Universidade Tecnológica Federal do Paraná – UTFPR
Departamento Acadêmico de Eletrônica– DAELN
Departamento Acadêmico de Informática– DAINF
Engenharia de Computação
Oficina de Integração 3 (IF66J) – S71 – 2016/2

Geanine Inglat – inglat.geanine@gmail.com
Mariana Carrião – mcarriao@alunos.utfpr.edu.br
Mariana F. M. Cabral – cabral.mariana@outlook.com

Dezembro de 2016

Resumo

Pedais para guitarras geralmente possuem custo elevado e funções específicas; isso faz com que as pessoas tenham que adquirir diversos pedais para ter uma maior possibilidade de efeitos sonoros. No presente relatório apresenta-se um pedal muitiefeitos, denominado GemFX, baseado no *Raspberry Pi* 2 e comunicação *Wi-Fi* com um aplicativo para *smartphone Android*, que atua como interface com o usuário. Os efeitos disponíveis são: *distortion, chorus, delay* e *wahwah* (que também foi chamado pela equipe de efeito GemFX) todos parametrizáveis. Estes efeitos podem ser utilizados de forma individual ou combinados. As ferramentas utilizadas para o desenvolvimento do GemFX foram: *Java* aplicado ao sistema *Android* para o desenvolvimento do aplicativo, *Pure Data* para desenvolver os efeitos *distortion, chorus e delay* e linguagem C (para o desenvolvimento do *Wahwah*) que, posteriormente foi importado como um objeto para *Pure Data*. Para o desenvolvimento do servido de comunicação entre o *Raspberry Pi* 2 e o aplicativo GemFX, foi utilizada a linguagem *Python*.

1 Introdução

Instrumentos musicais e acessórios para estes nem sempre são de fácil obtenção; geralmente possuem custo elevado e funções muito específicas. No caso dos pedais para guitarra, por exemplo, essa "restrição"de funcionalidades acaba por forçar o usuário a adquirir diversos produtos para diferentes efeitos desejados. Além disso, nem sempre o usuário é um músico profissional que possui familiaridade com os equipamentos e, para alguns, a utilização destes acessórios pode não ser trivial. A possibilidade de modificar o som (sinal de áudio) de um instrumento e obter uma saída totalmente diferente da entrada, proporciona aos músicos efeitos interessantes. Esse efeitos, que podem ser obtidos através do uso de pedais, podem ajudá-los no processo de criação, gerando sons de saída impactantes, com o objetivo de impressionar o público, além de acrescentar personalidade ao trabalho [1].

O funcionamento de uma guitarra consiste, basicamente, em transformar ondas mecânicas provenientes das vibrações que o usuário causa nas cordas em ondas elétricas [1], através de um transdutor denominado de captador - converte um sinal proveniente de uma forma física para um sinal correspondente

em outra forma física [2]. Os comprimentos e tensões de cada corda determinam a frequência de cada uma delas. O sinal do captador é aplicado a um amplificador de áudio e enviado para auto-falantes. Os pedais, conforme mencionado anteriormente, são acessório utilizados para alterar o sinal de áudio fornecido pelo captador, antes de enviá-lo ao amplificador.

Visando proporcionar ao usuário a alternativa de poder encontrar os diversos efeitos disponíveis em um único pedal, evitando assim a necessidade da aquisição de diversos pedais para efeitos específicos e, dessa forma, reduzindo os custos, foi decidido por desenvolver este projeto. No decorrer deste relatório, será discutido e apresentado o desenvolvimento do projeto do pedal multiefeitos para guitarra, GemFX, que possibilita a substituição dos pedais de guitarra utilizados para modular as ondas sonoras por um sistema no qual, através de um aplicativo para *smartphone Android*, por comunicação *Wi-fi* com o *Raspberry Pi* 2, é possível escolher diferentes efeitos e seus respectivos níveis, além de possibilitar que os efeitos sejam utilizados em conjunto e então reproduzidos no amplificador. Para isso, como é possível observar na Figura 1, o adptador de áudio, *USB Guitar Link Cable* [3] é conectado com os seguintes componentes: à guitarra (entrada), ao amplificador (saída) e também em uma porta USB do *Raspberry Pi* 2 (onde o áudio será processado e tratado antes de seguir para o amplificador). Então, após a conexão desses equipamentos, é necessário conectar o*smartphone* na mesma rede *Wi-fi* em que o *Raspberry Pi* 2 está conectado, abrir o aplicativo e o GemFX está pronto para o uso.

Para a programação dos efeitos sonoros, foi utilizada a linguagem de programação visual *Pure Data* [4] e também a linguagem C para receber, modular/tratar as ondas e gerar os seguintes efeitos e seus respectivos parâmetros:

- *Delay* - atraso definido pelo *time* e a amplitude/intensidade pelo *level*;
- *Distortion* - *level*;
- *Chorus* - *depth* (profundidade/intensidade) e *rate* (taxa de sinais repetidos);
- Equalizador + *Wahwah* - opções *low, mid, high* e *wahwah level*. Este último também está referido no aplicativo e neste relatório como *gemeffect*.

