Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

CybrusJam: software de acompanhamento musical para baixistas

Vitor Moraes Dellamora

Monografia apresentada como requisito parcial
para conclusão do Bacharelado em Ciência da Computação

Orientador
Prof. Dr. Carlos Eduardo Vianna de Mello

Brasília
2022

Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

CybrusJam: software de acompanhamento musical para baixistas

Vitor Moraes Dellamora

Monografia apresentada como requisito parcial
para conclusão do Bacharelado em Ciência da Computação

Prof. Dr. Carlos Eduardo Vianna de Mello (Orientador)
MUS/UnB

Prof.a Dr.a Carla Denise Castanho
CIC/UnB

Prof. Dr. Wilson Henrique Veneziano
CIC/UnB

Prof. Dr. Marcelo Grandi Mandelli
Coordenador do Bacharelado em Ciência da Computação

Brasília, 05 de maio de 2022

Dedicatória

Dedico este projeto à minha família, por sempre terem me influenciado a ter contato com a música e pela inspiração (Saibrus) ao nome de batismo deste projeto.

Agradecimentos

Agradeço à minha família que sempre me proporcionou o melhor para mim e sempre me permitiu ter acesso às melhores instituição de ensino, inclusive desde o ensino médio quando me incentivaram a estudar no Colégio Pedro II, onde começou minha paixão definitiva pela programação. Agradeço ao meu ex-professor do primeiro ano, João Calvano, por ter sido o primeiro professor a me ensinar a programar (em portugol). Agradeço ao professor Carlos Eduardo Mello por não só me provar que é possível trabalhar com programação e criação musical, como essas duas áreas são inseparáveis por meio da composição algorítmica e também por ter me orientado nesse trabalho ardiloso e pelas lapidadas necessárias na minha ideia original de projeto de graduação. Agradeço ao aluno de composição do Departamento de Música da Universidade de Brasília, José Bacelar de Abreu, pela grande contribuição com o conversor de áudio em MIDI no PureData; sem isso, o projeto seria impossível. Agradeço aos jogos da From Software, idealizados por Hidetaka Miyazaki, pois sem eles eu não teria mantido minha sanidade durante esses anos finais de faculdade.

Resumo

Uma das demandas constantes na rotina de um músico é a necessidade de prática individual para refinar suas técnicas. Esta monografia detalha o desenvolvimento de um *software* de acompanhamento musical para contrabaixistas no gênero *heavy metal*. O programa utiliza-se de conceitos da composição algorítmica em conjunto com uma biblioteca de C++ e orientação a objetos para gerar um acompanhamento musical em sinais MIDI a fim de trazer uma abordagem diferente para este tipo de *software*: em tempo real, captar o que o músico toca para gerar um material musical correspondente sem precisar de nenhuma configuração prévia de progressões harmônicas. Os resultados dos testes com o *software* mostram que ele demonstra bom desempenho na geração de acompanhamento.

Palavras-chave: composição algorítmica, contrabaixo elétrico, heavy metal, software de acompanhamento musical

Abstract

One of the constant demands in the routine of a musician is the necessity of individual practice in order to refine technical skills. This monograph details the development of a musical accompaniment software made for bass players in the heavy metal genre. The program uses concepts from algorithmic composition alongside a C++ library and object orientation to generate a musical accompaniment with MIDI signals. It brings a different approach to this kind of software: gathering, in real time, whatever the musician plays to generate a suitable accompaniment without needing any previous setting of harmonic progression. Test results show that the software is capable of satisfactory performance in generating musical accompaniment.

Keywords: algorithmic composition, electric bass, heavy metal, musical accompaniment software

Sumário

Lista de Figuras

Capítulo 1

Introdução

O presente trabalho propõe o desenvolvimento de um software que realize o acompanhamento musical automatizado para a prática do contrabaixo elétrico em um conjunto de metal.

1.1 Problema

A prática de música em conjunto, qualquer que seja o gênero musical, pressupõe a reunião dos membros do grupo em sessões de ensaio, onde se aprende o repertório do conjunto e se trabalha a interação entre os músicos, de modo a produzir um resultado musical satisfatório. Além da prática em conjunto, um músico frequentemente sente a necessidade de praticar individualmente para aprimorar sua técnica e se preparar melhor para as sessões coletivas com o grupo. Nesse contexto, é possível enriquecer a experiência musical do(a) instrumentista, fornecendo algum tipo de acompanhamento automatizado para servir de contexto/referência para o desenvolvimento da prática.

1.2 Hipótese

Embora já existam no mercado algumas opções de aplicativos para acompanhamento musical[1][2][3][4], a maior parte desses sistemas baseiam o acompanhamento em progressões harmônicas pré-definidas ou carregadas a partir de arquivos. Acreditamos que muitos instrumentistas possam se beneficiar de um sistema especializado que explore as características musicais específicas do gênero escolhido e ofereça a possibilidade de construir

[1] `https://www.bandinabox.com/`
[2] `https://www.jjazzlab.com/en/`
[3] `https://github.com/jsrmath/sharp11`
[4] `https://www.1manband.nl//`

o acompanhamento em cima de materiais musicais fornecidos pelo(a) próprio(a) instrumentista, no ato da prática.

1.3 Objetivo

Com esse propósito, este projeto propõe a definição de um sistema no qual o músico, no caso contrabaixista, inicie a prática tocando uma linha de baixo para que o sistema analise e defina uma progressão harmônica compatível com as notas tocadas, conforme as características do gênero metal. A partir dessa análise, então, o sistema deve iniciar a execução de uma sequência musical com sons de instrumentos típicos do gênero metal, de modo que o(a) contrabaixista possa tocar junto. Uma vez que a voz grave do baixo funciona frequentemente como base para a harmonia e o ritmo em grande parte dos gêneros musicais ditos populares, incluindo o metal, objeto do presente estudo, essa abordagem parece oferecer boas possibilidades para o desenvolvimento de um sistema efetivo de acompanhamento para os(as) praticantes desses instrumentos.

O projeto faz o uso de técnicas de composição algorítmica para a construção das vozes de acompanhamento e o uso de orientação a objeto para a construção de todo o sistema. Desse modo espera-se poder expandir mais facilmente, em futuras iterações do desenvolvimento, com a adição de novos componentes instrumentais, detalhes estilísticos e aperfeiçoamento da análise e composição do acompanhamento.

1.4 Metodologia

A metodologia deste projeto consistiu em etapas de pesquisa, desenvolvimento e testes. Inicialmente, foi feita uma pesquisa sobre o estilo *heavy metal* e que tipo de forma de execução de guitarras e baterias fazem parte desse gênero. Após isso, foram buscados exemplos de *softwares* parecidos com este que o projeto propõe, no intuito de entender o funcionamento deste tipo de ferramenta. Seguindo a etapa do desenvolvimento, foi feita uma versão limitada do programa, para que fossem aplicados os conhecimentos obtidos na etapa de pesquisa e os conhecimentos adquiridos da composição algorítmica e só então foi desenvolvida a versão completa do projeto apresentado por esta monografia. Finalmente, testes foram feitos para avaliar o quanto a ferramenta desenvolvida pode resolver o problema descrito anteriormente. Esses testes consistiram em realizar sessões de improvisação utilizando o CybrusJam e destacando a reação do usuário ao apresentar o acompanhamento gerado e se ele ficou satisfeito com tal resultado.

1.5 Conteúdo do Documento

Esta monografia apresenta a seguinte estrutura: o primeiro capítulo descreve o contexto, objetivos e metodologia do projeto. No capítulo seguinte é feita uma breve explicação de cada um dos conceitos teóricos utilizados no projeto, incluindo composição algorítmica, a biblioteca de materiais musicais MuM, o protocolo de sinais digitais MIDI, e uma descrição pragmática sobre acompanhamento em uma banda do gênero *heavy metal*. No terceiro capítulo, o projeto CybrusJam é explorado de modo geral: como o ambiente de testes foi montado para o funcionamento do programa, o que o usuário deve esperar do fluxo de uso do programa e como utilizá-lo e, por fim, a construção do projeto, detalhando como é feito o código que faz o programa rodar, passando por todas as classes e etapas de execução. No último capítulo, são apresentados resultados do projeto e seus testes, assim como considerações finais e futuras melhorias que podem ser adicionadas após o término deste projeto.

Capítulo 2

Fundamentação Teórica

Neste capítulo, os assuntos teóricos tratados no projeto são descritos e é feita uma explicação de como eles foram utilizados no desenvolvimento do programa de acompanhamento musical CybrusJam. Em ordem, são tratados a área de conhecimento composição algorítmica, a biblioteca MuM, o padrão MIDI e o modelo de um acompanhamento musical em uma banda de metal.

2.1 Conceitos de Composição Algorítmica

A composição algorítmica é uma área da computação musical que estuda o emprego de sistemas computacionais para gerar processos de criação musical[1]. Essa área tem se desenvolvido num campo muito vasto que inclui desde a exploração de processos totalmente externos à música, como o mapeamento de funções matemáticas a parâmetros de síntese sonora, até tentativas de reprodução do estilo de grandes compositores da história. Frequentemente empregam-se técnicas de inteligência artificial, muitas vezes com foco em sistemas especializados, baseados em conhecimento aprofundado da linguagem musical e das tendências estéticas de cada gênero ou estilo de música. Outra tendência nessa área é a tentativa, por parte de diversos compositores, de reproduzir através da programação de computadores, suas próprias ideias, técnicas e estéticas, na tentativa de expandir e diversificar sua produção musical através de processos computacionais.

