Virtual Hammond

Laboratórios de Informática

Universidade de Aveiro

Nelson Costa 42983
Pedro Martins 76551
Pedro Santos 76532
Ricardo Jesus 76613

7 de Junho de 2015

Virtual Hammond

Departamento de Electrónica, Telecomunicações e Informática

UNIVERSIDADE DE AVEIRO

Nelson Costa 42983, nelson.costa@ua.pt
Pedro Martins 76551, pbmartins@ua.pt
Pedro Santos 76532, pedroamaralsantos@ua.pt
Ricardo Jesus 76613, ricardojesus@ua.pt

7 de Junho de 2015

Resumo

Um órgão de Hammond, criado por Laurens Hammond, é um orgão, inicialmente utilizado em igrejas que mais tarde foi adotado por bandas de Jazz e Blues, ao qual foram adicionados efeitos que permitem enriquecer o som produzido. O objetivo final deste trabalho é a criação de uma aplicação Web com o objetivo de ser um Orgão de Hammond Virtual.

Através da mesma, é possível a criação de músicas a partir de uma pauta no formato RTTTL, um registo e um conjunto de efeitos, a visualização de gráficos WaveForm de cada uma e ainda será possível descarregar os ficheiros `WAVE` relativos a cada uma das interpretações da música, consoante os efeitos a ela aplicados.

Este documento visa descrever os procedimentos que foram adotados no projeto e descrever o funcionamento de todos os componentes envolvidos.

Conteúdo

Lista de Figuras

Capítulo 1

Introdução

Um orgão Hammond é um instrumento musical criado no século XX inicialmente utilizado em igrejas. Contudo, acabou por ser muito utilizado em bandas de Jazz e Blues, sendo mais tarde utilizado também em bandas de Rock. Este orgão permite utilizar até 9 osciladores (cilindros que permitem gerar sons quase puros) em simultâneo. A cada oscilador é também associada uma noção de amplitude, isto é, cada oscilador pode produzir sons com amplitude máxima (denotada pelo valor 8), "inexistente" (denotada por 0), ou qualquer valor inteiro intermédio (por exemplo ao valor 4 corresponderá metade da amplitude máxima). A soma dos sons gerados pelos osciladores constitui o som final produzido pelo orgão. A esta definição de amplitude a atribuir a cada oscilador chama-se *registo*, estando registos diferentes associados a timbres diferentes.

Para além disto, Laurens Hammond, seu criador, achou que os sons produzidos pelo seu orgão eram demasiado puros, então rapidamente lhe foram adicionados alguns efeitos que permitem enriquecer o som produzido por estes instrumentos.

Este projeto tem como objetivo a criação de um orgão de Hammond virtual que permita aos utilizadores criar músicas, sendo para isso apenas necessário fornecer a respetiva pauta (na linguagem RTTTL[1]), e o nome da respetiva música. Entretanto, se se desejar criar uma interpretação diferente, isto é utilizar efeitos ou registos diferentes é possível, enviando para o servidor através da interface web as respetivas configurações. A aplicação guardá a informação numa base de dados `SQLite` e no sistema de ficheiros, permitindo que as músicas possam ser ouvidas repetidamente após terem sido criadas. Para isso, criou-se interfaces web para o utilizador interagir com a aplicação, uma aplicação principal que constitui uma interface HTTP fornecendo respostas na forma de JSONs para as Interfaces Web e gere a lógica de funcionamento da aplicação, bem como um módulo capaz de levar a cabo todo o processamento relacionado com as músicas em si, desde

[1] `http://en.wikipedia.org/wiki/Ring_Tone_Transfer_Language`

interpretar pautas até produzir o ficheiro `WAVE` final.

Capítulo 2

Interfaces

2.1 Interface Web

Este módulo é o responsável por todas as interações entre o cliente e a aplicação. As interfaces Web comunicam com o utilizador e que, a partir das mesmas, consoante o pretendido, enviam pedidos ao servidor (de informações ou criação de ficheiros, por exemplo).

Criaram-se duas interfaces, uma aplicação para versão *desktop* e outra para versão *mobile*, estando elas disponíveis em `http://0.0.0.0:4506/web/` e `http://0.0.0.0:4506/mobile/`, respetivamente.

A versão *desktop* é construída em `HTML`, `CSS` e `JS`, tal como qualquer outra página Web, e tem como base o *template* Greyscale, desenvolvido pela Start Bootstrap, que foi construído utilizando a *framework* Twitter Bootstrap.

Por outro lado, mas utilizando as mesmas linguagens, está a aplicação para dispositivos móveis, na qual será utilizado um tema disponibilizado e baseado na *framework* Ratchet.

Criaram-se várias páginas de forma a dar resposta às diversas funcionalidades disponíveis pela aplicação:

Index A página principal, onde é apresentado o projeto e os seus autores, assim como hiperligações para versão móvel e para as diversas funcionalidades da versão *desktop*

Página da listagem de músicas Página onde são listadas todas as músicas presentes na base de dados e os seus respetivos gráficos relativos a representação das notas, assim como as suas interpretações. Relativamente ás interpretações é possível o utilizador dar *Like* ou *Dislike* e ver quantos "gostos"e não gostos tem cada interpretação. Outra opção que tem é ouvir ou fazer download da interpretação selecionada. Para além disso, é possível ser redirecionado para a página de criar interpretações da música escolhida previamente na página que lista as músicas.

Página de criação de uma música Página onde é possível a criação de uma música, fornecendo um nome e a pauta no formato RTTTL.

Página de criação de uma interpretação Acessível a partir da página da listagem de músicas carregando em *create interpretation* da respetiva música que se quer criar a interpretação, é a página onde será possível a criação de uma interpretação de uma certa música, aplicando-lhe um registo e efeitos disponíveis pela aplicação

A comunicação com o servidor é toda ela através de JavaScript e jQuery.

