TUTORIAL DE SONIC PI

SAM AARON

Traducció de Versembrant

Versembrant
Escola popular

beats.versembrant.cat

TUTORIAL DE SONIC PI

SAM AARON

Traducció de Versembrant

INDEX DE CONTINGUTS

1. BENVINGUT AMIC :-)

Benvingut a Sonic Pi. Espero que estiguis tan entusiasmat per començar a crear els teus propis sons com ho estic jo per ensenyar-te'n. Serà un viatge molt divertit en què ho aprendràs tot sobre la música, la síntesi, la programació, la composició, la interpretació i molt més.

Però... un moment, que groller que sóc! Deixa'm que em presenti: sóc Sam Aaron, el creador de Sonic Pi. Pots trobar-me a @samaaron a Twitter i estaré encantat de saludar-te. També et pot interessar saber més sobre les meves actuacions de codificació en viu, en les quals produeixo música amb Sonic Pi en directe davant del públic.
Si tens alguna idea per millorar Sonic Pi, si us plau, fes-me-la arribar, els comentaris són molt útils. Mai se sap, la teva idea podria ser la propera gran característica!

Aquest tutorial està dividit en seccions agrupades per categories. Encara que ho he escrit perquè tingui una progressió d'aprenentatge fàcil de principi a fi, sent-te molt lliure d'entrar i sortir de les seccions com et sembli.
Si creus que falta alguna cosa, fes-m'ho saber i ho tindré en compte de cara a una futura versió.
Finalment, veure el codi dels altres en directe és una forma magnífica d'aprendre. Regularment transmeto en viu a https://youtube.com/samaaron així que, si us plau, passa't, saluda i fes-me moltes preguntes :-)

Bé, comencem...

1.1 Codificació en viu

Un dels aspectes més emocionants de Sonic Pi és que et permet escriure i modificar el codi en directe per fer música, igual que podries actuar en directe amb una guitarra. Això significa que, amb una mica de pràctica, pots portar Sonic Pi a l'escenari i actuar-hi.

Allibera la teva ment

Abans d'entrar en els detalls reals de com funciona Sonic Pi a la resta d'aquest tutorial, m'agradaria mostrar-te l'experiència de codificar en viu. No et preocupis si no entens gaire (o res) d'això. Només es tracta de mantenir-te al teu seient i gaudir...

Un bulce en viu

Comencem, copia el següent codi en un *buffer* buit:

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

end
```

Ara, prem el botó *Run* i escoltaràs un bonic i ràpid bombo colpejant. Si en algun moment vols aturar el so, només has de prémer el botó Stop. Però no ho facis encara... Abans, segueix aquests passos:

1. Assegura't que el so de bombo segueix funcionant
2. Canvia el valor sleep de 0,5 per un valor una mica més alt, per exemple, 1.
3. Prem novament el botó *Run*
4. Observa com ha canviat la velocitat del bombo.
5. Finalment, recorda aquest moment, aquesta és la primera vegada que has codificat en viu amb Sonic Pi i és poc probable que sigui la darrera...

D'acord, això ha estat força senzill. Compliquem-ho una mica. Per sobre de `sample :bd_haus` afegeix la línia `sample :ambi_choir, rate: 0.3`. El teu codi hauria de tenir aquest aspecte:

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

Ara, comença a jugar. Canvia les freqüències - què passa quan utilitzes valors alts, valors baixos o valors negatius? Observa què passa quan canvies el valor de `rate`: per a la mostra `:ambi_choir` lleugerament (diguem a `0,29`). Què passa si tries un valor `sleep` realment petit? Prova si pots fer que vagi tan ràpid que el teu ordinador es col·lapsi perquè no pot seguir el ritme (si això passa, simplement escull un temps de repòs major i prem *Run* de nou).

Intenta comentar una de les línies *Sample* afegint un # al principi:

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

Fixa't com li diu a l'ordinador que ho ignori, per a que no ho escoltem. Això s'anomenta "comentari". A Sonic Pi podem utilitzar els comentaris per treure i afegir coses a la mescla.
Finalment, deixa'm proposar-te un exercici divertit perquè juguis. Agafa el codi de baix, i copia'l en un *buffer* lliure. Ara, no tractis d'entendre res més enllà de veure que hi ha dos bucles - és a dir, dues coses donant voltes al mateix temps. Ara, fes el que saps fer millor: experimentar i jugar. Aquí tens alguns suggeriments:

- Prova de canviar els valors blaus `rate`: per escoltar com canvia el so de la mostra.
- Prova de canviar els temps `sleep` i escolta que tots dos bucles poden girar a diferents velocitats.
- Prova de descomentar la línia de mostra (elimina el #) i gaudeix del so de la guitarra tocada al revés.
- Prova de canviar qualsevol dels valors blaus de `mix`: a números entre 0 (no al mix) i 1 (totalment al `mix`).

Recorda prémer *Run* i escoltaràs el canvi la propera vegada que el bucle faci una volta. Si no et funciona, no et preocupis: prem Stop, esborra el codi al buffer, enganxa una còpia nova i estaràs preparat per tornar a fer el bucle. Cometent errors és com aprendràs més ràpidament...

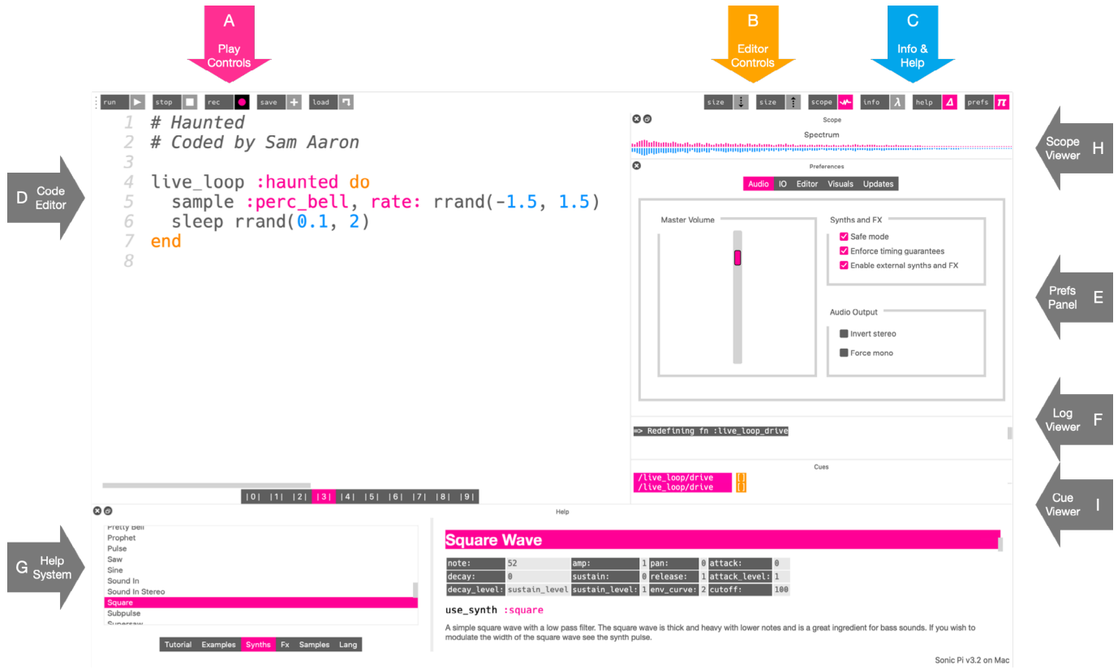
```
live_loop :guit do
  with_fx :echo, mix: 0.3, phase: 0.25 do
    sample :guit_em9, rate: 0.5
  end
#  sample :guit_em9, rate: -0.5
  sleep 8
end

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

Ara segueix jugant i experimentant fins que et piqui la curiositat per saber com funciona tot això i comencis a preguntar-te què més pots fer-ne. Ara estàs preparat per llegir la resta del tutorial.

Així doncs, a què esperes?

1.2 Exploració de la interfície de Sonic Pi

Sonic Pi té una interfície molt senzilla per codificar música. Dediquem una mica de temps a explorar-la.



A - *Play Controls* (Controls de reproducció)
B - *Editor Controls* (Controls de l'editor)
C - *Info and Help* (Informació i ajuda)
D - *Code Editor* (Editor de codi)
E - *Prefs Panel* (Panell de preferències)
F - *Log Viewer* (Visor de registres)
G - *Help System* (Sistema d'ajuda)
H - *Scope Viewer* (Visor d'espectre)
I - *Cue Viewer* (Visor de senyals)

A. *Play Controls* (Controls de reproducció)

Aquests botons roses són els principals controls per iniciar i aturar els sons. El botó *Run* serveix per executar el codi a l'editor, Stop per aturar tot el codi en execució, Save per guardar el codi en un fitxer extern i Record per crear un enregistrament (un fitxer WAV) del so que es reprodueix.

B. *Editor Control* (Controls de l'editor)

Aquests botons de color taronja et permeten manipular l'editor de codi. Els botons *Size +* i *Size -* us permeten fer el text més gran i més petit.

C. *Info and Help* (Informació i ajuda)

Aquests botons blaus et donen accés a la informació, ajuda i preferències. El botó *Info* obrirà la finestra d'informació que conté informació sobre Sonic Pi en si mateix - l'equip central, la història, els col·laboradors i la comunitat. El botó *Help* activa el sistema d'ajuda (G) i el botó *Prefs* activa la finestra de preferències que permet controlar alguns paràmetres bàsics del sistema.

D. *Code Editor* (Editor de codi)

Aquesta és l'àrea on escriuràs el teu codi i composaràs/interpretaràs música. És un editor de text senzill on pots escriure codi, esborrar-lo, tallar-lo i enganxar-lo, etc. Pensa-hi com una versió molt bàsica de Word o Google Docs. L'editor pintarà automàticament les paraules en funció del seu significat al codi. Això pot semblar estrany al principi, però aviat ho trobaràs molt útil. Per exemple, sabràs que alguna cosa és un número perquè és blava.

E. *Prefs Panel* (Panell de preferències)

Sonic Pi suporta una sèrie de preferències ajustables a les quals es pot accedir activant el botó *Prefs* al conjunt de botons d'informació i ajuda. Això activarà la visibilitat del panell de preferències, que inclou una sèrie d'opcions que es poden canviar. Alguns exemples són forçar el mode mono, invertir l'estèreo, alternar la verbositat de la sortida del registre i també un botó que regula el volum i un selector d'àudio a la Raspberry Pi.

F. *Log Viewer* (Visor de registres)

Quan executis el teu codi, la informació sobre el que el programa està fent es mostrarà al visor de registres. Per defecte, veuràs un missatge per a cada so que creïs amb l'hora exacta en què s'ha activat el so. Això és molt útil per depurar el teu codi i entendre què està fent exactament.

G. *Help system* (Sistema d'ajuda)

Una de les parts més importants de la interfície de Sonic Pi és el sistema d'ajuda que apareix a la part inferior de la finestra. Es pot activar i desactivar fent clic al botó blau *Help*. El sistema d'ajuda conté ajuda i informació sobre tots els aspectes de Sonic Pi, incloent aquest tutorial, una llista de sintetitzadors disponibles, mostres, exemples, efectes (FX per abreviar) i una llista completa de totes les funcions que Sonic Pi proporciona per codificar música.

H. *Scope Viewer* (Visor d'abast)

El visor d'abast et permet veure el so que estàs escoltant. Pots veure fàcilment que l'ona de serra sembla una serra i que el beep bàsic és una ona sinusoïdal corbada. També pots veure la diferència entre sons forts i silenciosos per la mida de les línies. Hi ha 3 àmbits per jugar - el predeterminat és un àmbit combinat per als canals esquerre i dret, hi ha un àmbit estèreo que dibuixa un àmbit separat per a cada canal. Finalment, hi ha un indicador de la corba de Lissajous que mostra la relació de fase entre els canals esquerre i dret i et permet dibuixar imatges boniques amb el so (https://en.wikipedia.org/wiki/Lissajous_curve).

I. *Cue Viewer* (Visor de senyals)

Tots els esdeveniments interns i externs -anomenats cues (senyals) a Sonic Pi- es registren automàticament al *Cue Viewer*. Per exemple, si tens una controladora MIDI connectada i prems un dels seus botons, veuràs un nou esdeveniment cue al *Cue Viewer* que t'indicarà el nom de la controladora i el botó que has premut. Quan hagis dominat els fonaments de la creació i producció de sons, començaràs a voler collar sons o seccions senceres de sons basats en esdeveniments com aquests. Un esdeveniment de cue és simplement una indicació que alguna cosa ha passat. Per exemple, cada vegada que un bucle en viu s'inicia, envia un esdeveniment *cue* que es registra al *Cue Viewer*. A més, els esdeveniments externs com els missatges MIDI d'equips MIDI connectats i els missatges OSC d'altres programes o ordinadors també es mostren al Cue Viewer. També és possible generar directament esdeveniments cue utilitzant la funció cue. Qualsevol cosa que aparegui al *Cue Viewer* es pot utilitzar per desencadenar alguna cosa. Això es tracta amb més detall a les seccions 10 a 12 d'aquest tutorial.

1.3 Aprendre jugant

Sonic Pi t'anima a aprendre informàtica i música a través del joc i l'experimentació. El més important és que et diverteixis, i abans que te n'adonis hauràs après accidentalment a codificar, composar i interpretar

No existeixen els errors

Ja que estem parlant d'aquest tema, permet-me donar-te un consell que he après durant els meus anys de codificació en directe amb música: no existeixen els errors, només les oportunitats. Això és una cosa que he sentit sovint en relació amb el jazz, però funciona igualment bé amb la codificació en directe. No importa l'experiència que tinguis -des d'un complet principiant fins a un experimentat codificador en viu-, executaràs algun codi que tingui un resultat completament inesperat. Pot ser que soni increïblement bé, i en aquest cas, guarda'l. No obstant això, pot sonar totalment discordant i fora de lloc. Tant se val què hagi succeït, el que importa és el que facis després amb allò. Pren el so, manipula'l i transforma'l en una cosa increïble. El públic es tornarà boig.

Comença de forma senzilla

Quan estàs aprenent, és temptador voler fer coses increïbles de seguida. Tanmateix, retén aquest pensament i fixa'l com un objectiu llunyà que podrà assolir més endavant. Ara per ara, pensa en el més senzill que puguis escriure, que sigui divertit i gratificant, que sigui un petit pas cap a allò sorprenent que tens al cap. Quan tinguis una idea sobre aquest pas senzill, intenta construir-lo, juga-hi i mira quines noves idees et dóna. En poc temps estaràs molt ocupat divertint-te i fent veritables progressos.
Això sí, assegura't de compartir la teva feina amb els altres!

2. SINTETITZADORS

Bé, fins aquí les introduccions: passem al so.

En aquesta secció abordarem els fonaments de l'activació i la manipulació dels sintetitzadors. Sintetitzador és una paraula elegant per referir-se a una cosa que crea sons. Normalment els sintetitzadors són força complicats d'utilitzar - especialment els sintetitzadors analògics com els mòduls Eurorack connectats entre si per un embolic de cables. Tanmateix, Sonic Pi us ofereix molta d'aquesta potència d'una manera molt senzilla i accessible.

No et deixis enganyar per la simplicitat immediata de la interfície de Sonic Pi. Pots endinsar-te en la manipulació de sons molt sofisticats si això és el que busques. Ves-te preparant...

2.1.Els teus primers *beeps*

Observa el següent codi:

```
play 70
```

Aquí és on comença tot. Endavant, copia'l i enganxa'l a la finestra de codi a la part superior de l'aplicació (el gran espai blanc sota el botó Run). Ara, prem Run...

Beep!

Intens. Prem-lo una altra vegada. I una altra vegada. I una altra vegada...

Uau, divertit! Segur que podries seguir fent això tot el dia. Però espera, abans de perdre't en un flux infinit de beeps, prova de canviar el número:

```
play 75
```

Aprecies la diferència? Prova un número més baix:

```
play 60
```

Com pots escoltar, els números més baixos fan beeps més greus i els números més alts fan beeps més aguts. Igual que en un piano, les tecles de la part inferior del piano (el costat esquerre) toquen notes més greus i les tecles de la part superior del piano (el costat dret) toquen notes més agudes.

Resulta que el Do de la 4a octava (C en notació anglesa) s'identifica amb el número 60. `play 60` per tant, toca el Do de la 4a octava. Per tocar la següent tecla de piano a la dreta, cal afegir 1 a 60 i després escriure `play 61`, que en aquest cas és la tecla negra del Do sostingut. Per tocar el Re, la següent tecla a la dreta, `play 62`.

No et preocupis si no li trobes el sentit, a mi també em passava quan vaig començar. L'únic que importa ara és que sàpigues que els números baixos fan beeps més greus i els números alts fan beeps més aguts.

Acords

Tocar una nota és força divertit, però tocar-ne moltes al mateix temps pot ser encara millor. Prova-ho:

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

Jazzy! Així doncs, quan escrius diverses `play`s, totes sonen alhora. Prova-ho tu mateix: quins números sonen bé junts? Quins sonen fatal? Experimenta, explora i descobreix-ho per tu mateix.

Melodia

Tocar notes i acords és divertit, però què tal una melodia? I si vols tocar una nota darrere l'altra i no alhora? Doncs és fàcil, només has d'escriure sleep entre les notes:

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

Que bonic, un petit arpegi. Aleshores, què vol dir `1` a `sleep 1` ? Bé, significa la durada del `sleep`. En realitat significa `sleep` durant un temps, però per ara podem pensar-hi com `sleep` durant 1 segon. Aleshores, què passa si volem que el nostre arpegi soni una mica més ràpid? Bé, hem de fer servir valors de `sleep` més curts. Provem la meitat, és a

```
dir, 0,5:
play 72
sleep 0.5
play 75
sleep 0.5
play 79
```

Fixa't que sona més ràpid. Ara, intenta-ho per tu mateix, canvia els temps - utilitza diferents temps i notes.
Una cosa que es pot provar són les notes intermèdies, com ara tocar `52,3` o `52,63`. No hi ha cap necessitat de cenyir-se a les notes senceres estàndard. Juga i diverteix-te.

Noms tradicionals de notes

Per a aquells que ja saben una mica de notació musical (no et preocupis si no la domines, no la necessites per divertir-te), pot ser que vulguis escriure una melodia utilitzant noms de notes com Do i Fa# en lloc de números. Sonic Pi t'ho posa fàcil. Pots fer el següent:

```
play :C
sleep 0.5
play :D
sleep 0.5
play :E
```

Recorda posar els dos punts : davant del nom de la nota perquè aparegui en rosa. A més, pots especificar l'octava afegint un número després del nom de la nota:

```
play :C3
sleep 0.5
play :D3
sleep 0.5
play :E4
```

Si vols fer una nota sostinguda, afegeix una `s` després del nom de la nota, per exemple, `play :Fs3` i si vols fer una nota bemoll, afegeix una `b`, per exemple, `play :Eb3`.

També existeix una pràctica drecera per a un descans (una forma musical de dir que no es toqui res en el lloc d'una nota) usant :r, :rest o nil.

Ara, diverteix-te creant les teves pròpies melodies.

2.2 Opcions de sintetitzador: Amp i Pa

A més de permetre't controlar quina nota tocar o quina mostra disparar, Sonic Pi ofereix tot un seguit d'opcions per elaborar i controlar els sons. N'abordarem moltes en aquest tutorial i hi ha una extensa documentació per a cadascuna d'elles al sistema d'ajuda. Tanmateix, ara per ara en presentarem dues de les més útils: l'amplitud (*amp* per abreujar) i el panorama (*pan* per abreujar). En primer lloc, vegem quines opcions hi ha.