Qualquer que seja a abordagem utilizada, alguns elementos comuns se destacam, em particular a natureza aberta do problema da composição musical. Uma vez que não existe uma única solução para a criação musical, sistemas dessa natureza são muitas vezes validados, na prática, pela percepção estética dos resultados sonoros, dentro de um estilo ou gênero musical definido.

Neste contexto, o problema proposto pelo presente projeto parece se configurar bastante bem como um problema de composição algorítmica. A performance instrumental

em diversos tipos de grupos de música popular se baseia na combinação de diversas texturas produzidas pelos instrumentos, com base em uma fundação harmônica em comum. Aqui essa base da harmonia musical é fornecida por um processo de análise em cima de materiais melódicos apresentados pelo próprio instrumentista. A partir dessa estrutura, o programa precisa produzir uma realização musical norteada pelo vocabulário típico de cada instrumento do grupo, de modo que o resultado seja unificado pela estrutura analisada anteriormente e ao mesmo tempo apresente o tipo de sonoridade esperada dentro do gênero proposto

2.2 Biblioteca MuM

Embora, como vimos anteriormente, as abordagens de composição algorítmica possam ser bastante diversas, muitas vezes é possível encontrar elementos em comum que permitem a reutilização de algoritmos e estruturas. Este é o caso, por exemplo, de sistemas de composição baseados em construções da linguagem musical tradicional, tais como notas, acordes, escalas, etc. Nesses casos muitas abstrações utilizadas na música podem ser modeladas de modo a facilitar o trabalho envolvido na criação de um sistema automatizado de composição.

No caso do projeto desenvolvido neste trabalho, foi utilizada a Biblioteca MuM (Music Material) [2] para fornecer essa infraestrutura musical básica. Essa biblioteca foi desenvolvida no Laboratório de Tecnologia Musical da UnB (LTM/MUS) em 2009, tendo sido utilizada em diversos projetos de desenvolvimento de software musical desde então. Ela é escrita na linguagem C++, explorando extensivamente os recursos de orientação a objeto da linguagem, criando novos tipos de dados e sobrecarregando funções e operadores, para ajudar o compositor a implementar algoritmos musicalmente interessantes, de forma intuitiva e com pouco código.

A Biblioteca MuM é baseada no conceito de "Material Musical", um termo muito usado na composição tradicional para designar qualquer agrupamento de notas musicais que possa servir de base para gerar novos materiais e assim construir um texto musical completo. Nesta biblioteca, um material musical é uma classe de objetos que podem ser manipulados de diversas formas, armazenando, modificando e produzindo qualquer conteúdo musical que possa ser descrito como combinações de notas musicais organizadas no tempo.

2.2.1 MuMaterial

A classe MuMaterial é a mais importante da biblioteca. Com ela são realizadas todas as operações básicas de construção dos algoritmos de composição. Um objeto da classe

MuMaterial pode ser usado como uma caixa preta, na qual são armazenadas informações musicais para cópia, edição e transformações diversas. Também é possível explorar a estrutura interna do objeto para auxiliar na organização e manipulação dos elementos musicais. Essa estrutura se baseia na distribuição de notas musicais em subdivisões independentes chamadas vozes, em referência às linhas melódicas encontradas em uma partitura musical tradicional. Cada uma dessas vozes internas do material é modelada por uma classe chamada MuVoice, a qual consiste basicamente de uma lista encadeada de notas (objetos da classe MuNote) associada a diversos métodos de transformação das mesmas. As notas são armazenadas em ordem cronológica em cada voz. Dentro de um material, as vozes são numeradas a partir de 0 e representam linhas paralelas no tempo, de modo que quando um material é tocado, as notas das diversas vozes soam simultaneamente, como em um programa de gravação de áudio com múltiplas trilhas. Quando, por exemplo, uma operação de atribuição ou cópia é realizada sem a indicação de um número de voz específico, todas as notas são inseridas na voz 0. Isso permite tratar o material como um *container* opaco para operações rápidas. Mas se, por outro lado, forem designados números de voz específicos, o material pode ser tratado como uma estrutura de várias camadas ou trilhas, que podem ser acessadas separadamente para organizar e manipular os elementos musicais mais facilmente. Além de notas, vozes, e materiais, diversas outras classes de objetos fornecem funcionalidades que completam a infra-estrutura da biblioteca. Objetos dessas classes são contidos uns dentro dos outros, como mostram as descrições abaixo:

- MuError: classe de comunicação de erros nas operações da biblioteca. Todo material possui um objeto dessa classe que armazena o erro gerado pela última operação realizada.
- MuMIDIMessage: estrutura usada para representar um evento MIDI independente. Essas estruturas são geradas como saída da classe MuNote para converter as informações das notas em mensagens MIDI.
- MuMIDIBuffer: *container* para transferir blocos de eventos MIDI; usado pelas classes MuRecorder e MuPlayer na entrada e saída de informações em um material.
- MuParamblock: vetor de valores contidos nos objetos da classe MuNote para transportar parâmetros opcionais de performance com Csound
- MuNote: define um evento sonoro discreto, unidade básica para a especificação das informações musicais.
- MuVoice: lista de objetos da classe MuNote; contida nos objetos da classe MuMaterial.

- MuMaterial: contem um *array* dinâmico de objetos MuVoice; utilizado como entrada de dados para a classe MuPlayer; pode receber MuMIDIBuffers da classe MuRecorder.
- MuRecorder: realiza a entrada assíncrona de dados oriundos de um dispositivo MIDI do sistema e produz *buffers* de MIDI para objetos da classe MuMaterial.
- MuPlayer: recebe objetos da classe MuMaterial, converte a informações do material em *buffers* MIDI e envia os dados MIDI de forma assíncrona para dispositivos MIDI (sintetizadores) disponíveis no sistema.

Em suas primeiras implementações, a Biblioteca MuM produzia apenas saídas no formato de partituras para o programa Csound, que podiam ser ouvidas através deste programa, de forma separada ou por de uma chamada de sistema embutida no próprio código da biblioteca. Isso permitia ao usuário ouvir o resultado sonoro dos programas e manter a biblioteca totalmente multiplataforma, podendo ser utilizada em qualquer sistema com um compilador padrão de C++. Esta forma de *playback*, entretanto, era feita apenas de forma síncrona. Em 2016, com o projeto MuMRT, foram acrescentadas as classes de entrada e saída MIDI descritas acima (MuRecorder e MuPlayer), expandindo a biblioteca com a possibilidade de *playback* em tempo real e em qualquer parte do código. Esses novos recursos limitaram, porém, a facilidade de adaptação para as diversas plataformas. Atualmente a biblioteca pode ser usada com *playback* MIDI no sistema MacOS (CoreMIDI) e no Linux (RtMidi)

2.3 MIDI

MIDI é um padrão de sinais digitais criado no início da década de 80 por uma série de fabricantes de sintetizadores musicais. É uma comunicação serial em tempo real utilizada em controladores (como teclados musicais), computadores, sintetizadores de áudio e outros dispositivos musicais que envia, em tempo real, mensagens de 24 bits para tocar e silenciar notas, alterar instrumentos, ativar certos sinais de controle, entre outras utilidades. A sigla MIDI significa interface digital para instrumento musical (*Musical Instrument Digital Interface*)[3][4].

O padrão MIDI conta com 16 canais que podem ser tocados independentemente. A cada canal pode ser atribuído um instrumento diferente e cada um terá um conjunto de instruções a ser executado que diz respeito à ativação e desativação de notas em determinados momentos; não é necessário que todos os canais estejam preenchidos. Apesar dessa definição, o MIDI por si só não define o timbre de um instrumento associado a um canal, isso depende do sintetizador que recebe o conjunto de instruções para tocar

as notas (no caso desse projeto, o programa Timidity). Cada nota em MIDI possui uma altura correspondente a um número inteiro que vai de 0 a 127, sendo 60 o dó central e uma amplitude (volume) que vai de 0 a 127.

Por convenção, o décimo canal é reservado para instrumentos de percussão, sendo cada nota um componente diferente (desde bumbos, caixas, tons, pratos até outros tipos de percussão como reco-recos, triângulos, agogôs, bongôs, etc.) e a mudança de instrumento nesse canal provoca uma mudança completa de quais componentes estarão presentes para serem tocados.

Neste projeto, o MIDI é utilizado tanto para recebimento de instruções como para envio dos sinais. Os sinais MIDI do baixo são lidos para determinar a forma de execução e quais notas serão executadas. O projeto monta uma trilha de guitarra e uma trilha de bateria para serem tocados pelo sintetizador MIDI Timidity, como acompanhamento para o usuário.

Para poder trabalhar com MIDI, foi construído um conversor de áudio digital para sinais MIDI feito em PureData [5] que é uma linguagem de programação visual feita para tratamento e geração de áudio digital. A programação é feita através de blocos de código (literalmente), onde cada bloco possui um identificador, seus parâmetros, entradas destacadas em cima e saídas destacadas em baixo, dependendo de qual componente ele representa.

2.4 Acompanhamento em uma banda de Metal

O metal é um gênero musical derivado do *rock'n'roll* e surgiu no final dos anos 60 com bandas como Black Sabbath e Judas Priest [6]. As características marcantes desse gênero são o som pesado de guitarras distorcidas, baterias aceleradas e vocais poderosos, elementos que se aplicam a vários subgêneros do metal, porém não todos. Uma banda de metal é geralmente composta por um baterista, um baixista, um conjunto de guitarristas e um vocalista, entretanto a composição dos músicos pode variar de inúmeras formas, acrescentando-se à fórmula tecladistas, vocais de apoio, outros instrumentos como sanfonas e violinos, corais inteiros de orquestra e até possuindo múltiplos percussionistas, em alguns casos. A Figura 2.1 mostra um concerto da banda Iron Maiden, um exemplo de banda com três guitarristas.

