Termpot: criação, edição e execução de funções no navegador em tempo de execução .

Guilherme Lunhani[1] **, Flávio Luiz Schiavoni**[2]

[1]Instituto de Artes e Design – Universidade Federal de Juiz de Fora
Juiz de Fora, MG

gcravista@gmail.com, ttonmeister@gmail.com

[2]Departamento de Computação – Universidade Federal de São João Del Rei
São João Del Rei, MG

fls@ufsj.edu.br

Abstract. *Researchs on sound synthesis using Internet browsers aren't new, but a group of developers used algebraic representations of waves as a online compositional strategy. Javascript language is used to process in a single data vector. With the research on this approach, we have developed a software with a different compositional viewpoint, using synthesis process. We'll present motivations, current, technical details, problems and future plans.*

Resumo. *Pesquisas sobre síntese sonora utilizando navegadores de internet não são novas, mas um grupo de desenvolvedores, Wavepot.com, utilizou representações algébricas de ondas como estrategia composicional online. Em linguagem javascript é processado um unico vetor de dados. Com a pesquisa desta abordagem, desenvolvemos um software com um ponto de vista da composicional diferente, utilizando o processo de síntese. Apresentaremos motivações, detalhes técnicos, problemas e planos futuros.*

1. Introdução

A *Web Audio API* [**?**] possibilitou a emancipação de áudio generativo para jogos e aplicativos de síntese sonora em navegadores de rede tornando o navegador uma plataforma musical viável de diversas formas [**?**, **?**].

Esta API traz diversos *nós de áudio* padrões que podem ser concatenados em um grafo de DSP. Um exemplo desta concatenação pode ser vista na Figura **??** onde são utilizados três nós diferentes em um sintetizador, em sequência: uma instância de oscilador de ondas (OscilatorNode), uma instância de ganho (GainNode) e uma última instância que representa os alto-falantes (DestinationNode).

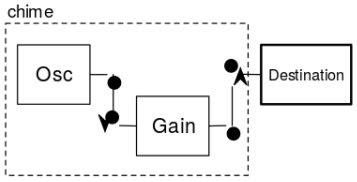

Figura 1: Estrutura de síntese da API webaudio. Fonte: [?].

Um sintetizador mais complexo pode ter diversos nós concatenados, entre eles um nó "especial" chamado *ScriptProcessorNode*, que permite o cálculo de cada *sample* durante o processamento de sua saída.

Baseado nesta API e principalmente neste nó, alguns *frameworks* são eles mesmos os instrumentos musicais; seja pelo encapsulamento de funcionalidades, seja pela maneira de pensar como o músico – profissional ou amador – codificará o processamento de sinais de áudio no navegador. Entre estes *frameworks*, destacam-se o trabalhos como *Gibber* e *Wavepot*.

Este trabalho apresenta uma outra ferramenta *open source*, desenvolvida a partir do estudo dos *softwares* acima. Chamamos de Termpot: *Term* por uma questão de similaridade com um terminal de computador, inspirado no GROOVE de Max Mathews e *Pot* por uma questão de similaridade técnica com o *Wavepot*. Nossa motivação foi desenvolver um *framework* como uma plataforma de luteria composicional[**?**] aberta e colaborativa, a partir de algumas reflexões de notação e execução musical.

A Seção **??** deste artigo apresenta os frameworks supracitados e uma breve comparação entre eles. A Seção **??** apresenta a ferramenta proposta. A Seção **??** traz os resultados desta pesquisa e a Seção **??** apresenta as conclusões do trabalho até o presente momento.

2. Trabalhos relacionados

Nesta seção apresentaremos a API webaudio e as ferramentas / frameworks Gibber e Wavepot.

WebAudio API

O primeiro trabalho relacionado é a própria API Webaudio e seu nó especial para processamento de sinais (*ScriptProcessorNode*), que permite os três trabalhos serem correlacionados. Uma característica lógica do ScriptProcessorNode é sua linearidade, possibilitando calcular cada amostra do vetor de saída de maneira direta. Isto permite *a)* uma possível redução no custo computacional para sínteses de ondas complexas; *b)* a definição matemática de um vetor de saída a partir da definição de uma função; e *c)* a criação de qualquer tipo de nó de processamento de áudio. No Código **??** demonstramos uma simples utilização desta API.

```
1  // Chrome ou firefox?
2  var context = new AudioContext();
3  var node = context.createScriptProcessor(1024, 0, 1);
4
5  // DSP
6  var noise = function(amplitude){
7      var sample = Math.random() * 2 - 1
8      return sample * amplitude
9  }
10
11 node.onaudioprocess = function(e) {
12   var output = e.outputBuffer.getChannelData(0);
13   for (var i = 0; i < data.length; ++i) {
14     output[i] = noise(0.5)
15   }
16 }
17 node.connect(context.destination); // TOCAR
18 node.disconnect(); // PARAR
```