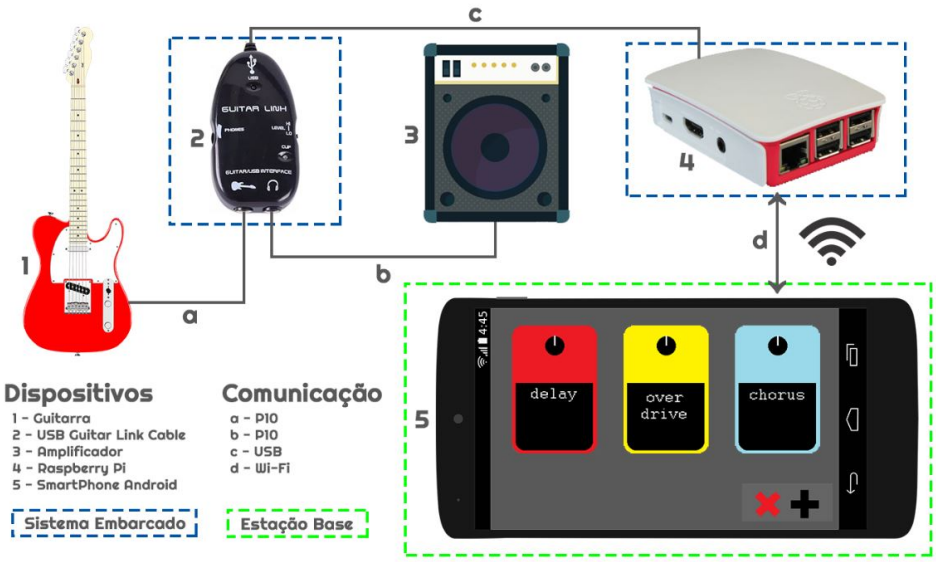

Figura 1: Visão geral do projeto pedal multi efeitos para guitarra.

Os requisitos funcionais especificados do projeto são:

1. O *software* deverá permitir ao usuário selecionar o efeito sonoro desejado;
2. O *software* deverá permitir ao usuário selecionar uma amplitude desejada dentro de limites pré-definidos;
3. O *software* deverá permitir ao usuário selecionar mais de um efeito sonoro ao mesmo tempo;
4. O sistema deverá permitir ao usuário a troca de efeitos em tempo real;
5. O sistema deverá permitir ao usuário fazer ajustes e configurações dos efeitos sonoros através de uma interface em uma estação base;

E os requisitos não funcionais,

1. O *software* deverá se comunicar via *Wi-Fi* com o restante do sistema;
2. O *software* deverá ter quatro efeitos sonoros para o usuário utilizar de forma individual ou combinada;
3. O *software* deverá ser desenvolvido em linguagem Java com *Android*;
4. A documentação final deverá conter um Manual do Usuário;

2 Aplicativo Musical para *Android* - GemFX

Android é um sistema operacional desenvolvido para *smartphones* e *tablets* desenvolvido pela empresa *Google*. É um sistema aberto, o que possibilita que diversos desenvolvedores (e empresas fabricantes de *hardware*) apliquem modificações para melhor atender suas necessidades [5]. Por conta disso, é o sistema operacional com público predominante sobre os demais, com quase 97%

do mercado mundial de *smartpones* e *tablets* vendidos no mundo atualmente [6].

Aplicativos são ferramentas de *software* executadas no *smartphone* ou *tablet* e que possuem fins específicos que visam auxiliar a experiência do usuário em diversas funções como, por exemplo, simular uma calculadora, armazenar lembretes de tarefas, calendário, etc [7].

O aplicativo GemFX foi desenvolvido para ter sua execução suportada pelo sistema operacional *Android* com o intuito de que o usuário possa obter entretenimento com seu uso, além de incentivar a criatividade. Considerando o propósito do aplicativo desenvolvido, além dos objetivos e funcionalidades propostas, a aparência e usabilidade deste é um fator muito importante em seu processo de desenvolvimento [8]. Ou seja, a interface gráfica do aplicativo tem uma alta relevância neste projeto. Com foco nesses aspectos, a interface do aplicativo foi planejada e desenvolvida para ser limpa e intuitiva de modo a facilitar o entendimento e visualização dos recursos na tela multitoque do usuário, independente do tamanho desta [9].

O aplicativo desenvolvido foi denominado de GemFX e foi desenvolvido em linguagem Java voltado para *Android*, conforme especificado. Após o levantamento de requisitos e prototipação, a seguinte interface gráfica para a tela inicial - e principal - do aplicativo foi obtida e apresentada na Figura 2:

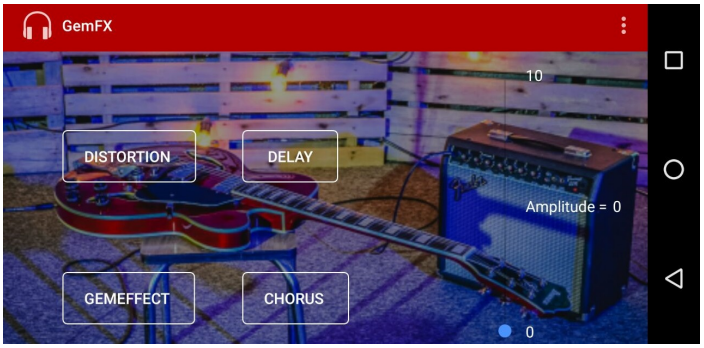

Figura 2: Tela inicial do aplicativo GemFX.

Cada tela do *Android* é chamada de *activity*. O aplicativo para *Android* GemFX possui três (3) *activities*: *main_activity*, *settings_activity* e *toolbar*, sendo a última uma *activity* utilizada sobre as outras e não uma tela inteira. Na tela incial (*main_activity*) é possível conferir quatro (4) botões do tipo *toggle* - esse botão possui um *switch*, o qual muda entre dois estados (ligado - *on* e desligado - *off*)correspondentes aos efeitos sonoros disponíveis (que estão na cor branca, indicando que eles estão desativados/desligados): *distortion*, *delay*, *gemeffect* e *chorus*.

