<xs:complexType name="subMeshType">
        <xs:complexContent>
                <xs:extension base="meshType">
                        <xs:sequence>
                                <!-- This node is similar to mainMeshType but
                                in this case only can have one texture
                                associated.-->
                                <xs:element name="subMeshTexture"
                                minOccurs="0" maxOccurs="1">
                                        <xs:complexType>
                                                <xs:choice>
                                                        <xs:element
                                                        name="baseShadowTextureSubMesh"
                                                type="baseShadowTextureSubMeshType"/>
                                                        <xs:element
                                                        name="simpleTextureSubMesh"
                                                type="simpleTextureSubMeshType"/>
                                                        <xs:element
                                                        name="doubleTextureSubMesh"
                                                type="doubleTextureSubMeshType"/>
                                                        <xs:element
                                                        name="multiOptionTextureSubMesh"
                                                type="multiOptionTextureSubMeshType"/>
```

```
                                        </xs:choice>
                                    </xs:complexType>
                                </xs:element>
                                <xs:element name="iconPath"
                                type="xs:string"/>
                        </xs:sequence>
                        <!-- This attribute indicates the panel where it has
                        to load this texture -->
                        <xs:attribute name="idPanelRef" type="xs:string"
                        use="required"/>
                        <!--This attribute indicates the bone which the
                        submesh is associated.-->
                        <xs:attribute name="associatedBone" type="xs:string"
                        use="required"/>
                        <!-- This attribute indicates the id of this submesh
                        must be unique -->
                        <xs:attribute name="idSubMesh" type="xs:ID"
                        use="required"/>
                        <!-- If default is true, this submesh will appear
                        when the model is loaded initially.-->
                        <xs:attribute name="default" type="xs:boolean"
                        use="required"/>
                        <xs:attribute name="text" type="xs:string"
                        use="required"/>
                </xs:extension>
        </xs:complexContent>
</xs:complexType>

<!-- This node is the generic type of a texture. All the textures have
these fields. -->
<xs:complexType name="textureType">
        <xs:sequence>
                <xs:element name="iconPath" type="xs:string"/>
        </xs:sequence>
        <!-- This attribute indicates the panel where it has to load
        this texture -->
        <xs:attribute name="idPanelRef" type="xs:string" use="required"/>
        <!-- This attribute indicates the priority of this texture.
        Lower value indicates higher priority.(Highest priority is
        layer=0) -->
        <xs:attribute name="layer" type="xs:integer" use="required"/>
        <!-- This attribute indicates the id of this texture. Must be
        unique! -->
        <xs:attribute name="idTexture" type="xs:ID" use="required"/>
        <xs:attribute name="text" type="xs:string" use="required"/>
</xs:complexType>

<xs:complexType name="baseShadowTextureType">
        <xs:complexContent>
                <xs:extension base="textureType">
                        <xs:sequence>
                                <!-- base path.-->
                                <xs:element name="path" type="xs:string"/>
                                <!-- RGB value which is initially loaded in
                                the texture.-->
                                <xs:element name="colorDefault"
                                type="colorType"/>
                                <xs:element name="shadowPath"
                                type="xs:string"
                                minOccurs="0" maxOccurs="1"/>
                        </xs:sequence>
                        <!-- If default is true, this texture will appear
                        when the model is loaded initially.-->
                        <xs:attribute name="default" type="xs:boolean"
                        use="required"/>
                </xs:extension>
        </xs:complexContent>
</xs:complexType>
```

```
<xs:complexType name="colorType">
	<xs:sequence>
		<xs:element name="red" type="xs:integer"/>
		<xs:element name="green" type="xs:integer"/>
		<xs:element name="blue" type="xs:integer"/>
	</xs:sequence>
</xs:complexType>

<xs:complexType name="simpleTextureType">
	<xs:complexContent>
		<xs:extension base="textureType">
			<xs:sequence>
				<xs:element name="path" type="xs:string"/>
			</xs:sequence>
			<xs:attribute name="default" type="xs:boolean"
			use="required"/>
		</xs:extension>
	</xs:complexContent>
</xs:complexType>

<xs:complexType name="doubleTextureType">
	<xs:complexContent>
		<xs:extension base="textureType">
			<xs:sequence>
				<xs:element name="basePath"
				type="xs:string"/>
				<xs:element name="detailsPath"
				type="xs:string"/>
			</xs:sequence>
			<xs:attribute name="default" type="xs:boolean"
			use="required"/>
		</xs:extension>
	</xs:complexContent>
</xs:complexType>