A página da listagem de músicas foi construída utilizando uma função que faz uso do `$.get()`, função de jQuery, que faz um GET ao servidor do qual é retornada uma lista de dicionários JSON, com informações relativas a todas as músicas, e, a partir delas, é construída a lista de todas as músicas presentes na base de dados. A cada elemento da lista é também associado um link da forma `song-page.html?id=value1&name=value2`, em que value1 e value2 correspondem, respetivamente, ao identificador na base de dados e ao nome da música escolhida, assim quando se cria uma nova interpretação vai ser associada à música certa, porque vai buscar o value1 e value2 ao link. A listagem das interpretações de cada música usa o mesmo processo que a listagem das músicas, a diferença é que para listar as interpretações a função usa como argumentos SongId e SongName, para poder associar corretamente á música.

A página que lista as música, como já foi referido, mostra as informações relativas a uma certa música e das suas interpretações. Nessa página também é possível visualizar o WaveForm da música, assim como descarregar ou ouvir os ficheiros `WAVE` associados às interpretações disponíveis. Para tal, é feito um GET ao servidor usando o método já descrito, onde é passado como argumento o ID da interpretação em questão, e onde é retornado a localização da imagem (WaveForm) da música, ou do `WAVE` específico a cada interpretação, consoante o pretendido. Em javascript foi implementado ainda uma função de pesquisa, que torna possível a pesquisa de músicas ou de interpretações de uma dada música.

Também é possível votar numa certa interpretação, isto é, o utilizador pode dar a sua opinião relativamente à mesma, onde pode deixar um voto favorável ou desfavorável. Para tal, é feito um POST usando o método `$.post()` (semelhante ao anterior, mas que envia a informação que queremos passar ao servidor pelo corpo do pedido e não pelo cabeçalho como acontece usando `$.get()`), mas a uma das funções existentes no servidor (`vote(self, id, ip, type)`).

A partir da página de criação de uma música, tal como o nome indica, é possível a adição de uma nova música à base de dados. Tal é feito através de um POST ao servidor com um dicionário JSON com o nome e a pauta RTTTL da música que o utilizador pretende criar.

Por outro lado, também é possível a criação de novas interpretações. Nessa página, será possível escolher os efeitos que se pretendem aplicar, assim como os valores associados a cada um. Também é feito um POST ao servidor com um dicionário JSON com as informações em questão, a partir das quais é construído um novo ficheiro `WAVE`.

2.2 Interface Móvel

Para o seu desenvolvimento, foi também utilizado `jQuery`, para diversas funções de pesquisa, pressionar de botões e comunicação com o servidor CherryPy.

A aplicação está dividida em 4 páginas distintas: a página inicial (onde são listadas todas as músicas disponíveis), a página de criação de músicas, a página individual de cada música e a página de criação de interpretações.

2.2.1 Página Principal

A página principal é constituída por uma barra da classe `bar-nav` no topo, onde existem 2 hiperligações: a da esquerda (About), mostra um objeto do tipo `modal` (próprio da framework) com as informações relativas ao projeto e aos seus criadores, enquanto que a da direita é uma hiperligação para a página de criação de músicas.

Mais abaixo está uma barra de pesquisa, para o utilizador pesquisar pela música pretendida. Essa barra ativa a função de `JavaScript searchSong` a cada "levantar"de tecla (keyup), que a percorre todos os elementos da listagem de músicas abaixo utilizando a função `each` da biblioteca `jQuery` e compara o conteúdo da `span` "name"de cada elemento da listagem com o conteúdo da barra de pesquisa. Se o resultado for positivo, é mostrado esse elemento, casa contrário, é escondido. Caso o utilizador pressione o "X"ao lado da barra, será apagada a pesquisa e mostradas todas as músicas existentes na base de dados.

Depois do slide, construído utilizando classes do Ratchet, está a listagem das músicas existentes. Aquando o carregamento da página, é feito um pedido ao servidor para que sejam devolvidas todas as músicas e as informações relativas às mesmas. Para tal, é utilizada a função `$.get()` da biblioteca `jQuery`, que simplesmente faz um GET ao servidor, utilizando o endereço `/listSongs`, e, a partir da resposta, é criada toda a listagem. Uma particularidade desta lista de músicas é que, a cada elemento, está associado um link do tipo `song-page.html?id=xpto1&name=xpto2`, sendo *xpto1* o identificar na base dados e *xpto2* o nome da música, respetivamente. Assim, é possível "passar"informação da página principal para a página individual de cada música utilizando apenas o endereço.

As restantes funções (de pesquisa, etc.), encontram-se dentro de uma função chamada `updateList`, que é apenas chamada dentro da função `$.get()`.

Recorreu-se a este método, visto que, como se iria comunicar com classes que ainda não estavam atribuídas (só depois de construída a lista é que isso aconteceria), as funções inicialmente construídas não teriam qualquer efeito. Isto deve-se a uma particularidade da função `$.get()`, que, depois de enviado o pedido ao servidor, permite que as funções depois dela sejam logo executadas, surgindo este tipo de problemas.

2.2.2 Página de Criação de Músicas

A página de criação de músicas é deveras simples. Visualmente, é apenas constituída por elementos `<input>` e por um botão, para além dos objetos de alerta (sucesso, erro, etc.), que estão inicialmente escondidos. Depois de devidamente preenchidos todos os campos (nenhum deles poderá estar em branco, caso contrário, serão mostrados alertas de erro) e de pressionado o botão de criar a música, é enviado ao servidor um dicionário `JSON` com o nome e a pauta `RTTTL` da música, utilizando a função `post`, de `jQuery`. Ao contrário da `$.get()`, que envia no cabeçalho as informações relativas a argumentos (ID, etc,), esta última envia as informações no corpo do pedido, evitando assim alguns erros (no caso da pauta, como é uma String enorme, poderiam surgir erros).

Caso a resposta do servidor seja de que conseguiu criar a música e adicioná-la à base de dados com sucesso, será mostrado um alerta informando isso mesmo. Por outro lado, caso seja recebida uma mensagem de erro, será também exibido um alerta de erro.