O acompanhamento provido por este projeto é pensado para músicos deste estilo, com composições geradas que se encaixam bem em um ensaio de uma banda de metal. Contudo, somente bateria e guitarra foram levados em consideração, com o objetivo de definir bem o escopo do projeto. Dessa forma, é como se o usuário fosse tocar com um baterista e um guitarrista ao seu dispor.

Figura 2.1: Concerto da banda Iron Maiden (Fonte: [7]).

Muitos guitarristas de metal não precisam fazer uso de acordes complexos acrescentando sétimas, fazendo uso de pestanas ou utilizando todas as cordas presentes no instrumento. É muito comum o uso de *power chords*: acordes simples com um formato bem fácil de lembrar que possuem apenas duas ou três notas, utilizando o dedo indicador, o anelar e o mínimo para pressionar a nota fundamental, uma quinta e uma oitava acima. Apesar de parecer pouco variado para um acorde, devemos lembrar que a guitarra no metal está quase sempre altamente distorcida, por isso o som resultante ainda preenche muito bem a harmonia da música. A Figura 2.2 mostra alguns exemplos de *power chords*, onde E representa a sexta corda da guitarra (mi grave), que fica no topo e as demais letras representam as cordas subsequentes: lá, ré, sol, si, mi.

No próximo capítulo, o programa CybrusJam é apresentado, junto com uma descrição do fluxo de uso para usuários e detalhes da implementação, explicando como ele foi feito e como funciona seu código.

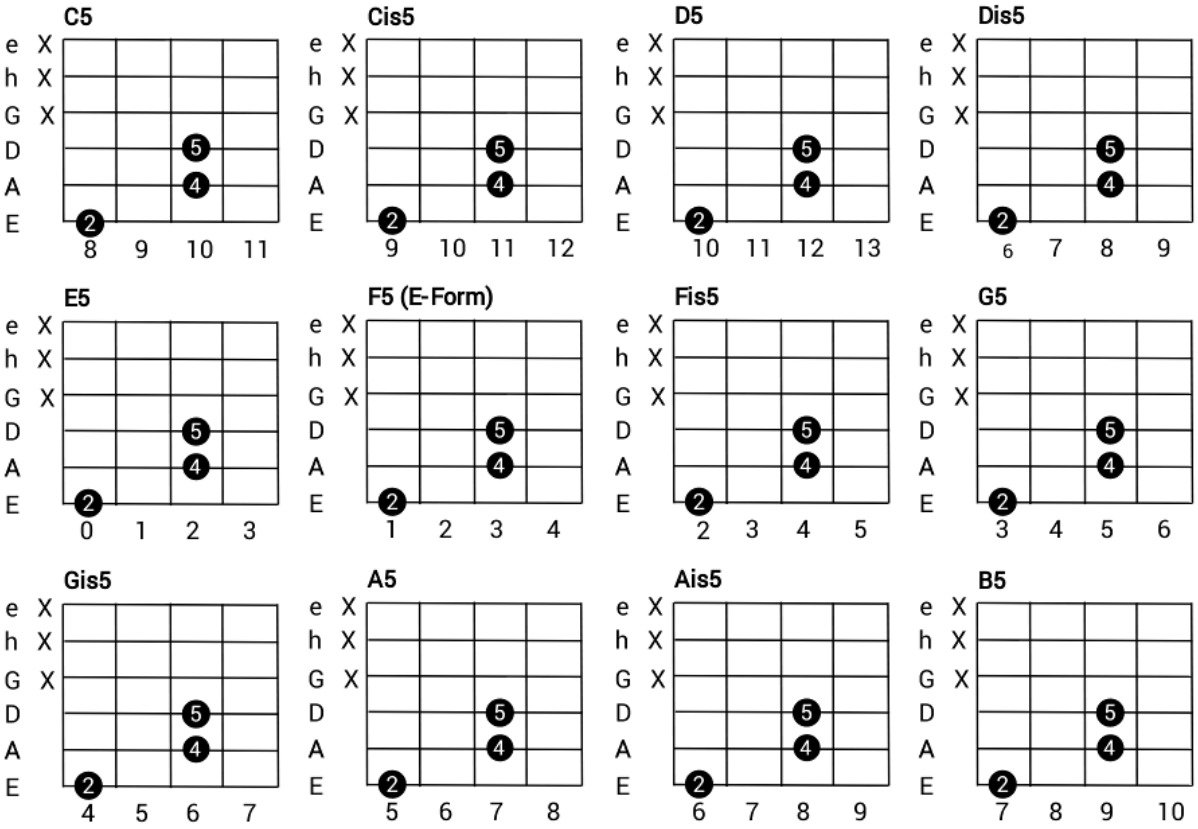


Figura 2.2: Exemplos de *power chords* (Fonte: [8]).

Capítulo 3

O CybrusJam - Acompanhamento Musical

Neste capítulo, o projeto CybrusJam é descrito, tanto do ponto de vista do programador com uma apresentação do que acontece em seu código, como do ponto de vista do usuário, mostrando seus casos de uso, o fluxo de utilização do programa e uma demonstração de como foi montado o ambiente de testes durante seu desenvolvimento.

3.1 Visão Geral

O CybrusJam é um programa de terminal de comandos e foi programado utilizando a linguagem C++[9][10] incluindo a biblioteca MuM descrita anteriormente. Um conversor externo de entrada de áudio digital para sinais MIDI em PureData foi utilizado para a entrada de dados e o sintetizador Timidity foi utilizado para a saída de dados (acompanhamento). Além disso, foi empregado um pedal MIDI, com um controlador Teensy, como controlador do programa, ligado ao computador de testes. O código foi todo escrito no ambiente de desenvolvimento Visual Studio Code[11][12], por preferência do autor.

3.2 Fluxo de Uso

O usuário precisa estar com o baixo ligado na interface de áudio e este, por sua vez, ligado ao computador. Alternativamente, o baixo pode estar ligado diretamente ao computador pela entrada de microfone. O usuário também precisa ligar o pedal a uma das entradas USB do computador. Com estes itens em ordem, é necessário abrir o programa de conversão do PureData (ProjetoCybrusJam-MidiConverter.pd) e configurar a entrada de áudio para o dispositivo em que o baixo se encontra.

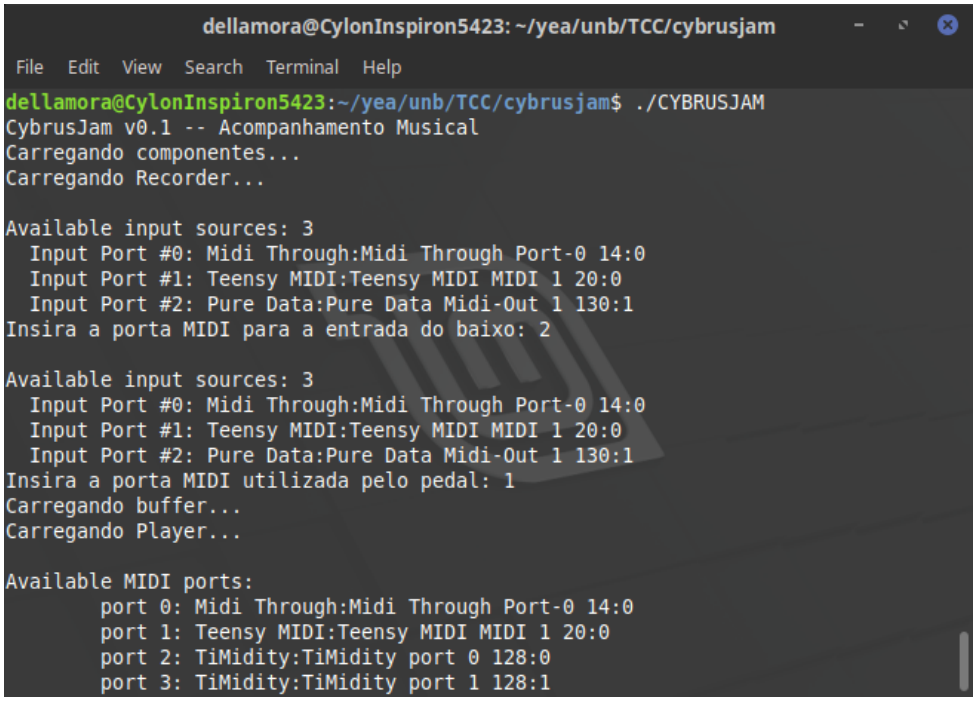


Figura 3.1: Opções de entrada de MIDI para o baixo e o pedal.

