INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DO
RIO GRANDE DO SUL
CAMPUS CANOAS
CURSO TÉCNICO EM DESENVOLVIMENTO DE SISTEMAS INTEGRADO AO ENSINO MÉDIO

GUILHERME VIANA DOS SANTOS

FRETCAT: Simulador de pedaleiras e efeitos para guitarras

Orientador: Prof. Msc Ígor Lorenzato Almeida

Canoas, novembro de 2023

GUILHERME VIANA DOS SANTOS

FRETCAT: Simulador de pedaleiras e efeitos para guitarras

Trabalho de Conclusão de Curso apresentado como requisito parcial para obtenção do grau de Técnico em Desenvolvimento de Sistemas pelo Instituto Federal de Educação, Ciência e Tecnologia do Rio Grande do Sul – Campus Canoas.

Prof. Msc Ígor Lorenzato Almeida

Orientador

Canoas, novembro de 2023

Dedico este trabalho aos meus pais que sempre me apoiaram e financiaram meus estudos. Também agradeço meus tios, sem eles não seria o musico e programador que sou hoje.

RESUMO

O presente trabalho de conclusão de curso apresenta um programa *desktop* que visa simular efeitos de guitarra digitalmente. Este trabalho foi elaborado com o intuito de fornecer uma ferramenta gratuita e intuitiva para alterar o timbre de uma guitarra por meio de computadores, retirando a necessidade de uma pedaleira. Foi realizada uma pesquisa de natureza aplicada buscando e agrupando diversos algoritmos já existentes para computação de efeitos. A partir disso foram feitos diagramas na linguagem UML assim como foi feita a seleção das ferramentas e tecnologias utilizadas. Após a implementação do Fretcat, foram feitos testes com músicos e integrantes da banda do Instituto Federal de Educação, Ciência e Tecnologia do Rio Grande do Sul – Campus Canoas, os quais após realizarem a testagem, responderam um questionário medindo a satisfação com o *software*. Através dos resultados pode ser ver que os músicos selecionados acreditam que o sistema permite a criação de timbres satisfatórios de maneira simples e intuitiva. Por fim, foram traçados trabalhos futuros com o intuito de refinar o programa.

Palavras-Chave: Desktop. Processamento de áudio. Efeito digital. VST

LISTA DE ABREVIATURAS E SIGLAS

VST	*Virtual Studio Technology*
DAW	*Digital Audio Workstation*
DSP	*Digital Signal Processing*
DLL	*Dynamically Linked Library*
SO	*Shared Object*
SVF	*State Variable Filter*
IIR	*Infinite Impulse Response*
JSON	*JavaScript Object Notation*

LISTA DE FIGURAS

LISTA DE QUADROS

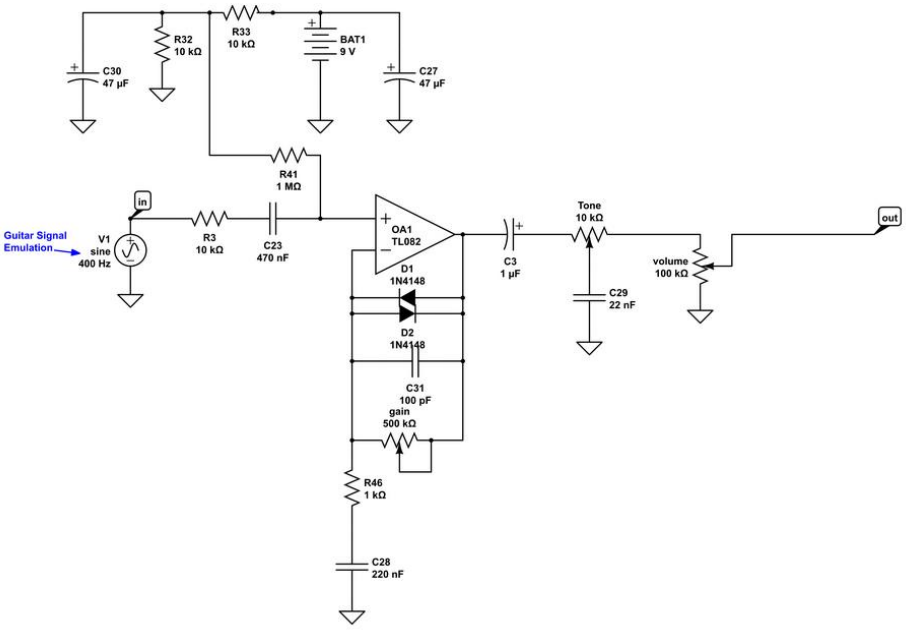
Sumário

1. INTRODUÇÂO

Esse capitulo irá introduzir as origens dos efeitos digitais, descrevendo o processo de digitalização dos meios músicas. Ademais, irá ser definindo o problema a ser resolvido pelo presente trabalho, a proposta de solução adotada e os objetivos a serem concluídos.

Com a digitalização de muitos meios analógicos, como a televisão, surgiu um novo ramo na área da programação, chamado *Digital Signal Processing* (Processamento de sinais digitais). Este ramo se encarrega de processar e analisar sinais discretos, sendo utilizado tanto para fins musicais, como até a análise de dados de experimentos científicos (SMITH, 1999). A produção musical se beneficiou com esse salto tecnológico com a invenção da *Digital Audio Workstation* (Estação de trabalho de áudio digital), substituindo as gravadoras de bobina[1]. A guitarra também se beneficiou com esse processo, sendo agora possível manipular seu timbre com o uso de *software*, retirando a necessidade de se utilizar uma pedaleira[2]. A Figura 1 demonstra o circuito de uma pedaleira de distorção e a Figura 2 demonstra o algoritmo que o replica, essa pedaleira é a origem do som normalmente associado com guitarras de Rock e Metal.

Figura 1- Circuito de um pedal de distorção



Fonte: *WAMPLER* (2020)

[1] Gravador de bobina, ou gravador de rolo, são aparelhos de som que armazenavam áudio em fitas magnéticas ao redor de um rolo, algo similar às fitas VHS.

[2] Uma pedaleira é um circuito elétrico que visa alterar um sinal de entrada conforme os parâmetros ajustados pelo usuário

Figura 2 - Código que imita um pedal de distorção

```
let gain = ((self.boost / 100.0) * 100.0) + 1.0;

*left *= gain;
*right *= gain;

let a = (((self.drive + 1.0) / 101.0) * (PI / 2.0)).sin();
let k = 2.0 * a / (1.0 - a);

let drive_l = (1.0 + k) * *left / (1.0 + k * left.abs());
let drive_r = (1.0 + k) * *right / (1.0 + k * right.abs());

*left = drive_l;
*right = drive_r;
```

Fonte: elaborado pelo autor

1.1 DESCRIÇÂO DO PROBLEMA

Dado o conhecimento técnico necessário na área de física, matemática, programação e música, os efeitos digitais e DAWs disponíveis no mercado possuem preços altos, sendo inacessível para novos músicos. Com isso, se torna necessário o desenvolvimento de uma aplicação gratuita e *open-source*, assim possibilitando novos interessados no mundo da música a não terem que arcar com altos custos de entrada e permite a contribuição da comunidade musical para o desenvolvimento e avanço do presente trabalho.

1.2 PROPOSTA DE SOLUÇÂO

Tendo em mente os altos custos e a complexidade previamente citados, o presente trabalho tem como objetivo geral criar um efeito digital *open-source* gratuito por meio do uso do protocolo VST3, a terceira versão e mais atualizada versão do protocolo VST. O programa simulará efeitos fundamentais como distorção, reverberação, eco e equalização, mantendo uma interface simples e dando total controle ao usuário sobre como combiná-los. Com a intenção de concluir o objetivo geral, foram definidos os objetivos específicos, eles são:

- Implementar um efeito digital utilizando a biblioteca nih-plug, o qual é uma abstração do protocolo VST, mais especificamente a versão 3 do mesmo, de uma forma que o programador não precisa se preocupar com os detalhes de mais baixos níveis especificados no capitulo **Erro! Fonte de referência não encontrada.** (HELM, 2023). O protocolo VST será usado, pois é amplamente implementado em diversas DAWs e é o padrão da indústria musical.
- Implementar efeitos básicos como *Drive*, *Fuzz*, *Gain Booster*, *Bit Crush*, *Low Pass*, *Band Pass*, *High Pass*, *Delay*, *Twin Delay* e *Studio Reverb*
- Disponibilizar o efeito no GitHub[3] sob a licença GPLv3[4], firmando o *software* como *open-source*

O próximo capitulo tem como objetivo descrever e introduzir os conceitos computacionais, matemáticos e musicais para a efetiva compreensão do trabalho aqui descrito.

[3] Site *web* que permite o armazenamento de código-fonte
[4] *GNU General Public License v3.0*

2. REFERENCIAL TEÓRICO

Este capitulo ira introduzir conceitos importantes de processamento de sinais com computadores, assim como irá apresentar como os efeitos são calculados e como os efeitos digitais são implementados computacionalmente, porém, antes de apresentar os conteúdos mencionados, é necessário esclarecer dois conceitos, as unidades de volume e amplitude, e a diferença entre eco e reverberação

Começando com o volume, ele normalmente é expresso em decibéis (dB), a qual é uma unidade logarítmica usada para expressar a relação entre duas quantidades, geralmente em termos de potência ou amplitude, já a amplitude refere-se à magnitude ou intensidade de uma onda. No contexto das ondas sonoras, a amplitude está relacionada à altura (volume) do som, em termos mais técnicos, a amplitude é a medida do deslocamento máximo de uma onda a partir de sua posição de equilíbrio, normalmente zero. Quanto maior a amplitude, mais alto será o som ou a intensidade do sinal. Para se obter o valor em decibéis de um sinal deve ser utilizado a formula abaixo.

$dB = 10 * \log_{10}(\frac{P}{P_0})$, onde P é a potência medida e P_0 é uma potência de referência.

A diferença entre eco e reverberação se dá na quantidade de atraso presente no som e a percepção humana sobre o som. O cérebro humano entende dois sons como sendo distintos caso haja por volta de 50 milissegundos de atraso entre eles, logo, para o cérebro perceber um som como eco, ele deve demorar mais de 50 milissegundos até retornar para o ouvinte, senão ele será percebido como reverberação.

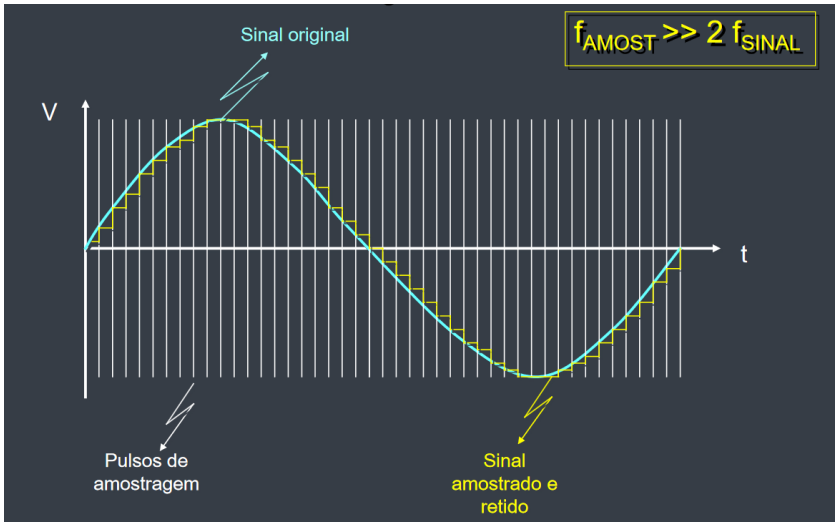
2.1 COMPUTADORES E SINAIS

A maneira que um sinal de áudio é processado em um computador pode ser quebrada em três etapas: conversão de analógico para digital; processamento do quadro pelos processos abertos no sistema operacional; conversão de digital para analógico. A última etapa não será discutida, pois não é de importância para este trabalho, visto que apenas lida com a reprodução do sinal para o usuário final.

A primeira etapa depende da taxa de amostragem (*sample rate*), definida pelo *driver* de áudio do sistema operacional, ela é uma medida dada em Hertz e define quantas amostragens do sinal analógico serão feitas por segundo, cada amostra representa a amplitude do sinal de entrada em um determinado momento. Conforme a conversão é feita, as amostras são guardadas em um *buffer* de tamanho fixo, o qual quando preenchido, será distribuído pelo sistema operacional aos processos abertos. O tamanho do *buffer* influencia a latência e a carga de processamento imposta sobre o processador, isto é, quanto menor o seu tamanho, menor é a latência, porem o processador precisa fazer mais chamadas, aumentando a chance do som “pipocar”. Esses dois valores são de extrema importância, pois o *sample rate* é utilizado para calcular o intervalo de tempo entre duas amostras distintas, algo importante para efeitos baseados em tempo como a reverberação e eco, e o tamanho do *buffer* determina quantas amostras um efeito pode computar em uma única iteração

de sua função *process*. Visto que a audição humana começa em 20Hz e termina em 20kHz, valores para o *sample rate* normalmente são 44100Hz ou 44800Hz, pois, segundo o teorema de Nyquist, o *sample rate* deve ser o dobro da frequência que se deseja converter, se não ocorrerá perda de qualidade nas frequências mais agudas (SCHIABEL, 2023). O processo descrito na primeira etapa pode ser visualizado na Figura 3. A onda senoidal azul é o sinal analógico e o sinal amarelo é o sinal digitalizado.

Figura 3 - Digitalização de um sinal analógico



Fonte: (SCHIABEL, 2023)

Na segunda etapa é feita o processamento do sinal em si. Os programas recebem os blocos e processam eles como desejar, e, quando finalizado, o bloco processado pelo programa é adicionado ao sinal de saída, sendo nessa etapa em que o presente trabalho atua.

2.2 STATE VARIABLE FILTERS

Todos os tipos de filtros dependem de uma frequência de corte para delimitar a frequência em que eles irão agir. Essa frequência pode ser expressa tanto como um valor em Hertz, tanto como um valor normalizado entre 0 e a frequência Nyquist do *sample rate* atual.

Um filtro de áudio de estrutura de estado variável (SVF, do inglês *State Variable Filter*) é um tipo de filtro usado em processamento de áudio para modificar características espectrais de um sinal de áudio. Ele é tem esta classificação de estado variável porque pode operar em diferentes modos, como passa-baixas, passa-altas e passa-bandas, dependendo da configuração dos parâmetros. Este tipo de filtro é particularmente flexível e versátil.

Os filtros SVF são frequentemente utilizados para implementar filtros de 2ª ordem, proporcionando uma estrutura modular e flexível. Um filtro de segunda ordem refere-se a um tipo específico de filtro que possui uma resposta de frequência que é caracterizada por uma atenuação ou aumento da amplitude da resposta em uma taxa de 12 dB por oitava. Isso significa que, ao se mover para o dobro (ou a metade) da frequência de corte, a amplitude do sinal é reduzida (ou aumentada) em 12 decibéis.

O processamento de um SVF inicia com o cálculo de coeficientes, o qual depende da frequência de corte, do parâmetro *Q*, ou ressonância, e do tipo de filtro desejado, tai como passe-baixo, passe-alto e etc. A variável T nos próximos cálculos é o intervalo de amostragem, ou seja, o intervalo de tempo entre cada amostra.

Com os coeficientes do filtro calculados, é feito o estágio de integração, o qual tem como objetivo fornecer uma aproximação da integral do sinal de entrada $x[n]$. O seu cálculo é descrito pela formula:

$$v_0[n] = v_0[n-1] + \frac{T}{2Q} * (x[n] - v_2[n-1])$$

A próxima fase é o de atraso, o qual atrasa o sinal integrado $v_0[n]$ por um intervalo de tempo T. Esse processo é descrito pela formula:

$$v_1[n] = v_1[n-1] + T * v_0[n]$$

Depois disso vem a fase de combinação, a qual combina os resultados dos estágios de integração e atraso para produzir a saída final do filtro. BP Gain e LP Gain são ganhos ajustados com base no modo do filtro.

$$y[n] = BP\ Gain * v_0[n] + x[n] - LP\ Gain * v_1[n]$$

2.3 INFINITE IMPULSE REPONSE FILTERS

Um filtro IIR (*Infinite Impulse Response*) *Butterworth* é um tipo comum de filtro digital que é projetado para ter uma resposta de magnitude plana na banda passante[5] e uma transição suave para a banda de rejeição[6]. A resposta em frequência de um filtro Butterworth é caracterizada por ter uma curva de atenuação mais suave em comparação com outros tipos de filtros.

A implementação de um filtro IIR *Butterworth* geralmente é feita usando estruturas de seções de filtro de segundo grau, conhecidas como seções *biquad*. Cada seção *biquad* é responsável por uma parte específica da resposta de frequência total do filtro. A função de transferência de um filtro Butterworth de n-ésima ordem pode ser decomposta em $\frac{n}{2}$ seções *biquad*.

A função de transferência[7] geral de um filtro IIR Butterworth é dada por:

$$H(z) = \frac{1}{(1 - a_1 z^{-1} + a_2 z^{-2}) * (1 - b_1 z^{-1} + b_2 z^{-2}) * \ldots * (1 - c_1 z^{-1} + c_2 z^{-2})}$$

Cada seção biquad tem a seguinte forma geral:

$$H_{biquad}(z) = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2}}{1 + a_1 z^{-1} + a_2 z^{-2}}$$

Os coeficientes a_1, a_2, b_0, b_1, e b_2 são os parâmetros do filtro que determinam suas características de resposta em frequência. Esses coeficientes podem ser calculados a partir das especificações do filtro (frequência de corte, banda de passagem, banda de rejeição, etc.) usando métodos de design de filtro digital.

A implementação prática de um filtro IIR *Butterworth* em código, demonstrado na Figura 4, geralmente envolve a utilização de uma estrutura de dados para armazenar os estados passados do filtro, bem como funções para atualizar esses estados e calcular a saída do filtro a cada nova amostra de entrada.

Figura 4 - Calculo de *biquad* em python

```python
def filter(self, x):
    y = self.b0 * x + self.b1 * self.x1 + self.b2 * self.x2 - self.a1 * self.y1 - self.a2 * self.y2
    self.x2 = self.x1
    self.x1 = x
    self.y2 = self.y1
    self.y1 = y
    return y
```

Fonte: elaborado pelo autor

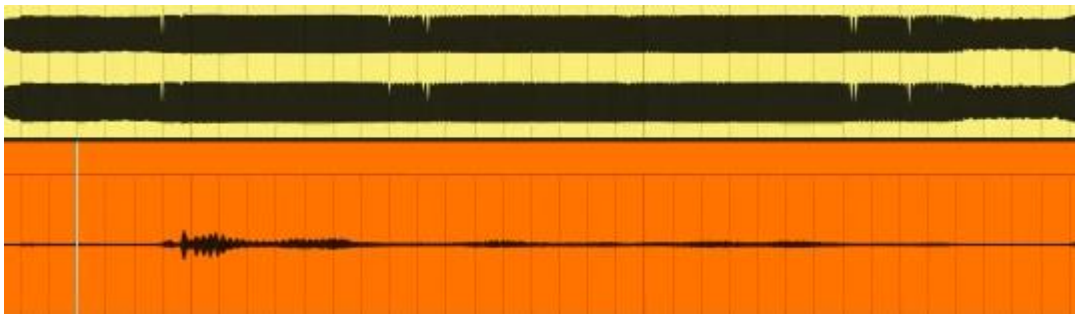
[5] A banda passante é a parte do sinal que não será afetada pelo filtro
[6] A banda de rejeição é a parte do sinal que será afetado pelo filtro
[7] A função de transferência descreve a relação matemática entre a entrada e a saída de um sistema dinâmico.