Opcions

Les opcions (o *opts* per abreujar) són controls que passes a `play` que modifiquen i controlen aspectes del so que escoltes. Cada sintetitzador té el seu conjunt d'opcions per ajustar de forma precisa el seu so. Tot i això, hi ha conjunts comuns d'opts compartides per molts sons, com les opts d'`amp`: i d'envolupant (tractades en una altra secció).

Les opts tenen dues parts principals, el nom (el nom del control) i el valor (el valor al qual vols ajustar el control). Per exemple, pots tenir una *opt* anomenada `cheese`: i voler-la establir amb un valor d' `1`.

Les opcions es passen a les crides play utilitzant una coma , i després el nom de l'opció com amp: (no oblidis els dos punts :) i després un espai i el valor de l'opció. Per exemple:

```
play 50, cheese: 1
```

(adona't que cheese: no es una opt vàlida, només l'estem utilitzat com a exemple).

Pots utilitzar múltiples opts separant-les amb una coma:

```
play 50, cheese: 1, beans: 0.5
```

L'ordre de les opts no importa, és a dir, l'exemple següent és idèntic al primer:

```
play 50, beans: 0.5, cheese: 1
```

Les opcions que no són reconegudes pel sintetitzador són simplement ignorades (com el formatge i les mongetes, que són noms d'opcions clarament ridículs).

Si accidentalment utilitzes la mateixa opció dues vegades amb valors diferents, guanya la darrera. Per exemple, beans: aquí tindrà el valor 2 en lloc de 0,5:

```
play 50, beans: 0.5, cheese: 3, eggs: 0.1, beans: 2
```

Moltes coses a Sonic Pi accepten opts, així que només has de dedicar una mica de temps a aprendre a fer-les servir i estaràs preparat. Comencem jugant amb la nostra primera *opt:* `amp:`.

Amplitud

L´amplitud és una representació informàtica de la intensitat d´un so. Una amplitud alta produeix un so fort i una amplitud baixa produeix un so fluix. Com Sonic Pi utilitza números per representar el temps i les notes, també utilitza números per representar l'amplitud. Una amplitud de 0 és silenciosa (no se sent res) mentre que una amplitud d'1 és un volum normal. Fins i tot pots pujar l'amplitud a 2, 10, 100. No obstant això, has de tenir en compte que quan l'amplitud total és massa alta, Sonic Pi utilitza el que s'anomena un compressor per reduir-la i assegurar-se que el so no sigui massa fort per a les teves orelles. Això sovint pot fer que el so sigui tèrbol i estrany. Per tant, tracta d'utilitzar amplituds baixes, és a dir, al rang de 0 a 0,5 per evitar la compressió.

Amplifica-ho

Per canviar l'amplitud d'un so, pots utilitzar l'opt `amp`: Per exemple, per tocar a la meitat de l'amplitud marca 0,5:

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

Per tocar a doble amplitud marca 2:

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

L'opció `amp:` només modifica la crida a què està associada. Així, en aquest exemple, la primera crida està a mig volum i la segona torna al valor per defecte (1):

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

Per descomptat, es poden utilitzar diferents valors d'amp: per a cada crida:

```
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
```

Panorama

Una altra opció divertida és `pan:`, que controla la panoràmica d'un so en estèreo. Desplaçar un so cap a l'esquerra significa que l'escoltaràs per l'altaveu esquerre, i desplaçar-lo cap a la dreta vol dir que l'escoltaràs per l'altaveu dret. Per als nostres valors, utilitzem un -1 per representar totalment l'esquerra, un 0 per representar el centre i un 1 per representar totalment la dreta al camp estèreo. Per descomptat, som lliures de fer servir qualsevol valor entre -1 i 1 per controlar la posició exacta del nostre so.

Reproduïm un beep per l'altaveu esquerre:

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

Ara, reproduïm-lo per l'altaveu dret:

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

Finalment, reproduïm-lo des del centre de tots dos (la posició per defecte):

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

Ara, diverteix-te canviant l'amplitud i la panoràmica dels teus sons!

2.3.Canvi de sintetitzadors

Fins ara ens hem divertit força fent *beeps*. No obstant això, probablement estiguis començant a avorrir-te del soroll bàsic dels *beeps*. És això tot el que ofereix Sonic Pi? No hi ha res més en la codificació en viu que reproduir *beeps*? Sí, n'hi ha, i en aquesta secció explorarem algunes de les emocionants gammes de sons que Sonic Pi té per oferir.

Sintetitzadors

Sonic Pi té una sèrie d'instruments diferents que anomena *synths* (que és l'abreviatura en anglès de sintetitzadors). Mentre que les mostres representen sons pregravats, els sintetitzadors són capaços de generar nous sons depenent de com els controlis (cosa que explorarem més endavant en aquest tutorial). Els sintetitzadors de Sonic Pi són molt potents i expressius i et divertiràs molt explorant-los i jugant-hi. En primer lloc, aprendrem a seleccionar el sintetitzador que utilitzarem.

Saw Wave i Prophet

Un so divertit és el *saw wave* - provem-lo:

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

Provem un altre so - el *prophet*:

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

Provem ara la combinació dels dos sons. Un després de l'altre:

```
use_synth :saw
```

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

Ara, diversos sons alhora (sense *sleep* entre les successives crides a play):

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

Adona't que el comandament *use_synth* només afecta a les següents crides.

Pensa-ho com un gran interruptor: les noves crides reproduiran el sintetitzador a què estigui apuntant en aquell moment. Pots canviar a un nou sintetitzador amb *use_synth*.

Descobrir els sintetitzadors

Per veure quins sintetitzadors té Sonic Pi per jugar, fes una ullada a l'opció *Synths* al menú de la part inferior d'aquesta pantalla d'ajuda (entre *Examples* & *Fx*). N'hi ha més de 20 per triar. Te'n deixo alguns dels meus favorits:

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

Ara juga canviant de sintetitzador durant la teva música. Diverteix-te combinant sintetitzadors per crear nous sons, així com utilitzant diferents sintetitzadors per a diferents seccions de la teva música.

2.4. Durada amb envolupants

En una secció anterior, vam veure com podem fer servir el comandament `sleep` per controlar quan activar els nostres sons. Tot i això, encara no hem estat capaços de controlar la durada dels nostres sons.

Per donar-nos una eina simple, però poderosa, de controlar la durada dels nostres sons, Sonic Pi proporciona la noció d'una envolupant d'amplitud ADSR (descobrirem què significa ADSR més endavant en aquesta secció). Una envolupant d'amplitud ofereix dos aspectes útils de control

- control sobre la durada d'un so
- control sobre l'amplitud d'un so

Durada

La durada és el temps que dura el so. Una durada més gran significa que el so s'escolta durant més temps. Tots els sons de Sonic Pi tenen una envolupant d'amplitud controlable, i la durada total d'aquesta envolupant és la durada del so. Per tant, en controlar l'envolupant se'n controla la durada.

Amplitud

L'envolupant ADSR no tan sols controla la durada del so, sinó que també et dóna un control fi sobre la seva amplitud. Tots els sons audibles comencen i acaben en silenci i contenen alguna part no silenciosa entre ells. Les envolupants et permeten lliscar i mantenir l'amplitud de les parts no silencioses del so. És com donar a algú instruccions sobre com pujar i abaixar el volum d'un amplificador de guitarra. Per exemple, pots demanar a algú que "comenci en silenci, pugi lentament fins al volum màxim, el mantingui una mica i després torni ràpidament al silenci". Sonic Pi us permet programar exactament aquest comportament amb les envolupants.

Per recapitular, com hem vist abans, una amplitud de `0` és el silenci i una amplitud d'`1` és el volum normal.

Ara, vegem cadascuna de les parts de les envolupants per separat.

Fase d'alliberament (release)

L'única part de l'envolupant que es fa servir per defecte és el temps d'alliberament. Aquest és el temps que triga el so del sintetitzador a apagar-se. Tots els *sintes* tenen un temps d'alliberament d'1, cosa que significa que per defecte tenen una durada d'1 beat (que al BPM per defecte de 60 és d'1 segon):

```
play 70
```

La nota serà audible durant un segon. Endavant, cronometra el temps :-) Aquesta és la versió curta de la versió més llarga i explícita:

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

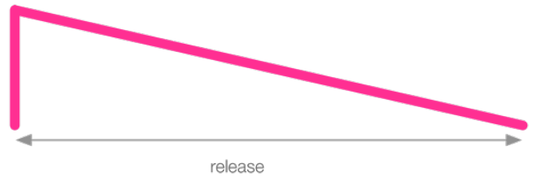
Nota com el so és exactament el mateix (el so dura un segon). Tanmateix, ara és molt més fàcil canviar-ne la durada modificant el valor de l'opció `release`:

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

Podem fer que el so del sinte duri molt poc marcant un temps d'alliberament molt petit:

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

La durada de l'alliberament del so s'anomena release phase (fase d'alliberament) i per defecte és una transició lineal (és a dir, una línia recta). El següent diagrama il·lustra aquesta transició:



La línia vertical a l'extrem esquerre del diagrama mostra que el so comença amb una amplitud de `0`, però puja a la màxima amplitud immediatament (aquesta és la fase d'atac que veurem a continuació). Un cop assoleix la màxima amplitud, es mou en línia recta fins a zero, prenent el temps especificat per `release:`. Els temps d'alliberament més llargs produeixen esvaïments de sintetitzador més llargs.

Per tant, pots canviar la durada del teu so modificant el temps d'alliberament. Juga a afegir temps d'alliberament a la teva música.

Fase d'atac (*attack*)

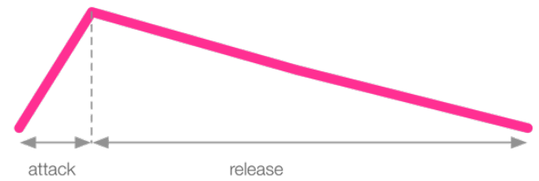
Per defecte, la fase d'atac és `0` per a tots els sintes, fet que significa que passen de `0` d'amplitud a `1` immediatament. Això dóna al sintetitzador un so percusssor inicial. No obstant això, és possible que vulguis esvair el so. Això es pot aconseguir amb l'opció `attack:`. Prova de difuminar alguns sons:

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

Pots utilitzar vàries opcions al mateix temps. Per exemple per a un atac curt i un alliberament llarg prova:

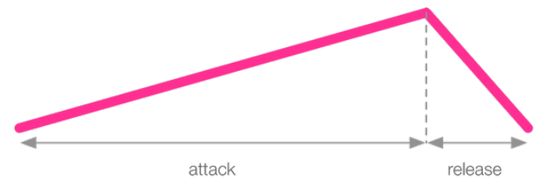
```
play 60, attack: 0.7, release: 4
```

Aquesta envolupant d'atac curt i alliberament llarg s'il·lustra en el diagrama següent:



Evidentment, pots fer-ho al revés. Prova un atac llarg i un alliberament curt:

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



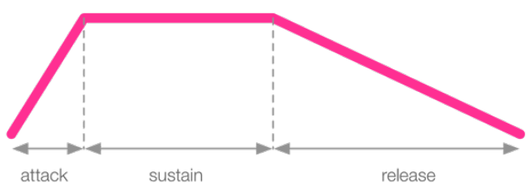
Finalment, també pots tenir atac i alliberament curts per a sons curts.

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

Fase de manteniment (sustain)

A més d'especificar els temps d'atac i alliberament, també pots especificar un temps de manteniment per controlar la fase de manteniment. Aquest és el temps durant el qual el so es manté a plena amplitud entre les fases d'atac i alliberament.

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

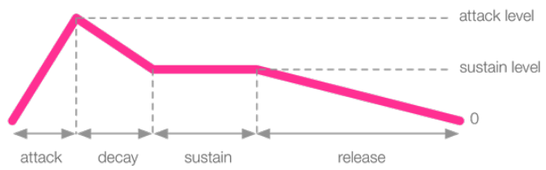


El temps de manteniment és útil per als sons importants als quals vulguis donar plena presència a la mescla abans d'entrar en una fase d'alliberament opcional. Per descomptat, és totalment vàlid ajustar les opts tant `attack:` com `release:` a `0` i només utilitzar el manteniment per no tenir absolutament cap fade in o fade out al so. No obstant això, tingues en compte que un alliberament de `0` pot produir espetecs a l'àudio i sovint és millor utilitzar un valor molt petit, com `0,2`.

Fase de decaïment (decay)

Per a un major nivell de control, també pots especificar un temps de decaïment. Aquesta és una fase de l'envolupant que encaixa entre les fases d'atac i manteniment i especifica un temps en què l'amplitud caurà des del `attack_level:` fins al `decay_level:` (que llevat que ho estableixis explícitament s'ajustarà al `sustain_level:`). Per defecte, l'opció `decay:` és `0` i tant el nivell d'atac com el de manteniment són `1`, per tant hauràs d'especificar-los perquè el temps de decaïment tingui algun efecte:

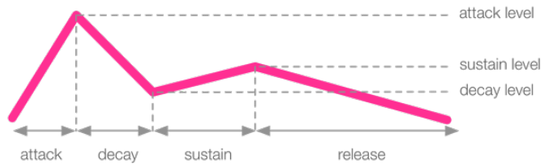
```
play 60, attack: 0.1, attack_level: 1, decay: 0.2,
sustain_level: 0.4, sustain: 1, release: 0.5
```



Nivell de decaïment

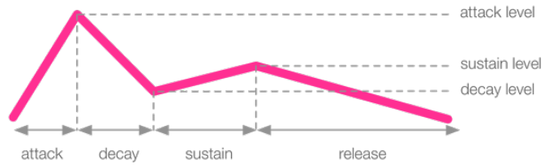
Un últim truc és que, encara que l'opció decay_level: té per defecte el mateix valor que sustain_level:, pots establir explícitament valors diferents per tenir un control total sobre l'envolupant. Això et permet crear envolupants com les següents:

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



També és possible establir el `decay_level:` per sobre del `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
```



Envolupants ADSR

En resum, les envolupants ADSR de Sonic Pi tenen les fases següents

1. attack (atac) - temps des de l'amplitud 0 fins al attack_level,
2. decay (decaïment) - temps per moure l'amplitud des de l'attack_level fins al decay_level,
3. sustain (manteniment) - temps per moure l'amplitud des del nivell de decay_level fins al nivell de sustain_level,
4. release (alliberament) - temps per moure l'amplitud des del nivell sustain_level a 0

És important assenyalar que la durada d'un so és la suma dels temps de cadascuna d'aquestes fases. Per tant, el so següent tindrà una durada de 0,5 + 1 + 2 + 0,5 = 4 temps:

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

Ara comença a jugar afegint envolupants als teus sons...

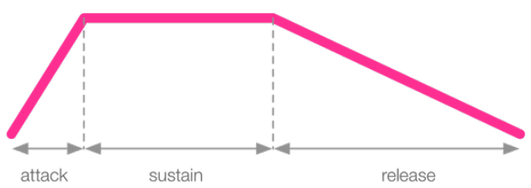
Finalment, també pots tenir atac i alliberament curts per a sons curts.

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

Fase de manteniment (sustain)

A més d'especificar els temps d'atac i alliberament, també pots especificar un temps de manteniment per controlar la fase de manteniment. Aquest és el temps durant el qual el so es manté a plena amplitud entre les fases d'atac i alliberament.

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

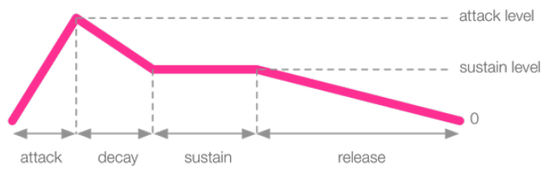


El temps de manteniment és útil per als sons importants als quals vulguis donar plena presència a la mescla abans d'entrar en una fase d'alliberament opcional. Per descomptat, és totalment vàlid ajustar les opts tant `attack:` com `release:` a `0` i només utilitzar el manteniment per no tenir absolutament cap fade in o fade out al so. No obstant això, tingues en compte que un alliberament de `0` pot produir espetecs a l'àudio i sovint és millor utilitzar un valor molt petit, com `0,2`.

Fase de decaïment (decay)

Per a un major nivell de control, també pots especificar un temps de decaïment. Aquesta és una fase de l'envolupant que encaixa entre les fases d'atac i manteniment i especifica un temps en què l'amplitud caurà des del `attack_level:` fins al `decay_level:` (que llevat que ho estableixis explícitament s'ajustarà al `sustain_level:`). Per defecte, l'opció `decay:` és `0` i tant el nivell d'atac com el de manteniment són `1`, per tant hauràs d'especificar-los perquè el temps de decaïment tingui algun efecte:

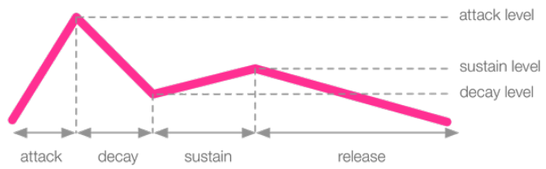
```
play 60, attack: 0.1, attack_level: 1, decay: 0.2,
sustain_level: 0.4, sustain: 1, release: 0.5
```



Nivell de decaïment

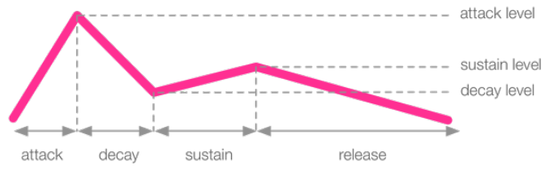
Un últim truc és que, encara que l'opció decay_level: té per defecte el mateix valor que sustain_level:, pots establir explícitament valors diferents per tenir un control total sobre l'envolupant. Això et permet crear envolupants com les següents:

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



També és possible establir el `decay_level:` per sobre del `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
```



Envolupants ADSR

En resum, les envolupants ADSR de Sonic Pi tenen les fases següents

1. attack (atac) - temps des de l'amplitud 0 fins al `attack_level`,
2. decay (decaïment) - temps per moure l'amplitud des de l'attack_level fins al `decay_level`,
3. sustain (manteniment) - temps per moure l'amplitud des del nivell de decay_level fins al nivell de `sustain_level`,
4. release (alliberament) - temps per moure l'amplitud des del nivell `sustain_level` a 0

És important assenyalar que la durada d'un so és la suma dels temps de cadascuna d'aquestes fases. Per tant, el so següent tindrà una durada de 0,5 + 1 + 2 + 0,5 = 4 temps:

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

Ara comença a jugar afegint envolupants als teus sons...

3. MOSTRES

Una altra forma estupenda de desenvolupar la teva música és utilitzar sons pregravats. A la gran tradició del hip-hop, anomenem aquests sons pregravats *samples* (o mostres). Així, si agafes un micròfon a l'exterior, vas i enregistres el suau so de la pluja colpejant la lona, acabes de crear un *sample*.

Sonic Pi us permet fer moltes coses divertides amb els *samples*. No només ve amb 130 mostres de domini públic a punt perquè puguis improvisar, sinó que et permet tocar i manipular les teves pròpies mostres. Veiem-ho...

3.1. Disparar mostres

Reproduir *beeps* és només el principi. Una cosa molt divertida és disparar mostres pregravades. Prova-ho:

```
sample :ambi_lunar_land
```

Sonic Pi inclou molts exemples de mostres perquè hi juguis. Pots utilitzar-les igual que utilitzes el comandament play. Per reproduir múltiples mostres i notes només les has d'escriure una darrere l'altra:

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

```
sample :ambi_drone
```

Si vols espaiar-les en el temps, utilitza el comandament sleep:

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

Fixa't que Sonic Pi no espera que acabi un so abans d'iniciar el següent. El comandament *sleep* només descriu la separació del tret dels sons. Això permet superposar fàcilment els sons creant interessants efectes de superposició.

