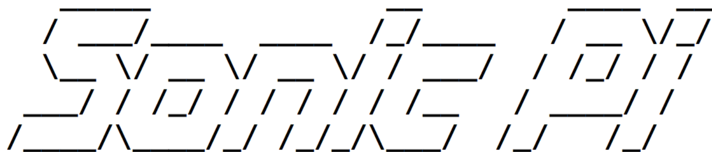
tutorial pt-br



música como :código

código como :arte

Sam Aaron

2015 - cc by-sa 4.0 - Sam Aaron

2017 - cc by-sa 4.0 - *Sonic Pi tutorial pt-br: música como código - código como arte.* Trad. Giuliano Obici. Rio de Janeiro: Pámphónos.

Sam Aaron

Sonic Pi

tutorial pt-br

música como **:código**

código como **:arte**

tradução: Giuliano Obici

atualizado em 29/12/2017

Pámphónos - Rio de Janeiro

Conteúdo

1 **Apresentação** 3
- 1.1 Live Coding 4
- 1.2 Interface 8
- 1.3 Aprender Tocando 11

2 **Sintetizador** 13
- 2.1 Primeiros Beeps 14
- 2.2 Opções de Síntese 17
- 2.3 Mudando Sintetizadores 20
- 2.4 Duração com Envelopes 22

3 **Sample** 31
- 3.1 Disparar Sample 31
- 3.2 Parâmetros de Sample 33
- 3.3 Time Stretch: esticando e contraindo o tempo dos samples 34
- 3.4 Envelopando Samples 38
- 3.5 Recortando Sample 41
- 3.6 Sample Externo 44
- 3.7 Pasta de Sample 45

4 **Randomização** 53

5 **Programando Estruturas** 59
- 5.1 Blocos 59
- 5.2 Iteração e Loops 60
- 5.3 Condicional 64
- 5.4 Threads 66
- 5.5 Funções 72
- 5.6 Variáveis 75
- 5.7 Sincronização de Thread 80

6 Efeitos 85
6.1 Adicionando Efeitos 86
6.2 Efeitos na Prática 90

7 Controle 93
7.1 Controlando Synths em Tempo Real 93
7.2 Controlando Efeitos 95
7.3 Opção Slide 95

8 Estruturas de dados 99
8.1 Listas 100
8.2 Acordes 102
8.3 Rings 105
8.4 Encadeamento de Ring 107

9 Live Coding 111
9.1 Fundamentos do Live Controling 111
9.2 Live Loops 114
9.3 Vários Live Loops 116
9.4 *Ticking* 118

10 Estado Temporal 123
10.1 *Set* e *Get* 124
10.2 *Sync* 128
10.3 Correspondência de Padrões 129

11 MIDI 135
11.1 MIDI In 136
11.2 MIDI out 140

12 OSC 145
12.1 Recebendo OSC 146
12.2 Enviando OSC 148

13 Áudio Multicanal 151
13.1 Entrada de Áudio 151
13.2 Live Audio 154
13.3 Saída de Som 157

14 Conclusões 161

Appendices 163

A Artigos MagicPi 165
A.1 Cinco Dicas Fundamentais 167

A.2 *Live Coding* 172
A.3 Batidas codificadas 178
A.4 Rifes de Sintetisadores 183
A.5 Acid Bass 189
A.6 Música e Minecraft 195

B Conhecimentos Fundamentais **201**
B.1 Usando Atalhos 202
B.2 Lista de Atalhos 205
B.3 Compartilhando 208
B.4 Performance 210

Da Tradução

Este texto é uma tradução do tutorial em inglês que acompanha o programa *Sonic Pi*.

Quando iniciei a tradução para o português brasileiro, meu objetivo era oferecer um rascunho simples aos alunos capaz de introduzir noções básicas de programação e música através do Sonic Pi. O entusiasmo e interesse dos alunos me levaram a terminar esta primeira versão do tutorial.

Como toda primeira versão, ela ainda necessita ser revista e aprimorada.[i] Por isso, se você tiver algum comentário que possa contribuir para melhorar este material por favor entre em contato.[ii]

Feito tais considerações vamos para a apresentação do Sonic Pi.

---

[i] A versão atualizada deste tutorial em pt-br se encontra neste link.
[ii] email [a] giulianobici.com

Capítulo 1

Apresentação

Sonic Pi é um programa feito para uma nova geração de músicos, simples de aprender, poderoso o suficiente para apresentações ao vivo e gratuito para *download*.[1] Ele é uma ótima maneira de aprender programação e ao mesmo tempo criar música. Sua simplicidade o torna um programa inteligente, eficiente e fácil de usar. É voltado tanto para jovens quanto para músicos experientes, curiosos, *experts* ou interessados em criar música através de códigos.

Como veremos, *Sonic Pi* é uma ótima forma de introduzir noções de programação, música, síntese, design sonoro, composição e *live code* além de servir como ferramenta para o ensino de conceitos de computação.[i]

Sonic Pi foi desenvolvido para promover o ensino de computação e música. O programa utiliza a linguagem Ruby e foi criado por Sam Aaron professor da University of Cambridge em colaboração com a

[i] *Sonic Pi* apresenta de maneira simples conceitos básicos de computação como seqüenciamento, iteração e seleção.[2]

Fundação Raspberry Pi – cujo objetivo é oferecer informática básica às escolas utilizando um mini computador de preço super acessível chamado *Raspberry Pi*[3].

Este tutorial está dividido em secções e agrupadas por categorias. Embora esteja escrito para ter uma evolução de aprendizagem fácil, do início ao fim, é esperado que você siga sua curiosidade, portanto fique à vontade para pular secções. Uma dica para aprender programação em *Sonic Pi* é observar sessões de *live coding*. Sam Aarom criador do *Sonic Pi* tem diferentes canais na Internet onde faz transmissões e demonstrações de *live coding* ao vivo.[ii]

Feito esta breve apresentação é hora de começar.

1.1 Live Coding

Um dos aspectos marcantes do *Sonic Pi* é poder escrever e modificar o código em tempo real, literalmente programar ao vivo (*live coding*) para fazer música. Diferente de apertar o *play* de um arquivo sonoro, o *live code* se assemelha ao ato de tocar e improvisar ao vivo. Após alguma prática qualquer pessoa consegue tocar e improvisar com *Sonic Pi* no palco.

Ao tocar ao vivo com *Sonic Pi* o computador passa a ser um instrumento musical que tem particularidades próprias. Assim como tocar um instrumento envolve conhecer suas características e potencialidades, no caso do computador a programação abre perspectivas próprias para a criação musical e o exercício à experimentação. Sendo assim, durante o percurso busque explorar lógicas, códigos e proces-

[ii] Alguns dos links a seguir podem ajudar a entender e aprender mais sobre o programa: Twitter Sam Aron, livecoding, Facebook Live Coding Performances,YouTube, Gitter Chat, Raspberry Pi, Google Groups, Twitter Sonic Pi, Facebook SonicPi, Vimeo Group, SoundCloud Playlist e oGitHub para os desenvolvedores interessados no código fonte.

samentos. Não esqueça: mantenha a mente e os ouvidos abertos.

Antes de entrar nos detalhes de como o *Sonic Pi* funciona, faremos uma breve experiência de programação ao vivo (*live coding*). Não se preocupe em entender tudo neste momento.

Caso não tenha ainda instalado o *Sonic Pi*, faça o download. Instale e abra o programa.

Agora sim.

Vamos colocar a mão na massa.

Live Loop

Copie o código (texto colorido) abaixo para o *buffer* (janela branca onde está escrito # *Welcome to Sonic Pi*) no *Sonic Pi*.:

```
live_loop :flibble do
  sample :bd_haus, rate: 1
  sleep 0.5
end
```

Agora, pressione o botão "**Run** ". Você ouvirá uma rápida batida de bumbo. Se não ouvir nada. Confira a configuração do código. Pode acontecer das linhas serem alteradas ao copiar e colar. Caso isso tenha ocorrido corrija e pressione novamente o botão "**Run** ".

Se quiser parar o som basta clicar no botão "**Stop**". Mas não faça isso ainda. Espere mais um pouco e siga os seguintes passos:

1. Certifique se de que o som do bumbo ainda toca.
2. Mude o valor do **sleep** de **0.5** para **1** por exemplo.
3. Pressione o botão **Run** novamente.
4. Note que a velocidade do bumbo mudou.

Lembre deste momento! Você acabou de descobrir o que é fazer *live code* programação ao vivo com o *Sonic Pi*.

Ok, isso foi simples. Vamos incrementar um pouco. Antes de "**sample :bd_haus** "adicione a linha "**sample :ambi_choir, rate: 0.3**". O código ficará parecido com o seguinte:

```
live_loop :flibble do
  sample :ambi_choir, rate: 0.3
  sample :bd_haus, rate: 1
  sleep 1
end
```

Mudou alguma coisa? Não? Lembre-se que após alterar o código, é necessário pressionar o botão **Run** que ativará o novo código no próximo ciclo (*loop*). Outra forma de iniciar e parar o som é utilizar as teclas de atalhos. **Run** = *Ctrl+R*, *Alt+R* ou *Ctrl+Enter*. **Stop** = *Ctrl+S* ou *Alt+S* dependendo do sistema operacional.

Fique à vontade para alterar os valores do código. Experimente mudar parâmetros como o **rate** por exemplo. Preste atenção ao que acontece utilizando valores altos, baixos ou negativos. Tente entender auditivamente o que ocorre quando valores são alterados. Mude o parâmetro **rate** do sample ":ambi_choir"(por exemplo para **0.29**). E se alterarmos o parâmetro **sleep** para um valor muito pequeno como **0.1**? Escute atentamente. Se o computador parar ou gerar erros. Fique tranquilo. Isso pode acontecer. Basta alterar novamente o parâmetro **sleep** para um valor maior e pressionar **Run** novamente.

Experimente comentar uma das linhas **sample** adicionando o caractere "#"no início e pressione **Run**:

```
live_loop :flibble do
  sample :ambi_choir, rate: 0.3
#  sample :bd_haus, rate: 1
  sleep 1
end
```

Note que a linha com "# "será ignorada e não se ouve mais. Isto porque o caractere "# "transforma a linha em um comentário, passando a não ser executado no próximo ciclo. No *Sonic Pi* como em outros programas comentários podem ser utilizados para habilitar ou desabilitar ações de um código. Musicalmente falando isso é uma forma simples de adicionar ou remover eventos e/ou processos sonoros durante a performance.

Vamos para um outro exemplo. Copie o código abaixo para um *buffer* vazio. De novo não é preciso compreendê-lo totalmente.

```
live_loop :guit do
  with_fx :echo, mix: 0.3, phase: 0.25 do
    sample :guit_em9, rate: 0.5
  end
#  sample :guit_em9, rate: -0.5
  sleep 8
end

live_loop :boom do
  with_fx :reverb, room: 1 do
    sample :bd_boom, amp: 10, rate: 1
  end
  sleep 8
end
```

Existem dois ciclos (**live_loop**) que tocam ao mesmo tempo. Rode esse novo exemplo e tente alterá-lo com as seguintes dicas:

1. Experimente mudar os valores do **rate** em azul para ouvir o que muda.
2. Experimente mudar o valor de **sleep** e note que os ciclos tocam em velocidades diferentes.
3. Experimente remover o comentário (símbolo "#") da linha **sample** e surgirá uma guitarra tocado ao contrário.
4. Experimente mudar um dos valores de **mix:** para números entre **0** (não entra no mix) e **1** (entra na mix com efeito completo).

Lembre-se de clicar no botão **Run** para ouvir as mudanças da próxima vez que o ciclo iniciar. Se o código for invalidado, não te preocupe – clique em **Stop**, apague o código e cole-o novamente. Errando se aprende mais rápido.

Continue alterando o código com os ouvidos atentos. O que mais seria possível fazer a partir deste exemplo? Como seria tocá-lo ao vivo? Gostou do resultado e quer grava-lo? Tem o botão **Rec** ao lado do botão **Run** que salvará sua performance em arquivo de áudio. Espero que esses minutos tenham despertado seu interesse em descobrir e aprender mais sobre como *Sonic Pi* funciona. Vamos explorar um pouco mais? [iii]

1.2 Interface

A interface do *Sonic Pi* possui a seguinte estrutura.

A - Controles de Execução
B - Controles do Editor
C - Informações e Ajuda
D - Editor de código
E - Preferências
F - Visualizador do Log
G - Sistema de Ajuda
H - Osciloscópio

[iii] Boa parte da documentação do programa está traduzida. É possível seguir o tutorial diretamente pelo programa *Sonic Pi*. Vale lembrar que o tutorial do *Sonic Pi* é apresentado conforme a língua do sistema operacional de seu computador. Se quiser alterar é necessário mudar as configurações do sistema.

Fig. 1.1: Interface do *Sonic Pi*

A - Controles de Execução

Os botões de cor rosa são os controles principais para tocar e executar, parar e gravar. O botão **Run** serve para reproduzir o código no editor, **Stop** para parar todos os processos, **Record** para gravar o som que estiver tocando em um arquivo .WAV, **Save** para salvar e o **Load** para abrir um arquivo.

B - Controles do Editor

Os botões de cor laranja **Size +** e **Size -** tornam o texto maior ou menor.

C - Informações, Ajuda e Opções

Os botões de cor azul acessam informações, ajuda e opções. **Info** abre a janela de informação sobre o *Sonic Pi* – a equipe, histórico de atualizações, contribuidores e a comunidade. O botão **Help** abre e fecha o sistema de ajuda e o botão **Opções** abre e fecha a janela das opções que te permitem controlar alguns parâmetros básicos do sistema.

D - Editor de Código

Esta é a área onde se escreve o seu código para compor/tocar sua música. Trata-se de um editor de texto simples onde se pode escrever código, apagá-lo, ou modificá-lo. O editor irá mudar de cores automaticamente conforme as propriedades do código. Isto é bastante útil, por exemplo, para saber as funções e características de um código.

E - Preferências

As preferências do *Sonic Pi* podem ser acessadas pelo botão **Prefs**. Entre as opções estão forçar saídad de audio mono **mono mode**, inverter os canais de **stereo**, activar e desativar **logs** além de um *slider* para o volume e selector de audio no Raspberry Pi.

F - Visualizador do Log

Quando rodar o código, informação sobre o que o programa está fazendo aparecerão no vizualizador do **log**. Por princípio, a cada som executado aparece no **log** uma mensagem no exato momento de sua execução. Isto é útil para *debugging* e perceber como seu código está

funcionando.

G - Sistema de Ajuda

Uma das partes mais relevantes da interface do *Sonic Pi*, é o sistema de ajuda que aparece por baixo da janela. Isto pode ser exibido através do botão azul **Help**. O sistema contém informação de ajuda acerca de todos os aspectos do *Sonic Pi*, incluindo este tutorial, e uma lista dos sintetizadores e samples disponíveis, assim como efeitos e uma lista completa de todas as funções que o *Sonic Pi* disponibiliza para criação musical.

H - Osciloscópio

O osciloscópio permite visualizar a forma de onda do sinal de áudio em execução. É possível observar facilmente a forma de uma onda triangular e constatar que o sinal se parece uma forma de onda triangular e que um beep básico tem uma curva de sinosoidal. É também possível constatar a diferença entre um som alto e um som baixo através do tamanho das linhas. Existem três osciloscópios com os quais é possível trabalhar-se - o pré-definido é uma combinação do osciloscópio para o canal esquerdo e direito. Existe um osciloscópio estéreo que desenha um osciloscópio com linhas separadas para cada canal. Finalmente, existe um osciloscópio com curva *Lissajous* que permite ver a relação de fase entre o canal esquerdo e direito e permite desenhar imagens bonitas a partir da forma do som.

1.3 Aprender Tocando

O *Sonic Pi* incentiva o aprendizado tanto de computação como de música ao tocar e experimentar. A coisa mais importante é se diver-

tir, e ao perceber terá aprendido, como se codifica, compõe e toca.

Não Existem Erros

Uma das coisas que aprendi ao longo dos meus anos de *live coding* é que não existem erros, apenas oportunidades. Isto é algo que se ouve em relação ao jazz mas também funciona com *live coding*. Independentemente do quão experiente você for, de principiante em *algoraver*[iv] a um *expert* em códigos, quando pressionar o botão para validar o código, o resultado é sempre uma surpresa. Pode soar extremamente bem, e nesse caso salve e volte a usar o código, ou pode soar um tanto desajustado. Não interessa o que tenha acontecido, o que importa é o que faz com ele a seguir, mudando-o e/ou transformando-o em algo singular. As pessoas na plateia irão gostar.

Começe Simples

Quando estiver escrevendo código, será tentador querer fazer coisas espetaculares. No entanto, guarde essas ideias como um objetivo a ser alcançado mais tarde. Por agora, pense no mais simples, divertido e recompensador de escrever. Isso te ajudará a chegar naquilo que tens em mente. Uma vez que tiver uma ideia, tente construí-la passo a passo partindo do mais simples. Brinque e explore os elementos mais simples dela e verás que novas ideias surgirão. Rapidamente estará entretido e irá progredir se divertindo.

Lembre-se de partilhar seu trabalho!

OK, chega de introduções. Vamos começar!

[iv] Algorave é a abreviação de "rave com algorítmico", ou seja um evento em que as pessoas dançam a música gerada a partir de algoritmos, muitas vezes usando técnicas de codificação ao vivo.

Capítulo 2

Sintetizador

Vamos escrever e ouvir o que podem fazer os códigos. Para isso começaremos manipulando alguns sintetizadores (*synth*).

Você já deve ter visto sintetizadores modulares analógicos, aqueles parecidos com centrais telefônicas ou computadores primitivos, com muitos fios. Os sintetizadores analógicos podem ser complexos de manipular. Por sua vez no *Sonic Pi* os sintetizadores guardam aquela elegância sonora dos sintetizadores analógicos proém são simples de serem utilizados. Apesar de utilizar sintetisadores no Sonic Pi é possível ir longe com eles conseguindo alcançar resultados sonoros bem sofisticados.

Não se deixe enganar pela simplicidade imediata da interface do *Sonic Pi*. Você pode se aprofundar em manipulações se este for seu interesse.

2.1 Primeiros Beeps

Vejamos o seguinte código:

```
play 70
```

Tudo começa assim. Copie e cole no editor o código acima e depois pressione o botão **Run**... Ouviu o beep? Pressione outra vez. E outra vez. E mais uma vez... Agora mude o número.

```
play 75
```

Ouviu a diferença? Experimente mais uma vez com um número menor:

```
play 60
```

Percebeu a diferença? Os números baixos produzem *beeps* mais graves e números acima produzem *beeps* mais agudos. Tal como num piano, as teclas na parte mais baixa do piano (lado esquerdo) tocam notas mais graves e as teclas na parte mais alta do piano (lado direito) tocam notas mais agudas. De fato, os números estão relacionados com as notas no piano e o protocolo MIDI.[i] "play 47"na realidade significa tocar a 47ª nota do piano. O que significa que "play 48"está uma nota acima (nota seguinte à direita). Acontece que a nota Dó na 4ª oitava é o número 60. Vamos lá, toque-a: "play 60"

Se nada disto fizer sentido para você, não se preocupe. O que interessa no momento é entender a relação de que números pequenos fazem sons graves e números grandes fazem sons agudos.

---

[i]MIDI (Musical Instrument Digital Interface) é um padrão de interconexão física (interface digital, protocolo e conexão) criado em 1982 por um grupo de fabricantes de sintetizadores japoneses e americanos, para facilitar a comunicação em tempo real entre instrumentos musicais eletrônicos, computadores e dispositivos relacionados.

Acordes

Se tocar uma nota já é divertido, tocar várias ao mesmo tempo pode ser ainda melhor. Experimente:

```
play 72
play 75
play 79
```

Quando se tem vários **play**s, eles tocam ao mesmo tempo. Experimente outros números. Quais soam mal? Explore e descubra.

Melodia

Como seria tocar uma nota depois da outra e não ao mesmo tempo? Para, é preciso um intervalo entre elas dado pelo **sleep**:

```
play 72
sleep 1
play 75
sleep 1
play 79
```

Que bom, um pequeno arpejo. Então o que significa o **1** em **sleep 1**? Bem, é a duração da pausa. Na realidade significa pausar durante um tempo, podemos pensar como uma pausa de 1 segundo. Então e se quisessemos que o nosso arpejo tocasse mais rápido? Bem, temos de diminuir o valor da pausa. E que tal metade, **0.5**:

```
play 72
sleep 0.5
play 75
sleep 0.5
play 79
```

Notou a diferença? Agora, tente usar pausas e notas diferentes.

Teste valores intermédios como **play 52.3** e **play 52.63**. Experimente e se divirta.

Notação por Cifras

Para aqueles que quiserem escrever uma melodia baseado nas notas musicais e não no protocolo MIDI o *Sonic Pi* utiliza também a notação por cifras:

a = lá	play :a
	sleep 0.2
b = si	play b
	sleep 0.2
c = dó	play :c
	sleep 0.2
d = ré	play :d
	sleep 0.2
e = mi	play :e
	sleep 0.2
f = fá	play :f
	sleep 0.2
g = sol	play :g

Lembre de pôr dois pontos **":"** antes do nome da nota para que fique rosa. Também é possível especificar a oitava adicionando um número na frente da nota:

```
play :c3
sleep 0.5
play :d3
sleep 0.5
play :e4
```

Se quiser adicionar sustenido, de modo a incrementar meio tom, adiciona um 's' depois da nota (**play :Fs3**) e se quiseres adicionar um bemol, adiciona um 'b' (**play :Eb3**).

Chegou a hora de experimentar algumas melodias.

2.2 Opções de Síntese

Amplitude e Panorâmica

Além de controlar nota ou sample, o *Sonic Pi* dispõe de um conjunto de opções para criar e controlar os sons. Vamos apresentar apenas alguns, a documentação detalhada está no sistema de ajuda. Veremos dois dos parâmetros mais utilizados: **amplitude** e **pan** (panorâmica). Primeiro, vamos ver quais são as nossas opções.

Opções

Os sintetizadores de *Sonic Pi* suportam opções (opts), que são parâmetros de controle usados no **play** que modificam aspectos do som. Cada sintetizador tem suas próprias *opts*. No entanto, existem conjuntos de *opts* gerais partilhados por muitos **synth**s tal como **amp:** e opts **envelope** que veremos adiante.

Opts têm duas partes principais, o seu nome e o seu valor. Por exemplo, você pode ter uma *opt* chamada 'queijo' e atribuir o valor '1'.

Opts são acionados pelo **play** ao usar uma virgula (',') e depois o nome da opt, como 'amp:' (não te esqueça dos dois pontos ':') e depois do espaço e o valor. Por exemplo:

```
play 50, queijo: 1
```

Note que 'queijo:' não é um parâmetro para play, só estamos exemplificando. É possível aplicar várias *opts* separando-as por vírgulas:

```
play 50, queijo: 1, feijões: 0.5
```

A ordem das *opts* não altera o resultado, por isso o seguinte exemplo é idêntico:

```
play 50,  feijões: 0.5, queijo: 1
```

Opts que não são reconhecidas pelo sintetizado serão simplesmente ignoradas (como ‘queijo‘ e ‘feijões‘, que são claramente nomes fictícios para *opts*!)

Se usar a mesma *opt* mais de uma vez com valores diferentes, será a última que valerá. No exemplo abaixo, ’feijões:’ terá o valor ‘2‘ em vez de ‘0.5‘:

```
play 50, feijões: 0.5, ovos: 0.1, feijões: 2
```

Muitas coisas no *Sonic Pi* aceitam *opts*, portanto sugiro perder um tempinho, aprender como usá-las e estarás preparado! Vamos tocar com a primeira *opt*: **amp:**.

Amplitude

A amplitude é a representação do volume de um som. Uma amplitude alta produz um som forte e uma baixa amplitude produz um som silencioso. Uma amplitude de 0 é silenciosa (você não ouvirá nada) enquanto uma amplitude de 1 é o volume normal. Você pode até aumentar a amplitude para 2, 10, 100. No entanto, deve-se notar que, quando a amplitude geral de todos os sons ficar muito alta, *Sonic Pi* usa o que é chamado de um compressor para balancear todos e garantir que não estejam demasiado alto para seus ouvidos. Isso pode gerar sonoridades ruidosas e estranhas. Então tente usar amplitudes baixas, isto é, na faixa de 0 a 0,5 para evitar a compressão.

Aumentando o Volume

Para mudar a amplitude de um som, use a *opt* **amp:**. Por exemplo, para tocar com metade da amplitude use ‘0.5‘:

```
play 60, amp: 0.5
```

Para tocar o dobro da amplitude:

```
play 60, amp: 2
```

A opt ‘amp:‘ só modifica a amplitude do ‘play‘ com que está associada. Então, neste exemplo, o primeiro ‘play‘ está pela metade e o segundo está normal (‘1‘):

```
play 60, amp: 0.5
sleep 0.5
play 65
```

Claro, é possível usar diferentes valores de ‘amp:‘ para cada ‘play‘:

```
play 50, amp: 0.1
sleep 0.25
play 55, amp: 0.2
sleep 0.25
play 57, amp: 0.4
sleep 0.25
play 62, amp: 1
```

Pan

Outra opt é a ‘pan:‘, que controla a panorâmica de um som stereo. Usar *panning* para a esquerda num som significa que só o vais ouvir pela caixa esquerda, e usar *panning* para a direita significa que que só ouvirá na caixa da direita. Como valores, usamos ‘-1‘ para representar a esquerda, ‘0 ‘ para o centro e ‘1‘ para a direita, em stereo. Claro, estamos livres para usar qualquer valor entre ‘-1‘ e ‘1‘ para controlar a posição exata do som.

Vamos tocar um beep na caixa da esquerda:

```
play 60, pan: -1
```

Agora, vamos tocar um beep na direita:

```
play 60, pan: 1
```

Finalmente vamos tocar um beep no centro (padrão):

```
play 60, pan: 0
```

Agora, divirta-se mudando amplitude e pan dos sons!

2.3 Mudando Sintetizadores

O Sonic Pi tem um conjunto de instrumentos chamados synths (abreviatura para sintetizadores). Enquanto que os samples de áudio representam sons pré-gravados, sintetizadores são capazes de criar novos sons dependendo de como os controlas. Os sintetizadores do Sonic Pi são muito poderosos e expressivos. Primeiro, vamos aprender como escolher qual synth usar.

Dente de Serra e Prophet

Um som divertido é a (onda dentes de serra) – vamos experimentar-lo:

```
use_synth :saw
play 38
sleep 0.25
play 50
sleep 0.25
play 62
sleep 0.25
```

Tentemos outro synth o prophet:

```
use_synth :prophet
play 38
sleep 0.25
play 50
sleep 0.25
play 62
sleep 0.25
```

Se combinarmos os dois.

```
use_synth :saw
play 38
sleep 0.25
play 50
sleep 0.25
use_synth :prophet
play 57
sleep 0.25
```

Agora ao mesmo tempo:

```
use_synth :tb303
play 38
sleep 0.25
use_synth :dsaw
play 50
sleep 0.25
use_synth :prophet
play 57
sleep 0.25
```

Observe que o comando **use_synth** só afeta as chamadas que seguem o **play**. Pense nisso como uma chave - novos comandos para **play** reproduzirão qualquer sintetizador que foi selecionado para aquele momento. Você pode mudar para um novo sintetizador com o comando **use_synth**.

Descobrindo Synths

Para saber quais synths o Sonic Pi tem dê uma olhada na opção **Synth** no sistema de ajuda (à esquerda de *Fx*). Existem mais de vinte. Aqui alguns favoritos:

```
:prophet
:dsaw
:fm
:tb303
:pulse
```

É hora de experimentar os sintetizadores. Combine sintetizadores para criar sons novos e use synths diferentes em diferentes secções da música.

2.4 Duração com Envelopes

Em secção anterior vimos como usar o comando **sleep** para controlar quando disparar os sons. Mas ainda não sabemos como controlar a duração dos nossos sons.

Uma maneira simples e eficaz de controlar a duração dos sons, é o envelope tipo ADSR (vamos falar sobre o que ADSR significa mais à frente). O envelope de amplitude oferece dois parâmetros úteis de controle:

- controle da duração
- controle da amplitude

Duração

Duração é o tempo que conseguimos ouvir o som. Uma duração mais longa faz com que ouçamos o som durante mais tempo. Todos os

sons no Sonic Pi têm um envelope de amplitude controlável, e a duração total desse envelope é a duração do som. Portanto, controlando o envelope, controla-se a sua duração e consequentemente a forma da onda do som.

Além de controlar a duração, o envelope ADSR também oferece um controle excelente sobre a amplitude do som. Todos os sons audíveis começam e acabam com silêncio e contêm uma parte não silenciosa no meio. Os envelopes permitem deslizar e manter a amplitude das partes não silenciosas do som. É como dar instruções de como aumentar e diminuir o volume de um amplificador de guitarra. Por exemplo, podes pedir a alguém para "começar com silêncio, aumentar o volume lentamente até ao máximo, manter durante um bocado e depois diminuir rapidamente até silêncio". O Sonic Pi permite programar este comportamento com envelopes.

Para recapitular, uma amplitude 0 é silêncio e uma amplitude de 1 é volume normal.

Agora, vamos ver cada uma das partes do envelope.

