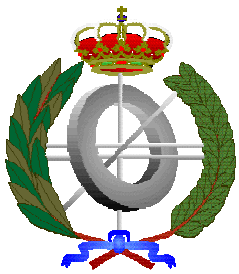
Sistemas Informáticos

Curso 2009–2010



Tecnologías de reconocimiento por voz y su aplicabilidad en videojuegos.

Realizado por:

Pablo Caloto Crespo

Manuel Moranchel Edras

Ángel Ruiz Alonso

Dirigido por:

Luis Hernández Yáñez

Pedro Antonio González Calero

Departamento de Ingeniería del Software e Inteligencia Artificial

Facultad de Informática

Universidad Complutense de Madrid

ÍNDICE DE CONTENIDOS:

1 Agradecimientos

En primer lugar agradecer a nuestros tutores Luis Hernández Yáñez y Pedro Antonio González Calero todo el apoyo y la confianza que han depositado en nosotros. Por la tranquilidad que nos han transmitido en momentos en los cuales no nos veíamos capaces de llegar a buen puerto...

Mención especial requiere David Llansó, ya que ha sido el que nos ha ayudado y explicado cualquier duda que nos surgía acerca de la arquitectura, diseño e implementación de Javy.

También agradecer a todos los profesores y compañeros con quienes hemos compartido tanto durante estos años, junto a los cuales hemos pasado buenos y malos momentos, y que han estado ahí para tendernos una mano cuando lo hemos necesitado. Además de vivir esta experiencia junto a ellos, hemos conocido a alguno de nuestros mejores amigos.

A nuestras familias por habernos apoyado y comprendido durante estos años, por haber tenido tanta paciencia en especial durante estas últimas semanas tan frenéticas, y sobre todo por haber confiado desde el primer día y no haber dudado nunca de nosotros.

Por último pero no por ello menos importante, agradecer a nuestros amigos todo su apoyo, sin duda necesario para sacar fuerzas de flaqueza. Gracias por estar ahí siempre, pero sobre todo en los momentos difíciles, y también por distraernos y ayudarnos a evadirnos de la presión y nervios que supone entregar un proyecto

Este proyecto está dedicado a todos vosotros, GRACIAS.

2 Resumen/Abstract

2.1 Resumen

El proyecto ha consistido en la investigación en el campo de modelos acústicos, generación de dichos modelos en distintos idiomas, e integración de los mismos en la infraestructura de un juego con el objetivo de manejar el avatar del juego a través de órdenes de voz, sin necesidad de utilizar ninguna interfaz física, como pueda ser el teclado o el ratón.

Una vez conseguido esto hemos realizado una aplicación web de cara a la recolección de las grabaciones, gracias a la cual se ha automatizado el proceso de entrenamiento de un modelo acústico previamente generado manualmente.

También hemos tratado de mostrar el potencial de estos sistemas reconocedores cuando son integrados en aplicaciones, no solo de escritorio sino videojuegos desde los que podemos manejar un lenguaje lo más próximo al natural.

2.2 Abstract

The Project has consisted in researching into acoustic model field, the way to generate them in different languages, and a later application in a game, with the aim of controlling the avatar by voice commands, without using any other physical interface like keyboard or mouse.

Once we have this, we create a web application to record voice transcriptions and train an acoustic model previously generated in an automated way.

3 Resumen

Analizando las tendencias actuales se advierte la importancia de los sistemas de reconocimiento por voz. Actualmente los mayores referentes en desarrollos informáticos están concediendo grandes cantidades de sus fondos a investigación en estas áreas.

Todos tenemos conocimiento de estos procedimientos desde hace algún tiempo, en sistemas telefónicos y otras aplicaciones. Sin embargo su funcionamiento no se podía considerar ni próximo a lo aceptable y su evolución se había visto truncada dado que la tecnología no permitía una mayor tasa de acierto.

Es ahora, en estos tiempos, cuando podemos decir que la tecnología se encuentra en su punto álgido para el desarrollo de estos sistemas. Pero no solo la tecnología corre a favor, el mercado también. Actualmente se vive una autentica revolución en la manera de interactuar con las máquinas. Esto, lógicamente, influye directamente en los sistemas que estamos tratando.

Lo expuesto anteriormente tuvo sin duda un gran peso en nuestra decisión al escoger este proyecto, sin embargo, hay otro motivo que nos impulsó a hacerlo: la conciencia social. Estamos rodeados de gente con trabas para utilizar los sistemas, para los que la tecnología de reconocimiento del habla supone un importantísimo avance.

Prácticamente todo el mundo ha disfrutado de juegos, tanto en ordenadores como en consolas con mando, *joystick* o teclado, es hora de cambiar esto y eliminar las barreras que la sociedad impone a estas personas. Estas cuestiones que para nosotros supone un lujo, un divertimento, para determinados usuarios significan puertas que se abren, barreras que se superan.

Es un modo no definitivo, sin duda, pero una forma de acercar

a estas personas al futuro, a las posibilidades. Dotarles un poco mas a la independencia que les permita valerse por si mismo y pasar por el sufrimiento de tener que depender de alguien para hacer las cosas mas nimias.

Así es como decidimos aceptar la propuesta que nuestros tutores Pedro y Luis nos propusieron. Pero entonces deberíamos atacar nuevos retos.

Los desafíos iniciales que surgieron eran averiguar como es el reconocimiento por voz. Que programas implementan eso y como funcionan. Así en este primer momento descubrimos programas como *Sphinx* y *Julius* desarrollados en Java y C respectivamente. También deberíamos saber donde lo debíamos integrar.

Buscando videojuegos dimos con *OpenSim*, una modalidad de código abierto de *Second Life* para este juego, también sería necesario usar *Hippo Open viewer*, visor necesario para acceder de un modo grafico al servidor *OpenSim* y donde deberíamos integrar el reconocedor. A estas alturas vimos que la mayoría de los videojuegos estaban implementados en C o C++ lo que ya nos empujaba a elegir *Julius* como motor de reconocimiento.

Para terminar optamos por informarnos acerca de las características de cada uno. Este fue el factor definitivo, vimos que *Julius* tenia una tasa de acierto sensiblemente mayor que la de su competencia. Por lo que finalmente nos inclinamos por el reconocedor *Julius*.

Intentando poner en marcha este reconocedor en su versión compilada, decidimos migrarnos a Linux lo que nos facilitaría, mas adelante, todas las labores de compilación. Documentándonos sobre la puesta en marcha de la aplicación descubrimos dos nuevos conceptos: Modelo Acústico y de Lenguaje.

El Modelo de Lenguaje hace referencia a la gramática sobre la que el reconocedor va a trabajar y especificará las frases que podrá identificar. Esta se debe especificar de una manera muy concreta para *Julius*. Así se deben crear dos ficheros .*voca* y .*grammar*. En los primeros se especifica la lista de palabras que contendrá la gramática así como los grupos a los que pertenece, que más adelante nos permitirán especificar como se combinarán. También aparecerá la descomposición en fonemas de cada palabra. En los archivos .*grammar* se especifican las construcciones de frases apoyándose en las clases que se han definido. Esto permitirá establecer las conexiones entre las clases para discriminar como se construirán las frases. Todo esto debe ser compilado para que se creen unos ficheros sobre los cuales se apoyará el reconocedor.

Otro concepto que aprendimos fue el de Modelo Acústico. Este se encarga de relacionar ciertas ondas sonoras con ciertos fonemas. Así cada fonema tendrá una representación como sonido. Esto lógicamente tiene muchas variaciones. Una persona no pronuncia nunca de igual manera una misma palabra. También cambia la representación de un hombre a otro. Otro factor a tener en cuenta será el dialecto, no es lo mismo una persona con Inglés Americano, que un inglés de Reino Unido, ni de Canadá.

Como se ve el modelo acústico resulta un problema de significada envergadura. Entonces ¿Como se consigue que todo esto funcione? Pues se ha demostrado que se pueden conseguir resultados más que aceptables empleando modelos estadísticos como los *Modelos Ocultos de Markov.*

Se pueden obtener representaciones numéricas de cada sonido o fonema. Pero como hemos dicho este puede variar notablemente según ciertos factores. Aquí es donde entran *los Modelos Ocultos de Markov*, se encargan de recopilar información de distintas fuentes y

obtener una representación que resulte aceptable para estas fuentes. De esta manera, si conseguimos varias representaciones de un fonema, podemos analizarlos con los modelos de *Markov* y obtener una representación que satisfaga el mayor número de individuos.

Aquí es donde pasa *Voxforge* a formar parte de nuestro proyecto. Esta página se encarga de recoger grabaciones de todos los idiomas y de todas las personas que quieran. Todas las noches construyen modelos acústicos basados en los de *Markov*. Es una página inglesa por lo que los modelos más completos se encuentran para esta lengua. Como trabajan con gente de todo el mundo y tienen un importante número de grabaciones tienen unos modelos muy completos que funcionan casi a la perfección.

En este punto ya éramos capaces de poner en marcha el reconocedor y que éste identifique algunas gramáticas básicas en ingles. En este punto nos dimos cuenta de que el reconocedor era sensible a la pronunciación. Era necesaria una pronunciación americana para que el reconocedor tuviese mas tasa de acierto. Esto se debe a la desviación que sufre el modelo al tener más grabaciones del continente americano. Pese a esto también fallaba para cierto tipo de palabras. Con el tiempo advertimos que no era por las palabras en sí, sino por que éstas contenían fonemas poco comunes y que, por tanto, estaban poco entrenados.

Con estas salvedades teníamos un modelo en inglés que reconocía órdenes simples como *Left, right, go y back* , siento esta última una de las palabras que no son bien reconocidas.

Puesto que el número de grabaciones necesaria para el modelo es muy elevada y la pagina es inglesa, el modelo más completo se encuentra desarrollado para esta lengua.

Para el lenguaje español existen también otro tipo de trabas. La

mayoría de las grabaciones que tiene *Voxforge* son de gente hispanoamericana y la pronunciación varia notablemente, así una palabra como accidentados en España se pronunciaría como A K C I D E N T A D O S mientras que para los hispanohablantes sería A K S I D E N T A D O S, nótese la s tras la k. Esto supone un problema considerable, tanto por que no es posible construir un modelo para España basado en el americano y viceversa.

Lo expuesto anteriormente nos condujo a decidirnos por el idioma inglés. Así hasta este momento teníamos el reconocedor, el lenguaje ingles y la conjunción de *Hippo Open Viewer* y *OpenSim* para el juego donde lo ensamblaríamos. El sistema operativo, Linux.

Comenzamos a intentar atacar el problema de la integración, buscamos documentación sobre el código de los juegos pero nos encontramos que ésta era somera, poco concisa y anticuada. Nos pusimos en contacto con los administradores de los sistemas y obtuvimos poca ayuda, o ninguna. Finalmente conseguimos dar con el emplazamiento de las acciones sobre el avatar. La problemática era que estos sistemas permitían una serie de movimientos muy limitados sobre éste. Todas estas conclusiones nos hicieron descartar este juego.

Ahora la problemática era encontrar un nuevo juego, éste debería estar implementado en C o C++ para poder continuar con *Julius* y la investigación que llevábamos hasta el momento. Fue así como nuestro tutor Pedro nos propuso introducirlo en un juego que actualmente se está desarrollando en el departamento: Javy2 desarrollado en Visual Studio. Una vez adoptado este cambio procedimos a obtener el código fuente del juego y compilarlo.

De manera simultánea en Japón el equipo de desarrollo de *Julius* publicó nuevos avances entre los cuales se incluía soporte para el IDE

Visual Studio, lo cual pudimos emplear en la elaboración de nuestro proyecto *Vocalius*.

Ahora ya teníamos todo lo necesario para empezar a desarrollar la funcionalidad. Comenzamos a investigar como interactuar con la arquitectura de *Julius* y así obtuvimos un primer ejecutable con el funcionamiento del reconocedor por consola.

Una vez familiarizados con la arquitectura de *Julius* tomamos la decisión de aplicar un patrón *Facade* al reconocedor, para abstraer la funcionalidad de éste y darle un punto de acceso común. Por lo que para adaptarse a Javy decidimos construir la librería *Vocalius* y desarrollamos un prototipo que nos permitiese probar su funcionamiento. También decidimos dotarla de los métodos con los que funcionaba Javy para facilitar su posterior integración. Otra decisión de diseño fue la aplicación de un patrón *Observer*. Este nos permite abstraer el reconocimiento de manera que el objeto que se establezca como observador decida que funcionalidad implementar cuando se lance el disparador *(trigger)* de reconocimiento.

Tras haber desarrollado la librería se paso a integrarla con Javy. Creamos una clase conectora para enlazar todo. Era necesario que el reconocedor corriese en un hilo a parte. Se empleó también una cola para las frases reconocidas, con esto surgió un problema ya que podían acceder a ésta dos hilos al mismo tiempo. El reconocedor accedería a ella como productor mientas que el hilo de Javy accedería a ella como consumidor. Así se habilitó un semáforo *(mutex)* que protegiese el acceso a ella.

Una vez integrado el reconocedor, era necesario conectarlo con los movimientos del avatar. Decidimos hacer un paso intermedio, este consistió en la inclusión de un analizador léxico que se encargase se decodificar las frases según acciones. De esta manera añadíamos una

capa de abstracción. Lo que nos permitía, en caso de tener que hacer algún cambio, como en el idioma, que tan solo tengamos que cambiar la lista de acciones y no todo el sistema.

Así con esta capa intermedia ya podíamos empezar a relacionar lo reconocido con las acciones. La decisión fue sustituir la clase encargada del control del avatar extendiéndola con la funcionalidad que queríamos añadir. De este modo manteníamos el control con el teclado y añadíamos el control por voz.

Así fue como conseguimos el control básico del sistema pero no estábamos del todo conformes con el hecho de que no existiese un modelo acústico en español. Por lo que decidimos estudiar aun más la construcción de estos modelos, estudiando el número de grabaciones que son necesarias para conseguir uno aceptable. Establecimos una serie de gramáticas de control sobre las cuales trabajaríamos y pasamos a la tarea de construir algunos modelos.

De este modo topamos con HTK, una herramienta de Cambridge University Engineering Department que permite construir modelos basándose en los Modelos Ocultos de *Markov*. Esta herramienta funciona bajo el sistema operativo Linux, y su compilación requiere de una versión muy específica del compilador gcc, exactamente la versión 3.4, nosotros nos encontramos ante la situación de tener instalada en nuestros equipos la versión 4.0 debido a tener unas versiones recientes de linux, por lo que tuvimos que investigar la manera de poder cambiar de una versión a otra. Investigando por internet descubrimos como cambar la versión del compilador a la versión que necesitamos, aunque fue una tarea que nos llevo unos días debido a nuestra experiencia en este tipo de actividades.

4 Introducción

En la actualidad, existe una auténtica revolución conceptual en la informática. Podemos ver como algunos sistemas, Wii de Nintendo, han revolucionado el mundo de las consolas y videojuegos simplemente modificando el mando para que reconociese movimientos. Un reciente proyecto de Microsoft, proyecto Natal, nos permite a través de un sistema de *webcams*, interactuar directamente con la consola tan solo moviéndonos delante de ella.

Vemos también, en el cine, una autentica revolución que bien podría estar liderada por Avatar de James Cameron, en la que el modo de ver las películas da un giro de ciento ochenta grados, sumergiéndonos en escenarios 3D. Cada vez más ordenadores y teléfonos móviles se dotan de pantallas táctiles que permiten directamente tocar lo que queremos.

En efecto, vivimos una revolución hacia algún lugar que hace unos años podría tacharse de utópica, salida de una novela de Orwell. Se está renovando la manera de interactuar con los ordenadores, dotándolos de cierta humanidad.

En este contexto no podíamos dejar de lado los sistemas de reconocimiento de habla. Vemos cada día sistemas de atención telefónica con los que interactuamos a través de órdenes vocales. Nuevos sistemas operativos como Windows 7 o Mac OS X incluyen opciones de reconocimiento por voz.

No podemos dejar de lado a Google, con su importante papel marcando tendencias tecnológicas. Actualmente ha mostrado una fuerte inclinación a este tipo de sistemas. Podemos ver en los últimos modelos de móviles del gigante como integran búsquedas en internet

empleando tan solo la voz o búsquedas en su sistema Google Maps.

En este marco es donde se sitúa este proyecto de sistemas informáticos. Analizamos las actuales tecnologías para este tipo de sistemas y mostraremos su aplicabilidad.

Otra característica de estos sistemas es que están basados en sistemas de Inteligencia Artificial de aprendizaje. Es por esto que la fiabilidad del resultado en su reconocimiento dependerá directamente del entrenamiento que tengan. Por tanto también estudiaremos las necesidades de estos sistemas.

5 Objetivos

A continuación enunciaremos los principales objetivos del proyecto, explicando en qué consiste cada uno de ellos:

Investigación en el campo del reconocimiento de voz con tecnologías de código libre. De esta tarea, que podemos denominar como principal, será de donde surjan el resto de líneas que a continuación detallaremos.

- ✓ Elección de un motor de reconocedor. Dicha decisión vendrá condicionada por el lenguaje en el que se encuentren desarrolladas, la tasa de acierto de reconocedor así como el soporte existente para ella.

- ✓ Elección de un modelo acústico que funcione adecuadamente y que permita mostrar el potencial de este tipo de aplicaciones. Para esta decisión se tendrá en cuenta la tasa de acierto, el coste del modelo y el idioma.

- ✓ Elección del modelo de lenguaje. Esta decisión dependerá directamente del reconocedor que se escoja, así como el modelo acústico que se ha empleado. El modelo del lenguaje será uno de los requisitos más cambiantes con los que nos encontraremos. Puede anotarse como riesgo del proyecto.

