Pure Data e as Linguagens Visuais

Cristiano Medeiros Dalbem
Computação e Música - INF01062 Tópicos Especiais em Computação XVII
Instituto de Informática
Universidade Federal do Rio Grande do Sul
cmdalbem@inf.ufrgs.br

29 de novembro de 2010

Sumário

1 Introdução

1.1 Por quê Pure Data?

Comecei a "brincar" com a linguagem Pure Data, ou Pd, pois parecia ser uma que me possibilitaria pra botar em prática os conhecimentos desenvolvidos na disciplina de Computação e Música. O que eu sabia é que era um ambiente de tratamento e síntese de som bastante flexível, graças a abordagem baixo-nível com os dados de áudio.

Acabei descobrindo que, além de isso tudo ser a mais pura verdade, Pd ainda traz diversos outros benefícios quase tão importantes quanto o poder de expressão que ele proporciona. Seja pela portabilidade, seja pela comunidade engajada e criativa, seja pelo relacionamento com outras linguagens e softwares, Pure Data vale muito a pena, e é mais ou menos isso que quero dizer neste artigo.

Tratarei de aspectos técnicos da linguagem, analisando as características (mais) objetivas (possíveis). Mas falarei também dos usos de Pd, das cenas no qual ele mais se destaca, das diversas possiblidades de integração com as mais variadas interfaces e frameworks. E além de tudo isso ainda deixarei a imparcialidade de lado para fazer críticas - sempre de maneira construtiva, é claro -, assim como elogios espalhados por esse texto.

1.2 Linguagem visual? Como assim?

Pure Data (ou Pd) é "um ambiente gráfico de programação em tempo real para áudio, vídeo e processamento gráfico" [1]. Mais do que isso, é uma linguagem de programação interpretada multiplataforma voltada à convergência de interfaces (MIDI, por exemplo e principalmente).

Pd é inspirada e baseada na Max, da empresa Cycling '74, uma outra linguagem visual de propósitos semelhantes mas proprietária, diferentemente de Pd que é Free Software [1]. O Pure Data foi criado e continua sendo mantido por Miller Puckette, que aliás ajudou no desenvolvimento do Max no passado. Para informações sobre outras linguagens semelhantes veja o Apêndice A.

Por ser uma linguagem de código aberto, desenvolvido por uma comunidade de desenvolvedores de todo mundo, ela vem ao longo do tempo acumulando um grande número de extensões, chamados *externals*. Esse grande número de extensões culminou no lançamento do Pd-Extended, uma edição da linguagem que já vem com uma grande quantidade de módulos adicionais, além de um ambiente melhorado.

[1] Free/Libre Software, ou Software Livre, significa software que pode ser usado, estudado e modificado sem restrições, além de poder ser copiado e redistribuído.

2 Pure Data

2.1 Fundamentos

Pure Data não é uma linguagem de programação imperativa nem funcional: o paradigma é o dataflow, uma filosofia que lembra muito o funcionamento de um circuito - não é à toa a semelhança visual. Os dados circulam felos fios, que são as linhas que ligam as caixas. Caixas são componentes independentes do código, como se fossem Objetos da programação O.O., e tem entradas e saídas (*inlets* e *outlets*, respectivamente). Pure Data é uma linguagem de programação visual que funciona sob o paradigma de Dataflow, o qual lembra muito um circuito, onde sinais são passados por um "caminho" - o qual chamamos de *patch*- e sendo interpretados, processados e repassados pelos componentes que o compõe.

Diferentemente de um circuito eletrônico, em Pure Data pode circular mais de um tipo de sinal/dados. Estes circulam pelo *patch* sendo processados e repassados pelas caixas que encontram no caminho.

Nem toda caixa é necessariamente um objeto, e nem tudo num patch é necessariamente uma caixa. Em Pd lidamos com 4 tipos de entidades: [7]

Objetos são os elementos principais do patching, e são caixas especiais cujo texto nelas (uma string) é parseada em atoms (um conceito similar ao de token). O primeiro atom representará a "classe" do objeto, podendo ser desde operações aritméticas ou lógicas até referências a externals ou subpatches. Os outros atoms depois do primeiro serão argumentos de criação da classe determinada pelo primeiro.

Mensagens são caixas que guardam strings e cujo comportamento é de, quando ativado, repassar uma mensagem com esse conteúdo. Uma mensagem é um dos tipos de sinais que navegam pelo patch. Um tipo especial de Mensagem é o Bang, que é uma mensagem de string nula e que serve para ativar entidades. Uma caixa de mensagem pode ser ativada tanto com outra mensagem quanto com um clique.

Elementos de GUI (*Graphical User Interface* são partes essenciais da experiência Pd, pois aqui que entra a parte mais customizável da linguagem. GUI pode ser tanto um number box, que é uma simples caixa cujo conteúdo é o dado que esta está recebendo - excelente para debugging sem precisar estar imprimindo coisas no terminal - até controladores de arrays e matrizes cujos dados o usuário pode entrar utilizando o mouse. Os elementos de GUI mais comuns são sliders dis-

cretos e contínuos, bangs (de fato, uma mensagem que é graficamente representada em formato de botão), toggles, etc.

Comentários dispensam comentários.

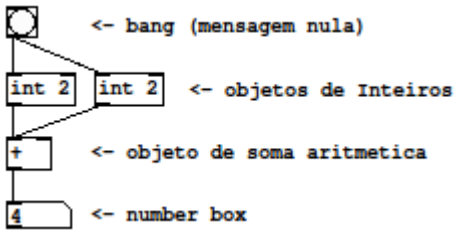


Figura 1: Amostra do look&feel de um código em Pd, demonstrando o uso de entidades básicas da linguagem.

Sobre o fluxo em um patch, resta dizer que há um tipo de dado especial que não é só uma mensagem de comando trocada por objetosos quais são manipulados pelos chamados *control objects*. Há ainda os *tilde objects*, que são os que trabalham com sinais de som, os quais são identificados por um ~no final do nome. Esta é a única limitação de "tipos" em Pd, pois não é permitido ter um objeto que esperavam uma entrada de som receber dados, ou vice-versa. Haverá objetos especiais para realizar a interface entre os dois domínios, como no caso do objeto osc~do exempplo da figura 2. Há uma diferenciação visual na conexão entre componentes, ou seja, na linha que os liga: ela é fina para conexões onde trafegam dados normais e grossa para onde trafegam sinais de som.

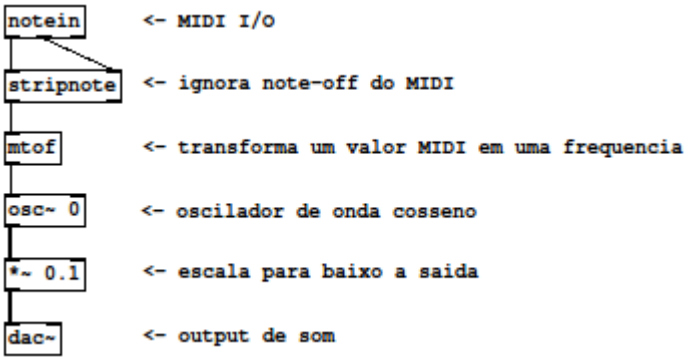


Figura 2: Exemplo simplíssimo de sintetizador MIDI. Desmontra como é a separação entre fluxo de sinais e de dados em um programa.

Em questão de subprogramação, Pure Data nos oferece a habilidade de fazer subpatches. Subpacthes são declarações de novos objetos, os quais são editados em uma nova janela, separada da do programa principal, e onde definimos inlets e outlets, que são, respectivamente, as entradas e saídas desse novo objeto. O resultado de um subpatch no programa que o instancia pode ser tanto uma caixa de objeto normal, onde aparece o nome do objeto, quanto uma representação arbitrária utilizando o recurso de *graph-on-parent* (figura 3). Existem dois tipos de subpatches:

One-off subpatches são objetos específicos do programa actual, e são definições salvas no arquivo deste.

Abstractions são os objetos que são salvos em arquivos separados e podem ser referenciados por qualquer programa com o path devidamente configurado.

