Universidade Estadual de Campinas
Faculdade de Tecnologia

Marcelo Augusto Stefanini Faria

# Sheet Gen

## Software para criação de partituras – Relatório TCC I

Limeira

2019

Universidade Estadual de Campinas
Faculdade de Tecnologia

Marcelo Augusto Stefanini Faria

# Sheet Gen

## Software para criação de partituras - Relatório TCC I

Trabalho de Conclusão de Curso I apresentado à Faculdade de Tecnologia da UNICAMP, para obtenção do título de Bacharel em Sistemas de Informação, sob a orientação do Prof. Dr. João Roberto Bertini Junior.

Limeira

2019

# Resumo - Sheet Gen

Este trabalho tem como objetivo resolver de um dos grandes problemas no universo da música: a escrita de partituras.

Existem máquinas que escrevem partituras, assim como as nossas conhecidas máquinas de escrever textos, porém as dificuldades para realizar esses trabalhos e os problemas que surgem quando um erro é cometido são tão grades que, com grande ímpeto, os programas de computadores tomaram esse lugar e hoje em dia são as principais formas de escrever.

Existem diversos softwares para escrever partituras, porém além de se apresentarem muito difíceis de usar, requerirem muita experiência e prática para que se escreva rápido, ainda são muito lentos.

A idealização desse software se dá justamente na tentativa de vencer essas grandes barreiras utilizando as mais altas tecnologias disponíveis no mercado de informática.

A escrita das partituras será feita a partir da execução de um músico. O sistema, através de diversas análises, entende o que está sendo executado e retorna o produto final: a partitura.

# Sheet Gen

## 1. Versão do Documento

- Versão 1.4

## 2. Introdução

O sistema de computadores Sheet Gen é capaz de imprimir partituras através da captação de sons pelo microfone do computador. Isso é possível graças a diversos procedimentos de análise de dados que, trabalhando juntos, conseguem retornar exatamente o que foi executado pelo músico.

Utilizando Inteligência Artificial, Aprendizado de Máquina e diversos outros algoritmos, Sheet Gen consegue entregar partituras com alta confiabilidade e precisão além de alta performance em escrevê-las em tempo real enquanto o músico executa seu instrumento.

## 3. Linguagem de programação

- Python 3.7

## 4. Bibliotecas a utilizar

- pyaudio
- numpy
- matplotlib.pyplot
- pandas
- plotly
- pydot
- turtle.py
- scikit-learn
- pil
- pygameintro
- time

# 5. Algoritmos a utilizar – Must Have

## 5.1. Funcionalidades Básicas

### 5.1.1. Captar áudio pelo microfone

Uma das premissas básicas do sistema é que o produto final, que é a partitura, seja gerada a partir da música que um músico executar. Para que isso seja possível é necessário utilizar algum componente que seja capaz de captar ondas sonoras. A melhor forma de se resolver esse problema é utilizando um microfone de boa qualidade e, de preferência, utilizar o sistema em um ambiente calmo, sem barulho externo.

### 5.1.2. Plotar ondas sonoras em gráfico

Plotar o gráfico das ondas sonoras captadas pelo microfone é um passo importante para a execução desse trabalho, pois com ele será possível identificar as notas através de sua frequência em Hertz (Hz). A exibição do gráfico em tempo real será fundamental para auxiliar no desenvolvimento e é um recurso estético que considero muito importante para compor as telas do sistema final.

- Identificar melhor formato para exibição de ondas.
- Avialiar o impacto de performance que isso causará no sistema

### 5.1.3. Implementar a taxa de atualização do gráfico de ondas sonoras

Um dos grandes desafios desse item é que quanto maior a taxa de atualização, mais nítido, fluido e legível fica o gráfico, porém maior o custo computacional. Então, implementar a possibilidade do usuário mudar a taxa de atualização seria interessante para que o sistema possa ser rodado em computadores mais simples. Neste caso, é muito importante atribuir um limite de segurança para evitar quebras no sistema.

- Identificar melhor taxa para atualização das ondas.