<xs:complexType name="multiOptionTextureType">
	<xs:complexContent>
		<xs:extension base="textureType">
			<xs:sequence>
				<xs:element name="optionTexture"
				minOccurs="1"
				maxOccurs="unbounded">
					<xs:complexType>
						<xs:sequence>
							<xs:element name="path"
							type="xs:string"/>
						<xs:element name="iconPath"
							type="xs:string"/>
						</xs:sequence>
						<!-- This attribute indicates
						the id of
						this subtexture. Must be unique!
						-->
						<xs:attribute
						name="idSubTexture"
						type="xs:ID" use="required"/>
						<!-- If default is true, this
						subtexture will appear when the
						model is loaded initially.-->
						<xs:attribute name="default"
						type="xs:boolean"
						use="required"/>
						<xs:attribute name="text"
						type="xs:string"
						use="required"/>
					</xs:complexType>
				</xs:element>
```

```
                </xs:sequence>
            </xs:extension>
        </xs:complexContent>
</xs:complexType>

<!-- Textures SubMeshes -->
<xs:complexType name="textureSubMeshType">
        <!-- This attribute indicates the priority of this texture.
                Lower value indicates higher priority.(Highest priority is
                layer=0) -->
        <xs:attribute name="layer" type="xs:integer" use="required"/>
        <xs:attribute name="idTexture" type="xs:ID" use="required"/>
</xs:complexType>

<xs:complexType name="baseShadowTextureSubMeshType">
        <xs:complexContent>
                <xs:extension base="textureSubMeshType">
                        <xs:sequence>
                                <!--Base path.-->
                                <xs:element name="path" type="xs:string"/>
                                <xs:element name="shadowPath"
                                type="xs:string"
                                minOccurs="0" maxOccurs="1"/>
                        </xs:sequence>
                </xs:extension>
        </xs:complexContent>
</xs:complexType>

<xs:complexType name="simpleTextureSubMeshType">
        <xs:complexContent>
                <xs:extension base="textureSubMeshType">
                        <xs:sequence>
                                <xs:element name="path" type="xs:string"/>
                        </xs:sequence>
                </xs:extension>
        </xs:complexContent>
</xs:complexType>

<xs:complexType name="doubleTextureSubMeshType">
        <xs:complexContent>
                <xs:extension base="textureSubMeshType">
                        <xs:sequence>
                                <xs:element name="basePath"
                                type="xs:string"/>
                                <xs:element name="detailsPath"
                                type="xs:string"/>
                        </xs:sequence>
                </xs:extension>
        </xs:complexContent>
</xs:complexType>

<xs:complexType name="multiOptionTextureSubMeshType">
        <xs:complexContent>
                <xs:extension base="textureSubMeshType">
                        <xs:sequence>
                                <xs:element name="optionTexture"
                                minOccurs="1"
                                maxOccurs="unbounded">
                                        <xs:complexType>
                                                <xs:sequence>
                                                        <xs:element name="path"
                                                        type="xs:string"/>
                                                        <xs:element
                                                        name="iconPath"
                                                        type="xs:string"/>
                                                </xs:sequence>
                                                <!-- This attribute indicates
                                                the id
```

```
                                        of this subtexture. Must be
                                        unique! -->
                                        <xs:attribute
                                        name="idSubTexture"
                                        type="xs:ID" use="required"/>
                                        <!-- If default is true,
                                         this subtexture will appear when
                                         the model is loaded initially.--
                                         >
                                        <xs:attribute name="default"
                                        type="xs:boolean"
                                        use="required"/>
                                        <xs:attribute name="text"
                                        type="xs:string"use="required"/>
                                </xs:complexType>
                        </xs:element>
                </xs:sequence>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>

<xs:complexType name="bonesType">
    <xs:sequence>
        <xs:element name="bone" type="boneType" minOccurs="1"
        maxOccurs="unbounded"/>
    </xs:sequence>
</xs:complexType>

<xs:complexType name="boneType">
    <!--This attribute indicates the controller defined in XML family
    with which it is associated.-->
    <xs:attribute name="idControllerRef" type="xs:string"
    use="required"/>
    <xs:attribute name="defaultValue" type="xs:float"
    use="required"/>
    <xs:attribute name="minValue" type="xs:float" use="required"/>
    <xs:attribute name="maxValue" type="xs:float" use="required"/>
    <xs:attribute name="boneName" type="xs:string" use="required"/>
</xs:complexType>

<xs:complexType name="physicalBuildsType">
    <xs:sequence>
        <xs:element name="physicalBuild" type="physicalBuildType"
        minOccurs="1" maxOccurs="10"/>
    </xs:sequence>
</xs:complexType>

<xs:complexType name="physicalBuildType">
    <!--This attributes indicates the escalations sequence that will
    be applied to model.-->
    <xs:sequence>
        <xs:element name="escalation" type="escalationType"
        minOccurs="1" maxOccurs="unbounded"/>
    </xs:sequence>
    <!-- This attribute indicates the panel where it has to load this
    texture -->
    <xs:attribute name="idPanelRef" type="xs:string" use="required"/>
    <xs:attribute name="label" type="xs:string" use="required"/>
    <!-- This attribute indicates the id of this physical build. Must
    be unique! -->
    <xs:attribute name="idPhysicalBuild" type="xs:ID"
    use="required"/>
</xs:complexType>