2.4 DISTORÇÂO

Este capitulo irá definir conceitos importantes relacionados à distorção, tais como distorção assimétrica e funções de recorte, assim como será explicado fenômeno por de trás da distorção

A distorção ocorre quando a amplitude de um sinal excede a amplitude que o dispositivo de reprodução consegue reproduzir, fazendo com que o som seja “esmagado” no processo. Esse fenômeno é visível quando se analisa o formato de onda de um sinal antes de ser distorcido e após ser distorcido. A Figura 5 demonstra um sinal antes e depois de ser distorcido, sendo o laranja o sinal antes da distorção e o amarelo após a distorção.

Figura 5 - Fenômeno de distorção

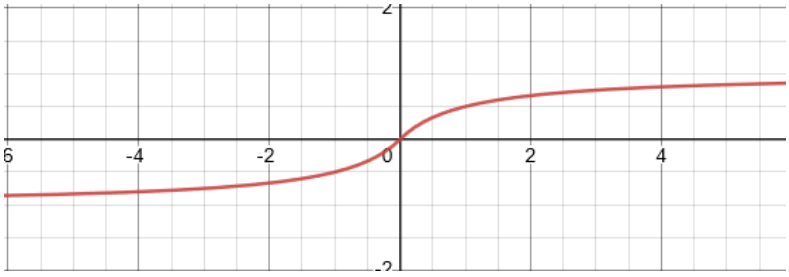


Fonte: elaborado pelo autor

Pode se observar que além do volume estar mais alto, o formato da onda se assemelha a um retângulo e é graças a essa mudança que o timbre[8] muda.

Com a finalidade de reproduzir esse fenômeno de forma controlada, pode ser utilizado uma função de recorte, ou *clipping*. Uma função de *clipping* limita o volume entre -1.0 e 1.0. A Figura 6 representa o gráfico da função $y = \frac{x}{1+|x|}$.

Figura 6 - Gráfico representando uma função de clipping



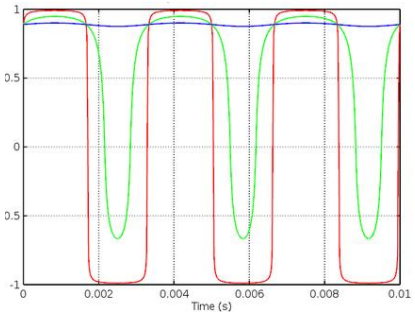
Fonte: elaborado pelo autor

[8] O timbre é som característico de um instrumento, sendo o elemento que torna possível distinguir um piano e uma guitarra tocando a mesma nota.

O eixo Y representa o volume de saída e o eixo X representa o volume de entrada, podendo ser observado que quando mais alto o valor de x, mais próximo a 1 o valor de y ficara. Por meio dessa função é possível forçar o fenômeno de distorção a ocorrer em um sinal sem a necessidade de aumentar o volume do mesmo (ZDSP, 2017).

A distorção assimétrica ocorre quando for somado um valor ao sinal, fazendo com o que eixo Y se desloque, e então é aplicado a função de *clipping*, caso o valor somado seja positivo, as partes com fase[9] positiva serão mais distorcidas do que as de fase negativa. Esse fenômeno é utilizado em efeitos como o *Fuzz*, o qual será abordado futuramente no presente trabalho, porem ele possui um problema, a adição de um valor constante à amplitude do sinal faz com que a função dependa do volume do sinal. A Figura 7 demonstra esse problema em ação podendo se observar que a onda verde está adequadamente afetada, mas as ondas vermelha e azul não reagiram ao efeito.

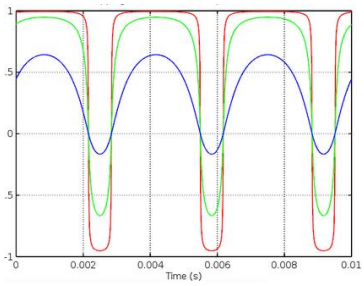
Figura 7 - Clipping com soma de valor constante



Fonte: (ZDSP, 2017)

Para evitar esse problema deve ser utilizado o valor absoluto do sinal em conjunto com um valor constante k, dando origem a equação $x = |x| * \frac{k}{100}, onde\ 0 < k < 100$. Assim invés de ser adicionado um valor constante, é adicionado um valor proporcional a amplitude atual. A Figura 8 mostra a distorção assimétrica sendo calculada corretamente paras os três sinais da Figura 7.

Figura 8 - Distorção assimétrica proporcional



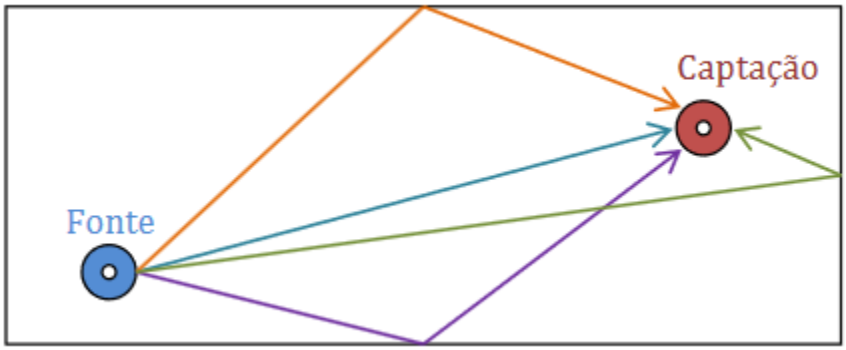
Fonte: (ZDSP, 2017)

[9] A fase de uma amostra determina se ela é positiva ou negativa, ou seja, uma amostra de valor -1.0 e 1.0 possuem a mesma amplitude, mas com fases inversas.

2.5 REVERBERADOR DE SCHROEDER

O reverberador de Schroeder, nomeado após o engenheiro acústico Manfred R. Schroeder, é um tipo especifico de algoritmo de reverberação. O funcionamento desse algoritmo depende de quatro conceitos, os *comb filters*, realimentação, *Allpass filters*, e decaimento exponencial (SMITH, 2007). O intuito desse algoritmo é simular a reverberação de uma sala fechada através das reflexões geradas pelas paredes, como a Figura 9 demonstra.

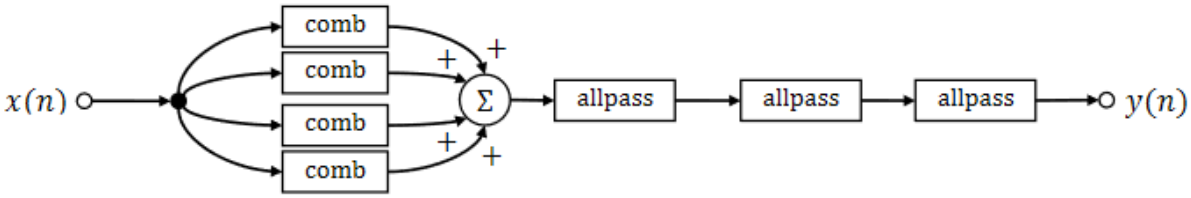
Figura 9 - Reverberação dentro de uma sala



Fonte: (TASCHETTO, 2009)

Os *comb filters* em paralelo criam diferentes atrasos na mesma fonte sonora, onde cada *comb* representa um caminho percorrido pelo som refletido nas paredes. Os atrasos destes filtros representam o "tamanho" do ambiente. Os filtros *allpass*[10] em série simulam a superfície refletida, dando um aspecto mais denso ao som com o deslocamento de fase, mas mantendo a magnitude de suas componentes espectrais inalteradas. A formula geral de um reverberador de Schroeder é ilustrada na Figura 10.

Figura 10 - Formula geral do reverberador de Schroeder



Fonte: (TASCHETTO, 2009)

[10] Não alteram a magnitude de nenhuma frequência. Porém, alteram a fase de diversas componentes espectrais. Ver Zölzer (2002, p. 33-34).

2.6 PROTOCOLO VST

O protocolo VST pode ser quebrado em duas partes, o *host* e o efeito em si. O *host* é encarregado de gerenciar as entradas e saídas de áudios, enviar as configurações de áudio do usuário para os efeitos e manejar os efeitos atualmente carregados.

Na parte do efeito são definidas uma série de funções que devem estar presentes para que possa ocorrer a comunicação entre o *host* e o efeito sendo desenvolvido. As funções que serão utilizadas são a *process* encarregada de processar o sinal proveniente do *host*, *initialization* encarregada de iniciar o efeito e *user interface* que se encarrega de mostrar a interface gráfica do efeito. O efeito é compilado em um DLL (*Dynamically Linked Library*) no Windows, e no Linux em um SO (*Shared Object*), ambos tipos de arquivos são bibliotecas compartilhadas, o motivo disso é que esses arquivos binários precisam ser carregados e acessados pelo *host* sem ter que estarem compilados junto ao mesmo. O *host* então se comunica com o efeito através das funções definidas no protocolo VST previamente citadas, assim tornando possível a criação de um ecossistema em que os *hosts* e os efeitos podem se comunicar livremente e de forma interoperável (STEINBERG, 2022).

Vale mencionar que efeitos VST funcionam, com suas exceções, em qualquer sistema operacional utilizando-se de uma técnica chamada *bridging*. Quando essa ferramenta é utilizada pode se obter uma DLL a partir de um SO ou vice-versa. Esse processo é muito utilizado para converter VSTs feitos originalmente para Windows no Linux.

A próxima seção irá explicar como é feito a arquitetura de um efeito digital que implementa o protocolo VST, assim como ira esclarecer algumas dificuldades no desenvolvimento de sistemas em tempo real.

2.7 ARQUITETURA DE UM EFEITO DIGITAL E REALTIME SAFETY

O sistema depende de duas *threads* principais, princípio da programação denominado *multithreading*, sendo uma delas encarregada do processamento de som, a qual sempre existirá enquanto o programa estiver aberto, e outra encarregada da exibição da interface gráfica, tendo sua execução dependente se o editor do efeito está aberto. Ambas *threads* são executadas em paralelos e a *thread* encarregada de exibir a interface jamais deve interromper a *thread* de áudio. Deve ser evitado quaisquer operações *non-realtime safe*, ou seja, operações com tempo de execução variável dependendo de fatores externos. Operações como alocar memória na *Heap*[11] não são *realtime-safe*, pois depende do tamanho da memória sendo requisitada e do tempo de resposta do sistema operacional (HELM, 2023).

[11] A memória Heap é uma região de memória dinâmica usada para alocar espaço para objetos durante a execução de um programa de computador. Ao contrário da memória estática, que é alocada durante a compilação e permanece fixa durante a execução do programa, a memória Heap é alocada e desalocada dinamicamente durante a execução do programa.

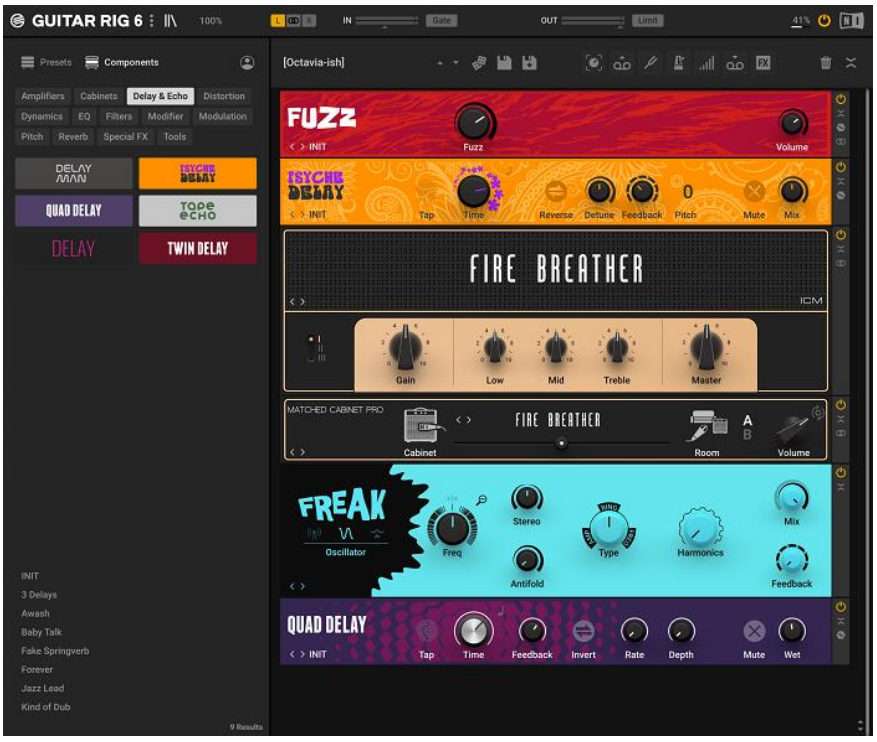
3. TRABALHOS RELACIONADOS

Serão discutidos três VSTs reconhecidos por sua capacidade de emular amplificadores e efeitos de guitarra de forma realista: Guitarix, Guitar Rig 6 e Neural DSP. Enquanto essas três ferramentas oferecem funcionalidades de processamento de áudio poderosas para músicos e produtores, elas possuem diferenças de interface, funcionalidades, preço e compatibilidade com diferentes sistemas operacionais.

3.1 GUITAR RIG 6

Guitar Rig 6 foi feito por uma grande empresa na área de efeitos digitais, a *Native Instruments*, responsável por outros programas famosos como Kontakt e Traktor. É um programa capaz de ser executado tanto dentro de uma DAW, quanto como um programa independente. Apresenta uma grande variedade de efeitos e amplificadores, uma interface simples de se entender, porém não apresenta suporte no Linux, requerendo o uso de ferramentas de *bridging.* Sua licença completa custa 200 dólares. A Figura 11 demonstra a interface do Guitar Rig 6 com alguns efeitos carregados.

Figura 11 - Interface gráfica do Guitar Rig 6

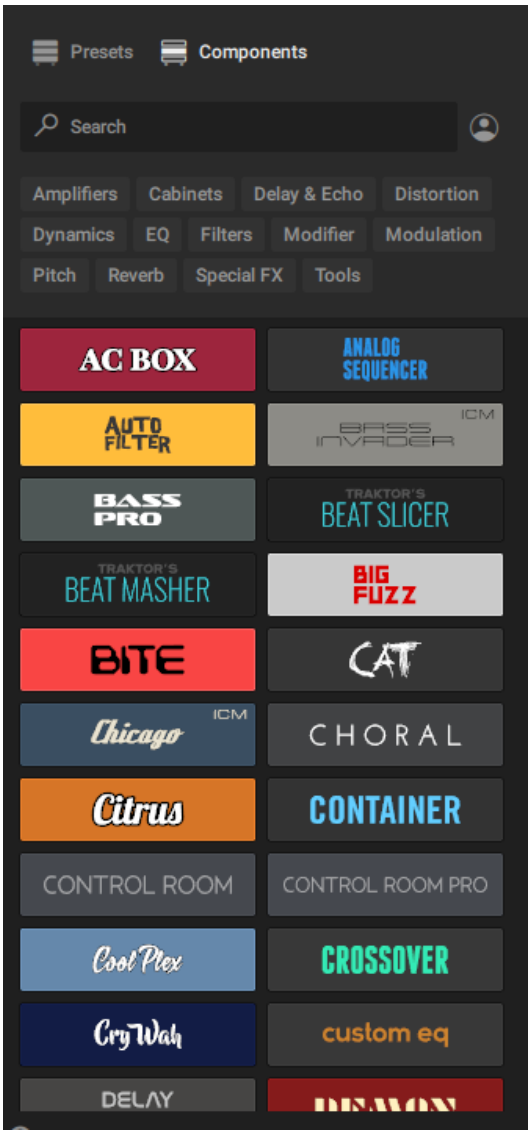


Fonte: Site oficial da Native Instruments[12]

[12] Disponível em: <https://www.native-instruments.com/en/products/komplete/guitar/guitar-rig-6-pro/>. Acesso em 26 jun. 2023

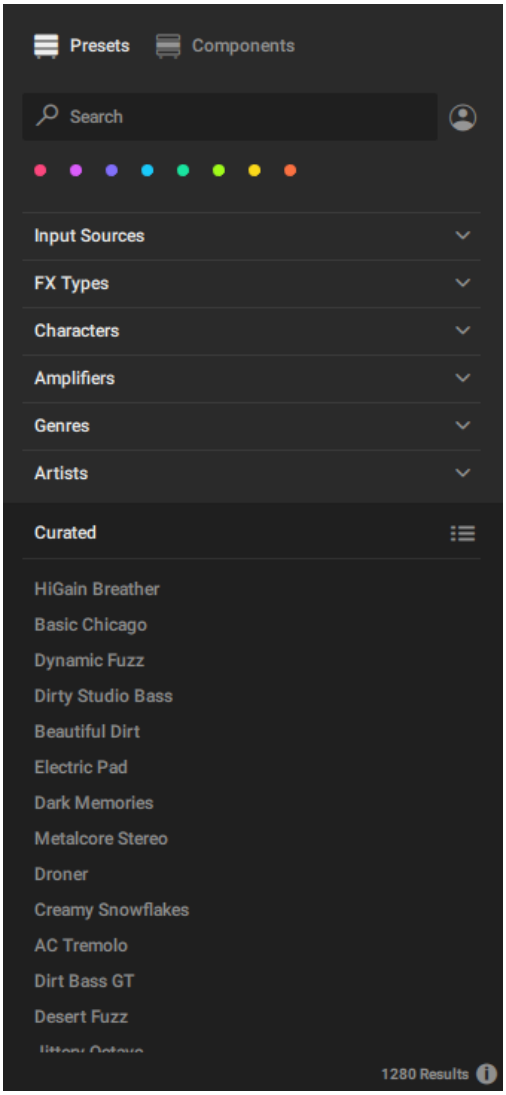
O programa apresenta uma barra lateral que possui dois modos, uma lista dos efeitos disponíveis divididos em categorias, demonstrada na **Erro! Fonte de referência não encontrada.**, e uma lista de configurações predefinidas incluídas com o programa ou feitas pelo próprio usuário, demonstrada na **Erro! Fonte de referência não encontrada.**.