Descobrir mostres

Hi ha dues maneres de descobrir la gamma de mostres que ofereix Sonic Pi. En primer lloc, pots utilitzar aquest sistema d'ajuda. Fes clic a *Samples* al menú de la part inferior d'aquesta pantalla d'ajuda, escull la teva categoria i trobaràs una llista de sons disponibles.

També pots utilitzar el sistema d'autocompletat. Simplement escriu l'inici d'un grup de mostres com: `sample :ambi_` i veuràs que apareix un desplegable de noms de mostres perquè les seleccionis. Prova els següents prefixos de categoria:

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

Ara comença a mesclar *samples* a les teves composicions!

3.2. Paràmetres de les mostres: Amp i Pan

Com hem vist amb els sintetitzadors, podem controlar fàcilment els nostres sons amb paràmetres. Les mostres admeten exactament el mateix mecanisme de parametrització. Tornem a visitar els nostres amics `amp:` i `pan:`

Amplificar mostres

Pots canviar l'amplitud de les mostres exactament de la mateixa manera que ho has fet amb els sintetitzadors:

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

Panoramitzar mostres

També podem utilitzar el paràmetre `pan:` amb les mostres. Per exemple, així és com tocaríem la pausa d'amén a l'orella esquerra i després a mig camí la tornaríem a tocar a través de l'orella dreta:

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

Fixa't que 0.877 és la meitat de la durada de la mostra `:loop_amen` en segons.

Finalment, tingues en compte que si estableixes alguns valors predeterminats del sintetitzador amb `use_synth_defaults` (dels quals parlarem més endavant), aquests seran ignorats pel sample.

3.3. Estirar mostres

Ara que podem tocar una varietat de sintetitzadors i mostres per crear una mica de música, és hora d'aprendre a modificar tant els sintetitzadors com les mostres per fer la música encara més única i interessant. En primer lloc, explorarem la possibilitat d'estirar i aixafar les mostres.

Representació de la mostra

Les mostres són sons pregravats que s'emmagatzemen en forma de números que representen com cal moure el con de l'altaveu per reproduir el so. El con de l'altaveu es pot moure cap a dins i cap a fora, per la qual cosa els números només han de representar com de lluny ha d'estar el con a cada moment. Per poder reproduir fidelment un so gravat, la mostra acostuma a necessitar emmagatzemar molts milers de números per segon. Sonic Pi pren aquesta llista de números i els alimenta a la velocitat adequada per moure l'altaveu del teu ordinador cap a dins i cap a fora de la manera correcta per reproduir el so. Tanmateix, també és divertit canviar la velocitat amb què s'alimenten els números a l'altaveu per canviar el so.

Canvi de freqüència

Juguem amb un dels sons ambientals: `ambi_choir`. Per reproduir-lo amb la taxa per defecte, pots passar una opció `rate:` a `sample`:

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

Aquí ho estem reproduint a la velocitat normal (1), així

que encara no hi ha res d'especial. Tanmateix, som lliures de canviar aquest número per un altre. Què tal `0,5`?

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

Uau! Què està passant aquí? Bé, dues coses. En primer lloc, la mostra triga el doble de temps a reproduir-se i, en segon lloc, el so és una octava més baixa. Explorem aquestes coses amb una mica més de detall.

Estirar (*stretch*)

Una mostra que és divertida d'estirar i comprimir és l'*Amen Break*. A un ritme normal, podríem imaginar-nos llançant-lo en una pista de *drum 'n' bass*:

```
sample :loop_amen
```

Tanmateix, canviant-ne la velocitat podem canviar de gènere. Prova amb mitja velocitat per al hip-hop old school:

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

Si l'accelerem, entrem en territori salvatge:

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

Ara, el nostre últim truc per a la festa: vegem què passa si utilitzem una rate negativa:

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

Uau! El reprodueix al revés! Ara intenta jugar amb moltes mostres diferents a diferents velocitats. Prova amb velocitats molt ràpides. Prova amb freqüències molt lentes. Mira quins sons interessants pots produir.

Una explicació senzilla de la freqüència de mostreig

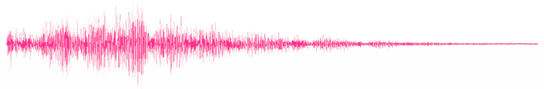
Una manera útil de pensar en les mostres és com si fossin molles. La velocitat de reproducció és com aixafar i estirar la molla. Si reprodueixes la mostra a velocitat 2, estàs aixafant la molla a la meitat de la longitud normal. Per tant, la mostra triga la meitat de temps a reproduir-se, ja que és més curta. Si reprodueixes la mostra a la meitat de la velocitat, estàs estirant la molla al doble de la longitud. Per tant, la mostra triga el doble de temps a reproduir-se, ja que és més llarga. Com més s'esclafa (taxa més alta), més s'escurça, com més s'estira (taxa més baixa), més s'allarga.

En comprimir una molla augmenta la seva densitat (el nombre de bobines per cm) això és similar al fet que la mostra soni més aguda. Estirar la molla disminueix la seva densitat i és similar al fet que el so tingui un to més baix.

Les matemàtiques darrere de la freqüència de mostreig

(Aquesta secció és per a aquells que estiguin interessats en els detalls. Sent-te lliure de saltar-te-la...)

Com hem vist anteriorment, una mostra està representada per una gran llista de números que representen on hauria de ser l'altaveu al llarg del temps. Podem prendre aquesta llista de números i utilitzar-la per dibuixar un gràfic que tindria un aspecte similar al següent:



És possible que abans hagis vist imatges com aquesta. S'anomena la forma d'ona d'una mostra. És un gràfic de números. Normalment, una forma d'ona com aquesta tindrà 44.100 punts de dades per segon (això és degut al teorema de mostreig de Nyquist-Shannon). Així que, si la mostra dura 2 segons, la forma d'ona estarà representada per 88.200 números que alimentarien l'altaveu a un ritme de 44.100 punts per segon. Per descomptat, podríem reproduir-la a doble velocitat, que seria de 88.200 punts per segon. Per tant, només trigaria 1 segon a reproduir-se. També podríem reproduir-la a mitja velocitat, cosa que suposaria 22.050 punts per segon i trigaria 4 segons en reproduir-se.

La durada de la mostra es veu afectada per la velocitat de reproducció:

- Si es duplica la velocitat de reproducció, es redueix a la meitat el temps de reproducció,
- Si es redueix a la meitat, es duplica el temps de reproducció,
- Utilitzar una taxa de reproducció d'una quarta part quadruplica el temps de reproducció,
- Utilitzar una taxa de reproducció de 1/10 fa que la reproducció duri 10 vegades més.

Podem representar-ho amb la fórmula:

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

Canviar la velocitat de reproducció també afecta el to de la mostra. La freqüència o el to d'una forma d'ona es determina per la rapidesa amb què es mou cap amunt i cap avall. D'alguna manera, el nostre cervell converteix el moviment ràpid dels altaveus en notes agudes i el moviment lent dels altaveus en notes greus. Per això, de vegades fins i tot es pot veure com es mou un altaveu de greus gran mentre bombeja greus super greus: en realitat es mou molt més lentament cap a dins i cap a fora que un altaveu que produeix notes més agudes.

Si prens una forma d'ona i l'esclafes, es mourà cap amunt i cap avall més vegades per segon. Això farà que soni més agut. Resulta que en duplicar la quantitat de moviments cap amunt i cap avall (oscil·lacions) se'n duplica la freqüència. Per tant, si reprodueixes la teva mostra al doble de velocitat, duplicaràs la freqüència amb què l'escoltes. Així mateix, si es redueix la velocitat a la meitat, la freqüència es reduirà a la meitat. Altres taxes afectaran la freqüència en conseqüència.

3.4. Mostres amb evolupant

També pots modificar la durada i l'amplitud d'una mostra mitjançant una envolupant ADSR. Tanmateix, això funciona de forma lleugerament diferent a l'envolupant ADSR que trobaràs en els sintetitzadors. Les envolupants de mostra només permeten reduir l'amplitud i la durada d'una mostra, però no augmentar-la. La mostra s'aturarà quan s'hagi acabat de reproduir o l'envolupant s'hagi completat, el que passi primer. Per tant, si utilitzes un `:release` molt llarg, no s'estendrà la durada de la mostra.

Envolupants d'*Amen*

Tornem al nostre fidel amic l'Amen Break:

```
sample :loop_amen
```

Sense *opts*, escoltem la mostra completa amb tota l'amplitud. Si la volem esvair en més d'1 segon podem fer servir el paràmetre `attack:`:

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

Per a un fade in més curt, tria un valor d'atac més curt:

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

Auto Sustain

El comportament de l'envolupant ADSR difereix del de l'envolupant del sintetitzador estàndard en el valor de *sustain*. A l'envolupant del sintetitzador estàndard, el valor de *sustain* és per defecte 0, llevat que el configuris manualment. En el cas de les mostres, el valor de sustain s'ajusta per defecte a un valor automàtic: el temps restant per reproduir la resta de la mostra. Per això sentim la mostra completa quan no passem cap valor per defecte. Si els valors d'atac, decaïment, manteniment i alliberament fossin tots 0, no sentiríem ni piu. Per tant, Sonic Pi calcula la durada de la mostra, dedueix els temps d'atac, decaïment i alliberament i utilitza el resultat com a temps de manteniment. Si els valors d'atac, decaïment i alliberament sumen més que la durada de la mostra, el manteniment simplement es posa a 0.

Fade Outs

Per explorar això, considerem la nostra ruptura d'*Amen* amb més detall. Si preguntem a Sonic Pi quant dura la mostra:

```
print sample_duration :loop_amen
```

S'imprimirà `1,753310657596372` que és la durada de la mostra en segons. Arrodonim-ho a 1,75 per comoditat. Ara, si establim l'alliberament a `0.75`, passarà una cosa sorprenent:

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

Reproduirà el primer segon de la mostra a plena amplitud abans d'esvair-se durant un període de 0,75 segons.

Aquest és el sosteniment automàtic en acció. Per defecte, l'alliberament sempre funciona des del final de la mostra. Si la nostra mostra tingués una durada de 10,75 segons, reproduiria els primers 10 segons a plena amplitud abans d'atenuar-se durant 0,75 segons.

Recorda: per defecte, `release:` s'esvaeix al final de la mostra.

Fade In i *Out*

Podem utilitzar tant `attack:` com `release:` juntament amb el comportament de sosteniment automàtic per esvair tant l'entrada com la sortida durant la durada de la mostra:

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

Com que la durada completa de la mostra és de 1,75s i les nostres fases d'atac i alliberament sumen 1,5s, el sustain s'ajusta automàticament a 0,25s. Això ens permet esvair fàcilment la mostra cap a dins i cap a fora.

Explicit sustain

Podem tornar fàcilment al nostre comportament ADSR normal del sintetitzador ajustant manualment el sustain: a un valor com a 0:

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

Ara la nostra mostra només es reprodueix durant 0,75 segons en total. Amb els valors predeterminats d'`attack:` i `decay:` en 0, la mostra salta directament a l'amplitud màxima, es manté durant 0 segons i després torna a baixar a l'amplitud 0 durant el període d'alliberament - 0,75 segons.

Plats de percussió

Podem utilitzar aquest comportament amb bons resultats per convertir les mostres que sonen més llargues en versions més curtes i percussives. Considereu la mostra `:drum_cymbal_open`:

```
sample :drum_cymbal_open
```

Pots escoltar el so del platet sonant durant un període de temps. Tot i això, podem utilitzar la nostra envolupant per fer-lo més percusiu:

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

Aleshores pots emular el cop del platet i després esmorteir-lo augmentant el període de sustain:

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

Ara comença a divertir-te posant envolupants sobre les mostres. Prova de canviar-ne també la velocitat per obtenir resultats realment interessants.

3.5. Mostres parcials

Aquesta secció conclourà la nostra exploració del reproductor de mostres de Sonic Pi. Fem una recapitulació ràpida. Fins ara hem vist com podem disparar mostres:

```
sample :loop_amen
```

A continuació, hem analitzat com podem canviar la velocitat de les mostres, per exemple, reproduint-les a mitja velocitat:

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

Després, hem vist com podem esvair una mostra (fem-ho a mitja velocitat):

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

També hem estudiat com podem utilitzar l'inici d'una mostra de forma percussiva donant a sustain: un valor explícit i establint que tant l'atac com l'alliberament siguin curts:

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

No seria bo, però, que no haguéssim de començar sempre pel principi de la mostra i que no haguéssim d'acabar sempre al final de la mostra?

Escollir un punt de partida

És possible triar un punt de partida arbitrari a la mostra com a valor entre 0 i 1, on 0 és l'inici de la mostra, 1 és el final i 0,5 és la meitat de la mostra. Intentem reproduir només la darrera meitat de la pausa d'*amen*:

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

Què tal el darrer quart de la mostra?

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

Triar un punt final

De la mateixa manera, és possible triar un punt final arbitrari a la mostra com un valor entre 0 i 1. Acabem la pausa d'*amen* a mig camí:

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

Especificar un inici i un final

Per descomptat, podem combinar aquests dos elements per reproduir segments arbitraris del fitxer d'àudio. Provem tan sols amb una petita secció al mig:

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

Què passa si triem una posició de sortida posterior a la d'arribada?

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

Genial! Ho reprodueix al revés!

Combinació amb la freqüència

Podem combinar aquesta nova capacitat de reproduir segments arbitraris d'àudio amb la nostra amiga `rate:`. Per exemple, podem reproduir una secció molt petita de la meitat de la pausa d´amen molt lentament:

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

Combinació amb envolupants

Finalment, podem combinar tot això amb les nostres envolupants ADSR per produir resultats interessants:

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

Ara pots començar a jugar a barrejar mostres amb tot aquest material...

3.6. Mostres externes

Encara que les mostres incorporades et poden ajudar a començar ràpidament, és possible que vulguis experimentar amb altres sons gravats de la teva música. Amb Sonic Pi és totalment possible fer-ho. No obstant, en primer lloc, tindrem una ràpida discussió sobre la portabilitat de la teva producció.

Portabilitat

Si composes la teva peça únicament amb sintetitzadors i mostres incorporades, el codi és tot el que necessites per reproduir fidelment la teva música. Pensa-ho per un moment: és increïble! Un simple tros de text que pots enviar per correu electrònic o enganxar a un Gist representa tot el que necessites per reproduir els teus sons. Això fa que sigui molt fàcil de compartir amb els teus amics, ja que només s'han de familiaritzar amb el codi.

En canvi, si comences a utilitzar les teves pròpies mostres pregravades, perds aquesta portabilitat. Això és perquè, per reproduir la teva música, altres persones no només necessiten el teu codi, sinó també les teves mostres. Això limita la capacitat dels altres per manipular, mesclar i experimentar amb la teva feina. Per descomptat, això no t'hauria d'impedir utilitzar les teves pròpies mostres, només cal tenir-ho en compte.

Mostres locals

Aleshores, com pots reproduir qualsevol arxiu WAV, AIFF, OGG, OGA o FLAC arbitrari al teu ordinador? Tot el que has de fer és passar la ruta d'aquest fitxer a `sample:`

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

# Windows
sample "C:/Users/sam/Desktop/my-sound.wav"
```

Sonic Pi carregarà i reproduirà automàticament la mostra. També pots introduir tots els paràmetres estàndard que estàs acostumat a passar a 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. Paquets de mostres

Nota: aquesta secció del tutorial aborda el fet de treballar amb grans directoris de les vostres pròpies mostres. Aquest serà el teu cas si has descarregat o comprat els teus propis paquets de mostres i vols utilitzar-los amb Sonic Pi.

Sent-te lliure d'ometre aquest apartat si ets feliç treballant amb les mostres incorporades.

Quan es treballa amb grans carpetes de mostres externes pot ser molest haver d'escriure tota la ruta cada vegada que es dispara una mostra individual.

Per exemple, diguem que tens la següent carpeta al teu ordinador:

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

A dins d'aquesta carpeta hi trobem les mostres següents:

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

Normalment, per reproduir la mostra del piano podem utilitzar la ruta completa:

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

Si volem després reproduir la mostra de la guitarra podem fer servir també la seva ruta completa:

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

Tot i això, ambdues crides a la mostra requereixen que coneguem els noms de les mostres dins del nostre directori. I si només volem escoltar ràpidament cadascuna de les mostres?

Indexació de paquets de mostres

Si volem reproduir la primera mostra d'un directori només hem de passar el nom del directori a `sample` i l'índex `0` com a continuació:

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

Fins i tot podem fer un accés directe a la ruta del nostre directori fent servir una variable:

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

Ara, si volem reproduir la segona mostra del nostre directori, només cal afegir 1 al nostre índex:

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

Observa que ja no necessitem conèixer els noms de les mostres al directori - només necessitem conèixer el directori mateix (o tenir-hi un accés directe). Si demanem un índex més gran que el nombre de mostres, simplement s'arrodoneix com a Rings (anells). Per tant, sigui quin sigui el número que utilitzem, tenim garantit que obtindrem una de les mostres del directori.

Filtrar paquets de mostres

Normalment la indexació és suficient, però de vegades necessitem més potència per classificar i organitzar les nostres mostres. Per sort, molts paquets de mostres afegeixen informació útil als noms dels fitxers. Fem una altra ullada als noms dels arxius de mostra del nostre directori:

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

Observa que en aquests noms de fitxer hi tenim força informació. En primer lloc, tenim els BPM de la mostra (pulsacions per minut) al principi. Així, la mostra de piano és a 120 BPM i les nostres tres primeres melodies són a 100 BPM. A més, els noms de les nostres mostres contenen la tonalitat. Així, la mostra de guitarra és a Sib i les melodies a La#. Aquesta informació és molt útil per mesclar aquestes mostres amb el nostre altre codi. Per exemple, sabem que només podem tocar la mostra de piano amb un codi que estigui a 120 BPM i a la tonalitat de Sib.

Resulta que podem utilitzar aquesta particular convenció de nomenclatura dels nostres conjunts de mostres al codi per ajudar-nos a filtrar les que volem. Per exemple, si estem treballant a 120 BPM, podem filtrar totes les mostres que continguin la cadena "`120`" amb el següent:

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

Això ens reproduirà el primer match. Si volem el segon match només hem de fer servir l'índex:

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

Fins i tot podem utilitzar diversos filtres alhora. Per exemple, si volem una mostra el nom de fitxer de la qual contingui les subcadenes "120" i "A#" podem trobar-la fàcilment amb el codi següent:

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

Finalment, seguim sent lliures d'afegir les nostres opcions habituals quan disparem la sample:

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

Fonts

El sistema de precàrrega de filtres de mostres entén dos tipus d'informació: sources (fonts) i filters (filtres). Les fonts són la informació utilitzada per crear la llista de possibles candidats. Una font pot adoptar dues formes:

1. "/path/to/samples" - una cadena que representa una ruta vàlida a un directori
2. "/path/to/samples/foo.wav" - una cadena que representa una ruta vàlida a una mostra

La fn de `sample` reunirà primer totes les fonts i les utilitzarà per crear una gran llista de candidats. Aquesta llista es construeix afegint primer totes les rutes vàlides i després afegint tots els fitxers `.flac`, `.aif`, `.aiff`, `.wav`, `.wave` vàlids que continguin els directoris.

Per exemple, observa el codi següent:

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

sample samps, samps2, path, 0
```

Aquí estem combinant el contingut de les mostres dins de dos directoris i afegint una mostra específica. Si `"/path/to/my/samples/"` conté 3 mostres i `"/path/to/my/samples2/"` en conté 12, tindríem 16 potencials mostres per indexar i filtrar (3 + 12 + 1).
Per defecte, només els fitxers de mostres dins d'un directori es reuneixen a la llista de candidats. De vegades pots tenir diverses carpetes imbricades de mostres en què vols cercar i filtrar. En aquest cas, pots fer una cerca recursiva de totes les mostres dins de totes les subcarpetes d'una carpeta concreta afegint `**` al final de la ruta:

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

Ves en compte, ja que la cerca en un conjunt molt gran de carpetes pot portar molt de temps. Tanmateix, el contingut de totes les fonts de carpetes s'emmagatzema en

memòria cau, de manera que el retard només es produirà la primera vegada.