<xs:complexType name="escalationType">
    <xs:attribute name="boneName" type="xs:string" use="required"/>
    <!-- These fields define the values of the transformations on
    each axis. -->
    <xs:attribute name="valueX" type="xs:float" use="required"/>
```

```
            <xs:attribute name="valueY" type="xs:float" use="required"/>
            <xs:attribute name="valueZ" type="xs:float" use="required"/>
      </xs:complexType>

      <xs:complexType name="transformationType">
            <xs:attribute name="transformationType" use="required">
                  <!-- This field defines the type of the transformation
                  (scale, rotate or translate). -->
                  <xs:simpleType>
                        <xs:restriction base="xs:string">
                              <xs:enumeration value="scale"/>
                              <xs:enumeration value="rotate"/>
                              <xs:enumeration value="translate"/>
                        </xs:restriction>
                  </xs:simpleType>
            </xs:attribute>
            <!-- These fields define the values of the transformations on
            each axis. -->
            <xs:attribute name="valueX" type="xs:float" use="required"/>
            <xs:attribute name="valueY" type="xs:float" use="required"/>
            <xs:attribute name="valueZ" type="xs:float" use="required"/>

      </xs:complexType>
</xs:schema>
```

Un último dato que hay que tener en cuenta a la hora de crear las nuevas familias y modelos, es que se debe hacer respetando el sistema de internacionalización que se ha creado para dotar de cierta flexibilidad a la interfaz (comentado en el apartado 4.6 del capítulo *Funcionalidades*). Para respetar este sistema, todos los campos en los que se requieran cadenas de texto que se muestran en la interfaz, deben aparecer como identificadores. Posteriormente y en la carpeta que se haya definido en el nodo metadatos de la familia, deberá aparecer un fichero por cada idioma que se quiera ofrecer. Estos ficheros tendrán una lista de pares de la forma <clave, valor> donde la clave es el identificador que se le ha asignado a la cadena de texto y el valor es la cadena de texto en el idioma correspondiente, listo para que se muestre en la interfaz.

Si el usuario ha configurado la aplicación para usarla en un idioma que por alguna razón no se encuentre disponible para la familia, la aplicación va a intentar ofrecer algún idioma similar. Si no consiguiera encontrar algún idioma similar al configurado, los textos asociados a la familia en cuestión se van a mostrar en alguno de los idiomas que disponga.

10. Método de trabajo

Este capítulo incluye información detallada acerca del proceso del que se ha seguido para el desarrollo de la aplicación. En un primer apartado se enumeran las herramientas que se han utilizado para el desarrollo del código y la comunicación entre todos los miembros del proyecto. En el segundo apartado de este capítulo se comenta la planificación seguida para cumplir los objetivos establecido de cara a la entrega final. Además este capítulo incluye un apartado con actas de todas las reuniones que se han mantenido a lo largo del curso entre alumnos y tutores del proyecto.

10.1. Herramientas utilizadas

Este proyecto se ha desarrollado apoyándose en diferentes aplicaciones en lo referente al mantenimiento del código, elaboración de documentos y comunicación entre los integrantes del grupo. Las aplicaciones que se han utilizado se describen a continuación.

- **Dropbox**

Dropbox es un servicio de alojamiento de todo tipo de archivos en línea. Dropbox permite sincronizar los archivos en línea en cualquier ordenador y compartirlos con otros usuarios. Se puede acceder a este servicio mediante la web www.dropbox.com o mediante las aplicaciones de PC, Android, iOS o Blackberry.

Para el proyecto se ha utilizado una versión gratuita de Dropbox. Inicialmente se compartían tutoriales y manuales de las herramientas que han sido utilizados durante el desarrollo del proyecto. También se ha utilizado para compartir diversos recursos, como documentos de la memoria, diseños de la interfaz, y sobre todo diferentes versiones de los distintos componentes de la interfaz. Las primeras versiones del código del proyecto también fueron compartidas mediante este sistema, pero debido a la complejidad para controlar el estado de las versiones se decidió usar un repositorio Git.

- **Google Drive**