Figura 12 - Aba de efeitos do Guitar Rig 6



Fonte: elaborado pelo autor

Figura 13 - Aba de predefinições do Guitar Rig 6



Fonte: elaborado pelo autor

A interface segue uma lógica de arrastar e soltar. O usuário arrasta um efeito para a direita da tela, o qual é adicionado na posição que o usuário o soltou. Após isso é possível ajustar diversos valores usando os diversos botões presentes na interface de cada efeito. A ordem de preferência do programa se dá de cima para baixo, os efeitos de cima são processados e aplicados primeiro do que os debaixo, como demonstrada na Figura 14.

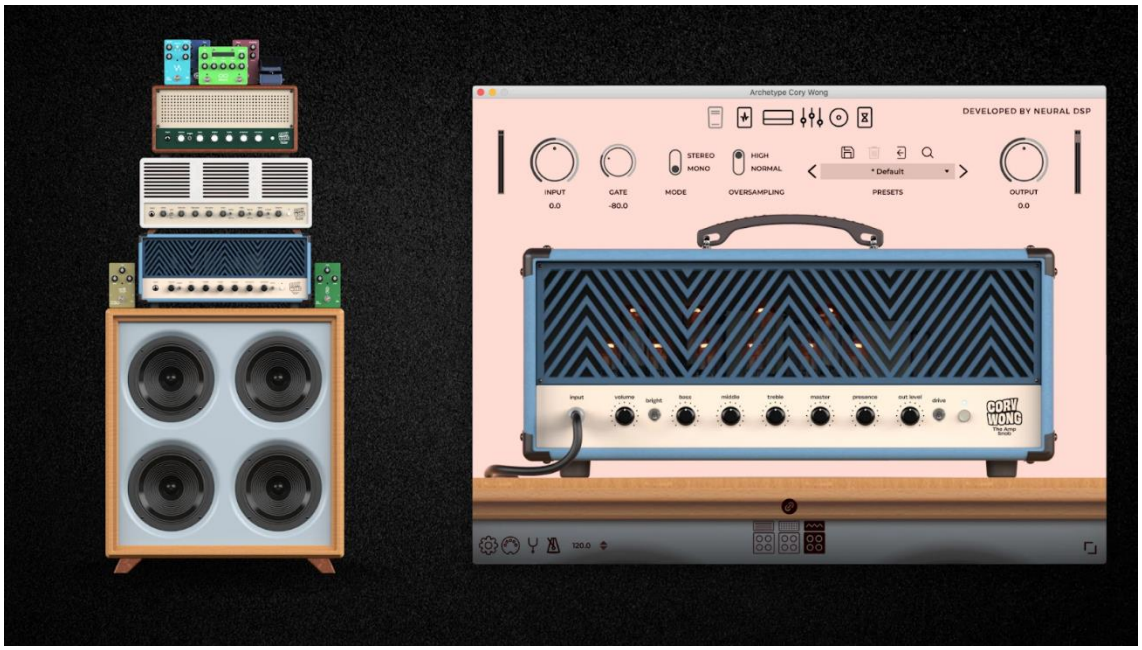
Figura 14 - Lista de efeitos do Guitar Rig 6

Fonte: elaborado pelo autor

3.2 NEURAL DSP

O Neural DSP funciona de forma diferente ao Guitar Rig 6. Ele é quebrado em diferentes *Archetypes* (arquétipos) que imitam os timbres de guitarristas atualmente famosos. Também apesenta uma interface simples e elegante, porém apresenta preços ainda mais elevados do que os previamente mencionados, tendo seus arquétipos variando entre 100 a 150 euros. A Figura 15 demonstra uma ilustração com a interface do programa ao lado.

Figura 15 - Interface do Neural DSP



Fonte: Site oficial do Neural DSP[13]

[13] Disponível em: <https://neuraldsp.com/>. Acesso em 26 jun. 2023.

O programa é dividido em seções, sendo cada uma delas responsável por um tipo diferente de efeito, a barra com as diferentes seções é demonstrada na Figura 16. É possível desativar ou ativar cada seção individualmente com um clique.

Figura 16 - Barra de seções do Neural DSP

Fonte: elaborado pelo autor

Assim como o programa previamente citado, o controle dos efeitos se dá através de botões, como mostra a Figura 17, porém a ordem na qual os efeitos são aplicados é determinada pelo programa. Também não é possível encadear diversos efeitos, visto que cada um apenas aparece uma vez em sua própria seção.

Figura 17 - Controle de parâmetros do Neural DSP

Fonte: elaborado pelo autor

3.3 GUITARIX

O Guitarix é um programa gratuito, porém tem uma quantidade menor de efeitos e sua interface é menos limpa e fácil de entender do que os previamente citados, sendo somente suportado no Linux e não funciona de forma convencional quando usado em alguns hospedeiros, exibindo cada efeito separadamente invés de serem contidos em um único efeito digital. A Figura 18 mostra a interface do Guitarix com alguns efeitos carregados.

Figura 18 - Interface do Guitarix

Fonte: Site oficial do Guitarix[14]

O programa possui uma barra lateral com um menu *drop-down*[15] para cada tipo de efeito. Diferente do Guitar Rig 6, cada efeito não possui seu próprio ícone, mas ainda é retido a funcionalidade de arrastar e soltar para adicionar e controlar a ordem dos efeitos dentro do programa. Assim como os outros trabalhos citados, o controle dos efeitos se dá através de botões deslizantes. Ele também possui um controle de configurações predefinidas, porém possui uma funcionalidade extra na qual o usuário pode baixar predefinições feitas por outros usuários.

[14] Disponível em: <https://guitarix.org/> . Acesso em: 26 jun. 2023

[15] Menu que ao receber um clique do usuário apresenta seu conteúdo, tornando-se uma forma simples e elegante de compactar e agrupar informações.

O Quadro 1 compara as qualidades dos efeitos previamente citados juntamente com o presente trabalho.

Quadro 1 - Comparação entre os diferentes trabalhos relacionados

	Fretcat	Neural DSP	Guitar Rig 6	Guitarix
Preço	Gratuito	100-150 EUR	200 USD	Gratuito
Clareza da interface	Boa	Excelente	Boa	Ruim
Sistemas operacionais suportados	Windows e Linux	Windows e Linux (*bridging*)	Windows e Linux (*bridging*)	Somente Linux
Open-source	Sim	Não	Não	Sim
Permite o usuário combinar efeitos livremente	Sim	Não	Sim	Sim
Variedade de efeitos	Apresenta os efeitos essenciais, com variações.	Variados, porém divididos em arquétipos.	Grande quantidade de efeitos incluídos em um só pacote, tendo várias variações dos mesmos efeitos.	Apresenta os efeitos essenciais, com variações.

Fonte: elaborado pelo autor

Os efeitos essenciais são os mais usados nas músicas famosas, como: distorção, reverberação, eco e filtragem. As variações seriam mudanças nesses efeitos, como por exemplo um efeito de reverberação de mola ou um de placa. Ambos emulam o mesmo efeito, porém oferecem timbres diferentes.

O próximo capitulo irá abordar a metodologia utilizada de pesquisa utilizada, assim como irá abordar as ferramentas e tecnologias adotadas para o cumprimento dos requisitos que foram levantados.

4. METODOLOGIA

Para desenvolver o programa, foi adotado uma abordagem de pesquisa aplicada, pois o intuito do trabalho é integrar várias tecnologias já existentes em um único pacote gratuito e simples de utilizar (GIL, 2019).

Como base para a pesquisa, foram analisadas e compiladas as funcionalidades mais importantes e recorrentes, assim como problemas que, ou complicam o uso do programa, ou o tornam inutilizável. Feito esse procedimento foi determinado o escopo do programa, assim como as tecnologias que seriam utilizadas. As funcionalidades selecionadas foram: carregar efeitos em qualquer ordem e quantidade, salvar e carregar predefinições e uma interface arrasta-solta simples e intuitiva. A principal tecnologia selecionada foi o protocolo VST3, visto sua grande abrangência e por ser o padrão da indústria de música.

Após a análise de requisitos, foi pensado na estrutura do trabalho. Então, foram feitos diagramas de pacotes, classes, atividades e casos de uso para ilustrar as principais funcionalidades, fluxos e estrutura de dados do programa.

4.1 TECNOLOGIAS ADOTADAS

Foi adotada o Rust[16] como linguagem de programação, visto que suas ferramentas de compilação e gerenciamento de pacote são mais ricas e fáceis do que as do C++, a qual é a linguagem principal para o desenvolvimento de efeitos digitais. O protocolo VST3[17] foi selecionado por ser o padrão de indústria, sendo implementado na vasta maioria das DAWs. Como forma de armazenamento de dados foi escolhido o formato JSON[18].

4.2 FERRAMENTAS ADOTADAS

Como forma de agilizar o desenvolvimento, foram adotadas ferramentas facilitadoras, simplificando a implementação das tecnologias mencionadas acima, assim como foram utilizadas ferramentas para a organização do desenvolvimento como um tudo.

A edição do código-fonte se deu através do Visual Studio Code[19], pois é o padrão do mercado. Para o controle de versão do código foi utilizado o Git[20], sendo o repositório armazenado no GitHub[21]. Foi utilizado a biblioteca nih_plug[22] para abstrair a implementação do protocolo VST em conjunto com a biblioteca Vizia[23] pra a construção dos elementos gráficos. Foi utilizado a biblioteca serde-rs[24] para facilitar a serialização e desserialização de arquivos JSON.

[16] Disponível em: https://www.rust-lang.org/pt-BR
[17] Disponível em: https://steinbergmedia.github.io/vst3_dev_portal/pages/index.html
[18] Disponível em: https://www.json.org/json-en.html
[19] Disponível em: https://code.visualstudio.com/
[20] Disponível em: https://git-scm.com/
[21] Disponível em: https://github.com/
[22] Disponível em: https://github.com/robbert-vdh/nih-plug
[23] Disponível em: https://github.com/vizia/vizia
[24] Disponível em: https://serde.rs/

5. MODELAGEM

Neste capitulo serão apresentados os diagramas de classes, os diagramas de atividades e a especificação de cada Caso de uso.

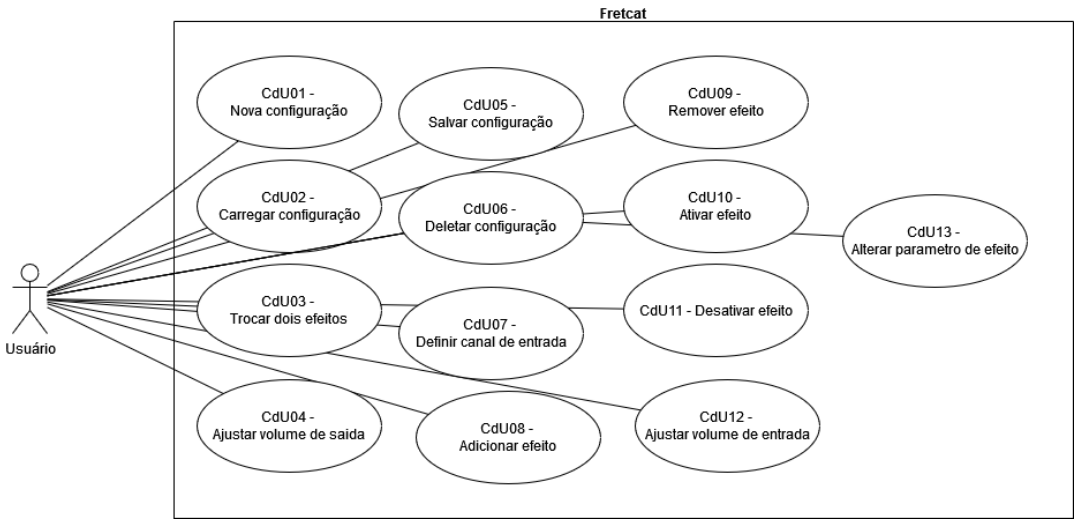
Tendo em mente a natureza *multithreaded* de efeitos digitais, a modelagem do sistema pode ser pensando em duas *threads*, uma que se encarrega dos elementos gráficos, podendo ou não estar sendo executada caso o usuário esteja com o editor aberto, e outra encarregada de processar o sinal de entrada e produzir uma saída. Outra complicação é a necessidade de garantir a execução em tempo real da *thread* de áudio, assim, as alocações de memória devem ser apenas feitas na *thread* gráfica. No que tange o armazenamento de dados, o programa apenas irá guardar predefinições em arquivos JSON localmente, não sendo necessário o uso de um banco de dados.

5.1 DIAGRAMA DE CASOS DE USO

Os diagramas de casos de uso modelam o comportamento de um sistema e ajudam a capturar os requisitos do sistema. Eles descrevem as funções de alto nível e o escopo de um sistema. Esses diagramas também identificam as interações entre o sistema e seus atores. Os casos de uso e atores nos diagramas de casos de uso descrevem o que o sistema faz e como os atores o utilizam, mas não como o sistema opera internamente (IBM, 2021).

O sistema apresenta 13 casos de uso. Os principais casos de uso são: adicionar, remover e trocar efeitos; configurar, carregar, inicializar e salvar predefinições. Outras funcionalidades que aprimoram a experiência do usuário são desativar e ativar efeitos sem removê-los, controle de qual canal estéreo será usado como entrada e ajustes de volume para os sinais de saída e entrada. O sistema possui apenas um ator, sendo ele qualquer pessoa com acesso ao programa. A Figura 19 ilustra os casos de uso do sistema e como eles interagem entre si. A especificação dos casos de uso pode ser conferida no APÊNDICE A – ESPECIFICAÇÂO DE CASOS DE USO.

Figura 19 - Diagrama de caso de uso



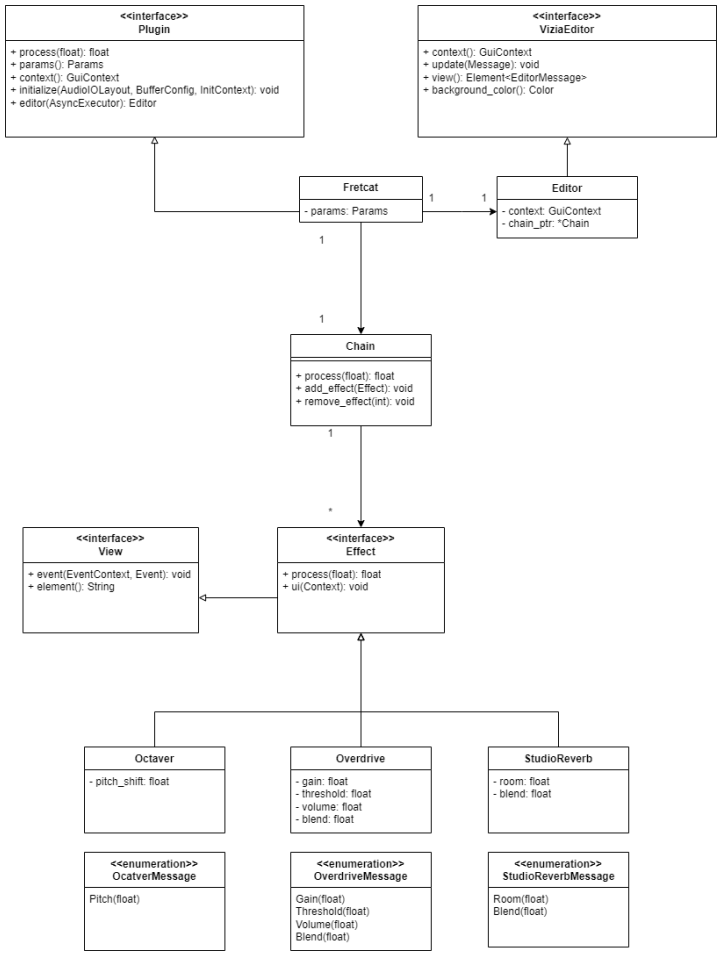
Fonte: elaborado pelo autor

5.2 DIAGRAMAS DE CLASSES

Os diagramas de classe são uma ferramenta essencial na modelagem de sistemas complexos, permitindo aos desenvolvedores compreenderem e comunicarem eficientemente a arquitetura de um software. Sua utilidade vai além da mera documentação, sendo um recurso valioso durante todo o ciclo de vida do desenvolvimento de software, desde a concepção até a implementação e manutenção (IBM, 2021). Foi desenvolvido um diagrama de classe mostrando a relação entre as principais classes do sistema

A Figura 20 ilustra, através do UML, as principais estruturas de dados do programa, sendo elas os efeitos e a cadeia de efeitos. Um efeito é constituído por parâmetros que controlam sua intensidade e uma equação que altera um sinal de entrada. A cadeia de sinais é a estrutura encarregada de manter a ordem dos efeitos e manter o estado atual do programa. Apenas alguns efeitos estão representados no diagrama, pois seria redundante demonstrar todos eles, visto que todos possuem as mesmas relações.

Figura 20 - Diagrama de classes



Fonte: elaborado pelo autor

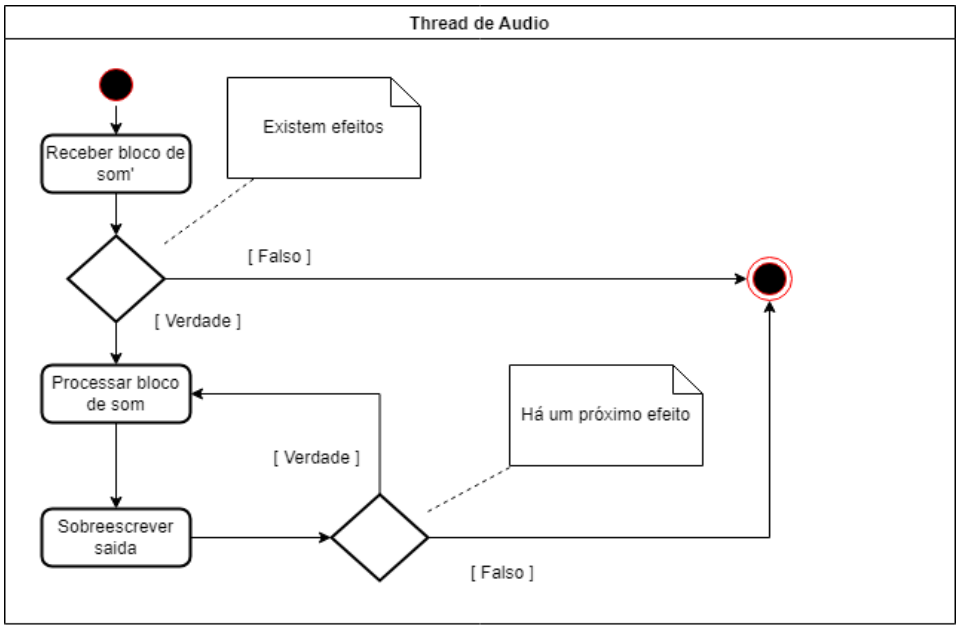
5.3 DIAGRAMAS DE ATIVIDADES

O sistema proposto também conta com um diagrama de atividades, o qual fornece uma visualização do comportamento do sistema descrevendo a sequência de ações em um processo, sendo semelhante a um fluxograma, visto que ilustra o fluxo entre ações em uma atividade (IBM, 2021). Para o Fretcat foram elaborados quatro diagramas de atividade, os quais podem ser visualizados nas figuras subsequentes.

