

**FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO**



**FEUP**

# **Transcrição de Canto para Pauta Musical**

**Miguel Garcia**

Mestrado Integrado em Engenharia Electrotécnica e de Computadores

Orientador: Aníbal Ferreira (Prof. Dr. Eng.)

Janeiro de 2012



# Resumo

A escrita musical é algo que, apesar de comum e ao mesmo tempo essencial entre os músicos, exige não só alguma formação musical para a sua leitura ou escrita limitando este sistema tão útil apenas ao músico com formação, como também é um processo moroso. A criatividade associada à composição musical deve ser fomentada, seja individualmente ou em grupo, com ou sem formação musical. As novas tecnologias proporcionam atualmente suporte aos processos de escrita: recorrendo a instrumentos MIDI é possível proceder à escrita automática utilizando *software* apropriado. Há ainda aplicações informáticas dedicadas à composição que reproduzem os sons indicados na partitura escrita, facilitando de certa forma o trabalho do compositor. Considera-se ainda assim que o potencial da tecnologia neste campo não está ainda a ser devidamente explorado. A exploração de *software* no auxílio do ensino da música também se encontra ainda bastante aquém das metas possíveis e faltam aplicações que não só auxiliem a escrita mas que a façam por inteiro, deixando a tarefa de verdadeira criação artística ao compositor.

Coloca-se então a hipótese de juntar ambos os pontos: um *software* de auxílio ao canto e que permitisse também a escrita musical. Ao identificar as notas cantadas, o intérprete teria uma verdadeira noção do que estava a fazer e o professor teria um suporte visual para traduzir um fenómeno sonoro. O SingingStudio é um programa que faz esse reconhecimento de notas, assim como a representação do espectrograma do som recolhido. A este programa prevê-se integrar um sistema de transcrição musical automática, explorando os diferentes algoritmos de discriminação de nota passíveis de implementar. Será necessário também a implementação de um metrónomo e a procura de estratégias que procurem ir de encontro não só ao que foi cantado mas ao que deveria ser registado na partitura de forma a reconhecer a diferença, por exemplo, entre a duração exata da nota cantada e a duração suposta. A partitura deverá apresentar-se com um grafismo de qualidade superior com a possibilidade de ser aplicada em *workshops* ou aulas de canto ou outro ramo musical.

Neste trabalho de dissertação procurou dar-se seguimento a estas preocupações, tendo-se implementado sobre a plataforma base do *software* interativo SingingStudio um módulo que realiza a referida transcrição automática da voz cantada. Descrevem-se também os testes realizados para aferir do correto funcionamento e desempenho das funcionalidades implementadas. Termina-se o documento olhando em retrospectiva o trabalho desenvolvido e sugerindo desenvolvimentos futuros, indicando as respetivas vantagens.



# Abstract

A music score is something that, although common among musicians and essential on the other hand, demands for some musical knowledge to read or write, therefore limiting this useful system to the educated musician alone, but takes some time as well. The creativity associated with music composing must be fed, either individually or within a group, with or without superior musical knowledge. New technologies have been helping music writing and changing our way of doing it: when using MIDI instruments it is possible to write music automatically through the appropriate software. There are even some applications dedicated to music composition that play the written score, making the composer's work easier. Still, informatic's role on this field is considered far from its huge potential. There is also much to be explored concerning software dedicated to music teaching and we lack of applications that not only help writing but also do it fully, leaving the composer with no more than the task of true artistic creation.

We now consider joining both fields: a software that helps singing and that also makes music writing much easier. When identifying the notes he or she is singing, the musician would have a real acknowledgement of what he was doing and the teacher would have some visual means of translating a sound phenomenon. SingingStudio is a software that does precisely that kind of note recognition as well as the spectrogram representation of the recorded sound. The goal is to add an automatic musical transcription system to it and explore the different algorithms to distinguish between note values that can be used. It is also necessary to implement a metronome and to search strategies that guess what is to be written rather than just use what was recorded regardless of the singers intention and tries to recognise, for example, the difference between the exact duration of a note and its supposed value. The score must be presented with superior quality images looking forward to applications in workshops or singing classes or even other areas of music.

On this thesis one tried to follow the mentioned concerns, having built over the interactive software SingingStudio, a module that transcribes automatically the singing voice. Tests to the developed product are also described in order to evaluate its performance. At the end, the entire work is analyzed and some future developments are suggested, indicating their advantages.



# Agradecimentos

Deixo um agradecimento ao prof. Aníbal Ferreira pelo constante apoio dado ao longo do trabalho realizado, cuja atitude dinâmica foi fonte de motivação. Gostaria também de agradecer ao Ricardo Sousa, ao Vitor Almeida e ao Tiago Campos pela constante companhia e pela disponibilidade no apoio sempre que necessário.

Este trabalho não seria também possível sem o constante apoio da família, amigos e namorada que também contribuíram com tudo o que podiam, nomeadamente com a cedência da sua voz para a realização de testes sempre que necessário. Outros contributos essenciais nestes últimos tempos que merecem menção foram os do João Terleira que disponibilizou a sua voz de cantor para contribuir no período de testes, do Tomé Salvaterra pelos esclarecimentos na programação e da minha irmã pelo seu sentido estético valioso na criação das imagens.

O Autor





*“Se não consegues explicar de forma simples  
É porque não percebeste suficientemente bem.”*

Albert Einstein



# Conteúdo

<b>1</b>	<b>Introdução</b>	<b>1</b>
1.1	Motivação . . . . .	1
1.2	Objetivos e Aplicações . . . . .	2
1.3	Estrutura da Dissertação . . . . .	3
1.4	Enquadramento . . . . .	3
1.5	Ferramentas a utilizar . . . . .	3
1.5.1	SingingStudio . . . . .	3
1.5.2	Qt . . . . .	4
<b>2</b>	<b>Revisão Bibliográfica</b>	<b>7</b>
2.1	O Som e o Ser Humano . . . . .	7
2.2	Introdução à escrita musical . . . . .	9
2.2.1	História da escrita musical . . . . .	9
2.2.2	A escrita musical moderna . . . . .	11
2.3	Análise de <i>software</i> existente . . . . .	13
2.4	Conclusão . . . . .	17
<b>3</b>	<b>Criação da Partitura</b>	<b>19</b>
3.1	Requisitos . . . . .	19
3.2	Desenvolvimento do <i>software</i> . . . . .	21
3.2.1	Análise aprofundada dos módulos existentes . . . . .	21
3.2.2	Estrutura de dados . . . . .	22
3.2.3	Metrónomo . . . . .	25
3.2.4	Escolha da figura musical . . . . .	26
3.2.5	Imagens . . . . .	32
3.2.6	Outras características e principais dificuldades . . . . .	33
3.3	Conclusão . . . . .	34
<b>4</b>	<b>Resultados</b>	<b>35</b>
4.1	Testes e análise de resultados . . . . .	35
4.2	Resumo e conclusões . . . . .	41
<b>5</b>	<b>Conclusões finais e Trabalho futuro</b>	<b>43</b>
5.1	Satisfação dos Objectivos . . . . .	43
5.2	Trabalho futuro . . . . .	44

<b>A</b>	<b>Código elaborado</b>	<b>47</b>
A.1	Metronomo.cpp . . . . .	47
A.2	Playbeep.cpp . . . . .	49
A.3	Partitura.cpp . . . . .	51
	<b>Referências</b>	<b>70</b>

# Lista de Figuras

1.1	Pormenor da janela principal do SingingStudio . . . . .	4
2.1	Esquema representativo do ouvido humano(Henrique, 2002) . . . . .	9
2.2	Exemplo de virga e punctum . . . . .	10
2.3	Tipos de Compasso . . . . .	12
2.4	Programa Encore da GVOX . . . . .	15
2.5	Programa Sibelius . . . . .	16
2.6	Programa Music Masterworks . . . . .	16
3.1	Pormenor da delimitação da nota no SingingStudio . . . . .	23
3.2	Medidas em pixels utilizadas na partitura . . . . .	25
3.3	Janela do metrónomo . . . . .	26
3.4	Diagrama de Funcionamento do Metrónomo . . . . .	27
3.5	Representação esquemática da amostragem por frames . . . . .	28
3.6	Janela com as opções de áudio do SingingStudio . . . . .	29
3.7	Esquema da divisão temporal das figuras . . . . .	29
3.8	Algoritmo da impressão das notas na partitura . . . . .	32
3.9	Representação PNG das figuras . . . . .	33
3.10	Comparação entre as imagens PNG e SVG . . . . .	34
4.1	Partitura inicial do primeiro teste . . . . .	36
4.2	Teste 1 - resultado obtido pelo SingingStudio . . . . .	36
4.3	Teste 1 - transcrição obtida . . . . .	36
4.4	Teste 2 - partitura inicial a reproduzir . . . . .	37
4.5	Teste 2 - resultado da voz interpretada pelo SingingStudio . . . . .	37
4.6	Teste 2 - partitura transcrita da voz cantada . . . . .	38
4.7	Teste 2 - resultado da melódica interpretado pelo SingingStudio . . . . .	38
4.8	Teste 2 - transcrição obtida da melódica . . . . .	38
4.9	Teste 3 - partitura inicial da melodia . . . . .	39
4.10	Teste 3 - sinal cantado interpretado pelo SingingStudio . . . . .	40
4.11	Teste 3 - Partitura transcrita da voz cantada . . . . .	40
4.12	Teste 3 - sinal tocado na melódica . . . . .	41
4.13	Teste 3 - Partitura interpretada da melódica pelo SingingStudio . . . . .	41
5.1	Comparação entre colcheias ligadas e não ligadas . . . . .	45



# Lista de Tabelas

2.1	Figuras Musicais . . . . .	12
2.2	Comparação do <i>Software</i> Existente . . . . .	14
3.1	Comparação do tempo da semifusa para diferentes frequências de amostragem . .	30





# Abreviaturas e Símbolos

ARTTS	Assistive Real-Time Technology in Singing
BPM	Beats Per Minute
FCT	Fundação para a Ciência e Tecnologia
MIDI	Musical Instrument Digital Interface
GUI	Graphical User Interface
PDF	Portable Document Format
PNG	Portable Network Graphics
QWT	Quaternion Wavelet Transform
SQL	Structured Query Language
SVG	Scalable Vector Graphics
XML	Extensible Markup Language



# Capítulo 1

## Introdução

### 1.1 Motivação

A música é uma arte em constante evolução. Consegue ser um desafio para o ser humano na exigência da sua interpretação, seja na afinação das notas do violino ou na fusão de um quarteto “*barbershop*”. A época moderna trouxe com a eletroacústica um novo desafio para a engenharia na música mas não é a primeira vez que tal acontece. Esta arte sempre desafiou a engenharia com a construção de mais e melhores instrumentos. O mecanismo de percussão nas cordas de um piano em contraponto com o mecanismo de dedilhação do cravo é um exemplo da forma como a engenharia é essencial na evolução da música. Hoje colocam-se constantes desafios na área da eletrônica musical mas, apesar de ser uma área que floresce, não é a única que precisa de avanços essenciais.

O trabalho de um compositor é fundamental para proporcionar a boa música, tanto ou mais importante que o do intérprete. É ele que estuda o impacto da música no ouvinte e que explica ao intérprete como o fazer; é ele, portanto, o criador. A composição musical implica um exercício complexo de imaginação e de experimentação. Atualmente é auxiliado por *software* que permite esta escrita mas que pode tornar o processo igualmente lento, apenas com a vantagem de poder ouvir o resultado no imediato. O improviso continua sem ser salvaguardado nesta situação. É necessário gravar a melodia improvisada e daí retirar “de ouvido” as notas para posteriormente as passar para uma partitura. Um dos avanços mais interessantes neste campo seria obter uma partitura diretamente do som. Se um grupo ou solista improvisasse e precisasse de se lembrar dessa mesma improvisação para a registar numa partitura, até que ponto se manteria a pureza do improviso? Se o compositor pudesse entoar as várias partes que compõem a sua orquestração e assim a escrevesse automaticamente, não traria tal projeto celeridade à composição? Qualquer intérprete compositor gostaria de poder compor somente no seu instrumento. E se fosse possível providenciar um sistema que, tanto quanto possível, o possibilitasse?

## 1.2 Objetivos e Aplicações

O trabalho desta dissertação consiste na criação de um sistema que permita a escrita musical a partir do reconhecimento de um som captado. Baseando-se num programa pré-existente que contém um algoritmo de reconhecimento de melodias cantadas, inclui-se uma funcionalidade de interpretação dos registos efetuados para transformação em notação musical, tentando aproximar o resultado o mais possível do desejado pelo intérprete. Posteriormente, este poderá fazer algumas alterações se necessário, assim como a impressão da partitura, a exportação em PDF ou ainda a conversão para MIDI para que possa ser aberto por outros programas, nomeadamente por alguns que permitam uma edição mais profunda da partitura. Desde logo, tal programa será útil para escrever partituras de solistas. No entanto, para adaptar a solução para grupos independentemente do seu tamanho, basta colocar um microfone para cada instrumento e executar o programa sequencialmente. Executado este processo, basta compilar o resultado numa só peça. Com este método, a composição deixa de ser limitada pela instrução na notação, passando a estar ao alcance de todo o cantor.

O *software* original tinha já como objetivo ser um auxílio no ensino do canto e é algo que esta funcionalidade complementar virá reforçar. A sua utilização poderá figurar na comparação entre o que o aluno de canto deveria cantar e aquilo que foi efetivamente transcrito pelo método automático. Com este método será possível obter um *feedback* visual da melodia entoada, facilitando a identificação dos erros cometidos. Um aluno iniciado tem frequentemente problemas desta natureza, mas o aluno mais experiente depara-se com outro tipo de questões mais relacionadas com o domínio da interpretação. Neste contexto, a transcrição automática pode também dar a sua contribuição devido à sua extrema minúcia na deteção das notas musicais. Certos aspetos específicos do canto como o *legato*, o *staccato* ou a coloratura poderão beneficiar do reconhecimento efetuado pelo programa em utilização, demonstrando ou não pausas entre as notas. Na utilização do *legato* - uma forma de canto que consiste numa continuidade da linha vocal - não deverá haver pausas enquanto no *staccato*, em antítese do primeiro, que se baseia na interpretação de notas mais curtas do que realmente são, as pausas deverão salientar-se. A coloratura é um outro efeito que deverá ser feito sem transições bruscas entre frequências, no qual esta aplicação terá um contributo semelhante ao do *legato*.

O programa de base inclui outras funcionalidades que podem ser igualmente úteis para os cantores aquando da utilização da componente da transcrição. Pondera-se uma aplicação desta tecnologia na Casa da Música precisamente para ver os seus resultados aplicados em *workshops* de canto. A formação musical é também um dos alvos deste programa que permite comprovar a aplicação de cada figura musical, permitindo ao iniciado compreender na prática a dimensão de cada uma e a sua contribuição para a partitura final. Apesar de ser específico para o canto, o programa reconhece também instrumentos monofónicos sejam sopros, cordas, teclas ou percussão de altura definida, com a condição de se tocar apenas uma nota de cada vez. A sua adaptação para a polifonia está prevista para trabalho futuro.

## 1.3 Estrutura da Dissertação

Para além da introdução, esta dissertação contém outros quatro capítulos principais. No primeiro é realizada uma análise contextual sobre a qual recai a necessidade da solução a desenvolver e foram apresentadas aplicações possíveis para a tecnologia da transcrição.

No segundo capítulo é feita uma descrição da evolução da escrita musical assim como do modelo atual, uma vez que se tratam de conceitos essenciais para a compreensão do desenvolvimento do projeto. É descrito o estado da arte, analisando-se programas semelhantes ou com funcionalidades cuja exploração seja relevante para perceber quais são aquelas que se traduzem numa mais-valia para a implementação. São exploradas as linguagens de programação e extensões aplicadas, assim como as plataformas utilizadas durante o trabalho realizado na dissertação.

No terceiro capítulo, faz-se uma descrição e análise aprofundada do trabalho desenvolvido, salientando os requisitos que se concluíram essenciais e todas as dificuldades encontradas ao longo do desenvolvimento. Explica-se ainda todo o trabalho efetuado e faz-se uma retrospectiva da sua evolução.

O quarto capítulo incide sobre os testes feitos para comprovar as medidas adotadas na discriminação temporal das notas e aqueles que foram feitos entretanto ao *software* desenvolvido. São apresentados os resultados obtidos, comparados com o resultado teórico, analisadas as diferenças e é feita uma crítica dos mesmos.

O quinto e último capítulo serve de conclusão ao trabalho realizado, fazendo uma reflexão sobre todo o projeto e procurando os pontos cujo desenvolvimento poderá ser continuado, se for pretendida a exploração posterior do software indo de encontro ao seu potencial .

## 1.4 Enquadramento

Esta dissertação está inserida num projeto da FCT denominado “Assistive Real-Time Technology in Singing”(ARTTS). A parte da qual decorre esta dissertação corresponde à tarefa 3: Transcrição de Canto para Pauta Musical e Composição (“TASK3 - singing to musical score transcription and music composition”).(Ferreira, 2010) A evolução desta parte do projeto será acompanhada e posteriormente continuada por um bolseiro da Universidade Católica Portuguesa, que servirá de ligação com a Casa da Música e será coordenador da sua aplicação nesta instituição. O projeto ARTTS começou em outubro de 2010 e tem a duração de 3 anos, apesar do Singing Studio ter sido desenvolvido, previamente, em 2004.

## 1.5 Ferramentas a utilizar

### 1.5.1 SingingStudio

O SingingStudio não pode ser considerado somente uma ferramenta uma vez que ele, mais que isso, é a base de trabalho. Este *software* foi um dos frutos da SEEGNAL Research, uma empresa criada em 2004 como resultado de um *spin-off* da Faculdade de Engenharia da Universidade do

Porto e do INESC-Porto. Trata-se de um programa de reconhecimento melódico que permite gravar e reconhecer sons musicais que podem, posteriormente, ser transportados para MIDI.

O algoritmo de reconhecimento - *Searchtonal* - foi desenvolvido pelo Professor Aníbal Ferreira. O SingingStudio grava a voz cantada e, por meio do *Serchtonal*, analisa os seus harmónicos para reconhecer com exatidão a frequência fundamental. (Ventura) Funciona também com outros instrumentos cuja extensão seja semelhante. Teria, por exemplo, dificuldades em lidar com um contrabaixo pela baixa frequência das suas notas para a qual o algoritmo não está adaptado. É mostrada a forma de onda em escala linear numa janela superior, a nota em reprodução numa pauta em *widget* anexo e uma esquematização dos harmónicos detetados. Após o processo de reconhecimento da frequência, é mostrada na janela principal uma sequência de pontos indicativos dessas mesmas frequências que são automaticamente ligados, criando uma linha progressiva das frequências entoadas. Estas frequências são então filtradas para distinguir quais correspondem a notas musicais efetivas, evitando pequenos ruídos de altura definida ou sons de transição. As notas são contornadas por retângulos azuis que facilitam a perceção visual. Tudo isto é representado na divisória mais ampla da janela principal do SingingStudio. Do lado esquerdo desta, numa posição vertical, temos um *widget* com a representação de um piano que reproduz um som ao ser pressionada uma das suas teclas virtuais com o rato. Cada uma dessas teclas delimita a área visual designada para cada nota, onde constará o retângulo respetivo. A figura 1.1 retrata um exemplo da janela principal deste programa, tendo reconhecido uma melodia. Este programa é completado pelas funcionalidades do VoiceStudio nas aplicações direcionadas para a análise e diagnóstico de sinais de voz.

