UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE CIÊNCIA DA COMPUTAÇÃO

MATHEUS MARCHINI

Minstrel: Composição Algorítmica para Leigos em Música

Monografia apresentada como requisito parcial para a obtenção do grau de Bacharel em Ciência da Computação.

Orientador: Prof. Dr. Marcelo Soares Pimenta

Porto Alegre
2016

"Sou biólogo e viajo muito pela savana do meu país. Nessas regiões encontro gente que não sabe ler livros. Mas que sabe ler o seu mundo. Nesse universo de outros saberes, sou eu o analfabeto."

COUTO, M. **E se Obama fosse africano? e outras interinvenções**
Editorial Caminho SA, 2009

RESUMO

O presente trabalho tem por objetivo o desenvolvimento e avaliação de uma aplicação de composição algorítmica para leigos em música. Foi escolhido um algoritmo de composição previamente avaliado, sobre o qual foram feitas alterações para que a sua utilização fosse possível por leigos em música. Uma interface *web* para a aplicação foi criada utilizando tecnologias recentes e boas práticas de desenvolvimento. A avaliação por parte dos usuários sobre a aplicação foram positivas, provando que é possível criar aplicações de composição algorítmica que possam ser utilizados por leigos em música.

Palavras-chave: Composição Algorítmica. Performance Expressiva. Aplicações web. Usabilidade.

Minstrel: Algorithmic composition for laypersons in music

ABSTRACT

The goal of this work is the development and evaluation of an algorithmic composition application for laypersons in music. A previously evaluated composition algorithm was chosen, and a some changes were made to enable it to be used by laypersons in music. A web interface was created for this application by using current technologies and development best practices. Feedback given by the participants was positive, proving that it's possible to develop an algorithmic composition application for laypersons in music.

Keywords: Algorithmic composition. Expressive performance. Web applications. Usability

LISTA DE FIGURAS

LISTA DE GRÁFICOS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

ACOPM	Algoritmic Composition of Popular Music
BPM	Batidas Por Minutos
CMERS	Computational Music Emotion Rule System
DM	Director Musices
GM	General MIDI
POO	Programação Orientada a Objetos
SF	SoundFont

SUMÁRIO

1 INTRODUÇÃO

A composição algorítmica tem suas origens muito antes do surgimento dos primeiros computadores. As primeiras técnicas de composição algorítmica registradas datam de aproximadamente 1025, quando Guido de Arezzo – mais conhecido por inventar o *solfeggio* e por suas contribuições para notação musical – criou uma técnica capaz de transformar textos em peças musicais (NIERHAUS 2009). Desde então, diversos outros compositores famosos criaram técnicas para composição de forma algorítmica, como a técnica dos doze tons de Shrönberg e os jogos de dados de Mozart e Haydn (NIERHAUS 2009).

Com o advento e evolução dos computadores no século XX, começaram a surgir trabalhos de composição algorítmica gerados completamente por computadores, como é o caso dos trabalhos produzidor por Lejaren Hiller e Leonard Isaacson entre 1955 e 1956, conhecidos como “Illiac Suit” por terem sido criados utilizando um computador ILLIAC na Universidade de Illinois, sendo o primeiro conjunto de peças geradas por computadores (NIERHAUS 2009). Diversos outros trabalhos foram desenvolvidos durante a segunda metade do século XX, principalmente dentro de universidades.

Apesar da popularização dos computadores e da ascensão da Internet no final do século XX e início do século XIX, poucos sistemas de composição algorítmica foram criados para utilização por uma grande quantidade de usuários (BOZHANOV 2014) ou para leigos em música. Por exemplo, o *Computoser* (BOZHANOV 2014) permite a utilização em larga escala, e oferece alguns parâmetros de simples compreensão, porém é necessário conhecimento musical por parte dos usuários para utilizar a maioria dos parâmetros disponíveis de forma efetiva. O trabalho proposto por Elowsson (2012) permite a composição de músicas populares, mas possui parâmetros complexos, além de não reproduzir a música gerada.

Um sistema de composição algorítmica que possa ser utilizado em larga escala por leigos em música possui diversas aplicações. Por exemplo, um sistema com essas características poderia criar protótipos musicais que podem ser utilizados durante o desenvolvimento de *video games*. Vale lembrar que um sistema assim não tem por objetivo substituir o papel do compositor.

Esse trabalho tem por objetivo o desenvolvimento de um sistema de composição algorítmica que possa ser utilizado por usuários leigos. Esse sistema permitirá que os usuários criem composições diferentes a partir da alteração dos parâmetros de entrada, gerando composições em tempo real. Para isso, será necessário a implementação de um algoritmo de

composição, bem como a elaboração e desenvolvimento de uma aplicação web, que possa ser utilizada através de diversas plataformas.

O trabalho está dividido em três partes. Na primeira, será feita uma breve revisão bibliográfica sobre composição algorítmica e performance expressiva, seguido pela apresentação dos trabalhos relacionados. A segunda parte irá abordar a implementação do sistema, a definição dos parâmetros, a elaboração da interface e a aplicação dos testes com usuários. A última parte apresentará as conclusões do trabalho e possibilidades para trabalhos futuros.

2 ESTADO DA ARTE

Essa seção está divida em duas partes. Na primeira é feita uma revisão bibliográfica dos trabalhos sobre composição algorítmica e performance expressiva pesquisados durante a realização do trabalho. Na segunda parte, os trabalhos selecionados como relevantes serão apresentados com mais detalhes, fornecendo as fundamentações teóricas para o desenvolvimento do sistema.

2.1 Revisão Bibliográfica

A escolha de trabalhos apropriados é importante para a criação de um sistema capaz de proporcionar uma experiência satisfatória para os usuários. Por esse motivo, essa seção trazer uma revisão bibliográfica de trabalhos existentes nas áreas de Composição Algoritmia e Performance Expressiva, ambas áreas importantes no desenvolvimento do sistema proposto.

2.1.1 Composição Algorítmica

De acordo com Nierhaus (2009), a composição algorítmica pode ser definida como a "composição utilizando métodos formais". Dentro da área da computação, essa técnica é geralmente associada à composição automatizada, realizada por um computador, com ou sem a intervenção de um usuário externo.

Diversos trabalhos foram desenvolvidos nessa área nas últimas décadas, sendo possível encontrar a utilização de técnicas que vão desde Cadeias de Markov simples até sistemas com regras complexas ou algoritmos evolutivos. Alguns desses trabalhos ficaram bem conhecidos na área, como é o exemplo do "*Experiments in Musical Intelligence*" (EMI), desenvolvido por David Cope, que ficou famoso por produzir um sistema capaz de imitar de forma satisfatória o estilo de compositores clássicos (NIERHAUS 2009). Outro trabalho que produz bons resultados é o GenJam, um sistema desenvolvido por John A. Biles, que utiliza um algoritmo evolutivo capaz de criar improviso em tempo real para peças de Jazz (MIRANDA 2007).

Alguns trabalhos menos famosos também foram capazes de obter bons resultados. Trabalhos mais recentes como o *Computoser* (BOZHANOV 2014) e o ACOPM

(ELOWSSON 2012). O *Computoser* foi criado com o intuito de ser um sistema *web* capaz de compor peças satisfatórias para uma grande quantidade de usuários sem a necessidade de supervisão durante a composição, enquanto que o ACOPM é um algoritmo produzido com o objetivo de compor musicas populares. Ambos os algoritmos conseguiram altas taxas de aprovação nas avaliações feitas com ouvintes.

Nem todos os algoritmos de composição buscam imitar gêneros ou estilos musicais conhecidos. Temos o exemplo do GenDash, um algoritmo evolutivo com o objetivo de criação de peças completamente novas a partir de uma população inicial (MIRANDA 2007).

Fazendo uma análise comparativa dos algoritmos apresentados até o momento, podemos ver que cada um deles foi projetado para atingir objetivos diferentes. Para a utilização nesse trabalho, é preciso escolher um algoritmo que (1) possibilite a criação peças musicais variadas e que (2) aceite uma grande quantidade de parâmetros de entrada.

Levando em consideração o primeiro requisito, podemos descartar algoritmos como o EMI e o GenJam, pois são algoritmos criados para produzir peças de um determinado gênero ou estilo. Apesar de ser capaz de compor músicas variadas, o GenDash será desconsiderado, pois seu parâmetro é uma população inicial composta por trechos musicais, e o desenvolvimento dessas populações necessita de um conhecimento musical ausente no autor.

Dentre os trabalhos restantes, o *Computoser* possui 9 tipos de parâmetros de entrada que permitem 43.680 combinações diferentes que afetam a composição em alto nível (escala, instrumentos, velocidade, etc.). Esses números foram calculados através da contagem dos parâmetros presentes na interface do *Computoser* disponível em Computoser (2016). Através das imagens disponíveis em Elowsson (2012), é possível identificar 27 tipos de parâmetros diferentes, capazes de ajustes tanto em alto nível (escala, velocidade) quanto num nível mais baixo (harmonia entre acordes e nota, probabilidades de acordes, do tamanho das frases, etc.). Não é possível calcular quantas combinações possíveis o ACOPM aceita, porém todos os parâmetros vistos na interface possuem pelo menos 5 opções diferentes, sendo que a maioria possui mais de 10, enquanto que o *Computoser* possui um parâmetro com 12 opções, um com 7 e um com 5, e os outros todos possuem menos de 5 opções. Logo, o ACOPM possui uma variedade muito maior de parâmetros, além de permitir ajustes mais sofisticados desses parâmetros para controlar melhor o resultado da composição gerada.

2.1.2

2.1.3 Performance Expressiva

Com o intuito de fazer composições executadas pelo computador soarem mais realistas, diversos algoritmos de Performance Expressiva (PE) foram desenvolvidos nos últimos anos. Esses algoritmos podem mudar a percepção da música executada, fazendo ela soar mais parecida com uma música executada por um instrumentista e até alterar a emoção transmitida pela sua execução. Para esse trabalho, que propõe um sistema que componha *e* execute uma peça musical, um algoritmo de performance expressiva é necessário por dois motivos: 1) fazer com que a música gerada pareça menos artificial e 2) abrir a possibilidade para novos parâmetros que o usuário possa selecionar.

Em Kirke (2009), foi apresentada uma revisão bibliográfica de diversos trabalhos na área de performance expressiva, expondo seus pontos fortes e fracos. Kirke (2009) dividiu os algoritmos em diversas categorias, sendo a categoria *nonlearning* aquela que apresenta a maior quantidade de algoritmos. As outras categorias utilizam técnicas de aprendizagem para melhorar os seus resultados, porém elas permitem menor controle dos resultados por parâmetros, já que são dependentes de execuções anteriores (Miranda 2009). Por esse motivo, o foco será nos algoritmos *nonlearning*. Com o intuito de adicionar o parâmetro "Emoção" no sistema, serão considerados apenas os trabalhos capazes de manipular essa característica das músicas.

A partir dessas restrições, dois algoritmos dentre os apresentados em Kirke (2009) se destacam: o Director Musices (DM) e o Computational Music Emotion Rule System (CMERS). Ambos são algoritmos *nonlearning*, possuem um conjunto diversificado de regras e possibilitam a alteração da emoção transmitida pela música.

O DM engloba um conjunto de regras, que foram sendo adicionadas aos poucos desde a sua criação em 1982. O sistema possui várias versões e influenciou diversos outros algoritmos. As regras, se configuradas de forma apropriada, podem mudar a emoção transmitida pela música (Kirke 2009).

Já o CMERS possui um foco maior na emoção transmitida pela música quando reproduzida. Ele também segue uma arquitetura de regras, onde cada uma das regras realiza alguma alteração na música. Algumas dessas regras são inspiradas no DM. Os autores afirmam que, nos testes realizados, o CMERS obteve um resultado melhor do que o DM para expressar emoções (LIVINGSTONE 2010).

A partir das análises de Kirke (2009) e dos resultados apresentados por Livingstone (2010) para os testes comparativos entre o CMERS e o DM, o algoritmo de performance expressiva escolhido para utilização nesse trabalho será o CMERS.

2.2 Trabalhos Relacionados

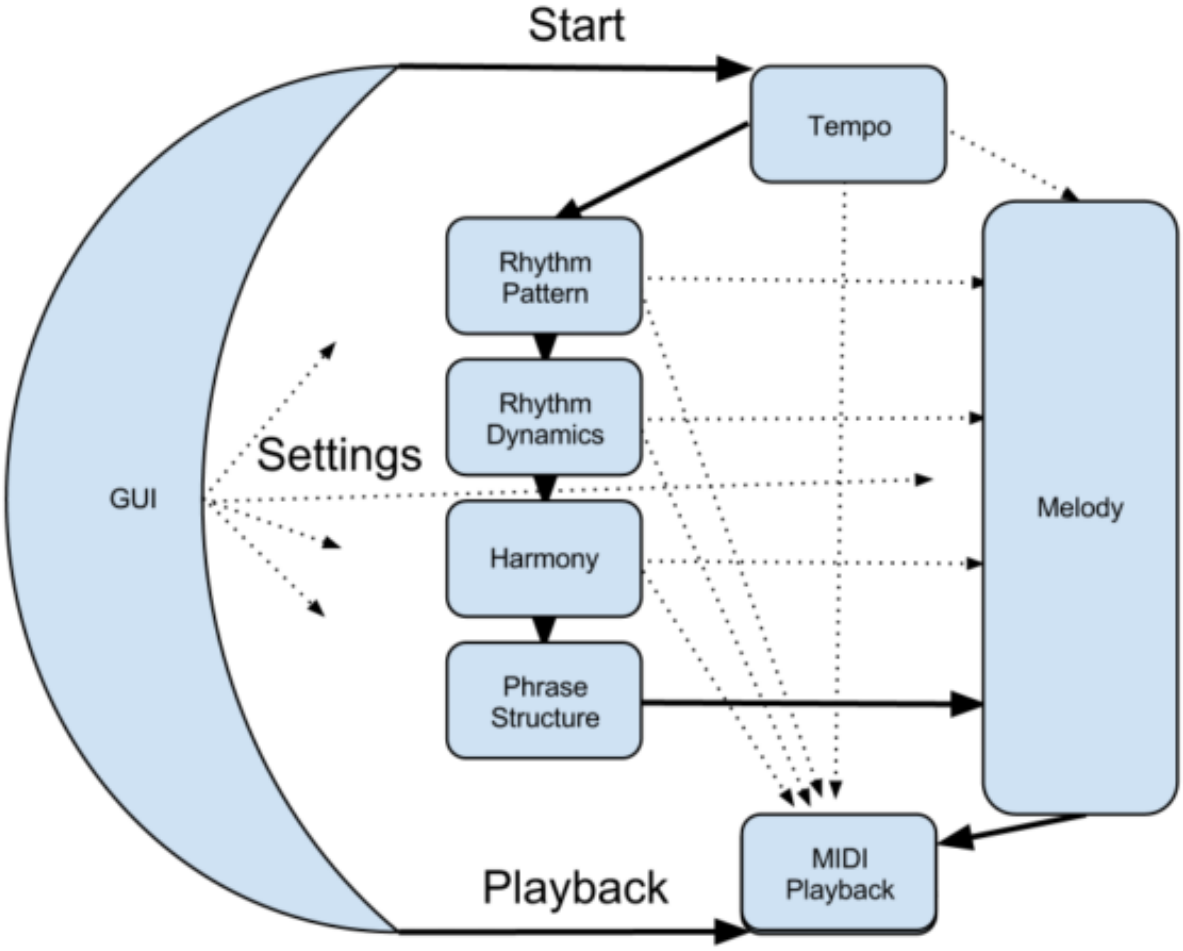
A partir da revisão bibliográfica, foi possível escolher os algoritmos que serão utilizados na implementação do sistema, bem como as tecnologias que serão utilizadas. O trabalho de (ELOWSSON 2012) foi escolhido como algoritmo de composição, pela sua diversidade de parâmetros, diversidade nos resultados e por ser voltado para a composição de músicas populares. Para complementar o sistema, uma parte das regras criadas no trabalho apresentado em (LIVINGSTONE 2010) foi utilizado. Além dos trabalhos acadêmicos citados, também foram utilizadas diversas ferramentas e tecnologias para a implementação do sistema, que serão vistas na Seção 3.1.