Código 1: Exemplo de utilização do ScriptProcessorNode

Esta abordagem é conveniente do posto de vista musical entre as linhas e 6-9 do Código **??**. Para o músico, o restante pode ser especificidade técnica que pode desviar a atenção composicional. Nesse sentido, o *Gibber* e *Wavepot* oferecem soluções possíveis para aqueles que se interessam em manter um controle dos algoritmos musicais.

Gibber

O Gibber[1] é descrito por [**?**] como um ambiente de *live coding*[2]. Possui um *layout* de abas divididas entre: *1)* editor de texto; *2)* um *canvas* contendo imagens resultantes dos códigos visuais; *3)* um navegador do aplicativo, contendo rotinas diversas; *4)* alternador de códigos sonoros e códigos visuais *5)* navegador de códigos musicais do servidor; *6)* um console de informações; *7) login* para um círculo de contas restritas ao aplicativo. Nosso interesse maior tange ao item 1, e em como o código escrito processa os sinais digitais.Uma outra abordagem técnica, descrita na próxima sessão, elucida as linhas 136-175 do código-fonte da máquina de áudio do *Gibber*, *Gibberish*[3]. Descartaremos portanto o estudo do *Gibber* para focar no *Wavepot*.

Figura 2: Aplicativo *Gibber*. Fonte: `http://gibber.mat.ucsb.edu`

Wavepot

É uma *webaudio-daw*[4]. Esta ferramenta possui as seguintes características: *1)* layout que expõe a partitura-programação [**?**] do lado esquerdo. Geralmente é dividida em dois editores, semelhante ao método *instrumento-orquestra* [**?**, **?**]; *2)* do lado direito, controles de tocar, parar e gravar, menu de exemplos, módulos, músicas, ajuda, alternáveis; *3)* compartilhamento e atualizações dos códigos através do *Github*; *4)* um analisador gráfico da formas de onda resultantes; *5)* uma lista de cabeçalhos dos códigos abertos; *6)* um console de retorno de informações.

Existe um modo padronizado de programação que requer observações. A primeira é que o improvisador utilize uma variável t. Esta variável pode ser descrita como a fase de um tempo discretizado $td = 1/sampleRate$. É util para calcular formas de ondas diversas. No exemplo **??** demonstramos o uso do t para criar uma senóide. A segunda exige a utilização do *token* `export` antes de definir a função dsp.

---

[1]Disponível em `http://gibber.mat.ucsb.edu/`.

[2]Para mais informações sobre a prática *live coding*, sugerimos a leitura dos seguintes autores: [**?**, **?**, **?**, **?**, **?**, **?**]. Seu foco está em tornar a atividade de codificação produtiva do ponto de vista musical.

[3]Disponível em `https://github.com/charlieroberts/Gibberish`.

[4]*Webaudio Digital Audio Workstation* disponível em `https://github.com/wavepot/wavep0t/blob/master/index.js`.