2.2.3 Página Individual de uma Música

Esta página foi construída com o intuito de mostrar todas as informações relativas a uma música, sejam elas o seu gráfico WaveForm, as notas `RTTTL`, as interpretações disponíveis, etc. Para a criação da mesma, é necessário o ID e o nome de uma música, informações essas que serão extraídas do URL da página, algo já referido na secção Subseção 2.2.1. Para isso, é utilizada a função `getURLInfo(field)` [1], em que *field* é o nome do campo do qual queremos o valor; para extrairmos o ID e o nome, foi necessário chamar duas vezes esta função, passando como argumentos `id` e `name`, respetivamente.

Depois de obtidas as informações relativas à música através do URL, é construída a hiperligação para a página de criação de novas interpretações com os mesmos campos (ID e nome da música), e é feito um pedido ao servidor (usando novamente a função `$.get()`), desta vez ao endereço `/listSongFiles?id=SongID`. Daí será retornado um dicionário `JSON`, com a localização da imagem do WaveForm e com uma lista de outros dicionários com as informações relativas a cada interpretação, de entre as quais,

[1] Esta função foi baseada na construção apresentada aqui: `http://www.jquerybyexample.net/2012/06/get-url-parameters-using-jquery.html`

o ID, o nome, o número de votos positivos e negativos e ainda a localização do ficheiro `WAVE` gerado, a partir das quais é construída a listagem de interpretações na página.

Utilizando ainda a resposta anterior do servidor (nomeadamente a localização do WaveForm e o ID da música), é executada a função `updateModal(SongID, WaveForm)` dentro do pedido anteriormente feito, que irá fazer outro GET ao servidor (endereço `/getNotes?id=SongID`), onde serão retornadas as notas da música para completar o objeto da classe `modal` (da framework Ratchet).

E tal como na secção Subseção 2.2.1, também foi criada uma função (`updateSongPageList()`) de maneira a que as funções dentro da mesma já tenham referências (classes, id, etc.) na página, visto que a listagem já tinha sido anteriormente construída.

Dentro desta função, estão todas as funções de comunicação com a listagem de interpretações, de entre as quais a função de pesquisa e eliminação da mesma (em tudo semelhantes às descritas na secção Subseção 2.2.1), as funções de "gostar"ou não de uma interpretação e ainda a que permite ouvir o `WAVE` de cada uma.

A função para "gostar"ou não de uma interpretação (`likeInterpretation(InterpretationID, vote)` aceita como argumentos o ID da interpretação em causa e ainda o respetivo voto: caso seja pressionado o botão "Like", será passado como argumento "PositiveVotes", caso seja pressionado o botão "Dislike", será passado "NegativeVotes". Assim que esta função é executada, faz um GET ao endereço `https://api.ipify.org?format=jsonp&callback=?`, que irá retornar um dicionário `JSON`, como várias informações do cliente, nomeadamente o seu IP. A partir daí, é construído outro dicionário com o IP, ID da interpretação e o tipo de voto, e feito um POST ao servidor através do endereço `/vote`. Caso a resposta seja positiva, isto é, caso o utilizador ainda não tinha votado na interpretação em causa e não tenha existido num erro interno do servidor, será chamada a função `updateLikeText(div, InterpretationID)` (o argumento *div* corresponde à `<span>` que será atualizada; pode ser da classe `like-text` ou `dislike-text`), que irá atualizar o texto relativo aos votos. Caso a resposta seja "Not Allowed.", ou seja, o utilizador já votara anteriormente na interpretação em causa, o texto não será atualizado.

Por último, existe uma pequena função que apenas serve para mostrar e esconder a área onde está o leitor de música de cada interpretação, utilizando para o efeito a função `slideToggle` da biblioteca `jQuery`.

2.2.4 Página de Criação de Interpretações

Por fim, a página de criação de interpretações é, a par da de criação de músicas, as únicas que não são quase totalmente criadas dinamicamente. Esta também é composta por uma série de elementos do tipo `<input>` e `<button>`, assim como objetos (`<div>`) de alertas. Tal como a página individual de cada música, também é extraído do URL o ID e o nome da música

para a qual se vai criar uma interpretação (sendo estes utilizados para gerar o link para voltar à página individual da música em questão, para além de que o nome servirá também como título da página).

Foram também criadas funções para aumentar ao diminuir o valor de cada um dos elementos do registo do orgão (`addUnit(id)` e `subtractUnit(id)`, sendo o argumento id o identificador de cada um dos elementos do bloco da classe `segmented-control`), consoante o utilizador clivasse num elemento botão com um "+"ou um -".

Depois de devidamente preenchidos todos os campos (caso o nome esteja em branco e o registo apenas tenha zeros, será mostradas mensagens de erro) e depois de o utilizador clicar no botão de criação, serão executadas funções para retornar os valores do nome, registo (`getRegistration()`) e efeitos (`getEffects()`), é construído um dicionário `JSON` e executada a função `post("/createInterpretation", data, callback)`, sendo *data* o dicionário e *callback* a função executada depois de receber a resposta do servidor. Caso a resposta seja positiva, é mostrada uma mensagem de sucesso, caso contrário, é mostrada uma mensagem de erro.

Membros Encarregues

2.3 Interface Móvel

As interfaces Web foi desenvolvida por Pedro Santos e Pedro Martins. Por um lado, a criação das páginas da aplicação Web foi desenvolvida por Pedro Santos, enquanto que a aplicação móvel foi desenvolvida por Pedro Martins. A construção da plataforma de comunicação com servidor foi realizada maioritariamente pelo Pedro Martins, no entanto, a adaptação da mesma à versão *desktop* foi realizada por Pedro Santos, daí que Pedro Martins tenha contribuído com 60% e Pedro Santos com 40%.