Google Drive, al igual que Dropbox, es un servicio de almacenamiento de archivos en línea. Este servicio concede a cada usuario 5 Gigabytes de almacenamiento ampliable mediante pago. Dispone de aplicaciones en Android e iOS y se accede mediante la web www.google.com/drive. Este servicio se caracteriza por la creación de documentos en línea y la posibilidad de colaborar en grupo.

En el proyecto se ha utilizado para compartir las actas de las reuniones que se han tenido durante el curso. La elaboración de la memoria final también se ha llevado a cabo con este servicio, ya que la posibilidad de la colaboración en grupo es mucho más sencilla que en otros servicios, como por ejemplo Dropbox.

- **Gmail**

Gmail es un servicio de correo electrónico. Este servicio proporciona al usuario 10 Gigabytes de almacenamiento para gestionar su correo electrónico. Este servicio también incluye Hangouts, que es un servicio de mensajería instantánea por el que

cada usuario se puede comunicar con sus contactos. Gmail tiene aplicación para Android e iOS y se accede mediante la web www.gmail.com.

En el proyecto ha sido el pilar fundamental de la comunicación de los integrantes con los tutores. Fundamentalmente se ha utilizado el servicio Google Groups para esta comunicación.

- **Google Groups**

Google Groups es un servicio de correo que proporciona Google para la comunicación de varios integrantes dentro de un mismo grupo. Se elaboró un correo electrónico de este tipo y se agregaron a todos los componentes del proyecto y a los dos tutores, para que toda la comunicación que se realizase mediante este servicio se compartiera con todos los integrantes.

- **Trello**

Trello es una web que facilita la gestión de proyectos. Trello utiliza un sistema de gestión denominado Kanban. Los proyectos se representan por tablas, que contienen listas, y cada lista contiene tarjetas, que son las tareas. Cada tarjeta debe pasar de una lista a otra, desde el inicio de la idea, hasta su implementación. También se pueden asignar tarjetas a usuarios.

Se empezó a utilizar en una fase bastante avanzada del proyecto sustituyendo a los documentos de actas almacenados en Google Drive. Esta fase final del desarrollo de la aplicación se dividió en cuatro hitos, y en cada uno de ellos deberían alcanzarse una serie de objetivos explicados con más detalle en el apartado 10.3 *Actas e Hitos* dentro de este capítulo.

- **GitHub**

GitHub es una plataforma de desarrollo colaborativo de software, que aloja proyectos utilizando un repositorio Git. Un repositorio Git es un software de control de versiones diseñado por Linus Torvalds. Sus características más significativas son el fuerte apoyo al desarrollo no lineal, basándose en la gestión de ramas y su posterior mezclado, y la gestión distribuida, proporcionando a cada usuario una copia local del proyecto.

Este servicio almacena el código de forma pública, aunque se puede hacer de forma privada en la versión de pago. También contiene diversas características que ayudan al proyecto almacenado como una wiki, una web para cada proyecto, y gráficos para ver cómo trabajan los desarrolladores en sus repositorios.

En el proyecto ha sido fundamental su uso, ya que ha alojado todas las versiones que se han ido desarrollando de la aplicación.

- **TortoiseGit**

Es un cliente Git de control de revisiones. Es software libre, publicado bajo la licencia pública general GNU.

El uso de esta aplicación ha sido diario, ya que era la conexión que teníamos entre la versión de la aplicación en GitHub con la aplicación alojada en los discos duros de los componentes del grupo.

10.2. Planificación del proyecto

Para poder llevar a cabo la realización del proyecto, se decidió utilizar el modelo en espiral. El modelo en espiral, propuesto originalmente por Boehm, es un modelo de proceso de software evolutivo que conjuga la naturaleza iterativa de construcción de prototipos con los aspectos controlados y sistemáticos del modelo lineal secuencial. Proporciona el potencial para el desarrollo rápido de versiones incrementales del software. En el modelo espiral, el software se desarrolla en una serie de versiones incrementales. Durante las primeras iteraciones, la versión incremental podría ser un modelo en papel o un prototipo. Durante las últimas iteraciones, se producen versiones cada vez más completas del sistema diseñado.

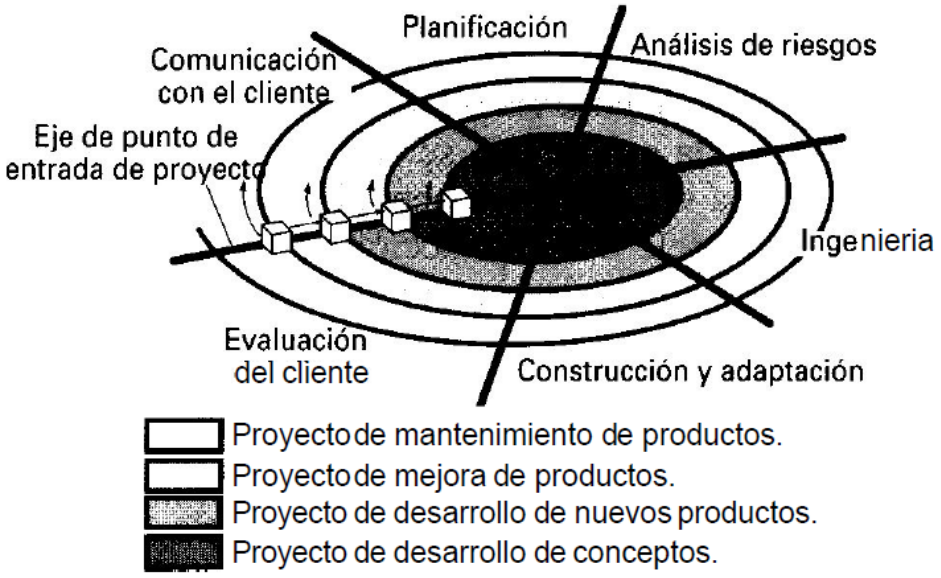

Figura 10.1: Diagrama de planificación en espiral. Obtenida de: (Pressman, 1997).

1. Primera iteración (8/10/2012 - 31/10/2012)