2.2.1 Algorithmic Composition of Popular Music

O trabalho de Elowsson (2012) tem como objetivo melhor entender como o processo de composição de músicas populares pode ser formalizado, bem como melhor entender músicas populares em geral. Para atingir esse objetivo, os autores do trabalho desenvolveram um algoritmo de composição de músicas populares. A estrutura desse algoritmo segue os seguintes passos: criação de "uma fundamentação rítmica (bateria), uma progressão de acordes, uma estrutura de frases e, por fim, a melodia" (ELOWSSON 2012). Essa estrutura pode ser observada na Figura 2.1.

O algoritmo foi desenvolvido através de análises estatísticas de dados extraídos da *Essen Folksong Collection*, usando como preceito o fato observado por Huron (2006) de que "humanos aparentemente possuem um entendimento estatístico intrínseco de música, onde eventos previsíveis são experienciados como prazerosos" (ELOWSSON 2012). O conceito de *Global Joint Accent Structure* foi introduzido no trabalho, com o objetivo de melhor entender como as relações entre melodia e ritmo podem deixar eventos futuros mais previsíveis para o ouvinte (ELOWSSON 2012).

Figura 2.1 - Arquitetura do CMERS



2.2.1.1 Tempo

"O Tempo é criado a partir de uma distribuição normal onde o usuário pode definir uma média preferida para a distribuição." (ELOWSSON 2012)

2.2.1.2 Ritmo

"O programa cria um ritmo básico de bateria usando o Tempo como parâmetro de entrada" (ELOWSSON 2012). O ritmo é composto por *kicks*, *hihats*, *snares* e *rides*. O *kick* sempre aparece na primeira batida, podendo aparecer também nas demais batidas. Uma distribuição normal é utilizada para determinar em quantas batidas – além da primeira – o *kick* será utilizado, e a posição desses *kicks* é determinado utilizando uma probabilidade diferente para cada posição (ELOWSSON 2012).

"O *hihat* pode tocar notas de 1/4, 1/8 ou 1/16 enquanto que o *ride* pode tocar notas de 1/4 ou 1/8. As probabilidades de cada duração dependem do Tempo, onde um tempo maior aumenta a probabilidade das notas mais longas" (ELOWSSON 2012). Como não é mencionado quais notas o *snare* pode tocar, foi assumido que ele poderá tocar as mesmas notas do *hihat*.

Para mais detalhes sobre a criação de ritmos, ver Elowsson (2012).

2.2.1.3 Harmonia

O ACOPM permite a criação de sequências de quatro ou oito acordes, e utiliza cadeias de Markov para essa função (ELOWSSON 2012). Para as sequências de quatro acordes, são utilizadas cadeias de 3ª ordem, enquanto que para sequências de oito acordes, são utilizadas cadeias de 2ª ordem (ELOWSSON 2012). Apenas os acordes mais comuns são usados, deixando de lado o 7º acorde (ELOWSSON 2012).

Para mais detalhes sobre a criação da harmonia, ver Elowsson (2012).

2.2.1.4 Frases

As frases influenciam em diversos aspectos da composição. Um desses aspectos são as repetições. O algoritmo utiliza diversas técnicas para realizar repetição nos mais variados níveis, permitindo a repetição de frases inteiras ou apenas de trechos. Ele também permite que a melodia repita apenas o ritmo das notas ou a sequência de notas (ou ambos). Além das repetições, uma frase determina quantas notas e quanto silêncio aparecerão dentro da frase.

Para uma explicação mais detalhada sobre a geração das frases e como elas influenciam a composição, ver (ELOWSSON 2012)

2.2.1.5 Melodia

A geração da melodia é a etapa mais complexa do algoritmo. Essa etapa utiliza um conjunto de dez regras que são aplicadas a um grupo de possíveis notas, atribuindo

pontuações para cada uma dessas notas, que posteriormente serão utilizadas para determinar a probabilidade de ocorrência delas. Nas palavras de Elowsson (2012):

> "A melodia é gerada de forma iterativa, criando uma nota de cada vez. Primeiro, uma série de notas com duração e tom são sugeridas. Para cada uma dessas notas uma pontuação é calculada multiplicando-se diferentes sub-pontuações. Dez regras provêem essas sub-pontuações (...). A nota final é selecionada pela probabilidade distribuída criada por todas essas pontuações" (ELOWSSON 2012).

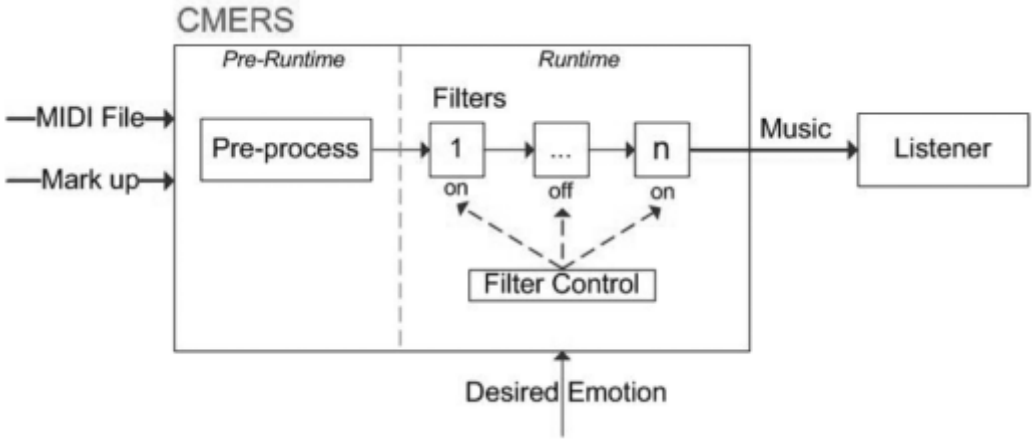
O algoritmo aceita um parâmetro opcional *p*, que é uma potência ao qual serão elevadas as pontuações de todas as notas sugeridas. Caso o valor de *p* for inferior a 1, a diferença de probabilidade entre as notas diminuí, e caso for superior a diferença aumenta. Esse parâmetro influencia a aleatoriedade percebida na melodia (ELOWSSON 2012).

Para mais detalhes sobre as regras utilizadas no cálculo da pontuação das notas, ver (ELOWSSON 2012).

2.2.2 Computational Music Emotion Rule System

O objetivo do trabalho de Livingstone (2010) é explorar a capacidade de sistemas computacionais para controlar a emoção percebida através de peças musicais, e para isso foi desenvolvido um sistema em tempo real que modifica aspectos da música em nível de partitura e em nível de execução. As avaliações do sistema mostraram uma taxa de sucesso na alteração da emoção das músicas superior à 70%.

Figura 2.2 – Arquitetura de alto nível do CMERS



Fonte: Livingstone, 2010

"O CMERS foi projetado como um sistema baseado em filtros para alteração de MIDI em tempo real" (LIVINGSTONE 2010). Apesar da nomenclatura utilizada no trabalho de Livingstone (2010) denominar os componentes que alteram a música de "filtros", esse nome não é apropriado filtros são "dispositivos que permitem apenas alguns tipos de luz e som passem por ele" (OXFORD 2010), quando na verdade esses componentes processam a música, alterando a informação ao invés de removê-la. Por esse motivo, a nomenclatura de "filtro" será substituída nesse trabalho por "processador" (*processors*).

Quatorze regras foram implementadas como processadores no CMERS, sendo que oito delas são classificadas como Regras de Emoção Musical e as outras seis são classificadas como Regras de Aspectos Expressivos. Algumas das regras relevantes para o presente trabalho podem ser vistas na Tabela 2.1.

Tabela 2.1 – Regras do CMERS relevantes para o trabalho

Emoção	**Tipo de Regra**	**Variação**	**Detalhes**
1 Feliz	Tempo	Aumentar	10 BPM
	Modo	Maior	
	Sonoridade	Aumentar	5 dB
	Altura do Tom	Subir	4 semitons
2 Irritado	Tempo	Aumentar	10 BPM
	Modo	Menor	
	Sonoridade	Aumentar	7 dB
	Altura do Tom	Sem Alterações	
3 Triste	Tempo	Diminuir	15 BPM
	Modo	Menor	
	Sonoridade	Diminuir	5 dB
	Altura do Tom	Descer	4 semitons
4 Tenro	Tempo	Diminuir	20 BPM
	Modo	Maior	
	Sonoridade	Diminuir	7 dB
	Altura do Tom	Subir	4 semitons

Fonte: Livingstone, 2010

3 MINSTREL – PROJETO E IMPLEMENTAÇÃO

O objetivo dessa seção é a apresentação do trabalho proposto, passando pela sua concepção, desenvolvimento e validação, e está dividida em cinco partes. Na primeira, será apresentado um panorama geral da arquitetura do sistema, que auxiliará na compreensão das sessões posteriores. Na segunda parte, será apresentada a implementação do algoritmo de composição escolhido, destacando os pontos em que a implementação e o algoritmo original diferem. A terceira parte falará sobre a definição dos parâmetros e sobre como eles foram implementados. A elaboração da interface, com o objetivo de facilitar a utilização para leigos em música, será abordada na quarta parte. O procedimento de validação com usuários, bem como os resultados obtidos nessa validação, será visto na última parte.

O trabalho proposto tem por objetivo a criação de um sistema de composição algorítmica que possa ser utilizado por usuários leigos em música de forma satisfatória. Para atingir o objetivo do trabalho, foi necessário escolher o algoritmo de composição a ser utilizado, definir os parâmetros que estarão à disposição do usuário – de tal forma que eles não necessitem de conhecimento musical prévio para a sua utilização -, elaboração da interface do sistema, que deve auxiliar na fácil interpretação dos parâmetros e possuir uma usabilidade simples.

Após finalizar o processo de desenvolvimento, foram realizados testes de validação com usuários reais. 51 pessoas testaram o sistema e responderam a um questionário sobre a sua utilização. O conhecimento musical dos participantes varia desde pessoas sem nenhum conhecimento musical até pessoas até pessoas experientes. A avaliação dos resultados desses testes permite determinar se foi possível atingir o objetivo do trabalho, bem como identificar pontos que podem ser melhorados em trabalhos futuros.

3.1 Arquitetura

A escolha de tecnologias atuais, amplamente utilizadas e confiáveis, além de utilizar padrões de desenvolvimento e de interface já consolidados, permite agilizar o desenvolvimento e obter um melhor resultado final. Além disso, a escolha de uma arquitetura que permita que o sistema seja utilizado em várias plataformas pode aumentar o acesso de usuários (MILETO 2007). As tecnologias escolhidas permitiram o desenvolvimento da aplicação visando a sua usabilidade e performance na composição e execução das composições.

Por essas razões, o sistema foi implementado utilizando tecnologias *web*. O algoritmo de composição foi implementado em Python (2016), com auxilio da biblioteca Mingus, uma biblioteca avançada para trabalhar com teoria e notação musical, com suporte a arquivos MIDI (MINGUS 2016). O componente do sistema que implementa o algoritmo de composição foi denominado *Popgen*. A música composta é salva em um arquivo MIDI, que é posteriormente renderizado utilizando a ferramenta FluidSynth (2016), com a SoundFont (SF) (MUSESCORE 2016) Arachno (ARACHNO 2016) para banco de instrumentos. A renderização é feita em tempo real, melhorando a percepção do usuário em relação a velocidade de resposta do sistema. O resultado da renderização é um *stream* de dados no formato WAV, que é codificado para MP3 utilizando a ferramenta *avconv*, presente no pacote *libav-tools* (LIBAV 2016).

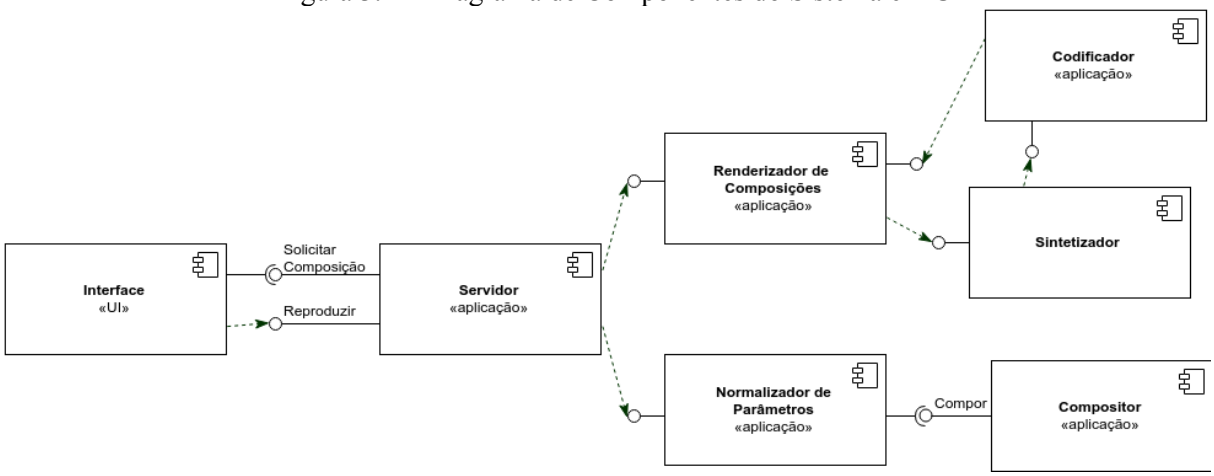
O servidor também é implementado em Python, o que possibilita uma melhor comunicação entre a implementação do algoritmo e o servidor. O *framework* Django (2016) foi escolhido para a implementação do servidor, por auxiliar no rápido desenvolvimento de sistemas e possuir a maior base de usuários dentro da comunidade de desenvolvedores Python para *web*. Para auxiliar no desenvolvimento da interface, foi utilizada a biblioteca Twitter Bootstrap 3, em conjunto com a biblioteca de *front-end* AngularJS, que permite a criação de interfaces web sem a necessidade de atualizar a página, passando uma sensação de fluidez na utilização do sistema.