Finalment, tingues en compte que les fonts han d'anar primer. Si no hi ha cap font, se seleccionarà el conjunt de mostres incorporades com a llista per defecte de candidats per treballar.

Filtres

Quan tinguis una llista de candidats, pots utilitzar els següents tipus de filtre per reduir encara més la selecció:

- `"foo"` - Les cadenes filtraran per l'aparició de subcadenes al nom del fitxer (menys la ruta del directori i l'extensió).
- `/fo[oO]/` - Les expressions regulars filtraran per la coincidència de patrons al nom del fitxer (menys la ruta del directori i l'extensió).
- `:foo` - Les paraules clau filtraran els candidats en funció de si la paraula clau coincideix directament amb el nom del fitxer (menys la ruta del directori i l'extensió).
- `lambda{|a| ... }` - Els procs amb un argument seran tractats com una funció de filtre o generador de candidats. T'apareixerà la llista de candidats actuals i hauràs de tornar una nova llista de candidats (una llista de rutes vàlides a fitxers de mostra).
- `1` - Els números seleccionaran el candidat amb aquest índex (embolicant-los com un anell si cal).

Per exemple, podem filtrar totes les mostres d'un directori que contingui la cadena `"foo"` i reproduir la primera mostra que coincideixi a mitja velocitat:

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

Consulta l'ajuda de `sample` per veure'n exemples detallats. Tingues en compte que es respecta l'ordre dels filtres.

Compostos

Finalment, pots utilitzar les llistes sempre que puguis col·locar una font o un filtre. La llista s'aplanarà automàticament i el seu contingut es tractarà com a fonts i filtres normals. Per tant, les següents crides a sample són semànticament equivalents:

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

Conclusió

Aquesta ha sigut una secció avançada per a persones que necessiten poder real per manipular i utilitzar paquets de mostres. Si no has entès la major part d'aquesta secció, no et preocupis. És probable que encara no necessitis cap d'aquestes funcionalitats. Tot i així, sabràs quan la necessites i ho podràs tornar a llegir quan comencis a treballar amb grans directoris de mostres.

4. ALEATORIETAT

Una bona manera d'afegir una mica dinterès a la teva música és utilitzar alguns números aleatoris. Sonic Pi té una gran funcionalitat per afegir aleatorietat a la teva música, però abans de començar hem de saber una veritat impactant: a Sonic Pi l'aleatorietat no és realment aleatòria. Què vol dir això? Bé, vegem.

Repetibilitat

Una funció aleatòria realment útil és `rrand`, que et donarà un valor aleatori entre dos números - un mínim i un màxim. (`rrand` és l'abreviatura de ranged random). Intentem tocar una nota aleatòria:

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

Ooh, ha sonat una nota a l'atzar. Ha sonat la nota `83.7527`. Una bonica nota aleatòria entre 50 i 95. Uau, espera, acabo de predir la nota aleatòria exacta que tens tu també? Alguna cosa estranya està passant aquí. Intenta executar el codi de nou. Què? Ha tornat a sortir `83.7527` de nou? Això no pot ser aleatori!

La resposta és que no és veritablement aleatori, és pseudo-aleatori. Sonic Pi et donarà números semblants als aleatoris de manera repetible. Això és molt útil per assegurar que la música que creis a la teva màquina soni idèntica a la dels altres, fins i tot si utilitzes una mica d'aleatorietat a la teva composició.

Per descomptat, en una peça musical determinada, si seleccionés "a l'atzar" `83,7527` cada cop, no seria gaire interessant. Però no és així. Prova el següent:

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

Sí, per fi sona aleatori. Dins d'una determinada execució, les crides següents a les funcions aleatòries tornaran valors aleatoris. Tot i això, la següent execució produirà exactament la mateixa seqüència de valors aleatoris i sonarà exactament igual. És com si tot el codi de Sonic Pi retrocedís en el temps fins exactament el mateix punt cada cop que es prem el botó Run. És el Dia de la Marmota de la síntesi musical!

Campanes embruixades

Una bonica il·lustració de l'aleatorietat en acció és l'exemple de les campanes embruixades, que fa un bucle de la mostra `:perc_bell` amb una velocitat aleatòria i un temps de repòs entre els sons de les campanes:

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

Tall aleatori

Un altre exemple divertit d'aleatorització és modificar el tall (cutoff) d'un sintetitzador aleatori. Un bon sintetitzador per provar-ho és l'emulador `:tb303`:

```
use_synth :tb303

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

Llavors aleatòries *(Random seeds)*

Aleshores, què passa si no t'agrada aquesta seqüència particular de números aleatoris que Sonic Pi proporciona? Bé, és totalment possible triar un punt de partida diferent mitjançant `use_random_seed`. La llavor per defecte és 0, així que escull una llavor diferent per a una experiència aleatòria diferent.

Considera el següent:

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

Cada cop que executis aquest codi, escoltaràs la mateixa seqüència de 5 notes. Per obtenir una seqüència diferent simplement canvia'n la llavor:

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

Això produirà una seqüència diferent de 5 notes. Canviant-ne la llavor i escoltant-ne els resultats pots trobar alguna cosa que t'agradi – i quan ho comparteixis amb altres, ells també escoltaran exactament el mateix que tu.

Vegem altres funcions aleatòries útils.

Choose

Quelcom molt comú és triar un element a l'atzar d'una llista d'elements coneguts. Per exemple, puc voler tocar una nota de les següents: 60, 65 o 72. Puc aconseguir-ho amb `choose`, que em permet triar un element d'una llista. Primer, necessito posar els meus números en una llista, cosa que es fa introduint-los entre claudàtors i separant-los amb comes: `[60, 65, 72]`. A continuació, només els he de passar a `choose`:

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

Escoltem com sona:

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

rrand

Ja hem vist `rrand`, però ho repassarem de nou. Aquest retorna un nombre aleatori entre dos valors exclusivament. Això vol dir que mai tornarà ni el nombre superior ni l'inferior - sempre alguna cosa entre ambdós. El nombre sempre serà un real - el que significa que no és un número sencer sinó una fracció d'un nombre. Exemples de valors reals retornats per `rrand(20, 110)`:

- 87.5054931640625
- 86.05255126953125
- 61.77825927734375

rrand_i

Ocasionalment voldràs un nombre aleatori sencer, no pas un número real. Aquí és on `rrand_i` ve al rescat. Funciona de forma similar a `rrand`, excepte que pot retornar els valors mínim i màxim com a possibles valors aleatoris (cosa que significa que és inclusiu en lloc d'exclusiu del rang). Exemples de números retornats per `rrand_i(20, 110)` són:

- 88
- 86
- 62

rand

Això retornarà un nombre real aleatori entre 0 (inclòs) i el valor màxim que especifiquis (exclusiu). Per defecte tornarà un valor entre 0 i un. Per tant, és útil per triar valors aleatoris d'`amp:`:

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

rand_i

De forma similar a la relació entre `rrand_i` i `rrand`, `rand_i` retornarà un nombre sencer aleatori entre 0 i el valor màxim que especifiquis.

dice

De vegades es vol emular un llançament de daus – aquest és un cas especial de `rrand_i` on el valor més baix és sempre 1. Una crida a `dice` requereix que s'especifiqui el nombre de cares del dau. Un dau estàndard té 6 cares, per la qual cosa `dice(6)` actuarà de forma molt similar, retornant valors de 1, 2, 3, 4, 5 o 6. No obstant això, igual que en els jocs de rol de fantasia, pots trobar valor en un dau de 4 cares, o en un de 12, o en un de 20, potser fins i tot en un de 120!

one_in

Finalment, és possible que vulguis emular el llançament de la puntuació més alta d'un dau, com un 6 en un dau estàndard. `one_in`, per tant, retorna veritable amb una probabilitat d'un al nombre de cares del dau. Per tant `one_in(6)` retornarà veritable amb una probabilitat de 1 a 6 o fals en cas contrari. Els valors veritables i falsos són molt útils per a les sentències `if`, que veurem en una secció posterior d'aquest tutorial.

Ara, prova de manipular el teu codi amb una mica d'aleatorietat!

5. ESTRUCTURES DE PROGRAMACIÓ

Ara que has après els fonaments de la creació de sons amb `play` i `sample` i de la creació de melodies i ritmes senzills afegint latència (`sleep`) entre sons, potser et preguntaràs què més et pot oferir el món del codi...

Doncs bé, t'espera un regal emocionant! Resulta que les estructures bàsiques de programació, com els bucles (looping), els condicionals (conditionals), les funcions (functions) i els fils (threads), ens ofereixen eines increïblement potents per expressar les nostres idees musicals.

Comencem amb els fonaments...

5.1. Blocs

Una estructura que veuràs molt a Sonic Pi és el bloc. Els blocs ens permeten fer coses útils amb grans fragments de codi. Per exemple, amb els paràmetres del sintetitzador i de la mostra només vam poder modificar el què passava en una sola línia. De vegades, però, volem fer alguna cosa significativa amb un nombre superior de línies de codi. Per exemple, podem voler fer un bucle, afegir-hi reverberació, executar-lo només 1 de cada 5 vegades, etc. Considera el codi següent:

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

Per treballar un fragment de codi, necessitem indicar a Sonic Pi on comença i on acaba el bloc de codi. Fem servir `do` per començar i `end` per acabar. Per exemple:

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

Però si només fas això no funcionarà (intenta-ho i obtindràs un error) ja que no li hem dit a Sonic Pi el que volem fer amb aquest bloc do/end. Ho diem a Sonic Pi escrivint un codi especial abans del do. Veurem alguns d'aquests caràcter de codi especials més endavant en aquest tutorial. Ara per ara, és important saber que col·locar el teu codi entre `do` i `end` indica a Sonic Pi que vols fer alguna cosa especial amb aquest tros de codi.

5.2. Iteració i bucles

Fins ara hem dedicat molt de temps a veure els diferents sons que es poden fer amb els blocs de `play` i `sample`. També hem après a activar aquests sons a través del temps fent servir `sleep`.

Com ja hauràs comprovat, et pots divertir molt amb aquests blocs de construcció bàsics. Tot i així, tota una nova dimensió de diversió s'obre quan comences a utilitzar el poder del codi per estructurar la teva música i les teves composicions. En les properes seccions explorarem algunes d'aquestes noves i poderoses eines. En primer lloc, la iteració i els bucles.

Repetició

Has escrit algun codi que t'agradaria repetir diverses vegades? Per exemple, pots tenir una cosa així:

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

I si volguessim repetir-ho 3 vegades? Bé, podríem simplement copiar-ho i enganxar-ho tres vegades:

```
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
```

Això sí que és un munt de codi! Què passaria si volguessis canviar la mostra a `:elec_plip`? Hauries de trobar tots els llocs amb el `:elec_blup` original i canviar-los. I el que és més important, què passaria si volguessis repetir el tros de codi original 50 cops o 1000? Això sí que seria un munt de codi, i un munt de línies de codi a alterar si volguessis fer un canvi.

Iteració

De fet, repetir el codi hauria de ser tan fàcil com dir "fes-ho tres vegades". Bé, més o menys ho és. Recordes el nostre vell amic el bloc de codi? Podem fer-lo servir per marcar l'inici i el final del codi que volem repetir tres vegades. Aleshores fem servir el codi especial `3.times`. Així, en lloc d'escriure "fes-ho tres vegades", escrivim `3.times do` - no és gaire difícil. Només recorda escriure `end` al final del codi que vols repetir:

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

Això no és molt més net que tallar i enganxar? Podem utilitzar-ho per crear un munt de boniques estructures repetitives:

```
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
```

Iteracions de nidificació

Podem posar iteracions dins d'altres iteracions per crear patrons interessants. Per exemple:

```
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
```

Fer bucles *(looping)*

Si vols que alguna cosa es repeteixi moltes vegades, potser et trobaràs utilitzant números realment grans, com ara `1000.times do`. En aquest cas, probablement és millor demanar a Sonic Pi que es repeteixi per sempre (almenys fins que premis el botó de stop!). Fem un bucle d'amen break per sempre:

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

L'important és saber que els bucles actuen com a forats negres per al codi. Quan el codi entra en un bucle, mai no en pot sortir fins que prems stop - simplement donarà voltes i voltes al bucle per sempre. Això vol dir que si tens codi després del bucle mai no l'escoltaràs. Per exemple, el platet després d'aquest bucle no sonarà mai:

```
loop do
  play 50
  sleep 1
end

sample :drum_cymbal_open
```

Ara ja pots començar a estructurar el teu codi amb iteracions i bucles.

5.3 Condicionals

Una cosa que probablement voldràs fer és no només tocar una nota a l'atzar (ves a la secció anterior sobre l'aleatorietat), sinó també prendre una decisió aleatòria i sobre la base del resultat executar algun codi o un altre. Per exemple, podries voler tocar aleatòriament un tambor o un platet. Això ho podem aconseguir amb una sentència `if`.

Llançar una moneda

Així que llancem una moneda: si surt cara, toca un tambor, si surt creu, toca un platet. És fàcil. Podem emular el llançament d'una moneda amb la nostra funció `one_in` (introduïda en la secció sobre aleatorietat) especificant una probabilitat d'1 entre 2: `one_in(2)`. Aleshores podem utilitzar-ne el resultat per decidir entre dues peces de codi, el codi per tocar el tambor i el codi per tocar el platet:

```
loop do

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

  sleep 0.5

end
```

Observa que les declaracions `if` tenen tres parts:

- La pregunta a fer
- La primera opció de codi a executar (si la resposta a la pregunta és sí)
- La segona opció de codi a executar (si la resposta a la pregunta és no)

Normalment, en els llenguatges de programació, la noció de `if` està representada pel terme `true` i la noció no està representada pel terme `false`. Així que hem de trobar una pregunta que ens doni una resposta `true` o `false`, que és exactament el que fa `one_in`.

Fixa't que la primera opció està sitada entre l'`if` i l'`else` i la segona opció entre l'`else` i l'`end`. Igual que els blocs do/end pots posar diverses línies de codi en qualsevol dels dos llocs. Per exemple:

```
loop do

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

end
```

Aquest cop "dormirem" una quantitat de temps diferent depenent de l'elecció que fem.

Simple if

De vegades voldrem executar opcionalment una sola línia de codi. Això ho podem fer col·locant l'`if` i després la pregunta al final. Per exemple:

```
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
```

Això farà que es reprodueixin acords de diferents números amb la possibilitat que cada nota soni amb una probabilitat diferent.

5.4 Fils (*Threads*)

Així que has creat la teva línia de baix assassina i un ritme genial. Com els pots tocar alhora? Una solució és entrellaçar-los manualment: tocar una mica de baix, després una mica de bateria, després més baix... Tot i això, aviat es fa difícil pensar en la sincronització, especialment quan comences a entrellaçar més elements.

I si Sonic Pi pogués entrellaçar les coses per tu automàticament? Bé, pot, i ho fa amb una cosa especial anomenada fil (thread).

Bucles infinits

Perquè aquest exemple sigui senzill, t'has d'imaginar que es tracta d'un ritme fort i una línia de baix assassina:

```
loop do
  sample :drum_heavy_kick
  sleep 1
end

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

Com hem comentat anteriorment, els bucles (loops) són com forats negres per al programa. Quan entres en un bucle mai no en pots sortir fins que prems stop. Com reproduïm tots dos bucles alhora? Hem de dir-li a Sonic Pi que volem iniciar alguna cosa alhora que la resta del codi. Aquí és on el fils (threads) vénen al rescat.

Fils al rescat

```
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
```

Quan emboliquem el primer bucle en un bloc `in_thread` do/end estem diem a Sonic Pi que executi el contingut del bloc do/end exactament al mateix temps que la següent sentència després del bloc do/end (que resulta ser el segon bucle). Prova-ho i escoltaràs com s'entrellacen la bateria i la línia de baix.

Ara, què passaria si volguéssim afegir un sintetitzador a sobre? Una cosa així:

```
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
```

Ara tenim el mateix problema que abans. El primer bucle es reprodueix al mateix temps que el segon a causa de l' `in_thread`. No obstant això, el tercer bucle mai no s'assoleix. Per tant, necessitem un altre fil:

```
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
```

Les execucions (*runs*) són fils

El que et pot sorprendre és que quan prems el botó Run, en realitat estàs creant un nou fil perquè s'executi el codi. Per això, si el prems diverses vegades, els sons se superposaran. Com que les execucions en si mateixes són fils, automàticament entrellaçaran els sons per tu.

Abast (*Scope*)

A mesura que aprenguis a dominar Sonic Pi, entendràs que els fils són els blocs de construcció més importants per a la teva música. Una de les coses importants que fan és aïllar la noció de *current settings* d'altres fils. Què vol dir això? Bé, quan canvies de sintetitzador mitjançant `use_synth` en realitat només estàs canviant el sintetitzador del fil actual - els altres fils seguiran amb el sinte original. Observem-ho en acció:

```
play 50
sleep 1

in_thread do
  use_synth :tb303
  play 50
end

sleep 1
play 50
```


Notes com el so del mig era diferent dels altres? La declaració `use_synth` només ha afectat al fil al qual fa referència i no al fil d'execució principal extern.

Herència

Quan es crea un fil nou amb `in_thread`, el fil nou heretarà automàticament totes les configuracions actuals del fil actual. Vegem-ho:

```
use_synth :tb303
play 50
sleep 1
in_thread do
  play 55
end
```

Veus com la segona nota es reprodueix amb el sintetitzador `:tb303` encara que s'hagi reproduït des d'un fil independent? Qualsevol dels ajustaments modificats amb les diverses funcions `use_*` es comportarà de la mateixa manera.

Quan es creen fils, aquests hereten tots els ajustaments del seu pare però no comparteixen cap canvi.

Posar nom als fils

Finalment, podem posar noms als nostres fils:

```
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
```

Mira el tauler de registre quan executes aquest codi. Veus com el registre informa del nom del fil amb el missatge?

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

Només es permet un fil per a cada nom

Una darrera cosa que cal saber sobre els fils amb nom és que només no podem tenir més d'un fil amb el mateix nom funcionant alhora. Explorem-ho. Considera el codi següent:

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

Enganxa-ho en un buffer i prem el botó Run. Torna'l a prémer un parell de vegades. Escolta la cacofonia de les múltiples pauses d'amen que es desajusten entre si. Bé, ara pots prémer Stop.

Aquest és el comportament que hem vist una vegada i una altra: si prems el botó Run, el so se superposa a qualsevol so existent. Per tant, si tens un bucle i prems el botó Run tres vegades, tindràs tres capes de bucles reproduint-se simultàniament.