- ✓ Elección del sistema sobre el que se va a integrar la lógica citada anteriormente. Características que deberemos tener en cuenta serán el sistema operativo para el que se esté implementado, lenguaje de programación, documentación y facilidad para obtener el código fuente.

- ✓ Desarrollo de una aplicación web que permita crear un modelo acústico mediante la cual cualquier persona pueda realizar

grabaciones, enviar dichas grabaciones para completar el modelo existente y descargar el nuevo modelo.

6 Conceptos teóricos

En este punto vamos a tratar unos cuantos conceptos teóricos que consideramos clave para el correcto seguimiento de la memoria.

6.1 Modelo acústico

Un Modelo Acústico es la representación de un sonido basada en la comprensión del comportamiento del sonido en la disciplina de la Acústica. Esta representación puede ser visual, o basada en computadoras, y puede ser usada para la o para la composición.

En un sentido más amplio, una investigación basada en la percepción de las características y el comportamiento de un sonido podría involucrar también a un modelo acústico.

En nuestro caso de estudio emplearemos un caso muy concreto de modelo acústico. Este nos permitirá hacer comparaciones de lo que se está escuchando con el modelo existente.

El modelo acústico se crea a partir de muestras de audio y las transcripciones de sus textos. Con un software específico se crean representaciones estadísticas, a partir del análisis de *Markov*, de cada sonido con su palabra correspondiente.

6.1.1 Caso particular

Para este proyecto estudiamos las alternativas que figuraban en la red de código abierto. Tras el análisis decidimos que la opción más completa y que mejor resultados nos aportaba era el proyecto *Voxforge[1]. Voxforge[1]* tiene como objetivo recoger voz que la gente done para ser usada con herramientas de reconocimiento de voz libre y de Código abierto.

Las grabaciones que los usuarios vayan enviando al sistema serán

accesibles por cualquiera y también se publicará una versión compilada de las grabaciones del idioma inglés para utilizar directamente con los programas de reconocimiento de voz más comunes.

En el programa de reconocimiento *Julius*, este modelo se especifica con los ficheros *hmmdefs* y *tiedlist*. Este tipo de modelos se pueden "entrenar" para que reconozcan un mayor número de palabras o se pueden crear nuevos modelos para distintos idiomas, distintos tipos de pronunciación, etc. Para ello se usan programas específicos como HTK. Estos programas están basados en diversas colecciones con grabaciones de palabras que, posteriormente, querremos poder reconocer. Como se puede deducir, a mayor número de grabaciones de cada palabra, mayor probabilidad de acertar a la hora del reconocimiento ya que se tendrá un modelo acústico más completo. Pero este tipo de sistemas trabajan a nivel de fonemas. Esto implica que si tenemos dos palabras que comparten un mismo fonema, entrenando ambas conseguiremos una mayor fiabilidad en el reconocimiento de este.

Parte del proyecto consistirá en la creación de un modelo acústico en español, por lo que desarrollaremos una descripción más extensa del proceso en dicho apartado.

6.2 Modelo de lenguaje

El reconocimiento de la gramática restringida trabaja reduciendo las típicas frases reconocidas a un tamaño más pequeño que la gramática formal. Este tipo de reconocimiento trabaja mejor cuando el hablante proporciona respuestas breves a cuestiones o preguntas específicas: las preguntas de “si” o “no”, al elegir una opción del menú, un artículo de una lista determinada, etc. La gramática especifica las palabras y frases más típicas que una persona diría

como respuesta rápida y después asocia esas palabras o frases a un concepto semántico.

Si el hablante dice algo que gramaticalmente no tiene sentido, el reconocimiento fallará. Normalmente, si el reconocimiento falla, la aplicación incitará al usuario a repetir lo que ha dicho y el reconocimiento se intentará de nuevo.

6.2.1 Caso particular

En nuestro caso *Julius* lo que hace es mapear una salida a partir de la entrada que recoge como parámetro de entrada, y mostrará la salida que más se asemeje a las palabras y gramática que tenga, por lo que es sistema debe estar bien entrenado para que no digamos una frase gramaticalmente correcta y el módulo de reconocimiento de voz interprete otra frase, que en consecuencia implicará que el avatar realizará una acción distinta a la que realmente queríamos.

La manera clásica para especificar el modelo de lenguaje se especifica inicialmente con dos ficheros, uno con extensión .*voca* y otro .*grammar*. En el primero se incluirán las palabras con sus representaciones en fonemas. A su vez estas palabras deberán estar agrupadas en conjuntos con una etiqueta identificativa que luego se usará para realizar la gramática.

Por ejemplo para declarar los operadores de movimiento deberá ponerse de este modo:

```
% OPMOV

LEFT        l eh f t

RIGTH       r ay t
```

Aquí se puede ver como se crea la clase OPMOV que contiene dos

palabras: LEFT y RIGHT. La cadena de palabras "l eh f t" es la descomposición en fonemas de la palabra *left*.

Por otro lado el fichero .*grammar* se usará para definir la gramática que queremos reconocer. El punto de entrada a la gramática se representa con el símbolo S seguido de dos puntos ":" y los conjuntos que formarán las palabras en el orden indicado.

A continuación vemos un ejemplo:

S : NS_B SENT NS_E

SENT: TAKE_V FRUIT

En este ejemplo vemos como el punto de entrada es S. Tras esto leemos NS_B SENT NS_E. Los símbolos NS_B y NS_E son marcas de silencio para identificar el comienzo y fin de cada frase. El símbolo sent se detalla en la siguiente línea. En la segunda línea podemos ver dos clases TAKE_V y FRUIT. TAKE_V identifica la acción de coger, es un símbolo que se tendrá que definir en el correspondiente fichero .voca. Finalmente FRUIT es otra clase donde incluiremos los distintos tipos de fruta.

Estos ficheros no pueden incluirse directamente en *Julius*, es necesario realizar un tratamiento sobre ellos para obtener distintos ficheros, entre los que se encuentra un autómata y otros ficheros adaptados para *Julius*. Para realizar este tratamiento *Julius* proporciona el *script mkdfa.pl*. Esto generará distintos ficheros .*term*, .*dfa* y .*dict*. Los últimos serán los que se le pasaran a la aplicación *Julius* para tratar con el modelo de lenguaje.

Lógicamente los fonemas de las palabras que empleemos en nuestro vocabulario tienen que estar entrenados en el modelo acústico ya que sino el sistema no podrá reconocer las palabras, o la tasa de acierto

con la que el sistema reconozca correctamente las palabras que pronunciamos será muy baja.

Fichero .*voca*:

Sirve para especificar la descomposición en fonemas de cada palabra y el conjunto de palabras asociadas a cada símbolo terminal de la gramática. Un ejemplo donde se puede ver la estructura de este fichero es el siguiente:

```
% NS_B

<s>  sil

% NS_E

</s>  sil

% HMM

FILLER  f m

FILLER  w eh l

% TAKE_V

take t ey k

% PLEASE

please  p l iy z

% FRUIT_N_1

apple  ae p ax l

orange  ao r ax n jh

banana  b ax n ae n ax
```

plum p l ah m

% FRUIT_N

apples ae p ax l z

oranges ao r ax n jh ax z

bananas b ax n ae n ax z

% NUM

one w ah n

two t uw

three th r iy

four f ao r

Como podemos ver, el fichero consiste en una lista de palabras (las palabras de nuestra gramática) junto con su descomposición en fonemas. Estas palabras aparecen agrupadas según el símbolo terminal en el que se engloben (en este ejemplo, los símbolos terminales serían NUM, FRUIT_N,...).

Ficheros .*grammar*:

Los símbolos terminales aparecerán por otro lado en las reglas de la gramática, especificadas este fichero. Un posible ejemplo de ese fichero, en línea con el ejemplo anterior, podría ser:

S : NS_B HMM SENT NS_E

S : NS_B SENT NS_E

SENT: TAKE_V FRUIT PLEASE

SENT: TAKE_V FRUIT

SENT: FRUIT PLEASE

SENT: FRUIT

FRUIT: NUM FRUIT_N

FRUIT: FRUIT_N_1

Si nos fijamos un poco, podemos observar que las líneas anteriores no son más que las reglas de nuestra gramática: los símbolos en negrita son los símbolos no terminales, lo que hay a la derecha de ':' es el conjunto de símbolos terminales y no terminales a los que llegamos a partir del símbolo no terminal de la producción,... Aquí, por tanto, es donde se especifican las formas de frase que posteriormente querremos reconocer.

Los ficheros *.voca* y *.grammar* no pueden ser leídos por *Julius*, es necesario compilarlos previamente antes de pasarle la información al programa. Para ello, disponemos de este *script* que nos convertirá estos ficheros en otros equivalentes con extensiones *.dict* y *.dfa*. Con un formato que *Julius* si puede tratar (estos son los ficheros que *Julius* usará para saber qué es lo que debe de reconocer).

Por último, también podemos hablar de fichero *.term*, generado junto con los archivos *.dict* y *.dfa*, que tampoco es de vital importancia pero que sirve para reunir el conjunto de símbolos terminales del fichero *.voca*. Siguiendo el ejemplo, el contenido de este fichero sería:

0 NS_B

1 NS_E

2 HMM

3 TAKE_V

4 PLEASE

5 FRUIT_N_1

6 FRUIT_N

7 NUM

N-Gram:

Es una subsecuencia de n elementos de una secuencia dada. Los elementos en cuestión pueden ser fonemas, sílabas, letras o palabras de acuerdo a la aplicación sobre la que se esté aplicada la gramática. Algunos modelos de lenguaje construidos sobre n-grams son los Modelos de Markov de orden n-1.Un modelo n-gram es un tipo de modelo probabilístico para predecir el siguiente elemento en una secuencia. Se utilizan en varias áreas de procesamiento de lenguaje natural y análisis de la secuencia genética.

HMM:

Los modelos ocultos de Markov (HMM del inglés *Hidden Markov Model*) asumen que el sistema estudiado sigue un proceso de *Markov* con parámetros desconocidos.

La tarea fundamental consiste en determinar los parámetros ocultos a partir de los parámetros observados. La diferencia fundamental respecto a un modelo de *Markov* habitual consiste en que los estados no son directamente visibles para el observador, pero sí lo son las variables influenciadas por el estado. Cada estado tiene una

distribución de probabilidad asociada sobre el conjunto de posibles valores de salida. La secuencia de valores de salida generados a partir de un HMM nos dará cierta información sobre la secuencia de estados.

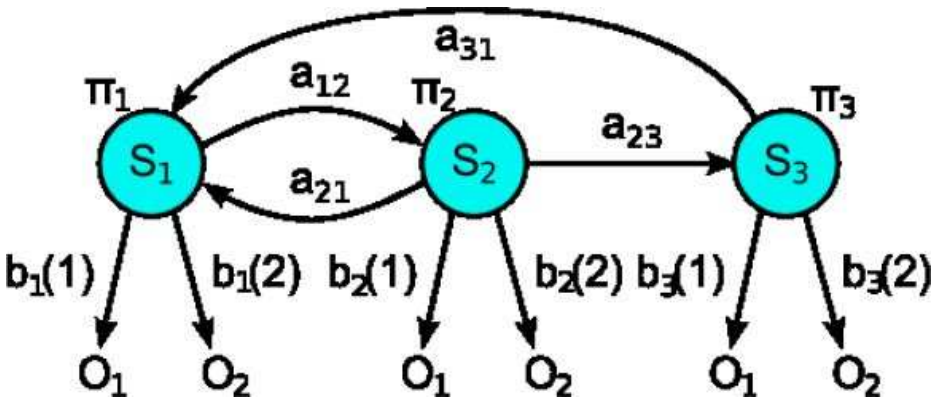
Los tres problemas fundamentales a resolver en el diseño de un modelo HMM son: la evaluación de la probabilidad (o verosimilitud) de una secuencia de observaciones dado un modelo HMM específico; la determinación de la mejor secuencia de estados del modelo; y el ajuste de los parámetros del modelo que mejor se ajusten a los valores observados.

Una cadena de *Markov* es un proceso estocástico discreto en el que el pasado es irrelevante para predecir el futuro dado el presente. Es decir un proceso de *Markov* es una secuencia X_0, X_1, X_2, etc. de variables aleatorias cuyo conjunto de sus posibles valores se denomina el espacio de estados.

El valor de X_n es el estado del proceso en el tiempo n. Se cumple que (propiedad de *Markov*):

$P(X_{n+1}|X_0, \bullet\bullet\bullet X_n) = (X_{n+1}|X_n)$

En este caso las variables Xi solo pueden tomar un conjunto finito de valores. Una forma de visualizar un proceso es mediante un grafo dirigido. El arco que va del estado i al j estará etiquetado con $P(X_{n+1} = i|X_n = j)$ (probabilidad de transición) Este grafo se puede representar mediante una matriz (matriz de transición).



Un Modelo Oculto de *Markov* (HMM del inglés -*Hidden Markov Model*-) es una 5-tupla (N,M,A,B,∏) donde:

N: número de estados el conjunto de estado se denota como:

$S = \{S_1, \ldots, S_N\}$

M: número de símbolos observables el conjunto de observables se denota como $O = \{O_1, \ldots O_M\}$

$A = \{a_{ij}\}$: probabilidades de transición entre estados $a_{ij} = P(q_{t+1} = S_j \mid_{qt} = S_i)$

$B = \{b_j(k)\}$: probabilidad de observar Ok en el estado S_j $b_j(k) = P(O_k \mid q_t = S_j)$

$\Pi = \{\Pi_i\}$: probabilidades iniciales de cada estado $\Pi_i = P(q_1 = S_i)$

7 Elección de Tecnologías y Herramientas

A continuación enunciaremos las distintas tecnologías sobre las que hemos investigado y las herramientas que hemos utilizado para el desarrollo del proyecto.

7.1 Tecnologías

7.1.1 Software de reconocimiento de voz

7.1.1.1 CMU Sphinx

También llamado *Sphinx* de forma abreviada, es el término que describe un grupo de sistemas de reconocimiento de voz desarrollado en *Carnegie Mellon University*, utilizando modelos ocultos de *Markov* y un modelo acústico estadístico basado en gramáticas. Incluye una serie de reconocedores de habla (Sphinx 2-4) y un entrenador de modelo acústico (SphinxTrain).

En 2001, el grupo de *Sphinx* de la universidad de *Carnegie Mellon* empezó a desarrollar varios componentes de los sistemas de reconocimiento de habla en código abierto, incluyendo *Sphinx* 2 y posteriormente *Sphinx* 2 en 2001. Los decodificadores del habla incluían modelos acústicos y aplicaciones de ejemplo. Los recursos disponibles incluyen software para el entrenamiento del modelo acústico, para la compilación del modelo de lenguaje y un diccionario de pronunciación de dominio público.

Sphinx:

Es un sistema de reconocimiento de habla independiente de la persona que realice la locución. Utiliza Modelos ocultos de Markov, y un modelo de lenguaje basado en n-gramáticas estadístico. Fue desarrollado por Kai-Fu Lee. Únicamente tiene interés histórico ya

que ha sido mejorado en posteriores versiones.

Sphinx 2:

Un sistema de reconocimiento del habla más rápido, desarrollado originalmente por X-D Huang y publicado como código abierto bajo una licencia BSD en *SourceForge* por Kevin Lenzo en Linux World en el año 2000. Se centra en el reconocimiento en tiempo real del habla de cara a aplicaciones manejadas por voz. Incorpora utilidades como generación parcial de hipótesis, cambio dinámico de lenguaje, etc. Se utiliza en sistemas de diálogo y sistemas de aprendizaje de lenguaje. Se puede utilizar en ordenadores basados en PBX como Asterisk. El código de *Sphinx* 2 ha sido utilizado en varios programas comerciales. El desarrollo del decodificador en tiempo real está teniendo lugar dentro del proyecto Pocket Sphinx.

Sphinx 3:

Sphinx 2 utiliza una representación semicontínua de modelos acústicos. *Sphinx* 3 adopta la representación de modelo oculto de Markov que es el predominante, y ha sido utilizado para aumentar la precisión en el reconocimiento en tiempo no real. Los desarrollos recientes tanto en algoritmia como en hardware, ha hecho que *Sphinx* 3 se aproxime al reconocimiento en tiempo real, pese a esto, todavía no es apto para aplicaciones interactivas críticas. Está bajo un continuo proceso de desarrollo y junto con *SphinxTrain* provee acceso a un número de modernas técnicas de modelado como LDA/MLLT, MLLR y VTLN, que proveen precisión de reconocimiento.

Sphinx 4:

Es una reescritura completa del motor *Sphinx* con el objetivo de proveer un sistema más flexible para la investigación en reconocimiento de habla, escrito completamente en Java. *Sun*

Microsystems mantiene el proyecto de Sphinx 4 y contribuye con software para el desarrollo del proyecto. Otros participantes son MERL, MIT and CMU.

Los objetivos del desarrollo actual incluyen:

- ✓ Desarrollar un entrenador de modelo acústico nuevo
- ✓ Mejorar el manejo de la configuración.
- ✓ Creación de un entorno de diseño para diseño asistido por ordenador.

Los inconvenientes que encontramos a Sphinx para decidir no utilizarlo finalmente fueron:

- ✓ Esta desarrollado en java por lo que habría que integrar Java con el proyecto C++.
- ✓ Tampoco tiene modelo Acústico propio, habría que crearlo.
- ✓ Necesita de una maquina virtual para ejecutarse, lo que supone una pérdida de rendimiento.

