Composição Algorítmica de Música - Desenvolvimento de Software (*HarmoniMIDI*)

Gabriel Ferreira Morais

Universidade Federal de Minas Gerais (UFMG)

Departamento de Ciência da Computação

gabriel.morais@dcc.ufmg.br

Orientador(a): Mirella Moura Moro

mirella@dcc.ufmg.br

Resumo. *Com o advento da computação e suas tecnologias associadas, é possível, hoje, simular alguns processos de composição musical. Este artigo busca apresentar, descrever e explicar a aplicação "**HarmoniMIDI**", desenvolvida para resolver o problema de harmonização de melodias. Esta aplicação segue o modelo matemático de Cadeias de Markov, que, segundo pesquisa realizada em um estudo prévio, se provou um dos modelos mais interessantes para aplicações que buscam resolver algum problema relacionado à composição algorítmica de música.*

Palavras chave*: Composição algorítmica. Algoritmo. Harmonização. Melodia. Obras Musicais.*

1. Introdução

Composição musical caracteriza-se pela geração de obra ou trecho de música com auxílio ou por meio de um algoritmo. Neste estudo, serão considerados apenas algoritmos computacionais.

A composição utilizando algoritmos e computadores propriamente dita data de meados de 1955, tendo sido proposta primeiramente por Lejaren Hiller e Leonard Isaacson, da Universidade de Illinois nos Estados Unidos. Devido ao sempre crescente uso da computação para diversas áreas (inclusive a da Música), ainda é um interessante objeto de estudo, já que

ambas as áreas (Computação e Música) possuem um vasto campo de possibilidades de novas pesquisas, descobertas e aplicações.

Pode ser categorizada pelos seguintes quesitos:

- Quantidade de intervenção humana: Autonomia do algoritmo em relação ao processo e ao resultado [MAZUROWSKI, 2012];
- Produto produzido: A saída pode ser uma obra notada musicalmente (exemplo: partitura), uma síntese real de som ou a mistura dos dois (combinação de notação com execução sonora da composição);
- O modelo de algoritmo. As principais categorias são [FREITAS, 2015] e [PAPADOPOULOS; WIGGINS, 2000]: Tradutivo, Matemático (Cadeias de Markov), Modelos Gramáticos, Modelos Evolucionários (Genéticos), Modelos de Improvisação de Máquina.

Esse trabalho possui uma motivação pessoal muito importante, uma vez que o autor, além de aluno de Ciência da Computação, também faz Formação Complementar na Escola de Música da UFMG.

O objetivo específico deste trabalho é utilizar do conhecimento obtido pelos estudos acerca do tema para desenvolver uma aplicação/algoritmo voltada à composição algorítmica. O nome da aplicação é "*HarmoniMIDI*" (juntando "harmonia" com "MIDI"), e utiliza de regras da Teoria Musical para harmonizar uma melodia fornecida pelo usuário.

O trabalho é estruturado em:

- Pesquisa e leitura de artigos e projetos voltados ao tema;
- Definição dos parâmetros de trabalho;
- Desenvolvimento do algoritmo/programa;
- Elaboração do presente artigo.

2. Referencial Teórico

2.1 Conceitos

Na presente seção serão definidos conceitos importantes para o entendimento da aplicação e do presente documento.

2.1.1 Conceitos computacionais

Algoritmo: Em poucas palavras, um conjunto determinado de passos ou regras que, aplicadas a um certo número finito de dados de entrada resolvem um problema (o que pode ser de várias naturezas, até musical). Para o escopo do trabalho, o algoritmo pode ser descrito como o conjunto de regras que, após ter recebido algum tipo de instrução, poderá gerar, como sua saída, uma composição musical, seja esta em forma de notação de partitura ou até mesmo algum tipo de som (desde que esse seja, de fato, um som que apresente musicalidade, e não somente ruídos aleatórios).

Função: Uma sequência definida de comandos a serem executados, podendo manipular e/ou retornar dados. Também pode receber parâmetros de entrada.