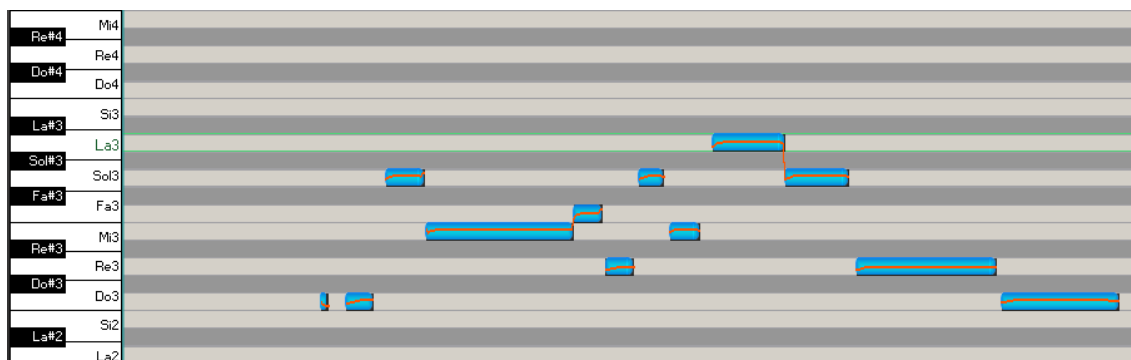


Figura 1.1: Pormenor da janela principal do SingingStudio

### 1.5.2 Qt

O SingingStudio utiliza o *framework* Qt para a criação dos efeitos gráficos e a sua compilação foi feita utilizando a versão 2005 do Microsoft Visual Studio em Windows. Utilizando a linguagem C++, esta ferramenta permite criar uma interface gráfica (GUI) facilmente compilável em qualquer plataforma. A Nokia, empresa possuidora do Qt, disponibiliza também algumas ferramentas para facilitar a sua utilização como o QtDesigner, QtAssistant, QtLinguistic e QtCreator. O primeiro

torna muito intuitiva a criação de janelas pela disponibilização de determinados componentes pré-feitos, resultando num ficheiro de tipo *form* com extensão “.ui”. É também neste componente que está definido o ficheiro da janela principal que contém o único bloco com a biblioteca QWT: o da forma de onda. O QWT é uma biblioteca utilizável no Qt que contém componentes GUI e classes de utilização que são úteis principalmente para programas de índole técnica relacionados com a visualização de gráficos. O QtAssistant é utilizado para o menu de ajuda e o QtLinguistic para facilitar as mudanças de língua no *software*. O último, o QtCreator, permite analisar o código interpretando-o e tornando-o mais claro para o utilizador, englobando também os restantes programas assim como um compilador.(Molkentin e Inc., 2007)

É de ressaltar que o ambiente gráfico do SingingStudio, apesar de usar Qt, não está totalmente adequado ao QtDesigner pelo que para o modificar é necessário analisar o código manualmente, tornando o processo mais moroso. Pressupõe-se que esta divergência daquele que é aparentemente o caminho mais fácil decorre da utilização da biblioteca QWT num dos elementos utilizados na janela principal e que limita a sua utilização em QtDesigner, assim como a relação ente os botões da barra de ferramentas que, ao implementar diretamente no código, será mais fácil criar condições de funcionamento entre eles por intermédio de um gestor de eventos (*event handler*). O Qt inclui módulos específicos para determinadas áreas como a criação de uma interface gráfica, o tratamento de imagens SVG ou a utilização de SQL.(Blanchette e Summerfield, 2006)

Inicialmente ponderou-se a compilação deste *software* para MacOSX mas tal investimento foi abortado não pelas funcionalidades do Qt (que se adequam totalmente a tal operação) mas pelo tempo que demoraria a converter todas as bibliotecas específicas do programa. Utilizou-se então o ficheiro de projeto pré-existente, o que obrigou à utilização das mesmas ferramentas e sistema operativo, tendo o Qt sido atualizado para a versão 4.7.





## Capítulo 2

# Revisão Bibliográfica

O presente capítulo trata de introduzir o leitor a certos conceitos essenciais a respeito do som e do sistema de recepção e interpretação do ser humano. Faz-se uma iniciação à história da escrita musical para perceber a razão da sua existência e a dimensão da sua importância, assim como para distinguir o essencial do acessório. Resume-se depois o modelo moderno de escrita musical, aprofundando os componentes da partitura cuja inclusão no *software* se concluiu fundamental. Termina-se com uma análise de *software* existente cujas funcionalidades demonstrem alguma utilidade para o programa a desenvolver e que permitam analisar as principais falhas a colmatar.

### 2.1 O Som e o Ser Humano

O som é a percepção que temos ao receber no cérebro impulsos nervosos provenientes da parte periférica do sistema auditivo humano como consequência da chegada de uma onda sonora. Mais especificamente, trata-se da própria interpretação cerebral e não da onda sonora. É importante esclarecer esta diferenciação: a onda sonora é a vibração das partículas de um meio causando nós e ventres de pressão - uma onda mecânica, portanto - e apenas quando essa onda encontra o tímpano e tem algum efeito no cérebro é que se obtém o som. (Levitin, 2006) O som abunda no nosso dia a dia, seja com sons de alerta dos mais diversos aparelhos, ruídos variados, como forma de arte ou de comunicação. Apenas quando pensamos que lhe é exclusivamente dedicado um dos cinco sentidos do ser humano conseguimos compreender verdadeiramente a sua importância. As duas utilizações do som que mais facilmente lhe são associadas são a fala e a música. A fala é vista como a forma mais prática de estabelecer comunicação por ser tão imediato para o ser humano a utilização da voz. A música é a outra face do som, mais exigente para o ouvido e cuja comunicação se encontra mais subjacente. Esta pode ainda estar associada ou não à voz. Tendo alguns pontos em comum, torna-se então necessário clarificar a diferença entre a fala e a música. Apenas são audíveis ondas sonoras com frequências entre os 20Hz e os 20kHz, e a sua intensidade é igualmente um requisito à sua condição de perceptível. É a relação harmónica entre os parciais

do som que lhe vai conferir a denominação de som de altura definida. Um som é constituído por uma frequência fundamental (chamada  $f_0$ ) mas, geralmente, também por outras ondas em número teoricamente infinito cujas frequências estão relacionadas com a fundamental. O harmónico de um som musical terá uma frequência que consiste num número inteiro de vezes superior à frequência fundamental. Henrique (2002) A título de exemplo, um som produzido pela queda de um livro não será de altura definida devido à relação variável dos seus harmónicos. A relação entre os harmónicos é dos mecanismos mais complexos da física do som pois define o conceito de timbre. Prova da dificuldade de compreensão do timbre é a própria definição da palavra pela American Standards Association: o atributo sensitivo que permite a distinção entre dois sons com a mesma intensidade e altura sonora (a altura sonora é o efeito psicoacústico causado pela variação da frequência); ou seja, a componente do som que não é nenhuma das que conseguimos efetivamente definir. Se quisermos fazer a analogia da audição à visão, o timbre seria a cor que difere os instrumentos musicais. Ao abordarmos a captação de trechos musicais por *hardware* temos de ter noção do funcionamento do sistema auditivo humano e de alguma física subjacente ao próprio som porque é isto que o sistema artificial terá de imitar. Por outro lado, a própria interpretação do som captado terá de ter em conta o aparelho auditivo humano porque o intérprete se fia instintivamente no *feedback* que tem de si próprio relativamente à sua interpretação. Ou seja, o que ouve e o que toca ou canta estão intrinsecamente relacionados.

Abordemos então o aparelho auditivo humano, baseando-nos no esquema apresentado na figura 2.1. A onda sonora atravessa o ouvido externo constituído pelo pavilhão e embate no tímpano, uma membrana oval, fazendo-o vibrar. Este movimento vibratório é depois transmitido para uma cadeia de três ossos de reduzida dimensão - os menores do corpo humano: o martelo, a bigorna e o estribo, também chamados ossículos - que o amplificam e transmitem para a janela oval. Considera-se a zona desde o tímpano até à janela oval como o ouvido médio. A janela oval é um pequeno orifício, uma janela portanto, que contém uma membrana. Esta amplifica também o sinal e encaminha-o para a cóclea, uma estrutura encaracolada de 3 a 4 centímetros que possui uma membrana no seu interior (membrana basilar), dividindo-a internamente em duas partes iguais. É esta estrutura que transmite o movimento vibratório em sinais elétricos através de processos químicos sobre os líquidos contidos em ambas as partes da cóclea. Os sons mais graves são processados no fim da cóclea (o que faz sentido, tendo em conta o seu comprimento de onda superior) na zona mais encaracolada e os mais agudos na zona inicial.

É até possível conceber uma analogia entre os sistemas de audição humana e um sistema artificial de análise sonora, em que neste último o microfone será o tímpano (composto igualmente por uma membrana) enquanto um processador cumprirá o papel da interpretação feito pelo cérebro. Mais à frente trataremos de forma mais dedicada o *software* necessário, assim como um possível método de reconhecimento e de transcrição.

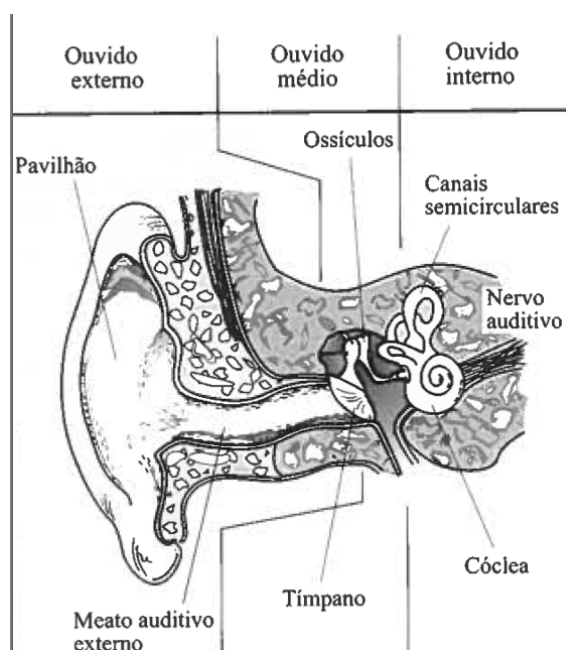


Figura 2.1: Esquema representativo do ouvido humano(Henrique, 2002)

## 2.2 Introdução à escrita musical

### 2.2.1 História da escrita musical

A escrita musical surge como registo de uma música que essencialmente terá de conter notas. Para cada uma, é necessário especificar a altura do som e a duração da nota. No entanto poderá ter outras indicações que sugiram a intensidade, o andamento, a expressão ou questões específicas relativas à execução em determinados instrumentos. A utilização destes escritos nem sempre foi muito ampla visto que a música era passada frequentemente de boca em boca e atravessava gerações nessa mesma base de transmissão de conhecimento. Poderá então surpreender que os primeiros sistemas de notação de que há registo sejam originários da Grécia Antiga e remontem a 600 a.C. . Estes eram compostos por símbolos que indicavam os dois parâmetros já referidos essenciais de definição das notas - a altura e a duração. Este sistema utilizava dois tipos diferentes de letras, uma para a música vocal e outra para a instrumental. Segundo o epitáfio de Seikilos em cada excerto é indicado a nota musical, o texto correspondente, o ritmo da música e o ritmo do texto. Eram também utilizados dois símbolos diferentes para representar as partes melódica e rítmica, perfazendo um conjunto muito vasto de símbolos necessários para a compreensão e consequente reprodução de uma melodia. Já aqui era possível empregar acidentes, atualmente os sustenidos e bemóis. Na altura, essa modificação da nota era feita escrevendo a letra na horizontal.

Boécio, filósofo romano, utilizou as primeiras 15 letras do alfabeto grego para denominar as notas que se usavam no fim do período romano. Já na idade Média, por volta de 600 d.C., o Papa Gregório Magno tentou uniformizar o canto religioso criando um Antifonário (compilação de cânticos católicos) assim como o canto Gregoriano, trabalhos que foram continuados especialmente

no papado de Carlos Magno (800 d.C.).(Grout e Palisca, 1988) O Canto Gregoriano é uma forma de cantochão: uma melodia monofónica que oscila em torno de uma nota principal em pequenos intervalos de um ou dois tons, sendo portanto um tipo de música pouco exigente no que toca aos requisitos de notação musical. O uso de escrita musical não era popular por esta altura. Como tal, a responsabilidade pela tradição oral recaía sempre sobre alguém com um certo grau de especialidade para o efeito e a composição de novas melodias era feita com base no conhecimento vasto de fórmulas pré-existentes. Para os católicos, este espírito de responsabilidade para com a tradição era baseado numa sacralidade associada à tradição gregoriana mas trazia sérias dificuldades na exigência requirida ao cantor solista, por exemplo, que precisava de memorizar todo o repertório litúrgico, processo que demorava cerca de uma década.

Foi talvez devido à necessidade de uniformização dos cânticos no mundo ocidental e da grande extensão do repertório que a escrita musical ganhou nesta altura alguma importância. Com o crescente número de melodias que apareciam nesta altura, a fiel memória dos cantores deixou, por si só, de ser suficiente. O ritmo era ditado pela prosódia, não carecendo assim a partitura de figuras muito complexas e os mosteiros tiveram aqui um papel central pela sua função de copistas e pela contribuição na evolução tanto na notação como na teoria musical. A representação de símbolos por cima do texto que sugeriam a evolução da melodia denominados neumas começa a ser difundida para ajudar a lembrar a melodia. Numa época inicial, estes não exigiam uma nota exata e pressupunham um conhecimento prévio da melodia. Os neumas derivavam dos sinais de acentuação da linguagem, sendo os mais famosos a virga e o punctum (ver figura 2.2). Tais sinais indicavam simplesmente que a nota seria respetivamente mais aguda ou mais grave que a anterior e bastava aglomerar estes sinais em pequenos grupos de 2 ou 3, distinguindo assim os neumas simples (utilizados sem aglomeração) dos compostos (aglomerados). Os próprios neumas sofreram uma grande evolução. A princípio a notação consistia em pequenas anotações gráficas, por norma curtas linhas curvas, que tentariam criar uma analogia com a evolução da linha melódica mas com o passar do tempo optou-se pela notação diastemática que permitia a distinção visual da altura dos sons. Foi também por esta altura que se deu a passagem para a notação quadrada, distinguindo a notação neumática do século IX da posterior, por volta do século XIII. A parte principal dos neumas na notação quadrada deve a sua forma ao material de escrita dos monges: a pena, ao ser cortada na ponta, ficaria com uma secção aproximadamente quadrada e recriava essa mesma forma geométrica ao ser tocar no papel com a sua tinta. Estes símbolos representativos das notas eram dispostos acima ou abaixo de uma só linha de referência, inicialmente a nota Fá de cor vermelha. Posteriormente surge uma segunda linha, amarela, correspondente ao Dó.



Figura 2.2: Exemplo de virga e punctum

A Idade Média é um vasto período da história que abrange desde o século V ao XV. Tendo















sido o período no qual se deu o maior salto na evolução da escrita musical, é também normal que esta escrita tenha variado bastante neste período tão abrangente. Com o passar do tempo o monge Beneditino Guido D'Arezzo introduziu o uso da pauta de 5 linhas tendo aconselhado o uso de cores diferentes, sediando as fundações da base da notação musical moderna. Colocou também letras em determinadas linhas, indicando a sua frequência e, por consequência, a das restantes linhas e espaços. Estas evoluíram para signos mais semelhantes à escrita neumática que entretanto foram precedidos pelas claves dos dias de hoje. Este teórico deu também às notas musicais os nomes por que as conhecemos hoje, tendo-se para isso baseado num hino a S. João Batista. É já numa época tardia da Idade Média que é adotada em França uma pauta de 5 linhas enquanto em Itália se populariza o uso de uma de 6 linhas. No entanto, a utilização do pentagrama vingou na Europa a partir do século XVI, estabilizando a pauta na versão que se utiliza atualmente. A escrita só então, devido não só à estabilização deste sistema mas também à complexificação da polifonia, se tornou indispensável. Foi também na Idade Média que surgiu a tablatura especialmente para as cordas mas aos dias de hoje só chegou aplicada à guitarra clássica. (Borges e Cardoso, 2003)

### 2.2.2 A escrita musical moderna

A evolução da escrita musical ocidental culminou num sistema que conjuga a duração com a sua altura num só parâmetro e que prima pela simplicidade do grafismo. Foram muitas as mudanças até se concluir que tal sistema seria o mais prático. O processo mais comum na nossa escrita musical baseia-se na partitura que para além da pauta - um simples conjunto de 5 linhas e 4 espaços que serve de base à notação musical - contém todas as notas, pausas e muitos outros símbolos de significado musical relevante. É neste preceito que se apoiam os atuais músicos seja para disponibilizar a sua música de forma pessoal ou comercial, seja para aprender ou para ensinar. Este registo, apesar de o considerarmos exato, quase ideal e relativamente intuitivo ao mesmo tempo, não deixa de ser um tipo de escrita que exige alguma formação musical e que é, para o criador menos experiente, um processo moroso. A música faz-se de sons e de silêncios representados por figuras musicais que indicam som durante um determinado intervalo de tempo e pausas associadas às figuras com igual valor em silêncio. As referidas figuras são as que se ilustram na tabela 2.1. A sequência destas representações gráficas indica o ritmo que, por sua vez, não é mais do que uma representação da duração temporal das notas, ou seja, distribuem-se na partitura formando um gráfico de duas dimensões que relaciona a altura do som com o tempo.

É lógico inferir que a transcrição musical está intimamente dependente do reconhecimento melódico e que para transcrever a música de forma fiável, é necessário começar por uma definição do tempo em bpm (batimentos por minuto). Um dos aspectos relevantes para entender a transcrição que será abordada nesta dissertação é a divisão em compassos, da qual dependerá a distribuição das figuras na partitura. A escolha do tipo de compasso e a divisão musical a ela associada depende somente da interpretação a dar à linha melódica em questão porque condicionará os tempos fortes da mesma, ou seja, a sua acentuação. Curiosamente, o mesmo trecho melódico pode ser transcrito em resultados diferentes, o que poderá resultar numa interpretação posterior que não corresponde ao desejo inicial.

Tabela 2.1: Figuras Musicais

Nome	Figura	Pausa	Tempo
Semibreve			4
Mínima			2
Semínima			1
Colcheia			$\frac{1}{2}$
Semicolcheia			$\frac{1}{4}$
Fusa			$\frac{1}{8}$
Semifusa			$\frac{1}{16}$

Examinem-se os 3 tipos de compasso mais frequentes e que serão utilizados no *software* produzido nos trabalhos desta dissertação: o compasso binário, o ternário e o quaternário. Cada um deles conterà dois, três ou quatro tempos respetivamente. Estes podem ser simples ou compostos, consoante sejam de divisão binária ou ternária. Em suma, a fracção do compasso define o seu valor indicando uma figura base com o denominador segundo a Tabela 2.1 e o numerador indica quantas figuras dessas deverão existir no compasso. Multiplicando o número de figuras pelo seu valor indicado na tabela obtém-se o valor total de tempos no intervalo do compasso.

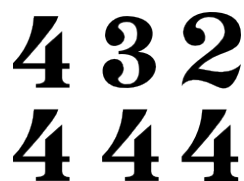


Figura 2.3: Tipos de Compasso

Olhemos ainda para o aspeto melódico. Certas frequências do espectro audível são indicadas como notas da escala musical igualmente temperada e o menor intervalo entre duas notas é chamado meio-tom ou semitom em Português do Brasil, vulgarmente encontrado na literatura). Estas frequências são escolhidas tendo como intervalo base a oitava: um intervalo de 12 meios-tons cuja frequência mais elevada é o dobro da inferior. A oitava é o intervalo entre duas notas com o mesmo nome e que estão no limite de uma escala. Os 12 meios-tons entre elas são distribuídos entre as sete notas que a compõem, utilizando cinco intervalos de um tom e dois de meio-tom.