O programa pode ser resumido da seguinte forma, em uma *thread* denominada *thread* de áudio, para cada bloco do sinal de entrada recebido, é chamada a função de processamento de todos os efeitos presentes na cadeia. Em outra *thread* denominada *thread* gráfica, a interface gráfica é criada com base nos efeitos atualmente carregados, adições, remoções ou alterações de efeitos serão refletidos no processamento da *thread* de áudio.

A Figura 21 representa a atividade de processamento de áudio do programa. Essa atividade é executada durante o programa inteiro.

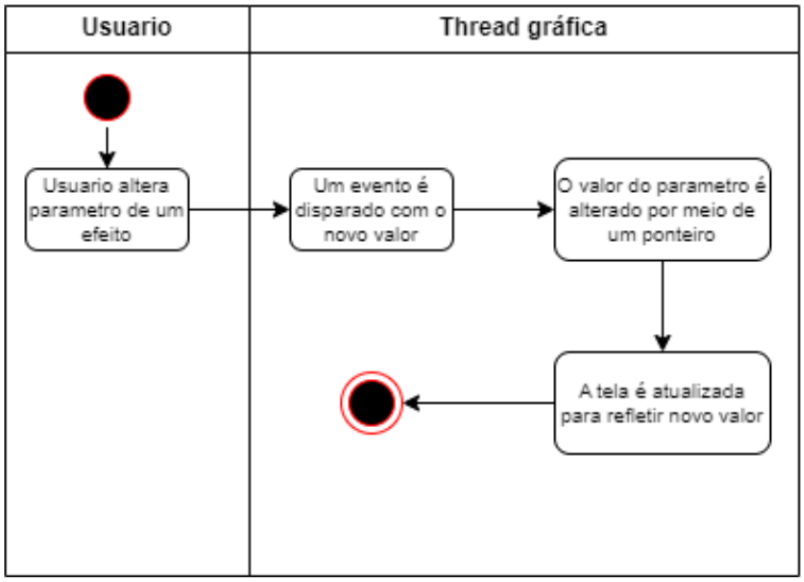
Figura 21 - Diagrama de atividade do processamento de áudio



Fonte: elaborado pelo autor

O diagrama ilustrado na Figura 22 demonstra como ocorre a alteração de um parâmetro de um efeito. O usuário interage com tela, disparando um evento. Esse evento carrega o valor novo daquele parâmetro que então é atualizado por meio de um ponteiro para o efeito na cadeia e também é atualizado na tela.

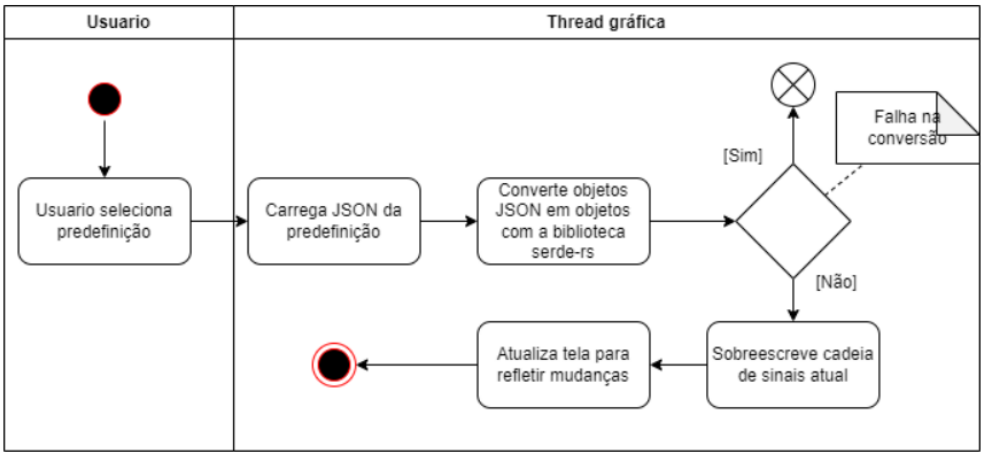
Figura 22 - Diagrama de atividade de alteração de parâmetro de efeito



Fonte: elaborado pelo autor

O diagrama ilustrado na Figura 23 representa o carregamento de uma predefinição. A atividade começa com uma ação do usuário, em seguida o sistema carrega o JSON daquela predefinição e tenta convertê-lo em instâncias dos efeitos. Caso ocorra alguma falha na conversão, seja por erro de sintaxe ou uma estrutura de dados errada, a atividade termina e a cadeia atual é preservada.

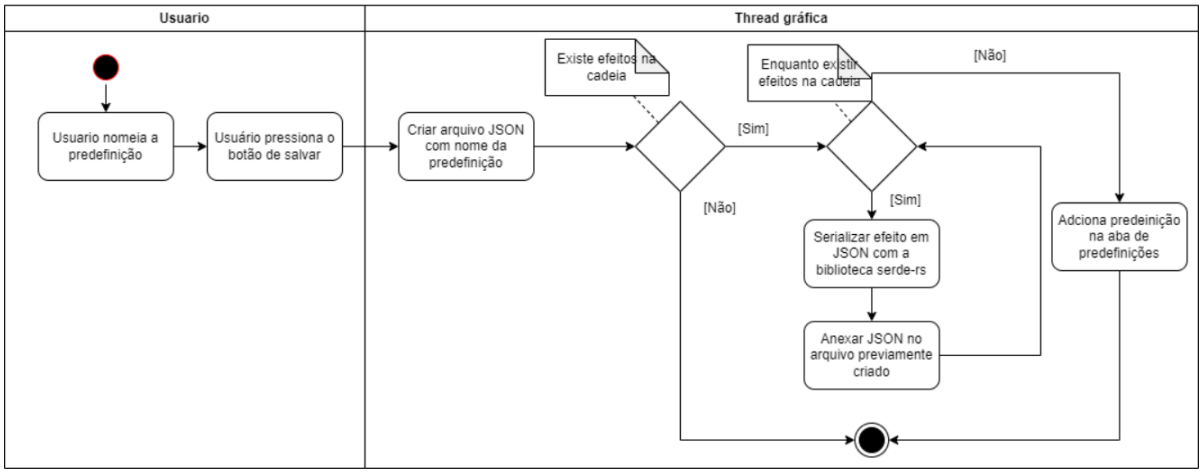
Figura 23 - Diagrama de atividade de carregar predefinições



Fonte: elaborado pelo autor

O diagrama representado na Figura 24 representa a atividade de salvar predefinições. O usuário nomeia e então pressiona o botão de salvar, iniciando o processo de salvamento. O sistema irá criar um arquivo com o nome da predefinição e então, caso existam efeitos carregados, cada efeito será serializado no formato JSON e adicionado ao arquivo previamente criado. Após essa operação, será adicionado à aba de predefinições a nova predefinição.

Figura 24 - Diagrama de atividade de salvar predefinição



Fonte: elaborado pelo autor

6. SISTEMA FRETCAT

Neste capitulo será descrito como está organizada a interface do sistema em conjunto com a descrição de cada efeito aqui implementado. O presente trabalho resultou em uma aplicação *desktop* com o objetivo de simular pedaleiras de guitarra de forma acessível e intuitiva.

6.1 ALGORITMOS

Este capitulo ira abordar como foi implementado cada efeito presente no sistema, demonstrando os cálculos feitos durante o processamento de áudio. Primeiramente será apresentado o como é feito o cálculo dos filtros, pois estão presentes em mais de um efeito, após isso será abordado cada efeito individualmente, excluindo o efeito *Gain Booster*, visto que ele é apenas um ganho simples.

6.1.1 FILTROS

Neste capitulo será abordado como é feito o processamento dos três tipos de filtros utilizados no Fretcat, os SVF, *Butterworth Lowpass* e *DCBlock*.

A Figura 25 demonstra a função *tick* de um SVF, cada chamada a função *tick* retorna um valor novo filtrado e atualiza o estado interno do filtro. Os filtros SVF presentes no programa foram modelados por Simper. Os valores a1, a2, a3, m0, m1, m2 são os coeficientes do filtro e os valores ic2eq, ic1eq são os valores calculados previamente pelo filtro.

Figura 25 - Implementação do filtro SVF

```rust
#[inline]
pub fn tick(&mut self, sample: f32) -> f32 {
    let v0 = sample;
    let v3 = v0 - self.ic2eq;
    let v1 = self.coeffs.a1 * self.ic1eq + self.coeffs.a2 * v3;
    let v2 = self.ic2eq + self.coeffs.a2 * self.ic1eq + self.coeffs.a3 * v3;
    self.ic1eq = 2.0 * v1 - self.ic1eq;
    self.ic2eq = 2.0 * v2 - self.ic2eq;
    self.coeffs.m0 * v0 + self.coeffs.m1 * v1 + self.coeffs.m2 * v2
}
```

Fonte: elaborado pelo autor

A Figura 26 demonstra a função *tick* de um *Butterworth Lowpass*, cada chamada a essa função retorna o valor novo filtrado e atualiza o estado interno do filtro. A diferença entre o *Butterworth Lowpass* e um SVF *Lowpass* se dá nos coeficientes, sendo o *Butterworth Lowpass* mais preciso do que o SVF. Os valores a1, a2, b0, b1, b2 são os coeficientes do filtro e os valores x1, x2, y1, y2 são os valores previamente processados pelo filtro.

Figura 26 - Código de processamento de um filtro *Butterworth Lowpass*

```rust
#[inline]
pub fn tick(&mut self, input: f32) -> f32 {
    let x0 = input;
    let y0 = self.coefs.b0 * x0 + self.coefs.b1 * self.x1 + self.coefs.b2 * self.x2
        - self.coefs.a1 * self.y1
        - self.coefs.a2 * self.y2;
    self.x2 = self.x1;
    self.x1 = x0;
    self.y2 = self.y1;
    self.y1 = y0;
    y0
}
```

Fonte: elaborado pelo autor

Por fim, ilustrado na Figura 27, temos o *DCBlock*, sendo ele o mais simples dos três. O *DCBlock* tem como objetivo compensar as oscilações indesejadas de fase acima e baixo da base zero[25]. O valor y0 representa o valor de deslocamento de amplitude a ser compensado, e os valores x1 e y1 são os valores previamente calculados pelo filtro.

Figura 27 - Implementação DCBlock

```rust
#[inline]
pub fn tick(
    &mut self,
    input: f32,
) -> f32 {
    let x = input;
    let y0 = x - self.x1 + self.coeff * self.y1;
    self.x1 = x;
    self.y1 = y0;
    y0
}
```

Fonte: elaborado pelo autor

[25] A base zero refere-se ao que seria o meio do sinal. Idealmente todo sinal deve ser centrado em torno do eixo X 0 para evitas problemas de fase.

6.1.2 DISTORÇÔES

Neste capitulo será demonstrado como foram feitas as implementações em código dos fenômenos descritos no capitulo 2.4.

O efeito de *Drive* é um efeito de distorção simétrica, sendo composto por dois filtros *DCBlock*, uma função de *clipping*, um controle de ganho e dois filtros *Butterworth Lowpass* para o controle da tonalidade. A Figura 28 demonstra o primeiro passo de aplicar os filtros *DCBlock* para remover quaisquer oscilação de fase indesejada

Figura 28 - Processamento de *Drive*: etapa 1

```
*left = self.pre_filter[0].tick(*left);
*right = self.pre_filter[1].tick(*right);
```

Fonte: elaborado pelo autor

Após isso, como ilustrado na O valor *a* representa a função de corte e o valor *k* representa o ganho a ser aplicado ao sinal. Essa parte da implementação causa o som característico associado à guitarras.

Figura 29, é calculado e aplicado o ganho, controlado pelo parâmetro *boost*, assim como é calculado a função de *clipping* baseado no valor do parâmetro *drive*. O valor *a* representa a função de corte e o valor *k* representa o ganho a ser aplicado ao sinal. Essa parte da implementação causa o som característico associado à guitarras.

Figura 29 - Processamento de Drive: etapa 2

```
let gain = ((self.boost / 100.0) * 100.0) + 1.0;

*left *= gain;
*right *= gain;

let a = (((self.drive + 1.0) / 101.0) * (PI / 2.0)).sin();
let k = 2.0 * a / (1.0 - a);

let drive_l = (1.0 + k) * *left / (1.0 + k * left.abs());
let drive_r = (1.0 + k) * *right / (1.0 + k * right.abs());

*left = drive_l;
*right = drive_r;
```

Fonte: elaborado pelo autor

Por fim é aplicado os filtros *Butterworth Lowpass* ao sinal distorcido, como ilustra a Figura 30. Esses filtros têm como função cortar as frequências agudas, dando liberdade ao usuário de deixar o som de sua guitarra mais “quente” ou “frio”.

Figura 30 - Processamento de *Drive*: etapa 3

```
*left = self.filter[0].tick(*left);
*right = self.filter[1].tick(*right);
```

Fonte: elaborado pelo autor

O efeito *Fuzz* é implementado de forma idêntica ao *Drive*, porem com a adição de distorção assimétrica. Assim como o *Drive*, o *Fuzz* utiliza dois filtros *DCBlock*, a mesma função de *clipping* e dois filtros *Butterworth Lowpass* para o controle de tonalidade, porém, antes de ser realizado o *clipping*, é adicionado um deslocamento ao sinal baseado no parâmetro *Fuzziness* e na amplitude atual do sinal. A Figura 31 ilustra a implementação do cálculo do deslocamento da distorção assimétrica. Sendo o restante da implementação do *Fuzz* é idêntica ao que é descrito na Figura 28, Figura 29 e Figura 30.

Figura 31 - Implementação de distorção assimétrica

```
let offset_l = left.abs() * (self.fuzziness / 100.0);
let offset_r = right.abs() * (self.fuzziness / 100.0);

*left += offset_l;
*right += offset_r;
```

Fonte: elaborado pelo autor

O efeito *Bit Crusher*, diferentemente do *Fuzz* e *Drive*, não utiliza uma função de clipping e não aplica nenhum ganho no sinal, a distorção trazida por esse efeito é oriunda da perca de qualidade proposital do áudio, resultando em um sinal de saída semelhante as músicas 8-bit presentes em muitos jogos antigos. Internamente o efeito depende do *sample rate* atual e do *bit rate*, a qual é a nova taxa de amostragem do sinal. Primeiramente é calculado o tamanho do passo, sendo ela uma medida que determina a quantidade de amostras consecutivas que serão afetadas pelo processo de *bit crushing*. Baseado no tamanho do passo, o programa começa a salvar amostras repetidas no sinal de saída, efetivamente diminuindo o *sample rate* do sinal.

Figura 32 - Implementação do *Bit Crusher*

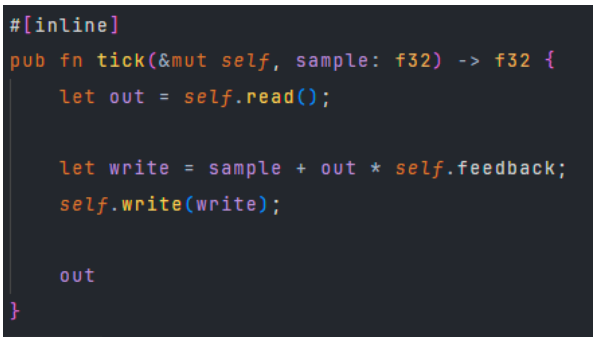
```
let step_size = self.step_size();

let mut sample_index = 0;
let buffer_size = input_buffer.len();

while sample_index < buffer_size {
    let first_index = sample_index;
    let limit_index = (sample_index + step_size).min(buffer_size);

    while sample_index < limit_index {
        input_buffer.process_channel(|channel| {
            let value = channel[first_index];
            channel[sample_index] = value;
        });
        sample_index += 1;
    }
}
```

Fonte: elaborado pelo autor

6.1.3 DELAYS

Este capitulo irá descrever como os dois efeitos de eco presentes, o *Delay* e *Twin Delay*, funcionam internamente, ambos os ecos do sistema implementam o mesmo algoritmo, logo somente será abordado o código em comum entre os dois. A Figura 33 ilustra a função *tick* do algoritmo de *Delay*, essa função atualiza o estado interno do efeito e também retorna um novo valor a ser gravado no sinal de saída.

Figura 33 - Função *tick* do efeito *Delay*



```
#[inline]
pub fn tick(&mut self, sample: f32) -> f32 {
    let out = self.read();

    let write = sample + out * self.feedback;
    self.write(write);

    out
}
```

Fonte: elaborado pelo autor

O efeito possui dois cursores, um de leitura, outro de escrita, sendo o de escrita deslocado n amostras a frente do de leitura. O número de amostras entre o cursores é determinado pelo intervalo de tempo entre cada eco em conjunto com o *sample rate* atual, sendo que um período de n segundos resultara em $n * S_r$ amostras, onde s é a duração de tempo em segundos e S_r é o *sample rate.* Conforme a função *read* é chamada, é calculado um valor interpolado[26] entre a amostra presente na posição atual e a amostra seguinte. Após calcular o valor interpolado, o cursor de leitura avança uma vez. A implementação desse comportamento é ilustrado na Figura 34.

Figura 34 - Implementação do *Delay: read*

```rust
pub fn read(&mut self) -> f32 {
    let delay_samples = self.delay_samples();
    let offset = delay_samples - delay_samples.floor();
    let buffer_size = self.buffer_size;

    let mut current_read_position = self.current_read_position;
    let delay_output = interpolate(
        self.delay_buffer[current_read_position],
        self.delay_buffer[(current_read_position + 1) % buffer_size],
        offset,
    );

    current_read_position += 1;
    if current_read_position >= buffer_size {
        current_read_position = 0;
    }
    self.current_read_position = current_read_position;

    delay_output
}
```

Fonte: elaborado pelo autor

Após a leitura, esse novo valor é multiplicado pelo parâmetro de *feedback*, o qual controla o decaimento de volume do efeito, e é somado ao sinal de entrada. Após a soma é feita a escrita do valor previamente calculado e só então é retornado o novo valor. A Figura 35 ilustra a função de escrita, na qual o novo valor calculado é salvo em um *buffer* para uso futuro.