Tot i això, amb els fils amb nom és diferent:

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

Intenta prémer el botó Run diverses vegades amb aquest codi. Només sentiràs un bucle de ruptura d'amen. També apareixerà aquest missatge:

```
⟹ Skipping thread creation: thread with name :amen
already exists.
```

Sonic Pi li està dient que un fil amb el nom `:amen` ja està sonant, per la qual cosa no n'està creant un altre.

Aquest comportament pot semblar-te poc rellevant ara per ara - però serà molt útil quan comencem a programar codi en viu.

5.5. Funcions

Quan comencis a escriure molt de codi, és possible que vulguis trobar una forma d'organitzar i estructurar les coses per fer-les més ordenades i fàcils d'entendre. Les funcions són una manera molt poderosa de fer-ho. Ens donen la possibilitat de posar nom a un munt de codi. Fem-hi una ullada.

Definint funcions

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

Aquí, hem definit una nova funció anomenada `foo`. Ho hem fet amb el nostre vell amic el bloc do/end i la paraula màgica `define` seguida del nom que volem donar a la nostra funció. No cal que es digui `foo`, la podríem haver anomenat `bar`, `baz` o qualsevol nom significatiu per a tu com `main_section` o `lead_riff`.

Recorda posar dos punts : abans del nom de la teva funció quan la defineixis.

Funcions de crida

Un cop definida la nostra funció podem anomenar-la només escrivint-ne el nom:

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

foo
```

```
sleep 1

2.times do
  foo
end
```

Fins i tot podem utilitzar `foo` dins dels blocs d'iteració o a qualsevol lloc on haguem escrit `play` o `sample`. Això ens dóna una gran possibilitat d'expressar-nos i de crear noves paraules amb sentit per utilitzar-les a les nostres composicions.

Les funcions es recorden a totes les execucions

Fins ara, cada cop que has premut el botó Run, Sonic Pi ha començat des d'una pissarra completament en blanc. Només sap el que hi ha al buffer. No pots referenciar codi en un altre buffer o en un altre fil. Tanmateix, amb les funcions podem canviar-ho. Quan defineixes una funció, Sonic Pi la recorda. Ho provarem. Esborra tot el codi del teu buffer i reemplaça-ho amb:

```
foo
```

Fes clic al botó Run i escolta com es reprodueix la teva funció. On ha anat el codi? Com sabia Sonic Pi què havia de reproduir? Sonic Pi simplement ha recordat la seva funció - fins i tot després d'esborrar-la del buffer. Aquest comportament només funciona amb les funcions creades usant `define` (i `defonce`).

Funcions parametritzades

Potser t'interessa saber que, igual que pots passar valors mínims i màxims a `rrand`, pots ensenyar a les teves funcions a acceptar arguments. Fem-hi una ullada:

```
define :my_player do |n|
  play n
end

my_player 80
sleep 0.5
my_player 90
```

Això no és gaire emocionant, però il·lustra la qüestió. Hem creat la nostra pròpia versió de `play` anomenada `my_player` que està parametritzada.

Els paràmetres han d'anar després del `do` del bloc `define` do/end, entre barres verticals | i separats per comes,. Pots utilitzar les paraules que vulguis per als noms dels paràmetres.

La màgia ocorre dins del bloc `define` do/end. Pots utilitzar els noms dels paràmetres com si fossin valors reals. En aquest exemple estic tocant la nota `n`. Pots considerar els paràmetres com una mena de promesa que quan el codi s'executi, seran reemplaçats per valors reals. Això ho aconsegueixes passant un paràmetre a la funció quan la crides. Jo faig això amb el `my_player 80` per tocar la nota 80. Dins la definició de la funció, `n` se substitueix ara per 80, per la qual cosa `tocar n` es converteix a `tocar 80`. Quan la crido de nou amb el `my_player 90`, `n` se substitueix ara per 90, de manera que `play n` es converteix en `play 90`.
Vegem-ne un exemple més interessant:

```
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
```

Aquí he utilitzat `repeats` com si fos un número a la línia `repeats.times do`. També he utilitzat `root` com si fos un nom de nota a la meva crida a `play`.

Mira com som capaços d'escriure una cosa molt expressiva i fàcil de llegir traslladant gran part de la nostra lògica a una funció.

5.6. Variables

Una cosa útil a fer en el teu codi és crear noms per a les coses. Sonic Pi ho fa molt fàcil: escrius el nom que vols fer servir, un signe igual (=), i després la cosa que vols recordar:

```
sample_name = :loop_amen
```

Aquí, hem 'recordat' el símbol `:loop_amen` a la variable `sample_name`. Ara podem utilitzar `sample_name` a tots els llocs on podríem haver utilitzat :loop_amen. Per exemple:

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

Hi ha tres raons principals per utilitzar variables a Sonic Pi: comunicar el significat, gestionar la repetició i capturar els resultats de les coses.

Comunicar el significat

Quan escrius codi, és fàcil pensar que estàs dient a l'ordinador com fer les coses, sempre que l'ordinador ho entengui. No obstant això, cal recordar que no només l'ordinador llegeix el codi. Altres persones també poden llegir-lo i intentar entendre el que passa. A més, és probable que tu mateix llegeixis el teu propi codi en el futur i tractis d'entendre què està passant. Encara que ara et sembli obvi, potser no ho sigui tant per als altres o fins i tot per al teu futur jo.

Una manera d'ajudar els altres a entendre el que està fent el teu codi és escriure comentaris (com vam veure en una secció anterior). Una altra és fer servir noms de variables amb sentit. Mira aquest codi:

```
sleep 1.7533
```

Per què utilitza el número `1,7533`? D'on procedeix aquest número? Què vol dir? Ara, mira aquest codi:

```
loop_amen_duration = 1.7533
sleep loop_amen_duration
```

Ara és molt més clar el que significa `1,7533`: és la durada

de la mostra `:loop_amen`! Per descomptat, podries preguntar-te que per què no escriure simplement

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

Això, per descomptat, és una forma molt bonica de comunicar la intenció del codi.

Gestionar la repetició

Sovint hi ha molta repetició al codi i quan es volen canviar coses, cal fer-ho a molts llocs. Fes una ullada a aquest codi:

```
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)
```

Fem moltes coses amb `:loop_amen`! I si volguéssim escoltar com sona amb una altra mostra de bucle com `:loop_garzul`? Hauríem de trobar i substituir tots els `:loop_amen` per `:loop_garzul`. Això pot estar bé si tens molt de temps, però què passa si estàs actuant a l'escenari? De vegades no et pots permetre el luxe de tenir temps, sobretot si vols que la gent segueixi ballant.

I si haguessis escrit el teu codi així?

```
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)
```

Ara, això fa exactament el mateix que l'anterior (prova-ho). També ens dóna la capacitat de només canviar una línia `sample_name = :loop_amen` a `sample_name = :loop_garzul` i ho canviem a molts llocs a través de la màgia de les variables.

Capturar resultats

Per últim, una bona motivació per utilitzar variables és capturar els resultats de les coses. Per exemple, és possible que vulguis fer coses amb la durada d'una mostra:

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

Ara podem utilitzar `sd` a qualsevol lloc on necessitem la durada de la mostra de `:loop_amen`.

Potser més important, una variable ens permet capturar el resultat d'una crida a `play` o `sample`:

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

Ara hem agafat i recordat `s` com una variable, cosa que ens permet controlar el sintetitzador mentre s'està executant:

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

Veurem el control dels sintetitzadors amb més detall a una secció posterior.

Alerta: Variables i fils

Mentre que les variables són genials per donar noms a les coses i capturar els resultats de les coses, és important saber que normalment només han de ser utilitzades localment dins un fil. Per exemple, no facis això:

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

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

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


A l'exemple anterior assignem un anell de números `a` una variable a i després l'utilitzem dins de dos bucles en directe (`live_loops`) separats. Al primer live_loop cada `0.5`s ordenem l'anell (a `(ring 1, 2, 3, 4, 5, 6)`) i després l'imprimim al registre. Si executes el codi, veuràs que la llista impresa no sempre està ordenada. Això et pot sorprendre, sobretot perquè de vegades la llista s'imprimeix ordenada i altres vegades no. Això s'anomena comportament no determinista i és el resultat d'un problema força desagradable anomenat condició de carrera. El problema es deu al fet que el segon live_loop també està manipulant la llista (en aquest cas mesclant-la) i pel moment en què s'imprimeix la llista, de vegades ha estat ordenada i de vegades ha estat barallada. Tots dos live_loop estan competint per fer alguna cosa diferent a la mateixa variable i cada vegada un bucle diferent "guanya".

Hi ha dues solucions per això. En primer lloc, no utilitzis la mateixa variable en múltiples bucles vius o fils. Per exemple, el codi següent sempre imprimirà una llista ordenada ja que cada live_loop té la seva pròpia variable:

```
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
```

De vegades, però, volem compartir coses entre fils. Per exemple, la clau actual, el BPM, el sintetitzador, etc. En aquests casos, la solució és fer servir el sistema d'estat especial de Sonic Pi a través dels fns `get` i `set`. Ho discutirem més endavant a la secció 10.

5.7. Sincronització de fils

Quan hagis avançat prou en la codificació en viu amb un nombre de funcions i fils simultàniament, probablement hauràs notat que és força fàcil cometre un error en un dels fils que el mata. Això no és un gran problema, perquè pots reiniciar fàcilment el fil fent clic a *Run*. No obstant això, quan reiniciis el fil ara està fora de temps amb els fils originals.

Temps heretat

Com hem comentat anteriorment, els nous fils creats amb `in_thread` hereten tots els ajustaments del fil pare. Això inclou l'hora actual. Això significa que els fils sempre estan en temps els uns amb els altres quan s'inicien simultàniament.

Tot i això, quan s'inicia un fil per si mateix, s'inicia amb la seva pròpia hora, que és poc probable que estigui sincronitzada amb qualsevol dels altres fils que s'estan executant actualment.

Cue i *Sync*

Sonic Pi proporciona una solució a aquest problema amb les funcions `cue` i `sync`.

`cue` ens permet enviar missatges de batec (*beat*) a tots els altres fils. Per defecte, els altres fils no hi estan interessats i ignoren aquests missatges de batec. Tanmateix, es pot registrar fàcilment l'interès amb la funció `sync`.

El més important és tenir en compte que `sync` és similar a `sleep` en el sentit que fa que el fil actual deixi de fer qualsevol cosa durant un període de temps. No obstant això, amb `sleep` s'especifica quant de temps es vol esperar, mentre que amb `sync` no se sap quant de temps s'esperarà, ja que aquest espera el següent senyal d'un altre fil, que pot ser aviat o trigar molta estona.

Explorem-ho amb una mica més de detall:

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

in_thread do
  loop do
    sync :tick
```

```
    sample :drum_heavy_kick
  end
end
```

Aquí tenim dos fils - un actuant com un metrònom, no reproduint cap so però enviant missatges de batec `:tick` a cada pulsació. El segon fil se sincronitza amb els missatges `tick` i quan en rep un hereta el temps del fil `cue` i segueix funcionant.

Com a resultat, sentirem la mostra de `:drum_heavy_kick` exactament quan l'altre fil enviï el missatge `:tick`, encara que els dos fils no hagin començat la seva execució alhora:

```
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
```

Aquesta entremaliada crida a `sleep` faria que el segon fil es desfasés amb el primer. No obstant això, com que estem usant `cue` i `sync`, sincronitzem automàticament els fils evitant qualsevol desfasament accidental.

Noms de Cue

Pots utilitzar el nom que vulguis per als teus missatges de `cue`- no només `:tick`. Només has d'assegurar-te que qualsevol altre fil se sincronitzi amb el nom correcte - en cas contrari, estaran esperant per sempre (o almenys fins que premis el botó de *Stop*).

Juguem amb uns quants noms de cue:

```
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
```

Aquí tenim un bucle `cue` principal que envia aleatòriament un dels noms de batecs `:foo`, `:bar` o `:baz`. També tenim tres bucles que se sincronitzen amb cadascun d'aquests noms de manera independent i que reprodueixen una mostra diferent. L'efecte net és que sentim un so cada 0,5 batecs mentre cada un dels `fils de sincronització` se sincronitza aleatòriament amb el fil de `cue` i en reprodueix la mostra.

Això, per descomptat, també funciona si ordenes els fils al revés, ja que els fils de `sync` simplement s'asseuran i esperaran el següent `cue`.

6. EFECTES (FX)

Un dels aspectes més gratificants i divertits de Sonic Pi és la possibilitat d'afegir fàcilment efectes (FX per abreujar) d'estudi als teus sons. Per exemple, és possible que vulguis afegir una mica de reverberació a parts de la teva peça, o una mica de ressò o potser fins i tot distorsionar o balancejar les teves línies de baix.

Sonic Pi ofereix una forma molt senzilla però potent d'afegir efectes. Fins i tot et permet encadenar-los (perquè puguis passar els teus sons per la distorsió, després pel ressò i després per la reverberació) i també controlar cada unitat individual de FX amb opts (de manera similar a donar paràmetres als sintetitzadors i mostres). Fins i tot pots modificar les opcions dels FX mentre s'estan executant. Així, per exemple, pots augmentar la reverberació del teu baix al llarg de la pista...

Pedals de guitarra

Si tot això et sona una mica complicat, no et preocupis. Quan hi hagis jugat una mica, tot s'aclarirà. Abans, però, una analogia senzilla és la dels pedals d'efectes de la guitarra. Hi ha molts tipus de pedals d'efectes que pots comprar. Alguns afegeixen reverberació, altres distorsió, etc. Un guitarrista connectarà la seva guitarra a un pedal FX –és a dir, de distorsió–, després prendrà un altre cable i connectarà (en cadena) un pedal de reverberació. La sortida del pedal de reverberació es pot connectar llavors a l'amplificador:

```
Guitar → Distortion → Reverb → Amplifier
```

Això s'anomena encadenament de FX. Sonic Pi suporta exactament això. A més, cada pedal sol tenir dials i lliscants que permeten controlar la quantitat de distorsió, reverberació, ressò, etc. que s'aplica. Sonic Pi també suporta aquest tipus de control. Finalment, imagina un guitarrista improvisant mentre una altra persona canvia i juga amb els controls dels seus pedals d'efectes. Sonic Pi també és compatible amb això, però en lloc de necessitar que una altra persona controli les coses per tu, és on entra l'ordinador.

Explorem els FX!

6.1. Afegir FX

En aquesta secció veurem un parell d'efectes: la reverberació i el ressò. Veurem com utilitzar-los, com controlar-ne les opcions i com encadenar-los.

El sistema de FX de Sonic Pi utilitza blocs. Així que si no has llegit la secció 5.1 potser vulguis fer-hi una ullada ràpida i després tornar.

Reverberació

Si volem utilitzar la reverberació, escrivim `with_fx :reverb` com a codi especial al nostre bloc, de la següent manera:

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

Ara reprodueix aquest codi i el sentiràs tocat amb reverberació. Sona bé, oi? Tot sona molt bé amb la reverberació.

Ara vegem què passa si tenim codi fora del bloc do/end:

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

sleep 1
play 55
```

Observa que l'últim `play 55` no es reprodueix amb reverberació. Això és perquè està fora del bloc do/end, per la qual cosa no és capturat per l'efecte de reverberació.

De la mateixa manera, si fas sons abans del bloc do/end, tampoc no seran capturats:

```
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
```

Ressò

Hi ha molts FX per triar. Què tal una mica de ressò?

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

Un dels aspectes més potents dels blocs FX de Sonic Pi és que se'ls poden passar opts similars a les opts que ja hem vist amb `play` i `sample`. Per exemple, una opció de ressò divertida amb què jugar és `phase:` que representa la durada d'un ressò determinat en pulsacions. Fem que el ressò sigui més lent:

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

Fem també que el ressò sigui més ràpid:

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

Fem que el ressò trigui més a esvair-se ajustant el temps de `decay:` a 8 temps:

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

Nidificació FX

Un dels aspectes més potents dels blocs FX és que els pots niar. Això permet encadenar fàcilment els FX. Per exemple, què passa si vols tocar un codi amb ressò i després amb reverberació? Fàcil, només has de posar-ne un dins de l'altre:

```
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
```

Pensa que l'àudio flueix de dins cap enfora. El so de tot el codi dins del bloc intern do/end com a `play 50` s'envia primer al ressò FX i el so del ressò FX s'envia al reverb FX.

Podem utilitzar nidificacions molt profundes per obtenir una enorme gamma de resultats. No obstant això, tingues en compte que els FX poden utilitzar molts recursos i quan els nidifiques estàs executant diversos FX simultàniament. Per tant, cal anar amb compte amb l'ús dels FX, especialment en plataformes de baixa potència com la Raspberry Pi.

Descobrir FX

Sonic Pi ve amb un gran nombre d'efectes perquè hi juguis. Per saber quins estan disponibles, fes clic a FX a l'extrem esquerre d'aquest sistema d'ajuda i veuràs una llista d'opcions disponibles. Aquí tens una llista d'alguns dels meus preferits:

- wobble,
- reverb,
- echo,
- distortion,
- slicer

Ara, arrisca't i afegeix FX per tot arreu per aconseguir sons nous i sorprenents!

6.2. FX a la pràctica

Encara que semblin aparentment senzills per fora, els FX són en realitat bèsties força complexes per dins. La seva simplicitat sol incitar la gent a utilitzar-los en excés a les seves peces. Això pot estar bé si tens un dispositiu potent, però si -com jo- fas servir una Raspberry Pi per improvisar, has d'anar amb compte amb la quantitat de treball que li demanes que faci si vols assegurar-te que els ritmes segueixin fluint.

Considera aquest codi:

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

En aquest codi estem tocant la nota 60 amb un temps d'alliberament molt curt, per la qual cosa és una nota curta. També volem reverberació, així que l'hem emmarcat en un bloc de reverberació. Tot bé fins ara. Excepte...

Vegem què fa el codi. Primer tenim un `loop`, cosa que significa que tot el que hi ha dins es repeteix per sempre. Després tenim un bloc `with_fx`. Això vol dir que crearem una nova reverberació FX cada vegada que fem el bucle. Això és com tenir un pedal de reverberació FX separat per cada vegada que prems una corda en una guitarra. Està bé que ho puguis fer, però no sempre és el que vols. Per exemple, aquest codi tindrà dificultats per funcionar bé a una Raspberry Pi. Tota la feina de crear la reverberació i després esperar fins que calgui aturar-la i eliminar-la la fas amb `with_fx`, però això roba la potència de la CPU, la qual és molt valuosa.

Com podem fer que s'assembli més a una configuració tradicional en què el nostre guitarrista té un sol pedal de reverberació per on passen tots els sons? Molt senzill:

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

Posem el nostre bucle dins el bloc `with_fx`. D'aquesta manera només creem una única reverberació per a totes les notes que es toquen al nostre bucle. Aquest codi és molt més eficient i funcionaria bé en una Raspberry Pi.

Un compromís és utilitzar with_fx sobre una iteració dins un bucle:

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

D'aquesta manera, hem tret el `with_fx` de la part interna del `bucle` i ara estem creant una nova reverberació cada 16 notes.

Aquest és un patró tan comú que `with_fx` suporta una opció per fer exactament el mateix però sense haver d'escriure el bloc `16.times`:

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

Tant els exemples de `reps: 16` i `16.times` do es comportaran de forma idèntica. El `reps: 16` essencialment repeteix el codi al bloc do/end 16 vegades, així que pots fer-los servir tots dos indistintament i triar el que et sembli millor.

Recorda que no hi ha errors, només oportunitats. No obstant això, alguns d'aquests enfocaments tindran un so diferent i també diferents característiques de rendiment. Així que juga i utilitza l'enfocament que et soni millor, dins de les limitacions de rendiment de la teva plataforma.

7. CONTROL

Fins ara hem vist com pots disparar sintetitzadors i mostres, i també com canviar les opcions per defecte, com l'amplitud, el panorama, la configuració de l'envolupant i molt més. Cada so disparat és essencialment el seu propi so amb la seva llista d'opcions establertes per a la durada del so.

No seria genial que poguessis canviar les opcions d'un so mentre segueix sonant, com podries doblegar una corda d'una guitarra mentre continua vibrant?

Estàs de sort: aquesta secció us mostrarà com fer-ho.

7.1. Control dels sintes en marxa

Fins ara només ens hem preocupat per activar nous sons i efectes. Tanmateix, Sonic Pi ens ofereix la capacitat de manipular i controlar els sons que s'executen actualment. Ho fem utilitzant una variable per capturar una referència a un sintetitzador:

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

Aquí, tenim una variable d'execució local s que representa la nota 60 de reproducció del sintetitzador. Tingues en compte que això és d'execució local: no hi pots accedir des d'altres funcions.

Un cop tinguem `s`, podem començar a controlar-la mitjançant la funció `control:`

