Escuela Técnica Superior
de Ingeniería Informática

Universidad de La Laguna

Memoria del Proyecto Fin de Carrera
de la Ingeniería Superior en Informática

PORTAD Y KEVAH:

INTERFAZ WEB Y PERSONAJE 3D PARA LA EVALUACIÓN DE TESTS ADAPTATIVOS EN XML

Autores:

Zebenzui Pérez Ramos
Francisco Javier Medina Santos
Daniel Jacobo Díaz González

Director:

Carina Soledad González González

Septiembre de 2004

Carina Soledad González González , profesora asociada del Departamento de Física Fundamental y Experimental, Electrónica y Sistemas de la Universidad de La Laguna, CERTIFICA:

Que el proyecto de Fin de Carrera de título: **'PORTAD Y HEVAH: Interfaz Web y Personaje 3D para la Evaluación de Tests Adaptativos en XML'** presentado por D. Zebenzui Pérez Ramos, D. Francisco Javier Medina Santos y D. Daniel Jacobo Díaz González ha sido realizado bajo su dirección.

Con esta fecha autoriza la presentación de la misma.

La Laguna, a 17 de Septiembre de 2004.

Agradecimientos

Ante todo, agradecer a Carina la oportunidad que nos ha brindado de realizar este proyecto, así como la confianza depositada a lo largo del desarrollo del mismo. Gracias por la libertad que nos has concedido en todo momento y por tu flexibilidad, ya que sin ella hubiera resultado imposible llevar a buen puerto este proyecto, debido a los diversos escollos que hemos tenido que sortear.

De forma menos personal, pero con igual entusiasmo, agradecer a la comunidad de Blender su dedicación y espíritu colaborativo. Gracias por ayudarnos a enfrentarnos a las dificultades intrínsecas del proyecto sin sentirnos aislados, así como por la gran cantidad de información que, de forma altruista, ponen a disposición de todos los usuarios en la red.

Por último, pero no menos importante, otro agradecimiento muy emotivo para nuestras familias y personas allegadas por la comprensión mostrada en este tiempo, especialmente a la madre de Dani por aceptarnos como huéspedes, especialmente en esta última fase del proyecto.

Índice general

Lista de Figuras

Capítulo 1

Introducción

1.1. Justificación del Proyecto

El objetivo inicial del proyecto era la creación de un agente virtual personificado que pueda guiar el proceso de evaluación de conocimientos de un alumno y brindar ayuda adaptada al nivel de cada alumno.

Un agente virtual es un agente 'personificado' (que adopta una representación, con forma humana o no) que 'habita' (no percibe observaciones de un entorno externo, sino que forma parte del mismo entorno, y debe ser capaz de desenvolverse, percibir e interactuar en él) en un entorno 'virtual' (el entorno en el que habita el agente es creado por la computadora (puede ser textual, bidimensional, tridimensional) y puede reproducir o no un entorno real con más o menos fidelidad).

El objetivo del proyecto conllevaba diferentes desafíos, el primero y el más visual , era la creación del personaje animado, el siguiente, crear el sistema de evaluación y el motor de inferencia que tomaría las decisiones para llevar todo este proceso usando técnicas de IA.

Para crear el personaje, debíamos seguir los siguientes pasos:

1. Modelarlo con algún software especializado en modelado 3D.
2. Crear las emociones o posibles reacciones que caracterizarían su conducta.
3. Permitir que este personaje pueda ser controlado y manejado desde el módulo de evaluación.

En este último punto pensamos que una buena idea era crear el personaje utilizando el motor del MS Agent de Microsoft que nos facilitaba enormemente el proceso de integra-

ción en otros módulos de programación y también la reutilización para cualquier otra aplicación.

Para modelar el personaje evaluamos distintos softwares comerciales que poseen un entorno de diseño especializado en la creación de personajes, tales como POSER, pero que no brindan ninguna posibilidad de interacción con la animación creada.

Debido a que nuestra filosofía es la del software libre, hemos intentado respetarla en nuestro proyecto. Por ello buscamos alternativas a este tipo de softwares en código abierto y nos encontramos con BLENDER, que respondía a nuestras necesidades de modelado y que además nos ofrecía la posibilidad de crear las interacciones con el personaje a través de su motor de juegos y su lenguaje de base, PYTHON.

Paralelamente, íbamos trabajando en el problema de creación del sistema de evaluación. En la idea inicial del proyecto pensamos que el sistema no solo evaluaría el conocimiento que el alumno tenía en un momento determinado, sino que podría enseñar y guíar al alumno en el proceso de enseñanza-aprendizaje utilizando el personaje animado como tutor, brindándole ayuda adaptada a su nivel de conocimiento según lo fuera precisando.

Para simplificar el problema decidimos descartar el objetivo de creación de un sistema tutorial y la ayuda adaptada, donde nos encontraríamos con el inconveniente de ir actualizando el conocimiento del alumno a medida que avanza con el sistema tutor y nos centramos en la evaluación de los conocimientos de un alumno sobre un tema en particular en un momento dado. Lo más adecuado para este problema era la creación de un test informatizado, pero para que este test fuera adaptado al usuario, decidimos crear un test adaptativo informatizado.

Nuestra premisa era que los tests que se creaban deberían ser reutilizables, fácilmente modificables por cualquier persona y que pudieran integrarse con el personaje. Por ello, decidimos utilizar XML para la creación de la batería de tests.

En relación a la representación del razonamiento que debía seguir el sistema de evaluación, decidimos utilizar redes bayesianas, ya que esta técnica es de probada eficacia para el caso de tests adaptativos y existe una numerosa bibliografía sobre este tema que así lo demuestra. El lenguaje de IA utilizado para crear el motor de inferencia fue CLIPS, al ser el más flexible y permitirnos la comunicación con otras aplicaciones externas.

En relación a la creación y edición de los tests , con el objetivo de facilitar este proceso a los usuarios finales, hemos desarrollado un entorno web completo de producción. Este portal permite a los profesores y alumnos utilizar los tests de forma local, a través del avatar y remota, a través de este portal. Además, los tests generados pueden utilizarse independientemente del personaje y pueden conectarse a cualquier otra aplicación.

Como resultados de este proyecto tenemos dos herramientas que trabajan de forma integrada o independiente:

- **HEVAH** (Hevah es un EValuador Adaptativo Humanoide) y
- **PORTAD** (PORtal para la creación y edición de Tests ADaptativos).

En esta memoria veremos como hemos diseñado y construido estas herramientas, así como los problemas de diseño y los fundamentos teóricos que nos guiaron en este desarrollo.

Como visión general del mismo, presentamos a continuación un esquema de la arquitecutra que hemos utiizado.

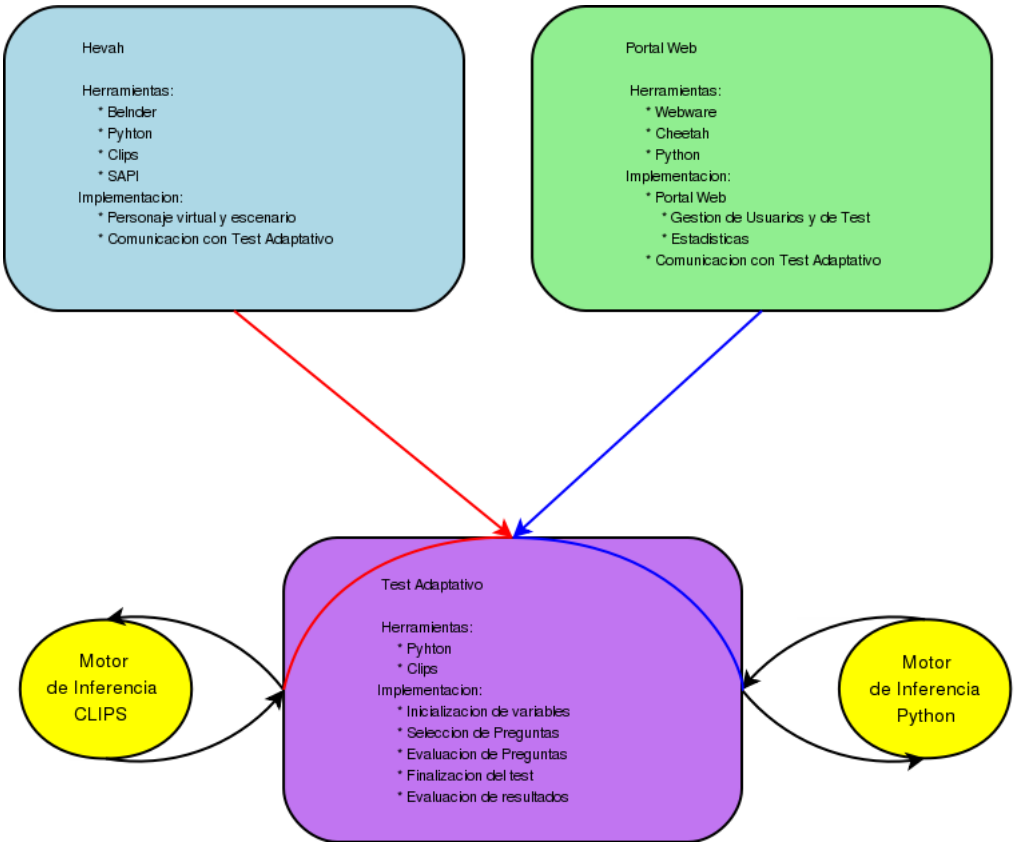


Figura 1.1: Esquema Global de la Arquitectura

1.2. Organización de la memoria

Hemos organizado la memoria de la siguiente manera.

En el capítulo 1 hacemos una breve disertación sobre el Software Libre, sus posibilidades en el mundo de la docencia, y los motivos que nos han llevado a optar por esta vía durante el desarrollo de este proyecto.

En el capítulo 2 introducimos la teoría de tests, deteniéndonos con particular interés en los test adaptativos informatizados y los tests adaptativos informatizados bayesianos.

En el capítulo 3 describimos las principales técnicas de modelado, animación e interacción, así como su aplicación en nuestro proyecto.

En el capítulo 4 hacemos un breve recorrido por las herramientas que utilizamos durante el desarrollo de HEVAH y los motivos que nos llevaron a decidirnos por ellas; en el capítulo 5 hacemos lo propio con el desarrollo de PORTAD.

En el capítulo 6 describimos el proceso lógico en el que se basa nuestro Sistema de Evaluación, explicando el algoritmo que utilizamos y las técnicas que empleamos para implementarlo.

En el capítulo 7 entramos en detalles sobre como implementamos HEVAH, y en el capítulo 8 hacemos lo propio con PORTAD.

Por último, en los apéndices hemos incluídos unos manuales de uso e instalación, así como los códigos fuente.

Capítulo 2

El software libre en la Docencia

En este capítulo intentaremos dar una visión general de lo que es el Software Libre, exponiendo algunos de los argumentos que nos llevaron a decantarnos por este tipo de herramientas a la hora de realizar nuestro proyecto.

2.1. ¿Qué es el software libre?

Se refiere a la libertad de los usuarios para ejecutar, copiar, distribuir, estudiar, cambiar y mejorar el software . De modo más preciso, se refiere a cuatro libertades de los usuarios del software:

- La libertad de usar el programa, con cualquier propósito (libertad 0).
- La libertad de estudiar cómo funciona el programa, y adaptarlo a tus necesidades (libertad 1). El acceso al código fuente es una condición previa para esto.
- La libertad de distribuir copias, con lo que puedes ayudar a tu vecino (libertad 2).
- La libertad de mejorar el programa y hacer públicas las mejoras a los demás, de modo que toda la comunidad se beneficie. (libertad 3). El acceso al código fuente es un requisito previo para esto.

2.2. ¿Qué implicaciones tiene el sofware libre en la docencia?

Uno de los primeros ámbitos de uso del software libre y donde, además, éste resulta inherente, es en el mundo de la docencia y la investigación. Quizá deberíamos destacar

algunos aspectos fundamentales:

- Tanto la investigación como la docencia giran en torno a la transmisión de conocimientos. Es decir, un investigador antes de realizar sus estudios se documenta en el ámbito en el que pretende trabajar apoyándose en artículos y trabajos de interés, con el fin de 'no reinventar la rueda'. Así mismo, un docente realiza en cada una de sus clases una transmisión de conocimientos directa hacia sus alumnos intentando que estos puedan asimilar los conceptos que este pretende enseñar. Justamente esta es una de las ideas más importantes en el software libre: la transmisión de conocimientos con el fin de evolucionar con mayor velocidad. Además se adapta a la perfección a los modelos que acabamos de observar.

- Por otro lado la posibilidad que nos brinda el software libre no únicamente de acceder a sus fuentes (elemento fundamental ya que no se puede enseñar el funcionamiento de un coche si no podemos ver el motor), sino de su libre manipulación, nos permite adaptarlo a nuestras necesidades y en consecuencia crear entornos y herramientas didácticas hechas a medida (que a su vez podrán ser estudiadas).

- Proporciona independencia del fabricante y en muchos casos de la plataforma. Una aplicación de software libre no esta ligada firmemente a su fabricante lo que puede llegar a evitar, entre otras cosas, que quede obsoleta ya que el trabajo de otros usuarios sobre dicha aplicación puede contribuir a mejorarla y mantenerla actualizada aligerando así la carga del creador.

- Con el software libre no son necesarias licencias, facilitando así el trabajo de los alumnos que podrán acceder libremente a las aplicaciones sin necesidad de un desembolso económico importante.

- El ahorro en adquisición de software debido a la utilización de aplicaciones libres puede resultar útil para las comunidades educadoras en otros ámbitos, cubriendo así otras necesidades que resulten relevantes.

2.3. ¿Utilidades para la docencia con Software Libre?

Existen gran cantidad: foros de discusión *Postnuke* o *Squishdot*, servidores de aplicaciones *Zope*, correo electrónico (incluido webmail), documentación en línea (uso de Wikis) y en conjunto opciones libres integradas como *Claroline*. Esto son, evidentemente, unos pocos de los muchísimos ejemplos que podríamos citar.

2.4. ¿Qué es Software Libre en nuestro proyecto?

El software Libre está presente en todas las fases de este proyecto.

En la parte del agente (modelado y animación) hemos utilizado Blender 2.25, una herramienta liberada en el 2002 y cuyo lenguaje de base es Python, y para el portal web nos decidimos por WebWare, una plataforma libre basada también en Python, y Cheetah, un sistema de plantillas soportado por Python.

Por útlimo destacar que el motor de inferencia de HEVAH ha sido escrito en CLIPS, una herramienta libre desarrollada por la NASA (detalle importante este a comentar a todos aquellos que afirman que el Software Libre no está preparado para uso profesional), y que todos los ficheros de almacenamiento utilizados (tanto en PORTAD como en HEVAH) son XML.

Capítulo 3

Tests Adaptativos Informatizados

En este capítulo vamos a dar una visión general sobre los tests informatizados; para ello explicaremos los modelos más utilizados a la hora de trabajar con estas herramientas.

3.1. ¿Qué es un test adapativo informatizado?

Actualmente los ordenadores han pasado a ser una herramienta fundamental en el diseño de tests. Los métodos tradicionales de evaluación mediante test, dependen del proceso estático para almacenar, manejar y analizar los datos, mientras que en los tests adaptativos este proceso es dinámico. Un test adaptativo es un test en el cuál la presentación de cada elemento y la decisión de finalizar el test son dinámicamente adoptados según el rendimiento del estudiante. La naturaleza dinámica de los tests adaptativos requiere:

1. Un algoritmo más complicado para la selección de las preguntas a plantear al alumno que los tests tradicionales.
2. Algún algoritmo para evaluar las respuestas dadas a cada pregunta.
3. Algún algoritmo para evaluar el conjunto total de respuestas dadas por el alumno cuando finaliza el test.
4. Por último, pero no menos importante, informar al estudiante de las consecuencias que podría tener el contestar al azar, dejar sin responder preguntas, etc...

Una clara ventaja de los tests adaptativos frente a los tradicionales, es que pueden limitar de manera efectiva el tiempo total que ocupa el test, ya que, generalmente, no se puede volver a las preguntas ya contestadas y el test va a depender del conocimiento de cada alumno.

3.2. ¿Por qué no un test tradicional?

Los modelos clásicos de test se han centrado en la creación de diferentes test para diferentes conjuntos de alumnos. Cada uno de estos test es diseñado de modo que un alumno que posea un conocimiento medio en el dominio del test, obtendrá una puntuación media. Estos modelos tienen al menos 3 limitaciones imoprtantes, las cuales comentaremos a continación.

- La primera de las limitaciones a la que hacemos referencia está relacionada con las estadísticas que podemos obtener a partir de las respuestas de los alumnos. Debido al esquema de construcción que sigue el modelo tradicional (comentado antes), las estadísticas obtenidas a partir de un grupo de alumnos son solamente aplicables a alumnos de características similares

- La segunda de las limitaciones afecta a la posibilidad de comparar los resultados obtenidos por estudiantes pertenecientes a distintos grupos. Con el modelo tradicional esta comparación es muy difícil de realizar, salvo en el caso de que los tests a los que hayan sido sometidos ambos grupos sean equivalentes, ya que en caso contrario no existe una relación lineal entre las estadísticas de ambos tests. El problema está en que generalmente el diseñador de un test no desarrolla otros tests equivalentes.

- La última de las limitaciones está relacionada con la efectividad real del test: ¿mide de manera precisa el conocimiento del dominio del test por parte del alumno? La mayor parte de los tests cometen un error entre la valoración que ofrecen y el conocimiento real del alumno (despistes, azar, ...); sin mebargo, y aunque esto es algo que se sabe a priori y se tiene en cuenta, se da por supuesto que el error es el mismo para todos los alumnos. De esta forma se intenta conseguir que la comparación entre estudiantes sea exacta, cosa que no ocurre.

En resumen, cada test diseñado según el modelo tradicional (o clásico) tiene un rango efectivo, dentro del cual la evaluación de los distintos niveles de conocimento es relativamente precisa, de tal forma que los alumnos cuyo conocimiento se encuentre en ese rango serán valorados con bastante exactitud. Sin embargo, en el momento en el que el conocimiento de un alumno se salga de ese rango efectivo, ya sea por arriba o por abajo, la valoración tenderá a ser menos exacta.

Como respuesta a todas estas limitaciones (y otras) surgió el modelo IRT (Item Response Theory)

3.3. IRT (Item Response Theory)

La IRT es una rama en la teoría de análisis de los tests.

Según la IRT, la medición educacional se puede definir como el proceso que nos permite cuantificar la capacidad de un alumno en un determinado dominio del conocimiento, a partir del comportamiento de dicho alumno frente a una muestra de ese dominio.

Además, IRT no nos ofrece únicamente el método para analizar los elementos que componen el test, sino que además nos permite transformar la escala de puntuación IRT de un test en una estimación de la puntuación aplicable al dominio que se pretende medir, es decir, generalizar los resultados. Para que esta transformación pueda tener lugar, han de cumplirse las siguientes condiciones:

1. Existencia de un conjunto de n items que puedan ser considerados como muestra del dominio.

2. Debe existir un modelo bien definido de respuesta para los items. Uno de los métodos más empleados para la definición de este modelo es la utilización de la probabilidad estadística de que un alumno responda correctamente a una pregunta en función de su nivel de conocimiento.

3. Este modelo de respuesta tiene que estar parametrizado. Estos parámetros deberían ser estimados a partir de grandes muestras de alumnos (preferentemente más de 500 individuos). IRT require de 1 a 3 parámetros: un parámetro de discriminación entre alumnos con distinto nivel de conocimiento; otro que indique el nivel de dificultad del item; y un último elemento que tenga en cuenta el factor de adivinanza.

4. Los items seleccionaos deben abarcar las dimensiones del dominio.

Algunas de las ventajas que se obtienen al utilizar la estimación IRT para puntuar los dominios a medir son las siguientes:

1. Los elementos de cualquier test no tienen que ser una muestra probabilística del dominio, y tienen un error estándar simple. Esta propiedad abre el camino de los tests adaptativos informatizados como herramientas para la medición del conocimiento en un determinado dominio.

2. El método IRT de estimación de puntuación se generaliza directamente a métodos IRT multidimensionales y a modelos de elementos con múltiples categorías de respuestas.

3. El estimador IRT de puntuación tiene mayor exactitud que la puntuación obtenida en los tests, evaluados con los métodos tradicionales (nº. de respuestas correctas, etc.).

En resumen, uno de los principales requisitos del modelo IRT es disponer de un gran banco de items, cada uno de ellos con un modelo de respuesta bien definido. Cuando un test es administrado para un determinado alumno, las preguntas presentadas a éste son seleccionadas según la respuesta dada a la pregunta anterior y según los modelos asociados a las preguntas del conjunto aún no efectuadas.

Por tanto, las limitaciones de los modelos clásicos ya no están presentes en el modelo IRT, ya que en éste no es necesario tener grupos de alumnos exactamente equivalentes cuando se toman las muestras de los items, debido a que el análisis IRT estima la dificultad del item y la discriminación de éste independientemente de esa muestra de alumnos. Así, las valoraciones que miden el conocimiento del alumno son dependientes de los items seleccionados para medir este conocimiento, y no de la dificultad del test (de hecho, el test en sí no tiene nivel de dificultad específico).

Sin embargo, no todo son ventajas en el modelo IRT. Algunas de las desventajas a descatar serían las siguientes:

1. El gran número de alumnos que tienen que ser evaluados (superior a 500) antes de que los resultados puedan ser formalmente aplicados.

2. La necesidad de valorar de manera precisa, el nivel de dificultad de cada uno de los tests del conjunto mediante algún modelo probabilístico.

3.4. Alternativas al modelo IRT

El uso del modelo IRT puede verse frenado por la gran cantidad de datos necesarios y la complejidad de las fórmulas a utilizar. Debido a estas dos razones surgen modelos alternativos, destacando esencialmente dos:

- **SPRT (Sequential Probability Ratio Test).** Una de las limitaciones que presenta este método es que no toma en cuenta, de manera explícita, los parámetros encargados de la dificultad y de la discriminación.
- **EXSPRT (EXpert SPRT).** Surge como una mejora del SPRT, a partir de la combinación de éste con el método de razonamiento de los sistemas expertos. Establece un peso a cada item del conjunto, permitiendo así, que la dificultad y el grado de discriminación de ésta, sean tomadas en cuenta en la elección de las preguntas. Existen varias versiones según el algoritmo de selección usado:
  - EXSPRT-R. Los items son seleccionados aleatoriamente.
  - EXSPRT-I. Los items son seleccionados de manera "inteligente", es decir, con este método el siguiente elemento a presentar al alumno es elegido según su grado de utilidad. A la hora de seleccionar la siguiente pregunta tenemos en cuenta el nivel que hasta ahora ha mostrado el alumno, intentando escoger aquella que nos lleve de una manera más rápida a la determinación del nivel real del mismo.

Común a SPRT, EXSPRT-R y EXSPRT-I es la formación de los porcentajes de probabilidad discreta. Es decir, en lugar de suponer una función continúa (como hace IRT), estos métodos clasifican a los alumnos en categorías discretas.

3.5. Estado Actual de los Tests Informatizados

A la hora de intentar valorar los resultados obtenidos mediante estos tests informatizados, tenemos que tener en cuenta, al menos, cuatro conceptos fundamentales:

1. Banco de elementos (o items) de un test.
2. Tests informatizados.
3. Tests adaptativos informatizados.
4. Efectividad del uso de ordenadores en la evaluación mediante tests.

A continuación haremos una breve introducción a cada uno de ellos.

3.5.1. Banco de elementos de un test

Cuando hablamos del banco de elementos de un test estamos haciendo referencia a cualquier procedimiento que se utilice para la creación, análisis, almacenamiento, dirección o selección de los elementos del test en cuestión.

Para la creación del banco de elementos podemos utilizar cualquiera de los modelos expuestos anteriormente; el modelo a seguir vendrá dado por los objetivos que se persigan con el test.

Evidentemente, el uso de los ordenadores ofrecen una mayor facilidad tanto a la hora de crear los bancos de elementos, como a la hora de analizar los resultados de los mismos. De hecho, se ha diseñado software específico para estos fines.

3.5.2. Tests informatizados

Se conocen como tests informatizados a todos aquéllos que son administrados a través de ordenadores. Tests como los de multi-elección, verdader-falso, basados en palabra clave... son muy fáciles de adaptar al ordenador, obteniendo dos beneficios claros e inmediatos:

1. Los tests pueden ser administrados individualmente, almacenando los resultados para, una vez tengamos la muestra completa, analizar los resultados, todo ello de forma rápida y cómoda.
2. El motivo de acotar el tiempo del test por el hecho de que una persona debe controlarlo, ya no es necesario.

3.5.3. Tests adaptativos informatizados

Los tests adaptativos informatizados se pueden considerar como un subconjunto de los tests informatizados, ya que, al igual que éstos, son administrados por ordenadores; sin embargo, poseen 3 características adicionales:

1. Los elementos que conforman el test son elegidos dinámicamente, a medida que el alumno lo va realizando, según su nivel de conocimientos. Esto significa que dos alumnos que estén realizando el mimso test no tienen por qué contestar a las mismas preguntas.

2. El test no acaba por agotamiento de los elementos, sino que acaba en el momento en que el ordenador localiza el nivel de conocimiento del alumno que está siendo evaluado.

3. Debido al punto anterior, los tests adaptativos generalmente requieren menos tiempo para su administración y para la estimación del nivel del alumno.

Los beneficios que aportan este tipo de tests respecto a los tests tradicionales son los siguientes:

1. El test debe poder realizarse las veces que se desee, limitadas éstas sólo por el número de ordenadores disponibles, restricciones del profesor de la materia, etc.

2. El tiempo que ocupa el test es típicamente más corto.

3. El test es administrado y puntuado de manera automática y, podría emitir un informe sobre la valoración global del alumno de manera casi inmediata.

4. Las preguntas presentadas al alumno, corresponden al nivel de éste: ni demasiado fáciles, ni demasiado difíciles.

El algoritmo general que siguen los tests adaptativos sería este:

1. El ordenador establece el nivel inicial con el que empezarán los elementos del test en función del conocimiento estimado del alumno. Hay diversas formas para estimar este conocimiento inicial: aleatoriamente, a través de tests anteriores realizados por el alumno, mediante entrada directa (se le pregunta que nivel cree que tiene, o un profesor lo introduce...).

2. Una vez llevada a cabo esta estimación inicial, se procede a intentar determinar el nivel exacto del estudiante, para lo cual el ordenador selecciona una pregunta del nivel adecuado entre los elementos del banco.

3. A medida que el alumno va dando respuestas, el ordenador va seleccionando la siguiente pregunta entre las existentes en el banco; el nivel de dificultad de la nueva pregunta irá dependiendo de las respuestas que ha dado hasta ese momento el alumno.

4. Una vez el ordenador determina que ha estimado de manera relativamente exacta el nivel del alumno, informa de su valoración y da por concluido el test. Dentro de esta valoración pueden ir recomendaciones orientadas a la mejora del rendimiento por parte del estudiante, tales como determinación de puntos flojos, inconsistencias en el dominio de los conceptos....

Para que los tests adaptativos sean realmente funcionales, tenemos que tener en cuenta una serie de puntos básicos:

- Lo primero que se debe hacer es enseñar a utilizar el ordenador, ya que de otra manera, si el alumno no está familiarizado con los ordenadores, el test pierde bastante eficacia.

- A la hora de seleccionar la siguiente pregunta, el algoritmo implementado debe contemplar la posibilidad de que, en el caso de que la respuesta a la pregunta anterior sea correcta, el alumno la haya adivinado. Esto significa que el algoritmo no debe ser simplemente aumentar el nivel en caso de respuesta correcta y disminuirlo en el caso de respuesta errónea, sino que además se deben contemplar otros factores.

- Se debe seguir el orden que estableza el ordenador, es decir, no se permitirá pasar a una nueva pregunta sin haber contestado la actual, y no se podrá volver a una ya formulada.

Generalmente, las distintas implementaciones que se realizan a la hora de elaborar tests adaptativos informatizados realizan modificaciones o adaptaciones de estas especificaciones generales, aplicando algunas de las técnicas de análisis de los elementos vistas anteriormente.

3.5.4. Test Adaptativos Informatizados Bayesianos

En esta sección veremos en detalle la forma de implementar y ejecutar "test adaptativos bayesianos". En primer lugar haremos una pequeqña introducción a la teoría bayesiana, para a continuación estudiar este tipo de tests, analizando su estructura, la evolución de dicho test durante su realización, los metodos de evaluación de las respuestas y las condiciones de parada.

Inferencia Bayesiana

El razonamiento bayesiano proporciona un enfoque probabilístico a la inferencia. Está basado en la suposición de que las cantidades de interés son gobernadas por distribuciones de probabilidad y que se pueden tomar decisiones óptimas razonando sobre estas probabilidades junto con los datos obtenidos. Este enfoque está siendo utilizado en multitud de campos de investigación.

Podemos distinguir dos formas de entender la probabilidad:

- frecuentista (objetiva): la probabilidad de un suceso es la frecuencia relativa de veces que ocurriría el suceso al realizar un experimento repetidas veces.
- Subjetiva (bayesiana): es el grado de certeza que se posee sobre un suceso (es personal).

En el lenguaje cotidiano es frecuente oir afirmaciones del tipo "es muy probable que el partido X gane las próximas elecciones", "es improbable que Juan haya sido quien llamó por teléfono.º "es probable que se encuentre un tratamiento eficaz para el sida en los próximos 5 años", pero estas expresiones no pueden cuantificarse formalmente, lo que las excluye en cierto modo de un marco frecuentista. Sin embargo, una cuantificación sobre base subjetiva resulta más familiar y apropiada para el enfoque bayesiano. Al admitir un manejo subjetivo de la probabilidad, se podrán emitir juicios de probabilidad sobre una hipótesis H y expresar por esa vía su grado de convicción al respecto, tanto antes como después de haber observado los datos.

Veamos la formula del teorema de bayes:

$$P(\frac{H}{D}) = \frac{P(\frac{D}{H}) * P(H)}{P(D)}$$

- P(H): es la probabilidad de que la hipótesis H sea la correcta, se suele llamar probabilidad a priori
- P(D): es la probabilidad de que los datos D sean correctos.

- P(D/H): es la probabilidad de observar los datos D sabiendo que se ha dado la hipótesis H. Se le suele denominar verosimilitud
- P(H/D): es la probabilidad de que se de la hipotesis H sabiendo que se han dado los datos D. Se suele denominar probabilidad a posteriori.

En la mayoría de problemas donde se plantea la inferencia bayesiana, se parte de un conjunto de hipótesis y se trata de encontrar la hipótesis más probable de dicho conjunto. A esta hipótesis más probable se le suele denominar hipótesis maximum a posteriori o MAP.

Utilizando el teorema de Bayes, diremos que h_{MAP} es una hipótesis MAP de acuerdo a:

$$h_{MAP} = argmax_{henH} P(\frac{h}{D}) = argmax_{henH}(\frac{P(\frac{D}{h}) * P(h)}{P(D)}) = argmax_{henH} P(\frac{D}{h}) * P(h)$$

En algunos casos todas las probabilidades en H son igualmente probables a priori ($P(h_i = P(h_j) \forall h_i, h_j \in H$). En este caso sólo utilizaríamos el término de verosimilitud, P(D—h), y podemos simplificar aún más la anterior ecuación:

$h_{ML} = argmax P(\frac{D}{h})$

donde a la hipótesis h_{ML} se le suele nombrar como hipótesis de máxima verosimilitud (Maximum Likelihood).

Supongamos ahora que debemos elegir entre dos hipótesis, h_1 y h_2, dados los datos D. El criterio de elección para responder de forma eficiente sería seleccionar la hipótesis más probable. Es decir, aplicaríamos lo que se conoce como regla de decisión:

$$\text{si } P(h_1|D) > P(h_2|D) \text{ elegir } h_1, \text{ sino elegir } h_2$$

Si aplicamos la regla de Bayes a cada término nos queda:

$$P(\frac{D}{h_1}) * (\frac{P(h_1}{)} P(D)) > P(\frac{D}{h_2}) * (\frac{P(h_2)}{P(D)})$$

- $\frac{P(\frac{D}{h_1})}{P(\frac{D}{h_1})}$ ratio de verosimilitud
- $\frac{P(h_2)}{P(h_1)}$ ratio a priori

Aplicando logaritmos a ambas partes nos queda:

$$Ln(\frac{P(\frac{D}{d_1})}{P(\frac{D}{d_1})}) > Ln(\frac{P(h_2)}{P(h_1)})$$

En ausencia de información a priori todas las hipótesis son igualmente probables y el término de la derecha es ln 1=0. La regla de decisión en ausencia de información a priori queda:

si $Ln(\frac{P(\frac{D}{d_1})}{P(\frac{D}{d_1})}) > 0$ elegir h_1, sino elegir h_2

Estructura de la red

Una estructura propicia en la construcción de una red bayesiana para un test adaptativo comprende tres niveles jerárquicos. Adoptando este criterio definimos los niveles de: "tema", çonceptoz "pregunta". Veamos gráficamente esta estructura:

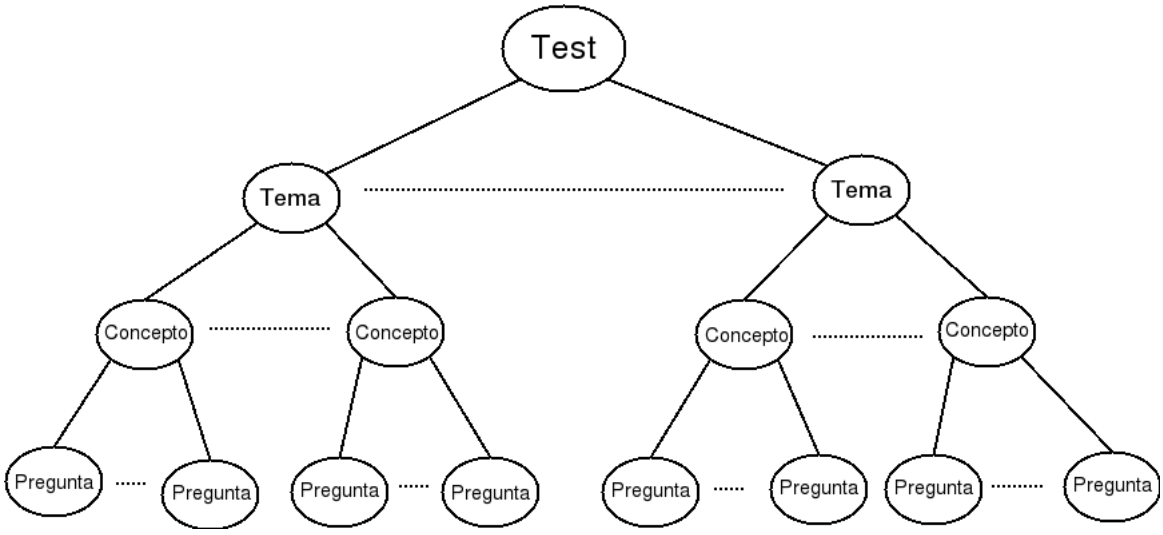


Figura 3.1: Red Bayesiana

A la hora de realizar la evaluación del test diferenciamos dos etapas. Por un lado una etapa de 'diagnóstico' en la cual se determina el grado de sabiduría que tiene el usuario de los conceptos presentados en función de sus preguntas. Por otro lado el proceso de 'evaluación' en la cual se estudian los resultados de la etapa anterior para determinar el grado de conocimiento que posee el usuario sobre el tema que se le está preguntando.

Elementos básicos del test adaptativo bayesiano

Los elementos básicos de un test adaptativo bayesiano son:

- El modelo de respuesta asociado a cada pregunta: viene dado por la distribución de probabilidad de la pregunta condicionada a sus padres.

- El método de puntuación: viene dado por un modelo bayesiano utilizando propagación de las probabilidades.
- El banco de ítems o preguntas: se debe tener en cuenta diversos factores como la posibilidad de que el alumno adivine la respuesta correcta o los errores no intencionados.
- El nivel inicial: se valora el conocimiento que se tenga del alumno o en su defecto se entiende que tiene la misma probabilidad de saberlo que de no saberlo.
- El método de selección de preguntas: el criterio adoptado en este punto determinará el grado en el que el test se adaptará a los conocimientos del usuario. Así se pretende cumplir el objetivo del test adaptativo, mejorar la precisión del diagnóstico reduciendo el número de preguntas. Distintos criterios son:
  - Criterio Aleatorio: según este criterio todas las preguntas tienen la misma probabilidad de ser elegidas.
  - Criterios Adaptativos: según este criterio se tienen en cuenta las respuestas anteriores dadas por el usuario para determinar la siguiente pregunta.
    - Criterios basados en la información.
    - Criterios condicionados.
- El criterio de parada: puede haber dos motivos para la conclusión de un tests; o bien se agotas las preguntas, o bien se estima que ya se puede determinar de forma efectiva el nivel del alumno.

3.5.5. Efectividad del uso de ordenadores en la evaluación mediante test

A la hora de valorar la efectividad de los ordenadores en la evaluación del conocimiento mediante tests debemos analizar las ventajas y desventajas de este sistema.

Hemos dividido las ventajas en dos grandes grupos:

- Consideraciones de evaluación.
- Consideraciones desde el punto de vista del trabajo humano.

Entre las consideraciones de evaluación hemos destacado las siguientes:

- Los ordenadores son mucho más exactos que los humanos puntuando cada pregunta.
- Los ordenadores son mucho más exactos a la hora de emitir un informe sobre las puntuaciones alcanzadas en la totalidad del test.
- El dianóstico y el 'feedback' (información de refuerzo que se le ofrece al estudiante, tales como consejos, recomendaciones, etc.) es proporcionado mucho más rápidamente. Si además incorporamos elementos de inteligencia artificial, este 'feedback' puede llegar a ser altamente descriptivo.
- A través del uso de los tests adaptativos y el modelo IRT (o sus alternativas) es posible determinar el nivel de conocimientos de un estudiante de una manera relativamente precisa.
- El uso de este tipo de tests minimiza los riesgos de trampas o memorización de test, ya que es personal para cada alumno; es más, a un mismo alumno que se someta al mismo test dos veces no se le presentarán las mismas preguntas (por norma general).

Desde el punto de vista del trabajo humano hemos destacado las siguientes:

- El uso de los ordenadores permite a los alumnos trabajar a su ritmo.
- Los tests adaptativos duran menos, y son más eficientes.
- A través de la utilización de los tests adaptativos deberíamos conseguir un mayor nivel de satisfacción del alumnado, ya que pueden trabajar a su ritmo y el nivel de dificultad de las preguntas estará acorde son su nivel de conocimientos.
- Se ha comprobado que los estudiantes ven los tests adaptativos menos intimidatorios que los tradicionales, ya que las preguntas se presentan una a una, en lugar de en batería.
- Evita distracciones y dudas entre qué pregunta contestar antes.

Evidentemente, el trabajo con tests adaptativos no supone únicamente ventajas. Hemos divido las desventajas en dos categorías.

- Inconvenientes tecnológicos.
- Inconvenientes de rendimiento.

Desde el punto de vista tecnológico nos podemos encontrar con limitaciones, algunas de las cuáles presentamos a continuación:

- No siempre existen equipos adecuados.
- La capacidad de la pantalla puede suponer una limitación en algunos dominios.

Al hablar de rendimiento, nos encotramos con los siguientes inconvenientes:

- Dependiendo del dominio a evaluar, el empleo de tests informatizados puede ser más o menos eficiente.
- El rendimiento del alumno puede verse influenciado por el empleo del ordenador, de ahí que sea importante que previamente haya un contacto con el entorno de trabajo, de tal forma que se pueda familiarizar con el mismo.
- Al utilizar bancos de elementos de grandes dimensiones, la información de un elemento puede, inadvertidamente, contribuir a la respuesta de otro.

3.6. ¿Qué utilizamos en nuestro proyecto?

Tanto en HEVAH como en PORTAD nosotros hacemos uso de los tests adaptativos bayesianos, ya que hemos considerado que son los más adecuados para nuestro caso y que facilitan la implementación de los mismos. Una de las principales bazas a su favor es el hecho de contar con una función de probabilidad discreta (al igual que algunas de las variantes de IRT), en lugar de una función continúa.

Capítulo 4

Técnicas de Modelado, Animación e Interacción en Blender

4.1. Técnicas Gráficas

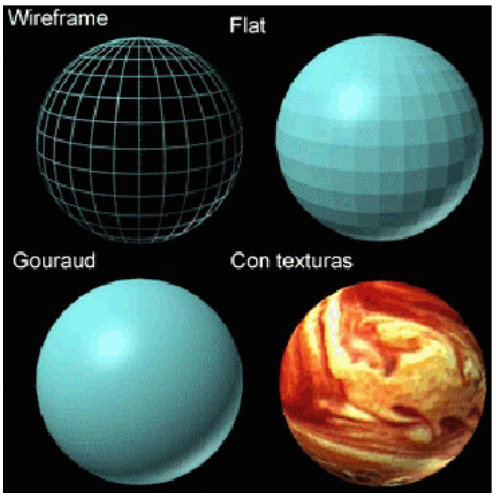
4.1.1. Texturas

El mapeado de texturas es una técnica gráfica que consiste en aplicar una serie de dibujos o plantillas a las caras de un objeto tridimensional. Por ejemplo, si pegamos la foto del rostro de una persona en una esfera plana, la habremos convertido en una cabeza. Lo mismo ocurre si aplicamos la textura de la madera a un cilindro: obtendremos un tronco bastante real.

La técnica de mapeado de texturas comenzó con la tesis doctoral de 1974 de Ed Catmull, lo que llevó al mundo de los gráficos por ordenador un nuevo nivel de realismo. (En un principio, la técnica mas usada para dotar de realismo los polígonos fue el trazado de rayos (raytracing), que es mas rápido que OpenGL, aunque de menor calidad. Posteriormente, se implementaron muchas operaciones por hardware, con lo que la velocidad del proceso creció y su uso también.)

En la imagen que aparece debajo muestran los cuatro pasos utilizados en la creación de un objeto 3D con texturas.

En el primer paso se crea el objeto mediante la unión de cientos de polígonos. Posteriormente estos se rellenan de un color para dar a la figura un aspecto sólido. Con el sombreado Gouraud eliminamos las aristas de los polígonos de modo que la imagen gane en realismo y de la impresión de ser un sólo objeto sólido. Por fin, el mapeado de texturas se ocupa de "pegar"encima de la esfera un dibujo determinado



Para que estas texturas se ajusten debidamente al objeto donde se pegan, se suele aplicar una corrección de perspectiva o textura inversa, que estira o comprime la textura según su posición, evitando cualquier anomalía y pérdida de realidad. Básicamente se trata de buscar el pixel de la pantalla que se corresponde con cada pixel de la textura.

Las características y ventajas del mapeado de texturas son obvias, ya que permite aumentar enormemente la complejidad visual de un modelo sin necesidad de aumentar su complejidad geométrica. En cualquier escena que se precie, la iluminación y las texturas suelen tener mayor importancia que la propia geometría que sustenta los modelos. Convertir la visualización de un simple polígono rectangular en un muro de ladrillo o una sencilla esfera en un planeta, abre nuevas vías de explotación para multitud de disciplinas, desde la simulación civil y militar hasta la arquitectura, el diseño industrial o la aeronáutica, entre muchas otras.

Mapeado de Texturas

Para una mayor eficiencia en el mapeado, una textura debería ser rectangular. Podremos codificar una textura como un array de dos dimensiones. También es posible el uso de texturas no rectangulares. Por ejemplo, si deseáramos una textura hexagonal, podríamos introducirla en el interior de un rectángulo más grande, y definir un polígono en el interior del mismo que cubra los pixels adecuados.

Es posible que parte del polígono quede sin recubrir por la textura. En estos casos se debe optar por expandir dicha textura o por repetirla hasta cubrir todo el polígono, creando un efecto de mosaico.

Debemos conocer en que posición del espacio está situada la textura, al igual que

debíamos conocer la posición de cada vértice del polígono en la pantalla. En el caso de la texturai, esto es más sencillo, ya que sabemos a priori que se trata de un rectángulo. Por lo tanto, nos bastará con almacenar la posición de su vértice superior izquierda y dos direcciones. Dichos parámetros deberán ser transformados para conocer su posición relativa respecto al observador.

Por último se debe tener en cuenta que la textura no tiene por qué encontrarse en el mismo plano que el polígono. Por ejemplo, supongamos que deseamos aplicar una textura de cielo nublado a un entorno 3D correspondiente a una casa sin techo. Para poder representar esto, podemos darle a P (esquina superior izquierda del rectángulo de la textura) un valor mucho más alto para la coordenada que la del techo abierto de la casa. Así, al movernos por la habitación, el movimiento de la textura del cielo será menor, causando la sensación de que se encuentra mucho más lejos.

Tipos de Texturas

En un sentido amplio, podemos definir una textura como un array de uno o dos dimensiones de pixels. A cada uno de estos pixels se le denomina texels. Veamos los distintos tipos de texturas:

- Textura 1D: tienen un único pixel de ancho o de alto
- Textura 2D: tienen mas de un pixel de alto y de ancho. Se componen de valores de color RGB y pueden tener valores alpha.
- Texturas 3D: además de cubrir un espacio plano poseen profundidad.
- Texturas Matemáticas: se obtienen mediante algoritmos de cálculo de patrones.

Tipos de Mapeados de las texturas

En general, el proceso involucra el mapeado matemático de un dominio (el de los pixels de la pantalla) a otro (los pixels de la textura). Dicho proceso matemático se basa principalmente en operaciones con matrices.

Distintos tipos de mapeado son:

- Mapeado plano: coloca la imagen sobre la superficie
- Mapeado cúbico: proyecta la imagen de forma plana desde los tres ejes
- Mapeado esférico: se encoge por los polos y se amplía por la zona central

- Mapeado cilíndrico: envuelve al objeto siguiendo uno de sus ejes
- Mapeado UV: la textura esta anclada a las coordenadas UV correspondientes. Abordamos en detalle este tipo de Mapeado en otra sección

Tipos de Algoritmos para el mapeado de texturas

Dichas técnicas se basan en aplicar unas determinadas ecuaciones de mapeado de texturas para conseguir determinar que texels de la textura deben ser dibujados y cual es su pixel correspondiente dentro del polígono.

Los algoritmos más usados son:

- Affine Mapping (muy rápido pero de mala calidad)
- Area Subdivision (buena cálidad y regular en velocidad)
- Scanline Subdivision (muy rápido y de muy buena calidad)
- Parabolic Mapping (buena velocidad pero de calidad regular)
- Constant-Z Mapping (rápido pero de regular cálidad)
- Perfect Mapping (de muy buena calidad pero lento)

Texturas Multimapa (Mip Mapping)

Mip-Mappping es la técnica que se encarga de mapear muchas texturas en una sola, dependiendo de la distancia. Esta técnica aumenta el grado de realismo y acelera el proceso.

La idea es realizar un preprocesado de la textura usando un filtrado que imite al comportamiento del ojo humano.

El mapeado de texturas funciona de la misma manera que en el caso de que no se utilice Mip-Mapping. Se deben seleccionar los texels de la textura que serán colocados en cada pixel del polígono. La única diferencia es que la textura de la que obtener esos texels cambia conforme el observador se aleja del polígono a mapear.

Se debe seleccionar el Mip-Map para el cual el tamaño del texel se encuentre más cercano al del pixel de la pantalla. Por ejemplo, si para un polígono dado cada pixel en la pantalla cubre cinco texels, se debería seleccionar el Mip-Map cuyo tamaño es 1/4 de la textura original.

Bump Mapping

El Bump-mapping intenta imitar las formas reales de la naturaleza, donde no suelen presentarse objetos geométricos.

Funciona de la siguiente forma: tenemos una superficie sobre la que se ha aplicado una textura y después se le aplica un foco de luz que lo ilumine. Para dar la sensación de que la superficie de esa textura es rugosa, se aplica una tercera capa denominada "Patrón de Relieve.º "Bitmap de alturas". El Bump-mapping a veces se define como textura 3D virtual, pues aporta información en el eje Z.

Alpha Blending

Se trata de una técnica que permite crear objetos transparentes. Normalmente, un píxel que aparece en pantalla tiene valores de rojo, verde y azul. Si el escenario 3D permite usar un valor alfa para cada pixel, tenemos un canal alfa. Un objeto puede tener diferentes niveles de transparencia: por ejemplo, una ventana de cristal limpia tendría un nivel muy alto de transparencia (un valor alfa muy bajo), mientras que un cubo de gelatina podría tener un valor alfa medio.

4.1.2. Materiales

Los materiales determinan como debe verse la superficie de un objeto mediante parámetros como el color, el brillo, la reflectividad, etc. Un parámetro muy importante en un material es la textura.

Los materiales son muy importantes para hacer los objetos más realistas. Por ejemplo, si se diseña una moneda, para hacerla más real, se le asignará una textura de bronce brillante.

4.1.3. Mapeado UV

El mapeado, mapeo o mapping es el proceso de desarrollo y asignación de atributos de material a un objeto.

La técnica conocida como Mapeado UV consiste en la asignación de una textura a una superficie en 3D a través de cada una de sus caras. Dicha asignación se realiza mediante las coordenadas UV de la superficie de la cara. Las coordenadas UV son similares al sistema de coordenadas de imagen XY, pero se ajustan a la malla, independientemente de cómo se doble o se retuerza. Dada una textura (JPG, GIF, etc.) el mapeado UV dice 'esta

porción de imagen va aquí'. El mapeado UV es generalmente realizado por el software de modelización 3D, como 3D Studio MAX, Blender, Rhino, etc.

Esta técnica posiblemente se acerca más a la realidad que cualquier otra, puesto que la textura está 'anclada' a las coordenadas UV correspondientes (para entendernos, las coordenadas UV son como la longitud y la latitud en una esfera, un valor determinado por dos números que indica la posición exacta de un punto en el espacio). Sólo tiene que colocar la textura, y retorcer, estirar o doblar la figura, para que la textura aplicada se acomode a los nuevos valores. Aunque sus ventajas son numerosas en cuanto a texturizar objetos deformados o moldeados, donde verdaderamente se aprecia el poder del mapeado UV es en la animación de personajes.

Blender soporta nativamente el mapeado UV tradicional. Éste consiste en 'mapear' la geometría de nuestro modelo sobre un plano. Normalmente se dispondrá de un editor UV y una serie de algoritmos que realicen el proceso de despliegue de nuestro modelo sobre un plano.

4.1.4. Iluminación

Para obtener imágenes realistas se aplican efectos de iluminación sobre las superficies visibles. Para ello se utilizan los modelos de iluminación y sombreado.

Modelos de iluminación

Determinan el color de una superficie en un punto dado. Veamos los distintos modelos:

- **Iluminación local**: Esta luz llega a un objeto o a un conjunto de objetos específicos dentro de la escena. Podemos distinguir:
  - *Luz ambiente*. La luz afecta por igual a todas las superficies desde todas las direcciones. La luz se refleja por igual en todas las direcciones.
  - *Luz difusa*. Este modelo considera una única fuente de luz puntual que emite uniformemente en todas las direcciones. Las intensidades en cada punto del objeto varían dependiendo de la dirección y distancia a la fuente de luz. La variación de la intensidad se rige por la Ley de Lambert, válida para superficies mates, es decir , que aparecen igualmente brillantes desde cualquier punto de vista.