Inteligência Artificial (IA): Ramo da Ciência da Computação que busca construir sistemas que simulem a capacidade humana de resolver problemas.

MIDI: Musical Instrument Digital Interface ou Interface Digital de Instrumentos Musicais, é um protocolo, linguagem e interface que permite a comunicação entre computadores, instrumentos musicais e outros dispositivos. É considerada uma das ferramentas mais importantes e poderosas na Música.

Software de código aberto: É um programa cujo código fonte é disponibilizado, e é licenciado de tal forma que permite que o usuário, além de ter acesso, modifique e distribua o *software* sem custos.

API: Application Programming Interface ou Interface de Programação de Aplicações é um conjunto de rotinas e padrões de programação para acesso a uma aplicação ou serviço Web. Com ela é possível que se desenvolva software baseado em um outro serviço.

Biblioteca (lib): Coleção de subprogramas e métodos utilizados para desenvolvimento de *software*. Permite o compartilhamento de código e facilita a modularização de programas, evitando repetição desnecessária de código.

Objeto: É uma referência a um local da memória que possui um valor. Pode ser uma variável, uma função ou estrutura de dados.

Variável: É um objeto capaz de reter um valor (ou expressão). Existem apenas em tempo de execução. Durante o desenvolvimento, são associadas à nomes para referenciação.

2.1.2 Conceitos musicais ([FREITAS, 2007], [PAZ; MOREIRA, 2012])

Melodia: Sequência de notas que faz sentido musicalmente, com sons que têm, normalmente, diferentes durações.

Acorde: Conjunto de notas com alguma relação harmônica entre elas. São percebidas soando em conjunto, mas podem soar simultaneamente ou sucessivamente. Normalmente, os acordes têm 3 ou mais notas.

Harmonia: Uso simultâneo de várias alturas, ou acordes. É também um importante campo de estudo na Música.

Ritmo: Define como os sons e silêncios (pausas) estão organizados em uma peça musical.

Tonalidade: Quando tratamos de um sistema tonal, cada música (ou trecho) tem, geralmente, um tom que é considerado o centro tonal da obra. Este costuma ser o primeiro e último acordes de uma peça.

Compasso: São segmentos agrupados dos pulsos de uma música, com duração definida pelo tempo da música. Cada compasso contém um conjunto de notas e figuras musicais que forma a melodia e a harmonia. Geralmente o número de pulsos por compasso é especificado no começo da partitura.

Pausa: Ausência de nota. É expressa com sua duração, assim como uma nota normal.

Cifra: É um sistema de notação musical, utilizado para denotar acordes a serem tocados por algum instrumento musical.

Escala: Escala musical é uma série de notas selecionadas dentre as doze notas musicais. Cada nota da escala está atribuída a um grau equivalente à sua posição. Sendo assim, a primeira nota é chamada de 1° grau ou tônica, a segunda nota é o 2° grau, e assim respectivamente. As notas também podem ser identificadas pela sua posição, como terça e quinta, referentes ao 3° e 5° graus, respectivamente. Neste programa trabalha-se com duas escalas, a maior e a menor (também chamadas de modo maior e menor).

Cadência: Sequência de acordes que produz um efeito harmônico característico. Por exemplo: cadência perfeita: Dominante (grau V) -> Tônica (grau I).

Campo Harmônico: Campo harmônico trata dos acordes disponíveis para a harmonia de uma escala. Uma explicação mais generalizada seria apresentar o campo harmônico como um conjunto de acordes que mais harmoniza com uma determinada escala.

Dissonância: É uma percepção de tensão gerada por duas ou mais notas, geralmente tocadas simultaneamente. Na música popular, costuma ser evitada.

2.2 Cadeias de Markov

O modelo utilizado para o algoritmo de harmonização no "HarmoniMIDI" foi o de **Cadeias de Markov**. O estudo feito na primeira parte do projeto mostrou que esse, junto com o modelo de Algoritmos Genéticos, é um dos mais interessantes quando se está trabalhando com Composição Algorítmica.