Além disso, é possível verificar uma barra vertical com um sinalizador (em azul claro) que corresponde ao nível de amplitude selecionado, que por padrão, está zerado no momento em que se inicia o aplicativo. Todas as configura-

ções estão zeradas quando o aplicativo é aberto a fim do usuário testar o som "limpo"vindo da guitarra para o devido ajuste das cordas.

Para ativar um ou mais efeitos, basta selecionar o(s) efeito(s) desejado(s) e a coloração passará de branca para verde (estado ligado); além disso, a palavra *"ON"* aparecerá escrita após o nome desse efeito, dentro do botão. É possível observar na Figura 3 os efeitos *delay* e *gemeffect* (*wahwah*) ligados (em verde) e o *distortion* e *chorus* desligados (em branco).

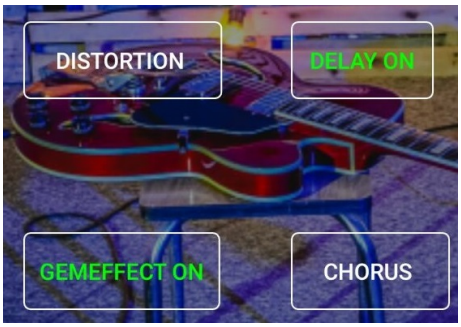

Figura 3: Efeitos *Delay* e *Gemeffect (Wahwah)* ativados.

No canto superior direito (que é a *toolbar*), é possível acessar a outra *activity* presente no aplicativo, as configurações *settings_activity*, a qual contém todos os parâmetros (de cada efeito) que podem ser escolhidos conforme o usuário desejar. A interface onde o usuário escolhe esses parâmetros pode ser vista na Figura 4.

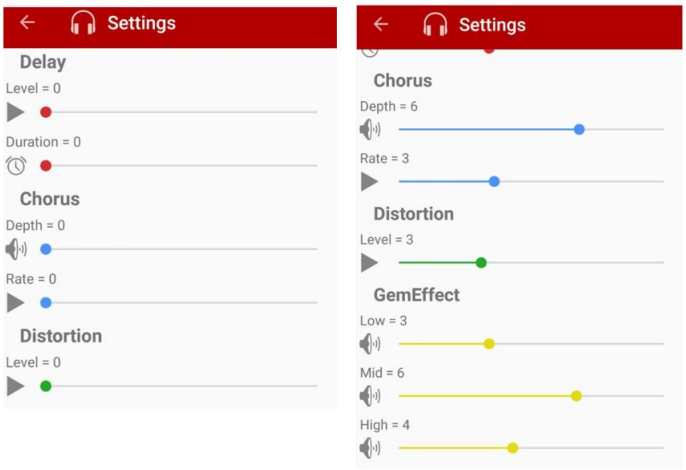

Figura 4: Tela *Settings* do aplicativo GemFX.

Após escolher os valores para cada parâmetro desejado, basta voltar à tela inicial que estes são salvos de forma automática e permanente, ou seja, se o

usuário utilizar novamente o aplicativo, todos os valores configurados anteriormente são carregados com os devidos parâmetros na *settings_activity*.

3 Efeitos sonoros

Os quatro efeitos sonoros desenvolvidos foram: *Fuzz*, *Delay*, *Chorus* e *WahWah* (*gemeffect*). A equipe optou por desenvolver três dos quatro efeitos em uma linguagem de programação visual chamada *Pure Data* [4]. Essa linguagem pode ser executada em computadores, *Raspberry Pi, Arduino* e até em *smartphones*. Pode ser utilizada para processar e gerar sons, vídeos, MIDI (*Musical Instrument Digital Interface*). A construção do código se dá através de caixas chamadas de objetos que possuem funções específicas e que podem ter diferentes níveis de complexidade, como uma operação matemática ou até mesmo uma transformada FFT (*Fast Fourier Transform*) [11]. O quarto efeito sonoro (*Wahwah* ou *Gemeffect*) foi desenvolvido em linguagem C, onde foi necessário receber a onda de entrada utilizando-se de um filtro passa-faixa com uma faixa pequena tolerada e uma frequência de faixa que varia com o tempo sendo que a implementação deste exige que ele sofra variações na frequência. A equipe optou por integrar o efeito desenvolvido em linguagem C ao *Pure Data* para que todos ficassem "centralizados"em uma única linguagem; o *Pure Data* possui essa flexibilidade: é possível desenvolver caixas (objetos) personalizados e integrar em projetos. Todos os efeitos foram convertidos de analógico para digital, para então sofrerem as alterações necessárias para que os efeitos fossem obtidos e então convertidos novamente de digital para analógico para então saírem no amplificador conectado ao sistema. A teoria explanada no tópico acima foi estudada de modo aprofundado e compreendida pela equipe para que fosse possível o desenvolvimento e apresentação dos efeitos já descritos.

3.1 Efeito *Distortion*

Também conhecido como *Fuzz* ou *Overdrive*, ocorre quando o sinal de entrada tem sua amplitude aumentada até que ocorra ceifação (corte) do sinal. Em dispositivos eletrônicos, este tipo de distorção ocorre quando o ganho do amplificador é maior do que o suportado, fazendo com que a tensão de saída seja maior do que a capacidade do amplificador. Para fazer o efeito de forma digital é necessário ceifar os sinais manualmente, uma vez que os limites suportados para o sinal de saída são significativamente mais altos.