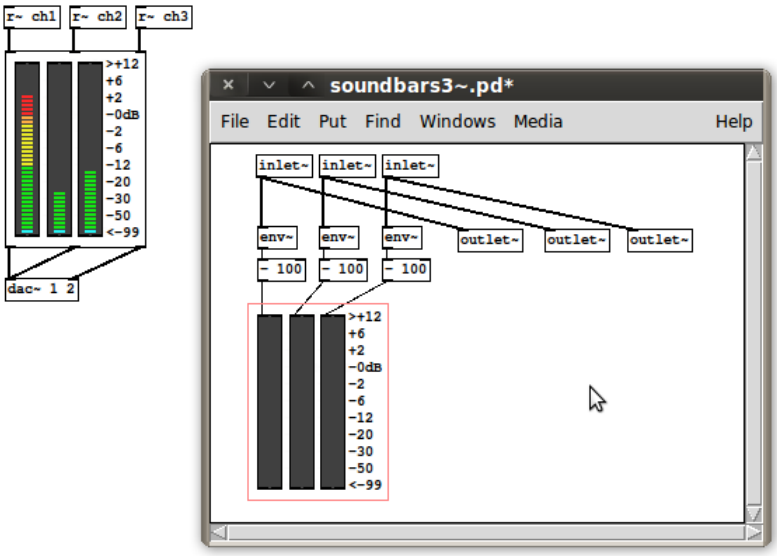


Figura 3: Exemplo de uso do recurso *graph-on-parent*, que nos permite definir uma representação gráfica para um subpatch, escondendo a implementação.

2.2 Análise de características

Utilizarei este espaço para estender a descrição dos fundamentos da linguagem, pontuando algumas características críticas no contexto de Pd. Estas análises são baseadas em um trabalho da cadeira de Modelos de Linguagens de Programação da UFRGS.

2.2.1 Simplicidade

Pure Data é uma linguagem sintaticamente simples. O conceito de inlets e outlets cria uma abstração muito intuitiva para o uso dos objetos: eles são representações visuais do que seriam entradas e saídas de uma função, com o adicional se só haver 2 tipos para entrada e saída (dados e sinais).

Já semanticamente a coisa muda de figura. Por conviver principalmente com o domínio "baixo nível" da computação musical, o usuário pode ficar bastante frustrado com seus programas se não souber exatamente o que está fazendo. Ainda assim, foi feito um excelente trabalho na documentação dos objetos: todos objetos tem uma documentação própria, que já vem com o ambiente de programação. Esta documentação mistura textos que explicam o funcionamento e *patches*. Na realidade, cada arquivo de documentação já é um patch, cujo texto é escrito em forma de comentário. Ou seja, pode-se executar copiar e modificar os exemplos como se fosse parte do teu programa (figura 4).

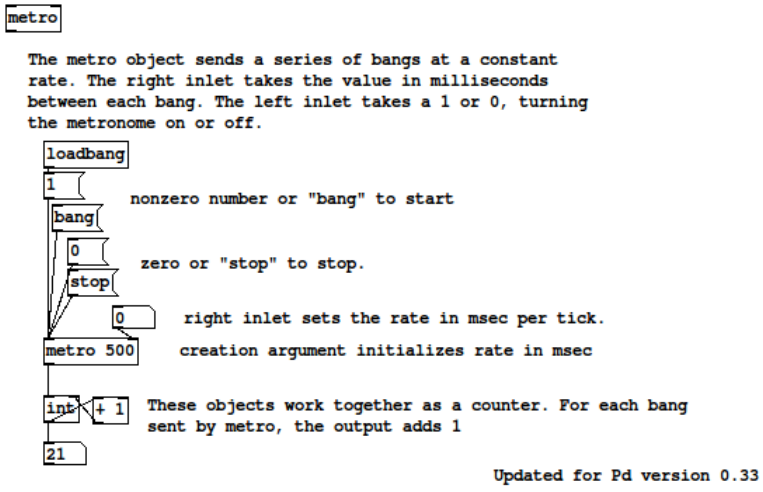


Figura 4: Exemplo de documentação do objeto de metrônomo. Isto é um patch totalmente editável, que nos permite inclusive copiar exemplos de aplicação do objeto para o nosso programa.

2.2.2 Portabilidade

Este é dos carros-chefe da linguagem: Pd roda em GNU/Linux, Mac OS X, iOS, Android, Windows, e há portes até para FreeBSD e IRIX. Isso se deve principalmente ao fato de ser uma linguagem de código aberto, permitindo

que qualquer um que tiver interesse portá-la para um sistema operacional diferente possa fazer essa implementação.

2.2.3 Ortogonalidade

Pd é extremamente ortogonal! Alguns dos fatores que contribuem para isto é a quase inexistência de tipos em Pd, e a possibilidade de controlarmos argumentos de criação de objetos com objetos externos (utilizando os outlets específicos para isto). Com algum domínio na manipulação de listas e vetores as possibilidades se tornam imensas, como no exemplo da figura 10, onde utilizamos vetores para armazenar acordes inteiros que alimentarão objetos de saída MIDI.

2.2.4 Expressividade

Esta é a característica mais interessante de ser comentada sobre Pure Data! É porque a expressividade dela está intimamente ligada com sua filosofia.

Pd naturalmente é pouquíssimo expressiva. A linguagem é altamente baixo nível, mexendo com os conceitos mais básicos da computação musical. Além disso os objetos tem nomes estranhos, e nem sempre a maneira como eles estão ligados e distribuídos na tela representam corretamente a semântica do programa, já que suas partes podem, e muito comumente serão, altamente paralelizáveis.

Apesar de tudo isso, o que costuma ocorrer é que os programas em Pure Data ficam altamente expressivos! E isso se dá ao fato de a linguagem ter sido feita justamente para que o programador desenvolve as abstrações que achar que mais convém para os dados que estão sendo processados. Ou seja, as representações, sejam numéricas quanto gráficas, dos dados com que o programa trabalha, são altamente personalizadas para o contexto de sua execução. E isso vale também para os controles e paineis da interface gráfica, que graças às diversas externs disponíveis e ao recurso de *graph-on-parent*, tomam o formato que o programador desejar.

2.3 Funcionalidades

Pure Data foi feito para dar a maior liberdade possível ao programador. Não é possível criar executáveis dos programas, mas quanto à execução deste temos uma certa liberdade. O ambiente onde um programa em Pd é rodado é interativo, mas a linguagem permite que sejam feitos programas completamentes automatizados, assim como também montar ou desenvolver do zero

interfaces gráficas.

2.4 Problemas, defeitos, limitações e críticas

O ambiente visual de programação do Pd é muito desconfortável. Além de ter apenas um estado de UNDO (o famoso CTRL+z), ele tem sérios problemas na parte visual. Ele é fixo na resolução do computador onde está rodando, sendo que bordas e linhas tem o tamanho de 1 pixel cada e isso não muda. Ou seja, ou o usuário precisa trocar a resolução do computador pra próximo de 1024 por 768, o que é um insulto a qualquer um que tenha algo melhor que um monitor CRT, ou é obrigado a diminuir a sensibilidade do mouse, caso contrário será uma tortura acertar os cliques. Além disso, como o grid é do tamanho de um pixel, para termos um programa com um visual organizado e agradável aos olhos temos que ajustar manualmente, pixel a pixel, a posição de cada elemento pra que as linhas entre eles fiquem retas, sem serrilhados. Isso poderia ser facilmente melhorado tendo uma opção de gridsize, com passos mais discretos, ou ainda com o uso de *magnetic points*, recurso extremamente comum em softwares hoje em dia. MAX, por exemplo, tem um botão que alinha automaticamente as caixas selecionadas - como isso faz falta...

Além da constante luta contra os serrilhados, outro problema do ambiente que dificulta a elaboração de programas agradáveis aos olhos é que não há transparênicas, sombras ou qualquer outros efeitos que dêem maior noção de profundidade e diferenciabilidade entre os objetos. Também não há recursos de desenho livre de conexões, por exemplo, que poderia ser um adicional bacana pra criar código mais legíveis.

Acredito que isso tudo se deve ao fato de ter sido feita uma escolha por uma implementação mais simples da IDE, facilitando o trabalho daqueles que fossem portar para diferentes sistemas. Mas é fato é que ninguém se preocupou em criar uma IDE personalizada que nos desse todas essas possibilidades. O resultado disso é que um programa mais complexo gera uma tela bastante poluída e homogênea, sem hierarquias que ressaltem trechos de código mais ou menos importantes - com exceção dos recursos como bangs e sliders, que podem ter seu tamanho e cor modificados.