É um modelo relacionado com probabilidade e estatística. Esses processos, que são chamados de estocásticos, descrevem uma sequência aleatória de eventos e dependente de tempo [FREITAS, 2007], e possuem um conjunto finito de estados. A distribuição de probabilidade do cálculo de cada próximo estado depende exclusivamente do estado atual, ou seja, independe de eventos precedentes. É possível usar cadeias de "alta ordem", o que significa que mais de um estado pode afetar o estado seguinte (uma cadeia de segunda ordem usa o estado atual e o último para calcular o próximo). Quanto maior a ordem, maior a complexidade do sistema.

A aplicação utiliza uma cadeia de Markov de primeira ordem, com algumas modificações. Para cada compasso da melodia provida, são analisadas suas notas, fazendo um *lookup* na matriz de pesos. O estado anterior é analisado, assim como uma eventual cadência formada pelo acorde anterior e o atual.

3. Desenvolvimento

Esta seção detalha o processo de desenvolvimento e outras informações úteis sobre a aplicação "*HarmoniMIDI"*.

3.1 Linguagem de programação e bibliotecas

A linguagem de programação escolhida para ser utilizada no desenvolvimento do programa foi a Python. É uma linguagem de programação de sintaxe simples, curva de aprendizado consideravelmente razoável e bastante utilizada para soluções que abrangem problemas de natureza musical.

A principal biblioteca utilizada, *Mingus,* fornece várias facilidades ao utilizar-se *MIDI*, manipular notas, acordes, escalas e progressões. Sua *API* e documentação não são atualizadas há alguns anos, e alguns métodos tiveram de ser reescritos. Porém, foi possível tirar bastante proveito das funcionalidades presentes.

3.2 Parâmetros

3.2.1 Entrada

- Caminho do arquivo .midi contendo a melodia
- (Opcional) Opção de exportar áudio
- (Opcional) Opção de gerar partitura

3.2.2 Saída

- (Opcional) Arquivo WAV com áudio do resultado
- (Opcional) Arquivo PDF com partitura do resultado
- (Opcional) Arquivo WAV com áudio do resultado (rearmonizado)
- (Opcional) Arquivo PDF com partitura do resultado (rearmonizado)

3.3 Algoritmo

Aqui serão explicadas as partes mais importantes do algoritmo desenvolvido para esta aplicação.

3.3.1 Harmonizador

Recebidos os parâmetros de entrada, usando método da classe midi_file_in da biblioteca *Mingus*, se obtém um objeto que contém as informações do arquivo .midi de

entrada, com a seguinte hierarquia: (figura: composição que contém faixas que contém compassos que contém notas)

Nesse momento, se tem todas as notas da melodia, assim como os compassos separados. Usando métodos de *Mingus* e também de outra biblioteca Python chamada *music21*, obtém-se a tonalidade da melodia, e se esta é em modo menor ou maior.

Só então é executado o método harmonizador propriamente dito. Este percorre os compassos da melodia, e, para cada nota, faz um *lookup* em uma matriz de pontuações (**Figura 1**), onde está expresso a pontuação que aquela nota representa para cada grau da escala. Além disso, quando a nota é a primeira do compasso, ou seja, está no primeiro tempo do compasso, que é um tempo forte, é recebida uma pontuação extra.

```
scores = {
        'I':   {0: 0.5,   1: -0.2,  2: 0,     3: 0,     4: 0.35,
        'ii':  {0: 0,     1: 0.1,   2: 0.5,   3: -0.2,  4: 0,
        'iii': {0: 0,     1: 0,     2: 0,     3: 0.1,   4: 0.5,
        'IV':  {0: 0.35,  1: 0,     2: 0,     3: 0,     4: 0.1,
        'V':   {0: 0,     1: 0,     2: 0.35,  3: 0,     4: 0,
        'vi':  {0: 0,     1: 0.35,  2: 0,     3: 0,     4: 0.35,
        'vii': {0: -0.2,  1: 0,     2: 0,     3: 0.35,  4: 0,
}
```