Capítulo 3

Aplicação Principal

Recorreu-se à *framework* `CherryPy` para implementar um servidor capaz de disponibilizar os diferentes serviços suportados pela aplicação, recebendo pedidos das **Interfaces Web 2.1 e Móvel 2.3** e dando-lhes resposta, para isso interagindo com uma base de dados `SQLite` e/ou com o módulo **Hammond 4**. O código deste módulo está escrito na linguagem `Python` e pode ser consultado em **VirtualHammond.py**. Este servidor está associado à porta 4506 e correndo nos servidores *xcoa* em `http://xcoa.av.it.pt/cherry/p5g6/`.

Esta interface recebe pedidos das interfaces web e móveis, às quais dá respostas que constrói recorrendo a informações presentes na base de dados da aplicação, ou a processos levados a cabo pelo módulo de Hammond (4). Efetivamente, de grosso modo este módulo foca-se em construir respostas admissíveis às interfaces que expõem toda a aplicação e com que comunica. Por outro lado estas respostas tanto podem ser por exemplo texto no formato `HTML` que deve ser interpretado pelo browser utilizado pelo utilizador, de forma a expor a página de rosto da aplicação, ou texto no formato `JSON` utilizado pelo `JavaScript` do lado dos clientes, que por sua vez pode permitir listar todas as músicas existentes na base de dados, com o número das suas interpretações e visualização das suas notas.

Foi também utilizada uma base de dados `SQLite` com o objetivo de guardar informações relativas às músicas e interpretações criadas, bem como votos registados por IP (de forma a se restringir votos múltiplos para um mesmo indivíduo). A estrutura utilizada é a exposta na figura 3.1.

Os diferentes serviços disponibilizados por esta aplicação são:

/**createSong?name=text&notes=text** Método que permite adicionar uma nova música à base de dados. Para isso duas Strings, uma com o nome a atribuir à música e outra com a pauta no formato RTTTL, e recorre a funções do **Interpretador de Pautas 4.1** de forma a gerar não só uma estrutura utilizável pelo resto do programa (aquando da criação de interpretações) como também uma visualização das notas (na

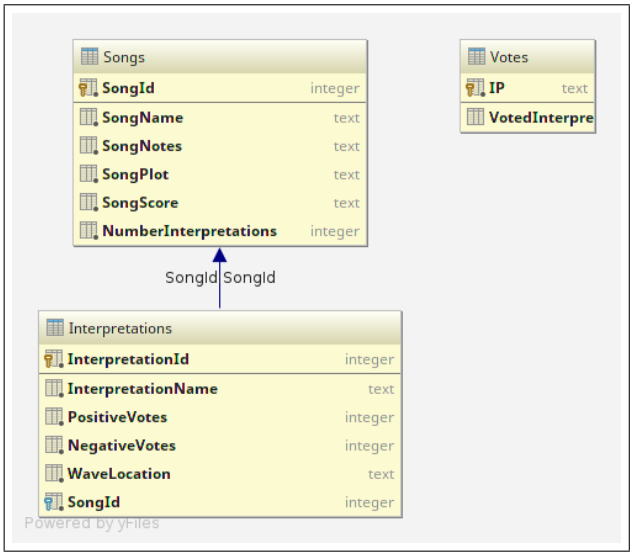


Figura 3.1: Estrutura da base de dados utilizada. Na tabela **Songs** são registadas informações relativas às diferentes músicas criadas, na tabela **Interpretations** informações associadas às diferentes interpretações e por fim na tabela **Vots** são registados os IPs a partir dos quais já foram registados votos para interpretações.

forma de um gráfico Frequência por tempo). Note-se que podia ser evitada a criação da primeira estrutura referida, no entanto optou-se por esta abordagem já que assim estas informações são criadas apenas uma vez, evitando chamadas constantes à função **interpret** do interpretador. Assim, adiciona-se uma nova entrada à base de dados (tabela **Songs**) com o nome da música, a sua pauta, a representação da estrutura gerada pelo interpretador e a localização no sistema de ficheiros da visualização gerada.

/createInterpretation?id=number&name=text&registration=text&effects=text

Permite a criação e adição à base de dados de uma nova interpretação. Para isso recebe o ID (valor numérico) relativo à música para a qual criar a interpretação, o nome (String) a lhe dar, o registo (String) e efeitos a aplicar (dicionário conforme exposto em 4.3). Para isso, obtém da tabela **Songs** a estrutura guardada aquando da criação da música referenciada por ID, de forma a a partir destes valores e do registo, recorrendo ao sintetizador, gerar uma síntese da interpretação. Por fim, fornecendo esta síntese e o dicionário de efeitos ao processador de efeitos, os efeitos são aplicados e o ficheiro WAVE criado. Neste momento pode ser adicionada à base de dados (à tabela **Interpretations**) uma nova entrada com o nome da interpretação criada, o ID da música para a qual é uma interpretação e a localização do ficheiro

WAVE criado. É também incrementado o número de interpretações que a música associada ao ID possui, na tabela **Songs**.

/**listSongs** Esta função permite retornar todas as músicas existentes na tabela Songs da base dados sobre a forma de uma lista de dicionários `JSON`. Para o efeito são devolvidos o identificador chave, o nome, a localização da imagem do gráfico gerado pela biblioteca `matplotlib` e o número de interpretações associadas a uma certa música, em cada uma das entradas da lista. Para que a resposta do servidor pudesse ser interpretada como um `JSON`, é adiciona aos cabeçalhos da resposta o tipo de conteúdo (`Content-Type`), que, neste caso, seria `text/json`.

/**listSongsFiles?id=number** Semelhante à função anterior; também esta envia ao cliente uma resposta em `JSON` num dicionário onde o primeiro elemento é a localização da imagem do WaveForm (também enviada na função anterior, mas foi repetida aqui para facilitar a integração com as páginas Web) e o segundo é um array com todas as interpretações associadas à música ao qual corresponde o ID passado como argumento da função. Para tal, é feito um pedido à tabela Interpretations da base de dados, onde é devolvida uma listagem de *tupples* com o identificador, o nome, votos positivos e negativos e, por último, a localização do ficheiro `WAVE` gerado pelo Processador de Efeitos associados a cada uma das interpretações.