En esta primera iteración se lleva a cabo la toma de contacto con el entorno de desarrollo jMonkey. La investigación inicial estuvo compuesta por la implementación de 10 tutoriales básicos que nos ayudaron a comprender su funcionamiento.

También se buscan modelos 3D en internet que ayuden en la implementación de la futura aplicación.

2. Segunda iteración (1/11/2012 - 19/12/2012)

Se desarrolla el primer prototipo de la aplicación final, con una interfaz muy poco vistosa y un único modelo al que se le puede cambiar el color de piel, la camiseta, el pantalón y los zapatos. El modelo carece de submallas, como por ejemplo el pelo.

3. Tercera iteración (20/12/2012 - 23/01/2013)

Se implementa una nueva interfaz más elaborada que la anterior. También se implementa un sistema de escalado en el modelo, en el que se pueden seleccionar complexiones básicas o escalado avanzado. En el escalado avanzado se diferencian huesos como los brazos, el torso, las piernas y los pies.

También se diseña el cambio de color de las texturas y se aplica a las texturas de camisetas, pantalones y zapatos dotándolos de una mayor personalización para el usuario.

Se toma la decisión de implementar un modelo subyacente explicado en el apartado de diseño de la aplicación, para que la inclusión de nuevos modelos y familias, sea más sencillo. También se incluye un sistema de configuración de la interfaz en este modelo subyacente.

4. **Cuarta iteración (24/01/2013 - 13/02/2013)**

Se introduce un nuevo modelo en la aplicación y se adapta para que siga funcionando. Se dota al nuevo modelo de las mismas características que poseía el modelo ya residente en la aplicación.

También se incorporan a la aplicación las submallas. Se implementa una nueva etapa en la interfaz donde se puede cambiar el estilo de peinado a los modelos de la aplicación.

5. **Quinta iteración (14/02/2013 - 4/04/2013)**

Se amplía la aplicación con una familia completa de modelos. Para esta ampliación se adapta la interfaz con una nueva pantalla de selección de familias.

También se implementa el sistema de exportación de los modelos personalizados en la aplicación, y como consecuencia la adición de una nueva pantalla en la interfaz que recoja esta nueva funcionalidad.

6. **Sexta iteración (5/04/2013 - 5/05/2013)**

En esta iteración se lleva a cabo un rediseño de la interfaz para hacerla más intuitiva y minimalista. También se introducen efectos en los botones y en el menú dotando a la interfaz más elegancia y preparándola para su distribución.

También se introduce una segunda familia en la aplicación. El sistema se adapta a esta nueva familia solucionando los conflictos que ocasiona incluirla.

7. **Séptima iteración. (6/05/2013 - 12/06/2013**

En esta última iteración se dan los últimos detalles en la implementación de la aplicación para su distribución.

También se lleva a cabo la escritura de la memoria final, adaptando documentos ya existentes producidos durante el desarrollo de la aplicación.

10.3. Actas e Hitos

Para ilustrar mejor la planificación seguida durante todo el año se decidió realizar actas de todas las reuniones mantenidas entre los alumnos y los tutores del proyecto. Esta sección recoge los puntos principales tratados y objetivos marcados en cada reunión.

10.3.1. Actas

1. Acta 24/10/2012

Se trató la introducción de texturas para los modelos de jMonkey y buscar en la red nuevos modelos 3D con distintos formatos, indicando el origen y la licencia que poseen.

Se inició la escritura de documentos básicos que posteriormente agilizaron la elaboración de la memoria del proyecto.

2. Acta 13/11/2012

Se habló sobre la importancia del escalado de los modelos, tanto generales, para implementar complexiones básicas, como delgado o corpulento y complexiones más avanzadas como alargar uno de los brazos.

En esta reunión también se comentó el proceso de los personajes diseñados. Para ello es necesario capturar los frames a una determinada frecuencia de muestreo que lo impone la calidad de la exportación.

Por último se comentó la posibilidad de añadir mallas complementarias al modelo, como puede ser el pelo.

3. Acta 21/11/2012

Se comentó el cambio de los colores en las texturas, para dar más opciones de configuración a los usuarios.

Se siguió comentando la captura de frames necesaria para la exportación y cómo implementarla.

Se presentó un pequeño documento de casos de uso de la aplicación.

Por último se propuso el inicio de uso de un repositorio donde almacenar la aplicación. El repositorio elegido fue GitHub.