Outra coisa que me incomoda são alguns nomes de objetos, que são extremamente pouco intuitivos. Por exemplo, um objeto cujo comportamento é semelhante ao de um IF se chama "spigot". Outro, que funciona semelhantemente a um IF, mas com teste de grandeza entre as entradas, se chama "moses". Outro exemplo ainda é do objeto slice, que é um bom nome, pois serve pra cortar um vetor (extrair um subvetor); agora, a sua versão inversa, que corta o vetor ao contrário, se chama "ecils"(!!!).

3 Usos

Basicamente o que é feito com Pd são sintetizadores. A linguagem nos permitirá que deixemos na tela apenas os controles gráficos dos módulos do sintetizador que nos interessam, escondendo por baixo o que é o "código" que gera e manipula o som. Assim como também é possível criar mapeamentos de teclas do teclado para ações (objeto [key]) e outros dispositivos de entrada, como joysticks (objeto [hid]). Estes são recursos que auxiliam na utilização de Pd em performances ao vivo, como em shows ou sessões de Live Coding[2].

Um exemplo que demonstra a habilidade do Pure Data de trabalhar com as mais diversas interfaces é o de seu uso como pedal de guitarra elétrica. Este é exemplo de Pierre Massat, da banda de rock francesa Pierre et le Loup[3]:

> Yes, I've been using Pd live for a few months now, and it works like a charm. My guitar is constantly pluged into my soundcard and routed through Pd via JACK, using Planet CCRMA's rt kernel. I use a hacked gamepad as a foot controller, with a small dozen of switches plus an expression pedal. I also use the second input in my soundcard for vocal effects (in Pd too). [4]

De fato, Pd pode - e foi feito para - ser usado na manipulação em tempo real de qualquer tipo de sinal, de maneira aberta e livre. Essa foi a filosofia por trás da criação da linguagem, e por isso seu nome ("dados puros" em tradução livre), que

3.1 Extensões e integrações

Todos os exemplos nesta seção funcionarão apenas no Pd-Extended, pois utilizam componentes adicionais ao Pd original.

3.1.1 hid

Com o uso do objeto [hid][10] temos acesso fácil a dispositivos físicos como teclados, mouses, joysticks ou qualquer tipo de controlador USB como touchscreens, controles remotos, etc. Na figura 3.1.1 criei um patch que recebe os valores dos analógicos e dos 4 botões frontais de um joystick padrão.

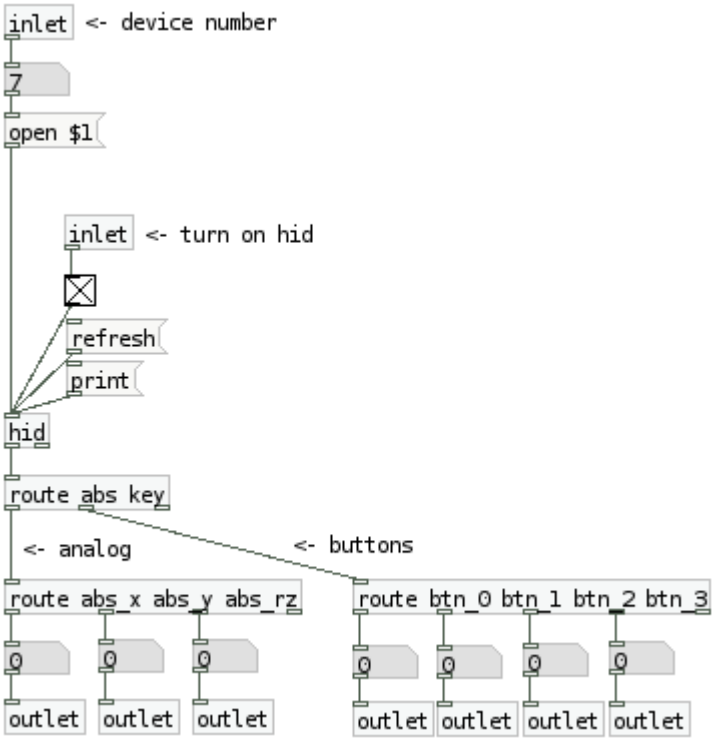


Figura 5: Exemplo de uso do objeto [hid] para receber comandos de um joystick padrão. Baseado em [9]

3.1.2 GEM

A Graphics Environment for Multimedia (GEM) é um portal para uma nova dimensão em Pd. O que ela nos permite, basicamente, é multiplicar as possibilidades de nossos patches e sintetizadores e geradores em geral para plataformas multimídia completas.

O GEM não permite apenas manipular imagens e vídeos em tempo real, mas também gerar e manipular geometrias 3D com efeitos de iluminação.

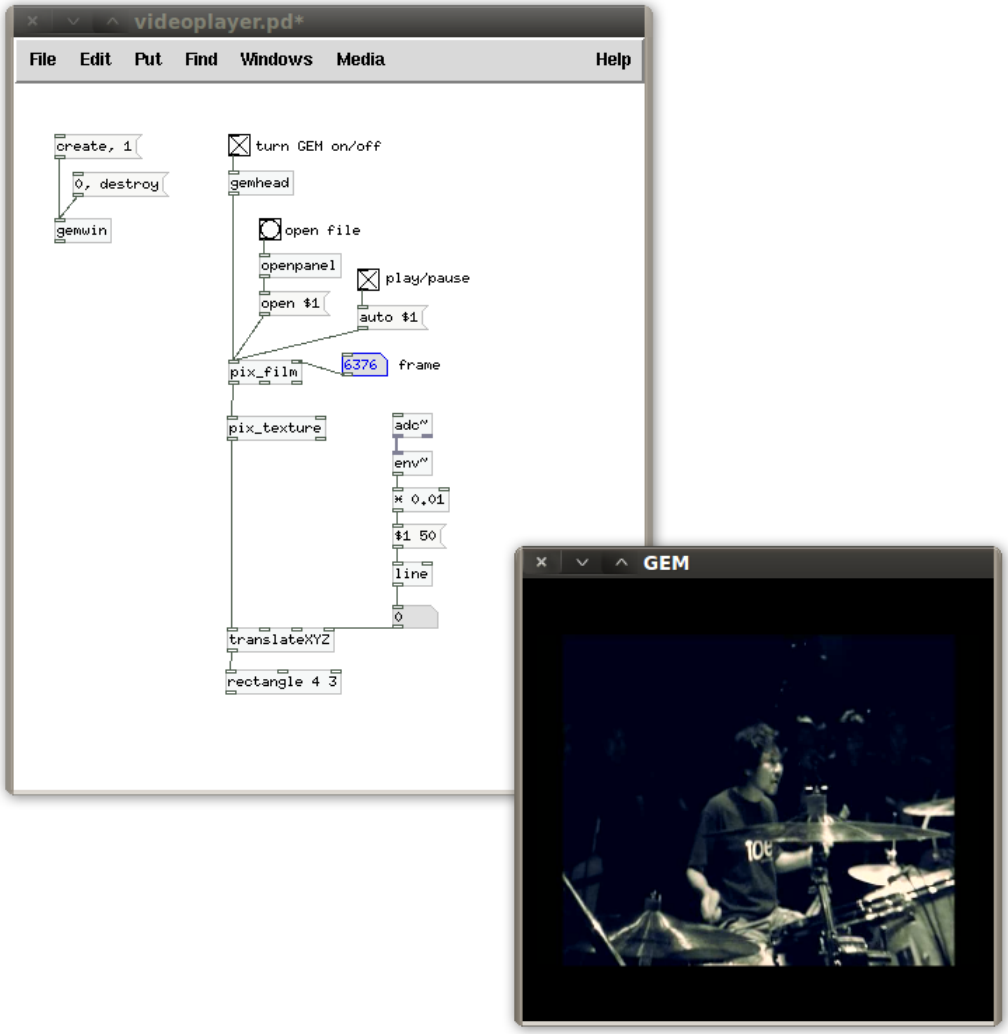
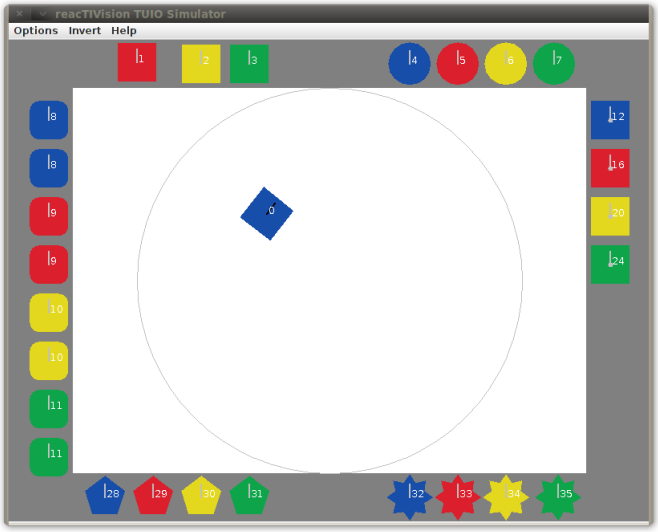