Figura 1: Matriz de pontuações para uma escala maior. Quando a melodia é menor, é usada outra matriz, semelhante. Cada grau tem os números 0 a 11, representando as 12 notas (Dó, Dó sustenido, …, Si)

Essa matriz leva em conta que, para cada grau da escala, a primeira nota (tônica) tem a maior pontuação, seguida dos quarto e quinto graus, sétimo, e por último segundo grau bemol (que tem pontuação negativa, pois geralmente causa uma dissonância muito grande).

No final da iteração de cada compasso, é conferido qual grau possui maior pontuação. Neste momento, algumas considerações são feitas: caso seja o primeiro ou último compasso, e o 1o grau é um dos graus com maior pontuação, mesmo que não o mais pontuado, este é o escolhido para o compasso. Isso se dá porque, geralmente, as obras iniciam e terminam com o primeiro grau da escala.

Além disso, é levado em consideração o acorde anterior, para que não se repita acordes.

No término do algoritmo, é retornado uma lista com os acordes escolhidos em forma de números romanos (I, ii, iii, IV, V, vi, vii), uma escrita bastante comum quando se trata de graus em escalas musicais.

3.3.2 Reharmonizador

Percorrendo os acordes, é utilizado o método *substitute* da classe *Progressions* da biblioteca *Mingus*, que retorna uma lista de possíveis substitutos para um acorde, levando em conta sua função harmônica. É escolhido, aleatoriamente, um dos acordes da lista, e esse acorde é avaliado segundo a regra: caso ele "combine" com a parte da melodia referente ao compasso que esse acorde está, ele entra na sequência, substituindo o antigo.

3.3.3 Exportador de áudio

Para fazer a exportação do resultado do programa em formato de áudio, foi usada a aplicação de código aberto *fluidsynth.* O código para essa exportação foi baseado no código do programador Devon Bryant feito em 2012 (todos os direitos reservados), com mudanças de adequação.

3.3.4 Gerador de partitura

Para gerar a partitura do resultado final, foi utilizado o módulo open source *Lilypond*, porém com modificações para atender às necessidades do programa. As modificações foram:

- Para poder escrever a cifra dos acordes na partitura, foi desenvolvido um método que, a partir das notas de um acorde, devolve sua notação cifrada. A biblioteca *Mingus* já possui um método semelhante, porém, a notação devia ser formatada especificamente do jeito que *Lilypond* entendesse, então não pôde ser utilizada.
- Para determinar qual clave utilizar na partitura (Sol ou Fá), é analisada a média de alturas das notas das partituras.
- Na partitura, é indicada a tonalidade da obra, passando o tom e o modo (maior ou menor). Originalmente, essa informação não aparecia.
- O nome da obra e do autor, caso fornecidos pelo usuário, são escritos na partitura.

3.4 Funcionalidades

3.4.1 Harmonizar melodia MIDI

A principal premissa do aplicativo. Recebe uma melodia em arquivo .midi, e gera uma sequência possível de acordes para a melodia.

3.4.2 Detecção modo menor ou maior

O código é capaz de detectar qual o modo da melodia, ou seja, se é uma melodia pertencente à uma escala maior ou menor.

3.4.3 Detecção de pausas

Ao "ler" a melodia e gerar os acordes, o código é capaz de detectar pausas na melodia, e isso influencia o resultado. Um compasso composto apenas de pausas, por exemplo, gera uma pausa na sequência de acordes.

3.4.5 Possibilidade de exportar o resultado em formato WAV

Depois de gerada a harmonia, é apresentado ao usuário um botão para ouví-la. Porém, também existe a opção de exportar o resultado para o formato WAV, que é executável em quase todos os dispositivos de áudio. Essa opção é apresentada ao abrir o programa.