Temos então as sete notas cujos nomes nos são normalmente familiares (dó, ré, mi, fá, sol, lá e si) e outras entre elas para as quais necessitamos de algo a que chamamos acidentes: o sustenido ( $\sharp$ ) e o bemol ( $\flat$ ). Estes símbolos aumentam e diminuem respetivamente meio-tom à nota na qual são

aplicados. Para os anular utiliza-se o bequadro (♯). A escala é um arquétipo de tons e meios-tons, variando consoante o tipo de escala (maior, menor, cigana, húngara, de blues, e muitas outras). É normalmente constituída por 7 notas com intervalos entre elas que se mantêm constantes em determinado tipo de escala, independentemente da nota inicial (chamada tónica e que dá nome à escala) como se de uma forma se tratasse. Para identificar a escala basta identificar a relação entre as suas notas indicando os sustenidos e bemóis que esta deve ter junto à clave naquilo a que se chama armação de clave. Se se ordenar todas os meios-tons entre um intervalo de oitava, obtém-se uma escala cromática.

Após esta explicação, é então bastante fácil encontrar vantagens num *software* que permita a transcrição da voz cantada ou de outro instrumento musical monofónico para uma partitura musical. Não só permitiria um avanço no ensino da música como também facilitaria a escrita musical tornando-a mais acessível a todos os que queiram compor bastando para isso entoar a melodia desejada.

## 2.3 *Análise de software existente*

Os sistemas musicais interativos existem nas mais variadas formas o que exige uma consideração especial por determinadas características que consideramos importantes quando se pretende agrupá-los em categorias. (Rowe, 1993) Optou-se por analisar 2 tipos de programas: aqueles que relacionavam a captação de trechos musicais com a gravação e aqueles que frisavam a escrita com detalhe ignorando outras funcionalidades. Essa divisão é clara na tabela 2.2. Como o primeiro tipo de programa permite a leitura de MIDI, pode obter-se a transcrição musical de uma linha melódica gravando-a neste formato, usando para tal o outro tipo de *software* analisado. No entanto, falta uma integração entre as 2 técnicas que otimize o processo e o facilite para o utilizador. Não só permitiria efetuar a transcrição com a utilização de 1 só programa em vez de 2 mas também abria portas à correção e adaptação da linha melódica captada à transcrição desejada.

A tabela 2.2 indica se o programa efetua gravação de som que depois identifica e se exporta esse ficheiro em MIDI. No que toca à escrita musical, indica-se se o *software* analisado mostra a nota numa pauta seja apenas temporariamente e uma nota de cada vez ou a partitura completa,

Atualmente existem alguns programas que reconhecem a nota captada quase imediatamente ou previamente gravada num ficheiro de som. A robustez desse reconhecimento é muito variável devido à complexidade do processo que está por trás desta funcionalidade. Os programas mais promissores neste campo são o Music Masterworks e o Sing & See. Fez-se, neste ponto, uma crítica ao *software* analisado, concluindo-se que nenhum tem a precisão necessária para uma utilização eficiente do ponto de vista da produtividade musical. Apesar do seu interesse inovador no campo da utilização de tecnologias na música, não está ainda no ponto em que permite uma utilização geral e despreocupada por qualquer utilizador. É de sublinhar que apenas uma pequena parte do *software* experimentado realizava algum tipo de transcrição musical e nunca de forma totalmente satisfatória (excetuando a captação por instrumentos MIDI), mostrando que neste campo

Tabela 2.2: Comparação do *Software* Existente

Nome	Gravação	Gravar em MIDI	Mostrar Nota	Partitura Editável	Mostrar Teclado	Pontuação	Transformar de WAV
<b>SingingStudio</b>	✓	✓	✗	✗	✓	✗	✓
Sing & See	✓	✗	✓	✗	✓	✗	✓
Encore	✓	✗	✓	✓	✓	✗	✗
Music Masterworks	✓	✓	✓	✓	✗	✓	✓
Singstar	✓	✗	✗	✗	✗	✓	✗
Transcribe	✓	✓	✗	✗	✓	✗	✓
Twelve Keys	✗	✓	✗	✗	✓	✗	✓
Muse Score	✗	✓	✓	✓	✓	✗	✗
Carry-a-tune	✓	✗	✓	✓	✓	✗	✓
SlowMP3	✗	✓	✗	✗	✓	✗	✓
Sibelius	✗	✓	✓	✓	✓	✗	✗
Finale	✗	✓	✓	✓	✓	✗	✗

ainda é necessária a implementação de um método que cumpra por inteiro os requisitos mínimos dos utilizadores.

Explorando ambos os tipos de programa vemos que os de edição de escrita musical são especialmente dedicados à escrita de complexas partituras com recurso à reprodução imediata. São baseados maioritariamente em MIDI, permitindo a associação de diferentes instrumentos musicais para cada pauta - vulgarmente dito “para cada voz- facilitando assim a reprodução de partituras com múltiplas vozes. Precisam de ser altamente detalhados no que toca aos símbolos musicais e imitar com a maior naturalidade possível a musicalidade humana em indicações dinâmicas (como *accelerando*, *ritenuto*, entre outros). A título de exemplo temos um dos mais simples - o Encore - cujo ambiente gráfico está representado na figura 2.4. A diferença de complexidade entre este programa ou o Muse Score e outros de cariz profissional como o Sibelius ou o Finale no fim da tabela, é especialmente o número de sinais específicos que cada um possui e o seu design. No caso particular do Encore, a reprodução do documento indicado distancia-se claramente da perfeição. Os principais defeitos detetados foram o facto de ser demasiado brusco nas transições de tempo como é o caso do *accelerando* e alguns *bug's* no que toca a repetições de compassos ou pequenas alterações que saiam da normal “leitura corrida” de uma partitura.

Os programas Sibélius e Finale apresentam a partitura de uma forma muito mais cuidada e explorada ao pormenor. A sua dedicação à criação e ajustes da partitura conferiu-lhes a possibilidade da simplicidade, apesar do enorme número de itens que têm disponíveis. O design é extremamente apelativo, como se pode comprovar pela figura 2.5, e a performance da construção musical é excelente. Exigem algum tempo de ambientação para compreender todas as suas funções devido ao seu elevado número.



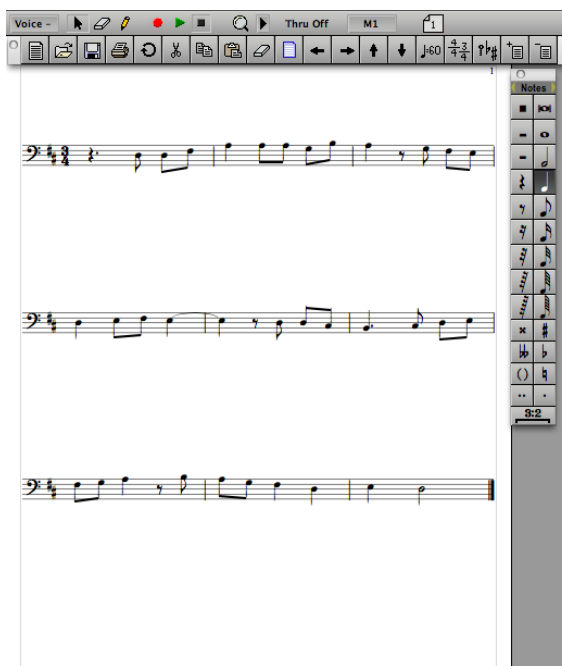


Figura 2.4: Programa Encore da GVOX

Qualquer partitura pode ser transferida de um programa de edição vulgar para outro semelhante aos dois acima referidos (e que são líderes de mercado nessa área) via MIDI, no entanto este formato regista uma quantidade de informação muito limitada. Falham-lhe pormenores como por exemplo a distinção entre dó $\sharp$  ou réb, dependendo da armação de clave, ligações entre notas, divisões de compassos e indicações de terminação de partitura ou repetição. Para colmatar essa falha, programas como o Muse Score utilizam MusicXML para servirem de alternativa de baixo custo aos programas mais populares e permitirem por outro lado a exportação para os seus concorrentes através desta linguagem de marcação baseada em XML. Apesar de o Finale conseguir importar ficheiros MusicXML, o Sibelius precisa que lhe seja adicionada uma pequena extensão gratuita.(Good, 2001)

O reconhecimento melódico efetuado pelo Sing & See foi considerado bastante fiável mas a sua aplicação não é fácil por lhe faltarem formas de exportação dos dados registados. Parece ser adequado a um registo meramente temporário apesar de ser de fácil compreensão e estar acompanhado de uma pequena pauta que indica a nota a ser registada naquele momento (e somente naquele momento) e um teclado que localiza a nota nas suas teclas, indicando-a com cor azul. Um fator diferenciador deste *software* é a inclusão de um pequeno ponto vermelho na frequência indicada para a última nota tocada, ainda que a melodia pare, e a capacidade de ignorar pausas. Este ponto poderia ser desastroso quando aplicado a um sistema de reconhecimento automático da transcrição musical mas para o efeito do Sing & See, que é apenas o de reconhecimento da nota com base na frequência gravada, parece agradar a muitos utilizadores. O Transcribe é um programa semelhante, que grava uma melodia e indica quais são as notas ou acordes a que deve corresponder. Apesar de ter a possibilidade de reconhecer acordes, a sua fiabilidade no reconhe-

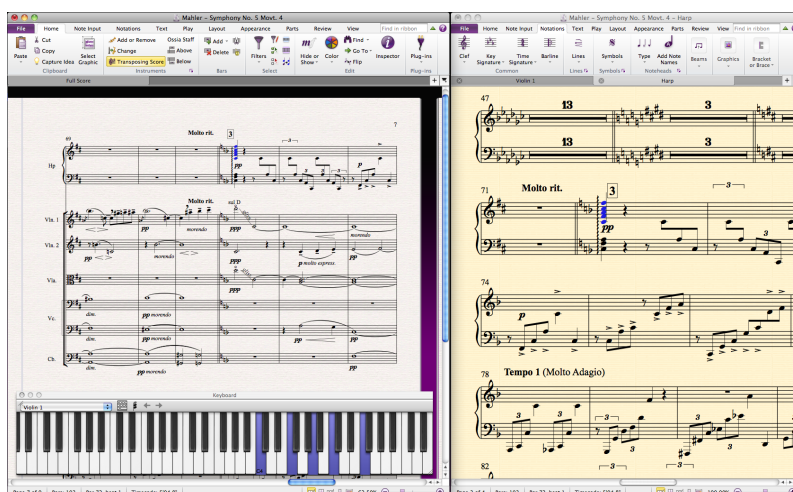


Figura 2.5: Programa Sibelius

cimento das notas não se mostrou tão boa como no Sing & See ou no SingingStudio. O programa mais promissor nas funções desejadas é o Music Masterworks. Com capacidade para registrar uma melodia monofônica e convertê-la para MIDI, apresenta as notas convertidas num ambiente muito complexo incluindo até vários botões semelhantes com diferentes funções, o que o torna de difícil utilização e a fiabilidade do reconhecimento é inferior à dos restantes programas com essa função. 2.6 Exige também certas condições para o seu correto funcionamento como a gravação do ficheiro “wav” para possibilitar a gravação. Ainda assim, consegue ser de todos os mais completo.

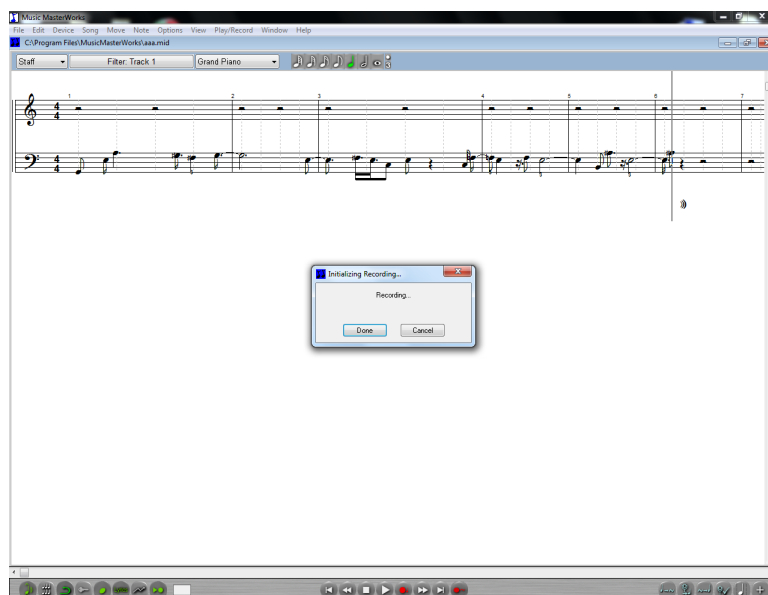


Figura 2.6: Programa Music Masterworks

Os restantes programas analisados têm as características indicadas na tabela comparativa mas não se salientam por nenhum aspeto significativamente relevante para a análise desejado.

## 2.4 Conclusão

Relativamente à escrita musical, a análise feita permitiu concluir que as figuras musicais do sistema atual são fruto da evolução dos diversos sistemas anteriores. Serviu também a análise para verificar quais os aspetos da escrita que se manifestam como uma real necessidade para uma transcrição musical como a desejada para o *software* a desenvolver. Existe algum software de reconhecimento musical que trabalha a partitura musical e outro que reconhece as notas cantadas (havendo ainda alguns que conseguem abordar ligeiramente ambas as funcionalidades) mas nenhum cuja simplicidade seja comparável à do SingingStudio. A robustez deste último permite potenciar ainda mais a funcionalidade da transcrição musical automática.



## Capítulo 3

# Criação da Partitura

Neste capítulo apresenta-se o programa desenvolvido e todas as dificuldades encontradas que importa mencionar, assim como os pontos principais sobre os quais assentou o trabalho desenvolvido.

### 3.1 Requisitos

Analisando-se o *software* existente no estado da arte, inferiu-se sobre as necessidades das funcionalidades da pauta musical. No programa utilizado foram então considerados alguns pontos que teriam de ser modificados para permitir a aplicação da transcrição para pauta musical e que serão enumerados seguidamente. Antes de mais, define-se um objectivo: pretende-se tornar o *software* numa plataforma que consiga gravar uma melodia monofónica, que a reconheça melódica e ritmicamente e que escreva a sua partitura, permitindo reproduzir esse resultado que poderá entretanto ter sofrido alterações para se aproximar dos desejos do intérprete. Tendo este marco como ponto de chegada podemos dividir os problemas de aquisição e transcrição em dois tipos: problemas rítmicos e melódicos.

As soluções passarão igualmente por dois processos: automático ou manual. O processo automático é preferível para os problemas sistemáticos e torna a utilização do programa mais simples. No entanto é menos fiável que a alternativa manual por obrigar à implementação de um conjunto de medidas que o utilizador pode não querer inteiramente. As características típicas da voz cantada produzem, por vezes, alguns erros no reconhecimento das notas, gerando um *output* afetado pelas limitações do processo automático. Para os corrigir, é necessário replicar a estrutura reconhecida pelo algoritmo do SingingStudio para, posteriormente, poder corrigi-la. A solução manual é a ideal para quem quer um resultado mais preciso. Apesar de exigir que o utilizador tenha noção das alterações a efectuar, permite que os resultados sejam utilizáveis no futuro com muito mais confiança. Preferir-se-á a solução automática devido à vasta gama de produtos de *software* de extrema qualidade que permitem a alteração (e criação) de partituras, sendo apenas necessária

a exportação no formato adequado. A componente manual seria adaptada para correções finais, caso o tempo o permitisse.

O programa regista a voz ou MIDI e tem de ser capaz de converter as notas reconhecidas para uma partitura. Entre cada nota deve haver um espaçamento diretamente proporcional ao valor temporal da figura. Considerou-se a inclusão da selecção da clave mas tal concluiu-se dispensável por se tratar somente da aplicação de voz cantada, cuja aplicação é muito extensa e se situa relativamente centrada na clave de sol. As restantes claves são mais graves e tornaria mais confusa a apresentação de notas cantadas por vozes femininas, devido ao elevado número de linhas suplementares superiores necessárias. A decisão da armação de clave pode também constar dos requisitos e, em caso afirmativo, deve obviamente ter repercussões na partitura, podendo ser feita antes de ser apresentada a escrita ou posteriormente. Nesta última opção pressupõe-se uma clave predefinida como Dó Maior pela ausência de acidentes. Neste contexto, acidentes referem-se a sustenidos e bemóis, ou seja, alterações de meio-tom das notas contidas na tonalidade. A sua inclusão seria um complemento que facilitaria a leitura e seria também um fator diferenciador dos restantes programas, aprimorando a componente de edição musical. Um processo semelhante ao da armação de clave deve acontecer para o tipo de compasso que, segundo a introdução, poderá ficar-se por três das possibilidades mais usadas já indicadas. Este é avaliado como sendo de carácter obrigatório pois a sua não inclusão tornaria a utilização desta funcionalidade do SingingStudio demasiado limitada.

Para que seja possível transcrever a partitura a partir de um registo cantado no momento é crucial a existência de um metrónomo. A velocidade do metrónomo condicionará a partitura obtida, *i.e.*, um mesmo registo poderá dar origem a partituras diferentes para diferentes valores de tempo do metrónomo. O caso mais claro desta modificação é quando se utiliza o dobro do valor do metrónomo - o que significa que o tempo base passa a metade do tempo inicial, duas vezes mais rápido - e não se altera o tempo da melodia. As notas a transcrever neste caso irão sofrer uma duplicação do seu valor: a colcheia passa a semínima, a semínima a mínima e assim sucessivamente.

O ideal seria juntar a este programa uma funcionalidade de *scoring* que permitiria ao utilizador confirmar se o trecho musical que acabou de produzir corresponde de facto ao que deveria ter feito e, em caso negativo, onde errou. Idealmente poderia também ser criado um conceito de escrita musical em tempo real e permitir a gravação e processamento (incluindo transcrição) individual de faixas separadas, dando opção de as aglutinar formando uma só partitura. Esta funcionalidade permitiria a pequenos grupos instrumentais ou coros de câmara criar a sua própria música e escreve-la sem qualquer esforço, necessitando somente de um microfone para cada instrumento monofónico ou para cada voz.

Ponderou-se a existência de um factor de precisão regulável que indicaria a duração mínima aceite na escrita musical. Estaria disponível um *slider* semelhante ao apresentado no metrónomo mas de complexidade muito inferior e com alguns valores predefinidos, que permitiria tal regulação. Foi estudada também a utilização de um reconhecimento automático do tempo da gravação, uma técnica ensaiada já desde meados dos anos 90 pelos holandeses Peter Desain e Henjkan Ho-

ning [Desain e Honing \(1992\)](#) mas concluiu-se que a sua adaptação a este programa em concreto poderia ser uma mais-valia apenas numa fase posterior, uma vez que o resultado final pode igualmente ser obtido de forma mais simples com a solução inicial de definição manual do valor do metrónomo.

## 3.2 Desenvolvimento do software

Para iniciar o desenvolvimento do módulo adicional da partitura, é primeiro necessário conhecer bem o programa sobre o qual se trabalha. Instalou-se o *software* em causa e analisou-se alguma bibliografia relativa ao C++ e ao Qt. A utilização do código fonte implicou a compilação do Qt 4.7 e a alteração de todos os parâmetros associados como variáveis de ambiente e adição de *includes* e bibliotecas específicas. Sob pretexto da familiarização com a programação nesta linguagem e com o intuito de o fazer de forma útil, iniciou-se uma adaptação do SingingStudio a uma plataforma táctil. A estação de trabalho providenciada para o decorrer deste projecto contempla um monitor táctil de grandes dimensões que motivou esta alteração. Os botões da janela principal foram todos aumentados, deixando o ícone com a mesma dimensão e alterando somente a área sensível envolvente. Foi necessário alterar também o *layout* da janela principal nas medidas de alguns dos seus elementos. Estas alterações iniciais permitiram obter a experiência inicial necessária para dar início aos trabalhos esperados.

### 3.2.1 Análise aprofundada dos módulos existentes

Nesta fase foi analisado em detalhe o código do SingingStudio procurando a função da maioria dos objetos, apesar de este conhecimento ser sempre aprofundado ao longo de todo o trabalho de desenvolvimento, não só através de métodos investigação e experimentação mas também por contacto com os criadores do programa original.