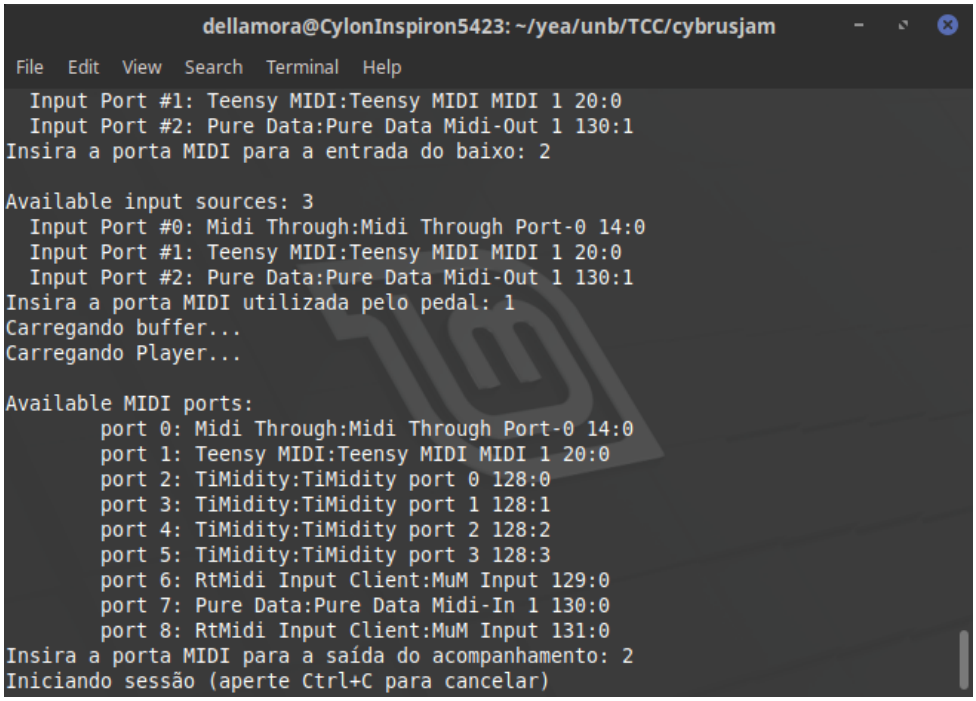


Figura 3.2: Opções de saída MIDI com o Timidity.

Com os dispositivos conectados adequadamente, o usuário pode executar o programa ao digitar o comando descrito no manual de uso (ver Anexo II). Como mostra o diagrama

Figura 3.3: Diagrama de casos de uso.

de casos de uso presente na Figura 3.3, será pedido ao usuário para que ele informe quais são os dispositivos que representam os sinais MIDI que vêm do conversor, os referentes ao pedal e para qual interpretador MIDI o projeto deve enviar a saída. A Figura 3.1 e a Figura 3.2 mostram o programa solicitando as configurações de entrada e saída, respectivamente. Com essas configurações definidas, o programa, então, inicia a primeira sessão de performance, a Fase 1.

Na Fase 1 o usuário toca seu instrumento de qualquer maneira que achar adequada; uma sessão de improvisação individual sem nenhuma restrição na forma de tocar. Quando o contrabaixista estiver satisfeito com sua linha de baixo, ele pressiona o pedal ligado para encerrar essa Fase 1 e deixa o programa analisar a forma musical executada ao utilizar o instrumento na improvisação. Após a análise da entrada de dados, inicia-se a Fase 2, onde o usuário pode tocar acompanhado de um material musical gerado pelo programa, com base no que ele tocou na fase anterior. Após usufruir do acompanhamento, o instrumentista encerra sua "apresentação" e ao ficar cinco segundos sem tocar, o programa irá encerrar o acompanhamento e automaticamente finalizar sua execução.

```
11  ; ========================================
12  ; VOICE: 0, Instr.: 0
13  ; ========================================
14
15  i1  0.000       2.326       6.07        0.000
16  i1  2.326       2.629       6.09        0.000
17  i1  4.955       2.355       6.05        0.000
18  i1  7.310       2.311       6.07        0.000
19
20  ; ========================================
21  ; END VOICE 0
22  ; ========================================
23
```

Figura 3.4: Exemplo de entrada do usuário.

A Figura 3.4 mostra um exemplo de entrada que o usuário poderia fornecer. Da esquerda para a direita, a primeira coluna representa o instrumento MIDI utilizado (que

não importa, nesse caso), o tempo inicial da nota em segundos, a duração em segundos da nota, a nota tocada (em notação Csound, onde o primeiro número representa a oitava e o segundo representa o semitom dentro da oitava) e o volume (parâmetro que será utilizado para reconhecimento da forma de execução posteriormente). Aqui, o usuário tocou as notas sol, lá, fá e sol novamente.

```
11  ; ========================================
12  ; VOICE: 0, Instr.: 2
13  ; ========================================
14
15  i2  0.000       2.405       6.07        0.400
16  i2  0.000       2.405       7.02        0.400
17  i2  0.000       2.405       7.07        0.400
18  i2  2.405       2.405       6.09        0.400
19  i2  2.405       2.405       7.04        0.400
20  i2  2.405       2.405       7.09        0.400
21  i2  4.810       2.405       6.05        0.400
22  i2  4.810       2.405       7.00        0.400
23  i2  4.810       2.405       7.05        0.400
24  i2  7.215       2.405       6.07        0.400
25  i2  7.215       2.405       7.02        0.400
26  i2  7.215       2.405       7.07        0.400
27
28  ; ========================================
29  ; END VOICE 0
30  ; ========================================
```

Figura 3.5: Exemplo de saída do programa: guitarra.

A Figura 3.5 é o exemplo de saída da figura anterior, onde a voz 0 representa a guitarra. Aqui, o programa gerou *power chords* referentes às notas presentes na Figura 3.4. Na Figura 3.6 aparece o resultado de saída do programa na voz 1, que diz respeito à bateria. As linhas com amplitude (volume) 1 são as notas de bumbo, uma correção de volume necessária para que o usuário possa escutar mais claramente o instrumento.

```
32    ; ====================================
33    ; VOICE: 1, Instr.: 10
34    ; ====================================
35
36    i10 0.000       0.601       5.11        1.000
37    i10 0.000       0.601       7.01        0.400
38    i10 0.301       0.301       6.06        0.400
39    i10 0.601       0.601       5.11        1.000
40    i10 0.601       0.601       6.02        0.400
41    i10 0.902       0.301       6.06        0.400
42    i10 1.203       0.301       5.11        1.000
43    i10 1.503       0.301       5.11        1.000
44    i10 1.503       0.301       6.06        0.400
45    i10 1.804       0.601       5.11        1.000
46    i10 1.804       0.601       6.02        0.400
47    i10 2.104       0.301       6.06        0.400
48    i10 2.405       0.601       5.11        1.000
49    i10 2.706       0.301       6.06        0.400
50    i10 3.006       0.601       5.11        1.000
51    i10 3.006       0.601       6.02        0.400
52    i10 3.307       0.301       6.06        0.400
53    i10 3.608       0.301       5.11        1.000
54    i10 3.908       0.301       5.11        1.000
55    i10 3.908       0.301       6.06        0.400
56    i10 4.209       0.601       5.11        1.000
57    i10 4.209       0.601       6.02        0.400
58    i10 4.510       0.301       6.06        0.400
59    i10 4.810       0.601       5.11        1.000
60    i10 5.111       0.301       6.06        0.400
61    i10 5.412       0.601       5.11        1.000
62    i10 5.412       0.601       6.02        0.400
63    i10 5.712       0.301       6.06        0.400
64    i10 6.013       0.301       5.11        1.000
65    i10 6.313       0.301       5.11        1.000
66    i10 6.313       0.301       6.06        0.400
67    i10 6.614       0.601       5.11        1.000
68    i10 6.614       0.601       6.02        0.400
69    i10 6.915       0.301       6.06        0.400
70    i10 7.215       0.601       5.11        1.000
71    i10 7.516       0.301       6.06        0.400
72    i10 7.817       0.601       5.11        1.000
73    i10 7.817       0.601       6.02        0.400
74    i10 8.117       0.301       6.06        0.400
75    i10 8.418       0.301       5.11        1.000
76    i10 8.719       0.301       5.11        1.000
77    i10 8.719       0.301       6.06        0.400
78    i10 9.019       0.601       5.11        1.000
79    i10 9.019       0.601       6.02        0.400
80    i10 9.320       0.301       6.06        0.400
81
82    ; ====================================
83    ; END VOICE 1
84    ; ====================================
85
```

Figura 3.6: Exemplo de saída do programa: bateria.

3.3 Hardware utilizado

A Figura 3.7 é uma representação do ambiente de testes utilizado ao longo do desenvolvimento do projeto CybrusJam. O baixo elétrico, que envia o sinal analógico nas faixas de frequência entre 20Hz e 9kHz, é ligado, utilizando um cabo com conector P10, à interface de gravação de áudio Focusrite Scarlett 2i2. A interface, então, envia o sinal digital correspondente (em 24-bits e amostrado a uma taxa de 96Khz) ao Notebook através de cabo USB. O notebook está rodando o sistema operacional Linux Mint versão 19.1 (uma distribuição de Linux Ubuntu). O sinal de áudio digital é enviado ao *software* de conversão para sinais MIDI no PureData, os quais serão recebidos pelo projeto em execução. Além disso, um pedal com um circuito eletrônico e um botão é ligado em outra entrada USB, enviando um sinal MIDI de controle que também é tratado pelo projeto em execução. O projeto gera sinais MIDI de acompanhamento que serão interpretados pelo *software* sintetizador Timidity, para que o som saia em qualquer dispositivo de áudio conectado ao computador.