```
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
```

El que cal notar és que aquí no estem activant 4 sintetitzadors diferents només estem disparant un sintetitzador i després canviem el to 3 vegades, mentre sona.

Podem passar qualsevol de les opcions estàndard a `control`, de manera que pots controlar coses com `amp:`, `cutoff:` o `pan:`.

Opcions no controlables

Algunes de les opcions no es poden controlar un cop s'ha iniciat el sintetitzador. Aquest és el cas de tots els paràmetres de l'envolupant ADSR. Pots esbrinar quines opcions es poden controlar consultant la seva documentació al sistema d'ajuda. Si la documentació diu *Can not be changed once set*, saps que no és possible controlar l'opció després que s'hagi iniciat el sintetitzador.

7.2. Control dels efectes

També és possible controlar els FX, encara que això

s'aconsegueix d'una manera lleugerament diferent:

```
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
```

En lloc de fer servir una variable, utilitzarem els paràmetres de porteria del bloc do/end. Dins de les barres `|`, hem d'especificar un nom únic per al nostre efecte en execució que després referenciarem des del bloc do/end que el conté. Aquest comportament és idèntic a l'ús de funcions parametritzades.

Ara prova de controlar alguns sintetitzadors i FX!

7.3. Opcions de lliscament

En explorar les diferents opcions de sintetitzador i efectes, hauràs notat que hi ha diverses opcions que acaben en `_slide`. Potser fins i tot hagis provat a usar-los i no hagis vist cap efecte. Això és perquè no són paràmetres normals, sinó que són opts especials que només funcionen quan es controlen els sintes, tal com s'ha introduït a la secció anterior.

Considera el següent exemple:

```
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
```

Aquí, pots escoltar el to del sintetitzador canviant immediatament a cada crida de `control`. Tanmateix, podem voler que el to llisqui entre els canvis. Com estem controlant el paràmetre `note:`, per afegir el lliscament (slide), hem d'establir el paràmetre `note_slide` del sintetitzador:

```
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
```

Ara sentim com es dobleguen les notes entre les crides a `control`. Sona bé, oi? Pots accelerar el lliscament indicant un temps més curt a `note_slide: 0.2` o alentir-lo utilitzant un temps de lliscament més llarg.

Cada paràmetre que pot ser controlat té el corresponent paràmetre `_slide` perquè hi juguis.

El lliscament és enganxós

Quan hagis establert un paràmetre `_slide` en un sintetitzador en funcionament, es recordarà i s'utilitzarà cada vegada que facis lliscar el paràmetre corresponent. Per deixar de lliscar has de posar el valor de `_slide` a 0 abans de la següent crida a `control`.

Opcions FX de lliscament

També és possible lliscar les opcions d'efectes:

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

Ara diverteix-te lliscant les coses per aconseguir transicions suaus i un control fluid...

8. ESTRUCTURES DE DADES

Una eina molt útil a la caixa d'eines d'un programador és una estructura de dades.

De vegades, pots voler representar i utilitzar més d'una cosa. Per exemple, et pot resultar útil tenir una sèrie de notes per tocar una darrere l'altra. Els llenguatges de programació tenen estructures de dades que et permeten fer exactament això.

Hi ha moltes estructures de dades interessants i exòtiques a disposició dels programadors, i la gent sempre n'està inventant de noves. No obstant, per ara només hem de considerar una estructura de dades molt senzilla: la llista.

Vegem-la amb més detall. Veurem la seva forma bàsica i també com es poden fer servir les llistes per representar escales i acords.

8.1. Llistes

En aquesta secció veurem una estructura de dades molt útil: la llista. La vam conèixer molt breument abans a la secció sobre aleatorietat quan vam triar a l'atzar entre una llista de notes per tocar:

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

En aquesta secció explorarem l´ús de llistes per representar també acords i escales. Primer recapitulem com podríem tocar un acord. Recorda que si no fem servir `sleep`, els sons es produeixen tots alhora:

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

Vegem altres maneres de representar aquest codi.

Reproducció d'una llista

Una opció és col·locar totes les notes en una llista: `[52, 55, 59]`. La nostra simpàtica funció `play` és prou intel·ligent per saber com reproduir una llista de notes. Prova-ho:

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

Ooh, això ja és més agradable de llegir. Tocar una llista de notes no t'impedeix utilitzar qualsevol dels paràmetres de forma normal:

```
play [52, 55, 59], amp: 0.3
```

Per descomptat, també pots utilitzar els noms tradicionals de les notes en lloc dels números MIDI:

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

Ara bé, els que tinguin la sort d'haver estudiat una mica de teoria musical potser reconeguin aquest acord com el Mi Menor tocat a la tercera octava.

Accedir a una llista

Una altra característica molt útil d'una llista és la possibilitat de treure'n informació. Això pot semblar una mica estrany, però no és més complicat que si algú et demana que obris un llibre a la pàgina 23. Amb una llista, diries, quin és l'element a l'índex 23? L'única cosa estranya és que en programació els índexs solen començar en 0 i no en 1.

Amb els índexs de les llistes no comptem 1, 2, 3... Sinó 0, 1, 2...

Vegem-ho amb una mica més de detall. Mira aquesta llista:

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

No hi ha res especialment aterridor en això. Ara, quin és el segon element d'aquesta llista? Sí, és clar, és el 55. Això ha estat fàcil. Vegem si podem fer que l'ordinador ens ho respongui també:

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

D'acord, sembla una mica estrany si mai no has vist res semblant. Però creu-me, no és tan difícil. Hi ha tres parts a la línia anterior: la paraula `puts` , la nostra llista `52, 55, 59` i el nostre índex `[1]`. Primer diem `puts` perquè volem que Sonic Pi ens imprimeixi la resposta al registre. Després, estem donant la nostra llista, i finalment el nostre índex està demanant el segon element. Hem de col·locar el nostre índex entre claudàtors i com que el recompte comença en `0`, l'índex per al segon element és `1`. Mira:

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

Prova a executar el codi `puts [52, 55, 59][1]` i veuràs que apareix `55` al registre. Canvia l'índex `1` per altres índexs, prova amb llistes més llargues i pensa com podries utilitzar una llista en el teu proper improvisació de codi. Per exemple, quines estructures musicals podrien representar-se com una sèrie de números...

8.2. Acords

Sonic Pi disposa de suport nadiu pels noms dels acords, que retorna llistes. Prova-ho tu mateix:

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

Ara sí que estem arribant a alguna cosa. Això es veu molt més bonic que les llistes en brut (i és més fàcil de llegir per a altres persones). Quins altres acords suporta Sonic Pi? Bé, molts. Prova-ho amb alguns d'aquests:

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

Arpegis

Podem convertir fàcilment els acords en arpegis amb la funció `play_pattern`:

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

Ok, això no és tan divertit - ha sonat molt lent. `play_pattern` tocarà cada nota de la llista separada amb una crida a `sleep 1` entre cada crida a `play`. Podem fer servir la funció `play_pattern_timed` per especificar els nostres propis temps i accelerar les coses:

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

Fins i tot podem passar una llista de temps que serà tractada com un cercle de temps:

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

Això és l'equivalent 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
```

Quina preferiries escriure?

8.3. Escales

Sonic Pi és compatible amb una àmplia gamma d'escales. Què et semblaria tocar una escala de Do major?

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

Fins i tot podem demanar més octaves:

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

I si volguéssim totes les notes en una escala pentatònica?

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

Notes aleatòries

Els acords i les escales són bones maneres de limitar una elecció aleatòria a alguna cosa significativa. Juga amb aquest exemple que tria notes aleatòries de l'acord Mi menor:

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

Prova de canviar diferents noms d'acords i rangs de tall.

Descobrir els acords i les escales

Per saber quines escales i acords són compatibles amb Sonic Pi, simplement fes clic al botó *Lang* a l'extrem esquerre d'aquest tutorial i després tria un acord o una escala a la llista de l'API. A la informació del panell principal, desplaça't cap avall fins que vegis una llarga llista d'acords o escales (depenent del que estiguis mirant).

Diverteix-te i recorda: no hi ha errors, només oportunitats.

8.4. Anells (*Rings*)

Un gir interessant a les llistes estàndard són els anells (*rings*). Si saps una mica de programació, potser t'has trobat amb buffers d'anell o matrius d'anell. Aquí, només anem a buscar l'anell: ras i curt.

A la secció anterior sobre llistes vam veure com podríem treure'n elements mitjançant el mecanisme d'indexació:

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

Ara, què passa si vols l'índex `100`? Bé, és evident que no hi ha cap element a l'índex 100, ja que la llista només té tres elements. Així que Sonic Pi et retornarà `nil`, cosa que no vol dir res.

Tanmateix, considera que tens un comptador, amb el ritme actual del beat, que augmenta contínuament. Creem el nostre comptador i la nostra llista:

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

Ara podem utilitzar el nostre comptador per accedir a una nota de la nostra llista:

```
puts notes[counter]
```

Genial, obtenim la `52`. Ara, augmentem el nostre comptador i obtenim una altra nota:

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

Súper, ara obtenim la `55` i si ho tornem a fer, obtenim la `59`. Tanmateix, si ho tornem a fer, ens quedarem sense nombres a la nostra llista i ens quedarem amb `nil`. Què passaria si només volguéssim tornar enrere i començar de nou al principi de la llista? Per això serveixen els anells.

Crear anells

Podem crear anells de dues maneres diferents. O fem servir la funció `ring` amb els elements de l'anell com a paràmetres:

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

O podem prendre una llista normal i convertir-la en un anell enviant-li el missatge `.ring`:

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

Indexar anells

Un cop tinguis un anell, el pots utilitzar exactament de la mateixa manera que faries servir una llista normal, amb l'excepció que pots utilitzar índexs que siguin negatius o més grans que la mida de l'anell que apuntaran sempre a un dels elements de l'anell:

```
(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
```

Usar anells

Suposem que estem utilitzant una variable per representar el nombre de ritme (beat) actual. Podem utilitzar-la com a índex al nostre anell per buscar notes per tocar, o temps de llançament o qualsevol cosa útil que hem emmagatzemat al nostre anell, independentment del número de ritme en què ens trobem actualment.

Les escales i els acords són anells

Una cosa útil a saber és que les llistes retornades per `scale` i `chord` també són anells i permeten accedir-hi amb índexs arbitraris.

Constructors d'anells

A més de `ring`, hi ha una sèrie d'altres funcions que ens construiran un anell.

- `range` et convida a especificar un punt inicial, un punt final i una mida del pas.
- `bools` et permet utilitzar `1` i `0` per representar de manera sucinta els booleans.
- `knit` et permet teixir una seqüència de valors repetits.
- `spread` crea un anell de bools amb una distribució euclidiana.

Fes una ullada a la seva documentació respectiva per obtenir-ne més informació.

8.5. Cadenes d'anells

A més dels constructors com `range` i `spread`, una altra forma de crear nous anells és manipulant els anells existents.

Comandaments de cadena

Per explorar-ho, pren un anell senzill:

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

I si ho volguéssim al revés? Bé, utilitzaríem l'ordre de cadena `.reverse` per prendre l'anell i donar-li la volta:

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

Ara, i si volguéssim els tres primers valors de l'anell?

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

I per acabar, i si volguéssim barrejar els elements de l'anell?

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

Cadenes múltiples

Aquesta ja és una manera potent de crear nous anells. Tanmateix, el poder real arriba quan encadenes algunes d'aquestes ordres.

Què et semblaria barrejar els elements de l'anell, deixar caure 1 element i després agafar els 3 següents?

Prenem això per etapes:

1. `(ring 10, 20, 30, 40, 50)` - El nostre anell inicial
2. `(ring 10, 20, 30, 40, 50).shuffle` - barregem - `(ring 40, 30, 10, 50, 20)`
3. `(ring 10, 20, 30, 40, 50).shuffle.drop(1)` - en deixem 1 - `(ring 30, 10, 50, 20)`
4. `(ring 10, 20, 30, 40, 50).shuffle.drop(1).take(3)` - n'agafem 3 - `(ring 30, 10, 50)`

Pots veure com podem crear una cadena llarga d'aquests mètodes només unint-los. Podem combinar-los en qualsevol ordre que vulguem creant una manera extremadament rica i potent de generar nous anells a partir dels existents.

Immutabilitat

Aquests anells tenen una propietat potent i important. Són immutables, és a dir, no poden canviar. Això vol dir que els mètodes d'encadenament descrits en aquesta secció no canvien els anells sinó que en creen nous. Això vol dir que ets lliure de compartir anells entre fils i començar a encadenar-los dins d'un fil sabent que no afectaràs cap altre fil amb el mateix anell.

Mètodes de cadena disponibles

Aquí tens una llista dels mètodes de cadena disponibles amb els quals pots jugar:

- `.reverse:` retorna una versió invertida de l'anell
- `.sort` - crea una versió ordenada de l'anell
- `.shuffle` - crea una versió barrejada de l'anell
- `.pick:` retorna un timbre amb els resultats de la crida `.choose` una vegada
- `.pick(3)` - retorna un anell amb els resultats de la crida `.choose` 3 vegades
- `.take(5)` - retorna un nou anell que només conté els 5 primers elements
- `.drop(3)` - retorna un nou anell amb tot menys els 3 primers elements
- `.butlast` - retorna un nou anell amb l'últim element que falta
- `.drop_last(3)` - retorna un nou anell amb els darrers 3 elements que falten
- `.take_last(6)` - retorna un nou anell amb només els darrers 6 elements
- `.stretch(2)` - repeteix cada element de l'anell dues vegades
- `.repeat(3)` - repeteix l'anell sencer 3 vegades
- `.mirror` - afegeix l'anell a una versió invertida de si mateix
- `.reflect` - igual que mirall però no duplica el valor mitjà
- `.scale(2)` - retorna un nou anell amb tots els elements multiplicats per 2 (suposa que l'anell només conté números)

Per descomptat, els mètodes de cadena que prenen números també poden prendre altres números! Així que no dubtis a fer una crida a `.drop(5)` en lloc de `.drop(3)` si vols deixar anar els 5 primers elements.

9. CODIFICACIÓ EN VIU

Un dels aspectes més interessants de Sonic Pi és que et permet escriure i modificar el codi en directe per fer música, igual que si toquessis en directe amb una guitarra. Un dels avantatges d'aquest enfocament és que et dóna més retroalimentació mentre composes (fes un simple bucle i segueix ajustant-lo fins que soni perfecte). Tot i això, el principal avantatge és que pots portar Sonic Pi a l'escenari i actuar.

En aquesta secció cobrirem els fonaments per convertir les teves composicions de codi estàtic en actuacions dinàmiques.

Agafa't a la cadira...

9.1 Fonaments de la codificació en viu

Codificació en viu

Ara ja en sabem prou com per començar a divertir-nos de debò. En aquesta secció ens basarem en totes les seccions anteriors i et mostrarem com pots començar a fer les teves composicions musicals en directe i convertir-les en una actuació. Per això necessitarem 3 ingredients principals:

- Una habilitat per escriure codi que faci sons - CORRECTE!
- Una habilitat per escriure funcions - CORRECTE!
- Una habilitat per utilitzar fils (amb nom) - CORRECTE!!

Molt bé, comencem. Codificarem en viu els nostres primers sons. Primer necessitem una funció que contingui el codi que volem reproduir. Comencem de forma senzilla. També volem fer un bucle de crides a aquesta funció en un fil:

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

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

Si això et sembla massa complicat, torna a llegir les seccions sobre funcions i fils. No t'ho hauria de semblar si ja t'has fet la idea d'aquestes coses.

El que tenim aquí és una definició de funció que simplement reprodueix la nota 50 i dorm durant un temps. Després definim un fil amb nom anomenat `:looper` que simplement fa un bucle cridant a `my_sound` repetidament.

Si executes aquest codi, escoltaràs la nota 50 repetint-se una vegada i una altra...

Canviem-ho

Ara és quan comença la diversió. Mentre el codi s'està executant, canvia el 50 per un altre número, diguem el 55, i torna a prémer el botó Run. Woah! Ha canviat! ¡En viu!

No hi ha afegit una nova capa perquè estem usant fils amb nom que només permeten un fil per a cada nom. A més, el so ha canviat perquè hem redefinit la funció. Hem donat a `:my_sound` una nova definició. Quan el fil de `:looper` ha començat a sonar en bucle, simplement ha cridat a la nova definició.

Intenta tornar-ho a canviar, canvia la nota, canvia el temps de repòs. Com afegir una declaració `use_synth`? Per exemple, canvia'l per:

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

Ara sona força interessant, però podem donar-li més sabor. En lloc de tocar la mateixa nota una vegada i una altra, intenta tocar un acord:

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

Què tal si toques notes de l'acord a l'atzar?

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

O utilitzes un valor de tall aleatori:

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

Finalment, afegeix alguns tambors:

```
define :my_sound 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
```

Ara les coses es posen emocionants!

No obstant això, abans que saltis i comencis a codificar en viu amb funcions i fils, deixa el que estàs fent i llegeix la secció següent sobre `live_loop` que canviarà la forma de codificar a Sonic Pi per sempre...

9.2 Bucles en viu

Bé, aquesta secció del tutorial és la veritable joia de la corona. Si només llegissis una secció, hauria de ser aquesta. Si has llegit la secció anterior sobre els fonaments de la codificació en viu, `live_loop` és una manera senzilla de fer exactament el mateix però sense haver d'escriure tant.

Si no has llegit la secció anterior, `live_loop` és la millor manera d'improvisar amb Sonic Pi.
zt
Comencem a jugar. Escriu el següent en un nou buffer:

```
live_loop :foo do
  play 60
  sleep 1
end
```

Ara prem el botó Run. Se sent un xiulet bàsic cada vegada que es pitja. Res divertit. Però no premis Stop encara. Canvia el `60` pel `65` i prem Run de nou.

Vaja. Ha canviat automàticament sense perdre el ritme. Això és codificació en viu.

Per què no canviar-ho perquè sigui més semblant a un baix? Només has d'actualitzar el teu codi mentre s'està reproduint:

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

Aleshores, prem *Run*.

Fem que el tall es mogui:

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

Prem *Run* de nou.

Afegeix alguns tambors:

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

Canvia la nota de `e1` a `c1`:

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

Ara deixa d'escoltar-me i juga tu mateix! Diverteix-te!

9.3 Múltiples bucles en viu

Considera el següent bucle en viu (live_loop):

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

T'hauràs preguntat per què necessita el nom `:foo`. Aquest nom és important perquè vol dir que aquest bucle en viu és diferent de tots els altres bucles en viu.

