tidalcycles

tutorial pt-br

V.0.21

TidalCycles

tutorial pt-br

V.0.21

tradução
Giuliano Obici

Rio de Janeiro, 2021.

Conteúdo

Apresentação

TidalCycles, ou simplesmente *Tidal*, é um ambiente de codificação ao vivo (*live coding*) concebido para improvisação musical e composição algorítmica. Incorpora uma linguagem de programação puramente funcional com inferência de tipo e avaliação preguiçosa.[1]

Criado por Alex McLean, a estrutura de *Tidal* foi concebida inicialmente para música percussiva e polirítmica. Atualmente está pautada em representação reativa flexível e funcional de padrões, operada em tempo racional. *Tidal* pode ser utilizado em diferentes estilos musicais, no entanto, por sua abordagem cíclica de tempo, o torna extremamente versátil para criar estilos musicais repetitivos voltados à música de pista.[2]

TidalCycles oferece resultados rápidos e versáteis, mesmo àqueles que não possuem qualquer conhecimento em programação. Por conta de sua versatilidade intuitiva, *Tidal* tem se popularizado no meio de produção e performance de música eletrônica e aproximando a atividade de músicos, compositores e DJ com a prática do *live coding*.[3] Além de apresentar um novo modo de se produzir música, o *live coding* tem criado tendências, com as festas que reúnem música algorítmica e dança chamadas de Algorave (junção das palavras algoritmo e rave).[2]

[1] Haskell é uma linguagem desenvolvida para ser adequada ao ensino, pesquisa e aplicação industrial, pioneira em várias características avançadas de linguagem de programação a que é utilizada por TidalCycles.

[2] O termo Algorave foi cunhado por Alex McLean, criador do TidalCycle, e Nick Collins.[1] [4]

Sobre Tradução

Este tutorial é a tradução de uma pequena parte da documentação de *TidalCycles* disponível online. Mais especificamente, ele cobre a parte introdutória (*Geting Started* - Como Iniciar). Ou seja, este Tutorial em português-br apresenta os fundamentos do *TidalCycles* como instalar, atualizar e iniciar pela primeira vez (partes 1, 2 e 3) bem como o Tutorial em si (parte 4).[3]

TidalCycle oferece uma forma intuitiva e versátil, ágil e simples de produzir música ao vivo a partir do código. Conhecida como codificação ao vivo (*live coding*), esta prática tem se difundido cada vez mais entre músicos e entusiastas, artistas e *experts* interessados em música e arte eletrônica. Mesmo quem nunca programou, conseguirá, a partir deste tutorial, criar rapidamente padrões sonoros complexos digitando pequenas sequências de código, como se estivesse escrevendo em um editor de texto.

A primeira vez que tive contato com *CiclosDaMaré* (tradução literal de *TidalCycles*), anos atrás, fiquei impressionado e, ao mesmo tempo, espantado com sua versatilidade intuitiva de produzir música. Desde então, tenho me perguntado. O que acontecerá quando autodidatas e músicos descobrirem *CiclosDaMaré*? Conseguirão fazer o código sambar, o ritmo com algoritmo rimar e a pista de dança desfilar na avenida?

Embalado pelos ciclos algorítmicos da *Maré*, enquanto testava os exemplos e traduzia este documento, sonoridades, afetos e ideias iam e vinham. Como o próprio movimento do mar que avança e recua, lembrei do refrão. "A onda do mar leva. A onda do mar traz". Fiquei então me perguntando. O que *Maré* leva? O que *Maré* traz?

Passado alguns anos, desde a primeira imersão nas operações lógicas do algorítmico, percebo que *Maré* me trouxe mais questões do que certezas. Ao mesmo tempo, *Maré* me levou para perspectivas instigantes que estão quase por emergir. Que esta tradução possa levar e trazer ciclos inventivos à arte digital em especial à música popular, eletrônica e experimental brasileira.

Rio de Janeiro, 2 de fevereiro 2021.

Giuliano Obici

[3] Esta é uma tradução que não foi revisada, por isso, é muito provável que existam erros ou incongruências terminológicas que precisarão ser aprimorados. Caso você tenha encontrado algum erro no documento, favor entre em contato pelo *email [a] giulianobici.com*. Toda contribuição será mais do que bem vinda. Vale lembrar também que esta é a versão V.0.21. Ela foi atualizada no dia 28 de Fevereiro de 2021. Um nova versão poderá estar disponível neste link.

Como Iniciar

Capítulo 1

Instalar do Zero

TidalCycles é multi plataforma e pode ser instalado em computadores com sistema operacional Linux, MacOS ou Windows. Ele utiliza um editor integrado ao Haskell que irá executar o processamento de áudio do SuperCollider. Isso significa que durante o processo de instalação serão instalados Atom Editor[1], Haskell e SuperCollider.

1.1 Linux

1.1.1 Pré-requisitos necessários

Há algumas coisas para instalar como parte de um sistema completo de Tidal:

- Git
- Haskell
- SuperCollider
- Atom Editor[2]

Esperamos que sua distribuição Linux torne os pré-requisitos facilmente disponíveis para você através de um gerenciador de pacotes. Por exemplo, se você estiver usando

[1] A documentação aqui recomenda o editor Atom para digitação do código Tidal. Se você deseja usar um editor diferente, dê uma olhada na lista de editores que suportam Tidal.

[2] Se você não gosta do editor Atom por algum motivo, por favor verifique a lista de editores que suportam Tidal.

uma versão recente do Ubuntu ou similar, você pode instalar o Haskell com o seguinte comando em uma janela do terminal:

```
sudo apt-get install build-essential cabal-install git jackd2
```

1.1.2 Instalar SuperCollider

A instalação do SuperCollider através deste método [sudo apt-get install supercollider sc3-plugins] geralmente não funciona. Ou a versão do SuperCollider é muito antiga (o SuperDirt precisa pelo menos da versão 3.7), ou a versão do SuperCollider é incompatível com os sc3-plugins. Se você estiver usando ubuntu, mint ou uma distribuição similar, meu conselho é ignorar os pacotes de supercollider e simplesmente compilá-los você mesmo. As instruções para compilação a partir da fonte em várias distribuições estão disponíveis aqui. Se você estiver usando uma distribuição recente baseada no Debian (por exemplo, Ubuntu >= 18.04), estes *scripts* facilitam a tarefa: https://github.com/lvm/build-supercollider

Basta colar as quatro instruções da linha de comando abaixo no terminal:

```
$ git clone https://github.com/lvm/build-supercollider
$ cd build-supercollider
$ sh build-supercollider.sh
$ sh build-sc3-plugins.sh
```

Pré-requisitos opcionais

(Esta seção pode ser ignorada se na seção anterior 'Install Supercollider' você decidiu executar os scripts em https://github.com/lvm/build-supercollider uma vez que esses *scripts* já contêm a instalação dos sc3-plugins através de '$ sh build-sc3-plugins.sh')

O seguinte é opcional, mas recomendado:

- **SC3 Plugins** - você pode precisar dos plugins SC3 do SuperCollider se quiser usar qualquer um dos sintetizadores incluídos no SuperDirt. A maioria dos exemplos na documentação ainda funcionará, portanto você pode pular esta etapa e retornar a ela mais tarde.

1.1.3 Instalar TidalCycles

Abra o Terminal. Se você não tem certeza de como abrir uma janela de terminal no Linux, ela varia de acordo com a distribuição, mas geralmente encontra "Terminal"nos menus. Então digite e execute estes dois comandos (ignorando quaisquer reclamações que a cabal tenha sobre o "estilo de uso legado v1"):

```
cabal update
cabal install tidal --lib
```

Se você nunca instalou TidalCycles antes, então a etapa de instalação da cabal de Tidal pode levar algum tempo. Ao fim do processo a mensagem que aparecerá no Terminal dever ser: Installed tidal-x.x.x.x (onde x.x.x é o último número de versão) sem nenhum erro.

1.1.4 Instalar SuperDirt

O TidalCycles destina-se a funcionar sobre o SuperDirt, portanto você terá que rodá-lo primeiro para fazer som. Veja aqui como instalá-lo.

Inicie o SuperCollider IDE (para iniciar o IDE a partir de um terminal digite 'scide'), e na pasta da janela do editor na seguinte linha de código:

```
Quarks.checkForUpdates({Quarks.install("SuperDirt", "v1.1.3"); thisProcess.recompile()})
```

Execute o código clicando sobre ele, tenha certeza de que o cursor está na mesma linha de código copiada, então pressione Shift-Enter.

Vai demorar um pouco para instalar. Você verá algo como o seguinte:

```
Installing SuperDirt
Installing Vowel
Vowel installed
Installing Dirt-Samples
Dirt-Samples installed
SuperDirt installed
compiling class library....*********************************************
*********************************( blah blah, e finalmente algo como :)
*** Welcome to SuperCollider 3.10.0. ***********************************
For help press Ctrl-D.
```

1.1.5 Usando o intérprete do SuperCollider em um terminal

Você também pode instalá-lo usando o intérprete terminal. Você pode querer se familiarizar com ele se preferir usar seu próprio editor de texto. Há ótimos plugins de integração SuperCollider disponíveis para Emacs, Vim ou Atom.

Para iniciar o intérprete basta executar o **sclang** em um terminal, depois basta colar a linha de comando de cima e pressionar Enter para executá-lo. Uma vez concluída a instalação, você pode sair do intérprete pressionando Ctrl + C.

1.1.6 Instalar as extensões do editor

O TidalCycles foi feito para ser executado em um ambiente interativo. A maneira de fazê-lo é obter um editor de texto e instalar uma extensão para ele. Aqui está uma lista de extensões que você pode querer tentar:

- Extensão do Atom
  Iniciar o Atom e instalar o plugin TidalCycles. Você pode encontrá-lo através dos menus em edit / settings/preferences / install, depois digitando "tidalcycles"na caixa de busca. Uma vez instalado, reinicie o Atom.