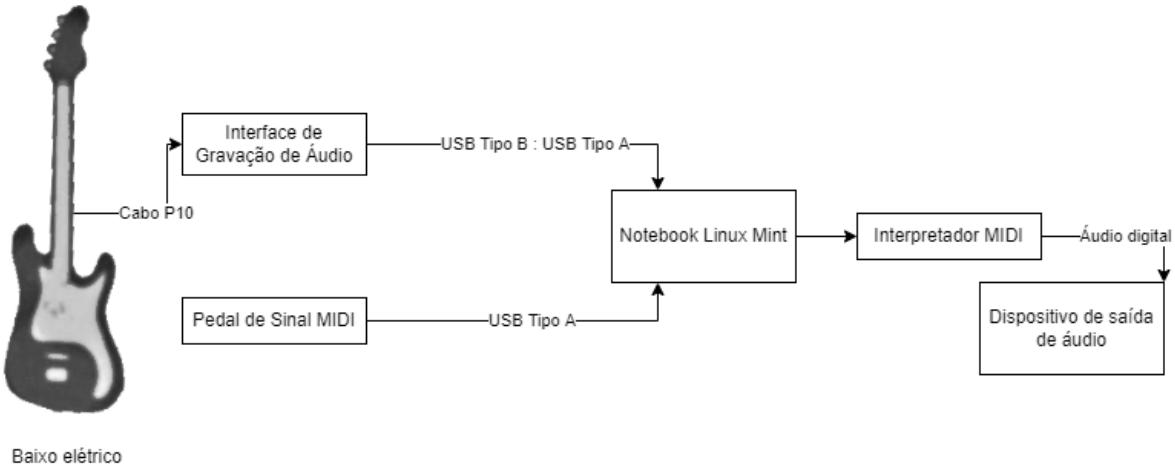


Figura 3.7: Visão geral do ambiente de testes.

3.4 Detalhes de Implementação

Conforme o diagrama de classes no Anexo I, o projeto conta com quatro classes, sendo uma delas a principal, “CybrusMain”, que controla o fluxo de execução com a entrada dos sinais MIDI do baixo e do pedal (os objetos MuRecorder input e pedal, respectivamente), fazendo a análise da forma de execução do baixo com o objeto “bAnalise”, gerando o acompanhamento com os objetos “guitarra” e “bateria” e projetando uma saída MIDI através de “output” (um objeto MuPlayer).

O guitarrista do acompanhamento provido pelo projeto faz uso apenas de *power chords* que vão do mi da sexta corda até o ré sustenido da mesma, provendo uma harmonia suficiente numa banda de metal. Outro guitarrista para prover os icônicos solos e outras melodias típicas do estilo foi considerado para ser incluído neste projeto, mas devido à restrição de tempo e escopo, ele foi apenas cotado como melhoria futura.

O baterista foi pensado com um conjunto de bateria padrão, contendo os seguintes componentes: um bumbo com pedal duplo, uma caixa, dois tom-tons diferentes, um surdo, um prato de contratempo, um prato de *crash* e um prato de condução. Ele está programado pra fazer dois tipos de ritmos diferentes, conforme for pedido: uma batida mais leve, utilizando apenas um pedal de bumbo para andamentos mais devagar e outra mais pesada, utilizando o pedal duplo nos bumbos para andamentos mais acelerados.

3.4.1 CybrusJam e MuM

Dentre as várias funcionalidades da biblioteca MuM, é preciso destacar três componentes utilizados no projeto CybrusJam: MuMaterial, MuRecorder e MuPlayer. O escopo do projeto permitiu que fossem utilizadas essas três classes da seguinte forma:

A classe MuMaterial é a mais usada. Com ela são feitas todas as operações de composição algorítmica: transformar *buffers* MIDI do MuRecorder em materiais musicais; criar novos materiais a partir de notas separadas; identificar as notas usadas nos materiais de entrada para realizar a escolha dos acordes; modificar de inúmeras maneiras os materiais carregados. A classe MuRecorder, por sua vez, é usada para receber a entrada de sinais MIDI enquanto que a classe MuPlayer realiza o *playback* do acompanhamento.

Durante a execução do programa, um objeto da classe MuRecorder recebe os dados MIDI vindos do baixo. Um outro objeto da mesma classe recebe uma segunda entrada MIDI gerada pelo pedal MIDI auxiliar. Uma instância da classe MuMaterial é utilizada para analisar os sinais recebidos e gerar o material de acompanhamento necessário. Este material é então enviado para um objeto da classe MuPlayer que toca esse acompanhamento, enquanto o primeiro MuRecorder continua recebendo sinais MIDI para alertar o programa de que ele deve continuar reproduzindo o material do acompanhamento.

3.4.2 Configuração

Inicialmente, são preparados os componentes para o funcionamento do programa. São carregadas as classes MuRecorder (tanto para o baixo como para o pedal), MuPlayer (para a saída MIDI) e outras variáveis de controle para a parte inicial da execução. O programa imprime na tela instruções para o usuário selecionar suas configurações de entrada e saída e dá início à entrada de dados, denominada de Fase 1.

3.4.3 Fase 1

A Fase 1 consiste em o baixista realizar uma improvisação de quaisquer notas ele quiser tocar. O conversor envia as notas tocadas como sinais MIDI para o programa principal receber através do gravador musical (MuRecorder) e montar um *buffer* de mensagens MIDI correspondente (MuMIDIbuffer) para ser enviado para análise. Toda vez que o *buffer* é atualizado para receber notas, as mensagens antigas são apagadas. De modo a contornar isso, um outro *buffer* é utilizado acumulando as mensagens do primeiro, para finalmente ser enviado por inteiro para outra classe (BaixoAnalise) e ser transformado em material musical e analisado de forma correta.

O algoritmo espera que a última nota seja finalizada e o usuário pressione o pedal de controle para encerrar essa fase e enviar o material coletado para a análise da forma de execução feita pelo baixista.

3.4.4 Análise

Ao final da Fase 1, o programa cria um objeto da classe "BaixoAnalise", enviando o acumulado de notas MIDI coletadas, gerado pelo usuário e passado pelo conversor. Na análise, acontece a transformação da coletânea de sinais MIDI em um material musical (MuMaterial) e acontece um ajuste do tempo inicial das notas, devido ao fato do usuário sempre começar a tocar com algum atraso a partir do início da Fase 1.

Com o material musical corrigido, a classe de análise faz uma busca de quantas pulsações o usuário tocou, pois ele não necessariamente tocou uma nota por pulsação e também há a possibilidade de existirem "notas fantasma" (esbarradas no instrumento que são computadas como notas completas e válidas) que não contribuem para o que o usuário visionou como a melodia escolhida para essa improvisação. Essa busca leva em consideração quais são as notas fundamentais (as notas mais próximas do início de cada pulsação) do material; notas essas que representarão qual acorde da guitarra será selecionado (a nota mais grave de cada acorde corresponde à fundamental). A busca consiste em experimentar divisões do material em 1, 2, 4, 8 até a mesma quantidade de notas existentes em quantidades de pulsações. A Figura 3.8 ilustra como é feita a divisão do material, onde cada marcação vermelha representa uma nota qualquer tocada pelo usuário em uma linha do tempo. O algoritmo, então, verifica se há uma nota fundamental no início de cada compasso, checando se o tempo inicial da nota está próximo o bastante do tempo inicial do compasso, com uma pequena margem de erro. A cada vez que uma nota fundamental for encontrada no início dos compassos formados, um contador aumenta, sendo que o algoritmo selecionará a divisão com o maior contador. O algoritmo destaca as notas fundamentais através de sua amplitude, guarda as informações de quantas pul-

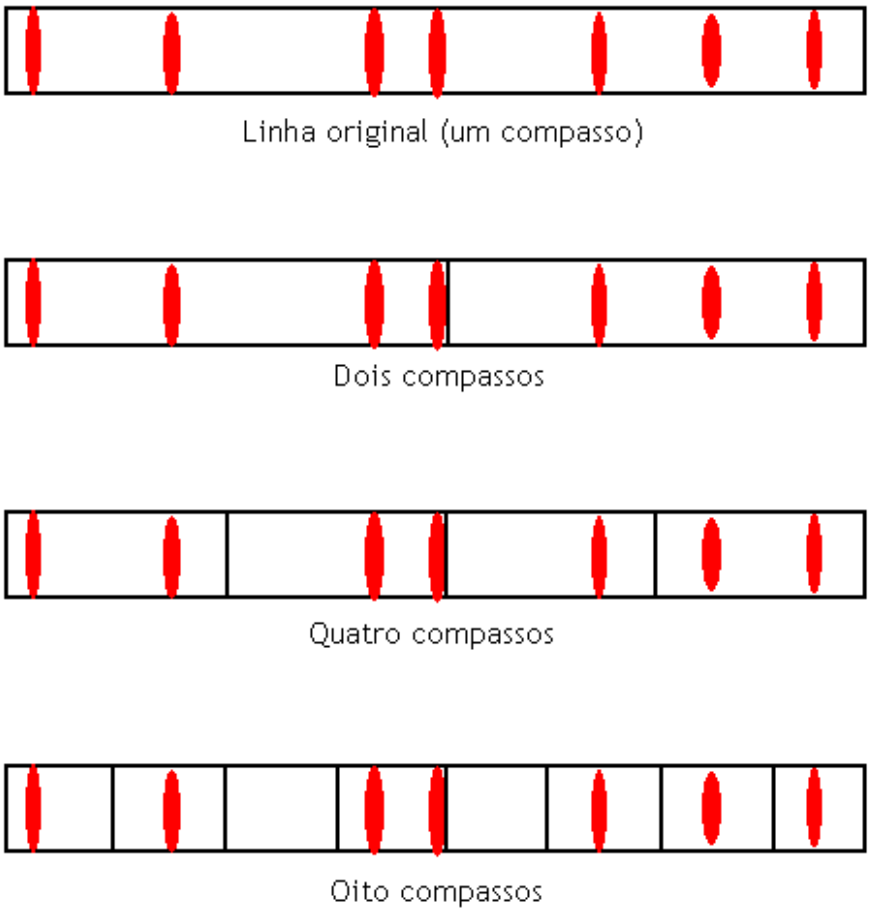


Figura 3.8: Ilustração de como é feita a busca por notas fundamentais.

sações o material tem e a duração de uma pulsação e vai, a seguir, gerar a guitarra e a bateria em suas respectivas classes.