- *Luz especular (Modelo de Phong).* Los objetos con esta propiedad reflejan la luz en una única dirección. Se manifiesta con un brillo o toque de luz en alguna de las direcciones de vista.
- *Modelo de Warn.* La luz proviene de la reflexión especular de una superficie iluminada por una fuente.

Iluminación global: Las fuentes de luz llegan a todos los objetos de la escena, además influye la luz reflejada por los otros objetos. A continuación explicaremos los dos tipos de iluminación más frecuentes correspondientes a este modelo:

- *Trazado de Rayos, o modelos dependientes de la vista.* Dada la dirección de vista se discretiza el plano de vista para determinar los puntos en los que evaluar el modelo de iluminación.
- *Radiosidad, o modelos independientes de la vista.* Discretizan el entorno y lo procesan para tener disponibles los valores necesarios para la ecuación de iluminación en cualquier punto y desde cualquier punto de observación.

Modelos de Sombreado

Especifican cómo aplicar el modelo de iluminación escogido. Esto se puede hacer, bien aplicando dicho modelo sobre cada píxel, o bien, aplicándolo en algunos píxeles e interpolando el resto. Las distintas interpolaciones dan lugar a los sombreados:

- **Constante**: se determina un único valor de intensidad para sombrear un polígono completo.
- **Gouraud**: los valores de la intensidad en cada punto se determinan por la interpolación de los valores de intensidad en los vértices de los polígonos.
- **Phong**: se determinan las intensidades en cada punto utilizando la interpolación de los vectores normales en los vértices

Tipos de fuente de luz

- **Luz puntual**: arroja luz en todas las direcciones. Por ello es llamada luz omnidireccional (modelo de iluminación local)
- **Foco**: arroja luz en forma cónica o piramidal en una dirección específica. Características: ángulo del cono variable y factor de caída (modelo de iluminación local)

- **Luz infinita**: Están tan lejos de los elementos de la escena que los rayos inciden paralelamente entre sí (lo podemos identificar con un modelo de iluminación global).
- **Área de luz**: grupos de luces en un área de luz única (o superficies que se comportan como emisores de luz). Pueden escalarse a cualquier tamaño (modelo de iluminación local).
- **Luz lineal**: tienen longitud pero no profundidad. Requieren muchos más cálculos que la combinación de puntos discretos
- **Luz ambiental**: distribuida a través de toda la escena. Viene de todas las direccines. Determina el nivel de iluminación general (modelo de iluminación local).

Componentes de una fuente de luz

- **Posición y orientación**: aplicamos transformaciones geométricas para su definición, colocando y orientando la fuente de luz en la escena.
- **Color**: habitualmente se usan modelos aditivos como:
  - **RGB (red green blue)**: el color de la luz se hace más claro cuanta más cantidad de cada color mezclemos.
  - **HSB(Hue Saturation Brightness)**: podemos especificar la intensidad de la luz independientemente de su color o matiz.
- **Intensidad**: varía habitualmente entre 0 y 1 y genera efectos ópticos relacionados con el color.
- **Factor de Caída**: controla la fuerza de una fuente de luz y hasta donde llega. En el mundo real es dependiente de la intensidad. En focos se denomina *decay* y en fuentes puntuales *fall-off*.
- **Brillo (Glow) y Cono de luz**: se forma alrededor del foco debido a que la luz es reflejada por partículas del entorno. A menudo se denominan Luces Volumétricas.

Iluminación de una escena

Una de las formas más populares de iluminar una escena es mediante el esquema clásico de iluminación de tres puntos útilizado en cine.

Para modelar una superficie con luz, deberemos asignar diferentes valores a cada plano de la superficie. Distinguimos tres tipos:

- **Luz clave**: define la iluminación principal y ángulo dominante. Es habitualmente la más brillante y arroja las sombras más visibles en la escena.
- **Luz de relleno**: se añade en ángulo opuesto a la luz clave.
- **Luz trasera**: sirve para separar de forma más clara el objeto del fondo. Este tipo de luz se aplica fundamentalmente con reflexión difusa ya que con ambiental o especular no da muy buenos resultados.

4.2. Animación

El efecto óptico de imágenes animadas se debe al fenómeno fisiológico de la persistencia de la visión. La animación tradicional hace uso de la muestra de una serie de imágenes en un corto espacio de tiempo para crear la ilusión del movimiento. Podemos considerar que una animación describe el cambio de una imagen a lo largo del tiempo, con el suficiente número de fotogramas por segundo para dar un efecto de continuidad.

La animación 3D es bastante más compleja. Existen bastantes técnicas que estudiaremos más adelante.

4.2.1. Bases Fisiológicas

La base fisiológica de la animación es la persistencia de la visión. Cada imagen permanece en la retina entre 20 y 100 milisegundos.

Si las imágenes utilizadas para estimular la retina son muy parecidas entre sí, se crea la sensación de movimiento. Esto no es más que una simulación de la realidad, en la visión del mundo real, la imagen actual que percibimos en un instante apenas se diferencia de la percibida milisegundos después.

Este fenómeno se conoce desde 1824, cuando Peter Roget lo dio a conocer a la comunidad científica.

4.2.2. Definición

La animación es el cambio de una imagen a lo largo del tiempo, con el suficiente número de fotogramas por segundo para dar un efecto de continuidad.

Aunque según esta definición, las capturas de vídeo también podrían considerarse una animación. La diferencia está en que en la animación los fotogramas se generan uno a uno. Existen muchos métodos para generar estas imágenes:

- Las imágenes son dibujadas (películas clásicas de 'Disney')
- Las imágenes se generan por ordenador (películas de 'Pixar')
- Las imágenes se obtienen fotografiando modelos en plastilina, modelos articulados, etc. ('Chicken Run', 'Pesadilla antes de Navidad', etc.)

4.2.3. Técnicas de animación tradicional

La animación 2D es el origen y la base de la animación 3D, por ello pasamos a explicar con más detenimiento sus técnicas:

Animación por fotogramas clave

Es la técnica utilizada en los dibujos animados tradicionales. Cada fotograma consiste en un dibujo que conecta con el fotograma anterior. Muchas veces los dibujos no se realizan sobre papel, sino sobre acetatos. Esto permite no tener que redibujar totalmente la escena (el fondo en papel y los personajes en acetato).

Animación por sprites

Un sprite es la imagen de un actor. La sensación de movimiento se consigue concatenando la aparición de los distintos sprites. Esta técnica se usaba mucho en los primeros videojuegos. Por ejemplo, en los juegos de pelea, los luchadores solían realizarse mediante sprites.

Recortes

Al mover un personaje, este no se redibuja por completo, sólo se dibuja la parte móvil y luego se monta junto con el resto. Esta técnica presenta el problema de la unión de los recortes, debe trabajarse mucho para que no se note. Un ejemplo de este tipo de animación es la serie 'Los Simpsons'.

4.2.4. Tecnología digital

Con el auge de la informática han surgido técnicas que hacen uso de la tecnología, permitiendo crear y almacenar las imágenes en formato digital, prescindiendo así (aunque no totalmente) del lápiz y el papel.

Animación por trayectoria

El objeto se mueve por la pantalla independientemente de si cambia o mantiene el mismo aspecto. El software utilizado permitirá establecer los puntos de origen y final y la trayectoria a seguir por el objeto (path).

Animación de gráficos vectoriales

Los gráficos vectoriales son descripciones matemáticas de los objetos de la imagen. Pueden ser utilizados en lugar de los mapas de bits. La principal desventaja es la perdida de realismo, sin embrago, nos permite un tamaño menor de almacenamiento y una escalabilidad mayor.

4.2.5. Técnicas de animación digital 3D

La animación digital 3D es extremadamente compleja y por tanto, necesita una gran potencia de cálculo.

A la hora de abordar un proyecto en 3D, generalmente se diseñan las partes por separado y luego se unen.

Una vez finalizado este proceso, se decide cuales son los frames interesantes, se realiza el proceso de renderizado de dichos frames. Finalmente se incluyen los frames en una secuencia que trata de aportar realismo al movimiento.

Rotoscopia

La rotoscopia es una técnica que se basa en la grabación de los movimientos o comportamientos reales para la generación de fotogramas clave. Esta técnica se usa típicamente para reproducir la locomoción y otros movimientos de personas y animales. Tradicionalmente, la grabación se realizaba mediante fotografía o filmación y se 'calcaba' a mano sobre los dibujos, adaptando la forma de los personajes. Este sistema ha sido utilizado por la compañía 'Disney' desde el primer largometraje de dibujos animados ('Blancanieves'). Este método se ha extendido a la animación por ordenador mediante la captura de movimiento, que consiste en introducir directamente en el ordenador valores numéricos que representan, por ejemplo, las posiciones de las articulaciones en ciertos puntos seleccionados, medidas mediante dispositivos mecánicos, ópticos o electromagnéticos. De este modo se consiguen secuencias muy realistas de movimiento, que luego pueden mezclarse o deformarse en tiempo real (por ejemplo, los movimientos de los luchadores o de los jugadores de fútbol en la última generación de videojuegos).

Animación paso a paso

Utilizando un ordenador, se pueden definir manualmente cada uno de los fotogramas de una animación. Por ejemplo, podemos dibujar manualmente cada imagen de una ani-

mación con el ordenador. Esta técnica es muy lenta, no se suele usar sino en animaciones muy pequeñas.

Animación por cotas

Se establecen una serie de fotogramas clave y dejamos que el sistema genere mediante interpolación los fotogramas intermedios. Debemos cuidarnos de introducir el número suficiente de fotogramas, y además estos deben proporcionar información suficiente para realizar la interpolación. Está técnica se basa en la animación tradicional, en la que los dibujantes más expertos dibujaban los fotogramas clave, los intermedios se dejaban para los dibujantes menos cualificados.

Animación procedural

Consiste en generar el movimiento de manera algorítmica. Se definen una serie de reglas que controlan el comportamiento de los objetos tanto en forma como en posición.

En los métodos procedurales una función o procedimiento define la evolución del conjunto de parámetros por medio de una expresión paramétrica en función del tiempo o bien mediante una expresión integral o incremental, también en función del tiempo.

4.2.6. Cinemática y Dinámica

Muchos movimientos cotidianos son muy difíciles de reproducir debido a las propias leyes de la naturaleza. Para realizar animaciones realistas se hace uso de la cinemática y la dinámica.

Cinemática

La cinemática estudia el movimiento con independencia de las fuerzas que actúan

- **Cinemática directa**: es la posibilidad de mover algunas de las 'piezas' de un personaje o montaje 3D actuando sobre un punto y produciendo un movimiento sobre su eje o centro de rotación.
- **Cinemática inversa**: es la posibilidad de que, moviendo elementos más sencillos en la jerarquía, el programa interpola el resto de articulaciones o puntos de giro, que pueden ser configurados por el animador, para conseguir que se muevan acorde a eso.

Dinámica

La dinámica estudia el movimiento teniendo en cuenta las fuerzas que lo producen.

- **Dinámica directa**: a partir de las masas y fuerzas aplicadas, se calculan las aceleraciones.
- **Dinámica inversa**: a partir de las masas y aceleraciones, se calculan las fuerzas que hay que aplicar.

4.3. Cámaras

Las cámaras ayudan a definir desde dónde, cómo y qué vemos en cada plano. Además, sitúa la escena permitiendo al observador una mayor comprensión de la imagen o las imágenes.

Las cámaras de los programas de diseño 3D proyectan las escenas, bien de 'forma ortográfica' o bien 'en perspectiva'.

4.3.1. Proyecciones

En una escena tridimiensional se requiere una conversión al plano, o lo que es lo mismo, un enfoque bidimensional dotado de profundidad. Para conseguir dicha profundidad se realizan proyecciones.

Veamos un poco más en detalle las proyecciones mencionadas anteriormente:

- *Proyección ortográfica*: se elimina la componente Z de las coordenadas de visualización del objeto, con ello se pierde la noción de distancia.
- *Proyección en perspectiva*: se utiliza para dar un mayor realismo a la imagen final haciendo uso de una perspectiva propiamente dicha.

4.3.2. Cono de Visión

Hablaremos ahora de la pirámide o cono de visión, que no es más que la porción del entorno 3D que se percibe a través de la cámara. El nombre de cono o pirámide proviene de la figura que extenderemos desde el punto donde se encuentra situada la cámara.

Esta pirámide tiene los siguientes parámetros:

- *Punto de vista (POV)*: punto donde esta situada la cámara.
- *Punto de interés (POI)*: es el punto del espacio hacia el que se apunta con la cámara
- *Linea de Vista*: perpendicular desde la cámara que va desde el POV al POI.
- *Planos de recorte*: perpendiculares a la linea de vista. Se establecen los planos de recorte cercano (entre la cámara y el objeto) y lejano (posterior al objeto).
- *Campo de visión*: se define entre los planos de recorte. Los objetos contenidos en este campo son los proyectados al plano de imagen.

- *Longitud focal*: controla el modo en que los objetos son vistos por la cámara.
- *Plano focal*: depende de la lente de la cámara y es el plano en el cual la cámara capta de forma nítida.
- *Profundidad de campo*: esta definido por el área entre los planos focales.

4.3.3. Posicionamiento de las cámaras

El posicionamiento de una o varias cámaras resulta crucial para el observador. Para ajustar la posición o el movimiento de la cámara o el conjunto de caras se utilizan fundamentalmente traslaciones y rotaciones al rededor de uno o varios ejes. Para dichas transformaciones se suele utilizar un sistema de coordenadas esféricas. Teniendo en cuenta esto y ajustando los parámetros anteriormente mencionados, se debe conseguir transmitir al observador la escencia de la escena (o animación).

Para conseguir esto veamos distintos tipos de planos a los que podemos optar, y que a continuación describimos.

Atendiendo al punto de vista

- *Punto de vista*: sitúan la cámara mirando directamente a la acción

Atendiendo al ángulo

Tenemos hasta tres posibles posiciones.

- *Ángulo inferior*: se coloca la cámara en un ángulo inferior a la acción, mirando hacia arriba.
- *Ángulo superior*: se coloca la cámara en un ángulo superior a la acción, mirando hacia abajo.
- *Ángulo inverso*: es el que se suele utilizar cuando se producen conversaciones.

Atendiendo a la distancia

En este caso podemos distinguir hasta 5 posiciones.

- *Primer plano*: se coloca la cámara muy cerca del objeto para observar los detalles. Estos son los planos que se suelen utilizar en la animación facial.

- *Plano medio*: presenta los personajes de la cintura para arriba, centrándose en los gestos y el lenguaje corporal.
- *Plano americano*: se ven los personajes desde las rodillas.
- *Plano general*: se muestran todos los personajes de la escena al completo.
- *Plano largo y panorámico*: se utiliza para situar al espectador presentándole el entorno.

4.3.4. Lentes

Este es otro de los elementos fundamentales en las cámaras, ya que se encargan de proyectar el mundo 3d al plano de proyección de la cámara. Entre los distintos tipos de lentes podemos destacar los siguientes:

- *Lentes normales(50-55mm)*: se útilizan para planos de medios a amplios.
- *Lentes de ángulo ancho(24-28mm)*: tienen una pequeña distorción en los ejes de la imagen ya que se fuerza la proyección de la perspectiva.
- *Lentes telefotográficas(135mm)*: poseen facilidad para cerrar los planos.

4.3.5. Movimientos y rotaciones de las cámaras

Son otra herramienta fundamental a la hora de lograr buenas escenas. Veamos los distintos movimientos que se pueden realizar con la cámara.

- *Dolly*: traslación a lo largo del eje horizontal
- *Truck*: traslación en el eje de profundidad
- *Boom*: traslación en el eje vertical

En cuanto a las rotaciones, estas son las más destacadas.

- *Pan*: rotación respecto al eje vertical
- *Tilt*: rotación respecto al eje horizontal
- *Roll*: rotación respecto al eje de profundidad

4.4. Interacción en tiempo real

La interacción en tiempo real con nuestro personaje era algo fundamental en el desarrollo del proyecto. En este aspecto, la potencia que nos ofrece Blender 2.25 más su módulo de juegos (también conocido como GameBlender) es enorme; de hecho, fue una de las principales razones por la que nos inclinamos por esta herramienta.

Otro de los elementos escenciales a la hora de la interacción era la capacidad de nuestro personaje para hablar; para ello recurrimos a la utilización de TTS.

En este apartado daremos una visión general de las características del módulo de juegos, haciendo una pequeña descripción de los elementos más destacados del mismo, así como una breve introducción al mundo de los conversores de texto a voz o TTS.

4.4.1. GameBlender

4.4.2. Propiedades

Las propiedades son un componente fundamental para la interacción con la escena en tiempo real. Cada objeto puede tener un gran número de propiedades asociadas, que nos van a permitir identificar características, estados, acciones....

Estas propiedades son el equivalente a las variables locales, de tal forma que no son visibles entre distintos objetos. Sin embargo, existen métodos para pasar estas propiedades entre objetos.

Estas propiedades pueden ser de diversos tipos:

- Boolean (Bool). Nos permite almacenar valores booleanos; los valores son TRUE y FALSE (sensitivo a mayúsculas)
- Integer (Int). Permite almacenar números enteros pertenecientes al intervalos [-2147483647, 2147483647].
- Float. Nos permite almacenar números en punto flotante.
- String. Permite almacenar cadenas de texto.
- Timer. Controla el tiempo (en segundos) que ha transcurrido desde que se inicia el juego.

4.4.3. Sensores

Actúan como auténticos sensores; a través de ellos se pueden detectar colisiones, proximidad,... Hay diversos tipos de sensores, de los cuáles vamos a hacer una pequeña descripción:

Sensor Always

Es el más simple de los sensores. Siempre está activo, por lo que se suele usar para acciones generales de control.

Sensor Keyboard

Como su propio nombre indica, son sensores que permiten detectar las pulsaciones de teclado, proporcionando un interfaz para la comunicación entre el usuario y Blender.

Sensor Mouse

Detecta los movimientos y acciones del ratón, proporcionando otro interfaz que permite la comunicación entre usuario y Blender.

Sensor Touch

Permite detectar toques entre objetos. Es posible, incluso, restringir esta detección a ciertos materiales, es decir, conseguir que el sensor solo se dispare cuando el objeto toque un cierto material.

Sensor Collision

Son los sensores de colisión entre objetos. Se puede conseguir que este sensor solo reaccione ante ciertos materiales o ante objetos con ciertas propiedades.

Sensor Near

Son los sensores de proximidad.

Sensor Radar

Es un sensor que actúa como un auténtico radar. Podemos limitarlo a objetos con una propiedad, limitar el ángulo de apertura o la distancia de alcance.

Sensor Property

Permite evaluar las propiedades del objeto, ofreciendo la posibilidad de ordenar acciones en función de los valores que dichas propiedades vayan tomando.

Sensor Random

Son sensores que se disparan de forma aleatoria, dependiendo de la semilla que se les indique.

Sensor Ray

Este sensor nos permite determinar la 'visión directa' del objeto. Es similar al sensor de radar, con la salvedad de que en éste se evalúa la interposición de objetos, mientras que en el radar no.

Sensor Messsage

Detecta la llegada de mensajes, ya sean provenientes del mismo objeto o de otros.

4.4.4. Controladores

Los controladores nos permiten establecer las reglas de decisión del GameBlender, permitiendo gestionar el comportamiento de los objetos dentro del mismo. Toma como entrada los sensores del objeto, y tiene como salida los actuadores (que los explicaremos a continuación).

Hay cuatro tipos de controladores:

Controlador AND

Debe estar conectado a uno, dos o más sensores del objeto. Todos estos sensores deben estar activos para que el controlador AND dispare el actuador asociado.

Controlador OR

Debe estar conectado a uno, dos o más sensores del objeto. Desde que se active uno de estos sensores, el actuador al que esté asociado será disparado.

Controlador Expression

Permite establecer condiciones complejas, combinando AND's, OR's y operadores de igualdad.

Controlador Python

Aquí es donde se encuentra la auténtica potencia del GameBlender. Este controlador permite activar python-scripts, en los cuales se puede implementar desde simples movimientos, hasta módulos de inteligencia artificial.

4.4.5. Actuadores

A través de los actuadores se ejecutan las distintas acciones. Si identificáramos los sensores con los sentidos y los controladores con el cerebro, los actuadores serían los músculos.

Motion

Permite desplazar, rotar, acelerar o frenar un objeto.

Constraints

Da la posibilidad de limitar la libertad de acción de un objeto.

IPO

Activa las curvas IPO (animaciones predefinidas) de un objeto.

Camera

Permite establecer un seguimiento a un determinado objeto, para que éste no salga del campo de visión y se mantenga a una distancia determinada.

Sound

Reproduce sonidos cargados en el GameBlender.

Property

Asigna un valor o el resultado de una expresión a una propiedad.

Edit

Permite realizar acciones tales como añadir o eliminar objetos.

Scene

Permite alternar entre escenas o cámaras en el GameBlender.

Random

Asigna a una propiedad un valor aleatorio.

4.4.6. TTS (TextToSpeech)

La gran expansión que ha experimentado en los últimos años el mundo de los ordenadores ha hecho que estos hayan pasado de ser utilizados por unos pocos (científicos, ingenieros,...) a convertirse en una herramienta habitual de trabajo en prácticamente cualquier profesión, además de un instrumento de ocio y cultura. Todo ello ha hecho evolucionar de forma vertiginosa la manera que tiene el ser humano de comunicarse con los ordenadores. Hemos pasado de cables y tarjetas perforadas a ratones, teclados, plotters, TPV's... El siguiente paso lógico es incorporar el método humano de comunicación por excelencia, la voz. Es aquí donde entran en juego las tecnologías de reconocimiento automático del habla (RAH) y las de Conversión de Texto a Voz (CVT o TTS). Nosotros nos centraremos específicamente en los sistemas TTS o CVT, que son los que hemos utilizado en la realización del proyecto.

Los sistemas TTS tienen multitud de aplicaciones en multitud de contextos, ofreciendo enormes ventajas: desde sistemas de lectura para invidentes o sintetizadores de voz para mudos, hasta asistencia telefónica o lectura de e-mails.

Es evidente que cuando hablamos de un sistema de estas características es fundamental que la voz sintetizada sea perfectamente entendible, y bastante recomendable que sea lo más natural posible. La reacción ante una viz mecánica y carente de emociones es totalmente distinta a la que se produce cuando nos encontramos ante una voz "humanizada".

Podemos obtener, por tanto, dos niveles de exigencia: un primer nivel, inexcusable para que el sistema cumpla su función, que es la inteligibilidad del habla; y un segundo nivel, en el que también es recomendable trabajar, y que lo hace más agradable de cara al usuario, como es la naturalidad en la voz.

En la conversión de texto a voz hay una distinción muy extendida entre parámetros **segmentales** y parámetros **suprasegmentales**.

- **Parámetros segmentales.** Son las unidades mínimas para caracterizar el habla, como puede ser el fonema. Parámetros de este tipo son las frecuencias y anchos de banda, la amplitud de los formantes en los diferentes sonidos o la frecuencia del cero nasal.

- **Parámetros suprasegmentales.** Son aquellos que afectan a las unidades de orden superior al segmento (sílabas, palabras, grupos fónicos, ...). La disciplina encargada del estudio de estos parámetros es la prosodia.

La inteligibilidad de la voz viene dada por los parámetros segmentales, pero una alta inteligibilidad no implica necesariamente una gran naturalidad. De hecho, la calidad de un sintetizador se obtiene en su mayor parte al actuar sobre los parámetros suprasegmentales, especialmente sobre tres de ellos:

1. La frecuencia fundamental. Es la curva que reprensenta la frecuencia de vibración de las cuerdas vocales, y se puede considerar como el parámetro que más influye en la naturalidad de la voz sintetizada.

2. Las duraciones. Tiene gran influencia tanto en la naturalidad como en la inteligibilidad de la voz, ya que permite la elocución y marcar el ritmo de la misma.

3. Las intensidades. Se suelen considerar como parámetros de segundo orden, y no se suelen utilizar en los sistemas actuales de conversión de texto a voz.

A continuación daremos algunos detalles de lo que es el esquema general de un conversor de texto a voz.

Estructura de un conversor de texto a voz.

Como su propio nombre indica, el objetivo fundamental de un CTV es pasar textos escritos a voz. Además, en función de la aplicación en la que se esté trabajando, tendrá unos requerimientos de naturalidad e inteligibilidad.

Desde el punto de vista de los mensajes a convertir, no es lo mismo si nos enfrentamos a mensajes predeterminados que si lo hacemos con un texto libre, sin restricción alguna. Atendiendo a este criterio los mensajes se suelen agrupar en cuatro categorías:

1. Mensajes Prefijados. Se representa mediante la asignación de una clave a cada mensaje.

2. Combinaciones limitadas de frases. Se representa mediante la asignación de una clave a cada frase.

3. Palabras en estructura fijada. Se representa como texto con características de entonación.

4. Estructura libre. Se representa como texto escrito.

Dependiendo del tipo de mensajes que se maneje, la naturaleza del habla puede variar entre los siguientes tipos:

1. Mensajes Pregrabados. Se utiliza con mensajes prefijados; se asigna una tabla de correspondencia uno a uno.

2. Concatenación de frases pregrabadas. Se utiliza cuando tenemos un conjunto predefinido de frases; se asigna una tabla de correspondencia uno a uno.

3. Reproducción de sonidos simples. Se utiliza con las palabras en estructura fija. Tenemos distintas representaciones de sonidos, y se elige cual es el que se va a reproducir.

4. Reproducción de sonidos simples y máxima flexibilidad prosódica. La utilizamos cuando nos enfrentamos a textos libres. Es de gran complejidad, ya que supone tener una representación para el mayor número posible de sonidos y características prosódicas posibles.

Es evidente la diferencia abismal que hay entre los puntos 1 y 2 y los puntos 3 y 4. Son estos últimos los que se pueden considerar como síntesis real de voz.

En todo proceso de conversión de texto a voz debemos distinguir dos fases fundamentales: el análisis del texto y la síntesis de la voz. Hagamos una pequeña introducción a cada una de ellas.

Análisis del Texto El análisis de texto se encarga de obtener la información que después se procesará en la fase de síntesis. generalmente se basa en cinco pasos fundamentales:

1. Detección de Palabras.

2. Asignación de Acentos.

3. Trascripción fonética.

4. Cálculo de valores prosódios para cada fonema.

5. Clasificación gramatical de palabras.

Síntesis de Voz Para la síntesis de voz necesitaremos almacenar segmentos de voz, tratar las transiciones entre estos segmentos y modificarlos para añadirles las características prosódicas necesarias.

La clasificación más utilizada para las técnicas de sintetizado está basada en el análisis de la forma en que manejan las transiciones y modifican la prosodia de cada sonido, dando lugar a dos categorías:

1. Síntesis concatenativa; utiliza tratamiento digital de la señal.

2. Síntesis por reglas; utliza reglas que definen la evolución de los parámetros y la prosodia.

Las tareas fundamentales a cumplir en esta fase son las siguientes:

- Obtención de unidades del habla y almacenamiento de las mismas.
- Concatenación.
- Modificación de la prosodia.

4.5. Aplicación a HEVAH

HEVAH es el resultado de la combinación de muchas de las técnicas explicadas anteriormente. Desde texturización y edición de materiales, hasta la incorporación del TTS para dotar de voz a nuestro personaje, pasando por los movimientos de cámara y la creación de claves para la animación, hemos tenido que familiarizarnos con multitud de nuevos conceptos.

Así, en texturización utilizamos texturas 2D, Mapeado UV y Alpha Blending; la animación la hicimos mediante fotogramas claves; para la iluminación utilizamos el modelo tradicional o clásico de iluminación en el cine; recurrimos a la utilización de planos cortos y panorámicos para destacar las animaciones faciales; y añadimos un TTS. HEVAH es el resultado de todo ello.

Capítulo 5

Decisiones de diseño en HEVAH.

En este capítulo haremos una breve descripción de HEVAH, sus características y las herramientas utilizadas en su desarrollo, así como las razones que nos llevaron a elegirlas. Además, especificaremos algunos conceptos vistos en capítulos anteriores y explicaremos como los hemos aplicado a HEVAH.

En primer lugar, señalar que Hevah es un personaje animado que nace como interfaz de un sistema de evaluación basado en tests adaptativos. El desarrollo de este personaje lo hemos dividido en distintos módulos o fases.

5.1. Modelado y Animación.

Para el modelado de nuestro personaje requeríamos de la utilización de un software especializado para diseño en 3d.

En un principio estuvimos manejando la posibilidad de utilizar soluciones propietarias, tales como Poser, 3DStudio o Maya, pero debido al alto coste de las licencias y a nuestra filosofía personal, decidimos buscar una herramienta libre y, a ser posible, multiplataforma.

Tras evaluar algunas otras alternativas que no nos ofrecieron garantías, optamos por Blender, un sotware de modelado y animación en 2D y 3D que cumplía con los requesitos necesarios; es más, no solo cumplía estos requisitos, sino que nos ofrecía otras características que nos resultaron muy interesantes, tales como la interacción con el personaje.

5.2. Interacción con HEVAH.

Este es uno de los puntos claves que nos hicieron decidirnos por Blender, ya que su motor de juegos o GameBlender nos brinda muy buenas prestaciones sin necesidad de recurrir a aplicaciones externas, utilizando la lógica que tiene incorporada.

Para el habla utilizamos una herramienta gratuita de Microsoft, el SAPI 4, ya que el resto de alternativas que manejamos presentaban distintos inconvenientes (no estaba implementado el español, no eran sistemas gratuitos...); esta herramienta proporciona una interfaz a través de clases para el Mircosoft TextToSpeech. En el momento del desarrollo se encontraba en la versión 5, pero no habían disponibles voces en español.

Aquí nos encontramos con bastantes dificultades para integrar el TTS y Blender. A la hora de la implementación de los script en Blender nos encontramos con algunos problemas derivados del funcionamiento del módulo de juegos. Esto hizo necesario la utilización de threading.

5.2.1. ¿Qué es el Threading?

La programación multihilo es una técnica alternativa de programación concurrente que se apoya en que nuestro sistema operativo nos proporcione hilos (threads).

En un S.O. multihilo nuestros programas pueden tener varios 'hilos de ejecución', es decir, estar realizando (aparentemente) varias tareas a la vez. El programador escribe partes de su programa que se ejecutan en paralelo, y a estas partes las denominamos hilos. Pero a diferencia de escribir varios programas (procesos) que se ejecuten a la vez colaborativamente, los hilos comparten la memoria (es decir, acceden a las mismas variables globales o dinámicas), por lo que no necesitan de costosos mecanismos de comunicación entre procesos para sincronizarse. Tampoco necesitan guardar gran parte de su contexto para preservar las variables, por lo que la penalización por cambio de contexto es menor.

Existen dos tipos de hilos: los hilos en el espacio del kernel y los hilos en el espacio de usuario. Los hilos del kernel son llamados a veces 'procesos ligeros', porque pueden verse como una versión simplificada de los procesos. Es el kernel el que se encarga de darles soporte y de planificarlos. En los hilos a nivel de usuario, en cambio, es una biblioteca la encargada de realizar la multiprogramación por hilos. Esta biblioteca se enlaza con el programa principal, y ella es la que se encarga de proporcionar el soporte a una concurrencia (por supuesto, simulada) mediante un planificador interno. Esta técnica tiene algunas ventajas, especialmente máquinas multiprocesador -la planificación de hilos en procesadores diferentes acarrea penalizaciones por el hecho de no poder utilizar los datos

de la memoria caché de cada procesador- y también algunas desventajas.

En los programas con varios hilos hay que tener cuidado para evitar el acceso simultáneo de dos hilos a la misma zona de memoria, lo que podría provocar corrupción de datos. Se usan 'semáforos' o 'secciones críticas' para 'proteger' un recurso. Esto es asegurar que sólo un hilo a la vez está en posesión del recurso protegido.

En la ejecución de hilos no podemos decir que haya simultaneidad, sino más bien intercalación. Hay varios flujos de programa, pero sólo uno de ellos está activo a la vez. El flujo activo completa alguna tarea y él mismo cede el control saltando a otra corrutina que sigue ejecutándose donde lo dejó la última vez.

La ventaja de los hilos es el acceso más cómodo a la información compartida. Tan cómodo como acceder a una variable global. Pero también es más inseguro. La programación concurrente es difícil. Es importante mantener el esquema sencillo, porque a la mínima complicación se pueden producir bloqueos mutuos y otras sorpresas.

A estas alturas podríamos preguntarnos: ¿para qué sirven en realidad?. Si el ordenador no tiene varios procesadores, ¿qué ventaja hay en hacer discurrir el programa por varios caminos a la vez?. Hay dos respuestas fundamentales. De cara a la velocidad, evita que un programa esté ocioso mientras espera a que un dispositivo lento complete una operación. Por ejemplo, nuestro programa tiene que leer desde el disco, la red o esperar respuesta del usuario. Desde que ha terminado de pedir al sistema esa lectura o esperar esa respuesta, y hasta que los datos solicitados llegan, el programa puede dedicarse a otras tareas. En la práctica el programa va más rápido porque no espera. Está todo el tiempo usando el procesador tanto como puede.

La otra respuesta es la sencillez para el programador. Mediante hilos podemos expresar algunos sistemas de forma más natural, más sencilla. Si llevas cierto tiempo programando es probable que, cuando estás planeando cómo comunicar dos objetos, no seas capaz de decidir quién llama a quién. Es más fácil si se hace separando en hilos la parte que capta la entrada de la que procesa la salida.

5.3. El Sistema de Evaluación

El Sistema de Evaluación de HEVAH está basado en tests adaptativos informatizados bayesianos. Para el almacenamiento de las estructuras de datos optamos por ficheros XML, lo cual supuso un enorme ahorro en complejidad de la infraestructura, ya que la otra alternativa que barajamos era el almacenamiento en bases de datos. En python existen módulos específicos para el tratamiento de documentos XML, que nos ha brindado enormes facilidades a la hora de trabajar con ellos.

El motor de inferencia que utilizamos es el CLIPS, un lenguaje específico de IA. Si bien en estos momentos el motor de inferencia es bastante simple, pensando en futuras ampliaciones del mismo consideramos que era oportuno implementarlo en un lenguaje específicamente diseñado para tales menesteres. Esto nos llevó a crear un túnel que nos permitiera la comunicación entre éste y Blender. Para ello hicimos uso de unas librerías denominadas win32api y de los módulos de gestión de procesos propios de python.

Hagamos una breve introducción a lo que es CLIPS y sus funcionalidades.

5.3.1. CLIPS (C Language Integrated Production System)

Clips representa un entorno completo para el desarrollo de sistemas expertos; incluye un editor de programas y herramientas de depuración. Fue creado en Johnson Space Center (NASA). Se diseño para facilitar el desarrollo de software que modele el conocimiento humano con propósitos específicos de alta portabilidad, bajo costo y facilidad de integración. Otra de sus virtudes es la independencia de la plataforma.

Clips distingue entre mayúsculas y minúsculas (es decir es case-sensitive). La versión actual de clips es la 6.1. Esta versión soporta distintos tipos de programación, siendo más explicitos: programación basada en reglas, procedural y orientada a objetos.

Los componentes básicos de clips son:

- la base de hechos (lista de datos introducidos e inferidos). Se entiende por hecho una de las formas básicas de representar información(conocimiento estático del dominio de la información
- la base de conocimientos (reglas, funciones,...). Es aquí donde se define la forma de actuación del sistema.
- el mecanismo de inferencias (que controla la ejecución)

Veamos ahora las distintas características:

- Esta diseñado para facilitar la integración con otros lenguajes
- Puede ser llamado desde otros lenguajes ya que clips puede ejecutar una función y retornar tanto el valor devuelto como el control del flujo.
- Asi mismo puede llamar a funciones externas que le devuelvan un valor y el control del flujo

La programación basada en reglas (o conocimiento heurístico)

Las reglas especifican las acciones a realizar en una situación dada. Es el desarrollador el encargado de introducir el conjunto de reglas en el sistema. Estas reglas en su forma mas simple tienen la siguiente estructura: 'si pasa X entonces haces Y '. Donde X sería el antecedente (o suceso) e Y el consecuente (o acción a realizar).

La programación procedural (funciones y objetos)

En clips también se permite la representación del conocimiento utilizando funciones definidas por el usuario y programación orientada a objetos. La programación orientada a objetos en clips permite las cinco características básicas: clases, mensajes, abstracción, encapsulamiento, herencia y polimorfismo.

Al realizar implementaciones en clips se pueden utilizar estas metodologías por separado o llevando a cabo una mezcla de ellas. La 'shell' de clips es la parte encargada de realizar inferencias o razonamientos y provee los elementos básicos de un sistema experto.

Se puede decir que un sistema experto basado en reglas escrito en CLIPS es un programa dirigido por los datos (data driven), es decir, hechos y objetos. Las reglas pueden casar con objetos y hechos, aunque los objetos también pueden usarse por sí solos (mediante el envío de mensajes) sin utilizar el motor de inferencia.

Capítulo 6

Decisiones de diseño en PORTAD.

A continuación haremos una breve descripción de las herramientas utilizadas a lo largo del desarrollo de PORTAD, el portal Web que complementa a HEVAH.

PORTAD surgió a partir de la necesidad de elaborar un interface para la creación y gestión de los tests adaptativos. Una vez elaboramos la parte de gestión, decidimos también desarrollar un módulo para la realización de tests, con lo que nos quedó una plataforma completa.

En el desarrollo de PORTAD podemos distinguir básicamente 3 módulos, el de contenidos, en el que utilizamos WebWare, el de diseño para el que utilizamos Cheetah y CSS, y el del motor de inferencia, que en este caso está escrito en Python.

6.1. WebWare

WebWare es una plataforma, para desarrollo web, de código abierto. El corazón de esta plataforma es un servidor al estilo del Tomcat de Java, pero para scripts escritos en python. Esta fue la principal razón que nos llevó a utilizarlo. Dado que los módulos de HEVAH ya implementados (tts, sistema de evaluación, ...) estaban desarrollados en python, el WebWare nos daba la posibilidad de reutilizar la mayor parte del código ya escrito. Además, es mucho más ligero que otros servidores similares (como el ya citado Tomcat).

6.2. Cheetah

Cheetah es un potente motor de plantillas y generador de código. Esto le da la posibilidad de ser utilizado en solitario o como complemento a otras herramientas. Puese ser

útil en muchos casos, pero suele ser utilizado como alternativas a ASP, JPS, PHP y PSP.

Algunas de las características de Cheetah que nos hicieron decantarnos por este lenguaje son las siguientes:

- Es capaz de generar HTML, SGML, XML, SQL, Postscripts, LaTeX y otro formatos basados en texto.
- Permite la separación de contenido, diseño gráfico y código. Esto nos brinda la posibilidad de tener sitios web con una arquitectura altamente modular, flexible y reutilizable y acelerar significativamente la velocidad de desarrollo, algo fundamental en nuestro caso.
- Combina la potencia y flexibilidad del python con un lenguaje de plantillas sencillo que los no programadores pueden entender.
- Permite acceder a cualquier estructura, módulo, función, objeto o método de python.

6.3. CSS

CSS es un lenguaje de estilos de presentación. Define colores, posiciones y fuentes. Facilita el mantenimiento de los documentos, desde el punto de vista de la presentación, pues lo que hace es separar las propiedades de presentación de documentos en una sola ubicación.

Una CSS consta de reglas que se aplican a elementos de un determinado tipo. Las CSS se pueden colocar directamente en el documento HTML o XML, o pueden colocarse en documentos de hojas de estilos externas con extensión .css.

La aplicación de las reglas de estilo, está determinada por selectores, que son construcciones de CSS para identificar partes de un documento HTML o XML. En síntesis, lo que hace un selector es crear un vínculo entre un elemento del documento y su determinado estilo.

6.4. Motor de Inferencia

Si bien en HEVAH nos decantamos por utlizar el CLIPS para la implementación del motor de inferencia, en el caso de PORTAD tuvimos que cambiar esta decisión debido a la naturaleza del Web.

Mientras que HEVAH es un sistema local, con un único usuario simultáneo, en el caso de PORTAD nos encontramos frente a un entorno web, en el que el número de usuarios que va a estar accediendo al mismo de forma simultánea puede llegar a ser bastante elevado (o al menos eso esperamos). Esto hace que sea inviable el lanzar un proceso de CLIPS por usuario, ya que la carga a la que se somete el sistema puede llegar a ser muy elevada. De hecho, en pruebas realizadas con 3 usuarios los tiempos de respuestas eran altamente insatisfactorios, por no decir que intolerables.

Esto nos llevó a implementar el motor de inferencia en python, e incluirlo en el propio servidor WebWare.

Capítulo 7

Implementación del test adaptativo

Con este test se pretende realizar una evalución del alumno de forma dinámica, es decir, se van evaluando sus respuestas y en consecuencia con las mismas se incrementa o decrementa la dificultad de las preguntas, con ello, se pretende la mayor aproximación posible al conocimiento del alumno. Para la realización del test adaptativo utilizamos técnicas de inferencia bayesiana, que nos ayudan considerablemente a conseguir el objetivo mencionado anteriormente.

Recordemos, antes que nada, la estructura de los ficheros XML que contienen los test. Un test está dividido en temas, a su vez estos temas están divididos en conceptos, éstos en preguntas y cada pregunta tiene asociada un número variable de respuestas. Para los conceptos distinguimos cuatro niveles de complejidad: simple, intermedio, complejo y avanzado. Así mismo, cada pregunta lleva asociada dos probabilidades, la probabilidad de saber la respuesta a dicha pregunta y la probabilidad de adivinarla (o lo que es lo mismo, acertar la respuesta por azar). Convertimos estas probabilidades en conjuntos de inferencia bayesiana similares a los de los conceptos. Para ello sumamos ambas probabilidades obteniendo la probabilidad total de saber dicha pregunta. Suponiendo que dichas probabilidades son independienes entre si, tendríamos una primera formula:

$$P(acertar) = P(saber) + P(adivinar)$$

Agrupando el resultado distinguimos tres niveles de dificultad para las preguntas: baja, media y alta.

Con respecto al usuario también distinguimos distintos conjuntos bayesianos en función de sus conocimientos. Los conjuntos son: novel, intermedio, avanzado y experto. Este nivel varía a medida que el usuario va realizando los test. A los usuarios que no han realizado ningún test se les asigna un nivel por defecto de novel.

Llegados a este punto parece necesario resumir los distintos conjuntos difusos con lo que vamos a trabajar:

- *Conceptos*: simple, intermedio, avanzado y complejo
- *Preguntas*: baja, media, alta
- *Usuarios*: novel, intermedio, avanzado y experto

7.1. Evaluación del test

Veamos cual es el proceso que sigue el sistema para la realización del test.

En primer lugar se establecen las probabilidades iniciales, que determinan el conocimiento que tienen los distintos conjuntos de usuarios sobre los distintos conjuntos de dificultad de los conceptos, es decir, para cada par 'nivel de usuario'-'dificultad del concepto' tendremos una probabilidad inicial.

Una vez establecidas estas probabilidades pasamos a seleccionar el primer concepto del test que va a ser evaluado. Los conceptos se encuentran agrupados según su nivel de dificultad. El concepto que será presentado al usuario está en relación directa con su nivel de la siguiente forma:

Nivel del Usuario	**Nivel del Concepto**
Novel	Simple
Intermedio	Intermedio
Avanzado	Avanzado
Complejo	Complejo

De esta forma obtenemos el nivel del primer concepto. Del conjunto de conceptos de este nivel se elige uno al azar.

Para evaluar este concepto debemos saber, a priori, la probabilidad que tiene el usuario de saberlo. Esta es una de las probabilidades inicialmente establecidas.

A continuación debemos seleccionar el nivel de dificultad de la pregunta que se le va a presentar al usuario de las disponibles para ese concepto. Por defecto se establece dificultad 'Media'. De las preguntas de ese nivel, al igual que los conceptos, seleccionamos una al azar.

Durante la selección de preguntas puede suceder que un concepto no disponga de más preguntas del nivel requerido,en cuyo caso, se pasaría a un nuevo concepto de la misma dificultad que el anterior con el fin de conseguir una pregunta del nivel deseado.

Una vez tenemos la pregunta a realizar se le presenta al usuario junto con las respuestas asociadas a la misma. Cuando el usuario responde se pasa al proceso de evaluación de la respuesta, el cual dará como resultado el nuevo nivel del usuario y el nivel tanto de la nueva pregunta como del nuevo concepto. Veamos en detalle este proceso.

Una vez que el usuario responde comprobamos si la respuesta es correcta o no. Si la respuesta es correcta, actualizamos la probabilidad que tiene el usuario de saber al concepto de la siguiente forma:

$$P(acertar) = P(saber) * P(inicial) + P(adivinar) * (1 - P(inicial))$$

Donde:

- $P(acertar)$: probabilidad que tenía de acertar la pregunta o probabilidad a priori de acertar
- $P(saber)$: probabilidad que tenía el usuario de acertar la pregunta sabiendo el concepto
- $P(inicial)$: probabilidad inicial que tiene el usuario de saber el concepto
- $P(adivinar)$: probabilidad de acertar la pregunta sin saber el concepto

$$P(saber\_concepto) = \frac{(P(inicial) * P(saber))}{P(acertar)}$$

Donde:

- $P(saber\_concepto)$: es la nueva probabilidad de saber el concepto

Si la respuesta es incorrecta, actualizamos la probabilidad que tiene el usuario de saber al concepto de la siguiente forma:

$$P(fallar) = (1 - P(saber)) * P(inicial) + (1 - P(adivinar)) * (1 - P(inicial))$$

Donde:

- P(fallar): probabilidad que tenía de fallar la pregunta o probabilidad a priori de fallar

$$P(saber\_concepto) = \frac{(P(inicial) * (1 - P(saber)))}{P(fallar)}$$

Estas probabilidades 'P(saber_concepto)' se van almacenando con el fin de calcular el nivel de conocimiento que tiene el usuario sobre el concepto en cuestión. Para calcular este conocimiento hacemos la media de las probabilidades calculadas hasta el momento:

$$Conocimiento = \frac{P(saber\_concepto)_{1an}}{n}$$

Una vez que tenemos el conocimiento actual del usuario, el nivel de la pregunta actual y el nivel del concepto actual, pasamos a buscar la nueva pregunta. Esto se hace atendiendo a las siguientes reglas:

Primero calculamos si se aumenta o disminuye el nivel

- Si probabilidad_saber_concepto = baja $\Longrightarrow$ disminuir_nivel
- Si probabilidad_saber_concepto = alta $\Longrightarrow$ aumentar_nivel
- Si probabilidad_saber_concepto = media $\Longrightarrow$ (nueva pregunta del mismo concepto y del mismo nivel) o (nuevo concepto del mismo nivel)

Si debemos disminuir el nivel

- Si (nivel_pregunta = baja) y (nivel_concepto = simple) y (disminuir_nivel) $\Longrightarrow$ (pregunta de nivel bajo del mismo concepto) o (nuevo concepto simple)
- Si (nivel_pregunta = baja) y (nivel_concepto = intermedio) y (disminuir_nivel) $\Longrightarrow$ (nuevo concepto simple)
- Si (nivel_pregunta = baja) y (nivel_concepto = avanzado) y (disminuir_nivel) $\Longrightarrow$ (nuevo concepto intermedio)
- Si (nivel_pregunta = baja) y (nivel_concepto = complejo) y (disminuir_nivel) $\Longrightarrow$ (nuevo concepto avanzado)
- Si (nivel_pregunta = media) y (disminuir_nivel) $\Longrightarrow$ (nueva pregunta del mismo concepto de nivel bajo)
- Si (nivel_pregunta = alta) y (disminuir_nivel) $\Longrightarrow$ (nueva pregunta del mismo concepto de nivel medio)

Si debemos aumentar el nivel

- Si (nivel_pregunta = alta) y (nivel_concepto = simple) y (aumentar_nivel) $\Longrightarrow$ (nuevo concepto intermedio)

- Si (nivel_pregunta = alta) y (nivel_concepto = intermedio) y (aumentar_nivel) $\Longrightarrow$ (nuevo concepto avanzado)
- Si (nivel_pregunta = alta) y (nivel_concepto = avanzado) y (aumentar_nivel) $\Longrightarrow$ (nuevo concepto complejo)
- Si (nivel_pregunta = alta) y (nivel_concepto = complejo) y (aumentar_nivel) $\Longrightarrow$ (pregunta de nivel alto del mismo concepto) o (nuevo concepto complejo)
- Si (nivel_pregunta = media) y (aumentar_nivel) $\Longrightarrow$ (nueva pregunta del mismo concepto de nivel alto)
- Si (nivel_pregunta = baja) y (aumentar_nivel) $\Longrightarrow$ (nueva pregunta del mismo concepto de nivel medio)

Como podemos observar estas reglas determinaran cual será el nivel del siguiente concepto o la siguiente pregunta que se va a realizar. Debemos destacar varias cosas:

- Si se produce un cambio de concepto el nivel de la pregunta no varía, es decir, si tenemos una pregunta de nivel medio y pasamos de un concepto simple a un concepto intermedio, el nivel de la pregunta que se realizará seguirá siendo medio. Esto es asi ya que no tenemos datos en ese momento para evaluar al usuario en conceptos de otro nivel.
- Si la probabilidad de saber el concepto es media, es decir, no esta muy claro si sabe o no sabe el concepto, simplemente realizamos otra pregunta del mismo nivel y del mimo concepto, o en su defecto, otra pregunta del mismo nivel de otro concepto del mismo nivel. Esto es asi porque en este caso aún no somos capaces de valorar el conocimiento del usuario sobre ese concepto.
- Si el nivel de la pregunta que estamos evaluando es bajo, el nivel del concepto es simple y vemos que el usuario no lo sabe pasamos a presentarle una nueva pregunta del mismo nivel del mismo concepto o en su defecto determinamos que no sabe el concepto y pasamos a uno nuevo del mismo nivel.
- Para el resto de los casos en los cuales se considere que el usuario no sabe el concepto o bien se disminuye el nivel de la pregunta o bien se disminuye el nivel del concepto, con el fin de aproximarnos lo más posible a los conocimientos de dicho usuario en ese momento.

- Si el nivel de la pregunta que estamos evaluando es alto, el nivel del concepto es complejo y vemos que el usuario lo sabe pasamos a presentarle una nueva pregunta del mismo nivel del mismo concepto o en su defecto determinamos que el usuario sabe el concepto y pasamos a presentarle un nuevo concepto del mismo nivel.

- Para el resto de los casos en los cuales se considere que el usuario sabe el concepto o bien se aumenta el nivel de la pregunta o bien se aumenta el nivel del concepto, con el fin de aproximarnos lo más posible al nivel del usuario.

Al margen de la selección de la nueva pregunta también se actualiza la 'tabla de resultados' del alumno. Esta tabla es una matriz cuyas filas son los distintos niveles de los conceptos y cuyas columnas son los distintos niveles de las preguntas. De esta forma sabremos, una vez finalizado el test, los porcentajes de aciertos que ha tenido el usuario en función de la dificultad tanto de las preguntas como de los conceptos. Así mismo esta tabla resulta fundamental para realizar la evaluación que se explicará posteriormente con los resultados obtenidos en el test.