4. Acta 28/11/2012

Se profundizó en las transformaciones de las texturas como el cambio de color o la introducción de detalles como las sombras.

También se habló sobre la importancia de optimizar el código y calcular el coste que suponían los procedimientos más lentos.

5. Acta 12/12/2012

Se trató el estilo del código y poner cabeceras de la licencia en las clases. También se habló del correcto uso del repositorio GitHub.

6. Acta 19/12/2012

Se inició la adaptación de la aplicación a un sistema que fuese flexible a la hora de introducir más recursos, como más texturas para los modelos ya existentes. Para esta adaptación se utilizan ficheros XML.

También se buscaron recursos con licencia *Open Source* en internet para realizar un prototipo de la interfaz gráfica.

7. Acta 09/01/2013

Se comenzó a diseñar una interfaz intuitiva y simple para el usuario y a tomar decisiones en la disposición de los paneles de opciones, botones y menús.

Se siguió dando pinceladas al nuevo sistema de introducción de los datos con XML.

8. Acta 23/01/2013

Se habló sobre las vistas de la cámara para la exportación de los frames.

En la interfaz se propuso la adición de una etapa de exportación que recogiese diferentes opciones para que el usuario pudiese decidir el formato en el que desea obtener su personaje. También se comentó la adición de una nueva pantalla para seleccionar las familias de modelos, ya que en un futuro podrían introducirse nuevos modelos en la aplicación.

9. Acta 06/02/2013

Se siguió elaborando las nuevas pantallas de la interfaz y tomando las decisiones de la disposición de los componentes.

A partir de esta última acta se comenzó a utilizar la aplicación Trello, y los objetivos se introdujeron como tarjetas en la aplicación.

10.3.2. Hitos

Como se puede observar las actas llegan hasta primeros de Febrero. En este punto del proyecto se decidió usar el sistema de gestión de proyectos Trello, para dotar a los objetivos de una mayor claridad a la hora de marcar los que están en desarrollo o los que están cumplidos. A continuación se recoge la serie de hitos marcados.

1. Aplicación funcional con una familia

Dentro de este hito se ha implementado un sistema de internacionalización para la carga de varios idiomas. Se desarrolló una interfaz para que fuese sencillo, la introducción de nuevos idiomas en la aplicación.

2. Rediseño de la interfaz y carga de una segunda familia

Se introdujo una nueva familia en la aplicación y se adaptó el sistema con este nuevo cambio. También se llevó a cabo el rediseño de la interfaz gráfica para que la experiencia de usuario sea más agradable.

3. Distribución

Se finalizó el rediseño de la interfaz introduciendo nuevas mejoras y se estudió la forma de distribuir la aplicación para las plataformas Windows, Linux y Mac OSX.

4. Integración con eAdventure

Se integró la aplicación desarrollada durante el presente curso en la plataforma eAdventure, dotando a ésta última de un potente configurador de personajes para los juegos existentes de la plataforma.

11. Contribuciones, conclusiones y trabajo futuro

11.1. Contribuciones

El proyecto *eCharacter* supone una aportación al mundo de desarrollo de videojuegos de código libre (open source). En la actualidad existe un gran número de herramientas libres (o de código abierto) tanto para el desarrollo de videojuegos como herramientas para el modelado y diseño de contenido artístico para videojuegos. Sin embargo, sigue existiendo un claro déficit de recursos artísticos libres (sin derechos de propiedad intelectual), especialmente de ciertos tipos, como pueden ser personajes animados. *eCharacter* puede llenar ese hueco, pues fue ideada para tener un largo recorrido como proyecto libre y que trascienda a la duración del proyecto de Sistemas Informáticos. Esta idea se ha plasmado en las siguientes características de la aplicación *eCharacter:*

- **Proyecto abierto y libre**. Su licencia es LGPL, su código está disponible en GitHub y los modelos que contiene son de uso libre auspiciados por la iniciativa Creative Commons, por lo que podrán beneficiarse de esta aplicación un gran número de usuarios. Asimismo puede atraer a otros desarrolladores que quieran contribuir al proyecto.

- **Facilidad de integración**. *eCharacter* incorpora un sistema para facilitar la integración con otras herramientas de creación de juegos para que los personajes creados puedan utilizarse en el mayor número de plataformas posible. Esto es en parte gracias a un sistema que permite al usuario exportar el modelo configurado de manera muy versátil.