3.4.6 Possibilidade de escrever a partitura do resultado em um arquivo PDF (com melodia e acordes cifrados)

Também é possível checar a opção de escrever o resultado em uma partitura em um arquivo PDF. Caso seja marcada, o usuário também pode informar nome do autor e/ou da obra. A opção de gerar partitura é apresentada ao abrir o programa.

3.4.4 Possibilidade de rearmonizar os acordes encontrados

Após gerar a harmonia, o usuário tem a opção de rearmonizar a sequência de acordes gerada. Isso é, para cada acorde (segundo uma regra de probabilidade), existe a possibilidade de trocá-lo por outro acorde que possua a mesma função harmônica do que o original. Isso pode levar a combinações interessantes, e diferentes, de acordes.

Obs: caso a opção de exportar o resultado para WAV e/ou gerar partitura tiverem sido marcadas, a rearmonização também será exportada.

3.4.7 Detecção de clave mais adequada para partitura (Sol ou Fá)

O código é capaz de detectar qual clave (Sol ou Fá) é mais adequada para escrever a harmonia (quando a opção de gerar partitura é marcada).

3.4.8 Busca de não repetição de acordes

O código faz o possível para que, na sequência de acordes gerados, não ocorra dois iguais consecutivos, pois isso, geralmente, causa um efeito desagradável para o ouvinte (mesmo que faça sentido harmonicamente).

3.4.9 Interface Gráfica

Para tornar o programa mais funcional e facilitar seu uso, foi desenvolvida uma interface gráfica simples, usando a biblioteca *Tkinter* do Python. Ao iniciar o programa, é requisitado ao usuário que digite o caminho do arquivo .midi, e que escolha se deseja exportar para WAV e/ou exportar a partitura. A próxima seção (Interface Gráfica, 3.4) contém mais detalhes.

3.4.10 Executável

Para facilitar uma possível e futura distribuição da aplicação, foi utilizada a ferramenta *pyinstaller* para gerar um executável autocontido para o aplicativo. Isso significa que o usuário que obter esse executável não precisaria, em teoria, ter Python (nem as outras bibliotecas utilizadas) instalado em sua máquina. Na realidade, apenas uma dependência necessária para sua execução não pôde ser carregada pela ferramenta, a *Lilypond*, módulo responsável por gerar a partitura. Por isso, para executar o *HarmoniMIDI,* é preciso instalar *Lilypond* antes (sudo apt-get install lilypond). Depois disso é só clicar no executável.

3.5 Interface Gráfica

Na janela inicial, é requisitado que o usuário forneça o caminho para o arquivo .midi que contém a melodia de entrada, e são apresentadas opções pós-execução (Figura 2).

Figura 2: Janela inicial

Caso o usuário clique no botão "HarmoniMIDI-me" sem especificar um caminho para a melodia, é apresentado uma mensagem de erro (Figura 3).

Figura 3: Erro ao não especificar caminho

O mesmo ocorre quando o programa não consegue localizar o arquivo especificado (Figura 4).

Figura 4: Erro ao especificar caminho inexistente

Caso marque a opção “Gerar partitura”, o usuário pode fornecer nome da obra e do autor (Figura 5).

Figura 5: Especificando nome e autor da obra

Caso desmarque novamente, as opções desaparecem (Figura 6).

Figura 6: Apenas opção de exportar áudio selecionada

Ao clicar em "HarmoniMIDI-me", tendo fornecido as informações corretamente, o programa começa a gerar a harmonia, mostrando uma barra de progresso (Figura 7).