De esta forma se van presentando al alumno preguntas sucesivamente y evaluando sus respuestas, para determinar su nivel de conocimiento sobre cada uno de los conceptos. El test finaliza por varias razones:

- No hay más preguntas que realizar al usuario, ya que podría suceder que nesecitáramos más preguntas de un determinado nivel y el test no disponga de ellas. No podríamos optar por presentar preguntas de otro nivel ya que se rompería el fujo del test.

- No hay más conceptos que evaluar, al igual que antes podría suceder que se agotaran los conceptos de una determinada dificultad, lo cual nos obligaría a parar el test.

- Se llega a un límite de 20 preguntas, ya que, a pesar de que la naturaleza del test adaptativo sea presentar el menor número posible de preguntas para determinar el nivel de conocimiento del usuario, éstas pueden ser excecivas. Por las pruebas que hemos echo y observando el proceso de evaluación consideramos un límite adecuado el de las 20 preguntas, ya que evitamos el agotamiento o la saturación del usuario y favorecemos su capacidad de atención y el tiempo de respuesta.

7.2. Valoración de los resultados

Veamos un ejemplo de probabilidades a priori entre el nivel de los usuarios y la dificultad de los conceptos.

	Simple	**Intermedio**	**Avanzado**	**Complejo**
Novel	0 a 6	0 a 4	0 a 2	0 a 2
Intermedio	0 a 4	0 a 4	0 a 2	0 a 2
Avanzado	0 a 2	0 a 3	0 a 4	0 a 1
Experto	0	0 a 2	0 a 4	0 a 4

Como podemos observar en la tabla, un usuario, dependiendo de su nivel, tendrá más o menos probabilidad de saber un concepto según la complejidad que este posea. La valoración final para un conjunto de usuarios esta entre 0 y 10, con la excepción de los dos primeros niveles a los que les hemos otorgado 'premio' por saber conceptos que en principio deberían estar fuera de su alcance.

Ahora debemos determinar cual es la puntuación para cada usuario según sus respuestas a cada uno de los conjuntos de los conceptos (dicho gráficamente, cómo rellenamos la tabla anterior). Para ello vemos la siguiente tabla.

Dificultad Preguntas	**Fallos**	**Aciertos**	**Valoración**
Baja	X1	Y1	20 %
Media	X2	Y2	30 %
Alta	X3	Y3	50 %

Como podemos observar el valor que se le asigna a las preguntas esta relacionado directamente con su complejidad, más concretamente cuanto más compleja sea la pregunta más porcentaje tendrá su valoración. Estos porcentajes se determinan de forma directa al número de aciertos, es decir, si un usuarios acierta todas las preguntas de dificultad baja que se le plantean tendrá un 20 %, que se sumará al resto de los porcentajes obtenidos en los otros conjuntos de preguntas.

Veamos un ejemplo:

A un usuario 'U' se le plantean 3 preguntas de dificultad baja, 3 de dificultad media y 2 de dificultad alta. Este usuario acierta todas la de dificultad baja, dos de dificultad media y ningúna de dificultad alta.

Atendiendo al enunciado el usuario 'U' tendría un 20 % por las preguntas de dificultad baja, un 20 % de las preguntas de dificultad media y un 0 % por las preguntas de dificultad alta. En total tendría un 40 %.

Si suponemos que el nivel del usuario es novel y el concepto al que pertenecen las preguntas es simple, el usuario 'U' tendría una puntuación de 2.4, a la que habría que sumarle la de los conceptos: itermedio, avanzado y complejo; obteniendo así la puntuación final.

7.3. Tratamiento de los datos

Como ya hemos comentado, este pretende ser un proyecto multiplataforma y abierto; con ello en mente elegimos la tecnología XML como forma de alamacenamiento de datos, ya que cumple ambos objetivos. Si bien pensamos que es de sobra conocida, hemos creído oportuno hacer aquí una breve descripción de esta tecnología.

XML es una tecnología muy sencilla que tiene a su alrededor otras tecnologías que la complementan y la hacen mucho más potente

La principal novedad que aporta XML al mundo del tratamiento de los datos es la filosofía de compartir los datos con los que se trabaja a todos los niveles, de tal forma que estos puedan ser entendidos por cualquier aplicación o soporte. Es una tecnología que permitirá compartir la información de forma fácil, segura y fiable, consiguiendo con ello avanzar hacia la compatibilidad de los sistemas.

Otra de las ventajas es que permite al programador y los soportes dedicar sus esfuerzos a las tareas importantes cuando trabaja con los datos, ya que algunas tareas tediosas como la validación de estos o el recorrido de las estructuras corre a cargo del lenguaje y está especificado por el estándar, de modo que el programador no tiene que preocuparse por ello.

Capítulo 8

Implementación de HEVAH

En este capítulo explicaremos el desarrollo de HEVAH, nuestro personaje virtual. Comentaremos las técnicas y herramientas utilizadas, haciendo hincapié en aquellos puntos que consideramos claves.

8.1. Modelado y animación

Este es uno de los puntos que nos llevó más tiempo. A continuación iremos describiendo los elementos que conforman el entorno de HEVAH y las técnicas que hemos utilizado en su implementación.

Cabe señalar que el modelo inicial de la cabeza no fue realizado por nosotros. Fue extraido de un fichero de ejemplo encontrado en Internet. Lamentablemente, no disponemos de los datos del autor para darle crédito en esta memoria. Este modelo inicial se corresponde únicamente con la malla de la cabeza. Decidimos utilizar este modelo debido a nuestras limitaciones en el campo de las Bellas Artes.

Sin embargo, la malla es una parte ínfima del proyecto, durante esta sección se verá el amplio trabajo que se realizó en el modelado y la animación.

8.1.1. Los ojos

Globo Ocular

Se suele decir que los ojos son el reflejo del alma. Dan mucha vida al personaje, por lo cuál desde el principio decidimos seguir este turorial[1].

[1]http://www.nicodigital.com/tutorial/Creacion de un ojo estilo Pixar/BlenderChar-Creacion de un ojo estilo Pixar.htm

En este tutorial se explica el procedimiento a utilizar para obtener unos ojos similares a los que crea la empresa Pixar para los personajes de sus películas.

Este tipo de ojos se caracteriza por dar mucha personalidad al carácter. Esto se consigue mediante la profundidad.

La profundidad en la mirada es fácil de simular. Para ello se crea el ojo a partir de 4 partes, como se muestra en la siguiente imagen:

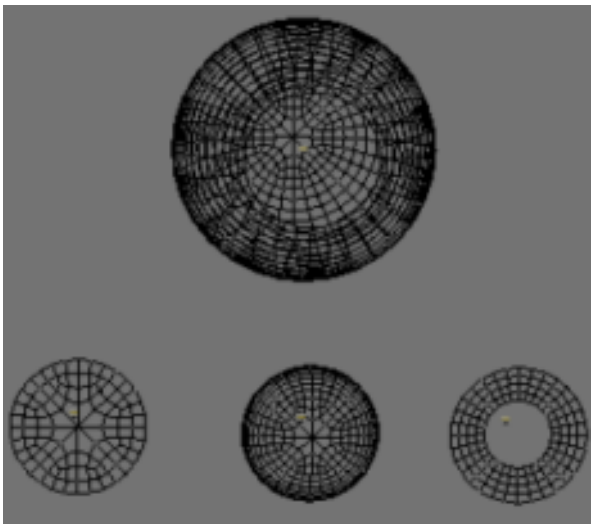


Figura 8.1: Partes del ojo

Estas 4 partes son el globo ocular, la córnea, el iris y la pupila.

Para crear las partes simplemente partimos de una esfera en Blender. Le separamos una parte. La parte más grande resultante sería el globo ocular, la parte más pequeña será utilizada para crear el resto. Duplicamos esta parte pequeña y le quitamos la parte central. Con esto obtenemos el iris.

Duplicamos el objeto e invertimos la curvatura. Con esto obtenemos la cornea. El objeto que nos queda es la pupila.

A la hora de darle color a cada uno de los elementos, a diferencia del tutorial, no nos complicamos. Esos efectos de color que se muestran en el tutorial no son apreciables en el módulo de juegos. Con lo cual, simplemente pintamos cada una de las partes con el color adecuado. Este proceso lo realizamos mediante un 'vertexpaint', o sea, pintando 'a mano' cada cara del ojo.

Una vez pintado el objeto, unimos las 4 partes para que sean una (CTRL + J).

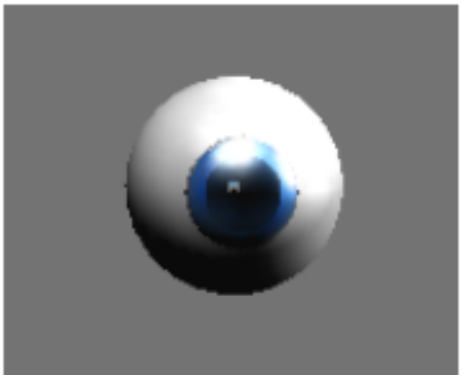


Figura 8.2: Unión de las partes del ojo

Cejas

Para la realización de las cejas del personaje, simplemente se realizaron una serie de 'extrudes' hasta la obtención del resultado adecuado.

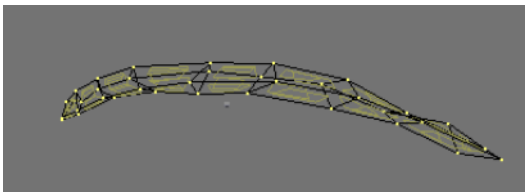


Figura 8.3: Malla de las cejas

Para darle color, aplicamos un material de color negro sin texturas (siguiendo los pasos adecuados para su correcta visualización en el módulo de juegos).

Figura 8.4: Ceja texturizada

Finalmente, para hacer que se mueva junto con la cabeza, realizamos el parentizado de turno.

Pestañas

Por supuesto, toda cabeza que se precie debe tener unas pestañas para darle mayor realismo. A la hora de modelar las pestañas nos encontramos con el problema de que había bastantes técnicas y ninguna de ellas era demasiado atractiva.

A continuación describiremos las alternativas que utilizamos.

El primero de los métodos que estudiamos fue el de partículas; este método para hacer pestañas da unos resultados impresionantes, como se puede ver en la siguiente imagen:

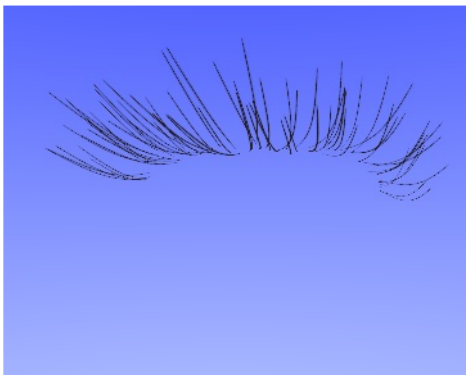


Figura 8.5: Pestañas con partículas

Sin embargo, ni nos lo planteamos, debido a que no funciona en el módulo de juegos. Está pensado para realizar renderizado y animación.

También cabía la posibilidad de modelar los pelos individualmente. Esto se haría creando cada pelo como una unión de segmentos. Esta técnica fue descartada por ser demasiado laboriosa y no garantizar unos resultados óptimos.

Por último, tenemos la solución que hemos adoptado: plano con canal alpha.

A continuación describiremos la técnica que utilizamos:

Realización de la malla

- En primer lugar se inserta un plano.
- Tras pasar a modo edición se seleccionan dos de los vértices del plano.
- Una vez tenemos la selección hecha, arrastramos con el ratón hasta conseguir un rectángulo del grosor aproximado de una pestaña.

- Llegados a este punto tendremos seleccionados todos los puntos. Ahora subdividiremos la malla para poder darle curvatura a la pestaña. Para subdividir la malla pulsamos repetidas veces la tecla W y seleccionamos la opción 'Subdivide'.

Obtener la imagen Para obtener la imagen hemos utilizado el programa para la manipulación de imagenes GIMP. La idea es crear una imagen con lineas negras (las pestañas) y el resto transparente.

- Una vez dentro del GIMP generamos un fichero nuevo (Archivo-Nuevo).
- Tras esto, rellenamos todo la imagen de color negro (Mayúsculas + B). Luego generaremos las partes transparentes.
- Hacemos visible (si es que no lo es) la ventana de 'Capas' (CTRL + L).
- Hacemos click derecho sobre 'Fondo' y seleccionamos 'Añadir canal alfa'
- Hacemos click derecho de nuevo sobre 'Fondo', pero esta vez seleccionamos 'Añadir máscara de capa'.
- La máscara nueva deberá ser 'Blanca (Opacidad total)'
- Ahora debemos pintar con el lápiz (N) la zona que va a ser transparente. O sea, el espacio entre pestaña y pestaña.
- Una vez pintada toda la transparencia, hacemos click derecho otra vez sobre 'Fondo' (en la ventana de 'Capas'). En esta ocasión seleccionamos 'Aplicar máscara de capa'.
- El último paso es guardar el fichero como TGA. Para ello deberemos hacer click derecho sobre la imagen y seleccionar 'Guardar como...'. Como extensión se debe elegir 'TGA'. El resto de las opciones son las que vienen por defecto. Guardamos TGA porque es el formato que reconoce Blender.

Integrar la imagen en el Blender

- Se seleciona el plano de las pestañas.
- Se presiona 'F' para pasar al modo caras.
- En la ventana de visualización de imágenes seleccionamos el fichero generado por el GIMP.

- Tras esto solo nos queda ir a los 'Paint buttons' y seleccionar el botón 'Alpha'. Esto hace que Blender interprete la imagen como una imagen con transparencias.

Curvar la malla

- El objetivo es darle curva a la malla para obtener el efecto de rizado de las pestañas.
- Para ello situamos la cámara en una vista lateral y seleccionamos sucesivamente las agrupaciones de puntos de los extremos de las pestañas.
- Una vez seleccionado el grupo de puntos, situamos el cursor del Blender para conseguir una rotación adecuada.
- Tras esto pulsamos R para rotar los puntos y obtener el efecto de rizado.

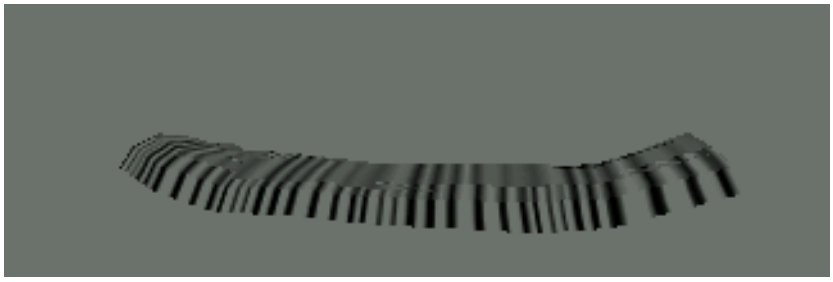


Figura 8.6: Pestañas definitivas

Módulo de juegos A la hora de integrarlo en el módulo de juegos tuvimos que hacer que se moviera junto con el párpado a la hora de parpadear. Existe el problema añadido de que el movimiento de los párpados se consigue mediante RVKs, con lo cual no podemos parentizar las pestañas con el párpado, ya que a pesar de que se anime el párpado, las pestañas no se moverán (lo harían si la animación fuera de rotación o posición). En principio sólo nos interesaba que las pestañas tuvieran dos posiciones: abierto y cerrado el ojo. Para hacer esto duplicamos las pestañas. Pusimos unas pestañas para ojo abierto y otras para ojo cerrado. Al principio sólo están activas las de ojo abierto. Cuando se cierra el ojo, se manda un mensaje mediante código para que las pestañas para ojo abierto se oculten y las otras se muestren. Al abrir de nuevo el ojo, se realiza el proceso inverso.

En la imagen final se muestran las pestañas de 'ojo cerrado' también. Al ejecutar en el módulo de juegos, lógicamente, no se verían sino dos pestañas por ojo, las convenientes en cada momento.

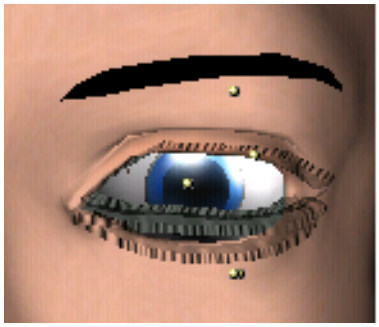


Figura 8.7: Pestañas integradas con el ojo

8.1.2. El Pelo

El pelo es uno de los elementos que das más realismo a un personaje animado. Pero a la vez, es una de la partes más complicadas de abordar. Un ejemplo lo encontramos en la superproducción cinematográfica 'Final Fantasy', cuyo presupuesto ronda los 100 millones de dólares. Gran parte del presuesto se destinó a la creación del pelo de la protagonista. Según los propios responsables del proyecto, lo más costoso y difícil de todo fue reproducir la cabellera lacia de la joven, con el fin de que tuvieran movimiento independiente los 60.000 pelos que la adornan. Nos hubiera gustado acercarnos aunque sea mínimamente al realismo de esta película pero nuestro presupuesto queda a alguna distancia del de 'Final Fantasy' (unos 100 millones de dólares).

Por lo tanto decidimos escoger alguna técnica más económica en tiempo y dinero. Las posibilidades que nos encontramos en Blender fueron escasas y de díficil implementación.

Primeras tentativas

Pelo estático Nuestra primera idea, por cuestión de simplicidad fue crear una cabellera estática. O sea, una malla se encargaría de modelar el pelo. Para dar mayor credibilidad a la falta de movimiento, hicimos pruebas con pelo corto, que consistían en generar la malla en base a extrusiones que nos permitieran crear una estructura que se ciñera a la cabeza. Para realizar esto, seguimos este tutorial[1].

Por supuesto, esta técnica fue descartada casi desde un principio. Se generaba un pelo muy poco realista y poco estético. Además, la falta de movimiento le restaba vistosidad.

[1]http://www.3dtotal.com/ffa/tutorials/max/joanofarc/hair1.asp

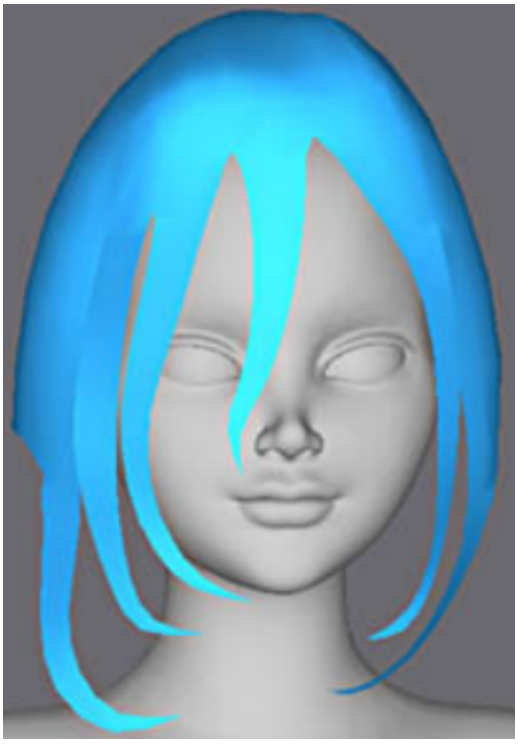


Figura 8.8: Cabellera Estática

Fiber Ante los desalentadores resultados de la prueba anterior, decidimos utilizar un plugin para Blender que nos ayudara a tal efecto. El plugin en cuestión está realizado en python, y se llama Fiber[1]. La versión que probamos es la 2.0.

El programa en principio da la opción de generar estructuras similares al pelo (hierba, pelo, etc). Se deberían poder conseguir buenos resultados, pero al llevarlo a la práctica, la cosa cambia.

Los problemas venían cuando queríamos aplicar la creación del pelo a nuestro tamaño de cabeza. Primero intentamos generar una estructura de pelo a medida para nuestra cabeza mediante el programa Fiber. El programa permite cambiar una serie de parámetros que se muestran en un menú. Se podían cambiar el grosor del pelo, el espesor, el estilo, etc.

Mediante las líneas de guía definimos la estructura general del pelo. Estas guías eran unas líneas finas que determinaban la dirección y longitud del cabello.

El resultado final obtenido no era para nada óptimo. La cabellera que se generaba no se ajustaba perfectamente a la cabeza. Con lo cual tuvimos que retocarla usando Blender

[1]http://www.elysiun.com/forum/viewtopic.php?t=14867&highlight=makehair

Figura 8.9: Fiber (lateral)

para eliminar las zonas de pelo que no eran necesarias.

Figura 8.10: Fiber (posterior)

A pesar de todas estas pegas, lo que nos impulsó definitivamente a deshechar esta idea fue el exagerado tamaño de los ficheros de pelo generados. Además, para conseguir unos resultados dignos debíamos darle mucha frondosidad al pelo, lo cual redundaba negativamente en el peso de los ficheros creados. Esto provocaba una ralentización más que notable en el Blender.

Bola de pelo dinámico (FakeFur) Buscando en el foro de http://www.elysiun.com, encontramos un fichero .blend que sería crucial a la hora de elaborar un pelo creíble.

En este fichero se mostraba un cubo con una capa de pelo que además se movía junto con el cubo dando una gran sensación de realismo. Además, lo mejor de todo es que funcionaba en el módulo de juegos.

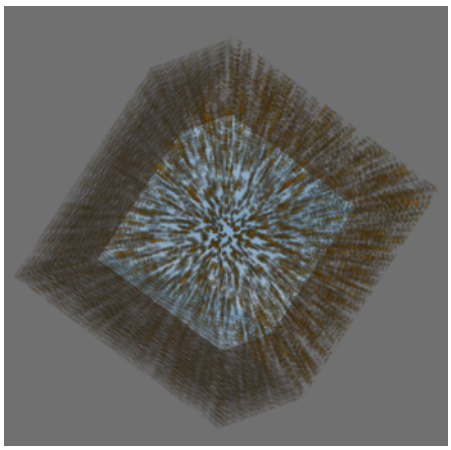


Figura 8.11: FakeFur

El efecto se consigue en este caso concreto con varias capas de cubos con una textura alpha aplicada. O sea, una textura semi-transparente, con algunos puntos opacos que serán los encargados de simular el pelo. Cada una de las capas posee un 'slow parent' distinto respecto del cubo central.

Cada una de las capas tiene una relación de parentesco (parent) con el cubo central, pero esta relación es débil (slow). Con esto se consigue que cada una de las capas se mueva después de moverse el cubo central.

Cada capa posee un parámetro de desplazamiento (offset) que determina cuanto tarda en moverse después de que lo haya hecho el cubo central. Para conseguir la sensación de que el cubo está rodeado de pelo, damos valores de offset distintos a cada capa. Los valores serán crecientes hacia el exterior de la estructura.

De manera similar al pelo real, lo primero que se moverá será la parte del cabello más cercana a la raíz (offset bajo) y lo último en moverse será la punta del cabello (offset alto).

Como ya comentamos, está técnica da mucho realismo al moviento. Sin embargo, la idea de utilizar mapas de puntos con canal alpha no nos sirve de mucho, ya que solo es válida para crear estructuras muy regulares de pelo, debido a que el cabello debería tener siempre el mismo tamaño, independientemente de la zona de la cabellera que tratáramos. Otro inconveniente es la cantidad de clareas que se verían en el pelo.

En cualquier caso, es una idea muy ingeniosa. Desgraciadamente no se encuentra muy bien documentado el fichero, por lo cual desconocemos el nombre del autor.

Solución definitiva: malla con 'slow parent'

Tras hacer todas estas tentativas y 'perder' mucho tiempo buscando y probando, se nos ocurrió una idea: mezclar las dos mejores ideas que habíamos visto, esto es, modelar el pelo con una malla (más sencillo, se consigue un pelo con la forma que deseamos y además el fichero resultante ocupa poco); y el 'slow parent' (esta técnica nos permite dar sensación de movimiento a la malla creada).

La idea a seguir, pues, era la siguiente:

1. Creación de la malla que representa el pelo siguiendo el tutorial de creación de pelo estático
2. División del pelo por 'alturas'
3. Asignación de un offset distinto a cada 'altura' del pelo.

A continuación detallamos el proceso.

Creación de la malla Para la creación de la malla que representa el pelo empezamos de cero siguiendo el tutorial disponible en este enlace[1].

Creamos únicamente uno de los lados de la cabellera. Posteriormente duplicamos para obtener una cabellera completa.

Para extender la malla simplemente usamos la función 'extrude'. Al terminar de modelar la malla, y para que no quedara tan pixelada subdividimos las caras para cuadruplicar el numero de estas.

Para dar mayor sensación de volumen al pelo duplicamos 2 veces la cabellera. A cada una de las dos copias le redujimos el volumen para crear sensación de espesor.

División de la malla por alturas Para cada una de las 3 capas de pelo, dividimos (separamos) la malla en distintas alturas. Cada altura está relacionada con el tiempo que tardaría en responder a un movimiento de la cabeza. Procuramos incluir dentro de la misma 'altura' aquellas parte que tardarían el mismo tiempo en desplazarse. En la figura en la que se muestra la mitad de la cabellera, podemos ver seleccionada una de las 'alturas' del pelo.

Por último, le dimos un color negro al pelo, ya que era una manera sencilla de conseguir una visión realista del mismo. Para darle este color, simplemente modificamos el color del material. Este es el resultado de la malla final (en modo rejilla, sin color):

[1] http://www.3dtotal.com/ffa/tutorials/max/joanofarc/hair1.asp

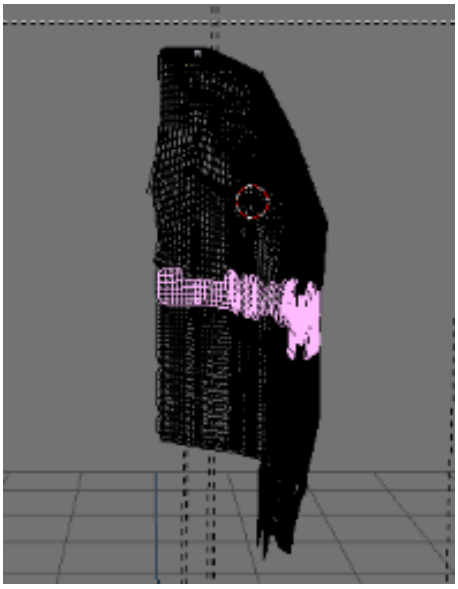


Figura 8.12: División por alturas

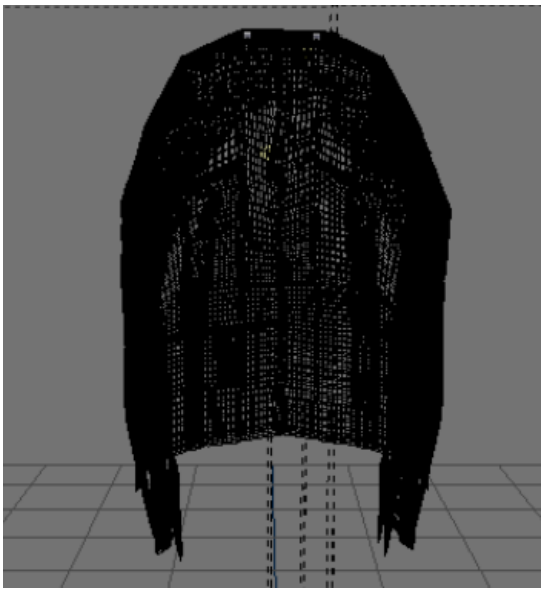


Figura 8.13: Malla del Pelo

Asignación de los offset Sólo nos quedaba dotar de movimiento a esta bonita cabellera. Para ello recurrimos al 'slow parent'.

Para cada una de las alturas del pelo, activamos la opción de 'slow parent' y fuimos incrementando el offset a medida que nos íbamos alejando de la raíz del pelo.

Finalmente, para probarlo añadimos un botón que hiciera girar la cabeza en el módulo

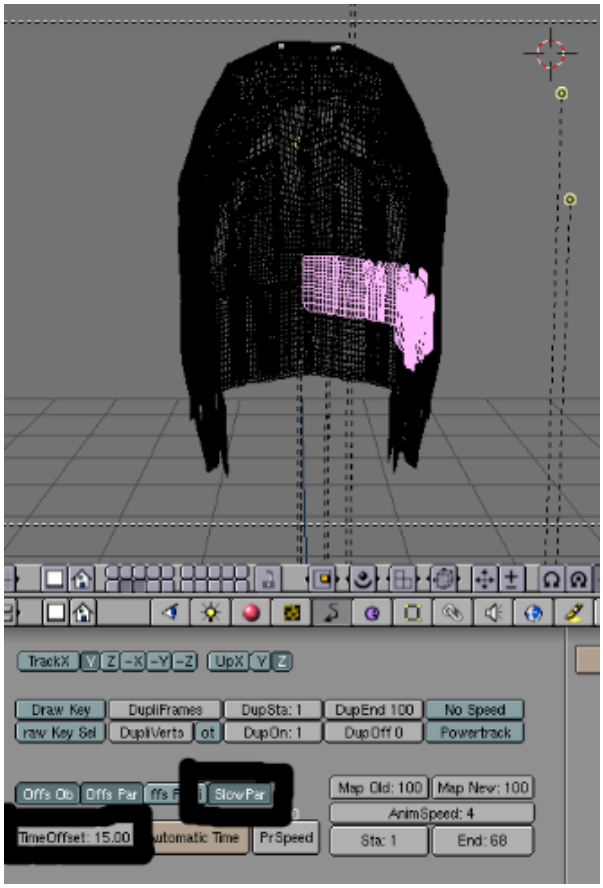


Figura 8.14: Asignación de los Offset

de juegos. El resultado obtenido fue muy satisfactorio, y es el que se aprecia en el proyecto.

8.1.3. Escenario

El escenario utilizado se compone de las partes básicas que pasamos a detallar

Mesa

La mesa sirve de soporte para la cabeza. Está situada en el interior de una habitación.

Para realizar la mesa, modelamos dos partes básicas: la parte inferior y la parte superior de la mesa. Ambas son cubos modificados para obtener las dimensiones deseadas. Se pueden observar en estas figuras

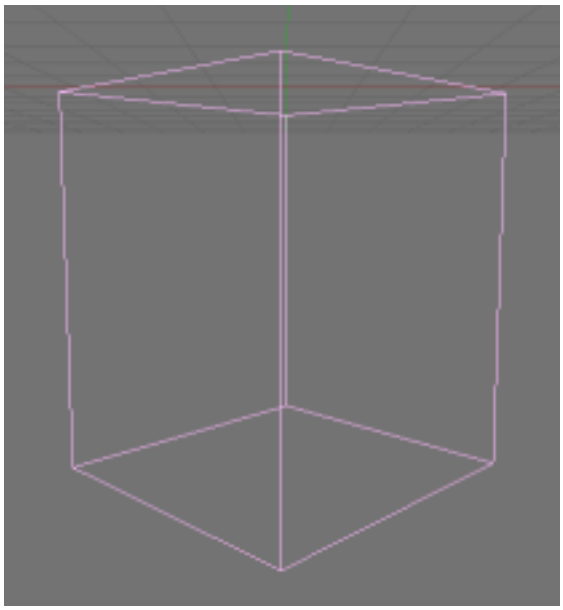


Figura 8.15: Parte inferior de la mesa

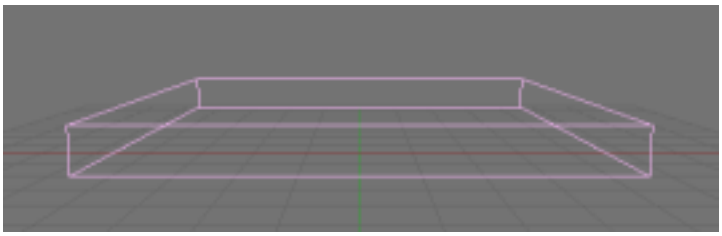


Figura 8.16: Parte superior de la mesa

Queríamos dotar a la mesa de una textura metalizada, así que buscamos en Internet hasta encontrar la siguiente textura:

Figura 8.17: Textura para la mesa

Esta textura fue aplicada directamente desde el editor UV, para poder ser vista tal cual es, desde el módulo de juegos.

Habitación

La cabeza está dentro de una habitación con paredes metálicas. Toda esta estructura ha sido iluminada con focos de tipo lámpara (lamp). La iluminación de la habitación no ha sido creada con un esquema predefinido, se ha ido probando hasta encontrar el efecto deseado.

Está habitación fue creada a partir de un cubo. Dos de las caras se hicieron transparentes para una correcta visualización del interior.

El resto de caras se corresponden con dos paredes, el techo y el suelo.

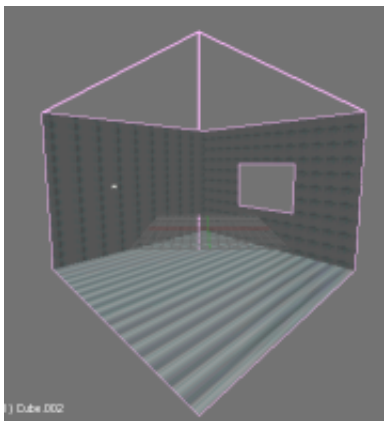


Figura 8.18: Estructura de la habitación

Paredes Las dos paredes poseen la misma textura:

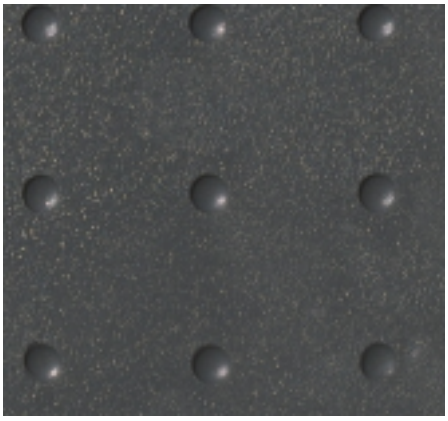


Figura 8.19: Textura de las paredes

Sin embargo, una de ellas posee una ventana. Esta ventana se implementó realizando un agujero en la superficie de la pared. Previamente se subdividió la pared, para luego poder eliminar algunas caras de ésta, lo cual dio como resultado la ventana deseada.

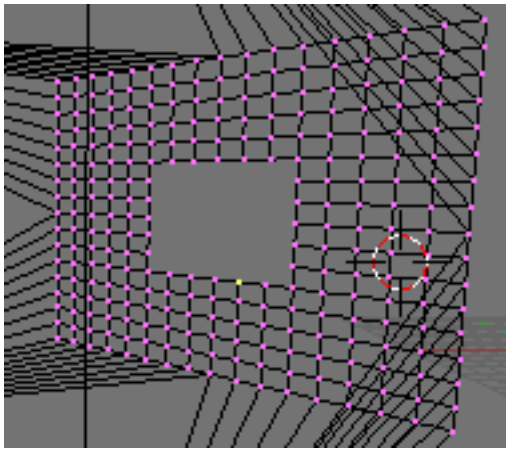


Figura 8.20: Ventana en una de las paredes

Suelo Al suelo simplemente se le aplico la misma textura que a la mesa

Techo fijo Al techo se le aplicó la misma textura que a las paredes. Para poder ver el cielo, se le realizó un agujero de manera similar a una de las paredes. Eso sí, en este caso el agujero es bastante más grande, de hecho, sólo se ve una pequeña porción del plano correspondiente al techo. El efecto se puede observar en la siguiente figura:

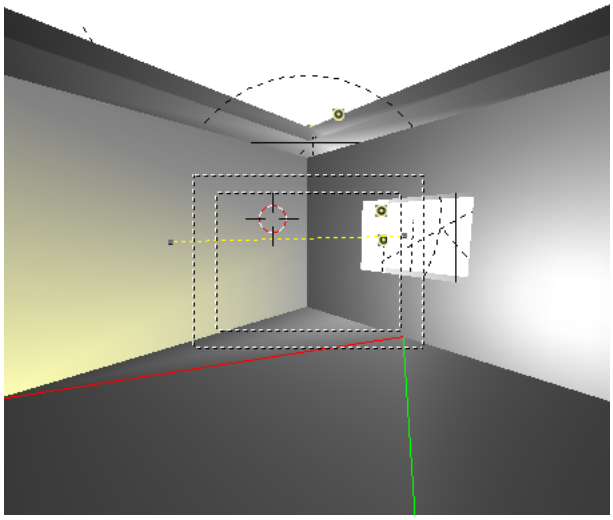


Figura 8.21: Agujero en el techo

Ventana

El hueco dejado en una de las paredes lo hemos rellenado con una ventana. Está compuesta por un marco y un vidrio.

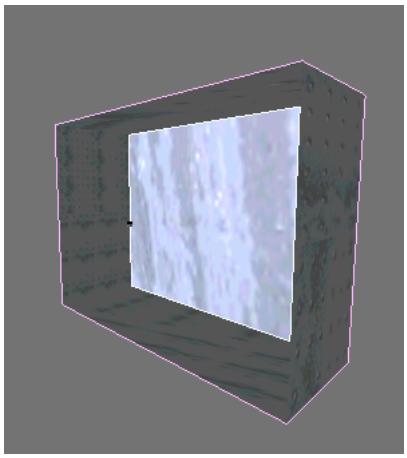


Figura 8.22: Estructura de la ventana

Todo esto se ha modelado con un sólo cubo.

El marco se utiliza para dar sensación de profundidad a la pared perforada. Se le aplica la misma textura que a la pared.

El cristal se corresponde con una de las caras del cubo. A dicha cara se le ha aplicado una textura mediante el Editor UV. Posteriormente se ha activado el parámetro 'add'

para dotar de transparencia a la cara del cubo.

Cielo

El cielo que se puede observar en el techo y a través de la ventana ha sido creado usando dos planos. A estos planos se le ha aplicado la siguiente textura:

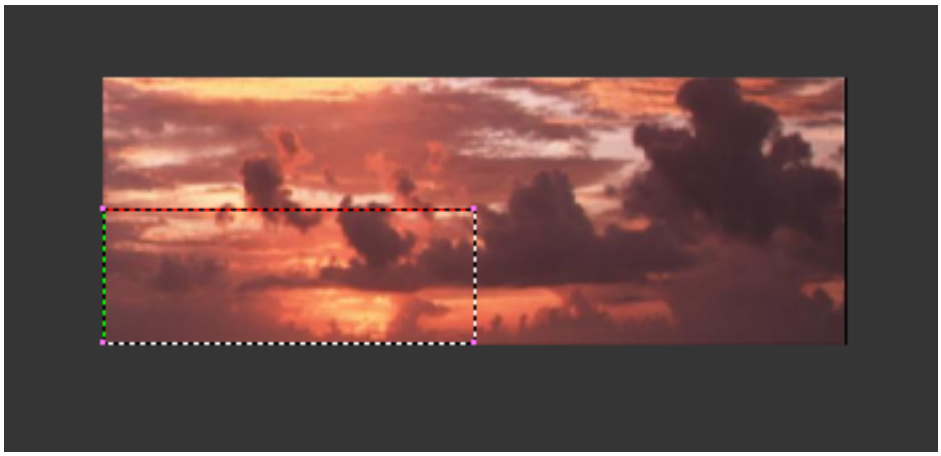


Figura 8.23: Textura del cielo

En la figura, además se observa la porción de la textura realmente utilizada y la manera de realizar el mapeado UV.

Edificios

Los edificios son cubos con una textura aplicada. Para simular la aparición de ventanas iluminadas de manera irregular, tuvimos la ocurrencia de utilizar está textura de letras:

Figura 8.24: Textura de letras para los edificios

Los resultados obtenidos fueron óptimos.

Cuadro

Para representar las respuestas dadas por los usuarios, diseñamos un cuadro que muestra la opción elegida.

Figura 8.25: Cuadro para las respuestas

Tras formular la pregunta, el usuario debe introducir su elección mediante teclado. Tras esto, el cuadro reflejará la respuesta hasta el momento en que se le solicite una nueva respuesta (o sea, al terminar de formular la siguiente pregunta)

La estructura física del cuadro se realizó mediante 'extrudes' a partir de un plano. Con esto se consiguió la estructura tridimensional.

Cuando se teclea la respuesta, el cuadro efectua una animación y emite un sonido. La animación consiste en un crecimiento momentaneo del tamaño del cuadro.

Para explicar mejor el funcionamiento de este proceso, observamos la lógica de juegos que acompaña al cuadro:

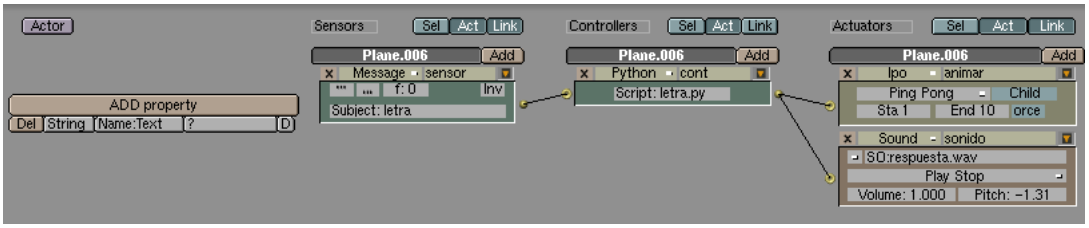


Figura 8.26: Lógica de juegos del cuadro

La propiedad 'Text' almacena la cadena a mostrar en el cuadro. El nombre de la propiedad no es arbitrario, debido a las dificultades de Blender a la hora de mostrar texto, se utiliza este proceso. El nombre de la propiedad debe ser obligatoriamente 'Text'. El cuadro (más bien, una de sus caras) debe tener asociada una textura que contenga las letras a mostrar. Cuando se modifica la propiedad 'Text', inmediatamente Blender modifica el aspecto del plano en cuestión para que refleje el aspecto de la letra deseada.

El proceso de interacción del cuadro con el resto de elementos se produce mediante el paso de mensajes. El mensaje 'letra' indica al cuadro que debe actualizar su aspecto. Este mensaje es enviado cuando se modifica la variable que define que respuesta es la elegida ('elegida'), dicha modificación se realiza en 'teclado.py'. Otro momento en el que se modifica el aspecto del cuadro, es el final de la formulación de una pregunta. En ese momento el sistema debe esperar una respuesta de teclado. Mientras tanto, se muestra una interrogante en el cuadro. El mensaje para mostrarla se envía desde 'moverboca.py' en el preciso momento en que se gesticula la última letra.

Por su parte, el script 'letra.py' simplemente se encarga de cambiar el valor de 'Text' y activar la animación y el sonido cuando sean necesarios (al modificar el valor por una interrogación no se realiza ni la animación ni se lanza el sonido).

Para terminar con esta parte, simplemente decir que el aspecto del cuadro se debe a un texturizado UV realizado mediante el editor UV.

Cartel de publicidad

En el exterior de la habitación, sobre los edificios se puede observar un cartel de publicidad. Es publicidad con el logo de Blender. El objetivo era simular el aspecto de una gran ciudad y darle algo más de vida al exterior de la habitación.

El logotipo de Blender fue capturado de su página oficial 'blender.org'[1].

Asignamos la imagen de Blender a las caras de un prisma mediante el editor UV.

Para dar el efecto de parpadeo del letrero, se superpuso un marco de color azul al borde negro que se puede observar en la figura anterior. Este marco azul se oculta y se muestra en relación con el tiempo. El comportamiento de este marco se puede observar en sus sensores y actuadores en el módulo de juegos.

Entre el segundo 1 y el segundo 2 el marco se vuelve invisible. Entre el segundo 3 y el segundo 4 se muestra el marco y se resetea el temporizador.

[1] http://www.blender.org

Figura 8.27: Logotipo del Blender

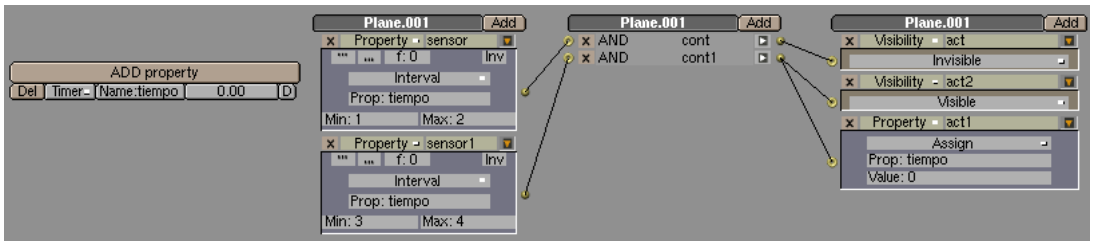


Figura 8.28: Comportamiento del marco del anuncio

Techo y persiana

Hemos diseñado una persiana y un techo para ocultar el exterior de la habitación. Los explicamos en el mismo apartado porque siguen un comportamiento análogo, por ello vamos a centrarnos en explicar la persiana.

La estructura física de la persiana se consiguió mediante un plano. Para darle textura usamos el editor UV.

Para conseguir el movimiento de la persiana se crearon dos botones mediante los cuales el usuario pudiera interactuar para subir y bajar las persianas a su gusto.

Estos botones envían los mensajes necesarios para que la ventana suba o baje. Concretamente vamos a explicar el botón de subida de la persiana.

Analizando su comportamiento en el módulo de juegos observamos lo siguiente: En el lado izquierdo se puede observar que existe una variable ('persiana') que almacena la

Figura 8.29: Botón-flecha para subir la persiana

posición actual de la persiana.

Cuando se pulsa el botón con el ratón y no se ha llegado al tope superior de la persiana (-4), se activa el movimiento de la persiana. Para ello se envía el mensaje 'sube' (para bajar, 'baja'). También decrementamos el valor de la variable que nos determina la posición de la ventana y emitimos el sonido de la persiana.

Pero la variable 'persiana' también debe ser actualizada cuando se baje la persiana. Por ello, hemos dispuesto los sensores y actuadores adecuados.

Al recibir el mensaje 'baja' simplemente incrementamos el valor de la variable. El resto de sensores y actuadores se encargar de hacer el botón más atractivo (animación y sonido de botón).

Nubes

En el cielo se puede observar como se produce un movimiento de nubes. Esto se consiguió mediante un plano con una textura aplicada.

El movimiento del plano se logra mediante un sensor 'always' que realiza una animación

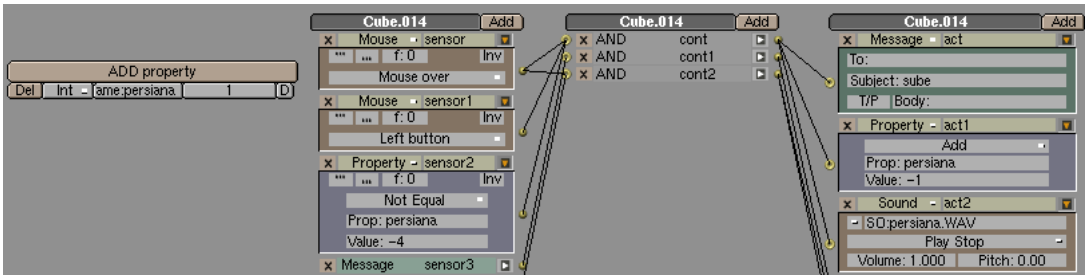


Figura 8.30: Lógica de juegos del botón-flecha para subir la persiana (I)

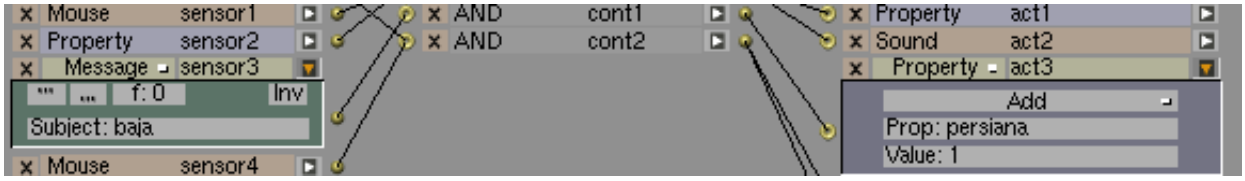


Figura 8.31: Lógica de juegos del botón-flecha para subir la persiana (II)

del plano para dar la sensación de moviento. Para ello usamos esta textura:

Figura 8.32: Textura inicial para las nubes

Tras aplicar transparencia (parámetro 'Add') a esta textura, el resultado es muy bueno, parece que estemos ante auténticas masas de nubes. El problema venía cuando terminaba la animación y volvía comenzar el bucle. En ese instante se producía un salto muy perceptible. Intentamos modificar la animación para encontrar dos puntos (fin e inicio) en los que el salto no fuese tan evidente, pero no conseguimos encontrarlos. Por ello decidimos cambiar la textura aplicada. El objetivo era conseguir a principio y al final de la animación una zona despejada de nubes, con esto se conseguiría una animación sin saltos. Para elaborar esta textura hicimos uso del Gimp para modificar la textura inicial. Este fue el resultado:

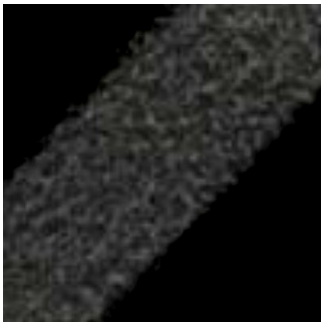


Figura 8.33: Textura final para las nubes

Era necesario conseguir bordes negros pero al mismo tiempo tener una transición suave entre la parte central (nubes) y la parte clara (negro). Este efecto se consiguió mediante un degradado de colores en las esquinas de la textura. Tras aplicar esta textura el resultado obtenido fue satisfactorio.

Bombilla

En una de las paredes se puede observar una lámpara (una lámpara real, no confundir con una lámpara en Blender). Dentro de ésta, econtramos una bombilla que emite luz amarilla (mediante una lámpara de Blender). Esta luz puede ser apagada y encendida por el usuario.

Para ello hemos creado un botón en tres dimensiones con el aspecto de una imagen de pagado y encendido estándar:

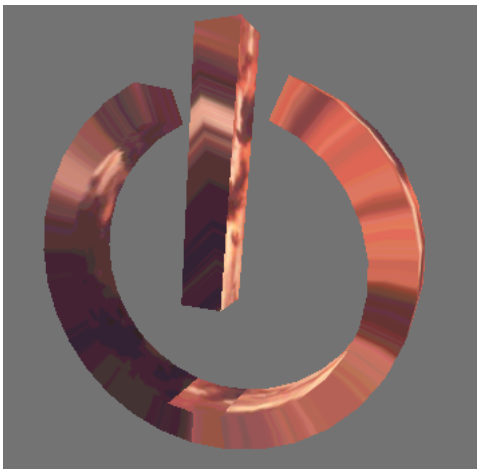


Figura 8.34: Botón de encendido/apagado

La textura ha sido añadido mediante el editor UV.