Fase de Extinção *release:*

A única parte do envelope que é padrão é o tempo de extinção. Este é o tempo necessário para o som do sintetizador desaparecer. Todos os sintetizadores têm um tempo de extinção igual a 1, o que significa que, por padrão, eles têm uma duração de 1 batida (que no BPM padrão de 60 é de 1 segundo):

```
play 70
```

A nota será audível por 1 segundo. Vá em frente e cronometre :-). Esta é a versão simplificada, existe um outro modo de escrever o mesmo porém explicitando o tempo de extinção da nota:

```
play 70, release: 1
```

Note como soam iguais (o som dura 1 segundo). No entanto, agora é muito fácil alterar a duração mudando o valor da *opt* ‘release:‘:

```
play 60, release: 2
```

Podemos fazer com que o som dure muito pouco tempo usando um valor mais baixo para ‘release:‘

```
play 60, release: 0.2
```

A duração do repouso do som chama-se "fase de extinção"(ou release phase) e por padrão é uma transição linear (isto é, uma linha reta). O diagrama seguinte ilustra esta transição:

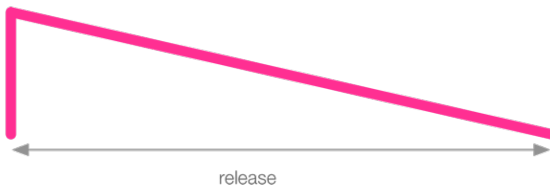


Fig. 2.1: curva de envelope da fase de extinção do som (release phase)

A linha vertical à esquerda do diagrama mostra que o som começa com amplitude 0, mas vai até a um nível de amplitude máximo imediatamente (isto é a fase de ataque, tal como vamos analisar de seguir). Tendo em conta a amplitude máxima, esta volta a baixar em linha reta até 0, levando o tempo especificado pela *opt* ‘release:‘. "Tempos de repouso mais longos produzem *fade outs* mais longos."

É possível mudar a duração do som ao mudar o tempo de repouso. Experimente mudar o **release** de seu código.

Fase de Ataque - *attack:*

Por padrão, a "fase de ataque"é 0 para todos os synths, o que significa que a amplitude sobe de 0 para 1 imediatamente. Isto resulta num som inicialmente mais percursivo. No entanto, podes querer que o teu som tenha um *fade in*. Isto pode ser feito com a opt **attack:**. Experimenta usar *fade in* em alguns sons:

```
play 60, attack: 2
sleep 3
play 65, attack: 0.5
```

Você pode querer usar várias *opts* ao mesmo tempo. Por exemplo, para um ataque curto e repouso longo, experimente:

```
play 60, attack: 0.7, release: 4
```

Este ataque curto e repouso longo está ilustrado no diagrama seguinte:

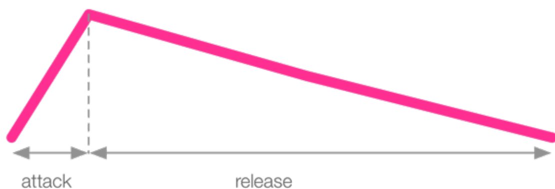


Fig. 2.2: envelope de ataque curto e extinção longo

Claro que é possível trocar. Tente um ataque longo e repouso curto:

```
play 60, attack: 4, release: 0.7
```

Por fim, também é possível ataque e release curtos.

```
play 60, attack: 0.5, release: 0.5
```

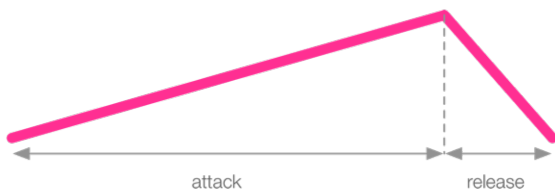


Fig. 2.3: envelope de ataque longo e extinção curto

Fig. 2.4: envelope de ataque e extinção curto

Fase de Sustentação - *sustain:*

Além de especificar os tempos de ataque e repouso, também é possível mudar o tempo de sustentação para controlar a "fase de sustentação". Isto é o máximo de tempo que a amplitude se mantém, entre o ataque e o repouso.

```
play 60, attack: 0.3, sustain: 1, release: 1
```

O tempo de sustentação é útil quando se pretende criar sons com mais presença na mistura, antes de entrar na fase de decaimento e extinção. Claro, é válido dar o valor de ‘0‘ às opts ‘attack:‘ e ‘release:‘ e usar só a sustentação para o som não ter nenhum fade in ou fade out. No entanto, vale avisar, um tempo de repouso de 0 pode produzir cliques sendo melhor usar um valor muito baixo como ‘0.2‘.

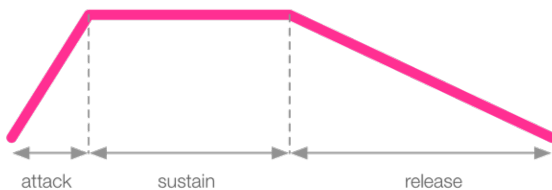


Fig. 2.5: envelope ASR (ataque, sustentação e extinção)

Fase de Decaimento - *decay:*

Para um controle maior, também é possível especificar o tempo de decaimento *decay:*. Esta fase fica entre o ataque e a sustentação e define o tempo em que a amplitude vai baixar do *attack_level:* para o *decay_level:* (a não ser que o mude, vai ser o mesmo que *sustain_level:*. Por padrão, a opt *decay:* é 0 e tanto o ataque *attack_level:* como a sustentação *sustain_level:* são 1, portanto é preciso especificá-los para o tempo de decaimento fazer efeito:

```
play 60, attack: 0.1, attack_level: 1, decay: 0.2,
sustain_level: 0.4, sustain: 1, release: 0.5
```

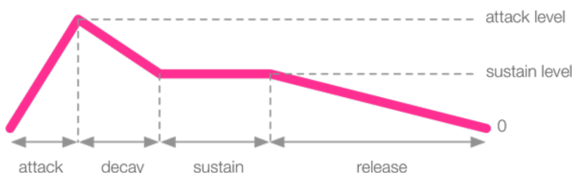


Fig. 2.6: envelope ADSR (ataque, decaimento, sustentação e extinção)

Nível de Decaimento -

Um último truque é que apesar da opt **decay_level:**, por padrão, ser igual ao **sustain_level:** é possível dar-lhes valores diferentes para controle total sobre o envelope. Isto permite criar envolventes assim:

```
play 60, attack: 0.1, attack_level: 1, decay: 0.2,
decay_level: 0.3, sustain: 1,sustain_level: 0.4, release: 0.5
```

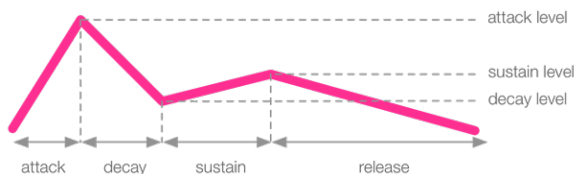


Fig. 2.7: envelope ADSR (ataque, decaimento, sustentação e extinção)

Também é possível contar um **decay_level:**, com um valor superior ao do **sustain_level**

```
play 60, attack: 0.1, attack_level: 0.1, decay: 0.2,
decay_level: 1, sustain: 0.5, sustain_level: 0.8, release: 1.5
```

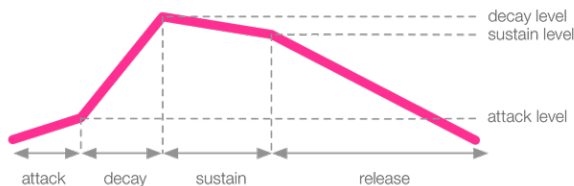


Fig. 2.8: envelope ADSR (ataque, decaimento, sustentação e extinção)

ADSR Envelopes

Resumindo, curvas de envelope do tipo ADSR no Sonic Pi têm as seguintes fases:

ataque - tempo de amplitude 0 até ao *attack_level:*
decaimento - tempo para mudar a amplitude de *attack_level:* até *sustain_level:*
sustentação - tempo para mudar a amplitude de *decay_level:* até *sustain_level:*
extinção - tempo para mudar a amplitude de *sustain_level:* até 0

É importante notar que a duração de um som é a soma dos tempos de cada uma das fases. Portanto, o seguinte som vai durar 0.5 + 1 + 2 + 0.5 = 4 tempos:

```
play 60, attack: 0.5, attack_level: 1, decay: 1,
sustain_level: 0.4, sustain: 2, release: 0.5
```

Experimente tocar com envelopes.

Capítulo 3

Sample

Outra boa forma de criar música é usar sons pré-gravados. Em hip-hop chamamos, tradicionalmente, estes sons de *samples*. Se levar um microfone para a rua, e gravar o som da chuva, irá criar um *sample*.

O Sonic Pi permite fazer coisas espetaculares com samples. Além de ter um banco de dados com mais de noventa samples de domínio público prontos para serem usados, ele permite tocar e manipular o seu próprio banco de sons.

3.1 Disparar Sample

Tocar samples pré-gravados é tão simples quanto tocar synths. Experimente:

```
sample :ambi_lunar_land
```

É possível tocar um **sample** assim como se toca um **play** criando um bloco de sons ou uma sequência:

```
play 36
play 48
sample :ambi_lunar_land
sample :ambi_drone
```

Se quiser pode criar uma sequência separando-os usando o comando **sleep:**

```
sample :ambi_lunar_land
sleep 1
play 48
sleep 0.5
play 36
sample :ambi_drone
sleep 1
play 36
```

Note que o Sonic Pi não espera que os sons terminem de tocar antes de começar o seguinte. O comando **sleep** só define o tempo entre o início de dois sons(sejam eles samples ou beeps). Isto permite criar sons facilmente, assim como sobreposição de efeitos. Veremos depois como controlar a duração de sons com envelopes.

Descobrindo Samples

Existem duas maneiras de descobrir quais samples existem no Sonic Pi. Primeiro, ir no o sistema de ajuda. Clique em *Samples* no menu vertical da esquerda, escolha uma categoria e verá uma lista com os samples disponíveis.

Outra maneira, é utilizar o sistema auto-complete. Basta escrever o início de um grupo de samples, como: **sample :ambi_** e verá um menu suspenso com nomes de samples. Experimente os prefixos de categorias seguintes:

:ambi_
:bass_
:elec_
:perc_
:guit_
:drum_
:misc_
:bd_

Experimente agora misturar samples nas suas músicas!

3.2 Parâmetros de Sample

Amp and Pan

Como vimos com os sintetizadores, podemos, controlar sons através do seu sistema de parametrização. Os samples suportam exatamente o mesmo mecanismo de parametrização dos synths. Vamos reexaminar os parâmetros 'amp:' e 'pan:'.

Amplificando Samples

É possível alterar a amplitude de samples com a mesma abordagem dos sintetizadores:

```
sample :ambi_lunar_land, amp: 0.5
```

Pan com Samples

Também podemos usar o parâmetro **pan:** com samples. Por exemplo:

```
sample :loop_amen, pan: -1
sleep 0.877
sample :loop_amen, pan: 1
```

Note que 0.877 é metade da duração da sample ':loop_amen' em segundos. Como veremos mais tarde, é importante lembrar que se alterar algum valor pré-definidos dos sintetizadores com **use_synth_defaults** esses valores vão ser ignorados pelo **sample**.

3.3 Time Stretch: esticando e contraindo o tempo dos samples

Agora que sabemos como tocar uma variedade de sintetizadores e samples para criar música, é hora de aprender a modificar sintetizadores e samples para criar personificar suas músicas tornando-as mais interessante. Primeiro, vamos explorar a possibilidade de aplicar time-stretching em samples.

Representação de Samples

Samples são sons pré-gravados, arquivados como uma sequencia de números que representam como a membrana do alto-falante se movimentará para gerar o som. A membrana do alto-falante pode mover para dentro ou fora. Os números representam o quão dentro ou fora tem de estar o cone a cada instante. Um sample normalmente tem de guardar vários milhares de números por segundo para para poder ser reproduzido com qualidade e precisão. O Sonic Pi pega nesses números e aplica a velocidade certa para mover a membrana para dentro e fora e assim reproduzir o som. Mas é possível alterar a velocidade com que os números de um sample são alimentam o processo de conversão de sinal digital analógico para que gerar o som na caixa de som.

Alterando a Velocidade de Leitura - *rate:*

Para tocar um sample em sua velocidade (*rate*) normal, o parâmetro **rate:** terá valor '1':

```
sample :ambi_choir, rate: 1
```

Podemos mudar esse valor para outro qualquer. Que tal '0.5':

```
sample :ambi_choir, rate: 0.5
```

O que se passou aqui? Bem, duas coisas. Primeiro, o sample demora o dobro do tempo a ser reproduzido. Segundo, o som está uma oitava abaixo. Vamos explorar isso com mais de cuidado.

Vamos esticar e contrair.

Um sample interessante para aplicar time-stretching é o "Amen Break". Em velocidade normal, poderíamos pô-lo numa música de "drum 'n' bass":

```
sample :loop_amen
```

Contudo, ao mudar a velocidade podemos eventualmente mudar o gênero musical também. Tente metade da velocidade para obter uma espécie de "hip-hop old school":

```
sample :loop_amen, rate: 0.5
```

Se aumentarmos a velocidade entramos no território *jungle*:

```
sample :loop_amen, rate: 1.5
```

Agora o que acontece se usarmos um valor de **rate** (velocidade) negativo?

```
sample :loop_amen, rate: -1
```

Isso mesmo! Toca no sentido contrário! Agora é hora de experimentar samples diferentes com rates diferentes. Experimente rates altos e lentos, altíssimos e lentíssimo. Vejamos até que ponto é possível obter sons interessantes alterando o a velocidade dos samples.

Breve Explicação Sobre *Sample Rate*

Uma maneira de pensar em samples é como molas. Mudar a rate é como esticar ou encolher um mola. Se tocarmos a sample com 'rate: 2', estamos encolhendo a mola para metade do seu comprimento normal, e então o sample demora metade do tempo a tocar por ser mais curta. Se tocarmos a sample com 'rate: 0.5', estamos a esticar a mola para o dobro do seu comprimento e, portanto, a sample demora o dobro do tempo a tocar por ser mais longa. Quanto a encolhermos (rate mais alta), esta fica mais curta, quanto mais esticada (rate mais baixa), mais longa fica.

Ao comprimir uma mola a sua densidade aumenta (número de voltas por cm)

- isto faz com que o sample soe "mais agudo". Ao esticar a mola a sua densidade diminui e isso faz com que o sample acabe por soar "mais grave".

Taxa de Amostragem: Perspectiva Matemática

(Esta parte é para os interessados em detalhes. Fique a vontade em passar para outro tópico ...)

Como vimos anteriormente, uma amostragem (*rate*) é representada por uma grande e longa lista de números que representam onde o alto-falante deve estar no tempo. Podemos pegar esta lista de números e usá-la para desenhar um gráfico como este:

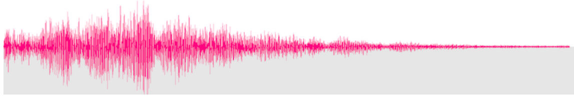


Fig. 3.1: gráfico de um sample

Você deve ter visto imagens como esta antes. Ela é uma representação da "forma de onda"de um sample. Na verdade é um gráfico com uma sequência grande de números. Normalmente, uma forma de onda como essa tem 44100 pontos de dados por segundo (isto ocorre devido ao teorema de amostragem de Nyquist-Shannon). Se a amostra durar 2 segundos, a forma de onda será representada por 88200 números que seriam tocados pelo alto-falante a uma taxa de 44100 pontos por segundo. Claro, poderíamos executar o mesmo sample ao dobro da taxa que seria de 88200 pontos por segundo. Assim o sample de 88200 duraria apenas 1 segundo. Também seria possível tocar novamente pela metade da taxa de 44100, o que seria 22050 pontos por segundo, isso demoraria 4 segundos para sua execução.

A duração de um sample sample é afetada pela taxa (rate):

- O dobro do rate diminui a duração para metade,
- Metade do rate aumenta a duração para o dobro,
- Um rate de 1/4 multiplica a duração por quatro,
- Com um rate de 1/10 o sample acaba por durar 10 vezes mais.

Podemos representar isto com a seguinte fórmula:

```
new_sample_duration = (1 / rate) * sample_duration
```

Alterar a taxa de reprodução também afeta o tom do sample. A frequência ou o tom de uma onda é determinado pela rapidez com que ela se move para cima e para baixo. Nosso cérebro, de alguma forma, percebe o movimento rápido do alto-falante como som agudo

e o movimento lento como som graves. É por isso que, às vezes, você pode ver o movimento da membrana dos alto-falantes quando eles reproduzem sons graves, isso ocorre pelo fato do falante estar se movendo muito devagar diferente quando um alto-falante que reproduz notas mais altas (agudas).

Se você pegar uma forma onda e contraí-la, ela se moverá para cima e para baixo mais vezes por segundo. Isso fará com que ela soe mais aguda. Acontece que duplicar a quantidade de movimentos para cima e para baixo (oscilações) duplica também a frequência. Então, "tocar um sample com uma taxa (*rate*) dupla dobrará a freqüência que você ouve". Além disso, "reduzir pela metade a taxa reduzirá pela metade a freqüência". Outras taxas afetarão a freqüência da mesma maneira.

3.4 Envelopando Samples

Também é possível modificar a "duração"e "amplitude"de um sample usando um envelope tipo ADSR. Contudo, isto funciona de forma ligeiramente diferente com os envelopes ADSR disponíveis em sintetizadores. Os envelopes dos samples apenas permitem reduzir a amplitude e duração de um sample mas não permitem aumentar alargar ou estender o sample. O sample irá parar quando ele tiver terminado de tocar ou quando o envelope tiver encerrado. Portanto, se você usar um valor de **release** muito longo, ele não estenderá a duração do sample.

Envelope Amen

Retomemos o nosso sample preferido Amen Break:

```
sample :loop_amen
```

Sem opts, ouvimos o sample completo, com toda a sua amplitude. Se quisermos fazer um fade de 1 segundo usamos o parâmetro 'ataque':

```
sample :loop_amen, attack: 1
```

Para um *fade in* mais curto, escolha um valor de ataque menor:

```
sample :loop_amen, attack: 0.3
```

Sustain Automático

O que difere o comportamento do envelope ADSR do envelope de um sintetizador padrão é o valor de sustentação. No envelope do sintetizador padrão, o *sustain* é padronizado para zero, a menos que você o configure manualmente. Com samples, o valor de *sustain* é definido automaticamente pelo tempo que demora a tocar o resto do sample. É por isto que nós ouvimos o sample completo quando tocamos sem alterar as definições. Se os valores de ataque, decay, sustain, e release forem todos 0 nós acabamos por nunca ouvir qualquer bip. O Sonic Pi, calcula a duração que o sample tem, deduz os tempos de ataque, decay e release e usa o resultado como valor de sustain. Se o valor de ataque, decay e release for superior à duração do sample, o sustain é simplesmente definido como zero.

Fade Out

Para explorar o *fade out*, vamos considerar o Amen Break com mais atenção. Se perguntarmos ao Sonic Pi qual o tempo do sample:

```
print sample_duration :loop_amen
```

Teremos o valor '1.753310657596372' que é a duração de um sample em segundos. Vamos arrendondá-lo para 1.75 por conveniência.

Agora, se definirmos o release para ’0.75’, algo surpreendente acontecerá:

```
sample :loop_amen, release: 0.75
```

Ele tocará o primeiro segundo do sample em amplitude total antes de fazer o *fade out* durante um período de 0.75 segundos. Este é como o "*auto sustain*"funciona. Por padrão, o release sempre funciona a partir do final da amostra. Se a nossa amostra tivesse 10.75 segundos de duração, reproduziria os primeiros 10 segundos em amplitude total antes de desaparecer em 0.75 segundos.

Lembre-se: por definição, o valor de ’release:’ diminui até ao fim do sample.

Fade In e Fade Out

Podemos usar o **attack:** e o **release:** juntos com o funcionamento do *auto sustain* para o *fade in* e para o *fade out* aplicando sobre a duração da amostra:

```
sample :loop_amen, attack: 0.75, release: 0.75
```

Como o sample tem 1.75s o nosso ataque e release demora aproximadamente de 1.5s, sendo o sustain automaticamente definido para 0.25s. isto permite facilmente fazer fazer fade in e out do sample.

Sustain explicito

Podemos voltar ao valor normal de ADSR do comportamento do sintetizador definindo manualmente o ’sustain’ como 0:

```
sample :loop_amen, sustain: 0, release: 0.75
```

Agora, nossa amostra só é tocada por 0,75 segundos. Com o padrão para 'attack:' e 'decay:' em 0, a amostra salta diretamente para amplitude total, como o sustain é de 0s, ele voltará para 0 amplitude durante o período de 0,75s que é o valor total do release.

Címbalos percussivos

Esse funcionamento tem uma boa aplicação quando para transformar sample longos em versões curtas e percussivas.

```
sample :drum_cymbal_open
```

É possível ouvir o som dos pratos (cymbal) durante um determinado período de tempo. Contudo, podemos usar o envelope para o torná-lo mais percussivo:

```
sample :drum_cymbal_open, attack: 0.01, sustain: 0, release: 0.1
```

É possível emular o ataque de um prato (cymbal) e mantê-lo ativo, aumentado o período de sustain:

```
sample :drum_cymbal_open, attack: 0.01, sustain: 0.3, release: 0.1
```

Agora divirta-se colocando envelopes nos sample. Tente alterar o *rate* para resultados interessantes.

3.5 Recortando Sample

Esta seção irá concluir nossa exploração do jogador de amostra da Sonic Pi. Vamos fazer uma breve recapitulação. Até agora, vimos como disparar sample:

```
sample :loop_amen
```

Vimos modos de mudar a velocidade de um sample tocando-os pela metade da sua velocidade:

```
sample :loop_amen, rate: 0.5
```

Em seguida, vimos como fazer *fade in* de um sample:

```
sample :loop_amen, rate: 0.5, attack: 1
```

Nós também analisamos como podemos usar o início de um sample de forma percussiva ao dar ao **sustain:** um valor explícito e definindo tanto o ataque e o release como valores curtos:

```
sample :loop_amen, rate: 2, attack: 0.01, sustain: 0, release: 0.35
```

No entanto, não seria bom se não tivéssemos que começar sempre no início da amostra? Não seria bom se não tivéssemos que terminar sempre no final da amostra?

Definindo Início

É possível escolher um ponto de partida arbitrário na amostra como um valor entre 0 e 1 onde 0 é o início da amostra, 1 é o fim . Vamos tentar tocar apenas a última metade do amen break:

```
sample :loop_amen, start: 0.5
```

E se for o ultimo quarto do sample?

```
sample :loop_amen, start: 0.75
```

Definindo Fim

Da mesma forma, é possível escolher um ponto final arbitrário no sample como um valor entre 0 e 1. Vamos terminar o amen pela metade:

```
sample :loop_amen, finish: 0.5
```

Definindo Inicio e Fim

Porque não combinar dois pontos para tocar segmentos arbitrários do arquivo de áudio? Que tal apenas uma pequena seção no meio:

```
sample :loop_amen, start: 0.4, finish: 0.6
```

O que acontece se escolher uma posição inicial posterior a posição final?

```
sample :loop_amen, start: 0.6, finish: 0.4
```

Legal! Toca ao revés!

Combinando com Rate

Podemos combinar esta capacidade de tocar segmentos arbitrários de áudio com o **rate:**. Por exemplo, podemos tocar uma seção muito pequena no meio do amen, muito lentamente:

```
sample :loop_amen, start: 0.5, finish: 0.7, rate: 0.2
```

Combinando com envelope

Finalmente, podemos combinar tudo isso com envelopes ADSR para produzir resultados interessantes:

```
sample :loop_amen, start: 0.5, finish: 0.8, rate: -0.2,
attack: 0.3, release: 1
```

Agora siga e toque misturando os sample com todas essas coisas divertidas ...

3.6 Sample Externo

Embora os sample ajudem a começar a tocar rapidamente, você pode querer experimentar outros sons gravados. Sonic Pi é totalmente compatível com isso. Primeiro, vamos ter uma discussão rápida sobre a portabilidade de sua peça.

Portabilidade

Quando você compõe sua peça puramente com sintetizadores e amostras integrados, o código é tudo o que você precisa para reproduzir fielmente sua música. Pense nisso por um momento - isso é incrível! Um simples texto que você pode enviar por email ou ficar em um Gist representa tudo o que você precisa para reproduzir seus sons. Isso faz com que seja muito fácil compartilhar com seus amigos, pois eles só precisam ter acesso ao código.

No entanto, se você começar a usar suas próprias amostras pré-gravadas, você perderá essa portabilidade. Isso porque reproduzir sua música, outras pessoas não só precisarão do seu código, mas também precisarão de seus sample. Isso limita a capacidade de outros manipula-

rem, mexerem e experimentarem seu trabalho. Claro que isso não deve impedir você de usar suas próprias amostras, é apenas algo a considerar.

Sample Local

Então, como tocar qualquer arquivo WAV, AIFF ou FLAC em seu computador? Tudo o que você precisa é indicar o caminho desse arquivo:

```
# Raspberry Pi, Mac, Linux
sample "/Users/sam/Desktop/my-sound.wav"
# Windows
sample "C:/Users/sam/Desktop/my-sound.wav"
```

Sonic Pi carregará automaticamente e tocará o sample. Você também pode informar os parâmetros padrões que você usou para executar o sample:

```
# Raspberry Pi, Mac, Linux
sample "/Users/sam/Desktop/my-sound.wav", rate: 0.5, amp: 0.3
# Windows
sample "C:/Users/sam/Desktop/my-sound.wav", rate: 0.5, amp: 0.3
```

3.7 Pasta de Sample

Nota: esta seção do tutorial cobre tópico avançado que trabalha com grandes diretórios de suas próprias amostras. Este será o caso se você tiver baixado ou comprado seus próprios pacotes de amostra e quiser usá-los dentro do Sonic Pi.

Não hesite em ignorar se você está feliz trabalhando com as amostras existentes.

Ao trabalhar com grandes pastas de amostras externas, pode ser complicado ter que digitar o caminho do diretório toda vez que desencadeie um sample específico.

Por exemplo, imageine que você tem a seguinte pasta na sua máquina:

```
/path/to/my/samples/
```

Quando olhamos dentro dessa pasta, encontramos as seguintes amostras:

```
100_A#_melody1.wav
100_A#_melody2.wav
100_A#_melody3.wav
120_A#_melody4.wav
120_Bb_guit1.wav
120_Bb_piano1.wav
```

Normalmente, para tocar a amostra do piano, usamos o caminho completo:

```
sample "/path/to/my/samples/120_Bb_piano1.wav"
```

Se quiser então tocar o sample de guitarra pode usar o caminho completo também:

```
sample "/path/to/my/samples/120_Bb_guit.wav"
```

No entanto, ambas endereçamentos para o sample exigem que conheçamos seus nomes dentro do nosso diretório. E se quisermos ouvir rapidamente cada amostra?

Indexação de Sample e Diretórios

Se quisermos tocar a primeira amostra em um diretório, precisamos passar o nome do diretório para amostra e o índice 0 da seguinte maneira:

```
sample "/path/to/my/samples/", 0
```

Podemos até fazer um atalho no nosso caminho do diretório usando uma variável:

```
samps = "/path/to/my/samples/"
sample samps, 0
```

Agora, se quisermos jogar a segunda amostra no nosso diretório, precisamos apenas adicionar 1 ao nosso índice:

```
samps = "/path/to/my/samples/"
sample samps, 1
```

Observe que não precisamos mais conhecer os nomes das amostras no diretório - precisamos apenas conhecer o próprio diretório (ou ter um atalho para isso). Se escrevemos um índice maior do que o número de amostras do diretório, ele simplesmente retorna num ciclo. Portanto, qualquer número que utilizar, é garantido que você obtenha um dos sample contido nesse diretório.

Filtrando Sample

Normalmente, a indexação é suficiente, mas às vezes precisamos de mais poder para classificar e organizar nossos sample. Felizmente, muitos pacotes de amostras adicionam informações úteis nos nomes dos arquivos. Vamos dar uma nova olhada nos nomes dos arquivos de exemplo em nosso diretório:

```
100_A#_melody1.wav
100_A#_melody2.wav
100_A#_melody3.wav
120_A#_melody4.wav
120_Bb_guit1.wav
120_Bb_piano1.wav
```

Observe que, nesses nomes de arquivos, temos um pouco de informação. Em primeiro lugar, temos o BPM da amostra (batimentos por minuto) no início. Assim, a amostra do piano é de 120 BPM e as nossas primeiras três melodias estão em 100 BPM. Além disso, nossos nomes de exemplo contêm o tom. Então, a amostra de guitarra está em Bb e as melodias estão em A#. Esta informação é muito útil para misturar essas amostras com nosso outro código. Por exemplo, sabemos que só podemos tocar a amostra de piano com código que está em 120 BPM e no tom Bb.