- Extensão do Emacs
  Um pacote MELPA é fornecido para a integração de TidalCycles dentro do Emacs. Você deve primeiro certificar-se de ter o MELPA instalado em sua máquina (aqui estão as instruções em inglês); basicamente modificando seus arquivos init.el ou .emacs com o primeiro trecho de código e depois executando M-x package-refresh-contents no Emacs; Aqui algumas informações de atualização do chaveiro (inglês) se ele não conseguir verificar a assinatura após executar o último comando), então simplesmente execute M-x package-install [return] tidal [return]. Para mais informações ou solução de problemas, verifique a página do Emacs.

  Esta extensão fornece um modo principal para arquivos ***.tidal**. Uma vez instalado o pacote, você pode simplesmente abrir um *script* Tidal e pressionar **C-c- C-s** para iniciar Tidal no Emacs, depois **C-return** para executar a declaração sob seu cursor.

- Extensão do Vim

1.1.7 Teste sua instalação

Agora você está pronto para Iniciar TidalCycles pela 1ª vez.

1.1.8 Solução de problemas de instalação

O Supercollider é executado em um servidor de áudio Jack a fim de fornecer som a seus alto-falantes. Caso apareça na janela de postagem do SuperCollider um erro

```
Couldn't set realtime scheduling priority 1:  Operation not permitted
```

você precisará configurar Jack com o comando `sudo dpkg-reconfigure jackd2` e adicionar seu nome de usuário ao grupo de áudio com `sudo addgroup -username- audio` (com seu nome de usuário Linux ao invés de -username-).

Você pode verificar se seu nome de usuário já está no grupo de áudio, digitando o comando `groups -username-`. Você pode precisar sair e voltar a entrar para que isto tenha efeito.

Mais Solução de problemas em Troubleshooting a Tidal installation.

1.2 MacOS

1.2.1 Instalação automatizada MacOS

Ainda estamos trabalhando nesta instalação automatizada, mas ela parece funcionar bem para a maioria das pessoas. Se você tiver problemas, por favor, junte-se a nós no canal de bate-papo #tidal-install e faremos o melhor para ajudar. Também ficaremos muito felizes em saber dos sucessos!

1.2.2 Executando o script

Você pode executar o script de instalação simplesmente abrindo uma janela de terminal, colando o seguinte comando e pressionando enter:

```
https://raw.githubusercontent.com/tidalcycles/tidal-bootstrap/master/tidal-bootstrap.command -sSf | sh
```

Provavelmente, em algum momento, ele irá pedir sua senha. Conforme você digita, os caracteres não aparecerão na tela!

Muitas informações confusas irão passar pelo terminal. Basta deixar o script rodar até o final.

1.2.3 O que o script faz?

O script instala as ferramentas mencionadas no guia de instalação manual do TidalCycles (inglês). Em particular, o script verifica se os seguintes programas estão instalados no sistema, e os instala se estiverem faltando.

- SuperCollider (e SuperDirt)
- Atom (e o plugin TidalCycles)
- Linguagem Haskell (ghcup)
- O Tidal pattern engine

1.3 Windows

Esta instalação instala um ambiente Tidal completo e suas dependências (Git, Haskell, Atom, SuperCollider, SuperDirt e os samples padrões de Dirt). Neste processo, se supõe que estes componentes ainda não tenham sido instalados. Caso já tenham sido instalados, talvez seja melhor instalar os componentes à mão.

1.3.1 Instalação automatizada em 3 etapas

A instalação é realizada em três etapas. A maior parte é automatizada, mas é esperado que algumas janelas de segurança aparecem como pop-ups para você confirmar a instalação.

No Windows 7 é preciso fazer uma pequena preparação antes de seguir as etapas a seguir, veja as instruções na seção ao final.

1. Abra um powershell do Windows em modo 'admin'

   No Windows 10 - Você pode fazer isso mantendo pressionada a tecla windows e pressionando 'x', depois escolhendo Windows PowerShell (admin) no menu que aparecer

   No Windows 7 - Clique no botão iniciar, digite "powershell", depois clique com o botão direito do mouse e escolha "Run as Adminstrator"(Executar como Administrador)

2. Você precisará instalar o gerenciador de pacotes chocolatey. Se você não o instalou anteriormente, você pode fazê-lo colando e executando este comando:

   ```
   Set-ExecutionPolicy Bypass -Scope Process -Force; iex ((New-Object System.Net.WebClient).DownloadString('https://chocolatey.org/install.ps1'))
   ```

3. Uma vez executado, instale Tidal e todas as suas dependências, colando no seguinte comando:

   ```
   choco install tidalcycles
   ```

Ainda estamos trabalhando neste material, por isso, muitas informações confusas vão aparecer. Por favor, ignore qualquer mensagem sobre o reinício da powershell. Apenas deixe o processo correr até o final.

Você pode então Iniciar TidalCycles pela 1ª vez.

Por favor, informe-nos como foi o processo de instalação, tanto se for bem ou mal sucedido! Não deixe de nos enviar qualquer mensagem de erro que aparecer durante o processo. Ou nos siga pelo canal TOPLAP de tidal-install (inglês), ou na categoria "installation help"no fórum do Tidal Club.

1.3.2 Windows 7

Caso esteja usando o Windows 7 é necessário alguma preparação extra.

1. Certifique se o Windows 7 está atualizado .
2. Instale e atualize a versão .NET 4.5. Baixe aqui.
3. Instale o Visual C++ redistribuível. Baixe aqui.

1.4 Problemas durante a instalação

Caso não tenha conseguido colocar o Tidal em funcionamento após as instruções de instalação acima, eis algumas instruções para localizar qual o problema. Acesse aqui o link com instruções em inglês.

Capítulo 2

Atualizar o Tidal

Para evitar problemas, recomendamos calorosamente a atualização conjunta dos três componentes do TidalCycles:

- Biblioteca TidalCycles
- Plug-in do editor
- Sintetizador SuperDirt

2.1 Biblioteca TidalCycles

Atualize a Biblioteca TidalCycles com o seguinte código a partir do terminal:

```
cabal update
cabal install tidal --lib
```

Se você estiver utilizando uma versão mais antiga do haskell, você pode ter que perder o '–lib' do comando acima.

2.2 Plug-in do editor

Para atualizar seu plug-in do editor de TidalCycles (que pode ser chamado de "extensão"ou "pacote"). No Atom, você pode fazer isso através do menu de preferências / atualizações.

Em VSCode, você pode atualizar o pacote TidalCycles visualizando suas extensões (menu Exibir / Extensões, ou clique no ícone Extensões na barra de atividades à esquerda), procurando por TidalCycles e então escolha atualizar.

2.3 SuperDirt

Para atualizar o sintetizador/biblioteca de som SuperDirt, abra o SuperCollider copie e cole o código abaixo e depois execute-o da seguinte forma (coloque o cursor na linha copiada, pressione shift-enter):

```
Quarks.update("SuperDirt")
```

Depois disso, você precisará recompilar a biblioteca da classe. Você pode fazer isso reiniciando o SuperCollider, ou através da entrada "Recompilar biblioteca de classes"no menu "Idioma".

Capítulo 3

Iniciar TidalCycles pela 1ª vez

Existem duas etapas para iniciar os TidalCycles:

1. Iniciar o SuperCollider e iniciar o SuperDirt dentro do SuperCollider.
2. Inicie o TidalCycles dentro do seu editor.

3.1 Iniciar o SuperDirt dentro do SuperCollider

Há duas maneiras de iniciar o SuperDirt. Você pode tanto rodá-lo pelo IDE do SuperCollider[1] como pelo terminal usando o **sclang** (interpreter do SuperCollider). Recomenda-se aos novos usuários que utilizem a IDE.

3.1.1 Usando a IDE do SuperCollider

Para iniciar o SuperDirt, cole o seguinte código em uma janela do SuperCollider[2]:

```
SuperDirt.start
```

Em seguida, posicione o cursor na linha do código, pressione shift-enter. A janela do post deve mostrar o progresso da inicialização da SuperDirt e, no final, você deve ver:

[1] IDE Integrated Development Environment ou Ambiente de Desenvolvimento Integrado.

[2] (para usuários de macOS 10.14.6, você provavelmente precisará copiar o script de inicialização e colá-lo em ~/Library/Application Support/SuperCollider/startup.scd)

```
SuperDirt: listening to Tidal on port 57120
```

Inicialização automática do SuperDirt na IDE

Se você quiser que o SuperDirt comece automaticamente ao abrir o SuperCollider, adicione a linha `SuperDirt.start` ao arquivo de inicialização do SuperCollider. Você pode editar o arquivo inicial do SuperCollider a partir do próprio SuperCollider, escolhendo File >Open Startup File no menu superior. Se você desejar usar um arquivo de inicialização mais abrangente do SuperDirt com mais opções, consulte este exemplo.

3.1.2 Usando o terminal de interpretação do SuperCollider

Você pode iniciar SuperDirt usando este script de inicialização. Para isso basta digitar no terminal `sclang superdirt_startup.scd`.

Uma vez tudo carregado, você deve ser capaz de encontrar a seguinte declaração na saída:

```
SuperDirt: listening to Tidal on port 57120
```

O script inicial padrão é bem simples, ele fornece duas órbitas e carrega as amostras padrão, mas você pode querer editá-lo para configurar as amostras personalizadas (e adicionar órbitas também).

3.2 Iniciar o TidalCycles dentro do editor de texto

3.2.1 Instruções para Atom

1. Iniciar Atom
2. Criar um novo arquivo e salvá-lo com um nome de arquivo que termina em .tidal, por exemplo, test.tidal.
3. Abrir o menu Packages e selecione TidalCycles ->Boot TidalCycles. Uma pequena janela se abrirá na parte inferior da janela contendo o prompt ‘t>‘ (e, espera-se, sem mensagens de erro).