- **Extensibilidad**. *eCharacter* incorpora un mecanismo que permite añadir nuevos modelos y texturas a la aplicación de forma sencilla. Esto puede convertir a *eCharacter* en un espacio atractivo para artistas 3D que puedan estar interesados en contribuir con nuevos modelos como forma de dar a conocer su trabajo, y permite que el proyecto pueda seguir creciendo en el futuro incluso sin invertir en nuevas funcionalidades.

  Esto es posible gracias al esquema de representación subyacente para los modelos gráficos. Se ha desarrollado un sistema para que de una manera sencilla, mediante el uso de ficheros XML, el usuario sea capaz de insertar sus propios modelos y sus propias texturas, así como las etapas del configurador. Mediante este proceso de configuración XML se puede no sólo incluir nuevos modelos sino también cambiar el comportamiento y apariencia del configurador sin tener que desarrollar nuevo código.

- **Internacionalización**. También se ha hecho hincapié en la internacionalización, desarrollando un sistema que permite que el producto se abra al mercado internacional dándole un plus de calidad. Se ha diseñado un sistema de internacionalización lo más flexible posible para que de una manera sencilla, el usuario pueda agregar más idiomas al interfaz de la aplicación.

Actualmente *eCharacter* se ofrece en 3 idiomas diferentes (español, inglés y francés).

- **Multiplataforma.** *eCharacter* se ha desarrollado teniendo en cuenta las actuales tendencias en la producción de software. Por ello, el sistema se ha diseñado de modo multiplataforma, pudiendo ser ejecutado en Windows, Linux, Mac OS X y para cualquier sistema que pueda ejecutar Java y OpenGL.

11.2. Conclusiones

Durante el proceso de desarrollo de esta aplicación nos hemos enfrentado a situaciones que no se nos habían presentado antes. Gracias a ellas hemos podido aprender nuevas técnicas de desarrollo, pero que después de haberlas aplicado creemos son importantes y necesarias a la hora de realizar un buen proyecto y conseguir lanzar un producto al mercado que sea usable, funcional, y con una estética actual.

Debido a una planificación demasiado optimista, se han retrasado algunas actividades que no han podido incluirse en esta memoria. Una de ellas es la recopilación y análisis de los resultados de las pruebas de usuario planteadas. Otros de los puntos que se han quedado pendiente es la posible integración con otros sistemas que utilicen avatares personalizados, ya que actualmente sólo se ha integrado en eAdventure.

El desarrollo de este proyecto se ha realizado con la idea de que el proyecto pueda tener un largo recorrido en el mundo del código libre, pero sí es cierto, que no se le ha podido dar la difusión deseada para poder contrastar este pensamiento inicial con la opinión de otros usuarios expertos en la materia

11.3. Trabajo futuro

eCharacter se ha desarrollado pensando en que la aplicación pueda crecer y mejorarse introduciendo mejoras y nuevas funcionalidades. Es una aplicación escalable y se ha desarrollado un plan de futuro para dotar de nueva funcionalidad a la aplicación. Estas nuevas funcionalidades son:

- Repositorio online de familias: Esta funcionalidad consiste en crear un repositorio online, en el que estén alojadas todas las familias disponibles para *eCharacter*. De esta manera el usuario podrá incorporar fácilmente nuevas familias a su aplicación y el proceso sería invisible de cara al usuario.
- Modos adicionales de exportación: La idea es que se permita poder realizar una exportación del modelo ya configurado en formatos asociados con programas de diseño y modelado 3D, como puede ser Collada (.dae), Ogre3D (.mesh.xml) y Maya (.mb).
- Nuevas familias. Incluir un mayor número de familias de modelos en el núcleo inicial de *eCharacter*: De esta forma se dota a la aplicación de una mayor riqueza de recursos y se le ofrecen al usuario un mayor número de posibilidades a la hora de configurar su modelo.
- Aplicación Android: Con la idea en mente de seguir las tendencias actuales de desarrollo de software, se añadirá soporte en la plataforma Android para que

de esta manera *eCharacter* pueda cubrir también el mercado de smartphones y tablets.