/**getNotes?id=number** Função muito simples que retorna um dicionário `JSON` com as notas da música associada ao ID passado com argumento, depois de ser feito um pedido à tabela Songs.

/**getWaveFile?id=number** Esta função tem como objetivo retornar o ficheiro `WAVE` associado ao ID da interpretação passada como argumento da função. Primeiro é feito um pedido à tabela das interpretações onde é retornada a localização do ficheiro. De seguida, configura-se que o tipo de resposta do servidor, que será `application/octet-stream`, isto é, um ficheiro binário, e também o `Content-Disposition`, que, muito resumidamente, serve para o servidor dar uma sugestão do nome do ficheiro que será posteriormente guardado no dispositivo do cliente (neste caso, atribuiu-se ao nome a localização do ficheiro). Deste modo, resta abrir o ficheiro e retornar o seu conteúdo usando uma simples função de leitura: `open(file_name, "r").read()`.

/**getWaveForm?id=number** Esta função apenas retorna um dicionário `JSON` com a localização do WaveForm da música associada ao ID passado como argumento.

/**vote?id=number&ip=text&type=text** Método que permite votar numa dada interpretação, com *like* ou *dislike*. Para isso recebe como argu-

mentos um id (valor númerico) que indica a interpretação em que se quer votar, ip, uma String que indica o IP do cliente que fez o pedido ao servidor e um type (String) que indica se o voto é *like* ou *dislike*. A respetiva coluna da tabela **Interpretations** é incrementada de acordo, no entanto antes verifica-se, recorrendo-se à tabela **Votes**, se o IP recebido já votou na interpretação em questão. Caso a resposta seja afirmativa, então é devolvida uma mensagem de insucesso, caso contrário a execução continua.

Realça-se o facto de na tabela **Songs** ser guardada uma representação da estrutura de dados gerada pelo interpretador de pautas (4.1). Optou-se por esta abordagem de forma a evitar a geração de interpretações sempre que se pretendesse gerar uma nova interpretação da música. Assim, tira-se partido da função `eval()` de `Python` e consegue-se reconstruir a interpretação de pautas através da representação guardada. Para além disso, em ambas as tabelas **Songs** e **Interpretations** se guardam o local, no sistema de ficheiros, onde foram guardados a imagem relativa à visualização das notas de uma música bem como o ficheiro WAVE que resulta de uma interpretação, respetivamente. No sistema de ficheiros, estes ficheiros devem ser guardados nos diretórios `img` e `wav` respetivamente existentes no mesmo diretório que o ficheiro da aplicação.

Por fim, relativamente à tabela **Votes**, esta serve apenas para limitar o número de votos possíveis por IP e por interpretação, efetivamente garantindo que um dado IP apenas pode votar uma e uma só vez (com *like* ou *dislike*) em cada interpretação. Apesar desta não ser a melhor implementação (outra eventualmente melhor seria a autenticação de utilizadores), considera-se que sem a introdução de autenticação é difícil evitar votos múltiplos de uma forma verdadeiramente eficiente, e portanto escolhemos a que achamos melhor enquadrar-se ao pedido. É de notar que, no entanto, esta implementação é algo limitada e não garante que um dado utilizador apenas possa votar uma vez, tendo consciência desse facto.

Membros Envolvidos

O desenvolvimento deste módulo foi levada a cabo por Pedro Martins e Ricardo Jesus. Os dois primeiros métodos, `createSong` e `createInterpretation` bem como o desenvolvimento da base de dados estiveram ao encargo de Ricardo Jesus, tendo também contribuído na implementação do controlo de votos por IP. Já os restantes métodos foram desenvolvidos por Pedro Martins, tendo também reestruturado certos pontos da base de dados de forma a facilitar a integração com as interfaces, e resolvido questões com diretórios estáticos servidos pelo servidor `CherryPy`. Assim, considera-se que Ricardo Jesus contribuiu com 40% e Pedro Martins com 60% para o desenvolvimento deste módulo.

Capítulo 4

Módulo Hammond

Este módulo foi desenvolvido com o intuito de dar resposta a todas as tarefas relacionadas com processamento de áudio, e portanto engloba um **Interpretador de Pautas 4.1**, **Sintetizador 4.2** e **Processador de Efeitos 4.3**. Desta forma os utilizadores da aplicação não têm acesso direto às funções por ele disponibilizadas, em vez disso apenas o utilizam consoante as opções disponibilizadas e selecionadas das interfaces Web, que comunicam com o servidor principal que por sua vez executa o pretendido através deste módulo, eventualmente recorrendo a métodos deste módulo.

As principais tarefas que devem recorrer a este módulo serão a criação de uma música, que utiliza funções do interpretador de pautas de forma a a partir de uma pauta no formato RTTTL[1] gerar uma estrutura utilizável pelo resto do programa, e a criação de uma interpretação que deve recorrer ao sintetizador e processador de efeitos com o objetivo de gerar um ficheiro `WAVE` com a aplicação de determinados efeitos e segundo um determinado registo (escolhidos pelo utilizador).

Foram definidos três testes simples aquando do desenvolvimento, um para o interpretador, outro para o sintetizador, e finalmente um para o processador de efeitos de forma a garantir que as principais funções expressas acima deveriam funcionar quando integradas com a aplicação principal, comparando os resultados obtidos pelas nossas implementações com o exemplo disponibilizado pelo professor João Paulo Barraca. No final deste capítulo serão também incluídos printscreens que pretendem evidenciar o bom funcionamento deste módulo.

4.1 Interpretador de Pautas

Este componente utiliza a função `interpretation` para gerar uma lista de sons constituída pelas durações e frequências das notas a partir de uma

[1] `http://en.wikipedia.org/wiki/Ring_Tone_Transfer_Language`

pauta no formato RTTTL, i.e., uma string composta por 3 partes separadas pelo caracter ':', onde se definem o nome, os parâmetros de referência (duração, oitava e batidas por minuto) e as notas referentes à música, respetivamente.