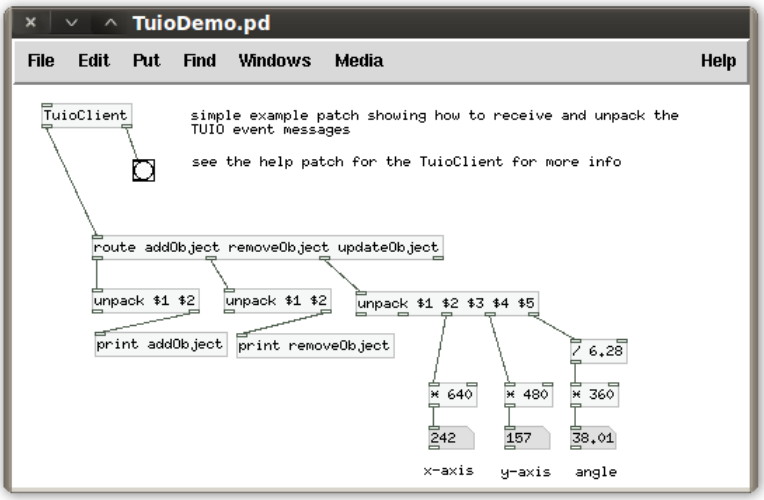
Figura 6: Patch que reproduz um vídeo, manipulando o zoom segundo a amplitude do espectro de som capturado por um microfone. Baseado em [9]

3.1.3 reacTIVision

A reacTIVision é uma biblioteca de Visão Computacional, open-source e multi-plataforma, que torna muito fácil criar interfaces físicas no estilo Realidade Aumentada. Tudo que fazemos com ela lembra o projeto da reacTable, e isso é porque esta é uma biblioteca criada justamente o projeto.



(a) TUIO Simulator



(b) Patch decodificador

Figura 7: Exemplo de patch que decodifica *TUIO messages*, mostrando a posição e o ângulo de um objeto da tela. No exemplo foi utilizado o TUIO Simulator, que serve para simular mensagens da biblioteca, sem precisar do uso de uma webcam durante os testes.

3.2 Projetos que utilizam Pure Data

Segue uma breve descrição de alguns projetos interessantes, de variadas magnitudes, que utilizam Pure Data.

Engranaje (Sprocket) , por Maíra Sala, é uma interface para edição de sequencias de videos baseada na experiencia multi-touch, em uma visualização por meio de um projetor em uma tela. Utiliza Pure Data e GEM para a manipulação dos videos e reacTIVision para tratamento dos toques. A interface foi feita especialmente para o filme *Ressaca* de Bruno Vianna. Mais em [11].

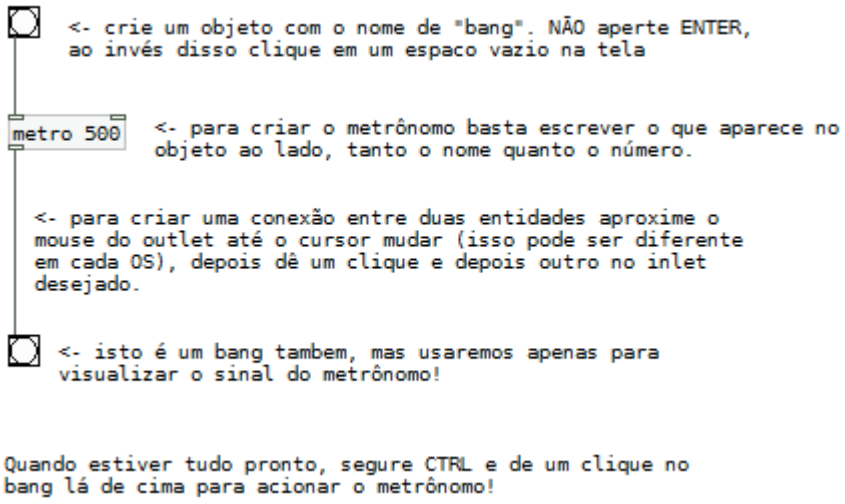
Spore , da EA Games é um jogo do tipo *god game* bastante popular, lançado em 2008 pra diversas plataformas, e cujo sistema de som teve grande parte feita em Pure Data [12].

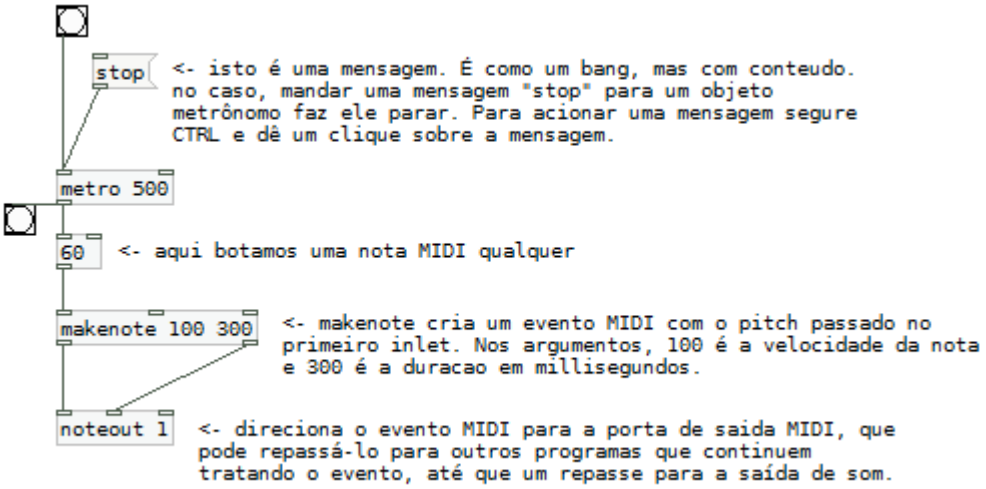
Automatenklavier é um projeto muito interessante que pretende, analisando o espectro de uma pessoa que lê um texto, transformá-lo em sequencias de notas que são tocadas em piano, de maneira que aquele conjunto de notas dá a impressão de que o piano está falando aquelas palavras. Algumas são completamente reconhecíveis, enquanto outras nem tanto. A análise do espectro foi feita em Pure Data, e o resto do projeto com outras ferramentas. Mais em [15].