Acontece que podemos usar esta convenção de nomenclatura específica de nossos conjuntos de sample no código para nos ajudar a filtrar os que queremos. Por exemplo, se trabalharmos com 120 BPM, podemos filtrar para todas as amostras que contêm a seqüência "120"com o seguinte:

```
samps = "/path/to/my/samples/"
sample samps, "120"
```

Isso nos interpretará na primeira combinação. Se quisermos a segunda combinação, precisamos usar o índice:

```
samps = "/path/to/my/samples/"
sample samps, "120", 1
```

Podemos até mesmo usar vários filtros ao mesmo tempo. Por exemplo, se quisermos uma amostra cujo nome de arquivo contenha as substrings "120"e "A#", podemos encontrá-lo facilmente com o seguinte código:

```
samps = "/path/to/my/samples/"
sample samps, "120", "A#"
```

Finalmente, ainda podemos adicionar opts usuais à chamada do sample:

```
samps = "/path/to/my/samples/"
sample samps, "120", "Bb", 1, lpf: 70, amp: 2
```

Fontes

O sistema de filtro de sample **pre-arg** reconhece dois tipos de informações: fontes e filtros. Fontes são informações usadas para criar a lista de potenciais candidatos. Uma fonte pode assumir duas formas:

1. "/path/to/samples- uma string que representa um caminho válido para um diretório
2. "/path/to/samples/foo.wav- uma string que representa um caminho válido para um sample

Um sample fn irá primeiro reunir todas as fontes e usá-las para criar uma grande lista de candidatos. Esta lista é construída primeiro adicionando todos os caminhos válidos e adicionando todos os arquivos válidos .flac, .aif, .aiff, .wav, .wave contidos nos diretórios.

Por exemplo, veja o seguinte código:

```
samps = "/path/to/my/samples/"
samps2 = "/path/to/my/samples2/"
path = "/path/to/my/samples3/foo.wav"

sample samps, samps2, path, 0
```

Aqui, estamos combinando o conteúdo dos sample dentro de dois diretórios e adicionando uma amostra específica. Se "/ path / to /

my / samples /"continha 3 amostras e "/ path / to / my / samples2 /"continha 12, teríamos 16 amostras potenciais para indexar e filtrar (3 + 12 + 1).

Por padrão, apenas os arquivos de sample dentro de um diretório são reunidos na lista de candidatos. Às vezes você pode ter uma série de pastas aninhadas de amostras que deseja pesquisar e filtrar dentro. Você pode, portanto, fazer uma pesquisa recursiva para todas as amostras em todas as subpastas de uma determinada pasta, adicionando ** ao final do caminho:

```
samps = "/path/to/nested/samples/**"
sample samps, 0
```

Tenha cuidado no entanto, enquanto pesquisar através de um conjunto muito grande de pastas, isso pode demorar muito. No entanto, o conteúdo de todas as fontes da pasta está em cache, então o atraso só acontecerá na primeira busca.

Finalmente, note que as fontes devem ir primeiro. Se nenhuma fonte for fornecida, o conjunto de amostras embutidas será selecionado como a lista padrão de candidatos para trabalhar.

Filtros

Depois de ter uma lista de candidatos, você pode usar os seguintes tipos de filtragem para reduzir ainda mais a seleção:

- "foo"Strings irá filtrar na ocorrência de substring no nome do arquivo (menos o caminho do diretório e a extensão).
- / fo [oO] / Regular Expressões irá filtrar na correspondência de padrões do nome do arquivo (menos o caminho do diretório e a extensão).

- : foo - As palavras-chave filtram os candidatos se a palavra-chave é uma combinação direta do nome do arquivo (menos o caminho do diretório e a extensão).
- lambda | a | ... - Procs com um argumento serão tratados como um filtro candidato ou função geradora. Será aprovada a lista de candidatos atuais e deve retornar uma nova lista de candidatos (uma lista de caminhos válidos para arquivos de amostra).
- 1 - Os números selecionarão o candidato com esse índice (envolvendo rodada como um anel, se necessário).

  Por exemplo, podemos filtrar todas as amostras em um diretório contendo a string "foo"e reproduzir a primeira amostra correspondente a meia velocidade

  ```
  sample "/path/to/samples", "foo", rate: 0.5
  ```

Veja o help de sample para mais exemplos e detalhes de uso. Lembre-se que a ordem dos filtro é respeitada.

Compósitos

Por fim, você pode usar listas onde você puder colocar uma fonte ou filtro. A lista será automaticamente achatada e o conteúdo será tratado como fontes e filtros comuns. Portanto, as seguintes chamadas para amostra são equivalentes semanticamente:

```
sample "/path/to/dir", "100", "C#"
sample ["/path/to/dir", "100", "C#"]
sample "/path/to/dir", ["100", "C#"]
sample ["/path/to/dir", ["100", ["C#"]]]
```

Empacotando

Esta foi uma seção avançada para pessoas que precisam de ter um real controle para manipular e usar pacotes de sample. Se a maior parte desta seção não faz muito sentido, não se preocupe. Provavelmente você ainda não precisa de nenhuma dessas funcionalidades. No entanto, quando você começar a trabalhar com grandes diretórios de amostras você sabe agora que se precisar poderá voltar e reler.

Capítulo 4

Randomização

Uma boa maneira de deixar sua música mais interessante é incluindo números aleatórios. Sonic Pi tem uma ótima funcionalidade para incluir aleatoriedade, mas antes de começar, precisamos aprender uma verdade chocante: no Sonic Pi o aleatório não é verdadeiramente aleatório. Que diabos isso quer dizer? Bem vamos ver.

Repetibilidade

Uma função aleatória realmente útil é **rrand**, que lhe gera um valor aleatório entre dois números - um mínimo e um máximo. (**rrand** é a abreviação de *range random*). Vamos tentar tocar uma nota aleatória:

```
play rrand(50, 95)
```

Ooh, tocou uma nota aleatória. A nota 83.7527 que está entre 50 e 95. Mas, espere, acabei de prever a nota aleatória exata que você também obteve? Alguma coisa estranha está acontecendo aqui. Tente

executar o código novamente. O que? Sorteou 83.7527 novamente? Isso não pode ser aleatório!

A resposta é que o sorteio não é realmente aleatório mas pseudo-aleatório. Sonic Pi lhe dará números aleatórios de uma maneira repetitiva. Isso é muito útil para garantir que a música que você crie em sua máquina seja idêntica na máquina de todos os outros - mesmo se você usar aleatoriedade na sua composição.

Claro, em uma determinada música, se o *random* "escolhesse aleatoriamente"sempre 83.7527, então não seria muito interessante. No entanto, não. Experimente o seguinte:

```
loop do
  play rrand(50, 95)
  sleep 0.5
end
```

Finalmente soa aleatório. Ao executar o random em loop, as operações subseqüentes retornarão valores aleatórios. No entanto, a próxima operação produzirá exatamente a mesma seqüência de valores aleatórios e tocará exatamente o mesmo. É como se todo o código Sonic Pi voltasse no tempo para exatamente o mesmo ponto sempre que o botão **Run** foi pressionado. É como o Feitiço do Tempo (Groundhog Day) da síntese musical!

Sinos Amaldiçoados

Uma adorável ilustração da aleatoriedade em ação é a dos sinos amaldiçoados em que o loop do sample **:perc_bell** é tocado a **:rate** e **:sleep** aleatório entre os sons dos sinos.

```
loop do
  sample :perc_bell, rate: (rrand 0.125, 1.5)
  sleep rrand(0.2, 2)
end
```

Cutoff Aleatório

Outro exemplo interessante de aleatoriedade é a modificação aleatória do **cutoff** de um sintetizador. Um bom sintetizador para experimentar isto é o emulador **:tb303**:

```
use_synth :tb303

loop do
  play 50, release: 0.1, cutoff: rrand(60, 120)
  sleep 0.125
end
```

Sementes Randômicas

Então, caso não goste da sequência de números aleatórios gerado, é possível escolher uma outra usando **use_random_seed**. A **seed** (semente) padrão é 0, então escolha um valor diferente para ter outra sequência!

```
5.times do
  play rrand(50, 100)
  sleep 0.5
end
```

Toda vez que executar este código, ouvirá a mesma seqüência de 5 notas. Para obter uma sequência diferente, basta mudar o valor de **use_random_seed**:

```
use_random_seed 40
5.times do
  play rrand(50, 100)
  sleep 0.5
end
```

Isso produzirá uma seqüência diferente de 5 notas. Ao mudar **seed** e ouvir os resultados, você pode encontrar algo que goste - e quando

compartilhar seu código com os outros, eles ouvirão exatamente o que você também ouviu.

Vamos olhar outras funções aleatórias.

Choose (Escolha)

Uma coisa comum é escolher um item aleatoriamente de uma lista de itens conhecidos. Por exemplo, talvez queira tocar uma das seguints notas: 60, 65 ou 72. Isso pode ser feito com **choose**, que escolherá um item de uma lista. Primeiro, eu preciso colocar meus números em uma lista (definida pelos colchetes) e separando-os com vírgulas: [60, 65, 72]. Em seguida, só é preciso escolhê-los (**choose**):

```
choose([60, 65, 72])
```

Vamos ouvir como soa.

```
loop do
  play choose([60, 65, 72])
  sleep 1
end
```

rrand

Já vimos **rrand**, mas vale revê-lo. Ele retorna um número aleatório entre dois valores. Isso significa que nunca retornará um número superior ou inferior - sempre algo entre os dois. O número sempre será decimal - o que significa que não é um número inteiro, mas uma fração de um número. Exemplos de números decimais retornados por rrand (20, 110):

- 87.5054931640625

- 86.05255126953125
- 61.77825927734375

rrand_i

O **rrand_i** funciona como o **rrand** para números inteiros. A única diferença é que ele pode gerar os valores mínimo e máximo definidos (o que significa que é incluído, em vez de excluído do intervalo). Exemplos de números retornados por **rrand_i** (20, 110) são:

- 88
- 86
- 62

rand

Rand retornará um número decimal aleatório entre 0 (inclusive) e o valor máximo que você especificou (exclusivo). Por padrão, ele retornará um valor entre 0 e 1. Por isso, é útil para escolher valores aleatórios de **amp:** :

```
loop do
  play 60, amp: rand
  sleep 0.25
end
```

rand_i

Semelhante a **rrand_i** e **rrand**, **rand_i** retornará um número total aleatório entre 0 e o valor máximo especificado.

dice (dados)

Às vezes se quer imitar um lance de dados - este é um caso especial de **rrand_i** onde o valor mais baixo é sempre 1. Usar **dice** implica dizer o número de lados do dado. Um dado padrão tem 6 lados, então o **dice** (6) retornará 1, 2, 3, 4, 5 ou 6. No entanto, ter dados de 4, 12, 17 ou20 lados - talvez até com 120 lados! Por que não?

one_in

Finalmente é possível emular o lançamento de um para 6 como um dado normal. **one_in** devolve *true/false* (verdadeiro ou falso) conforme o número de lados do dado. Assim **one_in(6)** irá devolver *true* ou *false* com a probabilidade de 1 em 6. *True* e *false* são valores muito úteis nas declarações condicionais **if**, que serão descritas nas secções seguintes deste tutorial.

Agora, aproveite para testar seu código com processos aleatórios !

Capítulo 5

Programando Estruturas

Agora que você aprendeu conceitos básicos de criação de sons com **play** e **sample** , criando melodias e ritmos simples com **sleep** , fica a pergunta: o que mais o mundo do código pode oferecer?

Acontece que estruturas de programação básicas, como o **loop**, **condicionais**, funções e tópicos, são ferramentas poderosas para expressar ideias musicais.

Vamos ficar presos com o básico ...

5.1 Blocos

Uma estrutura comum no Sonic Pi é o bloco. Os blocos permitem fazer coisas úteis com grandes pedaços de código. Por exemplo, com

parâmetros de sintetizadores e amostras, conseguimos mudar algo que aconteceu em uma única linha. No entanto, às vezes queremos fazer algo significativo com várias linhas de código. Por exemplo, talvez desejemos fazer o **loop**, adicionar reverb a ele, apenas executá-lo 1 vez em 5, etc. Vejamos:

```
play 50
sleep 0.5
sample :elec_plip
sleep 0.5
play 62
```

Para fazer algo com um pedaço de código, precisamos dizer ao Sonic Pi, onde o bloco de código começa e onde termina. Usamos **do** para começar e **end** para terminar. Por exemplo:

```
do
  play 50
  sleep 0.5
  sample :elec_plip
  sleep 0.5
  play 62
end
```

Mas, este exemplo ainda não está finalizado (experimente rodar e terá um erro) e não funcionará, já que não informamos ao Sonic Pi o que queremos fazer com o bloco entre **do** e **end**. É preciso informar antes do **do** o que será feito com esse código. Veremos vários desses fragmentos especiais de código. Por enquanto, é importante saber que ao envolver seu código com **do** e **end** você informa ao Sonic Pi que deseja fazer algo especial com esse pedaço de código.

5.2 Iteração e Loops

Até agora passamos um bom tempo a olhar para diferentes sons que podemos fazer com blocos **play** e **sample**. Também aprendemos

como disparar esses sons no tempo usando **sleep**.

Como você provavelmente descobriu, é possível se divertir bastante com esses blocos de construção básicos. No entanto, uma nova dimensão se abre quando você começar a usar o código para estruturar suas músicas e composições. Nas próximas seções, exploraremos algumas dessas novas ferramentas. Primeiro iteração e loops.

Repetição

Já escreveu algum código e gostaria de repetir algumas vezes? Por exemplo:

```
play 50
sleep 0.5
sample :elec_blup
sleep 0.5
play 62
sleep 0.25
```

Se quiser repetir isto 3 vezes você pode copiar e colar três vezes:

```
play 50
sleep 0.5
sample :elec_blup
sleep 0.5
play 62
sleep 0.25

play 50
sleep 0.5
sample :elec_blup
sleep 0.5
play 62
sleep 0.25

play 50
sleep 0.5
sample :elec_blup
sleep 0.5
```

```
play 62
sleep 0.25
```

É muito código! O que acontece se quiser mudar o sample para **:elec_plip**? Teria de encontrar todos **:elec_blup** e mudá-los. Não seria muito prático.

Iteração

Alterar o código deve ser fácil como dizer "faça isto 3 vezes". Na verdade é exatamente isso. Lembra do bloco de código (**do end**)? Podemos usá-los para marcar o inicio e o fim do código que queremos repetir 3 vezes. Usamos então o código especial **3.times**. Assim, em vez de escrever "faça isto 3 vezes", escrevemos **3.time do**. Não esqueça de escrever **end** no final do código que deseja repetir:

```
3.times do
  play 50
  sleep 0.5
  sample :elec_blup
  sleep 0.5
  play 62
  sleep 0.25
end
```

Isto simplifica bastante e podemos usar isto para criar estruturas repetitivas elegantes:

```
4.times do
  play 50
  sleep 0.5
end

8.times do
  play 55, release: 0.2
  sleep 0.25
end
```

```
4.times do
  play 50
  sleep 0.5
end
```

Iterações aninhadas

E se colocarmos iterações dentro de iterações?

```
4.times do
  sample :drum_heavy_kick
  2.times do
    sample :elec_blip2, rate: 2
    sleep 0.25
  end
  sample :elec_snare
  4.times do
    sample :drum_tom_mid_soft
    sleep 0.125
  end
end
```

Looping

Se quiser repetir algo muitas vezes, poderá usar números grandes como ‘1000.times do‘. Neste caso é provavelmente melhor pedir ao Sonic Pi para repetir para sempre (pelo menos até premires o botão stop). Vamos repetir o sample amen break para sempre:

```
loop do
  sample :loop_amen
  sleep sample_duration :loop_amen
end
```

Algo importante saber sobre loops é que eles atuam como buracos negros. Uma vez que o código entra em loop nunca sairá até pressionar **stop** - irá repetir para sempre. Isso significa que se tiveres código após o loop nunca o irás ouvir. Por exemplo, o címbalo depois deste loop nunca será tocado:

```
loop do
  play 50
  sleep 1
end

sample :drum_cymbal_open
```

Agora pode estruturar o teu código com iterações e loops!

5.3 Condicional

Em algum momento você irá querer tocar só notas aleatórias (ver secção anterior sobre aleatoriedade) mas estabelecer decisões aleatórias baseadas no resultado do código que corre ou nalgum outro código. Por exemplo, tocar aleatoriamente um bateria ou pratos. Isso é possível com uma declaração "se"**if**.

Lançar Moeda

Então, vamos lançar uma moeda: se for cara, toca a bateria, se for coroa, toca o prato. Podemos emular o lançamento da moeda com a função **one_in** (ver secção randomização), especificando a probabilidade de 1 em 2: **one_in(2)**. Podemos usar esse resultado para decidir entre duas partes de código, o código que toca a bateria e o código que toca o címbalo.

```
loop do
```

```
  if one_in(2)
    sample :drum_heavy_kick
  else
    sample :drum_cymbal_closed
  end

  sleep 0.5

end
```

Lembre que a declaração **if** tem 3 partes:

- A questão a ser feita
- A primeira escolha do código a executar (se a questão à pergunta for sim)
- A segunda escolha do código a executar (se a resposta à questão for não)

Tipicamente em linguagens de programação, a noção de sim é representada pelo termo **'true'** e a noção de não é representada pelo termo **'false'**. Assim temos que arranjar uma questão que nos dirá uma resposta 'true' ou 'false'. É exatamente isso que o **one_in** faz.

Repare como a primeira escolha está envolvida entre o **'if'** (se) e o **'else'** (então) e a segunda escolha está envolvida entre o **'else'** (então) e o **'end'**. Tal como os blocos **do/end** é possível colocar múltiplas linhas de código nesses espaços. Por exemplo:

```
loop do

  if one_in(2)
    sample :drum_heavy_kick
    sleep 0.5
  else
    sample :drum_cymbal_closed
    sleep 0.25
  end

end
```

Neste caso o tempo do **sleep** difere conforme escolha feita.

If simples

As vezes queremos executar apenas uma linha de código. Isto é possível colocando o **if** e depois a questão no final. Por exemplo:

```
use_synth :dsaw

loop do
  play 50, amp: 0.3, release: 2
  play 53, amp: 0.3, release: 2 if one_in(2)
  play 57, amp: 0.3, release: 2 if one_in(3)
  play 60, amp: 0.3, release: 2 if one_in(4)
  sleep 1.5
end
```

O exemplo toca acordes de números diferentes com a chance de cada nota tocar com uma probabilidade diferente.

5.4 Threads

Imagine que você tem linha de baixo e uma batida. Como tocar ao mesmo tempo? Uma solução é interligá-los manualmente - tocar algumas notas do baixo, depois um pouco de bateria e depois mais baixo ... N No entanto, o tempo torna-se difícil de pensar, especialmente se começares a entrelaçar mais elementos.

E se o Sonic Pi pudesse interligar os elementos automaticamente? Bem, isso é possível através da **thread.**

Loops Infinitos

Para manter este exemplo simples, imagine que isto é um beatida e uma linha de baixo:

```
loop do
  sample :drum_heavy_kick
  sleep 1
end

loop do
  use_synth :fm
  play 40, release: 0.2
  sleep 0.5
end
```

Como falamos anteriormente, loops são buracos negros para o programa. Uma vez neles nunca poderá sair até pressionar **stop**. Então como tocar ambos os loops ao mesmo tempo? Teremos que dizer ao Sonic Pi que queremos iniciar algo ao mesmo tempo que o restante do código. É ai que os **threads** entram.

Threads ao Auxilio

```
in_thread do
  loop do
    sample :drum_heavy_kick
    sleep 1
  end
end
loop do
  use_synth :fm
  play 40, release: 0.2
  sleep 0.5
end
```

Envolvendo o primeiro loop num **in_thread** do bloco **do/end** dizemos ao Sonic Pi para correr o conteúdo do bloco "exactamente"ao

mesmo tempo que a declaração seguinte ao bloco (que até é o segundo loop). Experimente e ouvirá a bateria e o baixo entrelaçados.

Agora tente adicionar um synth por cima. Algo como:

```
in_thread do
  loop do
    sample :drum_heavy_kick
    sleep 1
  end
end

loop do
  use_synth :fm
  play 40, release: 0.2
  sleep 0.5
end

loop do
  use_synth :zawa
  play 52, release: 2.5, phase: 2, amp: 0.5
  sleep 2
end
```

Teremos o mesmo problema de antes. O primeiro loop é tocado ao mesmo tempo que o segundo devido ao **in_thread**. No entanto, o terceiro loop nunca é tocado. Portanto necessitamos de outra thread:

```
\\
in_thread do
  loop do
    sample :drum_heavy_kick
    sleep 1
  end
end

in_thread do
  loop do
    use_synth :fm
    play 40, release: 0.2
    sleep 0.5
  end
end
```

```
loop do
  use_synth :zawa
  play 52, release: 2.5, phase: 2, amp: 0.5
  sleep 2
end
```

Run como Threads

Algo que pode supreender é quando pressiona o botão **Run**, você cria um novo **thread** para o código correr. Por isso que ao pressionar várias vezes você irá sobrepor camadas de som. O **run** são **threads**, que irão entrelaçar automaticamente os sons.

Osciloscópio

Assim que aprender mais sobre o Sonic Pi, entenderá que os **threads** são os blocos mais importante de construção da música. Seu papel é importante por isolar a noção de "settings atuais"de outras threads. Isso significa que quando trocar de synth usando o **use_synth** apenas mudará de sintetizador na *thread atual* - nenhuma outra thread terá a mudança de sintetizador. Vamos ver isto em pratica:

```
play 50
sleep 1

in_thread do
  use_synth :tb303
  play 50
end

sleep 1
play 50
```

Reparou como o som do meio foi diferente dos restantes? A declaração **use_synth** apenas afetou a thread onde estava e não a thread principal.

Herança

Quando criar uma nova *thread* com **in_thread**, a nova *thread* irá automaticamente herdar todas os parâmetros atuais da *thread* corrente. Vejamos:

```
use_synth :tb303
play 50
sleep 1

in_thread do
  play 55
end
```

Repare como a segunda nota foi tocada com o sintetizador **:tb303** apesar de estar em uma thread separada. Todos os parâmetros modificadas com as várias funções '**use_***' irão se comportar da mesma maneira.

Quando as *threads* são criados elas herdam todas as características dos seus pais mas não passam para qualquer mudança que tenham.

Nomeando Threads

Finalmente, podemos dar nomes às nossas **threads**:

```
in_thread(name: :bass) do
  loop do
    use_synth :prophet
    play chord(:e2, :m7).choose, release: 0.6
    sleep 0.5
  end
```

```
end

in_thread(name: :drums) do
  loop do
    sample :elec_snare
    sleep 1
  end
end
```

Veja o painel de registro enquanto roda o código. Observe como é reportado o nome do *thread* na mensagem?

```
[Run 36, Time 4.0, Thread :bass]
 |- synth :prophet, {release: 0.6, note: 47}
```

Única Thread por Nome

Uma ultima informação a respeito da *in_thread(name:)*. Não é possível rodar ao mesmo tempo duas *in_thread(name:)* com mesmo nome. Vejamos!

```
in_thread do
  loop do
    sample :loop_amen
    sleep sample_duration :loop_amen
  end
end
```

Cola isso num *buffer* e pressione o butão **Run** várias vezes. Ouve a cacofonia de vários amen break looping fora de tempo entre eles? Ok, pode pressionar **Stop** . Este é o comportamento que vemos sempre - se pressionar o botão **Run**, as camadas de som se sobrepõem. Assim se tiver um loop e pressionar o botão **Run** 3 vezes, terás 3 camadas de loops rodando simultaneamente.

No entanto, com *in_thread(name:)* é diferente:

```
in_thread(name: :amen) do
  loop do
    sample :loop_amen
    sleep sample_duration :loop_amen
  end
end
```

Experimente rodar 3 vezes com este código. Apenas irá ouvir um loop e poderá observar isto na janela de registro:

```
==> Skipping thread creation: thread with name :amen already exists.
```

Sonic Pi diz que uma *thread* com o nome ':amen' já existe, e assim não irá criar outro.

Esta informação pode não ser útil agora - mas será fundamental quando começar o *live code*...

5.5 Funções

Quando se habituar a escrever código, você buscará formas elegantes e de fácil compreensão para organizar e estruturar suas ideias. As funções são muito eficazes pela sua capacidade de nomear um monte de código. Vamos dar uma olhada.

Definindo funções

```
define :foo do
  play 50
  sleep 1
  play 55
  sleep 2
end
```

Aqui definimos uma nova função chamada ‘foo‘. Utilizamos para isso o bloco **do/end** e a palavra **‘define‘** seguida por um nome qualquer que definirá a função. Podemos definir a função com qualquer nome como ‘bar‘, ‘baz‘ idealmente com um nome que faça sentido para você como ‘secção_principal‘ou ’riff_condução’.

Lembre de acrescentar dois pontos ‘:‘ ao nome da função.

Chamando funções

Depois de definir a função chame escrevendo o seu nome:

```
define :foo do
  play 50
  sleep 1
  play 55
  sleep 0.5
end
foo
sleep 1
2.times do
  foo
end
```

Podemos usar ‘foo‘ dentro de um bloco de iteração ou em qualquer lugar do código que tenha ‘play‘ ou ‘sample‘. Isto é uma excelente maneira de expressar criando palavras significativas ao projeto musical em desenvolvimento.

As funções são recordadas ao longo das execuções

Até agora, cada vez que pressiona o botão **Run**, o Sonic Pi começa completamente do zero. Ele não sabe nada exceto o que está no buffer. Não é possível referir a um código de outro *buffer* ou em outra *thread*. Com as **funções** isso muda. Quando se define uma função, o

Sonic Pi se lembra dela. Vamos experimentar. Apague todo o código do buffer e substitui por:

```
foo
```

Pressione **Run** - ouve a função tocando? Onde foi o código? Como é que o Sonic Pi sabe o que tocar? O Sonic Pi lembra-se da tua função - mesmo depois de apagar do buffer. Este comportamento apenas funciona com funções criadas usando o '**define**' e '**defonce**'.

Funções parametrizadas

Assim como você pode passar valores mínimos e máximos para 'rrand', você pode ensinar suas funções a aceitar argumentos.

```
define :my_player do |n|
  play n
end
my_player 80
sleep 0.5
my_player 90
```

Isto não é muito excitante, mas ilustra o conceito. Criamos a nossa versão de 'play' chamada 'my_player' que é parametrizada. Os parâmetros seguem **do** do bloco 'define', delimitados pela barra vertical '|' e separados por virgulas ','. Você pode usar qualquer palavra para nomear os parâmetros.

A mágica acontece dentro do bloco 'define'. Você pode querer usar nomes de parâmetros como se fossem valores reais. Neste exemplo estou tocando a nota 'n'. Você pode considerar que os parâmetros são uma espécie de promessa, quando o código é executado os parâmetros são substituídos por valores atuais. Isso ocorre passando um parâmetro às função quando a chamas. Isso é feito com 'my_player 80' para tocar a nota 80. Dentro da definição da função, 'n' é então

substituído por 80, assim ‘play n‘ fica ‘play 80‘. Quando chamar novamente com ‘my_play 90‘, o ‘n‘é substituído por 90 e ‘play n‘fica ‘play 90‘.

Vamos ver um exemplo mais interessante:

```
define :chord_player do |root, repeats|
  repeats.times do
    play chord(root, :minor), release: 0.3
    sleep 0.5
  end
end

chord_player :e3, 2
sleep 0.5
chord_player :a3, 3
chord_player :g3, 4
sleep 0.5
chord_player :e3, 3
```

Aqui ‘repeat‘ foi usado como se fosse um número na linha ‘repeat.times do‘. Também usei ‘root‘ como se fosse o nome de uma nota para ‘play‘.

Veja como podemos escrever algo muito expressivo e fácil à leitura mudando a lógica para uma função!

5.6 Variáveis

Algo muito útil em programação é dar nome às coisa. Escreva o nome de algo que queira usar com um sinal de igual (‘=‘):

```
sample_name = :loop_amen
```

Com isso "registramos"o símbolo ‘:loop_amen‘ na variável ‘sample_name‘ e podemos agora usar ‘sample_name‘em qualquer momento do có-

digo quando quiser chamar ':loop_amen'. Por exemplo:

```
sample_name = :loop_amen
sample sample_name
```

Existem 3 razões principais para usar variáveis no Sonic Pi: comunicar significado, gerenciar repetições e capturar o resultado das coisas.

Comunicar Significado

Quando você escreve um código é fácil pensar que você está dizendo ao computador como fazer coisas - desde que o computador entenda isso. No entanto é importante lembrar que não é só o computador que lê código. Outras pessoas podem ler e tentar entender o que se está acontecendo. Além disso, é provável que você venha ler o código no futuro e tentará que entendê-lo novamente. Apesar do código parecer óbvio para você agora - pode não ser tão óbvio para os outros ou mesmo para você no futuro!

Uma maneira de lidar com isso é adicionando comentários no código (como vimos na secção anterior). Outra forma é usar nomes de variáveis que façam sentido. Vejamos este código:

```
sleep 1.7533
```

Porque usar o numero '1.7533'? De onde ele veio? O que significa? Agora veja este código:

```
loop_amen_duration = 1.7533
sleep loop_amen_duration
```

Agora fica mais claro o que '1.7533' significa: a duração do sample ':loop_amen'! Claro. Por que não escrever?

```
sleep sample_duration(:loop_amen)
```

Claro que esta é uma bela maneira de comunicar a intenção através do código.