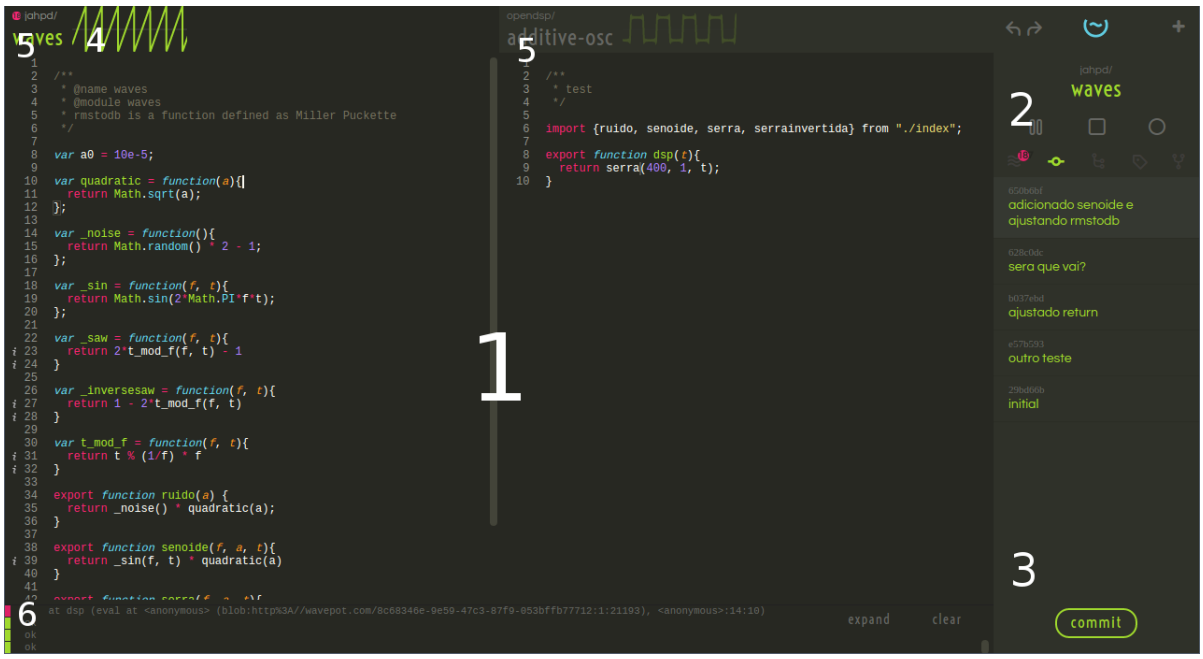

Figura 3: Aplicativo *Wavepot*. Fonte: `https://www.wavepot.com`

```
var noise = function(amplitude){
    return (Math.random() * 2 - 1) * a;
}

var sin = function(freq, amp, t){
    return Math.sin(Math.PI * 2 * freq * t) * a
}

export function dsp(t){
    return noise(1) * sin(1, 1, t)
}
```

Código 2: Sintetizando um ruído controlado por um LFO senoidal

2.1. Comparação de trabalhos relacionados

Para entender a relação entre estas duas ferramentas, montamos um quadro comparativo a partir de alguns parâmetros: tipo de música realizada nos códigos exemplares, suporte a vídeo, compilação em tempo de execução, estratégia de partitura-programação, biblioteca de síntese e suporte ao compartilhamento dos códigos. Esta comparação é apresentada na Tabela **??**. Em ambos os casos, ocorre a tradução de um código em linguagem *javascript* para o processamento de áudio. Uma discussão musicológica envolve entre outros fatores, um tipo de música tocado em eventos conhecidos como *algoraves* [**?**, **?**].

3. Desenvolvimento do Termpot

Semelhante ao *Wavepot* pelo método de síntese, mas diverso pela abordagem de codificação, o Termpot utiliza para enviar comandos uma mistura de comandos e linguagem *coffeescript*[**?**][11]. Com a extensão média de uma linha, é possível criar sonoridades utilizando as funções pré-definidas existentes no ambiente: ruído (branco), senóide, dente-de-serra, triangular, pulso, envelope e sequenciamento. Também é possível prototipar rapidamente novas funções para encapsular novas funcionalidade sonoras ao ambiente. Além da atividade de codificação, elaboramos uma maneira para a criar e utilizar GUIs que se assemelham a *sliders* de uma mesa de som.

[11] `http://coffeescript.org/`, acessado em 4 de novembro de 2015.

Tabela 1: Tabela comparativa de funcionalidades entre o *Gibber* e o *Wavepot*.

Característica	**Wavepot**	**Gibber**
Linguagem de programação	Javascript	Javascript
Tipo de música	Algorave	Algorave + glitch/noise
Vídeo	Não	Sim - Síntese de imagens 2D e 3D através de uma implementação OpenGL. Podem sincronizar com alguns parâmetros sonoros.
Compilação em tempo de execução	Sim – qualquer edição no código é automaticamente reconhecido enquanto estiver sendo processado algum sinal de áudio.	Sim – o executante musical deve utilizar teclas de atalho para requerir ao servidor o processo de compilação.
Partitura-programação	Editor de texto desenvolvido com a biblioteca *Ace*[56] em duas partes, subdividido pelos autores deste artigo como *instrumentais* e *orquestrais*	Editor de texto, com alternância dos códigos sonoros (*javascript*) e códigos visuais (*OpenGL*)
Biblioteca de síntese	$Audio-process$[7]	Gibberish[8]
Criação de novos módulos em tempo de execução	Sim – o improvisador pode criar funções através de variáveis ou as palavras-chave $exportdefaultfunction$	Não
Compartilhamento de códigos	O usuário pode criar, ou reutilizar módulos e músicas, através da rede social $Github$	Códigos musicais podem ser publicados no servidor do Gibber abrindo uma conta no próprio site.
Elementos de GUI	Não existe na versão oficial, no entanto um experimento foi desenvolvido para sincronizar a máquina de síntese *Wavepot* com controladores MIDI[9]. A GUI servia, portanto, para averiguar o funcionamento do MIDI.	Sim – Pode ser utilizado com a biblioteca *Interface.js*[10]