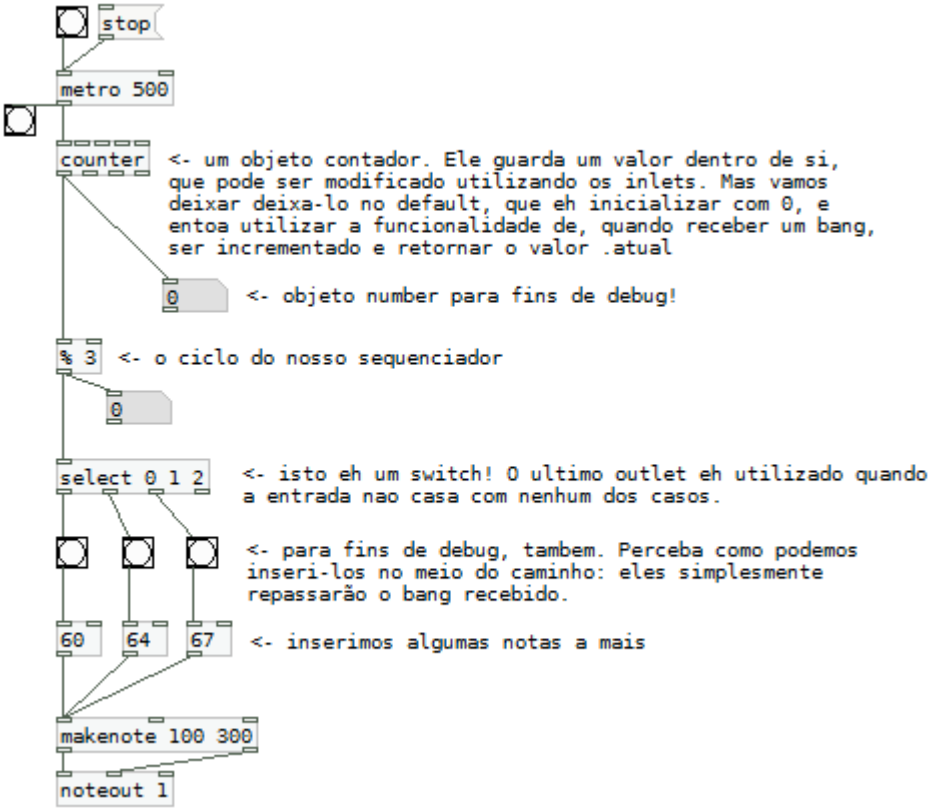
bang-tris é um simples jogo de Tetris que roda no ambiente de programação do Pd. Nada mais do que uma ideia curiosa. Mais em [13].

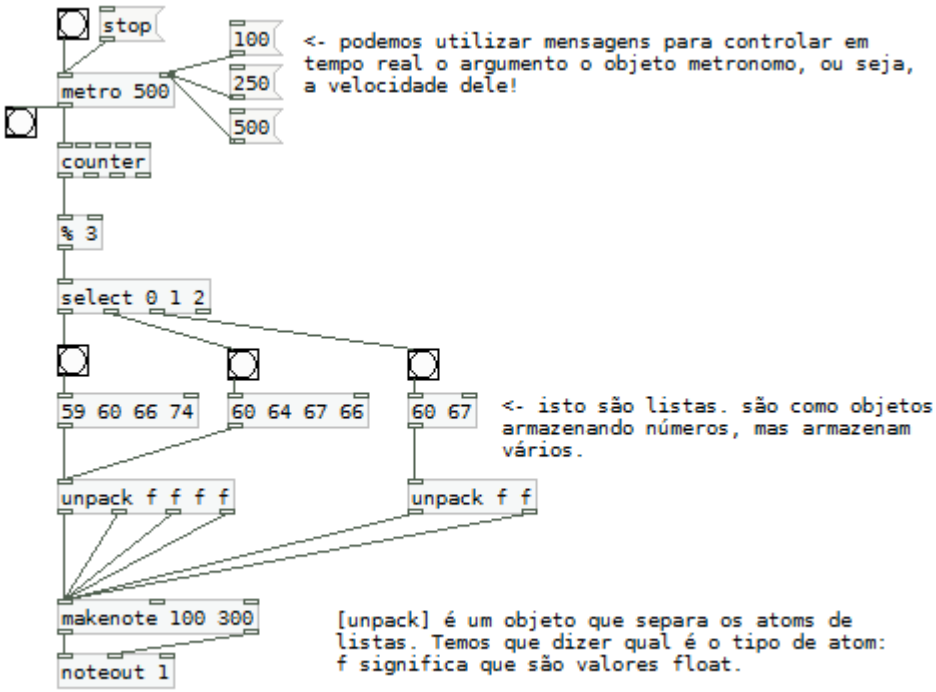
4 Minitutorial

Neste minitutorial faremos um sequenciador MIDI bastante simples, tentando explorar conceitos recorrentes na programação em Pd.




stop
<- isto é uma mensagem. É como um bang, mas com conteudo.
no caso, mandar uma mensagem "stop" para um objeto
metrônomo faz ele parar. Para acionar uma mensagem segure
CTRL e dê um clique sobre a mensagem.
metro 500
60
<- aqui botamos uma nota MIDI qualquer
makenote 100 300
<- makenote cria um evento MIDI com o pitch passado no
primeiro inlet. Nos argumentos, 100 é a velocidade da nota
e 300 é a duracao em millisegundos.
noteout 1
<- direciona o evento MIDI para a porta de saida MIDI, que
pode repassá-lo para outros programas que continuem
tratando o evento, até que um repasse para a saída de som.


stop
metro 500
counter
<- um objeto contador. Ele guarda um valor dentro de si,
que pode ser modificado utilizando os inlets. Mas vamos
deixar deixa-lo no default, que eh inicializar com 0, e
entoa utilizar a funcionalidade de, quando receber um bang,
ser incrementado e retornar o valor .atual
0
<- objeto number para fins de debug!
% 3
<- o ciclo do nosso sequenciador
0
select 0 1 2
<- isto eh um switch! O ultimo outlet eh utilizado quando
a entrada nao casa com nenhum dos casos.
<- para fins de debug, tambem. Perceba como podemos
inseri-los no meio do caminho: eles simplesmente
repassarão o bang recebido.
60
64
67
<- inserimos algumas notas a mais
makenote 100 300
noteout 1



5 Considerações finais

Programar em Pure Data é uma experiência instigante. Mas digo isso de um ponto de vista de um estudante de Ciência da Computação, que se interessa por linguagens de programação e paradigmas diferentes que te façam pensar diferente. Mas é uma experiência para poucos: se aventurar por esse mundo sem ter uma base musical pode ser muito frustrante. Ou talvez monótona, já que não terás muito pra onde ir, mesmo. Além disso, é muito importante que, para que se faça algo realmente interessante com Pd, é necessário ter conhecimento de uma suíte de programas que vão ser integrados com o ambiente, desde processadores MIDI até interfaces físicas para composição musical, alguns deles introduzidos neste documento.

Meu processo de aprendizado da linguagem foi bastante rápido, no sentido de eficiência: rapidamente consegui estar desenvolvendo aplicações que além de atender às minhas necessidades ainda me instigaram a tentar coisas novas e cada vez mais complexas. Mas isso tudo se deve muito pela parte da comunidade ativa que há pela Internet, o que resulta em um grande número de tutoriais, referências e discussões sobre a linguagem. Isso ajuda a aumentar o número de pessoas leigas interessadas na linguagem e, consequentemente, um número maior de tutoriais para iniciantes.

Pessoalmente tenho acho divertida a programação em Pd, principalmente porque é realmente muito fácil processar som com ela. Mas quando me deparo com um problema de lógica ou de matemática que seja um pouco mais complicado eu me atrapalho bastante. Isso se dá em razão do paradigma de dataflow, que faz tu te sentires como se estivesse programando um circuito, o que não é uma posição confortável para um estudante de Ciência da Computação acostumado com o paradigma Imperativo, o mais comum entre as linguagens de programação mais populares.

Um uso interessantíssimo de Pd o qual ainda não tive a oportunidade de vivenciar é nas sessões de Live Coding, ao lado de outras linguagens visuais, as quais certamente estarei estudando, ou pelo menos dando uma olhada mais atenta no futuro. A existência e uso dessas linguagens não só factualiza um anseio por novos meios de composição musical nos dias do hoje, mas também na revolução das dinâmicas de programação como um todo. Hoje em dia programação não é mais como no filme Matrix, onde a tela mostra código decifráveis apenas pelos mais hábeis e solitários mestres dessa arte. E nem uma parede de texto verborrágica como nos tempos do COBOL. Programar hoje em dia é muito mais agradável, mais eficiente, mais acessível, e por que não dizer, até mais coletivo?

6 Referências

[1] http://puredata.info *Pure Data — PD Community Site.*

[2] Collins, N., McLean, A., Rohrhuber, J. & Ward, A. (2003), "Live Coding Techniques for Laptop Performance", *Organised Sound* 8(3):321–30.

[3] http://www.mail-archive.com/pd-list@iem.at/msg39040.html *Re: [PD] Re : Music made with Pd*

[4] http://pierreetleloup.bandcamp.com *Pierre et le loup*

[5] Danks, M. 1997. Real-time image and video processing in gem. In *Proceedings of the International Computer Music Conference*, pp. 220–223, Ann Arbor. International Computer Music Association.