Gerenciar Repetições

Frequentemente você encontra muita repetição no seu código e quando quer mudar, acaba tendo que mudar em muitos lugares. Veja este código:

```
sample :loop_amen
sleep sample_duration(:loop_amen)
sample :loop_amen, rate: 0.5
sleep sample_duration(:loop_amen, rate: 0.5)
sample :loop_amen
sleep sample_duration(:loop_amen)
```

Estamos fazendo muitas coisas com o ':loop_amen'! E se quisermos ouvir como soa com outro loop sample como o ':loop_garzul'? Teremos de encontrar e substituir todos os ':loop_amen's por ':loop_garzul'. Isso pode ser Ok se tiveres muito tempo - mas se tiveres a tocar no palco? As vezes não temos esse tempo - especialmente se quiser manter a atenção das pessoas.

E se você escrevesse seu código assim:

```
sample_name = :loop_amen
sample sample_name
sleep sample_duration(sample_name)
sample sample_name, rate: 0.5
sleep sample_duration(sample_name, rate: 0.5)
sample sample_name
sleep sample_duration(sample_name)
```

Experimente. Isto faz exatamente a mesma coisa que o código anterior. Mas agora é possível mudar o sample apenas na linha 'sam-

ple_name = :loop_amen‘ para ‘sample_name = :loop_garzul‘ e todos os outros valores do código se adaptarão por conta da maneira como foi escrito o código. Este é a mágica de escrever com variáveis.

Obtendo informações

Finalmente, um bom motivo para usar variáveis é para obter informações. Por exemplo você pode querer fazer algo com a duração do sample:

```
sd = sample_duration(:loop_amen)
```

Agora podemos usar a variável ‘sd‘ em qualquer lugar que necessitarmos da informação que ela oferecem, ou seja a duração do sample ‘:loop_amen‘.

Talvez o fato mais importante seja que uma variável nos permite capturar a informação de uma execução do ‘play‘ ou ‘sample‘:

```
s = play 50, release: 8
```

Agora capturamos e guardamos a variável ‘s‘, isso nos permite controlar o synth enquanto ele é tocado:

```
s = play 50, release: 8
sleep 2
control s, note: 62
```

Veremos como controlar os *synths* com mais detalhe numa secção posterior.

Atenção com variáveis e threads

Embora as variáveis sejam ótimas para dar nomes às coisas e obter informações, é importante saber que elas normalmente devem ser

usadas localmente dentro de um tópico. Por exemplo, **NÃO FAÇA ISSO:**

```
a = (ring 6, 5, 4, 3, 2, 1)

live_loop :shuffled do
  a = a.shuffle
  sleep 0.5
end

live_loop :sorted do
  a = a.sort
  sleep 0.5
  puts "sorted: ", a
end
```

No exemplo acima, nós atribuímos uma lista (*ring*) de números a uma variável e usamos isso em dois **live_loops** separados. No primeiro **live_loop** a cada 0,5s ordenamos a lista para que toque ((ring 1, 2, 3, 4, 5, 6)) e em seguida a imprima no log. Se você executar o código, verá que a lista impressa nem sempre está ordenada! Isso pode surpreendê-lo, porque às vezes a lista é impressa corretamente e às vezes não. Isso é chamado de comportamento não-determinista e resulta de um problema desagradável chamado *race-condition* (competição condicionada). O problema ocorre porque o segundo **live_loop** também está utilizando a lista (a) (neste caso, arrastando-a) no momento em que a lista é impressa, às vezes ela acabou de ser ordenada (**a.sort**) e, às vezes, ela acaba de ser embaralhada (**a.shuffle**). Ambos os **live_loops** estão competindo para fazer algo diferente com a mesma variável e sempre um loop diferente é que 'ganha'.

Existem duas soluções para isso. Em primeiro lugar, não use a mesma variável em vários loops ou threads ao vivo. Por exemplo, o código a seguir imprimirá sempre uma lista ordenada, pois cada loop ao vivo possui sua própria independentemente sua variável :

```
live_loop :shuffled do
```

```
  a = (ring 6, 5, 4, 3, 2, 1)
  a = a.shuffle
  sleep 0.5
end

live_loop :sorted do
  a = (ring 6, 5, 4, 3, 2, 1)
  a = a.sort
  sleep 0.5
  puts "sorted: ", a
end
```

No entanto, às vezes nós queremos compartilhar coisas em tópicos. Por exemplo, o atual BPM, sintetizador, nota, etc. Nesses casos, a solução é usar o sistema especial de estado do *thread-safe* do Sonic Pi através das funções (fns) **get** e **set**. Isso será discutido mais adiante.

5.7 Sincronização de Thread

Quando estiver mais familiarizado com *live coding* utilzando várias funções e *threads* simultaneamente, irá notar que é bastante fácil gerar um erro numa das *threads* . Isso não é muito problemático porque é só reiniciar **Run**. No entanto, quando reiniciar o *thread* ele ficará "fora do tempo"com as threads originais.

Herança Temporal

Como discutimos anteriormente, novos tópicos criados com **in_thread** herdam as configurações da **thread** original. Isso inclui o horário atual. Ou seja, as threads estão sempre sincronizadas quando iniciados simultaneamente.

No entanto, quando você inicia uma **thread** separadamente, ela começa no seu próprio tempo, o que torna improvável que ela entre em sincronia com outra *thread* em execução.

Cue e Sincronização

Sonic Pi fornece uma solução para este problema com as funções **cue** e **sync**.

Cue nos permite enviar pulsos de mensagens (*heartbeat messages*) para todas **threads**. Por padrão, as **threads** ignoram essas mensagens. No entanto, você pode habilitar o recebimento dos pulsos de mensagens com a função **sync**.

É importante estar ciente de que **sync** é semelhante ao **sleep** no sentido de impedir a **thread** atual de fazer qualquer coisa por um período de tempo. No entanto, com **sleep**, você especifica por quanto tempo deseja aguardar enquanto com **sync** você não sabe quanto tempo irá aguardar, pois a **sync** aguarda a próxima **cue** de outro *thread* que pode ter ocorrido brevemente ou muito tempo depois.

Vamos explorar isso com mais detalhes:

```
in_thread do
  loop do
    cue :tick
    sleep 1
  end
end

in_thread do
  loop do
    sync :tick
    sample :drum_heavy_kick
  end
end
```

Aqui temos duas "threads". Uma que atua como um metrônomo, sem reproduzir som mas enviando ":tick"pulsações de mensagens a cada batida. Outra que está sincronizando as mensagens "tick"e quando recebe uma mensagem, recebe também a hora da **thread** "cue"e continua a ser executada.

Como resultado, ouviremos o exemplo **:drum_heavy_kick** exatamente quando o outra **thread** envia a mensagem **:tick**, mesmo que as duas threads não iniciem ao mesmo tempo:

```
in_thread do
  loop do
    cue :tick
    sleep 1
  end
end

sleep(0.3)

in_thread do
  loop do
    sync :tick
    sample :drum_heavy_kick
  end
end
```

Esse travesso "sleep"normalmente tornaria a segunda linha fora de fase com a primeira. No entanto, à medida que estamos usando **cue** e **sync**, sincronizamos automaticamente as **threads** ignorando qualquer deslocamentos temporal acidental.

Nomeando *sync*

Você é livre para usar o nome que você quiser para as mensagens de **cue**, não apenas **:tick**. Você só precisa garantir que qualquer outra **thread** esteja sincronizada **sync** com o mesmo nome de **cue**, caso contrário, elas estarão sempre esperando (ou pelo menos até você pressionar o botão **Stop**).

Vamos testar alguns exemplos:

```
in_thread do
  loop do
```

```
    cue [:foo, :bar, :baz].choose
    sleep 0.5
  end
end

in_thread do
  loop do
    sync :foo
    sample :elec_beep
  end
end

in_thread do
  loop do
    sync :bar
    sample :elec_flip
  end
end

in_thread do
  loop do
    sync :baz
    sample :elec_blup
  end
end
```

Aqui temos um loop principal de "cue"que envia aleatoriamente um dos nomes dos batimentos cardíacos :foo, :bar ou :baz. Nós também temos três loops de threads sincronizados com um desses nomes de forma independente e, em seguida, reproduzindo um sample diferente. O efeito em rede é que ouvimos um som a cada 0,5 batimentos, pois cada um das threads sync está sincronizada aleatoriamente com o **cue** thread que toca o sample.

Isso, claro, também funciona se você ordenar as threads no sentido inverso, pois as **threads sync** simplesmente aguardam a próxima sugestão **cue** .

Capítulo 6

Efeitos

Um dos aspectos mais gratificantes e divertidos do Sonic Pi é a capacidade de adicionar facilmente efeitos de estúdio aos seus sons. Por exemplo, você pode querer adicionar um *reverb* em partes da sua peça, algum eco ou talvez distorção ou *wooble* em suas linhas de baixo.

O Sonic Pi fornece uma maneira muito simples e poderosa de adicionar efeitos (FX). Ele até permite encadear-los (passando por distorção, eco e depois *reverb*) e além de poder controlar cada efeito individualmente com opções (opts) (assim como os parametros de **synth** e **sample**). Você pode até mesmo modificar as opções do FX enquanto toca. Por exemplo, você poderia aumentar a reverberação do seu baixo durante toda a faixa ...

Efeitos de Guitarra

Se tudo isso parecer um pouco complicado, não se preocupe. Assim que você começar a tocar tudo ficará mais claro. Antes disso, pense nos pedais de guitarra. Existem muitos tipos de pedais que você pode comprar. Alguns adicionam *reverb*, outros *distorcem* etc. Um guitarrista conectará sua guitarra em um pedal de efeito - ou seja de distorção -, então pegue outro cabo e conecte (chain) em um pedal de *reverb*. A saída do pedal de reverbe pode então ser conectada ao amplificador:

Guitar -> Distortion -> Reverb -> Amplifier

Isso é chamado de encadeamento de efeitos. Sonic Pi foi feito para isso. Além disso, cada pedal geralmente possui botões para permitir que você controle a quantidade de distorção, reverberação, eco, etc. No Sonic Pi também é possível ter esse tipo de controle. Finalmente, você pode imaginar um guitarrista tocando enquanto alguém controla os efeitos. Isso também é possível fazer no Sonic Pi - mas em vez de ter alguém que controle as coisas para você, o computador pode fazer essa função.

Vamos explorar alguns efeitos!

6.1 Adicionando Efeitos

Nesta seção, veremos alsungs efeitos (FX): reverb e eco. Veremos como usá-los, como controlar suas **opts** e como encadear-los.

O sistema de efeitos em Sonic Pi usa blocos. Então, se você não leu a seção 5.1, você pode dar uma olhada rápida e depois voltar.

Reverb

Se quiser usar o reverb, escreva **with_fx: reverb** sobre um bloco de código como este:

```
with_fx :reverb do
  play 50
  sleep 0.5
  sample :elec_plip
  sleep 0.5
  play 62
end
```

Agora, toque este código e você ouvirá isso com o reverb. Parece bom, não é? Tudo parece muito legal com reverb.

Agora vamos ver o que acontece se tivermos código fora do bloco **do/end**:

```
with_fx :reverb do
  play 50
  sleep 0.5
  sample :elec_plip
  sleep 0.5
  play 62
end

sleep 1
play 55
```

Observe como o útimo **play 55** não foi tocado com o reverb. Isso ocorre porque está fora do bloco **do/end**, portanto não executa o efeito.

Da mesma forma, se você fizer sons antes do bloco **do / end**, eles não terão efeito:

```
play 55
sleep 1
```

```
with_fx :reverb do
  play 50
  sleep 0.5
  sample :elec_plip
  sleep 0.5
  play 62
end

sleep 1
play 55
```

Eco

Existem muitos efeitos para escolher. Que tal o eco?

```
with_fx :echo do
  play 50
  sleep 0.5
  sample :elec_plip
  sleep 0.5
  play 62
end
```

Um dos aspectos interessante de utilizar os blocos de efeito em Sonic Pi é que suas opts podem ser habilitadas da mesma forma como em **play** e **sample**. Por exemplo, um jeito divertido de utilizar o eco é alterando a **phase:**, ou seja a duração de um determinado eco em batidas. Vamos criar um eco lento:

```
with_fx :echo, phase: 0.5 do
  play 50
  sleep 0.5
  sample :elec_plip
  sleep 0.5
  play 62
end
```

Agora mais lento.

```
with_fx :echo, phase: 0.125 do
  play 50
  sleep 0.5
  sample :elec_plip
  sleep 0.5
  play 62
end
```

Vamos experimentar uma duração maior para o eco definindo sua **decay:** em 8.

```
with_fx :echo, phase: 0.5, decay: 8 do
  play 50
  sleep 0.5
  sample :elec_plip
  sleep 0.5
  play 62
end
```

Encadeando Efeitos

Um dos aspectos mais poderosos dos blocos de efeitos é que você pode encadeá-los. Isso permite que você combine efeitos. Por exemplo, e se você quisesse tocar algum código com eco e depois com reverb? Simplesmente coloque um dentro do outro:

```
with_fx :reverb do
  with_fx :echo, phase: 0.5, decay: 8 do
    play 50
    sleep 0.5
    sample :elec_blup
    sleep 0.5
    play 62
  end
end
```

Pense que o fluxo de áudio parte de dentro para fora. O fonte emissora de todo o código está dentro do bloco **do/end**. Ela é o **play 50**

que passa primeiro pelo eco e que passa depois pelo reverb.

Podemos usar encadeamentos densos que terão resultados alucinantes. No entanto, fique atento, pois nesse tipo de encadeamento de efeitos, você está executando vários processamentos simultaneamente, isso pode consumir bastante do seu computador. Portanto, o uso efeitos pode comprometer a performance, especialmente em plataformas de baixa potência, como o Raspberry Pi.

Descobrindo Efeitos

Sonic Pi vem com efeitos para você tocar. Para descobrir quais estão disponíveis, clique em FX no extremo esquerdo do sistema de ajuda e você verá uma lista de opções disponíveis. Aqui tem alguns dos mais usados:

- wobble
- reverb
- echo
- distortion
- slicer

Aproveite gora para enlouquecer um pouco adicionando efeitos em seus códigos!

6.2 Efeitos na Prática

Embora pareçam simples os efeitos são bastante complexos internamente. Sua simplicidade muitas vezes leva as pessoas a abusar deles em suas peças. Isso pode ser bom se você tiver uma máquina poderosa, mas se você usa o Raspberry Pi, vale ter atenção com a quan-

tidade de processamento que você usa para garantir que as batidas seguirão fluindo.

Considere este código:

```
loop do
  with_fx :reverb do
    play 60, release: 0.1
    sleep 0.125
  end
end
```

Neste código, estamos tocando a nota 60 com um tempo **release** muito curto, então é uma nota curta. Nós também queremos reverb, então embalamos ela um bloco com reverb. Tudo bem até agora. Mas vejamos o que o código faz.

Primeiro, temos um loop que significa que tudo dentro dele é repetido sempre. Em seguida, temos um bloco **with_fx**. Isso significa que vamos criar um novo reverb a cada **loop**. Isto é como ter um pedal de reverb separado para cada vez que você toca uma corda em uma guitarra. É ótimo poder fazer isso, mas nem sempre é o que se deseja. Por exemplo, este código terá dificuldade em executar bem em um Raspberry Pi. Todo o trabalho de criação do reverb e, em seguida, aguardando até que ele precise ser interrompido e removido é tratado pelo **with_fx** para você, mas isso eleva a energia da CPU que é precioso.

Como podemos torná-lo mais parecido com uma configuração tradicional onde o guitarrista tem apenas um pedal de reverb onde todos os sons passam? Simples:

```
with_fx :reverb do
  loop do
    play 60, release: 0.1
    sleep 0.125
  end
end
```

Colocamos o **loop** dentro do bloco **with_fx.** Desta forma, criamos um único reverb para todas as notas tocadas em nosso loop. Este código é muito mais eficiente e funcionaria bem em um Raspberry Pi.

Tente usar efeitos **with_fx** envolvendo uma iteração em loop:

```
loop do
  with_fx :reverb do
    16.times do
      play 60, release: 0.1
      sleep 0.125
    end
  end
end
```

Desta forma, subimos o **with_fx** da parte interna do loop e agora estamos criando um novo reverb a cada 16 notas.

Este é um padrão tão comum que o **with_fx** suporta sem ter que escrever o bloco **16.times**:

```
loop do
  with_fx :reverb, reps: 16 do
    play 60, release: 0.1
    sleep 0.125
  end
end
```

Os exemplos **reps: 16** e **16.times** se comportam de forma idêntica. Os **reps: 16** basicamente repete o código no bloco **do / end** 16 vezes para que você possa usá-los de forma intercambiável e escolher aquele que for melhor para você.

Lembre-se, não há erros, apenas possibilidades. No entanto, algumas dessas abordagens terão um som e características diferentes. Então, explore e use a abordagem que lhe soar melhor, e trabalhar com as limitações de sua plataforma.

Capítulo 7

Controle

Até agora analisamos como tocar synths e samples, e também como modificas as opções padrões: amplitude, pan, envelope entre outras. Cada som disparado é essencialmente seu próprio som com sua lista de opções que definem sua duração do som.

Não seria legal poder alterar as opções de um som ele soa, da mesma foram que faz o guitarrista altera o som das cordas enquanto estão vibrando?

Esta seção irá mostrar como fazer exatamente isso.

7.1 Controlando Synths em Tempo Real

Sonic Pi oferece a capacidade de manipular e controlar sons em execução. Fazemos isso usando uma variável para capturar uma referência a um sintetizador:

```
s = play 60, release: 5
```

Aqui, temos uma variável *run-local* **s** que representa a nota de reprodução do synth 60. Observe que esta é executada localmente - você não pode acessá-la de outras operações como funções.

Uma vez que temos **s**, podemos começar a controlá-lo através da função de controle:

```
s = play 60, release: 5
sleep 0.5
control s, note: 65
sleep 0.5
control s, note: 67
sleep 3
control s, note: 72
```

Note que não estamos desencadeando quatro sintetizadores diferentes. Estamos apenas acionando um sintetizador e depois mudando o tom dele três vezes enquanto ele toca.

Podemos alterar qualquer uma das opções padrões para controlar, para que você possa controlar coisas como amp:, cutoff: ou pan:.

Opções Não Controláveis

Uma vez que o sintetizador esteja tocando algumas das opts não podem ser controladas. Este é o caso de todos os parâmetros do envelope ADSR. Você pode descobrir quais opts são controláveis, observando a documentação deles no sistema de ajuda. Se a documentação informar que uma vez definida não pode ser alterada , você saberá que não é possível controlar a opção após o início do sintetizador.

7.2 Controlando Efeitos

Também é possível controlar os efeitos, embora isso ocorra de uma maneira ligeiramente diferente:

```
with_fx :reverb do |r|
  play 50
  sleep 0.5
  control r, mix: 0.7
  play 55
  sleep 1
  control r, mix: 0.9
  sleep 1
  play 62
end
```

Em vez de usar uma variável, usamos os parâmetros no poste de meta do bloco **do/end**. Dentro das barras ", precisamos especificar um nome específico para o efeito em execução, que então fazemos referência ao bloco contendo do/end. Esse comportamento é idêntico ao uso de funções parametrizadas.

Agora tente controlar alguns sintetizadores e efeitos!

7.3 Opção Slide

Enquanto explorava as opções dos sintetizador e efeitos, você pode ter percebido que existem também opções que terminam com o *_slide*. Se você tentou utilizar os slides deve ter percebido que eles não tiveram efeito. Isso acontece porque os *slides* não são parâmetros normais, eles são **opts** especiais e só funcionam quando você controla sintetizadores como mostramos na seção anterior.

Considere o seguinte exemplo:

```
s = play 60, release: 5
```

```
sleep 0.5
control s, note: 65
sleep 0.5
control s, note: 67
sleep 3
control s, note: 72
```

Aqui, você pode ouvir o som do sintetizador mudar imediatamente a cada chamada de controle. No entanto, podemos querer que o tom seja deslocado entre as mudanças. Como estamos controlando a nota: parâmetro, para adicionar slide, precisamos definir o parâmetro *note_slide* do sintetizador:

```
s = play 60, release: 5, note_slide: 1
sleep 0.5
control s, note: 65
sleep 0.5
control s, note: 67
sleep 3
control s, note: 72
```

Agora, ouvimos as notas deslizando entre as chamadas de controle. Parece bom, não é? Você pode acelerar o slide usando um tempo mais curto, como *note_slide: 0.2* ou desacelerar usando um tempo de deslize mais longo.

Cada parâmetro que pode ser controlado tem seu slide correspondente para você usar.

O Slide Cola

Depois de configurar um parâmetro *_slide* em um sintetizador em execução, ele será lembrado e usado sempre que você deslizar o parâmetro correspondente. Para parar de deslizar, você deve definir o valor *_slide* para 0 antes da próxima chamada de controle.

Slide FX opts

Também é possível deslizar as opções dos efeitos:

```
with_fx :wobble, phase: 1, phase_slide: 5 do |e|
  use_synth :dsaw
  play 50, release: 5
  control e, phase: 0.025
end
```

Aproveite para deslizar parâmetros e fazer transições suaves de controle e fluxo nos seus códigos.

Capítulo 8

Estruturas de dados

Um instrumento que todo programador deve ter em sua caixa de ferramentas é uma estrutura de dados.

Às vezes você pode querer representar e usar mais de uma coisa. Por exemplo, você pode achar útil ter uma série de notas para tocar sequencialmente. As linguagens de programação possuem estruturas de dados que permitem fazer exatamente isso.

Existem muitas estruturas de dados interessantes e exóticas disponíveis - e as pessoas estão sempre inventando novas. Por enquanto vamos considerar uma estrutura de dados muito simples - a lista.

Vejamos com mais detalhe sua forma básica e, em seguida, também como as listas podem ser usadas para representar escalas e acordes.

8.1 Listas

Nesta seção, veremos uma estrutura de dados muito útil - a lista. Vimos isso rapidamente na seção de *randomização* quando escolhemos aleatoriamente notas em um lista para tocar:

```
play choose([50, 55, 62])
```

Nesta seção, exploraremos o uso das listas para representar acordes e escalas. Primeiro vamos recapitular como tocar um acorde. Lembre-se que, se não usarmos o sleep, tudo acontecerá ao mesmo tempo:

```
play 52
play 55
play 59
```

Vejamos outras formas de representar este código.

Tocando Lista

Uma opção é colocar todas as notas em uma lista: [52, 55, 59]. A função **play** é amigável e inteligente o suficiente para saber como tocar uma lista de notas. Tente:

```
play [52, 55, 59]
```

Oh, isso já é mais agradável de ler. A reprodução de uma lista de notas não impede que você use qualquer um dos parâmetros como usual:

```
play [52, 55, 59], amp: 0,3
```

Claro, você também pode usar os nomes das notas tradicionais em vez dos números MIDI:

```
play [:E3,:G3,:B3]
```

Agora, aqueles que tiveram a oportunidade de estudar teoria musical poderão reconhecer esse acorde. Um Mi menor tocado na 3a oitava.

Acessando Lista

Outra característica muito útil de uma lista é a capacidade de obter informações sobre ela. Isso pode soar um pouco estranho, mas não é mais complicado do que alguém pedir para você abrir um livro na página 23. Com uma lista, você diria, qual o elemento no índice 23? A única diferença é que os índices de programação geralmente começam em 0 e não 1.

Com índices de lista, não contamos 1, 2, 3 ... Em vez disso, contamos 0, 1, 2 ...

Vejamos isso com mais detalhes. Dê uma olhada nesta lista:

```
[52, 55, 59]
```

Não há nada especialmente assustador sobre ela. Agora, qual o segundo elemento dessa lista? Sim, claro, é 55. Isso foi fácil. Vamos ver se podemos conseguir o computador para responder também para nós:

```
puts [52, 55, 59] [1]
```

OK, isso pode parece um pouco estranho se você nunca viu nada assim antes. Porém, não é muito difícil. Existem três partes na linha acima: primeiro a palavra *puts*; depois a lista 52, 55, 59 e por fim o índice [1]. Primeiramente a palavra *puts* insere uma informação ao Sonic Pi que depois será impressa no log. Em seguida, estamos

dando a nossa lista e, finalmente, nosso índice pede o segundo elemento. Precisamos cercar o índice com colchetes e pelo fato da contagem começar em 0, o índice para o segundo elemento é 1. Confira:

```
# índices:  0   1   2
           [52 ,55, 59]
```

Tente executar o código *puts [52, 55, 59] [1]* e você verá 55 aparecer no log. Altere o índice 1 para outros índices, tente listas mais longas e pense em como você pode usar uma lista em seu próximo código. Por exemplo, quais estruturas musicais podem ser representadas como uma série de números ...

8.2 Acordes

O Sonic Pi suporta nomes de acordes que irão retornar listas. Experimente por exemplo:

```
play chord(:E3, :minor)
```

Agora, estamos realmente chegando a algum lugar. Isso parece muito mais simples do que as listas (e é mais fácil de ler para outras pessoas). Então, quais outros acordes o Sonic Pi suporta? Bem, muitos. Experimente alguns destes:

- chord(:E3, :m7)
- chord(:E3, :minor)
- chord(:E3, :dim7)
- chord(:E3, :dom7)

Arpegio

Podemos facilmente transformar acordes em arpejos com a função **play_pattern:**

```
play_pattern chord(:E3, :m7)
```

Ok, isso não é tão divertido - tocou muito devagar. O *play_pattern* reproduz cada nota de uma lista no intervalo *sleep 1*. Podemos usar outra função **play_pattern_timed** para especificar o intervalo de tempo e acelerar um pouco as coisas:

```
play_pattern_timed chord(:E3, :m7), 0.25
```

Podemos até mesmo passar uma lista com os tempos que irão será executada em loop:

```
play_pattern_timed chord(:E3, :m13), [0.25, 0.5]
```

Isso é equivalente a:

```
play 52
sleep 0.25
play 55
sleep 0.5
play 59
sleep 0.25
play 62
sleep 0.5
play 66
sleep 0.25
play 69
sleep 0.5
play 73
```

Qual você prefere escrever?

Escalas

Sonic Pi suporta uma ampla gama de escalas. Que tal tocar uma escala maior de dó na 3a oitava?

```
play_pattern_timed scale(:c3, :major), 0.125, release: 0.1
```

Podemos convocar um maior número oitavas:

```
play_pattern_timed scale(:c3, :major, num_octaves: 3), 0.125, release: 0.1
```

E que tal uma escala pentatônica?

```
play_pattern_timed scale(:c3, :major_pentatonic, num_octaves: 3), 0.125, release: 0.1
```

Notas Aleatórias

Utilizar acordes e escalas é uma ótima maneira de restringir uma escolha aleatória a algo significativo. Experimente com este exemplo que escolhe notas aleatórias de um acorde E3 menor:

```
use_synth :tb303
loop do
  play choose(chord(:E3, :minor)), release: 0.3, cutoff: rrand(60, 120)
  sleep 0.25
end
```

Tente explorar acordes e extensões diferentes para *cutoff:*.

Descobrindo Acordes e Escalas

Para descobrir quais acordes e escalas que o Sonic Pi suporta, clique no botão *Lang* do sistema de ajuda. Depois escolha *chord* (acorde)

ou *scale* (escala) na lista da API. Nas informações role a janela para baixo até encontrar uma longa lista de acordes ou escalas.

Divirta-se e lembre-se: não existem erros, apenas oportunidades.

8.3 Rings

Uma forma interessante de rodar o padrão de listas é o *ring*. Se você conhece algo de programação provavelmente já ouviu falar de *ring buffer* ou *ring array*. Aqui, mostraremos rapidamente como é simples de usar *ring* .

Na seção anterior vimos como extrair elementos de listas usando o mecanismo de indexação:

```
puts[52, 55, 59] [1]
```

Agora, o que aconteceria se buscarmos um índice maior que o número de elementos da lista? Por exemplo 100? Bem, não existem 100 elementos pois a lista tem apenas três números. Então, Sonic Pi irá retornar *nil* (nada).

No entanto, imagine que você tem um contador, como o ritmo atual que aumenta continuamente. Vamos criar um contador e uma lista:

```
counter = 0
notes = [52, 55, 59]
```

Podemos usar um contador para acessar uma nota da lista:

```
puts notes[counter]
```

Ótimo, temos 52. Agora, vamos incrementar nosso contador e obter outra nota:

```
counter = (inc counter)
puts notes[counter]
```

Fantástico, agora temos 55 e, se fizermos de novo, obtemos 59. No entanto, se o fizermos mais uma vez, ficaremos sem números na nossa lista e obteremos *nil*. E se quisermos apenas voltar a rodar a lista do início? É para isso que serve o *ring*.

Criando Rings

Podemos criar anéis de duas maneiras. Ou usamos a função *ring* com os elementos do *ring* entre parâmetros:

```
(ring 52, 55, 59)
```

Ou pegamos uma lista normal e convertemos em um *ring* enviando a mensagem *.ring*:

```
[52, 55, 59] .ring
```

Indexando Rings

Uma vez que temos o *ring*, ele pode ser usado igual uma lista. A única exceção é que é possível usar índices negativos ou maiores do que o tamanho do *ring* e eles vão rodar a lista até apontar um dos elementos do anel:

```
(ring 52, 55, 59)[0]  #=> 52
(ring 52, 55, 59)[1]  #=> 55
(ring 52, 55, 59)[2]  #=> 59
(ring 52, 55, 59)[3]  #=> 52
(ring 52, 55, 59)[-1] #=> 59
```

Usando Rings

Digamos que estejamos usando uma variável para representar o número da batida atual. Nós podemos usar isso como um índice no ring para selecionar as notas a serem tocadas, ou tempos do *:release* bem como qualquer coisa útil armazenada no ring, independentemente do número de batidas que esteja.