3.4.5 Construção do Acompanhamento

As classes de guitarrista e baterista são bem parecidas: ambas possuem uma referência ao material gerado pela análise do baixo, geram seu próprio material a ser escolhido posteriormente por suas devidas análises e acrescentam seu acompanhamento ao material musical que será tocado na Fase 2, mas cada um executa sua geração de uma maneira diferente. Os materiais gerados pela guitarra e bateria estão demonstrados nas partituras presentes no Anexo III

A classe do guitarrista trabalha com *power chords*, como discutido na parte teórica deste documento. Ela então gera *power chords* para cada nota dentro de uma oitava

inteira, desde a nota mi (E38) até o ré sustenido (D#49), como se cobrisse as doze primeiras casas de uma guitarra. Esses acordes cobrem os valores de altura MIDI de 38 até 61. Com esse banco de acordes em memória, o próximo passo da classe é gerar os possíveis ritmos de palhetadas.

Após a geração do material próprio do guitarrista (acordes, ritmos), a classe começa a ler o material musical da análise do baixo. Utilizando os dados de quantas pulsações há no material e a duração delas, é montada uma equação para determinar qual o andamento do material. A classe, então, busca no material as notas fundamentais destacadas anteriormente para definir os acordes para cada compasso. Busca-se qual o semitom da nota fundamental no banco de acordes e o *power chord* é separado. Dependendo de em qual compasso está e qual o andamento encontrado, um dos ritmos montados será escolhido. O material resultante será uma série de palhetadas contendo o acorde inteiro ou somente a nota fundamental do acorde dependendo de como o ritmo escolhido faz uso do acorde. No final, o material de acompanhamento do guitarrista será adicionado como uma nova voz ao material analisado do baixo.

A classe do baterista usufrui diretamente de notas musicais (MuNotes), uma para cada componente de uma bateria comum utilizada em muitas bandas de metal (utilizando as notas 35, 38, 41, 45, 50, 42, 49 e 51, correspondendo a bumbo, caixa, surdo, dois tipos de tons, contratempo, *crash* e condução, respectivamente). A classe gera os parâmetros necessários para que cada nota-componente esteja de acordo com a convenção General MIDI de uma bateria (cada componente tem uma altura de nota específica), bem como outras configurações de volume e definição de qual canal MIDI tocar a bateria (segundo a convenção de General MIDI, o canal 10). O próximo passo da classe é montar o banco de ritmos da bateria, similarmente ao banco de ritmos da guitarra, contudo são utilizadas apenas as notas aqui descritas, visto que a execução da bateria não depende de qual nota foi tocada no baixo, somente depende de qual componente será usado em determinado momento do ritmo (seja só um bumbo, um bumbo com uma caixa, preencher o som com o contratempo, por exemplo).

Quando a geração de componentes e ritmos houver terminado, a classe começa a analisar o material do baixo e, da mesma forma que a classe da guitarra, utiliza a mesma equação para encontrar o andamento em que o usuário tocou. O método de análise prossegue fazendo uma varredura nos compassos do material, inserindo os ritmos adequados para o andamento encontrado. O resultado será copiado para mais uma voz recém-adicionada ao material, totalizando três vozes: uma para o baixo (que não será tocada), uma para a guitarra e uma para a bateria.

3.4.6 Fase 2

Antes de dar seguimento ao fluxo do programa com a Fase 2 e iniciar o acompanhamento gerado pela fase de análise, é feita uma pequena modificação no material resultante para tirar o volume da voz em que as notas do baixo se encontram, para não serem tocadas e confundir o instrumentista. O programa então solicita ao MuPlayer que comece a tocar o material de acompanhamento. Enquanto o material é tocado, uma das entradas MuRecorder é novamente ativada para receber a sinais do baixo do usuário, só que dessa vez ele não será guardado em um material, somente servirá para manter o acompanhamento repetindo toda vez que chegar ao final, enquanto houver notas sendo tocadas.

Quando o usuário, por qualquer motivo, quiser interromper esta fase do programa e encerrar sua execução, basta ele ficar cinco segundos sem tocar nenhuma nota. O programa irá perceber a falta de notas neste período e irá encerrar as repetições quando terminar a reprodução atual do acompanhamento, limpando a memória e finalizando sua execução.

O código está disponível no GitHub: `https://github.com/vdellamora/cybrus-jam`

No próximo capítulo, é feita a conclusão desta monografia e são detalhadas as possíveis melhorias futuras para o projeto CybrusJam.

Capítulo 4

Conclusão

Após inúmeros testes com diferentes linhas de baixo executadas, com levadas variadas, uma série de combinações de notas diferentes, mudando a velocidade para mais rápido ou mais devagar, o *software* provou-se bem capaz de prover um acompanhamento adequado para o gênero musical *heavy metal*. O contrabaixista consegue, sem nenhum esforço, tocar qualquer improvisação que sentir vontade e o algoritmo gerará um material musical que combina com o que ele tocou, com mínima margem de erro (erro humano, o que o código consegue contornar suficientemente bem através da eliminação de "notas-fantasma").

Neste projeto foram utilizados conceitos de composição algorítmica e orientação a objetos, em conjunto com as tecnologias MIDI, PureData e a linguagem C++ com a biblioteca MuM para fazer um *software* de acompanhamento musical que possa auxiliar baixistas que tocam o gênero metal com a prática individual de seu instrumento. Pesquisas sobre o gênero musical abordado nesse trabalho e sobre outros programas semelhantes permitiram que o desenvolvimento do CybrusJam pudesse tomar forma e o produto final ser testado suficientemente para que ele cumprisse o objetivo especificado na metodologia deste projeto.

A biblioteca MuM e o padrão de sinais digitais MIDI possuem, em sua genética, os fatores necessários para se trabalhar com música em tempo real, possibilitando o desenvolvimento de um aplicativo como o apresentado neste trabalho, que provê um acompanhamento sem a necessidade de o usuário ficar definindo quais instrumentos estarão presentes e o que eles tocarão em determinado compasso (função já existente em outros programas de acompanhamento musical que estão difundidos no mercado).

4.1 Melhorias Futuras

Certamente há espaço para melhorias na ferramenta CybrusJam e para expandir o leque de funcionalidades que ele já apresenta. A estrutura montada utilizando a orientação

à objetos para este projeto fornece a facilidade de atualizações e outras ideias a serem incluídas para se adequar a outros ambientes em que futuros programadores e músicos queiram utilizá-lo.

Entre possíveis melhorias já consideradas, pode-se citar:

- Fornecer uma interface de usuário mais amigável (possivelmente GUI) que permita a definição de preferências de forma persistente;
- Melhorar a análise do instrumento, considerando não só notas soltas;
- Acrescentar novos instrumentos utilizando a mesma estrutura de classe apresentada pela bateria e guitarra;
- Acomodar outros gêneros musicais;
- Reescrever o código pensando em outro instrumento de entrada utilizado pelo usuário (guitarras, saxofones, trombones, etc.);
- Incluir no próprio código C++ um conversor de áudio analógico/digital em sinais MIDI;
- Testar em diferentes sintetizadores MIDI como alternativas de saída;
- Testar em outros sistemas operacionais, visto que o escopo deste projeto estava limitado ao Linux;

Antes de qualquer outra coisa, o CybrusJam foi pensado para ser lido e expandido por outros programadores, apresentando uma estrutura fácil de entender e código legível, com comentários que acompanham o leitor no entendimento do funcionamento do projeto. Essa é uma característica essencial para que novos projetos surjam a partir deste apresentado.