- Herramienta de creación del esquema XML subyacente: Como ya se ha comentado una de las principales funcionalidades que ofrece *eCharacter* es permitir al usuario la creación e incorporación de sus propias familias de modelos. Para llevar a cabo este proceso el usuario deberá crear una serie de ficheros XML, que puede que requieran un cierto conocimiento avanzado. Para facilitar esta tarea a usuarios menos avanzados se pretende desarrollar una herramienta, similar a un editor, que facilite este proceso de creación.

Al ser un proyecto de código libre y abierto se va a promocionar entre los interesados en el desarrollo de videojuegos para que parte de estos desarrollos se puedan realizar de forma colaborativa con la comunidad de desarrollo ya existente. Si *eCharacter* consigue una cierta repercusión y se logra que parte de la comunidad se implique en su desarrollo y en su diseño y mejora de funcionalidades puede evolucionar de una forma muy rápida.

12. Referencias

ADeSe. (n.d.). Asociación Española de Distribuidores y Editores de Software de Entretenimiento. *El Anuario del Videojuego*. Retrieved from http://www.adese.es/docs/documentacion/el-anuario-del-videojuego

ADeSe. (2012). *Anuario de la Industria del Videojuego* (p. 98). Retrieved from http://www.adese.es/anuario2012/ANUARIO_ADESE_2012.pdf

Brooke, J. (1996). SUS - A quick and dirty usability scale. *Usability evaluation in industry* (pp. 1–8). Retrieved from B5_During_the_trial_usability_scale_V1_09Aug11.pdf

ESA, E. S. A. (2012). *Essential facts about the computer and video game industry* (p. 13).

F A S. (2006). *Summit on Educational Games: Harnessing the power of video games for learning* (p. 53).

Hwang, G.-J., & Wu, P.-H. (2012). Advancements and trends in digital game-based learning research: a review of publications in selected journals from 2001 to 2010. *British Journal of Educational Technology*, *43*(1), E6–E10. doi:10.1111/j.1467-8535.2011.01242.x

Johnson, L., Adams Becker, S., Cummins, M., Estrada, V., Freeman, A., & Ludgate, H. (2013). *NMC Horizon Report: 2013 Higher Education Edition* (p. 44). Austin, Texas, USA.

Mayo, M. (2007). Games for science and engineering education. *Communications of the ACM*, *50*(7), 30–35. Retrieved from citeulike-article-id:1450072

Moreno-Ger, P., Burgos, D., Sierra, J. L., & Fernández-Manjón, B. (2008). Educational Game Design for Online Education. *Computers in Human Behavior*, *24*(6), 2530–2540. Retrieved from http://dx.doi.org/10.1016/j.chb.2008.03.012

Overmars, M. (2004, April). Teaching Computer Science through Game Design. *Computer*, *37*(4), 81–83. doi:10.1109/MC.2004.1297314

Pressman, R. S. (1997). *Software Engineering: A Practitioner's Approach*. New York: McGraw Hill.

Resnick, M., Maloney, J., Monroy-Hernández, A., Rusk, N., Eastmond, E., Brennan, K., Millner, A., et al. (2009). Scratch : Programming for all. *Communications of the ACM*, *52*(11), 60–67. doi:10.1145/1592761.1592779

Rogers, E. (2012). The Rise of Costs, the Fall of Gaming. *notenoughshaders.com*. Retrieved June 20, 2013, from http://www.notenoughshaders.com/2012/07/02/the-rise-of-costs-the-fall-of-gaming/

Sharkey, M. (2010). Report: Game Development Costs Have Skyrocketed. *gamespy.com*. Retrieved June 20, 2013, from http://uk.gamespy.com/articles/108/1082176p1.html

Torrente, J., Moreno-Ger, P., Fernández-Manjón, B., & Sierra, J. L. (2008). Instructor-oriented Authoring Tools for Educational Videogames (pp. 516–518). Santander, Spain: IEEE Computer Society.

13. Bibliografía

Este capitulo recoge todo la bibliografía que se ha consultado recurremente durante el desarrollo del proyecto.

- jMonkeyEngine
  - Tutoriales y documentación de la página oficial: http://hub.jmonkeyengine.org/wiki/doku.php/jme3
  - Repositorio de Google Code: https://code.google.com/p/jmonkeyengine/
  - JavaDoc de jMonkeyEngine: http://hub.jmonkeyengine.org/javadoc/
  - LWJGL Wiki: http://www.lwjgl.org/wiki/index.php?title=Main_Page

- Nifty GUI
  - Apartado en la documentación de jMonkeyEngine: http://hub.jmonkeyengine.org/wiki/doku.php/jme3:advanced:nifty_gui
  - Wiki de Ninty en Source Forge: http://sourceforge.net/apps/mediawiki/nifty-gui/index.php

- XML Schema
  - Documentación y tutoriales: http://www.w3schools.com/schema/default.asp

- JAXB
  - Tutoriales de la página oficial de JAXB: https://jaxb.java.net/tutorial/
  - Documentación de Oracle sobre JAXB: http://docs.oracle.com/javase/tutorial/jaxb/intro/index.html

- Java
  - API de Java en Oracle: http://docs.oracle.com/javase/7/docs/api/

- Blender
  - Wiki de Blender: http://wiki.blender.org/

- IZPACK
  - Wiki IZPACK http://docs.codehaus.org/display/IZPACK/User+documentation

- GIT
  - Documentación de la página oficial de GIT http://git-scm.com/docs