Uma característica singular do *Wavepot* original, é a possibilidade de separação da programação-partitura em dois arquivos, muito semelhante ao método *Instrumento-Orquestra* descrito por Max Mathews e utilizado no CSound [**?**, **?**]. Isso é possível adicionando um marcador $@module$ aos comentários iniciais de um código. Desta forma, serão reconhecidos dois arquvivos durante a performance de improvisação: $index.js$ e $test.js$. O primeiro permite elaborar os instrumentos, enquanto o segundo realiza o DSP (teste).

Já no Termpot, buscamos utilizar outro método de codificação focado em uma abordagem mais performática.

Arriscamos a comentar uma inspiração no GROOVE de [**?**, **?**], quando este propõe

a criação de novas funções em tempo de execução. Ao mesmo tempo em que utilizamos a biblioteca *Ptty.js* dando ao *Termpot* as características de um emulador de terminal no que tange a utilização de comandos em tempo de execução, como em um terminal, integrado com controles manuais. Por esta razão, acreditamos que esta ferramenta é inspirada no conceito de compor, memorizar, editar e controlar funções do tempo, algoritmicamente e manualmente [**?**].

Metodologia de desenvolvimento

Para a implementação, três tarefas foram necessárias:

1) Customizar um emulador terminal

Utilizamos a biblioteca *Ptty.js* é uma biblioteca documentada e pode ser facilmente implementada seguindo instruções de seu arquivo *README*. É baseada em jQuery e permitiu uma rápida prototipação.

2) Implementar um ambiente de síntese sonora integrado ao emulador

Uma das facilidades do *Ptty* é definir ambientes; elaboramos um ambiente que controla a *Web Audio API* nos moldes descritos na Seção 2.

3) Comandos diversos

Ajuda, inspeção de funções, definição de novas funções, tocar, parar, pausar, gravar e download e criação de controles gráficos. Para gravação, utilizamos o *recorderWorker.js*[12] de Matt Diamond.

4. Resultados

Ao iniciar a página, o improvisador é colocado diante de emulador de terminal. Para obter ajuda de uso, o improvisador pode digitar $help$. O comando *Wavepot* pode ser executado com um argumento, que é o tamanho do *buffer*. Valores válidos devem ser potência de dois (2), entre 256 e 16384, sendo que o padrão é 1024 (ver Código **??**). Alguns comando gerais como *play*, *stop*, *pause* e *reset* são utilizados para execução. *Record* e *export* para gravação de uma performance e exportação desta performance para um arquivo de extensão *.wav*.

O comando *play* inicia uma incrementação da variável t; o comando $pause$ "congela"t e coloca o sintetizador em estado suspenso; $reset$ redefine $t = 0$, porém não congela a incrementação. $Stop$ realiza a mesma tarefa que $pause$, porém desliga o sintetizador.

```
1  ......................................
2  . Virtual machine started at           .
3  . Thu Sep 03 2015 13:32:15 GMT-0300 (BRT).
4  . type help for instructions           .
5  ......................................
6  $\$$ wavepot 1024
7  ..............................
8  . sintetizador de sample a sample.
9  . amostragem: 44100            .
10 . canais: 2                    .
11 . buffer: 1024                 .
12 ..............................
13 wavepot > play
14 wavepot > sin 440, 0.5
15 true
```

Código 3: Console do *wavepot* aguardando dados de entrada do improvisador.

[12] https://github.com/mattdiamond/Recorderjs/blob/master/recorderWorker.js.

Realizando sínteses

A síntese no ambiente *wavepot* é realizada fornecendo uma função que retorne dados entre -1.0 e 1.0, atuando diretamente sobre a linha 8 do exemplo apresentado na Seção **??**. No exemplo apresentado no Código **??**, ao utilizar a expressão `Math.random() * 2 - 1`, a cada intervalo de tempo $td = 1/SR$ (taxa de amostragem), um novo número randômico é mapeado entre o intervalo permitido, resultando em um ruído branco.