Referências

[1] Maurer, John e A John: *A brief history of algorithmic composition*. Unpublished manuscript. Available at https://ccrma.stanford.edu/˜ blackrse/algorithm.html, 1999. 4

[2] Mello, Carlos Eduardo Vianna de: *Musical materials and algorithmic composition*. Anais do XIX Congresso da ANPPOM, páginas 526–528, 2009. 5

[3] Loy, Gareth: *Musicians make a standard: the midi phenomenon*. Computer Music Journal, 9(4):8–26, 1985. 7

[4] Association, MIDI Manufacturers *et al.*: *The complete midi 1.0 detailed specification*. Los Angeles, CA, The MIDI Manufacturers Association, 1996. 7

[5] Puckette, Miller S *et al.*: *Pure data*. Em *ICMC*, 1997. 8

[6] Wallach, Jeremy, Harris M Berger e Paul D Greene: *Metal rules the globe: Heavy metal music around the world*. Duke University Press, 2011. 8

[7] CroatiaWeek: *Iron maiden announce croatia date for 'legacy of the beast' tour*. `https://www.croatiaweek.com/iron-maiden-announce-croatia-date-for-legacy-of-the-beast-tour/`, acesso em 2022-05-10. 9

[8] Dill, Thomas: *Power chords for the guitar: Guitar chords for rock and roll*. `https://music2me.com/en/magazine/power-chords`, acesso em 2022-04-27. 10

[9] Stroustrup, Bjarne: *An overview of c++*. Em *Proceedings of the 1986 SIGPLAN workshop on Object-oriented programming*, páginas 7–18, 1986. 11

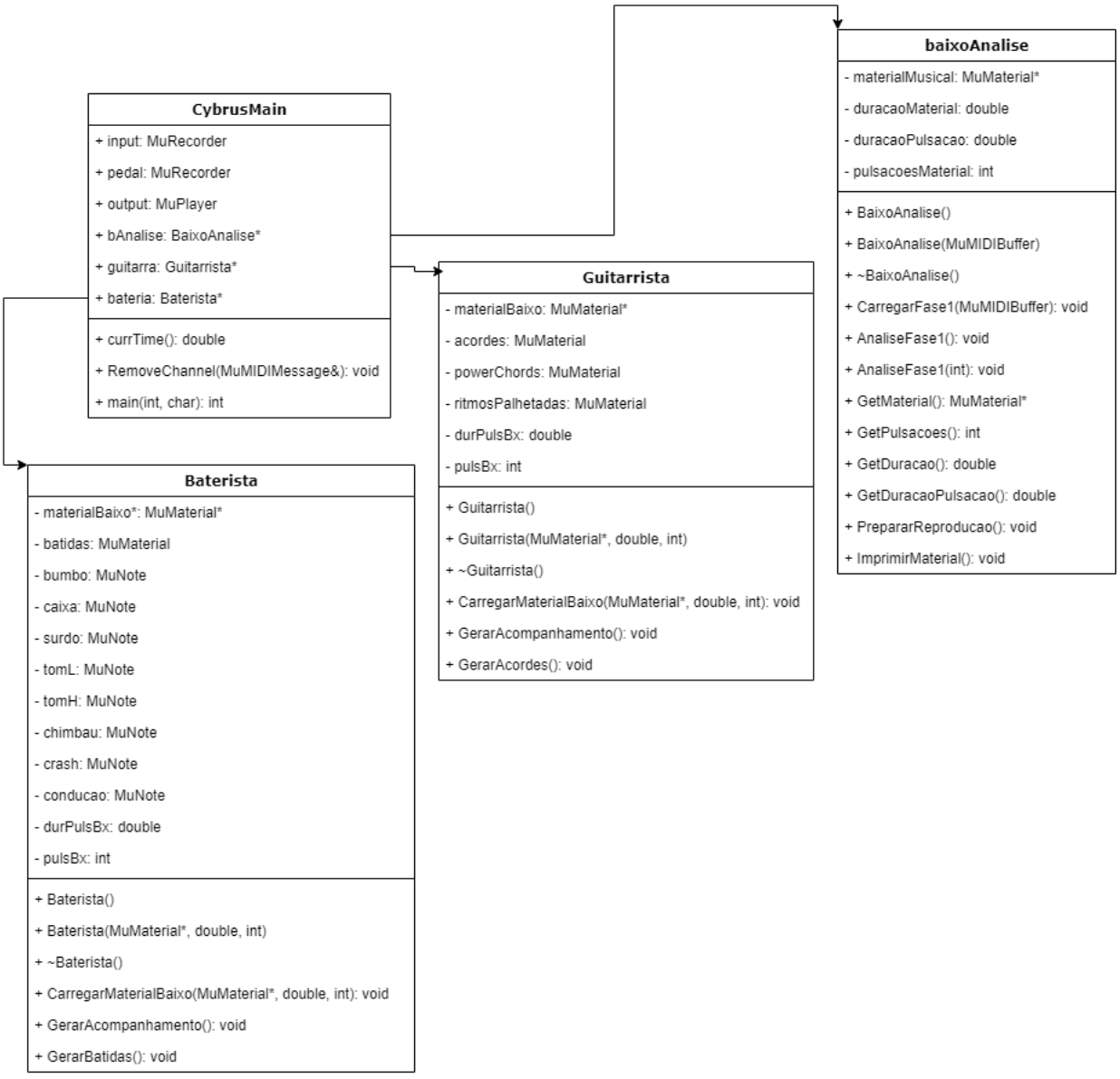
[10] W3Schools: *C++ tutorial*. `https://www.w3schools.com/CPP/default.asp`, acesso em 2022-05-10. 11

[11] Del Sole, Alessandro: *Introducing visual studio code*. Em *Visual Studio Code Distilled*, páginas 1–15. Springer, 2021. 11

[12] Microsoft: *Visual studio code - code editing. refined*. `https://code.visualstudio.com/`, acesso em 2022-05-10. 11

Anexo I

Diagrama de Classes



Anexo II

Manual de Uso

README.md

cybrus-jam

CybrusJam é um software de acompanhamento musical que fornece instrumentos a uma sessão de improvisação para músicos que queiram praticar sozinhos, fazendo uso da biblioteca MuM do professor Dr. Carlos Eduardo Vianna de Mello, do Departamento de Música da Universidade de Brasília. Este é o projeto de trabalho de graduação do curso de Ciência da Computação na Universidade de Brasília do aluno Vitor Moraes Dellamora.

Conexões necessárias

- Conecte o baixo em uma interface de áudio ou diretamente na entrada de microfone do computador;
- Conecte o pedal Teensy na porta USB do computador;
- Abra o arquivo do PureData (ProjetoCybrusJam-MidiConverter.pd);
- Abra as portas MIDI do sintetizador que será utilizado (por exemplo, Timidity);
- Defina um dispositivo de saída de áudio do computador como a saída padrão;

Instruções de execução

Para compilar o software, compile a biblioteca utilizando: `make compileMUM`

Em seguida, compile o projeto utilizando: `make compile`

Configure a entrada de áudio e as portas ALSA-MIDI no PureData.

Execute o programa com `./CYBRUSJAM`

A seguir, selecione as entradas de MIDI para o baixo e para o pedal. Escolha a entrada com "PureData" para o baixo e "Teensy" para o pedal. Após isso, selecione a saída MIDI desejada (por exemplo, TiMidity port).

Execute uma linha de baixo à sua escolha e pressione o pedal quando estiver satisfeito. Toque junto do acompanhamento ou fique cinco segundos sem utilizar o instrumento para encerrar a execução.

Anexo III

Partituras do Material de Acompanhamento

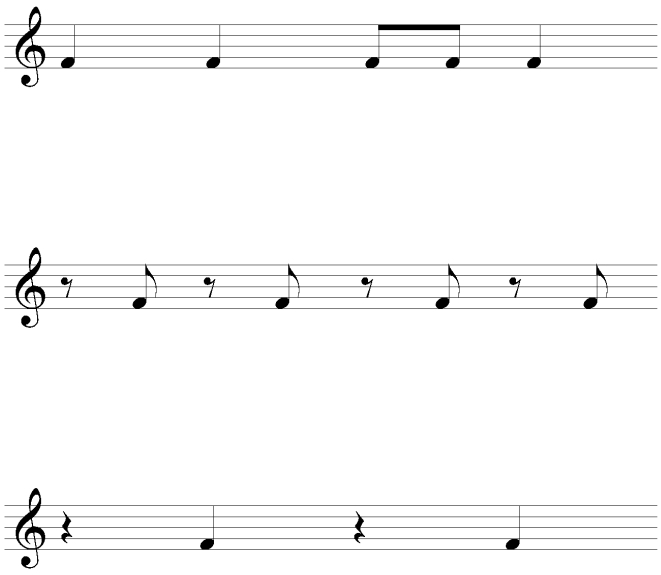


Figura III.1: Bateria leve A.

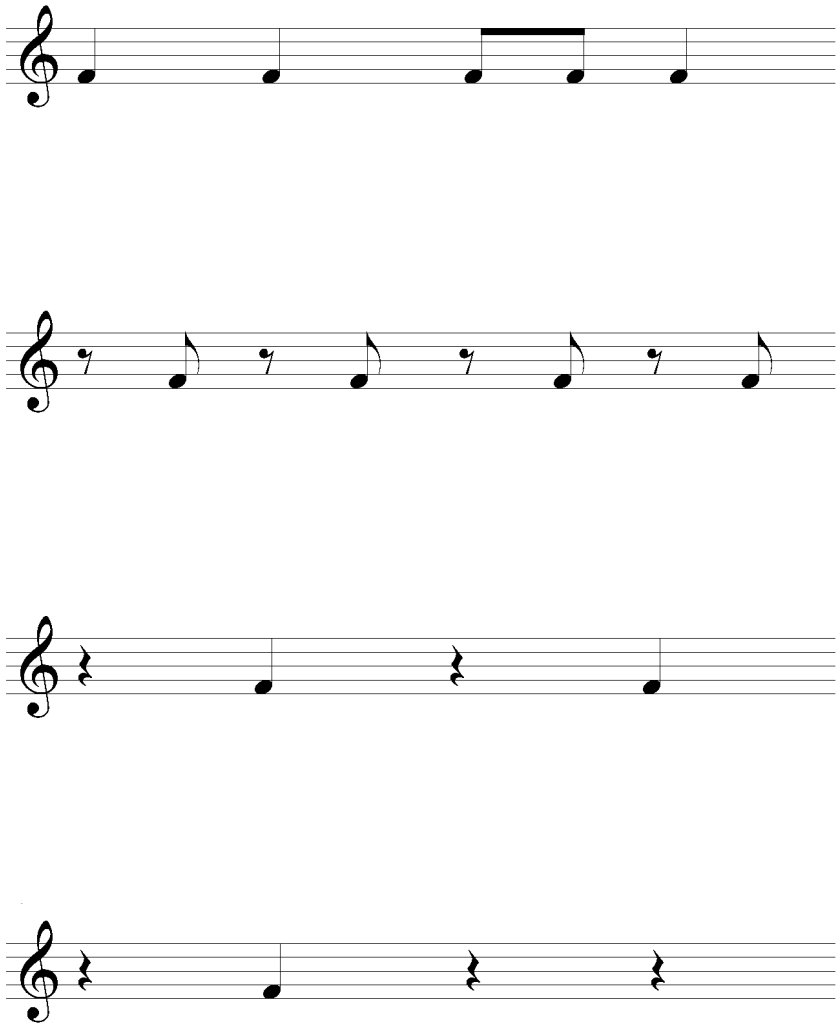


Figura III.2: Bateria leve B.