Na Figura 3.1, temos um diagrama de componentes que representa a arquitetura do sistema, com as comunicações entre as diferentes partes do mesmo. Após o usuário solicitar uma música através da Interface, o Servidor processa a requisição e envia os parâmetros escolhidos para um Normalizador de Parâmetros. Esse Normalizador é responsável por converter os parâmetros disponíveis pelo sistema para parâmetros válidos do *Popgen*, e enviar esses parâmetros para realizar a composição. Essa composição ficará salva no servidor, e o caminho para reproduzi-la será enviado para a interface.

Quando a interface recebe o caminho para a composição, ela realiza uma nova requisição - sem a necessidade de intervenção do usuário - para que o servidor renderize a composição em um arquivo reproduzível. O servidor processa essa nova requisição, identifica a composição e a encaminha para um Renderizador de Composições. O Renderizador possui duas funções: sintetizar a música e codificá-la em um arquivo compacto e reproduzível. Essa renderização é feita em tempo real, e o servidor envia - em forma de *streaming* de dados - o arquivo reproduzível para a interface, mesmo que o renderizador não tenha finalizado a sua tarefa. Como o processo de sintetização e codificação, mesmo sendo demorado, leva menos

tempo do que a duração das músicas, o usuário tem a sensação de que a composição é realizada em poucos segundos.

Figura 3.1 – Diagrama de Componentes do Sistema em UML2



Fonte: Diagrama do sistema desenhado pelo autor (2016)

3.2 Popgen – Implementação do Algoritmo de Composição

Mesmo que o foco desse trabalho não seja a estética das músicas compostas, a escolha de um bom algoritmo de composição, que produza resultados satisfatórios, permita a composição de músicas variadas e possua diversos parâmetros de entrada, é essencial para o desenvolvimento do trabalho. O algoritmo produzir resultados satisfatórios e uma boa diversidade de composições é importante para que o usuário consiga ter uma visão mais clara da influência de cada parâmetro na composição. Se o algoritmo produzir resultados insatisfatórios, ou se as músicas compostas por ele forem muito parecidas, o usuário pode não identificar relação entre a alteração dos parâmetros e a diferença entre as músicas compostas. Já a diversidade de parâmetros de entrada ajuda na criação de novos parâmetros de mais alto nível.

Levando esses fatores em consideração, o trabalho de Elowsson (2012) foi escolhido como algoritmo de composição utilizado no trabalho. Nessa seção serão apresentados alguns detalhes da sua implementação, como tecnologias utilizadas e diferenças entre a implementação feita e o algoritmo original.

3.2.1 Estrutura do Algoritmo

A implementação do algoritmo de composição foi realizada utilizando a linguagem de programação Python. Os motivos que levaram a escolha dessa linguagem foram a melhor comunicação com o servidor do sistema, ampla utilização da linguagem e familiaridade do autor. Para auxiliar na implementação, uma versão modificada do pacote *Mingus* foi utilizada. As modificações foram necessárias para corrigir alguns *bugs* existentes no pacote, bem como adicionar algumas funcionalidades, dentre elas a geração de escalas pentatônicas e alteração do volume das notas nas composições. O pacote modificado pode ser encontrado em Marchini (2016).

Foram utilizados conceitos de Programação Orientada a Objetos, separando cada etapa de composição - Ritmo, Harmonia, Estrutura de Frases e Melodia - em diferentes classes. Uma classe responsável por unificar o resultado de cada etapa foi criada e denominada *Compositor*. Ela instancia as classes que representam cada etapa da composição com base nos parâmetros recebidos, que podem ser passados como um arquivo YAML (YAML 2016) ou como objetos (nesse caso, não é necessário a instanciação). O sistema começa realizando os cálculos para determinar o ritmo da música. Após concluído, o ritmo é passado como parâmetro para a criação da harmonia da música, onde são escolhidos os acordes da música e em quais batidas eles serão tocados (de acordo com o ritmo). A harmonia gera duas faixas no arquivo MIDI, a faixa de acordes e a do instrumento base. A estrutura de frases é passada por parâmetro para o *Compositor*, e não é feito nenhum cálculo para modificá-la. Por fim, a criação da melodia recebe por parâmetro o ritmo, a harmonia e a estrutura de frases, e aplica as regras definidas em Elowsson (2012) para criar uma composição que seja adequada aos parâmetros envolvidos. Após a conclusão da composição, é possível salvá-la em um arquivo MIDI.

A implementação difere do algoritmo original em alguns pontos. Primeiro, o ACOPM foi criado para geração da composição, mas não para a sua execução – que fica a cargo de um instrumentista. Como o sistema Minstrel permite diversos instrumentos, e os instrumentos GM não conseguem tocar qualquer nota – ficando com uma sonoridade estranha quando sai da faixa confortável para aquele instrumento -, foi necessário adicionar um parâmetro Instrumentos no Popgen, para definir previamente quais instrumentos podem ser utilizados e suas faixas preferíveis de execução. Por conta dessa definição das faixas confortáveis, foi necessário modificar os parâmetros de faixa preferida e máxima da melodia. No algoritmo original, esses parâmetros são as notas que delimitam as faixas, enquanto que no Popgen esses

parâmetros são porcentagens que indicam a partir de qual e até qual nota a melodia pode utilizar. Outras diferenças serão abordadas na Seção 3.2.2.

Por ser um algoritmo complexo e com vários cálculos, seu tempo de execução pode ficar acima do tolerável para o usuário final. Algumas otimizações foram necessárias para que o algoritmo ficasse dentro do tempo aceitável. A utilização de *caches* para tarefas repetidas permitiu uma grande redução do tempo de execução do algoritmo. Também foi necessário substituir a utilização da estrutura de dados *Decimal*, nativa e implementada em Python, por uma versão otimizada, denominada *m3-cdecimal*, implementada em C. Essa implementação otimizada, que pode ser utilizada no lugar do Decimal nativo do Python sem alterações no código, pode ser até 80 vezes mais rápida quando utilizada em programas que realizem uma quantidade significativa de cálculos (MPDECIMAL 2016).

Além das otimizações feitas, outras medidas podem ser tomadas em trabalhos futuros para melhorar o desempenho do algoritmo. Primeiro, algumas das etapas onde *caches* foram utilizadas podem ter seus resultados pré-calculados e salvos em um banco de dados, precisando apenas carregar os valores necessários para as *caches* do sistema. A estrutura do programa pode ser alterada para que a composição seja enviada em tempo real para o FluidSynth (2016), seja utilizando *pipelines* ou através do protocolo MIDI, possibilitando que a música seja executada enquanto a composição é feita.

3.2.2 Etapas da Composição

Os algoritmos, cálculos e parâmetros utilizados para a criação do *Popgen* serão apresentados nas próximas seções.

3.2.2.1 Ritmo

O algoritmo que cria o ritmo da composição foi feito utilizando como base as análises estatísticas apresentadas por Elowsson (2012) e na Seção 2.2.1.2. A classe *Rhythm* é responsável pela realização dos cálculos de criação de Ritmo, e a declaração dessa classe pode ser vista nas Figuras 3.2, 3.3 e 3.4.

O método *calculate_notes*, que pode ser visto na Figura 3.2, é responsável por distribuir batidas do ritmo dentro de uma barra. Ele recebe por parâmetro a quantidade de batidas, as durações possíveis – uma lista de tuplas no formato “(duração, probabilidade)” – e

opcionalmente uma lista com batidas já existentes na barra. A posição das batidas é calculada individualmente, e para cada batida é feito um sorteio da sua duração, levando em consideração as probabilidades de cada duração. As posições possíveis para cada batida são calculadas levando em consideração a duração sorteada e evitando a sobreposição com batidas já determinadas. Por fim, é a posição da batida é sorteada levando em consideração as posições possíveis. Esse procedimento se repete por Q vezes, sendo Q a quantidade passada por parâmetro para o método.

Quatro tipos de batidas são utilizadas no ritmo: *kicks*, *hihats*, *snares* e *rides*. As posições e durações delas são determinadas individualmente pelos métodos *generate_kicks*, *generate_snares*, *generate_hihats* e *generate_rides*, que utilizam o método *calculate_notes* para determinar as posições e durações de cada tipo de batida. Esses métodos podem ser vistos na Figura 3.3.

A quantidade de *kicks* é determinado pelo método *number_of_kicks*, que utiliza uma distribuição normal para determinar essa quantidade. A quantidade dos outros tipos é determinada por uma probabilidade pré determinada, como pode ser visto na Figura 3.3.

Figura 3.2 - Construtor da classe Rhythm e métodos number of kicks e calculate notes

```
45  class Rhythm(object):
46
47      def __init__(self, tempo, instrument=AcousticPercussion):
48          self.tempo = tempo
49          self.instrument = instrument
50
51      def number_of_kicks(self):
52          kicks_mean = 1. + ((abs((self.tempo - 160.) / 40.)))
53          kicks = int(round(random.gauss(kicks_mean, .45)))
54
55          return kicks
56
57      def calculate_notes(self, quantity, possible_durations, notes=[]):
58          u""" Calcula a posição das notas de percussão dentr de uma barra.
59          O parâmetro `quantity` determina a quantidade de notas,
60          `possible_durations` as durações possíveis para as notas,
61          e o parâmetro opcional `notes` determina notas já presentes
62          na barra.
63          """
64          duration_chooser = WeightedChoice(*possible_durations)
65
66          # Determina a posição e duração das notas
67          for i in range(quantity):
68              note_duration = duration_chooser.choose()
69              note_step = 16 / note_duration
70
71              possible_positions = []
72              # Itera sobre as posições possíveis, baseado na duração da nota
73              for i in range(0, 16, note_step):
74                  # Verifica se a nota atual não se sobrepõe as notas
75                  # já escolhidas
76                  if not any([i <= note[0] < i + note_step for note in notes]):
77                      possible_positions.append(i)
78
79              # Escolhe uma das notas possíveis
80              chosen_position = random.choice(possible_positions)
81              notes.append((chosen_position, note_duration))
82
83              # Ordena as notas de acordo com as suas posições
84              notes = sorted(notes, key=itemgetter(0))
85          return notes
```

Fonte: Imagem do código do Popgen

Figura 3.3 - Métodos que calculam as posições e durações de cada tipo de batida

```
87      def generate_kicks(self):
88          u""" Gera os `kicks` que aparecerção na composição """
89          possible_durations = [(4, 0.60), (8, 0.40)]
90
91          return (self.instrument.kick, self.calculate_notes(
92              quantity=self.number_of_kicks()-1,
93              possible_durations=possible_durations,
94              notes=[
95                  (0, WeightedChoice(possible_durations).choose())
96              ]
97          ))
98
99      def generate_snares(self):
100         u""" Gera os `snares` que aparecerção na composição """
101         number_of_snares = WeightedChoice(
102             (1, 0.2), (2, 0.4), (3, 0.3), (4, 0.09), (0, 0.01)
103         ).choose()
104
105         return (self.instrument.snare, self.calculate_notes(
106             quantity=number_of_snares,
107             possible_durations=[(4, 0.5), (8, 0.3), (16, 0.2)],
108         ))
109
110     def generate_hihats(self):
111         u""" Gera os `hihats` que aparecerção na composição """
112         number_of_hihats = WeightedChoice(
113             (1, 0.2), (2, 0.4), (3, 0.3), (4, 0.09), (0, 0.01)
114         ).choose()
115
116         return (self.instrument.hi_hat, self.calculate_notes(
117             quantity=number_of_hihats,
118             possible_durations=[(4, 0.5), (8, 0.3), (16, 0.2)],
119         ))
120
121     def generate_rides(self):
122         u""" Gera os `rides` que aparecerção na composição """
123         number_of_rides = WeightedChoice(
124             (1, 0.2), (2, 0.4), (3, 0.3), (4, 0.09), (0, 0.01)
125         ).choose()
126
127         return (self.instrument.ride, self.calculate_notes(
128             quantity=number_of_rides,
129             possible_durations=[(4, 0.60), (8, 0.40)],
130         ))
```

Fonte: Imagem do código do Popgen

Figura 3.4 - Método que calcula o rítmo da composição

```
132     def generate_bar(self):
133         u""" Gera o ritmo da música, utilizando todas as notas de percussção
134         possivels. Retorna esse ritmo representado por um objeto do tipo
135         `mingus.container.Bar` """
136         beats = [
137             self.generate_kicks(),
138             self.generate_rides(),
139             self.generate_snares(),
140             self.generate_hihats(),
141         ]
142 
143         p = set()
144         for instr, beat in beats:
145             p |= set(map(itemgetter(0), beat))
146 
147         bar = Bar()
148         for i in range(16):
149             bar.place_notes(None, 16)
150 
151         for i in sorted(p):
152             container = NoteContainer()
153             for instr, beat in beats:
154                 beat = (filter(lambda b: b[0] == i, beat) or [None]).pop()
155                 if beat:
156                     container.add_note(instr)
157 
158             bar.bar.pop(bar.bar.index([i / 16., 16, None]))
159             bar.bar.append([i / 16., 16, container])
160         bar.bar = sorted(bar.bar, key=itemgetter(0))
161         bar.current_beat = 1.0
162 
163         return bar
```

Fonte: Imagem do código do Popgen

O método *generate_bar* cria o rítmo da música, calculando as posições e durações de todos os tipos de batidas. Após realizar esses cálculos, esse método agrupa os resultados em uma barra de compasso.

3.2.2.2 Harmonia

O trabalho de Elowsson (2012) permite a criação de sequências de quatro ou oito acordes, utilizando uma cadeia de Markov de 3ª ordem para sequências de quatro acordes, e uma cadeia de Markov de 2ª ordem para a sequência de oito acordes. Porém, em Elowsson (2012) é apresentado apenas o primeiro nível da cadeia para a sequência de quatro acordes. Como os parâmetros iniciais do Popgen são baseados nos resultados obtidos nesse artigo, a

implementação foi limitada a cadeias de Markov de 1ª ordem e em sequências de quatro acordes. A cadeia de Markov utilizada pode ser vista na Tabela 3.1.

Tabela 3.1 - Cadeia de Markovo utilizada

	I	**II**	**III**	**IV**	**V**	**VI**
I	24	35	0	20	70	5
II	2	2	5	1	1	5
III	2	1	0	1	2	1
IV	39	4	85	1	13	49
V	20	86	2	76	1	39
VI	35	4	8	1	14	1

Fonte: Elowsson (2012)

3.2.2.3 Estrutura de Frases

No trabalho de Elowsson (2012), as Estruturas de Frases possuem diversas regras para a criação de silencio e deslocamento da melodia. A implementação dessa etapa foi simplificada para que as estruturas de frases influenciem apenas as repetições em nível de frases. A Estrutura pode ser passada como parâmetro para o algoritmo, no formato de uma lista de tuplas, onde cada tupla é composta por dois elementos: o primeiro elemento é o nome da frase e o segundo é a duração. A duração é um número inteiro e determina quantas vezes haverá a repetição da sequência de acordes dentro da frase.