Tente executar um padrão simples digitando o código abaixo, selecionando a linha e pressionado shift-enter para validá-lo (Pressionar ctrl-enter também funcionará selecionando múltiplas linhas).

```
d1 $ sound "bd sn"
```

Se você ouvir som, parabéns!

Você deve ouvir um som de bumbo eletrônico seguido por um som de caixa. Rode o seguinte código para silenciar os instrumentos :

```
silence
```

Se ficar enroscado, escreva e compartilhe os problemas no fórum, ou no canal #tidal do Lurk RocketChat.

Como é comum no software livre, você tem escolhas alternativas para os diferentes componentes que compõem um sistema TidalCycles. Atom e SuperDirt podem ser tudo o que você precisa, mas há outros editores e sintetizadores que você pode usar.

3.2.2 Instruções para Emacs

Vá até a instalação do Linux para instalar o plugin Emacs, caso ainda não o tenha feito. Uma vez instalado o plugin Tidal, tudo que você precisa fazer é abrir um arquivo `*.tidal` e pressionar `C-c C-s` para iniciar o Tidal no Emacs. Se tudo der certo, você deve ver a saída da Tidal aparecendo em uma janela separada.

Agora você pode simplesmente pressionar `C-return` para avaliar qualquer declaração sob seu cursor. Você pode tentar esta como um exemplo para verificar se tudo correu bem:

```
d1 $ sound "bd sn"
```

Você deve ouvir um som de bumbo eletrônico seguido por um som de caixa. Em seguida, avalie o seguinte para silenciar todos os seus instrumentos :

```
silence
```

Capítulo 4

Tutorial

Com o TidalCycles e (Super)Dirt instalado, provavelmente já tenha feito alguns sons. Este tutorial ajudará a entender como funciona tidal e como usar ele para criar padrões simples e composições complexas. Caso você queira um breve resumo da sintaxe dos padrões disponíveis, verifique sintaxe do analisador de seqüências.

Por que não tocar com o código na medida em que os lê, fazendo suas próprias experiências, mudando os exemplos e escutando para onde eles o levam?

4.1 Criando Seqüências Rítmicas

4.1.1 Tocando um único sample

Tidal fornece 16 'conexões' com o sintetizador SuperDirt, nomeadas de d1 a d16. Aqui está um exemplo mínimo, que toca um bumbo a cada ciclo:

```
d1 $ sound "bd"
```

Rode o código acima no editor Atom (ou Emacs) pressionando Ctrl+Enter. Se você quiser parar o som novamente, leia abaixo a seção sobre silêncio.

No código acima, o sound nos diz que estamos executando um padrão com um sample "bd"que contém um único som. "bd"é a amostra de um bumbo. As amostras estão dentro da pasta Dirt-Samples que vem com SuperDirt. Cada subpasta dentro da pasta Dirt-Sample corresponde ao nome de amostra (como por exmplo bd, sn, glitch, etc).

Para encontrar os sample de SuperDirt em seu sistema, na IDE SuperCollider selecione o item de menu Arquivo > Abrir Diretório de Suporte ao Usuário. A partir daí, abra as marcas de download e, finalmente, as amostras de sujeira. Você deve encontrar muitas pastas, cada uma delas é um banco de amostras contendo arquivos wav padrão. Sinta-se livre para criar novas pastas e adicionar seus próprios sons, veja a página Base de Usuários de Amostras Personalizadas (Custom Samples) para mais informações.

Podemos escolher uma amostra diferente na pasta bd adicionando dois pontos (:) e depois um número. Por exemplo, isto escolhe o quarto sample (inicia-se a contagem do zero, então :3 lhe dá o quarto sample da pasta):

```
d1 $ sound "bd:3"
```

Se você especificar um número maior do que o número de amostras em uma pasta, então Tidal apenas "envolve"de volta à primeira amostra novamente (começa a contar a zero, por exemplo, em uma pasta com cinco amostras, "bd:5"tocaria "bd:0").

Também é possível especificar o número da amostra separadamente:

```
d1 $ sound "bd" # n "3"
```

A utilidade de fazer isso será evidente mais tarde.

4.1.2 Seqüências de múltiplos sample

Colocar as coisas entre aspas permite que você defina uma seqüência. Por exemplo, o que se segue lhe dá um padrão de bumbo e, em seguida, uma caixa:

```
d1 $ sound "bd sd:1"
```

Quando você executa o código acima, você está substituindo o padrão anterior por outro em tempo real. Parabéns, você está codificando ao vivo.

4.1.3 Tocando mais de uma sequência

A maneira mais fácil de reproduzir várias seqüências ao mesmo tempo é usar duas ou mais conexões com o sintetizador:

```
d1 $ sound"bd sd:1"
d2 $ sound "hh hh hh hh"
d3 $ "arpy" sound
```

NOTA I: cada conexão deve ser executada separadamente em seu editor de texto. Ou seja, você deve pressionar Ctrl+Enter três vezes, uma para cada linha acima. Certifique-se de que haja uma linha em branco entre eles cada padrão, ou Tidal os avaliará juntos e ficará confuso (se você quiser avaliar apenas uma linha, você pode pressionar Shift+Enter).

NOTA II: Se você preferir se referir a padrões pelo nome, ao invés de pelo número, você pode fazer isso com p, por exemplo:

```
p "susan" $ sound "bd sd:1"
p "gerard" $ sound "hh hh hh hh"
```

4.1.4 O que é um ciclo?

Um ciclo é o "loop"de tempo principal em Tidal. O ciclo se repete continuamente em segundo plano, mesmo quando você parar de tocar as amostras. A duração do ciclo permanece sempre a mesma, a menos que você o modifique com *setcps*, como será explicado mais tarde. Por padrão, um ciclo por segundo.

Note que este ciclo continuo do *looping* não precisa lhe constranger, por exemplo, é comum estender um padrão fora de um único loop, e variar os padrões de um loop para o outro. Veremos várias maneiras de fazer isso mais tarde.

Todas as amostras dentro de um padrão são controladas em um único ciclo. Todos os loops no exemplo a seguir são executados durante o mesmo período de tempo:

```
d1 $ s "bd sd"
d1 $ s "bd sd hh cp mt arpy drum"
d1 $ s "bd sd hh cp mt arpy drum odx bd arpy bass2 feel future"
```

Observe como quanto mais passos você acrescenta ao padrão, mais rápido ele os toca, a fim de encaixá-los a todos. Não importa quantas amostras você coloque em um padrão desta maneira, elas sempre serão distribuídas uniformemente dentro de um único ciclo.

4.2 Silêncio

Neste ponto, você provavelmente quer saber como parar os loops que você começou. Um loop vazio é definido como silêncio, então se você quiser 'desligar' um loop, você pode simplesmente defini-lo para isso:

```
d1 silence
```

Se você quiser configurar todas as conexões (de d1 a d9) para ficar em silêncio de uma só vez, há um atalho com uma só palavra para isso:

```
hush
```

Você também pode isolar uma única conexão e silenciar todas as outras com a função solo. Você pode fazer isso dessa forma:

```
d1 $ sound "bd"
d2 $ sound " ~ cp"
```

4.3 Padrões dentro de Padrões

Você pode usar a sintaxe dos colchetes de Tidal para criar um agrupamento de padrões:

```
d1 $ sound "[bd sd sd] cp"
```

Os colchetes permitem que vários eventos sejam tocados dentro de uma única batida. Você pode pensar no padrão acima como tendo duas batidas. A primeira batida dentro dos colchetes subdividida tem três batidas [bd sd sd] e a segunda batida cp . Na prática, isto significa que você pode criar sub-divisões de ciclos mais densos:

```
d1 $ sound "bd [sd sd]"
d1 $ sound "bd [sd sd sd]"
d1 $ sound "bd [sd sd sd sd]"
d1 $ sound "[bd bd] [sd sd sd sd]"
d1 $ sound "[bd bd bd] [sd sd]"
d1 $ sound "[bd bd bd bd] [sd]"
```

Você pode até mesmo aninhar grupos dentro de grupos para criar padrões cada vez mais densos e complexos:

```
d1 $ sound "[bd bd] [bd [sd [sd sd] sd] sd]"
```

Uma abreviação para este tipo de agrupamento é colocar um período . entre grupos, em vez de envolvê-los entre colchetes. Chamamos esta técnica de "marcação com os pés". Por exemplo, estes dois padrões são equivalentes:

```
d1 $ sound "bd bd . sd sd sd . bd sd"
d1 $ sound "[bd bd] [sd sd sd] [bd sd]"
```

A primeira abordagem é mais fácil de digitar, porém ela foi implementada recentemente ao TidalCycles. Muitos dos exemplos usados ainda usam os colchetes.

4.3.1 Camadas (Polirrítmicas) em vez de Agrupamento

Você pode sobrepor vários lopps, usando vírgulas para executar diferentes partes:

```
d1 $ sound "[bd bd bd, sd cp sd cp]"
```

Isto toca a seqüência `bd bd bd` ao mesmo tempo que a `sd cp sd cp`. Note que a primeira seqüência tem apenas três eventos, e a segunda quatro. Como o Tidal garante que ambos os loops se encaixem dentro da mesma duração cíclica, o resultado é uma polirritmia de 3/4.

Você pode sobrepor qualquer número destes subpadrões para criar polirritmias diversas:

```
d1 $ sound "[bd bd bd, sd cp sd cp, arpy arpy, moog]"
```

Ou ainda você pode usar agrupamentos dentro das camadas:

```
d1 $ sound "[bd bd bd, [sd sd] cp, arpy [arpy [arpy arpy] arpy arpy], moog]"
```

4.3.2 Tocando uma batida por ciclo

Para especificar um grupo onde apenas um passo é jogado por ciclo, use colchetes de ângulo. Por exemplo:

```
d1 $  sound "bd <arpy:1 arpy:2 arpy:3>"
```

A linha acima resulta na seqüência `bd arpy:1 bd arpy:2 bd arpy:3`, ao longo de três ciclos.