Escalas e Acordes São Rings

Uma informação importante é que as escalas e acordes também são *rings* e permitem que você as acessem com índices arbitrários.

Construções de Rings

Além de tocar, há uma série de outras funções que constroem *rings*.

- **range** convida você a especificar um ponto de partida, ponto final e tamanho de cada etapa.
- **bools** permite que você use 1s e 0s para representação booleana.
- **knit** permite que você faça uma série de valores repetidos.
- **spread** cria um *ring* de bools com uma distribuição euclidiana.

Dê uma olhada na documentação para obter mais informações.

8.4 Encadeamento de Ring

Além dos construtores, como *range* e *spread*, uma outra maneira de criar novos *rings* é manipular *rings* existentes.

Comandos em Cadeia

Para explorar isso, faça um simples *ring*:

```
(ring 10, 20, 30, 40, 50)
```

E se desejássemos inverter? Bem, usamos o comando em cadeia *reverse* para transformá-lo:

```
(ring 10, 20, 30, 40, 50).reverse #=> (ring 50, 40, 30, 20, 10)
```

E se quisermos os três primeiros elementos?

```
(ring 10, 20, 30, 40, 50).take(3) #=> (ring 10, 20, 30)
```

Por fim, se quisermos embaralhar o *ring*?

```
(ring 10, 20, 30, 40, 50).shuffle  #=> (ring 40, 30, 10, 50, 20)
```

Encadeamento Múltiplo

Esta é uma maneira eficiente de criar novos *rings*. No entanto, sua real eficácia está na forma de encadear alguns desses comandos juntos.

Que tal arrastar o *ring*, soltar um elemento e depois seguir os 3 próximos?
Vamos fazer isso por etapas:

1. (ring 10, 20, 30, 40, 50) - ring inicial
2. (ring 10, 20, 30, 40, 50).shuffle - embaralhar - (ring 40, 30, 10, 50, 20)
3. (ring 10, 20, 30, 40, 50).shuffle.drop(1) - largar 1 - (ring 30, 10, 50, 20)
4. (ring 10, 20, 30, 40, 50).shuffle.drop(1).take(3) - pegar 3 - (ring 30, 10, 50)

Veja como é simples criar uma longa cadeia destes métodos apenas colocando eles juntos. Podemos combiná-los em qualquer ordem, criando uma forma extremamente interessante de gerar novos *rings*.

Imutabilidade

Esses *rings* possuem uma propriedade importante. Eles são imutáveis, o que significa que eles não podem mudar. Isso significa que os métodos de encadeamento descritos nesta seção não alteram os *rings* e ao mesmo tempo criam novos. Isso significa que você é livre para compartilhar *rings* em tópicos e começar a encadeá-los dentro de um tópico sabendo que não estará afetando qualquer outro segmento usando o mesmo *ring*.

Métodos de Encadeamento

Aqui existe uma lista dos métodos de cadeia disponíveis para você explorar:

- *.reverse* - inverte o ring
- *.sort* - ordena o ring
- *.shuffle* - embaralha
- *.pick(3)* - pega 3 elementos convocanto *.choose* 3 vezes
- *.pick* - similar to .pick(3) only the size defaults to the same as the original ring
- *.take(5)* - novo ring com os 5 primeiros elementos
- *.drop(3)* - novo ring excluindo os 3 primeiros elementos
- *.butlast* - novo ring sem o ultimo elemento
- *.drop_last(3)* - novo ring sem os 3 últimos elementos
- *.take_last(6)*- novo ring com os 6 últimos elementos
- *.stretch(2)* - repete cada elemento duas vezes
- *.repeat(3)* - repete o ring 3 vezes
- *.mirror* - adiciona uma versão invertida do ring
- *.reflect* - igual ao *mirror* sem duplicar o valor do meio
- *.scale(2)* - novo ring com todos os elementos multiplicados por 2 (assume que o anel contém apenas números)

Claro que esses métodos de encadeamentos podem ter números diferentes dos apresentados acima! Então, sinta-se à vontade para escrever *.drop(5)* em vez de *.drop (3)* e assim por diante.

Capítulo 9

Live Coding

Um dos aspectos mais instigantes do Sonic Pi é que ele permite você escrever e modificar o código ao vivo enquanto faz música. Algo parecido com tocar violão ao vivo. Uma das vantagem desta abordagem é a inter-relação que existe entre criar o código e ao mesmo tempo improvisar com ele ao vivo, podendo tocar com Sonic Pi no palco.

Nesta seção, abordaremos alguns fundamentos para transformar seus códigos e composições em performances dinâmicas.

9.1 Fundamentos do Live Controling

Até aqui aprendemos o bastante para começar a aprofundar em estratégias de performance ao vivo com código. Com exemplos das seções anteriores, mostraremos como você pode começar a criar suas composições musicais e transformá-las em uma performance. Para

isso, precisamos de 3 coisas:

- A capacidade de escrever código que produz sons - OK!
- A capacidade de escrever funções - OK!
- A capacidade de nomear *threads* - OK!

Então vamos começar. Vamos ensaiar nossos primeiro *live coding*. Para isso, precisamos de uma função que contenha o código que queremos tocar. Lembre-se de começar simples! Também queremos fazer um loop que chame essa função em uma *thread*:

```
define :my_loop do
  play 50
  sleep 1
end

in_thread(name: :looper) do
  loop do
    my_loop
  end
end
```

Se isso parecer um pouco complicado para você, volte e releie as seções sobre funções e *threads*. Não é muito complicado se você acondicionou sua cabeça sobre essas coisas.

O que fizemos aqui foi definir uma função (*:my_loop*) que toca a nota 50 e aguarda um segundo (sleep 1). Em seguida, definimos um tópico (*in_thread*) com o nome *:looper*. Essa thread apenas executa em loop a função *:my_loop* .

Se você executar este código, você ouvirá a nota 50 repetindo várias vezes ...

Alterando Código ao Vivo

Agora, é aqui que começa a diversão. Enquanto o código está em execução, mude 50 para outro número, 55 por exemplo, em seguida, pressione o botão **Run** novamente. Uoh! Mudou! Viva!

Não adicionou uma nova camada porque estamos usando threads endereçadas que permitem apenas um tópico para cada nome. Além disso, o som mudou porque redefinimos a função. Demos para *:my_loop* uma nova definição. Enquanto o tópico *thread: looper* executa em loop chamou a nova definição.

Tente alterar o código novamente, mude a nota, o tempo de *sleep*. E se adicionarmos um *synth* declarando *use_synth*?

```
define :my_loop do
  use_synth :tb303
  play 50, release: 0.3
  sleep 0.25
end
```

Soa bem! Não? Que tal apimentar um pouco mais? Em vez de tocar a mesma nota repetidas vezes, tente tocar um acorde:

```
define :my_loop do
  use_synth :tb303
  play chord(:e3, :minor), release: 0.3
  sleep 0.5
end
```

Que tal escolher notas aleatórias de um acorde:

```
define :my_loop do
  use_synth :tb303
  play choose(chord(:e3, :minor)), release: 0.3
  sleep 0.25
end
```

Ou então usar um valor randômico para (*:cutoff*):

```
define :my_loop do
  use_synth :tb303
  play choose(chord(:e3, :minor)), release: 0.2,
  cutoff: rrand(60, 130)
  sleep 0.25
end
```

Finalmente, adicione umas batidas.

```
define :my_loop do
  use_synth :tb303
  sample :drum_bass_hard, rate: rrand(0.5, 2)
  play choose(chord(:e3, :minor)), release: 0.2,
  cutoff: rrand(60, 130)
  sleep 0.25
end
```

Agora, as coisas estão ficando um pouco mais interessantes!

No entanto, antes de continuar o *live coding* explorando funções e *threads*, pare o que você está fazendo e leia a próxima seção sobre *live_loop*. Ela mudará para sempre sua maneira escrever no Sonic Pi ...

9.2 Live Loops

Esta seção do tutorial é uma verdadeira preciosidade. Se você quer ler apenas uma seção, esta é a indicada. Se você leu a seção anterior sobre Fundamentos do *live coding*, live_loop é uma maneira simples de fazer o mesmo, mas sem ter que escrever muito.

Se você não leu a seção anterior, saiba que live_loop é a melhor maneira de interferir com o Sonic Pi.

Vamos tentar então escrever o seguinte em um novo buffer:

```
live_loop :foo do
```

```
  play 60
  sleep 1
end
```

Agora, pressione o botão **Run**. Você ouve um sinal sonoro básico a cada pulso. Nada divertido ainda. Mas não pressione **Stop** ainda. Altere o número 60 para 65 e pressione **Run** novamente.

Muito bem! Mudou o som automaticamente sem perder a batida. Isto que é *live coding*.

Por que não mudar para um som mais grave? Basta atualizar seu código enquanto toca:

```
live_loop :foo do
  use_synth :prophet
  play :e1, release: 8
  sleep 8
end
```

Então, pressione **Run**.

Vamos fazer com que *cutoff:* se mova:

```
live_loop :foo do
  use_synth :prophet
  play :e1, release: 8, cutoff: rrand(70, 130)
  sleep 8
end
```

Pressione **Run** novamente. Adicione um pouco de batida.

```
live_loop :foo do
  sample :loop_garzul
  use_synth :prophet
  play :e1, release: 8, cutoff: rrand(70, 130)
  sleep 8
end
```

Mude a nota :e1 para :c1.

```
live_loop :foo do
  sample :loop_garzul
  use_synth :prophet
  play :c1, release: 8, cutoff: rrand(70, 130)
  sleep 8
end
```

Agora pare de seguir o tutorial e experimente um pouco!

9.3 Vários Live Loops

Veja o seguinte *live_loop*:

```
live_loop :foo do
  play 50
  sleep 1
end
```

Você pode ter se perguntado por que incluir o nome *:foo*. Esse nome é importante porque significa que esse *live_loop* é diferente de todos os outros *live_loop*s. Nunca pode haver dois *live_loop*s com o mesmo nome.

Isso significa que, se quisermos executar múltiplas *live_loop*, apenas precisamos dar-lhes nomes diferentes:

```
live_loop :foo do
  use_synth :prophet
  play :c1, release: 8, cutoff: rrand(70, 130)
  sleep 8
end

live_loop :bar do
  sample :bd_haus
  sleep 0.5
end
```

Agora você pode atualizar e alterar cada loop ao vivo de forma independente e tudo funciona.

Sincronizando live_loops

Uma coisa que você talvez já tenha notado é que os *live_loop*s funcionam automaticamente com o mecanismo de *thread cue* que exploramos anteriormente. Toda vez que os *live_loop* retorna, ele gera um novo evento de *cue* com o nome do *live_loop*. Portanto, podemos sincronizar *cue* para garantir que nossos loops estejam sincronizados sem ter que parar nada.

Veja este código sincronizado incorretamente:

```
live_loop :foo do
  play :e4, release: 0.5
  sleep 0.4
end

live_loop :bar do
  sample :bd_haus
  sleep 1
end
```

Vejamos se é possível corrigir o tempo e sincronizar sem interromper a performance. Primeiro, vamos consertar o *loop :foo* para fazer que o fator *sleep* 1 - seja algo como 0.5:

```
live_loop :foo do
  play :e4, release: 0.5
  sleep 0.5
end

live_loop :bar do
  sample :bd_haus
  sleep 1
end
```

Ainda não acabamos, você notará que as batidas não se alinham corretamente. Isso ocorre porque os loops estão fora de fase. Vamos consertar isso sincronizando um com outro. Para isso é adicionado no bloco *live_loop :bar* a linha *sync :foo* que sincroniza com o *live_loop :foo*. Ouça!

```
live_loop :foo do
  play :e4, release: 0.5
  sleep 0.5
end

live_loop :bar do
  sync :foo
  sample :bd_haus
  sleep 1
end
```

Uau, tudo está perfeitamente no tempo - tudo sem parar.

Agora, siga em frente seu *live coding* com *live_loops*!

9.4 *Ticking*

Algo que provavelmente você encontrará muito fazendo nas sessões de live code será o ring loop. Você colocará notas em *rings* para criar melodias, ritmos, encadeamento de acordes, variações timbrística, etc.

Sistema Tiquetaque

Sonic Pi é uma ferramenta muito útil para trabalhar com *rings* dentro de *live_loops*. Isso é chamado de sistema tiquetaque (*tick system*) . Na seção sobre os *rings*, falamos sobre o contador que aumenta constantemente, com a batida atual. *Tick* apenas implementa essa ideia. Ele fornece a habilidade de marcar os *rings*. Vejamos um exemplo:

```
counter = 0
live_loop :arp do
  play (scale :e3, :minor_pentatonic)[counter], release: 0.1
  counter += 1
  sleep 0.125
end
```

Isso é igual a

```
live_loop :arp do
  play (scale :e3, :minor_pentatonic).tick, release: 0.1
  sleep 0.125
end
```

Aqui, estamos pegando a escala pentatônica em mi menor na 3a oitava e tocando cada nota. Isso é feito adicionando *.tick* ao final da declaração da escala. Esse *.tick* é local para o live_loop, assim cada *live_loop* pode ter seu próprio controle independente:

```
live_loop :arp do
  play (scale :e3, :minor_pentatonic).tick, release: 0.1
  sleep 0.125
end

live_loop :arp2 do
  use_synth :dsaw
  play (scale :e2, :minor_pentatonic, num_octaves: 3).tick,
  release: 0.25
  sleep 0.25
end
```

Tick

Você também pode chamar *tick* como uma função padrão (*fn*) e usar o valor como um índice:

```
live_loop :arp do
  idx = tick
```

```
  play (scale :e3, :minor_pentatonic)[idx], release: 0.1
  sleep 0.125
end
```

No entanto, é muito melhor convocar *.tick* no final. A *função tick*é para quando você quer fazer coisas extravagantes com o valor do *tick* e para quando você quer usar *tick*s para outras coisas além de indexar *ring*s.

Look

A coisa mágica sobre o *tick* é que não só retorna um novo índice (ou o valor do *ring* nesse índice) mas também garante que a próxima vez que você chamar o *tick*, será o valor seguinte. Dê uma olhada nos exemplos de tick nos documentos lá você encontrará muitas maneiras de trabalhar com ele. No entanto, por enquanto, é importante ressaltar que às vezes você quer apenas ver o valor atual do tick e não incrementá-lo. Isso é disponível através do *look fn*. Você pode chamar *look* como uma fn padrão ou adicionando *.look* ao final de um *ring*.

Nomeando *Tick*

Finalmente, às vezes você precisará de mais de um *tick* por *live_loop*. Isto é possível dando um nome ao *tick*:

```
live_loop :arp do
  play (scale :e3, :minor_pentatonic).tick(:foo), release: 0.1
  sleep (ring 0.125, 0.25).tick(:bar)
end
```

Aqui estamos usando dois *tick*s: um para tocar a nota e outro para a duração de sleep. Como eles estão ambos no mesmo live_loop, para mantê-los independentes, é preciso dar-lhes nomes únicos. Este é

exatamente o mesmo tipo de nomeando *live_loops* - apenas passamos um símbolo com um prefixo:. No exemplo acima, chamamos um *tick* de *:foo* e o outro de *:bar*. Se quisermos ver isso, também precisamos passar o nome do tick para *look*.

Não Complique

A maior parte do poder no sistema de tiques não é útil quando você está começando. Não tente aprender tudo nesta seção. Apenas se concentre em assinalar um único *ring*. Isso lhe dará mais diversão e simplicidade para assinalar os *ring*s em seu *live_loops*.

Dê uma olhada na documentação, nela há muitos exemplos úteis e táticas sobre isso!

Capítulo 10

Estado Temporal

Muitas vezes, é útil ter informações compartilhadas em várias *threads* ou *live_loop*s. Por exemplo, você pode querer compartilhar uma noção da tonalidade, do BPM ou ainda de conceitos abstratos a ser interpretado de maneiras diferentes em diversas *threads*). Também não queremos perder nenhuma dos determinismos existentes ao fazer isso. Em outras palavras, gostaríamos ainda de poder compartilhar código com outros e saber exatamente o que eles ouvirão quando o executarem. No final da Seção 5.6 deste tutorial, discutimos brevemente por que não devemos usar variáveis ??para compartilhar informações através de threads devido a perda de determinismo (por sua vez, devido aos conflitos entre as condicionais).

A solução de Sonic Pi para o problema de trabalhar facilmente com variáveis ??globais de forma determinista é o Estado Temporal (*Time State*). Isso pode parecer complexo e difícil (na verdade, no Reino Unido, a programação com vários tópicos e a memória compartilhada geralmente é um assunto universitário). No entanto, como

você verá, assim como tocar sua primeira nota, o Sonic Pi torna incrivelmente simples compartilhar o estado em todas as *threads*, mantendo o determinismo de seus programas seguros.

Conheça o *set* e o *get* ...

10.1 *Set* e *Get*

Sonic Pi tem um armazenamento de memória global chamada *Time State*. As duas principais coisas que ela faz são: definir *set* e obter *get* informações. Vamos mergulhar um pouco nisso ...

Conjunto

Para armazenar informações no Time State, precisamos de duas coisas:

1. a informação que queremos armazenar,
2. um nome exclusivo (chave) para a informação.

Por exemplo, podemos querer armazenar o número 3000 com a chave: intensidade. Isso é possível usando a função *set*:

```
set: intensidade, 3000
```

Podemos usar qualquer nome para nossa chave. Se várias informações tiverem sido armazenadas na chave, a ultima substituirá a anterior:

```
set: intensidade, 1000
set: intensidade, 3000
```

No exemplo acima, ao armazenar os dois números sob a mesma chave, o último *set* 'ganha', então o número associado a *:intensidade* será 3000, portanto o primeiro *set* será efetivamente anulado.

Get

Para buscar informações no *Time State*, precisamos apenas da chave que costumávamos definir, o que, em nosso caso, é *:intensidade*. Então, precisamos utilizar *get [: intensidade]* que podemos ver o resultado impresso no log:

print get [: intensidade] # => imprimir 3000

Observe que ao convocar uma informação com *get* você poderá ter informações definidas da execução anterior. Uma vez que a informação foi definida pelo *set* ela estará armazenada até ser substituída ou até o Sonic Pi ser encerrado.

Múltiplas *Threads*

O principal benefício do sistema de armazenamento do *Time State* é que ele pode ser usado com segurança em *threads* ou *live_loops*. Por exemplo, você poderia ter um live_loop para configurar informação *set* e outro live_loop para obter informação *get*:

```
live_loop :setter do
  set :foo, rrand(70, 130)
  sleep 1
end

live_loop :getter do
  puts get[:foo]
  sleep 0.5
end
```

A vantagem de o usar *set* e *get* nas *threads* é de sempre produzir

o mesmo resultado quando você for executar o código. Continue e experimente. Veja se você obtém o seguinte no seu log:

```
run: 0, time: 0.0 ?? 125.72265625
run: 0, time: 0.5 ?? 125.72265625
run: 0, time: 1.0 ?? 76.26220703125
run: 0, time: 1.5 ?? 76.26220703125
run: 0, time: 2.0 ?? 114.93408203125
run: 0, time: 2.5 ?? 114.93408203125
run: 0, time: 3.0 ?? 75.6048583984375
run: 0, time: 3.5 ?? 75.6048583984375
```

Tente executá-lo algumas vezes - veja se o resultado é igual sempre. Isso é o que chamamos de comportamento determinista e é realmente muito importante quando queremos compartilhar nossa música através de código e saber se a pessoa que está tocando o código está ouvindo exatamente o que queríamos que eles ouvissem (assim como tocar um arquivo MP3 pela internet, que soa igual para todos os ouvintes).

Sistema Determinista Simples

Na Seção 5.6, discutimos porque o uso de variáveis em *threads* pode levar a um comportamento aleatório. Isso nos impede de executar de forma confiável um código como este:

```
## Um exemplo de comportamento não determinista
## (devido a condições de conflitos causadas por múltiplas
## live_loops manipulando a mesma variável
## ao mesmo tempo).
##
## Se você executar este código, você notará
## que a lista impressa é
## nem sempre está ordenada!
a = (ring 6, 5, 4, 3, 2, 1)

live_loop :shuffled do
```

```
  a = a.shuffle
  sleep 0.5
end

live_loop :sorted do
  a = a.sort
  sleep 0.5
  puts "sorted: ", a
end
```

Vamos dar uma olhada usando *get* and *set*:

```
## Um exemplo de comportamento determinista
## (apesar do acesso simultâneo do estado compartilhado)
# usando o Time State da Sonic Pi.
##
## Quando este código é executado, a lista que é
## impressa estará sempre ordenada!
set :a, (ring 6, 5, 4, 3, 2, 1)

live_loop :shuffled do
  set :a, get[:a].shuffle
  sleep 0.5
end

live_loop :sorted do
  set :a, get[:a].sort
  sleep 0.5
  puts "sorted: ", get[:a]
end
```

Observe como esse código é praticamente idêntico à versão anterior. No entanto, quando você executa o código, ele faz a mesma coisa sempre neste caso, graças ao sistema *Time State*.

Portanto, ao compartilhar informações em live_loops e threads, use *get* e *set* em vez de variáveis para obter um comportamento determinístico e reprodutível.

10.2 *Sync*

Na Seção 5.7 introduzimos as funções *cue* e *sync* ao lidar com a questão da sincronização de *threads*. O que não explicamos foi que o sistema Time State é quem fornece essa funcionalidade. Acontece que *set* é uma variação de *cue* e é construído em cima da mesma funcionalidade que insere informações no sistema *Time State*. Além disso, *sync* também foi projetado para funcionar perfeitamente com o Time State - qualquer informação que planejamos armazenar no Time State em que podemos sincronizar. Em outras palavras - nós sincronizamos eventos que ainda não foram inseridos no Time State.

À Espera de Eventos

Vamos dar uma rápida olhada em como usar *sync* para aguardar novos eventos a serem adicionados no Time State:

```
in_thread do
  sync :foo
  sample :ambi_lunar_land
end

sleep 2

set :foo, 1
```

Neste exemplo, primeiro criamos uma *thread* que aguarda o evento *:foo* ser adicionado ao *Time State*. Após a declaração da *thread*, ela espera por 2 tempos e define *:foo* como 1. Isso libera *sync*, que então se move para a próxima linha, que dispara o *sample : ambi_lunar_land*.

Observe que *sync* sempre aguarda eventos futuros e que bloqueará a *thread* atual esperando um novo evento. Além disso, herdará o tempo lógico da *thread* desencadeada por meio de *set* ou *cue* para que também possa ser usado para sincronizar o tempo.

Passando Valores para o Futuro

No exemplo acima, definimos *:foo 1*, com o qual não fizemos nada. Na verdade, podemos obter esse valor para a *thread* convocando-o com *sync*:

```
in_thread do
  amp = sync :foo
  sample :ambi_lunar_land, amp: amp
end

sleep 2

set :foo, 0.5
```

Observe que os valores informados através de *set* e *cue* devem ser salvos na *thread* - ou seja, *rings*, números, símbolos ou listas. Sonic Pi informará erro se o valor que você estiver tentando armazenar no Time State não for válido.

10.3 Correspondência de Padrões

Ao obter e configurar informações no *Time State*, é possível usar chaves mais complexas do que símbolos básicos, tais como *:foo* e *:bar*. Você também pode usar o estilo URL de endereçamento como "/foo/bar/baz". Uma vez que começamos a trabalhar com endereçamento, podemos começar a aproveitar o sofisticado sistema de correspondência de padrões da Sonic Pi para *get* e *sync* com caminhos "semelhantes"em vez de "idênticos". Vamos dar uma olhada.

Combinando Diretórios

Vamos supor que queremos aguardar o próximo evento que tem três segmentos de caminho:

```
sync "/*/*/*"
```

Isso combinará qualquer evento Time State com exatamente três segmentos de caminho, independentemente de seus nomes. Por exemplo:

- cue "/foo/bar/baz"
- cue "/foo/baz/quux"
- cue "/eggs/beans/toast"
- cue "/moog/synths/rule"

No entanto, ele não combinará caminhos com menos ou mais segmentos. O seguinte não irá corresponder:

- cue "/foo/bar"
- cue "/foo/baz/quux/quaax"
- cue "/eggs"

Cada * significa qualquer conteúdo. Então, podemos combinar caminhos com apenas um segmento com / * ou caminhos com cinco segmentos com /*/*/*/*/*

Segmentos Parciais Correspondentes

Se sabemos o como o segmento vai começar ou terminar, podemos usar um * além de um nome de segmento parcial. Por exemplo: "/foo/b*/baz"irá combinar com qualquer caminho que tenha três segmentos, o primeiro é *foo*, o último *baz* e o segmento do meio pode ser qualquer coisa que comece com *b*. Então, ele combinaria:

- cue "/foo/bar/baz"
- cue "/foo/baz/baz"

- cue "/foo/beans/baz"

No entanto, não corresponderia ao seguinte:

- cue "/foo/flibble/baz"
- cue "/foo/abaz/baz"
- cue "/foo/beans/baz/eggs"

Você também pode colocar o * no início do segmento para especificar os últimos caracteres de um segmento: "/foo/*zz/baz"que irá corresponder a qualquer sugestão de 3 segmentos ou definir onde o primeiro segmento é *foo*, o último é *baz* e o segmento do meio termina com *zz* como "*cue* "/foo/whiz/baz".

Correspondência de Segmentos de Diretórios Aninhado

Às vezes você não sabe quantos segmentos do caminho você deseja combinar. Nestes casos, você pode usar estrela dupla: ** como "/foo/**/baz"que irá combinar:

- cue "/foo/bar/baz"
- cue "/foo/bar/beans/baz"
- cue "/foo/baz"
- cue "/foo/a/b/c/d/e/f/baz"

Correspondência de Letras Únicas

Você pode usar o ponto de interrogação *?* para combinar com um único caractere, como "/?oo/bar/baz", que irá corresponder:

- cue "/foo/bar/baz"
- cue "/goo/bar/baz"
- cue "/too/bar/baz"
- cue "/woo/bar/baz"

Combinando Várias Palavras

Se você sabe que um segmento pode ser um número seleto de palavras, você pode usar os *and* para especificar uma lista de escolhas como "/foo/bar, feijão, ovos/quux"que somente corresponderá os seguintes:

- cue "/foo/bar/quux"
- cue "/foo/beans/quux"
- cue "/foo/eggs/quux"

Correspondendo Várias Letras

Finalmente, você pode combinar contra uma seleção de letras se usar [and] para especificar uma lista de escolhas, como "/foo/[abc]ux/baz", que somente corresponderá:

- cue "/foo/aux/baz"
- cue "/foo/bux/baz"
- cue "/foo/cux/baz"

Você também pode usar o caractere - para especificar intervalos de letras. Por exemplo, "/foo/[a-e]ux/baz", que apenas corresponderá:

- cue "/foo/aux/baz"

- cue "/foo/bux/baz"
- cue "/foo/cux/baz"
- cue "/foo/dux/baz"
- cue "/foo/eux/baz"

Combinando Correspondentes

Ao convocar *sync* ou *get*, você é livre para combinar correspondentes em qualquer ordem que considere apropriada para combinar qualquer evento do Time State criado por *cue* ou *set*. Vejamos um exemplo louco:

```
in_thread do
  sync "/?oo/[a-z]*/**/ba*/{quux,quaax}/"
  sample :loop_amen
end

sleep 1

cue "/foo/beans/a/b/c/d/e/bark/quux/"
```

Correspondência com OSC

Para aqueles curiosos, essas regras de correspondência são baseadas na especificação de correspondência de Padrão OSC (Open Sound Control) que é explicada em detalhes aqui: http://opensoundcontrol.org/spec-1_0.

Capítulo 11

MIDI

Depois de controlar a conversão de código em música, você pode se perguntar - o que vem depois? Às vezes, as restrições de trabalhar apenas dentro da sintaxe e sistema de som do Sonic Pi podem ser interessante colocá-lo em uma nova situação criativa. No entanto, às vezes é essencial sai do código e dialogar com mundo real. Nesse sentido buscaremos duas coisas extras:

1. Usar ações do mundo real para coordenar eventos no Sonic Pi
2. Usar o Sonic Pi para controlar e manipular objetos no mundo real

Felizmente, existe um protocolo que existe desde a década de 1980 o qual permite exatamente esse tipo de interação - MIDI (*Music Interface Digial Instruemnt*). Há uma enorme quantidade de dispositivos externos, incluindo teclados, controladores, sequenciadores e programas de áudio profissionais que suportam MIDI. Podemos usar MIDI para receber dados e também usá-lo para enviar dados.

O Sonic Pi suporta o protocolo MIDI, permitindo que você conecte seu código ao vivo ao mundo real. Vamos explorá-lo mais ...

11.1 MIDI In

Nesta seção, aprenderemos a conectar um controlador MIDI para enviar eventos para o Sonic Pi e assim controlar nossos sintetizadores e sons. Para seguir pegue um controlador MIDI, como um teclado ou uma interface de controle, e vamos explorar fisicamente!