Foi dado um ênfase especial a alguns objectos cujos ficheiros e funções são indicados de seguida:

**PianoGui.cpp** - cria o *widget* com o *display* das teclas do piano.

**NoteSee.cpp** - cria o *widget* com a pequena pauta cuja nota muda e que é tocada ao selecciona-la por *PianoGui.cpp*.

**semitom.cpp** - estabelece a correspondência entre a frequência de cada nota e o seu valor MIDI.

**midiInstrument.cpp** - define o instrumento MIDI com base na selecção num campo apropriado, apresentado na janela principal.

**Note.cpp** - indica o nome da nota identificada por nomenclatura latina e internacional e vice-versa.

**PitchMeterSS** - Calcula as frequências das notas tendo em conta a frequência de amostragem utilizada para o ficheiro em causa.

**MidiCalls.cpp** - exporta e importa ficheiros MIDI e transcreve o seu conteúdo para a variável *pitchNoteData\_external*. É também o responsável pela concatenação das frequências calculadas pelo ficheiro PitchMeterSS no vetor *pitchNoteData*.

**MidiFile.cpp** - Inicializa os ficheiros MIDI utilizados, adquire o seu tempo em BPM e o tempo de amostragem do ficheiro. Este ficheiro apresenta duas funções cruciais para a evolução do trabalho: *getBPM()* e *getTimeDivision()*. Há duas variáveis que estarão interligadas na definição do tempo: *tpms* e *time\_division*. *Tpms* significa *Ticks per Millisecond* e pode ser calculado usando:

$$tpms = bpm \frac{time\_division}{60000} \quad (3.1)$$

Posteriormente, explorar-se-á a melhor forma de utilizar estas variáveis para definir o tempo da forma mais simples possível.

Como foi já referido, as aplicações deste software são as mais variadas, mas se nos focarmos em *workshops* em organizações como a Casa da Música ou outras de cariz cultural inovador utilizam por norma a plataforma Mac, cujo sistema operativo é MacOSX, como base das suas aplicações. Como tal, a equipa que desenvolve o SingingStudio foi obrigada a refletir sobre a possibilidade de migrar o seu programa para um outro ambiente. O Qt tem a vantagem de ser multi-plataforma, significando que a compilação noutro sistema operativo nesse aspeto seria simples mas há certos componentes do programa como é o caso de bibliotecas especificamente construídas que tinham como alvo o Windows, utilizando funções deste, assim como toda a formatação a elas associada. Pesando o tempo dispendido nessa migração de bibliotecas, concluiu-se que a sua prioridade seria baixa e a sua execução deveria ter lugar mais tarde.

### 3.2.2 Estrutura de dados

Criou-se um ficheiro para gerar o objeto da partitura denominado *partitura.cpp* com uma classe homónima: *Partitura*. Foi neste ficheiro e o seu correspondente *header*, *partitura.h*, que foram realizados os avanços no desenho e processamento da partitura.

A representação das notas na partitura exige uma comunicação óbvia entre este objeto e os restantes que efetuam o reconhecimento das notas captadas e guardam os parâmetros identificativos das notas e a sua duração. Os dados recolhidos são guardados numa estrutura apropriada e precisam de ser oportunamente acedidos por esta classe. Quando um sinal sonoro é gravado, passa pelo algoritmo onde o número de harmónicos que possui permite, ou não, a identificação da sua frequência. Em caso de se tratar de uma nota de altura definida (por altura entenda-se o efeito psicoacústico consequente do *pitch*), esta é desenhada a laranja na janela principal. Posteriormente, é feita outra análise considerando as frequências na sua vizinhança para classificar a nota como tal, segue-se a atribuição de um valor MIDI representativo do *pitch* e define-se a sua duração. Esta análise é visível pela delimitação com um retângulo de cor azul como consta na figura 3.1. Na mesma figura, são visíveis pequenos retângulos sobre os quais se situa a área azul a considerar.



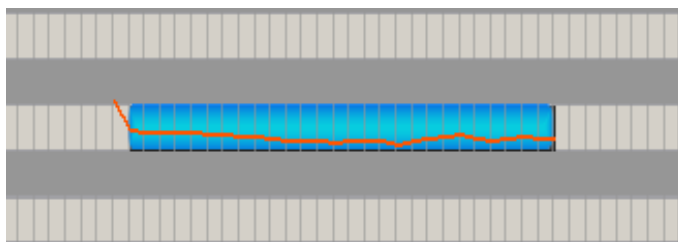


Figura 3.1: Pormenor da delimitação da nota no SingingStudio

Procurou-se então o objeto que trata a nota após o primeiro reconhecimento e que posteriormente faz a referida delimitação. A exploração intensiva do *software* permitiu concluir que a responsabilidade desta discriminação pertence ao objeto *PitchMeterSS*, uma adaptação do original *PitchMeter* criado para o *VoiceStudio* (um outro *software*), posteriormente adequado ao *software* que é agora motivo de trabalho, reunindo os dados no objeto *pitchsee*. O *PitchMeterSS*, como foi já mencionado na descrição dos módulos principais, mede a frequência das amostras captadas ou interpretadas do ficheiro importado e devolve o seu valor MIDI. Os dados recolhidos são alojados numa estrutura chamada *pitchNoteData* que é um vector do tipo *NoteLimits*, um tipo de vector definido no *SingingStudio* que contém três parâmetros para cada nota: *start* que indica o valor do tempo inicial da nota, *end* que aponta o instante de tempo final e *note* que indica o *pitch* nota em valor MIDI passado pelo *PitchMeterSS*. As notas neste vetor são ordenadas pelo índice *i* que efetua a seriação por ordem de chegada. A primeira abordagem para o reconhecimento das notas foi aproveitar o algoritmo deste objeto e copiá-lo para uma estrutura própria. Este assunto foi frequentemente discutido com o professor orientador e ponderou-se, a princípio, se valeria a pena essa duplicação da estrutura. A conclusão final foi que traria benefícios para a alteração da partitura que implica uma mudança dos valores da estrutura que, desta forma, poderia ser independente da estrutura principal.

A estrutura de dados está declarada no ficheiro *pitchsee.cpp* e é necessário passá-la desse objeto para o *Partitura* para poder ser lá utilizada. Para isso, apesar de ser mais direto aceder-lhes de imediato por intermédio da fonte - o *PitchMeterSS* - foi mais intuitivo aproveitar a existência do vector *PitchNoteData* e passá-lo para o objecto em questão. Inicialmente pensou-se em declarar a classe *Partitura* como classe amiga (do inglês *friend class*), de forma a que o objeto da primeira pudesse aceder diretamente ao conteúdo do vetor da outra classe. No entanto, optou-se por fim por utilizar uma particularidade do Qt: o *connect*. Consiste numa ligação entre duas classes onde a primeira emite um sinal e a segunda deverá ter uma função designada *slot* que será chamada com o mesmo argumento. Esta ligação terá de ser declarada no objeto que chama ambos os envolvidos na ligação. São declaradas variáveis, neste caso no *MainWindow*, que apontam para a classe emissora e para a recetora, e a comunicação é feita entre duas funções: um *signal* e um *slot*. O *signal*, proveniente da classe emissora, é uma simples função que deverá ser declarada como tal e que tem a sua tipologia predefinida para comunicar com o *slot* ou lançar um outro *signal*, passando sempre o mesmo número de argumentos entre a função emissora e a recetora.

Neste caso particular, a ligação entre as classes *Partitura* e *PitchSee* foi feita no objeto *MainWindow* mas os valores, por alguma razão, não chegavam à função destino. O *SingingStudio* possui algumas bibliotecas próprias criadas para o efeito que não estão preparadas para o modo *debug*, apenas para *release*. A inexistência de uma consola que retorne os valores de cada função faz com que a avaliação da localização do erro seja somente por exame do código. Após investigação exaustiva das bibliotecas nativas do Qt, foi encontrada uma denominada *QDebug* que, entre outras funcionalidades interessantíssimas, contém uma função que emite mensagens com o rótulo de *Debug*. Criou-se então - na função *main*, para que não houvesse erros de declaração - um *handler* de mensagens de erro que as escrevia num ficheiro do tipo "log", discriminadas pelo seu tipo. Esta discriminação é decidida pelo *QDebug* e o *handler* construído deve indicá-la como sendo *QDebug*, *QtWarning*, *QtCritical* ou *QtFatal*.

Com a utilização desta função foi possível verificar que as ligações *connect* declaradas no *MainWindow* não estavam a funcionar, pois as classes estavam a ser inicializadas nesse ficheiro apenas após serem chamadas pela ligação. Verificou-se também que o Visual Studio não reconhecia uma das funções de destino da ligação e que passava a reconhecê-la após ser redenominada. Este problema poderá derivar de erros de compilação ao fazer o *Build* do ficheiro. Concluiu-se que se os ficheiros *header* forem modificados deverá ser feito o *Rebuild All* para evitar situações semelhantes.

O primeiro elemento a ser criado para efetuar a transcrição musical será a pauta. O Qt providencia um comando próprio para desenhar certos objetos predefinidos como linhas, bastando para isso chamar o comando apropriado (no caso das linhas *painter.drawLine()*) utilizando como argumentos as coordenadas dos pontos que definem a localização do objeto. As 5 linhas foram criadas com um espaçamento que parecia adequado ao encaixe perfeitamente distinto de figuras musicais entre elas e que, após experimentação, se concluiu ideal. Um problema semelhante tinha sido apontado no *widget* da janela principal do *SingingStudio*: as notas apresentadas por vezes não se enquadravam na pauta da forma desejada.

A pauta deveria ser de largura pouco inferior à da janela. Começou por se configurar esta dimensão ajustável à própria janela mas, da apreciação feita no estado da arte, percebeu-se que esta deveria ser apresentada com um tamanho fixo. Concluiu-se ser preferível que a partitura fosse toda ela do mesmo tamanho, procurando-se um tamanho ideal para a compreensão do utilizador. A janela, por sua vez, terá um tamanho inicial predefinido que abarque a largura da partitura, dando já alguma margem.

Tendo a pauta completa é necessário definir as posições para todos os objetos a colocar. Entre estes estão a clave, a armação de clave e o tipo de compasso. O tipo de compasso será predefinido no metrónomo como quaternário - questão que será explicitada adiante -, a clave será sempre de sol por razões já explicadas nos requisitos e a armação será variável, sendo a sua escolha feita nas opções relativas ao módulo da transcrição musical. A geografia considerada, tirando a armação de clave que terá de incluir uma área variável, está descrita na figura 3.2 com as coordenadas indicadas em número de *pixels* por ser a medida utilizada no código para utilizar o Qt.

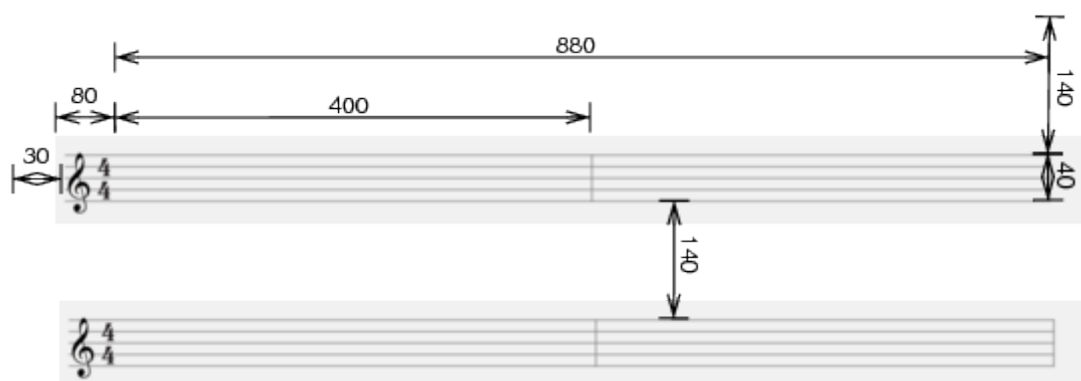


Figura 3.2: Medidas em pixels utilizadas na partitura

Aproveitou-se a clave disponível para a aplicação já existente no programa, apesar da sua qualidade de imagem deixar muito a desejar por ser utilizada num tamanho reduzido. Esta imagem foi posteriormente substituída por outra de qualidade significativamente superior, em formato SVG, aquando do desenho das figuras neste formato. Uma das dificuldades encontradas foi a aplicação desta imagem (ou de quaisquer outras) na janela da pauta já existente, apesar de qualquer outro objeto ser mostrado sem qualquer problema. O problema residia no caminho indicado para as imagens uma vez que a pauta começou a ser feita como *widget* em QtDesigner utilizando um sistema MacOSX (aproveitando a pluralidade de utilização deste *framework*) e só depois transportada para o sistema Windows para ser integrada no código original do SingingStudio. Ao efetuar esta transição, a indicação do caminho da imagem deveria ser feita considerando a forma diferente de armazenamento de dados nestes dois sistemas operativos. A utilização de caminhos relativos, que seria a solução mais óbvia, mostrou-se frequentemente problemática, mesmo na integração final da última versão do SingingStudio.

### 3.2.3 Metrónomo

Obtendo a duração das notas, é essencial saber a que base temporal é que essa duração se refere. É prática comum em música definir um valor base em bpm associado a uma determinada nota. No caso dos tipos de compasso considerados -  $\frac{2}{4}$ ,  $\frac{3}{4}$  e  $\frac{4}{4}$  - a base é sempre a semínima segundo a explicação dada no capítulo 2.2.2. É esse mesmo valor que será interpretado pelo metrónomo que indica, por sua vez, a base temporal para o cantor utilizar. Para recriar um metrónomo criou-se uma janela utilizando o QtDesigner, à qual se deu o nome de *metronomo.ui*, exemplificada na figura 3.3. Este ficheiro é chamado por *metronomo.cpp*.

O valor em bpm é indicado na janela ou pode ser escolhido alterando a posição da barra deslizante vertical que se encontra ligada à caixa. Este valor está predefinido para 120 bpm e o seu alcance é de 40 a 240, valores típicos nos metrónomos normais. É nesta janela que se escolhe o tipo de compasso que será posteriormente enviado por intermédio de uma ligação *connect* declarada no *MainWindow*, quer para a *Partitura*, quer para o objeto responsável pela reprodução dos sons

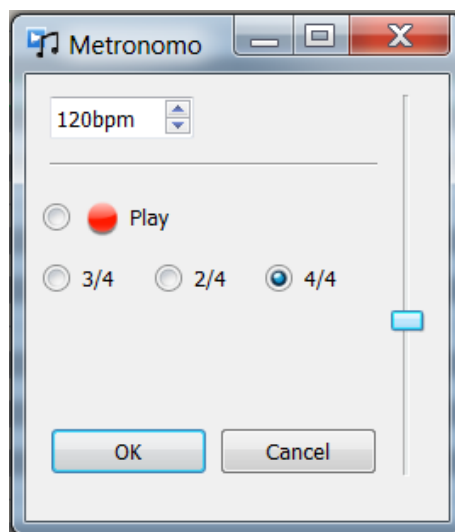


Figura 3.3: Janela do metrónomo

do metrónomo. A *Partitura* precisa de reconhecer o valor da base temporal para poder calcular o tempo correspondente a cada figura musical, como será adiante explicado. Ao seleccionar *play*, o objeto *Metronomo* vai chamar um outro objeto que cria um *thread* para poder reproduzir o som do metrónomo; este realiza um ciclo que reproduz o som e verifica se o botão *play* voltou a ser pressionado com a intenção de parar o metrónomo. As suas funções passam também por receber da interface gráfica o tipo de compasso seleccionado para descobrir a primeira batida para que a indique com um som diferente (mais agudo), identificando esse tempo como tempo forte. Este mecanismo está descrito na figura 3.4. Criaram-se duas funções no objeto *Partitura* que constituem *slots* para onde são passados os valores do tipo de compasso e tempo em BPM do metrónomo. O mesmo acontece no *PlayBeep* relativamente ao tempo BPM para poder colocar esse valor no temporizador inicializado. A criação de um *thread* permite repetir estas operações indefinidamente sem bloquear a aplicação e permitindo que o *thread* principal continue a realizar a função de gravação das notas e respectiva identificação, entre as suas restantes obrigações, contribuindo para manter os padrões de responsividade que se deseja para o programa.

O código relativo às classes *Metronomo* e *PlayBeep* encontra-se no anexo A para facilitar a compreensão do algoritmo.

### 3.2.4 Escolha da figura musical

A captação do som é feita pela reunião de amostras em pequenas janelas - *frames* - que se sobrepõem de forma a otimizar os resultados obtidos. As *frames* são conjuntos de 1024 amostras com sobreposição de 50%. Cria-se com isto um *buffer* de 512 amostras, metade do tamanho das *frames*. A figura 3.5 é uma representação esquemática deste processo com o intuito de facilitar o seu entendimento. No funcionamento inicial do *SingingStudio*, a frequência de amostragem está predefinida a 44100Hz apesar de poder adoptar outros valores. Todos os parâmetros anteriormente

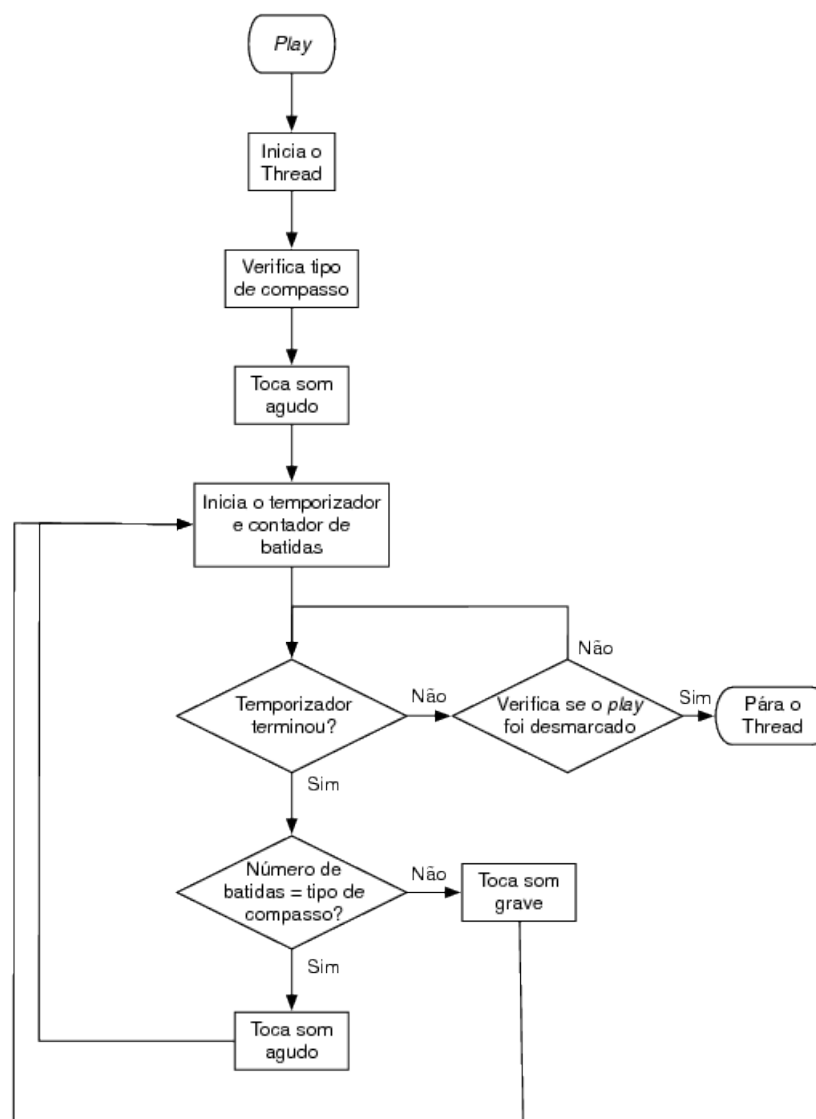


Figura 3.4: Diagrama de Funcionamento do Metrónomo

referidos são definidos nas opções do programa, como se pode ver na figura 3.6. A variação dos valores da frequência de amostragem provou ser um obstáculo ao bom funcionamento da transcrição musical.