4.4 Padrão de repetição e velocidade

4.4.1 Repetição

Há dois símbolos que você pode usar dentro de padrões para acelerar ou retardar as coisas: * e /. Você pode pensar neles como multiplicador e divisor.

Use o símbolo * para repetir quantas vezes quiser um padrão ou sub-padrão:

```
d1 $ sound "bd*2"
```

Isto é o mesmo que fazer `d1 $ sound "bd bd"`.

O código acima usa *2 para reproduzir o sample duas vezes.

Você pode usar o símbolo / para fazer com que uma parte de um padrão desacelere, ou que ocorra com menos freqüência:

```
d1 $ sound "bd/2"
```

O código acima usa /2 para tocar um sample com a metade da freqüência, ou uma vez a cada 2 ciclos.

O uso de números diferentes funciona como você esperaria:

```
d1 $ sound "bd*3"  -- toca a amostra três vezes a cada ciclo
d1 $ sound "bd/3" -- toca a amostra uma vez a cada três ciclos
```

4.4.2 Usando * e / em grupos de padrões

Você pode aplicar os símbolos * e / em grupos de padrões:

```
d1 $ s "[bd sn]*2 cp"
d1 $ s "[bd sn] cp/2"
d1 $ s "[[bd sn] cp]*2" -- acelera 2 vezes todo o padrão
d1 $ s "[[bd sn] cp]/2" -- desacelera 2 vezes todo o padrão
```

Você também pode usar os símbolos em grupos aninhados para criar ritmos mais complexos:

```
d1 $ s "[bd sn*3]/2 [bd sn*3 bd*4]/3"
d1 $ s "[bd [sn]*2]/2 [bd [sn bd]/2]*2"
```

4.5 Modificando seqüências com funções

Tidal se torna ela própria quando você começa a construir padrões com funções que transformam os padrões de várias maneiras.

Por exemplo, `rev` inverte um padrão:

```
d1 $ rev (sound "bd*2 [bd [sn sn*2 sn] sn]")
```

Isso não soa tão interessante assim, mas as coisas podem ficar mais instigantes quando combinado funções são combinadas entre si. Por exemplo, a função**every** utiliza três parâmetros: um número, uma função e um padrão para aplicar a função. O número especifica a freqüência com que a função é aplicada ao padrão. O exemplo seguinte inverte o padrão no quarto ciclos:

```
d1 $ every 4 (rev) (sound "bd*2 [bd [sn sn*2 sn] sn]")
```

Demora um pouco para nos acostumarmos com o modo como os parênteses são utilizados pelo Tidal. No exemplo anterior, a função `rev` utiliza um apenas parâmetro, o padrão, que 'embrulhamos' entre parênteses (`sound "[bd bd] [bd [sn [sn] sn] sn]"`) para utilizá-lo como parâmetro de `rev`. No exemplo acima, `every` utiliza três parâmetros: um número, uma função e um padrão. Como antes, tivemos que embrulhar o pa-

drão (`sound "[bd bd] [bd [sn [sn] sn] sn]"`), mas também o (`rev`). Com a prática esse tipo de sintaxe se tornará mais clara.

Você também pode usar as funções `fast` para acelerar ou `slow` para desacelerar a reprodução em um quarto da velocidade padrão:

```
d1 $ slow 4 $ sound "bd*2 [bd [sn sn*2 sn] sn]"
```

Agora acelerando quatro vezes a velocidade:

```
d1 $ fast 4 $ sound "bd*2 [bd [sn sn*2 sn] sn]"
```

Note que o `slow 0.25` faria exatamente o mesmo que o `fast 4`.

Lembre-se, isto pode ser aplicado de forma seletiva:

```
d1 $ every 4 (fast 4) $ sound "bd*2 [bd [sn sn*2 sn] sn]"
```

Note novamente o uso de parênteses envolvendo `fast 4`. Isto é necessário, para agrupar a função rapidamente com seu parâmetro 4, antes de ser passado como parâmetro para a função `every`.

Nos exemplos acima, a função `sound` toma um padrão de samples e o transforma em um padrão de gatilhos sintetizadores. Entender como funciona essa técnica pode levar algum tempo. O importante agora é lembrar o que tudo é tratado aqui como padrões. Neste caso, isto significa, que, você pode operar no padrão interno dos nomes das amostras, ao invés do padrão externo de gatilhos de sintetizadores que o `sound` lhe dá:

```
d1 $ sound (every 4 (fast 4) "bd*2 [bd [sn sn*2 sn] sn]")
```

A função `fast` também é conhecida como densidade, `desity` , que na verdade é seu nome mais antigo. Portanto, muitos exemplos usarão `desity` em vez `fast` (que é um pouco mais rápida para digitar). Ambas fazem exatamente a mesma coisa.

4.5.1 Onde estão todas as funções?

Existem varias funções que ajudam a alterar padrões. Algumas delas reordenam seqüências, outras alteram o tempo, outras fornecem lógica condicional e outras podem auxiliar na composição de padrões mais complexos.

Apresentaremos muitas das funções principais nesta introdução, e uma lista mais completa das funções disponíveis em Tidal pode ser encontrada na página tidalcycles.org.

4.6 Aplicando efeitos como padrões de controle

O TidalCycles tem uma série de efeitos que você pode aplicar aos sons. Alguns efeitos realizam operações simples, como alterar o volume, outros realizam operações mais complexas como adicionar distorção e *delay*. Isto é feito a partir de padrões de controle. Na verdade, o próprio `sound` cria um padrão de controle e nós aplicamos efeitos combinando padrões de controle.

Você pode combinar padrões de controle adicionando o operador # entre eles:

```
d1 $ sound "bd*4" # crush "4"
```

O código acima usa a `crush` para criar um padrão de controle de *bitcruching* com um valor de 4 (que soa realmente grunhido), e usa # para unir isso com o padrão de controle de som.

Você pode combinar vários padrões de controle juntos, com o # operador:

```
d1 $ sound "bd*4" # crush "4" # speed "2"
```

O código acima executa dois efeitos ao mesmo tempo `crusch` e `speed` (distorção e velocidade). `#speed 2` significa tocar o sample 2 vezes mais rápido (agudo), dobrando assim o tom original do sample. `#speed 0.5` significa dividir o sample por 2 vezes mais lento (grave). `#speed -1` significa executar o sample de trás para frente.

4.6.1 Valores de controle também são padrões

Os valores dos efeitos são especificados em aspas duplas. Isto significa que os valores dos efeitos também podem ser padronizados:

```
d1 $ sound "bd*4" # gain "1 0.8 0.5 0.7"
```

O efeito `#gain` altera o ganho (volume) da amostra. Todos padrões de controle seguem as mesmas regras de agrupamento que os padrões de *sound*:

```
d1 $ sound "bd*4 sn*4" # gain "[[1 0.8]*2 [0.5 0.7]]/2"
```

Também é possível aplicar funções aos padrões de controle:

```
d1 $ sound "bd*4" # (every 3 (rev) $ gain "1 0.8 0.5 0.7")
```

Como no exemplo anterior `sound`, você deve usar parênteses após o `gain` para especificar uma função sobre o padrão de ganho.

Isto também funciona:

```
d1 $ sound "bd*4" # gain (every 3 (rev) $ "1 0.8 0.5 0.7")
```

No exemplo acima, `every 3 (rev)` está sendo aplicada ao padrão numérico "1 0.8 0.5 0.7". No exemplo anterior, a mesma função foi aplicada ao `gain "1 0.8 0.5 0.7"`, que é um padrão de controle de ganho. Neste caso, se você aplicar a função antes ou depois dos números serem transformados em controles o resultado será exatamente o mesmo.

4.6.2 Ordem do padrão de controle

Você pode especificar o controle de efeito antes do controle de `sound`:

```
d1 $ gain "1 0.8 0.5 0.7" # sound "bd"
```

Aqui a ordem dos elementos importa; com # a estrutura do padrão é dada pelo padrão da esquerda. Neste caso, apenas um som bd é dado, mas você ouve quatro, porque a estrutura vem do padrão de ganho à esquerda.

4.6.3 Modificando valores de controle

O operador # é apenas um atalho para uma forma mais longa de operador chamado |». O operador |> faz parte de uma família de operadores, e significa algo especial sobre a combinação de padrões tópico que iremos aprofundar logo mais. Tudo o que se precisa saber agora é que |> é usado para combinar padrões.

Usa-se |» para combinar padrões condicionalmente:

```
d1 $ every 2 (|> speed "2") $ sound "arpy*4" |> speed "1"
```

Há outros tipos de operadores que permitem realizar aritmética:

```
|+
|-
|*
|/
```

Por exemplo, o uso do `|+` realizará uma operação de adição e adicionará a um valor original:

```
d1 $ every 2 (|+ speed "1") $ sound  "arpy*4" |> speed "1"
```

O código acima resulta em uma velocidade de "2"a cada novo ciclo.

O seguinte multiplicará os valores:

```
d1 $ every 2 (|* speed "1.5") $ sound "arpy*4" |> speed "1"
```

É possível fazer padrões e encadeamentos mais complexos, e com qualquer efeito:

```
d1 $ every 3 (|- note "3") $ every 2 (|+ up "5") $ sound "arpy*4" |> note "0 2 4 5"
```

Por enquanto, pode valer a pena se ater apenas a estas formas de combinar padrões de controle. No entanto, caso esteja curioso, confira os outros padrões de estruturas que estão disponíveis.