7.1.1.2 HTK:

The Hidden Markov Model Toolkit (HTK) es un conjunto de instrumentos portátiles para construir y manipular modelos ocultos de Markov. HTK se utiliza sobre todo para la investigación de reconocimiento de voz, aunque se ha utilizado en numerosas otras aplicaciones, incluida la investigación en la síntesis de voz, reconocimiento de caracteres y la secuenciación del ADN. HTK está en uso en cientos de sitios de todo el mundo.

HTK consta de un conjunto de módulos de librería y herramientas disponibles en código fuente. Las herramientas proporcionan

facilidades para el análisis del habla, entrenamiento de HMM, pruebas y análisis de resultados. El software soporta HMM utilizando tanto la mezcla de densidad continua Gaussiana como distribuciones discretas, y se pueden utilizar para construir sistemas de HMM más complejos. El lanzamiento de HTK contiene una amplia documentación y ejemplos.

HTK fue originalmente desarrollado en el *Machine Intelligence Laboratory*, de la Universidad de *Cambridge* (CUED) donde se ha utilizado construir los sistemas de reconocimiento del habla de CUED. En 1993 el Laboratorio de Investigación de entropía Inc. adquirió los derechos para vender HTK y el desarrollo de HTK sido transferidos a *Entropic* en 1995. HTK fue vendido por *Entropic* en 1999, cuando Microsoft compró *Entropic*. Mientras que Microsoft se reserva el derecho de autor del código original de HTK, a todo el mundo se anima a hacer cambios en el código fuente y contribuir con ellos para su inclusión en HTK3. Debido a que no siempre dispondremos de un buen modelo acústico para reconocer las palabras de nuestra gramática, existe la posibilidad de crearnos nuestro propio modelo a partir de nuestras propias grabaciones.

7.1.1.3 Julius

Es un sistema de reconocimiento de habla, destinado a investigadores y desarrolladores. Está basado en una gramática N-gram, y un contexto dependiente de los HMM. Puede casi realizar decodificación en tiempo real en la mayoría de los ordenadores en una tarea de dictado con un diccionario de 60000 palabras.

La mayoría de las técnicas han sido incorporadas como pueden ser por ejemplo, tree lexicon, cruce de palabras en un contexto, selección gaussiana,…. Para conseguir eficiencia ha sido diseñado para ser independiente de los modelos de lenguaje, y soporta varios tipos de

Modelos Ocultos de Markov, como monofonemas o trifonemas.

La plataforma principal sobre la que se está utilizando es Linux, aunque también funciona en Windows. La última versión ha sido desarrollada para ambos sistemas operativos. Es distribuido con licencia *open-source*.

Julius ha sido desarrollado como una investigación software para los sistemas de reconocimiento de habla en japonés desde 1997. Y se ha seguido investigando en una herramienta de dictado entre 1997 y 2000*, Continuous Speech Recognition Consortium, Japan* (CSRC) (2000-2003) y actualmente *Interactive Speech Technology Consortium* (ISTC).

7.1.2 Juegos para la integración

7.1.2.1 Hippo OpenSim viewer

El cliente de visualización *Hippo OpenSim*, es una modificación del visualizador de *Second Life*, pero de código libre, enfocada con el objetivo de que sea utilizado por los usuarios de *OpenSim*. De esta forma podremos visualizar previa conexión al servidor de *OpenSim* el entorno o mundo que hemos creado en el cual se desenvolverá nuestro avatar. Debido a su carácter gratuito, y ser de código abierto nos da la posibilidad de poder integrar en él, el sistema de reconocimiento de voz de cara a manejar el avatar mediante órdenes vocales.

Este cliente tiene versión tanto para Windows como para Linux, por lo que en la página oficial en el apartado de descargas, tendremos que descargarnos la versión en función del sistema operativo que vayamos a utilizar. También nos permiten descargar tanto los binarios como el código fuente por si queremos realizar alguna

modificación en el código como es nuestro caso.

Finalmente descartamos esta opción ya que al no tener documentación acerca del código y al tratarse de un proyecto tan grande, nos era imposible averiguar y meternos a investigar donde deberíamos integrar nuestro sistema de reconocimiento de voz.

7.1.2.2 Open Simulator

Es un servidor de aplicación 3D. Puede ser utilizado para crear un entorno o mundo virtual, al cual se puede acceder a través de una gran variedad de clientes y a través de múltiples protocolos. Nos permite desarrollar nuestro entorno con las tecnologías con las que nos sintamos más cómodos.

Es distribuido bajo una licencia BSD, haciendo de él *open source* y hacerlo fácil de comercializar para poder integrarlo en otro productos.

Se puede utilizar para simular un entorno virtual similar *a Second Life*™ (incluyendo compatibilidad con el cliente). Aunque todavía se considera un software de tipo alfa, mucha gente lo está utilizando para desarrollar sus proyectos sobre él.

7.1.2.3 Javy2

Javy2 es un proyecto perteneciente al grupo GAIA en el departamento de Inteligencia Artificial e Ingeniería del Software de la Facultad de Informática de la Universidad Complutense de Madrid.

Javy2 es uno de los sistemas basados en el conocimiento, donde los estudiantes pueden aprender el funcionamiento de la Máquina Virtual Java, la estructura y la compilación de Java el lenguaje.

El sistema presenta un entorno virtual 3D que simula la JVM. El usuario es simbolizado como un avatar que se utiliza para interactuar con los objetos virtuales.

7.2 Herramientas

7.2.1 Programación

7.2.1.1 Microsoft Visual Studio

La herramienta principal que se usa en la parte de programación del proyecto de Software es el entorno de desarrollo (del inglés *Integrated Development Environment* o IDE) Visual Studio 2005, que es el IDE de Microsoft.

Visual Studio es una herramienta de desarrollo diseñada por Microsoft, que apareció en el año 1997 con el nombre de Visual Studio 97. Posteriormente, han ido apareciendo nuevas versiones, hasta llegar a la actual versión Visual Studio 2010. Sin embargo, como se ha indicado, la versión utilizada ha sido Visual Studio 2005. La versión empleada fue la Profesional, ya que, por medio de la Facultad, ésta se facilita a los alumnos de forma gratuita. Sin embargo, a partir de esta misma versión (2005), Microsoft empezó a distribuir versiones con menos características de forma gratuita, que igualmente se podrían haber usado para realizar este proyecto.

Como se comentaba con anterioridad, el proyecto se podría haber desarrollado con una de estas versiones gratuitas que se ha mencionado, ya que sólo se ha hecho uso de C++, uno de los muchos lenguajes de programación con los que se puede trabajar en esta plataforma.

La elección de este IDE radica en que todo el proyecto sobre el que teníamos que integrar nuestro sistema de reconocimiento estaba

desarrollado en Visual Studio 2005, lo cual nos decanto por seguir el desarrollo en el mismo IDE. Lo cual supuso un periodo de familiarización con el mismo, ya que ninguno de los componentes del grupo había tenía experiencia a la hora de programar sobre él.

7.2.1.2 SVN

Subversion es un sistema de control de versiones libre y de código fuente abierto. *Subversion* maneja ficheros y directorios a través del tiempo. Hay un árbol de ficheros en un repositorio central. El repositorio es como un servidor de ficheros ordinario, excepto porque recuerda todos los cambios hechos a sus ficheros y directorios. Esto le permite recuperar versiones antiguas de sus datos, o examinar el historial de cambios de los mismos. En este aspecto, mucha gente piensa en los sistemas de versiones como en una especie de "máquina del tiempo".

Subversion puede acceder al repositorio a través de redes, lo que le permite ser usado por personas que se encuentran en distintos ordenadores. A cierto nivel, la capacidad para que varias personas puedan modificar y administrar el mismo conjunto de datos desde sus respectivas.

Las principales características de SVN son:

Versionado de carpetas

Subversion implementa un sistema "virtual" de ficheros versionados que sigue la pista de los cambios en todos los árboles de directorios en el tiempo. Los ficheros y los directorios están versionados.

Confirmaciones atómicas

Una confirmación o bien entra en el repositorio completamente, o no entra en absoluto. Esto permite a los desarrolladores construir y

confirmar cambios como unidades lógicas.

Metadatos versionados

Cada fichero y directorio tiene un conjunto invisible de "propiedades" adjuntos. Puede inventarse y almacenar cualquier par de clave/valor que desee. Las propiedades se versionan en el tiempo, igual que el contenido de los ficheros.

Manejo de datos consistente

Subversion expresa las diferencias entre ficheros usando un algoritmo de diferenciación binario, que funciona exactamente igual tanto en ficheros de texto como en ficheros binarios. Ambos tipos de ficheros se almacenan igualmente comprimidos en el repositorio, y las diferencias se transmiten en ambas direcciones por la red.

7.2.1.3 Cliente SVN

Puesto que nuestro proyecto se ha integrado en un sistema existente que ya poseía servidor SVN tan solo nos será necesario conseguir un cliente para poder sincronizar nuestro código con el repositorio existente.

El cliente empleado ha sido *TortoiseSVN* herramienta gratuita.

TortoiseSVN es un cliente gratuito de código abierto para el sistema de control de versiones Subversión. Y ni siquiera está obligado a usar el Explorador de Windows. Los menús contextuales de *TortoiseSVN* también funcionan en otros administradores de archivos, y en la ventana Fichero/Abrir que es común a la mayoría de aplicaciones estándar de Windows.

Todos los comandos de *Subversion* están disponibles desde el menú contextual del explorador. *TortoiseSVN* añade su propio submenú allí,

con las siguientes características:

Integración con el Shell de Windows

TortoiseSVN se integra perfectamente en el Shell de Windows. Aunque ni siquiera está obligado a usar el Explorador de Windows. Los menús contextuales de *TortoiseSVN* también funcionan en otros administradores de archivos, y en el diálogo Fichero/Abrir que es común a la mayoría de aplicaciones estándar de Windows.

Iconos sobre impresionados

El estado de cada carpeta y fichero versionado se indica por pequeños iconos sobre impresionados. De esta forma, puede ver fácilmente el estado en el que se encuentra su copia de trabajo.

Fácil acceso a los comandos de Subversión

Todos los comandos de *Subversion* están disponibles desde el menú contextual del explorador. *TortoiseSVN* añade su propio submenú allí.

7.2.1.4 Cruise Control

La integración continua constituye un modo de trabajo para el desarrollo de software, en el cual, se conoce en cada instante el estado de la copia de trabajo y, en caso de fallo, se comunica dicho fallo a la persona que ha remitido el código conflictivo, para que solucione el error que ha aparecido en el sistema. En este sistema de trabajo, se suele tener una máquina de integración, en la que se almacena el código principal mediante algún sistema de control de código fuente y en el que se suelen correr pruebas del programa en los proyectos que lo necesiten, para así comprobar su correcto funcionamiento en cada actualización del código principal.

7.2.2 Generación de Modelos

7.2.2.1 Scripts de voxforge

Dentro de las herramientas que vienen con *Julius*, se encuentran diversos *scripts* que complementan al programa principal de reconocimiento automatizando diversas tareas. Gracias a ellos y los ficheros intermedios que se van generando durante el proceso de creación de un modelo acústico, se crean otros ficheros con datos estadísticos que contienen la información necesaria para posteriormente reconocer las palabras a través de los modelos ocultos de *Markov*.

7.2.3 Documentación

7.2.3.1 Doxygen

Las razones por las cuales hemos elegido esta herramienta es que nos permite generar archivos de *LaTeX*, *manpages*, diagramas de bloques, diagramas de clases. Por lo que es más potente, además de esta forma hemos conocido una nueva herramienta que no habíamos visto durante la carrera con la ventaja de que también puede ser aplicada para la documentación de código en lenguaje C++ sin grandes cambios en el archivo de configuración del *Doxygen*, algo que también buscábamos, el hecho de aprender cosas nuevas durante la realización del proyecto. Además dicha herramienta era la que se venía utilizando para la documentación de Javy por lo que mantener *Doxygen* nos pareció lo más adecuado a la hora de generar la documentación.

7.2.4 Grabación de sonido

7.2.4.1 Audacity

Audacity es un editor de audio libre, fácil de usar y multilingüe para Windows, Mac OS X, GNU/Linux y otros sistemas operativos. En nuestro caso elegimos este software ya que era el que nos sugirió el manual de voxforge[1] para realizar las grabaciones pero se puede utilizar cualquier otro programa que nos permita realizar grabaciones y guardar el resultado en formato no comprimido (wav). Algunas características de este software son las siguientes:

- ✓ Importar y exportar archivos:
  - ➢ Importar archivos de sonido, editarlos y combinarlos con otros archivos o nuevas grabaciones. Exportar grabaciones en varios formatos de sonido.
- ✓ Edición:
  - ➢ Edición sencilla mediante cortar, copiar, pegar y borrar.
  - ➢ Utiliza ilimitados niveles de deshacer (y rehacer) para volver a cualquier estado anterior.
  - ➢ Rápida edición de archivos grandes.
  - ➢ Edita y mezcla un número ilimitado de pistas.

8 Desarrollo

8.1 Sistema de control por voz del avatar

Una vez estudiadas las alternativas y capacidades de los sistemas de reconocimiento por voz, se integrará en un sistema real para poder demostrar el potencial de este tipo de aplicaciones.

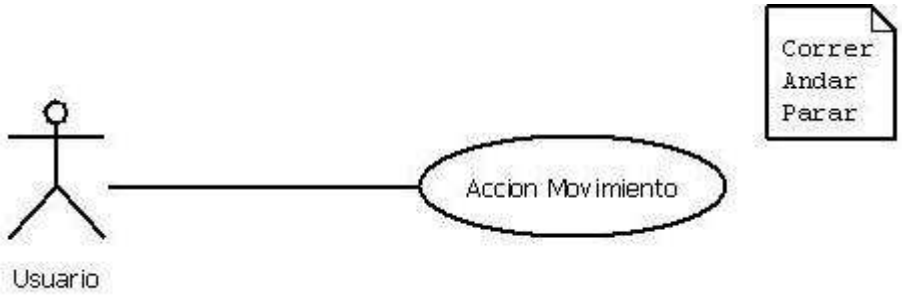
Como aplicación se ha elegido un juego. Nuestro sistema pretende controlar el avatar de dicho juego en una terminología lo más parecida al lenguaje natural. Esto permite mostrar cómo se pueden integrar sistemas de sentencias complejas de manera que el avatar responda como si de un humano se tratase.

Dado que las actividades del avatar son limitadas se restringirá el conjunto de frases reconocibles por el sistema, a una gramática específica, mejorando también así el resultado del reconocimiento.

Para el motor del reconocedor se usará *Julius*, y para el sistema donde se integrará será Javy2.

8.1.1 Especificación

8.1.1.1 Acción Movimiento



Caso de uso ID	Vocalius-02
Nombre Caso de Uso	Acción movimiento
Actores	El actor que participa en el caso de uso es el jugador.
Descripción	Este caso de uso permitirá al usuario enviar órdenes al avatar como correr, andar o detenerse
Disparador	El usuario pronuncia frases como "David corre", "Pablo para
Precondiciones	El usuario está con la aplicación cargada El micrófono así como la configuración del sonido es correcta
Postcondiciones	Una vez dicha la frase el sistema comenzará el movimiento citado

Flujo Normal	El usuario dice “Manuel corre” El avatar comienza a moverse por el escenario y a correr El usuario dice “Manuel para” El avatar se detiene al procesar la última orden.
Flujo Alternativo	
Excepciones	El usuario cita la frase correcta pero interpreta otra. El avatar actuará en consecuencia a la frase decodificada. El usuario cita la frase correcta pero falla la decodificación. El avatar no hace nada
Inclusiones	
Prioridad	Alta
Frecuencia de uso	Alta
Reglas de Negocio	
Requerimientos Especiales	
Asunciones	
Notas	

8.1.1.2 Acción Giro

Caso de uso ID	Vocalius-02
Nombre Caso de Uso	Acción giro
Actores	El actor que participa en el caso de uso es el jugador.
Descripción	Este caso de uso permitirá al usuario enviar órdenes al avatar como girar a la izquierda o gira a la derecha
Disparador	El usuario pronuncia frases como "David gira izquierda", "Pablo gira derecha"
Precondiciones	El usuario está con la aplicación cargada El micrófono así como la configuración del sonido es correcta
Postcondiciones	Una vez dicha la frase el sistema comenzará el movimiento citado.

Flujo Normal	El usuario dice “Pedro gira izquierda” El avatar comienza a girar sobre sí mismo hasta hacer un giro de 90 grados.
Flujo Alternativo	
Excepciones	El usuario cita la frase correcta pero interpreta otra. El avatar actuará en consecuencia a la frase decodificada. El usuario cita la frase correcta pero falla la decodificación. El avatar no hace nada.
Inclusiones	
Prioridad	Alta
Frecuencia de uso	Alta
Reglas de Negocio	
Requerimientos Especiales	
Asunciones	El usuario está diciendo las frases en el idioma para el cual la aplicación está configurada.
Notas	-

8.1.1.3 Acción sobre rutas

Caso de uso ID	Vocalius-03
Nombre Caso de Uso	Acción ruta
Actores	El actor que participa en el caso de uso es el jugador.
Descripción	Este caso de uso permitirá al usuario enviar órdenes al avatar para ejecutar rutas de un lugar a otro.
Disparador	El usuario pronuncia frases como "David ve a la caja", "ve a la consola"
Precondiciones	El usuario está con la aplicación cargada El micrófono así como la configuración del sonido es correcta
Postcondiciones	Una vez dicha la frase el sistema comenzará el movimiento citado.