### 5.1.4. Retornar valor em Hertz (Hz) de uma posição do gráfico

Para todo som que é emitido, existe uma referência para ele dada em Hertz.

O ouvido humano, por exemplo, é capaz de captar frequências a partir de 20 Hz que é o mais grave, até 20.000 Hz que é o mais agudo.

Logo, o sistema capta as ondas sonoras, e de acordo com a altura*, consegue determinar qual é a nota que foi entoada e qual foi sua oitava*

*altura: Propriedade do som que nos permite distinguir sons graves de agudos.

*oitava: Intervalos pré-definidos entre notas musicais de mesmo nome.

### 5.1.5. Identificar cristas do gráfico de ondas sonoras

Para que seja possível ter a maior precisão possível ao captar uma nota, o correto é identificá-la pela crista da onda que ela gera. Quanto mais para a crista, menor é a variação de espaço no eixo X do gráfico, ou seja, maior precisão no resultado.

### 5.1.6. Retornar valor da frequência em Hertz (Hz) da crista identificada

Tendo o algoritmo de identificação de cristas robusto, é necessário captar esse valor do eixo X que será dado em Hertz para se fazer os devidos processamentos.

### 5.1.7. Identificar mudança de crista

Existem vários desafios para que a identificação seja feita com sucesso.

Primeiro porque as notas podem ser muito próximas, ou seja, com frequências muito parecidas. Então é necessário prever esses cenários com antecedência para que o sistema não confunda uma mudança de nota com uma, por exemplo, oscilação que pode ocorrer durante a execução do músico. Será necessário treinar o sistema com as variações

O sistema com certeza terá que lidar com casos de ruído que podem interferir na boa compreensão dele e deverá tomar atitudes para fazer a tomada de decisão: escrever o que foi captado e avisar o usuário sobre a baixa confiabilidade do trecho para que seja possivelmente corrigido posteriormente

### 5.1.8. Retornar o tempo da mudança de uma crista para outra

Falamos até agora sobre como identificar o nome e a oitava da nota, porém para que o sistema consiga escrever na partitura, é indispensável saber por quanto tempo a nota foi tocada. A princípio o sistema contará com a ajuda do usuário: antes de começar a tocar alguma coisa, o músico deverá dizer ao sistema em que andamento* e em qual fórmula de compasso* ele irá tocar, além da armadura de clave*.

Tendo essas informações será possível categorizar através de computações e cálculos feitos pelo sistema quais notas deverão ser escritas de acordo com a quantidade de tempo que elas aconteceram durante a execução. Quando uma nota for tocada, um contador de tempo será disparado e só será interrompido quando o sistema detectar uma mudança de crista, determinando assim qual será o símbolo musical para atribuição na partitura. Ao mesmo tempo em que for interrompido, esse contador imediatamente será disparado novamente para que a próxima nota seja contabilizada e assim por diante.

*andamento: o conceito de batidas por minuto (bpm) define o andamento de uma música. Basicamente um “de-para” que diz respeito à velocidade em que se toca.

*fórmula de compasso: maneira de se dividir uma música em quantidades pré-definidas para que seja possível organizá-la em pequenos grupos a fim de deixá-la da forma mais legível possível

*armadura de clave: conjunto de acidentes musicais que determinam a tonalidade*

*tonalidade: unidade de medida para diferenciação das notas

## 5.2. Funcionalidades Intermediárias

### 5.2.1. Implementar sistema de configuração inicial

O sistema de configuração inicial deve contar a princípio, com opções mandatórias para o usuário selecionar. Para a primeira versão do sistema, não seria possível continuar sem elas, visto que para que essas informações fossem obtidas sem o auxílio do usuário, seria necessário aplicar conceitos de Inteligência Artificial e também, de preferência, Aprendizado de Máquina para funcionar corretamente.

Então, para o primeiro desenvolvimento do sistema, o usuário irá atribuir essas informações manualmente na partitura e somente depois disso estará apto para começar a executar.