Basicamente, a função `interpretation` é responsável por receber uma pauta no formato RTTTL e devolver uma lista de sons do tipo $[(t1, f1), (t2, f2), ...]$, onde $(tX, fX),\ X \in \mathbb{N}$ denota um par (tuplo) referente aos valores de duração (em segundos) e frequência fundamental (em Hertz) da cada nota da pauta. Os valores de duração e de frequência são obtidos através dos parâmetros de referência e das notas da pauta, sendo os valores de frequência obtidos também a partir de uma LUT[2] implementada como um dicionário, onde foram previamente definidas as frequências de todas as notas possíveis (com oitavas a variar de 0 a 8).

Por outro lado, este componente implementa outras funções que funcionam apenas de auxílio à função `interpretation`. Por exemplo, a função `plot` permite produzir uma representação gráfica das frequências das notas ao longo do tempo, onde a frequência fundamental (em Hertz) e a duração (em segundos) são as variáveis dependente e independente, respetivamente. Na Figura 4.1 é visível um gráfico exemplo gerado por esta função. De referir que esta função `plot` só é utilizada para efeitos de visualização, sendo que o resultado obtido por ela não será reutilizado por outros elementos deste módulo.

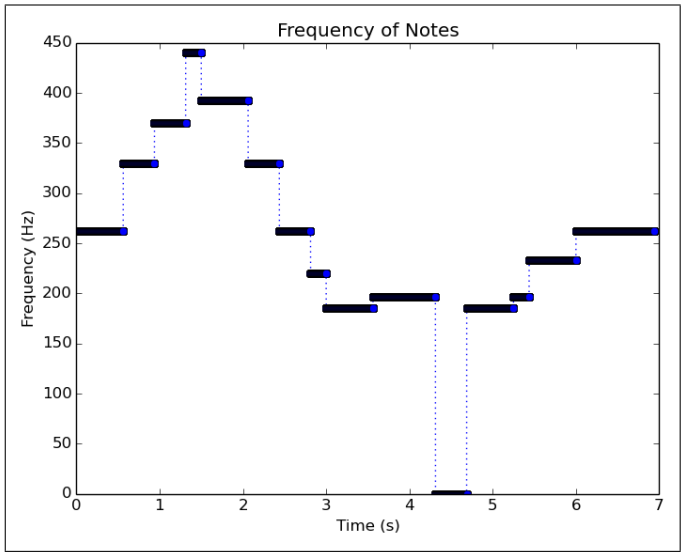


Figura 4.1: Gráfico gerado através da biblioteca *matplotlib*, referente à música **The Simpsons**. São visíveis as frequências e durações das notas ao longo do tempo.

[2] `http://en.wikipedia.org/wiki/Lookup_table`

4.2 Sintetizador

Deste componente realça-se a sua função `synthesize`, já que qualquer outra funciona apenas como auxiliar a esta. É responsabilidade deste componente, recebendo uma lista de notas no formato $[(dur1, freq1), (dur2, fre12), ...]$ bem como um registo a aplicar, gerar uma lista de dicionários onde cada dicionário é relativo a uma nota, segundo o formato $[\{'freq' : freq1, 'samples' : [sample1, sample2, ...]\}, ...]$.

Para isto teve-se em conta as seguintes condições:

- Na lista recebida como argumento, $durX$ denota a duração de uma nota em segundos e $freqX$ a sua frequência em Hz;
- Um registo é uma String de nove carateres cada um pertencente ao intervalo $[0, 8]$;
- Há nove osciladores: $\left[\frac{1}{2}, \frac{2}{3}, 1, 2, 3, 4, 5, 6, 8\right]$. Estes valores devem ser multiplicados pela frequência fundamental da nota, atribuindo a cada um um peso específico que depende do registo fornecido;
- A cada valor (caratere) do registo é atribuído um peso, valor pertencente a $\left[0, \frac{1}{8}, \frac{1}{4}, \frac{3}{8}, \frac{1}{2}, \frac{5}{8}, \frac{3}{4}, \frac{7}{8}, 1\right]$, utilizado no ponto acima;
- Para cada par $(durX, freqX)$ deve ser gerado um dicionário com duas chaves. Uma referente à frequência fundamental da nota, e outra referente a uma lista de amostras geradas de acordo com o exposto acima. O número de amostras é dado por $\lfloor durX * FrameRate \rfloor$;
- Cada amostra i de uma nota é dada por

$$\sum_{t=0}^{8} Registry[t] * \sin\left(\frac{2\pi * FrequencyOscillators[t] * i}{FrameRate}\right),$$

  onde $Registry[t]$ denota o valor da posição t do registo, $FrequencyOscillators[t]$ o valor na posição t da lista de valores de osciladores sendo cada posição multiplicada pela frequência fundamental da nota, e $FrameRate$ o número de amostras por segundo geradas. No caso deste projeto este último valor é de 44.1kHz;
- O conjunto de todas as amostras de uma nota constitui a lista de amostras que fica associada à chave *samples* do dicionário relativo a cada nota;
- O conjunto dos dicionários de todas as notas forma a lista de retorno da função.

Uma nova síntese é gerada sempre que for pretendido gerar uma nova interpretação. Essa síntese é depois fornecida ao processador de efeitos que se encarrega de aplicar os efeitos que foram pretendidos, criando depois o ficheiro WAVE final.