Para desenvolver o efeito no *Pure Data*, o sinal de entrada foi obtido utilizando o objeto "adc ", um objeto nativo da linguagem que captura o som recebido na placa de som. De forma similar, o objeto "dac "envia o sinal de audio do PureData para a placa de som. Ambos os objetos também foram utilizados nos demais efeitos. O objeto “* ” recebe como entrada o sinal e um parâmetro numérico. Sua saída consiste na multiplicação do sinal pelo número recebido. Este objeto foi utilizado para aumentar a amplitude do sinal de entrada. Para

ceifar o sinal, usou-se o objeto “clip ”, um ceifador simples também nativo da linguagem. O diagrama Figura 5 apresenta o efeito desenvolvido (assim como os objetos citados acima). Os demais objetos foram utilizados para a comunicação *Wi-fi* e serão explicados devidamente na seção 5.

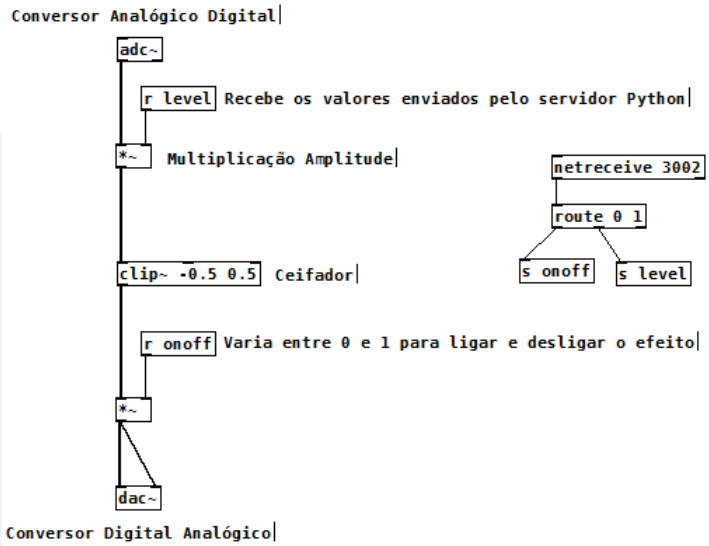

Figura 5: Diagrama em *Pure Data* que corresponde ao efeito *Distortion*

3.2 Efeito *Delay*

Conhecido como "atraso"em português, consiste apenas em adicionar uma cópia do sinal de entrada ao sinal original, com um determinado atraso [10].

No *Pure Data* os objetos “delwrite” e “delread” foram utilizados para criar este efeito. O "delwrite"cria uma nova linha de áudio e o “delread” reproduz a linha criada, tomando como parâmetro o tempo de atraso para reproduzir o áudio. A Figura 6 mostra o efeito utilizado no projeto:

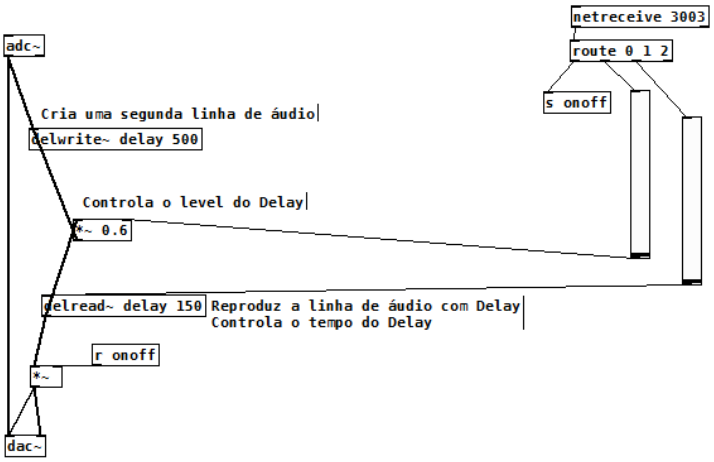

Figura 6: Diagrama em *Pure Data* que corresponde ao efeito *Delay*

3.3 Efeito *Chorus*

Este tipo de efeito acontece quando dois ou mais sinais de frequência e timbre aproximados são percebidos como apenas um sinal. O efeito Chorus é feito adicionando uma ou mais cópias do sinal original ao sinal de saída. Estas cópias tem geralmente seu tom modulado por osciladores de baixa frequência [10].

Para o projeto desenvolvido, foram adicionadas três cópias ao sinal original, todas moduladas com osciladores de baixa frequência que tem como parâmetro valores que controlam o *"rate"* e o *"depth"* do efeito. Para criar os osciladores foram utilizados os objetos "phasor " e "cos ", combinando ondas senoidais e cossenoidais. Também foram utilizadas funções matemáticas que multiplicam e somam sinais, os valores foram obtidos experimentalmente até que um som satisfatório fosse alcançado. A Figura 7 mostra o efeito criado em *Pure Data*.