Este botón envía el mensaje de apagado o encendido a la bombilla dependiendo de si se pulsa con el botón derecho o izquierdo del ratón. Como ejemplo, observamos el proceso de apagado en el módulo de juegos:

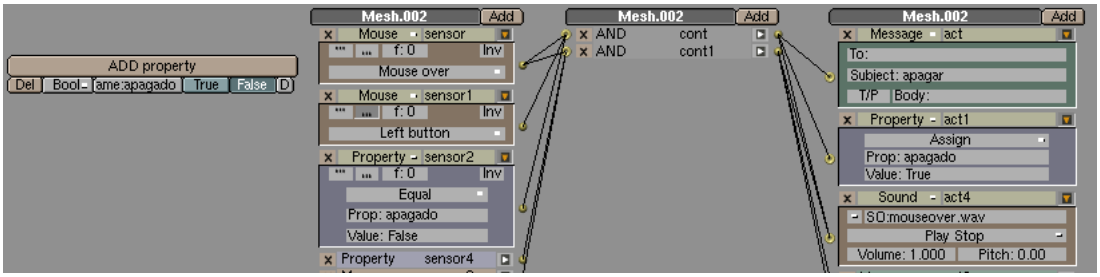


Figura 8.35: Lógica del botón de encendido/apagado

Mediante una variable booleana sabemos si la bombilla está encendida o apagada. Al pulsar con el botón izquierdo, y siempre que la luz este encendida ('apagada=False'), se procede a enviar el mensaje de apagado a la bombilla. Para evitar que se dispare este sensor de nuevo, ponemos la variable 'apagada' a 'True'. Además, emitimos un sonido que coincide con el apagado.

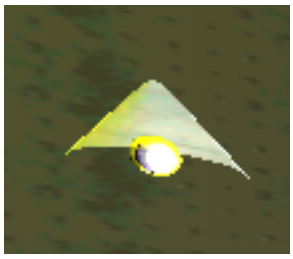


Figura 8.36: Bombilla encendida

Una vez pulsado el botón y enviado el mensaje de apagado, la bombilla amarilla se vuelve invisible. En el interior de esta bombilla se encuentra otra de menor tamaño y de color gris. Al ocultar la bombilla amarilla podemos ver la bombilla gris, que es la que da el efecto de apagado.

Para eliminar el efecto de iluminación producido por la lámpara de Blender, hacemos que ésta al recibir el mensaje de apagado se desplace hasta estar fuera del escenario. En el momento del encendido se produce un desplazamiento inverso.

Figura 8.37: Bombilla apagada

Naves

A través de la ventana, se puede observar un tráfico incesante de vehículos aéreos (naves). Este efecto ayuda a dar la impresión de que nos encontramos en una gran ciudad futurista.

Para conseguir dicho efecto hemos usado 18 naves (9 en un sentido y 9 en el sentido contrario). Este es el aspecto de una de las naves vista desde cerca:

Figura 8.38: Nave de cerca

Es una esfera con sus mitades separadas mediante un tubo.

Cada una de estas naves realiza el mismo movimiento una y otra vez. Para evitar la sensación de ciclado en el movimiento de estos vehículos, se opto por hacer que tuvieran distintas velocidades y distintas trayectorias. Para hacer que se mueva cada una de las naves, se ha utilizado un sensor 'always' que ejecuta una y otra vez la animación creada mediante una curva IPO.

Algunas de las naves también poseen un actuador que emite un sonido. Se decidió dotar de sonido a sólo algunas naves, ya que lo contrario hubiera sido extremadamente molesto para el usuario.

Tecla de salir

Para poder cerrar de manera adecuada el módulo de juegos, creamos un botón que se encargara de esta labor. Este es el aspecto del botón:

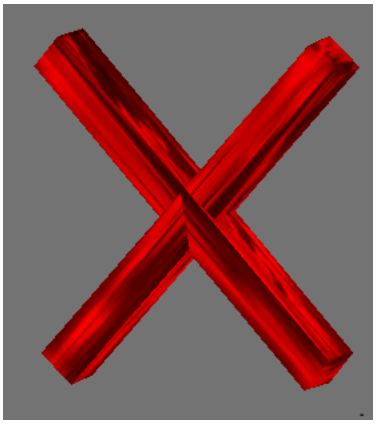


Figura 8.39: Tecla de salir

Este botón posee animaciones y sonido, la implementación de estas características se llevó a cabo de manera similar a la de otros botones explicados con anterioridad.

Al hacer click sobre el botón, se envía el mensaje 'mata'. El objetivo de dicho mensaje es el propio botón de salir. Este proceso es necesario para poder ejecutar el script en python 'matar.py'. Este script se encarga de realizar las operaciones necesarias antes de cerrar Blender y posteriormente salir del módulo de juegos. La única operación que se realiza antes de cerrar el módulo de juegos es salir del CLIPS. Merece mención especial la forma de cerrar el módulo de juegos. Se puede observar en la lógica de juegos del botón:

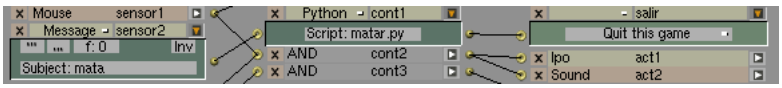


Figura 8.40: Lógica de juegos del botón de salir

El único actuador desplegado en la figura es el encargado de salir. Este actuador es lanzado desde 'matar.py'. En principio, este tipo de actuador no estaba implementado en Blender 2.25, pero sí en Blender 2.34. Al editar nuestro fichero y añadirle ese actuador en el Blender 2.34 nos encontramos con la sorpresa de que era plenamente funcional en la versión 2.25. Resumiendo: el actuador sólo se puede añadir en la versión 2.34, pero funciona en ambas versiones.

Tecla de comenzar

El botón de comenzar sólo se muestra antes de comenzar el test. Tiene forma de triángulo (similar al botón 'play' de un reproductor de vídeo o sonido).

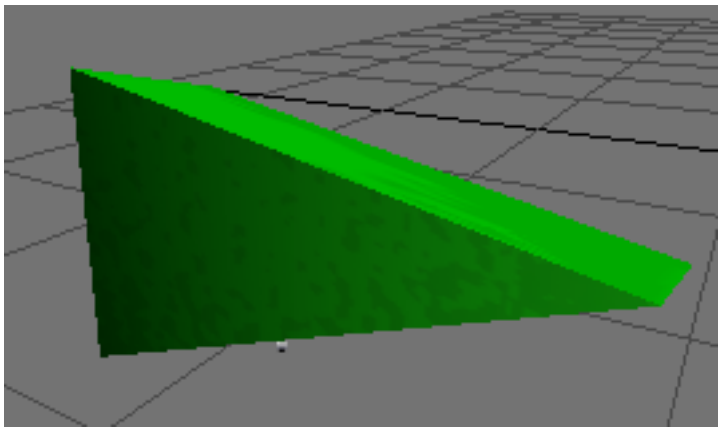


Figura 8.41: Tecla de Comenzar

En este caso la textura se consiguió mediante la mezcla de un material verde y un texturizado UV.

Este botón posee animación y sonido cuando se pasa sobre él sin pulsar. El botón sólo puede ser pulsado después de que el personaje termine el saludo inicial. Para hacer que se active el botón al terminar el saludo, el script 'moverboca.py' manda un mensaje que activa el botón. Al pulsar el botón, este se oculta y no se vuelve a mostrar durante el test.

Música

De fondo se puede escuchar una música que da ambiente futurista a la escena. Esta canción fue seleccionada de entre las canciones gratuitas encontradas en 'Epitonic'[1]. La canción se ejecuta continuamente gracias a un sensor 'always' asociado a la tecla de salir. La canción descargada estaba originalmente en formato MP3, y Blender sólo admite WAV. Debido a esto y al gran tamaño que generaría el fichero en WAV, nos vimos obligados a reducirlo a un fragmento de 30 segundos mediante el programa MP3DirectCut MP3DirectCut[2]. Para poder transformar el fragmento al formato WAV con las características deseadas utilizamos el programa DBPowerAmp Music Converter 'DBPowerAmp Music Converter'[3]. Ya que la múscia debía sonar en segundo plano con un volumen menor

[1] http://www.epitonic.com
[2] http://www.mpesch3.de/
[3] http://www.dbpoweramp.com

que el resto de elementos, decidimos crear el WAV con el menor tamaño posible (8 bits, 8KHz y sonido MONO). La canción seleccionada fue: 'Subpoena The Past - Altitude On Ice'.

8.1.4. Gestos

Para darle un poco más de vida y naturalidad al personaje, decidimos crear un script que ejecutará de manera aleatoria un conjunto de gestos.

Dentro de estos gestos encontramos el movimiento de cabeza, ojos, boca, etc.

Funcionamiento del script

El script en cuestión se llama 'gestos.py'. Su cometido es hacer que la cabeza realice unos gestos aleatorios cada cierto tiempo.

Está bastante parametrizado, es muy sencillo cambiar el tiempo que tarda en ejecutarse un gesto, los frames que ocupa y el numero de gestos.

Básicamente este el algoritmo seguido:

- Cada cierto tiempo (regulado por un timer) generamos un número aletorio que determinará el gesto.
- Durante el tiempo que dura un gesto lo ejecutamos de acuerdo con el tiempo transcurrido. Para esto se realizó una 'simple' regla de tres.
- Cuando acaba el gesto el personaje permanece en espera hasta que se genere el siguiente gesto.

Tipos de gestos

Durante la ejecución del algoritmo anterior se ejecutan diferentes tipos de gestos. Los gestos pueden ser:

- IPOs desde GameEngine: Para activar animaciones soportadas por el módulo de juegos. O sea: rotación y posición en nuestro caso.
- IPOs desde código: Invocamos desde código las IPOs que usan Relative Vertex Keys. No nos queda otro remedio que llamar a nuestro método para la ejecución de este tipo de IPO (ver 'basic.py')
- Motion: Usamos este tipo de animación, que es más simple. Es similar a las IPOs tradicionales desde el GameEngine pero no se muestra la animación intermedia, o sea, se salta del frame inicial al final de golpe (ideal para movimientos rápidos).

A continuación pasamos a explicar los movimientos implementados.

Movimientos de cabeza. Los movimientos de cabeza los ejecutamos con IPOs desde el GameEngine. Realizamos estos movimientos:

- Girar la cabeza a la derecha y volver a la posición inicial
- Girar la cabeza a la izquierda y volver a la posición inicial
- Girar la cabeza hacia abajo y volver a la posición inicial

Figura 8.42: Movimiento de cabeza hacia la izquierda

Para conseguir que la cabeza se moviera tuvimos que duplicar la malla de la cabeza. De esta manera, de cuello para abajo disponemos de una malla y de cuello para arriba, de otra. Así logramos movimientos de cabeza sólo con rotaciones y cambios de posición. Esto permite que los movimientos puedan ser realizados directamente desde el módulo de juegos.

Movimiento de boca. Los movimientos de la boca, al tratarse de deformación se han realizado desde código mediante RVKs. Estos han sido los movimientos de boca implementados:

- Leve sonrisa
- Leve gesto de enfado/decepción
- Retorno a la posición inicial

Figura 8.43: Leve sonrisa

Movimiento de cejas. El movimiento de cejas se realiza mediante IPOs desde el GameEngine, ya que sólo hemos desplazado la malla. Este movimiento consiste en levantar las cejas. El personaje durante el gesto levanta las cejas y las vuelve a bajar.

Figura 8.44: Cejas Levantadas

Movimiento de ojos. Otro de los gestos que dan mucha naturalidad es la rotación de los ojos a uno y otro lado. En este caso lo hemos realizado mediante 'Motion' en el GameEngine (indicándole los grados de giro que deseamos). Durante el gesto el ojo mira para uno de los dos lados y vuelve a la posición inicial.

Figura 8.45: Cejas en posición inicial

Figura 8.46: Ojo izquierdo mirando a la izquierda

8.2. Animación de la boca

Para hacer que la boca se deforme nos vemos obligados a modificar los vértices de la malla. Mediante las claves de vértices (vertex keys) conseguimos crear animaciones de deformación mediante la modificación de los vértices.

Podemos usar dos tipos de vertex keys:

- **Claves absolutas**: permiten realizar animaciones en las que se modifican posiciones de los vértices.
- **Claves relativas (RVK)**: igual que las anteriores pero con la gran ventaja de que sólo guardan la información de los vértices modificados (en lugar de toda la malla). Esto permite realizar varias deformaciones independientes (ojos, boca, etc.) dentro de la misma malla de forma simultánea.

Debido a la versatilidad que permiten, usamos RVKs. A continuación explicamos las distintas claves que utilizamos.

8.2.1. Claves usadas para el parpadeo

- Clave 1: Ojo derecho cerrado.
- Clave 2: Ojo izquierdo cerrado.

Figura 8.47: Ojo derecho cerrado

Figura 8.48: Ojo izquierdo cerrado

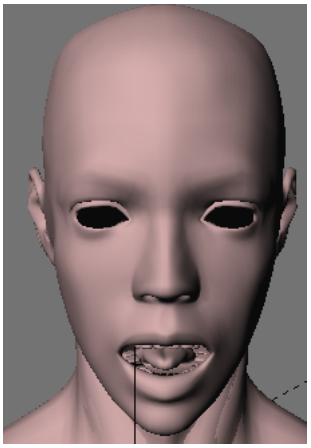


Figura 8.49: Labio inferior levemente bajado

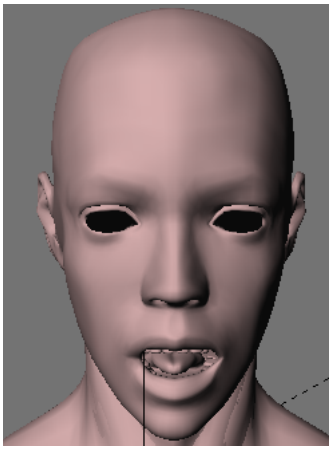


Figura 8.50: Movimiento de comisura derecha

8.2.2. Claves usadas para el habla

- Clave 3: Labio inferior levemente bajado.
- Clave 4: Movimiento de comisura derecha.
- Clave 5: Movimiento de comisura izquierda.
- Clave 6: Labio superior levemente subido.
- Clave 7: Lengua abajo.
- Clave 8: Lengua arriba

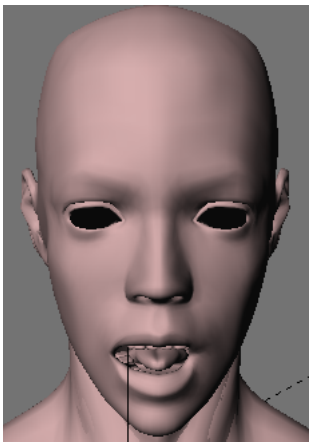


Figura 8.51: Movimiento de comisura izquierda

Figura 8.52: Labio superior levemente subido

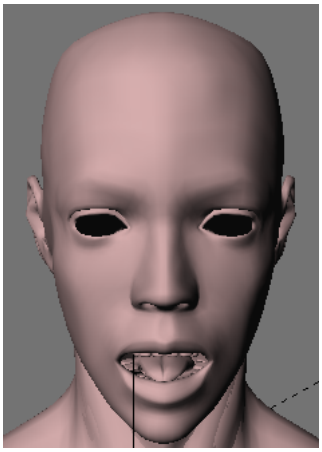


Figura 8.53: Lengua abajo

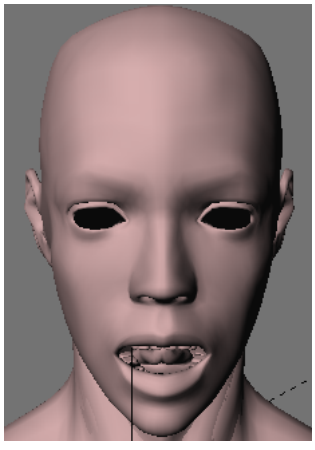


Figura 8.54: Lengua arriba

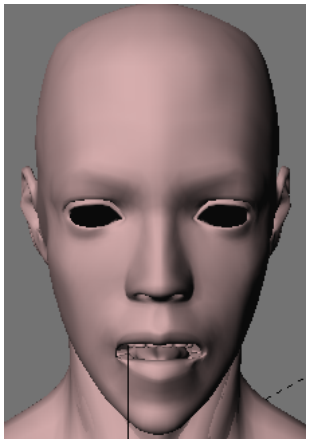


Figura 8.55: Labio inferior levantado

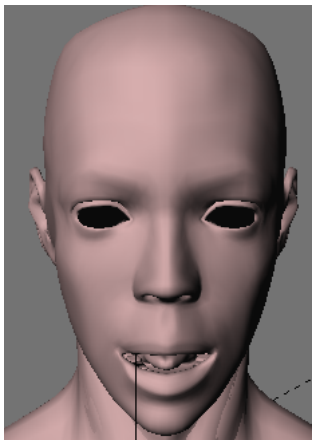


Figura 8.56: Labio superior bajado

- Clave 9: Labio inferior levantado
- Clave 10: Labio superior bajado

8.2.3. Otras claves

- Clave 11: Sonrisa

8.2.4. Conseguir los movimientos

Mediante el uso conjunto de estas claves, se puede conseguir que la boca adopte la forma deseada. A cada clave se le asigna un valor entre 0 y 1 dependiendo de si queremos

Figura 8.57: Sonrisa

que no se aplique (0), que se aplique totalmente (1) o que se aplique a medias (valores intermedios).

También se pueden asignar valores negativos para conseguir un movimiento de la boca contrario al de la clave. Por ejemplo, si se le da valor negativo a la sonrisa, se consigue enfado.

8.2.5. Fichero de configuración

Los valores de las claves son almacenadas en un fichero de configuración que lee HEVAH al ser lanzada. En este fichero se identifican los grupos bucales y las letras pertenecientes a cada uno de ellos con los valores óptimos para su representación. Esto nos permite un amplio margen de maniobra: desde conseguir afinar la simulación del movimiento de cada grupo bucal, hasta la internacionalización de estos movimientos creando el fichero correspondiente a distintos idiomas.

8.2.6. Threading para el TTS

El primer problema que nos encontramos al intentar comunicar nuestros scripts de python desde Blender con el TTS fue la secuencialidad, es decir, cuando se llamaba al TTS se detenía la ejecución en Blender hasta que finalizaba la ejecucion del sintetizador de voz. Esto resultaba totalmente inaceptable ya que el resultado sería que se oiría la voz, pero hasta que esta no terminara, no se movería la boca.

Una primera aproximación para solucionar este problema fue la de ejecutar el TTS en un hilo independiente del de la ejecución del presonaje en Blender. Realmente esto no solucionó el problema, debido a la naturaleza de los hilos en python. No basta con crear un hilo y ejecutarlo; además es necesario indicar que se ejecute como demonio para conseguir la independencia del hilo principal. Una vez que el TTS termina su ejecución el hilo muere sin que esto afecte de ninguna forma a la ejecución del resto del programa.

Una vez establecida esta independencia nos encontramos con otro problema, la posibilidad de que se ejecutara un nuevo hilo, o más bien demonio, sin que hubiera terminado el anterior. Para solucionar ésto simplemente colocamos un semáforo que indica si ha terminado la ejecución del hilo, de forma que solo permita una nueva llamada al TTS cuando la anterior haya finalizado.

8.2.7. Implementación de la iluminación

En general, la iluminación fue creada con lámparas estándar del Blender distribuidas de forma arbitraria.

Los diferentes elementos están distribuidos por capas, lo cual permite una iluminación individual para cada grupo de objetos.

Existen muchos elementos que no tienen iluminación, sencillamente porque no se ha considerado necesario. Estos elementos son:

- El pelo: al tener un color muy oscuro, no se apreciaría la iluminación.
- Las pestañas y las cejas: por el mismo motivo que el pelo.
- Los botones y las naves.

En el caso de los ojos, simplemente se situaron dos lámparas para dar algo de luz.

El escenario fue iluminado añadiendo lámparas a medida que creábamos partes del entorno.

Eso sí, para crear la iluminación de la parte más importante, la cabeza, usamos un esquema predefinido.

Iluminación de la cabeza

Para la iluminación de la cabeza hemos usado un esquema clásico en el cine. Este sistema está explicado en los fundamentos teóricos y utiliza tres fuentes de luz.

En la figura se pueden ver las tres fuentes utilizadas.

- Luz clave: es la luz situada delante de la cabeza. Es de tipo 'Spot' (luz dirigida) y es la más fuerte de la escena.
- Luz de relleno: es la luz central que posee forma circular (en la figura sólo se observa parte del círculo). La creamos de tipo 'Lamp' para poder rellenar mejor los huecos dejados por la luz clave.
- Luz trasera: es la luz situada detrás de la cabeza. En el Blender es una luz de tipo 'Spot' de menor fuerza que las otras dos.

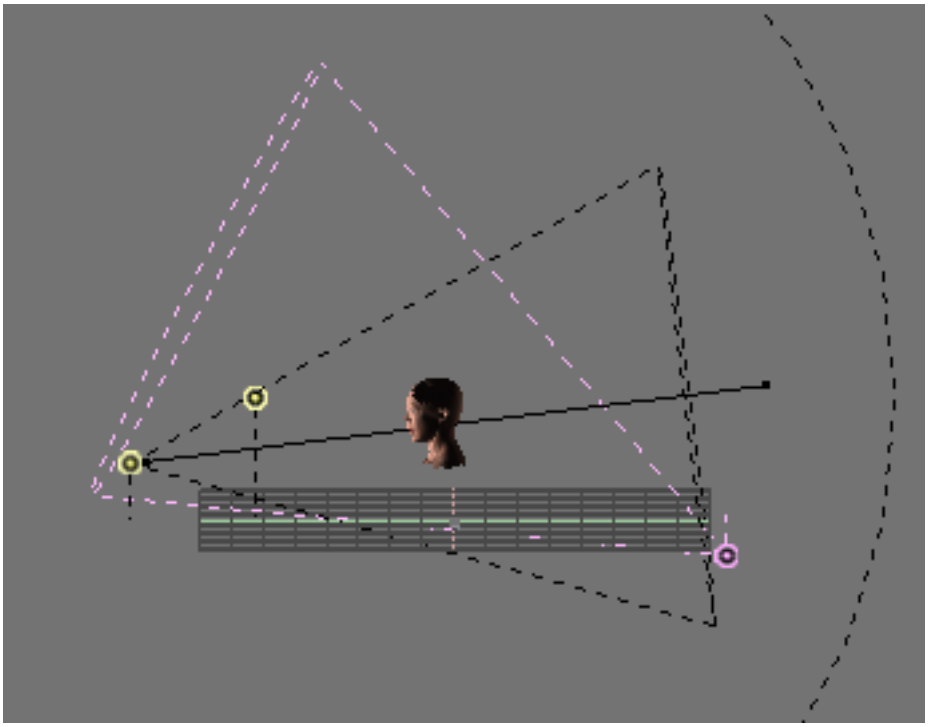


Figura 8.58: Iluminación de la cabeza

8.3. Interacción con el Personaje

El motor/módulo de juegos de Blender (GameEngine) nos ha permitido realizar operaciones muy complejas apoyándonos en los scripts en python. A continuación explicaremos la estructura básica de la lógica de juegos utilizada en el proyecto.

8.3.1. Lanzador

El lanzador es el primer interfaz con el que el usuario interactúa. A través de él selecciona el test que desea realizar y lanza a HEVAH.

8.3.2. Sistema de Control

Este fichero de Python realiza las principales labores del proyecto, es el núcleo de la lógica de juegos.

El núcleo de control de HEVAH está implementado en el script 'control.py'. Este módulo se está ejecutando continuamente desde el inicio del programa. Para conseguir este efecto, utilizamos el sensor 'Always'.

Los actuadores que se pueden observar a la derecha, envían un mensaje para que se cierren o abran los ojos, dependiendo de en que parte del ciclo de parpadeo nos encontremos. A la hora de poder lanzar un actuador desde código, dicho actuador debe estar enlazado con el fichero .py en el módulo de juegos (como es el caso).

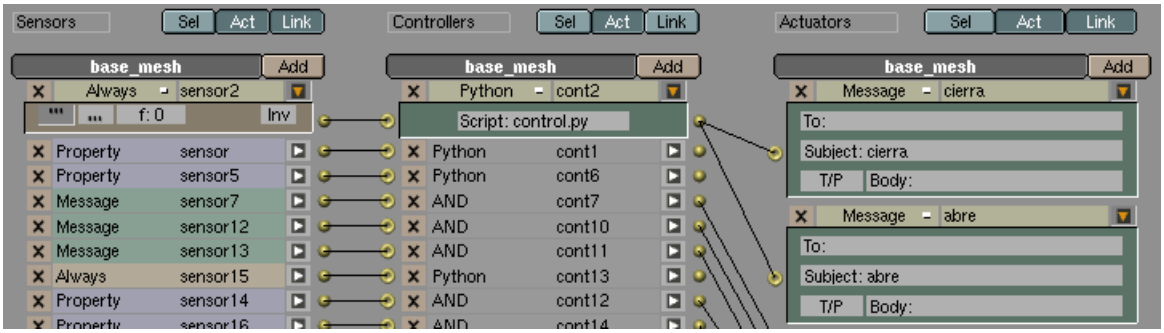


Figura 8.59: Sensor 'Always' para 'control.py'

El script 'control.py' realiza las siguientes funciones:

- Hacer que se muestre el ratón
- Inicializar el CLIPS
- Cargar el perfil del usuario
- Cargar todos los ficheros necesarios
- Pasa a la siguiente pregunta
- Lanzar el parpadeo (desde los actuadores que tiene enlazados el script)

8.3.3. Funciones del habla

Para implementar el habla en el GameEngine, hicimos uso de dos scripts en python, 'hablar.py' y 'moverboca.py'

Como se puede observar en la figura, ambos ficheros carecen de actuadores. Estos scripts desarrollan las siguientes labores:

- **hablar.py**. Se encarga de lanzar el Speech API, que a su vez se encarga de iniciar el sonido de la voz. Como el script se lanza cada vez que se modifica la propiedad 'frase', simplemente modificamos esta variable para hacer que el personaje empiece a

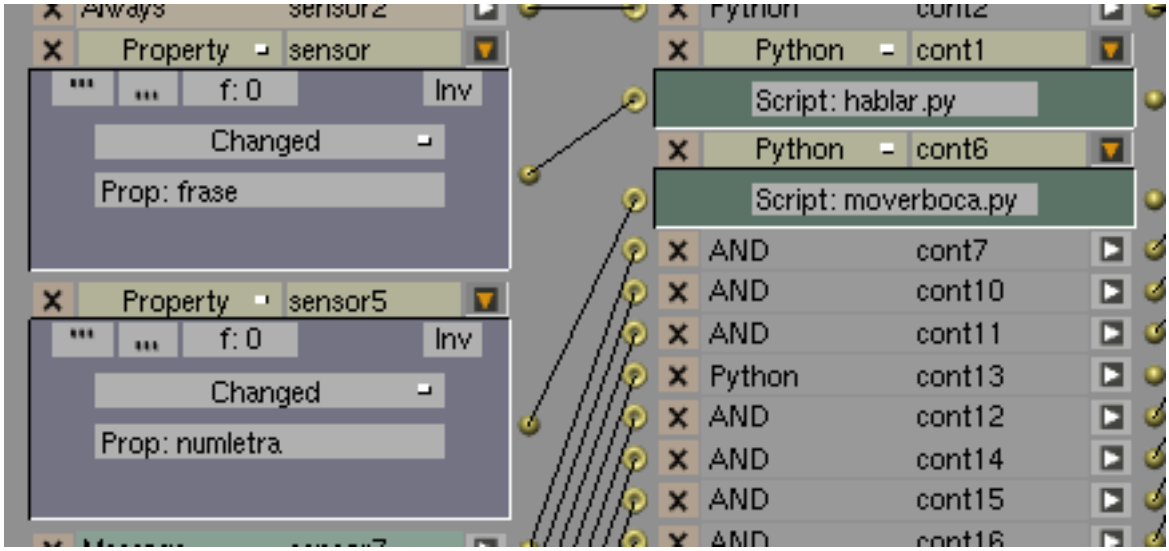


Figura 8.60: Scripts 'hablar.py' y 'moverboca.py'

hablar. Dentro de ese script también se inicializa la variable 'numletra'. Esta librería está bastante parametrizada, con lo cual permite realizar cambios sobre la voz del personaje de una manera bastante sencilla.

- **moverboca.py**. Su función es hacer que se mueva la boca del personaje mientras habla. Este script se lanza cada vez que se modifica 'numletra'. Este valor es un contador de la letra actual, por lo tanto, podemos decir que se lanza para cada letra. La manera en que se pone la boca está expresada en el fichero de configuración 'mbesp.conf'.

8.3.4. Gesticulación

Existe otro script que se ejecuta de manera contínua: 'gestos.py'. Este fichero se encarga de hacer que el personaje realice gestos aleatorios cada cierto tiempo.

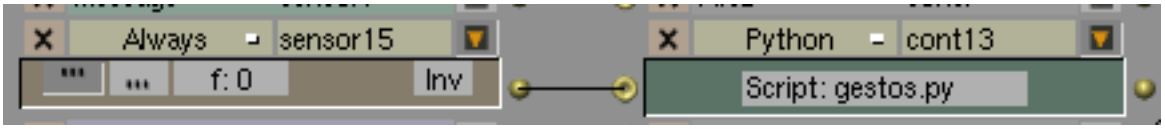


Figura 8.61: Scripts 'gestos.py'

Este script tampoco hace uso de actuadores. Su principal función es modificar el valor

de la variable 'k'. Este valor se obtiene de manera aleatoria y establece que tipo de gesto se debe ejecutar. En el apartado de gestos se explica que gestos se implementaron.

A continuación se muestra un ejemplo de un gesto mediante una IPO:

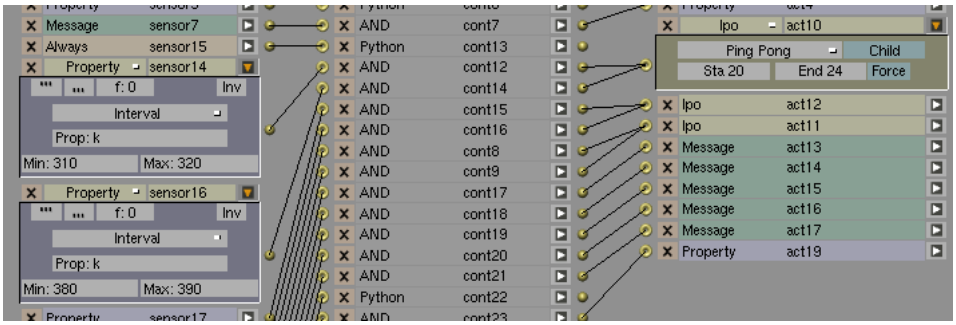


Figura 8.62: Gesto mediante IPO

Como se puede observar en los sensores desplegados, éstos se activan cuando la variable 'k' está en cierto intervalo. En este caso concreto, se trata del gesto que se produce entre 300 y 400. Existen dos sensores ya que el gesto consiste en dos movimientos, uno de ida y uno de vuelta.

Ambos sensores poseen el mismo actuador como destino, pero al tener activada la opción 'ping pong', la primera llamada ejecuta la animación en orden, mientras que la segunda llamada lo ejecuta a la inversa. De esta manera se consigue que el personaje vuelva a la posición inicial.

También existen gestos en los que se hace uso del paso de mensajes a otros objetos distintos de la cabeza del personaje. Este es un ejemplo de ello:

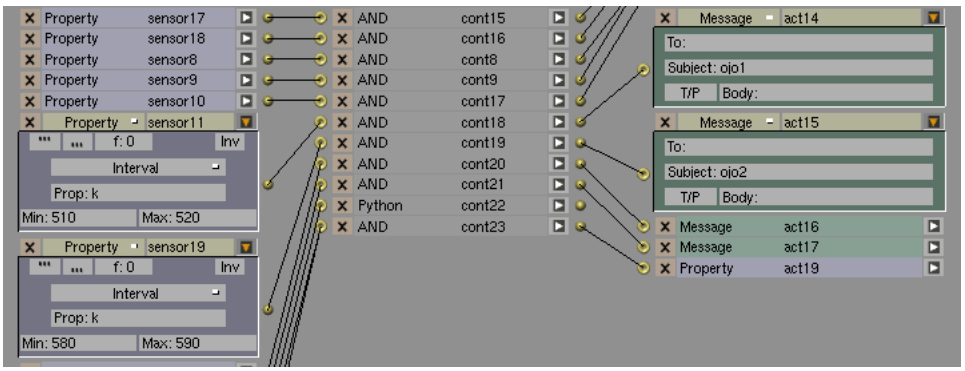


Figura 8.63: Gesto con paso de mensajes

Este gesto se realiza para un valor de 'k' comprendido entre 500 y 600. Al realizarlo,

los ojos giran hacia un lado. Primero se manda un mensaje para que los ojos vayan a un lado ('ojo1') y luego otro para que vuelvan a su posición inicial ('ojo2'). Este mensaje es tratado por los dos ojos, y de esta manera se mueven al mismo tiempo.

El otro tipo de gesto que existe, o sea, mediante RVKs, no tiene reflejo en el módulo de juegos. Este tipo de gesto se corresponde con las deformaciones de la cara.

8.3.5. Elección de la respuesta

El sistema permite responder con todas la letras del alfabeto. Para ello el usuario simplemente debe pulsarla en el teclado. El script que se encarga de recoger este valor es 'teclado.py'.

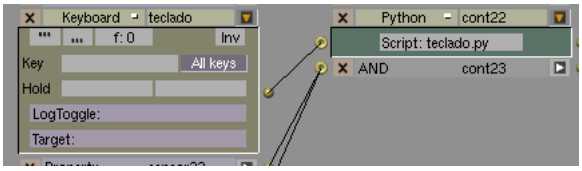


Figura 8.64: Script 'teclado.py'

Como se puede observar en la figura anterior, el script es lanzado mediante un sensor de teclado. Este sensor activará la ejecución tras pulsarse cualquier tecla. En el código de 'teclado.py' se modifica la variable 'elegida'.

Precisamente, la variable 'elegida' es la que define cuando se cambia de pregunta. Cuando se modifica esta variable (a un valor correcto, dentro del rango adecuado de respuestas) se pasa de pregunta. Este paso de pregunta se obtiene cambiando el valor de 'contestada' a 1. Al cambiar este valor, 'control.py' se encarga de proponer la siguiente pregunta.

La variable se modifica a 'VACIO' al pasar de pregunta, y sólo se modifica este valor cuando se pulsa una tecla adecuada.

Cambio de Pregunta

Esta es una de las interacciones fundamentales, ya que si no se produjera el cambio de pregunta, no podríamos realizar el test. Aquí tuvimos muchísimos problemas, ya que el motor de inferencia de HEVAH está implementado en CLIPS y necesitábamos comunicar con él desde Blender.

Para ello tuvimos que implementar un túnel entre el proceso de Blender y el proceso de CLIPS, utilizando las librerías de gestión de procesos de python. El funcionamiento

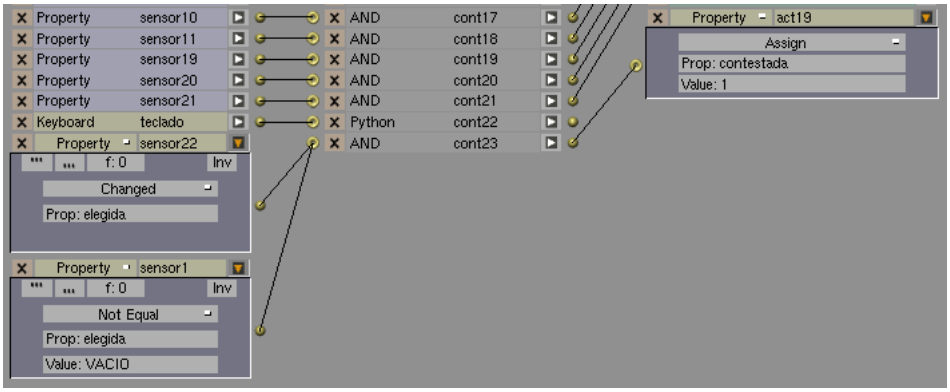


Figura 8.65: Contestación de preguntas

de este túnel es bastante sencillo conceptualmente hablando, pero bastante complicado de implementar; la idea es tener un proceso de CLIPS lanzado en segundo plano, y que HEVAH le vaya enviando las entradas y recogiendo las salidas; es decir, que HEVAH sea el 'usuario' que va 'introduciendo' los comandos oportunos en el CLIPS y 'leyendo' las salidas correspondientes.

Aquí nos encontramos frente a un problema a la hora de conseguir mantener el objetivo de la multiplataforma, ya que estas librerías son distintas dependiendo del sistema operativo utilizado. Mientras que para sistemas operativos Windows utilizamos la win32api y la win32sys, para sistemas Linux nos decantamos por utilizar una librería llamada pexpect, que permite interactuar con procesos lanzados en segundo plano.

Capítulo 9

Implementación de PORTAD

9.1. El test adaptativo

Debido a las limitaciones del web, nos vimos obligado a cambiar el motor de inferencia; mientras que en HEVAH el motor de inferencia está en CLIPS, en PORTAD tuvimos que desarrollarlo en python, implementando las mismas reglas que teníamos en CLIPS mediante sentencias condicionales, e incorporándolo al servidor WebWare. Esta implementación podría dar problemas en el caso de que el motor de inferencia se siguiera mejorando y complicando, ya que habría que comprobar cuáles son los tiempos de respuestas obtenidos. Almacenamiento de datos

Para realizar el almacenamiento de los datos relacionados con el sistema optamos por ficheros XML. Con el fin de hacer un sistema más compatible y más portable consideramos que esta sería una buena solución. Por otro lado la naturaleza independiente de los datos era otra motivación para crear una estructura de ficheros. Utilizar el estándar XML resulta fundamental ya que permite o puede permitir la integración con otros sistemas, elemento que quizá sea de interés en un futuro. El hecho de tener los datos almacenados en fichero supone un peligro para la integridad de los mismos, ya que un usuarios malintencionado podría modificarlos, provocando así inconsistencias en el sistema. Para solucionar este problema optamos por realizar un cifrado de los ficheros, que se explicara más adelante.

Veamos la estructura y funcionalidad de cada unos de los ficheros:

9.1.1. Índice

En este fichero se almacena un listado con todos los usuarios registrados del sistema, así como información útil de los mismos. Veamos un ejemplo:

```
<PACO GRUPO='PROFESORES' PASSWD='' USERNAME='PACO'/>
<PEPE GRUPO='ALUMNOS' PASSWD='' PROFESOR='PACO' USERNAME='PEPE'/>
<ROOT GRUPO='ADMINISTRADOR' PASSWD='root' USERNAME='ROOT'/>
```

Básicamente se almacena el nombre de usuario, el grupo al que pertenece y el password. Si el usuario pertenece al grupo de 'ALUMNOS' además se almacena el profesor que le dio de alta en el sistema. Es aquí donde el sistema valida al usuario cuando intenta acceder al sistema verificando que los datos introducidos son correctos.

9.1.2. Perfil de Alumno

Este fichero se crea cuando un profesor da de alta a uno de sus alumnos, y su nombre coincide con el nombre de usuario de dicho alumno. Contiene información relacionada con el alumno atendiendo a los tests que haya realizado. Veamos un ejemplo de dicho fichero:

Lo primero que veremos es el nivel que tiene el alumno en ese momento. Cuando se inserta un alumno por primera vez se le asigna un nivel por defecto de 'novel'. Un ejemplo es:

```
<USUARIO NIVEL='novel'>
```

Seguidamente vienen los datos personales del alumno. Por ejemplo:

```
'<DATOS APELLIDO1='DIAZ' APELLIDO2='GONZALEZ' DNI='45458621'
EMAIL='anyaterme@hotmail.com' NOMBRE='DANIEL' USERNAME='DANI'/>
```

A partir de aquí, aparece información relacionada con cada uno de los tests realizados hasta el momento. Veamos el ejemplo de un test:

- Primero se coloca información general sobre el test que ha realizado. Más concretamente la fecha y la hora de realización del test, el fichero donde se encuentra almacenado y la nota obtenida tras la evaluación.

  ```
  <TEST FECHA="02/09/2004" FICHERO="/tests/cambio.xml" HORA="11/18/31"
  NOTA="4.800">
  ```

- A continuación la tasa de aciertos y fallos atendiendo al grado de dificultad de los conceptos (simple, intermedio, complejo y avanzado). Para cada uno de los niveles de dificultad de los conceptos se explicita el número de preguntas acertadas y falladas según su nivel de dificultad (alta, baja y media).

```
<SIMPLE>
    <ALTA ACIERTOS='1.0' FALLOS='0.0'/>]
    <BAJA ACIERTOS='0.0' FALLOS='0.0'/>]
    <MEDIA ACIERTOS='1.0' FALLOS='0.0'/>]
</SIMPLE>
<INTERMEDIO>
    <ALTA ACIERTOS='1.0' FALLOS='0.0'/>
    <BAJA ACIERTOS='0.0' FALLOS='0.0'/>
    <MEDIA ACIERTOS='0.0' FALLOS='0.0'/>
</INTERMEDIO>
<COMPLEJO>
    <ALTA ACIERTOS='0.0' FALLOS='2.0'/>
    <BAJA ACIERTOS='0.0' FALLOS='0.0'/>
    <MEDIA ACIERTOS='0.0' FALLOS='0.0'/>
</COMPLEJO>
<AVANZADO>
    <ALTA ACIERTOS='0.0' FALLOS='0.0'/>
    <BAJA ACIERTOS='0.0' FALLOS='0.0'/>
    <MEDIA ACIERTOS='0.0' FALLOS='0.0'/>
/AVANZADO>
```

- A continuación se almacena un histórico con todas las preguntas que se le han realizado al usuario durante el test y si dicha pregunta fue respondida correctamente o no. Para almacenar dicha información se siguió la siguiente estructura:

  ```
  <PREGUNTA ID="1.1.2" VALOR="ACERTADA"/>
  ```

  Como podemos observar se almacena información en dos campos: 'ID' y 'VALOR'. El primero de ellos conforma un código del cual se extrae, el identificador del tema (primer número), el identificador del concepto (segundo número) y el identificador de la pregunta (tercer número). El segundo campo tan solo indica si dicha pregunta fue respondida correctamente o no.

9.1.3. Perfil de profesor

Este fichero se crea cuando un usuario administrador da de alta un usuario profesor en el sistema. Contiene la información relacionada con el profesor de la siguiente forma:

En primer lugar los datos personales, por ejemplo:

```
<DATOS APELLIDO1='Hernández' APELLIDO2='Rodriguez' DNI='33333333'
EMAIL='pepe@ull.es' NOMBRE='Pepe' USERNAME='Pepe'/>
```

A continuación aparecerá información relacionada con el entorno de trabajo de dicho profesor, concretamente información de los alumnos y de los test que haya dado de alta. Cada una de estas categorías vendrá especificada por un metatag.

Los usuarios estarán identificados de la siguiente forma:

```
'<PACO GRUPO='ALUMNO' PASSWD='' USERNAME='ZEBEN'/>
```

En este tag existen tres elementos identificativos que son los únicos necesarios: el nombre de usuario, su clave de acceso al sistema y el grupo al que pertenece. El nombre de usuario aparece duplicado para optimizar los accesos. No son necesarios más campos en este punto pues el nombre de usuario define unívocamente a los usuarios.

Por otra parte, las referencias a los tests siguen este otro formato:

```
<CAMPOS DESC="Conceptos de campos magnéticos" FICHERO="CAMPOS" TEXTO="Campos"/>
```

Almacenamos el nombre del fichero que contiene el test, la descripción del mismo y un texto que podría ser equivalente al título del test.

Tanto la información relacionada con los usuarios como la relacionada con los test están delimitadas de forma que su identificación resulta bastante cómoda.

Aunque la realización de los test está pensada para los alumnos, a un profesor podría interesarle realizar alguno de ellos para verificar que están elaborados correctamente o para comprobar que la evaluación que se lleva a cabo concuerda con lo que él esperaba. En este caso se almacena la información relacionada con dicha ejecución de la misma forma que en los ficheros de perfil de alumno.

9.1.4. Perfil de administrador

Este fichero se crea cuando se da de alta en el sistema un usuario administrador. Contiene la información relacionada con el administrador de la siguiente forma:

En primer lugar los datos personales, por ejemplo:

```
<DATOS APELLIDO1='Hernández' APELLIDO2='Rodriguez' DNI='33333333'
EMAIL='pepe@ull.es' NOMBRE='Pepe' USERNAME='Pepe'/>
```

A continuación aparecerá información relacionada con los usuarios que este administrador ha dado de alta en el sistema. La única restricción que tienen los usuarios administradores es la imposibilidad de crear usuarios del grupo 'ALUMNOS', ya que estos están asociados a un profesor. De esta forma aparecen en el perfil de los administradores dos categorías, una para los profesores y otra para los administradores. Cada una de estas categorías vendrá especificada por un metatag.

Tanto unos como otros están identificados de la siguiente forma:

```
'<PACO GRUPO='ALUMNO' PASSWD='' USERNAME='ZEBEN'/>
```

En este tag existen tres elementos identificativos que son los únicos necesarios: el nombre de usuario, su clave de acceso al sistema y el grupo al que pertenece. El nombre de usuario aparece duplicado para optimizar los accesos. No son necesarios más campos en este punto pues el nombre de usuario define unívocamente a los usuarios.

Los usuarios administradores también tienen la posibilidad de realizar los test, si bien no tiene mucho sentido que lo hagan. La información relacionada con dichas ejecuciones se almacena de forma idéntica a la de profesores y alumnos en el fichero de perfil del administrador.

9.1.5. Ficheros de Test

Es en este fichero donde se almacena toda la información relacionada con los test. Cada test tiene asociado un fichero. La estructura del fichero la describimos a continuación:

- En primer lugar información general del test:

  ```
  '<TEST DESC='TEST CAMBIO' TEXTO='CAMBIO'>'
  ```

- A continuación ponemos la información de cada uno de los temas del test. El tema comienza indicando el identificador de dicho tema y el nombre del mismo de la siguiente forma:

  ```
  '<TEMA ID='1' TEXTO='PRUEBA'>'
  ```

- Cada tema lleva asociado un conjunto de conceptos. Los conceptos tienen distintos niveles de dificultad distinguidos en cuatro grupos: simple, intermedio, complejo y avanzado. Además llevan asociado un identificador y el nombre del concepto. Estos conceptos comienzan de la siguiente forma:

```
<CONCEPTO ID='1' NIVEL='SIMPLE' TEXTO='TIPOS'>'
```

- Cada concepto lleva asociado un conjunto de preguntas. Las preguntas tienen la siguiente información: un identificador, la probabilidad que tiene el alumno de adivinar la pregunta (o lo que es lo mismo de acertarla sin saberla), la probabilidad que tiene el alumno de saber la pregunta, la respuesta correcta para dicha pregunta y el enunciado de la misma. Las probabilidades asociadas a cada pregunta determinan en cierto modo su nivel de dificultad. Las preguntas se almacenan de la siguiente forma:

```
'<PREGUNTA IDPREGUNTA='1' PADIVINAR='0.03' PSABER='0.95'
RESULTADO='A' TEXTO='>Cuántos valores puede tener un tipo booleano?'>'
```

- Cada pregunta lleva asociada un conjunto de respuestas. La información almacenada para las respuestas consta de un identificador y la posible respuesta a la pregunta planteada. Las respuestas se almacenan de la siguiente forma:

```
'<RESPUESTA IDRESPUESTA='A' TEXTO='2'> </RESPUESTA>'
```

La información descrita hasta aquí se encuentra anidada de la siguiente forma:

```
<TEMA ID='1' TEXTO='PRUEBA'>
    <CONCEPTO ID='1' NIVEL='SIMPLE' TEXTO='TIPOS'>
        <PREGUNTA IDPREGUNTA='1' PADIVINAR='0.03' PSABER='0.95'
         RESULTADO='A' TEXTO='>Cuántos valores puede tener un
         tipo booleano?'>
            <RESPUESTA IDRESPUESTA='A' TEXTO='2'> </RESPUESTA>
```

Una vez finalizadas las posibles respuestas para una pregunta se cierra dicha pregunta y se abre la siguiente. Lo mismo sucede con el concepto y con el tema. De esta forma se construye el fichero de test.

La creación, edición y borrado de cada uno de estos ficheros sera llevada a cabo por los usuarios según los permisos que cada uno de ellos tenga en el sistema. Así mismo, es el propio sistema el encargado de rellenar algunos de los ficheros de forma automática, como por ejemplo el del perfil del usuario, con los valores que se vayan generando.

9.1.6. Cifrado

En el almacenamiento de los datos se hace imprescindible adoptar algún mecanismo de seguridad. En este caso además resulta de vital importancia debido a que dicho almacenamiento se realiza en ficheros xml. Al margen de la seguridad que aporte el lugar donde se encuentren dichos ficheros debemos adoptar medidas más drásticas con el fin de evitar que usuarios malintencionados accedan a ellos y modifiquen los datos, ya que esto podría ocasionar inestabilidades en el sistema y en el peor de los casos un bloqueo del mismo.

La forma de reforzar y garantizar la seguridad de los ficheros viene de la mano del cifrado. Todos los ficheros relacionados con el sistema (ficheros de test, índice, perfiles...) se encuentran cifrados, de forma que solo el sistema es capaz de averiguar cuales son los datos originales de los mismos.

Para realizar el cifrado de los ficheros hemos utilizado una librería en python llamada 'pycrypto 2.0'[1]. Esta librería lleva asociada diversos algoritmos de cifrado de los cuales hemos seleccionado el AES (Advanced Encryption Standard). Veamos con un poco más de detalle este algoritmo.

El algoritmo AES fue desarrollado por los belgas Vincent Rijmen y Joan Daemen. Es un cifrador de bloque que opera con bloques y claves de longitudes variables, que pueden ser especificadas independientemente a 128, 192 ó 256 bits. El resultado intermedio del cifrado se denomina Estado, que puede representarse como un array rectangular de bytes de cuatro filas. La transformación que tiene lugar en cada vuelta de cifrado está compuesta a su vez de cuatro transformaciones diferentes:

- Sustitución de bytes no lineal, operando independientemente sobre cada uno de los bytes del Estado.
- Desplazamiento de las filas del Estado cíclicamente con offsets diferentes.
- Mezcla de columnas, que se realiza multiplicando las columnas del Estado módulo $x^4 + 1$, consideradas como polinomios en GF(2^8), por un polinomio fijo c(x).
- Adición de la clave de vuelta, en la que se aplica al Estado por medio de un simple XOR. La clave de cada vuelta se deriva de la clave de cifrado mediante el esquema de clave.

El esquema de clave consiste en dos operaciones: expansión de clave y selección de clave de vuelta de cifrado.

[1] http://www.amk.ca/python/code/crypto.html

El proceso de cifrado consta de tres pasos: una adición inicial de la clave de vuelta, n-1 vueltas de cifrado y una vuelta final.

Para ver una descripción detallada del algoritmo, se puede consultar el documento de los autores, AES Proposal: Rijndael[1].

[1]http://www.esat.kuleuven.ac.be/ rijmen/rijndael

9.2. Política de Usuarios

Con la implementación de un sitio web desde el cual se permite una manipulación mas cómoda tanto de la construcción de los test como de la ejecución de los mismos, nos encontramos con la necesidad de implementar la política de usuarios que a continuación se describe. Con esta política se pretende evitar usos mal intencionados de los recursos que ofrece el sitio, así como ofrecer un espacio personalizado a cada uno de los usuarios que accedan al mismo.