Conectando Controlador MIDI

Para obter informações de um dispositivo MIDI externo para o Sonic Pi, primeiro precisamos conectá-lo ao nosso computador. Normalmente, isto acorre através de uma conexão USB, embora equipamentos antigos tenham um conector *DIN* de 5 pinos, para o qual você precisará de suporte de *hardware* para o seu computador suportá-lo (por exemplo, algumas placas de som possuem conectores DIN MIDI). Depois de conectar seu dispositivo, inicie o Sonic Pi e veja a seção IO do painel de preferências. Você deverá encontrar seu dispositivo listado lá. Caso contrário, tente acertar o botão 'Redefinir MIDI' e veja se ele aparece. Se você ainda não está vendo nada, a próxima coisa é consultar a configuração MIDI do seu sistema operacional para ver se ele reconhece seu dispositivo. Falhando tudo isso, sinta-se à vontade para fazer perguntas na sala de bate-papo pública: http://gitter.im/samaaron/sonic-pi.

Recebendo MIDI

Uma vez com seu dispositivo conectado, o Sonic Pi receberá eventos automaticamente. Você pode ver que ao manipular seu dispositivo

MIDI os *cues* (registrador na parte inferior direita da janela do aplicativo) abaixo do *log* (se isso não estiver visível, vá em Preferências-> Editor-> Mostrar e Ocultar e ativar o 'Mostra cue log e habilite o botão). Os eventos têm a seguinte aparência:

```
/midi/nanokey2_keyboard/0/1/note_off [55, 64] /midi/nanokey2_keyboard/0/1/note_
[53, 102] /midi/nanokey2_keyboard/0/1/note_off [57, 64] /midi/nanokey2_keyboard/
[53, 64] /midi/nanokey2_keyboard/0/1/note_on [57, 87] /midi/nanokey2_keyboard/0
[55, 81] /midi/nanokey2_keyboard/0/1/note_on [53, 96] /midi/nanokey2_keyboard/0
[55, 64]
```

Se você ver um fluxo de mensagens como esta, seu dispositivo MIDI foi conectacdo com sucesso. Ótimo, agora vamos ver o que podemos fazer com isso!

MIDI *Time State*

Esses eventos são divididos em duas seções. Em primeiro lugar, há o nome do evento como /midi/nanokey2_keyboard/0/1/note_on e, em segundo lugar, há os valores do evento, como [18, 62]. Curiosamente, estas são as duas coisas que precisamos para armazenar informações no *Time State*. O Sonic Pi insere automaticamente os eventos MIDI recebidos no Time State. Isso significa que você pode obter o valor MIDI mais recente e também sincronizar esperando o próximo valor MIDI usando tudo o que aprendemos na seção 10 deste tutorial.

Controlando o Código

Agora que conectamos um dispositivo MIDI, vimos seus eventos no registro de *cue* e descobrimos que nosso conhecimento de Time State é tudo o que precisamos para trabalhar com os eventos, podemos começar. Vamos construir um simples piano MIDI:

```
live_loop :midi_piano do
  note, velocity = sync "/midi/nanokey2_keyboard/0/1/note_on"
  synth :piano, note: note
end
```

Há algumas coisas acontecendo no código acima, incluindo alguns problemas. Em primeiro lugar, temos um live_loop simples que repetirá para sempre executar o código entre o bloco *do/end*. Isto foi introduzido na Seção 9.2. Em segundo lugar, estamos convocando *sync* para aguardar o próximo evento correspondente do *Time State*. Nós usamos uma lista que representa a mensagem MIDI que procuramos (o que é o mesmo que foi exibido na janela de registro de *cue*). Observe que esta lista longa é fornecida pelo sistema auto-complete do Sonic Pi, portanto, você não precisa digitar tudo à mão. Na janela de registro *cue*, vemos que haviam dois valores para cada evento MIDI, então nós atribuímos o resultado para dessas duas variáveis separadamente *note* e *velocity*. Finalmente, disparamos: o *synth* de piano passando nossa note.

Agora, tente você. Digite o código acima, substitua a chave de sincronização por uma string que corresponda ao seu dispositivo MIDI específico e execute *Run*. Pronto, você tem um piano de funcionando! No entanto, você provavelmente notará alguns problemas: primeiro, todas as anotações estão no mesmo volume, independentemente de quão forte você pressione a tecla. Isso pode ser facilmente corrigido usando o valor MIDI *velocity* e convertendo-o na amplitude. Dado que MIDI tem um intervalo de 0-> 127, para converter esse número em um valor entre 0-> 1, precisamos dividir por 127:

```
live_loop :midi_piano do
  note, velocity = sync "/midi/nanokey2_keyboard/0/1/note_on"
  synth :piano, note: note, amp: velocity / 127.0
end
```

Atualize o código e pressione **Run** novamente. Agora, a intensidade (velocity) do teclado é respeitada. Em seguida, vamos nos livrar

dessa latência.

Removendo Latência

Antes de podermos remover a latência, precisamos saber por que ela existe. Para manter todos os sintetizadores e efeitos bem sincronizados em uma variedade de CPUs de diferentes capacidades de processamentos, o Sonic Pi por padrão gerencia primeiro o áudio com 0,5 segundo. (Observe que esta latência adicional pode ser configurada através do fns *set_sched_ahead_time*! E *use_sched_ahead_time*). Esta latência de 0,5s está sendo adicionada ao disparar o *synth de :piano* , pois é adicionado a todos os sintetizadores desencadeados pelo Sonic Pi. Normalmente, queremos essa latência adicional, pois isso significa que todos os sintetizadores estarão bem cronometrados. No entanto, isso só faz sentido para sintetizadores desencadeados pelo código usando *play* e *sleep*. Neste caso, estamos realmente desencadeando synth :piano com nosso dispositivo MIDI externo e, portanto, não queremos que o Sonic Pi controle o tempo para nós. Podemos desligar esta latência com o comando *use_real_time* que desabilita a latência do segmento atual. Isso significa que você pode usar o modo em tempo real para live_loops que têm seu tempo controlado pela sincronização com dispositivos externos e manter a latência padrão para todos os outros *live_loop*. Vamos ver:

```
live_loop :midi_piano do
  use_real_time
  note, velocity = sync "/midi/nanokey2_keyboard/0/1/note_on"
  synth :piano, note: note, amp: velocity / 127.0
end
```

Atualize seu código para coincidir com o código acima e execute o código novamente. Agora temos um piano de baixa latência com velocidade variável codificado em apenas 5 linhas. Não era tão difícil! Certo?

Obtendo Valores

Finalmente, à medida que nossos eventos MIDI estão diretos no Time State, também podemos usar a função *get* para recuperar o último valor. Isso não bloqueia a *thread* atual e retorna *nil* se não houver nenhum valor a ser encontrado (o qual você pode substituir passando um valor padrão - veja os documentos para *get*). Lembre-se de que você pode convocar *get* em qualquer *thread* a qualquer momento para ver o último valor correspondente do *Time State*. Você pode até usar *time_warp* para retroceder no tempo e convocar o *get* para ver eventos passados ...

Você no Controle

A coisa interessante agora é que agora você pode usar as mesmas estruturas de código para sincronizar e obter informações MIDI de qualquer dispositivo MIDI e fazer o que quiser com os valores. Agora você pode escolher o que seu dispositivo MIDI fará!

11.2 MIDI out

Além de receber eventos MIDI, também podemos enviar eventos MIDI para acionar e controlar sintetizadores, teclados e outros dispositivos e hardware externos. Sonic Pi fornece um conjunto completo de *fns* para enviar várias mensagens MIDI, tais como:

1. Note on - midi_note_on
2. Note off - midi_note_off
3. Control change - midi_cc
4. Pitch bend - midi_pitch_bend

5. Clock ticks - midi_clock_tick

Há muitas outras mensagens MIDI suportadas também - confira a documentação da API para todos os outros *fns* que começam com midi_.

Conectando Dispositivo MIDI

Para enviar uma mensagem MIDI para um dispositivo externo, primeiro devemos conectá-lo. Verifique a subseção "Conectando um controlador MIDI"na seção 11.1 para mais detalhes. Observe que se você estiver usando o USB, conectar-se a um dispositivo ao qual você está enviando (em vez de receber) é o mesmo procedimento. No entanto, se você estiver usando os conectores DIN clássicos, certifique-se de se está conectado à porta de saída MIDI do seu computador. Você deve ver seu dispositivo MIDI listado no painel de preferências.

Enviando MIDI

As diversas funções midi_ * funcionam exatamente como *play*, *sample* e *synth*, na medida em que enviam uma mensagem em tempo real (lógico). Por exemplo, para enviar disparos para *fns midi_ ** você precisa usar *sleep* exatamente como você fez com *play*. Vamos dar uma olhada:

```
midi_note_on :e3, 50
```

Isso enviará uma nota *midi_on* para o dispositivo MIDI conectado com velocidade 50. (Note que Sonic Pi converterá automaticamente as notas na forma *:e3* para o seu número MIDI correspondente, neste caso ":e3 = 52.)

Se o seu dispositivo MIDI conectado for um sintetizador, você poderá ouvi-lo tocar uma nota. Para desativá-lo use *midi_note_off*:

```
midi_note_off :e3
```

Selecionando Dispositivo MIDI

Por padrão, Sonic Pi enviará cada mensagem MIDI para todos os dispositivos conectados em todos os canais MIDI. Isto é para facilitar o trabalho com um único dispositivo conectado sem ter que configurar nada. No entanto, às vezes, um dispositivo MIDI tratará os canais MIDI de forma especial (talvez cada nota tenha um canal separado) e você também pode conectar mais de um dispositivo MIDI ao mesmo tempo. Em configurações mais elaboradas, você pode querer ser mais seletivo sobre qual dispositivo MIDI recebe qual mensagem e em qual canal.

Podemos especificar para qual dispositivo enviar usando a opt *port:*, usando o nome do dispositivo como mostrado nas preferências:

```
midi_note_on :e3, port: "moog_minitaur"
```

Também podemos especificar qual canal enviar usando a opt *channel:* (usando um valor entre 1-16):

```
midi_note_on :e3, channel: 3
```

Claro que também podemos especificar *port:* e *channel:* ao mesmo tempo para enviar a cada dispositivo específico em canais específico:

```
midi_note_on :e3, port: "moog_minitaur", channel: 5
```

Estúdio MIDI

Finalmente, uma coisa muito divertida é conectar a saída de áudio do seu sintetizador MIDI a uma das entradas de áudio da sua placa de som. Você pode então controlar o seu sintetizador com o código usando função *midi_ ** e também manipular o áudio usando live_audio e efeitos FX:

```
with_fx :reverb, room: 1 do
  live_audio :moog
end

live_loop :moog_trigger do
  use_real_time
  midi (octs :e1, 3).tick, sustain: 0.1
  sleep 0.125
end
```

(A função *midi* está disponível como um atalho acessível para enviar tanto *note on* e *note off* com um único comando. Consulte a documentação para obter mais informações).

Capítulo 12

OSC

Além do MIDI, outra maneira de obter informações dentro e fora do Sonic Pi é através da rede usando um protocolo simples chamado OSC - Open Sound Control. Isso permitirá que você envie e receba mensagens de programas externos (ambos em execução no seu computador e em computadores externos), o que abre o potencial de controle para além do protocolo MIDI, que tem limitações devido ao seu design dos anos 1980.

Por exemplo, você poderia escrever um programa em outra linguagem de programação que envia e recebe OSC (há bibliotecas OSC para praticamente todas as linguagens comuns). Mas para o que você pode usar o OSC. Isso dependerá da sua imaginação.

12.1 Recebendo OSC

Por padrão, quando o Sonic Pi é iniciado, ele escuta a porta 4559 para receber mensagens OSC de programas no mesmo computador (localhost). Isso significa que, sem qualquer configuração, você pode enviar Sonic Pi uma mensagem OSC e será exibida no *cue* log , assim como as mensagens MIDI recebidas. Isso também significa que qualquer mensagem entrada de mensagem OSC também é automaticamente adicionada ao Time State, o que significa que você também pode usar *get* e *sync* para trabalhar com os dados recebidos - assim como com MIDI.

OSC básico

Vamos construir um ouvinte (receptor) OSC básico:

```
live_loop :foo do
  use_real_time
  a, b, c = sync "/osc/trigger/prophet"
  synth :prophet, note: a, cutoff: b, sustain: c
end
```

Neste exemplo, descrevemos um caminho OSC "/osc/trigger/prophet"no qual estamos sincronizando. Este pode ser qualquer caminho OSC válido (todas as letras e números são suportados e o travessão **"/"** é usado como em um URL para quebrar o caminho para várias palavras). O prefixo /osc é adicionado pelo Sonic Pi a todas as mensagens OSC recebidas, por isso precisamos enviar uma mensagem OSC com o caminho /trigger/prophet para a nossa *sync* para interromper o bloqueio e o *synth :prophet* a ser ativado.

Enviando OSC para Sonic Pi

Podemos enviar OSC para Sonic Pi a partir de qualquer linguagem de programação que tenha uma biblioteca OSC. Por exemplo, se estamos enviando OSC da Python, podemos fazer algo como isto:

```
from pythonosc import osc_message_builder
from pythonosc import udp_client

sender = udp_client.SimpleUDPClient('127.0.0.1', 4559)
sender.send_message('/trigger/prophet', [70, 100, 8])
```

Ou, se estamos enviando OSC de Clojure, podemos fazer algo como isso da REPL:

```
(use 'overtone.core)
(def c (osc-client "127.0.0.1" 4559))
(osc-send c "/trigger/prophet" 70 100 8)
```

Recebendo de Máquinas Externas

Por motivos de segurança, por padrão, Sonic Pi não permite que máquinas remotas enviem mensagens OSC. No entanto, você pode ativar o suporte para máquinas remotas em Preferências-> IO-> Rede-> Receber mensagens OSC remotas. Depois de ativá-la, você pode receber mensagens OSC de qualquer computador em sua rede. Normalmente, a máquina que envia precisará conhecer seu endereço IP (um identificador exclusivo para o seu computador em sua rede - tipo como um número de telefone ou um endereço de e-mail). Você pode descobrir o endereço IP do seu computador, observando a seção IO do painel de preferências. (Se a sua máquina tiver mais de um endereço IP, passando o mouse sobre o endereço irá listar uma lista de todos os endereços conhecidos).

Note, alguns programas como o TouchOSC para iPhone e o Android suportam o envio de OSC como um recurso padrão. Então, uma vez que você está ouvindo máquinas remotas e conhece seu endereço IP, você pode começar a enviar mensagens instantaneamente de aplicativos como o TouchOSC, que permitem que você crie seus próprios controles de toque personalizados com controles deslizantes, botões, mostradores, etc. Isso pode lhe proporcionar uma enorme gama de opções de entrada.

12.2 Enviando OSC

Além de receber OSC e trabalhar com ele usando o Time State, também podemos enviar mensagens OSC no tempo da nossa música (assim como podemos enviar mensagens MIDI no tempo da música). Só precisamos saber qual endereço IP e porta estamos enviando. Vamos tentar:

```
use_osc "localhost", 4559
osc "/hello/world"
```

Se você executar o código acima, notará que o Sonic Pi está enviando uma mensagem OSC! Isso ocorre porque definimos o endereço IP como a mesma máquina que estamos trabalhando para a porta padrão OSC. Isto seria a mesma coisa que publicar uma carta enviada para você - o pacote OSC é criado, deixa o Sonic Pi, chega à pilha de rede do sistema operacional, que roteia o empacotamento para o Sonic Pi que recebe como uma mensagem OSC padrão que pode ser vista no *cue* log como a seguinte mensagem "/osc/hello/world."(Observe como o Sonic Pi prefixa automaticamente todas as mensagens OSC recebidas com /osc.)

Enviando OSC para Outros Programas

É claro que enviar mensagens OSC para o mesmo programa pode ser divertido, mas nem sempre é tão útil. O maior benefício deste protocolo é enviar mensagens para outros programas:

```
use_osc "localhost", 123456
osc "/hello/world"
```

Neste caso, estamos assumindo que há outro programa na mesma máquina que escuta a porta 123456. Se esta porta existir, então receberá uma mensagem OSC */hello/world* que pode você poderá fazer com ela o que desejar.

Se nosso programa estiver sendo executado em outra máquina, precisamos saber seu endereço IP que usamos em vez de "localhost":

```
use_osc "192.168.10.23", 123456
osc "/hello/world"
```

Agora, podemos enviar mensagens OSC para qualquer dispositivo acessível para nós através de nossas redes locais e até mesmo na internet!

Capítulo 13

Áudio Multicanal

Até agora, em termos de produção de som, exploramos desencadear sintetizadores e sons gravados via fns *play*, *synth* e *sample*. Isso gera áudio que toca através do nosso sistema de alto-falantes estéreo. No entanto, muitos computadores também têm entrada de áudio, através de um microfone, além da capacidade de enviar o sinal de áudio para mais de dois alto-falantes. Muitas vezes, essas capacidades é possível através do uso de uma placa de som externa - disponíveis para todas as plataformas. Nesta seção, daremos uma olhada como podemos tirar proveito dessas placas de áudio externas e trabalhar com vários canais de áudio dentro e fora do Sonic Pi.

13.1 Entrada de Áudio

Uma maneira simples (e talvez familiar) de acessar entradas de áudio é usar o *synth* especificando *synth: sound_in* .

```
synth: sound_in
```

Isso funcionará como qualquer synth, como synth :dsaw com a exceção de que o áudio gerado será lido diretamente da primeira entrada da placa de som do seu sistema. Em laptops, ela é normalmente o microfone embutido, mas se você possuir uma placa de som externa, pode conectar qualquer entrada de áudio à primeira entrada.

Aumentando Duração

Uma coisa que você irá notar é que apenas com *synth :dsaw* e *:sound_in* o som dura 1 batida, pois segue o envelope padrão. Se você quiser mantê-lo aberto por mais tempo, mude as configurações do envelope ADSR. Por exemplo, o seguinte manterá o sintetizador aberto por 8 batidas antes de fechar a conexão:

```
synth :sound_in, sustain: 8
```

Adicionando Efeitos

Assim como qualquer sintetizador normal, você pode facilmente aplicar uma camada sobre os efeitos com o bloco FX:

```
with_fx :reverb do
  with_fx :distortion do
    synth :sound_in, sustain: 8
  end
end
```

Se você conectou uma guitarra na primeira entrada de áudio, você deve poder ouvi-lo com distorção e reverb até o *synth* terminar conforme esperado.

Você é livre para usar *synth :sound_in* quantas vezes quiser (assim como você faria com qualquer sintetizador normal). O exemplo seguinte reproduz dois *synth :sound_in* ao mesmo tempo - um com distorção e um com reverb:

```
with_fx :distortion do
  synth :sound_in, sustain: 8
end

with_fx :reverb do
  synth :sound_in, sustain: 8
end
```

Múltiplas Entradas

Você pode selecionar qual entrada de áudio você quer tocar com a opção *input*. Você também pode especificar uma entrada estéreo (duas entradas consecutivas) usando o *synth sound_in_stereo*. Por exemplo, se você tiver uma placa de som com pelo menos três entradas, você pode tratar os dois primeiros como um sinal estéreo adicionando distorção e o terceiro como um sinal mono e adicionar reverb com o seguinte código:

```
with_fx :distortion do
  synth :sound_in_stereo, sustain: 8, input: 1
end

with_fx :reverb do
  synth :sound_in, sustain: 8, input: 3
end
```

Potenciais Problemas

No entanto, embora esta seja uma técnica útil, há algumas limitações nesta abordagem. Em primeiro lugar, ele só funciona por uma

duração específica (devido ao fato de ter um envelope ADSR) e, em segundo lugar, não há como mudar o efeito (FX) uma vez que o sintetizador tenha sido criado. Essas duas coisas são solicitações típicas ao trabalhar com alimentação de áudio tais como microfones, guitarras e sintetizadores externos. Vamos, portanto, examinar a solução para o problema de manipular um sinal (potencial) infinito de entrada de áudio ao vivo: live_audio.

13.2 Live Audio

O som *synth :sound_in* como descrito na seção anterior fornece um método muito flexível e familiar para trabalhar com entrada de áudio. No entanto, como discutido, tem alguns problemas ao trabalhar com uma única entrada de áudio como um único instrumento (como uma voz ou guitarra). A melhor abordagem para trabalhar com um único fluxo contínuo de áudio é usar *live_audio*.

Entrada de Áudio Nomeada

live_audio compartilha algumas restrições básicas de design como live_loop (daí o nome similar). Em primeiro lugar, deve ter um único nome, e segundo lugar, só pode existir um único sinal de live_audio com esse nome. Vamos dar uma olhada:

```
live_audio :foo
```

Este código atuará de forma semelhante ao *synth :sound_in* com algumas diferenças principais: ele roda sempre (até que você explicitamente o interrompa) e você pode movê-lo para novos contextos FX dinamicamente.

Trabalhando com Efeito

Ao iniciar o *live_audio* funciona exatamente como você pode esperar que ele funcione com efeito. Por exemplo, para iniciar um transmissão de áudio ao vivo com reverb adicione simplesmente o bloco de efeito reverb:

```
with_fx :reverb do
  live_audio :foo
end
```

No entanto, dado que o *live_audio* é executado para sempre (pelo menos até você parar), seria bastante limitado se, como os synth , o live_audio estivesse vinculado dentro do FX *:reverb* para sempre. Por sorte, este não é o caso e foi desenhado para ser fácil de se mover entre diferentes FX. Vamos tentar. Execute o código acima para ouvir áudio ao vivo diretamente da primeira entrada de sua placa de som. Observe, se você estiver usando um laptop, isso geralmente será fora do seu microfone embutido, por isso recomenda-se usar fones de ouvido para não gerar microfonia (*feedback*).

Agora, enquanto você ainda ouve o áudio ao vivo da placa de som com reverb, altere o código para o seguinte:

```
with_fx: echo do
  live_audio: foo
end
```

Agora, execute e você ouvirá imediatamente o áudio reproduzido com efeito *echo* e não mais através do *reverb*. Se você quisesse ambos, basta editar o código novamente e executar:

```
with_fx :reverb do
  with_fx :echo do
    live_audio :foo
  end
end
```

É importante ressaltar que você pode definir *live_audio: foo* de qualquer *thread* ou live_loop e isso moverá o sintetizador live_audio para o contexto FX atual dessa *thread*. Você poderia, portanto, facilmente ter vários live_loops convocando *live_audio: foo* em diferentes momentos, resultando no contexto FX sendo automaticamente trocado por alguns resultados interessantes.

Parando Áudio Ao Vivo

Ao contrário dos sintetizadores padrões, como live_audio não tem envelope, ele continuará funcionando indefinidamente (mesmo apagando o código ele permanece na memória). Para pará-lo, você precisa usar o argumento *:stop*

```
live_audio :foo, :stop
```

Ele pode ser reiniciado com facilidade chamando-o sem o seguinte: pare arg novamente:

```
live_audio :foo
```

Além disso, todos os sintetizadores de áudio em execução serão interrompidos quando pressionar o botão **Stop** parando também todos os outros sintetizadores e efeitos.

Entrada Estéreo

No que diz respeito aos canais de áudio, por padrão, *live_audio* age de forma semelhante ao *synth :sound_in*, na medida em que é necessário um único sinal de entrada mono que o converte em um sinal estéreo usando uma panorâmica especificada. No entanto, assim como *:sound_in_stereo* também é possível dizer ao *live_audio* que leia duas entradas de áudio consecutivas e trate-as como os canais esquerdo e

direito. Isto acontece através da *opt :stereo*. Por exemplo, para tratar a entrada 2 como o sinal esquerdo e a entrada 3 como o sinal direito, você precisa configurar o opt *:input* como 2 e ativar o modo estéreo da seguinte maneira:

```
live_audio :foo, stereo: true, input: 2
```

Observe que, uma vez que você iniciou uma transmissão de áudio ao vivo no modo estéreo, não conseguirá alterá-lo para mono sem recomeçar o processamento. Da mesma forma, se você iniciá-lo em mono não conseguirá mudar para estéreo sem reiniciar a transmissão.

13.3 Saída de Som

Até agora, nesta seção, analisamos como obter múltiplos sinis de áudio no Sonic Pi - quer através do *synth :sound_in* ou através do *live_audio*. Além de trabalhar com múltiplas entrada de áudio, o Sonic Pi também pode produzir vários saídas de áudio. Isto é conseguido através do FX *:sound_out*.

Contextos de Saída

Relembremos rapidamente como os sintetizadores e efeitos geram áudio no atual contexto atual de FX. Por exemplo, considere o seguinte:

```
with_fx :reverb do    # C
  with_fx :echo do    # B
    sample :bd_haus   # A
  end
end
```

A maneira mais simples de entender o que está acontecendo com a transmissão de áudio deste código é partir de centro para a extremidade. Nesse caso, o que está no centro é o *sample :bd_haus* (A). Depois o áudio passa pela camada B *with_fx :echo*. Só então é adicionado a o *reverb* na camada C que seguirá para as saídas de áudio no nível superior - que são os alto-falantes esquerdo e direito (saídas 1 e 2 da placa de áudio). O áudio segue para fora com um sinal estéreo.

Efeitos e Saída de Som

O comportamento acima funciona com todos os sintetizadores (incluindo *live_audio*) e a maioria dos efeitos com exceção do *:sound_out*. O efeito *:sound_out* faz duas coisas. Em primeiro lugar, ele produz seu áudio para o seu contexto externo conforme descrito acima. Em segundo lugar, ele também exibe seu áudio diretamente para uma saída na sua placa de som. Vamos dar uma olhada:

```
with_fx :reverb do                  # C
  with_fx :sound_out, output: 3 do # B
    sample :bd_haus                 # A
  end
end
```

Neste exemplo, o sample :bd_haus exibe seu áudio para o seu contexto externo, que é *:sound_out* FX. Isso, por sua vez, exibe seu áudio para o seu contexto externo: reverb FX (como esperado). No entanto, ele também produz uma mono mix para a 3ª saída da placa de som do sistema. O áudio gerado dentro *:sound_out* tem, portanto, dois destinos - o *:reverb* e saída 3 da placa de áudio.

Saída Mono e Stereo

Como já vimos, por padrão, o FX *sound_out* emite uma mistura mono da entrada estéreo para um canal específico, além de passar

a alimentação estéreo para o contexto externo (como esperado). Se a saída de uma mistura mono não é precisamente o que você quer fazer, existem algumas alternativas. Em primeiro lugar, usando *opt*, você pode exibir apenas o sinal de entrada à esquerda ou apenas à direita para a placa de áudio. Ou você pode usar FX *:sound_out_stereo* para as duas saídas consecutivas da placa de som. Consulte a documentação sobre função para mais informações e exemplos.

Saída Direta

Como vimos, o comportamento padrão para *:sound_out* e *:sound_out_stereo* é enviar ambos áudios para o seu ambiente externo (como é comum com todos os efeitos) e então para a saída especificada em sua placa de som. No entanto, ocasionalmente, você pode querer apenas enviar para a saída na sua placa de som e não para o ambiente externo (portanto existe a chance do som ser misturado e enviado para os canais de saída padrão 1 e 2). Isso é possível usando padrão FX opt *:amp* que intervem no áudio depois que FX o manipulou:

```
with_fx :sound_out, output: 3, amp: 0 do # B
  sample :loop_amen                      # A
end
```

No exemplo acima, o exemplo *:loop_amen* é enviado para o contexto externo *:sound_out*. Em seguida, envia uma mistura mono para saída 3 da placa de som e, em seguida, multiplica o áudio por 0, que acaba por silenciar. É este sinal silenciado que é então enviado para as caixas de som *sound_out* que é a saída padrão. Portanto, com este código, os canais de saída padrão não receberão nenhum áudio e o canal 3 receberá uma mistura mono da da sample de bateria.

Capítulo 14

Conclusões

Isso conclui o tutorial introdutório do Sonic Pi. Espero que você tenha aprendido algo ao longo dessa caminhada. Não se preocupe se você sentir que não entendeu tudo - apenas toque e se divirta. Você vai pegar as coisas no seu tempo. Sinta-se a vontade para retornar e mergulhar quando tiver uma pergunta que possa ser abordada em uma das seções.

Se você tiver alguma dúvida que não tenha sido abordada no tutorial, então passe pelos fóruns do Sonic Pi e faça sua pergunta. Você encontrará alguém disposto a dar uma mão.

Finalmente, convido você a ter uma visão mais profunda do resto da documentação pelo sistema de ajuda do programa. Lá existem vários recursos que não foram abordados neste tutorial aguardando serem descobertos.

Toque, divirta-se, compartilhe e execute códigos com seus amigos, mostre suas telas e lembre-se:

Não existem erros, apenas oportunidades.

Sam Aaron!

Appendices

Apêndice A

Artigos MagicPi

O Apêndice A reúne os artigos da Sonic Pi escritos para a revista MagPi.

Mergulhando nos Tópicos

Esses artigos não precisam ser lidos em uma ordem rigorosa. Ele reúne material de diferentes tópicos do tutorial. Em vez de tentar ensinar-lhe todo o Sonic Pi, eles se focam aspectos específico do Sonic Pi de forma divertida e acessível.