Figura 35 - Implementação do *Delay: escrita*

```rust
pub fn write(&mut self, sample: f32) {
    let mut current_write_position = self.current_write_position;
    self.delay_buffer[current_write_position] = sample;

    current_write_position += 1;
    if current_write_position >= self.buffer_size {
        current_write_position = 0;
    }
    self.current_write_position = current_write_position;
}
```

Fonte: elaborado pelo autor

[26] Interpolação é o método de aproximar os valores dos conjuntos discretos. Em matemática, denomina-se interpolação o método que permite construir um novo conjunto de dados a partir de um conjunto discreto de dados pontuais previamente conhecidos.

6.1.4 STUDIO REVERB

Neste capitulo será descrito como foi implementado o efeito *Studio Reverb*, sendo ele o efeito de reverberação provido pelo sistema. Este efeito implementa o reverberador de Schroeder descrito no capítulo 2.5. Internamente o reverberador é configurado por uma série de variáveis constantes demonstradas na Figura 36 , os quais determinam os diferentes atrasos do reverberador. A alteração desses valores resultará em timbres diferentes para o reverberador.

Figura 36 - Configuração do Studio Reverb

```
const SCALE_DAMPENING: f32 = 0.4;

const SCALE_ROOM: f32 = 0.28;
const OFFSET_ROOM: f32 = 0.7;

const STEREO_SPREAD: usize = 23;

const COMB_TUNING_L1: usize = 1116;
const COMB_TUNING_R1: usize = 1116 + STEREO_SPREAD;
const COMB_TUNING_L2: usize = 1188;
const COMB_TUNING_R2: usize = 1188 + STEREO_SPREAD;
const COMB_TUNING_L3: usize = 1277;
const COMB_TUNING_R3: usize = 1277 + STEREO_SPREAD;
const COMB_TUNING_L4: usize = 1356;
const COMB_TUNING_R4: usize = 1356 + STEREO_SPREAD;
const COMB_TUNING_L5: usize = 1422;
const COMB_TUNING_R5: usize = 1422 + STEREO_SPREAD;
const COMB_TUNING_L6: usize = 1491;
const COMB_TUNING_R6: usize = 1491 + STEREO_SPREAD;
const COMB_TUNING_L7: usize = 1557;
const COMB_TUNING_R7: usize = 1557 + STEREO_SPREAD;
const COMB_TUNING_L8: usize = 1617;
const COMB_TUNING_R8: usize = 1617 + STEREO_SPREAD;

const ALLPASS_TUNING_L1: usize = 556;
const ALLPASS_TUNING_R1: usize = 556 + STEREO_SPREAD;
const ALLPASS_TUNING_L2: usize = 441;
const ALLPASS_TUNING_R2: usize = 441 + STEREO_SPREAD;
const ALLPASS_TUNING_L3: usize = 341;
const ALLPASS_TUNING_R3: usize = 341 + STEREO_SPREAD;
const ALLPASS_TUNING_L4: usize = 225;
const ALLPASS_TUNING_R4: usize = 225 + STEREO_SPREAD;
```

Fonte: elaborado pelo autor

Definido a configuração do reverberador, é possível calcular a reverberação da sala definida pelos valores constantes. Esse processo se dá por meio da função *tick*, a qual retorna um valor novo, com a reverberação, e atualiza o estado interno do efeito, esse comportamento está ilustrado na Figura 37.

Figura 37 - Função *tick* do *Studio Reverb*

```rust
pub fn tick(&mut self, input: (f32, f32)) -> (f32, f32) {
    let input_mixed = (input.0 + input.1) * FIXED_GAIN * self.input_gain;

    let mut out = (0.0, 0.0);

    for combs in self.combs.iter_mut() {
        out.0 += combs.0.tick(input_mixed);
        out.1 += combs.1.tick(input_mixed);
    }

    for allpasses in self.allpasses.iter_mut() {
        out.0 = allpasses.0.tick(out.0);
        out.1 = allpasses.1.tick(out.1);
    }

    (
        out.0 * self.wet_gains.0 + out.1 * self.wet_gains.1 + input.0 * self.dry,
        out.1 * self.wet_gains.0 + out.0 * self.wet_gains.1 + input.1 * self.dry,
    )
}
```

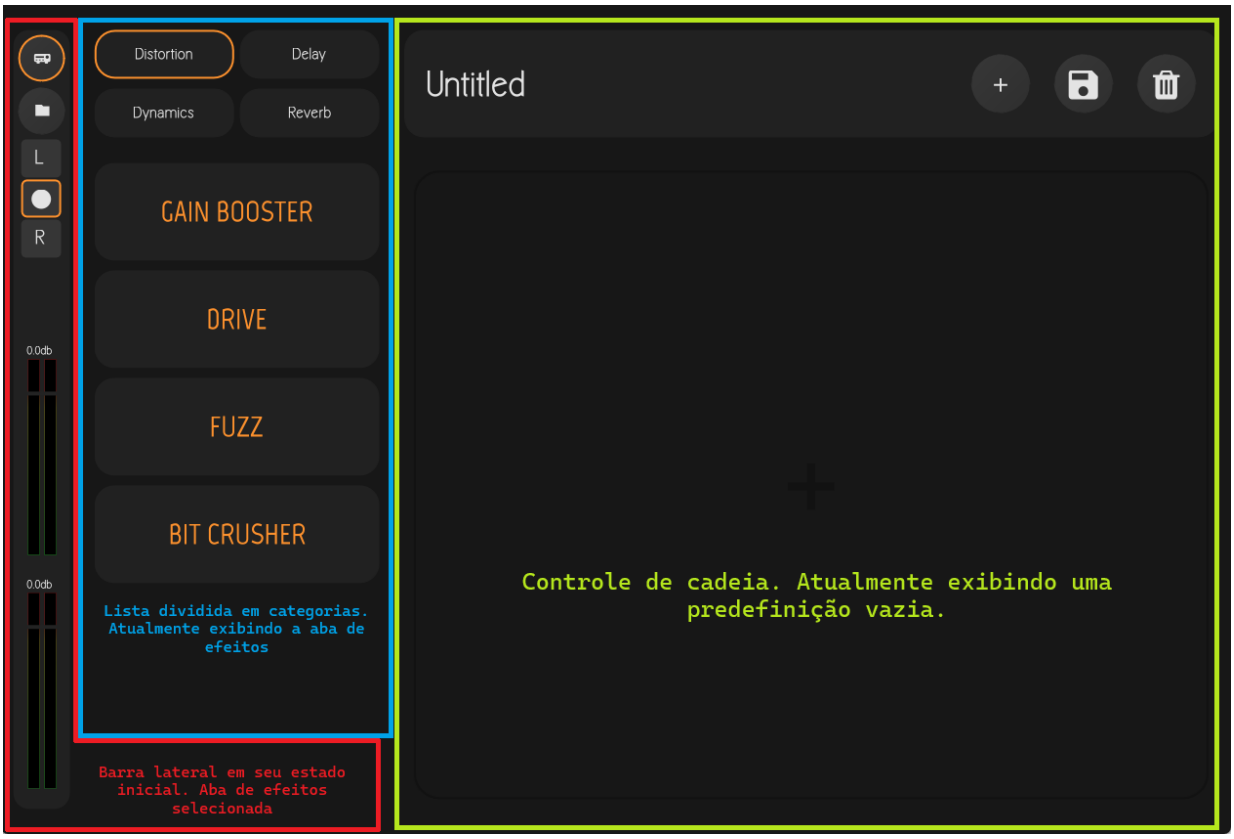
Fonte: elaborado pelo autor

Em primeira instancia, é feito a mistura dos canais esquerdo e direito em conjunto com a aplicação de um ganho no sinal. Feito isso, os *comb filters* são processados, adicionando os atrasos que criam a reverberação, cada um dos *combs* aplicados são configurados com um dos valores demonstrados na Figura 36. Após calculados os atrasos, é necessário corrigir a fase dos atrasos para evitar com que os eles e o sinal de entrada se cancelem, iniciando o processamento dos diferentes filtros *allpass* presente no efeito. No final de todo esse processo, é calculado o quanto do sinal reverberado será retornado em relação ao sinal de entrada.

6.2 VISÂO GERAL

O trabalho aqui implementado usufrui primariamente de botões deslizantes para o controle de parâmetros, assim como utiliza um sistema arrasta-solta para carregar os efeitos. O programa pode ser compreendido em três principais elementos visuais: a barra lateral; uma lista dividida em categorias; e o controle de cadeia. A Figura 38 apresenta os elementos visuais separados por cores, sendo a parte vermelha a barra lateral, a parte azul a lista dividida em categorias e a parte verde o controle de cadeia. Cada elemento será explicado em detalhes nos próximos capítulos.

Figura 38 - Visão geral do sistema



Fonte: elaborado pelo autor

6.3 BARRA LATERAL

A barra lateral está encarregada de oferecer controle sobre qual aba está sendo exibida pelo sistema, um seletor de canal de entrada e controle sobre o volume de entrada e saída. A Figura 39 apresenta a barra lateral com seus elementos divididos por cores

Figura 39 - Barra lateral do Fretcat

Fonte: elaborado pelo autor

6.3.1 CONTROLE DE ABAS

O controle de abas é composto por dois botões, sendo cada um deles representativos de uma aba do sistema. A Figura 40 representa o controle de abas com a aba de efeitos selecionada. O botão superior é a aba de efeitos, e o inferior é a aba de predefinições.

Figura 40 - Controle de abas Fretcat

Fonte: elaborado pelo autor

6.3.2 CONTROLE DE CANAL DE ENTRADA

Devido a guitarra ser um instrumento mono, existe a possibilidade do sinal recebido ser reproduzido por somente um dos dois canais estéreos. Visando solucionar esse problema, foi feito um controle de canal de entrada, demonstrado na Figura 41, o qual pode estar em três estados distintos, os quais são: L, curto para *left*, replica o som do canal esquerdo para o direito; o círculo representa o estado padrão, ou seja, o sinal será processado da maneira que for recebido, sem replicação de canal; R, curto para *right*, replica o som do canal direito para o esquerdo.

Figura 41 - Controle de canal de entrada

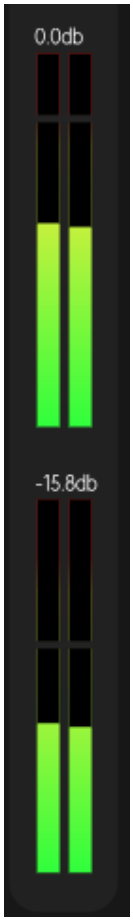
Fonte: elaborado pelo autor

6.3.3 CONTROLES DE VOLUME

O sistema oferece dois controles de volume, um para o sinal de entrada, prévio a qualquer processamento de efeitos, e um para o sinal de saída, posterior ao processamento de efeitos. O aumento/diminuição de volume são aplicados após a replicação de canais descritas no capitulo 6.3.2. Cada controle de volume é composto por medidor de pico, a qual representa o quão alto volume está atualmente, e uma escala linear que controla o ganho ou redução de volume em decibéis. No topo do medidor existe um contador, em decibéis, o qual representa quantos decibéis de ganho ou redução estão sendo aplicados.

A Figura 42 demonstra os controles de volume do sistema em funcionamento, sendo o superior representativo do sinal de entrada e o inferior representativo do sinal de saída. O controlador de entrada não está alterando volume, já o controlador de saída está reduzindo o volume.

Figura 42 - Controladores de volume



Fonte: elaborado pelo autor

O próximo capitulo irá abordar as duas listas contidas nas abas previamente mencionadas, bem como seus conteúdos e funcionalidades.

6.4 LISTAS

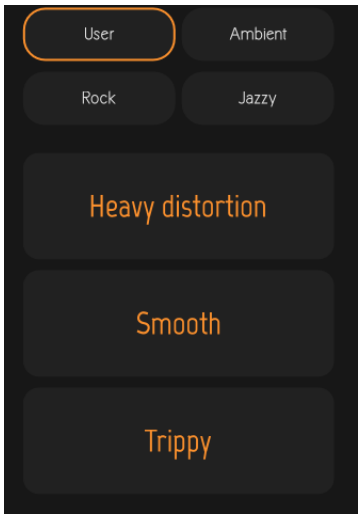
Este capitulo irá descrever o que cada lista do sistema representa, assim como também irá falar sobre o conteúdo das mesmas. Será explicado cada categoria da aba de efeitos, assim como será descrito cada efeito presente na mesma.

6.4.1 LISTA DE PREDEFINIÇÔES

A lista de predefinições está encarregada de exibir as diversas predefinições salvas do programa. Ela possui quatro categorias, sendo elas: a categoria *User*, que contém as configurações feitas pelo usuário; a categoria *Ambient*, a qual possui timbres de som ambiente, utilizando primariamente dos efeitos de eco e reverberação; a categoria *Rock*, a qual possui timbres comumente associados a guitarras, usufruindo dos efeitos de distorção e dinâmica; e a categoria *Jazzy*, a qual possui timbres mais limpos, utilizando de todos os efeitos presentes de maneira moderada.

A Figura 43 demonstra a aba de predefinições na aba *User*. Como pode se notar a lista apresenta um *card*[27] por predefinição, sendo identificada pelo seu nome, neste caso definido pelo usuário. As demais categorias seguem a mesma lógica visual.

Figura 43 - Lista de predefinições

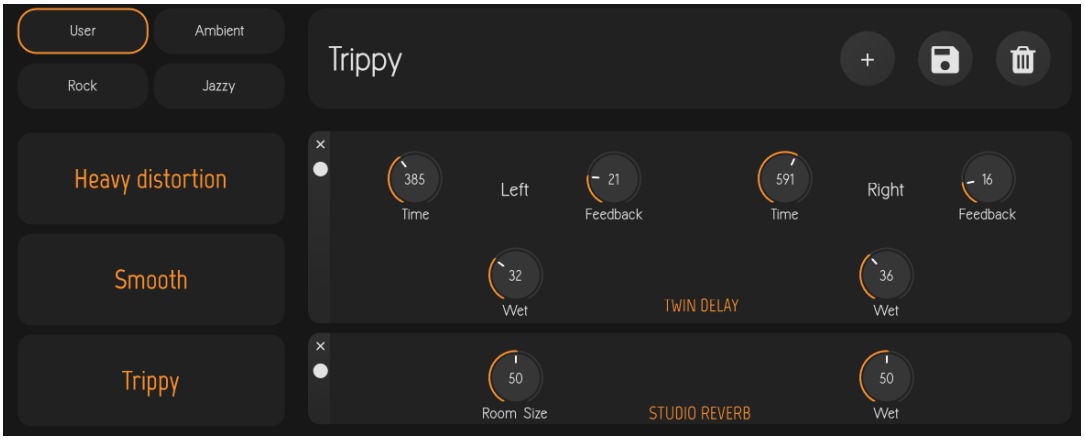


Fonte: elaborado pelo autor

[27] Um *card* refere-se a um componente de design que exibe informações ou conteúdo de forma modular e organizada. Os *cards* são frequentemente utilizados para apresentar pedaços de informação concisos e autônomos, tornando mais fácil para os usuários consumirem o conteúdo de maneira rápida e eficiente.

Ao clicar em uma predefinição ela será carregada na lista de efeitos. A Figura 44 demonstra a predefinição *Trippy* carregada com sucesso na cadeia de efeitos.

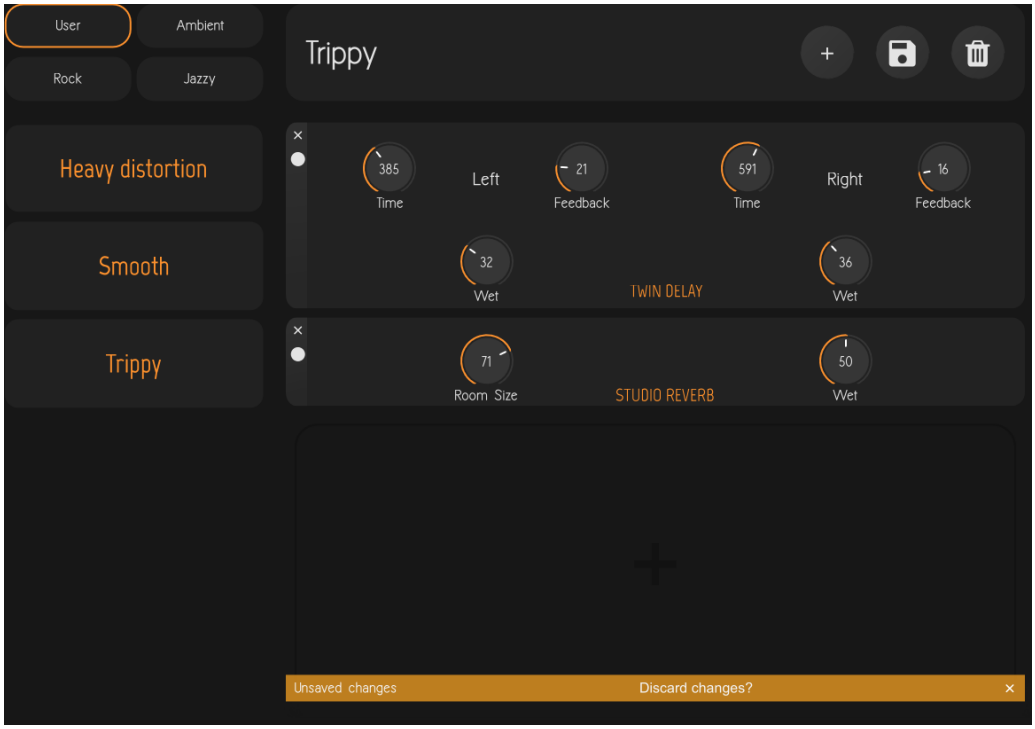
Figura 44 - Predefinição *Trippy* carregada



Fonte: elaborado pelo autor

Caso o usuário tenha alterações não salvas o sistema irá informar o usuário e esperar por uma ação do mesmo, que pode ser de descartar as alterações, ou salva-las. Isso é demonstrado na Figura 45, na qual é feito a tentativa de carregar a predefinição *Heavy Distortion*, mas a predefinição *Trippy* já está carregada com alterações no parâmetro *room size*, do efeito *studio reverb*.

Figura 45 - Aviso ao tentar carregar predefinição

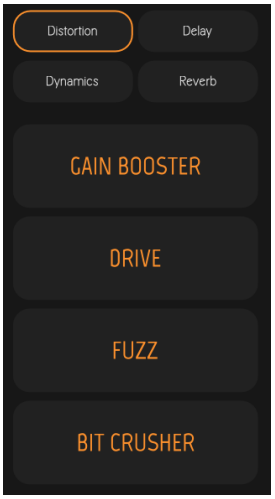


Fonte: elaborado pelo autor

6.4.2 LISTA DE EFEITOS

Esta parte da interface do sistema se encarrega de exibir os efeitos disponíveis, agrupados por suas categorias. Cada efeito é representado por um *card*, assim como as predefinições são, mantendo a interface gráfica coesa. A Figura 46 demonstra a lista de efeitos na aba de distorção.