9.2.1. Roles

Para empezar veamos un diagrama:

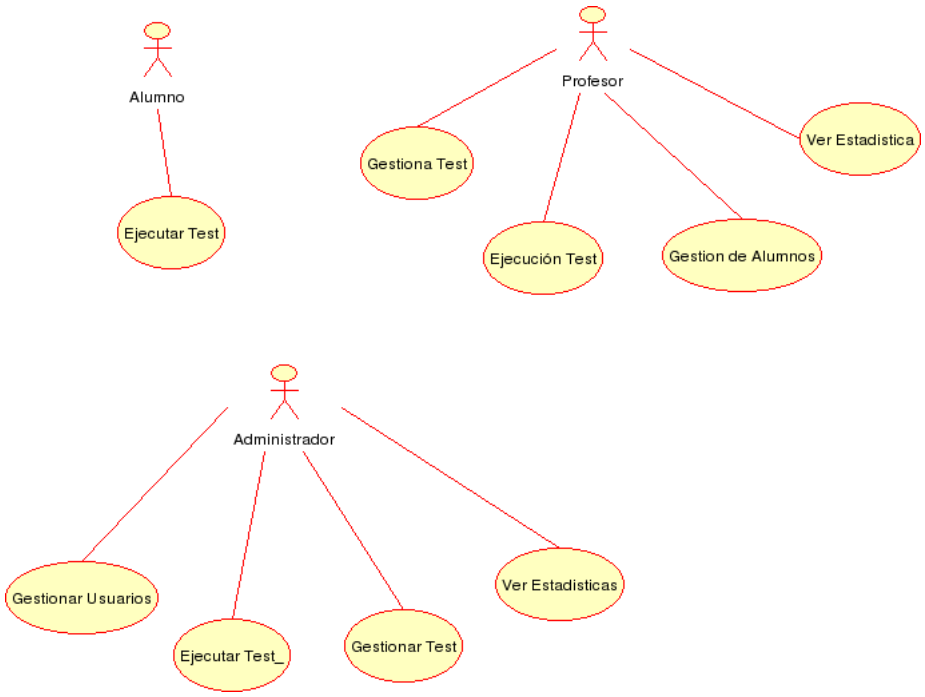


Figura 9.1: Diagrama de Roles

Como podemos observar en el diagrama tenemos tres roles o tipos de usuarios. Veamos en detalle cada uno de ellos:

- Alumnos: es el rol con más restricciones, debido a su naturaleza. Los estudiantes solo podrán ejecutar los tests que les asigne un profesor y ver las estadísticas de los tests que hayan realizado.

- Profesores: el profesor puede realizar las siguientes acciones:

  - Gestionar los test: desde la gesión de test el profesor puede crear, modificar, o borrar sus test, navegando de una forma cómoda a través de los mismos.
  - Ejecutar test: podrá ejecutar los test que haya creado con el fin de probarlos para comprobar su buen funcionamiento.
  - Gestión de Alumnos: cada profesor deberá gestionar los alumnos que puedan acceder a sus test. Dentro de dicha gestión encontramos las funciones básicas de inserción, modificación y borrado.
  - Ver Estadísticas: en este apartado los profesores podrán observar la evolución de sus alumnos observando los progresos que hayan realizado y los fallos cometidos al realizar los test.

- Administrador: es el usuario con más privilegios. Tiene acceso a todas las posibilidades que ofrece el sitio; así mismo, debe realizar determinadas operaciones que por seguridad le están restringidas al resto de los usuarios. Veamos sus posibilidades:

  - Gestionar los tests: esta labor está definida para los profesores, no obstante, puede resultar interesante la posibilidad de que el administrador tenga acceso a estas acciones.
  - Ejecutar test: esta opción es fundamentalmente para los alumnos pero al igual que la opción anterior, el administrador tiene la posibilidad de hacer uso de ella en caso de que lo crea conveniente.
  - Gestionar usuarios: esta es la opción que le distingue del resto, ya que es el único que tendrá permisos para crear usuario de tipo administrador y profesor. La única restricción para un usuario administrador es la de crear usuarios de tipo alumno ya que estos estan vinculados necesariamente a un profesor.
  - Ver estadísticas: el administrador tendrá acceso también a la visualización de las distintas estadísticas generadas en el sitio web.

Todos los usuarios que quieran acceder al sitio web deben estar registrados. La primera acción que deben realizar todos los usuarios es la de 'login', accediendo así a un entorno personalizado desde el cual podrán realizar las tareas que le sean permitidas.

9.2.2. Espacio de Trabajo

Cuando un usuario profesor accede al sitio entra en un espacio personalizado en el cual podrá crear sus test y dar de alta a sus alumnos, sobre los cuales podrá realizar las acciones de gestión que tenga asignadas. Un usuario profesor no puede ver test de otro usuario profesor ni tampoco sus alumnos. Podemos entender este entorno como un espacio de trabajo individual y cerrado al que solo tiene acceso dicho usuario profesor y el usuario administrador.

Cuando el que accede al sitio es un usuario alumno, también accede a un espacio restringido dependiente del usuario profesor que lo haya creado, es decir, el alumno solo tendrá acceso a los tests del usuario profesor que lo haya creado. De esta forma si un alumno quiere ejecutar dos test de profesores distintos tendrá que acceder al sistema con dos usuarios alumnos distintos, uno para cada profesor, ya que desde un espacio de trabajo no se pueden ver los test creados en otro (como explicamos anteriormente). Podemos entender más facilmente esta política si la identificamos con el registro de fichas que realiza cada profesor al comienzo del curso, ya que el alumno debe rellenar tantas fichas como profesores tenga.

El usuario administrador tiene acceso a cada uno de esos entornos personales ya que dispone de permisos para realizar cualquier acción en el sistema. Es por ello por lo que se espera un comportamiento correcto de los usuarios que tengan categoría de administrador y se recomienda que el número de usuarios con este perfil sea el más reducido posible.

9.2.3. Implementación de la política

Los datos realacionados con los roles y los perfiles de usuarios se encuentran almacenados en ficheros XML cuya estructura se explica en el capítulo 'Alamacenaminto de los datos'. Veamos, groso modo, como trabaja el sitio:

- Cuando se crea un usuario, se crean los ficheros de datos asociados (las operaciones de modificación y borrado se realizan sobre esos ficheros).
- Cuando un usuario se valida en el sistema se comprueba que esos datos son correctos comparándolos con los de los ficheros.
- Cuando un usuario accede al sistema se verifican los permisos que posee y en consecuencia se le muestran las opciones a la que puede acceder.
- Se le facilita la posibilidad de realizar las operaciones que considere oportunas dentro de sus limitaciones.

- Cuando abanona el sistema se eliminan los datos del perfil cargados durante la sesión activa del usuario.
- Cuando se elimina un usuario del sistema su fichero de perfil se mueve a una carpeta histórico, desde donde podrá ser recuperado por si fuera preciso.

Notas:

- Si un usuario deja una sesión inactiva durante un tiempo prolongado el sistema lo expulsa del mismo.
- Aunque pueda resultar obvio, un usuario solo podrá acceder al sistema si otro usuario con los privilegios suficientes le ha dado de alta en el mismo. Dicho de otra forma, no se permite el autoregistro.

Capítulo 10

Posibles mejoras

En este apartado veremos posibles mejoras que se pueden realizar en el personaje virtual, en el portal web y en el test adaptativo. Además de la mejora sobre los códigos implementados veremos algunos hilos posibles de desarrollo y expansión del proyecto. Los temas abordados ofrecen un sinfin de posibilidades, de las cuales expondremos las que consideramos más viables.

10.1. Mejoras para Hevah

A continuación veremos posibles mejoras que se pueden incorporar al personaje virtual para hacerlo más agradable, complejo y funcional.

Veamos algunas posible mejoras que se pueden añadir al entorno:

- Una posible mejora va directamente relacionada con el entorno en el que se muestra el personaje. A pesar de la complejidad asociada al escenario y su múltiple funcionalidad, nuestros conocimientos de bellas artes son limitados, así que con la ayuda de profesionales en el tema se podría mejorar el aspecto del escenario, convirténdolo en algo más amigable para el usuario.

- Otra de las mejoras que se puede incorporar al entorno va relacionada con los sonidos que se producen a lo largo de la interacción. Los sonidos que actualmente están incorporados pretenden ser lo más familiares al entorno y más concretamente al objeto que los produce. Como comentamos anteriormente, un cambio significativo en el escenario debe suponer también un cambio en los sonidos asociadso al mismo. La elección de varias bandas sonoras que intenten hacer más confortable la interacción con el usuario sería otra de las cosas deseables.

Hablemos ahora de mejoras relacionadas con el personaje en sí:

- Hemos procurado dotar de la mayor naturalidad posible al personaje en sus movimientos; no obstante, se podrían añadir algunos nuevos con el fin de mejorar aún más esa naturalidad. En este sentido, y con respecto a los movimientos bucales se podrían generar nuevos ficheros de claves para que los movimientos resultaran reales para otros idiomas. Con respecto a esto, y una vez conseguidos los movimientos naturales de la boca para la pronunciaión en otros idiomas, sería de interes la posibilidad de cambiar el idioma del sinetizador de voz (dentro de lo que este permita).

- Otra mejora interesante consistiría en dotar de cuerpo entero al personaje. A pesar de que lo fundamental para este proyecto resulta ser la cabeza, el cuerpo daría una mayor sensación de realidad. Una vez construido, habría que proporcionarle una movilidad natural.

- Un problema que nos encontramos desde el comienzo del proyecto es el sintetizador de voz. Todas las voces que encontramos resultaron muy artificiales, aún más cuando se trataba del español. Una mejora interesante sería mejorar el sintetizador para conseguir una voz más cordial y agradable, a la vez que más humana.

Dentro de las dificultades con las que nos encontramos, una de relativa importancia fue la imposibilidad de generar un ejecutable a partir del personaje en Blender. Básicamente el problema proviene de una llamada que hacemos desde los script en python a la librería de Blender, ya que cuando se genera el ejecutable dicha librería no resulta accesible. Resultaría interesante poder generar un ejecutable con el presonaje virtual.

De forma más ambiciosa se podría plantear la posibilidad de realizar una red de agentes colaborativos. Estos agentes se comunicarían entre sí para pasarse información que pudiera resultar de utilidad. Dicha información aumentaría en gran medida la funcionalidad del agente con lo cual sus tares podrían aumentar su nivel de complejidad. Veamos esto con un ejemplo: un usuario 'U' hace un test 'X' frente al agente 'A' y luego pretende realizar el mismo test 'X' frente a otro agente 'B'. Si el agente 'B' pudiera preguntar al agente 'A' si tiene algún test del usuario 'U', su nivel de conocimiento sobre dicho usuario ya no sería nulo, sino que tendría una base donde apoyarse. Por supuesto esto es un ejemplo ínfimo con respecto a las posibilidades que la colaboración entre agentes supone.

10.2. Mejoras para el portal

Al igual que en el caso del personaje virtual, la estética del portal web puede resultar susceptible de cambios. A pesar de resultar uno de los aspectos de menor importancia, con el consejo de expertos en la materia, quizá se podría conseguir un entorno más confortable para el usuario. No obstante, hemos intentado crear un espacio homogéneo e intuitivo para facilitar el acceso a cualquier tipo de usuario.

Otra de las mejoras de interés consistiría en la posibilidad de almacenar ciertas variables de usuario con el fin de presentarle en sucesivas sesiones un entorno más personalizado y acorde con sus gustos. Si bien esto aún no esta implementado, sí que hemos dejado la puerta abierta para que resulte una mejora fácil y cómoda de implementar.

Con respecto a la mejora mencionada anteriormente, y utilizando técnicas de agentes inteligentes o robots, se podría conseguir una personalización más detallada del entorno de trabajo de cada usuario. Se podría implementar un agente que aprendiera de la forma de trabajo del usuario para facilitarle dicho trabajo en el futuro.

Debido a que el proyecto incorpora un agente virtual en 3D quizá sería de interés también disponer en el web de algún agente, al menos en 2D, que realice una tarea similar a la del personaje virtual. Con esto conseguiriamos despertar la curiosidad del usuario además de facilitar su trabajo en el sitio. En el caso de la inclusión de este agente, además se podría utilizar como un ayudante para usuarios que estuvieran poco familiarizados con el portal.

10.3. Mejoras globales

Podría ser deseable la posibilidad de cambiar la respuesta tipo test por una respuesta introducida el usuario en lenguaje natural. Para ello sería necesario implementar un modulo que fuera capaz de procesar dicha respuesta y extraer información de la misma con el fin de evaluarla.

El agente también podría resultar útil para otras labores, como un sistema de ayuda. Ahora el personaje trataría de ayudar y aconsejar al usuario sobre algún tema concreto. El personaje sería así como un experto en alguna materia sobre la que el usuario solicita información.

Capítulo 11

Conclusiones

Tras muchas horas de trabajo e investigación, hemos construido un complejo sistema de realización de test adaptativos. Este sistema aborda dos frentes distintos pero no incompatibles, por un lado, un personaje virtual que de forma llamativa y amigable pretende hacer más amena la realización de los test para el usuario. No debemos olvidar nuestro compromiso de facilitar en la mayor medida posible la labor del usuario, por ello, con este personaje creemos haber conseguido un compendio de trabajo y entretenimiento. Por otro lado hemos construido un completo interfaz web desde el cual resulta muy cómoda la gestión de los test adaptativos con los que se pretenda trabajar. Además este portal web abre un sin fin de posibilidades para el trabajo concurrente de profesores (gestionando y probando sus test) y de alumnos (ejecutando los test del profesor).

Hevah (Hevah es un EValuador Adaptativo Humanoide) es nuestro personaje virtual. Desde el principio estaba clara la idea de crear un personaje virtual lo más humano posible, lo que no quedaba muy claro es como ibamos a conseguirlo. Después de algúnos meses buscando las herramientas apropiadas nos decantamos por Blender (las razones ya han sido expuestas en detalle), y al estudiar esta herramienta observamos las enormes posibilidades que nos ofrecía su módulo de juegos para nuestro fin. No resulto nada fácil el modelado del personaje, más aún cuando no conocíamos la herramienta, pero con muchas ganas y algún que otro tutorial conseguimos crear la malla principal de la cabeza de hevah. En este punto nuestros esfuerzos estaban divididos en dos frentes, por un lado el modelado del personaje y por otro en el estudio del sintetizador de voz (que también nos dio bastantes quebraderos de cabeza). A partir de aqui se fue entremezclando animación y modelado, el pelo, los ojos, el movimiento de la boca... hasta conseguir la cabeza actual. Todavía quedaba las texturas y la iluminación, dos aspectos que para nada resultaron triviales. Conseguido nuestro objetivo principal, abordamos el improvisado escenario que

teniamos hasta el momento añadiendole objetos y funcionalidad para intentar hacer más amena la interacción.

El diseño del espacio web surgio con la necesidad de manipular los ficheros XML donde se almacena la información para el test adaptativo. Resulta bastante complicado, sobre todo para usuarios inexpertos, el trabajo directo con los ficheros xml y optamos por realizar una interfaz para hacer más cómodo dicho trabajo. Después de estudiar algunas alternativas optamos por realizar dicha interfaz en web, ya que proporciona una forma cómoda de trabajar desde cualquier sitio expandiendo asi el alcance del proyecto, al margen de permitir concurrencia de usuarios. La creación de un espacio web, que ya no solo iba a resultar una interfaz gráfica para el tratamiento de los ficheros se convirtio en una entidad con identidad propia y cierta independencia del personaje virtual, ofreciendo un espacio de trabajo para un amplio grupo de personas divididas en dos grupos fundamentales: profesores y alumnos. Una política de usuarios estricta y el cifrado de los ficheros convierten el portal web en un espacio seguro donde los profesores podrán depositar los test y los alumnos realizaran sus pruebas sin que otros puedan manipular sus resultados. Este espacio abre las puertas para que muchos usuarios se aproximen a la realización de test adaptativos de una forma cómoda, lo cual resulta inviable con los test tradicionales en papel.

Aunque en un principio se barajaron otras opciones, la idea de la construcción de un test adaptativo fue la que tuvo más peso, tanto por su viabilidad como por sus características. Tras un estudio de la teoria de test adaptativos bayesianos nos lanzamos a la implementación de un software que fuera capaz de poner en práctica sus fundamentos. Para nuestra sorpresa unos minimos ajustes en las formulas que manipulamos inicialmente dieron muy buenos resultados, si bien es cierto que existió un estudio bastante exhaustivo previo a la implementación. Para desarrollo del test fue necesaria la creación de un motor de inferencias, para su construcción optamos por un lenguaje especifico de inteligencia artificial, decantándonos por CLIPS. Durante el desarrollo del portal web observamos la inviabilidad de la utilización de esta herramienta, optando por generar el motor de inferencia en el lenguaje que esta escrito el resto del código, es decir, Python.

Nuestros últimos intentos consisten en la migración del personaje virtual a la nueva versión de blender, la 2.34. Desde blender 2.25 (la versión que usamos), hasta blender 2.33, las veriones disponibles no incorporaban el módulo de juegos. En la versión 2.33 el módulo de juegos era aún bastante inestable pero ya en la versión 2.34 se consolida. La migración no resulta fácil pues han habido bastantes cambios y esta nueva versión nos llega en la última fase del proyecto haciendo inabordable la conclusión de la migración a tiempo. Otra

migración importante en la que nos encontramos a mitad de camino consiste en convertir a hevah en multiplataforma. El principal inconveniente deriva del sintetizador de voz. En linux hemos optado por usar festival, pero al igual que en la migración de versiones este cambio nos ha llegado fuera de tiempo. Estas migraciones están prácticamente concluidas pero a falta realizar pruebas definitivas para comprobar su correcto funcionamiento hemos preferido no incluirlas en el proyecto.

Como se ha comentado anteriormente, tanto con hevah como con el espacio web, pretendemos acercar lo más cómoda y fácilmente posible al usuario al mundo de los test adaptativos. Consideramos que este tipo de test pueden resultar muy útiles para evaluar el conocimiento de los usuarios sobre diversos temas, resultando menos traumático para el usuario que los test tradicionales.

Capítulo 12

Bibliografía

- '¿Qué es XML?'
  http://www.desarrolloweb.com/articulos/449.php?manual=27
- 'elYsiun.com :: Render Your Imagination'
  http://www.elysiun.com
- 'NicoDigital - El mejor sitio para aprender 3D en español con Blender'
  http://www.nicodigital.com
- 'blender.org :: the dot org era'
  http://www.blender.org
- 'Free Samples of Texture Maps and Clip Art for 2D 3D Graphics'
  http://www.realworldimagery.com/Samples/sample.html
- 'Texture Library'
  http://textures.forrest.cz/
- 'Carlos González Morcillo 'Animación para la comunicación'
  http://www.inf-cr.uclm.es/www/cglez/
- 'IAIC ULL'
  http://www.csi.ull.es/ iaic/
- 'CLIPS: A Tool for Building Expert Systems'
  http://www.ghg.net/clips/CLIPS.html

- 'CLIPS - Aitor San Juan Sánchez'
  http://www.geocities.com/aitorsjs/html/clips.pdf
- 'INTRODUCCION A CLIPS'
  http://intelec.dif.um.es/ gdia/Asignaturas/V33/Practicas/Fpract1.pdf
- 'Programación concurrente'
  http://espira.net/delphi/multithread/
- 'Hilos'
  http://libertonia.escomposlinux.org/story/2002/9/23/16452/3311
- 'Inferencia bayesiana'
  http://www.dccia.ua.es/ miguel/tesis/tesis006.html
- '¿Qué es la inferencia bayesiana?'
  http://www.atheneum.doyma.es/Socios/sala_l/lec06est.htm
- 'Bioestadística, probabilidad'
  http://campusvirtual.uma.es/est_fisio/apuntes/ficheros/estad_uma_04.ppt
- 'Test adaptativos'
  http://www.ugr.es/ icem2002/Ponencias/FernandezAlvarez.PDF
- 'Webware for Python'
  http://webware.sourceforge.net
- 'Cheetah - The Python-Powered Template Engine'
  http://www.cheetahtemplate.org
- 'Python Programming Language'
  http://www.python.org
- 'Mark Hammond's Python Extensions'
  http://starship.python.net/crew/mhammond/
- 'SAPI Reference'
  http://www.microsoft.com/speech

Apéndice A

Manuales de Instalación y Usuario

A.1. Instalación de la parte gráfica

Antes de poder utilizar la parte gráfica del proyecto, es necesario instalar ciertos módulos y aplicaciones. Este es el orden de instalación que recomendamos:

A.1.1. Python 2.0

Es necesario instalar Python 2.0 ya que es el lenguaje de programación utilizado por la versión del Blender que hemos utilizado (Blender Publisher 2.25).

Se puede encontrar en la página web oficial de 'Python'[1]. En nuestro caso hemos utilizado la versión 2.0.1, aunque no descartamos que funcione con versiones cercanas. El nombre del ejecutable es 'Python-2.0.1.exe'

A.1.2. Win32API

En el desarrollo del proyecto ha sido necesario hacer uso de algunas funcionalidades de Windows bajo Python, por eso es necesario tener instalada la API de Python para este sistema operativo, o sea, la Win32API.

Este módulo se puede descargar desde la página de 'Mark Hammond'[2].

En nuestro caso es necesario instalar la versión para Python 2.0, o sea, el fichero 'win32all-144.exe'.

[1] http://www.python.org
[2] http://starship.python.net/crew/mhammond/

A.1.3. Microsoft Speech API

Para hacer que el personaje hable, hay que instalar la SAPI de MS. Puede encontrarse en la página de Microsoft[1].

La versión utilizada es la 4, que en el momento de desarrollo del proyecto, era la última versión que poseía voces en español.

El nombre del fichero es 'spchapi.exe'. Opcionalmente se puede instalar la extensión para el 'Panel de control' de Windows ('SpchCpl.exe').

Las voces en español se encuentran en el fichero 'lhttsspe.exe'.

A.1.4. Blender Publisher 2.25

Finalmente, sólo queda por instalar el Blender Publisher 2.25.

Blender puede ser descargado desde su página oficial 'blender.org'[2]. El nombre del fichero es 'blender-publisher-2.25-windows.exe'.

A.2. Manual de usuario del personaje animado

Al ejecutar el fichero que contiene el personaje animado en el módulo de juegos, entraremos en una ventana con este aspecto:

Figura A.1: Entorno del personaje animado

Lo primero que hará el personaje será decir una frase a modo de saludo.

[1] http://www.microsoft.com/speech/
[2] http://www.blender.org

A.2.1. Realizar el test

Para poder comenzar el test, el usuario deberá pulsar sobre la tecla verde.

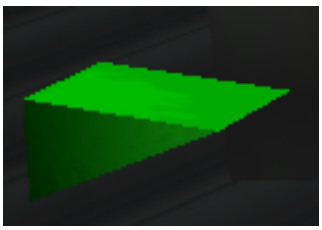


Figura A.2: Tecla de comienzo en el módulo de juegos

Tras esto, la tecla desaparecerá.

En este momento, la cámara se acercará a la cabeza y esta planteará la primera pregunta del test. En el momento en que se aleje la cámara, el usuario podrá responder la pregunta.

La respuesta será dada por el usuario mediante el teclado, pulsando la tecla que se corresponda con la respuesta que crea adecuada.

En el momento de presionar la tecla para responder, la elección realizada se reflejará en el cuadro situado a la izquierda de la cabeza.

Figura A.3: Cuadro reflejando las respuesta elegida

Tras esto, el personaje pasará a la siguiente pregunta, y así sucesivamente hasta terminar el test.

A.2.2. Interacciones con el entorno

Algunos elementos del escenario pueden ser modificados por el usuario:

- Persiana: puede ser subida y bajada pulsando en cualquier momento las teclas situadas a la derecha.

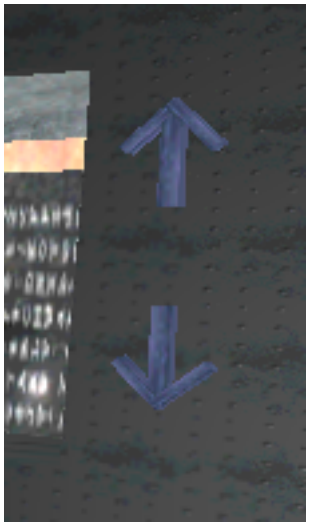


Figura A.4: Botones para subir/bajar la persiana

- Techo: puede ser plegado o desplegado mediante las teclas que se encuentran encima de la persiana.

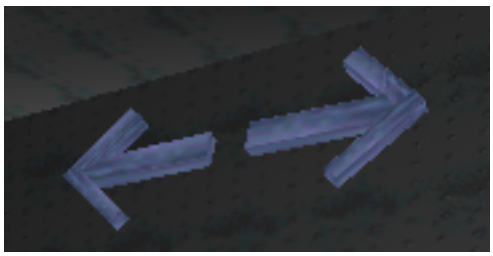


Figura A.5: Botones para plegar/desplegar el techo

- Luz: la bombilla puede ser apagada o encendida dependiendo de si pulsamos con el botón derecho o con el izquierdo sobre la tecla que se encuentra debajo de la lámpara.

Para salir en cualquier momento del módulo de juegos hay que pulsar la tecla con forma de aspa (la opción de presionar ESC para salir no es muy conveniente en este caso).

Figura A.6: Botón para encender/apagar la luz

Figura A.7: Botón para salir del módulo de juegos

A.3. Manual de usuario del portal

En este manual se pretende explicar la funcionalidad del portal indicando los pasos que debe realizar un usuario para efectuar las operaciones que considere oportunas y/o que le sean permitidas.

A.3.1. Bienvenida

En esta página de bienvenida encontramos los campos de validación, donde los usuarios registrados del sistema deben introducir su 'nombre de usuario' y su 'clave' para acceder al mismo. Los usuarios no registrados tan solo podrán ver esta página de bienvenida.

A.3.2. Principal

Una vez que un usuario registrado accede al sistema se le presentarán las opciones que tenga permitidas. Veamos que opciones tiene cada tipo de usuario:

- Alumnos: Ejecución de test y Estadísticas
- Profesores: Gestión de alumnos, Gestión de test, Estadísticas (de sus alumnos) y Ejecución de test

- Administrador: Gestión de usuarios (profesores y administradores), Gestión de test, Estadísticas y Ejecución de test.

A.3.3. Ejecución de Test

Todos los usuarios registrados en el sistema tienen permisos para utilizar esta opción pero con ciertas restricciones.

Para ejecutar un test simplemente habrá que ir a la sección de tests, donde aparecerán los tests disponibles, seleccionar el deseado, y darle a comenzar. Si es un alumno se le mostrarán los test del profesor que le ha dado de alta en el sistema. Si es un profesor se le muestran los test que ha creado. Si es un administrador se le muestran todos los test disponibles.

Una vez que se selecciona el test el procedimiento es sencillo, se van mostrando preguntas y sus correspondientes respuestas. Para responder a una pregunta simplemente se selecciona la respuesta que se considera correcta; automáticamente se mostrará la siguiente pregunta.

Una vez finalizado al test se presenta al usuario la nota del mismo.

A.3.4. Estadísticas

Esta opción también es común a todos los usuarios, y al igual que en el caso de la ejecución de los test, presenta restricciones para los distintos grupos de usuarios.

Si un alumno entra en estadísticas se le mostrarán los test que ha realizado. Si selecciona uno de los test se le muestra una primera tabla con los aciertos y fallos en función de la dificultad de las preguntas y de los conceptos. En esta pantalla además tiene la opción de acceder al histórico de las preguntas que le fueron realizadas; para ello solo tiene que hacer click en el boton con la etiqueta 'histórico'. En el histórico se le mostrará todas las preguntas que se le hicieron para ese test y si respondió correctamente o no.

A.3.5. Gestión de los tests

Cuando un usuario autorizado (Profesor a Administrador) entra a la zona de Tests, paarecen las opciones asociadas a la gestión de los mismos. Se puede entender dividida en niveles atendiendo a la estructura de como se construyen los test adaptativos. Dicha estructura de mayor a menor importancia es: test, tema, concepto, pregunta y respuesta.

En la zona central se mostrarán las instancias existentes del item correspondiente.

Las opciones son las siguientes:

Nuevo

Permite crear un nuevo item del nivel correspondiente.

Cambiar Nombre

Permite modificar los datos correspondientes a la instancia seleccionada.

Modificar

Accede al nivel inferior en la jerarquía que comentamos antas; es decir, si estamos viendo tests, seleccionamos uno, y accedemos a esta opción, pasaremos a ver los temas de este test.

Ordenar

Como su propio nombre indica, permite reordenar los items.

Eliminar

Elimina la instancia seleccionada.

A.3.6. Gestión de usuarios

Esta es la otra zona restringida a profesores y administradores. Inicialmente se muestra un listado con los usuarios del sistema (en el caso del profesor solo de sus alumnos). Seleccionando un usuario pasamos a una nueva página donde se muestran los datos de dicho usuario. Aquí tienen la posibilidad de realizar las operaciones de gestión básicas que consideren oportunas, más concretamente:

- Nuevo: se le muestra un formulario con los datos que debe rellenar para dar de alta al nuevo usuario. Una vez introducidos éstos, se debe aceptar (haciendo click en el botón). El usuario es insertado y se le devuelve a la pantalla principal de gestión de usuarios.

- Modificar: si se selecciona esta opción se pasa a una nueva página donde se mostrarán los datos del usuarios seleccionado. Aquí se realizaran las modificaciones oportunas.

Una vez realizadas se 'acepta' las modificaciones volviendo a la página principal de gestión de usuarios.

- Eliminar: una vez seleccionado el usuario a borrar se hace click en el botón al efecto, el usuario es eliminado y se vuelve a la página principal de gestión de usuario.

A.3.7. Salir

En todo momento el usuario que se encuentra trabajando en el sitema dispone de la posibilidad de abandonar el mismo. Para ello sencillamente debe hacer click sobre la pestaña que indica salir. Esta opción aún pareciendo trivial debe tenerse muy en cuenta puesto que si un usuario abandona su puesto de trabajo dejando su sesión abierta otro usuario podría hacer uso de ella, esto toma un matiz extremo si el usuario que está trabajando es un administrador.

Apéndice B

Código común

B.1. Evaluar

```
##                  HEVAH
## Proyecto de fin de carrera de Ingenieria Informática
## Universidad de La Laguna
## Tutora: Carina Gonzalez Gonzalez
## Autores: Fco. Javier Medina Santos
##     Daniel J. Díaz Gonzalez
##     Zebenzui Pérez Ramos
## Descripcion: Se encarga de evaluar el conocimiento del usuario.
##              Primero se genera una tabla con las probabilidades a priori
##              Estas probabilidades vienen determinadas por el nivel del
##              usuario que realiza el test y por la dificultad de los
##              conceptos. A continuación y a medida que se el van
##              presentando preguntas al usuario se va recalculando la
##              probabilidad que tiene éste de saber los conceptos que se
##              le van presentando. Una lógica de reglas determina el nivel
##              de las preguntas o conceptos que se le van presentando en
##              función de las probabilidades obtenidas. Cuando se produce
##              un cambio de concepto las probabilidades se inicializan.

import string

class Evaluar:
```

```
sabeconcepto = 0    #Probabilidad de saber el concepto
acumulado = []      #La probabilidad despues de cada respuesta
nivel_preg = ""
nivel_conc = ""
acertada = 0

TablaProb = {}      #Probabilidades a priori

def __init__ (self, lista, fichero_reglas):
 #Inicializamos las probabilidades a priori
    self.TablaProb ["novel", "simple"] = lista[0]
    self.TablaProb ["novel", "intermedio"] = lista[1]
    self.TablaProb ["novel", "complejo"] = lista[2]
    self.TablaProb ["novel", "avanzado"] = lista[3]

    self.TablaProb ["principiante", "simple"] = lista[4]
    self.TablaProb ["principiante", "intermedio"] = lista[5]
    self.TablaProb ["principiante", "complejo"] = lista[6]
    self.TablaProb ["principiante", "avanzado"] = lista[7]

    self.TablaProb ["medio", "simple"] = lista[8]
    self.TablaProb ["medio", "intermedio"] = lista[9]
    self.TablaProb ["medio", "complejo"] = lista[10]
    self.TablaProb ["medio", "avanzado"] = lista[11]

    self.TablaProb ["experto", "simple"] = lista[12]
    self.TablaProb ["experto", "intermedio"] = lista[13]
    self.TablaProb ["experto", "complejo"] = lista[14]
    self.TablaProb ["experto", "avanzado"] = lista[15]

# Funcion que realiza los cambios pertinentes cuando se produce un
# cambio de concepto.
# Parametros:
#     * user: usuario que esta realizando el test
```

```
#     * concepto: nuevo concepto a evaluar
# Actualiza:
#     * saberinicial: probabilidad a priori de saber el concepto
#     * sabeconcepto: probabilidad de saber el concepto (como no se han
#                     realizado preguntas aún coincide con la inicial)
#     * nivel_conc: nivel del nuevo concepto
#     * acumulado: se inicializa a vacio
def NuevoConcepto (self, user, concepto):
    usuario = user.GetNivel()
    concepto = string.lower (concepto)
    self.saberinicial = self.TablaProb[usuario, concepto]
    self.sabeconcepto = self.TablaProb[usuario, concepto]
    self.nivel_conc = concepto
    self.acumulado = []

# Calcula la nueva probabilidad que tiene el usuario de saber el
# concepto si ha respondido de forma correcta a la pregunta formulada.
# Ademas almacena esa probabilidad en el acumulado.
# Parametros:
#     * prob_saber: probabilidad que tiene el usuario de acertar la
#                   pregunta sabiendo el concepto.
#     * prob_adivinar: probabilidad que tiene el usuario de adivinar
#                      la respuesta a la pregunta.
# Actualiza:
#     * sabeconcepto: probabilidad de saber el concepto
#     * acumulado: probabilidades de saber el concepto de las preguntas
#                  realizadas de dicho concepto
#     * nivel_preg: nivel de la pregunta actual
#     * acertada: indicando que ha acertado

def PreguntaAcertada (self, prob_saber, prob_adivinar):
    #P(p1+)    = P (p1+/c+). P(c+) + P(p1+/c-). P(c-)
    prob_acertar = prob_saber * self.saberinicial +
 prob_adivinar * (1- self.saberinicial )
```

```
        #P*(c+) = P(c+/p1+)= [P(c+) . P (p1+/c+) ]/ P(p1+)
        self.sabeconcepto = self.saberinicial * prob_saber / prob_acertar

        self.acumulado.append(self.sabeconcepto)
        self.nivel_preg =  self.NivelPregunta (prob_saber)
        self.acertada = 1

    # Calcula la nueva probabilidad que tiene el usuario de saber el
    # concepto si ha respondido de forma incorrecta a la pregunta
    # formulada. Ademas almacena esa probabilidad en el acumulado.
    # Parametros:
    #     * prob_saber: probabilidad que tiene el usuario de acertar la
    #                   pregunta sabiendo el concepto.
    #     * prob_adivinar: probabilidad que tiene el usuario de adivinar
    #                      la respuesta a la pregunta.
    # Actualiza:
    #     * sabeconcepto: probabilidad de saber el concepto
    #     * acumulado: probabilidades de saber el concepto de las preguntas
    #                  realizadas de dicho concepto
    #     * nivel_preg: nivel de la pregunta actual
    #     * acertada: indicando que ha fallado
    def PreguntaFallada (self, prob_saber, prob_adivinar):
        #P(p1-)    = P (p1-/c+). P(c+) + P(p1-/c-). P(c-)
        prob_fallar = (1 - prob_saber) * self.saberinicial +
                (1 - prob_adivinar)
                * (1 - self.saberinicial )

        #P*(c+) = P(c+/p1-)= [P(c+) . P (p1-/c+) ]/ P(p1-)
        self.sabeconcepto = self.saberinicial * (1 - prob_saber)
                      / prob_fallar

        self.acumulado.append(self.sabeconcepto)
        self.nivel_preg =  self.NivelPregunta (prob_saber)
        self.acertada = 0
```

```
# Realiza la media de las probabilidades obtenidas en las distintas
# preguntas del concepto para determinar el nivel de conocimiento
# que tiene el usuario sobre el mismo.
# Devuelve:
#     * el resultado de esa media
def GetConocimiento (self):
    temp = 0
    for i in self.acumulado:
        temp = temp + i
    return (temp/len (self.acumulado))

# Funcion que determina en conjuntos bayesianos el nivel de la
# pregunta.
# Parametros:
#     * prob: probabilidad de responder correctamente a la pregunta
# Devuelve:
#     * el valor del conjunto bayesiano
def NivelPregunta (self, prob):
    if (prob > 0.7):
        return ("baja")
    elif ((prob <= 0.7) and (prob > 0.4)):
        return ("media")
    else:
        return ("alta")

# Funcion que determina el nivel de la nueva pregunta en funcion de si
# ha acertado la actual o no.
# Parametros:
#     * resp: respuesta dada por el usuario
#     * preg: pregunta actual (que se le ha planteado)
# Devuelve:
#     * El nivel de la nueva pregunta o concepto
def NuevaPreg (self, resp, preg):
    resp = resp.split('\n')[0]
    if (preg.EsCorrecta(resp)):
```

```
            self.PreguntaAcertada(preg.GetSaber(), preg.GetAdivinar())
        else:
            self.PreguntaFallada(preg.GetSaber(), preg.GetAdivinar())
        result = self.Reglas(self.GetConocimiento(),
           self.nivel_preg,str(self.nivel_conc))
        return (result)

    ## Nos permite testear si la ultima pregunta fue acertada o no
    def GetTestUltima (self):
        return (self.acertada)

    ## Conjunto de reglas para la inferencia bayesiana
    #  Parametros:
    #      * prob: probabilidad que tiene el alumno de saber el concepto
    #      * nivel_preg: nivel de la pregunta actual
    #      * nivel_conc: nivel del concepto actual
    #  Devuelve:
    #      * El nivel de la proxima pregunta o concepto que se realizara
    def Reglas (self, prob, nivel_preg, nivel_conc):
        aumenta = 0
        disminuye = 0
#Comprobamos si debemos aumentar, disminuir o mantener el nivel
        if (prob > 0.7):
            aumenta = 1
        if (prob <= 0.7) and (prob > 0.4):
            return ('pregunta_' + nivel_preg)
        if (prob < 0.4):
            disminuye = 1

#Vemos como cambia el nivel cuando tenemos que disminuirlo
        if ((nivel_preg == 'baja') and (nivel_conc == 'simple') and
      (disminuye)):
            return('concepto_SIMPLE')
        if (nivel_preg == 'baja') and (nivel_conc == 'intermedio') and
     (disminuye):
```

```
            return('concepto_SIMPLE')
        if (nivel_preg == 'baja') and (nivel_conc == 'complejo') and
     (disminuye):
            return('concepto_INTERMEDIO')
        if (nivel_preg == 'baja') and (nivel_conc == 'avanzado') and
     (disminuye):
            return('concepto_COMPLEJO')

        if (nivel_preg == 'media') and (disminuye):
            return('pregunta_BAJA')

        if (nivel_preg == 'alta') and (disminuye):
            return('pregunta_MEDIA')

#Vemos como cambia el nivel cuando tenemos que aumentarlo
        if (nivel_preg == 'alta') and (nivel_conc == 'simple') and
     (aumenta):
            return('concepto_INTERMEDIO')
        if (nivel_preg == 'alta') and (nivel_conc == 'intermedio') and
     (aumenta):
            return('concepto_COMPLEJO')
        if (nivel_preg == 'alta') and (nivel_conc == 'complejo') and
     (aumenta):
            return('concepto_AVANZADO')
        if (nivel_preg == 'alta') and (nivel_conc == 'avanzado') and
     (aumenta):
            return('concepto_AVANZADO')

        if (nivel_preg == 'media') and (aumenta):
            return('pregunta_ALTA')

        if (nivel_preg == 'baja') and (aumenta):
            return('pregunta_MEDIA')
```

B.2. procxml

```
##                  HEVAH
## Proyecto de fin de carrera de Ingenieria Informática
## Universidad de La Laguna
## Tutora: Carina Gonzalez Gonzalez
## Autores: Fco. Javier Medina Santos
##     Daniel J. Díaz Gonzalez
##     Zebenzui Pérez Ramos
## Descripcion: Contiene las cuatro clases siguientes:
##              * Convertir: encargada del tratamiento de caracteres
##                           extraños
##              * Pregunta: contiene y gestiona toda la información que
##                          lleva asociada una pregunta, incluidas las
##                          respuestas.
##              * Concepto: contiene y gestiona toda la información que
##                          lleva asociado un concepto, incluidas las
##                          preguntas.
##              * Tema: contiene y gestiona toda la información que lleva
##                      asociado un tema, incluidos los conceptos.
##              * MiXML: extrae la información del fichero de test,
##                       fundamentalmente relacionado con los temas que
##                       tiene el test.

from types import UnicodeType
from xml.dom import minidom
import encodings
import string
import os
from Cifrado import Cifrado

# Clase que se encarga del correcto funcionamiento de caracteres
# conflictivos como las tildes.
class Convertir:
    def __init__ (self, texto):
```

```
        temp1 = ''
        temp2 = ''
        try:
            for i in texto:
                temp1 = temp1 + self.convertir(ord(i))
                temp2 = temp2 + self.convertir2(ord(i))
        except:
            pass
        self.result = [temp1, temp2]


    def convertir (self, code):
        temp = ''
        if (code == 225):
            temp = temp +'$'+repr(code)
        elif (code == 233):
            temp = temp +'$'+repr(code)
        elif (code == 237):
            temp = temp +'$'+repr(code)
        elif (code == 243):
            temp = temp +'$'+repr(code)
        elif (code == 250):
            temp = temp +'$'+repr(code)
        elif (code == 252):
            temp = temp +'$'+repr(code)
        elif (code == 241):
            temp = temp +'$'+repr(code)
        elif (code == 191):
            temp = temp +'$'+repr(code)
        else:
            temp = chr (code)
        return (temp)


    def convertir2 (self, code):
```

```
        temp = ''
        if (code == 225):
            temp = temp +'a'
        elif (code == 233):
            temp = temp +'e'
        elif (code == 237):
            temp = temp +'i'
        elif (code == 243):
            temp = temp +'o'
        elif (code == 250):
            temp = temp +'u'
        elif (code == 252):
            temp = temp +'u'
        elif (code == 241):
            temp = temp +'n'
        elif (code == 191):
            temp = temp +''
        else:
            temp = chr (code)
        return (temp)

    def GetResultado (self):
        return self.result



class Pregunta:
    # Se inicializan los valores de la pregunta y de cada una de las
    # respuestas que tiene asociadas.
    # Parametros:
    #     * nodo: contiene la informacion de la pregunta y sus respuestas
    def __init__ (self, nodo):
        self.respuestas = {}
        self.correcta =""
        self.nodo = nodo
```

```
        self.probsaber = nodo.getAttribute('PSABER')
        self.probadivinar = nodo.getAttribute('PADIVINAR')
        self.enunciado = nodo.getAttribute ('TEXTO')
        self.correcta = nodo.getAttribute ("RESULTADO")
        self.id = nodo.getAttribute("IDPREGUNTA")
        lista_respuestas = nodo.getElementsByTagName ('RESPUESTA')
        for respuesta in lista_respuestas:
            self.respuestas[respuesta.getAttribute('IDRESPUESTA')] =
respuesta.getAttribute ('TEXTO')

    # Devuelve el conjunto de respuestas de una pregunta o la respuesta
    # especificada.
    # Parametros:
    #     * opcion: indica que respuesta se desea obtener
    def GetRespuesta (self, opcion = ""):
        # Si no se indica nada se devuelven todas las respuestas
        if (opcion == ""):
            result = []
            for resp in self.respuestas.keys():
                result.append (self.respuestas[resp])
            return (result)
        else:
    # Se devuelve la respuesta solicitada
            if (self.respuestas.has_key (opcion)):
                return (self.respuestas[opcion])
            else:
                return ("RESPUESTA INEXISTENTE")

    # Indica si la respuesta dada a una pregunta es correcta o no
    # Devuelve:
    #     * 0: ha fallado, 1: ha acertado
    def EsCorrecta (self, opcion):
        if (self.correcta == opcion):
            return (1)
        else:
```

```
            return (0)

    # Devuelve el conjunto bayesiano de dificultad de la pregunta en
    # funcion de sus probabilidades asociadas.
    def GetNivel (self):
        prob = self.GetSaber() +  self.GetAdivinar()
        if (prob > 0.7):
            return ("BAJA")
        elif ((prob <= 0.7) and (prob > 0.4)):
            return ("MEDIA")
        else:
            return ("ALTA")

    # Devuelve el enunciado de la pregunta
    def GetEnunciado (self):
        return (self.enunciado)

    # Devuelve la probabilidad que tiene el usuario de acertar la pregunta
    # sabiendo el concepto.
    def GetSaber (self):
        return string.atof(self.probsaber)

    # Devuelve la probabilidad que tiene el usuario de adivinar la pregunta
    def GetAdivinar (self):
        return string.atof(self.probadivinar)

    # Devuelve el identificador de la pregunta
    def GetID (self):
        return self.id


class Concepto:
    # Se inicializan los valores del concepto y de cada una de las
    # preguntas que tiene asociadas. Estas preguntas se distribuyen en tres
    # listas atendiendo a su nivel de dificultad (alta, media y baja), de
```

```
    # forma que luego al acceso resulte más cómodo.
    # Parametros:
    #     * nodo: contiene la informacion del concepto y sus preguntas
    def __init__ (self, nodo):
        self.preguntas={}
        self.ptr={}
        self.ptr['BAJA'] = -1
        self.ptr['MEDIA'] = -1
        self.ptr['ALTA'] = -1
        self.preguntas['BAJA']=[]
        self.preguntas['MEDIA']=[]
        self.preguntas['ALTA']=[]
        self.enunciado=""
        self.nodo = nodo
        self.enunciado = nodo.getAttribute ('TEXTO')
# Cogemos las preguntas del nodo
        lista_preguntas = nodo.getElementsByTagName('PREGUNTA')
# Las repartimos en las listas según su dificultad
        for preg in lista_preguntas:
            prob_total = string.atof(preg.getAttribute('PSABER')) +
             string.atof(preg.getAttribute('PADIVINAR'))
            self.preguntas[self.NivelPregunta(prob_total)].append(preg)
        self.id = nodo.getAttribute ('ID')

    # Devuelve el conjunto bayesiano de dificultad de la pregunta en
    # funcion de sus probabilidades asociadas.
    def NivelPregunta (self, prob):
        if (prob > 0.7):
            return ('BAJA')
        if (prob <= 0.4):
            return ('ALTA')
        return ('MEDIA')

    # Devuelve una pregunta de ese concepto del nivel dado
    # Parametros:
```

```
    #     * nivel: nivel del que se quiere la pregunta
    def GetPregunta (self, nivel):
     # Si quedan preguntas en ese nivel
        if (self.ptr[nivel] < len (self.preguntas[nivel]) - 1):
    # Seleccionamos y devolvemos la pregunta
            self.ptr[nivel] = self.ptr[nivel] + 1
            nodo = self.preguntas[nivel][self.ptr[nivel]]
            temp = nodo.getAttribute('TEXTO')
            lista = Convertir(temp).GetResultado()
            lista.append (Pregunta(nodo))
            return (lista)
        print "NO MAS PREGUNTAS"
        return ([None,None,None])

    # Devuelve el nivel del concepto
    def GetNivel(self):
        return (self.nodo.getAttribute ('NIVEL'))

    # Devuelve el enunciado del concepto
    def GetEnunciado (self):
        return (self.enunciado)

    # Devuelve el id del concepto
    def GetID (self):
        return self.id


class Tema:
    # Se inicializan los valores del tema y de cada uno de los conceptos
    # que tiene asociados. Estos conceptos se distribuyen en cuatro
    # listas atendiendo a su nivel de dificultad (simple, intermedio,
    # complejo y avanzado), de forma que luego al acceso resulte más cómodo
    # Parametros:
    #     * nodo: contiene la informacion del tema y sus conceptos
    def __init__ (self, nodo):
```

```
        self.ptrconcepto={}
        self.ptrconcepto['SIMPLE'] = -1
        self.ptrconcepto['INTERMEDIO'] = -1
        self.ptrconcepto['COMPLEJO'] = -1
        self.ptrconcepto['AVANZADO'] = -1
        self.lista_conceptos={}
        self.lista_conceptos['SIMPLE']=[]
        self.lista_conceptos['INTERMEDIO']=[]
        self.lista_conceptos['COMPLEJO'] = []
        self.lista_conceptos['AVANZADO'] =[]
        self.titulo = ""
    # Cogemos la lista de conceptos del tema
        lista_aux = nodo.getElementsByTagName('CONCEPTO')
        self.id = nodo.getAttribute("ID")
    # Los distribuimos en las listas
        for concepto in lista_aux:
            self.lista_conceptos[concepto.getAttribute('NIVEL')].append(
                    concepto)

# Devuelve un concepto del nivel dado de ese tema
# Parametros:
#     * nivel: nivel del que se quiere el concepto
def GetConcepto(self, nivel):
    # Si quedan conceptos de ese nivel
        if (self.ptrconcepto[nivel] < len (self.lista_conceptos[nivel])-1):
         # Se devuelve el concepto
            self.ptrconcepto[nivel] = self.ptrconcepto[nivel] + 1
            nodo = self.lista_conceptos[nivel][self.ptrconcepto[nivel]]
            temp = nodo.getAttribute('TEXTO')
            lista = Convertir(temp).GetResultado()
            lista.append (Concepto(nodo))
            return (lista) # [representacion Blender, representacion TTS,
                  concepto]
        print ("NO MAS CONCEPTOS")
        return ([None,None,None])
```

```
    # Devuelve el identificador del concepto
    def GetID (self):
        return self.id



class MiXML:
    # Se extraen del fichero del test los temas que contiene. Estos
    # temas se almacenan en una lista.
    # Parametros:
    #     * fich: fichero que contiene la información del test
    def __init__ (self, fich):
        self.ptrtema = -1
        self.lista_temas = []
        self.aux = fich
        document = Cifrado.Descifrar(fich = fich)
        u=unicode(document,'iso-8859-1')
        cadena=u.encode('utf-16')
        # parse an XML file by name
        self.docxml = minidom.parseString(cadena)
     # Cogemos los temas del fichero
        lista_aux = self.docxml.getElementsByTagName('TEMA')
     # Los almacenamos en una lista
        for tema in lista_aux:
            self.lista_temas.append(tema)

    # Devuelve un tema de la lista
    def GetTema(self):
        # Si aún quedan temas en la lista
        if (self.ptrtema < len (self.lista_temas) - 1):
         # Se devuelve el tema
            self.ptrtema = self.ptrtema+1
            tema = self.lista_temas[self.ptrtema]
            temp = tema.getAttribute('TEXTO')
```

```
lista = Convertir(temp).GetResultado()
lista.append (Tema (tema))
return (lista)
```