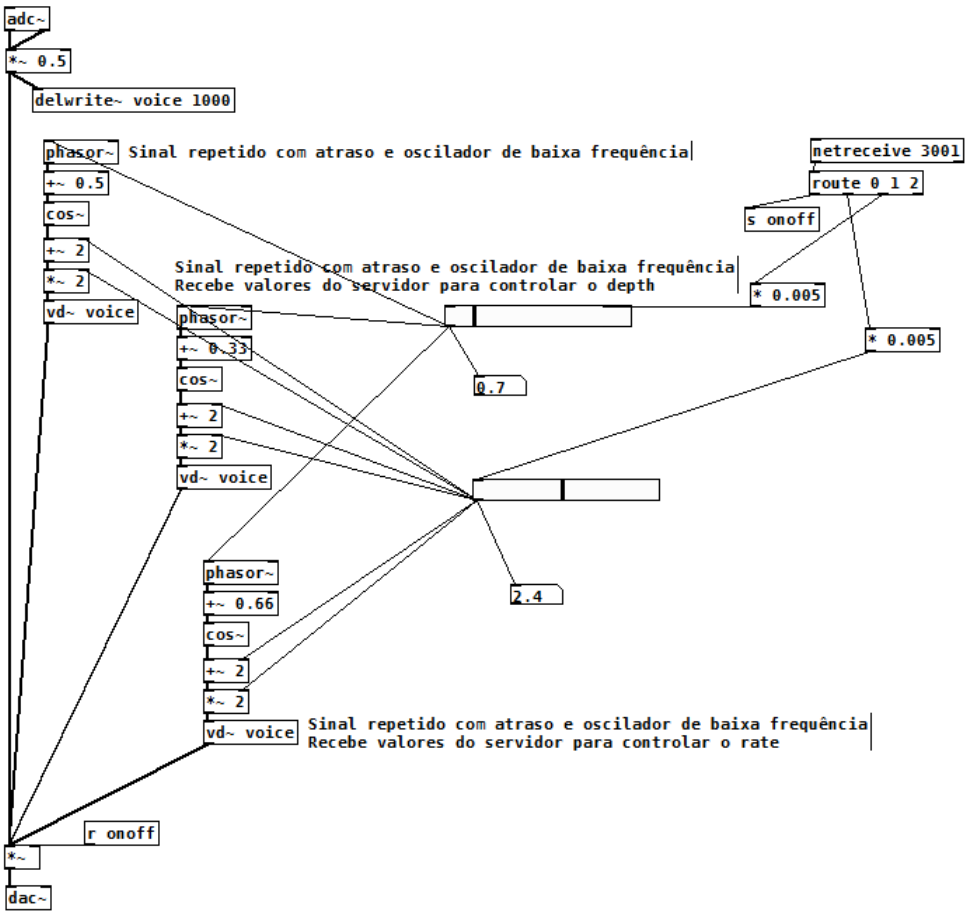

Figura 7: Diagrama em *Pure Data* que corresponde ao efeito *Chorus*

3.4 Efeito *WahWah* - GemFX

O nome real do gemeffect é *WahWah*, considerado um efeito de variação no tempo. Consiste em um filtro passa-faixa com uma faixa pequena e uma frequência de faixa que varia com o tempo. Para implementar o *WahWah* o sinal original passa pelo filtro passa-faixa sofrendo variações na frequência. Estas variações dependem da frequência de faixa do filtro. Em sistemas analógicos a frequência de faixa pode ser controlada por uma tensão variável que geralmente pode ser controlada pelo usuário. No efeito desenvolvido a frequência de faixa foi controlada por um oscilador senoidal onde o usuário pode controlar o período da onda [11].

O efeito *WahWah* foi desenvolvido na linguagem C e posteriormente integrado ao *Pure Data*. O *Pure Data* é uma linguagem visual *open source* com base na linguagem C, o que permite que novos objetos sejam criados em C e depois utilizados no *Pure Data*. Para realizar esta integração é necessário obter o có-

digo fonte e utilizar as bibliotecas já existentes do *Pure Data*. A biblioteca *Pd-Extended* possui um filtro passa-faixa, este foi utilizado como base para desenvolvimento do efeito *WahWah*. Os objetos do *Pure Data* são declarados como *structs*, e possuem como padrão a seguinte estrutura:

```
typedef struct wahwah
{
    t_object x_obj;
    t_wahwahctl x_cspace;
    t_wahwahctl *x_ctl;
} t_wahwah;
```

Um *"external"*, como são chamados os objetos integrados ao *Pure Data*, apresenta por padrão uma entrada de sinal. O efeito criado pela equipe apresenta quatro entradas que recebem os parâmetros que controlam as características do *WahWah*. A criação das entradas adicionais e da saída do objeto podem ser vistas no código abaixo:

```
    float *in1 = (float *)(w[1]);
    float *in2 = (float *)(w[2]);
    float *in3 = (float *)(w[2]);
    float *out1 = (float *)(w[3]);
```

Os quatro parâmetros do efeito criado controlam a banda do filtro passa-faixa. Uma senóide é criada a partir do primeiro parâmetro "in1"e o tempo do sistema. Esta senóide representa a abertura da banda do filtro, o que cria o efeito de *"WahWah"* percebido no resultado final. O pedaço de código abaixo mostra como a onda senoidal foi criada e também como foi adicionada à saída do filtro.

```
//Hora do sistema em milissegundos, atualizada sempre que há sinal
//de entrada no filtro
double time_in_mill = (tv.tv_sec) * 1000 + (tv.tv_usec) / 1000 ;
//Criação da onda senoidal
f1 = (float)sin((double)f1*time_in_mill);
//Saída do filtro passa-faixa com a banda multiplicada pela
//onda senoidal
*out1 = re = ampcorrect * oneminusr * f1 + coefr * re2 - coefi * im;
```

O código deve ser compilado e apresentar o mesmo diretório do código fonte da linguagem PureData, desta forma é possível integrar totalmente o objeto à linguagem. O diagrama da Figura 8 mostra o objeto criado sendo utilizado para o efeito *GemEffect*:

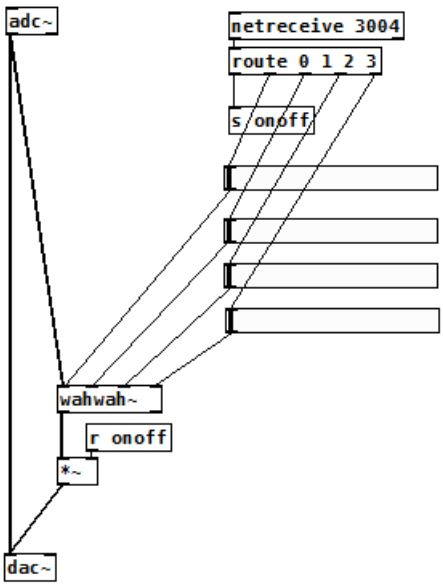

Figura 8: Diagrama em *Pure Data* que corresponde ao efeito *WahWah*

4 Transmissão de dados utilizando a biblioteca *Volley*

Para enviar os dados do aplicativo *Android* para o sistema embarcado via *Wi-fi*, é necessário o uso de algum protocolo HTTP. A equipe optou pela biblioteca *Volley* disponível em *Android*.

4.1 O que é *Volley*?

Volley é uma biblioteca de HTTP (*Hypertext Transfer Protocol*) desenvolvida inteiramente pela *Google*. A *Google* desenvolveu essa biblioteca devido a deficiência das classes de *networking* do *Android Studio*. As classes existentes não funcionavam corretamente o que acarretava em interferir na experiência do usuário. Antes da criação da *Volley*, uma comunicação HTTP no *Android Studio* era feita somente com a *Java class java.net.HttpURLConnection* e o *Apache org.apache.http.client*. Com essas ferramentas era possível desenvolver um servidor tipo REST (*Representational State Transfer*) [12], o qual será melhor explicado mais adiante, na seção seguinte.

4.2 O uso da biblioteca *Volley*

Ao utilizar a biblioteca, evita-se o uso de *HttpURLConnection* e *HttpClient*. A biblioteca *HttpClient* passou a ser obsoleta a partir da API (*Application Programming Interface*) 22.Para realizar uma conexão HTTP com essas classes, era obrigatório o uso de tarefas assíncronas como era utlizado anteriormente a sua

criação, porém um problema de tarefas assíncronas é que elas funcionam tipo FIFO (*First in First Out*), não podendo, assim, dar diferentes prioridades a *requests*, nem indicar quais *requests* devem ser enviados primeiro nem quais devem aguardar envio. A biblioteca *Volley* utiliza tarefas assíncronas, porém possui um melhor tratamento com tarefas no *background*, podendo assim, tratar melhor as prioridades das tarefas [12].

Outros benefícios:

- Programa automaticamente *requests* [13];
- É capaz de suportar múltiplas conexões [13];
- Armazena em *cache* qualquer *request* de forma automática [13];
- É possível cancelar qualquer *request* individualmente [13].

4.3 Funcionamento

Na Figura 9 é possível visualizar o diagrama que explica o funcionamento da biblioteca *Volley*.

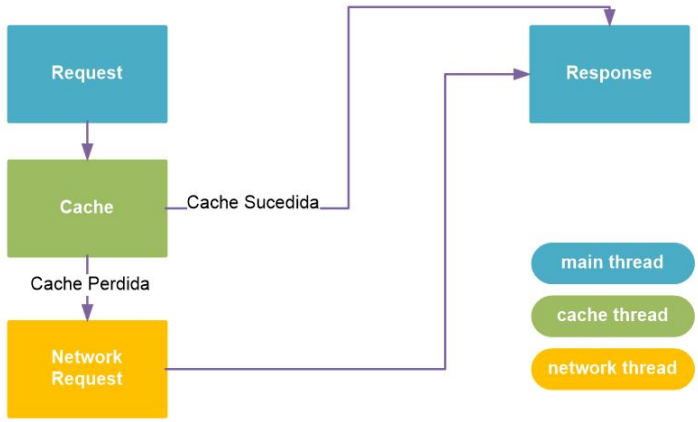

Figura 9: Diagrama de blocos do funcionamento da biblioteca *Volley*.

- ***Main Thread:*** A *Thread* Principal consiste somente em enviar um *request* e tratar a sua resposta;
- ***Cache*** e ***Network Thread***: quando adicionado um *request* na fila, a *Volley* verifica se esse *request* pode ser enviado pela *cache*, se sim, então esse *request* é enviado e a resposta é aguardada. Se não, então o *request* é passado para a *Network Thread*.

Dentro da *Network Thread* é realizado um *Round-Robin* (adaptação do escalonamento FCFS - *First come, first server* mas com um *quantum* - limite de tempo de processamento que cada tarefa recebe) [14]. A primeira tarefa disponível retira um *request* da fila de tarefas, realiza um HTTP *Request*, analisa a resposta e escreve na *cache*. No final do processo, a resposta analisada é enviada para a *Main Thread* onde os *listeners* estão esperando a resposta para tomar as devidas ações [12].