Flujo Normal	El usuario dice “Manuel ve caja” El avatar comienza a moverse por el escenario trazando una ruta en línea recta desde donde se encontraba hacia el objeto. El avatar se detiene un metro antes de colisionar con el objeto destino.
Flujo Alternativo	
Excepciones	El usuario cita la frase correcta pero interpreta otra. El avatar actuará en consecuencia a la frase decodificada. El usuario cita la frase correcta pero falla la decodificación. El avatar no hace nada. El usuario colisiona con un objeto al trazar la ruta.
Inclusiones	
Prioridad	Alta
Frecuencia de uso	Alta
Reglas de Negocio	
Requerimientos Especiales	
Asunciones	El usuario está diciendo las frases en el idioma para el cual la aplicación está configurada.

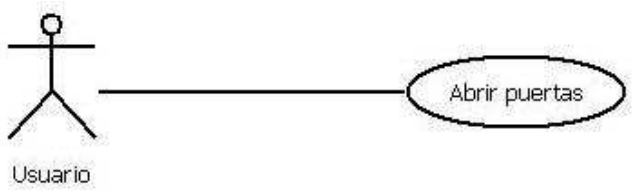
Notas	-

8.1.1.4 Acción mirar hacia los lados

Caso de uso ID	Vocalius-04
Nombre Caso de Uso	Acción cámara
Actores	El actor que participa en el caso de uso es el jugador.
Descripción	Este caso de uso permitirá al usuario enviar órdenes al avatar como girar cámara izquierda o girar cámara derecha.
Disparador	El usuario pronuncia frases como "Luis gira cámara izquierda", "Luis cámara derecha"
Precondiciones	El usuario está con la aplicación cargada El micrófono así como la configuración del sonido es correcta
Postcondiciones	Una vez dicha la frase el sistema comenzará el movimiento citado.

Flujo Normal	El usuario dice "Luis cámara derecha" El avatar quieto y la cámara empieza a rotar como si el avatar mirase a la derecha
Flujo Alternativo	
Excepciones	El usuario cita la frase correcta pero interpreta otra. El avatar actuará en consecuencia a la frase decodificada. El usuario cita la frase correcta pero falla la decodificación. El avatar no hace nada.
Inclusiones	
Prioridad	Alta
Frecuencia de uso	Alta
Reglas de Negocio	
Requerimientos Especiales	
Asunciones	El usuario está diciendo las frases en el idioma para el cual la aplicación está configurada.
Notas	-

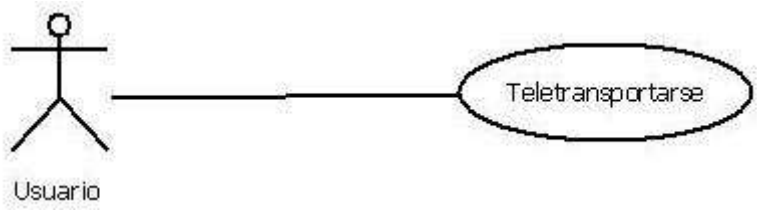
8.1.1.5 Acción abrir puertas



Caso de uso ID	Vocalius-05
Nombre Caso de Uso	Abrir puerta
Actores	El actor que participa en el caso de uso es el jugador.
Descripción	Este caso de uso permitirá al usuario enviar órdenes al avatar como abrir puertas.
Disparador	El usuario pronuncia frases como "abre puerta"
Precondiciones	El usuario está con la aplicación cargada El micrófono así como la configuración del sonido es correcta
Postcondiciones	Una vez dicha la frase el sistema comenzará el movimiento citado.
Flujo Normal	El usuario dice "abre puerta" La entidad puerta encontrada en el escenario se abrirá
Flujo Alternativo	

Excepciones	El usuario cita la frase correcta pero interpreta otra. El avatar actuará en consecuencia a la frase decodificada. El usuario cita la frase correcta pero falla la decodificación. El avatar no hace nada.
Inclusiones	
Prioridad	Alta
Frecuencia de uso	Alta
Reglas de Negocio	
Requerimientos Especiales	
Asunciones	El usuario está diciendo las frases en el idioma para el cual la aplicación está configurada.
Notas	-

8.1.1.6 Teletransportarse



Caso de uso ID	Vocalius-06
Nombre Caso de Uso	Teletransporte
Actores	El actor que participa en el caso de uso es el jugador.
Descripción	Este caso de uso permitirá al usuario enviar órdenes al avatar para que se teletransporte
Disparador	El usuario pronuncia frases como "ve teletransportador"
Precondiciones	El usuario está con la aplicación cargada El micrófono así como la configuración del sonido es correcta
Postcondiciones	Una vez dicha la frase el sistema comenzará el movimiento citado.

Flujo Normal	El usuario dice “ve teletransportador” El avatar comienza a moverse por el escenario en dirección al teletransportador Se introduce en él y ejecuta la acción de teletransportarse.
Flujo Alternativo	
Excepciones	El usuario cita la frase correcta pero interpreta otra. El avatar actuará en consecuencia a la frase decodificada. El usuario cita la frase correcta pero falla la decodificación. El avatar no hace nada.
Inclusiones	
Prioridad	Alta
Frecuencia de uso	Alta
Reglas de Negocio	
Requerimientos Especiales	
Asunciones	El usuario está diciendo las frases en el idioma para el cual la aplicación está configurada.
Notas	-

8.1.2 Diseño

8.1.2.1 VocaliusLib

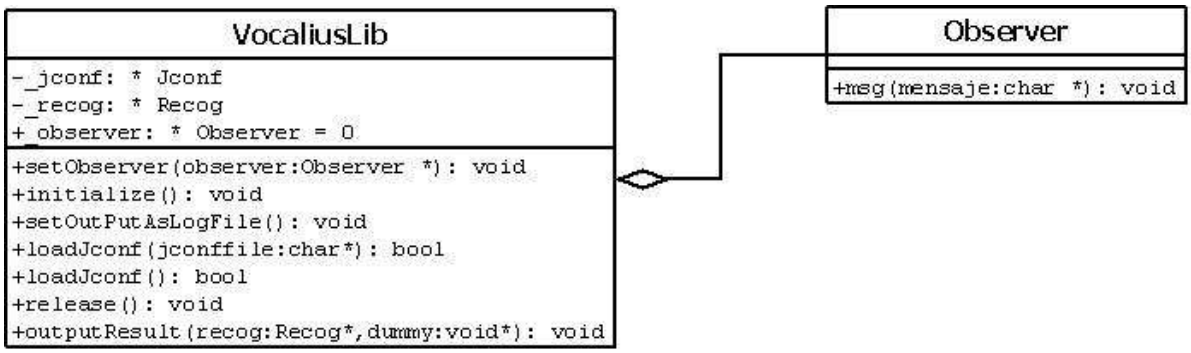
Para el diseño de nuestro sistema lo primero que tuvimos que hacer es abstraer la funcionalidad del reconocedor.

Esto supuso creación de una librería que proporcionase una interfaz unificada a la funcionalidad del reconocedor, simplificando métodos como inicialización y recoger el resultado. Todo esto se logró aplicando un patrón *Facade*. Así simplificaremos todo el sistema de llamadas de manera que con tan solo un par de invocaciones a métodos tengamos el reconocedor funcionando.

Dado que el reconocedor tiene muchas opciones, lo simplificaremos al máximo, descartando funcionalidades como la entrada por fichero o por red. También abstraeremos la configuración del reconocedor de manera que se pueda hacer todo desde un fichero de configuración externo.

Otra decisión de diseño fue la aplicación del patrón *Observer* para recoger el resultado del reconocimiento. Esta decisión facilita la reutilización y migración de la librería a otros proyectos. El encargado de tratar la salida tan solo tendrá que establecerse como observador y, de este modo, recogerá el resultado del reconocimiento y efectuará las operaciones que considere. Esto nos vino especialmente bien para migrar la librería al sistema Javy2 ya que solo teníamos que integrar la librería y establecer como Observador la clase que queríamos que se encargase de tratar el resultado.

De este modo, y con las consideraciones antes mencionadas, el diseño del reconocedor queda de la siguiente forma:



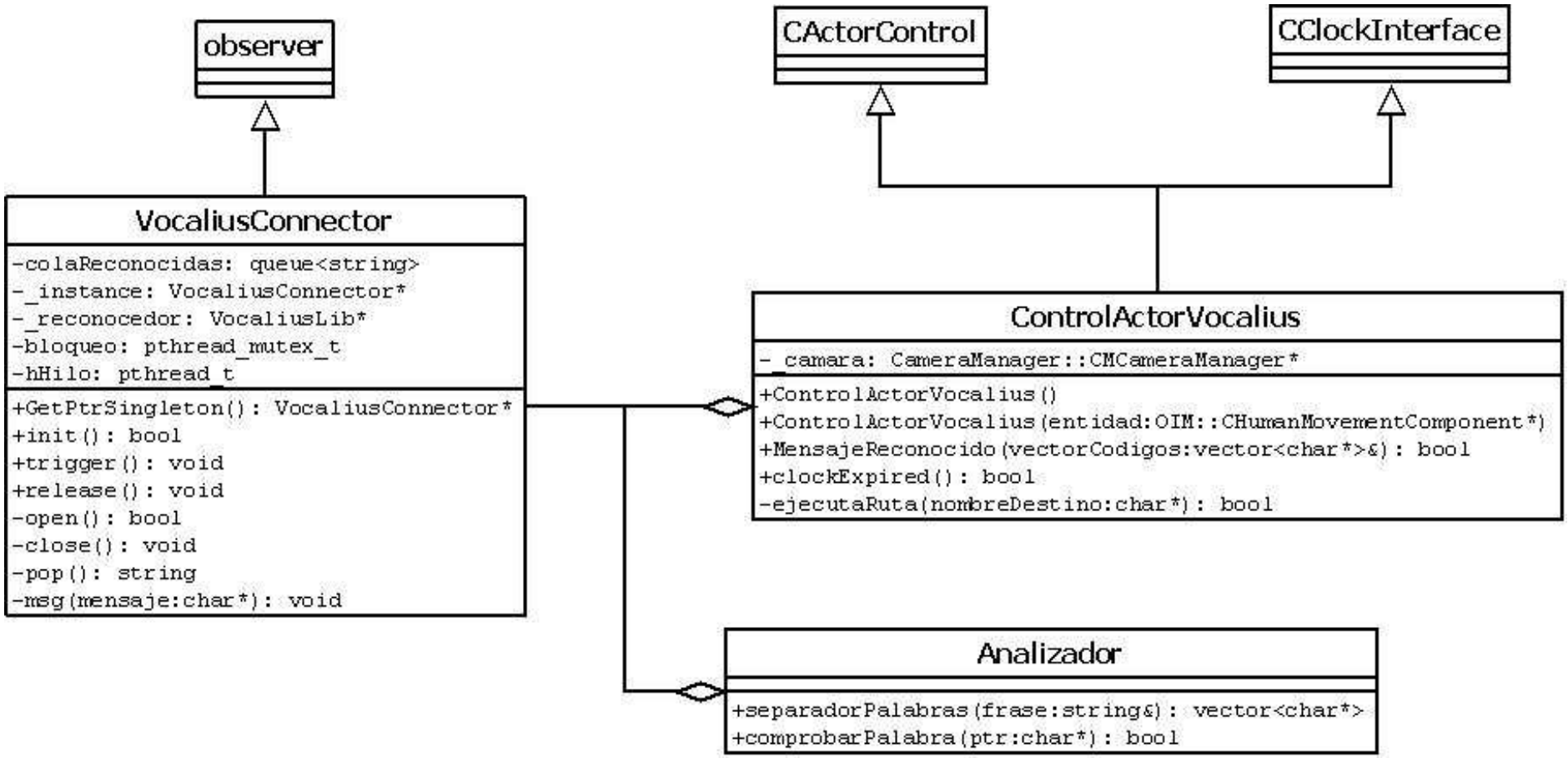
8.1.2.2 Vocalius

Todo esto se inserta en un subproyecto del videojuego citado que formará, de por sí, un videojuego independiente.

Se diseña una clase conectora que será encargada de vincular la librería con el videojuego. La librería con el reconocedor se ejecuta en un hilo aparte, por lo que se debe tener especial cuidado con la concurrencia. Esto anterior implicó la creación de una cola, con las palabras reconocidas. El reconocedor actuará como productor, mientras que el juego actuará como consumidor. Por supuesto la cola tiene que estar protegida por un semáforo.

Para la decodificación de las frases reconocidas el sistema realizará una conversión a ciertos códigos. De este modo hay un código para cada acción. Correr tendrá su código, andar, otro distinto. Esto hace necesario un analizador léxico que se encargue de tokenizar las frases reconocidas con los códigos específicos. Este analizador se encarga también de desechar palabras que carecen de significado para el sistema, como pueden ser determinantes, artículos, etc.

Finalmente es necesario conectar todo el sistema de códigos con la aplicación y por consiguiente con el sistema de control del avatar. Esto se logra creando una nueva clase que provea al sistema del nuevo comportamiento y que a su vez mantenga el comportamiento que el sistema ya tenía, para entrada por teclado y ratón.


observer
CActorControl
CClockInterface
VocaliusConnector
-colaReconocidas: queue<string>
-_instance: VocaliusConnector*
-_reconocedor: VocaliusLib*
-bloqueo: pthread_mutex_t
-hHilo: pthread_t
+GetPtrSingleton(): VocaliusConnector*
+init(): bool
+trigger(): void
+release(): void
-open(): bool
-close(): void
-pop(): string
-msg(mensaje:char*): void
ControlActorVocalius
-_camara: CameraManager::CMCameraManager*
+ControlActorVocalius()
+ControlActorVocalius(entidad:OIM::CHumanMovementComponent*)
+MensajeReconocido(vectorCodigos:vector<char*>&): bool
+clockExpired(): bool
-ejecutaRuta(nombreDestino:char*): bool
Analizador
+separadorPalabras(frase:string&): vector<char*>
+comprobarPalabra(ptr:char*): bool

8.1.3 Implementación

8.1.3.1 VocaliusLib

Para la creación de la librería se utilizo, inicialmente, Visual Studio 2008. Una vez desarrollada toda la librería fue necesario un cambio de versión a una anterior para facilitar la integración con el proyecto Javy y el *CruiseControl*.

Decir antes de nada que la versión de *Julius* sobre la que trabajamos es del 25 de diciembre del 2009 y solo en la versión anterior, del 2 de noviembre del mismo año se comenzó a dar soporte para el visual Studio. Esto hace una idea de la singularidad del empleo de esta librería. Las diferencias entre ambas distribuciones son la corrección de errores para este IDE.

8.1.3.2 Vocalius

Como ya hemos dicho, para este sistema creamos un subproyecto dentro del proyecto Javy empleando el esqueleto y estructura de este. Al hacerlo en un subproyecto pudimos modificar ciertos comportamientos y clase, siempre dentro de nuestro proyecto, para implementar lo que necesitábamos.

8.2 Sistema de entrenamiento del modelo acústico

Para facilitar el entrenamiento del modelo acústico y por tanto mejorar las tasas de acierto a la hora de interpretar las frases dichas por el usuario por parte del avatar, hemos puesto a disposición de todo el mundo una herramienta que permite realizar grabaciones de voz de una serie de frases que contienen palabras de la gramática del lenguaje que se utiliza en el juego.

Otra funcionalidad que hemos integrado dentro de esta herramienta en la capacidad de descargar el modelo acústico del lenguaje en su versión más completa, y la automatización del proceso de generación del modelo, actualizando el actual con las nuevas grabaciones que hayan sido almacenadas en el servidor, de cara a obtener la versión más reciente y poder utilizar dicha versión en el juego.

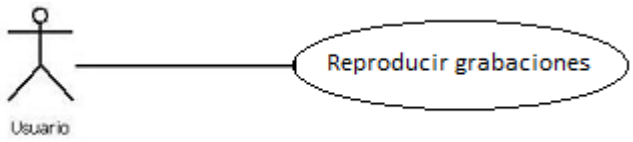
8.2.1 Especificación

8.2.1.1 Grabación de frases

Caso de uso ID	Entrenamiento-01
Nombre Caso de Uso	Grabación de frases
Actores	El actor que participa en el caso de uso es el usuario.
Descripción	Este caso de uso permitirá al usuario realizar las grabaciones de frases de cara al entrenamiento del modelo acústico.
Disparador	El usuario pulsa sobre el botón de grabar
Precondiciones	El usuario tiene conexión a internet El usuario tiene el micrófono de su equipo correctamente configurado.

Postcondiciones	Se guarda la grabación cuando el usuario pulsa sobre el botón de parar
Flujo Normal	El usuario pulsa sobre el botón de grabar El usuario pronuncia la frase en cuestión Cuando el usuario pulsa sobre el botón de parar se guarda dicha grabación
Flujo Alternativo	
Excepciones	
Inclusiones	
Prioridad	Alta
Frecuencia de uso	Alta
Reglas de Negocio	
Requerimientos Especiales	
Asunciones	El usuario está leyendo la frase asociada al botón de grabación que ha pulsado.
Notas	-

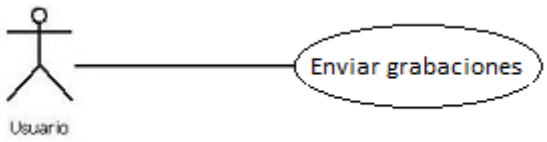
8.2.1.2 Reproducir Grabaciones



Caso de uso ID	Entrenamiento-02
Nombre Caso de Uso	Reproducir grabaciones
Actores	El actor que participa en el caso de uso es el usuario.
Descripción	Este caso de uso permitirá al usuario escuchar las grabaciones que haya realizado.
Disparador	El usuario pulsa sobre el botón de reproducir
Precondiciones	El usuario tiene conexión a internet El usuario tiene los altavoces de su equipo correctamente configurados. Por lo menos ha realizado una grabación.
Postcondiciones	El usuario escuchará la grabación asociada al botón de reproducir que haya pulsado.
Flujo Normal	El usuario pulsa sobre el botón de reproducir. El usuario escucha la correspondiente grabación
Flujo Alternativo	

Excepciones	
Inclusiones	
Prioridad	Alta
Frecuencia de uso	Alta
Reglas de Negocio	
Requerimientos Especiales	
Asunciones	
Notas	-

8.2.1.3 Envío de grabaciones



Caso de uso ID	Entrenamiento-03
Nombre Caso de Uso	Envío de grabaciones.
Actores	El actor que participa en el caso de uso es el usuario.