Mai no hi pot haver dos bucles en viu executant-se amb el mateix nom.

Això vol dir que si volem que s'executin diversos bucles en viu simultàniament, només els hem de donar noms diferents:

```
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
```

Ara pots actualitzar i canviar cada bucle en viu de forma independent i tot funciona.

Sincronització de bucles en viu

Una cosa que ja hauràs notat és que els bucles en viu funcionen automàticament amb el mecanisme de seguiment del fil que hem explorat anteriorment. Cada vegada que el bucle en viu fa un bucle, genera un nou `esdeveniment` amb el nom del bucle en viu. Per tant, podem `sincronitzar` aquests senyals per assegurar que els nostres bucles estan sincronitzats sense haver d'aturar res.

Considera aquest codi mal sincronitzat:

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

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

Vegem si podem arreglar el temps i la sincronització sense aturar-lo. Primer, arreglem el bucle `:foo` perquè sleep sigui un factor de 1 - una cosa com `0.5` servirà:

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

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

Vaja, ara tot està perfectament a temps - i sense haver hagut de parar.

Ja pots començar a codificar amb bucles en viu!

9.4 tic-tac (*ticking*)

Una cosa que probablement faràs molt quan codifiquis en directe és fer bucles a través d'anells. Posaràs notes als anells per a les melodies, *sleeps* pels ritmes, progressions d'acords, variacions tímbriques, etc., etc.

Anells de tic-tac (Ticking Rings)

Sonic Pi proporciona una eina molt útil per treballar amb anells dins de `live_loops`. Es diu el sistema de ticks (marques, o tic-tacs). A la secció sobre els anells parlàvem del comptador que augmenta constantment, com un nombre del compàs actual. Tick simplement implementa aquesta idea. Et proporciona la capacitat de fer un tick a través dels anells. Vegem-ne un exemple:

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

Això és equivalent a:

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

En aquest cas, només estem prenent l'escala de Mi menor pentatònica i fent un *tick* a cada element. Això es fa afegint `.tick` al final de la declaració de l'escala. Aquest *tick* és local per al bucle en viu, de manera que cada bucle en viu pot tenir el seu propi *tick* independent:

```
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

També pots cridar a `tick` com una fn estàndard i utilitzar el valor com a índex:

```
live_loop :arp do
  idx = tick
  play (scale :e3, :minor_pentatonic)[idx], release: 0.1
```

```
  sleep 0.125
end
```

No obstant això, és molt més agradable cridar a `.tick` al final. La fn `tick` és per quan vulguis fer coses extravagants amb el valor del tick i per quan vulguis fer servir els ticks per a altres coses que no siguin indexar en anells.

Look

La màgia de tick és que no només retorna un nou índex (o el valor de l'anell en aquest índex) sinó que també s'assegura que la propera vegada que cridi a `tick`, sigui el següent valor. Fes una ullada als exemples de la documentació de tick per veure diferents maneres de treballar amb això. Ara per ara, però, és important assenyalar que de vegades voldràs només mirar el valor actual del tick i no incrementar-lo. Això ho pots fer mitjançant la fn `look`. Pots cridar a `look` com una fn estàndard o afegir `.look` al final d'un anell.

Posar nom als *ticks*

Finalment, de vegades necessitaràs més d'un *tick* per bucle en viu. Això s'aconsegueix donant un nom al teu tick:

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

Aquí estem usant dos ticks, un per a la nota a tocar i un altre per al temps de repòs. Com que tots dos són al mateix bucle en viu, per mantenir-los separats necessitem donar-los noms únics. Això és exactament el mateix que posar nom a `live_loops` - simplement passem un símbol prefixat amb un `:`. A l'exemple anterior hem anomenat un bucle `:foo` i l'altre `:bar`. Si els volem `veure`, també hem de passar el nom del *tick* que volem veure.

No et compliquis la vida

La major part del poder del sistema de *ticks* no és útil quan estàs començant. No intentis aprendre-ho tot en aquesta secció. Només concentra't en el *ticking* a través d'un sol anell. Això t'aportarà la satisfacció i la simplicitat de fer *ticking* a través dels anells en els teus `live_loops`.

Fes una ullada a la documentació sobre *tick* on trobaràs molts exemples útils i gaudeix del *ticking*!

10. *TIME STATE*

Sovint és útil tenir la informació que es comparteix mitjançant múltiples fils o bucles en viu. Per exemple, és possible que vulguis compartir una noció de la clau actual, el BPM o fins i tot conceptes més abstractes com la "complexitat" actual (que potencialment s'interpretaria de diferents maneres a través de diferents fils). Tampoc volem perdre cap de les nostres garanties de determinisme existents en fer-ho. En altres paraules, ens agradaria ser capaços de compartir el codi amb altres persones i saber exactament el que escoltaran quan l'executin. Al final de la Secció 5.6 d'aquest tutorial hem comentat breument per què no hauríem de fer servir variables per compartir informació entre fils a causa de la pèrdua de determinisme (també a causa de les condicions de la carrera).

La solució de Sonic Pi al problema de treballar fàcilment amb variables globals de forma determinista és a través d'un nou sistema que anomena *Time State*. Això pot semblar complex i difícil (de fet, al Regne Unit, programar amb múltiples fils i memòria compartida sol ser una assignatura de nivell universitari). Tanmateix, com veuràs, com quan toques la teva primera nota, Sonic Pi fa que sigui increïblement senzill compartir l'estat entre fils sense deixar de mantenir els teus programes segurs i deterministes.

És hora de coneixer `get` i `set`...

10.1 *Set* i *Get*

Sonic Pi té un magatzem de memòria global anomenat *Time State*. Les dues coses principals que s'hi fan són establir (`set`) i obtenir (`get`) informació. Aprofundim-hi...

Set

Per emmagatzemar informació a *Time State* necessitem dues coses:

1. la informació que volem emmagatzemar,
2. un nom únic (clau) per a la informació.

Per exemple, podríem voler emmagatzemar el número `3000` amb la clau `:intensity`. Això és possible utilitzant la funció `set:`

```
set :intensity, 3000
```

Podem utilitzar qualsevol nom per a la nostra clau. Si ja s'ha emmagatzemat informació amb aquesta clau, el nostre nou conjunt l'anul·larà:

```
set :intensity, 1000
set :intensity, 3000
```

A l'exemple anterior, com emmagatzemem ambdós números sota la mateixa tecla, l'última crida a `set` 'guanya',

de manera que el número associat a `:intensity` serà `3000` ja que la primera crida a `set` ha estat efectivament anul·lada.

Get

Per obtenir informació del *Time State* només necessitem la clau que fem servir per establir-lo, que en el nostre cas és `:intensity`. Aleshores només hem de cridar a `get[:intensity]` que podem veure imprimint el resultat al registre:

```
print get[:intensity] #⇒ prints 3000
```

Observa que les crides a `get` poden retornar informació establerta en una execució anterior. Quan s'ha establert (`set`) una informació, aquesta està disponible fins que s'anul·li (igual que vam fer amb el valor de `: intensity` de `1000` a `3000` més amunt) o es tanqui Sonic Pi.

Múltiples fils

El principal avantatge del sistema Time State és que es pot utilitzar de manera segura entre fils o bucles vius. Per exemple, pots tenir un live_loop establint informació i un altre obtenint-la:

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

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

El millor d'usar `get` i `set` a través de fils com aquest és que sempre produirà el mateix resultat cada vegada que prems *Run*. Va, prova-ho. Mira si obtens el següent al teu registre:

```
{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
```

Prova d'executar-ho diverses vegades i veuràs que sempre és igual. Això és el que anomenem comportament determinista i és realment molt important quan volem compartir la nostra música com a codi i saber que la persona que reprodueix el codi està escoltant exactament el que volíem que escoltés (igual que la reproducció d'un MP3 o d'un *stream* d'Internet sona igual per a tots els oients).

Un sistema d'estats determinista simple

A la secció 5.6 vam explicar per què l'ús de variables entre fils pot provocar un comportament aleatori. Això ens impedeix reproduir de manera fiable codi com aquest:

```
## An Example of Non-Deterministic Behaviour
## (due to race conditions caused by multiple
## live loops manipulating the same variable
## at the same time).
##
## If you run this code you'll notice
## that the list that's printed is
## not always sorted!
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
```

Vegem com podria ser això usant `get` i `set`:

```
## An Example of Deterministic Behaviour
## (despite concurrent access of shared state)
## using Sonic Pi's new Time State system.
##
## When this code is executed, the list that's
## printed is always sorted!
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
```

Fixa't que aquest codi és pràcticament idèntic a la versió que utilitza una variable abans. No obstant això, quan executes el codi, aquest es comporta com esperaries amb qualsevol codi típic de Sonic Pi – fa el mateix cada vegada en aquest cas gràcies al sistema *Time State*.

Per tant, quan comparteixes informació a través de bucles en viu i fils, utilitzes `get` i `set` en lloc de variables per a un comportament determinista i reproduïble.

10.2 Sincronització (*Sync*)

La secció 5.7 va introduir les funcions `cue` i `sync` en tractar el tema de la sincronització de fils. El que no explicava era que el sistema *Time State* és el que proporciona aquesta funcionalitat. Succeeix que `set` és en realitat una variació de `cue` i està construïda sobre la mateixa funcionalitat principal que és inserir informació al sistema *Time State*. A més, `sync` també està dissenyada de manera que funciona sense problemes amb *Time State* - qualsevol informació que planegem emmagatzemar a *Time State* podem sincronitzar-la. En altres paraules, sincronitzem els esdeveniments que encara no han estat inserits a *Time State*.

Esperar els esdeveniments

Vegem ràpidament com utilitzar `sync` per esperar que s'afegeixin nous esdeveniments a *Time State*:

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

sleep 2

set :foo, 1
```

En aquest exemple primer creem un fil que espera un esdeveniment `:foo` per ser afegit al Time State. Després de la declaració d'aquest fil fem sleep durant 2 temps i després establim que `:foo` sigui `1`. Això allibera el `sync`, que passa a la línia següent, activant la mostra `:ambi_lunar_land`.

Tingues en compte que el `sync` sempre espera esdeveniments futurs i bloquejarà el fil actual esperant un nou esdeveniment. A més, heretarà el temps lògic del fil que el va disparar via `set` o `cue` per la qual cosa també pot ser usat per sincronitzar el temps.

Passar valors al Futur

A l'exemple anterior hem establert `:foo` a `1`, és a dir que no hem fet res. En realitat podem obtenir aquest valor del fil cridant a `sync`:

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

sleep 2

set :foo, 0.5
```

Tingues en compte que els valors que es passen a través de `set` i `cue` han de ser segurs per al fil - és a dir, anells immutables, números, símbols o cadenes congelades. Sonic Pi llançarà un error si el valor que s'intenta emmagatzemar al *Time State* no és vàlid.

10.3 Coincidència de patrons

En obtenir i establir informació al *Time State*, és possible utilitzar claus més complexes que els símbols bàsics com `:foo` i `:bar`. També es poden utilitzar cadenes d'estil URL anomenades rutes com `"/foo/bar/baz"`. Quan comencem a treballar amb rutes, podem començar a aprofitar el sofisticat sistema de coincidència de patrons de Sonic Pi per obtenir i sincronitzar amb rutes "similars" en lloc d'"idèntiques". Fem-hi una ullada.

Fer coincidir qualsevol segment del patró

Suposem que volem esperar el següent esdeveniment que té tres segments de ruta:

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

Això coincidirà amb qualsevol esdeveniment del Time State amb exactament tres segments de ruta, independentment dels noms. Per exemple:

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

Tot i això, no coincidirà amb rutes amb menys o més segments de ruta. Les següents no coincidiran:

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

Cada `*` significa qualsevol contingut. Així que podríem fer coincidir rutes amb un sol segment amb `/*` o rutes amb cinc segments amb `/*/*/*/*/*`

Segments parcials coincidents

Si sabem amb què començarà o acabarà el segment, podem utilitzar un `*` a més d'un nom de segment parcial. Per exemple: `"/foo/b*/baz"` coincidirà amb qualsevol ruta que tingui tres segments, el primer dels quals és `foo`, l'última `baz` i el segment del mig pot ser qualsevol cosa que comenci per `b`. Així, coincidiria:

- `cue "/foo/bar/baz"`
- `cue "/foo/baz/baz"`
- `cue "/foo/beans/baz"`

Tanmateix, no coincidiria amb el següent:

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

També pots col·locar el `*` al principi del segment per especificar els últims caràcters d'un segment: `"/foo/*zz/baz"`, que coincidirà amb qualsevol tac o conjunt de 3 segments `cue` o `set`, en què el primer segment sigui `foo`, l'últim sigui `baz` i el segment del mig acabi en `zz`, com `"cue "/foo/whizz/baz"`.

Segments de camí coincidents

De vegades no se sap quants segments de la ruta es volen fer coincidir. En aquests casos pots utilitzar la poderosa estrella doble: `**` com a `"/foo/**/baz"` que coincidirà:

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

Caràcters coincidents

Pots utilitzar el caràcter ? per comparar amb un sol caràcter com a "/?oo/bar/baz" que coincidirà:

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

Paraules coincidents

Si saps que un segment pot ser un nombre concret de paraules, pots utilitzar els comparadors `{` i `}` per especificar una llista d'opcions com `"/foo/{bar,beans,eggs}/quux"` que només coincidirà amb el següent:

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

Lletres coincidents

Finalment, pots comparar amb una selecció de lletres si utilitzes els comparadors [i] per especificar una llista d'opcions com `"/foo/[abc]ux/baz"` que només coincidirà:

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

També pots utilitzar el caràcter `-` per especificar intervals de lletres. Per exemple `"/foo/[a-e]ux/baz"` que només coincidirà:

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

Combinar *matchers*

Quan es crida `sync` o `get` es poden combinar els matchers en l'ordre que es consideri oportú per fer coincidir qualsevol esdeveniment del *Time State* creat per `cue` o `set`. Vegem-ne un exemple molt rebuscat:

```
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/"
```

OSC *Pattern Matching*

Per als curiosos, aquestes regles de concordança es basen en l'especificació de concordança de patrons d'*Open Sound Control* que s'explica detalladament aquí: http://opensoundcontrol.org/spec-1_0

11. MIDI

Quan hagis dominat la conversió de codi en música, et preguntaràs: què és el següent? De vegades, les limitacions de treballar únicament amb la sintaxi i el sistema de so de Sonic Pi poden ser emocionants i col·locar-te en una nova posició creativa. No obstant això, de vegades és essencial sortir del codi i entrar al món real. Volem dues coses més:

- Ser capaços de convertir les accions del món real en esdeveniments de Sonic Pi amb què codificar
- Ser capaços d'utilitzar el fort model de temporització i la semàntica de Sonic Pi per controlar i manipular objectes al món real

Per sort, hi ha un protocol que existeix des dels anys 80 que permet exactament aquesta mena d'interacció: MIDI. Hi ha un nombre increïble de dispositius externs, incloent teclats, controladores, seqüenciadors i programari d'àudio professional que suporten MIDI. Podem fer servir MIDI per rebre dades i també per enviar-les.

Sonic Pi proporciona suport complet per al protocol MIDI, permetent connectar el teu codi en viu amb el món real. Explorem-ho més a fons...

11.1 MIDI In

En aquesta secció aprendrem a connectar una controladora MIDI per enviar esdeveniments a Sonic Pi per controlar els nostres sintetitzadors i sons. Agafa una controladora MIDI com un teclat o una superfície de control i posem-nos físics!

Connectar una controladora MIDI

Per obtenir informació d'un dispositiu MIDI extern a Sonic Pi, primer hem de connectar-lo al nostre ordinador. Normalment això serà a través d'una connexió USB, encara que els equips més antics tindran un connector DIN de 5 pins per al qual necessitaràs suport de maquinària per al teu ordinador (per exemple, algunes targetes de so tenen connectors MIDI DIN). Quan hagis connectat el teu dispositiu, engega Sonic Pi i fes una ullada a la secció IO del panell de Preferències. Hauries de veure el teu dispositiu a la llista. Si no és així, prova a prémer el botó *'Reset MIDI'* i miria si apareix. Si segueixes sense veure res, el pas següent és consultar la configuració MIDI del sistema operatiu per veure si veu el teu dispositiu. Si no ho aconsegueixes, no dubtis a preguntar als nostres amables fòrums: https://in-thread.sonic-pi.net

Rebre esdeveniments MIDI

Quan el dispositiu estigui connectat, Sonic Pi rebrà automàticament els esdeveniments. Pots comprovar-ho tu mateix manipulant el dispositiu MIDI i mirant el registre de cue a la part inferior dreta de la finestra de l'aplicació, sota el registre (si no està visible, ves a Preferències->Editor->Mostrar i Ocultar i activa la casella 'Mostrar registre de cue'). Veuràs un flux d'esdeveniments com:

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

Quan puguis veure un flux de missatges com aquest, voldrà dir que has connectat correctament el teu dispositiu MIDI. Enhorabona, a veure què en podem fer!

Estat de temps MIDI

Aquests esdeveniments es divideixen en dues seccions. En primer lloc hi ha el nom de l'esdeveniment com a `/midi:nanokey2_keyboard:0:1/note_on` i en segon lloc hi ha els valors de l'esdeveniment com `[18, 62]`. Curiosament, aquestes són les dues coses que necessitem per emmagatzemar la informació a *Time State*. Sonic Pi insereix automàticament els esdeveniments MIDI entrants a *Time State*. Això significa que pots obtenir l'últim valor MIDI i també sincronitzar l'espera del proper valor MIDI usant tot el que vam aprendre a la secció 10 d'aquest tutorial.

Controlar el codi

Ara que hem connectat un dispositiu MIDI, hem vist els seus esdeveniments al registre de `cue` i hem descobert que el nostre coneixement de *Time State* és tot el que necessitem per treballar amb els esdeveniments, podem començar a divertir-nos. Provem de construir un simple piano MIDI:

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

Hi ha algunes coses que succeeixen al codi anterior, incloent alguns problemes. En primer lloc, tenim un simple `live_loop` que es repetirà per sempre executant el codi entre el bloc `do`/`end`. Això serà introduït a la Secció 9.2. En segon lloc, cridem a `sync` per esperar el següent esdeveniment del *Time State* que coincideixi. Fem servir una cadena que representa el missatge MIDI que estem buscant (que és el mateix que es mostrava al *cue logger*). Fixa't que aquesta llarga cadena te la proporciona el sistema d'autocompletat de Sonic Pi, perquè no l'hagis d'escriure tota a mà. Al `log` vam veure que hi havia dos valors per a cada esdeveniment de nota MIDI on, així que vam assignar el resultat a dues variables separades, `note` i `velocity`. Finalment disparem el sintetitzador `:piano` passant la nostra nota.

Ara, prova-ho tu. Escriu el codi anterior, substitueix la tecla de `sync` per una cadena que coincideixi amb el teu dispositiu MIDI específic i prem *Run*. I llest, tens un piano que funciona! No obstant això, probablement notaràs un parell de problemes: en primer lloc, totes les notes tenen el mateix volum independentment de la força amb què colpeges el teclat. Això es pot arreglar fàcilment utilitzant el valor MIDI de la velocitat i convertint-ho en una amplitud. Atès que MIDI té un rang de 0->127, per convertir aquest número en un valor entre 0->1 només l'hem de dividir entre 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
```

Actualitza el codi i torna a prémer *Run*. Ara es respecta la

velocitat del teclat. A continuació, desfem-nos d'aquesta pausa molesta.