### 5.2.2. Normalizar ondas sonoras no que se diz respeito à intensidade do som

É muito importante que o sistema consiga tratar casos em que a captação esteja muito alta. O músico pode não perceber que o microfone está muito próximo, ou que o instrumento esteja alto demais.

Para estes casos, o sistema deve normalizar as ondas sonoras para que não perca a precisão enquanto captura o áudio.

### 5.2.3. Criar templates das páginas da partitura

Como o Sistema terá que fazer diversas computações em tempo real, é necessário diminuir sua carga de processamento em cada detalhe o quanto for possível. Uma maneira de fazer isso é deixar algumas atribuições prontas, pelo menos aquelas que não sejam variáveis.

Isso depende de como a biblioteca que vai renderizar os pentagramas* funciona. A ideia principal é deixar tudo o que for possível pronto. Porém, se o custo computacional for baixo, é possível renderizar na hora.

*pentagrama: conjunto de 5 linhas e 4 espaços onde as notas são inseridas.

### 5.2.4. Criar as figuras musicais

Duas opções estão cotadas para esse passo. A primeira e mais simples é baixar os símbolos musicais vetorizados da internet ou desenhar em algum sistema que permita. Porém existem bibliotecas em Python para desenhar e, dependendo da complexidade de código para criá-las, vale considerar a possibilidade caso a complexidade seja baixa.

### 5.2.5. Relacionar os conceitos musicais com as saídas das funcionalidades básicas

Conceito geral de como vai funcionar o motor de decisão do Sheet Gen. As regras estarão implementadas no sistema e ele deverá retornar as saídas de acordo com as entradas.

### 5.2.6. Converter a partitura gerada para .pdf

Após sua execução o sistema converterá a partitura gerada no formato PDF.

## 5.3.Funcionalidades Avançadas

### 5.3.1. Gerar a partitura em tempo real

Otimizar ao máximo para que seja o mais instantâneo possível. Isso fará com que esse passo seja, com certeza, o de maior custo computacional.

### 5.3.2. Integração de bibliotecas

Será necessário fazer as bibliotecas conversarem não só para que seja possível ter a melhor performance, como para chegar em resultados que talvez sejam impossíveis de alcançar utilizando um número limitado delas.

#### 5.3.2.1. Bibliotecas de desenho

Uma biblioteca de plotagem de gráficos seria uma excelente opção para desenhar um pentagrama, por exemplo, ao mesmo tempo que bibliotecas desenho mais específicas seriam ideais para os símbolos músicais.

#### 5.3.2.2. Bibliotecas de conversão

Como possivelmente várias bibliotecas vão compor o produto final do sistema, talvez seja necessário utilizar alguns componentes do sistema operacional além das bibliotecas de conversão para que seja possível entregar a partitura em .pdf.

# 6. Algoritmos a utilizar – Nice to Have

## 6.1.Funcionalidades Gerais

**Não estão em ordem de prioridade!**

### 6.1.1. Implementar animação indicando o progresso da música

Existem algumas bibliotecas em python que facilitam fazer a animação. A ideia não é fazer algo muito complexo. Somente uma espécie de cursor que percorre o pentagrama enquanto a partitura for gerada já seria o suficiente, a princípio.

### 6.1.2. Implementar animação para a plotagem dos desenhos

Para este caso, seria interessante implementar animações para as inserções das notas e plotagem dos desenhos para que fique mais bonito e fluido. Porém é uma ideia que deve ser implementada após o término do projeto.

### 6.1.3. Identificar várias melodias ao mesmo tempo

A proposta inicial do sistema é que ele seja capaz de escrever a partitura de somente uma melodia por vez. Porém, através de observações do gráfico sonoro, quando as notas são tocadas simultaneamente é possível identificar duas ou mais cristas ao mesmo tempo para aquela oitava em específico, por conseguinte, suas respectivas frequências resultando nas notas.