3.2.2.4 Melodia

Todo o cálculo da melodia é realizado pela classe *Melody* (Figura 3.5), que implementa o método de composição abordado na Seção 2.2.1.5. Ou seja, o cálculo das notas é feito uma nota de cada vez (Figuras 3.6 e 3.7), utilizando um conjunto de regras que calculas a probabilidade de ocorrência das notas (Figura 3.8). As regras são implementadas como métodos da classe *Melody*. As seguintes regras presentes no trabalho de Elowsson (2012) foram implementadas: “Ambitus” (Figura 3.9), “Harmonic Compliance” (Figuras 3.10 e 3.11), “Intervals and Harmonic Complicance” (Figura 3.12), “Note Length” (Figura 3.13), “Note Length and Harmonic Compliance” (Figura 3.14), “Good Continuation” (Figura 3.15).

Figura 3.5 - Construtor da classe Melody

```
16  HARMONIC_COMPLIANCE = [
17      [0.94, 0.30, 0.95, 0.16, 0.87, 0.26, 0.15],  # I
18      [0.20, 0.90, 0.26, 0.86, 0.24, 0.88, 0.02],  # II
19      [0.01, 0.18, 0.87, 0.09, 0.89, 0.24, 0.83],  # III
20      [0.90, 0.26, 0.18, 0.82, 0.29, 0.99, 0.01],  # IV
21      [0.28, 0.92, 0.28, 0.27, 0.95, 0.30, 0.75],  # V
22      [0.92, 0.28, 0.85, 0.03, 0.25, 0.91, 0.20],  # VI
23  ]
24
25  DYNAMICS = [10, 1, 3, 1, 6, 1, 3, 1, 8, 1, 3, 1, 6, 1, 3, 1]
26
27  CONTINUATION = [1.6, 1.35, 1.19, 0.9, 0.75, 0.6, 0.5, 0.45, 0.41, 0.35]
28
29
30  class Melody(object):
31
32      def __init__(self, scale="C", tempo=110, power=1,
33                   preferred_range=("A-3", "A-4"), maximum_range=("F-2", "D-4"),
34                   inner_drop_off=0.04, outer_drop_off=0.15,
35                   harmonic_compliance=HARMONIC_COMPLIANCE, dynamics=DYNAMICS):
36          self.preferred_range = preferred_range
37          self.maximum_range = maximum_range
38          self.inner_drop_off = inner_drop_off
39          self.outer_drop_off = outer_drop_off
40          self.power = power
41          self.scale = scale
42          self.tempo = tempo
43          self.harmonic_compliance = harmonic_compliance
44          self.dynamics = dynamics
45          self.phrase_structure = []
46          self.harmony = []
```

Fonte: Imagem do código do Popgen

Figura 3.6 - Método que gera a melodia da composição

```
97      def generate_melody(self, ):
98          melody_bars = []
99          self._phrases = {}
100         self._init_caches()
101
102         for phrase, bars in self.phrase_structure:
103             melody_bars.extend(self._generate_phrase_bars(phrase, bars))
104         return melody_bars
```

Fonte: Imagem do código do Popgen

Figura 3.7 - Método que gera a melodia de uma frase

```
106     def _generate_phrase_bars(self, phrase, bars):
107         if phrase in self._phrases:
108             return self._phrases[phrase]
109         phrase_bars = []
110         self.melody = []
111         self.last_note = (None, None)
112         for bar in range(bars):
113             for chord in self.harmony:
114                 self.current_chord = chord
115                 self.melody.append([])
116                 melody_bar = Bar()
117                 while not melody_bar.is_full():
118                     probabilities = []
119                     self.current_beat = melody_bar.current_beat
120                     for note, beat in self.suggested_notes:
121                         probabilities.append(self.calculate_score(note, beat))
122 
123                     total = Decimal(sum(probabilities))
124                     if not total:
125                         note, beat = None, 1 / (1 - self.current_beat)
126                     else:
127                         normalize = lambda p: p / total
128                         probabilities = map(normalize, probabilities)
129 
130                         values = [self.suggested_indexes, probabilities]
131                         index = rv_discrete(values=values).rvs()
132 
133                         note, beat = self.suggested_notes[index]
134                     self.melody[-1].append((note, beat))
135                     self.last_note = (note, beat)
136                     if note is not None:
137                         cbeat = int(16 * self.current_beat)
138                         velocity = 110 + self.dynamics[cbeat]
139                         velocity = velocity if velocity < 128 else 127
140                         note.velocity = velocity
141                         note = NoteContainer(note)
142                     note = note
143                     melody_bar.place_notes(note, beat)
144                 phrase_bars.append(melody_bar)
145         self._phrases[phrase] = phrase_bars
146         return phrase_bars
```

Fonte: Imagem do código do Popgen

Figura 3.8 - Método que calcula a pontuação de uma nota

```
334     def calculate_score(self, note, beat):
335         score = 0
336         # Ambitus
337         score += self.calculate_ambitus(note)
338 
339         # Harmonic Compliance
340         score += self.calculate_harmonic_compliance(note)
341 
342         # Intervals & Harmonic Compliance
343         score += self.calculate_intervals_n_harmonic_compliance(note)
344 
345         # Note Length
346         score += self.calculate_note_length(beat)
347 
348         # Note Length & Harmonic Compliance
349         score += self.calculate_note_length_n_harmonic_compliance(note, beat)
350 
351         # Good Continuation
352         score += self.calculate_good_continuation(note)
353 
354         if score < 0:
355             return 0
356 
357         return score
```

Fonte: Imagem do código do Popgen

Figura 3.9 - Método que calcula o a pontuação da regra “Ambitus”

```
181     def calculate_ambitus(self, note):
182         score = self.ambitus_cache.get(note, None)
183         if score:
184             return score
185         score = Decimal(1)
186         pref_notes = self._pref_notes
187         maximum_notes = self._max_notes
188 
189         if not getattr(self, 'lower_note', None):
190             self.lower_note = min(pref_notes)
191         if not getattr(self, 'upper_note', None):
192             self.upper_note = max(pref_notes)
193 
194         if getattr(self, 'lower', None) is None:
195             self.lower = filter(lambda r: r < min(pref_notes), maximum_notes)
196         if getattr(self, 'upper', None) is None:
197             self.upper = filter(lambda r: r > max(pref_notes), maximum_notes)
198 
199         if self.lower_note <= note <= self.upper_note:
200             if len(pref_notes) % 2 == 1:
201                 median = len(pref_notes) / 2
202             else:
203                 m = pref_notes[(len(pref_notes) / 2) - 1]
204                 n = pref_notes[len(pref_notes) / 2]
205                 if note <= m:
206                     median = pref_notes.index(m)
207                 else:
208                     median = pref_notes.index(n)
209             offset = abs(median - pref_notes.index(note))
210             lowered_by = self.inner_drop_off * offset
211         elif note <= self.lower_note:
212             lowered_by = (self.lower.index(note) + 1) * self.outer_drop_off
213         elif note >= self.upper_note:
214             lowered_by = (self.upper.index(note) + 1) * self.outer_drop_off
215         else:
216             raise ValueError("Invalid note", note)
217 
218         score *= Decimal(1 - lowered_by)
219         self.ambitus_cache[note] = score
220         return score
```

Fonte: Imagem do código do Popgen

Figura 3.10 - Método que calcula a conformidade de uma nota com um acorde

```
148     def get_note_chord_compliance(self, note, chord):
149         if not getattr(self, "_chord_compliance", None):
150             self._chord_compliance = {}
151         if not self._chord_compliance.get((note, chord)):
152             a = ['I', 'II', 'III', 'IV', 'V', 'VI']
153 
154             key_chords = map(determine, progressions.to_chords(a, self.scale))
155             key_chords = map(itemgetter(0), key_chords)
156 
157             p = self.harmonic_compliance[key_chords.index(chord)]
158             comp = p[get_scale(self.scale).ascending()
159                      .index(note.name)]
160 
161             self._chord_compliance[(note, chord)] = comp
162 
163         return self._chord_compliance.get((note, chord))
```

Fonte: Imagem do código do Popgen

Figura 3.11 - Método que calcula o a pontuação da regra “Harmonic Compliance”

```
222     def calculate_harmonic_compliance(self, note):
223         compliance = self.get_note_chord_compliance(note, self.current_chord)
224         return Decimal(compliance)
```

Fonte: Imagem do código do Popgen

Figura 3.12 - Método que calcula o a pontuação da regra “Intervals and harmonic compliance”

```
226     def calculate_intervals_n_harmonic_compliance(self, note):
227         score = 1
228         last_note = self.last_note[0]
229         if last_note is None:
230             return score
231         if not getattr(self, '_intervals_n_harmonic_compliance', None):
232             self._intervals_n_harmonic_compliance = {}
233         if self._intervals_n_harmonic_compliance.get((last_note, note)):
234             return self._intervals_n_harmonic_compliance[(last_note, note)]
235 
236         # Harmonic Compliance
237         C = self.get_note_chord_compliance
238         interval = self.get_interval(last_note, note)
239         last_note_compliance = C(last_note, self.current_chord)
240         current_note_compliance = C(note, self.current_chord)
241         # As it is not clear how they calculate, let's try guessing
242         last_note_score = interval * (last_note_compliance - 0.5)
243         current_note_score = interval * (current_note_compliance - 0.5)
244         score += (last_note_score + current_note_score) / 2.0
245 
246         # One pitch step
247         chord_notes = chords.from_shorthand(self.current_chord)
248         if interval >= 1:
249             if not (last_note.name in chord_notes or note.name in chord_notes):
250                 score -= 0.15
251         # Two Pitch steps 1
252         is_last_pentatonic = self.is_pentatonic(last_note)
253         is_this_pentatonic = self.is_pentatonic(note)
254         if interval >= 2:
255             if not (last_note.name in chord_notes or note.name in chord_notes):
256                 if is_last_pentatonic and is_this_pentatonic:
257                     score -= 0.1
258                 else:
259                     score -= 0.2
260         # Two Pitch steps 2
261         if interval >= 2:
262             if not (is_last_pentatonic and is_this_pentatonic):
263                 score -= 0.05
264 
265         score = Decimal(score)
266         self._intervals_n_harmonic_compliance[(last_note, note)] = score
267         return score
```

Fonte: Imagem do código do Popgen

Figura 3.13 - Método que calcula o a pontuação da regra “Note Length”

```
269     def calculate_note_length(self, beat):
270         score = self.note_length_cache.get((self.current_beat, beat), None)
271         if score:
272             return Decimal(score)
273 
274         if self.current_beat + (16. / beat) / 16. > 1.0:
275             self.note_length_cache[(self.current_beat, beat)] = -1000
276             return -1000
277 
278         beat_ = self.possible_beats.get(beat, None)
279 
280         if beat_:
281             if beat_[1] and self.current_beat not in beat_[1]:
282                 self.note_length_cache[(self.current_beat, beat)] = -1000
283                 return -1000
284             score = beat_[0]
285         else:
286             raise ValueError("Unexpected beat", beat)
287 
288         if score < 0:
289             score = -1000
290         self.note_length_cache[(self.current_beat, beat)] = score
291         return Decimal(score)
```

Fonte: Imagem do código do Popgen

Figura 3.14 - Método que calcula o a regra “Note Length and Harmonic Compliance”

```
293     def calculate_note_length_n_harmonic_compliance(self, note, beat):
294         last_note, last_beat = self.last_note
295         if not last_note:
296             return 0
297 
298         score = 1
299         C = self.get_note_chord_compliance
300         current_note_compliance = C(note, self.current_chord)
301 
302         # Poor
303         if current_note_compliance < 0.5:
304             # Shorter with poor = slightly higher
305             # Longer with poor = lower
306             self.poor_compliance_score[beat]
307 
308         # Good
309         else:
310             # Shorter with good = lower
311             # Longer with good = slightly higher
312             self.good_compliance_score[beat]
313 
314         return Decimal(score)
```

Fonte: Imagem do código do Popgen

Figura 3.15 - Método que calcula o a pontuação da regra “Good Continuation”

```
316     def calculate_good_continuation(self, note):
317         last_note = self.last_note[0] or note
318         direction = copysign(1, int(note) - int(last_note))
319         if self.current_direction == 0:
320             self.current_direction = direction
321             self.steps_in_same_direction = 0
322         elif copysign(1, self.current_direction) == direction:
323             self.steps_in_same_direction += 1
324         else:
325             self.current_direction = 0
326             self.steps_in_same_direction = 0
327 
328         if self.steps_in_same_direction == 0:
329             return Decimal(0.1)
330         elif self.steps_in_same_direction > len(CONTINUATION):
331             return Decimal(0.1)
332         return Decimal(CONTINUATION[self.steps_in_same_direction - 1])
```

Fonte: Imagem do código do Popgen

3.3 Definição dos Parâmetros

A escolha de parâmetros que sejam compreendidos por leigos em música, sem perder o poder de diversificação do algoritmo, é uma das tarefas mais importantes na concepção de um sistema voltado para a utilização por leigos em música. Esses parâmetros devem permitir uma fácil utilização por parte do usuário, sem exigir a utilização de mecanismos não convencionais de interação (MILETO 2007).

Em (MILETO 2007) foram apresentadas algumas sugestões para a elaboração de interfaces para atividades musicais que possam ser utilizadas por leigos e por músicos, e dentre essas sugestões estão a utilização de metáforas musicais presentes no dia-a-dia e que possam ser conhecidas por qualquer pessoa, não só por músicos, e evitar utilizar notação musical tradicional ou exigir do usuário conhecimento prévio em teoria musical. Essas sugestões, apesar de serem voltadas para a elaboração da interface, podem ser aplicadas na definição dos parâmetros do sistema.

3.3.1 Conjunto de Instrumentos

A seleção dos instrumentos utilizados no trabalho foi definida através de uma análise subjetiva da qualidade dos instrumentos disponíveis no *Arachno*. Após selecionar esses

instrumentos, eles foram agrupados em conjuntos, seguindo algumas regras de gênero musical e tipos dos instrumentos. Os instrumentos agrupados por Conjuntos podem ser vistos na Tabela 3.2.