Descripción	Este caso de uso permitirá al usuario enviar al servidor las grabaciones que haya realizado.
Disparador	El usuario pulsa sobre el botón de enviar
Precondiciones	El usuario tiene conexión a internet El usuario ha realizado por lo menos una grabación
Postcondiciones	Se almacenan en el servidor las grabaciones que ha realizado el usuario junto con un fichero de texto en el que estarán las frases grabadas por el usuario.
Flujo Normal	El usuario pulsa sobre el botón de enviar Se almacena en el servidor un fichero zip, que contiene los ficheros de audio y un fichero de texto en el que están las frases que ha sido grabadas por el usuario.
Flujo Alternativo	
Excepciones	
Inclusiones	
Prioridad	Alta
Frecuencia de uso	Alta
Reglas de Negocio	

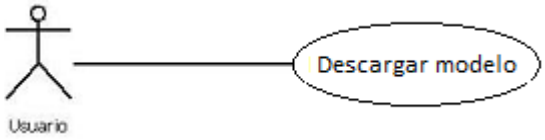
Requerimientos Especiales	
Asunciones	El usuario ha realizado por lo menos una grabación.
Notas	-

8.2.1.4 Generación del modelo

Caso de uso ID	Entrenamiento-04
Nombre Caso de Uso	Generación del modelo
Actores	El actor que participa en el caso de uso es el usuario.
Descripción	Este caso de uso permitirá al usuario generar de manera automática el modelo acústico, teniendo en cuentas las grabaciones nuevas que se encuentren en ese momento en el servidor
Disparador	El usuario pulsa sobre el botón de generar modelo
Precondiciones	El usuario tiene conexión a internet.

Postcondiciones	Se guarda la grabación cuando el usuario pulsa sobre el botón de parar
Flujo Normal	El usuario pulsa sobre el botón de generar modelo En la parte del servidor se generará un nuevo modelo acústico teniendo en cuenta las nuevas grabaciones que se encuentren almacenadas en el servidor.
Flujo Alternativo	
Excepciones	
Inclusiones	
Prioridad	Alta
Frecuencia de uso	Alta
Reglas de Negocio	
Requerimientos Especiales	
Asunciones	
Notas	-

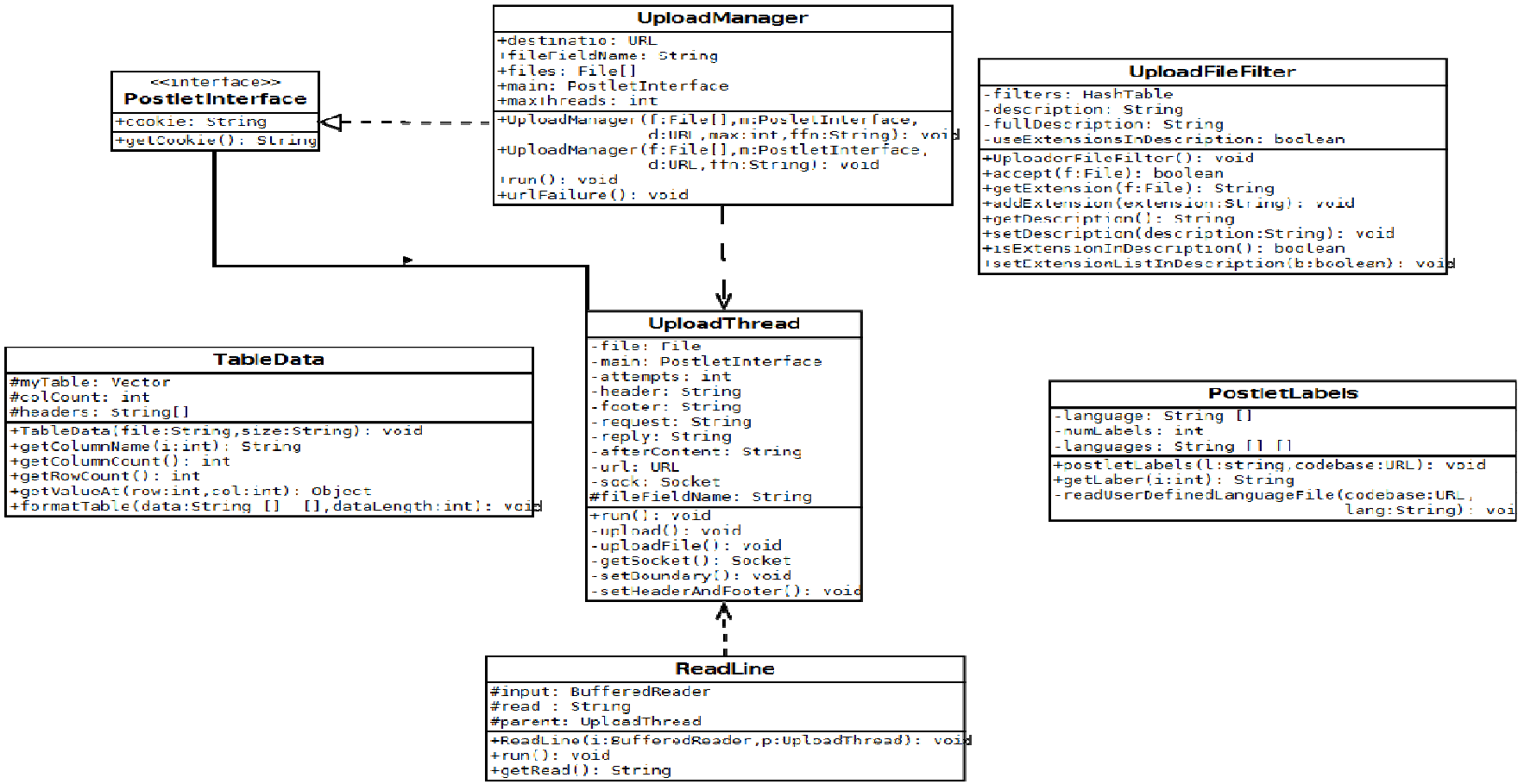
8.2.1.5 Descarga del modelo

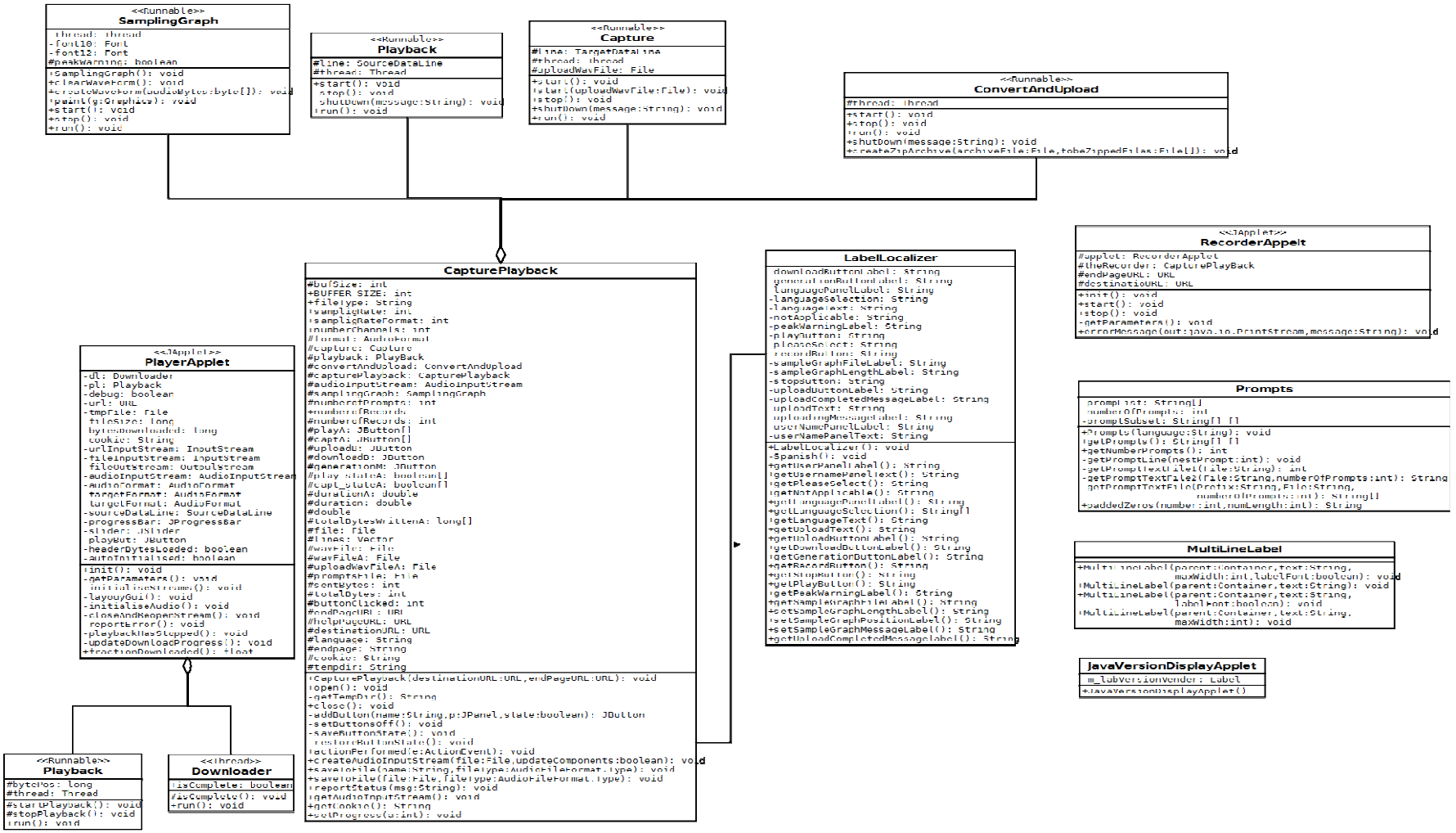


Caso de uso ID	Entrenamiento-05
Nombre Caso de Uso	Descarga del modelo
Actores	El actor que participa en el caso de uso es el usuario.
Descripción	Este caso de uso permitirá al usuario descargar el modelo acústico generado más reciente que se encuentre en el servidor
Disparador	El usuario pulsa sobre el botón de descargar modelo
Precondiciones	El usuario tiene conexión a internet.
Postcondiciones	Al final del proceso el usuario obtiene los ficheros del modelo acústico.
Flujo Normal	El usuario pulsa sobre el botón de descargar modelo Guarda el modelo acústico que estaba en el servidor en su equipo.
Flujo Alternativo	
Excepciones	

Inclusiones	
Prioridad	Alta
Frecuencia de uso	Alta
Reglas de Negocio	
Requerimientos Especiales	
Asunciones	
Notas	-

8.2.2 Diseño


UploadManager
+destinatio: URL
-fileFieldName: String
+files: File[]
+main: PostletInterface
+maxThreads: int
+UploadManager(f:File[],m:PostletInterface,
d:URL,max:int,ffn:String): void
+UploadManager(f:File[],m:PostletInterface,
d:URL,ffn:String): void
-run(): void
+urlFailure(): void
<<interface>>
PostletInterface
+cookie: String
+getCookie(): String
UploadFileFilter
-filters: HashTable
-description: String
-fullDescription: String
-useExtensionsInDescription: boolean
+UploadorFileFilter(): void
+accept(f:File): boolean
+getExtension(f:File): String
+addExtension(extension:String): void
+getDescription(): String
+setDescription(description:String): void
+isExtensionInDescription(): boolean
+setExtensionListInDescription(b:boolean): void
UploadThread
-file: File
-main: PostletInterface
-attempts: int
-header: String
-footer: String
-request: String
-reply: String
-afterContent: String
-url: URL
-sock: Socket
#fileFieldName: String
+run(): void
-upload(): void
-uploadFile(): void
-getSocket(): Socket
-setBoundary(): void
-setHeaderAndFooter(): void
TableData
#myTable: Vector
#colCount: int
#headers: string[]
+TableData(file:String,size:String): void
+getColumnName(i:int): String
+getColumnCount(): int
+getRowCount(): int
+getValueAt(row:int,col:int): Object
+formatTable(data:String [] [],dataLength:int): void
PostletLabels
-language: String []
-numLabels: int
-languages: String [] []
+postletLabels(l:string,codebase:URL): void
+getLaber(i:int): String
-readUserDefinedLanguageFile(codebase:URL,
lang:String): void
ReadLine
#input: BufferedReader
#read : String
#parent: UploadThread
+ReadLine(i:BufferedReader,p:UploadThread): void
+run(): void
+getRead(): String


«Runnable»
SamplingGraph
-thread: Thread
-font10: Font
-font12: Font
#peakWarning: boolean
+SamplingGraph(): void
+clearWaveform(): void
+createWaveform(audioBytes:byte[]): void
+paint(g:Graphics): void
+start(): void
+stop(): void
+run(): void
«Runnable»
Playback
#line: SourceDataLine
#thread: Thread
+start(): void
+stop(): void
+shutDown(message:String): void
+run(): void
«Runnable»
Capture
#line: TargetDataLine
#thread: Thread
#uploadWavFile: File
+start(): void
+start(uploadWavFile:File): void
+stop(): void
+shutDown(message:String): void
+run(): void
«Runnable»
ConvertAndUpload
#thread: Thread
+start(): void
+stop(): void
+run(): void
+shutDown(message:String): void
+createZipArchive(archiveFile:File,toBeZippedFiles:File[]): void
«JApplet»
RecorderApplet
#applet: RecorderApplet
#theRecorder: CapturePlayBack
#endPageURL: URL
#destinationURL: URL
+init(): void
+start(): void
+stop(): void
-getParameters(): void
+errorMessage(out:java.io.PrintStream,message:String): void
«JApplet»
PlayerApplet
-dl: Downloader
-pl: Playback
-debug: boolean
-url: URL
-tmpFile: File
fileSize: long
bytesDownloaded: long
cookie: String
-urlInputStream: InputStream
-fileInputStream: InputStream
fileOutStream: OutputStream
-audioInputStream: AudioInputStream
-audioFormat: AudioFormat
targetFormat: AudioFormat
targetFormat: AudioFormat
-sourceDataLine: SourceDataLine
-progressBar: JProgressBar
-slider: JSlider
playBut: JButton
-headerBytesLoaded: boolean
-autoInitialised: boolean
+init(): void
-getParameters(): void
-initialiseStream(): void
-layoutGui(): void
-initialiseAudio(): void
-closeAndReopenStream(): void
reportError(): void
-playbackHasStopped(): void
-updateDownloadProgress(): void
+fractionDownloaded(): float
CapturePlayback
#bufSize: int
+BUFFER_SIZE: int
+fileType: String
+samplingRate: int
+samplingRateFormat: int
numberChannels: int
#format: AudioFormat
#capture: Capture
#playback: PlayBack
#convertAndUpload: ConvertAndUpload
#capturePlayback: CapturePlayback
#audioInputStream: AudioInputStream
#samplingGraph: SamplingGraph
#numberOfPrompts: int
+numberOfRecords
#numberOfRecords: int
#playA: JButton[]
#captA: JButton[]
#uploadB: JButton
#downloadB: JButton
#generationM: JButton
#playStateA: boolean[]
#captStateA: boolean[]
#durationA: double
#duration: double
#double
#totalBytesWrittenA: long[]
#file: File
#lines: Vector
#waveFile: File
#wavFileA: File
#uploadWavFileA: File
#promptsFile: File
#contBytes: int
#totalBytes: int
#buttonClicked: int
#endPageURL: URL
#helpPageURL: URL
#destinationURL: URL
#language: String
#endpage: String
#cookie: String
#tempDir: String
+CapturePlayback(destinationURL:URL,endPageURL:URL): void
+open(): void
-getTempDir(): String
+close(): void
-addButton(name:String,p:JPanel,state:boolean): JButton
-setButtonsOff(): void
-saveButtonState(): void
restoreButtonState(): void
+actionPerformed(e:ActionEvent): void
+createAudioInputStream(file:File,updateComponents:boolean): void
+saveToFile(name:String,fileType:AudioFileFormat.Type): void
+saveToFile(file:File,fileType:AudioFileFormat.Type): void
+reportStatus(msg:String): void
+getAudioInputStream(): void
+getCookie(): String
+setProgress(n:int): void
LabelLocalizer
downloadButtonLabel: String
generationButtonLabel: String
languagePanelLabel: String
-languageSelection: String
-languageText: String
-notApplicable: String
-peakWarningLabel: String
-playButton: String
pleaseSelect: String
recordButton: String
-sampleGraphFileLabel: String
-sampleGraphLengthLabel: String
-stopButton: String
-uploadButtonLabel: String
-uploadCompletedMessageLabel: String
uploadText: String
uploadingMessageLabel: String
userNamePanelLabel: String
-userNamePanelText: String
+LabelLocalizer(): void
-Spanish(): void
+getUserPanelLabel(): String
+getusernamePanelText(): String
+getPleaseSelect(): String
+getNotApplicable(): String
+getLanguagePanelLabel(): String
+getLanguageSelection(): String[]
+getLanguageText(): String
+getUploadText(): String
+getUploadButtonLabel(): String
+getDownloadButtonLabel(): String
+getGenerationButtonLabel(): String
+getRecordButton(): String
+getStopButton(): String
+getPlayButton(): String
+getPeakWarningLabel(): String
+getSampleGraphFileLabel(): String
+getSampleGraphLengthLabel(): String
+setSampleGraphPositionLabel(): String
+setSampleGraphMessageLabel(): String
+getUploadCompletedMessageLabel(): String
Prompts
promptList: String[]
numberOfPrompts: int
-promptSubset: String[][]
+Prompts(language:String): void
+getPrompts(): String[][]
+getNumberPrompts(): int
-getPromptLine(nestPrompt:int): void
-getPromptTextFile1(file:String): int
-getPromptTextFile2(file:String,numberOfPrompts:int): String
getPromptTextFile(prefix:String,fileString,
numberOfPrompts:int): String[]
+paddedZeros(number:int,numLength:int): String
MultiLineLabel
+MultiLineLabel(parent:Container,text:String,
maxWidth:int,labelFont:boolean): void
+MultiLineLabel(parent:Container,text:String): void
+MultiLineLabel(parent:Container,text:String,
labelFont:boolean): void
+MultiLineLabel(parent:Container,text:String,
maxWidth:int): void
JavaVersionDisplayApplet
m_labVersionVendor: Label
+JavaVersionDisplayApplet()
«Runnable»
Playback
#bytePos: long
#thread: Thread
#startPlayback(): void
#stopPlayback(): void
+run(): void
«Thread»
Downloader
fileComplete: boolean
fileComplete(): void
+run(): void

8.2.3 Implementación

Para la implementación del *applet* para la automatización del proceso de generación del modelo y de entrenamiento hemos utilizado Eclipse. El cual nos permitía el desarrollo de aplicaciones web, y con el que más habituados nos encontrábamos.