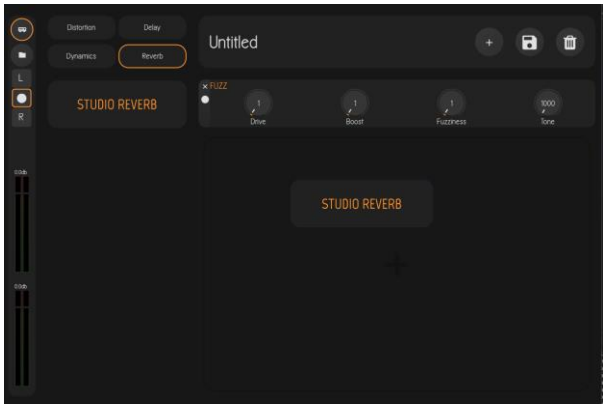
Figura 46 - Lista de efeitos



Fonte: elaborado pelo autor

A diferença presente entre essa lista para a lista de predefinições se dá no aspecto de que efeitos são adicionados através de um sistema arrasta-solta, no qual o usuário pressiona o *mouse* sob o efeito que deseja e o solta na posição que deseja no controle de cadeia. Essa funcionalidade é demonstrado na Figura 47, na qual está sendo adicionado um *Studio Reverb* abaixo de um *Fuzz*.

Figura 47 - Efeito sendo adicionado ao controle de cadeia



Fonte: elaborado pelo autor

O próximo capítulo irá descrever o funcionamento da interface dos efeitos presentes na lista de efeitos, bem como os algoritmos implementados por cada um deles.

6.5 EFEITOS

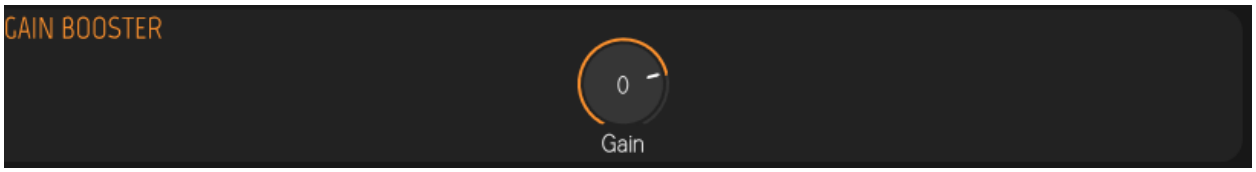
Este capitulo ira abordar como os efeitos presentes no sistema são apresentados, assim como irá relacionar os mesmos aos algoritmos descritos no capitulo 6.1. Os efeitos do sistema combinam diferentes algoritmos com o instituto de gerarem timbres únicos e ao mesmo tempo manter o código limpo através da reutilização.

6.5.1 EFEITOS DE *DISTORÇÂO*

Essa categoria está encarregada de agrupar os efeitos que, de uma forma ou outra, distorcem propositalmente o sinal de entrada. Aqui será descrito o uso e funcionamento da interface dos efeitos *Gain Booster, Drive, Fuzz* e *Bit Crusher.*

A Figura 48 demonstra a interface do efeito mais simples do sistema, o *Gain Booster*. Este efeito apenas aumenta ou diminui o volume do sinal, se assemelhando aos controles de volume descritos no capitulo 6.3.3. O parâmetro *Gain* representa a quantidade de decibéis que serão acrescentados ao sinal de entrada.

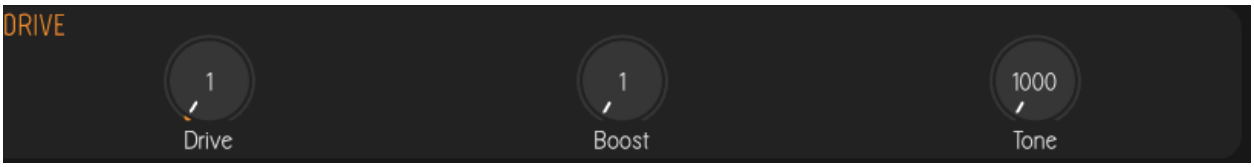
Figura 48 - Interface do efeito *Gain Booster*



Fonte: elaborado pelo autor

A Figura 49 demonstra a interface do efeito *Drive*, o qual permite o usuário controlar os parâmetros *Drive, Boost* e *Tone.* Internamente o efeito *Drive* usufrui do algoritmo de distorção descrito no capítulo 6.1.2. O efeito *Drive* tem a intenção de imitar um pedal de *Overdrive*, produzindo o som distorcido característico de guitarras.

Figura 49 - Interface do efeito *Drive*



Fonte: elaborado pelo autor

O parâmetro *Drive* e *Boost* em conjuntos controlam o nível de distorção do efeito, sendo o parâmetro *Drive* o quão agressiva a função de recorte será e o parâmetro *Boost* é um ganho de volume prévio à função de recorte. O parâmetro *Tone* controla a frequência de corte de um filtro IIR *Butterworth Lowpass.*

A Figura 50 demonstra a interface do efeito *Fuzz*, o qual é similar ao *Drive*, porem acrescenta o parâmetro *Fuzziness*. Diferentemente do *Drive*, o *Fuzz* utilizada distorção assimétrica, cuja implementação é descrita no capitulo 6.1.2.

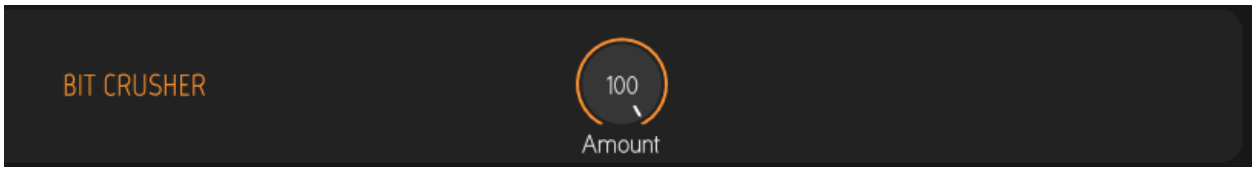
Figura 50 - Interface do efeito *Fuzz*

Fonte: elaborado pelo autor

O parâmetro *Drive*, *Boost* e *Tone* são iguais aos parâmetros do efeito *Drive*, porém, com a adição do parâmetro *Fuzziness*, é possível controlar o quão assimétrica será a distorção, emulando um amplificador com rasgos em seu cone[28].

A Figura 51 demonstra a interface do efeito *Bit Crusher*, o qual é um efeito fundamentado num conceito totalmente digital. O *Bit Crusher* internamente faz *downsampling*[29] do sinal de entrada baseado no parâmetro *Amount*.

Figura 51 - Interface do efeito *Bit Crusher*



Fonte: elaborado pelo autor

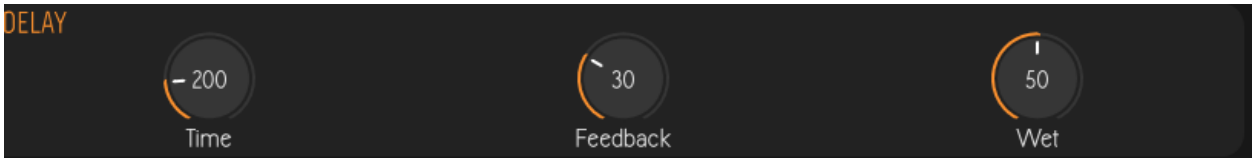
O efeito possui um único parâmetro, denominado *Amount*, que corresponde a porcentagem do *sample rate* do sinal de saída em relação ao sample rate do sinal de entrada, ou seja, quanto menor o parâmetro for, menor será a taxa de amostragem final e mais distorcido o som será.

[28] O cone de um amplificador ou caixa de som é parte que se encarrega de projetar e amplificar o som produzido pelo mesmo, seguindo os mesmos princípios físicos do cone de uma vitrola.
[29] A resolução ou a taxa de amostragem de um sinal ou conjunto de dados é reduzida. Na prática, isso significa diminuir o número de pontos de dados ou a complexidade do sinal, muitas vezes para simplificar a representação ou reduzir os requisitos computacionais.

6.5.2 EFEITOS DE *DELAY*

Essa categoria está encarregada de agrupar os efeitos que repetem um sinal baseado em um determinado intervalo, criando um efeito de eco. Aqui será descrito o uso e funcionamento da interface dos efeitos *Delay* e *Twin Delay*. A Figura 52 representa a interface do efeito *Delay*. Este efeito aplica um eco igual para os dois canais estéreo.

Figura 52 - Interface gráfica do efeito *Delay*

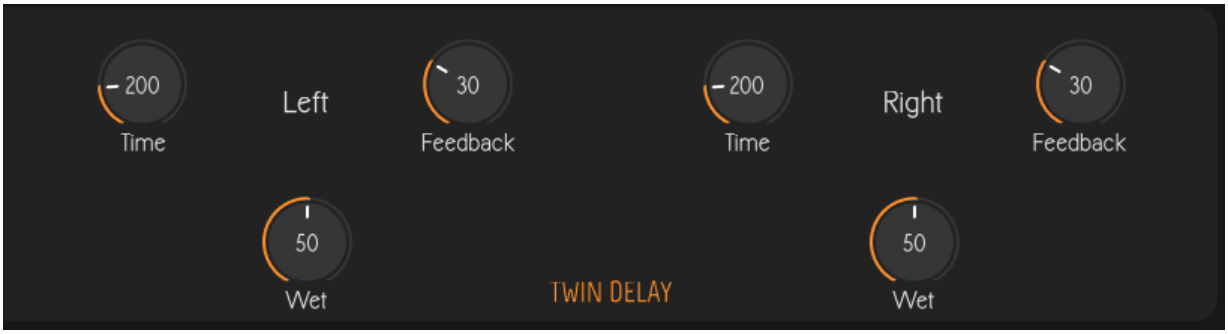


Fonte: elaborado pelo autor

Os parâmetros do efeito *Delay* são: *Time*, o qual determina o intervalo de tempo entre cada eco; *Feedback* que determina o decaimento do volume entre cada eco, ou seja, caso o *Feedback* esteja no máximo, o sinal ficará ecoando infinitamente, pois não haverá perda de volume; *Wet*, determina o quanto do sinal sem eco será usado em relação ao sinal com eco.

A Figura 53 representa a interface gráfica do *Twin Delay*, o qual tem funcionamento interno idêntico ao *Delay*, mas aplica ecos diferentes para cada canal estéreo.

Figura 53 - Interface gráfica do *Twin Delay*



Fonte: elaborado pelo autor

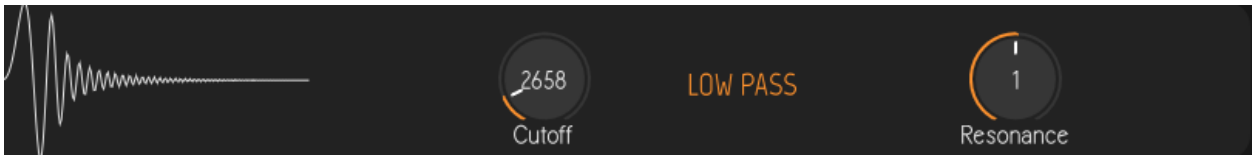
Os parâmetros do *Twin Delay* são os mesmos do *Delay*, porem duplicados, tendo um parâmetro para cada canal estéreo, ou seja, o *Time* da esquerda controla o intervalo de tempo entre ecos do canal esquerdo e o *Time* da direita controla o intervalo de tempo entre ecos do canal direito, o mesmo se aplica aos outros parâmetros.

6.5.3 EFEITOS DE *DYNAMICS*

Essa categoria está encarregada de agrupar os efeitos que alteram alguma qualidade do sinal de entrada, sem imitar um fenômeno físico como os demais efeitos mencionados. Aqui será descrito o funcionamento dos três tipos de filtros implementados, o *Low Pass*, o *Band Pass* e o *High Pass*.

A Figura 54 representa a interface do efeito *Low Pass*, o qual corta frequências agudas e mantem frequências graves. Esse efeito é utilizado para deixar o som mais "quente" ou "aconchegante".

Figura 54- Interface gráfica do efeito *Low Pass*

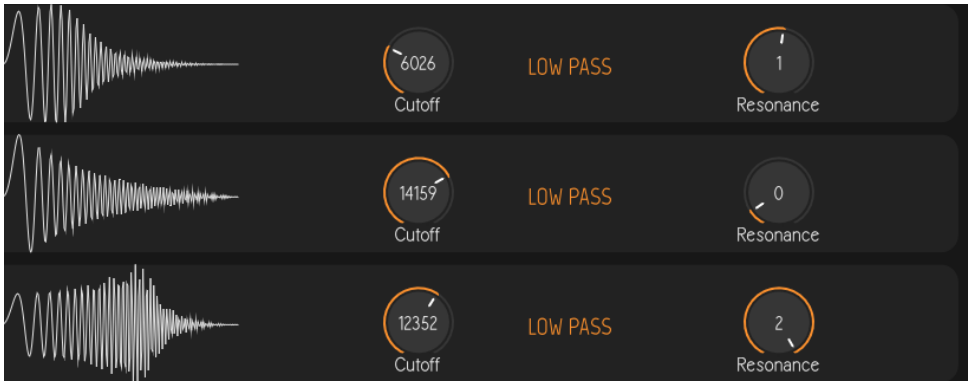


Fonte: elaborado pelo autor

O efeito possui dois parâmetros e um elemento gráfico para auxiliar a visualização e compreensão da frequência de corte atual. O parâmetro *Cutoff* determina a frequência de corte em Hertz de um SVF *Lowpass* e o parâmetro *Resonance* determina a ressonância do filtro, determinando quão suave a transição entre a linha de corte e as frequências acima da linha de corte será.

O elemento gráfico é computado em tempo real conforme alterações são feitas aos parâmetros descritos previamente e tem como finalidade demonstrar ao usuário como o filtro está afetando determinadas frequências. O lado esquerdo é o mais grave (20hz) e o lado direito a frequência Nyquist, ou seja, metade do *sample rate*, a frequência mais aguda que o dispositivo atual consegue capturar. A Figura 55 demonstra o como o elemento gráfico reage a diferentes parâmetros.

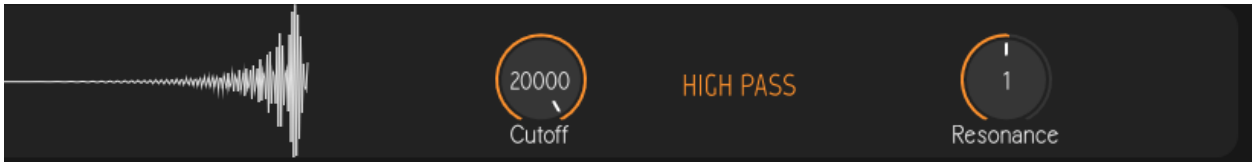
Figura 55 - Representação gráfica do *Low Pass* com parâmetros variados



Fonte: elaborado pelo autor

A Figura 56 demonstra um filtro *High Pass*, o qual é o oposto do filtro *Low Pass*, removendo frequências graves e mantendo as agudas. Esse efeito é utilizado para deixar o som mais “afiado” ou “brilhante”.

Figura 56 - Interface gráfica do efeito *High Pass*

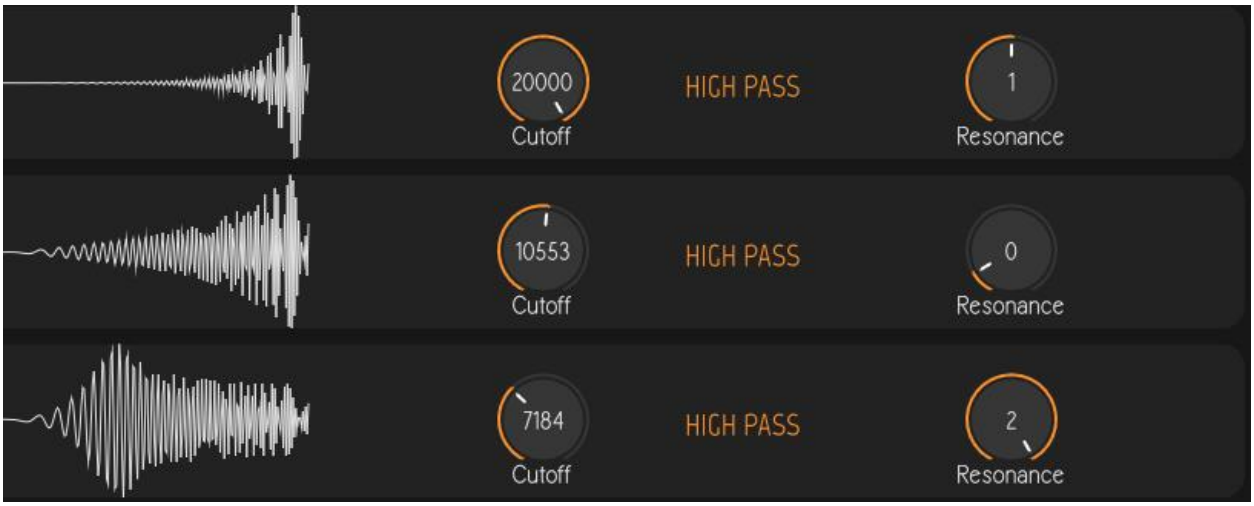


Fonte: elaborado pelo autor

Assim como o *Low Pass*, o *High Pass* também possui dois parâmetros, um para a frequência de corte que determina a frequência de corte de um SVF *Highpass* e outro para a ressonância do mesmo.

O *High Pass* também apresenta um elemento gráfica para facilitar a compreensão de como o filtro está agindo. A maneira como o gráfico é exibido se dá igualmente ao do *Low Pass*, porem agora cortando as frequências graves. A Figura 57 representa como o *High Pass* responde a diferentes parâmetros.