A frequência predileta para a utilização óptima do algoritmo de reconhecimento de notas é 22050 Hz, visto que demonstra melhores resultados na análise harmónica. No entanto, a utilização de uma amostragem mais reduzida tira precisão à identificação das figuras, pois o tempo correspondente à menor figura ronda um valor verdadeiramente baixo de amostras, como mais tarde será explicado. Os valores previamente indicados são fulcrais para o cálculo do valor temporal de cada figura musical. As figuras musicais medem-se normalmente pelo número de batimentos da sua duração (também chamados tempos), como se indicou no capítulo 2.2.2. O processo adoptado para esta discriminação foi a conversão de cada nota registada para o seu tamanho relativo, isto

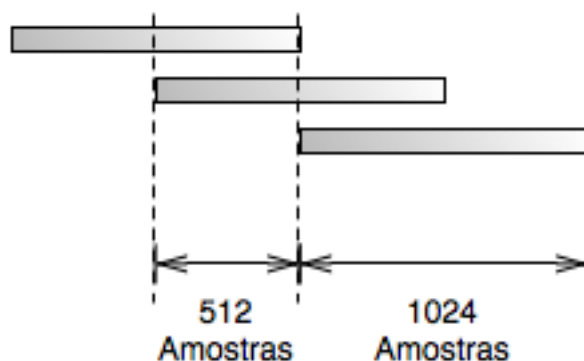


Figura 3.5: Representação esquemática da amostragem por frames

é, em tempos. A unidade base na duração das notas registadas na janela principal é o número de *frames*, sendo esta a unidade obtida ao subtrair o valor do tempo final da nota - *iend* - pelo inicial - *istart*. Cada *frame* tem uma certa duração que se traduz em milissegundos através da sua relação com a frequência de amostragem através de:

$$\frac{1024 \text{ amostras}}{44100 \text{ Hz}} 50\% = 11,6 \text{ ms.} \quad (3.2)$$

Introduz-se ainda outro conceito: o valor temporal em milissegundos por cada batimento. Este é indicado por *mspt* e pode calcular-se de forma trivial sabendo que um determinado número de batimentos em *bpm* demora, por definição, 1 minuto, ou seja 60 000 milissegundos.

$$mspt = \frac{60000(ms)}{bpm} \quad (3.3)$$

Juntando as equações 3.7 e 3.3 é possível calcular a duração de uma nota registada pelo seu tempo relativo, provando-se que esse é definido pela equação 3.4.

$$tamanho_{relativo} = \frac{(iend - istart) * 11,6}{mspt} \quad (3.4)$$

Obtido o tamanho relativo, compara-se este valor aos das figuras musicais, seleccionando a mais apropriada. A margem de erro da selecção das figuras foi considerada como sendo limitada pelo valor médio entre as durações de duas figuras consecutivas. A figura 3.7 esquematiza esta divisão, indicando o intervalo ao qual corresponde a selecção de cada figura que será depois impressa na partitura. O algoritmo de decisão da figura musical a escolher foi criado com o objetivo de tornar o SingingStudio numa aplicação abrangente em termos de utilizadores. Haveria outras hipóteses de maior precisão que incluíriam mais figuras pontuadas ou outras que se poderiam reger por múltiplos da semifusa, hipóteses que o tornariam também mais exposto a erros.

Na parte inferior do diagrama temos os valores temporais de cada figura em tempo relativo,

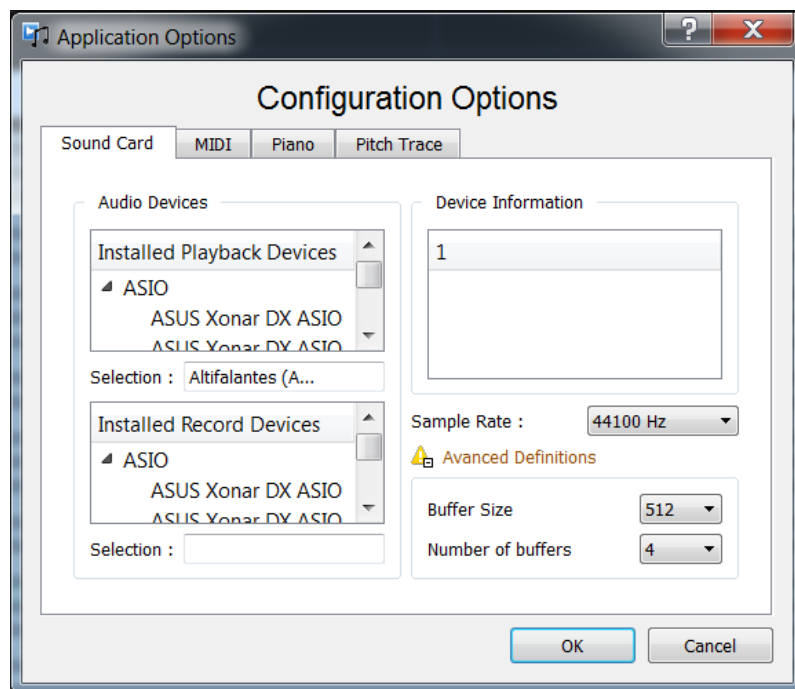


Figura 3.6: Janela com as opções de audio do SingingStudio

isto é, valores em batimentos. Para cada valor, está indicada uma janela em cujo intervalo a duração da nota analisada se inserirá. Este processo de associação é feito com valores relativos para que não seja preciso calcular os intervalos consoante a mudança do valor base BPM. Desta forma é apenas feito um cálculo por nota, permitindo a optimização da utilização dos recursos pela aplicação e contribuindo para o aumento da velocidade da transcrição. Identificada a nota, é necessário desenhar a figura respetiva nas coordenadas corretas. Este processo é feito utilizando o *include* `QSvgRenderer` com a função *Renderer*, que recebe as coordenadas, e precisa de ter já carregado as referidas imagens no construtor do objeto *Partitura*. Ao carregar todas as figuras no construtor obtém-se maior rapidez, já que este processo é realizado uma única vez, o que não

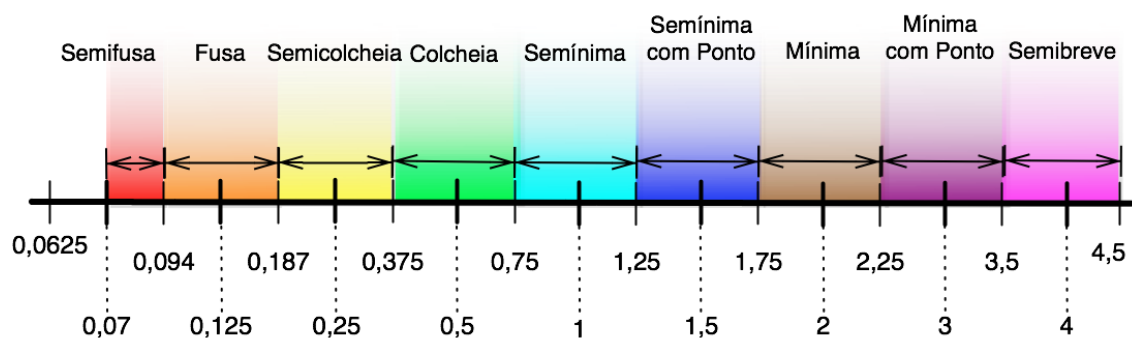


Figura 3.7: Esquema da divisão temporal das figuras

aconteceria caso o carregamento fosse efetuado numa qualquer função.

Explicado o processo de cálculo de cada figura musical, é possível agora justificar a questão da frequência de amostragem. A duração de uma semifusa será calculada pela equação 3.5, onde BPM é o valor de batimentos por minuto indicado no metrónomo.

$$\text{tempo da semifusa} = \frac{60000}{16 \times \text{BPM}} \quad (3.5)$$

Basta então dividir o resultado pelo tempo de cada buffer de amostras para se obter o número de amostras necessário para reconhecer uma nota, para diferentes valores de BPM. Um buffer de amostras a 44100 Hz tem a duração de 11,6 ms mas se a frequência de amostragem for alterada para 22050 Hz, a duração de cada amostra passa, por cálculos análogos, a 23,2 ms. Podemos observar na tabela 3.1 que para o menor valor temporal do metrónomo, a utilização da menor frequência de amostragem para reconhecer uma semifusa é inferior à duração de um buffer de amostras, o que não é um intervalo que permita confiança no resultado; daí a necessidade da utilização da maior frequência de amostragem como predefinido.

Frequência de Amostragem	40 BPM	240 BPM
44100 Hz	8,08 ms	1,35 ms
22050 Hz	4,04 ms	0,67 ms

Tabela 3.1: Comparação do tempo da semifusa para diferentes frequências de amostragem

Agora que se conhecem os processos que fornecem dados essenciais à criação da partitura, é possível resumir toda a algoritmia responsável pela sua criação. Ao ser chamada a partitura, o construtor cria a janela com os respetivos botões. Gera-se um *paintEvent* que iniciará a função de escrita na janela, cuja primeira ação será pedir a estrutura *pitchNoteData* e copiá-la para outra semelhante: *dadosNotas*. Após a replicação da estrutura, define-se o valor temporal em milissegundos de cada batimento, o já mencionado *mspt*. Seguidamente, caso a estrutura não se encontre vazia, utilizando os seus parâmetros, calcula-se o tamanho da nota em questão e da pausa entre esta nota e a anterior (exceptuando o caso de se tratar da primeira nota) da forma indicada na equação 3.3 chamando-lhes *tamanho* e *tamanhopausa*. Se for o caso da estrutura não possuir notas, é mostrado um aviso que indica essa falta, provavelmente por o programa não ter ainda um ficheiro gravado. São inicializadas algumas variáveis de notável importância para o cálculo das coordenadas das notas. Duas delas (*posh* e *npos*) servem para regular a posição horizontal das notas. Cada compasso é dividido num determinado número de posições, dependente do tipo de compasso. O cálculo é feito considerando que cada tempo pode ser sub-dividido em 8 posições, obtendo-se para cada compasso  $8 * msrp$ , em que *msrp* é 4 em caso de compasso quaternário, 3 em caso de ternário e 2 se se tratar de um compasso binário. Cada figura musical tem associado um determinado número de posições proporcional ao seu tamanho que é acrescentado ao valor total do compasso definido por *npos*. O valor desta variável nunca poderá exceder o valor máximo



do compasso, condição que é testada depois do cálculo do tamanho e, se isto não acontecer, é-lhe adicionado um valor associado com a figura que foi escolhida. É feito um processo análogo com a pausa, utilizando *tamanhopausa*. No caso do número de posições a adicionar associado à figura escolhida perfizer um total que não se encaixa no tempo máximo do compasso, a variável tamanho passa a ter somente o tamanho que cabe no compasso e tudo o restante irá para uma outra variável que passará pelo mesmo ciclo. Isto significa que se uma nota ultrapassar o tamanho permitido para aquele compasso, é desenhada apenas a figura correspondente aos tempos a montante no compasso em questão, enquanto o tempo sobranete será representado no compasso seguinte, exigindo uma atualização do compasso e da pauta na qual se desenhará esta segunda parte da nota. Estas duas notas serão unidas por uma ligadura que indicará ao intérprete a continuação da primeira nota pela segunda.

O número do compasso onde se escreve é indicado por uma terceira variável (*ncomp*) que incrementará quando as posições disponíveis forem ultrapassadas. Ao identificar a nota, para além da atualização da posição horizontal, é feita a apresentação da figura correspondente na localização correta, criada uma *flag* que implica verificar posteriormente se a nota necessita de linhas suplementares inferiores ou superiores e desenhá-las e repetir esse processo para sustenidos.

O sistema da divisão em  $8 * mspt$  partes iguais foi um sistema criado para uma resolução que parece aceitável na região de 400 pixels - como é indicado na figura 3.2 - por compasso, visto ter o seu valor máximo em compasso quaternário com 32 posições. No entanto, esta não é a coordenada final mas apenas mais um componente. Existe a outra variável (*posh*) que, essa sim, regula a posição horizontal na totalidade, considerando as notas já nela impressas. O seu cálculo considera o espaço entre o início da janela e o início da pauta, a ocupação da clave e do tipo de compasso, a posição anteriormente descrita que toma em conta o espaço ocupado por notas já impressas e o número do compasso em questão:

$$posh = 80 + ncomp * 400 + npos * 12.5 \quad (3.6)$$

Em que *ncomp* indica o número do compasso e 12,5 é calculado dividindo os 400 pixels disponíveis no compasso pelas 32 posições. O número de compasso (*ncomp*) serve apenas para indicar se se trata do primeiro (adquirindo o valor 0) ou do segundo compasso (com o valor 1).

Falta apenas considerar a posição vertical da imagem. Esta será definida pela frequência da nota, neste caso pelo valor MIDI da nota indicado no campo inote do vetor *dadosNotas*. A cada nota é associada uma coordenada vertical que depois é necessário atualizar consoante a pauta em que nos encontramos. Considerou-se que cada pauta terá 2 compassos o que implica criar uma variável de contagem de pauta (*nsistema*) que aumentará de forma constante a coordenada vertical. Esta é calculada pelo teto da metade do número total de compassos, ou seja, para além de saber se devemos escrever no primeiro compasso ou no segundo adicionou-se uma contagem total do seu número. A coordenada vertical é então calculada por

$$notap = notap * (nsistema - 1) 180. \quad (3.7)$$

O algoritmo acima descrito foi brevemente esquematizado na figura 3.8, começando no momento após o desenho da pauta, clave e tipo de compasso. Este diagrama apresenta a cor azul os passos em que são atualizadas as variáveis *ncomp*, *posh*, *npos* e *nsistema*; a amarelo estão todos os momentos que incluem impressão na partitura e a laranja a atribuição do valor de *notap*.

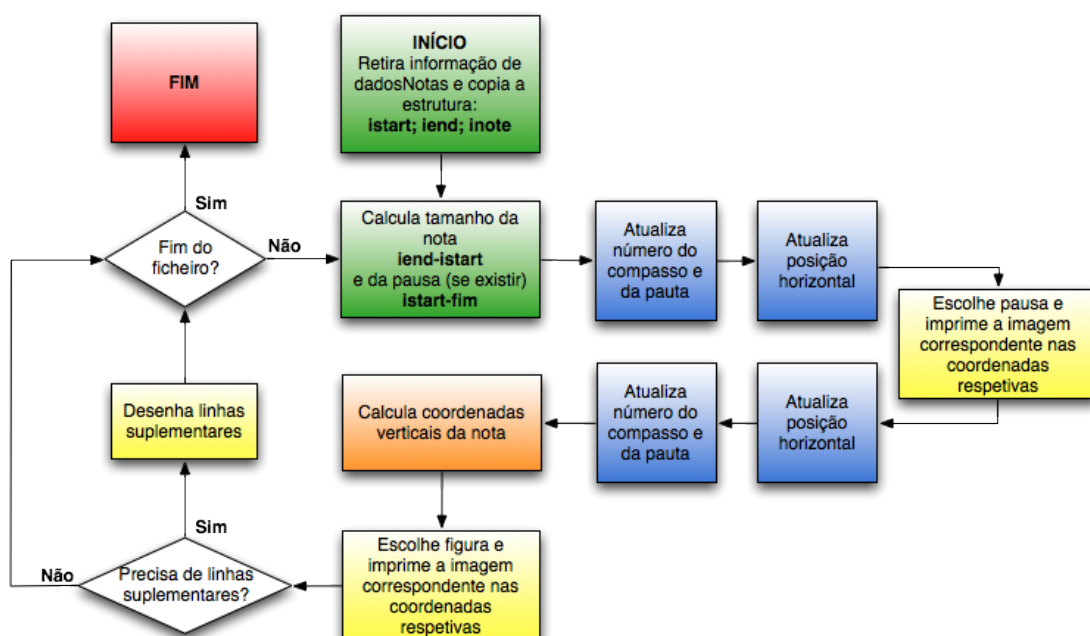


Figura 3.8: Algoritmo da impressão das notas na partitura

### 3.2.5 Imagens

Para utilizar os dados obtidos do reconhecimento das notas representam-se as figuras musicais na janela. Não foi fácil a escolha do tipo de ficheiro a utilizar para a criação das figuras. Considerou-se inclusivamente desenhá-las na janela como aconteceu com a pauta mas a relação entre a qualidade das formas obtidas e o trabalho exigido não convenceu. Avançou-se, nesta fase, com a criação de imagens em formato PNG por ser um tipo de imagem leve, compatível com o Qt e que não cria dificuldades na criação de imagens cujo fundo seja transparente, como se deseja. As imagens criadas foram as que se representam em seguida, na figura 3.9, já na pauta para se perceber a forma como se distribuem e juntamente com uma clave de sol igual à aplicada no *widget* que indica a nota na janela principal do SingingStudio.

Como se pode concluir, a qualidade da imagem não é a desejada. Serviram estas imagens para testar as funcionalidades que iam sendo implementadas até se criar algo de qualidade superior, mais aproximado dos principais programas de edição musical. Estes utilizam por norma imagens do tipo SVG. Este tipo de ficheiro baseia-se numa linguagem em formato XML que constrói o gráfico de duas dimensões através de funções matemáticas. Esta técnica permite obter imagens de

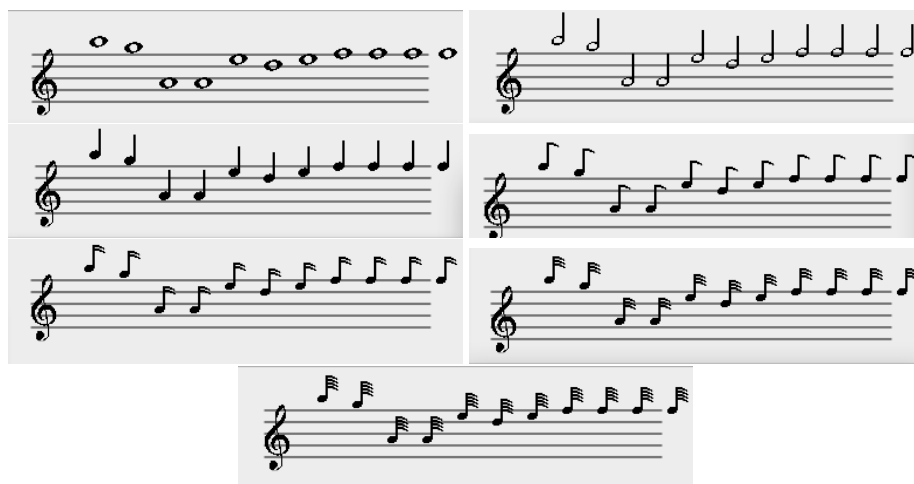


Figura 3.9: Representação PNG das figuras

grande qualidade seja em pequena ou grande escala, tornando-as perfeitas para sistemas escaláveis. Apesar de não ser o caso, a qualidade gráfica permitida pelo SVG tornou-o o formato eleito para esta aplicação. Para criar imagens vectoriais utilizou-se o Adobe Illustrator e criaram-se não só as figuras como na versão anterior mas também as pausas, uma clave de sol que se mostrou de qualidade significativamente superior à sua precedente, os tipos de compasso e os símbolos referentes aos acidentes. Para utilizar imagens SVG, o Qt possui um módulo próprio - QtSVG - que carrega o ficheiro que descreve a imagem e posteriormente o representa na janela com o comando *render*.

As imagens SVG provaram cumprir os objetivos de estética definidos para o programa e o seu carregamento não foi suficientemente lento para que se justificasse a criação de outro *thread* para a sua impressão. No entanto, ao serem impressas apoiam-se num pequeno ponto de ancoragem que é mostrado a amarelo. Esse pixel extra foi bastante útil para confirmar a localização das figuras mas poderá ter de ser removido posteriormente.