Tabela 3.2 - Instrumentos disponíveis em cada Conjunto

	Melodia	Harmonia	Base
Acústico	➔ Harmonica ➔ Tango Accordion ➔ Acoustic Guitar (steel) ➔ Accordion	➔ Acoustic Guitar (nylon)	➔ Acoustic Bass
Jazz	➔ Alto Sax ➔ Tenor Sax ➔ Trumpet	➔ Electric Guitar (jazz) ➔ Bright Acoustic Piano	➔ Electric Bass (finger) ➔ Cello ➔ Contrabass
Orquestra	➔ Violin ➔ Acoustic Grand Piano	➔ Viola	➔ Contrabass ➔ Cello
Piano	➔ Honky-tonk Piano ➔ Acoustic Grand Piano ➔ Electric Grand Piano	➔ Electric Piano 1 ➔ Bright Acoustic Piano	➔ Electric Piano 2 ➔ Bright Acoustic Piano
Rock	➔ Overdriven Guitar ➔ Distortion Guitar	➔ Electric Guitar (jazz) ➔ Electric Guitar (muted) ➔ Overdriven Guitar	➔ Slap Bass 1 ➔ Electric Bass (finger) ➔ Electric Bass (pick) ➔ Slap Bass 2
Eletrônico	➔ SynthBrass 1 ➔ Lead1 (square) ➔ Lead8 (bass + lead) ➔ Lead8 (bass + lead)	➔ SynthStrings 1 ➔ Pad4 (choir) ➔ Pad3 (polysynth) ➔ Pad3 (polysynth)	➔ Synth Bass 1 ➔ Synth Voice ➔ Pad7 (halo) ➔ Lead3 (calliope)

Fonte: Tabela criada pelo autor (2016)

3.3.1.1 Acústico

Também chamados de *instrumentos não eletrônicos* (ELSEA 1996), os instrumentos acústicos podem ser definidos como "Um instrumento musical sem amplificação elétrica" (OXFORD 2010), e são amplamente utilizados em músicas populares.

De acordo com Elsea (1996) e UFRGS (2016a), os instrumentos acústicos podem ser divididos em três categorias: cordas, ar/sopro e percussão. Seguindo as definições e categorizações acima, os instrumentos MIDI utilizados nesse conjunto devem simular instrumentos reais que são utilizados sem amplificação elétrica e que se enquadrem em alguma das três categorias. Logo, instrumentos MIDI que simulem sintetizadores não podem ser utilizados nesse conjunto.

Os instrumentos de corda acústicos utilizados foram o violão ("Acoustic Guitar (steel)", "Acoustic Guitar (nylon)") e o baixo acústico ("Acoustic Bass"), por serem instrumentos acústicos amplamente utilizados em gêneros musicais considerados populares. Dentre os instrumentos de ar/sopro, a gaita harmônica ("Harmonica") foi escolhida, pois também pode ser encontrada em diversas músicas populares. A flauta doce também foi considerada por ser um instrumento de sopro e ter uma sonoridade boa. Porém, o seu volume é muito baixo no banco de instrumentos utilizado, fazendo com que os instrumentos de harmonia e percussão se sobrepusessem ao seu som.

O acordeão também é considerado um instrumento acústico, apesar de haver divergências em torno da sua classificação. No *General MIDI*, existem dois acordeões: "Tango Accordion" e "Accordion". Ambos foram utilizados no trabalho.

3.3.1.2 Jazz

Não existem restrições ou regras que regulem quais instrumentos estarão presentes em uma música de jazz, porém alguns instrumentos são mais comuns, como o saxofone, trompete, piano, guitarras, dentre outros (JAZZ 2016). É importante lembrar que o objetivo desse conjunto não é que a composição seja uma música de jazz, mas sim que a sonoridade dos instrumentos passe a ideia de uma música de jazz.

Instrumentos de trompa, são usualmente utilizados para tocar a melodia das músicas de jazz (JAZZ 2016). Por essa razão, os instrumentos de melodia escolhidos para esse Conjunto foram os saxofones (“Alto Sax” e “Tenor Sax”) e o trompete (“Trumpet”).

Na harmonia, podemos encontrar tanto pianos quanto guitarras elétricas, e eles compõem, juntamente com os instrumentos de base e percussionistas, a seção rítmica das músicas de jazz (JAZZ 2016). Os instrumentos de harmonia escolhidos foram “Electric Guitar (jazz)” e “Bright Acoustic Piano”, e para a base “Cello”, “Electric Bass (finger)” e “Contrabass”.

3.3.1.3 Orquestra

Orquestras podem contar com uma grande diversidade de instrumentos, como pode ser visto em Philarmonia Orchestra (2016a), com várias possibilidades de combinações entre eles. Alguns instrumentos, no entanto, são considerados essenciais para a sonoridade da orquestra.

Um desses instrumentos é o violino (PHILARMONIA ORCHESTRA 2016b), sendo responsável muitas vezes pela execução da melodia das músicas. Também é comum encontrarmos pianistas executando a melodia com uma orquestra de fundo (PHILARMONIA ORCHESTRA 2016c; OREGON SYMPHONY 2016). A importância desses instrumentos na orquestra motivou a utilização dos instrumentos MIDI “Violin” e “Acoustic Grand Piano” para a execução da melodia desse conjunto.

Em muitas orquestras, a viola pode ser encontrada tocando a melodia das músicas (PHILARMONIA ORCHESTRA 2016d), porém a sua utilização ficou limitada à execução da harmonia através do instrumento MIDI “Viola”, pois o volume e as notas atingidas por ela no banco de instrumentos MIDI utilizado não permitem sua utilização de outra forma.

Por serem instrumentos mais graves, o violoncelo e o contrabaixo podem ser utilizados para executar a base nas músicas compostas pelo sistema. Além disso, o timbre do violoncelo permite a criação de uma sonoridade melancólica (PHILARMONIA ORCHESTRA 2016e). Os instrumentos MIDI utilizados para a base são o “Contrabass” e o “Cello”.

3.3.1.4 Piano

O padrão GM possui oito instrumentos na categoria “Pianos” e, como mencionado anteriormente, pianos são instrumentos capazes de executar músicas de forma satisfatória sem necessidade de acompanhamento por outros instrumentos. Seis dos oito pianos disponíveis no GM foram utilizados, sendo eles: “Acoustic Grand Piano”, “Bright Acoustic Piano”, “Electric Grand Piano”, “Electric Piano 1”, “Electric Piano 2”, “Honky-tonk Piano”. Os pianos foram escolhidos através de uma avaliação de sonoridade nas faixas de notas necessárias para harmonia, melodia e base.

3.3.1.5 Rock

Da mesma forma que pode ser observado no Jazz, o Rock possui uma grande variedade de instrumentos, podendo ir desde instrumentos puramente acústicos até a utilização de sintetizadores nas músicas. Porém, alguns instrumentos são clássicos nesse gênero musical. As guitarras elétricas, sejam elas limpas ou utilizando distorção, estão presentes em uma grande parte das músicas desse gênero, e podem ser consideradas um ícone do mesmo. Elas são utilizadas tanto para a execução da harmonia, como para a execução da melodia, em conjunto com o vocal. Acompanhando as guitarras, temos os baixos elétricos, que são responsáveis por reproduzir a base da música, acompanhando a harmonia.

Levando esses pontos em consideração, os instrumentos selecionados para esse conjunto foram: “Overdriven Guitar”, “Distortion Guitar”, “Electric Guitar (jazz)”, “Electric Guitar (muted)”, “Slap Bass 1”, “Electric Bass (finger)”, “Electric Bass (pick)” e “Slap Bass 2”.

3.3.1.6 Eletrônico

Conforme apontado na Seção 3.3.1.1, instrumentos “não eletrônicos” são todos instrumentos que não são amplificados de forma eletrônica. O “Conjunto de Instrumentos” Eletrônico tem por objetivo passar a sonoridade presente em músicas eletrônicas, sonoridade essa que, em sua grande maioria, é criada com a utilização de sintetizadores eletrônicos, sejam eles analógicos ou digitais. Para atingir esse objetivo, os instrumentos GM mais adequados são aqueles que simulam o som de sintetizadores, e existem três famílias de instrumentos com esse objetivo no GM, totalizando 24 instrumentos. Através de uma avaliação de sonoridade,

os intrumentos escolhidos foram “SynthBrass 1”, “Lead1 (square)”, “Lead8 (bass + lead)”, “Lead8 (bass + lead)”. “SynthStrings 1”, “Pad4 (choir)”, “Pad3 (polysynth)”, “Pad3 (polysynth)”, “Synth Bass 1”, “Synth Voice”, “Pad7 (halo)” e “Lead3 (calliope)”.

3.3.2 Emoção

O parâmetro Emoção foi escolhido pois, além de não ser algo exclusivo de músicas – o que possibilita a leigos em música entender seu efeito na mesma -, é algo presente na maioria das músicas, e é um dos fatores que colaboram para que a música soe mais realista, mesmo quando executada por um computador.

Existem diversas emoções, e elas podem ser percebidas de formas diferentes pelas pessoas. Então, como escolher quais emoções estarão disponíveis para o usuário nessa opção? Como fazer com que a opção selecionada seja refletida na música composta?

Para responder essas questões, olhamos novamente para o CMERS, apresentado na Seção 2.2.2. O sistema desenvolvido por Livingstone (2010) permite a alteração de características musicais de tal forma que a música resultante transmita uma determinada emoção. Durante o seu desenvolvimento, os autores analisaram diversos estudos anteriores para determinar como alterar os parâmetros para que a música vá em direção à emoção desejada. O CMERS possibilita a alteração da música para quatro emoções diferentes: “Feliz”, “Triste”, “Irritado” e “Tenro”.

Diferente do CMERS, que modifica as músicas já compostas, o Minstrel altera os parâmetros que serão passados para o Popgen, para que a composição criada tenha características associadas com a emoção escolhida. A definição dos parâmetros do Popgen são feitas levando em consideração algumas regras estruturais do CMERS, sendo elas: Tempo, Modo, Volume da Melodia e Altura do Tom.

O Tempo altera o BPM da música, que é definido por um sorteio na faixa da emoção. O Modo altera a escala da música para maior ou menor, dependendo da emoção. O Volume da Melodia influencia na dinâmica da melodia da música, e Altura do Tom na faixa de notas preferida e máxima da melodia.

Tabela 3.3 - Como cada emoção altera os parâmetros do Popgen

	Tempo	Modo	Volume da Melodia	Altura do Tom
Feliz	[90, 180] BPM	Maior	+[5, 14]	+4 semitons
Triste	[60, 120] BPM	Menor	+[4, 9]	Não varia
Irritado	[90, 180] BPM	Menor	+[8, 17]	-4 semitons
Sereno	[60, 100] BPM	Maior	+[1, 7]	+4 semitons

Fonte: Tabela criada pelo autor (2016)

O resultado da aplicação das regras antes da composição é inferior à forma como o CMERS foi projetado para ser utilizado por diversos motivos, porém a implementação foi feita dessa forma por questões de tempo.

3.3.3 Complexidade

A Complexidade afeta a quantidade de repetições e a faixa de notas que podem ser executadas na melodia da música, tendo relação direta com os parâmetros Emoção, Duração e Conjunto de Instrumentos. As repetições são feitas no nível das frases, ou seja, dependendo da complexidade escolhida, a música poderá (ou não) repetir algumas frases durante a sua execução. As faixas de notas preferidas e máximas da melodia são afetadas pela complexidade, sendo que, quanto maior a complexidade, maior o tamanho dessas faixas.

3.3.4 Duração

Com o objetivo de ser o parâmetro mais simples do sistema, a Duração afeta a quantidade de frases da música e, consequentemente, seu tempo de duração. Apesar dessa simplicidade, ele possui uma forte sinergia com os parâmetros de Emoção e Complexidade. A duração da música não é definida apenas pela quantidade de frases, mas sim pela combinação disso com o BPM da música, que é definido pelo parâmetro Emoção. Como o parâmetro Complexidade cria repetições no nível das frases, a distribuição dessas repetições na música é afetada tanto pela Complexidade quanto pela Duração escolhidas.

3.4 Elaboração da Interface

Com o intuito de elaborar uma interface que permita a utilização por leigos em música, algumas das sugestões feitas em Mileto (2010) foram levadas em consideração no trabalho. Conforme visto na Seção 3.3.1, os parâmetros foram definidos sem a utilização de uma notação musical tradicional, e não demandam conhecimento prévio de teoria musical ou outros conceitos de música, o que ajuda a deixar o sistema mais amigável para leigos em música (MILETO 2010). A escolha dos nomes desses parâmetros e das suas opções, com exceção do parâmetro Complexidade, também foi feita levando em consideração uma sugestão de Mileto (2010), buscando "utilizar metáforas musicais da vida real, conhecidas por qualquer um, e não apenas na realidade dos músicos"(MILETO 2010). Para complementar a escolha do nome dos parâmetros, foram utilizadas imagens que representam as opções dos parâmetros. As imagens utilizadas em cada parâmetro podem ser vistas nas Figuras 3.16, 3.17, 3.18 e 3.19.

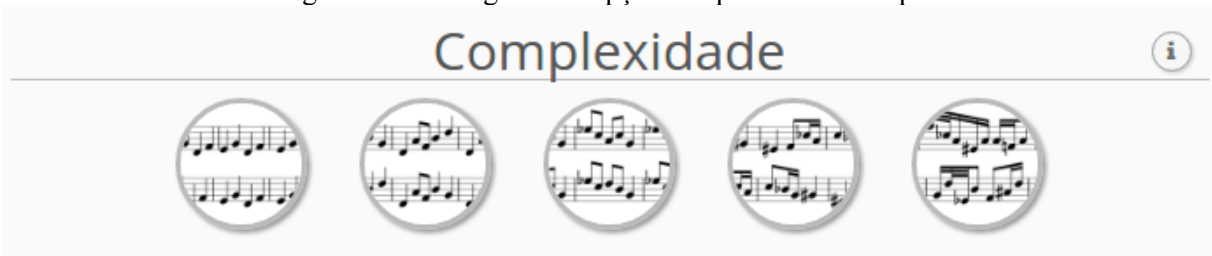
Figura 3.16 - Imagens das opções do parâmetro Conjunto de Instrumentos

Fonte: Imagem do sistema criado pelo autor

Figura 3.17 - Imagens das opções do parâmetro Emoção

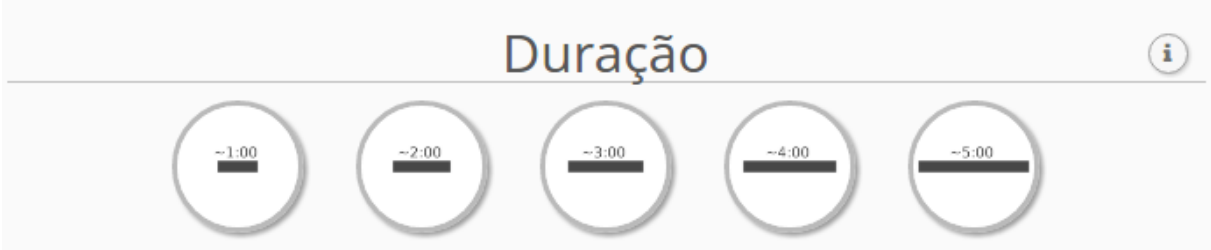
Fonte: Imagem do sistema criado pelo autor

Figura 3.18 - Imagens das opções do parâmetro Complexidade



Fonte: Imagem do sistema criado pelo autor

Figura 3.19 - Imagens das opções do parâmetro Duração

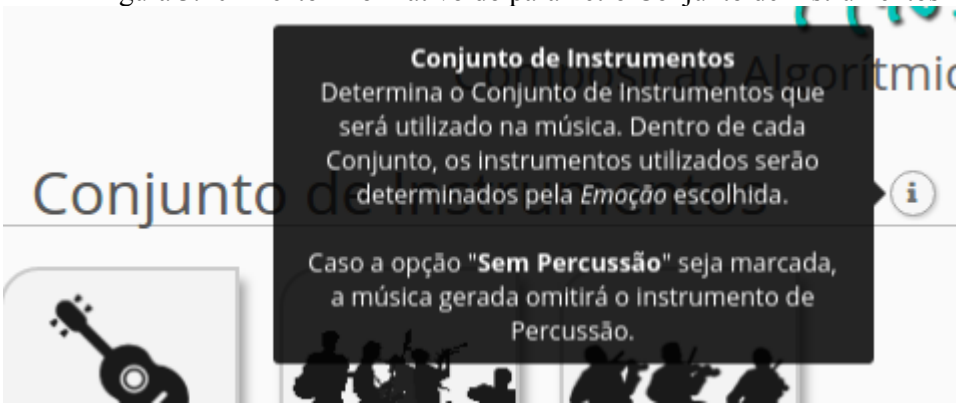


Fonte: Imagem do sistema criado pelo autor

Outro recurso utilizado para ajudar no entendimento do funcionamento dos parâmetros foram os textos de ajuda, que permitem adicionar mais informações à interface, removendo possíveis ambiguidades ou dúvidas que o usuário tenha. Esses textos podem ser encontrados

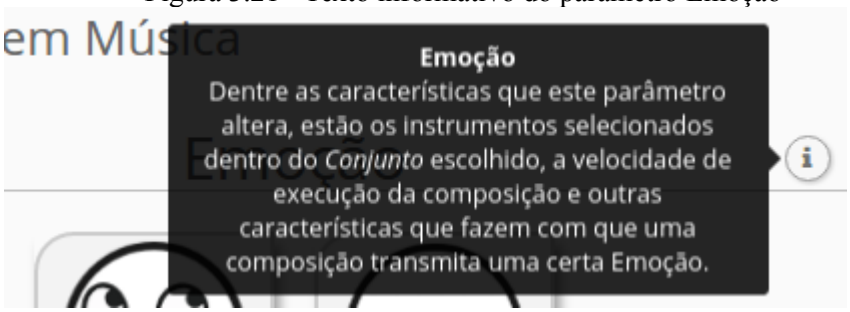
quando o usuário passar o mouse no ícone presente ao lado do nome dos parâmetros, como é possível ver nas Figuras 3.20, 3.21, 3.22 e 3.23.