9 Conclusiones y líneas de trabajo futuras

El proyecto nos ha servido para desarrollar en gran medida una de las facetas más importantes de un ingeniero, la investigación, y que durante la carrera consideramos se relega a un segundo plano. Hemos investigado, estudiado y decidido entre una serie de tecnologías de reconocimiento de voz, sin tener ningún conocimiento previo acerca de este campo, sopesando los pros y los contras de cada una de las alternativas, para finalmente decidirnos por la que mejor se adaptaba a nuestro objetivo.

Consideramos que hemos obtenido un resultado aceptable teniendo en cuenta que la mayoría del tiempo que hemos dedicado al proyecto ha sido enfocado a la investigación como se ha comentado anteriormente, lo cual ha implicado que en varias ocasiones hayamos tenido que desechar el trabajo realizado sobre una tecnología ya que descubríamos que no podíamos seguir utilizándola, aún así hemos conseguido integrar el sistema de reconocimiento en un videojuego, y hemos montado una aplicación web que sirve para entrenar el modelo acústico.

Las principales líneas de trabajo futuras de este proyecto son seguir ampliando el modelo de lenguaje, y el modelo acústico para ir ampliando las posibilidades de interacción con el avatar, todas las acciones que introduzcamos en el modelo acústico, tendremos que diseñar su implementación o en caso de que estén implementadas seguramente se tengan que modificar, por ejemplo si existen varias cajas en el escenario, se deberá mostrar un número para poder hacer referencia a cada una de las cajas, y en caso de que se le indique al avatar que se mueve hacia una caja sepa a cual debe ir.

El control de videojuegos por voz es una alternativa, que poco a poco se está introduciendo en el mercado, hasta el momento no están

apostando fuerte por esta posibilidad pero consideramos que es un campo todavía por explotar, además existen otra gran cantidad de disciplinas en las cuales se puede aplicar el reconocimiento de voz, como por ejemplo enfocado a la ayuda de personas discapacitadas, para que puedan manejar equipos,…

10 Apéndice A: Manual de Usuario

En este punto de memoria vamos a explicar los distintos pasos que se deben de ir dando para la correcta utilización correcta tanto del sistema de reconocimiento por voz integrado en Javy como de la aplicación web para el entrenamiento del modelo acústico correspondiente a la gramática que se utiliza en el juego.

En primer lugar explicaremos como utilizar la herramienta que nos permite realizar las grabaciones a través de internet para el entrenamiento y la mejora del modelo acústico del que se sirve el juego para poder interactuar con el avatar a través de comandos de voz. Introducimos la dirección:

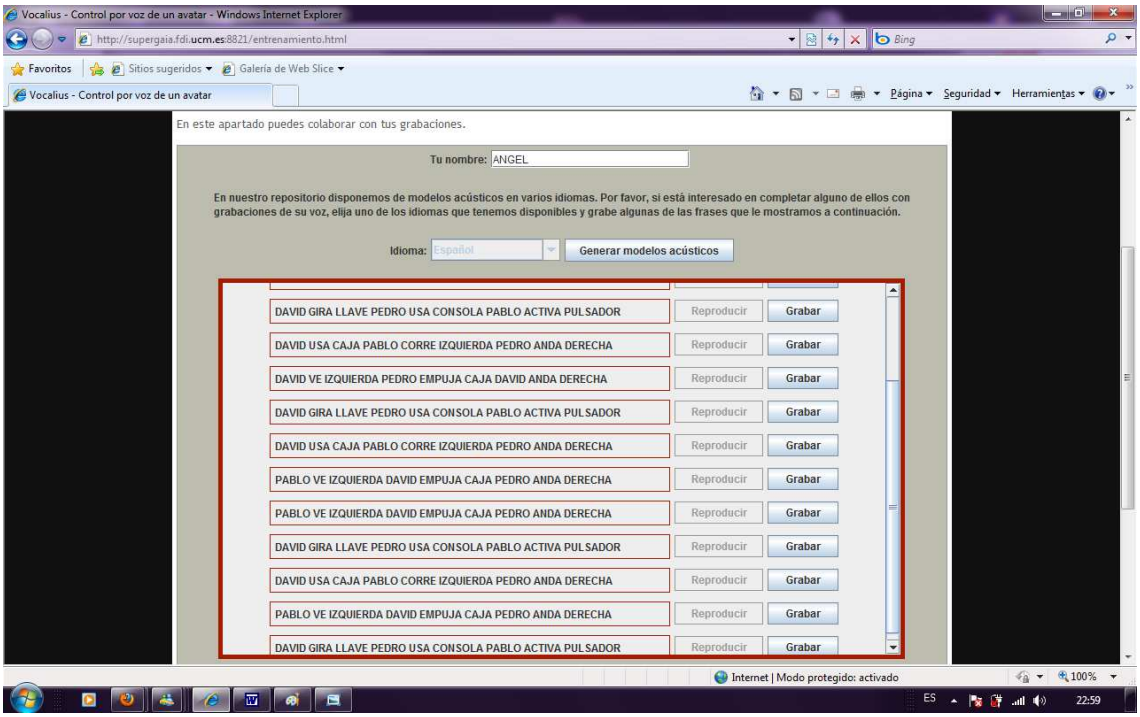
http://supergaia.fdi.ucm.es:8821 /index.html en el navegador de nuestro ordenador, nos cargará la página principal que hemos creado para el proyecto.

Una vez estamos en la página nos debemos dirigir a la parte de *Entrenamiento*, situada en el menú horizontal de la página, y

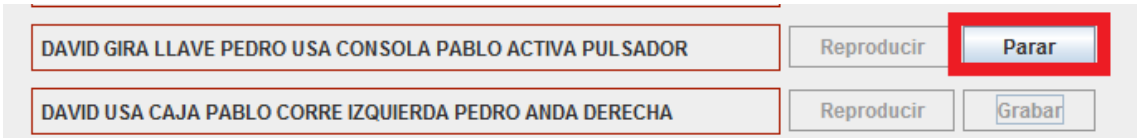
hacemos click sobre dicho menú.

Una vez pulsemos sobre el menú de *Entrenamiento* llegaremos a la página en la se podrán realizar las grabaciones para el posterior entrenamiento del modelo acústico, la pantalla que tendremos delante será la siguiente:

En el primer recuadro deberemos indicar el nombre del usuario que está aportando las grabaciones, y en el segundo al hacer click sobre el desplegable podremos seleccionar entre los idiomas español o inglés para realizar las grabaciones en cada uno de ellos respectivamente. Una vez seleccionemos uno de los idiomas se cargará la correspondiente lista de los prompts que podemos grabar, como podemos ver en la siguiente figura, lo más importante en esta parte es no realizar más de 10 grabaciones ya que sino el envío no se realiza con éxito.

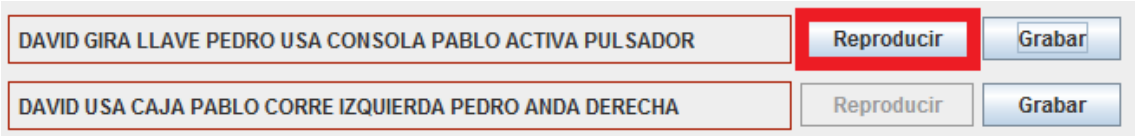


Una vez cargadas las frases solo nos queda realizar las grabaciones de aquellas frases que queramos para posteriormente enviarlas al servidor. Para ello utilizaremos los botones que se encuentran a la derecha de las frases, como vemos en la figura anterior, únicamente se encuentran habilitados los botones de *Grabar*, al pulsar sobre alguno de ellos, comenzaremos la grabación, el botón pasará a ser un botón de *Parar* con el objetivo de deterner la grabación en el momento en el que hayamos completado la lectura de la frase.



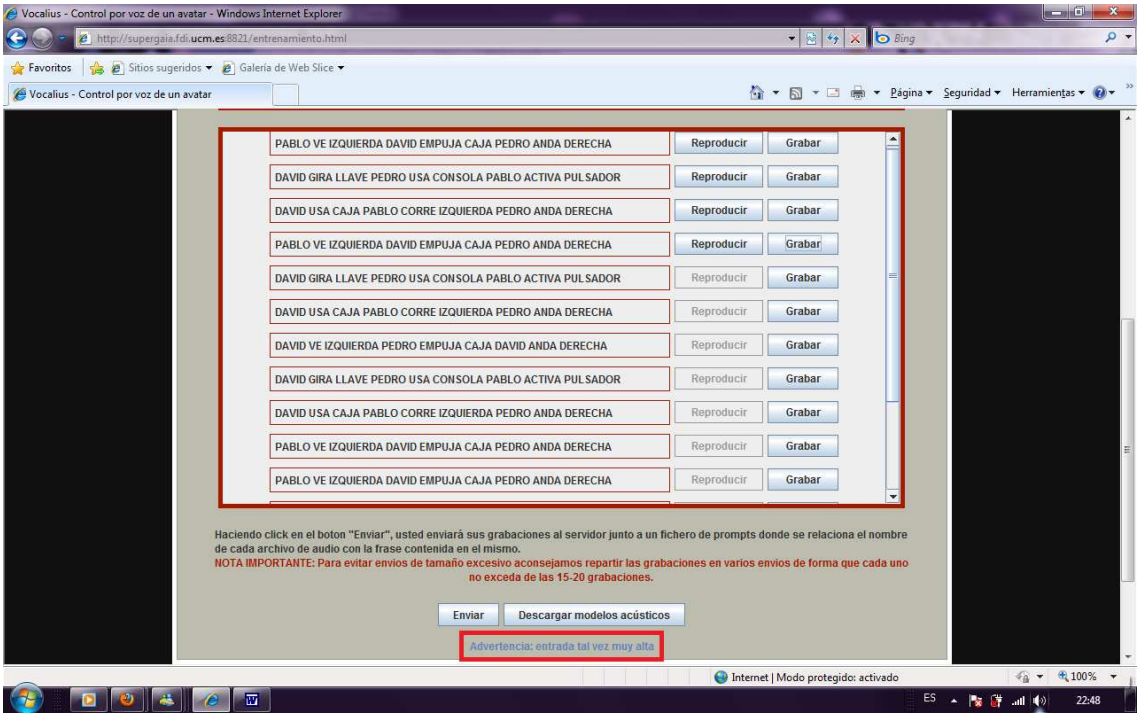
Hasta el momento el botón de *Reproducir* estaba deshabilitado, ya que no había ninguna grabación asociada a ninguna frase, pero cuando hayamos completado alguna grabación y hagamos click sobre el botón *Parar*, se habilitará el botón de *Reproducir* el cual nos permitirá escuchar la grabación que hayamos realizado, para detectar cualquier problema que haya podido ocurrir y para asegurarnos que efectivamente la grabación ha sido realizada correctamente, a

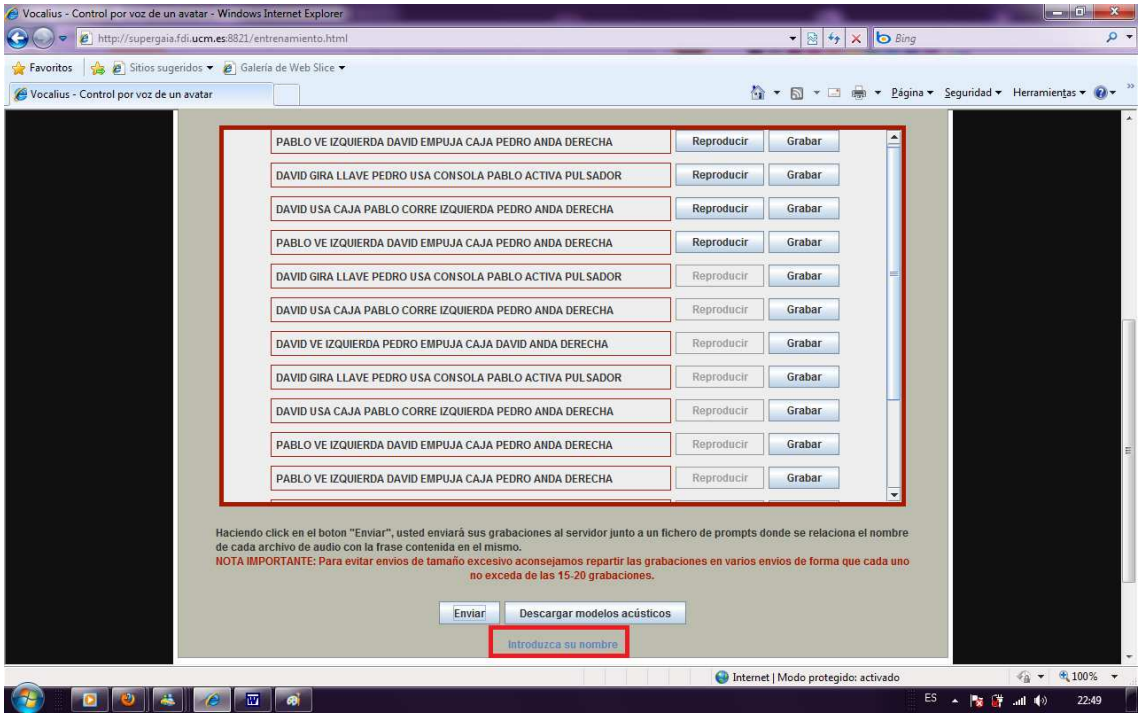
continuación mostramos una imagen con el botón de *Reproducir* activado.



En caso de que al reproducir detectemos que una grabación no es correcta, pulsando sobre el botón de *Grabar* comenzaremos una grabación nueva que machacará la anterior.

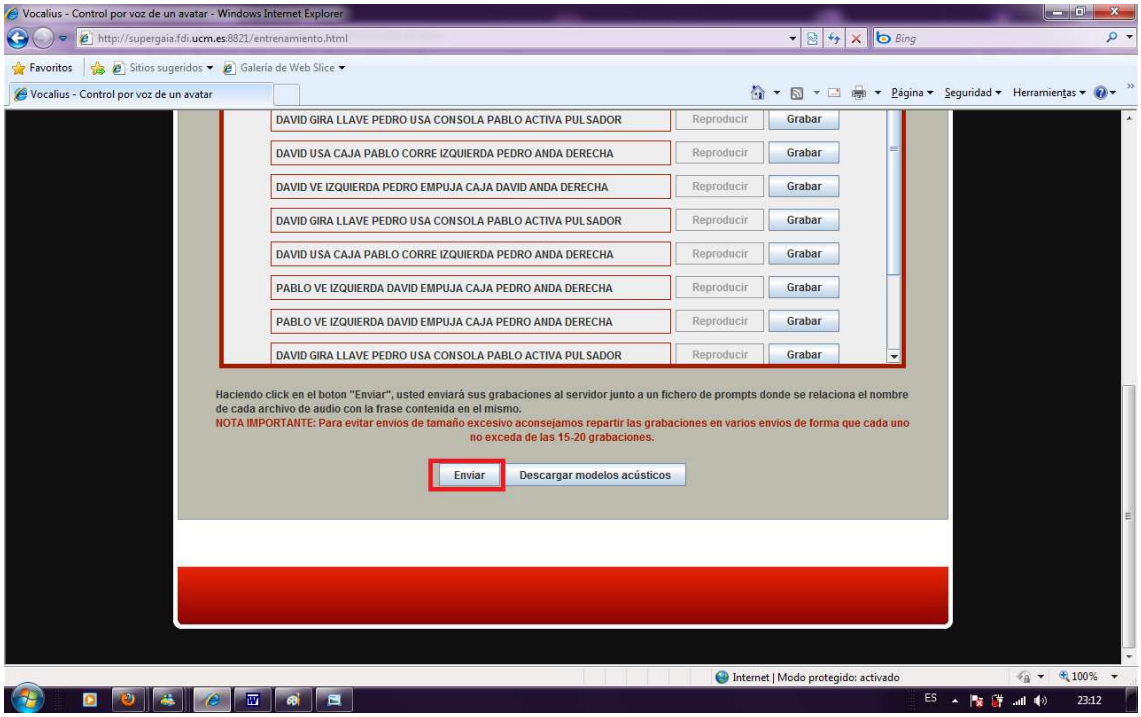
En caso de que realicemos una grabación en un tono demasiado elevado, o que se nos haya olvidado poner el nombre de usuario al principio, nos saldrán los siguientes mensajes avisándonos de dichos contratiempos, para que en el primer caso repitamos la grabación, y en el segundo caso escribamos el nombre de usuario que está realizando las grabaciones.