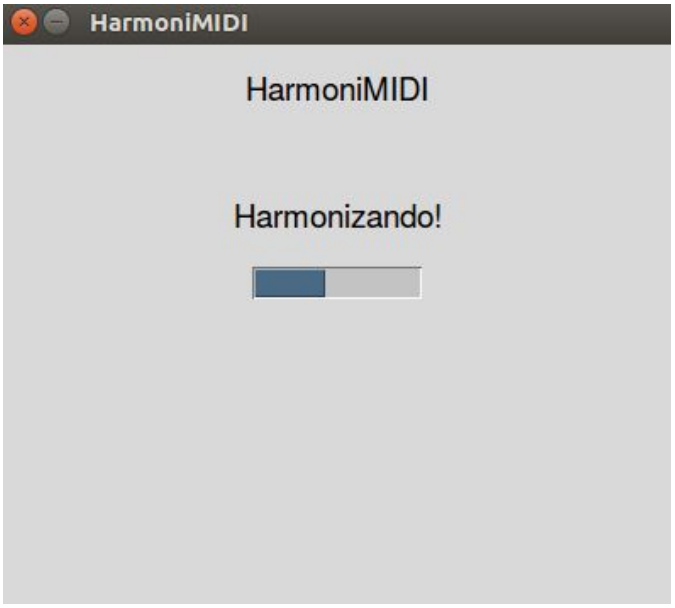

Figura 7: Algoritmo de harmonização em processamento

Terminada a execução do algoritmo de harmonização, é exibida uma mensagem de sucesso, e a opção para que o usuário possa ouvir o resultado (Figura 8).

Figura 8: Término de execução do algoritmo de harmonização

Após a execução do áudio, é apresentada a opção de rearmonização. Ao clicar no botão, o algoritmo executa de novo, e então o usuário pode ouvir a nova harmonização (Figura 9).

Figura 9: Opção para rearmonização

Após a execução do programa, caso o usuário tenha marcado as opções de exportação, são gerados arquivos de áudio e/ou partitura no diretório ./out (Figura 10).

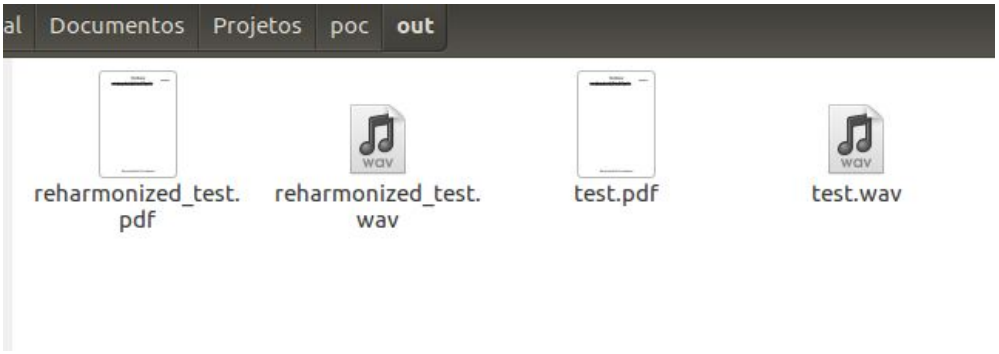

Figura 10: Arquivos gerados

A partitura gerada contém clave, tonalidade, ritmo, notas da melodia e os acordes gerados. Caso o usuário tenha fornecido, também contém nome da obra e do autor (Figura 11).

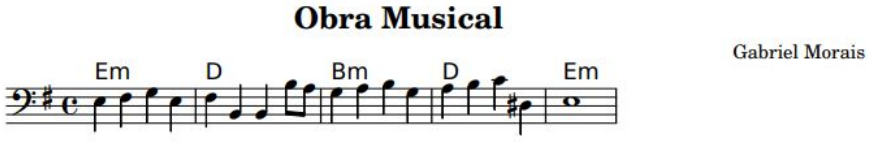

Figura 11: Partitura gerada

3.6 Dificuldades e problemas

O uso da linguagem Python foi, sem dúvida, uma dificuldade, mesmo que inicial, que se deve pelo fato de o autor ter tido pouco contato com esta anteriormente ao trabalho. Porém, Python se mostrou, ao desenvolver da aplicação, uma facilidade, e não impedimento, uma vez que possui sintaxe simples e muitas bibliotecas úteis para programação voltada para fins musicais.