Figura 3.20 - Texto informativo do parâmetro Conjunto de Instrumentos



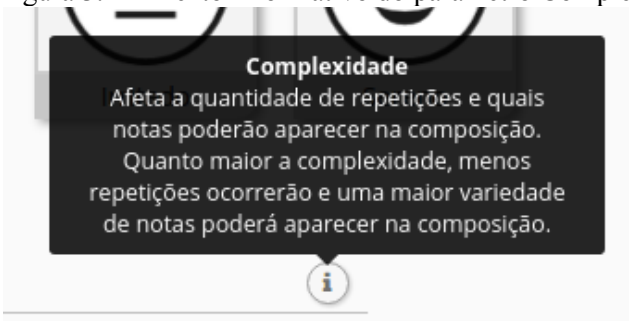
Fonte: Imagem do sistema criado pelo autor

Figura 3.21 - Texto informativo do parâmetro Emoção



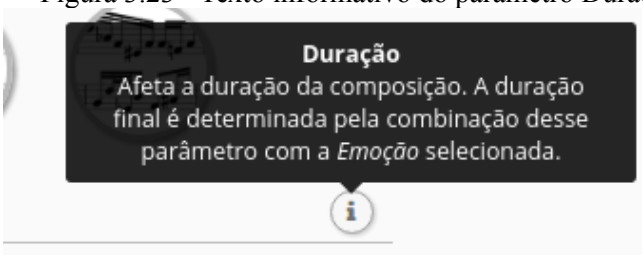
Fonte: Imagem do sistema criado pelo autor

Figura 3.22 - Texto informativo do parâmetro Complexidade



Fonte: Imagem do sistema criado pelo autor

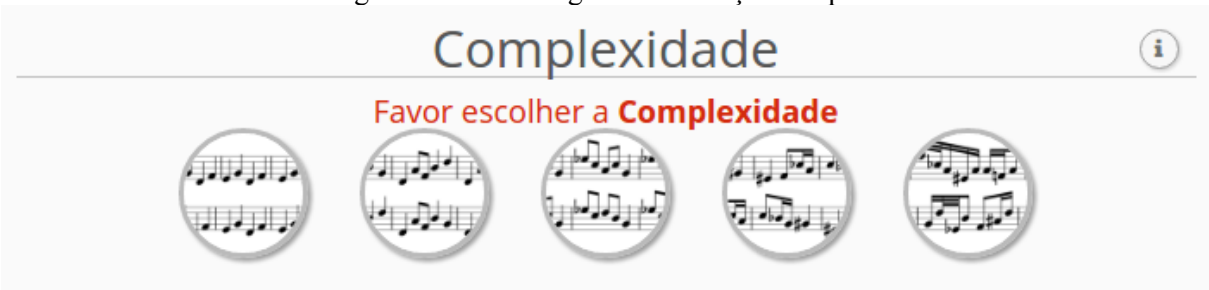
Figura 3.23 - Texto informativo do parâmetro Duração



Fonte: Imagem do sistema criado pelo autor

Por fim, a inteface foi projetada para que toda ação rearlizada passe um *feedback* para o usuário. Caso o usuário tente compor uma música sem selecionar os parâmetros, uma mensagem de validação irá aparecer logo abaixo do título do parâmetro, como é possível ver na Figura 3.24. Ao selecionar uma opção em algum parâmetro, o sistema irá destacar essa opção usando cores vivas, de tal forma que fique claro para o usuário qual a opção selecionada (Figura 3.25). Enquanto o sistema estiver compondo, uma mensagem irá aparecer até que a composição esteja concluída (Figura 3.26).

Figura 3.24 - Mensagem de validação dos parâmetros

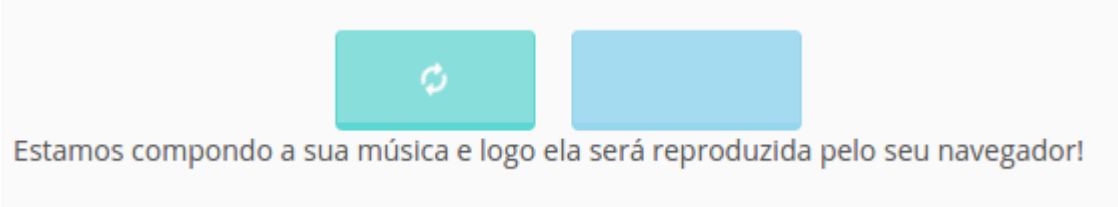


Fonte: Imagem do sistema criado pelo autor

Figura 3.25 - Ao selecionar uma opção em algum parâmetro, o sistema irá destacar essa opção de tal forma que fique claro para o usuário qual a opção selecionada

Fonte: Imagem do sistema criado pelo autor

Figura 3.26 - Mensagem que aparece enquanto o sistema estiver compondo

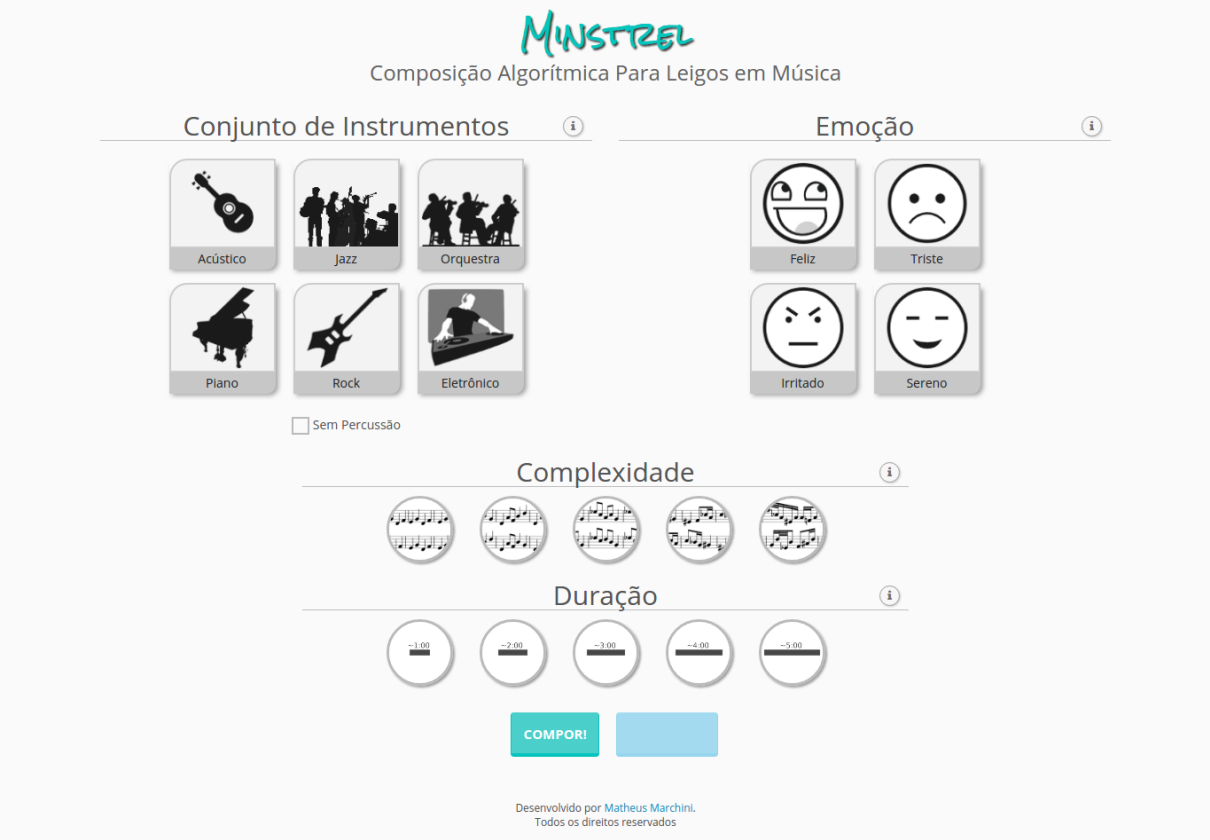


Fonte: Imagem do sistema criado pelo autor

3.4.1 Exemplo de Caso de Uso

O usuário acessa o sistema através do endereço http://minstrel.me/. A tela inicial do sistema pode ser vista na Figura 3.27. Após acessar o sistema, o usuário poderá clicar nos ícones das opções dos parâmetros, os quais ficarão destacados, conforme visto na Figura 3.28. Por exemplo, o usuário pode selecionar as opções: “Jazz”, “Triste”, “3ª Complexidade”, “2ª Duração”, e marcar a opção “Sem Percussão”, como pode ser visto na Figura 3.28.

Figura 3.27 - Captura de tela do estado inicial do sistema



Fonte: Figura do sistema criado pelo autor

Figura 3.28 - Captura de tela do sistema após selecionar os parâmetros

Fonte: Figura do sistema criado pelo autor

Caso o usuário fique com alguma dúvida a respeito do comportamento de algum dos parâmetros, ele poderá posicionar o *mouse* no ícone de informações (“i”) que fica localizado ao lado do título dos parâmetros. Ao posicionar o *mouse* sobre esse ícone, mais informações sobre o parâmetro são exibidas, como pode ser visto na Figura 3.29, que mostra a mensagem sendo exibida após posicionar o *mouse* sobre do ícone de informações de Duração.

Figura 3.29 - Captura de tela do sistema ao posicionar o mouse sobre o ícone de informações de Duração

Fonte: Figura do sistema criado pelo autor

Após selecionar os parâmetros, o usuário clica em “COMPOR!” para compor a música. O sistema exibirá uma mensagem avisando que a música logo será reproduzida, como pode ser visto na Figura 3.26, e assim que a composição estiver pronta – o que leva em média de 3 segundos – o sistema começa a reproduzir a música. A partir do momento que a música estiver reproduzindo, o botão ao lado do botão “COMPOR!” será utilizado para Pausar/Reproduzir a música, como pode ser visto nas Figuras 3.30 e 3.31. Se quiser, o usuário pode compor uma nova música repetindo os passos anteriores.

Figura 3.30 - Captura de tela do sistema durante a execução de uma música

Fonte: Figura do sistema criado pelo autor

Figura 3.31 - Captura de tela do sistema após a música ter sido pausada, ou terminar a sua execução

Fonte: Figura do sistema criado pelo autor

3.5 Testes com Usuários

Após finalizar o desenvolvimento do sistema, ele foi disponibilizado *online* para validação de usuários. Para avaliar o sistema, um questionário sobre a usabilidade do sistema foi feito. A escolha por disponibilizar o sistema *online* foi feita com o intuito de coletar uma quantidade maior de respostas para o questionário.

O sistema, juntamente com o questionário, ficaram disponíveis no período entre 07/06/2016 13:00 e 11/06/2016 23:59. Nesse período o sistema foi acessado 185 vezes por 131 usuários diferentes - de acordo com estatísticas coletadas pelo Google Analytics (2016) - e 51 desses usuários responderam ao questionário.

3.5.1 Elaboração dos Testes

O teste foi elaborado com o objetivo de coletar informações relevantes a respeito do conhecimento musical dos participantes, e a sua percepção sobre a usabilidade do sistema. Para completar esse objetivo, um Formulário de Perfil com algumas perguntas para determinar se o usuário pode ser considerado leigo foi criado.

Seis tarefas foram elaboradas para serem realizadas, utilizando o sistema, permitindo que o usuário perceba as diferenças nas composições causadas pelas alterações dos parâmetros. A primeira tarefa é a composição de uma música, selecionando os parâmetros de acordo com a preferência do usuário. As quatro tarefas seguintes consistem em alterar os parâmetros, um a um, e realizar novas composições, permitindo ao usuário perceber o efeito de cada parâmetro na composição. A última tarefa é, novamente, compor uma música escolhendo os parâmetros de acordo com a preferência do usuário. Ao final de cada tarefa, algumas perguntas são feitas para que o usuário dê sua opinião sobre a utilização do sistema.

O Questionário, juntamente com o Formulário de Perfil e as Tarefas, foi feito utilizando o Google Forms (2016), por facilitar a sua divulgação *online* e a análise dos dados após os testes, que podem ser encontrados no Apêndice.

3.5.2 Aplicação dos Testes

Os testes foram aplicados através da Internet. O sistema ficou hospedado em um servidor de 16GB RAM e 8 cores, no provedor de *cloud computing* DigitalOcean, e estava acessível através do domínio http://minstrel.me durante o período dos testes. O questionário foi divulgado nas redes sociais e grupos de e-mail, e, durante o período de testes, foi possível coletar 51 avaliações, variando entre pessoas leigas em música e pessoas com alguma experiência musical.

3.5.3 Análise dos Resultados

Após a conclusão dos testes, os dados coletados através do questionário foram analisados de duas formas. Primeiro, uma análise das perguntas presentes nas Tarefas 1 a 5, agrupando as respostas pelo Perfil de Usuário. O resultado dessa análise permite avaliar se o sistema possui uma interface e um conjunto de parâmetros satisfatórios para usuários leigos em música de forma objetiva.