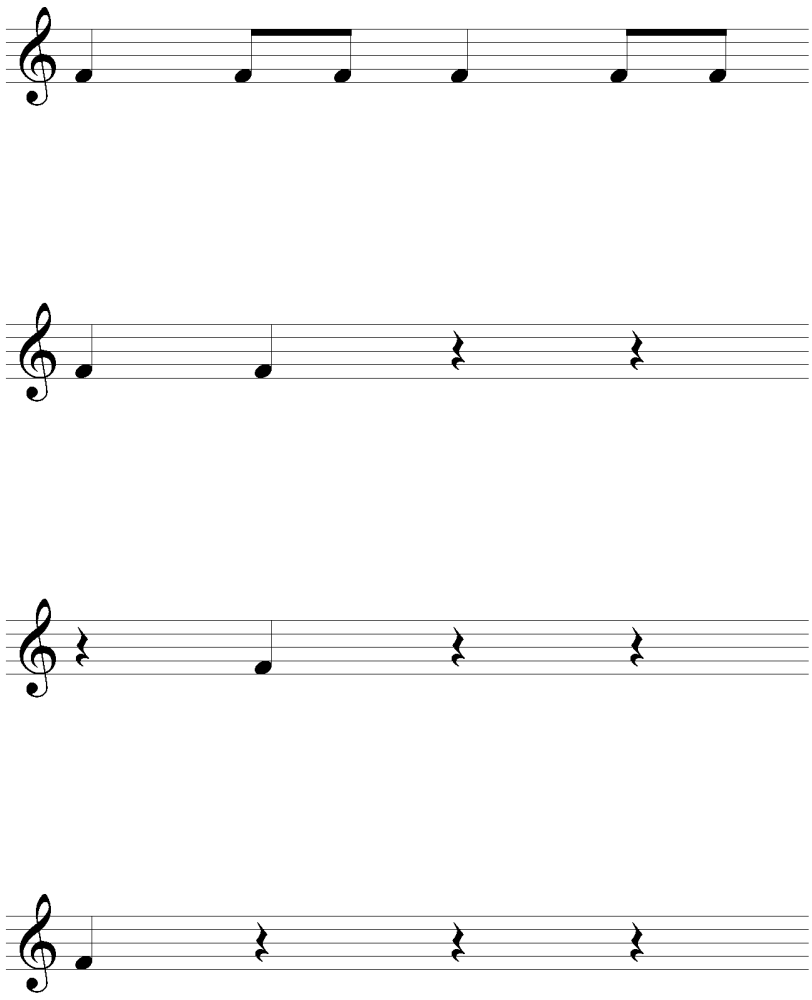


Figura III.3: Bateria rápida A.

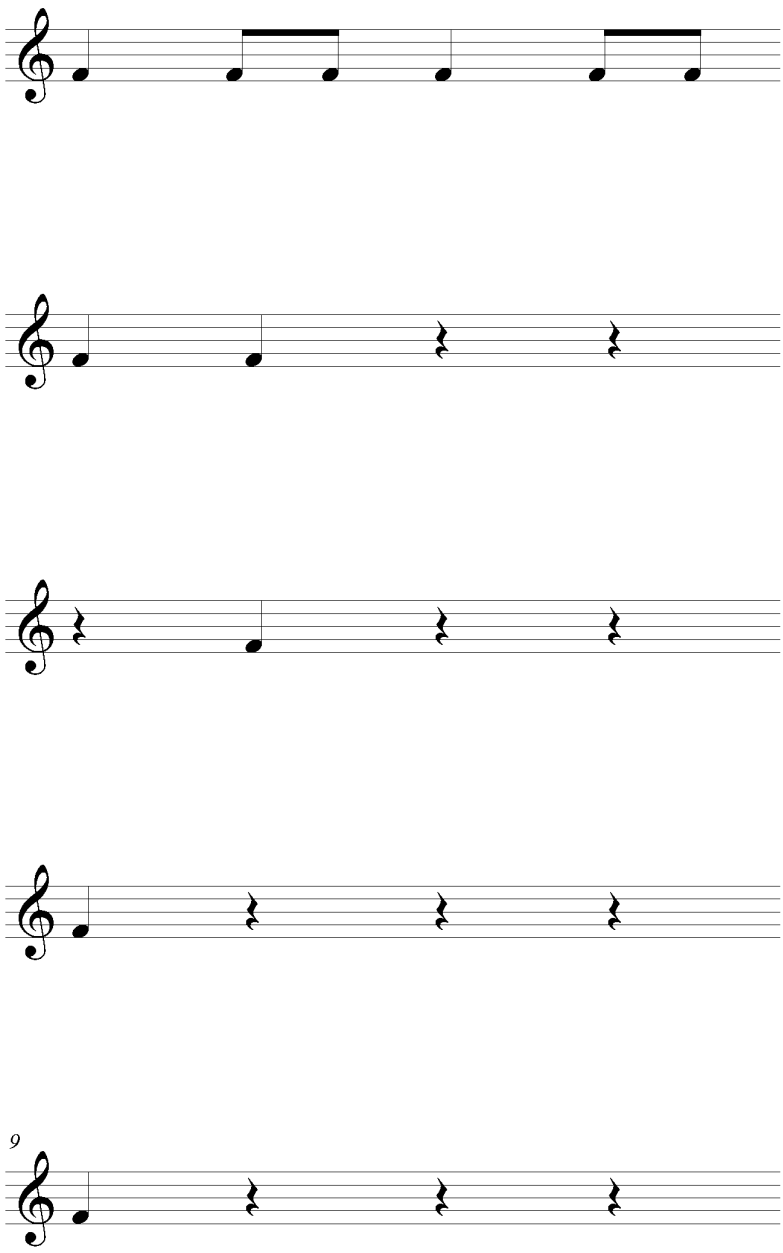


Figura III.4: Bateria rápida B.

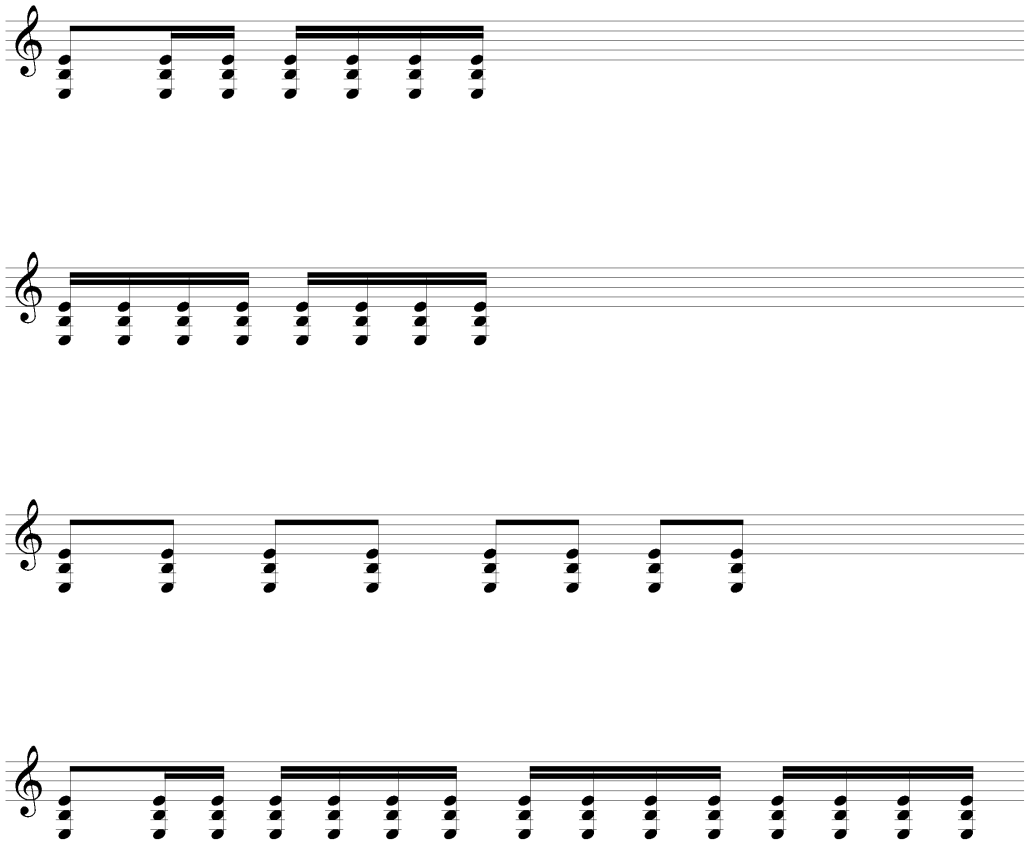


Figura III.5: Materiais da guitarra.