B.3. usuario

```
##                  HEVAH
## Proyecto de fin de carrera de Ingenieria Informática
## Universidad de La Laguna
## Tutora: Carina Gonzalez Gonzalez
## Autores: Fco. Javier Medina Santos
##     Daniel J. Díaz Gonzalez
##     Zebenzui Pérez Ramos
## Descripcion: Contiene dos clases: Usuario y Perfiles
##                  * Usuario: se encarga de realizar la evaluación del
##                             usuario una vez finalizado el test.Los datos
##                             se van almacenando a lo largo del test en
##                             una tabla de resultados en funcion del nivel
##                             del concepto y el nivel de la pregunta.
##                             Según el nivel del usuario y del nivel del
##                             concepto se predefinen las notas. La nota
##                             global será la suma de las notas parciales
##                             que se vayan obteniendo en los distintos
##                             niveles de dificultad de los conceptos.
##                  * Perfiles: Se encarga de gestionar el conjunto de
##                              usuarios del sistema.

from types import UnicodeType
from xml.dom import minidom
import encodings
import string
import os
import time
from Cifrado import Cifrado

class Usuario:

    def __init__ (self, username):
     #Inicialización de propiedades
```

```
        self.TablaResultados ={}
#Nota para cada uno de los niveles de dificultad de los conceptos
        self.ValorModulos = {"SIMPLE":[6,4,2,0],"INTERMEDIO":[4,4,3,2],
                        "COMPLEJO":[2,2,4,4],"AVANZADO":[2,2,1,4]}
        self.niveles_user = ["novel", "principiante", "medio", "experto"]
        self.niveles_conc = ["SIMPLE", "INTERMEDIO", "COMPLEJO",
                        "AVANZADO"]
        self.username = username        # Cadena alfanumerica
#Seleccionamos el fichero de perfil del usuario, si existe
        fichero = "%s.xml" % self.username
        if (os.path.exists ('hevah/%s' % fichero)):
            document = Cifrado.Descifrar(fich = 'hevah/%s' %fichero)
            self.docxml = minidom.parseString(document)
            self.raiz = self.docxml.getElementsByTagName ("USUARIO")[0]
            self.nivel = self.niveles_user.index(string.lower
             (self.raiz.getAttribute('NIVEL')))

            preguntas = ["ALTA","BAJA","MEDIA"]
            conceptos = ["SIMPLE","INTERMEDIO","COMPLEJO","AVANZADO"]
    #Inicializamos la tabla de resultados
            for i in conceptos:
                for j in preguntas:
                    self.TablaResultados[i,j] = {}
                    self.TablaResultados[i,j]["ACIERTOS"] = 0.0
                    self.TablaResultados[i,j]["FALLOS"] = 0.0
        self.Historico = {}

    # Devuelve:
    #     * el nivel del usuario
    def GetNivel (self):
        return (self.niveles_user[self.nivel])

    # Actualiza el histórico indicando si la pregunta fue repsondida
    # correctamente o no.
    # Parámetros:
```

```
    #     * id_pregunta: identificador de la pregunta actual
    #     * acertada: si la pregunta fue respondida correctamente
    def ActualizarHistorico (self, id_pregunta, acertada):
        if (acertada):
            self.Historico[id_pregunta] = 'ACERTADA'
        else:
            self.Historico[id_pregunta] = 'FALLADA'

    # Actualiza la tabla de resultados aumentando el número de aciertos
    # o fallos.
    # Parámetros:
    #     * concep: concepto que se esta evaluando
    #     * preg: pregunta que se ha presentado
    #     * acertada: indica si la pregunta fue respondida
    #                 correctamente o no.
    def ActualizarTabla (self, concep, preg, acertada):
        if (acertada):
            self.TablaResultados[concep.GetNivel(),
preg.GetNivel()]["ACIERTOS"] += 1.0
        else:
            self.TablaResultados[concep.GetNivel(),
preg.GetNivel()]["FALLOS"] += 1.0

    # Devuelve:
    #     * el nivel del concepto
    def GetConceptoInicial (self):
        return (str(self.niveles_conc[self.nivel]))

    # Muestra en consola la tabla de resultados
    def VerResultados (self):
        preguntas = ["ALTA","BAJA","MEDIA"]
        conceptos = ["SIMPLE","INTERMEDIO","COMPLEJO","AVANZADO"]
        for i in conceptos:
            print "CONCEPTOS %s " % i
            for j in preguntas:
```

```
                print "\tPREGUNTAS %s " % j
                print "\t\tACIERTOS :> %d" %
                     self.TablaResultados[i,j]["ACIERTOS"]
                print "\t\tFALLOS :> %d" %
                     self.TablaResultados[i,j]["FALLOS"]

    # Se encarga de realizar la evaluación de los resultados obtenidos por
    # el alumno durante la ejecución del test.Para calcular la nota primero
    # calcula el porcentaje de aciertos obtenidos para el par: dificultad
    # concepto - dificultad pregunta. Luego se multiplica ese porcentaje
    # por la nota predefinida para cada para. Esa nota predefinida esta en
    # funcion del nivel del usuario.
    def Evaluar (self):
        porcentaje = {"ALTA":0.5,"BAJA":0.2,"MEDIA":0.3}
        preguntas = ["ALTA","BAJA","MEDIA"]
        conceptos = ["SIMPLE","INTERMEDIO","COMPLEJO","AVANZADO"]
        nota = 0.0
        for i in conceptos:
            for j in preguntas:
     #se suman los aciertos y los fallos botenidos en los pares
#dificultad concepto - dificultad pregunta
                total = (self.TablaResultados[i,j]["ACIERTOS"] +
          self.TablaResultados[i,j]["FALLOS"])
#calculamos el porcentaje de la nota para ese par
                if (total != 0):
                    parcial = self.TablaResultados[i,j]["ACIERTOS"] / total
                    parcial *= porcentaje[j]
                else:
                    parcial = 0.0
#calculamos la nota
                nota += parcial * self.ValorModulos[i][self.nivel]
        if (nota > 10):
            nota = 10
        self.nota = nota
```

```
    # Devuelve:
    #     * la nota del alumno
    def GetNota(self):
        return self.nota

    def CrearPerfil (self, fichero_test):
        fichero = "hevah/%s.xml" % self.username
        self.raiz = self.docxml.getElementsByTagName ("USUARIO")[0]
        nodo = self.docxml.createElement ('TEST')

#Almacenamos los datos generales del test
        index_ = string.index (fichero_test, '/')
        nodo.setAttribute ("FICHERO", fichero_test[index_+1:])
        nodo.setAttribute ("NOTA", "%.3f"%(self.nota))
        fecha_hora = time.localtime(time.time())
        nodo.setAttribute ("FECHA", time.strftime("%d/%m/%Y",fecha_hora) )
        nodo.setAttribute ("HORA", time.strftime("%H/%M/",fecha_hora) )
#Almacenamos los aciertos y fallos para el par: dificultad-concepto
#dificultad pregunta
        preguntas = ["ALTA","BAJA","MEDIA"]
        conceptos = ["SIMPLE","INTERMEDIO","COMPLEJO","AVANZADO"]
        for i in conceptos:
            nodo_conc = self.docxml.createElement(i)
            for j in preguntas:
                nodo_preg = self.docxml.createElement(j)
                nodo_preg.setAttribute ("ACIERTOS", repr(
           self.TablaResultados[i,j]["ACIERTOS"]))
                nodo_preg.setAttribute ("FALLOS", repr(
           self.TablaResultados[i,j]["FALLOS"]))
                nodo_conc.appendChild (nodo_preg)
            nodo.appendChild (nodo_conc)
#Almacenamos la pregunta y si fue acertada o no en el histórico
        historico = self.docxml.createElement ('HISTORICO')
        for preg in self.Historico.keys():
            aux = self.docxml.createElement('PREGUNTA')
```

```
            aux.setAttribute ('ID',preg)
            aux.setAttribute ('VALOR', self.Historico[preg])
            historico.appendChild (aux)
        #Almacenamos la información
        nodo.appendChild (historico)
        self.raiz.appendChild (nodo)
        f = open (fichero,"w")
cad_cifrada = Cifrado.Cifrar(cadena = i
                             self.docxml.toxml().encode ("latin1"))
        f.write (cad_cifrada)
        f.close()


class Perfiles:
    # Parámetros:
    #     * fichero_index: el fichero indice de los perfiles de usuario
    #     * nombre_test: el nombre del test
    usuario = {}
    def __init__ (self, fichero_index, nombre_test = "__"):
        self.index = "/%s.xml" % (fichero_index)
        if (os.path.exists (self.index)):
            document = Cifrado.Descifrar(fich = self.index)
            u=unicode(document,'iso-8859-1')
            cadena=u.encode('utf-16')
        else:
            if (nombre_test == "__"):
                nombre_test = fichero_index
            cadena = '<INDEX TEST="%s"/>' % (nombre_test)

#Se crea la lista de usuarios
        self.docxml = minidom.parseString(cadena)
        self.raiz = self.docxml.getElementsByTagName ("INDEX")[0]
        lista = self.raiz.getElementsByTagName ('USUARIOS')
        for user in lista:
            self.usuario[user.getAttribute ('ID')] = user
```

```
    # Parámetros:
    #     * id: identificador del usuario
    # Devuelve:
    #     * el usuario solicitado
    def GetUsuario (self, id):
        if (self.usuario.has_key (id)):
            return (self.usuario[id])
        return (None)
```

Apéndice C

Código Fuente de HEVAH

C.1. basic

```
## Autores: Fco. Javier Medina Santos
##        Daniel J. Díaz Gonzalez
##        Zebenzui Pérez Ramos
## Descripcion: Conjunto de funciones de interes para el proyecto

import Blender
from Blender import Ipo
import GameLogic
import Rasterizer
import sys
from _winreg import *


# Script para leer la velocidad del procesador desde el registro de Windows
def cpuSpeed():
    aReg = ConnectRegistry(None,HKEY_LOCAL_MACHINE)

    aKey = OpenKey(aReg, r"HARDWARE\DESCRIPTION\System\CentralProcessor\0")
    for i in range(1024):
        try:
            n,v,t = EnumValue(aKey,i)
            if (n == '~MHz'):
```

```
                CloseKey(aKey)
                CloseKey(aReg)
                return v
        except EnvironmentError:
            print "ERROR: No se encontró la clave"
            break
    CloseKey(aKey)
    CloseKey(aReg)
    return None

# Script que modifica el valor de un conjunto de claves
# Parámetros:
#    * num_clave: Conjunto con los números de clave a modificar
#    * valor: Conjunto de los nuevos valores de las claves
def working_list(num_clave, valor):
    #función que actualiza un grupo de claves
    work_ipo=Ipo.get('KeyIpo')
    allcurves=work_ipo.curves
    index=0
    for num in num_clave:
        curve=allcurves[num]
        n_points=curve.points

        new_point= Blender.Ipo.BezierPoint()
        new_point=n_points[0]
        new_point.pt[1]=valor[index]
        del n_points[0]
        n_points.insert(0,new_point)
        curve.points=n_points
        index = index+1

    work_ipo.update()

# Script que modifica el valor de una clave
# Parámetros:
```

```
#    * num_clave: Número de clave a modificar
#    * valor: Nuevo valor de la clave
def working(num_clave, valor):
    "Función que actualiza los valores de la clave"
    work_ipo=Ipo.Get('KeyIpo')
    allcurves=work_ipo.curves
    curve=allcurves[num_clave]
    n_points=curve.points

    new_point= Blender.Ipo.BezierPoint()
    new_point=n_points[0]
    new_point.pt[1]=valor
    del n_points[0]
    n_points.insert(0,new_point)
    curve.points=n_points

    Ipo.Recalc(work_ipo)
```

C.2. control

```
##    Autores:    F. Javier Medina Santos
##                Daniel J. Diaz Gonzalez
##                Zebenzui Perez Ramos
##    Descripcion: Este script se encarga de gestionar las preguntas
##        y respuestas. En general, tambien se ocupa de coordinar el
##        resto de scripts.

import GameLogic
import Rasterizer
import basic
import parpadeo
from procesarfich import *
from procxml import MiXML
import time
import os

import usuario                    # Perfiles de usuario
import clipsxml                    # Comunicacion con el clips
from Evaluar import Evaluar        # Evaluacion de las respuestas


Rasterizer.showMouse(1)            # para que se vea el raton

cont=GameLogic.getCurrentController()

own=cont.getOwner()
# Numero maximo de intentos para buscar conceptos
max_intentos = 3

retraso_gesto = 200.0 # proporcional a lo que tarde en gesticular el habla

#Iniciliaciones
if (own.sem_inicio == 1):
```

```
own.sem_inicio = 0    #Indicamos que ya se inicializo

# obtener T
try:
    f_temp = open ("T.log","r")
except:
    f_temp = None

if (f_temp != None):
    # leer de fichero T
    tmp = f_temp.read()
    tmp2 = tmp.split ('|')
    GameLogic.T = float(tmp2[0])
    GameLogic.muestras = int(tmp2[1])
    f_temp.close()
else:
    # calibrar el movimiento de la boca en relacion con la
    # frecuencia de la CPU
    GameLogic.T = retraso_gesto / basic.cpuSpeed()
    GameLogic.muestras = 1

# inicializa el cuadro de respuestas
GameLogic.letra = 'VACIO'
# indica que de momento no se puede responder
GameLogic.responder = -1

own.finalizado = 0

#Arrancamos el proceso del clips

GameLogic.clips = clipsxml.Coprocess("CLIPSDOS.exe")

######### Cargamos el perfil del usuario ###########
f_temp = open ("login.pro","r")        #Este fichero lo crea el login.pyw
```

```
    id_user = f_temp.read()  # id | nombre | apellido1 | apellido2 | nivel
    f_temp.close()
    lista_param = id_user.split ('|')
    GameLogic.id_user = lista_param[0]
    GameLogic.user = usuario.Usuario (lista_param)

    #Inicialimos probabilidades y la clase de evaluacion
    prob_ini = [0.4,0.3,0.1,0.01,0.5,0.4,0.2,0.1,0.6,0.5,0.3,
                                          0.2,0.7,0.6,0.5,0.3]
    GameLogic.calculos = Evaluar(prob_ini, GameLogic.clips, "reglas.clips")

    basic.working_list([2,3,4,5,6,7,8,9],[0,0,0,0,0,0,1,1])    #Gesto neutral
    p = Preprocesado()
    p.leerfichero('mbesp.conf') #Cargamos el fichero de idioma
    own.movboca = p.cadena
    own.grupos = p.grupos
    GameLogic.enunciado='OK'
    GameLogic.arbol = MiXML('result.xml')

    # Conseguimos el primer tema y el primer concepto
    GameLogic.tema = GameLogic.arbol.GetTema()[2]
    GameLogic.concepto = GameLogic.tema.GetConcepto (
                        GameLogic.user.GetConceptoInicial())[2]
    # Le indicamos al evaluador que iniciamos un nuevo concepto
    GameLogic.calculos.NuevoConcepto (GameLogic.user,
                              GameLogic.concepto.GetNivel())
    GameLogic.nivel_preg = 'MEDIA'
    # Saludo inicial
    own.frase = "Bienvenidos al Proyecto EVA, Agente Evaluador Adaptativo.
          Espero ser de su agrado."

# Formamos la pregunta y testeamos la respuesta
if (own.next_preg == 1):
    own.next_preg = 0
```

```
if (GameLogic.concepto != None):
    GameLogic.pregunta = GameLogic.concepto.GetPregunta (
                                        GameLogic.nivel_preg)
    intento = 0 #Buscamos hasta en conceptos una pregunta de este
                #nivel; si no encontramos, acabamos el test
else:
    intento = max_intentos
# Loczalizacion de la pregunta
while ((GameLogic.pregunta[2] == None) and (intento < max_intentos)):
    aux = GameLogic.concepto.GetNivel()
      GameLogic.concepto = GameLogic.tema.GetConcepto (aux)[2]
    if (GameLogic.concepto == None):
        intento = max_intentos
        break
    GameLogic.calculos.NuevoConcepto(GameLogic.user,
                         GameLogic.concepto.GetNivel())
    if (GameLogic.concepto != None):
        GameLogic.pregunta = GameLogic.concepto.GetPregunta(
                                        GameLogic.nivel_preg)
    else :
        intento = max_intentos
        break
    intento += 1
if (intento == max_intentos):
    # FINALIZADO POR FALTA DE PREGUNTAS
    # Al no haber mas preguntas, no es necesario responder
    GameLogic.responder = -1

    GameLogic.user.Evaluar()
    GameLogic.user.CrearPerfil()
    temp = "%.3f" % GameLogic.user.GetNota()
    nota = string.replace (temp, '.', ' punto ')
    own.frase = "NO HAY MAS PREGUNTAS. TIENES UNA NOTA DE %s" % nota
    # indica que se ha llegado al final del test
    own.finalizado = 1
```

```
    if (own.finalizado != 1):
        #Accedemos al enunciado de la pregunta en "formato TTS"
        #y lo convertimos
        tmp = str(GameLogic.pregunta[0])

        #### FALTARIA MOSTRAR LAS RESPUESTAS ####

        #Mostramos alternativas
        frase = tmp#+'       ..'
        if (GameLogic.pregunta[2] != None):
            for i in "ABCDEFGHIJKLMNOPQRSTUVWXYZ":
                tmp=str(GameLogic.pregunta[2].GetRespuesta(i))
                if (tmp != 'RESPUESTA INEXISTENTE'):
                    frase = "%s Respuesta %s: %s." % (frase,i,tmp)
                else:
                    # 'limite' evita respuestas incoherentes
                    own.limite = i
                    break
            own.frase = str(frase)
            print own.frase

if (own.contestada == 1):
    own.next_preg = 1
    own.contestada = 0
    #Evaluamos la respuesta
    siguiente = GameLogic.calculos.NuevaPreg (GameLogic.clips,
                                own.elegida,GameLogic.pregunta[2])
    lista = siguiente.split('_')

    #Actualizamos Tabla de resultados
    GameLogic.user.ActualizarTabla (GameLogic.concepto,
          GameLogic.pregunta[2], GameLogic.calculos.GetTestUltima())

    #Actuamos segun lo que nos diga el clips
```

```
    tipo = lista[0]
    nivel = str(string.upper(lista[1].strip()))
    if (tipo == "concepto"):
        GameLogic.concepto = GameLogic.tema.GetConcepto (nivel)[2]
        if GameLogic.concepto != None :
            GameLogic.calculos.NuevoConcepto(GameLogic.user,
                                GameLogic.concepto.GetNivel())
    else:
        GameLogic.nivel_preg = nivel
    own.elegida = 'VACIO'

parpadeo.parpadear()
```

C.3. gestos

```
##    Autores:    F. Javier Medina Santos
##                Daniel J. Diaz Gonzalez
##                Zebenzui Perez Ramos
##    Descripcion: Este script se encarga de efectuar gestos
##        aleatorios cada cierto tiempo. Esta asociado con la cabeza

import GameLogic
import basic

cont = GameLogic.getCurrentController()
owner = cont.getOwner()
tiempo_gesto = 3 # segundos que tarda en ejecutarse un gesto
frames_gesto = 100 # frames que ocupa un gesto
num_gestos = 10 # numero total de gestos
num_gestos_ipo = 7 # numero de gesto con IPO en GameEngine


if (owner.parpadeo - owner.anterior > tiempo_gesto * 2):
# Generar el numero de gesto aleatorio (k)
    owner.anterior = owner.parpadeo
    owner.k = GameLogic.getRandomFloat() * 1000
    owner.k = owner.k % num_gestos
    owner.k = owner.k * frames_gesto
else:
    if (owner.parpadeo - owner.anterior > tiempo_gesto):
    # Si se ha terminado el gesto, se espera el siguiente
        owner.k = 0
    else:
        if owner.k < num_gestos_ipo * frames_gesto:
        # Si es un gesto mediante IPO se recalcula k
            x = frames_gesto / tiempo_gesto * (owner.parpadeo-owner.anterior)
            owner.k = owner.k - (owner.k % 100) + x
        else:
```

```
        # sino se actualiza el RVK conveniente
            k2 = owner.k / frames_gesto

            if k2 == num_gestos_ipo:
                basic.working_list([10],[0.60])
            if k2 == num_gestos_ipo + 1:
                basic.working_list([10],[0])
            if k2 == num_gestos_ipo + 2:
                basic.working_list([10],[-0.40])
```

C.4. hablar

```
##    Autores:    F. Javier Medina Santos
##            Daniel J. Diaz Gonzalez
##            Zebenzui Perez Ramos
##    Descripcion: Este script se encarga de lanzar el habla
##         mediante la SAPI. Esta asociado a la cabeza

import GameLogic
import win32com.client
import pythoncom
import threading

#<tts>
defaultNamedOptArg=pythoncom.Missing
defaultNamedNotOptArg=pythoncom.Missing
defaultUnnamedArg=pythoncom.Missing
directss = win32com.client.Dispatch(
           "{EEE78591-FE22-11D0-8BEF-0060081841DE}")
#</tts>


# Establece la velocidad del habla
def setSpeed(speed_percent):
    global directss
    #set voice speed
    rango = (directss.MaxSpeed - directss.MinSpeed)
    directss.Speed = directss.MinSpeed + (rango * speed_percent) / 100

# Establece en Pitch (voz aguda, voz grave,...)
def setPitch(pitch_percent):
    global directss
    #set voice pitch
    rango = (directss.MaxPitch - directss.MinPitch)
    directss.Pitch =directss.MinPitch + (rango * pitch_percent) / 100
```

```
# Establece el volumen
def setVolume(volume_percent):
    global directss
    #set voice volume left
     rango = (directss.MaxVolumeLeft - directss.MinVolumeLeft)
    directss.VolumeLeft = directss.MinVolumeLeft + (rango * volume_percent)
                          / 100


def hablar(directss, pythoncom,owner):

    speaking = 0
    spoken = 0
    saythis = owner.frase
    owner.frase = 'EXIT'
    while saythis != 'exit':
        if speaking == 0:
            if spoken == 0:
                directss.Speak(saythis)
                spoken = 1
            else:
                saythis = 'exit'
                spoken = 0
        pythoncom.PumpWaitingMessages()
        speaking = directss.Speaking
    owner.semaforo = 1
    return


cont = GameLogic.getCurrentController()
owner = cont.getOwner()
if owner.frase != 'EXIT':
    # Para que no se pueda responder mientras se habla
    GameLogic.responder = GameLogic.responder - 1
```

```
    actuator=cont.getActuator("acercar_camara")
    GameLogic.addActiveActuator(actuator,1)

if (owner.semaforo == 1):
    owner.numletra=0
    setPitch(40)
    setVolume(100)
    setSpeed(100)
    # semaforo para evitar dispara varias veces el TTS
    owner.semaforo = 0
    owner.gestual = owner.frase
    # se lanza en un hilo para evitar bloqueamiento
    hilo1 = threading.Thread(target = hablar, name = 'hilo1',
                             args=(directss,pythoncom,owner))
    hilo1.setDaemon(1)
    hilo1.start()
```

C.5. letra

```
##    Autores:    F. Javier Medina Santos
##            Daniel J. Diaz Gonzalez
##            Zebenzui Perez Ramos
##    Descripcion: Este script se encarga de actualizar la letra del
##        cuadro. Esta asociado con el cuadro

import GameLogic

cont = GameLogic.getCurrentController()
owner = cont.getOwner()


if (GameLogic.letra == 'VACIO'):
# si estamos a la espera d euna respuesta
    owner.Text = '?'
else:
    # Se lanza el sonido
    actuator=cont.getActuator("sonido")
    GameLogic.addActiveActuator(actuator,1)
    # Se actualiza la letra en el cuadro
    owner.Text = GameLogic.letra
    # Se anima el cuadro
    actuator=cont.getActuator("animar")
    GameLogic.addActiveActuator(actuator,1)
```

C.6. matar

```
##    Autores:    F. Javier Medina Santos
##            Daniel J. Diaz Gonzalez
##            Zebenzui Perez Ramos
##    Descripcion: Este script se encarga finalizar el CLIPS y
##        salir del modulo de juegos. Esta asociado a la tecla 'salir'

import GameLogic

GameLogic.clips.kill()

cont = GameLogic.getCurrentController()
owner = cont.getOwner()

actuator=cont.getActuator("salir")
GameLogic.addActiveActuator(actuator,1)
```

C.7. moverboca

```
##    Autores:    F. Javier Medina Santos
##                Daniel J. Diaz Gonzalez
##                Zebenzui Perez Ramos
##    Descripcion: Este script se encarga de hacer que la boca se mueva.
##        Esta asociado a la cabeza

import Blender
from Blender import Ipo
import GameLogic
import Rasterizer
import sys
import basic
import time
from procesarfich import *

# Se encarga de mover la boca de acuerdo con la informacion de los
# grupos vocales (contenida en 'cad')
def moverboca(frase, grupos, cad):
    p = Preprocesado()
    indice = p.simbolos(frase, grupos)
    curvas=[2,3,4,5,6,7,8,9]
    for j in indice:
        i = string.atoi(j)
        valores = {}
        for h in range (8):
            valores[h] = string.atoi (cad[i*24 + (h*3)] + cad[(i*24)+ (h*3+1)]
                        + cad[(i*24)+(h*3+2)] )
            valores[h] = valores[h] / 100 - 5
        basic.working_list(curvas, valores)
        time.sleep(GameLogic.pausa)

cont = GameLogic.getCurrentController()
owner = cont.getOwner()
```

```
if (owner.numletra != -1):
    if (owner.numletra < len (owner.gestual)):
        if (owner.numletra == 0):
            time.sleep(0.2)
        moverboca(owner.gestual[owner.numletra],owner.grupos, owner.movboca)
        owner.numletra = owner.numletra+2
    else:
        # Hemos terminado la frase
        owner.numletra = -1

        actuator=cont.getActuator("acercar_camara")
        GameLogic.addActiveActuator(actuator,1)
        basic.working_list([2,3,4,5,6,7,8,9],[0,0,0,0,0,0,1.2,1.2])

        if GameLogic.responder != -2:
            GameLogic.responder = 1
            # Actualizar el cuadro
            GameLogic.letra = 'VACIO'
            actuator=cont.getActuator("vaciar")
            GameLogic.addActiveActuator(actuator,1)
        else:
            # Entrara aqui antes y despues del test
            # o sea, cuando no debe contestar
            GameLogic.responder = 0
            actuator=cont.getActuator("empezar")
            GameLogic.addActiveActuator(actuator,1)
```

C.8. parpadeo

```
## Autores: Fco. Javier Medina Santos
##        Daniel J. Díaz Gonzalez
##        Zebenzui Pérez Ramos
## Descripcion: Este script se encarga de realizar el parpadeo del
##              personaje

import basic
import GameLogic

# De acuerdo con un timer cambia los valores de las claves para conseguir
# el efecto de parpadeo
def parpadear():
    cont = GameLogic.getCurrentController()
    owner = cont.getOwner()

    if (owner.parpadeo % 5 < 1):
        basic.working_list([0,1],[0.85,0.85])
        actuator=cont.getActuator("cierra")
        GameLogic.addActiveActuator(actuator,1)
    else:
        if (owner.parpadeo % 5 < 1.5):
            basic.working_list([0,1],[0,0])
            actuator=cont.getActuator("abre")
            GameLogic.addActiveActuator(actuator,1)
```

C.9. teclado

```
##    Autores:    F. Javier Medina Santos
##                Daniel J. Diaz Gonzalez
##                Zebenzui Perez Ramos
##    Descripcion: Este script se encarga de recoger las respuestas
##        desde teclado. Esta asociado a la cabeza

import GameLogic
import string

cont = GameLogic.getCurrentController()
own = cont.getOwner()

sensor = cont.getSensor("teclado")
lista = sensor.getPressedKeys()


if (lista[0][1] == 1) and (GameLogic.responder == 1):
    tmp = string.upper(chr(lista[0][0]))
    if (tmp < own.limite) and (tmp >= 'A'):
        own.elegida = tmp
        GameLogic.letra = own.elegida
```

Apéndice D

Código Fuente de PORTAD

D.1. ActivarDatos

```
## Autores: Fco. Javier Medina Santos
##         Daniel J. Díaz Gonzalez
##         Zebenzui Pérez Ramos
## Descripcion: Se encarga de verificar los datos introducidos a la hora
##              de insertar un nuevo usuario en el sistema.

from Cheetah.Template import Template
from xml.dom import minidom
from index import index
import string
from Cifrado import Cifrado

class ActivarDatos (index):
    def writeContent (self):
        req = self.request()
        resp = self.response()
        #El usuario está validado
        if (self.request().hasCookie('grupo')):
            #Es un profesor o un administrador
            if (self.request().cookie ('grupo') != "ALUMNO"):
                #recogemos los datos de la plantilla
                username = string.upper(req.field('username'))
```

```
                nombre = req.field('nombre')
                apellido1 = req.field('apellido1')
                apellido2 = req.field('apellido2')
                dni = req.field('dni')
                email = req.field('email')
                grupo = string.upper(req.field('grupo'))
                #Vemos si el fichero de perfil ya existe
                try:
                    cad_descifrada = Cifrado.Descifrar(
                   fich = 'hevah//%s.xml' % username)
                    usuario = minidom.parseString (cad_descifrada)
                except:
                    #Si un profesor crea un alumno
                    if (req.cookie ('grupo') == "PROFESORES"):
                        usuario = minidom.parseString('''
       <USUARIO NIVEL="novel">
                                <DATOS /></USUARIO>''')
                    else:
                        #Si un admin crea un profesor
                        if (grupo == "PROFESORES"):
                            usuario = minidom.parseString('''
         <USUARIO NIVEL="novel">
                                        <DATOS/><ALUMNO></ALUMNO><TEST></TEST>
</USUARIO>''')
                        #Si un admin crea otro admin
                        else:
                            usuario = minidom.parseString('''
 <USUARIO NIVEL="novel">
                                        <DATOS /><PROFESOR></PROFESOR>
<ADMIN></ADMIN>
</USUARIO>''')

                    #Creamos la entrada del usuario en el indice
                    cad_descifrada = Cifrado.Descifrar(fich =
                                    'hevah//indice.xml')
```

```
                    indice = minidom.parseString (cad_descifrada)
                    raiz = indice.getElementsByTagName ('PERFILES')[0]
                    nodo = indice.createElement ('%s'
                              % string.upper(username))
                    nodo.setAttribute ('USERNAME', string.upper(username))
                    nodo.setAttribute ('PASSWD', req.field('passwd'))
                    nodo.setAttribute ('GRUPO', grupo)
                    #Si insertamos un alumno ponemos su profesor asociado
                    if (req.cookie('grupo') == 'PROFESORES'):
                        nodo.setAttribute('PROFESOR',
                   req.cookie('username'))
                    raiz.appendChild (nodo)
                    f = open ('hevah//indice.xml','w')
                    cad_cifrada = Cifrado.Cifrar(cadena =
               self.codificar(indice.toxml()))
                    f.write (cad_cifrada)
                    f.close()
               #Metemos los datos del usuario en su fichero de perfil
               datos = usuario.getElementsByTagName ('DATOS')[0]
               datos.setAttribute ('NOMBRE',self.codificar(nombre))
               datos.setAttribute ('APELLIDO1', self.codificar(apellido1))
               datos.setAttribute ('APELLIDO2', self.codificar(apellido2))
               datos.setAttribute ('DNI', self.codificar(dni))
               datos.setAttribute ('EMAIL', self.codificar(email))
               datos.setAttribute ('USERNAME', self.codificar(username))
               f = open ('hevah//%s.xml' % username,'w')
               cad_cifrada = Cifrado.Cifrar(cadena = usuario.toxml())
               f.write (cad_cifrada)
               f.close()

               #Si un profesor crea un nuevo alumno, éste se me mete en
#su perfil
               if (req.cookie('grupo') == "PROFESORES"):
                    cad_descifrada = Cifrado.Descifrar(fich =
  'hevah//%s.xml' % req.cookie('username'))
```

```
                profesor = minidom.parseString (cad_descifrada)
                alumnos = profesor.getElementsByTagName ('ALUMNO')[0]
                nuevo_alumno = profesor.createElement ('%s'
             % string.upper(username))
                nuevo_alumno.setAttribute ('USERNAME',
                          string.upper(username))
                nuevo_alumno.setAttribute ('PASSWD', req.field('passwd'))
                nuevo_alumno.setAttribute ('GRUPO', grupo)
                alumnos.appendChild (nuevo_alumno)
                f = open ('hevah//%s.xml' % req.cookie('username'),'w')
                cad_cifrada = Cifrado.Cifrar(cadena =
            self.codificar(profesor.toxml()))
                f.write (cad_cifrada)
                f.close()
            else:
                cad_descifrada = Cifrado.Descifrar(fich =
        'hevah//%s.xml' % req.cookie('username'))
                admin = minidom.parseString (cad_descifrada)
                #Si un admin crea un nuevo profesor almacenamos
#éste en su perfil
                if (grupo == "PROFESORES"):
                    profesores = admin.getElementsByTagName(
         'PROFESORES')[0]
                    nuevo_profesor = admin.createElement ('%s'
             % string.upper(username))
                    nuevo_profesor.setAttribute ('USERNAME',
                         string.upper(username))
                    nuevo_profesor.setAttribute ('PASSWD',
                         req.field('passwd'))
                    nuevo_profesor.setAttribute ('GRUPO', grupo)
                    profesores.appendChild (nuevo_profesor)
                #Si un admin crea un nuevo admin almacenamos este en
#si perfil
                else:
                    admins = admin.getElementsByTagName ('ADMIN')[0]
```

```
                    nuevo_admin = admin.createElement ('%s'
          % string.upper(username))
                    nuevo_admin.setAttribute ('USERNAME',
                      string.upper(username))
                    nuevo_admin.setAttribute ('PASSWD',
                      req.field('passwd'))
                    nuevo_admin.setAttribute ('GRUPO', grupo)
                    admins.appendChild (nuevo_admin)
                f = open ('hevah//%s.xml' % req.cookie('username'),'w')
                cad_cifrada = Cifrado.Cifrar(cadena =
            self.codificar(admin.toxml()))
                f.write (cad_cifrada)
                f.close()
            resp.sendRedirect ('MostrarDatosUsuario?usuario=%s'
                % username)
        #Si no es profesor o administrador, se le expulsa por acceso
#no autorizado
        else:
            self.LimpiarCookies()
            self.response().sendRedirect ('index')
    # Al no estar validado, se le manda a la página de Bienvenida
    else:
        self.LimpiarCookies()
        self.response().sendRedirect ('index')
```

D.2. BorrarConcepto

```
##                  HEVAH
## Proyecto de fin de carrera de Ingenieria Informática
## Universidad de La Laguna
## Tutora: Carina Gonzalez Gonzalez
## Autores: Fco. Javier Medina Santos
##     Daniel J. Díaz Gonzalez
##     Zebenzui Pérez Ramos
## Descripcion: Se encarga de borrar un concepto dado del fichero de test
##              sobre el que se este trabajando en ese momento y de tema
##              que haya sido seleccionado


from xml.dom import minidom
from Cheetah.Template import Template
from TestDisponibles import TestDisponibles
from Cifrado import Cifrado

class BorrarConcepto (TestDisponibles):
    def writeContent (self):
        req = self.request()
        resp = self.response()
        file = req.cookie ('fichero')
        tema = req.cookie ('tema')
        concepto = req.cookie('concepto')
        cad_descifrada = Cifrado.Descifrar(fich = file)
        dom = minidom.parseString (cad_descifrada)
     # Buscamos en el fichero el tema con el que estamos trabajando
        lista_temas = dom.getElementsByTagName ('TEMA')
        for _tema in lista_temas:
            if (_tema.getAttribute('ID') == tema):
                nodo_tema = _tema
                break
        # Buscamos en el fichero el concepto que queremos eliminar
```

```
lista_conceptos = [e for e in nodo_tema.childNodes
                if e.nodeType == e.ELEMENT_NODE]
   for _concepto in lista_conceptos:
       if (_concepto.getAttribute('ID') == concepto):
           nodo_concepto = _concepto
           break
#eliminamos el concepto
   nodo_tema.removeChild (nodo_concepto)
   f = open (file,'w')
cad_cifrada = Cifrado.Cifrar(cadena = str(dom.toxml()))
   f.write (cad_cifrada)
   f.close()
   resp.sendRedirect ('VerTema')
```

D.3. BorrarPregunta

```
##                  HEVAH
## Proyecto de fin de carrera de Ingenieria Informática
## Universidad de La Laguna
## Tutora: Carina Gonzalez Gonzalez
## Autores: Fco. Javier Medina Santos
##     Daniel J. Díaz Gonzalez
##     Zebenzui Pérez Ramos
## Descripcion: Se encarga de borrar una pregunta. Dicha pregunta pertenece
##              a un concepo, éste a su vez a un tema y éste a su vez a un
##              test que previamente habremos seleccionado

from xml.dom import minidom
from Cheetah.Template import Template
from TestDisponibles import TestDisponibles
from Cifrado import Cifrado

class BorrarPregunta (TestDisponibles):
    def writeContent (self):
        req = self.request()
        resp = self.response()
        file = req.cookie ('fichero')
        tema = req.cookie ('tema')
        concepto = req.cookie('concepto')
        pregunta = req.cookie('pregunta')
        cad_descifrada = Cifrado.Descifrar(fich = file)
        dom = minidom.parseString (cad_descifrada)
        test = dom.getElementsByTagName ('TEST')[0]
     # Buscamos en el fichero el tema con el que estamos trabajando
        lista_temas = dom.getElementsByTagName ('TEMA')
        for _tema in lista_temas:
            if (_tema.getAttribute('ID') == tema):
                nodo_tema = _tema
                break
```

```
     # Buscamos en el tema el concepto con el que estamos trabajando
        lista_conceptos = [e for e in nodo_tema.childNodes
                     if e.nodeType == e.ELEMENT_NODE]
        for _concepto in lista_conceptos:
            if (_concepto.getAttribute('ID') == concepto):
                nodo_concepto = _concepto
                break
     # Buscamos en el concepto la pregunta que queremos borrar
        lista_preguntas = [e for e in nodo_concepto.childNodes
                     if e.nodeType == e.ELEMENT_NODE]
        for _pregunta in lista_preguntas:
            if (_pregunta.getAttribute('IDPREGUNTA') == pregunta):
                nodo_pregunta = _pregunta
                break
#borramos la pregunta seleccionada
        nodo_concepto.removeChild (nodo_pregunta)
        f = open (file,'w')
cad_cifrada = Cifrado.Cifrar(cadena = str(dom.toxml()))
        f.write (cad_cifrada)
        f.close()
        resp.sendRedirect ('VerConcepto')
```

D.4. BorrarRespuesta

```
##                  HEVAH
## Proyecto de fin de carrera de Ingenieria Informática
## Universidad de La Laguna
## Tutora: Carina Gonzalez Gonzalez
## Autores: Fco. Javier Medina Santos
##     Daniel J. Díaz Gonzalez
##     Zebenzui Pérez Ramos
## Descripcion: Se encarga de borrar una respuesta. Previamente se ha
##              seleccionado la pregunta, el concepto, el tema y el test
##              al que pertenece dicha pregunta.

from xml.dom import minidom
from Cheetah.Template import Template
from TestDisponibles import TestDisponibles
from Cifrado import Cifrado

class BorrarRespuesta(TestDisponibles):
    def writeContent (self):
        req = self.request()
        resp = self.response()
        file = req.cookie ('fichero')
        tema = req.cookie ('tema')
        concepto = req.cookie('concepto')
        pregunta = req.cookie('pregunta')
     respuesta = req.cookie('respuesta')
        cad_descifrada = Cifrado.Descifrar(fich = file)
        dom = minidom.parseString (cad_descifrada)
        test = dom.getElementsByTagName ('TEST')[0]
     # Buscamos en el fichero el tema
        lista_temas = dom.getElementsByTagName ('TEMA')
        for _tema in lista_temas:
            if (_tema.getAttribute('ID') == tema):
                nodo_tema = _tema
```

```
                break
    # Buscamos en el tema el concepto
       lista_conceptos = [e for e in nodo_tema.childNodes
                    if e.nodeType == e.ELEMENT_NODE]
       for _concepto in lista_conceptos:
           if (_concepto.getAttribute('ID') == concepto):
               nodo_concepto = _concepto
               break
    # Buscamos en el concepto la pregunta
       lista_preguntas = [e for e in nodo_concepto.childNodes
                    if e.nodeType == e.ELEMENT_NODE]
       for _pregunta in lista_preguntas:
           if (_pregunta.getAttribute('IDPREGUNTA') == pregunta):
               nodo_pregunta = _pregunta
               break
    #buscamos en la pregunta la respuesta que queremos eliminar
       lista_respuestas= [e for e in nodo_pregunta.childNodes
                    if e.nodeType == e.ELEMENT_NODE]
       for _respuesta in lista_respuestas:
           if (_respuesta.getAttribute('IDRESPUESTA') == respuesta):
               nodo_respuesta = _respuesta
               break
#eliminamos la respuesta
       nodo_pregunta.removeChild (nodo_respuesta)
       f = open (file,'w')
cad_cifrada = Cifrado.Cifrar(cadena = str(dom.toxml()))
       f.write (cad_cifrada)
       f.close()
       resp.sendRedirect ('VerPregunta')
```

D.5. BorrarTema

```
##                  HEVAH
## Proyecto de fin de carrera de Ingenieria Informática
## Universidad de La Laguna
## Tutora: Carina Gonzalez Gonzalez
## Autores: Fco. Javier Medina Santos
##     Daniel J. Díaz Gonzalez
##     Zebenzui Pérez Ramos
## Descripcion: Se encarga de borrar un tema de un test dado.

from xml.dom import minidom
from Cheetah.Template import Template
from TestDisponibles import TestDisponibles
from Cifrado import Cifrado

class BorrarTema (TestDisponibles):
    def writeContent (self):
        req = self.request()
        resp = self.response()
        file = req.cookie ('fichero')
        tema = req.cookie ('tema')
        cad_descifrada = Cifrado.Descifrar(fich = file)
        dom = minidom.parseString (cad_descifrada)
        test = dom.getElementsByTagName ('TEST')[0]
# Buscamos en el fichero el tema que queremos borrar
        lista_temas = dom.getElementsByTagName ('TEMA')
        for _tema in lista_temas:
            if (_tema.getAttribute('ID') == tema):
                nodo_tema = _tema
                break
#borramos el tema seleccionado
        test.removeChild (nodo_tema)
        f = open (file,'w')
cad_cifrada = Cifrado.Cifrar(cadena = str(dom.toxml()))
```

```
        f.write (cad_cifrada)
        f.close()
        resp.sendRedirect ('TestDisponibles')
```

D.6. BorrarTestOK

```
##                  HEVAH
## Proyecto de fin de carrera de Ingenieria Informática
## Universidad de La Laguna
## Tutora: Carina Gonzalez Gonzalez
## Autores: Fco. Javier Medina Santos
##     Daniel J. Díaz Gonzalez
##     Zebenzui Pérez Ramos
## Descripcion: Se encarga de borrar el fichero del test

from TestDisponibles import TestDisponibles
import os

class BorrarTestOK (TestDisponibles):
    def writeContent (self):
        req = self.request()
        resp = self.response()
        file = req.cookie ('fichero')
        os.remove(file)
        resp.sendRedirect ('TestDisponibles')
```

D.7. BorrarTest

```
##                  HEVAH
## Proyecto de fin de carrera de Ingenieria Informática
## Universidad de La Laguna
## Tutora: Carina Gonzalez Gonzalez
## Autores: Fco. Javier Medina Santos
##     Daniel J. Díaz Gonzalez
##     Zebenzui Pérez Ramos
## Descripcion: Se encarga de borrar un test. Elimina también el fichero
##              xml asociado al mismo.

from xml.dom import minidom
from Cheetah.Template import Template
from TestDisponibles import TestDisponibles
from Cifrado import Cifrado

class BorrarTest (TestDisponibles):
    def writeContent (self):
        req = self.request()
        resp = self.response()
        file = req.cookie ('fichero')
        cad_descifrada = Cifrado.Descifrar(fich = file)
        dom = minidom.parseString (cad_descifrada)
        raiz = dom.getElementsByTagName ('TEST')[0]
        t = Template (file='hevah//templates/BorrarTest.tpl.html')
        t.test = raiz.getAttribute ('TEXTO')
        self.mostrar (str(t))
```

D.8. BorrarUsuario

```
##                  HEVAH
## Proyecto de fin de carrera de Ingenieria Informática
## Universidad de La Laguna
## Tutora: Carina Gonzalez Gonzalez
## Autores: Fco. Javier Medina Santos
##     Daniel J. Díaz Gonzalez
##     Zebenzui Pérez Ramos
## Descripcion: Se encarga de borrar un usuario del sistema. Para ello
##              borra la información relacionada con dicho usuario de
##              los ficheros en los que se encuentre. El fichero de
##              perfil no se borra sino que se para a un directorio
##              histórico por si en algún momento fuera preciso recuperar
##              su información.

from Cheetah.Template import Template
from xml.dom import minidom
from index import index
import string
import os
import time
from Cifrado import Cifrado

class BorrarUsuario (index):
    def writeContent (self):
       #El usuario está validado
        if (self.request().hasCookie('grupo') and
     (self.request().hasField('borrar'))):
         #Es un profesor o un administrador
            if (self.request().cookie ('grupo') != "ALUMNO"):
                cad_descifrada = Cifrado.Descifrar(fich =
                                   'hevah//indice.xml')
                indice = minidom.parseString (cad_descifrada)
                raiz = indice.getElementsByTagName ('PERFILES')[0]
```

```
                username = self.request().field ('borrar')
            #borramos los datos del usuario del indice
                nodo = indice.getElementsByTagName (
               string.upper(username))[0]
                raiz.removeChild(nodo)
                f = open ('hevah//indice.xml','w')
            cad_cifrada = Cifrado.Cifrar(cadena = indice.toxml())
                f.write (cad_cifrada)
                f.close()
            #movemos el perfil del directorio actual al histórico
                os.rename('hevah//%s.xml' % string.lower(username),
             'hevah//Historico/%s%s.xml' % ( time.strftime
  ('%d%m%Y%H%M',time.localtime(time.time())),
  string.lower(username)))
                self.response().sendRedirect('GestionUsuarios')
              #Si no es profesor o administrador,
#se le expulsa por acceso no autorizado
            else:
                self.LimpiarCookies()
                self.response().sendRedirect ('index')
     # Al no estar validado, se le manda a la página de Bienvenida
        else:
            self.LimpiarCookies()
            self.response().sendRedirect ('index')
```

D.9. Cifrado

```
##                  HEVAH
## Proyecto de fin de carrera de Ingenieria Informática
## Universidad de La Laguna
## Tutora: Carina Gonzalez Gonzalez
## Autores: Fco. Javier Medina Santos
##     Daniel J. Díaz Gonzalez
##     Zebenzui Pérez Ramos
## Descripcion: Clase utilizada para realizar el cifrado de los ficheros
##              xml. Se importa la clase Cipher de la libreria Crypto
##              contenida en pycrypto 2.0. Esta libreria tiene diversos
##              algoritmos de cifrado de los cuales hemos seleccionado el
##              AES (Advanced Encryption Standard)

#!/usr/bin/python
import string
from Crypto.Cipher import AES

class Cifrado:
    #Se utiliza para cifrado de cadenas o ficheros utilizando el algoritmo
    #AES.
    #Parametros:
    #    * cadena: la cadena a cifrar
    #    * fich: fichero a cifrar
    #    * salvar: si es distinto de None indica que se quiere almacenar en
    #              fichero la información cifrada
    #En caso de que se pase una cadena y un fichero se cifra la cadena
    def Cifrar (self, cadena = None, fich = None, salvar = None):
        cad_aux = ""
        if (fich != None):
            f = open (fich,"r")
            cad_aux = f.read()
            f.close()
        if (cadena != None):
```

```
        cad_aux = cadena
    #Completamos la cadena a un multiplo de 16 con espacios en blanco
    cad = string.join([' '
              for i in range((((len(cad_aux)/16) + 1) * 16)
              - len(cad_aux))], '')
    cad_aux = cad_aux + cad
 #realizamos el cifrado
    obj = AES.new('This is a key456', AES.MODE_ECB)
    cadena_cifrada = obj.encrypt (cad_aux)
    if (salvar != None):
        if (fich != None):
            f = open (fich, "w")
            f.write (cadena_cifrada)
            f.close()
    return (cadena_cifrada)

#Se utiliza para el descifrado de cadenas o ficheros utilizando el
#algoritmo AES.
#Parametros:
#    * cadena: la cadena a descifrar
#    * fich: fichero a descifrar
#    * salvar: si es distinto de None indica que se quiere almacenar en
#              fichero la información descifrada
#En caso de que se pase una cadena y un fichero se descifra la cadena
def Descifrar (self, cadena = None, fich = None, salvar = None):
    cad_aux = ""
    if (fich != None):
        f = open (fich,"r")
        cad_aux = f.read()
        f.close()
    if (cadena != None):
        cad_aux = cadena
 #realizamos el descifrado
    obj = AES.new('This is a key456', AES.MODE_ECB)
    cadena_descifrada = obj.decrypt (cad_aux)
```

```
        if (salvar != None):
            if (fich != None):
                f = open (fich, "w")
                f.write (cadena_descifrada)
                f.close()
        return (cadena_descifrada)

    Cifrar = classmethod (Cifrar)
    Descifrar = classmethod (Descifrar)

if (__name__ == '__main__'):
    a = Cifrado.Cifrar (fich = "tests/test_javi.xml", salvar = 1)
    print a
```

D.10. ComienzaTest

```
##                  HEVAH
## Proyecto de fin de carrera de Ingenieria Informática
## Universidad de La Laguna
## Tutora: Carina Gonzalez Gonzalez
## Autores: Fco. Javier Medina Santos
##     Daniel J. Díaz Gonzalez
##     Zebenzui Pérez Ramos
## Descripcion: realiza las siguientes tareas:
##                  * Selccionar el fichero del test a realizar
##                  * Una vez seleccionado va mostrando las preguntas y
##                    evaluando las respuestas
##                  * Cuando no quedan más preguntas muestra la nota
##                    obtenida al usuario

from xml.dom import minidom
from Cheetah.Template import Template
from TestDisponibles import TestDisponibles
from Test import Test
import string

class ComienzaTest (TestDisponibles):
    def writeContent (self):
        req = self.request()
        resp = self.response()
     #Si aun no se ha seleccionado un test para realizar
        if (req.hasCookie('fich_test')):
         #seleccionamos el fichero del test
            self.test = Test (string.lower(req.cookie('fich_test')),
         req.cookie('username'))
            resp.delCookie('fich_test')
     #Si el usuario ha respondido a una pregunta
        if (self.request().hasField ("respuesta")):
         #Se evalua dicha respuesta para formular la siguiente pregunta
```

```
            self.test.EvaluarRespuesta(self.request().field("respuesta"))
     #Se selecciona la siguiente pregunta
        pregunta = self.test.getPregunta()
     #Si no se devuelve ninguna
        if (pregunta == None):
         #Termina el test y se muestra la nota
            pregunta = {"PREG":"FIN DEL TEST. NO QUEDAN PREGUNTAS",
                     "A":"","B":"","C":""}
            nota = "%.3f" % self.test.user.GetNota()
            self.mostrar ("Fin del test. No hay m#as preguntas.
               Tienes una nota de %s" % nota)
            self.test = None
     #Si aun quedan preguntas se muestra la seleccionada con
#sus respuestas
        else:
            t = Template (file='hevah//templates/ComienzaTest.tpl.html')
            t.enunciado = pregunta["PREG"]
            t.pregA = pregunta["A"]
            t.pregB = pregunta["B"]
            t.pregC = pregunta["C"]
            self.mostrar (str(t))

    #Mostramos las opciones del del usuario en el sistema
    def setTabs (self):
            self.tabs = []
            self.tabs.append (['Salir','Logout',''])
            self.tabs.append(['Bienvienida', 'index',''])
            if (self.request().cookie('grupo') != "ALUMNOS"):
                self.tabs.append (['Edición de Tests',
                    'TestDisponibles',''])
                self.tabs.append (['Gestión de Usuarios',
                    'GestionUsuarios',''])
            self.tabs.append(['Comenzar Tests','SeleccionTest','current'])
            self.tabs.append(['Estadisticas','Estadisticas',''])
```