4.6.4 Alguns Efeitos Comuns

Aqui segue uma rápida lista com alguns efeitos do Tidal:[1]

- `gain` - altera o volume, valores de 0 a 1
- `pan` - panorâmica sonora à esquerda e à direita, valores de 0 a 1
- `shape` - um tipo de amplificador, valores de 0 a 1

[1] A lista completa de efeitos está disponível na seção Referência

- `vowel` - um filtro formador de vogais, os parâmetros incluem a, e, i, o, u
- `speed` - muda a velocidade de reprodução de uma amostra, veja abaixo

4.7 Atalhos para padrões numéricos

A partir da versão 0.9 do Tidal, surgiram algumas maneiras simples de digitar quando se trabalha com padrões numéricos.

Por exemplo, especificar padrões de números simples:

```
d1 $ sound "arpy(3,8)"  # n "2"
```

Não é preciso aspas duplas, então isto funciona bem:

```
d1 $ sound "arpy(3,8)"  # n 2
```

Entretanto, se quisesse mais de um valor no padrão n, seria necessário as aspas:

```
d1 $ sound "arpy(3,8)" # n "2 5"
```

É possível lidar com padrões numéricos de outras maneiras. Por exemplo, fazendo álgebra:

```
d1 $ sound "arpy(3,8)" # n ("0 2" * 2)
d1 $ sound "arpy(3,8)" # n (every 4 (* 2) "0 2")
d1 $ n (off 0.125 (+12) $ off 0.25 (+7) $ slow 2 $ "0(3,8) [5 7]") # sound "supergong"
```

O sound do **supergong** requer a instalação de `sc3-plugins`.

Isto ainda é bastante novo, por isso não verá muitos exemplos nesta documentação.

É possível ainda especificar números crescentes ou decrescentes com um intervalo, por exemplo, este:

```
d1 $ n "[0 .. 7] [3 .. 1]" # sound "supergong"
```

Isso é a abreviatura de:

```
d1 $ n "[0 1 2 3 4 5 6 7] [3 2 1]" # sound "supergong"
```

4.8 Velocidade e Tonalidade do sample

É possível alterar a velocidade de reprodução de um sample em TidalCycles usando o efeito de velocidade speed. Com speed é possível alterar o tom, para criar um efeito estranho, ou para combinar a duração de uma amostra com um período específico do tempo do ciclo (mas veja a função loopAt para fazer isso de maneira mais fácil de fazer).

Você pode definir a velocidade de um *sample* usando o efeito speed com um número.

- speed "1" toca uma amostra em sua velocidade original
- speed "0,5" toca uma amostra com metade de sua velocidade original
- speed "2" toca uma amostra com o dobro de sua velocidade original

```
d1 $ sound "arpy"   #speed "1"
d1 $ sound "arpy" #speed "0.5"
d1 $ sound "arpy"  #speed "2"
```

Assim como outros efeitos, pode-se especificar um padrão de velocidade:

```
d1 $ speed "1 0.5 2 1.5"  #sound "arpy"
```

É possível tocar um sample de tras pra frente especificando valores negativos:

```
d1 $ speed  "-1 -0.5 -2 -1.5" #sound "arpy"
```

4.8.1 Tocar um sample em várias velocidades simultaneamente

Use a sintaxe de agrupamento de padrões com uma vírgula para fazer com que a velocidade de um sample seja executada em várias velocidades ao mesmo tempo:

```
d1 $ sound "arpy" # speed "[1, 1.5]"
d1 $ speed "[1 0.5, 1.5 2 3 4]" # sound "arpy"
```

4.8.2 Usando speed para escala dodecafônica

É possível usar a função up para alterar a velocidade do sample. up é um efeito que combina speed com progressão cromática em uma escala de 12 tons. O exemplo seguinte toca uma escala cromática:

```
d1 $ up "0 1 2 3 4 5 6 7 8 9 10 11" # sound "arpy"
```

Use a função run para criar um padrão de incremento de números inteiros:

```
d1 $ up (run 12) # sound "arpy"
```

Falaremos mais sobre a função run mais tarde.

4.9 Seqüências Euclidianas

Digitando dois números entre parênteses após um elemento dentro de um padrão o Tidal irá distribuir o primeiro números de sons em proporção ao segundo número de batidas:

```
d1 $ sound "bd(5,8)"
```

É possível usar a função euclid para fazer a mesma coisa que o exemplo acima.

```
d1 $ euclid 5 8 $ sound "bd"
```

Existem outras funções relativas ao euclid como euclidInv, que inverte a função euclid e euclidFull, que toca todas as batidas do padrão euclidiano adicionando um padrão diferente na batida off.[2]

```
d1 $ euclidInv 5 8 $ sound "sn"
d1 $ euclidFull 5 8 (s  "bd") (s "sn")
```

A notação de parênteses dentro de um único elemento em um padrão:

[2] Confirma mais informações no link

```
d1 $ s  "bd(3,8) sn*2"
d1 $ s  "bd(3,8) sn(5,8)"
```

Ao adicionar um terceiro parâmetro, o padrão "gira"começando em uma batida diferente:

```
d1 $ som "bd(5,8,2)".
```

A função `euclid` pode ser aplicada a um algoritmo Euclidiano a um padrão complexo, no entanto, a estrutura padrão é comprometida:

```
d1 $ euclid 3 8 $ sound "bd*2 [sn cp]".
```

No exemplo acima, três sons são pegos do padrão à direita, seguindo a estrutura dada por `euclid 3 8`. O resultado é dois bd, um cp, com sn excluído.

Como bônus, é possível padronizar parâmetros dentro do parêntese, por exemplo, para alternar entre 3 e 5 batidas dentro de uma estrutura com 8 pulsos:

```
d1 $ s "bd([5 3]/2,8)"
```

Estes tipos de sequências utilizam o "algoritmo de Bjorklund", criado para ser utilizado na área de física nuclear. Sua estrutura é semelhante ao primeiro algoritmo conhecido escrito no tratado matemático e geométrico "Os Elementos"escrito por Euclides em Alexandria por volta do ano 300 antes de cristo. Você pode ler mais sobre isto no artigo The Euclidean Algorithm Generates Traditional Musical Rhythms escrito por Godfried Toussaint. Alguns exemplos deste trabalho estão incluídos abaixo, em alguns casos com rotação.

- (2,5) : ritmo persa do século XIII chamado Khafif-e-ramal.
- (3,4) : padrão arquetípico da Cumbia da Colômbia, assim como de uma infinidade de ritmos latinos como Calypso de Trinidad.
- (3,5,2) : ritmo persa do século XIII chamado Khafif-e-ramal, bem como um ritmo de dança folclórica romena.
- (3,7) : ritmo Ruchenitza usado em uma dança folclórica búlgara.
- (3,8) : padrão tresilo-cubano.
- (4,7) : ritmo de dança folclórica búlgara do Ruchenitza.

- (4,9) : ritmo de Aksak da Turquia.
- (4,11) : padrão usado por Frank Zappa em sua peça intitulada Outside Now.
- (5,6) : padrão York-Samai, um ritmo árabe popular.
- (5,7) : padrão Nawakhat, ritmo árabe popular.
- (5,8) : padrão cubano cinquillo.
- (5,9) : ritmo árabe popular chamado Agsag-Samai.
- (5,11) : padrão métrico usado por Moussorgsky em Quadros de uma Exposição.
- (5,12) : padrão de aplausos de Venda de uma canção infantil sul-africana.
- (5,16) : padrão rítmico da Bossa-Nova .
- (7,8) : ritmo característico do Bendir (tambor tipo adufe ou pandeiro).
- (7,12) : padrão de sino comum da África Ocidental.
- (7,16,14) : padrão de samba.
- (9,16) : padrão rítmico usado na República Centro-Africana.
- (11,24,14) : padrão rítmico dos pigmeus da África Central.
- (13,24,5) : padrão rítmico dos pigmeus Aka Pygmies no Sangha superior.

4.10 Tempo

Até aqui nenhum dos exemplos alterou a unidade de tempo.

A unidade central de tempo em Tidal são os ciclos por segundo. Por padrão, ela é ajustada para 0,5625 ou 135 BPM (batidas por minutos). O tempo pode ser Ela pode ser ajustada com a função `setcps`:

```
setcps 1
```

Para executar `setcps` exatamente como um padrão (usando Shift+Enter no editor).

O `setcps` suporta valores numéricos positivos e pode incluir também frações decimais:

```
setcps 1.5
setcps 0.75
setcps 10
```

O timing de Tidal é baseado em ciclos, ao invés de batida, porém é mais comum as pessoas pensarem em batimentos por minuto (BPM). Caso prefira pensar desta forma, é possível decidir quantas batidas deseja por ciclo, e dividir de acordo. Por exemplo, caso queira tocar a 140 bpm, com quatro batidas por ciclo, então é só digitar:

```
setcps (140/60/4)
```

É possível padronizar o tempo com a função de controle cps, por exemplo:

```
d1 $ sound "cp(3,8)"
  # cps (slow 8 $ range 0.8 1.6 saw)
```

4.11 Função run

Há uma função especial muito útil chamada run que retornará um padrão de números inteiros até um máximo valor especificado. Você pode usar run em efeitos para ajudar a geração automaticamente de um padrão linear:

```
d1 $ s "arpy*8" # up (run 8)
d1 $ s "arpy*8" # speed (run 8)
```

No exemplo acima é especificado o número de sons duas vezes - no padrão sound, bem como no padrão up ou speed. Na verdade, há uma maneira simples de definir isso apenas uma vez, simplesmente trocando sound , de modo que o parâmetro de efeito fique à esquerda:

```
d1 $ up (run 8) # sound "arpy"
```

Isto funciona porque o TidalCycles sempre toma como estrutura de um padrão do parâmetro aquilo que está à esquerda. É comum que a estrutura venha do parâmetro sound, mas nem sempre é assim.

Como run gera um padrão, é possível aplicar funções ao seu resultado:

```
d1 $ sound "arpy*8" # up (every 2 (rev) $ run 8)
```

Para um exemplo mais prático de utilização da função run, leia abaixo sobre a seleção de sample a partir de uma pasta no seu computador.