Leia MagPi

As revistas MagPi que têm um formato editorial profissional pode ser encontrada para download gratuito no formato PDF aqui:

www.raspberrypi.org/magpi/.

Sugerir um Tema

Se você não encontrar um tema que lhe interesse nesses artigos - por que não sugerir um? A maneira mais fácil de fazer isso é enviar um *tweet* com sua sugestão para @Sonic_Pi. Nunca se sabe - sua sugestão pode ser tema do próximo artigo!

A.1 Cinco Dicas Fundamentais

Não Existem Erros

A lição mais importante para aprender com Sonic Pi é que realmente não existem erros. A melhor maneira de aprender é apenas experimentar, experimentar e experimentar. Tente várias coisas diferentes. Não se preocupe se o código soa bem ou não e comece a experimentar com quantos sintetizadores, notas, efeitos e *opts* diferentes forem possíveis. Você descobrirá muitas coisas que fazem você se divertir, porque eles soam horríveis e algumas verdadeira preciosidades que soam incríveis. Basta pular as coisas que você não gosta e manter as coisas que te interessam. Quanto mais "erros"você se permitir mais rápido você aprenderá e descobrirá seus sons.

Use Efeitos (FX)

Diga que você já dominou os conceitos básicos da Sonic Pi para fazer sons com amostra, jogar? Qual é o próximo? Você sabia que o Sonic Pi suporta mais de 27 efeitos de estúdio (FX) para mudar o som do seu código? FX são como filtros de imagem extravagantes em programas de desenho, exceto que em vez de desfocar ou fazer algo preto e branco, você pode adicionar coisas como reverb, distorção e eco para o seu som. Pense nisso como plugar o cabo da guitarra em um pedal de efeitos de sua escolha e depois enviar o processo ao amplificador. Sonic Pi faz uso de FX de maneira realmente fácil e não requer cabos! Tudo o que você precisa fazer é escolher qual seção do seu código você gostaria que o FX adicionasse e envolve-lo com o código FX. Vejamos um exemplo. Digamos que você tenha o seguinte código:

```
sample :loop_garzul
```

```
16.times do
  sample :bd_haus
  sleep 0.5
end
```

Se você quisesse adicionar FX para a amostra: *loop_garzul*, basta colocá-lo dentro de um bloco *with_fx* como este:

```
with_fx :flanger do
  sample :loop_garzul
end

16.times do
  sample :bd_haus
  sleep 0.5
end
```

Agora, se você quisesse adicionar FX ao tambor, o envolva com *with_fx* também:

```
with_fx :flanger do
  sample :loop_garzul
end

with_fx :echo do
  16.times do
    sample :bd_haus
    sleep 0.5
  end
end
```

Lembre-se, você pode envolver qualquer código dentro do *with_fx* e todos os sons criados passarão por esse FX.

Parameterize seus sintetizadores

Para realmente descobrir códigos e sons particularmente próprios, você precisa saber como modificar e controlar sintetizadores e FX.

Por exemplo, você pode querer alterar a duração de uma nota, adicionar mais reverb ou alterar o tempo entre o efeito de eco. Sonic Pi oferece um incrível nível de controle para fazer exatamente isso com coisas especiais chamadas parâmetros opcionais ou *opt*. Vamos dar uma rápida olhada. Copie este código para um *buffer* e clique **Run**:

```
sample: guit_em9
```

Ooh, um som de guitarra adorável! Agora, vamos começar a tocar com isso. Que tal mudar sua taxa (rate)?

```
sample: guit_em9, rate: 0.5
```

Ei, o que é essa taxa: 0,5 que acabei de adicionar no final? Isso é chamado de *opt*. Todos os sintetizadores e FX do Sonic Pi suportam *opts* e há muito para tocar. Eles também estão disponíveis para FX. Tente isso:

```
with_fx :flanger, feedback: 0.6 do
  sample :guit_em9
end
```

Agora, tente aumentar o *feedback* para 1 e ouvirá alguns sons malucos! Leia os documentos para obter os detalhes completos sobre as muitos *opts* disponíveis.

Live Code

A melhor maneira de experimentar e explorar o Sonic Pi rapidamente é o *live code*. Isso permite que você começar um código e mudá-lo continuamente e ajustando-o enquanto ele ainda está tocando. Por exemplo, se você não sabe o que o parâmetro de corte faz para uma amostra, basta tocar. Vamos tentar! Copie este código para um dos seus espaços de trabalho do Sonic Pi:

```
live_loop :experiment do
  sample :loop_amen, cutoff: 70
  sleep 1.75
end
```

Agora, execute **Run** e você ouvirá uma pausa de tambor ligeiramente abafada. Agora, altere *cutoff:* para 80 e pressione **Run** novamente. Você pode ouvir a diferença? Experimente 90, 100, 110 ...

Depois que pegar o jeito de usar *live_loops*, você não o abandonará mais. Sempre que faço um show de *live coding*, confio no *live_loop* tanto quanto um baterista em suas baquetas. Para obter mais informações sobre *live code*, consulte a Seção 9 deste tutorial.

Surfando Fluxos Aleatórios

Finalmente, uma coisa que eu adoro fazer é utilizar Sonic Pi para gerar composições. Uma boa maneira de fazer isso é utilizar processos aleatórios. Pode parecer complicado, mas não é. Vamos dar uma olhada. Copie o código a seguir para um *buffer*:

```
live_loop :rand_surfer do
  use_synth :dsaw
  notes = (scale :e2, :minor_pentatonic, num_octaves: 2)
  16.times do
    play notes.choose, release: 0.1, cutoff: rrand(70, 120)
    sleep 0.125
  end
end
```

Agora, quando você tocar isso, você ouvirá um fluxo constante de notas aleatórias da escala: *e2: minor_pentatonic* tocando o sintetizador *dsaw*. "Espere, espere! Isso não é uma melodia", ouço você gritar! Bem, aqui está a primeira parte do truque. Toda vez que circulamos pelo *live_loop*, podemos dizer ao Sonic Pi para redefinir o fluxo aleatório para um ponto conhecido. Este é um pouco como voltar no tempo utilizando uma máquina do tempo que permite ir para um

ponto específico no tempo e no espaço. Vamos tentar - adicione a linha *use_random_seed 1* no *live_loop*:

```
live_loop :rand_surfer do
  use_random_seed 1
  use_synth :dsaw
  notes = (scale :e2, :minor_pentatonic, num_octaves: 2)
  16.times do
    play notes.choose, release: 0.1, cutoff: rrand(70, 120)
    sleep 0.125
  end
end
```

Agora, a cada ciclo de *live_loop* o fluxo aleatório é reiniciado. Isso significa que ele escolhe as mesmas 16 notas de cada vez. E pronto! Uma melodia instantânea. Agora, aqui temos uma batida emocionante. Altere o valor de semente de 1 para outro número. Digamos 4923. Uau! Outra melodia! Então, ao mudar um número de *seed* (a semente aleatória), você pode explorar tantas combinações melódicas que puder imaginar! Essa é a magia do código.

A.2 *Live Coding*

Os feixes de laser cortaram a cortina de fumaça enquanto o subwoofer presssionava profundamente nos corpos da multidão. A atmosfera estava no clímax com uma mistura inebriante de sintetizadores e danças. No entanto, algo não estava bem nesta pista. Projetado em cores brilhantes acima do placo do DJ um texto futurista, movendo, dançando, piscando. Este não era um visual elegante, era apenas uma projeção de Sonic Pi executando em um Raspberry Pi. O ocupante da cabine do DJ não estava girando discos, ele estava escrevendo, editando e avaliando o código. Ao vivo. Isso é *live code.*

Fig. A.1: Sam Aaron criador do Soni Pi em uma sessão de *Live Coding*

Isso pode parecer uma história extravagante de um clube futurista, mas codificar a música como esta é uma tendência crescente e muitas vezes é descrita como *live coding*(http://toplap.org). Uma tendência atual que esta abordagem de criação de música gerou é o Algorave (http://algorave.com) - eventos em que artistas como eu codificamos música para pessoas dançar. No entanto, você não precisa estar em

uma boate para fazer *Live Code*. Pode tocar em qualquer lugar onde você possa pegar o seu Raspberry Pi e um par de fones de ouvido ou alguns alto-falantes. Depois de chegar ao final deste artigo, você estará programando suas próprias batidas e modificando-as ao vivo. Onde você vai depois só dependerá da sua imaginação.

Live Loop

A fórmula para codificação em tempo real com o Sonic Pi é dominar o live_loop. Vejamos um:

```
live_loop :beats do
  sample :bd_haus
  sleep 0.5
end
```

Existem 4 ingredientes principais para um *live_loop*. O primeiro é o nome. Nosso live_loop acima é chamado *:beats*. Você pode chamar seu *live_loop* do que quiser. Seja criativo. Costumo usar nomes que comunicam algo sobre a música que estou fazendo para o público. O segundo ingrediente é a palavra **do** que representa onde o *live_loop* **do** inicia. O terceiro é a palavra **end** que marca onde o *live_loop* termina e, finalmente, há o corpo do *live_loop* - parte do código entre **do** e **end**, início e fim - que será repetido pelo loop. Neste exemplo, estamos repetidamente tocando uma amostra de tambor e esperando uma meia batida. Isso produz uma batida regular grave. Vá em frente, copie-o para um buffer Sonic Pi vazio e execute (**Run**). Boom Boom Boom!.

Redefinindo Enquanto Toca

Ok, então, o que é tão especial sobre o *live_loop*? Até agora parece um loop mais elaborado! Bem, a beleza de *live_loops* é que você pode alterá-los enquanto toca. Isso significa que enquanto eles ainda estão

funcionando, você pode mudar o que eles fazem. Este é o segredo do *live coding* com o Sonic Pi. Vamos tocar:

```
live_loop :choral_drone do
  sample :ambi_choir, rate: 0.4
  sleep 1
end
```

Agora, pressione o botão **Run** ou a tecla de atalho Alt-r ou Comand-r dependendo do sistema operacional.[i] Agora você está ouvindo sons de coro. Enquanto ainda toca, altere **rate:** de 0.4 a 0.38. Execute o código novamente. Woah! Você ouviu a mudança do coro? Altere-o de volta para 0.4 para retornar como estava. Agora, deixe cair para 0.2, até 0.19 e, em seguida, volte para 0.4. Veja como mudar apenas um parâmetro sobre a marcha pode dar-lhe um controle real da música? Agora brinque com *rate:* você mesmo - escolha seus próprios valores. Experimente números negativos, números muito pequenos e grandes. Divirta-se!

Dormir é Importante

Uma das lições mais importantes sobre *live_loops* é que eles precisam dormir (**sleep**). Considere o seguinte *live_loop:*

```
live_loop :infinite_impossibilities do
  sample :ambi_choir
end
```

Se você tentar executar este código, verá que o Sonic Pi informa um erro dizendo que o live_loop não dormiu. Este é um sistema de segurança! Tome um momento para pensar sobre o que esta mensagem está pedindo para o computador fazer. Isso mesmo, o código pede que o computador execute uma quantidade infinita de amostras do

[i]Veja mais sobre teclas de atalho no Apêndice B - Usando Atalhos

sample coro em tempo zero. Sem o sistema de segurança, o computador tentará fazer isso e travar e/ou queimar durante o processamento. Então lembre-se, seu live_loops deve conter um tempo de descanso (**sleep**) para não fundir.

Combinando Sons

A música está cheia de coisas acontecendo ao mesmo tempo. Tambores tocam ao mesmo tempo que o baixo, ao mesmo tempo que os vocais, ao mesmo tempo que as guitarras ... Em computação chamamos isso de simultaneidade e o Sonic Pi nos fornece uma maneira incrivelmente simples de tocar coisas ao mesmo tempo. Basta usar mais de um *live_loop*!

```
live_loop :beats do
  sample :bd_tek
  with_fx :echo, phase: 0.125, mix: 0.4 do
    sample  :drum_cymbal_soft, sustain: 0, release: 0.1
    sleep 0.5
  end
end

live_loop :bass do
  use_synth :tb303
  synth :tb303, note: :e1, release: 4, cutoff: 120, cutoff_attack: 1
  sleep 4
end
```

Aqui, temos dois *live_loops*, um loop rapidamente fazendo batidas e outro loop lentamente fazendo um som de baixo louco.

Uma das coisas interessantes sobre o uso de múltiplos *live_loops* é que cada um gerência seu próprio tempo. Isso significa que é realmente fácil criar estruturas polirritmicas interessantes e até mesmo jogar com o estilo do compositor Steve Reich. Veja isso:

```
# Steve Reich's Piano Phase
```

```
notes = (ring :E4, :Fs4, :B4, :Cs5, :D5, :Fs4, :E4, :Cs5, :B4, :Fs4, :D5, :Cs5)

live_loop :slow do
  play notes.tick, release: 0.1
  sleep 0.3
end

live_loop :faster do
  play notes.tick, release: 0.1
  sleep 0.295
end
```

Recombinando Tudo

Em cada um destes tutoriais, terminaremos com um exemplo final na forma de uma nova música que desenhe as ideias introduzidas. Leia este código e veja se você pode imaginar o que ele faz. Em seguida, copie-o para um novo *buffer* e execute-o e ouça com o que parece. Finalmente, mude um dos números ou comente e descomente as linhas. Veja se você pode usar isso como um ponto de partida para uma nova performance e, acima de tudo, se divirta! Vejo você na próxima vez...

```
with_fx :reverb, room: 1 do
  live_loop :time do
    synth :prophet, release: 8, note: :e1, cutoff: 90, amp: 3
    sleep 8
  end
end

live_loop :machine do
  sample :loop_garzul, rate: 0.5, finish: 0.25
  sample :loop_industrial, beat_stretch: 4, amp: 1
  sleep 4
end

live_loop :kik do
  sample :bd_haus, amp: 2
  sleep 0.5
end
```

```
with_fx :echo do
  live_loop :vortex do
    # use_random_seed 800
    notes = (scale :e3, :minor_pentatonic, num_octaves: 3)
    16.times do
      play notes.choose, release: 0.1, amp: 1.5
      sleep 0.125
    end
  end
end
```

A.3 Batidas codificadas

Um dos desenvolvimentos técnicos mais instigantes e perturbadores da música moderna foi a invenção do sampler. Eles são como caixas que lhe permitiram gravar qualquer som e depois manipular e reproduzir-los de muitas maneiras interessantes. Por exemplo, você pode pegar uma gravação antiga, encontrar um solo de bateria (batida break), gravá-lo em seu sampler e depois reproduzi-lo repetidas vezes, alterando a velocidade para fornecer a base para as últimas batidas. Foi assim que surgiu hip-hop inicialmente e hoje é quase impossível encontrar música eletrônica que não incorpora sample de algum tipo. Usar sample é uma ótima maneira de introduzir facilmente elementos novos e interessantes em suas performances de *live coding*.

Então, onde você pode obter um sampler? Bem, você já tem um - o Raspeberry Pi! O Sonic Pi vem instalado nele e possui um sampler extremamente poderoso integrado ao seu núcleo. Vamos brincar com isso!

O Amen Break

Uma das amostras de batida break mais clássicas e reconhecíveis é chamada de Amen Break. Foi realizado pela primeira vez em 1969 na música "Amen Brother"pelos Winstons como parte de uma ritmo de bateria. No entanto, foi quando foi descoberto pelos primeiros músicos do hip-hop nos anos 1980 e usado em amostras que começou a ser fortemente usado em uma grande variedade de outros estilos, como tambor e baixo, breakbeat, hardcore, techno e breakcore.

Tenho certeza de que você está curioso por saber que também foi construído diretamente no Sonic Pi. Limpe um buffer e inclua o seguinte código:

```
sample :loop_amen
```

Pressione **Run** e boom! Você está ouvindo uma das batidas mais influentes na história da *dance music*. No entanto, esse sample não era famoso por ser tocado como um tiro, ele foi criado para ser tocado em loop.

Esticando Batidas

Vamos fazer um loop do Amen Break usando o conhecido *live_loop* apresentado neste tutorial anteriormente:

```
live_loop :amen_break do
  sample :loop_amen
  sleep 2
end
```

OK, então isto é looping, mas ele tem uma pausa desconfortante a cada ciclo. Isso é porque pedimos que ele durasse por 2 batidas e com o BPM padrão de 60: a amostra loop_amen só dura 1.753 batidas. Portanto, temos um silêncio de 2 - 1.753 = 0,247 batimentos. Mesmo que seja curto, ainda é perceptível.

Para corrigir esse problema, podemos usar a *opt beat_stretch:* para solicitar ao Sonic Pi que estique (ou encolha) a amostra para corresponder ao número especificado de batidas.

O *sample* e o *synth* de Sonic Pi fornecem muitos controles através de parâmetros opcionais, como *amp:* (amplitude), *cutoff:* (ponto de corte) e *release:* (liberação). Apesar do termo parâmetro opcional ser longo para se pronunciar o chamamos de *opt* para manter as coisas agradáveis e simples.

```
live_loop :amen_break do
  sample :loop_amen, beat_stretch: 2
  sleep 2
```

```
end
```

Agora sim dá pra dançar! Que tal aumentar ou diminuir a velocidade para adequar ao humor.

Alterando o Tempo

Ok, então, e se quisermos mudar os estilos para o tradicional hip hop ou o breakcore? Uma maneira simples de fazer isso é brincar com o tempo - ou, em outras palavras, alterar o tempo. Isso é super fácil no Sonic Pi - basta lançar um use_bpm em seu loop ao vivo:

```
live_loop :amen_break do
  use_bpm 30
  sample :loop_amen, beat_stretch: 2
  sleep 2
end
```

Enquanto você está batendo nessas lentas batidas, note que ainda estamos dormindo para 2 e nosso BPM é 30, mas tudo está no tempo. A *opt beat_stretch* funciona com o BPM atual para garantir que tudo funcione de forma simples.

Agora, aqui chegamos na parte divertida. Enquanto o loop ainda está tocando, altere o valor 30 na linha *use_bpm 30* para 50. Woah, tudo acabou de ficar mais rápido, mas manteve-se no tempo! Tente ir mais rápido ainda - até 80, até 120, agora enlouqueça em batidas de 200!

Filtragem

Agora, podemos fazer mudanças em *live_loop*, vejamos algumas das opções mais divertidas fornecidas pelo *sample* synth. Primeiro, o *cu-*

toff: o que controla o filtro de corte do *sample*. Por padrão, isso está desativado, mas você pode ativá-lo facilmente:

```
live_loop :amen_break do
  use_bpm 50
  sample :loop_amen, beat_stretch: 2, cutoff: 70
  sleep 2
end
```

Não se acanhe e altere os valores da *opt cutoff:*. Por exemplo, aumente para 100, execute novamente o código e aguarde o novo ciclo para ouvir a mudança. Observe que com valores baixos, como 50 o som fica mais brando e grave e valores altos como 100 e 120, soa mais intenso e estridente. Isso ocorre porque a *opt cutoff* cortará as freqüência do som, assim como um cortador de grama corta o topo da grama. A opt *cutoff:* define o ajuste do comprimento - parecido ao ajuste do cortador de grama que define o tamanho da grama.

Fatiamento

Outra ótima ferramenta para experimentar é o efeito *slicer* (fatiador). Ele irá fatiar o som. Escolha um sample e o envolva com o efeito slicer, algo como:

```
live_loop :amen_break do
  use_bpm 50
  with_fx :slicer, phase: 0.25, wave: 0, mix: 1 do
    sample :loop_amen, beat_stretch: 2, cutoff: 100
  end
  sleep 2
end
```

Observe como o som salta para cima e para baixo um pouco mais. Você pode ouvir o som original sem o FX mudando *mix:* para 0. Agora, tente brincar com a opt *phase:*. Esta é a taxa (em batimentos) do efeito de corte. Um valor menor como 0.125 irá cortar valores rapidamente e maiores como 0.5 cortará mais lentamente.

Observe que dividindo pela metade ou duplicando a opt *phase:* sempre soará bem. Finalmente, altere *wave:* por 0, 1 ou 2 e ouça como altera o som. Estas são as várias formas de onda. Zero é uma onda dente de serra, (hard in, fade out) 1 é uma onda quadrada (hard in, hard out) e 2 é uma onda triangular (fade in, fade out).

Juntando Tudo

Finalmente, voltemos no tempo e revisemos a cena inicial do tambor e do baixo de Bristol com o exemplo deste mês. Não se preocupe muito com o que tudo isso significa, basta digitá-lo, pressione Executar e, em seguida, comece a codificá-lo ao vivo alterando números de opções e veja onde você pode chegar. Por favor, compartilhe o que você criou! Vejo você na próxima vez...

```
use_bpm 100

live_loop :amen_break do
  p = [0.125, 0.25, 0.5].choose
  with_fx :slicer, phase: p, wave: 0, mix: rrand(0.7, 1) do
    r = [1, 1, 1, -1].choose
    sample :loop_amen, beat_stretch: 2, rate: r, amp: 2
  end
  sleep 2
end

live_loop :bass_drum do
  sample :bd_haus, cutoff: 70, amp: 1.5
  sleep 0.5
end

live_loop :landing do
  bass_line = (knit :e1, 3, [:c1, :c2].choose, 1)
  with_fx :slicer, phase: [0.25, 0.5].choose, invert_wave: 1, wave: 0 do
    s = synth :square, note: bass_line.tick, sustain: 4, cutoff: 60
    control s, cutoff_slide: 4, cutoff: 120
  end
  sleep 4
end
```

A.4 Rifes de Sintetisadores

Seja a deriva assombrosa dos osciladores ruidosos ou o soco desafinado das ondas dente de serra que passam pelo mixer, o sintetizador desempenha um papel essencial em qualquer som eletrônico. Na edição do mês passado da série deste tutorial, aprendemos como codificar nossas batidas. Agora, veremos como codificar os três componentes principais de um *riff* de sintetizador - timbre, melodia e ritmo.

Ok, então, ligue o seu Raspberry Pi, abra o Sonic Pi e vamos fazer barulho!

Possibilidades de Timbres

Um tópico essencial em qualquer *riff* de sintetizador está em brincar e alterar o timbre dos sons. Podemos controlar o timbre em Sonic Pi de duas maneiras:

- *bruscamente*: escolhendo sintetizadores diferentes para uma mudança direta;
- *sutilmente*: definindo os vários *opts* de sintetizador para modificações mais sutis.

Também podemos usar o FX, mas isso será abordado em outro momento.

Vamos criar um simples *live_loop* onde alteramos continuamente o sintetizador:

```
live_loop :timbre do
  use_synth (ring :tb303, :blade, :prophet, :saw, :beep, :tri).tick
  play :e2, attack: 0, release: 0.5, cutoff: 100
  sleep 0.5
end
```

Dê uma olhada no código. Estamos simplesmente percorrendo uma lista (ring) com nomes de sintetizadores (isso irá executar cada um deles por vez repetindo a lista em circulo). Passamos este nome de sintetizador para o use_synth fn (função) que alterará o sintetizador atual do live_loop. Nós também tocamos nota: e2 (Mi na segunda oitava), com um tempo de liberação *release* de 0.5 batidas (meio segundo do BPM padrão 60) e com o ponto de corte (*cutoff: opt*) em 100.

Ouça como os diferentes sintetizadores têm sons muito distintos apesar de tocarem a mesma nota. Agora experimente e toque. Mude o tempo de *release* para valores maiores e menores. Por exemplo, mude o *attack:* e *release: opt* para ver o enorme impacto que o tempo de fade in/out geram no som. Finalmente, altere o *cutoff: opt* para ver como diferentes valores de corte também influenciam no timbre (tente valores entre 60 e 130). Veja quantos sons diferentes você pode criar simplesmente alterando alguns valores. Depois de dominar isso, vá até a guia Synths no sistema de Ajuda para obter uma lista completa de todos os sintetizadores e todos os opts disponíveis para cada sintetizador individualmente para ver apenas a quantidade de controle do código que você possui sob as pontas de seus dedos.

Timbre

Timbre é uma palavra sofisticada para descrever a qualidade de um som. Se você tocar a mesma nota em diferentes instrumentos, como um violão, guitarra ou piano, o tom (o som alto ou baixo) seria o mesmo, mas a qualidade do som seria diferente. Essa qualidade de som - o que permite que você reconheça a diferença entre um piano e um violão é o timbre.

Composição Melódica

Outro aspecto importante para tocar nosso sintetizador é a escolha das notas que queremos tocar. Se você já tem uma ideia, então você pode simplesmente criar uma lista (*ring*) com suas notas e marcá-los:

```
live_loop :riff do
  use_synth :prophet
  riff = (ring :e3, :e3, :r, :g3, :r, :r, :r, :a3)
  play riff.tick, release: 0.5, cutoff: 80
  sleep 0.25
end
```

Aqui, definimos nossa melodia com uma list (ring) que inclui ambas as notas, como *:e3* e repousos representados por *:r*. Estamos então usando *.tick* para percorrer cada nota para nos dar um riff repetitivo.

Melodia Automática

Nem sempre é fácil encontrar um bom *riff* do zero. Muitas vezes é mais fácil pedir ao Sonic Pi uma seleção de riffs aleatórios e escolher aquele que você mais gosta. Para fazer isso precisamos combinar três coisas: lista (ring), randomização e random_seed (sementes aleatórias). Vejamos um exemplo:

```
live_loop :random_riff do
  use_synth :dsaw
  use_random_seed 3
  notes = (scale :e3, :minor_pentatonic).shuffle
  play notes.tick, release: 0.25, cutoff: 80
  sleep 0.25
end
```

Existem algumas coisas acontecendo aqui - vamos olhar uma de cada vez. Primeiro, especificamos que estamos usando sementes aleatórias 3 (use_random_seed 3). O que isso significa? Bem, a coisa útil

é que, quando estabelecemos a semente, podemos prever qual será o próximo valor aleatório - é o mesmo que foi a última vez que colocamos a semente número 3! Outra coisa útil a saber é que a manipulação de um anel de notas funciona da mesma maneira. No exemplo acima, estamos essencialmente pedindo o "terceiro shuffle"na lista padrão de *shuffles* - o que também é o mesmo toda vez que estamos sempre definindo a semente aleatória no mesmo valor antes do *shuffle*. Finalmente, estamos passando por nossas notas embaralhadas para tocar o riff.

Agora, aqui é onde começa a diversão. Se alterarmos o valor da semente aleatória para outro número, digamos 3000, obtemos uma sequência diferente de notas. Então agora é extremamente fácil explorar novas melodias. Basta escolher a lista de notas que quer embaralhar (shuffle) (as escalas são um excelente ponto de partida) e, em seguida, escolha a semente. Se não gostar altere uma dessas duas coisas e tente novamente. Repita até escutar algo interessante aos seus ouvidos!

Pseudo Randomização

A randomização de Sonic Pi não é aleatória, é o que se chama pseudo aleatoriedade. Imagine se você fosse rolar um dado 100 vezes e anotar o resultado de cada resultado em um pedaço de papel. Sonic Pi tem uma lista de resultados equivalentes que usa quando você solicita um valor aleatório. Em vez de rodar um dado real, ele apenas escolhe o próximo valor da lista. Definir a semente aleatória é apenas saltar para um ponto específico dessa lista.

Encontrando seu ritmo

Outro aspecto importante para o nosso riff é o ritmo - quando tocar ou não tocar uma nota. Como vimos acima, podemos usar *:r* em

nossas listas (rings) para inserir repousos. Outra maneira muito eficiente é usar *spreads* que abordaremos no futuro. Hoje, usaremos randomização para nos ajudar a encontrar nosso ritmo. Em vez de tocar cada nota, podemos usar um condicional para tocar uma nota com uma dada probabilidade. Vamos dar uma olhada:

```
live_loop :random_riff do
  use_synth :dsaw
  use_random_seed 30
  notes = (scale :e3, :minor_pentatonic).shuffle
  16.times do
    play notes.tick, release: 0.2, cutoff: 90 if one_in(2)
    sleep 0.125
  end
end
```

Um fn realmente útil para saber é one_in que nos dará um valor verdadeiro ou falso com a probabilidade especificada. Aqui, estamos usando um valor 2, então, em média, uma vez, cada duas chamadas para one_in retornará verdade. Em outras palavras, 50% do tempo retornará verdadeiro. O uso de valores mais altos fará com que ele seja retornado mais freqüentemente, introduzindo mais espaço no *riff*.

Observe que adicionamos alguma iteração aqui com *16.times*. Isso é porque só queremos redefinir nosso valor de semente aleatória a cada 16 notas para que nosso ritmo se repita a cada 16 vezes. Isso não afeta o baralhamento, pois isso ainda é feito imediatamente após a criação da semente. Podemos usar o tamanho da iteração para alterar o comprimento do riff. Tente mudar os 16 a 8 ou até 4 ou 3 e veja como isso afeta o ritmo do riff.