D.11. Convertir

```
##                  HEVAH
## Proyecto de fin de carrera de Ingenieria Informática
## Universidad de La Laguna
## Tutora: Carina Gonzalez Gonzalez
## Autores: Fco. Javier Medina Santos
##     Daniel J. Díaz Gonzalez
##     Zebenzui Pérez Ramos
## Descripcion: Se utiliza para mostrar correctamente determinados
##              caracteres que ocasionan problemas, como tildes o la
##              letra 'ñ'.

import string

class Convertir:
    def __init__ (self, cadena):
        self.cadena = cadena

    #Convierte caracteres de Latin1 a UTF-8
    def codificar (self):
        codificado = self.cadena
        codificado = string.replace(codificado,'\012',"#a")
        codificado = string.replace(codificado,'\341',"#e")
        codificado = string.replace(codificado,'\351',"#i")
        codificado = string.replace(codificado,'\355',"#o")
        codificado = string.replace(codificado,'\363',"#u")
        codificado = string.replace(codificado,'\372',"#A")
        codificado = string.replace(codificado,'\301',"#E")
        codificado = string.replace(codificado,'\311',"#I")
        codificado = string.replace(codificado,'\315',"#O")
        codificado = string.replace(codificado,'\323',"#U")
        codificado = string.replace(codificado,'\332',"#N")
        codificado = string.replace(codificado,'\277',"#n")
        codificado = string.replace(codificado,'\361',"#?")
```

```
        codificado = string.replace(codificado,'\321',"#!")
        return codificado

    #Convierte caracteres de UTF-8 a Latin1
    def decodificar (self):
        codificado = self.cadena
        codificado =string.replace(codificado,"#a",'\250').encode("latin1")
        codificado = string.replace(codificado,"#e","é")
        codificado = string.replace(codificado,"#i","í")
        codificado = string.replace(codificado,"#o","ó")
        codificado = string.replace(codificado,"#u","ú")
        codificado = string.replace(codificado,"#A","Á")
        codificado = string.replace(codificado,"#E","É")
        codificado = string.replace(codificado,"#I","Í")
        codificado = string.replace(codificado,"#O","Ó")
        codificado = string.replace(codificado,"#U","Ú")
        codificado = string.replace(codificado,"#N","Ñ")
        codificado = string.replace(codificado,"#n","ñ")
        codificado = string.replace(codificado,"#?",">")
        codificado = string.replace(codificado,"#!","<")
        return codificado
```

D.12. CrearConceptoOk

```
##                  HEVAH
## Proyecto de fin de carrera de Ingenieria Informática
## Universidad de La Laguna
## Tutora: Carina Gonzalez Gonzalez
## Autores: Fco. Javier Medina Santos
##     Daniel J. Díaz Gonzalez
##     Zebenzui Pérez Ramos
## Descripcion: Se encarga de rellenar los campos del nuevo concepto y
##              adjuntarlo al tema seleccionado.

from xml.dom import minidom
from Cheetah.Template import Template
from TestDisponibles import TestDisponibles
from Cifrado import Cifrado

class CrearConceptoOk (TestDisponibles):
    def writeContent (self):
        req = self.request()
        resp = self.response()
        file = req.cookie ('fichero')
        cad_descifrada = Cifrado.Descifrar(fich = file)
        dom = minidom.parseString (cad_descifrada)
        tema = req.cookie('tema')
#Seleccionamos el tema al que se añadira el concepto
        lista_temas = dom.getElementsByTagName ('TEMA')
        for _tema in lista_temas:
            if (_tema.getAttribute('ID') == tema):
                nodo_tema = _tema
                break
#Rellenamos los campos del nuevo concepo
        nuevo_concepto = dom.createElement('CONCEPTO')
        nuevo_concepto.setAttribute('ID', req.field('id_concepto'))
        nuevo_concepto.setAttribute('NIVEL', req.field('nivel'))
```

```
        nuevo_concepto.setAttribute('TEXTO', req.field('nuevo_concepto'))
#Se añade
        nodo_tema.appendChild(nuevo_concepto)
        f = open (file,'w')
cad_cifrada = Cifrado.Cifrar(cadena = str(dom.toxml()))
        f.write (cad_cifrada)
        f.close()
        resp.sendRedirect ('VerTest')
```

D.13. CrearPreguntaOk

```
##                  HEVAH
## Proyecto de fin de carrera de Ingenieria Informática
## Universidad de La Laguna
## Tutora: Carina Gonzalez Gonzalez
## Autores: Fco. Javier Medina Santos
##     Daniel J. Díaz Gonzalez
##     Zebenzui Pérez Ramos
## Descripcion: Se encarga de rellenar los campos de la nueva pregunta y
##              añadirla al concepto seleccionado.

from xml.dom import minidom
from Cheetah.Template import Template
from TestDisponibles import TestDisponibles
from Cifrado import Cifrado

class CrearPreguntaOk (TestDisponibles):
    def writeContent (self):
        req = self.request()
        resp = self.response()
        #Comprobamos que haya seleccionado una respuesta correcta
        file = req.cookie ('fichero')
        cad_descifrada = Cifrado.Descifrar(fich = file)
        dom = minidom.parseString (cad_descifrada)
        tema = req.cookie('tema')
        concepto = req.cookie('concepto')
     #Buscamos el tema al que se añadirá la pregunta
        lista_temas = dom.getElementsByTagName ('TEMA')
        for _tema in lista_temas:
            if (_tema.getAttribute('ID') == tema):
                nodo_tema = _tema
                break
     #Buscamos el concepto al que se añadirá la pregunta
        lista_conceptos = [e for e in nodo_tema.childNodes
```

```
                if e.nodeType == e.ELEMENT_NODE]
    for _concepto in lista_conceptos:
        if (_concepto.getAttribute('ID') == concepto):
            nodo_concepto = _concepto
            break

    #Creamos la nueva Pregunta
    nueva_pregunta = dom.createElement('PREGUNTA')
    nueva_pregunta.setAttribute('IDPREGUNTA', req.field('id_pregunta'))
    nueva_pregunta.setAttribute('PADIVINAR',req.field('prob_adivinar'))
    nueva_pregunta.setAttribute('PSABER', req.field('prob_saber'))
    nueva_pregunta.setAttribute('RESULTADO',
                          req.field('respuesta_correcta'))
    nueva_pregunta.setAttribute('TEXTO', self.codificar(
                          req.field('nueva_pregunta')))
#Se añade al concepto
    nodo_concepto.appendChild(nueva_pregunta)

    f = open (file,'w')
cad_cifrada = Cifrado.Cifrar(cadena = str(dom.toxml()))
    f.write (cad_cifrada)
    f.close()
    resp.sendRedirect ('VerConcepto')
```

D.14. CrearPregunta

```
##                  HEVAH
## Proyecto de fin de carrera de Ingenieria Informática
## Universidad de La Laguna
## Tutora: Carina Gonzalez Gonzalez
## Autores: Fco. Javier Medina Santos
##     Daniel J. Díaz Gonzalez
##     Zebenzui Pérez Ramos
## Descripcion: Se encarga de crear una nueva pregunta para un concepto
##              determinado. Asigna el identificador a esta nueva pregunta.

from xml.dom import minidom
from Cheetah.Template import Template
from TestDisponibles import TestDisponibles
from Cifrado import Cifrado

class CrearPregunta (TestDisponibles):
    def writeContent (self):
        req = self.request()
        resp = self.response()
        file = req.cookie ('fichero')
        tema = req.cookie ('tema')
        concepto = req.cookie ('concepto')
        cad_descifrada = Cifrado.Descifrar(fich = file)
        dom = minidom.parseString (cad_descifrada)
     #Buscamos el tema al que se añadira la pregunta
        lista_temas = dom.getElementsByTagName ('TEMA')
        for _tema in lista_temas:
            if (_tema.getAttribute('ID') == tema):
                nodo_tema = _tema
                break
     #Buscamos el concepto al que se añadira la pregunta
        lista_conceptos = [e for e in nodo_tema.childNodes
                    if e.nodeType == e.ELEMENT_NODE]
```

```
        for _concepto in lista_conceptos:
            if (_concepto.getAttribute('ID') == concepto):
                nodo_concepto = _concepto
                break
    #Buscamos la ultima pregunta para añadir la nueva al final
        lista_preguntas = [e for e in nodo_concepto.childNodes
                    if e.nodeType == e.ELEMENT_NODE]
        for _pregunta in lista_preguntas:
            nodo_pregunta = _pregunta

    #Miramos si ya habían preguntas
        if (lista_preguntas != []):
            id_pregunta = int(nodo_pregunta.getAttribute('IDPREGUNTA')) + 1
        else:
            id_pregunta = 1
        t = Template (file='hevah//templates/CrearPregunta.tpl.html')
        t.tema = nodo_tema
        t.concepto = nodo_concepto
        t.id_pregunta = id_pregunta
        self.mostrar (str(t))
```

D.15. CrearRespuestaOk

```
##                  HEVAH
## Proyecto de fin de carrera de Ingenieria Informática
## Universidad de La Laguna
## Tutora: Carina Gonzalez Gonzalez
## Autores: Fco. Javier Medina Santos
##     Daniel J. Díaz Gonzalez
##     Zebenzui Pérez Ramos
## Descripcion: Se encarga de rellenar los campos de la nueva respuesta y
##              añadirla a la pregunta correspondiente.

from xml.dom import minidom
from Cheetah.Template import Template
from TestDisponibles import TestDisponibles
from Cifrado import Cifrado

class CrearRespuestaOk (TestDisponibles):
    def writeContent (self):
        req = self.request()
        resp = self.response()
        #Comprobamos que haya seleccionado una respuesta correcta
        file = req.cookie ('fichero')
        cad_descifrada = Cifrado.Descifrar(fich = file)
        dom = minidom.parseString (cad_descifrada)
        tema = req.cookie('tema')
        concepto = req.cookie('concepto')
        pregunta = req.cookie('pregunta')
        #Buscamos el tema al que se añadirá la respuesta
        lista_temas = dom.getElementsByTagName ('TEMA')
        for _tema in lista_temas:
            if (_tema.getAttribute('ID') == tema):
                nodo_tema = _tema
                break
     #Buscamos el concepto al que se añadirá la respuesta
```

```
   lista_conceptos = [e for e in nodo_tema.childNodes
                if e.nodeType == e.ELEMENT_NODE]
   for _concepto in lista_conceptos:
       if (_concepto.getAttribute('ID') == concepto):
           nodo_concepto = _concepto
           break
#Buscamos la pregunta a la que se añadirá la respuesta
   lista_preguntas = [e for e in nodo_concepto.childNodes
                if e.nodeType == e.ELEMENT_NODE]
   for _pregunta in lista_preguntas:
       if (_pregunta.getAttribute('IDPREGUNTA') == pregunta):
           nodo_pregunta = _pregunta
           break

   #Creamos las respuestas
   nueva_respuesta = dom.createElement('RESPUESTA')
   nueva_respuesta.setAttribute('IDRESPUESTA',
                          req.field("id_respuesta"))
   nueva_respuesta.setAttribute('TEXTO', self.codificar(
                          req.field('respuesta')))
#Se añade a la pregunta
   nodo_pregunta.appendChild(nueva_respuesta)

   f = open (file,'w')
cad_cifrada = Cifrado.Cifrar(cadena = str(dom.toxml()))
   f.write (cad_cifrada)
   f.close()
   resp.sendRedirect ('VerPregunta')
```

D.16. CrearRespuesta

```
##                  HEVAH
## Proyecto de fin de carrera de Ingenieria Informática
## Universidad de La Laguna
## Tutora: Carina Gonzalez Gonzalez
## Autores: Fco. Javier Medina Santos
##     Daniel J. Díaz Gonzalez
##     Zebenzui Pérez Ramos
## Descripcion: Se encarga de crear una nueva respuesta para una pregunta
##              seleccionada. Se le asigna el identificador.

from xml.dom import minidom
from Cheetah.Template import Template
from TestDisponibles import TestDisponibles
from Cifrado import Cifrado

class CrearRespuesta (TestDisponibles):
    def writeContent (self):
        req = self.request()
        resp = self.response()
        file = req.cookie ('fichero')
        tema = req.cookie ('tema')
        concepto = req.cookie ('concepto')
        pregunta = req.cookie ('pregunta')
        cad_descifrada = Cifrado.Descifrar(fich = file)
        dom = minidom.parseString (cad_descifrada)
        #Buscamos el tema al que se añadira la respuesta
     lista_temas = dom.getElementsByTagName ('TEMA')
        for _tema in lista_temas:
            if (_tema.getAttribute('ID') == tema):
                nodo_tema = _tema
                break
     #Buscamos el concepto al que se añadirá la respuesta
        lista_conceptos = [e for e in nodo_tema.childNodes
```

```
                if e.nodeType == e.ELEMENT_NODE]
    for _concepto in lista_conceptos:
        if (_concepto.getAttribute('ID') == concepto):
            nodo_concepto = _concepto
            break
 #Buscamos la pregunta a la que se añadirá la respuesta
    lista_preguntas = [e for e in nodo_concepto.childNodes
                if e.nodeType == e.ELEMENT_NODE]
    for _pregunta in lista_preguntas:
        if (_pregunta.getAttribute('IDPREGUNTA') == pregunta):
            nodo_pregunta = _pregunta
            break
 #Buscamos la última respuesta
    lista_respuestas = [e for e in nodo_pregunta.childNodes
                 if e.nodeType == e.ELEMENT_NODE]
    for _respuesta in lista_respuestas:
        nodo_respuesta = _respuesta

 #Miramos si hay alguna
    if (lista_respuestas != []):
        id_respuesta =
chr(ord(nodo_respuesta.getAttribute('IDRESPUESTA')) + 1)
    else:
        id_respuesta = "A"

    t = Template (file='hevah//templates/CrearRespuesta.tpl.html')
    t.tema = nodo_tema
    t.concepto = nodo_concepto
    t.pregunta = nodo_pregunta
    t.id_respuesta = id_respuesta
    self.mostrar (str(t))
```

D.17. CrearTemaOk

```
##                  HEVAH
## Proyecto de fin de carrera de Ingenieria Informática
## Universidad de La Laguna
## Tutora: Carina Gonzalez Gonzalez
## Autores: Fco. Javier Medina Santos
##     Daniel J. Díaz Gonzalez
##     Zebenzui Pérez Ramos
## Descripcion: Se encarga de rellenar los campos del nuevo tema y
##              asignarlo al test en seleccionado.

from xml.dom import minidom
from Cheetah.Template import Template
from TestDisponibles import TestDisponibles
from Cifrado import Cifrado

class CrearTemaOk (TestDisponibles):
    def writeContent (self):
        req = self.request()
        resp = self.response()
        file = req.cookie ('fichero')
        cad_descifrada = Cifrado.Descifrar(fich = file)
        dom = minidom.parseString (cad_descifrada)
        test = dom.getElementsByTagName ('TEST')[0]
     #Rellenamos los campos del nuevo tema
        nuevo_tema = dom.createElement('TEMA')
        nuevo_tema.setAttribute('ID', req.field('id_tema'))
        nuevo_tema.setAttribute('TEXTO', req.field('nuevo_tema'))
     #Se añade al test
        test.appendChild(nuevo_tema)
        f = open (file,'w')
     cad_cifrada = Cifrado.Cifrar(cadena = str(dom.toxml()))
        f.write (cad_cifrada)
        f.close()
```

```
        resp.sendRedirect ('TestDisponibles')
```

D.18. CrearTema

```
##                  HEVAH
## Proyecto de fin de carrera de Ingenieria Informática
## Universidad de La Laguna
## Tutora: Carina Gonzalez Gonzalez
## Autores: Fco. Javier Medina Santos
##     Daniel J. Díaz Gonzalez
##     Zebenzui Pérez Ramos
## Descripcion: Se encarga de crear un nuevo tema para el test
##              seleccionado. Se asigna su identificador.

from xml.dom import minidom
from Cheetah.Template import Template
from TestDisponibles import TestDisponibles
from Cifrado import Cifrado

class CrearTema (TestDisponibles):
    def writeContent (self):
        req = self.request()
        resp = self.response()
        file = req.cookie ('fichero')
        cad_descifrada = Cifrado.Descifrar(fich = file)
        dom = minidom.parseString (cad_descifrada)
        test = dom.getElementsByTagName ('TEST')[0]
     #Buscamos el ultimo tema para añadirlo al final
        lista_temas = dom.getElementsByTagName ('TEMA')
        nodo_tema = None
        for _tema in lista_temas:
            nodo_tema = _tema
     #Miramos si es el primero
        if (nodo_tema != None):
            id_tema = int(nodo_tema.getAttribute('ID')) + 1
        else:
            id_tema = 1
```

```
t = Template (file='hevah//templates/CrearTema.tpl.html')
t.id_tema = id_tema
self.mostrar (str(t))
```

D.19. EditarConceptoOK

```
##                  HEVAH
## Proyecto de fin de carrera de Ingenieria Informática
## Universidad de La Laguna
## Tutora: Carina Gonzalez Gonzalez
## Autores: Fco. Javier Medina Santos
##     Daniel J. Díaz Gonzalez
##     Zebenzui Pérez Ramos
## Descripcion: Se encarga de rellenar los datos modificados del concepto y
##              actualizar el fichero de test.

from Cheetah.Template import Template
from TestDisponibles import TestDisponibles
from xml.dom import minidom
from Cifrado import Cifrado

class EditarConceptoOK(TestDisponibles):
    def writeContent (self):
        req = self.request()
        resp = self.response()
        file = req.cookie ('fichero')
        tema = req.cookie ('tema')
        concepto = req.cookie('concepto')
        cad_descifrada = Cifrado.Descifrar(fich = file)
        dom = minidom.parseString (cad_descifrada)
     #Buscamos el tema al que pertenece el concepto
        lista_temas = dom.getElementsByTagName ('TEMA')
        for _tema in lista_temas:
            if (_tema.getAttribute('ID') == tema):
                nodo_tema = _tema
                break
     #Buscamos el concepto a modificar
        lista_conceptos = [e for e in nodo_tema.childNodes
                    if e.nodeType == e.ELEMENT_NODE]
```

```
    for _concepto in lista_conceptos:
        if (_concepto.getAttribute('ID') == concepto):
            nodo_concepto = _concepto
            break
#Asignamos los nuevos valores
    if (req.hasField("nuevo_concepto")):
        nuevo_concepto = req.field ("nuevo_concepto")
        if (nuevo_concepto != ""):
            nodo_concepto.setAttribute ('TEXTO',
           self.codificar(str(nuevo_concepto)))
    nivel = req.field ("nivel")
    nodo_concepto.setAttribute ('NIVEL', nivel)
    f = open (file,'w')
cad_cifrada = Cifrado.Cifrar(cadena = str(dom.toxml()))
    f.write (cad_cifrada)
    f.close()
    resp.sendRedirect ('VerConcepto')
```

D.20. EditarConcepto

```
##                  HEVAH
## Proyecto de fin de carrera de Ingenieria Informática
## Universidad de La Laguna
## Tutora: Carina Gonzalez Gonzalez
## Autores: Fco. Javier Medina Santos
##     Daniel J. Díaz Gonzalez
##     Zebenzui Pérez Ramos
## Descripcion: Se encarga de presentar los datos del concepto seleccionado
##              para que puedan ser modificados.

from Cheetah.Template import Template
from TestDisponibles import TestDisponibles
from xml.dom import minidom
from Cifrado import Cifrado

class EditarConcepto(TestDisponibles):
    def writeContent (self):
        req = self.request()
        resp = self.response()
        file = req.cookie ('fichero')
        tema = req.cookie ('tema')
        concepto = req.cookie('concepto')
        cad_descifrada = Cifrado.Descifrar(fich = file)
        dom = minidom.parseString (cad_descifrada)
        #Buscamos el tema donde esta el concepto
        lista_temas = dom.getElementsByTagName ('TEMA')
        for _tema in lista_temas:
            if (_tema.getAttribute('ID') == tema):
                nodo_tema = _tema
                break
     #Buscamos el concepto
        lista_conceptos = [e for e in nodo_tema.childNodes
                    if e.nodeType == e.ELEMENT_NODE]
```

```
for _concepto in lista_conceptos:
    if (_concepto.getAttribute('ID') == concepto):
        nodo_concepto = _concepto
        break
t = Template (file='hevah//templates/EditarConcepto.tpl.html')
t.tema = nodo_tema.getAttribute ('TEXTO')
t.concepto = nodo_concepto
if (req.hasField ("enun_old")):
    t.new_enun = req.field("enun_old")
else:
    t.new_enun = ""
self.mostrar (str(t))
```

D.21. EditarPreguntaOK

```
##                  HEVAH
## Proyecto de fin de carrera de Ingenieria Informática
## Universidad de La Laguna
## Tutora: Carina Gonzalez Gonzalez
## Autores: Fco. Javier Medina Santos
##     Daniel J. Díaz Gonzalez
##     Zebenzui Pérez Ramos
## Descripcion: Se encarga de rellenar los datos modificados de la pregunta
##              y actualizar el fichero de test.

from Cheetah.Template import Template
from TestDisponibles import TestDisponibles
from xml.dom import minidom
from Cifrado import Cifrado

class EditarPreguntaOK(TestDisponibles):
    def writeContent (self):
        req = self.request()
        resp = self.response()
        file = req.cookie ('fichero')
        tema = req.cookie ('tema')
        concepto = req.cookie('concepto')
        pregunta = req.cookie('pregunta')
        cad_descifrada = Cifrado.Descifrar(fich = file)
        dom = minidom.parseString (cad_descifrada)
     #Buscamos el tema al que pertenece la pregunta
        lista_temas = dom.getElementsByTagName ('TEMA')
        for _tema in lista_temas:
            if (_tema.getAttribute('ID') == tema):
                nodo_tema = _tema
                break
     #Buscamos el concepto al que pertence la pregunta
        lista_conceptos = [e for e in nodo_tema.childNodes
```

```
                if e.nodeType == e.ELEMENT_NODE]
    for _concepto in lista_conceptos:
        if (_concepto.getAttribute('ID') == concepto):
            nodo_concepto = _concepto
            break
#Buscamos la pregunta modificar
    lista_preguntas = [e for e in nodo_concepto.childNodes
                if e.nodeType == e.ELEMENT_NODE]
    for _pregunta in lista_preguntas:
        if (_pregunta.getAttribute('IDPREGUNTA') == pregunta):
            nodo_pregunta = _pregunta
            break
#Actualizamos los campos
    if (req.hasField("nueva_pregunta")):
        nueva_pregunta = req.field ("nueva_pregunta")
        if (nueva_pregunta != ""):
            nodo_pregunta.setAttribute ('TEXTO',
           self.codificar(str(nueva_pregunta)))
    if (req.hasField ("prob_saber")):
        self.log (req.field ("prob_saber"))
        prob_saber = float (req.field("prob_saber"))
        prob_saber = prob_saber / 100
        self.log (prob_saber)
        nodo_pregunta.setAttribute ('PSABER', ("%.2lf" % prob_saber))
    if (req.hasField ("prob_adivinar")):
        prob_adivinar = float (req.field("prob_adivinar"))
        prob_adivinar = prob_adivinar / 100
        nodo_pregunta.setAttribute ('PADIVINAR', ("%.2lf"
                        % prob_adivinar))
    correcta = req.field ("correcta")
    nodo_pregunta.setAttribute ('RESULTADO', str(correcta))
    f = open (file,'w')
    cad_cifrada = Cifrado.Cifrar(cadena = str(dom.toxml()))
    f.write (cad_cifrada)
    f.close()
```

```
        resp.sendRedirect ('VerConcepto')
```

D.22. EditarPregunta

```
##                  HEVAH
## Proyecto de fin de carrera de Ingenieria Informática
## Universidad de La Laguna
## Tutora: Carina Gonzalez Gonzalez
## Autores: Fco. Javier Medina Santos
##     Daniel J. Díaz Gonzalez
##     Zebenzui Pérez Ramos
## Descripcion: Se encarga de presentar los datos de la pregunta
##              seleccionada para que puedan ser modificados.

from Cheetah.Template import Template
from TestDisponibles import TestDisponibles
from xml.dom import minidom
from Cifrado import Cifrado

class EditarPregunta(TestDisponibles):
    def writeContent (self):
        req = self.request()
        resp = self.response()
        file = req.cookie ('fichero')
        tema = req.cookie ('tema')
        concepto = req.cookie('concepto')
        pregunta = req.cookie('pregunta')
        cad_descifrada = Cifrado.Descifrar(fich = file)
        dom = minidom.parseString (cad_descifrada)
      #Buscamos el tema al que pertenece la pregunta
        lista_temas = dom.getElementsByTagName ('TEMA')
        for _tema in lista_temas:
            if (_tema.getAttribute('ID') == tema):
                nodo_tema = _tema
                break
     #Buscamos el concepto al que pertenece la pregunta
        lista_conceptos = [e for e in nodo_tema.childNodes
```

```
                if e.nodeType == e.ELEMENT_NODE]
    for _concepto in lista_conceptos:
        if (_concepto.getAttribute('ID') == concepto):
            nodo_concepto = _concepto
            break
#Buscamos la pregunta a modificar
    lista_preguntas = [e for e in nodo_concepto.childNodes
                if e.nodeType == e.ELEMENT_NODE]
    for _pregunta in lista_preguntas:
        if (_pregunta.getAttribute('IDPREGUNTA') == pregunta):
            nodo_pregunta = _pregunta
            break
    lista_respuestas = [e for e in nodo_pregunta.childNodes
                 if e.nodeType == e.ELEMENT_NODE]
#Mostramos los datos
    t = Template (file='hevah//templates/EditarPregunta.tpl.html')
    t.tema = nodo_tema.getAttribute ('TEXTO')
    t.concepto = nodo_concepto.getAttribute ('TEXTO')
    t.pregunta = nodo_pregunta
    t.lista_respuestas = lista_respuestas
    t.prob_saber = float(nodo_pregunta.getAttribute ('PSABER'))*100
    t.prob_adivinar = float(nodo_pregunta.getAttribute ('PADIVINAR'))
              *100
    if (req.hasField ("enun_old")):
        t.new_enun = req.field("enun_old")
    else:
        t.new_enun = ""
    self.mostrar (str(t))
```

D.23. EditarRespuestaOK

```
##                  HEVAH
## Proyecto de fin de carrera de Ingenieria Informática
## Universidad de La Laguna
## Tutora: Carina Gonzalez Gonzalez
## Autores: Fco. Javier Medina Santos
##     Daniel J. Díaz Gonzalez
##     Zebenzui Pérez Ramos
## Descripcion: Se encarga de rellenar los datos modificados de la
##              respuesta y actualizar el fichero de test.

from Cheetah.Template import Template
from TestDisponibles import TestDisponibles
from xml.dom import minidom
from Cifrado import Cifrado

class EditarRespuestaOK(TestDisponibles):
    def writeContent (self):
        req = self.request()
        resp = self.response()
        file = req.cookie ('fichero')
        tema = req.cookie ('tema')
        concepto = req.cookie('concepto')
        pregunta = req.cookie('pregunta')
        respuesta = req.cookie ('respuesta')
        cad_descifrada = Cifrado.Descifrar(fich = file)
        dom = minidom.parseString (cad_descifrada)
     #Buscamos el tema al que pertenece la respuesta
        lista_temas = dom.getElementsByTagName ('TEMA')
        for _tema in lista_temas:
            if (_tema.getAttribute('ID') == tema):
                nodo_tema = _tema
                break
     #Buscamos el concepto al que pertenece la respuesta
```

```
   lista_conceptos = [e for e in nodo_tema.childNodes
                if e.nodeType == e.ELEMENT_NODE]
   for _concepto in lista_conceptos:
       if (_concepto.getAttribute('ID') == concepto):
           nodo_concepto = _concepto
           break
#Buscamos la pregunta a la que pertenece la respuesta
   lista_preguntas = [e for e in nodo_concepto.childNodes
                if e.nodeType == e.ELEMENT_NODE]
   for _pregunta in lista_preguntas:
       if (_pregunta.getAttribute('IDPREGUNTA') == pregunta):
           nodo_pregunta = _pregunta
           break
#Buscamos la respuesta a modificar
   lista_respuestas = [e for e in nodo_pregunta.childNodes
                 if e.nodeType == e.ELEMENT_NODE]
   for _respuesta in lista_respuestas:
       if (_respuesta.getAttribute ('IDRESPUESTA') == respuesta):
           nodo_respuesta = _respuesta
           break
#Actualizamos los datos
   nueva_respuesta = req.field ("nueva_respuesta")
   self.log(nueva_respuesta)
   nodo_respuesta.setAttribute ('TEXTO', self.codificar
                          (str(nueva_respuesta)))
   f = open (file,'w')
cad_cifrada = Cifrado.Cifrar(cadena = str(dom.toxml()))
   f.write (cad_cifrada)
   f.close()
   resp.sendRedirect ('VerPregunta')
```

D.24. EditarRespuesta

```
##                  HEVAH
## Proyecto de fin de carrera de Ingenieria Informática
## Universidad de La Laguna
## Tutora: Carina Gonzalez Gonzalez
## Autores: Fco. Javier Medina Santos
##     Daniel J. Díaz Gonzalez
##     Zebenzui Pérez Ramos
## Descripcion: Se encarga de presentar los datos de la respuesta
##              seleccionada para que puedan ser modificados.

from Cheetah.Template import Template
from TestDisponibles import TestDisponibles
from xml.dom import minidom
from Cifrado import Cifrado

class EditarRespuesta(TestDisponibles):
    def writeContent (self):
        req = self.request()
        resp = self.response()
        file = req.cookie ('fichero')
        tema = req.cookie ('tema')
        concepto = req.cookie('concepto')
        pregunta = req.cookie('pregunta')
        respuesta = req.cookie ('respuesta')
        cad_descifrada = Cifrado.Descifrar(fich = file)
        dom = minidom.parseString (cad_descifrada)
     #Buscamos el tema al que pertenece la respuesta
        lista_temas = dom.getElementsByTagName ('TEMA')
        for _tema in lista_temas:
            if (_tema.getAttribute('ID') == tema):
                nodo_tema = _tema
                break
     #Buscamos el concepto al que pertenece la respuesta
```

```
        lista_conceptos = [e for e in nodo_tema.childNodes
                    if e.nodeType == e.ELEMENT_NODE]
        for _concepto in lista_conceptos:
            if (_concepto.getAttribute('ID') == concepto):
                nodo_concepto = _concepto
                break
    #Buscamos la pregunta a la que pertence la respuesta
        lista_preguntas = [e for e in nodo_concepto.childNodes
                    if e.nodeType == e.ELEMENT_NODE]
        for _pregunta in lista_preguntas:
            if (_pregunta.getAttribute('IDPREGUNTA') == pregunta):
                nodo_pregunta = _pregunta
                break
    #Buscamos la repuesta a modificar
        lista_respuestas = [e for e in nodo_pregunta.childNodes
                     if e.nodeType == e.ELEMENT_NODE]
        for _respuesta in lista_respuestas:
            if (_respuesta.getAttribute ('IDRESPUESTA') == respuesta):
                nodo_respuesta = _respuesta
                break

        t = Template (file='hevah//templates/EditarRespuesta.tpl.html')
        t.tema = nodo_tema.getAttribute ('TEXTO')
        t.concepto = nodo_concepto.getAttribute ('TEXTO')
        t.pregunta = nodo_pregunta.getAttribute ('TEXTO')
        t.respuesta = nodo_respuesta
        self.mostrar (str(t))
```

D.25. EditarTemaOK

```
##                  HEVAH
## Proyecto de fin de carrera de Ingenieria Informática
## Universidad de La Laguna
## Tutora: Carina Gonzalez Gonzalez
## Autores: Fco. Javier Medina Santos
##     Daniel J. Díaz Gonzalez
##     Zebenzui Pérez Ramos
## Descripcion: Se encarga de rellenar los datos modificados del tema y
##              actualizar el fichero de test.

from Cheetah.Template import Template
from TestDisponibles import TestDisponibles
from xml.dom import minidom
from Cifrado import Cifrado

class EditarTemaOK(TestDisponibles):
    def writeContent (self):
        req = self.request()
        resp = self.response()
        file = req.cookie ('fichero')
        tema = req.cookie ('tema')
        cad_descifrada = Cifrado.Descifrar(fich = file)
        dom = minidom.parseString (cad_descifrada)
     #Buscamos el tema a modificar
        lista_temas = dom.getElementsByTagName ('TEMA')
        for _tema in lista_temas:
            if (_tema.getAttribute('ID') == tema):
                nodo_tema = _tema
                break
     #Activamos los datos
        if (req.hasField("nuevo_tema")):
            nuevo_tema = req.field ("nuevo_tema")
            if (nuevo_tema != ""):
```

```
            nodo_tema.setAttribute ('TEXTO', self.codificar(
                    str(nuevo_tema)))
   f = open (file,'w')
cad_cifrada = Cifrado.Cifrar(cadena = str(dom.toxml()))
   f.write (cad_cifrada)
   f.close()
   resp.sendRedirect ('VerTema')
```

D.26. EditarTema

```
##                  HEVAH
## Proyecto de fin de carrera de Ingenieria Informática
## Universidad de La Laguna
## Tutora: Carina Gonzalez Gonzalez
## Autores: Fco. Javier Medina Santos
##     Daniel J. Díaz Gonzalez
##     Zebenzui Pérez Ramos
## Descripcion: Se encarga de presentar los datos del tema seleccionado
##              para que puedan ser modificados.

from Cheetah.Template import Template
from TestDisponibles import TestDisponibles
from xml.dom import minidom
from Cifrado import Cifrado

class EditarTema (TestDisponibles):
    def writeContent (self):
        req = self.request()
        resp = self.response()
        file = req.cookie ('fichero')
        tema = req.cookie ('tema')
        cad_descifrada = Cifrado.Descifrar(fich = file)
        dom = minidom.parseString (cad_descifrada)
        #Buscamos el tema a modificar
        lista_temas = dom.getElementsByTagName ('TEMA')
        for _tema in lista_temas:
            if (_tema.getAttribute('ID') == tema):
                nodo_tema = _tema
                break
        t = Template (file='hevah//templates/EditarTema.tpl.html')
        t.tema = nodo_tema
        if (req.hasField ("enun_old")):
            t.new_enun = req.field("enun_old")
```

```
        else:
            t.new_enun = ""
        self.mostrar (str(t))
```

D.27. EditarTestOK

```
##                  HEVAH
## Proyecto de fin de carrera de Ingenieria Informática
## Universidad de La Laguna
## Tutora: Carina Gonzalez Gonzalez
## Autores: Fco. Javier Medina Santos
##     Daniel J. Díaz Gonzalez
##     Zebenzui Pérez Ramos
## Descripcion: Se encarga de rellenar los datos modificados del test y
##              actualizar el fichero.

from Cheetah.Template import Template
from TestDisponibles import TestDisponibles
from xml.dom import minidom
from Cifrado import Cifrado

class EditarTestOK(TestDisponibles):
    def writeContent (self):
        req = self.request()
        resp = self.response()
        file = req.cookie ('fichero')
        cad_descifrada = Cifrado.Descifrar(fich = file)
        dom = minidom.parseString (cad_descifrada)
     #Buscamos el test a modificar y actualizamos los datos
        nodo_test = dom.getElementsByTagName ('TEST')[0]
        if (req.hasField("nuevo_test")):
            nuevo_test = req.field ("nuevo_test")
            if (nuevo_test != ""):
                nodo_test.setAttribute ('TEXTO', self.codificar(
                        str(nuevo_test)))
        f = open (file,'w')
     cad_cifrada = Cifrado.Cifrar(cadena = str(dom.toxml()))
        f.write (cad_cifrada)
        f.close()
```

```
        resp.sendRedirect ('TestDisponibles')
```

D.28. EditarTest

```
##                  HEVAH
## Proyecto de fin de carrera de Ingenieria Informática
## Universidad de La Laguna
## Tutora: Carina Gonzalez Gonzalez
## Autores: Fco. Javier Medina Santos
##     Daniel J. Díaz Gonzalez
##     Zebenzui Pérez Ramos
## Descripcion: Se encarga de presentar los datos del test seleccionado
##              para que puedan ser modificados.

from Cheetah.Template import Template
from TestDisponibles import TestDisponibles
from xml.dom import minidom
from Cifrado import Cifrado

class EditarTest (TestDisponibles):
    def writeContent (self):
        req = self.request()
        resp = self.response()
        file = req.cookie ('fichero')
        cad_descifrada = Cifrado.Descifrar(fich = file)
        dom = minidom.parseString (cad_descifrada)
        nodo_test = dom.getElementsByTagName ('TEST')[0]
        t = Template (file='hevah//templates/EditarTest.tpl.html')
        t.test = nodo_test
        if (req.hasField ("enun_old")):
            t.new_enun = req.field("enun_old")
        else:
            t.new_enun = ""
        self.mostrar (str(t))
```

D.29. EstadisticaAlumno

```
##                  HEVAH
## Proyecto de fin de carrera de Ingenieria Informática
## Universidad de La Laguna
## Tutora: Carina Gonzalez Gonzalez
## Autores: Fco. Javier Medina Santos
##     Daniel J. Díaz Gonzalez
##     Zebenzui Pérez Ramos
## Descripcion: Se encarga de presentar los test que ha relizado el alumno
##              para ver las estadisticas del que sea seleccionado.

from xml.dom import minidom
from Cheetah.Template import Template
from index import index
import os
import string
import re
from Cifrado import Cifrado

class EstadisticaAlumno (index):
    def writeContent (self):
        req = self.request()
        resp = self.response()
        if (req.hasField('radio_alu')):
            cad_descifrada = Cifrado.Descifrar(fich = "hevah//%s.xml"
                 % req.field('radio_alu'))
            dom = minidom.parseString (cad_descifrada)
         #Cogemos los test del fichero
            test = dom.getElementsByTagName ('TEST')
            resp.setCookie('alumno', req.field('radio_alu'))
            t = Template (file='hevah//templates/VerEstadistica.tpl.html')
            t.lista_test = test
            self.mostrar (str(t))
```

D.30. Estadisticas

```
##                  HEVAH
## Proyecto de fin de carrera de Ingenieria Informática
## Universidad de La Laguna
## Tutora: Carina Gonzalez Gonzalez
## Autores: Fco. Javier Medina Santos
##     Daniel J. Díaz Gonzalez
##     Zebenzui Pérez Ramos
## Descripcion: Se encarga de la presentacion de estadisticas. Para ello
##              primero distingue el grupo al que pertenece el usuario que
##              quiere ver las estadisticas ya que tienen distintos
##              provilegios. Si es un alumno, se le mostrarán los test que
##              haya realizado. Si es un profesor se le mostraran los test
##              de sus alumnos. Si es un administrador se le muestran
##              todos.

from xml.dom import minidom
from Cheetah.Template import Template
from index import index
import os
import string
import re
from Cifrado import Cifrado

class Estadisticas (index):
    def writeContent (self):
        req = self.request()
        resp = self.response()
        #Si es un profesor el que quiere ver las Estadisticas
  #de sus alumnos
        if (req.cookie('grupo') == 'PROFESORES'):
            cad_descifrada = Cifrado.Descifrar(fich = "hevah//%s.xml"
                                % req.cookie('username'))
            dom = minidom.parseString (cad_descifrada)
```

```
            alumnos = dom.getElementsByTagName ('ALUMNO')[0]
         #Mostramos sus alumnos para que seleccione cual quiere ver
            lista_alumnos= [e for e in alumnos.childNodes
                if e.nodeType == e.ELEMENT_NODE]
            t = Template (file='hevah//templates/SeleccionAlumno.tpl.html')
            t.lista_alumnos = lista_alumnos
            self.mostrar (str(t))
        else:
            #Si un alumno quiere ver sus estadisticas
            if (req.cookie('grupo') == 'ALUMNOS'):
                cad_descifrada = Cifrado.Descifrar(fich =
                  "hevah//%s.xml" % req.cookie('username'))
                dom = minidom.parseString (cad_descifrada)
          #Se le muestran los test que ha realizado
                test = dom.getElementsByTagName ('TEST')
                t = Template (file=
               'hevah//templates/VerEstadistica.tpl.html')
                t.lista_test = test
                self.mostrar (str(t))
            else:
                #Si un administrador quiere ver estadisticas
                if (req.cookie('grupo') == 'ADMINISTRADOR'):
                    cad_descifrada = Cifrado.Descifrar(fich =
                   "hevah//indice.xml")
                    dom = minidom.parseString (cad_descifrada)
              #Se le muestran todos los usuarios para que seleccione
              #el que quiere ver
                    usuarios = dom.getElementsByTagName ('PERFILES')[0]
                    lista_usuarios = [e for e in usuarios.childNodes
                    if e.nodeType == e.ELEMENT_NODE]
                    lista_alumnos = []
                    for usuario in lista_usuarios:
                        if (usuario.getAttribute('GRUPO') == "ALUMNOS"):
                            lista_alumnos.append(usuario)
                    t = Template (file=
```

```
            'hevah//templates/SeleccionAlumno.tpl.html')
                    t.lista_alumnos = lista_alumnos
                    self.mostrar (str(t))

    def setTabs (self):
        self.tabs = []
        self.tabs.append (['Salir','Logout',''])
        self.tabs.append(['Bienvienida', 'index',''])
        if (self.request().cookie('grupo') != "ALUMNOS"):
            self.tabs.append (['Edición de Tests','TestDisponibles',''])
            self.tabs.append (['Gestión de Usuarios','GestionUsuarios',''])
        self.tabs.append(['Comenzar Tests','SeleccionTest',''])
        self.tabs.append(['Estadisticas','Estadisticas','current'])
```

D.31. EstadisticaTest

```
##                  HEVAH
## Proyecto de fin de carrera de Ingenieria Informática
## Universidad de La Laguna
## Tutora: Carina Gonzalez Gonzalez
## Autores: Fco. Javier Medina Santos
##     Daniel J. Díaz Gonzalez
##     Zebenzui Pérez Ramos
## Descripcion: Se encarga de presentar la primera tabla de estadisticas.
##              Distingue entre si es un alumno el que quiere ver la
##              estadística de uno de sus test o si es un profesor o
##              administrador el que quiere ver la estadistica de algun
##              alumno. A continuación, teniendo los datos del test, se
##              muestra una primera tabla con los aciertos y fallos según
##              la dificultad tanto de los conceptos como de las preguntas.

from xml.dom import minidom
from Cheetah.Template import Template
from index import index
import os
import string
import re
from Cifrado import Cifrado

class EstadisticaTest (index):
    def writeContent (self):
        req = self.request()
        resp = self.response()
        if (req.hasField('test') ):
            id = req.field('test')
         #Si es un profesor o un administrador el que quiere ver las
         # Estadisticas de un alumno seleccionado
            if (req.hasCookie('alumno')):
                alumno = req.cookie('alumno')
```

```
                cad_descifrada = Cifrado.Descifrar(
                  fich = "hevah//%s.xml"
% req.cookie('alumno'))
                dom = minidom.parseString (cad_descifrada)
                resp.delCookie('alumno')
          #Si un alumno quiere ver sus estadisticas
             else:
                alumno = req.cookie('username')
             cad_descifrada = Cifrado.Descifrar(fich = "hevah//%s.xml"
                                  % req.cookie('username'))
             dom = minidom.parseString (cad_descifrada)
          #Seleccionamos el test para ver la estadistica
             tests = dom.getElementsByTagName ('TEST')
             for t in tests:
                if ((t.getAttribute('FECHA') + t.getAttribute('HORA'))
       == id):
                    test = t
             #Conseguimos los niveles de dificultad de los conceptos
             dif_conceptos = [e for e in test.childNodes
                  if e.nodeType == e.ELEMENT_NODE]
             dic = {}
             for dif in dif_conceptos:
                #Conseuimos los niveles de dificultad de las preguntas
                if (dif.nodeName != 'HISTORICO'):
                    dif_preg = [e for e in dif.childNodes
               if e.nodeType == e.ELEMENT_NODE]
                    lista_preg = []
                    for dif_p in dif_preg:
                        cadena = str("a: "+dif_p.getAttribute('ACIERTOS')+"
          /  f: "+dif_p.getAttribute('FALLOS'))
                        lista_preg.append(cadena)
                    dic[str(dif.nodeName)] = lista_preg
             t = Template (file=
               'hevah//templates/VerEstadisticaTest.tpl.html')
             t.dic = dic
```

```
            t.alumno = alumno
            t.orden = ['SIMPLE','INTERMEDIO','COMPLEJO','AVANZADO']
            t.id_tema = id
            self.mostrar (str(t))
        else:
            self.writeln("MAL ROLLO")
```

D.32. GestionUsuarios

```
##                  HEVAH
## Proyecto de fin de carrera de Ingenieria Informática
## Universidad de La Laguna
## Tutora: Carina Gonzalez Gonzalez
## Autores: Fco. Javier Medina Santos
##     Daniel J. Díaz Gonzalez
##     Zebenzui Pérez Ramos
## Descripcion: Se encarga de presentar los usuarios del sistema para
##              realizar las labores de gestión necesarias. Existen varias
##              restricciones:
##                  * Alumnos: no pueden acceder a esta zona
##                  * Profesores: solo pueden gestionar sus alumnos
##                  * Administradores: pueden gestionar todos los usuarios

from Cheetah.Template import Template
from xml.dom import minidom
from Plantilla import Plantilla
from Cifrado import Cifrado

class GestionUsuarios (Plantilla):
    def writeContent (self):
        req = self.request()
     #El usuario está validado
        if (self.request().hasCookie('grupo')):
         #Es un profesor o un administrador
            if (self.request().cookie ('grupo') != "ALUMNO"):
                plantilla = Template
          (file="hevah//templates/GestionUsuarios.tpl.html")
          #Si es un profesor mostramos sus alumnos
                if (req.cookie('grupo') == "PROFESORES"):
                    cad_descifrada = Cifrado.Descifrar(fich =
                "hevah//%s.xml" % req.cookie('username'))
                    dom = minidom.parseString (cad_descifrada)
```

```
                    alumnos = dom.getElementsByTagName ('ALUMNO')[0]
                    lista_usuarios = [e for e in alumnos.childNodes
                    if e.nodeType == e.ELEMENT_NODE]
          #Si es un administrador mostramos todos los usuarios
                else:
                    if (req.cookie('grupo') == "ADMINISTRADOR"):
                        cad_descifrada = Cifrado.Descifrar(fich =
                 "hevah//indice.xml")
                        dom = minidom.parseString (cad_descifrada)
                        raiz = dom.getElementsByTagName ('PERFILES')[0]
                        lista_usuarios = [e for e in raiz.childNodes
                     if e.nodeType == e.ELEMENT_NODE]

                plantilla.lista_usuarios = lista_usuarios
                self.mostrar (plantilla)
         #Si no es profesor o administrador, se le expulsa
#por acceso no autorizado
            else:
                self.LimpiarCookies()
                self.response().sendRedirect ('index')
     # Al no estar validado, se le manda a la página de Bienvenida
        else:
            self.LimpiarCookies()
            self.response().sendRedirect ('index')

    def setTabs (self):
        self.tabs = []
        self.tabs.append (['Salir','Logout',''])
        self.tabs.append(['Bienvienida', 'index',''])
        if (self.request().cookie('grupo') != "ALUMNOS"):
            self.tabs.append (['Edición de Tests','TestDisponibles',''])
            self.tabs.append (['Gestión de Usuarios','GestionUsuarios',
                   'current'])
        self.tabs.append(['Comenzar Tests','SeleccionTest',''])
        self.tabs.append(['Estadisticas','Estadisticas',''])
```

D.33. index

```
##                  HEVAH
## Proyecto de fin de carrera de Ingenieria Informática
## Universidad de La Laguna
## Tutora: Carina Gonzalez Gonzalez
## Autores: Fco. Javier Medina Santos
##     Daniel J. Díaz Gonzalez
##     Zebenzui Pérez Ramos
## Descripcion: Se encarga de la presentación de la pagina principal del
##              portal.

from Cheetah.Template import Template
from Plantilla import Plantilla

class index (Plantilla):
    # Página inicial de la aplicación
    def writeContent (self):
        """
            Escribe el cuerpo de la página
        """
        t = Template (file = 'hevah/templates/Login.tpl.html')
        t.validado = 0
        if (self.request().hasCookie ('grupo')):    #Si está validado
            t.validado = 1
        self.writeln ((t))

    def setTabs (self):
        """
            Establece los tabs que se verán en la barra superior
        """
        self.tabs = []
        if (self.request().hasCookie ('grupo')):    # Usuarios Validados
            self.tabs.append (['Salir','Logout',''])
#Estamos en esta página
```

```
            self.tabs.append(['Bienvenida','index','current'])
            if (self.request().cookie('grupo') != "ALUMNOS"):
                self.tabs.append (['Edición de Tests','TestDisponibles',
                    ''])
                self.tabs.append (['Gestión de Usuarios',
                    'GestionUsuarios',''])
            self.tabs.append(['Comenzar Tests','SeleccionTest',''])
            self.tabs.append(['Estadisticas','Estadisticas',''])

    def writeRight (self):
        if (not self.request().hasCookie ('username')):
            t = Template (file = 'hevah/templates/Login.tpl.html')
            self.mostrar (str(t))

    def writeFooter (self):
        self.writeln ('''
                        <font color="#73298c">
                        <hr/>
                        <table width="100%">
                            <tr align="center">
                                <td ><h2>UNIVERSIDAD DE LA LAGUNA</h2></td>
                            </tr>
                        </table>
                        </font>
                    ''')
```