4.4 Uso da biblioteca *Volley* no aplicativo GemFX

Na *main_activity* do aplicativo foi declarada uma fila de *Requests*.

```
@Override
protected void onCreate(Bundle savedInstanceState){
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    final RequestQueue queue=Volley.nevRequestQueue(this);
}
```

Essa fila de *requests* serve para gerenciar as diferentes tarefas em execução, avaliar as respostas e enviar novamente para a *Main Thread*. Dependendo do efeito selecionado, seu parâmetro é salvo numa *URL* (*Uniform Resource Locator* - endereço de um recurso disponível em uma rede) e é chamado o método para enviar o *request* [15].

```
//envia o novo level (nível)
String url_level = host + "settings/distortion/level/" +
Integer.toString(dil);
sendRequest(url_level, queue);
```

A função *sendRequest* tem como parâmetro a *URL* a ser enviada e a fila de *requests*. Em todos os *requests* feito no aplicativo GemFX é utilizado o método GET, o qual espera-se uma resposta. Assim que a resposta chegar é exibido um *Toast* - pequena *pop-up* de mensagem para informar *feedback* ao usuário. Após isso o *request* é adicionado a fila de *requests*.

```
public void sendRequest(String url, RequestQueue queue)
{
    StringRequest stringRequest = new
    StringRequest(Request.Method.GET, url, (response) -> {
    //exibe a resposta do toast
    Toast.makeText(getApplicationContext(), response,
    Toast.LENGTH_LONG.show();
    }, error -> {
    //caso falhe
```

```
    //Toast.makeText(getApplicationContext(), "falha de comunicação",
    Toast.LENGTHLONG.show();
    ]);
    queue.add(stringRequest);
}
```

No momento em que é dado o *queue.add(stringRequest)* esse *request* é tratado como no diagrama exposto na Figura 10. Os *requests* são enviados ao servidor *Python* explicado na seção 5.

5 Comunicação *Wi-fi*

Uma vez os dispositivos (*Smartphone Android* e *Raspberry Pi* 2) conectados na mesma rede, basta iniciar o Servidor em *Python* que o Cliente (Aplicativo GemFX) já estará pronto para enviar os *requests*. Cada *request* recebido no servidor é analisado, processado e os devidos valores são enviados para o Cliente *Pure Data* explicado a seguir:

5.1 *Servidor*

O servidor foi desenvolvido utilizando a linguagem *Python*, e é do tipo *RESTful*, esse tipo de servidor descreve um conjunto de princípios de arquitetura pelo qual os dados podem ser transmitidos através de um protocolo como o *HTTP*. Para acessar um recurso utilizando uma URI (*Uniform Resource Identifier*) e uma representação do recurso é devolvida. As operações *GET, PUT, DELETE* e *POST* podem ser executadas nesse recurso.

5.2 Implementação

O servidor *python* foi desenvolvido usando a *micro-framework flask*. Para instalar a *micro-framework* basta usar o gerenciador de pacotes pra *python* 'pip'. Após a instalação, primeiramente, importa-se a biblioteca:

```
fro flask import Flask
app = Flask(__name__)
```

Feito isso basta definir as rotas, isto é, os endereços que serão digitados no navegador:

```
@app.route("/")
def hello():
return "Hello World!"
```

Neste exemplo, ao acessar o endereço raiz ("/"), a função *'hello'* será executada, retornando a frase *"Hello World!"* ao navegador. Um retorno para a *thread*

que realizou o *request* é obrigatório, e a função pode executar qualquer código. Aqui esta um exemplo de uma rota do código da equipe no *Android*:

```
@app.route('/gemfx/settings/<effect>/<setting>/<int:value>')
```

Essa rota recebe 3 (três) valores como argumentos enviados pelo aplicativo GemFX. Os argumentos são *effects, setting* e *value*. Como, por exemplo,

```
@app.route('/gemfx/settings/distortion/level/5')
```

Isso é um *request* para ajustar o nível (*level*) do *distortion* em 5.

6 Resultados

Conforme pôde ser observado no decorrer do presente relatório, nas seções acima, a equipe considera que obteve êxito em todos os objetivos propostos:

- os efeitos sonoros foram desenvolvidos com sucesso (em *Pure Data* e em linguagem C).
- a comunicação *Wi-fi* entre *Raspberry Pi* 2 e *smartphone* funcionou corretamente.
- o aplicativo foi desenvolvido e está totalmente funcional - para *Android* em linguagem Java aplicado.
- a integração entre o código desenvolvido em C e o *Pure Data*.
- a integração entre os efeitos implementados em *Pure Data* (incluindo o efeito em C que foi importado também para essa linguagem) e o aplicativo para *Android* utilizando a linguagem *Python*.

Além disso, a equipe realizou o cálculo com o valor que o usuário teria que desembolsar para adquirir os equipamentos necessários para montar o pedal multiefeitos - que estão dispostos na Tabela 1 (os preços estão cotados em dólares e convertidos para reais utilizando a cotação do dólar no valor de R$3,36, considerando que algum desses equipamentos, em seu site original, possuem preço em dólar).

Tabela 1: Custo para adquirir equipamentos necessários para montagem do pedal multiefeitos proposto neste relatório.

Equipamento	Custo (U$)
Raspberry Pi 2 - *model B*	42
Wi-fi Dongle	9
USB *Guitar Link Cable*	8
Total	59

Logo, se convertido para reais, o custo total para adquirir os equipamentos é de aproximadamente 200 reais (R$198,24). A equipe reconhece que os preços podem variar de acordo com as lojas, mas foi realizada uma pesquisa e escolhido os preços em locais que a equipe considera confiável para adrquiir tais equipamentos com qualidade.

O *smartphone* com sistema operacional *Android* não foi colocado como um equipamento imprescindível, pois o usuário pode fazer uso de um emulador de celular em um computador através do *software Android Studio*. Porém, caso o usuário deseje (ou sinta necessidade de) adquirir um *smartphone* para utilizar o aplicativo GemFX, a equipe recomenda que seja utilizado um celular com alguma API mais recente - a partir do KitKat (versão 4.4). Foi realizada uma pesquisa e este poderia ser adquirido a partir de 600 reais.