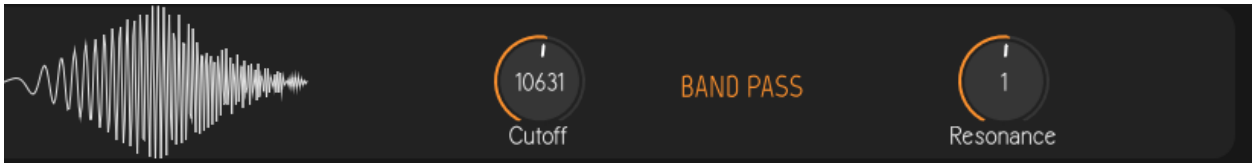
Figura 57 - Representação gráfica do *High Pass* com parâmetros variados



Fonte: elaborado pelo autor

A Figura 58 demonstra um filtro *Band Pass*, o qual, se diferenciando dos outros dois filtros, age sobre uma banda de frequência especifica, ou seja, qualquer frequência acima ou abaixo da frequência de corte será reduzida. É um efeito com casos de uso específicos, sendo mais usado em conjunto com a automação do parâmetro da frequência de corte, produzindo um efeito chamado *Auto-Wah*

Figura 58 - Interface gráfica do efeito *Band Pass*

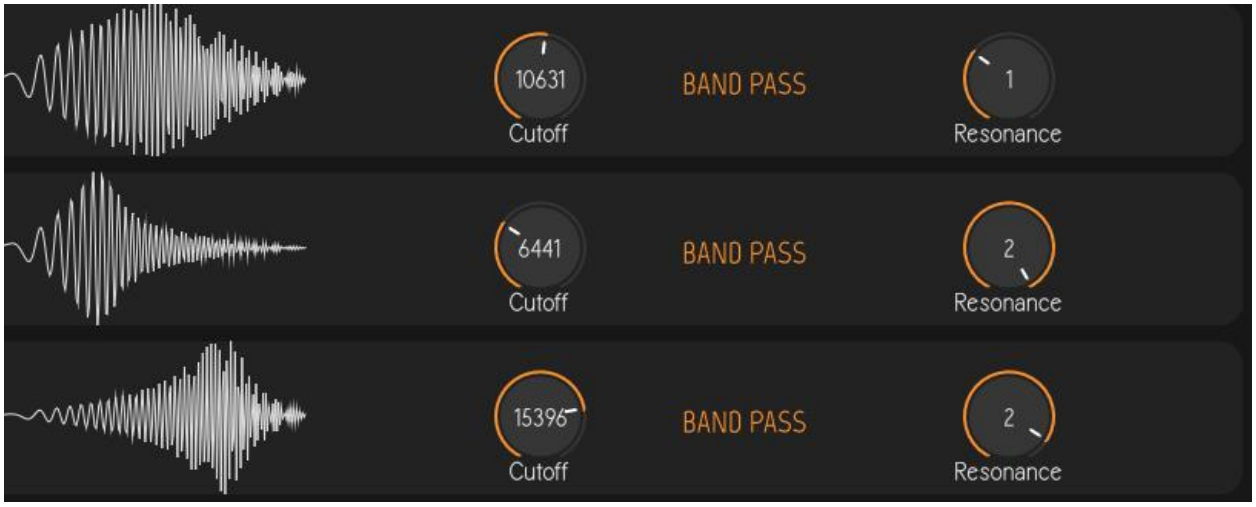


Fonte: elaborado pelo autor

Assim como os demais, o *Band Pass* possui dois parâmetros, um que determina a frequência de corte de um SVF *Bandpass* e outro que determina a ressonância do mesmo.

O *Band Pass* também apresenta um elemento gráfica para facilitar a compreensão de como o filtro está agindo. A maneira como o gráfico é exibido é totalmente diferente dos outros dois filtros, sendo possível observar na Figura 59 que as frequências acima e abaixo da frequência de corte são diminuídas.

Figura 59 - Representação gráfica de um *Band Pass* com parâmetros variados



Fonte: elaborado pelo autor

6.5.4 EFEITOS DE *REVERB*

.

Essa categoria está encarregada de agrupar os efeitos que imitam a reverberação de um ambiente. Aqui será descrito o funcionamento do efeito *Studio Reverb* que utiliza internamente do algoritmo *Freeverb*.

A Figura 60 ilustra a interface gráfica do efeito *Studio Reverb*, o qual tem intuito de replicar a reverberação que ocorre dentro de um estúdio.

Figura 60 - Interface gráfica do efeito *Studio Reverb*

Fonte: elaborado pelo autor

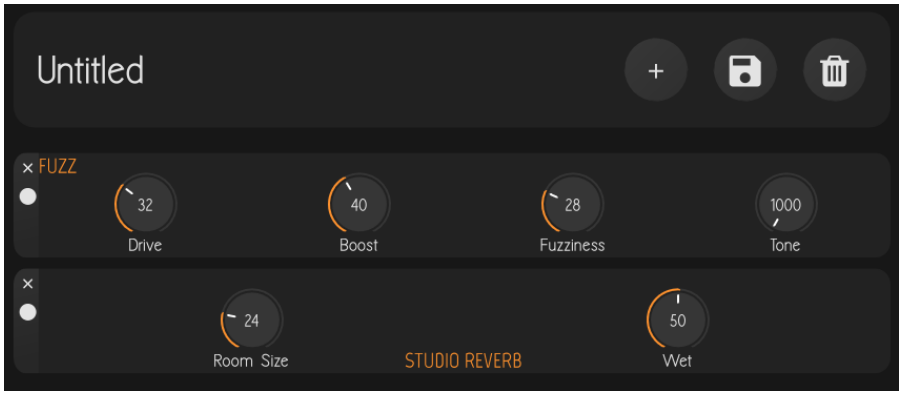
O parâmetro *Room Size* controla o tamanho da sala sendo simulada pelo algoritmo *Freeverb*. O parâmetro *Wet* determina o quanto do sinal sem reverberação será usado em relação ao sinal com reverberação.

No próximo capitulo será abordado como esses efeitos são encadeados e apresentados para o usuário durante o uso do programa, bem como será descrito as possíveis ações do usuário com a ordenação dos efeitos.

6.6 CONTROLE DE CADEIA

O controle de cadeia é a parte da interface gráfica na qual o usuário pode adicionar, remover, mover, ativar e desativar efeitos. A cadeia é composta pelas interfaces gráficas de cada efeito com a adição de uma barra para manipulação previamente mencionada dos efeitos. A Figura 61 ilustra o controle de cadeia com o efeito *Fuzz* e *Studio Reverb* adicionados.

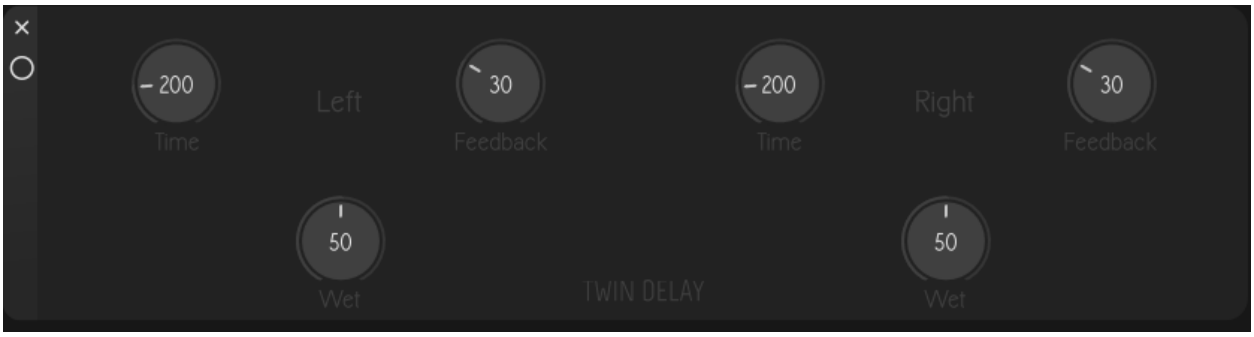
Figura 61 - Controle de cadeia com efeitos carregados



Fonte: elaborado pelo autor

A barra presente ao lado da interface de cada efeito serve três propósitos, eles são: deletar o efeito, ativar/desativar o efeito; e a barra em si é utilizado para mover o efeito. Ao pressionar o botão que desativa o botão, o efeito aparecerá apagado e o botão trocará de símbolo, agora mostrando um círculo vazio. Esse comportamento de desativar um efeito pode ser observado na Figura 62.

Figura 62 - *Twin Delay* desativado



Fonte: elaborado pelo autor

A Figura 63 ilustra a parte superior do controle de cadeia possui o nome da predefinição atual e os botões que executam as funções de criar, salvar e deletar predefinições.

Figura 63 - Controles de predefinição

Fonte: elaborado pelo autor

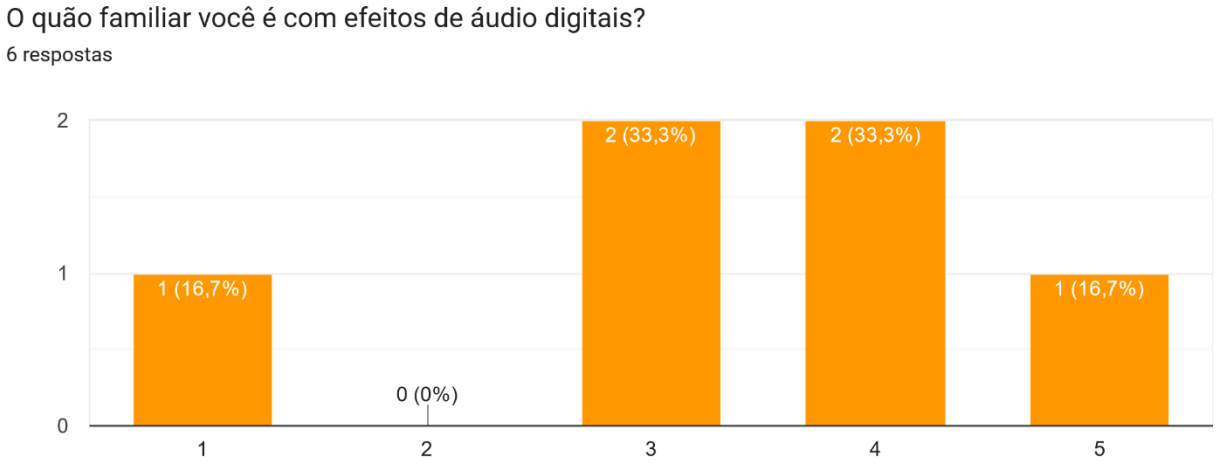
7. TESTES E DISCUSSÕES

Esta capitulo possui como objetivo discutir a avaliação do sistema por parte de seus usuários, além de verificar se o mesmo atingiu as metas estipuladas.

Para o teste do sistema Fretcat, foram selecionados participantes que se enquadravam no público alvo, isto é, músicos. Os testes foram feitos por seis usuários, sendo quatro deles integrantes da banda do Instituto Federal do Rio Grande do Sul – Campus Canoas, os testes foram realizados presencialmente com o equipamento fornecido pelo autor. Após realizarem os testes, os participantes preencheram um questionário com nove perguntas, as quais visavam saber se o Fretcat cumpriu seus objetivos. O questionário dado aos participantes pode ser consultado no APÊNDICE B – QUESTIONÀRIO DE AVALIAÇÂO DO SISTEMA FRETCAT.

A primeira pergunta do questionário, encontrada na Figura 64, demonstra que a 83,3% dos participantes possuem conhecimento mediano ou superior sobre efeitos digitais, comprovando a grande espaço que possuem no mundo musical. As respostas variam de um a cinco, sendo um total desconhecimento e cinco total familiaridade.

Figura 64 - Questionário de avaliação: primeira pergunta



Fonte: elaborado pelo autor

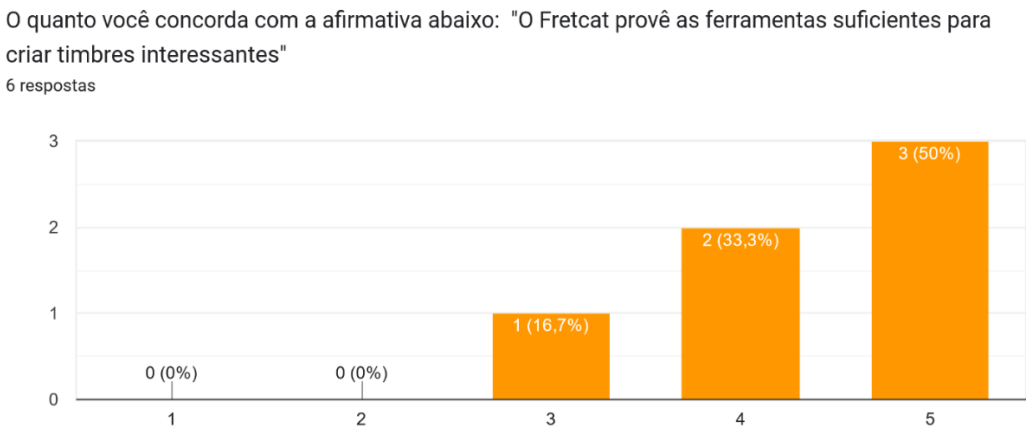
A segunda pergunta, ilustrada na Figura 65, questiona os participantes no que tange o interesse em usar efeitos digitais como meio de produção musical, tendo todos participantes interessados no uso de efeitos digitais. A partir disso, pode se concluir que os efeitos digitais são de grande importância para a produção musical como um todo.

Figura 65 - Questionário de avaliação: segunda pergunta

Fonte: elaborado pelo o autor

No que tange aos efeitos disponíveis no programa, a grande maioria dos participantes acharam os efeitos suficientes, isso é ilustrado na Figura 66, tendo 50% dos participantes totalmente satisfeitos. As respostas variam entre um e cinco, sendo um representativo de total insatisfação e cinco total satisfação.

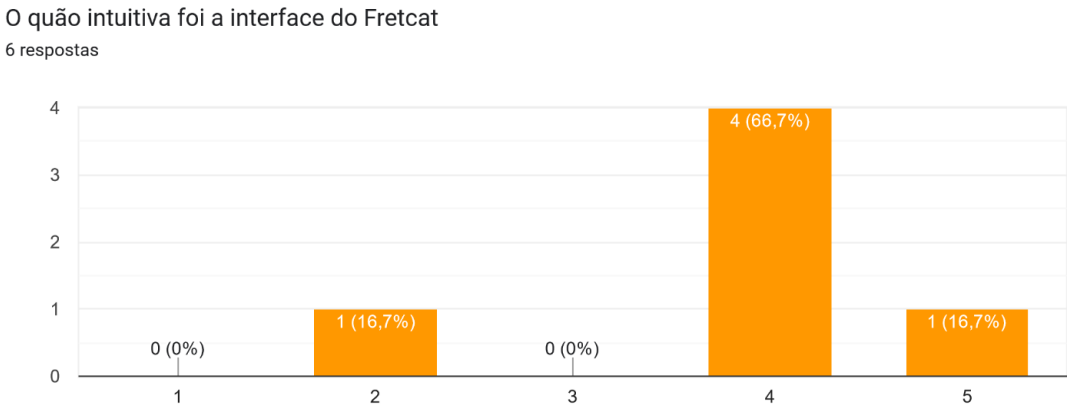
Figura 66 - Questionário de avaliação: terceira pergunta



Fonte: elaborado pelo autor

Com o intuito de verificar a experiência de usuário, foi questionado a quão intuitiva foi a interface gráfica do sistema. A Figura 67 demonstra que 66,7% dos participantes acharam a interface boa e 16,7% não tiveram nenhum problema. Vale mencionar que o participante sem conhecimento prévio sobre efeitos de áudio, demonstrado na Figura 64, conseguiu utilizar o programa sem problemas. As respostas se dão de um a cinco, sendo um nada intuitivo e cinco totalmente intuitivo.

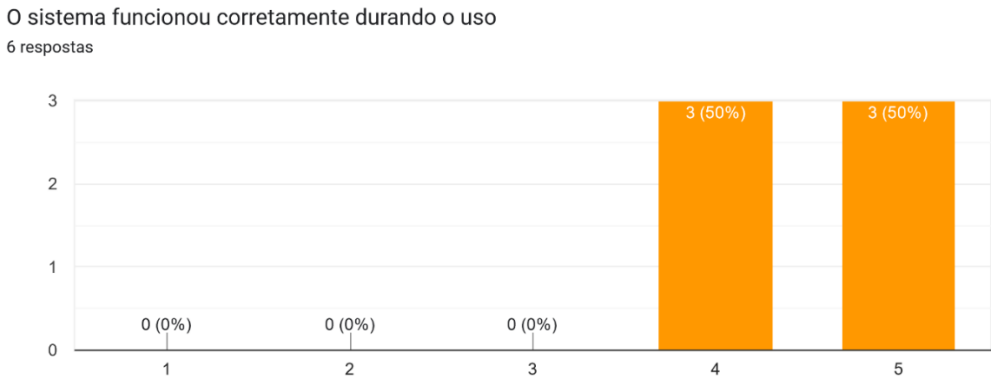
Figura 67 - Questionário de avaliação: quarta pergunta



Fonte: elaborado pelo autor

Em relação ao funcionamento do programa, foi disponibilizado a pergunta ilustrada na Figura 68. A partir dos dados obtidos pode se concluir que o programa funciona de forma adequada. As respostas se dão de um a cinco, sendo um representativo de uma falha crítica e cinco representativo de funcionamento totalmente correto.

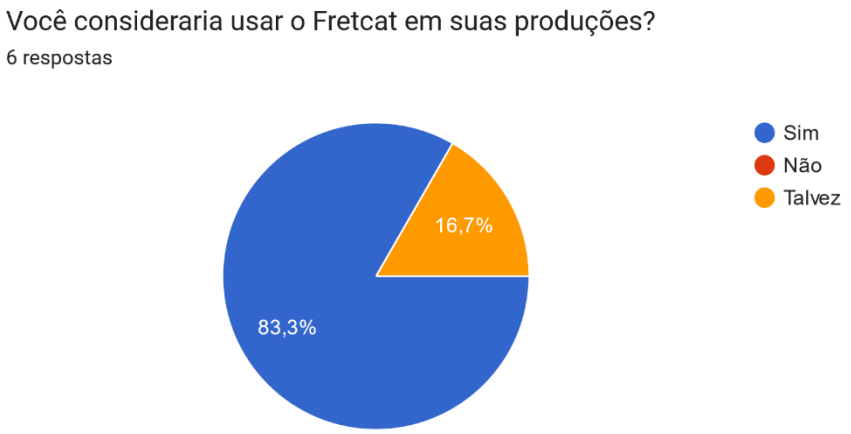
Figura 68 - Questionário de avaliação: quinta pergunta



Fonte: elaborado pelo autor

A seguinte pergunta, enunciada pela Figura 69, visava verificar o interesse dos participantes pelo programa. 83,3% dos participantes demonstraram interesse pelo sistema.