4.3 Processador de Efeitos

Este é o componente que engloba todas as funções necessárias para a adição de efeitos à música e criação do ficheiro WAVE final. Dele realça-se a função `process` que recebe como argumentos o nome a atribuir ao ficheiro WAVE, a síntese de uma música (vinda do sintetizador 4.2), à qual aplicar efeitos, e por fim um dicionário que simboliza a lista de efeitos que se pretende aplicar. Ainda em relação aos argumentos, o nome do ficheiro é apenas uma String, a síntese uma lista de dicionários onde cada dicionário é relativo a uma nota e guarda a frequência fundamental da mesma bem como o conjunto das suas amostras (conforme explicado na secção anterior), e o dicionário de efeitos segue a estrutura $\{'effect1' : \{'v1' : val1,' v2' : val2, ...\}, ...\}$.

Todos os efeitos que se pretendam implementar devem estar presentes no dicionário de efeitos recebido. Os valores a aplicar em cada um devem estar presentes no dicionário associado a cada efeito. Caso algum destes últimos valores não seja indicado assumem-se valores padrão.

De forma a serem aplicados todos os efeitos pretendidos, verifica-se se um determinado efeitos está presente no dicionário e caso esteja ele é aplicado, verificando-se a seguir os efeitos em falta. A lista de verificação é $precussion \rightarrow chorus \rightarrow envelope \rightarrow echo \rightarrow tremolo \rightarrow distortion$. É também de notar que para os três primeiros efeitos estes são aplicados a cada nota separadamente, portanto a função responsável por aplicar cada um deles recebe como argumento uma síntese da música (que pode eventualmente já ter sido exposta a outros efeitos). Já os últimos três são aplicados a toda a música por igual, e portanto recebem como argumento uma lista com todas as amostras seguidas, lista que é criada após a aplicação dos três primeiros efeitos. Abaixo deixam-se breves explicações sobre cada um dos efeitos, no entanto estas fundamentam-se muito nas já dadas no guião do projeto. A figura 4.3 evidencia a aplicação dos efeitos expostos à música "The Simpsons".

Echo Gera o efeito echo, adicionando o sinal atenuado por um valor a dum instante i ao de um instante $i + \Delta t$, onde Δt denota o tempo de atraso. Para isso é convertido o tempo de atraso no número de amostras de diferença que deve haver desde o instante presente ao instante de atraso através de $d = FrameRate * \Delta t$, aplicando-se depois o efeito segundo $samples[i + d] += a * samples[i]$, onde i denota um dado instante, d o número de amostras de diferença, a o valor de atenuação e $samples$ a lista de todas as amostras da música.

Tremolo Variação periódica f_t de uma magnitude a da amplitude do sinal. A fórmula utilizada é $samples[i] += a * \sin(\frac{2\pi * f_t * i}{f_r})$ onde i denota um dado instante, a a magnitude do efeito, f_t a sua frequência, f_r o número de amostras por segundo e por fim *samples* a lista de amostras da música.

Distorção Amplificação exagerada da amplitude de um sinal, de magnitude n. Foi utilizada a fórmula $samples[i] = samples[i]^n$, onde i denota um dado instante, n a magnitude do efeito e *samples* a lista de todos os efeitos.

Percussão Adição de uma harmónica (múltiplo da frequência de um sinal) no início da música ou após uma pausa, decaindo até se anular.

Chorus Adição de um sinal com uma frequência ligeiramente superior à do sinal original

Envelope Manipulação da amplitude do sinal de forma a simular o timbre natural de um orgão de Hammond. O resultado pretendido é o ilustrado na figura 4.2.

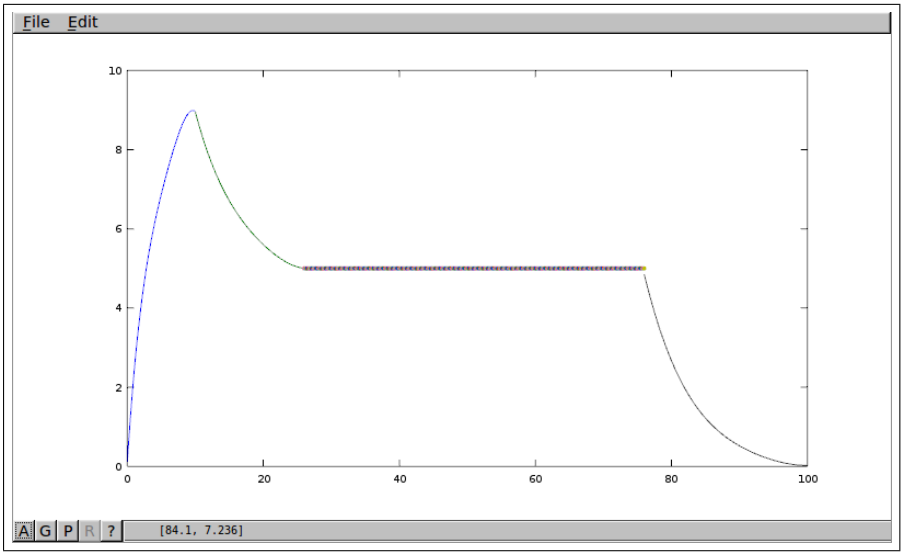
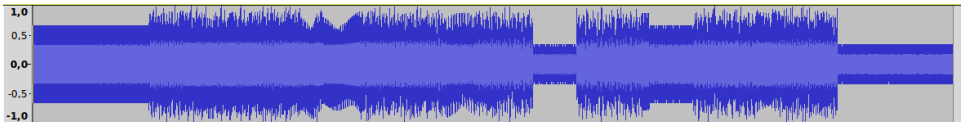
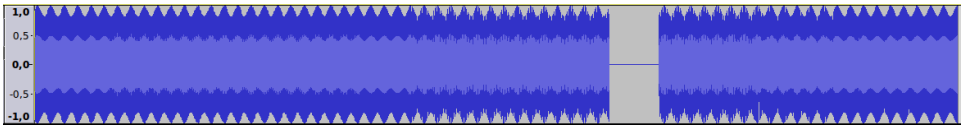


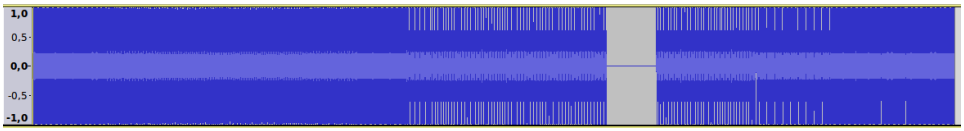
Figura 4.2: Efeito de envelope aplicado. Gráfico obtido recorrendo-se ao *software* GNU Octave, através do qual também se obtiveram as funções que aproximam o efeito.