A equipe optou por fazer uma segunda tabela de custos, já que para que seja possível realmente testar o projeto, é necessário a aquisição dos componentes listados na Tabela 2.

Tabela 2: Custo dos componentes necessários para utilizar, de fato, o pedal multiefeitos.

Equipamento	Custo (R$)
Guitarra	600
Amplificador	250
Cabos P-10 - P-10	30 (x2)
Total	910

Então para um custo total adquirindo todos os equipamentos o usuário gastaria 1110 reais para criar seu próprio pedal multiefeitos para guitarra.
Para efeito de comparação, um pedal multiefeitos para guitarra analógico, sem comunicação com o celular e sem amplificador custa 189 dolares (635 reais), e mesmo assim, o usuário ainda precisaria adquirir os componentes para teste - listados na Tabela 2.

Também foi confeccionado um Manual do Usuário, que está disponível no blog do projeto.

7 Conclusão

Os objetivos estipulados no início do semestre foram concluídos com sucesso pela equipe. O aplicativo GemFX foi desenvolvido para *smartphone Android*, realiza comunicação *Wi-fi* com o *Raspberry Pi 2* que recebe as informações e as interpreta realizando as modificações nos sinais de entrada da guitarra, além dos parâmetros de cada efeito de forma individual, alterando o sinal de entrada e utilizando-o como saída no amplificador. Conforme especificado inicialmente, os efeitos desenvolvidos foram: *chorus, distortion, delay e WahWah - Gemeffect* - sendo que os três primeiros foram desenvolvidos em *Pure Data* e o último em linguagem C integrado em *Pure Data*.

Como implementação e melhoria futura, a equipe pretende utilizar o GemFX em um show, dar continuidade a este projeto modificando o *Raspberry Pi* 2 para funcionar como um *access point* e incluindo a possibilidade de que os usuários consigam atualizar o aplicativo de forma a obterem mais efeitos sonoros para utilizar.

Agradecimentos

A equipe gostaria de agradecer aos familiares pela paciência e compreensão durante o desenvolvimento do projeto; ao *pub John Bull* que cedeu gentilmente o espaço para gravação de parte do vídeo da disciplina; ao Ruani Paz, guitarrista da banda Dia-D que foi convidado para participar do vídeo e testar o pedal, ao Lucas Chociay que auxiliou e apoiou a equipe sempre que foi necessário. André Fedalto que emprestou o *Raspberry Pi 2*, Arthur Floriani que emprestou o roteador e Edgar Pereira por emprestar a guitarra. A equipe também gostaria de agradecer aos profesores da disciplina de Oficina de Integração 3, Guilherme Alceu Schneider e Gustavo Benvenutti Borba por toda orientação prestada durante o desenvolvimento do projeto e também por estarem sempre disponíveis para os alunos. A equipe reconhece que sem o apoio e orientação das pessoas citadas, a realização e conclusão do projeto não teria tido o êxito na conclusão de tudo o que foi proposto.

Referências

[1] Gustavo Magalhães dos Santos. Efeitos digitais para guitarra elétrica, 2011. Download: `http://repositorio.uniceub.br/jspui/bitstream/123456789/3316/2/20317602.pdf`.

[2] Valner Brusamarello. Introdução de transdutores. Download: `https://chasqueweb.ufrgs.br/~valner.brusamarello/eleinst/ufrgs6.pdf`.

[3] Adaptador usb guitar linnk cable - cirilo cabos. Download: `http://www.cirilocabos.com.br/adaptador-usb-guitar-linnk-cable-7218/p`.

[4] Pure Data. What is pure data? Download: https://puredata.info/.

[5] Ricardo Bergher. O que é um celular android?, 2016. Download: https://www.zoom.com.br/celular/deumzoom/o-que-e-um-celular-android.

[6] Raquel Freire. 96,8 por cento de todos os smartphones vendidos no planeta são android ou iphone, 2015. Download: http://www.techtudo.com.br/noticias/noticia/2015/08/968-de-todos-os-smartphones-vendidos-no-planeta-sao-android-ou-iphone.html.

[7] Guilherme Santa Rosa. Mas afinal, o que é um app?, 2012. Download: http://fabricadeaplicativos.com.br/fabrica/mas-afinal-o-que-e-um-app/.

[8] Bruno Rohde, Cristiano Figueiró, Guilherme Rafael Soares. Explorações e estratégias em música móvel: Processo criativo de produção de código aberto para experimentações sonoras interativas em dispositivos com android, 2016. Download: https://siimi.medialab.ufg.br/up/777/o/34_musica_movel.pdf.

[9] Jerônimo Barbosa. Uma interface multitoque para processamento de áudio em tempo real, 2011. Download: http://sites.itaucultural.org.br/rumosarteciberneticapdf/Paper_Jeronimo-barbosa.pdf.

[10] Dave Hunter. Effects explained: Modulation—phasing, flanging, and chorus. Download: http://www.gibson.com/News-Lifestyle/Features/en-us/effects-explained-modulation.aspx.

[11] Cardiff University. Digital audio effects. Download: http://users.cs.cf.ac.uk/Dave.Marshall/CM0268/PDF/10_CM0268_Audio_FX.pdf.

[12] Gianluca Segato. An introduction to volley. Download: https://code.tutsplus.com/tutorials/an-introduction-to-volley--cms-23800.

[13] Transmitting network data using volley. Download: https://developer.android.com/training/volley/index.html.

[14] Dr. Carlos A. Maziero. *Sistemas Operacionais: Conceitos e Mecanismos*. 2014.

[15] Sending a simple request. Download: https://developer.android.com/training/volley/simple.html.