Juntando Tudo

OK, vamos combinar tudo o que aprendemos em um último exemplo.

```
live_loop :random_riff do
  #  uncomment to bring in:
  #  synth :blade, note: :e4, release: 4, cutoff: 100, amp: 1.5
  use_synth :dsaw
  use_random_seed 43
  notes = (scale :e3, :minor_pentatonic, num_octaves: 2).shuffle.take(8)
  8.times do
    play notes.tick, release: rand(0.5), cutoff: rrand(60, 130) if one_in(2)
    sleep 0.125
  end
end

live_loop :drums do
  use_random_seed 500
  16.times do
    sample :bd_haus, rate: 2, cutoff: 110 if rand < 0.35
    sleep 0.125
  end
end

live_loop :bd do
  sample :bd_haus, cutoff: 100, amp: 3
  sleep 0.5
end
```

A.5 Acid Bass

É impossível examinar a história da *dance music* sem mencionar o grande impacto do pequeno sintetizador Roland TB-303. Ele é o tempero secreto por trás do som original do *acid bass*. Esses clássicos riffs graves do TB-303 de *squealing* e *squelching* podem ser ouvidos na cena inicial da Chicago House ou até mesmo em artistas eletrônicos mais recentes, como Plastikman, Squarepusher e Aphex Twin.

Curiosamente, a Roland não imaginou que o TB-303 seria usado na *dance music*. Originalmente ele foi criado para acompanhar guitarristas. Imaginaram que as pessoas programariam o sintetizador para tocar linhas de baixo e que improvisarem por cima. Infelizmente tiveram vários problemas com o TB-303: não era simples de programar, não soava particularmente bem para substituir o baixo elétrico e era bastante caro. Decidindo cortar os prejuízos, a Roland parou de fabricá-lo depois de 10 mil unidades vendidas e, depois de vários anos nas prateleiras dos guitarristas, logo poderiam ser encontradas nas lojas de segunda mão. Os TB-303 descartados e solitários estavam esperando para serem descobertos por uma nova geração de experimentadores que começaram a usá-los de uma maneira diferente daquela que Roland imaginava para criar novos sons loucos. Nasceu assim a Acid House.

Ter em suas mãos um TB-303 original não é tão fácil. Contudo, você ficará satisfeito por saber que poderá transformar seu Raspberry Pi em um TB-303 usando o Sonic Pi. Observe, inicie o Sonic Pi e cole este código em um buffer vazio e clique em *Run*:

```
use_synth :tb303
play :e1
```

Acid Bass instantâneo! Vamos brincar ...

Squelch that Bass

Primeiro, vamos construir um live_arpegiador para tornar as coisas divertidas. No último tutorial, analisamos como os *riffs* podem ser apenas um toque de notas que marcamos um após o outro, repetindo quando chegamos ao fim. Vamos criar um live_loop que faça exatamente isso:

```
use_synth :tb303
live_loop :squelch do
  n = (ring :e1, :e2, :e3).tick
  play n, release: 0.125, cutoff: 100, res: 0.8, wave: 0
  sleep 0.125
end
```

Dê uma olhada em cada linha.

1. Na primeira linha, configuramos o sintetizador padrão para ser tb303 com a função *use_synth* .
2. Na segunda linha, criamos um live_loop chamado :squelch, que apenas irá rodar em loop.
3. A linha três é onde criamos nosso riff - um anel de notas (mi nas oitavas 1, 2 e 3), que simplesmente marcamos com *.tick*. Definimos *n* para representar a nota atual no riff. O sinal de igual significa apenas atribuir o valor à direita ao nome à esquerda. Isso será diferente a cada ciclo. A primeira vez *n* será definido como :e1. A segunda vez, será :e2, seguido de :e3, e depois novamente para :e1, seguindo o ciclo sempre.
4. A linha quatro é onde realmente desencadeamos o nosso : sintetizador :tb303. Estamos passando algumas opções interessantes aqui release:, cutoff: , res: e wave: o que discutiremos abaixo.
5. A linha cinco é o nosso *sleep* - estamos pedindo que o live_loop seja rodado a cada 0.125 segundos ou 8 vezes por segundo no padrão de 60 BPM (batidas por minuto).

6. A linha seis é o fim do live_loop. Isso apenas diz ao Sonic Pi, onde é o final do live_loop.

Enquanto você vai descobrindo o que está acontecendo, digite o código acima e pressione o botão *Run*. Você deve ouvir :tb303 kick em ação. Agora, é aqui que começa o *live coding*.

Enquanto o live_loop ainda estiver tocando, mude o *cutoff:* opt para 110. Agora, pressione o botão *Run* novamente. Você deve ouvir o som tornar-se um pouco mais áspero e constrito. Selecione em 120. Agora, 130. Escute como os valores de corte mais altos fazem parecer mais penetrante e intenso o som. Finalmente, desça até 80 onde você sentirá um descanso. Então, repita tantas vezes quanto você quiser. Não se preocupe, ainda estarei por aqui ...

Outra *opt* que vale a pena experimentar é *res:*. Ela controla o nível de ressonância do filtro. Uma alta ressonância é característica dos sons de Acid Bass. Atualmente, temos nossa res: definida para 0.8. Tente ativar até 0,85, depois 0,9 e, finalmente, 0,95. Você pode achar que um ponto de corte *cutoff:*, como 110 ou superior, tornará as diferenças mais fáceis de ouvir. Finalmente enlouqueça um pouco com 0,999 para alguns sons insanos. Às alturas altas, você está ouvindo o filtro de corte ressoar tanto que começa a fazer sons próprios!

Finalmente, para um grande impacto no timbre tente mudar o formato da onda *wave:* para 1. Esta é a escolha do oscilador de origem. O padrão é 0, que é uma onda de dente de serra, 1 é uma onda de pulso e 2 é uma onda triangular.

Claro, experimente *riffs* diferentes alterando as notas na lista (*ring*) ou mesmo escolhendo notas de escalas ou acordes. Divirta-se com o seu primeiro sintetizador de Acid Bass.

Deconstruindo o TB-303

O design do TB-303 original é bastante simples. Como você pode ver no diagrama a seguir, existem apenas 4 partes principais.

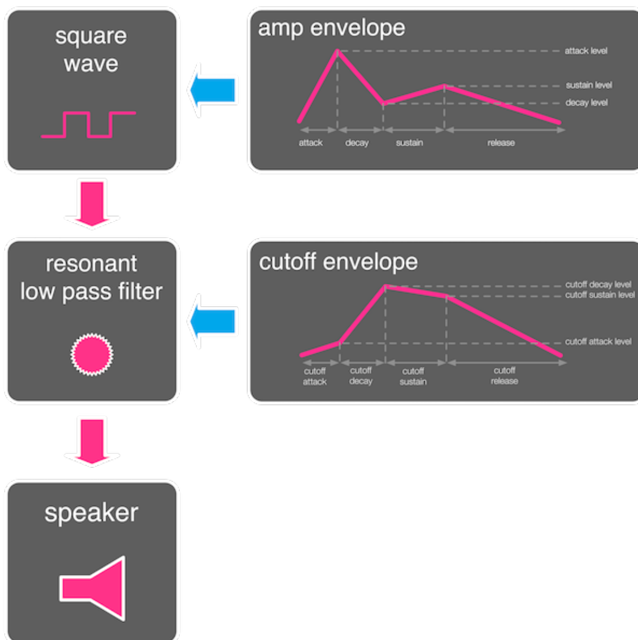


Fig. A.2: Design do sintetizador TB-303

Primeiro é a forma de onda do oscilador - o ingrediente bruto do som. Neste caso, temos uma onda quadrada. Em seguida, há o envelope de amplitude do oscilador que controla o amplificador da onda quadrada através do tempo. Estes são acessados em Sonic Pi pelo *attack:*, *decay:*, *sustain:* e *release:* opts junto com seus equivalentes de nível. Para obter mais informações, leia a Seção 2.4 "Duração com Envelopes". Em seguida, passamos a nossa envelopada onda qua-

drada através de um filtro de ressonância passa baixa. Isso corta as freqüências mais altas, além de ter esse bom efeito de ressonância. Agora, é aí que começa a diversão. O valor de corte deste filtro também é controlado por seu próprio envelope! Isso significa que temos um controle incrível sobre o timbre do som, jogando com ambos os envelopes. Vamos dar uma olhada:

```
use_synth :tb303
with_fx :reverb, room: 1 do
  live_loop :space_scanner do
    play :e1, cutoff: 100, release: 7, attack: 1, cutoff_attack: 4,
    cutoff_release: 4
    sleep 8
  end
end
```

Juntando tudo

Finalmente, aqui é uma peça que compus usando as ideias anteriores. Copie-o para um buffer vazio, ouça por um tempo e comece alterá-lo e fazer seu próprio live coding. Veja quais sons loucos você pode fazer com isso! Vejo você na próxima vez...

```
use_synth :tb303
use_debug false

with_fx :reverb, room: 0.8 do
  live_loop :space_scanner do
    with_fx :slicer, phase: 0.25, amp: 1.5 do
      co = (line 70, 130, steps: 8).tick
      play :e1, cutoff: co, release: 7, attack: 1, cutoff_attack: 4,
       cutoff_release: 4
      sleep 8
    end
  end

  live_loop :squelch do
    use_random_seed 3000
```

```
    16.times do
      n = (ring :e1, :e2, :e3).tick
      play n, release: 0.125, cutoff: rrand(70, 130), res: 0.9, wave: 1,
      amp: 0.8
      sleep 0.125
    end
  end
end
```

A.6 Música e Minecraft

Olá e bem-vindo novamente! Nos tutoriais anteriores, nos concentramos exclusivamente sobre as possibilidades musicais de Sonic Pi - (transformando seu Raspberry Pi em um instrumento musical preparado para performance). Até agora, aprendemos:

- Live Code - alterar sons durante o voo,
- Codificar algumas grandes batidas,
- Gerar potentes sintetizadores,
- Recriar o famoso som do Acid Bass TB-303.

Há muito mais para mostrar-lhe (o que exploraremos em futuras edições). No entanto agora vamos ver algo que a Sonic Pi pode fazer, você provavelmente não percebeu: controlar Minecraft.

Hellow Minecraft World

OK, vamos começar. Inicie seu Raspberry Pi, abra o Minecraft Pi e crie um novo mundo. Agora, crie o Sonic Pi e re-dimensione e mova suas janelas para que poder visualizar o Sonic Pi e o Minecraft Pi ao mesmo tempo.

Em um novo buffer digite o seguinte:

```
mc_message "Olá Minecraft da Sonic Pi!"
```

Agora, pressione *Run*. Tcham! Sua mensagem apareceu no Minecraft! Isso foi fácil não? Agora, pare de ler por um momento e brinque com suas próprias mensagens. Divirta-se!

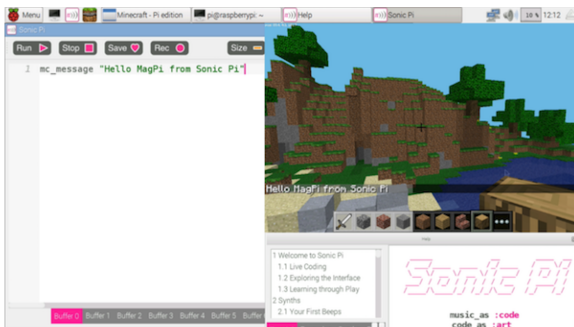


Fig. A.3: Interfaces de comunicação Sonic Pi e Minecraft

Teleporte Sônico

Agora vamos fazer algumas explorações. A opção padrão é pegar o mouse e o teclado e começar a caminhar. Isso funciona, mas é muito lento e chato. Seria muito melhor se tivéssemos algum tipo de máquina de teletransporte. Bem, graças a Sonic Pi, temos uma. Tente isso:

```
mc_teleport 80, 40, 100
```

Uau! Isso foi um longo caminho até. Se você não estivesse em modo de voo, então você teria caído de volta ao chão. Se você tocar duas vezes no espaço para entrar no modo voo e se deslocar novamente, você permanecerá pairando no local onde você zap.

Agora, o que esses números significam? Nós temos três números que descrevem as coordenadas de onde queremos ir. Damos a cada número um nome - x, y e z:

- x - quão longe esquerda e direita (80 em nosso exemplo)
- y - quão alto queremos ser (40 no nosso exemplo)

• z - quão longe para a frente e para trás (100 no nosso exemplo)

Ao escolher valores diferentes para x, y e z, podemos nos teletransportar em qualquer lugar do mundo. Tente! Escolha diferentes números e veja onde você pode parar. Se a tela ficar preta é porque você se teletransportou para o solo ou em uma montanha. Basta escolher um valor superior para voltar para a terra. Continue explorando até encontrar um lugar que você goste ...

Usando as ideias até agora, vamos construir um Sonic Teleporter, que faz um divertido som de teletransporte enquanto atravessamos o mundo de Minecraft:

```
mc_message "Preparing to teleport...."
sample :ambi_lunar_land, rate: -1
sleep 1
mc_message "3"
sleep 1
mc_message "2"
sleep 1
mc_message "1"
sleep 1
mc_teleport 90, 20, 10
mc_message "Whoooosh!"
```

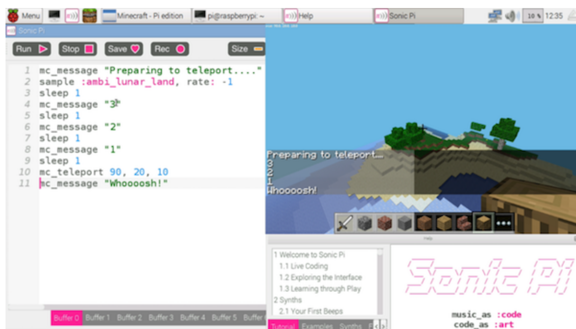


Fig. A.4: Som no Sonic Pi e teletransporte no Minecraft

Blocos Mágicos

Agora você encontrou um local legal, vamos começar a construir. Você poderia fazer o que estava acostumado e começar a clicar o mouse furiosamente para colocar blocos debaixo do cursor. Ou você poderia usar os truques do Sonic Pi. Tente isso:

```
x, y, z = mc_location
mc_set_block: melon, x, y + 5, z
```

Agora procure! Há um melão no céu! Dê uma olhada no código. O que nós fizemos? Na linha 1, nós fixamos a localização atual de Steve com as variáveis x, y e z. Estes correspondem às nossas coordenadas descritas acima. Usamos essas coordenadas no fn *mc_set_block* que colocará o bloco de sua escolha nas coordenadas especificadas. Para fazer algo mais alto no céu, precisamos aumentar o valor y, e é por isso que adicionamos 5. Vamos fazer uma longa caminhada deles:

```
live_loop :melon_trail do
  x, y, z = mc_location
  mc_set_block :melon, x, y-1, z
  sleep 0.125
end
```

Agora, passe para o Minecraft, certifique-se de estar no modo voador (toque duplo em espaço) e voe em todo o mundo. Olhe atrás de você para ver uma bela trilha de blocos de melão! Veja o tipo de padrões tortuosos que você pode fazer no céu.

Live Coding Minecraft

Aqueles que seguiram este tutorial nos últimos meses provavelmente terão suas mentes abertas neste momento. A trilha dos melões é muito legal, mas a parte mais emocionante do exemplo anterior é

que você pode usar o live_loop com o Minecraft! Para aqueles que não sabem, live_loop é a habilidade mágica especial de Sonic Pi que nenhuma outra linguagem de programação tem. Ele permite executar vários loops ao mesmo tempo e permite que você os altere enquanto eles correm. Eles são incrivelmente poderosos e divertidos. Eu uso live_loops para tocar música nas discotecas com o Sonic Pi - DJs usam discos e uso live_loops :-) No entanto, hoje vamos ao live coding tanto de música como de Minecraft.

Vamos começar. Execute o código acima e comece a fazer sua trilha de melão novamente. Agora, sem parar o código, basta simplesmente mudar *:melon* para *:brick* (tijolos) e rodar o código. Pronto, agora você está fazendo uma trilha de tijolos. Simples não! Gostaria de alguma música para acompanhar? Fácil. Tente isso:

```
live_loop :bass_trail do
  tick
  x, y, z = mc_location
  b = (ring :melon, :brick, :glass).look
  mc_set_block b, x, y -1, z
  note = (ring :e1, :e2, :e3).look
  use_synth :tb303
  play note, release: 0.1, cutoff: 70
  sleep 0.125
end
```

Agora, enquanto isso está começando a mudar o código. Altere os tipos de bloco - tente :water, :grass ou o seu tipo de bloco favorito. Além disso, tente mudar o valor de corte de 70 para 80 e, em seguida, até 100. Isso não é divertido?

Juntando tudo

Vamos combinar tudo o que vimos até agora com um pouco mais de magia. Vamos combinar nossa capacidade de teletransporte com colocação de blocos e música para fazer um Vídeo de Música Minecraft. Não se preocupe se você não entender tudo, basta digitar o

código e tocar, alterando alguns dos valores enquanto ele está sendo executado ao vivo. Divirta-se e até a próxima ...

```
live_loop :note_blocks do
  mc_message "This is Sonic Minecraft"
  with_fx :reverb do
    with_fx :echo, phase: 0.125, reps: 32 do
      tick
      x = (range 30, 90, step: 0.1).look
      y = 20
      z = -10
      mc_teleport x, y, z
      ns = (scale :e3, :minor_pentatonic)
      n = ns.shuffle.choose
      bs = (knit :glass, 3, :sand, 1)
      b = bs.look
      synth :beep, note: n, release: 0.1
      mc_set_block b, x+20, n-60+y, z+10
      mc_set_block b, x+20, n-60+y, z-10
      sleep 0.25
    end
  end
end

live_loop :beats do
  sample :bd_haus, cutoff: 100
  sleep 0.5
end
```

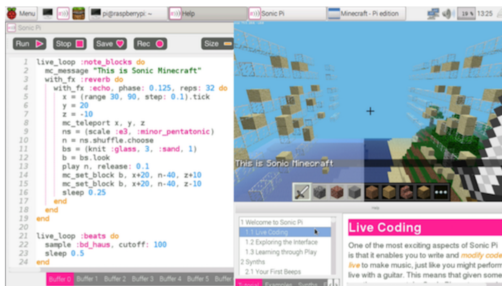


Fig. A.5: VJ com Sonic Pi e Minecraft

Apêndice B

Conhecimentos Fundamentais

Esta seção irá apresentar alguns conhecimentos fundamentais para obter maior proveito com Sonic Pi.

Vamos mostrar alguns atalhos de teclado disponíveis, como compartilhar seu trabalho e algumas dicas sobre a performance com Sonic Pi.

B.1 Usando Atalhos

Sonic Pi é tanto um instrumento como um ambiente de codificação. Os atalhos nos possibilita tocar de modo muito mais eficiente e natural - especialmente quando você está apresentando ao vivo na frente do público.

Muito do Sonic Pi pode ser controlado através do teclado. À medida que você ganha mais familiaridade trabalhando e atuando com o Sonic Pi, você provavelmente começará a usar os atalhos cada vez mais. Sou pessoalmente sou daquelas pessoas que digita sem ver o teclado (*touch-type*) e me sinto frustrado sempre que preciso usar o mouse. Por isso uso todos esses atalhos com regularidade.

Portanto, se você aprender os atalhos, você aprenderá a usar seu teclado de forma eficaz e estará codificado ao vivo como um profissional em nenhum momento.

No entanto, não tente aprendê-los todos de uma vez, apenas tente lembrar os que você usa mais e depois continue adicionando aos poucos à sua prática.

Consistência em plataformas

Imagine que você esteja aprendendo clarinete. Você esperaria que todos clarinetes de todas as marcas tivessem os controles de chaves semelhantes. Se não o fizeram, você teria dificuldade em alternar entre diferentes clarinetes e você ficaria preso a usar apenas uma marca.

Infelizmente, os três principais sistemas operacionais (Linux, Mac OS X e Windows) vêm com atalhos de teclado diferentes para atalhos como copiar, colar, etc. O Sonic Pi tenta seguir esses padrões. No entanto, em vez de tentar se adequar aos padrões de cada plata-

forma a prioridade do Sonic Pi é manter a consistência entre todas as plataformas. Isso significa que, quando você aprende os atalhos enquanto toca Sonic Pi no seu Raspberry Pi por exemplo, você poderá mover-se para o Mac ou PC e se sentir em casa.

Control e Meta

Parte da noção de consistência é a nomeação de atalhos. No Sonic Pi usamos os nomes *Control* e *Meta* para se referir às duas teclas de combinação principais. Em todas as plataformas, o *control* é o mesmo. No entanto, no Linux e no Windows, o *Meta* é realmente a tecla *Alt* enquanto no Mac *Meta* é a tecla *Command*. Para manter a consistência, usaremos o termo *Meta* - lembre-se de mapear isso para a tecla apropriada em seu sistema operacional.

Abreviaturas

Para ajudar a manter as coisas simples e legíveis, usaremos as abreviaturas *C-* para *Control* mais outra tecla e *M-* para *Meta* mais outra tecla. Por exemplo, se um atalho exige que você mantenha pressionado o *Meta* e a tecla *R*, vamos escrever isso como *M-r*. O traço -"significa "ao mesmo tempo que".

A seguir estão alguns dos atalhos mais comuns.

Run e Stop

Em vez de sempre buscar o mouse para executar seu código, você pode simplesmente pressionar **M-r** para **Run**. Da mesma forma, para parar de executar o código, você pode pressionar **M-s** para **Stop**.

Navegação

Eu realmente me perco sem os atalhos de navegação. Portanto, recomendo que você gaste um tempinho para aprendê-los. Estes atalhos funcionam extremamente bem caso você queira aprender *touch-type* pois eles usam letras padrões em vez de exigir que você mova a mão para o *mouse* ou as teclas em seu teclado.

Apenas utilizando o teclado você pode mover para o cursor ao início da linha com **C-a**, ao fim da linha com **C-e**, até a primeira linha com **C-p**, descer para a próxima linha com **C-n**, adiantar um caractere com **C-f** e voltar um caractere com **C-b**. Você pode até mesmo excluir todos os caracteres do cursor até o final da linha com **C-k**.

Arrumando o Código

Para auto-alinhar o seu código, simplesmente pressione **M-m**.

Sistema de Ajuda

Para visualizar o sistema de ajuda, você pode pressionar **M-i**. No entanto, um atalho muito mais útil para saber é **C-i**, que procurará a palavra embaixo do cursor e exibirá os documentos se encontrar alguma coisa. Ajuda instantânea!

Para uma lista completa, veja a seção 10.2 Cheatsheet de atalhos.

B.2 Lista de Atalhos

A lista a seguir é um resumo dos principais atalhos disponíveis no Sonic Pi. Consulte a Seção 10.1 para obter motivação e antecedentes.

Convenções

Nesta lista, usamos as seguintes convenções (onde o *Meta* é a tecla **Alt** no Windows e ou **Cmd** no Linux e no Mac):

- **C-a** significa segurar a tecla Control, depois pressionar a tecla **a** enquanto mantém os dois ao mesmo tempo e depois solte-os.
- **M-r** significa segurar a tecla Meta e, em seguida, pressione a tecla **r** enquanto mantém os dois ao mesmo tempo, depois soltando.
- **S-M-z** significa segurar a tecla Shift, depois a tecla Meta e, finalmente, a tecla **z** ao mesmo tempo e depois liberar.
- **C-M-f** significa segurar a tecla Controle, depois pressione a tecla Meta, finalmente a tecla **f**, ao mesmo tempo, e solte-a.

Manipulação de Aplicação Principal

- **M-r** - executar código
- **M-s** - parar código
- **M-i** - janela de ajuda
- **M-p** - janela de preferências
- **M-{** - Mudar o buffer para a esquerda
- **M-}** - Mudar o buffer para a direita
- **M-+** - Aumentar o tamanho do texto do buffer atual
- **M–** - Diminuir o tamanho do texto do buffer atual

Seleção / Copiar / Colar

- **M-a** - Selecionar tudo
- **M-c** - Copiar seleção para colar buffer
- **M-]** - Copiar seleção para colar buffer
- **M-x** - Cortar seleção para colar o buffer
- **C-]** - Cortar seleção para colar buffer
- **C-k** - Corte ao final da linha
- **M-v** - Colar do buffer de pasta para o editor
- **C-y** - Cole de pasta de buffer para editor
- **C-SPACE** - Definir marca. A navegação agora manipulará a região realçada. Use **C-g** para escapar.

Manipulação de texto

- **M-m** - Alinhar todo o texto
- **Tab** - Alinhar a linha ou seleção atual (ou selecionar preenchimento automático)
- **C-l** - Editor do centro
- **M-/** - Comentário / Desempenho linha atual ou seleção
- **C-t** - Transposição / permuta de caracteres
- **M-u** - Converte a próxima palavra (ou seleção) para maiúsculas.
- **M-l** - Converte a próxima palavra (ou seleção) em minúsculas.

Navegação

- **C-a** - Mover para o início da linha
- **C-e** - Mover para o fim da linha
- **C-p** - Mover para a linha anterior

- **C-n** - Mover para a próxima linha
- **C-f** - Avançar um caractere
- **C-b** - Mover para trás um caractere
- **M-f** - Avançar uma palavra
- **M-b** - Mover para trás uma palavra
- **C-M-n** - Mover linha ou seleção para baixo
- **C-M-p** - Mover linha ou seleção para cima
- **S-M-u** - Suba 10 linhas
- **S-M-d** - Desça 10 linhas
- **M-<** - Mover para o início do buffer
- **M->** - Mover para o final do buffer

Exclusão / Delete

- **C-h** - Excluir caractere anterior
- **C-d** - Exclua o próximo caractere

Recursos avançados do editor

- **C-i** - Mostrar docs por palavra sob o cursor
- **M-z** - Desfazer
- **S-M-z** - Redo
- **C-g** - Escape
- **S-M-f** - Alternar modo de tela cheia
- **S-M-b** - Alternar a visibilidade dos botões
- **S-M-l** - Alternar a visibilidade do log
- **S-M-m** - Alternar entre os modos luz / som
- **S-M-s** - Salve o conteúdo do buffer em um arquivo
- **S-M-o** - Carregar conteúdo do buffer a partir de um arquivo

B.3 Compartilhando

Sonic Pi tem tudo a ver com compartilhamento e aprendizagem uns com os outros.

Uma vez que você aprender a codificar música, verá que compartilhar suas composições é tão simples como enviar um e-mail. Por favor, compartilhe seu código com os outros para que eles possam aprender assim também com seu trabalho e até mesmo usar peças em um novo remix.

Se você não souber qual a melhor maneira de compartilhar seu trabalho com outras pessoas, recomendo colocar seu código no GitHub e sua música no SoundCloud. Dessa forma, você poderá facilmente ter um grande público.

Código -> GitHub

O GitHub é um site para compartilhar e trabalhar com o código. É usado por desenvolvedores profissionais, bem como artistas para compartilhar e colaborar com o código. A maneira mais simples de compartilhar um novo código (ou mesmo uma peça inacabada) é criar um *Gist*. O *Gist* é uma maneira simples de carregar o seu código de uma forma descomplicada de modo que outros podem ver, copiar e compartilhar.

Áudio -> SoundCloud

Outra maneira de compartilhar seu trabalho é gravar o áudio e carregá-lo para o SoundCloud. Uma vez que você carregou sua peça, outros usuários podem comentar e discutir seu trabalho. Também recomendo colocar um link para um *Gist* do seu código na descrição da faixa.

Para gravar o seu trabalho, aperte o botão **Rec** na barra de ferramentas e a gravação começará imediatamente. Clique em **Run** para iniciar o seu código se ainda não estiver em andamento. Quando terminar de gravar, pressione o botão piscando novamente e você será solicitado a inserir o nome do arquivo. A gravação será salva em um arquivo WAV, que pode ser editado e convertido em MP3 por qualquer programa gratuito (tente o Audacity, por exemplo).

Passando o Recado

Espero que você realmente compartilhe seu trabalho e que todos possamos aprender uns com os outros truques novos e progredir com Sonic Pi. Estou muito animado com o que você terá para mostrar.

B.4 Performance

Um dos aspectos mais emocionantes do Sonic Pi é que ele permite você usar o código como um instrumento musical. Isso significa que escrever código pode ser visto como uma nova maneira de tocar música. Chamamos isso de *live coding*.

Mostre sua tela

Quando você estiver "codando"recomendo que você mostre sua tela ao público. Caso contrário, é como tocar guitarra e esconder os dedos e as cordas. Quando pratico em casa, uso um Raspberry Pi e um pequeno projetor na parede da minha sala de estar. Você poderia usar sua TV ou em um projetor na escola/trabalho para fazer uma performance. Experimente, é muito divertido.

Forme uma banda

Não toque sozinho - forme uma banda de *live coding*! É muito bom improvisar e tocar com os outros. Um pode fazer as batidas, outro um ambiente de fundo, outro ainda melodias e harmonias, texturas etc. E por que não combinar solos, duos, ou estabelecer estratégias de pergunta e resposta? Veja quais combinações interessantes de sons vocês podem combinar. E que tal formar uma banda com outros músicos? Pode ser uma experiência e tanto.

TOPLAP

Live coding não é algo inédito - várias pessoas vêm fazendo isso há alguns tempo, geralmente usando sistemas personalizados que eles

próprios criaram. Um ótimo lugar para descobrir e explorar mais sobre é o TOPLAP.

Algorave

Outro excelente meio para explorar o mundo da codificação ao vivo é Algorave. Aqui você pode encontrar tudo sobre *live coding* para fazer música em casas noturnas.

Referência

[1] Sonic Pi. http://sonic-pi.net. 2013-17.

[2] Sam Aaron. Sonic pi–performance in education, technology and art. *International Journal of Performance Arts and Digital Media*, 12(2):171–178, 2016.

[3] Raspberry Pi. https://www.raspberrypi.org. 2012-17.