4.12 (Algoritmicamente) Seleção de sample

O parâmetro sound que temos usado até aqui pode ser dividido em outros dois parâmetros: s nome do sample, e n número do sample dentro da pasta. Os dois padrões a seguir fazem exatamente o mesmo:

```
d1 $ sound "arpy:0 arpy:2 arpy:3"
d1 $ n "0 2 3" # s "arpy"
```

É possível quebrar o parâmetro sound em dois padrões diferentes, ou seja, s que dá o nome do conjunto de sample e n que dá o índice da amostra dentro desse conjunto. Por exemplo, os dois padrões a seguir são os mesmos:

```
d1 $ sound "arpia:0 arpia:2 arpia:3".
d1 $ n "0 2 3" # s "arpia"
```

Isto nos permite separar o nome da pasta de sample do índice dentro da pasta, possivelmente com resultados surpreendentes!

Há também uma função especial chamada samples que permite fazer o mesmo usando o parâmetro sound.

```
d1 $ sound $ samples "drum*4" "0 1 2 3"
```

o código acima é o mesmo que este:

```
d1 $ sound "drum:0 drum:1 drum:2 drum:3"
```

Usar n e s juntos, ou sound com samples é indiferente.

Lembra da função run? Como a função run gera padrão de números inteiros, ela pode ser usada com n para percorrer automaticamente os índices de sample de uma pasta:

```
d1 $ n (run 4) # s "drum"
d1 $ sound $ samples "drum*4" (run 4) -- ou com samples
```

E claro é possível especificar um padrão diferente de nomes de sample:

```
d1 $ s "drum arpy cp hh" # n (run 10)
```

Novamente, trocando a ordem dos parâmetros s e n, ouve-se a diferença entre utilizar a estrutura de um ou de outro:

```
d1 $ n (run 10) # s "drum arpy cp hh"
```

NOTA: especificando um valor run maior que o número de amostras em uma pasta, o índice do número maior "envolverá"até o início das amostras na pasta (assim como na notação de dois pontos).

Às vezes é possível ver a função das amostras envoltas em parênteses:

```
d1 $ sound (samples "drum arpy cp hh" (run 10))
```

4.13 Combinando Padrões

Lembra de quando começamos a adicionar efeitos?

```
d1 $ sound "bd sn drum arpy" # pan "0 1 0.25 0.75
```

O código acima combina dois padrões que operam simultaneamente: o padrão sound e o padrão pan. Operadores especiais |>, |+, |-, |*, |/, ... permitem combinar dois padrões. Não esqueça que # é a abreviação de |>.

Mesmo trocando a ordem dos padrões eles soarão da mesma forma:

```
d1 $ pan "0 1 0.25 0.75" # sound "bd sn drum arpy"
```

É importante lembrar que ao combinar os padrões, o que estiver posicionado à esquerda determinará a estrutura rítmica. Ao remover um dos elementos do padrão pan à esquerda o ciclo de quatro samples será alterado para três.

```
d1 $ pan "0 1 0.25" # sound "bd sn drum arpy"
```

No código acima, o padrão pan determina o ritmo porque é o padrão posicionado à esquerda. O padrão sound determina apenas quais amostras são tocadas a cada momento. Ou seja, o padrão sound é mapeado em função o padrão pan.

Você pode estar se perguntando como o TidalCycles decide quais valores de sound são combinados com quais valores de pan no código acima. A regra é, cada valor à esquerda é associado a um valor à direita, onde o início do valor a esquerda, cai dentro do intervalo de tempo do valor da direita. Por exemplo, o segundo valor de pan 1 começa a um terço em seu padrão, e o segundo valor de sound sn começa a um quarto em seu padrão, e termina na metade. Como o primeiro começa (um terço) cai dentro do intervalo de tempo do segundo tempo (de um quarto até a metade), eles são combinados. O intervalo de tempo de arpy não contém nenhum conjunto do padrão de pan , por isso não combina com nada, e não é tocado.

A regra descrita acima pode ser demasiada confusa para se ter em mente durante a performance, mas na prática não há necessidade de se preocupar com ela. A proposta neste momento é experimentar e aos poucos internalizar o funcionamento e jogar com eles.

De qualquer forma, este tipo de composição a partir de combinações de padrões nos permite criar algumas sonoridades singulares:

```
d1 $ up "0 0*2 0*4 1" |> sound "[arpy, bass2, bd]"
```

Acima, o padrão sound especifica três sample para tocar em cada nota. Tanto o ritmo quanto a altura destas notas são definidos pelo padrão up.

É possível também mudar as coisas para que a estrutura venha da direita, usando os operadores `>|`, `*|`, `+|`, `/|` e `-|`, em vez de `|>`, `|*`, `|+` e `|-`, por exemplo:

```
d1 $ sound "drum" >|  n "0 1*2 ~ 3"
```

O lado do operador que a barra | está ligada, indica de onde vem a estrutura. Na verdade, se colocar a barra em ambos os lados, a estrutura vem de ambos os lados:

```
d1 $ sound "drum cp" >|  n "0 1 2"
```

4.14 Osciladores com Padrões Contínuos

Até agora, temos trabalhado apenas com padrões discretos, ou seja, padrões que contêm eventos que começam e terminam. Tidal também suporta padrões contínuos que variam continuamente ao longo do tempo. Você pode criar padrões contínuos usando funções senoidais `sine`, dente de serra `saw`, triângular `tri` e ondas quadradas `square`:

```
d1 $ sound "bd*16" # pan sine
```

O código acima usa o padrão `sine` para especificar uma oscilação em onda senoidal de valores entre 0 e 1 para os valores da pan, de modo que o bumbo se move suavemente entre os alto-falantes esquerdo e direito.

Tidal costumava ter padrões `sine` e `sine1` com intervalos diferentes, eles são pseudônimos, com ambos dando uma faixa de 0 a 1.

Além do padrão `sine`, Tidal também tem ondas `saw` (dente de serra), `tri` (triangular), e `square` (quadradas).

Assim como os padrões discretos, você pode controlar a velocidade dos padrões contínuos com lentidão ou densidade:

```
d1 $ sound "bd*16" # pan (slow 8 $ saw)
d1 $ sound "bd*8 sn*8" # pan (density 1.75 $ tri)
d1 $ sound "bd*8 sn*8" # speed (density 2 $ tri)
```

É possível combiná-los de diferentes formas:

```
d1 $ sound "bd*16" # pan (slowcat [sine, saw, square, tri])
d1 $ sound "sn:2*16" # speed ((range 0.5 3 sine) * (slow 4 saw))
```

4.14.1 Oscilador em escala

Você pode dizer as funções de oscilação para se escalar e oscilar entre dois valores usando `range`:

Uma mudança recente no Tidal redirecionou a antiga função `scale` para `range`, liberando `scale` para ser usada para outros propósitos.

```
d1 $ sound "bd*8 sn*8" # speed (range 1 3 $ tri)
d1 $ sound "bd*8 sn*8" # speed (slow 4 $ range 1 3 $ tri)
```

Você também pode usar `scale` para valores negativos, mas **atenção**, certifique-se de envolver valores negativos entre parênteses (caso contrário o intérprete pensa que você está tentando subtrair 2 de algo):

```
d1 $ sound "bd*8 sn*8" # speed (range (-2) 3 $ tri)
```

Esta técnica funciona bem com corte lento (slow) do filtro de frequências passa-baixo (cutoff):

```
d1 $ sound "hh*32" # cutoff (range 300 1000 $ slow 4 $ sine) # resonance "0.4"
```

NOTA: Apesar destes padrões contínuos produzem valores oscilantes, ainda é necessário combiná-los com padrões discretos.

4.15 Pausas

Até agora, temos criado padrões para produzir som. Como fazer para gerar um intervalo de silêncio? O "til" faz isso:

```
d1 $ sound "bd bd ~ bd"
```

Pense no como uma batida silenciosa em uma sequência de sons, que gera intervalos de silêncio.

4.16 Polimetria

Falamos sobre polirritmos antes, mas Tidal também pode produzir seqüências de polimetria. Um padrão polimétrico possui ao menos dois padrões com comprimentos e seqüências diferentes, mas compartilham o mesmo pulso ou tempo.

Você usa a sintaxe com colchetes para criar um ritmo de polimétrico:

```
d1 $ sound "{bd hh sn cp, arpy bass2 drum notes can}"
```

O código acima gera um ritmo em cinco sendo tocado ao pulso de um ritmo em quatro. Mudando a ordem dos grupos, o resultado será um ritmo de quatro sobre um ritmo de cinco:

```
d1 $ sound "{arpy bass2 drum notes can, bd hh sn cp}"
```

Às vezes, se deseja criar um ritmo polimetrico estranho sem precisar criar um ritmo de base. Isto pode ser feito com pausas :

```
d1 $ sound "{~ ~ ~ ~, arpy bass2 drum notes can}"
```

Mas uma maneira mais eficiente é usar o símbolo % após o colchete para especificar o número de notas no pulso base:

```
d1 $ sound "{arpy bass2 drum notes can}%4"
```

O exemplo acima é semelhante a este:

```
d1 $ sound "{~ ~ ~ ~, arpy bass2 drum notes can}"
```

Caso tenha interesse por "polimetria"é possível se aprofundar a partir deste link.

4.17 Deslocando tempo

Usando as funções > e < se desloca os padrões para frente ou para trás no tempo. Cada uma destas funções, define uma quantidade, em unidades de ciclo.

```
d1 $ (0.25 <~) $ sound "bd hh cp hh:10"
d1 $ (0.25 ~>) $ sound "bd hh cp hh:10"
```

O código acima desloca os padrões em um quarto de ciclo.