Justo debajo de las frases para las grabaciones se encuentran unas advertencias que conviene seguir, que nos indican que en caso de que queramos grabar varias frases, mandemos las correspodientes grabaciones al servidor en varias tandas nunca superando las 20 grabaciones, de cara a evitar un envío masivo al servidor.

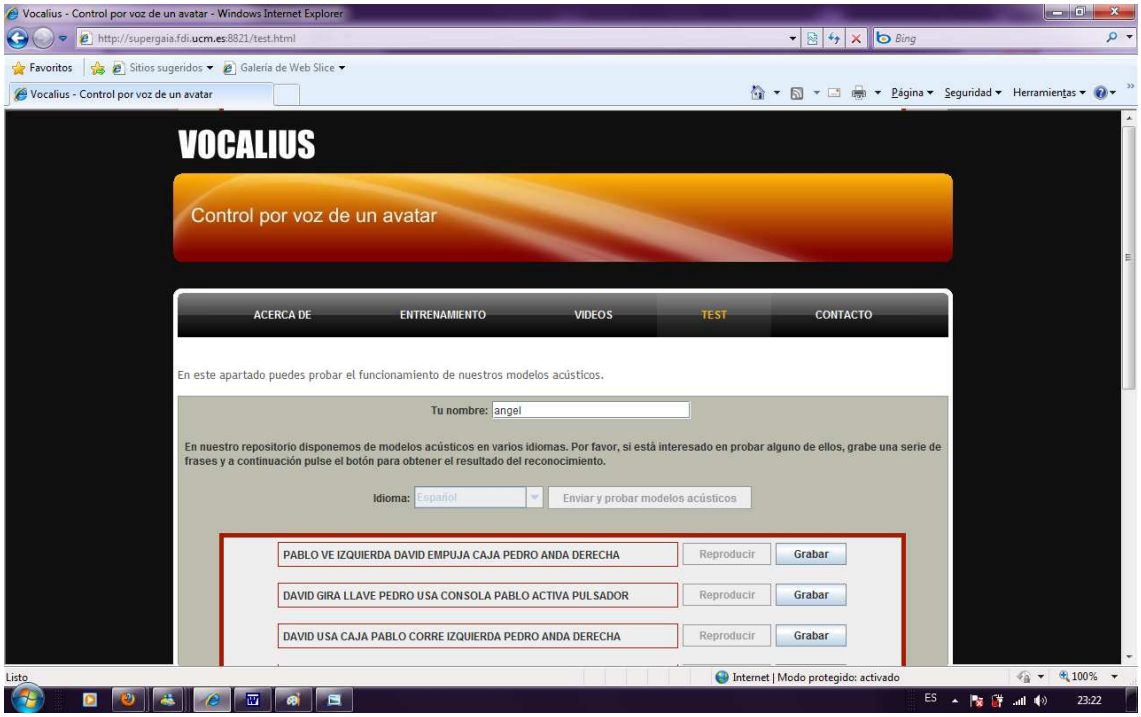
Una vez hayamos realizado las grabaciones correctas, con los campos correctamente cumplimentados, podremos pulsar sobre el botón *Enviar* el cual mandará dichas grabaciones al servidor donde serán almacenadas y posteriormente incorporadas en la siguiente generación del modelo acústico.



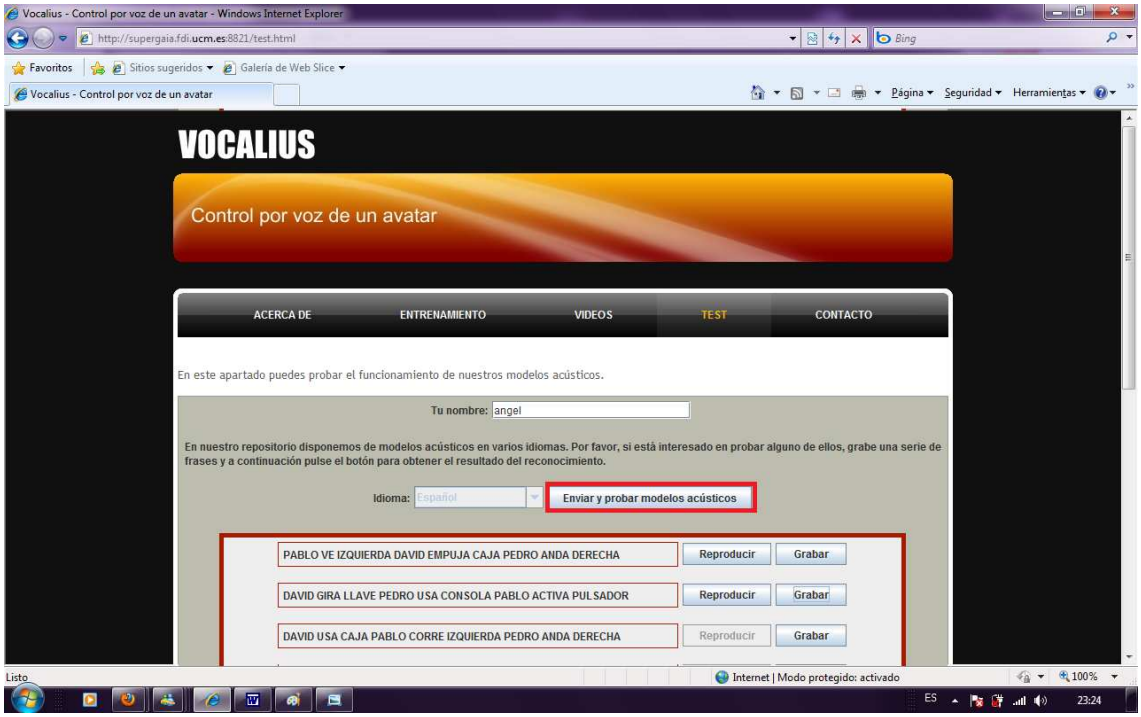
Y la última funcionalidad de esta parte de entrenamiento es la descarga de la última versión del modelo acústico, para ello pulsaremos sobre el botón de *Descarga modelo acústico*, que se encuenta a la derecha del botón enviar explicado anteriormente.

La otra funcionalidad que se encuentra en la página web se correponde con las pruebas del modelo acústico que se encuentra en el servidor, para ello nos deberemos dirigir en el menú horizontal a la pestana de *Test.*

Al pulsar sobre ese botón nos aparecerá una pantalla similar a la que aparecia al pulsar sobre el botón de *Entrenamiento* en la que tras poner el nombre de usuario y el idioma en el cual queremos realizar el test nos cargará una pantalla similiar a esta:



En ella nos aparecerán una serie de frases que deberemos leer, utilizando los botones de *Grabar, Parar* y *Reproducir*, explicados anteriormente, cuando hayamos realizado por lo menos una grabación se habilitará el botón *Enviar y probar modelos acústicos,* el cual se encargará de enviar las grabaciones al servidor y comprobar cuantas frases serían reconocidas correctamente con el modelo acústico actual.



Para la puesta en marcha del videojuego es necesario familiarizarse con la estructura de carpetas de Javy. Colgando del directorio principal encontramos diversas carpetas, de las que cabe destacar Exes, JavyAux, Proyects y src.

En Projects y src se encuentran los ficheros de programacion que permiten la funcionalidad de los diversos proyectos que dependen de Javy. En projects encontramos los archivos necesarios para cargar el proyecto en el IDE Visual Studio mientras que en src los ficheros de codigo fuente.

En la carpeta JavyAux tenemos los proyectos y librerias que son necesarias para las diversas aplicaciones que tenemos en el proyecto. En nuestro caso destacamos la librería Julius-4.1.4 que es la versión del reconocedor que se ha empleado en nuestro desarrollo. Dentro de esta, encontramos una carpeta llamada msvc. En ella tenemos todos los subproyectos que permiten al reconocedor funcionar. Tambien se

encuentra la carpeta Vocalius donde incluimos nuestra librería adaptada y la carpeta Resources.

En la carpeta Resources es donde podemos configurar nuestro reconocedor. Tenemos el fichero .jconf donde se pueden configurar una inmensa cantidad de parametros para el reconocedor. Tambien hay dos carpetas, Gramaticas y M_acusticos. En ellas podemos encontrar los modelos de lenguaje y acústicos, respectivamente. Especificar cúales se quiere cargar en un momento dado se escogerá en el fichero .jconf citado anteriormente.

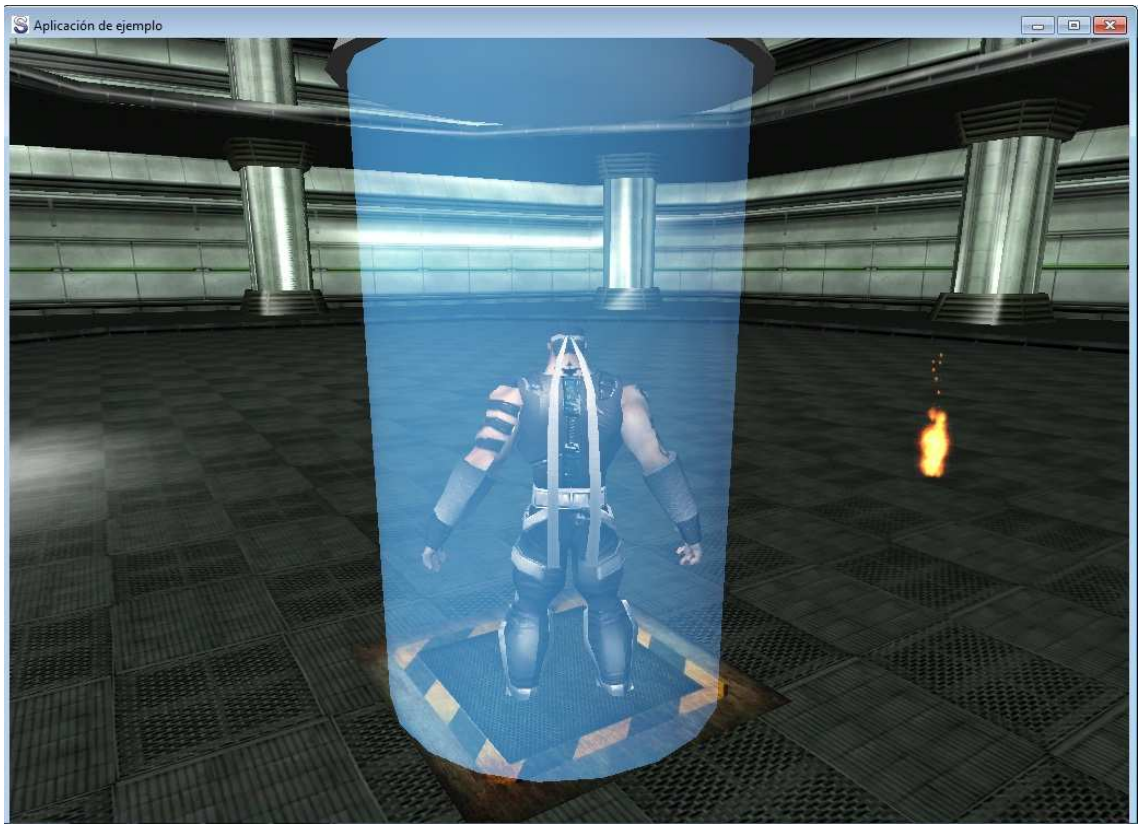
Finalmente en la carpeta Exes es donde podemos arrancar nuestro videojuego. Tiene la denominacion Vocalius.exe o VocaliusD.exe según se quiera arrancar la version para depurar o la version release del videojuego.

A continuación vemos unas capturas de pantalla del juego en el que está integrado el reconocedor.


Aplicación de ejemplo


Aplicación de ejemplo


Aplicación de ejemplo


Aplicación de ejemplo

11 Apéndice B: Manual de administrador

A continuación vamos a poner una serie de pauta a seguir para la correcta modificación de la parte de la aplicación web.

A la hora de modificar los *prompts* que son los ficheros que contienen las frases que mostrará el applet para que los usuarios realicen las distintas grabaciones, nos deberemos dirigir a la ruta */var/www/appletGen/speechrecorder/* en ella encontraremos todos los ficheros de prompts. A la hora de editar los siguientes ficheros hay que tener en cuenta lo siguiente:

- ✓ Frases deben tener las palabras en mayúsculas y sin acentos.
- ✓ Las frases deben tener un máximo de 10 palabras.
- ✓ Cada frase debe terminar sin ningún espacio al final.
- ✓ El fichero no debe contener un salto de línea al final del mismo.

En caso de querer modificar las frases de la aplicación de *Test* se deberá dirigir a la siguiente ruta */var/www/appletTest/speechrecorder*, para el fichero que se encuentra ahí deberá seguir las mismas pautas comentadas anteriormente.

El fichero *javaUploadServer.php* que se encuentra en la ruta */var/www/* sirve para una vez que el usuario haya enviado las grabaciones están se redirijan a la carpeta */usr/share/generador/enviosSinProcesar.*

El código html de la página se encuentra en la ruta */var/www/* en caso de que se quiera modificar algo de la apariencia de la página web.

En los archivos html donde están insertados los dos applets hay

una serie de parámetros dentro del código HTML como son por ejemplo la ubicación del archivo .php de subida de ficheros, el parámetro que indica la ruta en la que se encuentra el servlet que se deba llamar en cada caso, la ruta desde la que se descarga el fichero .zip con el modelo acústico generado.

A la hora de generar nuevos modelos debemos tener por lo menos una grabación en cada uno de los idiomas ya que sino al pulsar sobre el botón de generar modelo, intentará crear ficheros intermedios que necesitan esas grabaciones, y al realizar el proceso sobre todos los idiomas a la vez en caso de que no exista una grabación por lo menos de cada idioma e intentar acceder a esos ficheros intermedio dará un error y no finalizará el proceso correctamente.

Para la parte de los *servlets* el *tomcat* está montado en la ruta */usr/share/tomcat5.5*, los ficheros .*war* se encuentran en la carpeta *webapps* dentro de la ruta anteriormente comentada.

Para la generación de los modelos acústicos se debe seguir una estructura de carpetas como la que se encuentra en la ruta */usr/share/generador*. En caso de querer crear un modelo de cero, se deben mantener los ficheros que se encuentran en la raíz del anterior directorio que son los siguientes:

- ✓ config
- ✓ fixfullist.pl
- ✓ global.ded
- ✓ mkphones0.led
- ✓ mkphones1.led
- ✓ mktri.led

- ✓ proto
- ✓ sil.hed
- ✓ wav_config

También debe esta la carpeta *HTK_scripts* que contiene 4 ficheros que son:

- ✓ maketrihed
- ✓ mkclscript.prl
- ✓ prompts2mlf
- ✓ prompts2wlis

En la carpeta enviosSinProcesar se encontrarán las nuevas grabaciones nuevas que se hayan enviado desde la última generación del modelo acústico a través del applet de la página web.

En las carpetas *english_model* y *spanish_model* se deben borrar todos los ficheros excepto las siguientes carpetas:

- ✓ *language_model*: que contiene los ficheros correspondientes a la gramática.
- ✓ *lexicon*: que contiene el archivo *lexicon* correspondiente a cada lenguaje.
- ✓ *speechrecords*: que deberá estar vacía.
- ✓ speechrecords_mfcc: que deberá estar vacia.
- ✓ *hmms*: que contendrá 16 carpetas numeradas desde hmm0 hasta hmm15 que deberán estar presentes a la hora de generar el modelo.

- ✓ tree_original.hed: el fichero debe estar siempre en la carpeta /spanish_model/ y el equivalente para el inglés en la carpeta /english_model/.

12 Apéndice C: Pasos para la generación de un modelo acústico con Voxforge[1].

Con 16Khz de muestreo de 16 bits por el mismo, se tiene más información de audio con el que formar un modelo acústico con más precisión en el reconocimiento. Esto parece indicar que cuanto mayor es la frecuencia de muestreo, mayor será la tasa de reconocimiento que obtiene, pero, es necesario tener en cuenta el teorema de Nyquist:

La frecuencia máxima para una frecuencia de muestreo dado se llama la frecuencia de Nyquist. La mayoría de la información en el lenguaje humano es en frecuencias inferiores a 10.000 Hz, por lo que una frecuencia de muestre mayor resultaría innecesaria.

Lo primero que deberemos hacer es dirigirnos a la página de Voxforge[1], en nuestro caso el modelo acústico lo hemos desarrollado bajo el sistema operativo Linux, por lo que los pasos que vamos a comentar que hay que ir dando se refieren a este Sistema, también existe la posibilidad de generarlo desde Windows, y en la página de voxforge[1] hay una guía para aquellos que elijan esa opción, en nuestro caso nos decantamos por Linux ya que es más cómodo la ejecución de los distintos scripts, sin la necesidad de cygwin para que interprete las órdenes en perl.

Primero debemos preparar todo el entorno de desarrollo, para lo cual nos dirigimos a la siguiente dirección, desde la cual descargaremos los programas que comentaremos a continuación y que fueron explicados en profundidad en el apartado de tecnologías utilizadas.

HTK:

La licencia de HTK necesita que nos registremos en la página antes de poder descargar la herramienta. Es un software de código libre pero tiene restricciones en la distribución de la propia herramienta. Sin embargo no hay ninguna limitación en la distribución de los modelos que generemos con esta herramienta.