Uma maneira de simplificar o cálculo de um ruído, é definí-lo em uma função, como no Código **??**.

Já existe no aplicativo uma função `noise(a)`. Porém, como o objetivo é demonstrar seu uso, exemplificamos o processo ainda no Código **??**. Ao utilizar o *token* `def`, definimos em tempo de execução uma função chamada `ruidobranco(a)`.

```
1  wavepot > Math.random() * 2 - 1
2  true
3  wavepot> 0
4  true
5  wavepot > def rb(a) (Math.random() * 2 - 1) * amplitude #Ruido branco [a-amplitude = {0..1.0} ]
6  true
7  wavepot > inspect rb
8  /*
9   Ruido branco [a-amplitude = {0..1.0} ]
10  */
11 var rb;
12
13 rb = function(a) {
14   return a * (Math.random() * 2 - 1);
15 };
16 wavepot > rb(0.71)
17 true
18 wavepot > mute()
19 true
20 wavepot> rb 0.71
21 true
22 wavepot > stop
23 true
```

Figura 4: Exemplo de código para produzir e definir um ruído branco.

Uma característica emprestada do *wavepot* é o encorajamento ao uso de comentários como forma de documentar as funções criadas pelo usuário. No $Termpot$ são usados no final da definição da função, seguido do caracter #, sem espaço inicial.

Para silenciar o sistema, três formas de codificação são possíveis. Uma vez que o console aceita valores numéricos, ao digitar 0, ouvimos um silêncio. Existe também a função `mute()` pré-definida. A terceira desliga o sistema. As três maneiras estão demonstradas no Código **??**

Utilização da variável t

A utilização de uma variável que controla o tempo foi discutida brevemente no final da Seção **??**. No caso do $Termpot$ recorremos a uma supressão desta variável, como demonstrado no Código **??**, afim de atingir uma economia de caracteres.

A implementação acima foi possível como demonstrado na linha 48, e entre as linhas 105 e 138 no repositório git[13].

[13]Disponível em `https://github.com/jahpd/termpot/blob/master/wavepot-runtime.js`.

```
wavepot > inspect tau
/* Ciclo Trigonometrico completo */
var tau = 2*Math.PI;
true
wavepot > Math.sin tau * 440 * t
true
wavepot > 0
true
wavepot > def senoide(f, a) (Math.sin tau * f * t) * a
senoide defined
wavepot > senoide 415, 0.5
true
```

Código 4: Exemplo de código do Wavepot sem o controle explicíto do tempo (t)

Implementação estereofônica

Até o momento, os recursos utilizados apenas retornam um áudio monofônico expandido para dois canais. Todavia, há uma maneira simples de separar os canais, partindo da criação de uma função que aceite dois valores de entrada e retorne um Array de dois elementos, como demonstrado no Código **??**.

```
wavepot > def stereo(l, r) [l, r] #Retorna um audio stereo de
quaisquer inputs
wavepot > inspect stereo
/*
Retorna um audio stereo de quaisquer inputs
 */
var stereo;

stereo = function(l, r) {
  return [l, r];
};
wavepot > stereo sin(440,0.5), sin(330, 0.5)
true
```

Código 5: Exemplo de código estereofônico do Wavepot

4.1. Implementação de controles

Conforme apresentado na Figura **??**, a GUI do Termpot possui do lado direito um painel; que pode ser utilizado como uma mesa de *sliders*. Desta forma, as interfaces gráficas controlam parâmetros de funções usadas. Abaixo descrevemos o procedimento para criar um novo controle, ele mesmo uma função:

```
wavepot > slider("left", 0, 1024)
true
wavepot > slider("right", 0, 1024)
true
wavepot > stereo sin(440,0.5)*left(), sin(330, 0.5)*right()
true
```

Código 6: Exemplo de código do Wavepot

5. Conclusão

Este trabalho tratou de investigar a *Web Audio API* e alguns trabalhos derivados desta API partindo de uma abordagem técnica. A investigação desta API e suas possibilidades musicais e artísticas levaram ao desenvolvimento de uma ferramenta chamada Termpot.