### 3.2.6 Outras características e principais dificuldades

É possível imprimir a imagem da partitura através de um botão próprio colocado na barra de ferramentas no topo da janela. Este direcionará para o sistema de impressão do computador. Outro assunto, as opções da armação de clave já foram implementadas junto das restantes opções do SingingStudio. A informação relativa à clave seleccionada é então passada à partitura que imprime o número de sustenidos ou bemóis necessário. A razão pela qual esta implementação não foi concluída é porque, para além do desenho dos sustenidos ou bemóis, é necessário voltar a desenhar as notas na respetiva tonalidade, ou seja, voltar a calcular se as notas devem ou não ter acidente, e a falta de tempo não permitiu concluir tal operação. A exportação em formato MusicXML foi também analisada mas a sua implementação também foi travada pelo tempo.
























PNG	SVG	SVG
		
		
		
		
		
		
		
		

Figura 3.10: Comparação entre as imagens PNG e SVG

A principal dificuldade sentida durante o desenvolvimento da componente principal do trabalho da dissertação foi a compreensão do código já elaborado juntamente com os pequenos erros que ocorriam pelas mais variadas razões nos diferentes programas utilizados. Foram implementadas algumas soluções como a pauta escalável proporcionalmente ao tamanho da janela ou as imagens em formato PNG que demoraram algum tempo e se provaram infrutíferas.

### 3.3 Conclusão

Desenvolveu-se o *software* de transcrição da pauta musical, principalmente com a criação de 3 ficheiros. Um destes serve para exibir o metrónomo, outro para tocar o som deste e um último para imprimir na partitura as notas correspondentes. Os dados recebidos pelo objeto criado pelo ficheiro de criação da partitura são provenientes de um outro objeto que interpreta o som gravado. Para a apresentação da partitura foram criadas imagens adequadas assim como um algoritmo de discriminação das diferentes notas, tentando dimensionar a solução com uma precisão adequada à maioria das situações.

## Capítulo 4

# Resultados

Nesta secção são apresentados testes efetuados ao programa desenvolvido e são discutidos os resultados. Tentou-se, nestes testes, abarcar as principais qualidades e as falhas do programa, de forma a realizar uma análise imparcial.

### 4.1 Testes e análise de resultados

O facto do SingingStudio estar configurado especificamente para a voz humana traz sérias dificuldades à transcrição. O som que escutamos é formatado pelo trato vocal que salientará as frequências formantes e é essa moldagem que caracterizará a voz de um determinado cantor.

A caracterização própria da voz dá à sua frequência fundamental uma forma muito volátil, sendo especialmente notada nas transições entre notas. Mesmo um cantor profissional canta para agradar ao ouvido humano, indo de encontro a todos os efeitos psicoacústicos que sentimos e não das características físicas do som. As respirações, os *vibratos* e o *staccato* são 3 exemplos de acontecimentos no canto que consideramos bastante enquadrados e agradáveis mas que um programa de computador não consegue distinguir de um erro. Torna-se ainda mais complicado de criar uma regra que identifique esta situação quando não há uma ocasião específica onde ela é utilizada; são somente recursos artísticos ou necessidades do artista, no caso da respiração, mas feitas de forma natural. O algoritmo de reconhecimento de notas do SingingStudio possui também alguns problemas que não permitem a perfeita interpretação da linha melódica entoada, problemas que estão paralelamente a ser resolvidos por uma equipa de trabalho na qual foi, numa época final do projeto, integrada a presente funcionalidade da transcrição musical. Nem todos os problemas, porém, são solucionáveis em tão pouco tempo como o disponível para o desenvolvimento desta dissertação, uma vez que algum temperamento da parte do cantor ou uma transição entre notas que o ouvido ignora são rapidamente desmascarados pelo SingingStudio. Como não é possível educar um computador para a arte da música nem fazer com que a voz do cantor se torne descaracterizada de modo a estabilizar a sua frequência fundamental, a solução para testar o programa com alguma

fiabilidade extra foi a utilização de um instrumento de sopro: a melódica. Este instrumento tem teclas semelhantes às de um piano mas que apenas produzem som se, quando pressionadas, for insuflado ar pela entrada apropriada. O som produzido tem uma frequência fundamental de estabilidade significativamente superior à tipicamente conseguida através de voz cantada. Os testes realizados contêm melodias simples cuja extensão é variada, assim como a duração das notas.

O primeiro teste demonstrado teve como objetivo reproduzir a partitura representada na figura 4.1.



Figura 4.1: Partitura inicial do primeiro teste

Escolheu-se uma melodia simples que foi interpretada por uma jovem do sexo feminino, para quem o registo escolhido é confortável. Isto faz com que cante com o mínimo de esforço e dá um perfil mais suave e estável ao traçado da frequência captada.

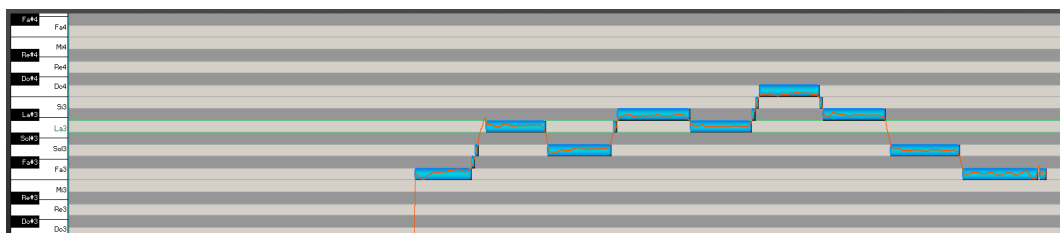


Figura 4.2: Teste 1 - resultado obtido pelo SingingStudio



Figura 4.3: Teste 1 - transcrição obtida

Este teste revelou-se extremamente positivo, transcrevendo na perfeição a melodia entoada. Neste caso não foi preciso comparar com a melodia tocada na melódica porque não traria qualquer

resultado que trouxesse valor à avaliação do desempenho.

Normalmente, os testes realizados no SingingStudio apresentaram resultados muito perto da perfeição, e o erro mais comum de constatar é o da divisão de uma nota musical em duas cuja soma perfaz um valor igual. Este fenómeno acontece sempre que o SingingStudio interpreta a nota em questão como duas por falta de intensidade do som, por exemplo. O *software* foi testado ao longo do desenvolvimento inúmeras vezes e não se vê vantagem em expor constantemente exemplos positivos. Em vez de se analisarem casos comuns de resultados positivos, analise-se antes resultados que não correspondem às expectativas, analisando os erros.

O segundo teste efetuado consistia na reprodução de uma curta melodia. A partitura a reproduzir foi a representada na figura 4.4. Foi escolhida para começar uma melodia de curta duração e com uma extensão limitada, apenas para demonstrar a base do funcionamento da partitura. As figuras escolhidas são variadas para podermos observar como reage o programa a diferentes durações das notas.



Figura 4.4: Teste 2 - partitura inicial a reproduzir

A interpretação cantada da voz está representada na figura 4.5 e são visíveis alguns erros, razão pela qual se escolheu este teste. No início observa-se algum ruído que se refletirá na partitura transcrita - figura 4.6 - criando várias pausas no primeiro compasso. O algoritmo de reconhecimento do SingingStudio reconhece várias notas mas, devido à sua reduzida duração, considera-as demasiado pequenas. No entanto, estes valores interrompem a contagem do tamanho da pausa, criando assim várias pausas de valor inferior mas cujo valor total será igual. A erro na primeira nota transcrita deve-se à afinação “baixa” que, durante um curto intervalo, foi interpretada como sendo a nota imediatamente inferior. No entanto, esta nota e a que lhe deu origem são de tão curta duração que a partitura não chega a considerar a sua discriminação em figura musical. Estes erros em pequenas notas fazem com que haja durações temporais que não são suficientemente grandes para serem considerados nota ou pausa e, consequentemente, não contribuem para o aumento da variável de contagem dos tempos no compasso.

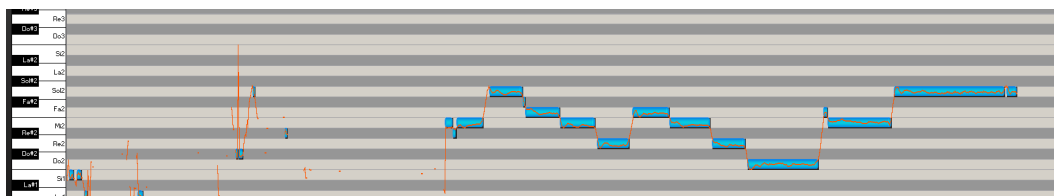


Figura 4.5: Teste 2 - resultado da voz interpretada pelo SingingStudio



Figura 4.6: Teste 2 - partitura transcrita da voz cantada

Os erros detetados na interpretação da voz cantada não foram suficientemente graves para tornar a partitura incompreensível, mas experimente-se agora utilizando a melódica. Com este instrumento, o SingingStudio revela uma muito maior capacidade de reconhecimento preciso de notas. O resultado obtido é demonstrado nas figuras 4.7 e 4.8, sendo a primeira o feedback visual da representação das notas na janela principal do SingingStudio e a segunda a transcrição obtida dessa identificação.

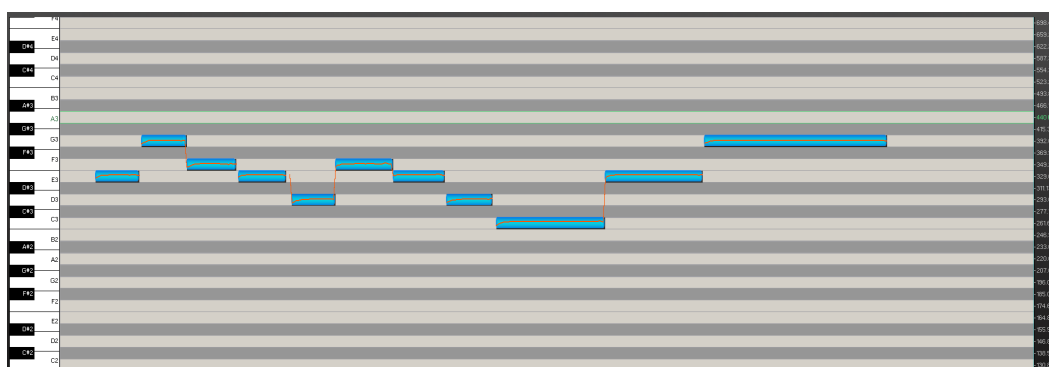


Figura 4.7: Teste 2 - resultado da melódica interpretado pelo SingingStudio



Figura 4.8: Teste 2 - transcrição obtida da melódica

A partitura obtida é muito mais clara e totalmente igual à inicialmente interpretada, o que reforça a confiança no mecanismo de transcrição. Neste caso já não há qualquer erro como no caso anterior, aliás, a ocorrência de erros com a utilização da melódica raramente aconteceu e, quando tal sucedeu, limitava-se a erros de reconhecimento de nota, interpretando o resultado uma oitava abaixo do devido apenas num curto intervalo.



Como terceiro teste, utilizou-se uma melodia mais longa, o que aumenta a probabilidade de acontecer um erro. Este aumento deve-se não só ao maior número de notas e com frequências mais abrangentes como também pelos processos pelo qual a interpretação pode passar, como se pode constatar no código, no anexo [A.3](#). Estes incluem, por exemplo, a passagem de uma nota demasiado grande para um determinado compasso para o seguinte, possivelmente numa pauta abaixo. A partitura inicial encontra-se representada na figura [4.9](#).



Figura 4.9: Teste 3 - partitura inicial da melodia

A melodia cantada vigora na janela principal como representado na figura [4.10](#). Foi cantada por um elemento do sexo masculino, razão pela qual se encontra uma oitava abaixo do original. Apresenta uma identificação irregular, com alguns erros causados pelo facto da frequência fundamental se encontrar no limite do intervalo de reconhecimento na nota resultando de uma identificação alternada entre a nota devida e a imediatamente inferior. Estas pequenas desafinações no canto são frequentes quando não se trata de um cantor profissional, como foi o caso deste teste. Nas transições e no ataque foram também identificadas erradamente algumas notas que não deveriam existir. Erros como os enumerados, em casos nos quais as notas erradas são suficientemente grandes para permitir a identificação ou nos que, ainda não tendo tamanho suficiente, reiniciam a

contagem da duração da nota, encurtando-a, implicam a impressão de um maior número de notas na partitura, embora com um valor inferior. Estas notas assumem posições muito próximas, dificultando a leitura do resultado, especialmente se, a estas notas, forem adicionados acidentes.

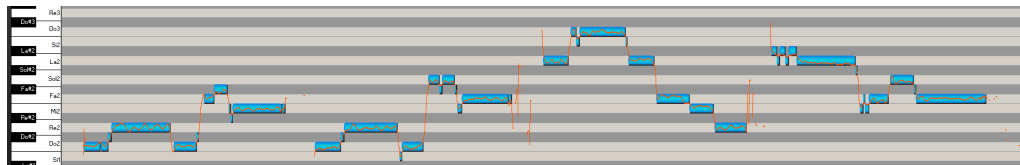


Figura 4.10: Teste 3 - sinal cantado interpretado pelo SingingStudio



Figura 4.11: Teste 3 - Partitura transcrita da voz cantada

Para confirmar a razão do erro, foi uma vez mais interpretada a melodia utilizando a melódica, cujo resultado se encontra na figura 4.12 e que gera a partitura na figura 4.13. Observa-se que a partitura obtida pela transcrição automática é idêntica à inicialmente interpretada. Isto prova que a identificação das figuras musicais efetuada pela partitura é bastante fiável no caso do traçado da frequência da melodia entoada ser estável, lidando bem com as pequenas variações naturais na duração das notas da melodia. Consegue-se, então, associar a linearidade das notas reconhecidas pelo SingingStudio com a clareza da partitura transcrita.

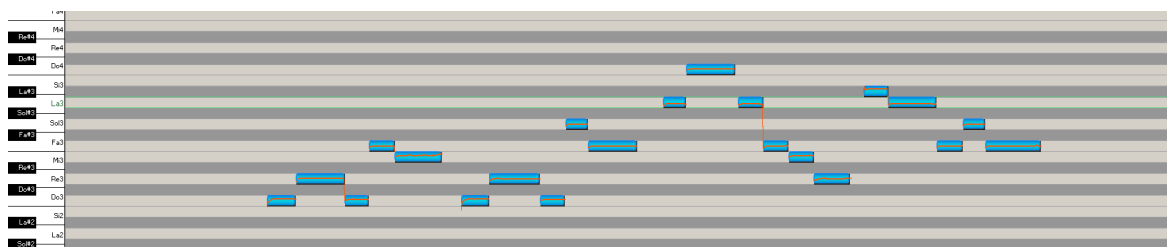


Figura 4.12: Teste 3 - sinal tocado na melódica



Figura 4.13: Teste 3 - Partitura interpretada da melódica pelo SingingStudio

## 4.2 Resumo e conclusões

A transcrição da pauta musical com base no reconhecimento das notas previamente apresentadas pelo SingingStudio funciona com relativa fiabilidade. Nos testes realizados, a resposta do módulo desenvolvido foi bastante positiva, normalizada pelos valores do metrônomo e congruente com os valores esperados. O primeiro teste demonstrado mostrou um exemplo claro de uma situação normal da utilização desta funcionalidade. O metrônomo também funciona segundo os requisitos previamente detalhados.

As situações de erro foram analisadas, dando-se mais ênfase aos erros mais típicos, como foi o caso dos testes 2 e 3. Concluiu-se que estes ocorrem quando o reconhecimento apresenta variações anormais da frequência fundamental, diminuindo a fiabilidade da partitura. As dificuldades mais frequentemente sentidas no processo da transcrição são resultantes de situações nas quais o canto é instável e apresenta ataques pouco sustentáveis ou em que a frequência se encontra perto do limiar de decisão da nota. Em termos estéticos, estas situações diminuem substancialmente a qualidade do resultado e tornam-no muito pouco prático para os objetivos previstos.

A alteração do sistema de reconhecimento da partitura será demasiado complexa para responder às situações indicadas, visto tratarem-se de situações específicas de efeitos artísticos no canto ou pequenas desafinações. A solução passa por melhorias no algoritmo de reconhecimento da nota, tendo em vista o aumento do tamanho geral das notas, ou seja, a concatenação de notas demasiado pequenas quando estas deveriam indicar uma só. Este tipo de ajustes visam situações que obrigam à definição da ténue separação entre a precisão física e a arte: é necessário especificar as situações nas quais o intérprete desejava cantar apenas uma nota e quais aquelas em que a oscilação é premeditada. O grau de inteligência necessário seria demasiado para o que se pretende com o *software* em questão, optando-se por uma segunda solução: a alteração da partitura manualmente, após a transcrição. A implementação deste ponto está já preparada no SingingStudio, pertencendo ao trabalho a desenvolver num futuro próximo.

## Capítulo 5

# Conclusões finais e Trabalho futuro

Este capítulo final serve para tirar conclusões do trabalho efetuado e inferir sobre a convergência dos resultados com os objetivos inicialmente delineados. Nesta fase, olha-se em retrospectiva todo o percurso percorrido no desenvolvimento deste projeto e são analisados os passos dados para se poderem criticar os erros cometidos e frisar os pontos mais positivos.

### 5.1 Satisfação dos Objectivos

Para a transcrição musical ser feita de forma automática é necessário compreender a música como forma de arte e tentar decompor determinados aspetos da sua génese nas várias situações em que normalmente acontecem, de modo a conseguir cobrir a maioria das situações nas quais estes acontecem. Segundo os testes efetuados, pode-se concluir que a transcrição musical da voz cantada padece ainda de alguns erros por parte do SingingStudio, impedindo o correto reconhecimento das notas e, conseqüentemente, a sua identificação e transcrição fiável. Tal problema será ultrapassado com a inclusão da possibilidade de fazer pequenas alterações na partitura, opção para a qual o módulo desenvolvido está já preparado. Quando se utilizou a melódica como instrumento responsável pela reprodução da linha melódica a identificar, constatou-se que o número de erros e a sua dimensão são significativamente menores. Esta técnica possibilitou a correcção de alguns pormenores da partitura como foi o caso da implementação da funcionalidade de adição de sustenidos, do ajuste do tamanho adequado a cada figura musical ou na exploração do da duração mínima da nota reconhecida; enfim, em todas as experiências que se regeram pelo conhecimento empírico.

A transcrição musical é efetuada na maior parte dos casos com um elevado nível de confiança, quer quanto às figuras indicadas e a sua altura, quer relativamente à distribuição temporal das notas nas figuras musicais. Esta seleção de figuras acarreta algum erro permitido para escolha de uma determinada figura e o somatório de todos estes erros pode aumentar moderadamente o tempo total de um determinado compasso. Para resolver esta questão deve ser implementada,

por exemplo, uma barra de precisão que ignoraria as figuras de menor duração temporal numa razão proporcional ao valor indicado nessa mesma barra. A utilização do *software* é bastante simples, como desejado, apresentando uma plataforma de utilização intuitiva de modo a facilitar a experiência do utilizador. A funcionalidade do metrónomo foi igualmente bem sucedida, correspondendo ao funcionamento normal do aparelho e integrando-se facilmente com a funcionalidade de transcrição.

O *software* deverá ser integrado com os restantes desenvolvimentos efetuados pela equipa incluída no projeto ARTTS, cujo resultado será então avaliado por potenciais utilizadores e só então se obterá um *feedback* fundamentado. No entanto, durante o período de testes do desenvolvimento do presente módulo, a aplicação foi experimentada por diversas pessoas ligadas ao ramo da música - nomeadamente alunos e professores de canto - que salientaram a utilidade da funcionalidade de transcrição automática e se mostraram geralmente satisfeitos com os resultados.