As ondas sonoras não se mistruram, então é possivel identificá-las individualmente. Com isso seria possível escrever, na mesma partitura, as várias melodias tocadas por um ou mais músicos. Esforços para resolver esse item:

- Identificar várias cristas do gráfico de ondas sonoras
- Retornar valor da frequência em Hertz (Hz) das cristas identificadas
- Identificar várias mudanças de cristas
- Retornar valor em milisegundos ou segundos das mudanças de várias cristas

### 6.1.4. Animação de load equanto o sistema gera o .pdf da partitura

Computar com antecedência o esforço que o sistema fará para gerar o pdf e calcular o impacto de cada uma individualmente. Tendo os números, posteriormente, a cada passo que ele executar, aumentar a barra de progresso de acordo com a porcentagem do total que cada passo exige do sistema.

### 6.1.5. Possibilitar o usuário fazer correções na partitura gerada

Após terminar a execução de uma seção, o usuário poderá corrigir alguns erros que o sistema pode ter cometido e também conferir as confiabilidades para cada nota.

### 6.1.6. Gerenciar as informações da partitura e metadados

Implementar um sistema que consiga além de escoher um nome para o arquivo da partitura e escrever no template dela, editar os metadados do arquivo que for gerado para aquela seção, considerando os seguintes itens:

- Nome do Arquivo
- Título da partitura
    - Subtítulo
- Autor
- Direitos autorais
- Descrição e etc.

### 6.1.7. Possibilitar a inserção de letra na partitura

Algumas músicas possuem letras. Seria interessante, ao abrir o template para o usuário corrigir os erros, deixá-lo apto para inserir a letra da música na partitura.

### 6.1.8. Implementar conceitos de Machine Learning

É muito importante que a cada vez que o sistema for usado, melhore sua precisão ao gerar as partituras e isso é possível com aprendizado de máquina. Alguns itens importantes que devem ser considerados para a implementação dessa ideia:

- Treinamento do sistema com algumas situações mais complexas que ele pode se deparar
- Treinamento de correção do sistema com situações em que esteja gerando a partitura de forma incorreta
- Criação de um sistema de confiança com porcentagens de 0% a 100% para cada nota e, ao final da execução, ressaltar os pontos mais críticos.

- Guardar na memória tudo o que foi executado para que, depois que o usuário parar de executar, verificar se existe alguma melhoria que pode ser feita através do que o computador já aprendeu. Como esse item pode levar algum tempo a mais de processamento, para que não impacte na performance da partitura em tempo real, é recomendável que seja feito após o processamento.

#### 6.1.9. Implementar sistema de aviso

Se trata de um sistema que detecta se o local está apropriado ou não para gerar partituras. Ele deve funcionar da seguinte forma: se o microfone estiver detectando muito ruído, ou seja, se estiver impossível para o sistema chegar em conclusões plausíveis para gerar a partitura, ele exibirá um aviso dizendo que o ambiente está inapropriado para executar o sistema.

#### 6.1.10. Implementar conceitos de IA

Utilizando conceitos de Inteligência Artificial, o sistema será capaz de reconhecer o andamento & bpm ( batidas por minuto ), fórmula de compasso e armadura de clave, dispensando assim as configurações iniciais para que o sistema seja executado.

# 7. Tarefas importantes

## 7.1. Categorizar timbres

Analisar as ondas sonoras para que o sistema seja capaz de identificar diferentes instrumentos.

## 7.2. Definir um instrumento ideal para o desenvolvimento

Definir um tipo de instrumento para auxiliar os testes. A princípio: Piano ou qualquer instrumento que não oscile quando executado.

## 7.3. Organizar a exibição da interface gráfica em toda sua execução

### 7.3.1. Tela inicial

### 7.3.2. Tela de execução de uma seção

### 7.3.3. Tela de encerramento de uma seção

# 8. Conclusão

A partir dos itens listados é possível chegar na conclusão que o sistema é factível principalmente porque os maiores esforços de programação serão auxiliados pelas vastas e específicas bibliotecas já existentes em python.

O sistema resolverá um problema comum no mundo da música e será muito útil nos dias de hoje, já que a tendência é digitalizar cada vez mais o mundo real.