A segunda análise foi feita sobre as respostas das perguntas com resposta em texto livre, e foi feita para verificar quais características do sistema foram bem aceitas, quais foram consideradas ruins e o que pode ser melhorado em versões futuras. Por avaliar respostas em campo texto, essa avaliação é mais subjetiva. As respostas foram classificadas em “Pontos Positivos”, “Pontos Negativos” e “Sugestões”.

3.5.3.1 Análise das perguntas presentes nas Tarefas 1 a 5

Com o objetivo de realizar uma análise efetiva, que fosse capaz de exibir o sucesso ou fracasso de cada parâmetro do sistema de acordo com o Perfil do Usuário, primeiro foi necessário definir quais Perfis de Usuário utilizaram o sistema. Através de uma análise das respostas do Formulário de Perfil, foi possível agrupar os resultados em três Perfis diferentes, que podem ser vistos na Tabela 3.4.

Tabela 3.4 - Respostas consideradas para cada Perfil

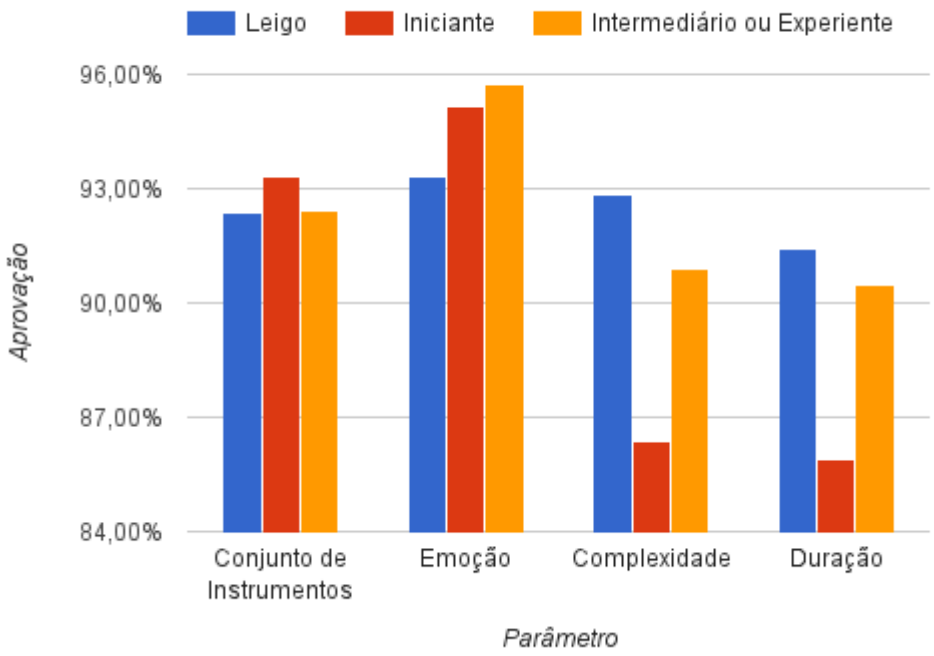
	Leigo	Iniciante	Intermediário ou Experiente
Quantos instrumentos já tocou?	0	1	2 ou mais
Sabe ler partitura?	Não	Um Pouco	Sabe ler
Já compôs alguma música?	Não	Não	Pode ter composto
Quantos softwares musicais já utilizou?	0	1 ou mais	1 ou mais

Fonte: Tabela criada pelo autor (2016)

No Gráfico 3.1 é possível ver a comparação dos resultados da avaliação de cada parâmetro, bem como a comparação da avaliação de cada Perfil. Através de uma análise das avaliações agrupadas, podemos ver que, independente do Perfil ou do Parâmetro, a aceitação dos usuários em relação à forma como os parâmetros foram apresentados na interface ficou acima de 84%. O parâmetro com maior aceitação foi o parâmetro “Emoção”, sendo melhor aceito entre usuários “Intermediários ou Experientes”.

Os parâmetros “Complexidade” e “Duração” foram os menos aprovados pelos usuários, principalmente por aqueles não considerados “Leigos”. A partir disso, podemos confirmar a tese anterior de que o parâmetro “Complexidade” é o mais complexo dos quatro, e refutar a tese de que o parâmetro “Duração” seria o mais simples.

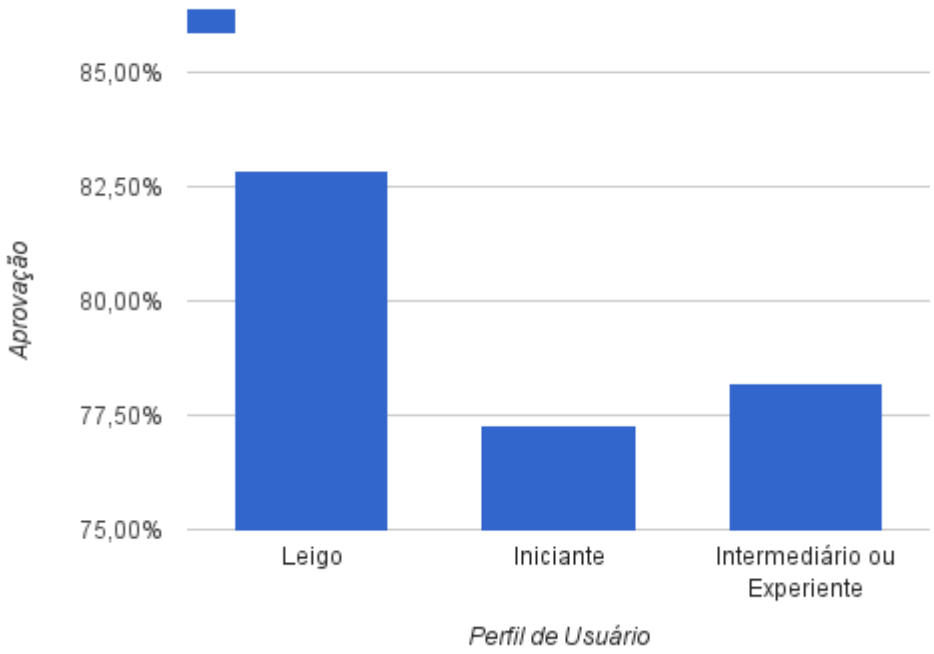
Gráfico 3.1 - Avaliação dos Parâmetros por Perfil de Usuário



Fonte: Gráfico criado pelo autor (2016)

Além da análise dos parâmetros, foi feita uma análise da satisfação dos usuários sobre a coerência entre a música gerada e os parâmetros escolhidos. Essa análise também foi feita agrupando as respostas por Perfil de Usuário. O resultado mostra que usuários leigos ficaram mais satisfeitos com as suas composições iniciais do que os usuários com algum conhecimento musical. A diferença entre a Aprovação os Perfil “Iniciante” e “Intermediário ou Experiente” é estatisticamente insignificante.

Gráfico 3.2 - Aprovação dos Usuários para o resultado da Tarefa 1, por Perfil de Usuário



Fonte: Gráfico criado pelo autor (2016)

3.5.3.2 Pontos Positivos

Vários usuários elogiaram a interface por sua simplicidade, clareza, intuitividade e objetividade, principalmente para usuários leigos. Outros pontos levantados como positivos pelos usuários foi a grande variedade de combinações possíveis para os parâmetros do sistema, e a facilidade/velocidade para alterá-los. O sistema também foi elogiado diversas vezes pela sua velocidade na composição das músicas, confirmando a tese anterior que o tempo para compor pode afetar a percepção do usuário a respeito do sistema. A diversidade de instrumentos que o sistema possui, a possibilidade de desabilitar a percussão e de alterar a Emoção da música foram pontos positivos apontados por uma quantidade significativa de usuários. Por fim, alguns usuários apontaram possíveis aplicações do sistema na criação de músicas de fundo, principalmente para prototipação de jogos.

3.5.3.3 Pontos negativos

Apesar dos pontos positivos referentes à interface, vários usuários mencionaram que a nomenclatura utilizada em algumas partes poderia ser melhorada, bem como os ícones de alguns parâmetros e opções. Os ícones mais criticados foram os de Complexidade (por passar uma ideia errada do seu impacto na composição), Duração (por serem ícones muito parecidos e não deixarem tão claros a duração final da música) e o ícone de Jazz (por ser muito poluído).

As composições foram criticadas em diversos pontos, os mais notórios sendo a invariância da Emoção da música em alguns casos, a SF utilizada e os resultados da opção Eletrônico. Apesar disso, também houve alguns elogios a respeito da SF utilizada, o que mostra que esse é um ponto bem subjetivo.

Apesar de não ser explicitamente criticado, foi possível notar que o parâmetro Conjunto de Instrumentos possa ter passado a ideia errada do seu impacto nas músicas. Algumas respostas – positivas, negativas e sugestões – mostraram que os usuários viram os Conjuntos de Instrumentos como um parâmetro que afeta o Gênero Musical da composição, como podemos ver nesse trecho de uma das respostas: “Todas as composições que eu gerei foram bem de acordo com o que eu escolhi, só estranhei um pouco o eletrônico. Eu esperava uma coisa mais agitada, dançante”. A alteração do nome das opções pode ajudar a reduzir a ambiguidade desse parâmetro.

Por fim, três usuários reportaram problema na utilização do sistema, informando que algumas composições não estavam sendo executadas algumas vezes. Pelas informações passadas por eles, em conjunto com a análise dos *logs* do sistema, esse problema parece ser relacionado à compatibilidade com o navegador utilizado. Como os problemas reportados foram intermitentes e isolados a alguns usuários, eles não afetaram a avaliação do sistema.

3.5.3.4 Sugestões

Várias sugestões interessantes foram apresentadas nas respostas ao questionário. Dentre elas, podemos encontrar sugestões voltadas para melhorar a usabilidade do sistema para qualquer perfil de usuário, e sugestões para deixar a interface e a usabilidade mais atrativas para usuários avançados.

Entre as sugestões para melhorar a interface e a usabilidade do sistema estão a utilização de cores para diferenciar os parâmetros, melhoria da interface para a utilização em dispositivos móveis, adição de um passo a passo do sistema e melhoria geral do visual da

interface. Além disso, para melhorar a nomenclatura do sistema foi sugerido a alteração do nome do parâmetro “Emoção” por “Humor”. Foi sugerida também a adição de uma seção no sistema que explique melhor o funcionamento do mesmo e a forma de composição, destacando o fato de que as composições são geradas utilizando um gerador de números aleatórios e que cada composição do sistema será diferente, mesmo sem alterar os parâmetros.

Algumas funcionalidades novas também foram sugeridas, como a adição de uma opção para baixar a composição em diversos formatos, como MP3, MIDI e PDF (partitura). Também foi sugerido uma opção semelhante à “Sem Percussão”, que removeria a melodia da música. Por fim, alguns usuários sugeriram que a reprodução da música fosse contínua e que a alteração dos parâmetros fosse refletida na música gerada em tempo real.

Entre as sugestões voltadas para os usuários mais avançados, temos a possibilidade de determinar o tempo das músicas e de escolher os instrumentos individualmente, e não apenas o conjunto deles.

Vários usuários com Perfil “Iniciante” sugeriram a adição de explicações mais detalhadas dos impactos dos parâmetros nas músicas, utilizando termos de teoria musical, apresentando os instrumentos dos Conjuntos, etc. Uma das sugestões foi adicionar um “Mais Informações” nas explicações dos Parâmetros.

4 CONCLUSÃO

O presente trabalho teve por foco o desenvolvimento e avaliação de uma aplicação de composição algorítmica voltada para leigos em música, denominado Minstrel. Seu desenvolvimento foi possível através da utilização de algoritmos existentes e tecnologias recentes, bem como a utilização de boas práticas de programação e para elaboração da interface. A avaliação do sistema foi feita através da Internet, o que possibilitou a participação de vários usuários.

A partir dos resultados desses testes de avaliação, é possível concluir que o sistema Minstrel conseguiu atingir o seu objetivo de desenvolver uma interface para a realização de composições algorítmicas por leigos em música. Alguns elementos de interface, como os ícones e a descrição dos parâmetros, foram apontados como sendo de grande ajuda para a compreensão do sistema por parte desses usuários.

Dentre as possibilidades de trabalhos futuros, encontram-se a melhoria da interface, tanto em termos visuais quanto na sua usabilidade, junto com uma melhoria na nomenclatura e do conjunto de ícones utilizados. Além disso, é possível adicionar funcionalidades para dar mais controle aos usuários avançados, fazendo com que o sistema seja adequado tanto para leigos em música quanto para usuários mais experientes.

Além das melhorias na interface, existem várias melhorias que podem ser realizadas no lado do servidor. Primeiramente, o aperfeiçoamento do algoritmo de composição, implementando todas as regras presentes no ACOPM, juntamente com a implementação completa do CMERS, podem fazer com que as composições reflitam melhor os parâmetros selecionados. A utilização de uma SF comercial, ou de um conjunto de SFs, pode melhorar a qualidade da música gerada pelo sistema.

Por fim, através das observações feitas na Seção 3.5.3.3 sobre a ambiguidade do parâmetro Conjunto de Instrumentos, surge a possibilidade de alterar esse parâmetro para "Gênero Musical". A implementação desse parâmetro requer diversas alterações no sistema, e algumas sugestões em relação à implementação com sucesso desse parâmetro seriam a utilização de outros algoritmos, dependendo do gênero musical escolhido. Outra sugestão é continuar utilizando o ACOPM para composição, e realizar uma análise estatística de um conjunto de músicas de um determinado gênero, utilizando algoritmos de extração de informações de músicas.

REFERÊNCIAS

NIERHAUS, G. **Algorithmic composition: paradigms of automated music generation**. Springer Science & Business Media, 2009.

LIVINGSTONE, S. R., et al. **Changing musical emotion: A computational rule system for modifying score and performance.** Computer Music Journal 34.1 (2010): 41-64.

MILETO, E. M., et al. **Interfaces for musical activities and interfaces for musicians are not the same: the case for CODES, a Web-based environment for cooperative music prototyping.** Proceedings of the 9th international conference on Multimodal interfaces. ACM, 2007.

ELOWSSON, A., Friberg, A. **Algorithmic composition of popular music.** Royal Institute of Technology, Stockholm, Sweden (2012).

BOZHANOV, B. **Computoser-rule-based, probability-driven algorithmic music composition.** 2014. Disponível em: <http://arxiv.org/abs/1412.3079>. Acessado em: 16/09/2015

COPE, D. **Virtual music: computer synthesis of musical style**. MIT press, 2004.

KIRKE, A., MIRANDA, E. R.. **A survey of computer systems for expressive music performance**. ACM Computing Surveys (CSUR) 42.1 (2009): 3.

MIRANDA, E. R. **Evolutionary computer music**. London: Springer, 2007.