Como música não é uma ciência exata, é difícil avaliar a eficiência de um algoritmo de composição. É claro, regras podem ser definidas, e parâmetros podem ser recebidos, mas, no final, continua sendo algo subjetivo. Como avaliar que uma melodia ou harmonia gerada são as melhores possíveis? As possibilidades são, quase literalmente, infinitas. Portanto, o meio de avaliar os resultados do programa é manual, e pode variar de pessoa para pessoa.

No início do desenvolvimento, foi decidido que seria usado como base o algoritmo de *Viterbi*, que resolve um problema do **Modelo Oculto de Markov** (uma alteração do modelo de Cadeias de Markov, onde, dados estados possíveis, tabela de probabilidades, tabela de

emissão e observações, é determinada a sequência de estados que mais provavelmente levaram às observações). No caso da aplicação, estados seriam os graus da escala (I, ii, iii, IV, V, vi e vii) e as observações as notas da melodia. Uma alteração no algoritmo de Viterbi poderia, em tese, gerar acordes para as melodias.

Porém, houve dificuldade em adaptar o algoritmo para as necessidades do problema, e certo receio de que o uso de um algoritmo "pronto" fizesse parecer que houve pouco trabalho por parte do autor. Sendo assim, foi decidido criar um algoritmo novo, que levasse em conta conceitos da **Teoria Musical**.

Um dos problemas mais marcantes foi com o arquivo de entrada em .midi. Foram testados alguns meios para obter esse arquivo, como, por exemplo, fazer uma gravação de melodia em formato MP3 e utilizar conversores *online* para gerar um .midi. Porém o programa, ao ler esse arquivo, recebia também muito "barulho": (muitas) notas a mais, pausas onde não havia e informação extra e desnecessária. Isso tudo inviabilizou esse processo.

Sendo assim, foi decidido que as melodias de entrada, usadas para teste, seriam geradas programaticamente, pelo próprio desenvolvedor. Não é a solução ideal, e, para o futuro, seria muito mais interessante descobrir uma forma de ler outros arquivos .midi que não foram gerados por programação, mas, para o escopo do trabalho, foi uma solução suficiente.

3.7 Código fonte

O código fonte do programa pode ser acessado em: *https://github.com/gabrissk/poc*.

4. Trabalhos futuros

Ao final do desenvolvimento da aplicação, é possível perceber várias oportunidades de complementações para esse.

A primeira oportunidade seria desenvolver uma versão *mobile*, já que, isso é, há alguns anos, a tendência para aplicações.

Seria interessante ser capaz de reconhecer vários tipos de arquivos .midi, além dos gerados programaticamente, como é na versão atual. Talvez até desenvolver uma aplicação extra que

geraria um .midi “limpo”, ou seja, sem informações inúteis e que atrapalham, a partir de um arquivo MP3.

Uma funcionalidade bastante interessante seria a de possibilitar ao usuário, além de poder fornecer um arquivo de entrada contendo a melodia, poder gravar diretamente no *app* sua melodia, seja usando algum instrumento ou até a voz ou mesmo assobiando.

Outra oportunidade de deixar o harmonizador mais robusto seria a de conseguir gerar acordes que estão fora do campo harmônico da tonalidade da melodia. Assim, poderiam ser geradas sequências de acordes mais diferenciadas, o que pode ser algo bastante positivo para fugir da monotonia de ter sempre acordes e sequências iguais.

Uma última mudança poderia ser a de gerar mais de um acorde por compasso. Isso requer uma análise muito mais profunda da melodia, dos compassos, do ritmo e dos tempos. Porém, tornaria o resultado mais “real”.

5. Conclusão

Como inicialmente era pretendido, foi desenvolvida uma aplicação que auxiliasse, de alguma forma, algum processo de composição musical. O processo escolhido foi o de harmonizar (escolher acordes) para uma melodia definida.

A pesquisa sobre Composição Algorítmica realizada na disciplina de Projeto Orientado em Computação (no primeiro semestre de 2019) foi muito importante, ajudando na definição do trabalho, bem como suas diretrizes e regras.