[6] http://gem.iem.at *Gem - PD Community Site*

[7] http://www.crca.ucsd.edu/~msp/Pd_documentation/index.htm *HTML documentation for Pd*

[8] Martin Kaltenbrunner, Sergi Jordá, Günter Geiger, and Marcos Alonso. The reactable*: A collaborative musical instrument. *In Proceedings of the Workshop on "Tangible Interaction in Collaborative En- vironments" (TICE), at the 15th International IEEE Workshops on Enabling Technologies*, 2006.

[9] http://www.youtube.com/user/cheetomoskeeto#g/c/12DC9A161D8DC5DC *YouTube - cheetomoskeeto's Channel - Pure Data Tutorials Playlist*

[10] http://at.or.at/hans/pd/hid.html *HID for Pd*

[11] http://geral.etc.br/engranaje *Engranaje*

[12] http://pc.gamespy.com/pc/spore/853810p1.html *GameSpy: The Beat Goes on: Dynamic Music in Spore*

[13] http://parasitaere-kapazitaeten.net *Marius Schebella's site*

[14] http://kickowang.blogspot.com *Kicko Wang - Daze Graffiti*

[15] http://ablinger.mur.at/voices_and_piano.html *Peter Ablinger, Voices and Piano*

[16] http://cycling74.com/products/maxmspjitter

[17] http://www.pawfal.org/al-jazari

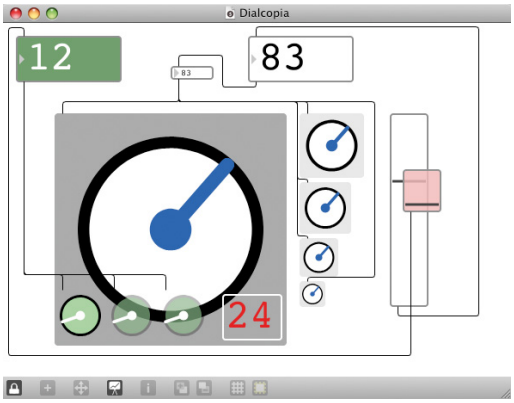
[18] http://vvvv.org

[19] http://ffnnkk.org

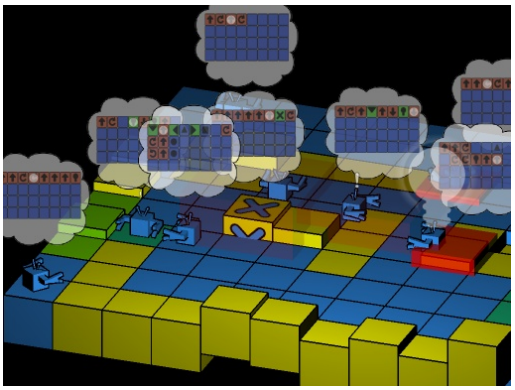
[20] http://www.native-instruments.com/#/en/products/producer/reaktor-5

[21] http://www.pawfal.org/fluxus

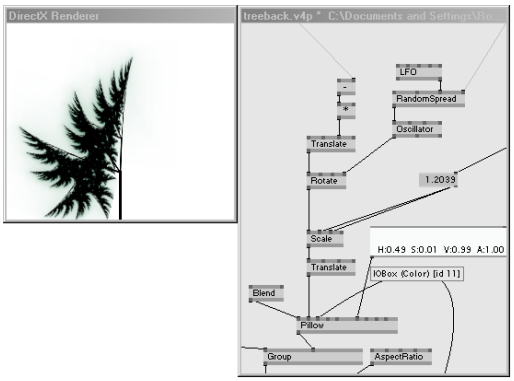
7 Apêndice A - Linguagens visuais marcantes



(a) Max [16]

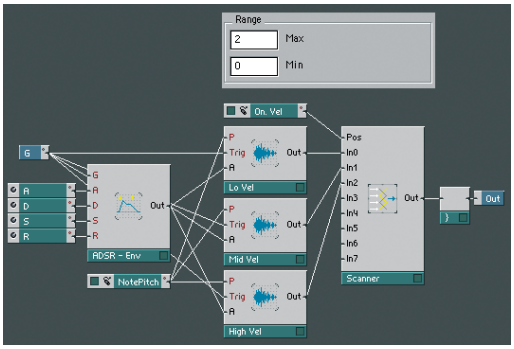


(b) Al Jazari [17]

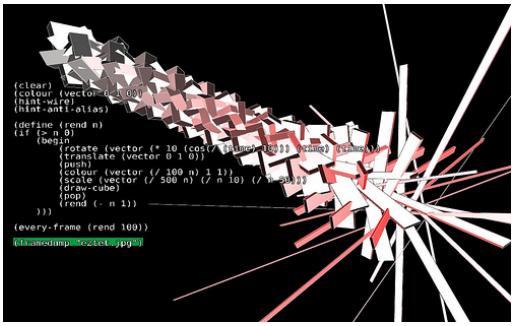


(c) vvvv [18]

(d) fnk [19]



(e) Reaktor [20]



(f) Fluxus [21]

8 Apêndice B - Exemplos de patches em Pd

8.1 Exemplo 1

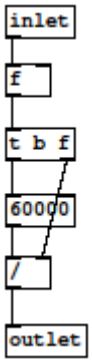


Figura 8: Exemplo de abstraction que recebe um valor de BPM (*Beats Per Minute*) e o converte em um valor em milisegundos. Útil para utilizar com metrônomos.

8.2 Exemplo 2

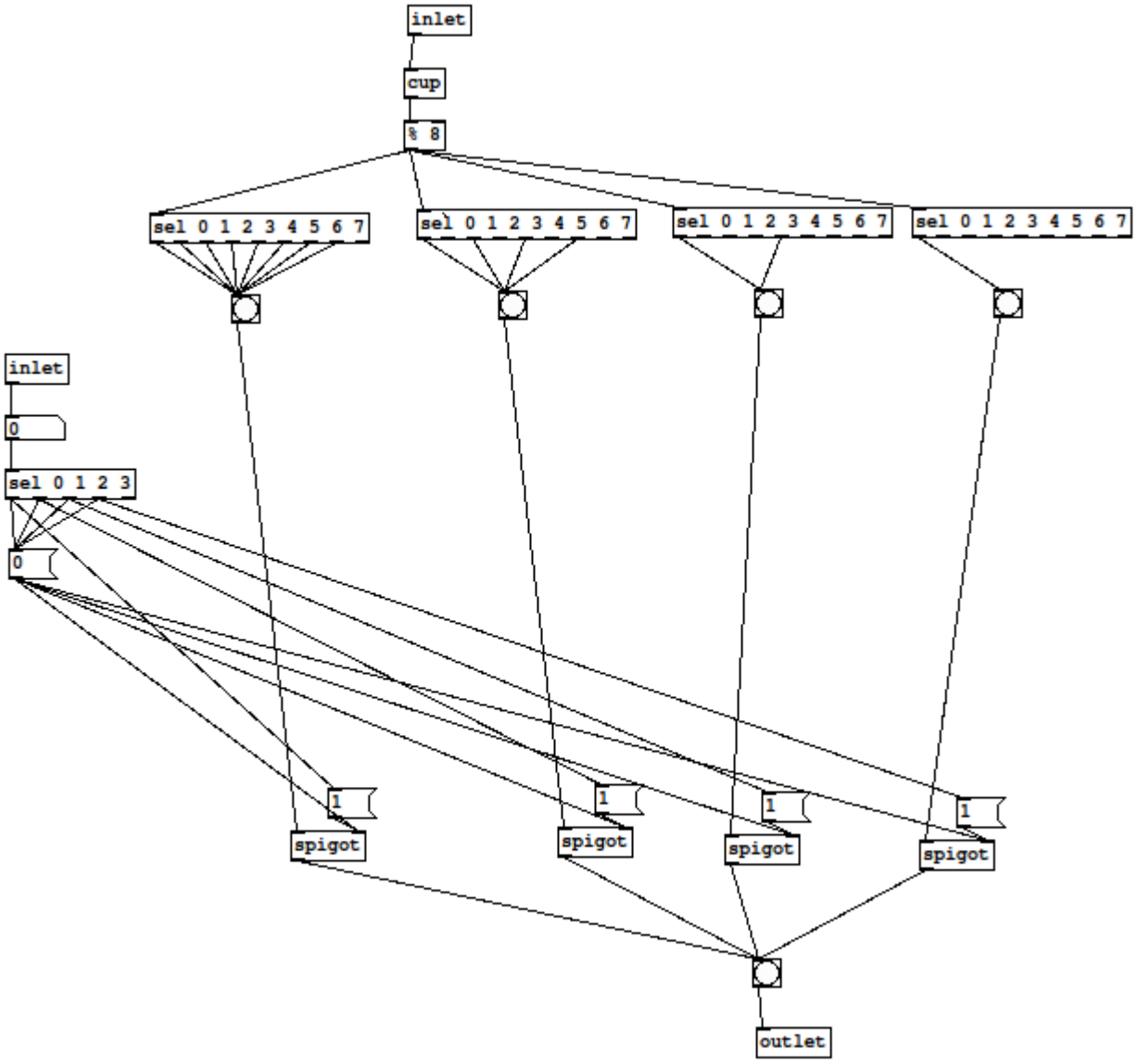


Figura 9: Exemplo de uma rhythm machine. Recebe um bang, que é a batida da música, e um inteiro que selecionará um dos 4 ritmos. No caso, ele seleciona entre semínima, colcheia, semicolcheia ou fusa.