HURON, D. **Sweet anticipation: Music and the psychology of expectation**. Cambridge, MA: MIT Press, 2006

ELSEA, P. **Acoustic Instruments: A look at un-eletronic musical instruments** UCSC Electronic Music Studio, Santa Cruz, California, 1996. Disponível em: <http://artsites.ucsc.edu/ems/music/tech_background/TE-13/teces_13.html>. Acesso em 07/06/2016

UFRGS **Instrumentos Musicais** UFRGS Instituto de Física, Porto Alegre. Disponível em: <http://www.if.ufrgs.br/tex/fis01043/20032/Ismael/instrumentos_musicais.htm>. Acesso em 07/06/2016

UFRGS **Instrumentos de Cordas Acústicos** UFRGS Instituto de Física, Porto Alegre. Disponível em: <http://www.if.ufrgs.br/tex/fis01043/20032/Ismael/instrumentos_acusticos_de_cordas.htm>. Acesso em 07/06/2016

UFRGS **Instrumentos de Sopro** UFRGS Instituto de Física, Porto Alegre. Disponível em: <http://www.if.ufrgs.br/tex/fis01043/20032/Ismael/instrumentos_de_sopro.htm>. Acesso em 07/06/2016

JAZZ IN AMERICA **What is Jazz? Jazz Sounds Jazz In America**. Disponível em : <http://www.jazzinamerica.org/lessonplan/5/1/246>. Acessado em 07/06/2016

PHILARMONIA ORCHESTRA **Instruments**, Londres. Disponível em <http://www.philharmonia.co.uk/explore/instruments>. Acessado em 07/06/2016

PHILARMONIA ORCHESTRA **Violin**, Londres. Disponível em <http://www.philharmonia.co.uk/explore/instruments/violin>. Acessado em 07/06/2016

PHILARMONIA ORCHESTRA **Keyboards**, Londres. Disponível em <http://www.philharmonia.co.uk/explore/instruments/keyboards>. Acessado em 07/06/2016

PHILARMONIA ORCHESTRA **Viola**, Londres. Disponível em <http://www.philharmonia.co.uk/explore/instruments/viola>. Acessado em 07/06/2016

PHILARMONIA ORCHESTRA **Cello**, Londres. Disponível em <http://www.philharmonia.co.uk/explore/instruments/cello>. Acessado em 07/06/2016

OREGON SYMPHONY **The Piano**, Portland. Disponível em: <http://www.orsymphony.org/edu/instruments/popups/piano.html>. Acessado em 07/06/2016

MINGUS **Mingus Documentation**. Disponível em: <http://bspaans.github.io/python-mingus/>. Acessado em 10/06/2016

MARCHINI, M. **Python Mingus Fork Repository**. Disponível em: <https://github.com/mmarchini/python-mingus/tree/master_modified>. Acessado em 10/06/2016

MPDECIMAL **mpdecimal project**. Disponível em: <http://www.bytereef.org/mpdecimal/index.html>. Acessado em 10/06/2016

MIDI ASSOCIATION **GM 1 SOUND SET**. Disponível em: <https://www.midi.org/specifications/item/gm-level-1-sound-set>. Acessado em 07/06/2016

COMPUTOSER **Listen to unique, computer-generated music**. Disponível em: <http://computoser.com/>. Acessado em 10/06/2016

YAML **YAML: YAML Ain't Markup Language**. Disponível em: <http://yaml.org/>. Acessado em 12/06/2016

MUSESCORE **SoundFonts**. Disponível em: <https://musescore.org/en/handbook/soundfont>. Acessado em: 12/06/2016

ARACHNO Arachno SoundFont. Disponível em: <http://www.arachnosoft.com/main/soundfont.php>. Acessado em: 16/10/2015

LIBAV **Libav: Open source audio and video processing tools**. Disponível em: <https://libav.org/>. Acessado em: 15/06/2016

GOOGLE ANALYTICS **Google Analytics: Turn insights into actions**. Disponível em: <http://www.google.com/analytics/>. Acessado em: 15/06/2016

GOOGLE FORMS **Google Forms**. Disponível em: <https://www.google.com/forms/about/>. Acessado em: 15/06/2016

FLUIDSYNTH **FluidSynth: A SoundFont Synthesizer**. Disponível em: <http://www.fluidsynth.org/>. Acessado em: 15/06/2016

PYTHON **Python**. Disponível em: <https://www.python.org/>. Acessado em: 15/06/2016

DJANGO **Django: The web framework for perfectionists with deadlines**. Disponível em: <https://www.djangoproject.com/>. Acessado em: 15/06/2016

ANGULARJS **AngularJS - SuperHeroic JavaScript MVW Framework**. Disponível em: <https://angularjs.org/>. Acessado em: 15/06/2016

AGILE MODELING **UML 2 Component Diagrams: An Agile Introduction**. Disponível em: <http://agilemodeling.com/artifacts/componentDiagram.htm>. Acessado em: 15/06/2016

OXFORD **Oxford Advanced Learner's Dictionary**. Oxford University Press, 2010

APÊNDICE

Minstrel - Composição para Leigos em Música

Prezado(a),

Estamos lhe convidando para participar de um estudo sobre a criação de um sistema para composição automatizada chamado Minstrel, onde buscamos desenvolver um conjunto de parâmetros que possibilite a utilização do sistema por usuários leigos em música. Os parâmetros sugeridos têm por objetivo facilitar a utilização, sem eliminar completamente a capacidade do sistema de gerar músicas variadas de acordo com a entrada do usuário. Para validar esse sistema, elaboramos um procedimento de testes e um questionário, que permitirá determinar se os parâmetros desenvolvidos podem ser considerados adequados para usuários leigos em música. Durante o procedimento, solicitaremos que você realize diversas composições utilizando o sistema, ajustando os parâmetros separadamente e em conjunto, para determinar se os parâmetros atingiram seus objetivos com sucesso.

*Obrigatório

Procedimento

A participação neste estudo está dividida em duas fases:

1) Preenchimento do Formulário de Perfil, disponível na próxima página, que será utilizado para determinar o seu conhecimento sobre músicas e sua familiaridade com sistemas computacionais. O tempo médio de preenchimento é de aproximadamente 5 minutos.
2) Realização de 6 Tarefas utilizando o sistema Minstrel, o qual encontra-se disponível em http://minstrel.me/. Após a realização de cada tarefa, algumas perguntas devem ser respondidas. O tempo médio para a realização das tarefas e para responder as perguntas é de aproximadamente 15 minutos.

O tempo médio previsto para a realização de todas as atividades é de 20 minutos. A realização das atividades pode levar mais (ou menos) tempo do que o previsto. A participação será online e não há necessidade de agendamento prévio para a sua realização. Lembrando que este estudo avalia a interface do ambiente, e não o participante.

O sistema Minstrel, bem como os questionários do estudo, estarão disponíveis na internet até 11/06/2016, às 23:59, horário de Brasília.

Riscos e Desconfortos

Esses procedimentos não possuem riscos diretos aos participantes, apenas a possibilidade de cansaço para responder os questionários e utilizar o sistema. Se achar necessário, você poderá realizar descansos ou intervalos durante o procedimento.

Alternativas de participação

A participação deste estudo é voluntária. Você pode se retirar do estudo ou descontinuar sua participação se desejar.

Confidencialidade

Toda informação coletada durante o período de estudo será mantida confidencialmente. A

pessoa responsável por acompanhar a pequisa será Matheus Marchini, aluno da graduação em Ciência da Computação pela UFRGS - Universidade Federal do Rio Grande do Sul, Brasil.

Termo de Consentimento

Declaro que fui informado sobre todos os procedimentos da pesquisa, que recebi de forma clara e objetiva todas as explicações pertinentes ao projeto e que todos os dados a meu respeito serão mantidos em sigilo.
Eu compreendo que, neste estudo, a interação com o sistema e as respostas fornecidas no questionário serão utilizadas, mas que meu nome e outras informações pessoais não serão citadas ou divulgadas em nenhum momento.

1. *
   Marque todas que se aplicam.
   - [ ] Li e aceito os Termos de Consentimento

Agradecemos pela sua Colaboração!

Formulário de Perfil

2. **Nome** *
   .....
3. **Idade** *
   .....
4. **Toca ou já tocou algum instrumento?** *
   Marcar apenas uma oval.
   - ◯ Sim
   - ◯ Não
5. **Quais instrumentos você já tocou?**
   .....
6. **Você sabe interpretar partituras?** *
   Marcar apenas uma oval.
   - ◯ Não sei
   - ◯ Um pouco
   - ◯ Satisfatoriamente
   - ◯ Bem
   - ◯ Muito bem

7. **Você já compôs alguma música?** *
Marcar apenas uma oval.

- ◯ Sim
- ◯ Não

8. **Quais dispositivos computacionais você utiliza no dia a dia?** *
Marque todas que se aplicam.

- ☐ Não utilizo nenhum dispositivo computacional
- ☐ Desktop (PC, Mac, etc.)/Notebook
- ☐ Celular/Smartphone
- ☐ Tablets
- ☐ Outro: .....

9. **Quais ferramentas computacionais você utiliza ou já utilizou?** *
Marque todas que se aplicam.

- ☐ Não utilizo nenhuma ferramenta computacional
- ☐ Navegar na Internet
- ☐ Jogos
- ☐ Redes Sociais/Mensageiros instantâneos
- ☐ Criação/Edição Visual (Photoshop, Fireworks, Illustrator, etc.)
- ☐ Criação/Edição Musical
- ☐ Desenvolvimento de Software ou Hardware
- ☐ Outro: .....

10. **Marque caso você já tenha utilizado algum dos programas abaixo:**
Caso você já tenha utilizado algum programa para criação/edição de áudio. Se você utilizou algum programa não listado abaixo, você pode informar na opção "Outro"
Marque todas que se aplicam.

- ☐ Audacity
- ☐ Adobe Audition
- ☐ FL Studio
- ☐ LMMS
- ☐ SuperCollider
- ☐ PureData
- ☐ Outro: .....

Tarefa 1 - Conhecendo o Sistema

O sistema encontra-se disponível em: http://minstrel.me/
Para uma melhor experiência, sugerimos que o sistema seja acessado em computador de mesa ou notebook, utilizando os navegadores Firefox ou Google Chrome.

Nessa tarefa, você deve ler a descrição dos parâmetros que estão disponíveis no ícone "i", que encontra-se à direita do nome de cada parâmetro.

Após ler a descrição dos parâmetros, componha uma música. Para fazer isso, selecione as opções desejadas em cada parâmetro e clique no botão "Compor". Depois que a composição estiver completa, ela será reproduzida no seu navegador. Escute a música por pelo menos alguns segundos e responda a questão abaixo.

P.S.: Em nenhuma das tarefas será necessário ouvir a música até o final.

11. **A música composta reflete os parâmetros selecionados.**
Marcar apenas uma oval.

	1	2	3	4	5	
Discordo Totalmente	◯	◯	◯	◯	◯	Concordo Totalmente

Tarefa 2 - Conjunto de Instrumentos

Agora, altere o "Conjunto de Instrumentos", mantendo os outros parâmetros inalterados. Se você quiser, pode marcar/desmarcar a opção "Sem Percussão". Clique em Compor e escute a música por alguns segundos. Depois, responda as questões abaixo.

P.S.: Em nenhuma das tarefas será necessário ouvir a música até o final.

12. **Os nomes dos Conjuntos de Instrumentos são claros e objetivos.** *
Marcar apenas uma oval.

	1	2	3	4	5	
Discordo Totalmente	◯	◯	◯	◯	◯	Concordo Totalmente

13. **Os ícones dos Conjuntos de Instrumentos são claros e objetivos.** *
Marcar apenas uma oval.

	1	2	3	4	5	
Discordo Totalmente	◯	◯	◯	◯	◯	Concordo Totalmente

14. **A descrição do parâmentro "Conjunto de Instrumentos" é clara e objetiva.** *
Marcar apenas uma oval.

	1	2	3	4	5	
Discordo Totalmente	◯	◯	◯	◯	◯	Concordo Totalmente

Tarefa 3 - Emoção

Altere a "Emoção", mantedo os outros parâmetros inalterados, e clique novamente em Compor. Escute a música por alguns segundos, e depois responda as questões abaixo.

P.S.: Em nenhuma das tarefas será necessário ouvir a música até o final.

15. **Os nomes das Emoções são claros e objetivos.** *
Marcar apenas uma oval.

	1	2	3	4	5	
Discordo Totalmente	◯	◯	◯	◯	◯	Concordo Totalmente

16. **Os ícones das Emoções são claros e objetivos.** *
Marcar apenas uma oval.

	1	2	3	4	5	
Discordo Totalmente	◯	◯	◯	◯	◯	Concordo Totalmente

17. **A descrição do parâmentro "Emoção" é clara e objetiva.** *
Marcar apenas uma oval.

	1	2	3	4	5	
Discordo Totalmente	◯	◯	◯	◯	◯	Concordo Totalmente

Tarefa 4 - Complexidade

Altere a "Complexidade", mantedo os outros parâmetros inalterados, e clique novamente em Compor. Escute a música por alguns segundos, e depois responda as questões abaixo.

P.S.: Em nenhuma das tarefas será necessário ouvir a música até o final.

18. **Os ícones das Complexidades são claros e objetivos.** *
Marcar apenas uma oval.

	1	2	3	4	5	
Discordo Totalmente	◯	◯	◯	◯	◯	Concordo Totalmente

19. **A descrição do parâmentro "Complexidade" é clara e objetiva.** *
Marcar apenas uma oval.

	1	2	3	4	5	
Discordo Totalmente	◯	◯	◯	◯	◯	Concordo Totalmente

Tarefa 5 - Duração

Altere a "Duração", mantedo os outros parâmetros inalterados, e clique novamente em Compor. Escute a música por alguns segundos, e depois responda as questões abaixo.

P.S.: Em nenhuma das tarefas será necessário ouvir a música até o final.

20. **Os ícones das Durações são claros e objetivos.** *
Marcar apenas uma oval.

	1	2	3	4	5	
Discordo Totalmente	◯	◯	◯	◯	◯	Concordo Totalmente

21. **A descrição do parâmentro "Duração" é clara e objetiva.** *
Marcar apenas uma oval.

	1	2	3	4	5	
Discordo Totalmente	◯	◯	◯	◯	◯	Concordo Totalmente

Tarefa 6 - Conclusão

Por fim, altere todos os parâmetros conforme a sua preferência, e então clique em Compor. Escute a música por alguns segundos, e depois responda as questões abaixo.

P.S.: Em nenhuma das tarefas será necessário ouvir a música até o final.

22. **De maneira geral, foi possível compreender a influência que cada um dos parâmetros tem na execução do sistema?** *

.....

.....

.....

.....

.....

23. **Cite pelo menos 2 pontos positivos que você identificou no programa Minstrel.** *

.....

.....

.....

.....

.....

24. **Cite pelo menos 2 pontos negativos que você identificou no programa Minstrel.** *

.....

.....

.....

.....

.....

25. **Quais suas sugestões para que os parâmetros sejam mais claros?** *

.....

.....

.....

.....

.....

26. **Quais suas sugestões para que a utilização do sistema fique mais simples?** *

.....

.....

.....

.....

.....