Después de seguir los pasos que nos indican para crear una estructura de carpetas determinada, la extracción de los ficheros en su lugar correspondiente para conseguir la estructura que nos indican, una de las cosas que más nos llamo la intención fue a la hora de compilar el código la necesidad de una versión específica del compilador de gcc, exactamente la versión 3.4, nosotros nos encontramos ante la situación de tener instalada en nuestros equipos la versión 4.0, por lo que tuvimos que investigar la manera de poder cambiar de una versión a otra.

Julius:

Es un motor de reconocimiento de voz (LVCSR). Julius no tiene limitaciones en la distribución, utiliza modelos acústicos en el formato de HTK y ficheros de gramáticas en su propio formato.

Julian es una versión especial de julius que nos permite comprobar si los ficheros que hemos generado con el HTK son válidos o no.

Audacity:

Es la herramienta con la cual realizaremos las grabaciones necesarias para la creación de un modelo acústico propio, se podría utilizar otro programa que nos permitiese realizar exportaciones a formato wav, pero se utiliza este por su fácil manejo y por lo intuitivo que es.

Una vez terminada la instalación de estos tres componentes podemos comenzar a dar los 10 pasos que debemos recorrer para conseguir crear nuestro propio modelo acústico.

Step 1- Task Grammar:

En este primer paso nos introducen en el ámbito de los sistemas de reconocimiento del habla explicándonos los componentes que forman este tipo de sistemas los cuales son:

LM (Language Model) o Grammar: Los modelos de lenguaje contienen una gran cantidad de palabras y su probabilidad de aparición en una secuencia determinada. Las gramáticas son ficheros mucho más pequeños con un conjunto de combinaciones de palabras ya predefinidas. Cada palabra de la gramática o del modelo de lenguaje está asociada a una lista de fonemas.

AM (Acoustic Model): Contiene una representación estadística para cada sonido que forma una palabra del LM o de la gramática.

Decoder: Es el programa que se encarga de recoger los sonidos de la locución y buscar en el AM los sonidos equivalentes para determinar el fonema al que corresponde ese sonido y de esta forma obtener la palabra correspondiente.

Con estos conceptos básicos claros podemos empezar a construir nuestra propia gramática. A continuación deberemos crear los siguientes ficheros:

.grammar: Este fichero define una serie de reglas para identificar el tipo de palabras que el sistema de reconocimiento de habla puede esperar recibir. Se utilizan "Categorías de Palabras" las cuales son definidas en el fichero .voca.

.voca: Este fichero define las "Palabras candidatas" dentro de cada

“Categoría de Palabras”, y la información de su pronunciación.

Una vez que sabemos para qué sirven cada uno de los ficheros que debemos generar nos dan la información necesaria para crearlos. Cuando ya tenemos ambos ficheros debemos utilizar un script de HTK para generar 3 ficheros más que son:

.dfa y .term: estos ficheros contienen la información del autómata finito.

.dict: Contiene la información de las palabras del diccionario.

Step 2 - Pronunciation Dictionnary:

En este paso lo que deberemos crear es un fichero en el cual aparezcan en varias líneas todas las palabras que tenemos en la gramática. El objetivo es conseguir que todos los fonemas que tenemos en nuestra gramática tengan un número de apariciones mínimo, ya que de no cumplirse esto luego se producirán fallos durante el proceso de reconocimiento, con el fin de obtener una gramática fonéticamente equilibrada se deberán incluir palabras para obtener como mínimo de 3 a 5 apariciones por fonema. Nos recomiendan desde voxforge[1] que como mínimo tengamos de 30-40 frases con 8-10 palabras cada una de ellas, (desde nuestra experiencia recomendamos que se realicen bastantes más grabaciones, como ejemplo citar que en nuestro primer modelo acústico tuvimos que realizar cerca de 240 grabaciones para conseguir una resultados aceptables, y se trataba de un gramática muy sencilla, además al realizar pruebas nos dimos cuenta que este número de grabaciones era válido si utilizaban el sistema aquellas personas que se habían encargado de realizar las grabaciones, en el momento en el que alguien que no hubiese realizado las grabaciones utilice el sistema las probabilidades de un reconocimiento correcto

disminuyen notablemente).

Step 3 - Recording the data:

En este paso lo que nos indican es la forma de realizar las grabaciones que posteriormente se unirán a las frases que anteriormente hemos creado.

Step 4 - Creating the Transcription Files:

En este paso lo que se hace es tratar cada una de las frases que escribimos y que posteriormente realizamos su grabación, de cara a un posterior tratamiento. Para ello gracias a los scripts que nos proporcionan, obtenemos al final unos ficheros en los cuales están desplegadas cada una de las frases que anteriormente creamos, en otro fichero en el cual en cada línea aparece una palabra, posteriormente se baja otro nivel para esas palabras descomponerlas en los fonemas que forman cada una de las mismas.

Step 5 - Coding the (Audio) Data:

En este último paso del primer bloque, lo que hacemos es pasar las grabaciones del formato ".wav" al formato interno de THK ya que trabaja mucho mejor con este último. Para ello se crean distintos ficheros en los cuales se indican ciertos parámetros para que la conversión de las grabaciones se lleve a cabo.

Una vez completado este quinto paso, hemos finalizado el punto de "Data preparation", por lo que estamos en disposición de pasar al paso que voxforge[1] ha llamado "Monophones". Este paso se desglosa en tres, a continuación explicaremos que se realiza en cada uno de ellos:

Step 6 - Creating Flat Start Monophones:

En este punto se define un prototipo de modelo de entrenamiento, el cual es necesario para el entrenamiento de un Modelo Oculto de Harkov. Para ello se crean una serie de ficheros y junto con los generados al ejecutar los scripts que nos indica el manual estamos listos para pasar al siguiente paso.

Step 7 - Fixing the Silence Models:

En este paso lo que se creamos siguiendo los pasos que nos dicen es crear un modelo oculto de Markov que incluya pausas pequeñas, (se refiere a las pausas que se producen entre palabras cuando hablamos normalmente). En el punto anterior si que creamos un modelo que incluía los silencios, (se refiere a las pausas que ocurren antes y al final de cada frase). Modificando alguno de los ficheros obtenidos anteriormente y ejecutando los scripts que nos indican conseguiremos completar este paso.

Step 8 - Realigning the Training Data:

Este paso es similar al que realizamos en el paso 4 del primer punto, sin embargo en este la orden HVite puede considerar todas las pronunciaciones de cada palabra (en el caso de que exista una palabra con varias pronunciaciones) y luego sacar la pronunciación que mejor coincida con la información acústica.

Una vez finalizado este octavo paso, podríamos utilizar nuestro modelo acústico con Julius, sin embargo la precisión de reconocimiento no sería demasiado buena, por lo que para mejorar la misma continuaremos con el paso que voxforge denomina como "Triphones" y que contiene los siguiente subapartados:

Step 9 - Making Triphones from Monophones

En este paso lo que se hace básicamente es mejorar la precisión de reconocimiento, para ello se introducen más restricciones para que dicha precisión aumente de manera significativa, para ello a cada fonema se le introducen 2 restricciones, consiste en ligar a cada fonema con aquel que tenga a su izquierda y aquel que tenga a la derecha, de esta forma será más difícil confundir a cada uno de ellos. Como ejemplo:

TRANSLATE [TRANSLATE] t r @ n s l e t

TRANSLATE [TRANSLATE] t+r t-r+@ r-@+n @-n+s n-s+l s-l+e l-e+t e-t

Lo que hacemos es pasar a mirar el contexto en el cual aparece cada fonema, ya que la mirar los fonemas que preceden y siguen al fonema en cuestión, lo que estamos haciendo es buscar una secuencia de 3 sonidos.

Step 10 - Making Tied-State Triphones

Una vez llegamos a este punto estamos a muy pocos pasos de conseguir nuestro modelo acústico, para ellos deberemos seguir los pasos que nos indican desde voxforge[1], el problema de este punto es que al ejecutar la siguiente orden:

HDMan -A -D -T 1 -b sp -n fulllist -g global.ded -l flog dict-tri ../lexicon/voxforge_lexicon

A nosotros nos saltaba un error ya que esta orden introduce algunos fonemas que únicamente están en el fichero voxforge_lexicon (Este es un fichero bastante grande que contiene cerca de 28000 palabras en ingles una en cada línea con el formato que necesita HTK y además cada una de ellas está separada en fonemas), no en nuestro

fichero dict, por lo que no tenemos ningún archivo de audio asociado a esos fonemas. Para resolver este error llegamos a la conclusión de que en vez de ../lexicon/voxforge_lexicon hay que poner el fichero dict que hemos generado en pasos posteriores y de esta forma podemos completar el resto de instrucciones. En nuestro caso la resolución de este fallo nos llevo varios días ya que en el momento en el que te da un error, si no lo encuentras en los hilos de la propia página, el libreo de referencia tampoco es demasiado útil ya que no te da una explicación demasiado detallada de cada error. Una vez terminado estos pasos ya podemos probar nuestro modelo acústico.

Otra cosa a tener en cuenta en este paso es que el fichero tree.hed, que nos aparece en la página en este punto, se corresponde con el idioma inglés, por lo que para generar un modelo acústico en otro idioma, tendríamos que construir o encontrar el fichero tree.hed adecuado al lenguaje con el que estemos trabajando.

PROBLEMAS ENCONTRADOS:

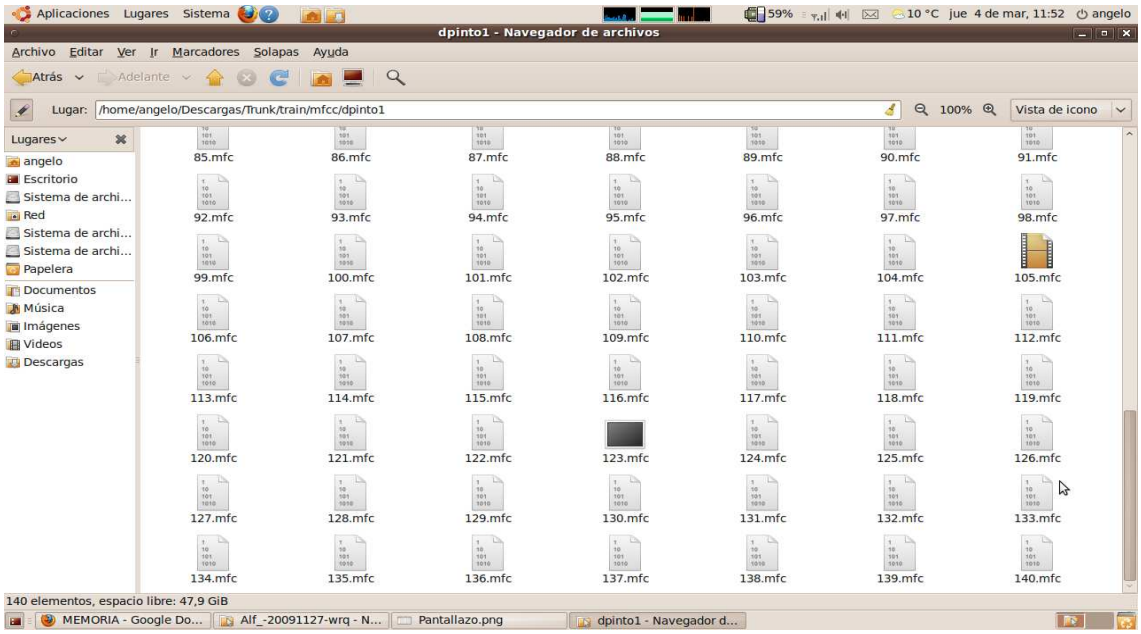
A la hora de aprovechar las grabaciones que pudimos conseguir del repositorio de voxforge nos encontramos con una serie de problemas que comentamos a continuación:

En primer lugar las muestras de audio estaban grabadas a una frecuencia de 8000 Hz, que según indica en el manual de voxforge deberían haber sido grabadas a la máxima que permitiese la tarjeta de sonido, en nuestro casos las muestras fueron grabadas a 48000Hz, por lo que la calidad de las muestras descargadas era bastante inferior a la de las nuestras. El problema que surge de esto, es que todas las grabaciones deben estar muestreados a la misma frecuencia. Así la única solución posible sería downsamplear nuestras muestras a 8 Khz., por lo que la calidad del modelo se vería comprometida.

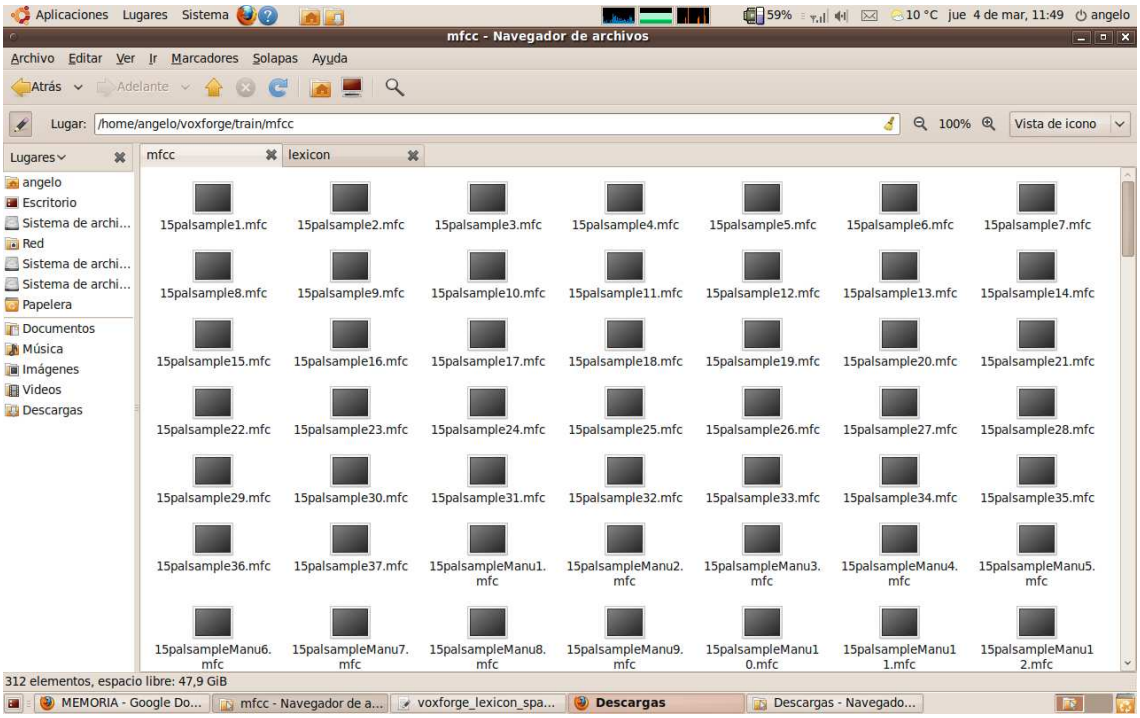
Otro de los problemas que nos encontramos es que la longitud de las frases que aparecían en los distintos ficheros prompts, eran muy superiores a la longitud que recomienda voxforge para las frases del mismo, la cual debe estar comprendida entre 8 y 10 palabras como nos indican en el paso 2 del manual.

Otro problema grave es que la mayoría de las grabaciones de hispanohablantes que se encuentran en Voxforge son de Sudamérica. Estos tienen maneras distintas de pronunciar diversos fonemas. Por ejemplo la palabra ACCIDENTADOS se descompone en fonemas para el sudamericano de esta manera: A K S I D E N T A D O S. En cambio para españoles sería de esta otra: A C C I D E N T A D O S. Como se ve presenta problemas que una vez mas envenenarán el modelo final creado.

En otra ocasión nos encontramos con que los ficheros .mfc que nos descargamos no tenían el icono característico de un terminal como se puede ver en la imagen siguiente



sino que la carpeta tenía archivos similares a estos:



lo cual nos indicaban que tanto el fichero de prompts a partir del cual se habían generado estaba mal construido, y que las grabaciones tampoco eran válidas por estas asociado a frases de un tamaño incorrecto, esta información la dedujimos nosotros ya que en una ocasión estuvimos atascados en un punto del manual hasta que comprobamos la carpeta mfcc y vimos que no todos los archivos tenían el icono del terminal, cambiamos el fichero prompts, realizamos las grabaciones de nuevo y pudimos continuar con el proceso de generación del modelo acústico.

Otro problema que encontramos es que las grabaciones estaban realizadas en un volumen muy bajo.

Todos estos problemas que encontramos junto que en ningún foro encontramos las indicaciones para ver como podíamos solucionar todos estos problemas nos hicieron decantarnos por crear una gramática propia y realizar nosotros las grabaciones necesarias para conseguir una tasa de acierto aceptable, el proceso y el número de grabaciones necesarias las comentaremos en el siguiente punto.

13 Bibliografía

[1] Voxforge: http://www.voxforge.org/

[2] Open Simulator: http://opensimulator.org

[3] Hippo OpenSim Viewer: http://mjm-labs.com/viewer/

[4] SVN: http://subversion.apache.org/

[5] TortoiseSVN: http://tortoisesvn.tigris.org/

[6] Audacity: http://audacity.sourceforge.net/

14 Palabras Clave

Julius

Modelo de lenguaje

Modelo acústico

Aventura gráfica

Juego

Javy

15 Autorización

Los abajo firmantes autorizan a la Universidad Complutense de Madrid a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a sus autores, tanto la propia memoria, como el código, la documentación y/o el prototipo desarrollado.

Pablo Caloto Crespo

Manuel Moranchel Edras

Ángel Ruiz Alonso

Madrid, a 29 de Junio de 2009