Para o desenvolvimento, foram aplicados conhecimentos adquiridos ao longo de todo o curso de Ciência da Computação, bem como as habilidades e conhecimentos musicais obtidos ao ter cursado disciplinas na Escola de Música. Na parte de computação, o desenvolvimento de aplicação, utilizando o que se foi aprendido em disciplinas como Algoritmos e Estruturas de Dados (I, II e III) e Engenharia de Software. Pela música, foi aplicado conhecimentos sobre Teoria Musical adquirido em disciplinas como Percepção Musical (I) e Fundamentos de Harmonia I.

A aplicação desenvolvida pode, além de ser um trabalho de conclusão de curso, um passo inicial para o desenvolvimento de outras ferramentas e aplicações voltadas para o universo da Música.

6. Referências

MAZUROWSKI, Łukasz. **Computer models for algorithmic music composition**. 2012. West Pomeranian University of Technology-Szczecin, Polônia, 2012.

EDWARDS, Michael. **Algorithmic Composition: Computational Thinking in Music**. 2011. Communications of the ACM, Julho de 2011, Vol. 54 No. 7, p. 58-67.

MORONI, Artemis et al. **Vox Populi: An Interactive Evolutionary System for Algorithmic Music Composition**. Leonardo Music Journal, Dezembro de 2000. Vol. 10, p. 49-54

FERNÁNDEZ, Jose David; VICO, Francisco. **AI Methods in Algorithmic Composition:A Comprehensive Survey**. 2013. Universidad de Málaga, Espanha, 2013.

PAPADOPOULOS, George; WIGGINS, Geraint. **AI Methods for Algorithmic Composition: A Survey, a Critical View and Future Prospects**. 2000. School of Artificial Intelligence, Division of Informatics, University of Edinburgh, Escócia, 2000.

ALPERN, Adam. **Techniques for Algorithm Composition of Music**. 1995. Hampshire College, Estados Unidos, 1995.

JACOB, Bruce. **Algorithm composition as a model of creativity**. 1996. University of Michigan, Estados Unidos, 1996.

ALBANO, Alexandre. **Composição Algorítmica: Histórico, exemplos e pesquisas recentes**. 2007. Universidade de São Paulo, Brasil, 2007.

FREITAS, Dr Alan. **Métodos de Composição Algorítmica**. Brasil, 2015.

CAMARATTA, Eduardo; FISCHER, Leonardo. **Composição de Música com o Uso de Algoritmos**. Universidade Federal do Rio Grande do Sul, Brasil.

PAZ, Fernando; MOREIRA, Benjamin. **Método de Monte Carlo para Composição Musical Algorítmica para Jogos**. 2012. Universidade do Vale do Itajaí, Brasil, 2012.

DE MATTOS, Fernando. **Harmonização de melodia**. Universidade Federal do Rio Grande do Sul, Brasil.

MARCHINI, Matheus. **Minstrel: Composição Algorítmica para Leigos em Música**. Universidade Federal do Rio Grande do Sul, Brasil, 2016.

KALIAKATSOS–PAPAKOSTAS, Maximos; CAMBOUROPOULOS, Emilios. **Probabilistic harmonization with fixed intermediate chord constraints**. Aristotle University of Thessaloniki, 2014, Grécia.

MAKRIS, Dimos; KARYDIS, Ioannis; SIOUTAS, Spyros. **Automatic Melodic Harmonization: An overview, challenges and future directions**. Ionian University, 2016, Grécia.

AI Project: Harmonizing Pop Melodies using Hidden Markov Models. Disponível em <https://luckytoilet.wordpress.com/tag/harmonization/>. Acesso em 27/09/2019.

DUREY, Adriane; CLEMENTS, Mark. **Melody Spotting Using Hidden Markov Models**. Georgia Institute of Technology, 2001, Estados Unidos

Cadência. Disponível em <https://www.descomplicandoamusica.com/cadencia/>. Acesso em 15/10/2019.