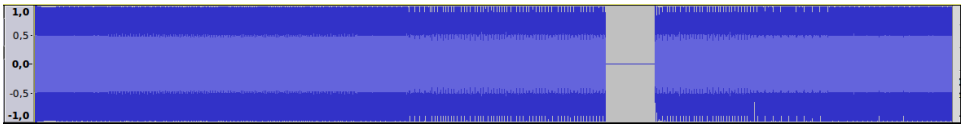
(a) Efeito Echo



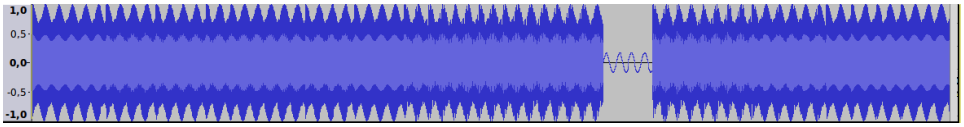
(b) Efeito Tremolo



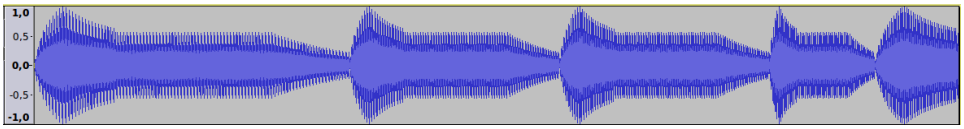
(c) Efeito Distortion



(d) Efeito Percussion. É visível no início de cada nova nota um pequeno aumento na intensidade da música. Isto é causado por este efeito. O facto de ser pouco visível deve-se à implementação, tendo-se optado por fazer este efeito pouco notório



(e) Efeito Chorus. Toda a música parece exposta a um tremolo, no entanto isto deve-se ao facto de haver uma grande diferença entre a frequência fundamental da nota e a frequência do efeito



(f) Efeito Envelope, aplicado a cada nota quando a sua frequência é diferente da anterior

Figura 4.3: Imagens obtidas recorrendo-se ao *software* Audacity

Membros Envolvidos

Neste módulo estiveram envolvidos Nelson Costa e Ricardo Jesus. O desenvolvimento do **Interpretador de Pautas 4.1** esteve ao encargo de Nelson Costa, enquanto que tanto o **Sintetizador 4.2** como o **Processador de Efeitos 4.3** foram trabalho de Ricardo Jesus.

Capítulo 5

Conclusões

Apesar de ter sido um projeto trabalhoso, obrigando-nos a aprender sobre diversos temas, consideramos também que foi extremamente enriquecedor. Pudemos trabalhar verdadeiramente em equipa e usufruir das enormes vantagens disso, por outro lado fomos obrigados a desenvolver melhores qualidades de comunicação e sentimos uma necessidade crescente de esclarecer ao máximo o que cada membro do grupo faria e quais as restrições para o fazer, de forma a evitar mais tarde terem de ser corrigidos pequenos problemas de compatibilidade entre módulos que acabavam por consumir o seu tempo, desnecessariamente. Por outro não consideramos este fator algo mau, pelo contrário. Sendo obrigados a clarificar corretamente o que será feito obriga a uma melhor estruturação de tudo, no final resultando (neste caso) numa melhor aplicação. Neste sentido ajudou também o bom ambiente entre o grupo, onde os seus elementos sempre cumpriram com aquilo a que se propunham, muitas vezes auxiliando outros mesmo em trabalho que não era seu de forma a o melhorar.

Em suma, consideramos que foi possível desenvolver a aplicação web como o pretendido, tendo-se cumprido com os objetivos iniciais estabelecidos no relatório preliminar. Contudo, durante a realização do projeto algumas ideias iniciais foram alteradas para melhorar e otimizar a aplicação, tais como a página das músicas conter simultaneamente a listagem das músicas e das interpretações. Outro exemplo poderá ser o método de votos disponibilizado pela aplicação principal. Inicialmente eram permitidos "votos infinitos" a cada utilizador, no entanto tendo considerado que isso era errado e podia (e devia) ser melhor implementado, passou-se a limitar um único voto para cada interpretação por IP.

No entanto, temos consciência de que vários pontos desta aplicação poderia ter sido melhor explorados e abordados. Por exemplo, tivesse sido implementado um sistema de autenticação de utilizadores e seria de facto possível garantir de forma muito mais correta que um único utilizador vota apenas uma vez numa interpretação. Outro exemplo poderá ser o da apli-

cação de efeitos, onde poderiam ter sido dadas outras opções de controlo aos utilizadores, por exemplo a intensidade com que se aplica o efeito de `chorus` ou `percussion`. No entanto, por falta de tempo e não nos tendo comprometido a implementar estas opções, elas foram postas de parte.

Por fim, mesmo apesar de vários pontos poderem ser melhorados como frisado anteriormente, mantemo-nos satisfeitos com o nosso trabalho. Consideramos que a possibilidade de conviver com tantos conceitos diferentes em simultâneo contribuiu significativamente para a nossa aprendizagem, deixando-nos alertas e bastante mais à vontade para embarcar em projetos futuros.

Referências

[1] *Synthesizing hammond organ effecss*, http://www.soundonsound.com/sos/jan04/articles/synthsecrets.htm, 2015.

[2] *Synthesizing tonewheel organs*, http://www.soundonsound.com/sos/nov03/articles/synthsecrets.htm, 2015.

[3] *Hammond organ*, http://en.wikipedia.org/wiki/Hammond_organ, 2015.