## 5.2 Trabalho futuro

Há ainda alguns passos a dar no futuro desenvolvimento do SingingStudio se se apontar o seu desenvolvimento no sentido de auxiliar do ensino do canto. Todos vão de encontro à perfeição da transcrição, embora não se exclua a possibilidade de incluir certos aspetos mais didáticos se os utilizadores finais o considerarem importante, como a classificação de intervalos entre notas ou a escolha de notas ser feita na partitura, ou seja, ao começar a cantar uma nota, o intérprete veria uma semicolcheia e, à medida que a nota fosse ficando mais longa, a figura mudaria em tempo real na pauta. Este tipo de aplicações não trazem nada de novo para o output final mas se o público alvo do *software* o considerar importante poderá facilmente ser implementado.

A armação de clave inclui já uma opção onde pode ser escolhida. Para concluir a sua aplicação, basta associá-la à implementação de sustenidos ou bemóis nas notas, na zona da *flag*, e criar um vetor onde se incluam todas as notas do presente compasso para verificar a necessidade de bequardos que anulem os acidentes prévios naquele compasso, se for caso disso.

A aparência da partitura foi um fator muito valorizado no decorrer dos trabalhos e será sempre um ponto que merece uma atenção especial porque, por mais eficiente que seja, o programa desenvolvido nunca cativará o utilizador se não for esteticamente agradável. As notas de valor igual ou inferior a colcheia podem ser ligadas entre si, em grupos cujo tempo totalize o valor base do compasso, no caso da implementação atual, os 3 tipos de compasso têm a semínima como valor base portanto seria de 1 tempo no total, o que implica ligar as colcheias duas a duas, as semicolcheias quatro a quatro e assim sucessivamente. Para o leitor da partitura, a sua compreensão fica significativamente mais fácil e esta simples medida terá um forte impacto na mancha gráfica. Na figura 5.1 comparam-se duas colcheias ligadas e outras duas separadas que não é mais do que duas simples notas numa partitura que pode ser verdadeiramente enorme, o que torna mais fácil compreender o contributo deste pequeno efeito.

Prosseguindo no campo da estética, existe ainda um outro ponto que a escassez do tempo não permitiu implementar: trata-se das barras finais da partitura. Quando uma partitura termina, são

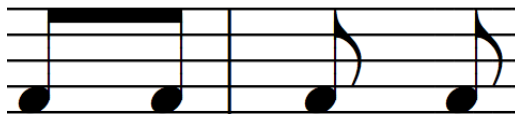


Figura 5.1: Comparação entre colcheias ligadas e não ligadas

colocadas no seu fim duas barras semelhantes à do compasso em que a última é mais grossa. Esta pode ser uma barra de repetição, precisando somente de dois pontos paralelos antes da barra para cumprir esse requisito. Um pormenor deste tipo dará ao utilizador uma noção de produto acabado, ao contrário do que acontece neste momento em que a partitura simplesmente não tem mais notas. Ao definir o compasso em que a partitura termina, encher-se-ia o espaço sobran-te com pausas para a completar.

A já mencionada exportação em MusicXML foi estudada como uma implementação que não é demasiado complexa para ser implementada e que poderá trazer algumas vantagens na quantidade de informação transmitida. No entanto, é necessário reunir com o utilizador final de modo a compreender a necessidade dessa implementação ou se este a considera desnecessária por se sentir satisfeito com a exportação em formato MIDI.

A adaptação do *Searchtonal* para reconhecer *multi-pitch*, ou seja, para conseguir reconhecer várias notas simultaneamente, é uma das previsões para um futuro bastante próximo. Este desenvolvimento dará à transcrição musical automática um potencial ainda maior pois vem expandir o seu já vasto leque de aplicações. No entanto, com a sua aplicação, é necessário adaptar a partitura para escrever em pautas diferentes e, possivelmente e no caso de haver vozes diferentes provenientes de instrumentos ou intervenientes diferentes, permitir que os utilizadores escolham ainda na janela principal quais são linhas melódicas que constituem cada uma das vozes pela aplicação de diferentes cores. Assim, aquando da transcrição, não serão já necessárias alterações desse cariz, ficando-se estas pelos erros típicos do reconhecimento.

As alterações à partitura final estão prontas a implementar; para isso é apenas necessário incluir a janela com as notas, pausas e acidentes a adicionar no campo *notes* da janela da partitura. A nível de código, foram já feitas algumas experiências no que toca à alteração dos valores da estrutura que demonstram ser possível fazê-lo. As funcionalidades “Drag& Drop” estão disponíveis no Qt e deverão alterar os valores da nota caso haja movimentação vertical da mesma na pauta ou alteração do acidente a ela associado, apagá-la da estrutura se esta for apagada e verificar a possibilidade de adicionar uma nova nota antes de o concluir efetivamente no caso de tal ser indicado pelo utilizador.

A inclusão de uma barra deslizante que controla a precisão deverá constar das futuras implementações e terá de levar ao novo cálculo e impressão de todas as notas na partitura. Deverá ter uma aspeto que se coadune com o grafismo geral do *software* e, para que seja de fácil utilização, sugere-se a predefinição de alguns valores no *slider* horizontal.

Inicialmente ponderou-se a interpretação da transição entre as notas de forma a compreender se estávamos perante algum tipo de efeito do canto (como um portamento ou um qualquer ornamento) ou uma normal transição entre duas notas. A análise das notas curtas sob o pretexto da procura de efeitos especiais foi retirada da equação, considerando-se apenas para zonas de vibrato nas quais pode ser indicada somente a nota média ou no caso de afinações no limiar da fronteira entre duas notas em que o programa por norma oscila entre as duas notas interpretando o sinal como uma sequência de pequenas notas próximas.



## Anexo A

# Código elaborado

Aqui apresenta-se uma cópia do código utilizado em determinadas secções para que o leitor consiga mais facilmente acompanhar os processos implementados.

### A.1 Metronomo.cpp

Código elaborado no ficheiro metronomo.cpp que apresenta a janela metronomo.ui, faz a ligação com a classe PlayBeep e trata todos os eventos associados ao metrónomo descritos na secção refsubsec:met:

```
#include "metronomo.h"
#include "ui_metronomo.h"
#include "playbeep.h"
#include <QSound>
#include <QElapsedTimer>
#include <QObject>
#include <QMainWindow>
#include <QtGui>
#include <QString>
#include <QThread>
#include <QEvent>
#include <QtCore/QCoreApplication>

// Miguel

bool on;
int compasso;
int time_division;
```

```

class playbeep;

Metronomo::Metronomo() : QMainWindow(), ui(new Ui::Metronomo())
{
    ui->setupUi(this);
    //on=true;
    bipThread = new PlayBeep();

    connect(ui->radioButton, SIGNAL(toggled(bool)), this, SLOT(Som(bool)));
}

void Metronomo::Som(bool on)                //recebe bpm
{
    qDebug() << "Vertical Slider: " << ui->verticalSlider->value();
    bpm = ui->verticalSlider->value();
    if(on==true)
    {
        bipThread->bpm2=bpm;
        bipThread->start();
        bipThread->stopped=false;
    }
    else
    {
        bipThread->stopped=true;
        bipThread->exit();
    }
}

void Metronomo::chgBPM(int bpmm)
{
    bpm = bpmm;
    bipThread->bpm2=bpm;
    //qDebug() << "chgBPM: " << bpm;
    //emit sendBPM(bpm);
}

Metronomo::~Metronomo()
{
    delete ui;
}

```

## A.2 Playbeep.cpp

Código elaborado no ficheiro PlayBeep.cpp, cuja função é a de tocar os sons do metrónomo criando um thread para o efeito:

```
#include <QSound>
#include <QElapsedTimer>
#include <QMainWindow>
#include <QThread>
#include "metronomo.h"
#include "ui_metronomo.h"
#include "playbeep.h"
#include <QMessageBox>
#include <QTimer>
#include <QString>
#include <QtCore>

using namespace std;
float mspt;

PlayBeep::PlayBeep()
{
    moveToThread(this);
    //metro = new Metronomo();
    //stopped = false;
}

PlayBeep::~PlayBeep(){}

void PlayBeep::run()
{
    //Debug: identifica o Thread
    /*
    qDebug("Thread id inside run %d", (int)QThread::currentThreadId());
    static int run = 0;
    QString temp = QString("Run: %1").arg(run++);
    qDebug("String address inside run %p", &temp);
    */
}
```

```

QElapsedTimer timer;
QSound bip2 (". / audio / start2 .wav");
QSound bip1 (". / audio / start1 .wav");
timer.start ();
int counter = measurepb;

forever{

    //qDebug() << "run BPM2: " << bpm2;

    //BPM correcto?
    if (bpm2>0)
        mspt= 60000/bpm2;
    else {
        qDebug() << "ERRO NO BPM! " << bpm2;
        mspt = 2000; //120bpm
    }    ///*****

    qDebug() << "measurepb:" << measurepb;
    qDebug() << "counter:" << counter;

    while(timer.elapsed() < mspt){
        if (this->stopped)
        {
            break;
        }
    }

    if (this->stopped)
    {
        break;
    }

    if (counter==measurepb)
    {
        bip1.play ();
        counter=1;
    }
    else
    {
        bip2.play ();

```

```

        counter++;
    }

    //qDebug() << "toquei ";
    timer.restart();

}
exec();
}

void PlayBeep::playMetro()
{
    // if(q==1)
    // stopped=false;
    qDebug() << "playMetro ";
    run();
}

void PlayBeep::stopMetro(int q)
{
    if(q==1)
        stopped = true;
    run();
}

```

### A.3 Partitura.cpp

Código escrito no ficheiro Partitura.cpp que define a sua classe e é responsável pela janela na qual consta a transcrição musical, assim como do tratamento das notas recebidas e da impressão de todos os objetos necessários na janela:

```

#include <QLabel>
#include <cmath>
#include <QtGui>
#include <QActionEvent>

```

```

#include <QEvent>
#include "Evento.h"

#include "MainWindow.h"
#include <vector>
#include "objsimples.h"
#include "Partitura.h"
#include "ui_Partitura.h"
#include "PitchSee.h"
#include "MidiCalls.h"
#include "metronomo.h"
#include <QVBoxLayout>
#include <qDebug>
#include <QGraphicsView>

#include <QtSvg>
#include "QtSvg/private/qsvggraphics_p.h"

float fim;

Partitura::Partitura(QWidget *parent) : QMainWindow(parent)
{

    ui=new Ui_Partitura();
    ui->setupUi(this);
    playbb = new PlayBeep();
    // metron = new Metronomo();

    setGeometry(50, 50, 1000, 850);
    fim=0;
    connect(ui->actionPrint , SIGNAL(activated()), this , SLOT(printas()));

    /*QWidget *window=new QWidget();
    QVBoxLayout *layout =new QVBoxLayout();
    QScrollBar *scroll=new QScrollBar();
    scroll->setOrientation(Qt::Vertical);
    scroll->setMinimum(0);
    scroll->setMaximum(100);
    scroll->setTracking(0);

```

```

scroll->resize(60,40);
layout->addWidget(scroll);
window->setLayout(layout);*/

//VBoxLayout *vBox = new QVBoxLayout(this);
//QLabel *networkPanel = new QLabel(hBox,"");
//QScrollBar *vScrollBar = new QScrollBar(Qt::Vertical);
//setCentralFrame( frame );

//      appoptions = new AppOptions2(this , iniSettings);

button_playmidip=new Button3();
button_playmidip->setImages(":/image/AudioMidi/playmidi-enable32.png",
                           ":/image/AudioMidi/playmidi-disable32.png",
                           ":/image/AudioMidi/playmidi-over32.png",
                           ":/image/AudioMidi/playmidi-click32.png",
                           ":/image/AudioMidi/playmidi-active32.png");
button_playmidip->setToolTip(tr("Play MIDI"));

button_stopp=new Button3();
button_stopp->setImages(":/image/AudioMidi/stop-enable32.png",
                       ":/image/AudioMidi/stop-disable32.png",
                       ":/image/AudioMidi/stop-over32.png",
                       ":/image/AudioMidi/stop-click32.png",
                       ":/image/AudioMidi/stop-active32.png");
button_stopp->setToolTip(tr("Stop"));

button_pausep=new Button3();
button_pausep->setImages(":/image/AudioMidi/pause-enable32.png",
                        ":/image/AudioMidi/pause-disable32.png",
                        ":/image/AudioMidi/pause-over32.png",
                        ":/image/AudioMidi/pause-click32.png",
                        ":/image/AudioMidi/pause-active32.png");
button_pausep->setToolTip(tr("Pause"));

ui->mainToolBar->addWidget(button_playmidip);
button_playmidip->adjustSize();

```

```

ui->mainToolBar->addWidget(button_pausep);
button_pausep->adjustSize();
ui->mainToolBar->addWidget(button_stopp);
button_stopp->adjustSize();

button_playmidip->setButtonHoldPress(true);
button_pausep->setButtonHoldPress(true);
button_stopp->setButtonHoldPress(true);

// Definicao das notas:
rsemib = new QSvgRenderer(QString(":/image/Figuras/semib.svg"));
rminim = new QSvgRenderer(QString(":/image/Figuras/minima.svg"));
rsemin = new QSvgRenderer(QString(":/image/Figuras/seminima.svg"));
rcolch = new QSvgRenderer(QString(":/image/Figuras/colch.svg"));
rsemic = new QSvgRenderer(QString(":/image/Figuras/semicolch.svg"));
rfusa = new QSvgRenderer(QString(":/image/Figuras/fusa.svg"));
// QSvgRenderer rsemif(QString(":/image/Figuras/semifus.svg"));

// Definicao das pausas:
rsemibp = new QSvgRenderer(QString(":/image/Figuras/pausasemibrev.svg"));
rminimp = new QSvgRenderer(QString(":/image/Figuras/pausaminim.svg"));
rseminp = new QSvgRenderer(QString(":/image/Figuras/pausaseminim.svg"));
rcolchp = new QSvgRenderer(QString(":/image/Figuras/pausacolch.svg"));
rsemicp = new QSvgRenderer(QString(":/image/Figuras/pausasemicolch.svg"));
rfusap = new QSvgRenderer(QString(":/image/Figuras/pausafusa.svg"));
// QSvgRenderer rsemifp(QString(":/image/Figuras/pausasemifusa.svg"));

// Acidentes
sust = new QSvgRenderer(QString(":/image/Figuras/sustenido.svg"));
beq = new QSvgRenderer(QString(":/image/Figuras/bequadro.svg"));
bemol = new QSvgRenderer(QString(":/image/Figuras/bemol.svg"));

quatro = new QSvgRenderer(QString(":/image/Figuras/4.svg"));
dois = new QSvgRenderer(QString(":/image/Figuras/2.svg"));
tres = new QSvgRenderer(QString(":/image/Figuras/3.svg"));
clave = new QSvgRenderer(QString(":/image/Figuras/clavesol_1.svg"));

// frame = new QFrame(this);

// connect(button_playmidip, SIGNAL(activated()),

```



```

mainw , SLOT(MainWindow::call_PlayVoiceToMIDI()));
// connect(button_stopp , SIGNAL(activated()),
mainw , SLOT(MainWindow::call_botaoParar()));
// connect(button_pausep , SIGNAL(activated()),
mainw , SLOT(MainWindow::call_botaoPausa()));
// connect(button_pausep , SIGNAL(disactivated()),
mainw , SLOT(MainWindow::call_botaoPausa()));

}

void Partitura::paintEvent(QPaintEvent *)
{
    QPainter painter(this);
    painter.setRenderHint(QPainter::Antialiasing , true);
    painter.setPen(QPen(Qt::darkGray , 1));
    painter.setBrush(QBrush(Qt::darkGray));

    // frame->show();

    //desenha as 5 linhas da Pauta
    for(int l=1; l<5; l++){
        int x1=20, x2=880; //x2=width()-30;
        int yt=140*l+40*(l-1);
        QPoint p11(x1, yt);
        QPoint p21(x2, yt);
        QPoint p12(x1, yt+10);
        QPoint p22(x2, yt+10);
        QPoint p13(x1, yt+20);
        QPoint p23(x2, yt+20);
        QPoint p14(x1, yt+30);
        QPoint p24(x2, yt+30);
        QPoint p15(x1, yt+40);
        QPoint p25(x2, yt+40);

        painter.drawLine(p11, p21);
        painter.drawLine(p12, p22);
        painter.drawLine(p13, p23);
        painter.drawLine(p14, p24);
    }
}

```

```

painter.drawLine(p15, p25);
clave->render(& painter, QRect(x1,yt-3,20,47));

if(msrp==3)      tres->render(& painter, QRect(x1+11,yt,45,37));
else
    if(msrp==2) dois->render(& painter, QRect(x1+11,yt+2,45,37));
    else quatro->render(& painter, QRect(x1+11,yt+2,45,37));

//      painter.drawImage(QRect(x1,yt-8,30,60),clave,QRect(0,0,80,160));

// Compassos
painter.drawLine(480, yt, 480, (yt+40));
painter.drawLine(880, yt, 880, (yt+40));
}

/*
QSvgRenderer rsemin(QString(
"C:/Users/Utilizador/Documents/Dropbox/tesemiguel/seminima.svg"));
int c=0, xx=480;
while(c<32){
    rsemin.render(& painter, QRect(xx+10*c,140,20,25));
    c++;
}

    //rsemin.render(& painter, QRect(60,140,20,25));

painter.end();

// Painter com caneta mais grossa para uniao de notas
QPainter grossa(this);
grossa.setRenderHint(QPainter::Antialiasing, true);
grossa.setPen(QPen(Qt::black, 2));
grossa.setBrush(QBrush(Qt::black));

grossa.drawLine(xx+15, 140, xx+44, 140);
grossa.drawLine(xx+15, 143, xx+44, 143);
grossa.end(); */

//NOTA DE TESTE
/* QImage smib;

```

```

        smib=QImage("C:/svn_static/SingingStudio/image/Figuras/colch.tiff");
        painter.drawImage(QRect(80,50,30,40),smib,QRect(0,0,503,820));*/

        emit pedi();
        ////qDebug() << "pedi()" ;
    }

    /*
    PAUTA ESCALAVEL

        painter.setPen(Qt::black);
        int unit = (int)(height()-20)/14;

        // painter.drawLine(5,2*unit,5,12*unit);
        for(int i=0; i<14;i++){
            if(i==0 || i==1 || i==7 || i==13) continue;
            painter.drawLine(5,i*unit,width()-10,i*unit);
        }
        int h1=pixmap_gclef.height();
        int a1=80;
        int a2=4*unit;
        float h2=(h1*a2)/(float)a1;
        int k1=34;
        float k2=(k1*a2)/(float)a1;
        QSize size_gclef(pixmap_gclef.size());
        size_gclef.scale(width(),h2,Qt::KeepAspectRatio);
        pixmap_gclef=pixmap_gclef.scaled(size_gclef,
        Qt::KeepAspectRatio,Qt::SmoothTransformation);
        painter.drawPixmap(7,2*unit-k2,pixmap_gclef);

    */

void Partitura::getNotas(std::vector<NoteLimits> *pitchNoteData){

    //Novo Construtor do painter
    QPainter painter(this);
    painter.setRenderHint(QPainter::Antialiasing, true);
    painter.setPen(QPen(Qt::black, 1));