Você pode ouvir melhor este efeito de mudança ao aplicá-lo condicionalmente. Por exemplo, o código abaixo desloca o padrão a cada três ciclos:

```
d1 $ every 4 (0.25 <~) $ sound "bd hh cp hh:10"
d1 $ every 4 (0.25 ~>) $ sound "bd hh cp hh:10"
```

Desloque o quanto desejar os padrões:

```
d1 $ every 3 (0.0625 <~) $ sound "bd*2 cp*2 hh sn"
d1 $ every 3 (1000 ~>) $ sound "bd*2 cp*2 hh sn"
d1 $ every 3 (1000.125 ~>) $ sound "bd*2 cp*2 hh sn"
```

No entanto, no caso acima os ciclos são iguais. Haverá deslocamento nos casos com frações. Não há diferença entre ciclos com número inteiro. 1 ou 1000 ciclos gera resultados iguais.

É possível especificar um padrão para a quantidade de deslocamentos:

```
d1 $ "[0 0.25]/4" <~ (sound "bd*2 cp*2 hh sn")
```

4.18 Introduzindo Aleatoriedade

Tidal pode produzir padrões aleatórios de números inteiros e decimais. Além disso, pode introduzir aleatoriedade nos padrões, removendo eventos aleatórios.

4.18.1 Padrões Decimais Aleatórios

Use a função `rand` para criar um valor aleatório entre 0 e 1. Isto é útil para efeitos:

```
d1 $ sound "arpy*4" # pan (rand)
```

Como no caso do run e todos os padrões numéricos, os valores que o rand podem ser escalonados com range, o exemplo a seguir gera números aleatórios entre 0,25 e 0,75:

```
d1 $ sound "arpy*4" # pan (range 0.25 0.75 $ rand)
```

4.18.2 Padrões de números inteiros aleatórios

Use a função irand para criar um inteiro aleatório até um determinado número máximo. O uso mais comum do irand é produzir um padrão aleatório de índices de amostra (semelhante ao run):

```
d1 $ s "arpy*8" # n (irand 30)
```

O código acima escolhe aleatoriamente entre 30 amostras na pasta "arpy".

Detalhes cabeludos: rand e irand são na verdade padrões contínuos. Em termos práticos significa que eles têm detalhes infinitos - você pode tratá-los como pura informação! Como em todos os padrões, eles também são funções determinísticas, sem estado temporal, de modo que, se você recuperasse um padrão do mesmo ponto de tempo lógico, os mesmos números seriam produzidos. Além disso, se você usar um rand ou irand em dois lugares diferentes, você obteria o mesmo padrão 'aleatório' - se isto não for o que você quer, você pode simplesmente mudar ou diminuir um pouco o tempo para um deles, por exemplo, diminuir 0.3 rand .

4.18.3 Removendo ou "Degradando"eventos do Padrão

Tidal possui diferentes maneiras de remover aleatoriamente eventos de um padrão. Incluindo o símbolo ? se introduz uma aleatoriedade de 50% na execução de um evento a cada ciclo:

```
d1 $ sound "bd? sd? sd? sd?"
```

No código acima, todos sample têm 50% de chance de serem executados ou permanecerem em silêncio.

É possível adicionar o ponto de interrogação ? após a qualquer evento ou grupo em um padrão:

```
d1 $ sound "bd*16?"
d1 $ sound "bd sn? cp hh?"
d1 $ sound "[bd sn cp hh]?"
```

O símbolo ? é a abreviatura para a função degrade. As duas linhas abaixo são equivalentes:

```
d1 $ sound "bd*16?"
d1 $ degrade $ sound "bd*16"
```

Relacionado a degrade está a função degradarBy, onde é possível especificar a probabilidade (de 0 a 1) a qual os eventos serão removidos em um padrão:

```
d1 $ degradeBy 0.25 $ sound "bd*16"
```

Há também sometimesBy, que executa uma função baseada em uma probabilidade:

```
d1 $ sometimesBy 0.75 (# crush 4) $ sound "bd arpy sn ~"
```

O código acima tem uma probabilidade de 75% de executar o padrão de efeito bitcrush # crush 4 a cada evento do padrão sound.

Há outros pseudônimos para às sometimesBy:

```
sometimes = sometimesBy 0.5
often = sometimesBy 0.75
rarely = sometimesBy 0.25
almostNever = sometimesBy 0.1
almostAlways = sometimesBy 0.9
```

por exemplo

```
d1 $ rarely (# crush 4) $ sound "bd*8"
```

4.19 Criando Variação nos Padrões

É possível criar muitas variações cíclicas nos padrões através de uma lógica condicional em camadas:

```
d1 $ every 5 (|+| speed "0.5") $ every 4 (0.25 <~) $ every 3 (rev) $ s "bd sn arpy*2 cp" # speed "[1 1.25 0.75 -1.5]/3"
```

Além da função condicional `every` existe `whenmod`. `Whenmod` utiliza dois parâmetros; executa uma função quando o restante do número do loop corrente dividido pelo primeiro parâmetro é maior ou igual ao segundo parâmetro.

Execute o exemplo a seguir para entender como `whenmod` funciona. O código abaixo `whenmod 8 6 (rev)` executa 8 ciclos. Do ciclo 1 a 6 executará a função `sound` normalmente, nos ciclos restantes executará a função associada ao `whenmod`, ou seja, tocando em reverso `(rev)` os ciclos 7 e 8. Terminado os 8 ciclos, refaz o processo tocando normalmente até seis ciclos, e em reverso para dois restantes, e assim por diante:

```
d1 $ whenmod 8 6 (rev) $ sound "bd*2 arpy*2 cp hh*4"
```

4.20 Criando preenchimentos e usando a função `const`

Pense em um "preenchimento"como mudança para um padrão regular que acontece constantemente. Por exemplo, a cada 4 ciclos fazer "xyz", ou a cada 8 ciclos fazer "abc".

Já temos usado `every` e `whenmod` para criar preenchimento de padrões:

```
d1 $ every 8 (rev) $ every 4 (density 2) $ sound "bd hh sn cp"
d1 $ whenmod 16 14 (# speed "2") $ sound "bd arpy*2 cp bass2"
```

No entanto, caso queira substituir condicionalmente um padrão por um novo é possível usar a função `const` .Ela substitui completamente um padrão em execução.

Vamos começar com um exemplo trivial onde usamos `const` para substituir um padrão inteiro o tempo todo:

```
d1 $ const (sound "arpy*3") $ sound "bd sn cp hh"
```

No código acima, substituímos completamente o padrão "bd sn cp hh"por um padrão

"arpy". `const` especifica assim um novo padrão.

Podemos aplicar condicionalmente `const` usando `every` ou `whenmod`:

```
d1 $ whenmod 8 6 (const $ sound "arpy(3,8) bd*4") $ sound "bd sn bass2 sn"
d1 $ every 12 (const $ sound "bd*4 sn*2") $ sound "bd sn bass2 sn"
```

4.21 Compondo Padrões de Múltiplas Partes

Existem algumas maneiras de se compor novos padrões a partir de outros padrões. É possível concatenar ou "anexar"padrões em série, ou ainda "empilhá-los"e tocá-los juntos em paralelo.

4.21.1 Concatenação de padrões em série

A função `fastcat` serve para adicionar padrões um após o outro:

```
d1 $ fastcat [sound "bd sn:2" # vowel "[a o]/2",
              sound "casio casio:1 casio:2*2"
             ]
```

A função `fastcat` espreme os padrões no espaço de um. Quanto mais padrões você acrescentar à lista, mais rápido cada padrão será reproduzido para que todos os padrões possam se encaixar em um único ciclo.

```
d1 $ fastcat [sound "bd sn:2" # vowel "[a o]/2",
              sound "casio casio:1 casio:2*2",
              sound "drum drum:2 drum:3 drum:4*2"
             ]
```

`cat` (também conhecido como slowcat), manterá a velocidade original de reprodução dos padrões:

```
d1 $ cat [sound "bd sn:2" # vowel "[a o]/2",
          sound "casio casio:1 casio:2*2",
          sound "drum drum:2 drum:3 drum:4*2"
         ]
```

`cat` é uma ótima maneira de criar seqüência linear de padrões (seqüência de seqüências), dando melhor forma a múltiplos padrões.

Há também o `randcat` que toca um padrão aleatório da lista.

4.21.2 Tocando padrões em paralelo

A função de `stack` (empilhar) pega uma lista de padrões e os combina em um novo padrão, reproduzindo todos os padrões da lista simultaneamente.

```
d1 $ stack [
 sound "bd bd*2",
 sound "hh*2 [sn cp] cp future*4",
 sound (samples "arpy*8" (run 16))
]
```

Isto é útil caso queira aplicar funções ou efeitos em múltiplos padrões simultaneamente:

```
d1 $ every 4 (slow 2) $ whenmod 5 3 (# speed "0.75 1.5") $ stack [
 sound "bd bd*2",
 sound "hh*2 [sn cp] cp future*4",
 sound (samples "arpy*8" (run 16))
] # speed "[[1 0.8], [1.5 2]*2]/3"
```

4.22 Truncando samples com `cut`

Até agora, todos os nossos exemplos têm usado sample curtos. Caso tenha experimentado algum sample longo, é possível perceber que existe uma sobreposição indesejada. Com o efeito `cut` é possível impedi-lo de tocar quando uma nova amostra é acionada.

Considere o seguinte exemplo que utiliza o padrão `sounds "arpy"`, tocado em velocidade reduzida, que acaba gerando sobreposições a cada novo ciclo:

```
d1 $ sound (samples "arpy*8" (run 8)) # speed "0.25"
```

Pode-se estancar este vazamento usando `cut` e atribuindo ao padrão um grupo de corte "1":

```
d1 $ sound (samples "arpy*8" (run 8)) # speed "0.25" # cut "1"
```

Sem sobreposições!

É possível usar qualquer número para o grupo de corte.