D.34. Login

```
##                  HEVAH
## Proyecto de fin de carrera de Ingenieria Informática
## Universidad de La Laguna
## Tutora: Carina Gonzalez Gonzalez
## Autores: Fco. Javier Medina Santos
##     Daniel J. Díaz Gonzalez
##     Zebenzui Pérez Ramos
## Descripcion: Se encarga de la validación de los usuarios. Para ello
##              verifica que: se haya itroducido login y password, el
##              usuario este registrado en el sistema y los datos
##              introducidos coincidan con los almacenados. Si todo es
##              correcto se activa informacion necesaria para la sesión
##              del usuario.

from Plantilla import Plantilla
from Cheetah.Template import Template
from xml.dom import minidom
import string
from Cifrado import Cifrado

class Login (Plantilla):
    def writeContent (self):
        req = self.request()
        resp = self.response()
     #Si no se introducen el login o el password no se le deja entrar
        if (not req.hasField("login")) or (not req.hasField("password")):
            self.LimpiarCookies()
            resp.sendRedirect ("index")
        else:
            login = string.upper(req.field ("login"))
            password = req.field ("password")

         #Buscamos el usuario
```

```
        cad_descifrada = Cifrado.Descifrar(fich = "hevah//indice.xml")
        dom = minidom.parseString (cad_descifrada)
        usuario = dom.getElementsByTagName (login)
    # Si no esta registrado
        if len (usuario) == 0:
            self.LimpiarCookies()
            resp.sendRedirect ("index")
        else:
            user = usuario[0]
            passwd = user.getAttribute ('PASSWD')
     #Si el password no es correcto
            if (passwd != password):
                self.LimpiarCookies()
                resp.sendRedirect ("index")
            else:
          #Se activa informacion necesaria para su sesion
                resp.setCookie ('username',login)
                grupo = user.getAttribute('GRUPO')
                resp.setCookie ('grupo', grupo)
                if (grupo == "ALUMNOS"):
                    resp.setCookie ('profesor',
            user.getAttribute('PROFESOR'))
                resp.sendRedirect ("index")
```

D.35. Logout

```
##                  HEVAH
## Proyecto de fin de carrera de Ingenieria Informática
## Universidad de La Laguna
## Tutora: Carina Gonzalez Gonzalez
## Autores: Fco. Javier Medina Santos
##     Daniel J. Díaz Gonzalez
##     Zebenzui Pérez Ramos
## Descripcion: Se encarga de la salida de los usuarios del sistema
##              elimiando las variables asociadas a ese usuario.

from Plantilla import Plantilla

class Logout(Plantilla):
    def writeContent (self):
        self.tabs = []
        self.LimpiarCookies()
        self.response().sendRedirect('index')
```

D.36. ModificarDatos

```
##                  HEVAH
## Proyecto de fin de carrera de Ingenieria Informática
## Universidad de La Laguna
## Tutora: Carina Gonzalez Gonzalez
## Autores: Fco. Javier Medina Santos
##     Daniel J. Díaz Gonzalez
##     Zebenzui Pérez Ramos
## Descripcion: Se encarga de buscar y presentar los datos del usuario
##              para que estos puedan ser modificados. Verifica que el
##              usuario que intenta acceder a los datos sea un usuario
##              registrado.

from Cheetah.Template import Template
from xml.dom import minidom
from index import index
import string
from Cifrado import Cifrado

class ModificarDatos (index):
    def writeContent (self):
       #El usuario está validado
        if (self.request().hasCookie('grupo') and
     (self.request().hasField('modificar'))):
         #Es un profesor o un administrador
            if (self.request().cookie ('grupo') != "ALUMNO"):
            username = self.request().field('modificar')
                plantilla = Template (file=
             "hevah//templates/ModificarDatos.tpl.html")
                cad_descifrada = Cifrado.Descifrar(fich =
                  "hevah//%s.xml" % username)
                dom = minidom.parseString (cad_descifrada)
          #Buscamos el usuario para coger sus datos
                cad_descifrada = Cifrado.Descifrar(fich =
```

```
                                    "hevah//indice.xml")
                usuarios = minidom.parseString (cad_descifrada)
                alumnos = usuarios.getElementsByTagName ('PERFILES')[0]
                lista_usuarios = [e for e in alumnos.childNodes
                   if e.nodeType == e.ELEMENT_NODE]
          for user in lista_usuarios:
              if (user.getAttribute('USERNAME') == username):
                 nodo_user = user
               break

                plantilla.user = nodo_user
                plantilla.u = dom.getElementsByTagName ('DATOS')[0]
                plantilla.username = self.request().field('modificar')
                self.mostrar (plantilla)
         #Si no es profesor o administrador, se le expulsa por acceso
#no autorizado
            else:
                self.LimpiarCookies()
                self.response().sendRedirect ('index')
     # Al no estar validado, se le manda a la página de Bienvenida
        else:
            self.LimpiarCookies()
            self.response().sendRedirect ('index')
```

D.37. MostrarDatosUsuario

```
##                  HEVAH
## Proyecto de fin de carrera de Ingenieria Informática
## Universidad de La Laguna
## Tutora: Carina Gonzalez Gonzalez
## Autores: Fco. Javier Medina Santos
##     Daniel J. Díaz Gonzalez
##     Zebenzui Pérez Ramos
## Descripcion: Se encarga de buscar y presentar los datos del usuario.
##              Verifica que el usuario que intenta acceder a los datos sea
##              un usuario registrado.

from Cheetah.Template import Template
from xml.dom import minidom
from index import index
import string
from Cifrado import Cifrado

class MostrarDatosUsuario (index):
    def writeContent (self):
        #El usuario está validado
        if (self.request().hasCookie('grupo')):
         #Es un profesor o un administrador
            if (self.request().cookie ('grupo') != "ALUMNO"):
            username = self.request().field('usuario')
                plantilla = Template (file=
            "hevah//templates/MostrarDatosUsuario.tpl.html")
                #Vamos a perfil del usuario para coger sus datos personales
          cad_descifrada = Cifrado.Descifrar(fich = "hevah//%s.xml"
                                    % username)
                dom = minidom.parseString (cad_descifrada)
          #Buscamos los datos de usuario
                cad_descifrada = Cifrado.Descifrar(fich =
                                    "hevah//indice.xml")
```

```
                usuarios = minidom.parseString (cad_descifrada)
                alumnos = usuarios.getElementsByTagName ('PERFILES')[0]
                lista_usuarios = [e for e in alumnos.childNodes
                   if e.nodeType == e.ELEMENT_NODE]
          for user in lista_usuarios:
              if (user.getAttribute('USERNAME') == username):
                 nodo_user = user
               break

          plantilla.user = nodo_user
                plantilla.u = dom.getElementsByTagName ('DATOS')[0]
                plantilla.username = self.request().field('usuario')
                self.mostrar (plantilla)
         #Si no es profesor o administrador, se le expulsa por acceso
#no autorizado
            else:
                self.LimpiarCookies()
                self.response().sendRedirect ('index')
     # Al no estar validado, se le manda a la página de Bienvenida
        else:
            self.LimpiarCookies()
            self.response().sendRedirect ('index')
```

D.38. NuevoTest

```
##                  HEVAH
## Proyecto de fin de carrera de Ingenieria Informática
## Universidad de La Laguna
## Tutora: Carina Gonzalez Gonzalez
## Autores: Fco. Javier Medina Santos
##     Daniel J. Díaz Gonzalez
##     Zebenzui Pérez Ramos
## Descripcion: Se encarga de mostrar el formulario para la creacion de un
##              nuevo test.

from TestDisponibles import TestDisponibles
from xml.dom import minidom
from Cheetah.Template import Template

class NuevoTest (TestDisponibles):
    def writeContent (self):
        t = Template (file = 'hevah//templates/nuevotest.tpl.html')
        self.mostrar (str(t))
```

D.39. NuevoUsuario

```
##                  HEVAH
## Proyecto de fin de carrera de Ingenieria Informática
## Universidad de La Laguna
## Tutora: Carina Gonzalez Gonzalez
## Autores: Fco. Javier Medina Santos
##     Daniel J. Díaz Gonzalez
##     Zebenzui Pérez Ramos
## Descripcion: Se encarga de mostrar el formulario donde se especificarán
##              los datos del nuevo usuario. Controla que el que intenta
##              realizar esta operación tiene permiso para ello.

from Cheetah.Template import Template
from xml.dom import minidom
from index import index
import string

class NuevoUsuario (index):
    def writeContent (self):
       #El usuario está validado
        if (self.request().hasCookie('grupo')
      and (self.request().hasField('anyadir'))):
         #Es un profesor o un administrador
            if (self.request().cookie ('grupo') != "ALUMNO"):
                plantilla = Template (file=
             "hevah//templates/NuevoUsuario.tpl.html")
                plantilla.g = self.request().cookie('grupo')
                self.mostrar (plantilla)
         #Si no es profesor o administrador, se le expulsa por acceso
#no autorizado
            else:
                self.LimpiarCookies()
                self.response().sendRedirect ('index')
     # Al no estar validado, se le manda a la página de Bienvenida
```

```
        else:
            self.LimpiarCookies()
            self.response().sendRedirect ('index')
```

D.40. Plantilla

```
##                  HEVAH
## Proyecto de fin de carrera de Ingenieria Informática
## Universidad de La Laguna
## Tutora: Carina Gonzalez Gonzalez
## Autores: Fco. Javier Medina Santos
##     Daniel J. Díaz Gonzalez
##     Zebenzui Pérez Ramos
## Descripcion: Se encarga del formato general que tendrá el web, indicando
##              donde y como se van a presentar cada uno de los elementos
##              que se presentarán en el mismo.

from WebKit.Page import Page

#color de la universidad #73298c
class Plantilla(Page):
    """
        Probando
    """
    ## Init ##
    def __init__ (self):
        Page.__init__(self)
        self.tab = ' '
        self.test = None
        self.tabs = []
        self.paso_actual = 0
        self.asistente = []
    ## Content methods ##

    def mostrar (self, cadena):
        cadena = str(cadena).replace ('#A','Á')
        cadena = str(cadena).replace ('#E','É')
        cadena = str(cadena).replace ('#I','Í')
        cadena = str(cadena).replace ('#O','Ó')
```

```
        cadena = str(cadena).replace ('#U','Ú')
        cadena = str(cadena).replace ('#a','á')
        cadena = str(cadena).replace ('#e','é')
        cadena = str(cadena).replace ('#i','í')
        cadena = str(cadena).replace ('#o','ó')
        cadena = str(cadena).replace ('#u','ú')
        cadena = str(cadena).replace ('#N','Ñ')
        cadena = str(cadena).replace ('#n','ñ')
        cadena = str(cadena).replace ('#?','>')
        cadena = str(cadena).replace ('#!','<')
        self.writeln (str(cadena))


    def codificar (self, cadena):
        cadena = str(cadena).replace ('Á','#A')
        cadena = str(cadena).replace ('É','#E')
        cadena = str(cadena).replace ('Í','#I')
        cadena = str(cadena).replace ('Ó','#O')
        cadena = str(cadena).replace ('Ú','#U')
        cadena = str(cadena).replace ('á','#a')
        cadena = str(cadena).replace ('é','#e')
        cadena = str(cadena).replace ('í','#i')
        cadena = str(cadena).replace ('ó','#o')
        cadena = str(cadena).replace ('ú','#u')
        cadena = str(cadena).replace ('Ñ','#N')
        cadena = str(cadena).replace ('ñ','#n')
        cadena = str(cadena).replace ('>','#?')
        cadena = str(cadena).replace ('<','#!')
        return (cadena)

    def htBodyArgs(self):
        """
        Returns the arguments used for the HTML <body> tag. Invoked by
        writeBody().
```

```
        With the prevalence of stylesheets (CSS), you can probably skip
        this particular HTML feature.
        """
        return 'color=white bgcolor=white vlink="yellow" alink="cyan"
         link="black"'

    def writeBodyParts(self):
        self.setTabs()
        self.writeHeader()
        self.writeln('<div id="contenido">')
        self.writeln ('<table width="100%" border="0"><tr>')
        self.writeAsistente()
        self.writeln ('<td width="75%%">')
        self.writeContent()
        self.writeln ('</td></tr></table>')
        self.writeln('</div>')
        self.writeln ('<div id="pie">')
        self.writeFooter ()
        self.writeln ('</div>')

    def addPaso (self, cadena):
        try:
            self.asistente.index(cadena)
        except:
            self.asistente.append (cadena)

    def delPaso (self, cadena):
        try:
            self.asistente.remove(cadena)
        except:
            pass

    def setActual (self, paso):
        try:
            self.paso_actual = paso
```

```
        except:
            pass

    def setAsistente (self):
        self.asistente = []

    def writeAsistente(self):
        self.setAsistente()
        if (len(self.asistente) > 0):
            self.writeln ('<td width="20%%">')
            self.writeln ('<table border="0">')
            cont = 0
            for paso in self.asistente:
                if (cont == self.paso_actual):
                    self.writeln('<tr><td align="left">
  <font size="3" color="white">
  <img src="images/boton_verde.gif"/>
  Paso %d. %s </font></td></tr>'% (cont, paso))
                else:
                    self.writeln ('<tr><td align="left">
  <font size="2"><img src="images/boton_verde_off.png"/>
  Paso %d. %s</font></td></tr>'% (cont, paso))
                cont += 1
            self.writeln ('</table>')
            self.writeln ('</td>')

    def writeTabs (self):
        if len (self.tabs) > 0:
            cadena = '<ul>'
            for tab in self.tabs:
                cadena = cadena + '<li id="%s"><a href="%s">%s</a></li>'
 % (tab[2],tab[1], tab[0])
            cadena = cadena + '</ul>'
            self.mostrar (cadena)
```

```
def setTabs (self):
    self.tabs = []

def writeHeader (self):
    self.writeln (’’’
                <div id="header">
                    <h1><font color="white">Proyecto HEVAH</font></h1>
                ’’’)
    self.writeTabs()
    self.writeln (’</div>’)

def writeFooter (self):
    pass

def LimpiarCookies (self):
    cookies = self.request().cookies()
    for cookie in cookies.keys():
        self.response().delCookie (cookie)

def writeStyleSheet(self):
    self.writeln(’\t<link rel=stylesheet href=estilos.css type=text/css>’)
```

D.41. Procesado

```
##                  HEVAH
## Proyecto de fin de carrera de Ingenieria Informática
## Universidad de La Laguna
## Tutora: Carina Gonzalez Gonzalez
## Autores: Fco. Javier Medina Santos
##     Daniel J. Díaz Gonzalez
##     Zebenzui Pérez Ramos
## Descripcion: Se encarga de gestionar la ejecucion del test. Llama a las
##              funciones pertinentes para evaluar la respuesta del usuario
##              solicitar nuevas preguntas o nuevos conceptos y actualizar
##              las variables necesarias. También se encarga de determinar
##              cuando ha finalizado el test.

import procxml
import usuario
from Evaluar import Evaluar
import clipsxml
import sys

class Test:
finpreguntas = 0

def __init__ (self, ficheroxml):
f = open ('login.pro','r')
id_user = f.read()
f.close
lista_param = id_user.split('|')
#Instanciamos un nuevo usuario
self.user = usuario.Usuario (lista_param)
self.xml = procxml.MiXML (ficheroxml)   # Cargamos el test
self.p = clipsxml.Coprocess("CLIPSDOS.exe")

#Inicialimos probabilidades y la clase de evaluación
```

```
prob_ini = [0.4,0.3,0.1,0.01,0.5,0.4,0.2,0.1,
            0.6,0.5,0.3,0.2,0.7,0.6,0.5,0.3]
self.calculos = Evaluar(prob_ini, self.p, "reglas.clips")

#Conseguimos el primer tema y el primer concepto
self.tema = self.xml.GetTema()[2]
self.concepto = self.tema.GetConcepto (
                self.user.GetConceptoInicial())[2]

# Le indicamos al evaluador que iniciamos un nuevo concepto
self.calculos.NuevoConcepto (self.user, self.concepto.GetNivel())
self.nivel_preg = 'MEDIA'
self.ProximaPregunta()

# Se encarga de buscar la siguiente pregunta a evaluar.
# Si no hay mas preguntas de un concepto se busca otro concepto del
# mismo nivel. Este proceso se repite tres veces. Si aún asi no se ha
# encontrado ninguna pregunta se entiende que estas se han agotado.
def ProximaPregunta (self):
self.pregunta = self.concepto.GetPregunta (self.nivel_preg)[2]
intento = 0
#Si no hay mas preguntas del concepto actual
while (self.pregunta == None) and (intento < 3):
#buscamos un nuevo concepto
aux = self.concepto.GetNivel()
self.concepto = self.tema.GetConcepto (aux)[2]
#Si no hay mas conceptos terminamos
if (self.concepto == None):
intento == 3
break
#si los hay, buscamos una nueva pregunta
self.calculos.NuevoConcepto(self.user, self.concepto.GetNivel())
self.pregunta = self.concepto.GetPregunta(self.nivel_preg)[2]
intento += 1
#No hay mas preguntas
```

```
if intento == 3:
self.finpreguntas = 1

# Devuelve:
#     * un el enunciado del concepto actual
def getConcepto (self):
return (self.concepto.GetEnunciado().encode("latin1"))

#Devuelve:
#     * una lista con el enunciado de la pregunta y las
#       posibles respuestas para la misma
def getPregunta (self):
if (self.FinTest()):
return (None)
lista = {}
lista["PREG"] = self.pregunta.GetEnunciado().encode("latin1")
for i in "ABC":
lista[i] = self.pregunta.GetRespuesta(i)
return (lista)

# Evalua la respuesta del usuario. Para ello recalcula las
# probabilidades y actuliza la tabla de probabilidades. Según
# lo que devuelvan las reglas cambia el concepto o actualiza el
# nivel de la pregunta. Si el test no ha finalizado solicita la
# siguiente pregunta.
# Parametros:
#     * respuesta: respuesta del usuario
def EvaluarRespuesta (self, respuesta):
siguiente = self.calculos.NuevaPreg (self.p,respuesta, self.pregunta)
lista = siguiente.split ('_')

#Actualizamos Tabla de resultados
self.user.ActualizarTabla (self.concepto, self.pregunta,
                           self.calculos.GetTestUltima())
```

```
#Actuamos según nos digan las reglas
tipo = lista[0]
nivel = str(string.upper(lista[1].strip()))
#Si hay un cambio de concepto
if (tipo == "concepto"):
#Se busca un nuevo concepto
self.concepto = self.tema.GetConcepto(nivel)[2]
if (self.concepto == None):
self.finpreguntas = 1
return
self.calculos.NuevoConcepto(self.user, self.concepto.GetNivel())
else:
#Se actualiza el nivel de la pregunta
self.nivel_preg = nivel
#Si no se ha terminado se busca la siguiente pregunta
if (not (self.FinTest())):
self.ProximaPregunta()

# Indica si el test ha finalizado
# Devuelve:
#     * 0: no ha terminado, 1: ha terminado
def FinTest (self):
#Si se terminan las preguntas se evalua el test y se guardan
#los datos
if (self.finpreguntas) or (self.pregunta == None):
self.user.Evaluar()
self.user.CrearPerfil ()
return (1)
return (0)
```

D.42. SeleccionTest

```
##                  HEVAH
## Proyecto de fin de carrera de Ingenieria Informática
## Universidad de La Laguna
## Tutora: Carina Gonzalez Gonzalez
## Autores: Fco. Javier Medina Santos
##     Daniel J. Díaz Gonzalez
##     Zebenzui Pérez Ramos
## Descripcion: Se encarga de mostrar los test para su ejecucion.
##              Si el usuario que va a realizar un test se le muestran los
##              de su profesor asociado. Si es un profesor se le muestran
##              los test que ha creado. Y si es un administrador se le
##              muestran todos.

from xml.dom import minidom
from Cheetah.Template import Template
from Plantilla import Plantilla
from Test import Test
import os
import string
import re
import threading
from Cifrado import Cifrado

class SeleccionTest (Plantilla):
    def writeContent (self):
        req = self.request()
        resp = self.response()
        if (not req.hasField('seleccion')):
            test_disponibles = []
            #Si es un profesor el que quiere ejecutar el test se
#le muestran los suyos
            if (req.cookie('grupo') == 'PROFESORES'):
                cad_descifrada = Cifrado.Descifrar(fich =
```

```
                  "hevah//%s.xml" % req.cookie('username'))
                dom = minidom.parseString (cad_descifrada)
                test = dom.getElementsByTagName ('TEST')[0]
          #Seleccionamos la lista de test
                lista_test = [e for e in test.childNodes
               if e.nodeType == e.ELEMENT_NODE]
                for t in lista_test:
                    test_disponibles.append(
                  ['hevah//tests/'+
t.getAttribute('FICHERO'),
t.getAttribute('TEXTO'),
t.getAttribute('DESC')])
            else:
                #Si es un alumno el que ejecuta el test se le
 #muestran los de su profesor asociado
                if (req.cookie('grupo') == 'ALUMNOS'):
                    cad_descifrada = Cifrado.Descifrar(fich =
  "hevah//%s.xml" % req.cookie('profesor'))
                    dom = minidom.parseString (cad_descifrada)
                    test = dom.getElementsByTagName ('TEST')[0]
              #Seleccionamos la lista de test
                    lista_test = [e for e in test.childNodes
                if e.nodeType == e.ELEMENT_NODE]
                    for t in lista_test:
                        test_disponibles.append(
                ['hevah//tests/'
 +t.getAttribute('FICHERO'),
 t.getAttribute('TEXTO'),
 t.getAttribute('DESC')])
                else:
                    #Si un administrador ejecuta un test
                    if (req.cookie('grupo') == 'ADMINISTRADOR'):
                        lista = os.listdir ('hevah//tests')
                        for file in lista:
                   #filtrado de ficheros;me quedo solo con los xml
```

```
                            if re.match('..*\.xml$', file):
                           cadena = Cifrado.Descifrar(fich =
                    'hevah//tests/'+file)
                                dom = minidom.parseString(cadena)
                                nodo = dom.getElementsByTagName ("TEST")[0]
                                test_disponibles.append (['hevah//tests/'+
                    string.split(file,".")[0],
nodo.getAttribute('TEXTO'),
nodo.getAttribute('DESC')])
            t = Template (file='hevah//templates/SeleccionTest.tpl.html')
            t.test_disponibles = test_disponibles
            self.mostrar (str(t))
        else:
            resp.setCookie('fich_test', '%s.xml' % req.field('radio_test'))
            resp.sendRedirect('ComienzaTest')

    def setTabs (self):
        self.tabs = []
        self.tabs.append (['Salir','Logout',''])
        self.tabs.append(['Bienvienida', 'index',''])
        if (self.request().cookie('grupo') != "ALUMNOS"):
            self.tabs.append (['Edición de Tests','TestDisponibles',''])
            self.tabs.append (['Gestión de Usuarios','GestionUsuarios',''])
        self.tabs.append(['Comenzar Tests','SeleccionTest','current'])
        self.tabs.append(['Estadisticas','Estadisticas',''])
```

D.43. TestDisponiblesOK

```
##                  HEVAH
## Proyecto de fin de carrera de Ingenieria Informática
## Universidad de La Laguna
## Tutora: Carina Gonzalez Gonzalez
## Autores: Fco. Javier Medina Santos
##     Daniel J. Díaz Gonzalez
##     Zebenzui Pérez Ramos
## Descripcion: Realiza las operaciones de gestión oportunas en funcion de
##              la seleccion del usuario:
##                   * Ver: se pasa a la gestión de temas
##                   * Borrar: se borra el test
##                   * Editar: se edita el test para modificaciones
##                   * Nuevo: se crea un nuevo test
##                   * Atrás: se retorna a la página anterior

from TestDisponibles import TestDisponibles

class TestDisponiblesOK (TestDisponibles):
    def writeContent (self):
        req = self.request()
        resp = self.response()
     #Pasamos a ver los temas del test
        if (req.hasField ('ver')):
            resp.setCookie ('fichero', req.field('radio_test'))
            resp.sendRedirect ('VerTest')
     #Pasamos a borrar el test
        if (req.hasField ('borrar')):
            if (req.hasField('radio_test')):
                resp.setCookie ('fichero', req.field('radio_test'))
                resp.sendRedirect ("BorrarTest")
     #Pasamos a editar el test
        if (req.hasField ('editar')):
            if (req.hasField ('radio_test')):
```

```
            resp.sendRedirect ('EditarTest')
        else:
            resp.sendRedirect ('TestDisponibles')
#Pasamos a crear un nuevo test
   if (req.hasField ('nuevo')):
        resp.sendRedirect ('NuevoTest')
#Volvemos a la página anterior
   if (req.hasField('atras')):
        resp.sendRedirect ('index')
```

D.44. TestDisponibles

```
##                  HEVAH
## Proyecto de fin de carrera de Ingenieria Informática
## Universidad de La Laguna
## Tutora: Carina Gonzalez Gonzalez
## Autores: Fco. Javier Medina Santos
##     Daniel J. Díaz Gonzalez
##     Zebenzui Pérez Ramos
## Descripcion: Se encarga de mostrar los test disponibles. Los alumnos
##              no tienen permiso para acceder a esta sección. A los
##              profesores se le muestran los test que ha creado. A los
##              administradores se le muestran todos.

from Cheetah.Template import Template
from Plantilla import Plantilla
import os
import re
from xml.dom import minidom
from convertir import Convertir
import string
from Cifrado import Cifrado

class TestDisponibles (Plantilla):
    def writeContent (self):
        req = self.request()
        if (req.hasCookie ('grupo')):
            t = Template (file =
              'hevah//templates/testdisponibles.tpl.html')
            lista = os.listdir ('hevah//tests')
            lista_tests = []    #Test que se muestran
            test_profesor = []     #Test del profesor
            #Si es un profesor, miramos los test que ha creado
            if (req.cookie('grupo') == 'PROFESORES'):
                cad_descifrada = Cifrado.Descifrar(fich =
```

```
                'hevah//%s.xml' % req.cookie('username'))
              profesor = minidom.parseString (cad_descifrada)
              test = profesor.getElementsByTagName ('TEST')[0]
              for _test in test.childNodes:
                  test_profesor.append("%s.xml" %
           string.lower(_test.getAttribute('FICHERO')))

              for file in lista:
            #filtrado de ficheros; me quedo solo con los xml
                  if re.match('..*\.xml$', file):
                #Si el test es del profesor
                      if (string.lower(file) in test_profesor):
                          cadena = Cifrado.Descifrar(fich =
                        'hevah//tests/'+file)
                          dom = minidom.parseString(cadena)
                          nodo = dom.getElementsByTagName ("TEST")[0]
                          lista_tests.append (['hevah//tests/'+file,
                     nodo.getAttribute('TEXTO'),
nodo.getAttribute('DESC')])
            else:
            #Si es un administrador se le muestran todos los test
                if (req.cookie('grupo') == 'ADMINISTRADOR'):
                    for file in lista:
                  #filtrado de ficheros; me quedo solo con los xml
                        if re.match('..*\.xml$', file):
                            cadena = Cifrado.Descifrar(fich =
                           'hevah//tests/'+file)
                            dom = minidom.parseString(cadena)
                            nodo = dom.getElementsByTagName ("TEST")[0]
                            lista_tests.append (['hevah//tests/'+file,
                     nodo.getAttribute('TEXTO'),
nodo.getAttribute('DESC')])
            t.lista_tests = lista_tests
            self.mostrar (str(t))
        else:
```

```
            self.response().sendRedirect ('Logout')

    def setAsistente(self):
        self.addPaso ('Tests')
        self.addPaso ('Temas')
        self.addPaso ('Conceptos')
        self.addPaso ('Preguntas')
        self.addPaso ('Respuestas')

    def setTabs (self):
        self.tabs = []
        self.tabs.append (['Salir','Logout',''])
        self.tabs.append(['Bienvienida', 'index',''])
        if (self.request().cookie('grupo') != "ALUMNOS"):
            self.tabs.append (['Edición de Tests','TestDisponibles',
                   'current'])
            self.tabs.append (['Gestión de Usuarios','GestionUsuarios',''])
        self.tabs.append(['Comenzar Tests','SeleccionTest',''])
        self.tabs.append(['Estadisticas','Estadisticas',''])
```

D.45. Test

```
##                  HEVAH
## Proyecto de fin de carrera de Ingenieria Informática
## Universidad de La Laguna
## Tutora: Carina Gonzalez Gonzalez
## Autores: Fco. Javier Medina Santos
##     Daniel J. Díaz Gonzalez
##     Zebenzui Pérez Ramos
## Descripcion: Se encarga de gestionar la ejecucion del test. Llama a las
##              funciones pertinentes para evaluar la respuesta del usuario
##              solicitar nuevas preguntas o nuevos conceptos y actualizar
##              las variables necesarias. También se encarga de determinar
##              cuando ha finalizado el test.

#!/usr/bin/python
import procxml
import usuario
from Evaluar import Evaluar
import clipsxml
import sys
import string

class Test:
    finpreguntas = 0

    def __init__ (self, ficheroxml, username):
     #Instanciamos un nuevo usuario
        self.user = usuario.Usuario (username)
        self.xml = procxml.MiXML (ficheroxml)   # Cargamos el test
        self.ficheroxml = ficheroxml

        #Inicialimos probabilidades y la clase de evaluación
        prob_ini = [0.4,0.3,0.1,0.01,0.5,0.4,0.2,0.1,
              0.6,0.5,0.3,0.2,0.7,0.6,0.5,0.3]
```

```
        self.calculos = Evaluar(prob_ini, "reglas.clips")

        #Conseguimos el primer tema y el primer concepto
        self.tema = self.xml.GetTema()[2]
        self.concepto = self.tema.GetConcepto (
                  self.user.GetConceptoInicial())[2]

        # Le indicamos al evaluador que iniciamos un nuevo concepto
        self.calculos.NuevoConcepto (self.user, self.concepto.GetNivel())
        self.nivel_preg = 'MEDIA'
        self.ProximaPregunta()

    # Se encarga de buscar la siguiente pregunta a evaluar.
    # Si no hay mas preguntas de un concepto se busca otro concepto del
    # mismo nivel. Este proceso se repite tres veces. Si aún asi no se ha
    # encontrado ninguna pregunta se entiende que estas se han agotado.
    def ProximaPregunta (self):
        self.pregunta = self.concepto.GetPregunta (self.nivel_preg)[2]
        intento = 0
     #Si no hay mas preguntas del concepto actual
        while (self.pregunta == None) and (intento < 3):
         #buscamos un nuevo concepto
            aux = self.concepto.GetNivel()
            self.concepto = self.tema.GetConcepto (aux)[2]
         #Si no hay mas conceptos terminamos
            if (self.concepto == None):
                intento == 3
                break
         #si los hay, buscamos una nueva pregunta
            self.calculos.NuevoConcepto(self.user,self.concepto.GetNivel())
            self.pregunta = self.concepto.GetPregunta(self.nivel_preg)[2]
            intento += 1
     #No hay mas preguntas
        if intento == 3:
            self.finpreguntas = 1
```

```
# Devuelve:
#     * un el enunciado del concepto actual
def getConcepto (self):
    return (self.concepto.GetEnunciado().encode("latin1"))

#Devuelve:
#     * una lista con el enunciado de la pregunta y las
#       posibles respuestas para la misma
def getPregunta (self):
    if (self.FinTest()):
        return (None)
    lista = {}
    lista["PREG"] = self.pregunta.GetEnunciado().encode("latin1")
    for i in "ABC":
        lista[i] = self.pregunta.GetRespuesta(i)
    return (lista)

# Evalua la respuesta del usuario. Para ello recalcula las
# probabilidades y actuliza la tabla de probabilidades. Según
# lo que devuelvan las reglas cambia el concepto o actualiza el
# nivel de la pregunta. Si el test no ha finalizado solicita la
# siguiente pregunta.
# Parametros:
#     * respuesta: respuesta del usuario
def EvaluarRespuesta (self, respuesta):
    siguiente = self.calculos.NuevaPreg (respuesta, self.pregunta)
    lista = siguiente.split ('_')
    id_pregunta = '%s.%s.%s' %(self.tema.GetID(),self.concepto.GetID(),
            self.pregunta.GetID())

 #Almacenamos la pregunta en el historico
    self.user.ActualizarHistorico (id_pregunta,
                                   self.calculos.GetTestUltima())
```

```
        #Actualizamos Tabla de resultados
        self.user.ActualizarTabla (self.concepto, self.pregunta,
                             self.calculos.GetTestUltima())

        #Actuamos según nos digan las reglas
        tipo = lista[0]
        nivel = str(string.upper(lista[1].strip()))
     #Si hay un cambio de concepto
        if (tipo == "concepto"):
     #Se busca un nuevo concepto
            self.concepto = self.tema.GetConcepto(nivel)[2]
            if (self.concepto == None):
                self.finpreguntas = 1
                return
            self.calculos.NuevoConcepto(self.user,self.concepto.GetNivel())
        else:
         #Se actualiza el nivel de la pregunta
            self.nivel_preg = nivel
     #Si no se ha terminado se busca la siguiente pregunta
        if (not (self.FinTest())):
            self.ProximaPregunta()

    # Indica si el test ha finalizado
    # Devuelve:
    #     * 0: no ha terminado, 1: ha terminado
    def FinTest (self):
        if (self.finpreguntas) or (self.pregunta == None):
            self.user.Evaluar()
            self.user.CrearPerfil (self.ficheroxml)
            return (1)
        return (0)

if (__name__ == '__main__'):
        test = Test (string.lower('tests/cambio.xml'), 'kk')
```

```
resp = ''
while (resp != 'xx'):
    pregunta = test.getPregunta()
    print 'PREGUNTA:', pregunta
    if (pregunta == None):
        pregunta = {"PREG":"FIN DEL TEST.
     NO QUEDAN PREGUNTAS","A":"","B":"","C":""}
        nota = "%.3f" % test.user.GetNota()
        print ("Fin del test. No hay m#as preguntas.
 Tienes una nota de %s" % nota)
        test = None
    else:
        print  pregunta["PREG"]
        print 'A: ', pregunta["A"]
        print 'B: ', pregunta["B"]
        print 'C: ', pregunta["C"]
        resp = raw_input()
        test.EvaluarRespuesta(resp)
```

D.46. VerConceptoOK

```
##                  HEVAH
## Proyecto de fin de carrera de Ingenieria Informática
## Universidad de La Laguna
## Tutora: Carina Gonzalez Gonzalez
## Autores: Fco. Javier Medina Santos
##     Daniel J. Díaz Gonzalez
##     Zebenzui Pérez Ramos
## Descripcion: Realiza las operaciones de gestión oportunas en función de
##              la selección del usuario:
##                   * Ver: se pasa a la gestión de respuestas
##                   * Borrar: se borra la pregunta
##                   * Editar: se edita la pregunta para modificaciones
##                   * Nuevo: se crea una nueva pregunta
##                   * Atrás: se retorna a la página anterior

from TestDisponibles import TestDisponibles

class VerConceptoOK (TestDisponibles):
    def writeContent (self):
        req = self.request()
        resp = self.response()
#Pasamos a ver las respuestas de la pregunta
        if (req.hasField ('ver')):
            resp.setCookie ('pregunta',req.field ('ver'))
            resp.sendRedirect ('VerPregunta')
#Pasamos a crear una nueva pregunta
        if (req.hasField ('crear')):
            resp.sendRedirect ('CrearPregunta')
#Pasamos a borrar la pregunta
        if (req.hasField ('borrar')) and (req.hasField('radio_concepto')):
            resp.setCookie ('pregunta', req.field ('radio_concepto'))
            resp.sendRedirect ('BorrarPregunta')
#Pasamos a editar la pregunta
```

```
if (req.hasField ('editar')):
    if (req.hasField('radio_concepto')):
        resp.setCookie ('pregunta', req.field ('radio_concepto'))
        resp.sendRedirect('EditarPregunta')
    else:
        resp.sendRedirect('VerConcepto')
```

D.47. VerConcepto

```
##                  HEVAH
## Proyecto de fin de carrera de Ingenieria Informática
## Universidad de La Laguna
## Tutora: Carina Gonzalez Gonzalez
## Autores: Fco. Javier Medina Santos
##     Daniel J. Díaz Gonzalez
##     Zebenzui Pérez Ramos
## Descripcion: Se encarga de mostrar las preguntas disponibles para un
##              concepto dado. Tanto el concepto como el tema al que
##              pertence deben haberse seleccionado previamente.

from Cheetah.Template import Template
from TestDisponibles import TestDisponibles
from xml.dom import minidom
from Cifrado import Cifrado

class VerConcepto(TestDisponibles):
    def writeContent (self):
        req = self.request()
        resp = self.response()
        file = req.cookie ('fichero')
        tema = req.cookie ('tema')
        concepto = req.cookie('concepto')
        if (req.hasCookie('pregunta')):
            resp.delCookie('pregunta')
        cad_descifrada = Cifrado.Descifrar(fich = file)
        dom = minidom.parseString (cad_descifrada)
     # Buscamos el tema con el que estamos trabajando
        lista_temas = dom.getElementsByTagName ('TEMA')
        for _tema in lista_temas:
            if (_tema.getAttribute('ID') == tema):
                nodo_tema = _tema
                break
```

```
# Buscamos el concepto con el que estamos trabajando
   lista_conceptos = [e for e in nodo_tema.childNodes
                if e.nodeType == e.ELEMENT_NODE]
   for _concepto in lista_conceptos:
       if (_concepto.getAttribute('ID') == concepto):
           nodo_concepto = _concepto
           break
# Selecionamos la lista del preguntas relacionadas con el concepto
   lista_preguntas = [e for e in nodo_concepto.childNodes
                if e.nodeType == e.ELEMENT_NODE]

   t = Template (file='hevah//templates/VerConcepto.tpl.html')
   t.tema = nodo_tema.getAttribute ('TEXTO')
   t.concepto = nodo_concepto.getAttribute ('TEXTO')
   t.lista_preguntas = lista_preguntas
   self.mostrar (str(t))
```

D.48. VerHistórico

```
##                  HEVAH
## Proyecto de fin de carrera de Ingenieria Informática
## Universidad de La Laguna
## Tutora: Carina Gonzalez Gonzalez
## Autores: Fco. Javier Medina Santos
##     Daniel J. Díaz Gonzalez
##     Zebenzui Pérez Ramos
## Descripcion: Se encarga de mostrar el histórico de las preguntas que le
##              fueron realizadas al usuario a lo largo del test y si éstas
##              fueron respondidas correcatamente. Las tuplas del hitórico
##              tiene el siguiente formato:
##                  * X.Y.Z: X es el identificador del tema, Y es el
##                           identificador del concepto y Z es el
##                           identificador de la pregunta, todos valores
##                           numéricos
##                  * Acertada o Fallada

from xml.dom import minidom
from Cheetah.Template import Template
from index import index
import os
import string
import re
from Cifrado import Cifrado

class VerHistorico (index):
    def writeContent (self):
        req = self.request()
        resp = self.response()
        #Si un profesor el que quiere ver las Estadisticas de sus alumnos
        id_tema = req.field('id_tema')
        #Vamos al fichero de perfil del alumno
        cad_descifrada = Cifrado.Descifrar(fich = "hevah//%s.xml" %
```

```
                                    req.field('alumno'))
        dom = minidom.parseString (cad_descifrada)
        #Cogemos el test del que se ha echo el historico
        tests = dom.getElementsByTagName ('TEST')
        for t in tests:
            if ((t.getAttribute('FECHA') + t.getAttribute('HORA')) ==
    id_tema):
                test = t
        #Cogemos el historico
        hijos = [e for e in test.childNodes
           if e.nodeType == e.ELEMENT_NODE]
        for h in hijos:
            if (h.nodeName == "HISTORICO"):
                historico = h

        #Nos vamos al fichero donde esta el test para ver los temas,
#conceptos y preguntas
        cad_descifrada = Cifrado.Descifrar(fich = "hevah/%s" %
                                    test.getAttribute('FICHERO'))
        fich_test = minidom.parseString (cad_descifrada)
        desc_test = fich_test.getElementsByTagName ('TEST')[0]

        lista_datos = []
        lista_hist = [e for e in historico.childNodes
                if e.nodeType == e.ELEMENT_NODE]
        for tupla in lista_hist:
            id = string.split(tupla.getAttribute('ID'), ".")
            _tema = id[0]
            _concepto = id[1]
            _pregunta = id[2]
            datos = []
            # Buscamos el tema por su identificador
            temas = [e for e in desc_test.childNodes
         if e.nodeType == e.ELEMENT_NODE]
            for tema in temas:
```

```
            if (tema.getAttribute('ID') == _tema):
                nodo_tema = tema
                datos.append(tema.getAttribute('TEXTO'))
                break
        # Buscamos el concepto por su identificador
        conceptos = [e for e in nodo_tema.childNodes
         if e.nodeType == e.ELEMENT_NODE]
        for concepto in conceptos:
            if (concepto.getAttribute('ID') == _concepto):
                nodo_concepto = concepto
                datos.append(concepto.getAttribute('TEXTO'))
                break
        # Buscamos la pregunta por su identificador
        preguntas = [e for e in nodo_concepto.childNodes
         if e.nodeType == e.ELEMENT_NODE]
        for pregunta in preguntas:
            if (pregunta.getAttribute('IDPREGUNTA') == _pregunta):
                datos.append(pregunta.getAttribute('TEXTO'))
                break
        datos.append(tupla.getAttribute('VALOR'))
        lista_datos.append(datos)

    t = Template (file='hevah//templates/VerHistorico.tpl.html')
    t.lista_datos = lista_datos
    self.mostrar (str(t))
```

D.49. VerPreguntaOK

```
##                  HEVAH
## Proyecto de fin de carrera de Ingenieria Informática
## Universidad de La Laguna
## Tutora: Carina Gonzalez Gonzalez
## Autores: Fco. Javier Medina Santos
##     Daniel J. Díaz Gonzalez
##     Zebenzui Pérez Ramos
## Descripcion: Realiza las operaciones de gestión oportunas en función de
##              la selección del usuario:
##                   * Borrar: se borra la respuesta
##                   * Editar: se edita la respuesta para modificaciones
##                   * Nuevo: se crea una nueva respuesta

from TestDisponibles import TestDisponibles

class VerPreguntaOK (TestDisponibles):
    def writeContent (self):
        req = self.request()
        resp = self.response()
# Borramos la respuesta seleccionada
        if (req.hasField ('borrar')):
            if (req.hasField ('radio_pregunta')):
                resp.setCookie ('respuesta',req.field ('radio_pregunta'))
                resp.sendRedirect ('BorrarRespuesta')
            else:
                resp.sendRedirect ('VerPregunta')
# Creamos una nueva respuesta
        if (req.hasField ('crear')):
            resp.sendRedirect ('CrearRespuesta')
# Editamos la respuesta seleccionada
        if (req.hasField ('editar')):
            if (req.hasField ('radio_pregunta')):
                resp.setCookie ('respuesta',req.field ('radio_pregunta'))
```

```
            resp.sendRedirect ('EditarRespuesta')
        else:
            resp.sendRedirect ('VerPregunta')
```

D.50. VerPregunta

```
##                  HEVAH
## Proyecto de fin de carrera de Ingenieria Informática
## Universidad de La Laguna
## Tutora: Carina Gonzalez Gonzalez
## Autores: Fco. Javier Medina Santos
##     Daniel J. Díaz Gonzalez
##     Zebenzui Pérez Ramos
## Descripción: Se encarga de mostrar las respuestas disponibles para una
##              pregunta dada. Tanto la pregunta como el concepto al que
##              pertence deben haberse seleccionado previamente.

from Cheetah.Template import Template
from TestDisponibles import TestDisponibles
from xml.dom import minidom
from Cifrado import Cifrado

class VerPregunta(TestDisponibles):
    def writeContent (self):
        req = self.request()
        resp = self.response()
        file = req.cookie ('fichero')
        tema = req.cookie ('tema')
        concepto = req.cookie('concepto')
        pregunta = req.cookie('pregunta')
        cad_descifrada = Cifrado.Descifrar(fich = file)
        dom = minidom.parseString (cad_descifrada)
     # Buscamos el tema con el que estamos trabajando
        lista_temas = dom.getElementsByTagName ('TEMA')
        for _tema in lista_temas:
            if (_tema.getAttribute('ID') == tema):
                nodo_tema = _tema
                break
     # Buscamos el concepto con el que estamos trabajando
```

```
        lista_conceptos = [e for e in nodo_tema.childNodes
                    if e.nodeType == e.ELEMENT_NODE]
        for _concepto in lista_conceptos:
            if (_concepto.getAttribute('ID') == concepto):
                nodo_concepto = _concepto
                break
    # Buscamos la pregunta con la que estamos trabajando
        lista_preguntas = [e for e in nodo_concepto.childNodes
                    if e.nodeType == e.ELEMENT_NODE]
        for _pregunta in lista_preguntas:
            if (_pregunta.getAttribute('IDPREGUNTA') == pregunta):
                nodo_pregunta = _pregunta
                break
    # Cogemos la lista de respuestas
        lista_respuestas = [e for e in nodo_pregunta.childNodes
                     if e.nodeType == e.ELEMENT_NODE]

        t = Template (file='hevah//templates/VerPregunta.tpl.html')
        t.tema = nodo_tema.getAttribute ('TEXTO')
        t.concepto = nodo_concepto.getAttribute ('TEXTO')
        t.pregunta = nodo_pregunta.getAttribute ('TEXTO')
        t.correcta = nodo_pregunta.getAttribute ('RESULTADO')

        t.lista_respuestas = lista_respuestas
        self.mostrar (str(t))
```

D.51. VerTemaOK

```
##                  HEVAH
## Proyecto de fin de carrera de Ingenieria Informática
## Universidad de La Laguna
## Tutora: Carina Gonzalez Gonzalez
## Autores: Fco. Javier Medina Santos
##     Daniel J. Díaz Gonzalez
##     Zebenzui Pérez Ramos
## Descripcion: Realiza las operaciones de gestión oportunas en función de
##              la selección del usuario:
##                   * Ver: se pasa a la gestión de preguntas
##                   * Borrar: se borra el concepto
##                   * Editar: se edita el concepto para modificaciones
##                   * Nuevo: se crea un nuevo concepto

from TestDisponibles import TestDisponibles

class VerTemaOK (TestDisponibles):
    def writeContent (self):
        req = self.request()
        resp = self.response()
     # Pasamos a la gestion de preguntas
        if (req.hasField ('ver')):
            resp.setCookie ('concepto',req.field ('ver'))
            resp.sendRedirect ('VerConcepto')
     # Creamos un nuevo concepto
        if (req.hasField ('crear')):
            resp.sendRedirect ('CrearConcepto')
     # Borramos un concepto
        if (req.hasField ('borrar')):
            if (req.hasField ('radio_concepto')):
                resp.setCookie ('concepto',req.field ('radio_concepto'))
                resp.sendRedirect ('BorrarConcepto')
            else:
```

```
            resp.sendRedirect ('VerTema')
    # Editamos un concepto seleccionado
       if (req.hasField ('editar')):
           if (req.hasField ('radio_concepto')):
               resp.setCookie ('concepto',req.field ('radio_concepto'))
               resp.sendRedirect ('EditarConcepto')
           else:
               resp.sendRedirect ('VerTema')
```

D.52. VerTema

```
##                  HEVAH
## Proyecto de fin de carrera de Ingenieria Informática
## Universidad de La Laguna
## Tutora: Carina Gonzalez Gonzalez
## Autores: Fco. Javier Medina Santos
##     Daniel J. Díaz Gonzalez
##     Zebenzui Pérez Ramos
## Descripcion: Se encarga de mostrar los conceptos disponibles para un
##              tema dado.

from Cheetah.Template import Template
from TestDisponibles import TestDisponibles
from xml.dom import minidom
from Cifrado import Cifrado

class VerTema(TestDisponibles):
    def writeContent (self):
        req = self.request()
        resp = self.response()
        file = req.cookie ('fichero')
        tema = req.cookie ('tema')
        if (req.hasCookie('concepto')):
            resp.delCookie('concepto')
        if (req.hasCookie('pregunta')):
            resp.delCookie('pregunta')
        cad_descifrada = Cifrado.Descifrar(fich = file)
        dom = minidom.parseString (cad_descifrada)
     # Seleccionamos el tema con el que estamos trabajando
        lista_temas = dom.getElementsByTagName ('TEMA')
        for _tema in lista_temas:
            if (_tema.getAttribute('ID') == tema):
                nodo_tema = _tema
                break
```

```
# Seleccionamos la lista de conceptos de ese tema
   lista_conceptos = [e for e in nodo_tema.childNodes
                if e.nodeType == e.ELEMENT_NODE]

   t = Template (file='hevah//templates/VerTema.tpl.html')
   t.tema = nodo_tema.getAttribute ('TEXTO')
   t.lista_conceptos = lista_conceptos
   t.color =''
   self.mostrar (str(t))
```

D.53. VerTestOK

```
##                  HEVAH
## Proyecto de fin de carrera de Ingenieria Informática
## Universidad de La Laguna
## Tutora: Carina Gonzalez Gonzalez
## Autores: Fco. Javier Medina Santos
##     Daniel J. Díaz Gonzalez
##     Zebenzui Pérez Ramos
## Descripcion: Realiza las operaciones de gestión oportunas en función de
##              la selección del usuario:
##                   * Ver: se pasa a la gestión de conceptos
##                   * Borrar: se borra el tema
##                   * Editar: se edita el tema para modificaciones
##                   * Nuevo: se crea un nuevo tema

from TestDisponibles import TestDisponibles

class VerTestOK (TestDisponibles):
    def writeContent (self):
        req = self.request()
        resp = self.response()
     # Pasamos a la gestión de conceptos
        if (req.hasField ('ver')):
            resp.setCookie ('tema',req.field ('radio_tema'))
            resp.sendRedirect ('VerTema')
     # Creamos un nuevo tema
        if (req.hasField ('crear')):
            resp.sendRedirect('CrearTema')
     # Borramos un tema
        if (req.hasField ('borrar')):
            if (req.hasField('radio_tema')):
                resp.setCookie ('tema', req.field ('radio_tema'))
                resp.sendRedirect('BorrarTema')
            else:
```

```
            resp.sendRedirect('VerTest')
    # Editamos un tema
    if (req.hasField ('editar')):
        if (req.hasField('radio_tema')):
            resp.setCookie ('tema', req.field ('radio_tema'))
            resp.sendRedirect('EditarTema')
        else:
            resp.sendRedirect('VerTest')
```

D.54. VerTest

```
##                  HEVAH
## Proyecto de fin de carrera de Ingenieria Informática
## Universidad de La Laguna
## Tutora: Carina Gonzalez Gonzalez
## Autores: Fco. Javier Medina Santos
##     Daniel J. Díaz Gonzalez
##     Zebenzui Pérez Ramos
## Descripcion: Se encarga de mostrar los temas disponibles para un
##              test dado.

from Cheetah.Template import Template
from TestDisponibles import TestDisponibles
from xml.dom import minidom
from Cifrado import Cifrado

class VerTest(TestDisponibles):
    def setAsistente (self):
        self.addPaso ('Tests')
        self.addPaso ('Temas')
        self.addPaso ('Conceptos')
        self.addPaso ('Preguntas')
        self.addPaso ('Respuestas')
        self.setActual (1)

    def writeContent (self):
        req = self.request()
        resp = self.response()
        file = req.cookie ('fichero')
        if (req.hasCookie('tema')):
            resp.delCookie('tema')
        if (req.hasCookie('concepto')):
            resp.delCookie('concepto')
        if (req.hasCookie('pregunta')):
```

```
            resp.delCookie('pregunta')

        cad_descifrada = Cifrado.Descifrar(fich = file)
        dom = minidom.parseString (cad_descifrada)
        raiz = dom.getElementsByTagName ('TEST')[0]
        temas = raiz.getElementsByTagName ('TEMA')
        t = Template (file='hevah//templates/VerTest.tpl.html')
        t.test = raiz.getAttribute ('TEXTO')
    # Seleccionamos la lista de temas del test
        t.lista_temas = []
        for tema in temas:
            t.lista_temas.append([tema.getAttribute('ID'),
                      tema.getAttribute('TEXTO')])
        self.mostrar (str(t))
```