```

```

painter.setBrush(QBrush(Qt::black));

bool flag=0, sus=0;
int istart, iend, inote, notap=0, ncomp=0,
ncompasso=1, npos=0, ok=0;
float mspt=500, tamanho, tamanhopausa=0, posh, ncomp2, resto=0;
float nsistema, nsistema2=0;

//qDebug() << "bpm3" << bpm3;
//qDebug() << "compasso" << msrp;
mspt=60000/(bpm3); //miliseconds per tick

/*QPixmap semib, mini, semin, colc, semic, fus, semif, semibp;
    semib=QPixmap("C:/svn_static/SingingStudio/image/Figuras/semibex2.png");
mini = QPixmap("C:/svn_static/SingingStudio/image/Figuras/mini3.png");
semin = QPixmap("C:/svn_static/SingingStudio/image/Figuras/semin3.png");
colc = QPixmap("C:/svn_static/SingingStudio/image/Figuras/colchex.png");
semic = QPixmap("C:/svn_static/SingingStudio/image/Figuras/semicolchex.png");
fus = QPixmap("C:/svn_static/SingingStudio/image/Figuras/fusex.png");
    semif = QPixmap("C:/svn_static/SingingStudio/image/Figuras/semifusex.png");

/* semibp=QPixmap("C:/svn_static/SingingStudio/image/Figuras/pausa1.png");
    minip = QPixmap("/Users/miguel/partst3/image/mini3.png");
seminp = QPixmap("/Users/miguel/partst3/image/mini3.png");
colcp = QPixmap("/Users/miguel/partst3/image/mini3.png");
semicp = QPixmap("/Users/miguel/partst3/image/mini3.png");
fusp = QPixmap("/Users/miguel/partst3/image/mini3.png");
semifp = QPixmap("/Users/miguel/partst3/image/mini3.png"); */

dadosNotas = pitchNoteData;

if (dadosNotas->size() == 0)
painter.drawText(50,80, "ERRO AO LER NOTAS");

for (int i=0; i<(int)dadosNotas->size(); i++) {

// dadosNotas->at(i).start = pitchNoteData->at(i).start;
// dadosNotas->at(i).end = pitchNoteData->at(i).end;

```

```

// dadosNotas->at(i).note = pitchNoteData->at(i).note;
// qDebug() << "dadosNotas->at(i).start" << dadosNotas->at(i).start;
istart= dadosNotas->at(i).start;
iend=   dadosNotas->at(i).end;
inote=  dadosNotas->at(i).note;
tamanho = ((iend-istart)*11.6)/mspt;

// if(tamanho < 5*11.6/mspt)
//     dadosNotas->at(i-1).end = dadosNotas->at(i).end;
// if(tamanhopausa < 7*11.6/mspt)
//     dadosNotas->at(i-1).end = dadosNotas->at(i).start;

if(fim !=0)
    tamanhopausa= (istart - fim)*11.6/mspt;

    // Prepara para correccao e alteracao das notas
/* if(tamanho < 5*5.8/mspt)
pitchNoteData->at(i-1).end = pitchNoteData->at(i).end;
if(tamanhopausa < 5*5.8/mspt)
iend=pitchNoteData->at(i+1).start;
// pitchNoteData->at(i-1).end = pitchNoteData->at(i).start; */

// qDebug() <<"istart:" << istart;
// qDebug() <<"iend:" << iend;
// qDebug() <<"fim:" << fim;
// qDebug() <<"Tamanho -" << i << ":" << tamanho;
// qDebug() <<"tamanhopausa: " << tamanhopausa;

// posh=i*30+40; // Trata da posicao das pausas

    if(i >=0){

        if(npos >= 8*msrp){
            if(ncomp==1)
                ncomp=0;
            else
                ncomp=1;

            ncompasso++;

```

```

        npos=0;
    }

    // para escrever noutro sistema
    ncomp2 = ((float)ncompasso/2);
    nsistema = ceil(ncomp2);
    // qDebug() << "1ncomp2:  " << ncomp2;
    // qDebug() << "1nsistema:" << nsistema;

    posh= 80 + ncomp*400 + npos*12.5;

    // qDebug() << "1.notap:      " << notap;
    // qDebug() << "1.posh:      " << posh;
    // qDebug() << "1.npos:      " << npos;
    // qDebug() << "1.ncompasso:" << ncompasso;
    // qDebug() << "1.ncomp:      " << ncomp;

    // Tratamento das pausas
    /* if (tamanhopausa >= 0.07 && tamanhopausa < 0.094){
        painter.drawPixmap(QRect(posh, 155+(nsistema)*180, 25, 25),
            semibp, QRect(0, 0, 600, 500));
        rsemifp.render(& painter, QRect
            (posh, 155+(nsistema-1)*180, 25, 25));
        npos += 1;
        // qDebug() << "PAUSA Semifusa ";
    } */

    if (tamanhopausa >= 0.094 && tamanhopausa < 0.187){
        // painter.drawPixmap(QRect(posh, 155+(nsistema)*180, 25, 25),
            semibp, QRect(0, 0, 600, 500));
        rfusap->render(& painter, QRect
            (posh, 155+(nsistema-1)*180, 20, 25));
        npos += 1;
        // if (npos > 31)
        // qDebug() << "PAUSA Fusa ";
    }

    if (tamanhopausa > (msrp-npos/8)){

```

```

        tamanhopausa=msrp-npos/8;
    }

    if (tamanhopausa >=0.187 && tamanhopausa <0.375){
        // painter.drawPixmap(QRect(posh,155+(nsistema-1)*180,25,25),
        semibp,QRect(0,0,600,500));
        rsemicp->render(& painter, QRect
        (posh,155+(nsistema-1)*180,20,25));
        npos += 2;
        //qDebug() <<"PAUSA Semicolcheia";
    }

    if (tamanhopausa >=0.375 && tamanhopausa <0.75){
        // painter.drawPixmap(QRect(posh,155+(nsistema-1)*180,25,25),
        semibp,QRect(0,0,600,500));
        rcolchp->render(& painter, QRect
        (posh,155+(nsistema-1)*180,20,25));
        npos += 4;
        //qDebug() <<"PAUSA Colcheia";
    }

    if (tamanhopausa >=0.75 && tamanhopausa <1.5){
        // painter.drawPixmap(QRect(posh,155+(nsistema-1)*180,25,25),
        semibp,QRect(0,0,600,500));
        rseminp->render(& painter, QRect
        (posh,145+(nsistema-1)*180,20,25));
        npos += 8;
        //qDebug() <<"PAUSA Seminima";
    }

    if (tamanhopausa >=1.5 && tamanhopausa <3){
        // painter.drawPixmap(QRect(posh,155+(nsistema-1)*180,25,25),
        semibp,QRect(0,0,600,500));
        rminimp->render(& painter, QRect
        (posh,155+(nsistema-1)*180,20,20));
        npos += 16;
        //qDebug() <<"PAUSA Minima";
    }

    if (tamanhopausa >=3){

```

```

        // painter.drawPixmap(QRect(posh,155+(nsistema-1)*180,25,25),
semibp,QRect(0,0,600,500));
        rsemibp->render(& painter , QRect
(posh,155+(nsistema-1)*180,20,20));
        npos += 32;
        //qDebug() <<"PAUSA Semibreve ";
    }

// Actualizacao da posicao horizontal
if (npos>=8*msrp){
    if (ncomp==1)
        ncomp=0;
    else
        ncomp=1;
        ncompasso++;
        npos=0;
    }

    //para escrever noutro sistema
    ncomp2 = ((float)ncompasso/2);
    nsistema = ceil(ncomp2);
    //qDebug() << "2ncomp2: " <<ncomp2;
    //qDebug() << "2nsistema:" <<nsistema;

    posh= 80 + ncomp*400 + npos*12.5;

// Tratamento das notas: posicao vertical
switch(inote){
    case 36 : notap=185; break; //Do1
    case 37 : notap=185; sus=1; break;
    case 38 : notap=180; break; //Re
    case 39 : notap=180; sus=1; break;
    case 40 : notap=175; break;
    case 41 : notap=170; break;
    case 42 : notap=170; sus=1; break;
    case 43 : notap=165; break; //Sol
    case 44 : notap=165; sus=1; break;
    case 45 : notap=160; break; //La1
    case 46 : notap=160; sus=1; break; //
    case 47 : notap=155; break; //

```



```

case 48 : notap=150; break ; //Do2
-O Traco Inferior !!!
case 49 : notap=150; sus=1; break ;
case 50 : notap=145; break ; //Re2
case 51 : notap=145; sus=1; break ;
case 52 : notap=140; break ; //Mi
case 53 : notap=135; break ; //Fa2
case 54 : notap=135; sus=1; break ;
case 55 : notap=130; break ; //Sol2
case 56 : notap=130; sus=1; break ;
case 57 : notap=125; break ; //La2
case 58 : notap=125; sus=1; break ;
case 59 : notap=120; break ; //Si2
case 60 : notap=115; break ; //Do3
case 61 : notap=115; sus=1; break ;
case 62 : notap=110; break ;
case 63 : notap=110; sus=1; break ;
case 64 : notap=105; break ; //Mi3
case 65 : notap=100; break ; //Fa3
case 66 : notap=100; sus=1; break ;//
case 67 : notap=95; break ;// Sol3
case 68 : notap=95; sus=1; break ;//
case 69 : notap=90; break ;// La3

Traco Superior !!!

case 70 : notap=90; sus=1; break ;//

|

case 71 : notap=85; break ;//

|

case 72 : notap=80; break ;// Do4

v

case 73 : notap=80; sus=1; break ;
case 74 : notap=75; break ;// Re4
case 75 : notap=75; sus=1; break ;
case 76 : notap=70; break ;//Mi4
case 77 : notap=65; break ;//Fa4
case 78 : notap=65; sus=1; break ;
case 79 : notap=60; break ;// Sol4
case 80 : notap=60; sus=1; break ;
case 81 : notap=55; break ;//La4
case 82 : notap=55; sus=1; break ;

```

```

        case 83 : notap=50; break;
        case 84 : notap=45; break ;//Do5
        case 85 : notap=45; sus=1; break ;
    }

    if (nsistema > 1)
        notap += (nsistema - 1) * 180;

    // qDebug() << "2. notap: " << notap;
    // qDebug() << "2. posh: " << posh;
    // qDebug() << "2. npos: " << npos;
    // qDebug() << "2. ncompasso:" << ncompasso;
    // qDebug() << "2. ncomp: " << ncomp;

    ok = 0;
    while (ok != 1) {

        if (tamanho >= 0.07 && tamanho < 0.094) {
            painter.drawPixmap(QRect(posh, notap + 12, 25, 35),
semif, QRect(0, 0, 600, 850));
            npos += 1;
            flag = 1;
            // qDebug() << "Semifusa";
        }

        if (resto != 0) {
            tamanho = resto;
            resto = 0;
        }

        if (tamanho > (msrp - npos / 8)) {
            resto = tamanho - (msrp - npos / 8);
            tamanho = msrp - npos / 8;
            ok = 0;
        }
        else
            ok = 1;

        if (tamanho >= 0.094 && tamanho < 0.187) {
            // painter.drawPixmap(QRect

```

```

(posh , notap + 12 , 25 , 35) , fus , QRect(0 , 0 , 600 , 850));
    rfusa->render(& painter , QRect
(posh , notap + 20 , 23 , 32));
    npos += 1;
    flag = 1;
    // qDebug() <<"Fusa ";
}

if(tamanho >= 0.187 && tamanho < 0.375){
    // painter.drawPixmap(QRect
(posh , notap + 12 , 25 , 35) , semic , QRect(0 , 0 , 600 , 850));
    rsemic->render(& painter , QRect
(posh , notap + 14 , 20 , 30));
    npos += 2;
    flag = 1;
    // qDebug() <<"Semicolcheia ";
}

if(tamanho >= 0.375 && tamanho < 0.75){
    // painter.drawPixmap(QRect
(posh , notap + 12 , 25 , 35) , colc , QRect(0 , 0 , 600 , 850));
    rcolch->render(& painter , QRect
(posh , notap + 14 , 20 , 30));
    npos += 4;
    flag = 1;
    // qDebug() <<"Colcheia ";
}

if(tamanho >= 0.75 && tamanho < 1.25){
    // painter.drawPixmap(QRect
(posh , notap + 9 , 20 , 35) , semin , QRect(0 , 0 , 400 , 750));
    rsemin->render(& painter , QRect
(posh - 3 , notap + 15 , 20 , 30));
    npos += 8;
    flag = 1;
    // qDebug() <<"Seminima ";
}

if(tamanho >= 1.25 && tamanho < 1.75){
    // painter.drawPixmap(QRect

```

```

(posh , notap+9,20,35),semin ,QRect(0,0,400,750));
        rsemin->render(& painter , QRect
(posh-3,notap+15,20,30));
        painter.drawPoint(posh+15, notap+40);
        npos += 12;
        flag = 1;
        //qDebug() <<"Seminima c/ ponto";
    }

    if (tamanho>=1.75 && tamanho<2.25){
        // painter.drawPixmap(QRect
(posh , notap+6,25,35),mini ,QRect(0,0,450,700));
        rminim->render(& painter , QRect
(posh , notap+14,20,30));
        npos += 16;
        flag = 1;
        //qDebug() <<"Minima ";
    }

    if (tamanho>=2.25 && tamanho<3.5){
        // painter.drawPixmap(QRect
(posh , notap+6,25,35),mini ,QRect(0,0,450,700));
        rminim->render(& painter , QRect
(posh , notap+14,20,30));
        painter.drawPoint(posh+15, notap+40);
        npos += 24;
        flag = 1;
        //qDebug() <<"Minima c/ ponto";
    }

    if (tamanho>=3.5 && tamanho<4.5){
        // painter.drawPixmap(QRect
(posh , notap+3,30,40),semib ,QRect(0,0,400,700));
        npos += 32;
        flag = 1;
        rsemib->render(& painter , QRect
(posh , notap+24,20,30));
        //qDebug() <<"Semibreve ";
    }

```

```

// Linhas suplementares se necessarias
if (flag)
{
    // qDebug() << "flag ";
    if (notap > 0 && notap <= (90 + ((nsistema - 1) * 180)) ) {
        for (int lin = 130 + (nsistema - 1)
* 180; lin >= notap + 40; lin = lin - 10) {
            painter.drawLine(posh, lin, posh + 18, lin);
            // qDebug() << "Desenhou traco superior";
        }
    }
    if (notap >= (150 + ((nsistema - 1) * 180)) ) {
        for (int lin = 190 + (nsistema - 1)
* 180; lin <= (notap + 40); lin = lin + 10) {
            painter.drawLine(posh, lin, posh + 17, lin);
            // qDebug() << "Desenhou traco inferior";
        }
    }

    // acidentes
    if (sus) {
        sust->render(& painter
, QRect(posh - 20, notap + 22, 35, 35));
        sus = 0;
    }
    flag = 0;
} // fim flag

// Actualizacao da posicao horizontal
if (npos >= 8 * msrp) {
    if (ncomp == 1)
        ncomp = 0;
    else
        ncomp = 1;
        ncompasso++;
        npos = 0;
}
// para escrever noutro sistema
ncomp2 = ((float) ncompasso / 2);

```

```

        nsistema2 = ceil(ncomp2);
        posh= 80 + ncomp*400 + npos*12.5;
        if (nsistema2>nsistema)
            notap += 180;

        //qDebug()<< "notap:" << notap;
        //qDebug()<< "nsistema:"<<nsistema;
        //qDebug() << "npos: " << npos;
        //qDebug() << "fimfim: " << fim;
    }

    fim = iend;
//Para medir tamanho da pausa seguinte , se existir
    //qDebug() <<" ";
    }
}

Partitura::~Partitura()
{
    delete ui;
    delete button_pausep;
    delete button_playmidip;
    delete button_stopp;
}

void Partitura::printas(){

    QLabel *previewLabel;
    QDesktopWidget *dsk = QApplication->desktop(); // dsk->screen()->winId()
    QPixmap pagtotal = QPixmap::grabWindow(dsk->window()->winId());
    previewLabel = new QLabel;
    previewLabel->setPixmap(pagtotal.scaled(previewLabel->size()));

    QPrinter printer;
    printer.setOrientation(QPrinter::Portrait);
    QPrintDialog dlg(& printer);
    if (dlg.exec()==QDialog::Accepted) {

```

```
        printer.newPage();
        QPainter p(& printer);
        QPixmap resized =
            pagtotal.scaledToWidth(printer.pageRect().width());
        p.drawPixmap(0,0, resized);
        p.end();
    }
}

void Partitura::getbpmvalue(int bpm)
{
    bpm3=bpm;
    //qDebug() << "bpmp:" << bpm3;
}

void Partitura::getmeasure(int msr)
{
    msrp=msr;
    //qDebug() << "msrp:" << msrp;
}
```





# Referências

Jasmin Blanchette e Mark Summerfield. *C++ GUI programming with Qt 4*. Pearson Hall in association with Trolltech Press, Upper Saddle River, NJ, 2006. 2006013376 9780131872493 Jasmin Blanchette, Mark Summerfield. ill. ; 24 cm. + 1 CD-ROM (4 3/4 in.) Includes index. "The accompanied CD-ROM includes the open source edition of Qt 4.1.1 for Windows, Mac, Linux, and many Unixes, as well as MinGW, a set of freely available development tools that can be used to build Qt application on Windows, and also the source code for the book's examples--P. [4] of cover. 1. Getting started – 2. Creating dialogs – 3. Creating main windows – 4. Implementing application functionality – 5. Creating custom widgets – 6. Layout management – 7. Event processing – 8. 2D and 3D graphics – 9. Drag and drop – 10. Item view classes – 11. Container classes – 12. Input/output – 13. Databases – 14. Networking – 15. XML – 16. Providing online help – 17. Internationalization – 18. Multithreading – 19. Creating plugins – 20. Platform-specific features – 21. Embedded programming – App. A. Installing Qt – App. B. Introduction to C++ for Java and C# programmers.

Maria José Borges e José Maria Cardoso. *História da música - manual do aluno*, 2003.

Peter Desain e Henkjan Honing. *Music, mind, and machine : studies in computer music, music cognition, and artificial intelligence*. Thesis Publishers, Amsterdam, 1992. Peter Desain and Henkjan Honing. ill. ; 24 cm. Includes bibliographical references. Studies in computer music, music cognition and artificial intelligence.

Aníbal Ferreira. Report regarding the first year of activities of the project: Assistive real-time technology in singing (artts). Technical report, FCT, 2010.

Michael Good. Musicxml: An internet-friendly format for sheet music, 2001.

Donald J Grout e Claude V Palisca. *História da música ocidental*. Gradiva, Lisboa, 1988. Donald J. Grout, Claude V. Palisca revisão técnica de Adriana Latino il. 24 cm Título original: A history of western music Latino, Adriana.

Luís L Henrique. *Acústica musical*. Fundação Calouste Gulbenkian, Lisboa, 2002. Luís L. Henrique 26 cm inclui 1 CD-ROM.

Daniel J. Levitin. *This is your brain on music : the science of a human obsession*. Dutton, New York, N.Y., 2006. 2006009055 9780525949695 Daniel J. Levitin. ill. ; 24 cm. Includes bibliographical references (p. [271]-300) and index. Introduction: I love music and I love science—why would I want to mix the two? – What is music? : from pitch to timbre – Foot tapping : discerning rhythm, loudness, and harmony – Behind the curtain : music and the mind machine – Anticipation : what we expect from Liszt (and Ludacris) – You know my name, look up the number : how we categorize music – After dessert, Crick was still four seats away from me : music, emotion, and the reptilian brain – What makes a musician? : expertise dissected – My favorite things : why do we like the music we like? – The music instinct : evolution's nr.1 hit.

Daniel Molkentin e Books24x7 Inc. The book of qt 4 the art of building qt applications, 2007. Qt 4, Einführung in die Applikationsentwicklung. English [electronic resource] : Daniel Molkentin. Computer document. (Norwood, Mass. : Books24x7.com [generator]) Mode of access: Internet via World Wide Web. Title from title screen. Digitized and made available by: Books24x7.com. Includes bibliographic references and index. \*\*See URL(s) Internet Resource.

Robert Rowe. *Interactive music systems : machine listening and composing*. MIT Press, Cambridge, Mass., 1993. 92016388 /MN Robert Rowe. ill. ; 24 cm. + 1 computer laser optical disk (4 3/4 in.) Includes bibliographical references (p. [265]-272) and index.

José Ventura. Biofeedback da voz cantada. Master's thesis, Faculdade de Engenharia da Universidade do Porto.