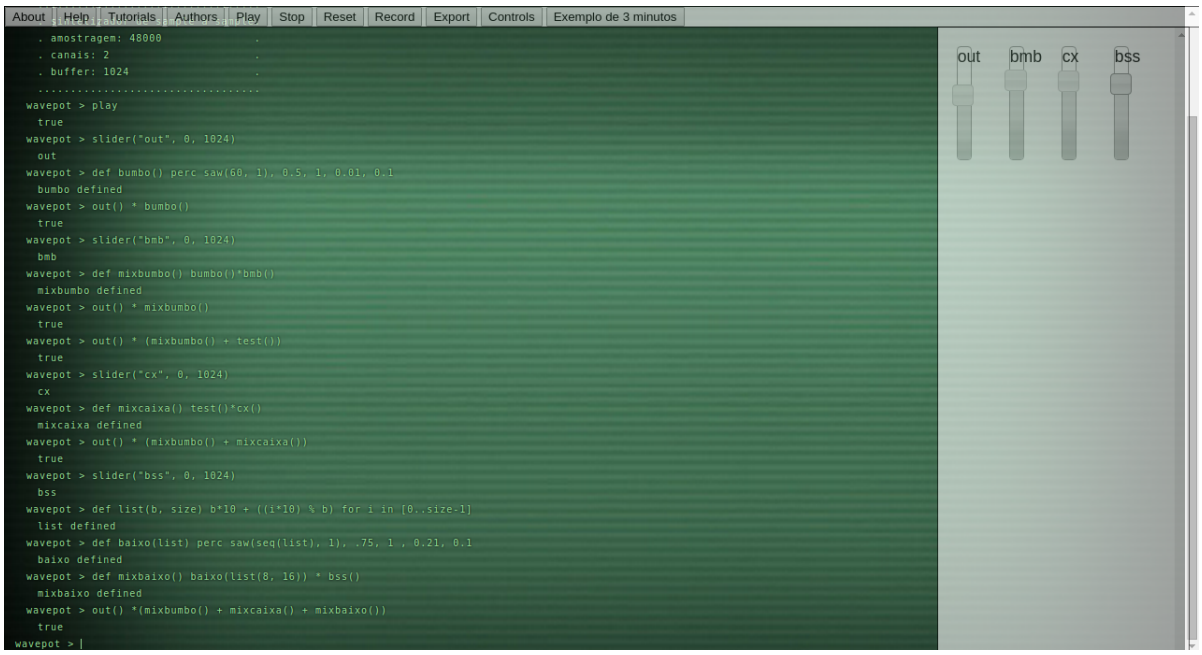

Figura 5: Aplicativo *Termpot*. Fonte: autores.

No processo de desenvolvimento do Termpot levamos em conta um processo de síntese semelhante ao *Wavepot* e realizamos algumas customizações. Características presentes nos ambientes Gibber e Wavepot que julgamos importantes foram adicionadas a nova ferramenta. Mesmo rudimentar se comparada a estas ferramentas, a ferramenta desenvolvida apresentou-se bastante versátil.

Entretanto, apontamos os seguintes problemas técnico : a biblioteca jQuery apontada como uma provável fonte de *xruns*[14].

Além disso é discutido na comunidade *WebAudio API* a substituição do

A criação musical baseada em funções matemáticas, como é proposto na abordagem do Termpot, permite ao músico explorar as técnicas tradicionais de síntese como síntese aditiva, subtrativa, FM, AM mas permite também explorar outras construções musicais que diferem destas técnicas. Neste sentido, a ferramenta desenvolvida apresenta uma abordagem técnico-composicional diferenciada de alguns métodos e ferramentas existentes por poder estabelecer-se no limiar entre a matemática e a música.

5.1. Trabalhos Futuros

i) criar um servidor; *ii)* otimizar o emulador, talvez substituindo o Ptty ou propondo melhorias; *iii)* suporte para amostras pré-gravadas.

[15].

5.2. Agradecimentos

Os autores agradecem ao Guilherme Rafael Soares por ter apresentado as biblioteca Ptty.js, aos desenvolvedores dos projetos investigados por disponibilizarem seus códigos e a FAPEMIG por subsidiar a pesquisa.

[14]*Google Chrome* 44.0.2403.89 Ubuntu 14.04. Embora o conceito seja original do ALSA (Linux), o fenômeno é semelhante do ponto de vista perceptivo. Sua definição técnica é: "Um 'xrun' pode ser um estouro de buffer ou de uma saturação de um buffer. Um aplicativo de áudio ou não foi rápido o suficiente para transmitir dados (...) ou não rápido o suficiente para processar os dados"[**?**]. Tradução nossa de *An "xrun"can be either a buffer underrun or a buffer overrun. In both cases an audio app was either not fast enough to deliver data (...) or not fast enough to process data (...).*

[15]Disponível em `https://jahpd.github.io/termpot`.