Os grupos de corte são globais, para o processo Tidal, portanto, se existir duas conexões Dirt, use dois valores para grupos de cortes diferentes. Isso garantirá que os padrões não engasgue um ao outro:

```
d1 $ sound (samples "arpy*8" (run 8)) # speed "0.25" # cut "1"
d2 $ sound (samples "bass2*6" (run 6)) # speed "0.5" # cut "2"
```

Isso também funciona para com padrões múltiplos de `stack`

```
d1 $ stack [
  sound (samples "arpy*8" (run 8)) # speed "0.25" # cut "1",
  sound (samples "bass2*6" (run 6)) # speed "0.5" # cut "2" ]
```

4.23 Transições entre Padrões

A mudança do padrão em um canal tem efeito (quase) imediato. Isto pode não ser desejável, especialmente em performances ao vivo!

Por isso, Tidal permite transições dinâmicas ou mesmo substituindo o padrão atual.

Assim, uma vez que tenhamos algo tocando em `d1`, podemos usar o mesmo número de canal (1), passado por uma agradável transição:

```
d1 $ sound (samples "hc*8" (iter 4 $ run 4))
anticipate 1 $ sound (samples "bd(3,8)" (run 3))
```

Para fazer a transição a partir daqui, basta mudar o padrão e, neste caso, também mudar a função de transição:

```
xfadeIn 1 16 $ sound "bd(5,8)"
```

O exemplo acima se desfaz em 16 ciclos desde o padrão anterior até o novo padrão dado. Além de `antecipate` e `xfadeIn`, existem muitas outras funções de transição, por exemplo, algumas que o forçarão a continuar mudando seus padrões para evitar performances repetitivas.

Aqui alguns fade out.

```
xfadeIn 1 16  silence
xfade silence
```

4.24 Samples

Caso esteja usando SuperDirt, todas os sample podem ser encontradas na pasta Dirt-Samples. É possível abrir a pasta executando Quarks.gui no SuperCollider, clicando em "Dirt-Samples"e depois em "abrir pasta". Caso esteja usando Dirt clássico, procure na subpasta sample. Aqui estão alguns sample para experimentar:

```
flick sid can metal future gabba sn mouth co gretsch mt arp h cp cr newnotes
bass crow hc tabla bass0 hh bass1 bass2 oc bass3 ho odx diphone2 house off ht
tink perc bd industrial pluck trump printshort jazz voodoo birds3 procshort
blip drum jvbass psr wobble drumtraks koy rave bottle kurt latibro rm sax
lighter lt arpy feel less stab ul
```

Cada mome acima é uma pasta que contém um ou mais arquivos wav. Por exemplo, quando executamos bd:1 em uma seqüência, se está selecionando o segundo arquivo wav na pasta bd. Caso execute a nona amostra de uma pasta que tem apenas sete sample, Tidal tocará o segundo sample. Lembrando que a contagem inicia-se com 0 como primeiro sample.

Caso queira adicionar sua própria biblioteca de sample, basta criar uma nova pasta na pasta sample, e incluir os arquivos wav na pasta.

4.25 Sintetizadores

Para que esta seção funcione, é necessário ter instalado o SuperCollider sc3-plugins. Instale a última versão a partir do git, ou se estiver usando Linux, pode encontrá-la em seu gerenciador de pacotes. No Fedora o pacote é chamado de supercollider-sc3-plugins.

SuperDirt foi criado com SuperCollider é um fantástico mecanismo de síntese e linguagem com imensas possibilidades sônicas. É possível acionar sintetizadores personalizados no SuperCollider a partir de TidalCycles da mesma forma que se tocam sample. SuperCollider tem um processamento de síntese fantástico. Experimente os exemplos a seguir.

```
d1 $ midinote "60 62*2" # s "supersaw"
```

O exemplo acima executa as notas 60 e 62 da escala MIDI, usando o parâmetro midinote. É possível também tocar as notas pelo seu nome, usando n:

```
d1 $ n "c5 d5*2" # s "supersaw"
```

Para semitons acrescenta-se os sufixos "f"ou "s"(f = bemol ou s = sustenido) à nota em questão.

```
d1 $ n "<[a5,cs5,e5,g5]*3 [d5,fs5,g5,c5]>" # s "supersquare" # gain "0.7"
```

Above is a two chord progression A7 D7. Notice cs5 and fs5 as `C#5` and `F#5`, respectively.

```
d2 $ every 4 (rev) $ n "<[g5 df5 e5 a5] [gf5 d5 c5 g5]*3>" # s "supersaw"
```

Agora os mesmos acordes (A7 D7) desta vez tocados como arpejos ascendentes e descendentes e cs5 escritos como df5 e fs5 como gf5. Toque os dois exemplos juntos para ficar mais divertido!

É possível especificar números de notas com n, onde 0 é a nota dó central (em vez de 60 com midinote).

```
d1 $ n "0 5" # s "supersaw"
```

A duração padrão do `sutain` é um pouco longo para que os sons se sobreponham, é possível ajustá-lo usando o parâmetro `sustain`.

```
d1 $ n "c5 d5*2" # s "supersaw" # sustain "0.4 0.2"
```

Muitos exemplos de sintetizadores podem ser encontrados no arquivo default-synths-extra.scd na pasta SuperDirt/library ou no arquivo default-synths.scd e tutorial-synths.scd na pasta SuperDirt/synths. Esles incluem:

- serie de tutoriais: tutorial1, tutorial2, tutorial3, tutorial4, tutorial5
- exemplos de modulação com o cursor ou entrada de som: pmsin, in, inr
- síntese por modelagem física: supermandolin, supergong, superpiano, superhex
- kit básico de sintetizado de bateria: superkick, superhat, supersnare, superclap, super808
- quatro sintetizadores estilo analógico: supersquare, supersaw, superpwm, super-comparator
- dois sintetizados estilo digital: superchip, supernoise

Para encontrar a pasta SuperDirt, simplesmente execute Quarks.folder no SuperCollider. A localização completa da pasta deve aparecer no terminal "postwindow"(que geralmente fica na parte inferior direita).

Muitos dos sintetizadores acima aceitam parâmetros adicionais do Tidal ou interpretam os parâmetros usuais de uma maneira ligeiramente diferente. Para documentação completa, veja default-synths.scd. De qualquer forma aqui estão alguns exemplos para experimentar:

```
d1 $ jux (# accelerate "-0.1") $ s "supermandolin*8" # midinote "[80!6 78]/8"
 # sustain "1 0.25 2 1"
```

```
d1 $ midinote (slow 2 $ (run 8) * 7 + 50) # s "supergong" # decay "[1 0.2]/4"
 # voice "[0.5 0]/8" # sustain (slow 16 $ range 5 0.5 $ saw1)
```

```
d1 $ sound "superhat:0*8" # sustain "0.125!6 1.2" # accelerate "[0.5 -0.5]/4"
```

```
d1 $ s "super808 supersnare" # n (irand 5)
 # voice "0.2" # decay "[2 0.5]/4" # accelerate "-0.1" # sustain "0.5" # speed "[0.5 2]/4"
```

```
d1 $ n (slow 8 "[[c5 e5 g5 c6]*4 [b4 e5 g5 b5]*4]") # s "superpiano"
 # velocity "[1.20 0.9 0.8 1]"
```

```
d1 $ n (slow 8 $ "[[c4,e4,g4,c5]*4 [e4,g4,b5,e5]*4]" + "<12 7>") # s "superpiano"
 # velocity (slow 8 $ range 0.8 1.1 sine) # sustain "8"
```

```
d1 $ n "[c2 e3 g4 c5 c4 c3]/3" # s "[superpwm supersaw supersquare]/24" # sustain "0.5"
 # voice "0.9" # semitone "7.9" # resonance "0.3" # lfo "3" # pitch1 "0.5" # speed "0.25 1"
```

```
d1 $ every 16 (density 24 . (|+| midinote "24") . (# sustain "0.3") . (# attack "0.05"))
 $ s "supercomparator/4" # midinote ((irand 24) + 24)
 # sustain "8" # attack "0.5" # hold "4" # release "4"
 # voice "0.5" # resonance "0.9" # lfo "1" # speed "30" # pitch1 "4"
```

```
d1 $ n "[c2 e3 g4 c5 c4 c3]*4/3" # s "superchip" # sustain "0.1"
 # pitch2 "[1.2 1.5 2 3]" # pitch3 "[1.44 2.25 4 9]"
 # voice (slow 4 "0 0.25 0.5 0.75") # slide "[0 0.1]/8" # speed "-4"

d2 $ every 4 (echo (negate 3/32)) $ n "c5*4" # s "supernoise"
 # accelerate "-2" # speed "1" # sustain "0.1 ! ! 1" # voice "0.0"
```

```
d1 $ s "supernoise/8" # midinote ((irand 10) + 30) # sustain "8"
 # accelerate "0.5" # voice "0.5" # pitch1 "0.15" # slide "-0.5" # resonance "0.7"
 # attack "1" # release "20" # room "0.9" # size "0.9" # orbit "1"
```

Isso tudo ainda é bastante recente e segue em desenvolvimento, mas é possível saber mais sobre como modificar e adicionar seus próprios sintetizadores ao SuperDirt através do repositório github.

Referência Bibliográfica

[1] Nick Collins and Alex McLean. Algorave: Live performance of algorithmic electronic dance music. In *Proceedings of the International Conference on New Interfaces for Musical Expression*, pages 355–358, 2014.

[2] Alex McLean. Making programming languages to dance to: live coding with tidal. In *Proceedings of the 2nd ACM SIGPLAN international workshop on Functional art, music, modeling & design*, pages 63–70, 2014.

[3] Alex McLean and Geoff Cox. *Speaking code: Coding as aesthetic and political expression*. MIT Press, 2013.

[4] Alex McLean and Roger T Dean. *The Oxford handbook of algorithmic music*. Oxford University Press, 2018.