8.3 Exemplo 3

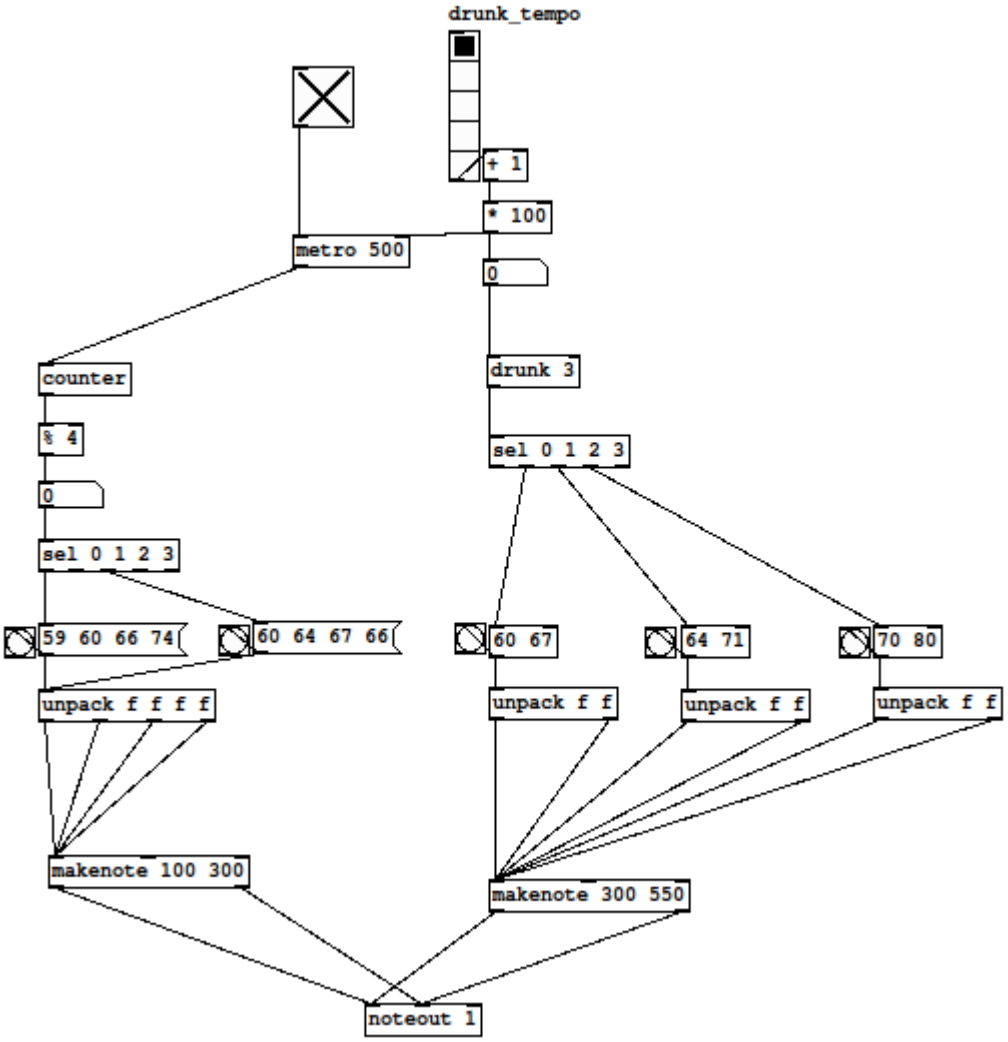


Figura 10: Um pequeno sequenciador que cria uma música de duas faixas. De um lado há um padrão composto por dois acordes (o "acompanhamento"), e de outro uma melodia randomizada pelo objeto [drunk]. O objeto drunk gera números aleatórios próximos um dos outros, gerando algo como se fosse uma coerência na melodia. No topo há uma pequena interface para controlar a velocidade do metrônomo.

8.4 Exemplo 4

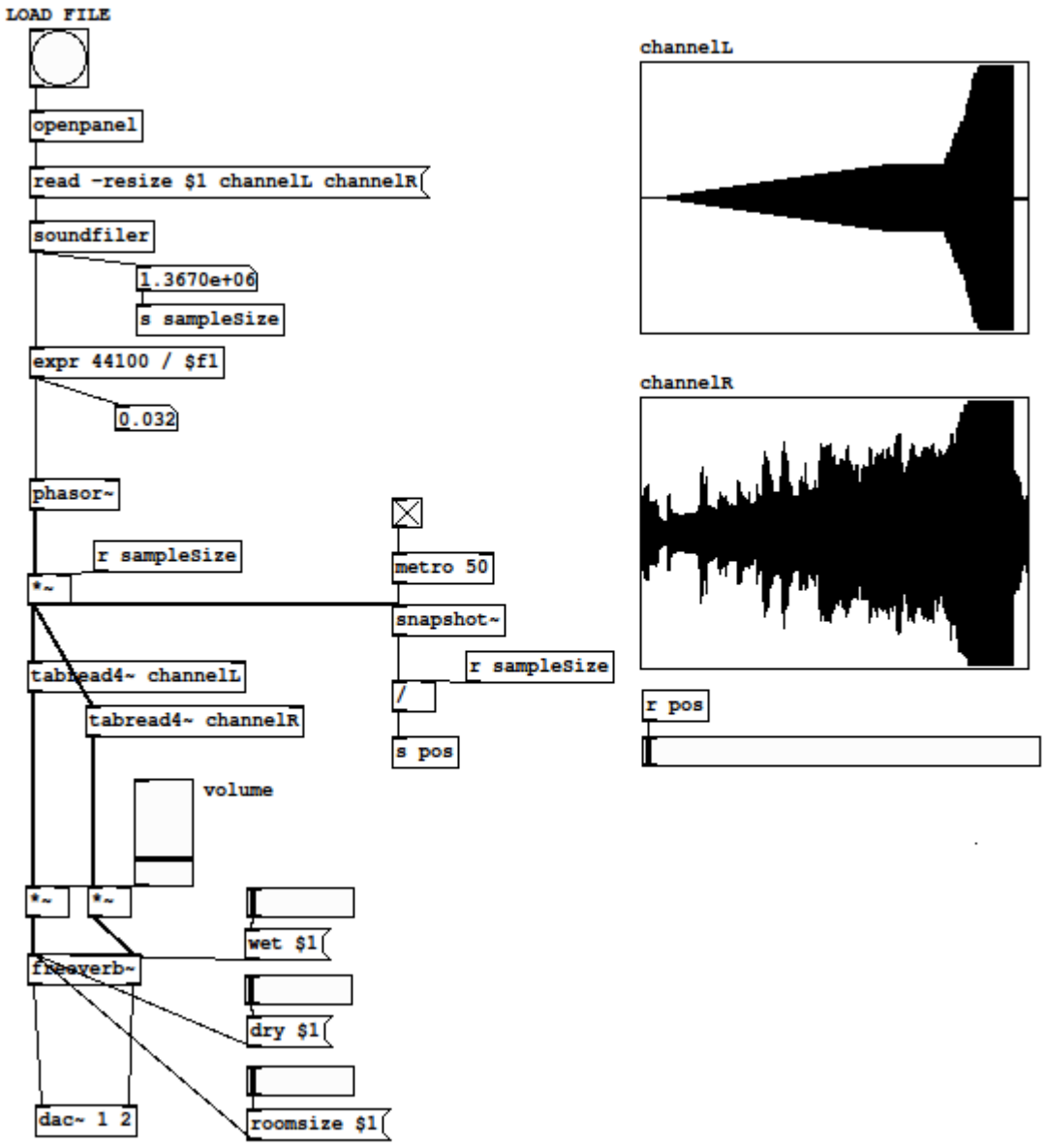


Figura 11: Um exemplo de programa que abre, filtra e executa um arquivo de som WAV. Os dois arrays na direita são alimentados com o espectro dos canais do arquivo de som, e o slider abaixo deles mostra a posição da execução. Há um controle de volume e 3 controles dos parâmetros do filtro de Reverb (objeto [freeverb~]).