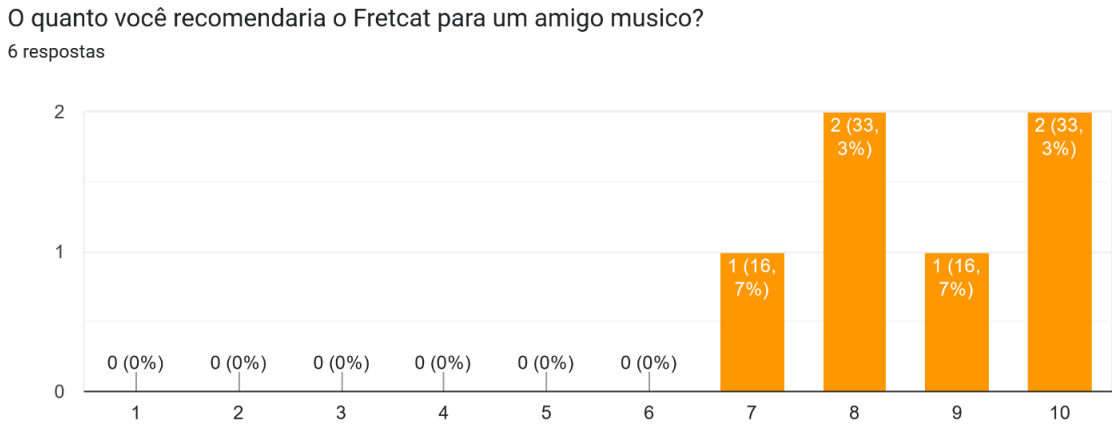
Figura 69 - Questionário de avaliação: sexta pergunta



Fonte: elaborado pelo autor

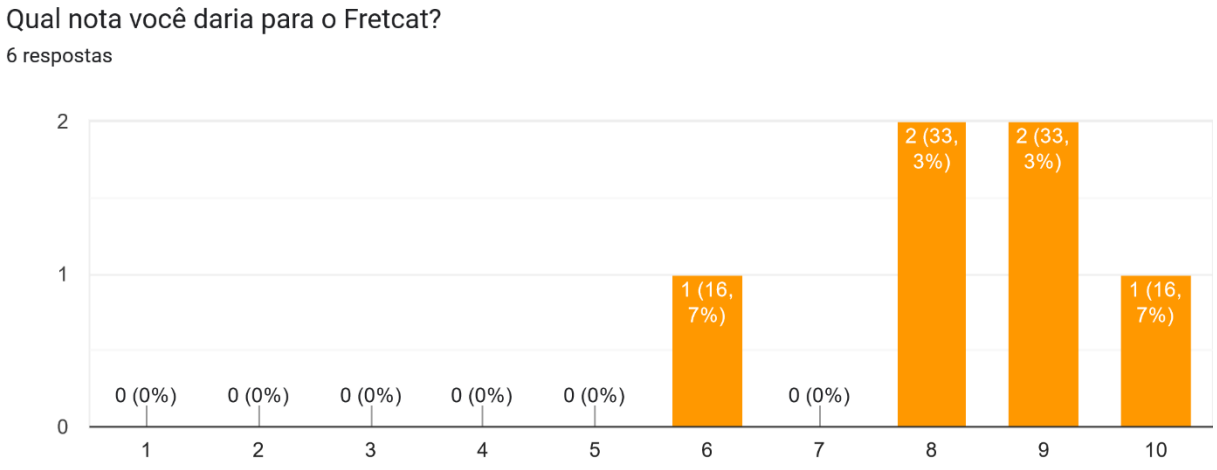
A fim de verificar o nível de aprovação do *software* foi feito a sétima pergunta, tendo suas respostas, ilustrada na Figura 70, e a oitava pergunta, enunciada na Figura 71. Com base nos dados obtidos, pode se afirmar que o sistema foi bem aceito pelos participantes, sendo a nota média 8,3. A sétima pergunta tem suas respostas variando entre não recomendaria e com certeza recomendaria.

Figura 70 - Questionário de avaliação: sétima pergunta



Fonte: elaborado pelo autor

Figura 71 - Questionário de avaliação: oitava pergunta



Fonte: elaborado pelo autor

8. CONCLUSÂO

O presente trabalho de conclusão descreveu o processo de construção do sistema Fretcat, abordando os aspectos computacionais, físicos, matemáticos e musicais requeridos para seu desenvolvimento.

O levantamento de requisitos e referencial teórico foi feito a partir da observação de trabalhos relacionados e da experiência própria do autor com produção musical. A partir dos dados obtidos, foram selecionadas as tecnologias que seriam usadas visando compreender as limitações e dificuldades logo no começo do desenvolvimento. Então, foi feito a modelagem do sistema buscando otimizar ao máximo a performance da *thread* de áudio e prover uma interface de usuário simples de se compreender. No fim da implementação foram feitos testes com quatro integrantes da banda do Instituto Federal Campus Canoas e outros dois participantes.

Analisando os dados obtidos pelos testes de aceitação, o Fretcat atingiu o objetivo geral de prover uma alternativa gratuita e intuitiva a programas mais caros, sendo seu desenvolvimento uma grande etapa para a aprendizagem do autor sobre conceitos de programação, como *multithreading* e DSP, assim como conceitos matemáticos e físicos relacionados à ondas e sinais.

8.1 TRABALHOS FUTUROS

Durante o processo de validação do trabalho pelos usuários foram sugeridas funcionalidades as quais o autor julgou como pertinentes ao aperfeiçoamento do trabalho.

Como trabalhos futuros, destacam-se: uma maneira de executar o programa fora de um *host*, gerenciando suas próprias entradas e saídas; adição de um guia introdutório para novos usuários; uma maneira de acessar predefinições guardadas em um banco de dados; e a adição de mais efeitos para o enriquecimento de combinações e timbres que o programa consegue gerar. Com isso, espera-se que usuários tenham um primeiro contato menos confuso e também tenham mais funcionalidades e aplicações para esse trabalho.

REFERENCIAS BIBLIOGRAFICAS

CARMO, Vera. O uso de questionários em trabalhos científicos. Departamento de Informática e Estatística da Universidade Federal de Santa Catarina, 19 ago. 2013. Disponível em: <http://www.inf.ufsc.br/~vera.carmo/Ensino_2013_2/O_uso_de_questionarios_em_trabalhos_cient%EDficos.pdf>. Acesso em: 15 nov. 2023.

TASCHETTO, Guilherme. Efeitos Digitais de Áudio utilizando DSP. [S.I], 2009.

GIL, Antônio Carlos. Como Elaborar Projetos de Pesquisa. 6. ed. São Paulo: Atlas, 2019.

HELM, Robbert van der. NIH Plug. [S.l.], 2023. Disponível em: <https://nih-plug.robbertvanderhelm.nl/nih_plug/>. Acesso em: 10 abr. 2023.

IBM. Diagramas de atividades. IBM Corporation, 2 mar. 2021. Disponível em: <https://www.ibm.com/docs/pt-br/rational-soft-arch/9.7.0?topic=diagrams-activity>.Acesso em: 15 nov. 2023.

IBM. Diagramas de classes. IBM Corporation, 2 mar. 2021. Disponível em: <https://www.ibm.com/docs/pt-br/rsas/7.5.0?topic=structure-class-diagrams>.Acesso em: 15 nov. 2023.

IBM. Use case diagrams. IBM Corporation, 2021. Disponível em: <https://www.ibm.com/docs/en/rational-soft-arch/9.6.1?topic=diagrams-use-case>. Acesso em: 17 jun. 2023.

JONES, Hollin. What Is a DAW and What Can You Do With It? Steinberg Media Technologies GmbH, 2023. Disponível em <https://www.steinberg.net/tutorials/what-is-a-daw>. Acesso em: 17 abr. 2023.

N, Juliana. Fundamentação teórica: exemplos e como fazer uma base em um trabalho acadêmico. Studybay, 01 dez. 2021. Disponível em: <https://mystudybay.com.br/blog/fundamentacao-teorica/?ref=1d10f08780852c55#o-que-e-fundamentacao-teorica>. Acesso em: 15 nov. 2023

SCHIABEL, Homero. Digitalização de Sinais. Universidade de São Paulo, 2023. Disponível em: <https://edisciplinas.usp.br/pluginfile.php/7510469/mod_resource/content/1/25-Amost.pdf> Acesso em: 14 nov. 2023.

SIMPER, Andrew. Solving the continuous SVF equations using trapezoidal integration and equivalent currents. Cytomic, 2023. Disponível em: <https://cytomic.com/files/dsp/SvfLinearTrapOptimised2.pdf>. Acesso em 15 nov. 2023

STEINBERG MEDIA TECHNOLOGIES GMBH. VST 3 Documentation. Steinberg Media Technologies GmbH, 2022. Disponível em: <https://steinbergmedia.github.io/vst3_doc/>. Acesso em: 10 abr. 2023.

SMITH, Julius O. Physical Audio Signal Processing for Virtual Musical Instruments and Audio Effects. Stanford, 2007. Disponível em: <http://www.dsprelated.com/dspbooks/pasp/> Acesso em: 15 nov. 2023.

SMITH, Steven W. The Scientist and Engineer's Guide to Digital Signal Processing. 2nd ed. California Technical Publishing, 1999. Disponível em: <https://users.dimi.uniud.it/~antonio.dangelo/MMS/materials/Guide_to_Digital_Signal_Process.pdf>. Acesso em: 10 abr. 2023.

WAMPLER, Brian. How to design a basic overdrive pedal circuit.Wampler, 2020. Disponível em: <https://www.wamplerpedals.com/blog/uncategorized/2020/05/how-to-design-a-basic-overdrive-pedal-circuit>. Acesso em: 17 abr. 2023.

ZDSP. Modelling Fuzz, Z Squared DSP Pty Limited, 4 sep. 2017. Disponível em: <https://z2dsp.com/2017/09/04/modelling-fuzz/>. Acesso em: 15 nov. 2023

ZDSP. Overdrive: What is it and how is it implemented in the Vector Drive? Part One, Z Squared DSP Pty Limited, 12 sep. 2017. Disponível em: <https://z2dsp.com/2017/09/12/overdrive-what-is-it-and-how-is-it-implemented-in-the-vector-drive/>. Acesso em: 15 nov. 2023

ZÖLZER, Udo. DAFX - Digital Audio Effects. Chichester: John Wiley & Sons, 2002. 533p.

APÊNDICE A – ESPECIFICAÇÂO DE CASOS DE USO

Quadro 2 - Especificação do CdU01

Código e Nome do Caso de Uso:	**CdU01 – Nova configuração**
Ator primário	Usuário
Fluxo principal de eventos	P1. O usuário pressiona o botão + no controle de cadeia P2. O sistema busca por dados não salvos P3 O sistema não encontra dados não salvos P4. O sistema volta para o estado padrão P5. O sistema limpa qualquer mensagem anterior
Fluxo alternativo de eventos	A1. **Alterações não salvas** A1.1. Em P2, o sistema encontra dados não salvos A1.2. O usuário descarta as alterações A1.3. O sistema volta para o estado padrão A1.4. O sistema limpa qualquer mensagem anterior

Fonte: elaborado pelo autor

Quadro 3 - Especificação do CdU02

Código e Nome do Caso de Uso:	**CdU02 – Carregar configuração**
Ator primário	Usuário
Fluxo principal de eventos	P1. O usuário pressiona seleciona uma predefinição P2. O sistema busca alterações na cadeia atual P3. O sistema não encontra alterações P4. O sistema carrega os efeitos da predefinição na memória P5. Os efeitos atuais são substituídos P6. O nome é atualizada para o nome da predefinição
Fluxo alternativo de eventos	A1. **Alterações não salvas** A1.1. Em P2, o sistema encontra alterações A1.2. O usuário aceita descartar as alterações A1.3. O sistema carrega os efeitos da predefinição na memória A1.4 O sistema substitui os efeitos atuais pelos novos A1.5. O sistema atualiza o nome da configuração atual
Fluxo de exceção	**E1. Falha ao ler disco** E1.1. Em P4 ou A1.3, o sistema é incapaz de ler o arquivo E1.2 O sistema avisa o usuário que algo deu errado **E2. Falha ao converter JSON** E2.1. Em P4 ou A2.3, o sistema é incapaz de converter o JSON em um objeto E2.2 O sistema avisa o usuário que algo deu errado

Fonte: elaborado pelo autor

Quadro 4 - Especificação do CdU03

Código e Nome do Caso de Uso:	**CdU03 – Trocar dois efeitos**
Ator primário	Usuário
Fluxo principal de eventos	P1. O usuário arrasta um efeito sob outro P2. O sistema troca os índices dos efeitos na cadeia P3. O sistema atualiza a tela

Fonte: elaborado pelo autor

Quadro 5 - Especificação do CdU04

Código e Nome do Caso de Uso:	**CdU04 – Ajustar volume de saída**
Ator primário	Usuário
Fluxo principal de eventos	P1. O usuário arrasta a barra deslizante inferior para cima P2. O sistema calcula quantos decibéis de ganho deve ser aplicado P3. O sistema altera o efeito de pós-processamento responsável pelo ganho de volume
Fluxo alternativo de eventos	**A1 – Redução de volume** A1.1 O usuário arrasta a barra deslizante inferior para baixo A1.2 O sistema calcula a quantidade de decibéis a ser reduzida A1.3 O sistema altera o efeito de pós-processamento responsável pelo ganho de volume

Fonte: elaborado pelo autor

Quadro 6 - Especificação do CdU05

Código e Nome do Caso de Uso:	**CdU05 – Salvar configuração**
Ator primário	Usuário
Fluxo principal de eventos	P1. O usuário pressiona o botão de salvar. P2. O sistema busca por configurações com o nome atual P3. O sistema não encontra outra configuração com o nome atual P4. O sistema salva a configuração P5. O usuário recebe uma confirmação do sistema
Fluxo alternativo de eventos	A1. **Já existe uma configuração com este nome** A1.1. Em P2, o sistema encontra uma configuração com o nome atual A1.2. O usuário aceita sobrescrever a outra configuração A1.3 O sistema sobrescreve a outra configuração A1.3. O usuário recebe uma confirmação do sistema
Fluxo de exceção	**E1. Falha ao escrever no disco** E1.1. Em P4 ou A1.3, o sistema é incapaz de salvar o arquivo E1.2 O sistema avisa o usuário que algo deu errado

Fonte: elaborado pelo autor

Quadro 7 - Especificação do CdU06

Código e Nome do Caso de Uso:	**CdU06 – Deletar configuração**
Ator primário	Usuário
Fluxo principal de eventos	P1. O usuário pressiona o botão de deleção P2. O sistema busca no disco a configuração com o nome atual P3. O sistema deleta o arquivo da configuração P4. O sistema retorna uma mensagem de sucesso para o usuário
Fluxo de exceção	**E1. Configuração não encontrada** E1.1. Em P2, o sistema é incapaz de encontrar a configuração E1.2 O sistema avisa o usuário que a configuração não existe **E2. Falha ao escrever no disco** E2.1. Em P3, o sistema é incapaz de escrever no disco E2.2 O sistema avisa o usuário que ocorreu um erro

Fonte: elaborado pelo autor

Quadro 8 - Especificação do CdU07

Código e Nome do Caso de Uso:	**CdU07 – Definir canal de entrada**
Ator primário	Usuário
Fluxo principal de eventos	P1. O usuário seleciona um dos três canais de entrada P2. O sistema atualiza o efeito de pré-processamento que lida com canais estéreo

Fonte: elaborado pelo autor

Quadro 9 - Especificação do CdU08

Código e Nome do Caso de Uso:	**CdU08 – Adicionar efeito**
Ator primário	Usuário
Fluxo principal de eventos	P1. O usuário arrasta um efeito até a seção com um grande símbolo de soma no meio P2. O sistema adiciona o novo efeito no fim da cadeia
Fluxo alternativo de eventos	**A1. Adicionar efeito em cima de outro** A1.1 O usuário arrasta o efeito na parte superior de outro efeito A1.2 O sistema calcula a posição do novo efeito A1.3 O sistema adiciona o novo efeito na posição calculada A1.4 O sistema atualiza a interface **A2. Adicionar efeito abaixo de outro** A2.1 O usuário arrasta o efeito na parte inferior de outro efeito A2.2 O sistema calcula a posição do novo efeito A2.3 O sistema adiciona o novo efeito na posição calculada A2.4 O sistema atualiza a interface

Fonte: elaborado pelo autor

Quadro 10 - Especificação do CdU09

Código e Nome do Caso de Uso:	CdU09 – Remover efeito
Ator primário	Usuário
Fluxo principal de eventos	P1. O usuário pressiona o botão de deletar efeito P2. O sistema remove o efeito e recalcula as posições dos demais efeitos

Fonte: elaborado pelo autor

Quadro 11 - Especificação do CdU10

Código e Nome do Caso de Uso:	CdU10 – Ativar efeito
Ator primário	Usuário
Fluxo principal de eventos	P1. Com o efeito desativado, o usuário pressiona o botão circular abaixo do botão de deletar P2. O efeito voltar a ser ativo e processado

Fonte: elaborado pelo autor

Quadro 12 - Especificação do CdU11

Código e Nome do Caso de Uso:	CdU11 – Desativar efeito
Ator primário	Usuário
Fluxo principal de eventos	P1. Com o efeito ativado, o usuário pressiona o botão circular abaixo do botão de deletar P2. O efeito deixa de ser ativo e para de ser processado

Fonte: elaborado pelo autor

Quadro 13 - Especificação do CdU12

Código e Nome do Caso de Uso:	**CdU12 – Ajustar volume de entrada**
Ator primário	Usuário
Fluxo principal de eventos	P1. O usuário arrasta a barra deslizante superior para cima P2. O sistema calcula quantos decibéis de ganho deve ser aplicado P3. O sistema altera o efeito de pré-processamento responsável pelo ganho de volume
Fluxo alternativo de eventos	**A1 – Redução de volume** A1.1 O usuário arrasta a barra deslizante superior para baixo A1.2 O sistema calcula a quantidade de decibéis a ser reduzida A1.3 O sistema altera o efeito de pré-processamento responsável pelo ganho de volume

Fonte: elaborado pelo autor

Quadro 14 - Especificação do CdU13

Código e Nome do Caso de Uso:	**CdU13 – Alterar parâmetro de efeito**
Ator primário	Usuário
Fluxo principal de eventos	P1. O usuário gira o botão circular na interface do efeito P2. O sistema atualiza o parâmetro correspondente do efeito sendo alterado

Fonte: elaborado pelo autor

APÊNDICE B – QUESTIONÀRIO DE AVALIAÇÂO DO SISTEMA FRETCAT

O quão familiar você é com efeitos de áudio digitais? *

Marcar apenas uma oval.

1 2 3 4 5

Você utilizaria efeitos digitais em suas apresentações e composições? *

Marcar apenas uma oval.

- Sim
- Não

O quanto você concorda com a afirmativa abaixo: *

"O Fretcat provê as ferramentas suficientes para criar timbres interessantes"

Marcar apenas uma oval.

1 2 3 4 5

O quão intuitiva foi a interface do Fretcat *

Marcar apenas uma oval.

1 2 3 4 5

O sistema funcionou corretamente durando o uso *

Marcar apenas uma oval.

1 2 3 4 5

Você consideraria usar o Fretcat em suas produções? *

Marcar apenas uma oval.

- Sim
- Não
- Talvez

O quanto você recomendaria o Fretcat para um amigo musico? *

Marcar apenas uma oval.

1 2 3 4 5 6 7 8 9 10

Qual nota você daria para o Fretcat? *

Marcar apenas uma oval.

1 2 3 4 5 6 7 8 9 10

Qual feedback você daria para o sistema?