8.5 Exemplo 5

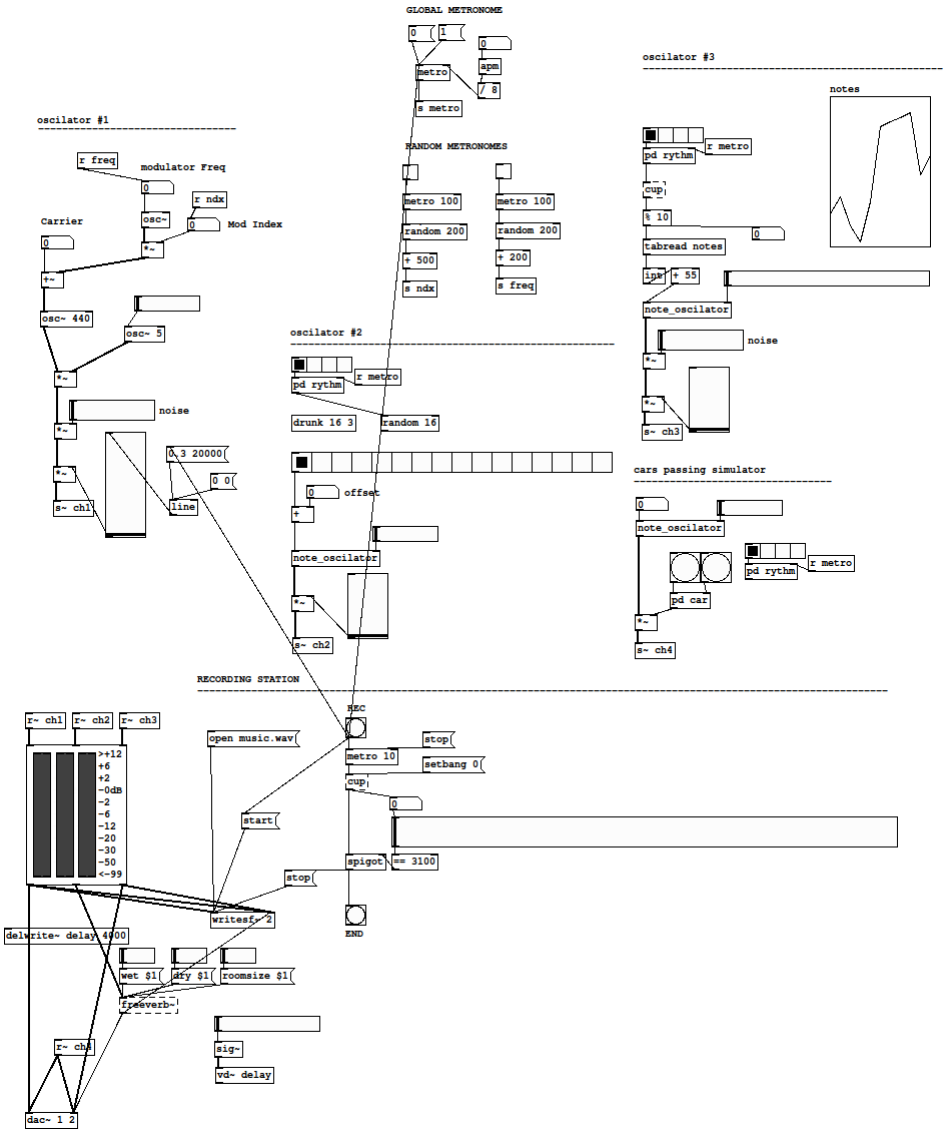


Figura 12: Uma orquestra formada por um sintetizador FM de parâmetros aleatórios e 2 osciladores controlados por notas MIDI, sendo um cuja nota é selecionada com técnica probabilística e outro que toca uma sequência de 10 notas geradas obtidas do array no canto superior esquerdo, cujos valores foram gerados por cliques de mouse. A parte de baixo contém o módulo de saída de som, onde estão alguns filtros aplicados em alguns canais, e a parte de gravação, o qual está feita para gravar 30 segundos. O botão de REC aciona o metrônomo, assim como um controle que faz o primeiro oscilador aumentar o volume gradualmente durante alguns segundos.

8.6 Exemplo 6

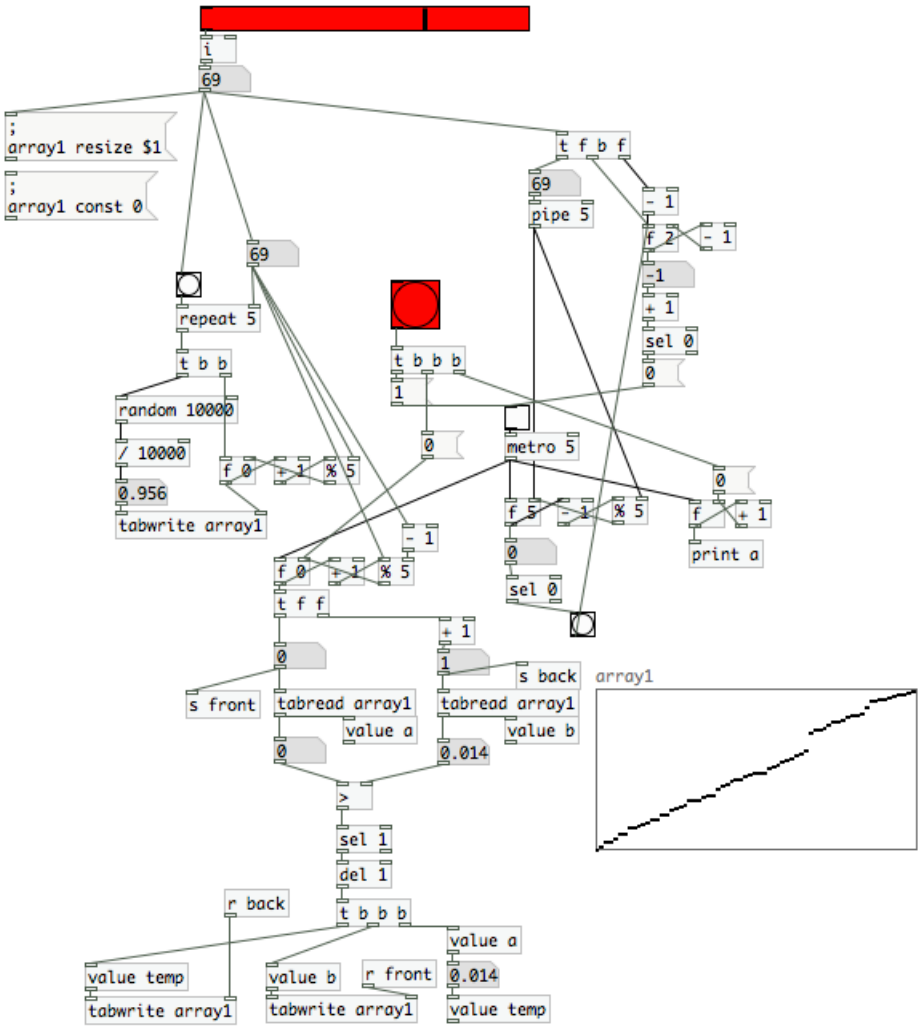


Figura 13: [14] Implementação em Pd de um algoritmos de ordenação (que deveria ser) bastante simples, o Bubble Sort. Tire suas próprias conclusões.