Eliminar la latència

Abans de poder eliminar la pausa, necessitem saber per què hi és. Per tal de mantenir tots els sintetitzadors i FX ben sincronitzats a través d'una varietat de CPUs de diferent capacitat, Sonic Pi programa l'àudio per avançat en 0,5s per defecte. (Tingues en compte que aquesta latència afegida es pot configurar a través de les fns `set_sched_ahead_time!` i `use_sched_ahead_time`). Aquesta latència de 0,5s s'ha afegit als disparadors del nostre sintetitzador `:piano` com s'afegeix a tots els *sintes* disparats per Sonic Pi. Normalment aquesta latència afegida la necessitem per tal de sincronitzar tots els sintetitzadors. Tanmateix, això només té sentit per als sintetitzadors disparats pel codi utilitzant `play` i `sleep`. En aquest cas, estem disparant el sintetitzador `:piano` amb el nostre dispositiu MIDI extern i, per tant, no volem que Sonic Pi controli la sincronització per nosaltres. Podem desactivar aquesta latència amb el comandament `use_real_time` que desactiva la latència per al fil actual. Això significa que pots utilitzar el mode de temps real per als bucles en viu la sincronització dels quals és controlada amb dispositius externs, i mantenir la latència per defecte per a tots els altres bucles en viu. Vegem-ho:

```
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
```

Actualitza el teu codi perquè coincideixi amb el codi anterior i torna a prémer *Run*. Ara tenim un piano de baixa latència amb velocitat variable codificat en només 5 línies. No sigut tan fàcil!

Obtenir valors

Finalment, com que els nostres esdeveniments MIDI van directament al *Time State*, també podem fer servir la fn `get` per recuperar l'últim valor vist. Això no bloqueja el fil actual i retorna `nil` si no hi ha cap valor a trobar (el qual pots anul·lar passant un valor per defecte - veure els documents de `get`). Recorda que pots cridar a `get` en qualsevol fil i en qualsevol moment per veure l'últim valor del *Time State* que coincideixi. Fins i tot pots utilitzar time_warp per saltar cap enrere en el temps i cridar a `get` per veure esdeveniments passats...

Ara tens el control

El que és emocionant ara és que pots utilitzar les mateixes estructures de codi per sincronitzar i obtenir informació MIDI de qualsevol dispositiu MIDI i fer el que vulguis amb els valors. Ara pots triar el que farà el teu dispositiu MIDI!

11.2 MIDI Out

A més de rebre esdeveniments MIDI també podem enviar esdeveniments MIDI per disparar i controlar sintetitzadors de maquinari externs, teclats i altres dispositius. Sonic Pi proporciona un conjunt complet de fns per enviar diversos missatges MIDI com:

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`

També hi ha molts altres missatges MIDI compatibles: consulta la documentació de l'API per a totes les altres fns que comencen amb `midi_`.

Connectar-se a un dispositiu MIDI

Per enviar un missatge MIDI a un dispositiu extern, primer l'hem d'haver connectat. Consulta la subsecció "Connexió d'un controlador MIDI" a la secció 11.1 per a més detalls. Tingues en compte que si utilitzes USB, connectar-se a un dispositiu al qual estàs enviant (en lloc de rebre) funciona de la mateixa manera. Tanmateix, si utilitzes els clàssics connectors DIN, assegura't de connectar-te al port de sortida MIDI del teu ordinador. Hauries de veure el teu dispositiu MIDI al panell de preferències.

Enviar esdeveniments MIDI

Les fns `midi_*` funcionen igual que `play`, `sample` i `synth` en el sentit que envien un missatge al temps actual (lògic). Per exemple, per repartir les crides a les fns `midi_*` necessites fer servir `sleep` igual que feies amb play. Fem-hi una ullada:

```
midi_note_on :e3, 50
```

Això enviarà una nota MIDI al dispositiu MIDI connectat amb velocitat 50. (Tingues en compte que Sonic Pi convertirà automàticament les notes de la forma `:e3` al seu número MIDI corresponent, com 52 en aquest cas).

Si el dispositiu MIDI connectat és un sintetitzador, hauries de poder sentir-lo tocar una nota. Per desactivar-lo utilitza `midi_note_off`:

```
midi_note_off :e3
```

Seleccionar un dispositiu MIDI

Per defecte, Sonic Pi enviarà cada missatge MIDI a tots els dispositius connectats a tots els canals MIDI. Això és així per facilitar el treball amb un sol dispositiu connectat i no haver de configurar res. No obstant això, de vegades un dispositiu MIDI tractarà els canals MIDI d'una manera especial (pot ser que cada nota tingui un canal separat) i també pots voler connectar més d'un dispositiu MIDI al mateix temps. En configuracions més complicades, és possible que vulguis ser més selectiu sobre quin dispositiu MIDI rep quin missatge(s) i en quin canal.

Podem especificar a quin dispositiu volem enviar a través de l'opció `port:`, indicant el nom del dispositiu tal i com es mostra a les preferències:

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

També podem especificar a quin canal enviar-lo mitjançant l'opció `channel:` (utilitzant un valor en l'interval 1-16):

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

Per descomptat, també podem especificar-los tots dos alhora per enviar a un dispositiu concret en un canal concret:

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

Estudi MIDI

Finalment, una cosa molt divertida és connectar la sortida d'àudio del teu sintetitzador MIDI a una de les entrades d'àudio de la teva targeta de so. Així, pots controlar el teu sintetitzador amb codi mitjançant les fns `midi_*` i també manipular l'àudio mitjançant `live_audio` i FX:

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

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

(La fn `midi` es pot utilitzar com a drecera útil per enviar esdeveniments d'activació i desactivació de notes amb una sola ordre. Consulta'n la documentació per obtenir-ne més informació).

12. OSC

A més de MIDI, una altra forma d'obtenir informació dins i fora de Sonic Pi és a través de la xarxa utilitzant un senzill protocol anomenat *OSC – Open Sound Control*. Això et permetrà enviar missatges a i des de programes externs (tant executats al teu ordinador com en ordinadors externs) cosa que amplia el potencial de control molt més enllà de MIDI que té limitacions a causa del seu disseny dels anys 80.

Per exemple, pots escriure un programa en un altre llenguatge de programació que envia i rep OSC (hi ha biblioteques OSC per a gairebé tots els llenguatges comuns) i treballar directament amb Sonic Pi. L'ús que se li pot donar a això només és limitat per la imaginació.

12.1. Rebre OSC

Per defecte, quan Sonic Pi es llança, escolta el port 4560 per als missatges *OSC* entrants dels programes al mateix ordinador. Això significa que sense cap configuració, pots enviar a Sonic Pi un missatge OSC i es mostrarà al registre de *cue* igual que els missatges MIDI entrants. Això també significa que qualsevol missatge OSC entrant també s'afegeix automàticament al *Time State*, el que significa que també pots utilitzar `get` i `sync` per treballar amb les dades entrants - igual que amb MIDI i la sincronització de `live_loops` - ves a les seccions 5.7 i 10.2 per recapitular com funciona això.

Un oient bàsic d'OSC

Construïm un oient bàsic d'*OSC*:

```
live_loop :foo do
  use_real_time
  a, b, c = sync "/osc*/trigger/prophet"
  synth :prophet, note: a, cutoff: b, sustain: c
end
```

En aquest exemple hem descrit una ruta *OSC* `"/osc*/trigger/prophet"` amb la qual estem sincronitzant. Pot ser qualsevol camí OSC vàlid (totes les lletres i números són compatibles i el / s'utilitza com en una URL per dividir el camí en diverses paraules). Sonic Pi afegeix el prefix `/osc` en tots els missatges OSC entrants, de manera que hem d'enviar un missatge OSC amb el camí `/trigger/prophet` perquè la nostra sincronització deixi de bloquejar-se i s'activi el sintetitzador *prophet*.

Enviar OSC a Sonic Pi

Podem enviar *OSC* a Sonic Pi des de qualsevol llenguatge de programació que tingui una biblioteca *OSC*. Per exemple, si estem enviant *OSC* des de *Python*, podríem fer alguna cosa com aquesta:

```
from pythonosc import osc_message_builder
from pythonosc import udp_client

sender = udp_client.SimpleUDPClient('127.0.0.1', 4560)
sender.send_message('/trigger/prophet', [70, 100, 8])
```

O, si estem enviant *OSC* des de *Clojure*, podríem fer alguna cosa com aquesta des del *REPL*:

```
(use 'overtone.core)

(def c (osc-client "127.0.0.1" 4560))
(osc-send c "/trigger/prophet" 70 100 8)
```

Rebre des de màquines externes

Per raons de seguretat, per defecte, Sonic Pi no permet que màquines remotes t'enviïn missatges *OSC*. Tanmateix, pots habilitar el suport per a màquines remotes a *Preferències->IO->OSC en xarxa->Permetre OSC* des d'altres ordinadors. Quan hagis habilitat això, podràs rebre missatges OSC des de qualsevol ordinador de la teva xarxa. Normalment, la màquina que envia els missatges necessitarà conèixer la seva adreça IP (un identificador únic per al teu ordinador a la seva xarxa - una cosa així com un número de telèfon o una adreça de correu electrònic). Pots saber l'adreça IP del teu ordinador mirant la secció *IO* del panell de preferències. (Si la teva màquina té més d'una adreça IP, en passar el ratolí per sobre de l'adreça llistada apareixerà una llista de totes les adreces conegudes).

Tingues en compte que alguns programes com *TouchOSC* per a iPhone i Android suporten l'enviament d'*OSC* com una característica estàndard. Per tant, un cop escoltis les màquines remotes i coneguis la teva adreça IP, podràs començar a enviar missatges a l'instant des d'aplicacions com *TouchOSC*, que et permeten construir els teus propis controls tàctils personalitzats amb lliscants, botons, dials, etc. Això et pot proporcionar una enorme gamma d'opcions d'entrada.

12.2 Enviar OSC

A més de rebre OSC i treballar-hi amb el *Time State*, també podem enviar missatges *OSC* a temps amb la nostra música (igual que podem fer-ho amb els missatges MIDI).

Només hem de saber a quin port i adreça IP ho estem enviant. Provem-ho:

```
use_osc "localhost", 4560
osc "/hello/world"
```

Si executes el codi anterior, notaràs que el propi Sonic Pi està enviant un missatge *OSC*! Això es deu al fet que hem establert l'adreça IP a la màquina actual i el port a l'*OSC* predeterminat al port. Això és bàsicament el mateix que enviar-te una carta a tu mateix: el paquet *OSC* es crea, surt de Sonic Pi, arriba a la pila de xarxa del sistema operatiu que després encamina el paquet de tornada a Sonic Pi i després es rep com a missatge *OSC* estàndard i és visible al *cue logger* com a missatge entrant `/osc:127.0.0.1:4560/hello/world`. (Observa com Sonic Pi prefixa automàticament tots els missatges *OSC* entrants amb `/osc` i després el nom d'amfitrió i el port del remitent.)

Enviar OSC a altres programes

Per descomptat, enviar-nos missatges *OSC* a nosaltres mateixos pot ser divertit, però no és gaire útil. El benefici real comença quan enviem missatges a altres programes:

```
use_osc "localhost", 123456
osc "/hello/world"
```

En aquest cas estem assumint que hi ha un altre programa a la mateixa màquina escoltant el port 123456. Si hi és, aquest rebrà un missatge "`/hello/world` *OSC* amb el qual podrà fer el que vulgui.

Si el nostre programa s'està executant en una altra màquina, necessitem conèixer la seva adreça IP, que utilitzarem en lloc de "`localhost`":

```
use_osc "192.168.10.23", 123456
osc "/hello/world"
```

Ara podem enviar missatges OSC a qualsevol dispositiu que ens pugui arribar a través de les nostres xarxes locals i fins i tot d'Internet!

13. ÀUDIO MULTICANAL

Fins ara, pel que fa a la producció de so, hem explorat l'activació de sintetitzadors i els sons enregistrats a través de les fns `play`, `synth` i `sample`. Aquests han generat àudio que s'ha reproduït a través del nostre sistema d'altaveus estèreo. Tot i això, molts ordinadors també tenen la capacitat d'introduir so, per exemple a través d'un micròfon, a banda de la capacitat d'enviar so a més de dos altaveus. Sovint, aquesta capacitat es fa possible mitjançant l'ús d'una targeta de so externa - disponibles per a totes les plataformes. En aquesta secció del tutorial farem una ullada a com podem aprofitar aquestes targetes de so externes i treballar sense esforç amb múltiples canals d'àudio dins i fora de Sonic Pi.

13.1. Sound In

Una forma senzilla (i potser familiar) d'accedir a les entrades de so és fer servir el nostre amic synth especificant el sintetitzador `:sound_in`:

```
synth :sound_in
```

Funcionarà igual que qualsevol sintetitzador com `synth :dsaw` amb l'excepció que l'àudio generat es llegirà directament de la primera entrada de la targeta de so del teu sistema. En els ordinadors portàtils, acostuma a ser el micròfon incorporat, però si tens una targeta de so externa, pots connectar qualsevol entrada d'àudio a la primera entrada.

Augmentar la durada

Una cosa que pots notar és que, igual que el sintetitzador `:dsaw`, el sintetitzador `:sound_in` només dura 1 temps, ja que té una envolupant estàndard. Si vols mantenir-lo obert durant més temps, canvia la configuració de l'envolupant ADSR. Per exemple, això mantindrà el sintetitzador obert durant 8 temps abans de tancar la connexió: `synth :sound_in, sustain: 8`

Afegir efectes

Per descomptat, igual que qualsevol sintetitzador normal, pots afegir efectes fàcilment amb el bloc FX:

```
with_fx :reverb do
  with_fx :distortion do
    synth :sound_in, sustain: 8
  end
end
```

Si has connectat una guitarra a la primera entrada, hauries de poder escoltar-la amb distorsió i reverberació fins que el sintetitzador acabi com s'espera.

Pots utilitzar el sintetitzador `:sound_in` tantes vegades com vulguis de forma simultània (com faries amb qualsevol sintetitzador normal). Per exemple, el següent reproduirà dos sintes `:sound_in` alhora: un amb distorsió i un altre amb reverberació:

```
with_fx :distortion do
  synth :sound_in, sustain: 8
end

with_fx :reverb do
  synth :sound_in, sustain: 8
end
```

Multiples entrades

Pots seleccionar quina entrada d'àudio vols reproduir amb l'*opt* `input:`. També pots especificar una entrada estèreo (dues entrades consecutives) utilitzant el sintetitzador `:sound_in_stereo`. Per exemple, si tens una targeta de so amb almenys tres entrades, pots tractar les dues primeres com un flux estèreo i afegir distorsió i la tercera com un flux mico i afegir reverberació amb el codi següent:

```
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
```

Problemes potencials

Tot i això, encara que es tracta d'una tècnica útil, aquest enfocament té un parell de limitacions. En primer lloc, només funciona durant una durada específica (atès que té una envolupant ADSR) i, en segon lloc, no hi ha manera de canviar els FX una vegada que el sintetitzador s'ha disparat. Ambdues coses són peticions típiques quan es treballa amb fonts d'àudio externes com micròfons, guitarres i sintetitzadors externs. Per tant, farem una ullada a la solució de Sonic Pi al problema de manipular un flux (potencialment) infinit d'entrada d'àudio en viu: `live_audio`.

13.2. Àudio en viu

El sintetitzador `:sound_in` explicat a la secció anterior proporciona un mètode molt flexible i familiar per treballar amb àudio d'entrada. Tot i així, com també s'ha discutit, dóna alguns problemes quan es treballa amb una sola entrada d'àudio com pot ser un sol instrument (una veu o una guitarra). Amb diferència, el millor enfocament per treballar amb un únic flux continu d'àudio és utilitzar `live_audio`.

Una entrada d'àudio amb nom

`live_audio` comparteix un parell de restriccions de disseny amb `live_loop` (d'aquí la semblança en el nom). En primer lloc, ha de tenir un nom únic i, en segon lloc, només hi pot haver un flux `live_audio` amb aquest nom en qualsevol moment. Fem-hi una ullada:

```
live_audio :foo
```

Aquest codi actuarà de manera similar a `synth :sound_in` amb algunes diferències clau: s'executa indefinidament (fins que l'aturis explícitament) i pots moure'l a nous contextos FX de forma dinàmica.

Treballar amb FX

A l'activació inicial, `live_audio` funciona exactament com s'espera que funcioni amb FX. Per exemple, per iniciar un flux d'àudio en directe amb reverberació afegida, simplement utilitza un bloc FX `:reverb`:

```
with_fx :reverb do
  live_audio :foo
end
```

Tanmateix, atès que `live_audio` s'executa indefinidament (almenys fins que l'aturis) seria bastant limitant si, com els sintetitzadors típics, l'àudio en viu estigués lligat

dins del reverb FX durant tota la seva existència. Per sort, aquest no és el cas i ha estat dissenyat perquè sigui fàcil moure's entre diferents FX. Ho provarem. Executa el codi anterior per escoltar l'àudio en viu que ve directament de la primera entrada de la targeta de so. Tingues en compte que si estàs utilitzant un portàtil, això sortirà normalment del teu micròfon incorporat, per la qual cosa es recomana fer servir auriculars per evitar la retroalimentació.

Ara, mentre segueixes escoltant l'àudio en directe des de la targeta de so amb reverberació, canvia el codi pel següent:

```
with_fx :echo do
  live_audio :foo
end
```

Ara, prem *Run*, i immediatament escoltaràs l'àudio reproduït a través del ressò FX i ja no a través de la reverberació. Si els vols tots dos, simplement edita el codi de nou i torna a prémer *Run*:

```
with_fx :reverb do
  with_fx :echo do
    live_audio :foo
  end
end
```

És important assenyalar que es pot cridar a `live_audio :foo` des de qualsevol fil o bucle en viu i es mourà el sintetitzador d'àudio en viu al context FX actual d'aquest fil. Per tant, és fàcil tenir diversos bucles en viu cridant a `live_audio :foo` en diferents moments, cosa que fa que el context FX s'intercanviï automàticament, amb resultats interessants.

Aturar l'àudio en viu

A diferència dels sintetitzadors estàndard, com `live_audio` no té envolupant, seguirà executant-se indefinidament (fins i tot si esborres el codi, igual que una funció es manté en la memòria si esborres el codi a l'editor). Per aturar-lo, cal utilitzar l'argument (*arg* per abreviar) `:stop`:

```
live_audio :foo, :stop
```

Es pot reiniciar fàcilment cridant-lo de nou sense l'*arg* `:stop`:

```
live_audio :foo
```

A més, tots els sintetitzadors d'àudio en viu que s'estan executant s'aturen quan es prem el botó global *Stop* (igual que la resta de sintetitzadors i efectes que s'estan executant).

Entrada estèreo

Pel que fa als canals d'àudio, per defecte `live_audio` actua de forma similar al sintetitzador `:sound_in` en el sentit que pren un únic flux d'entrada d'àudio mono i el converteix en un flux estèreo utilitzant la panoràmica especificada. No obstant això, igual que amb `:sound_in_stereo` també és possible dir a `live_audio` que llegeixi dues entrades d'àudio consecutives i les tracti com els canals esquerre i dret directament. Això s'aconsegueix mitjançant l'opció `:stereo`. Per exemple, per tractar l'entrada 2 com el senyal esquerre i l'entrada 3 com el senyal dret, cal configurar l'opt `input:` a 2 i activar el mode estèreo de la següent manera:

```
live_audio :foo, stereo: true, input: 2
```

Tingues en compte que un cop hagis iniciat un flux d'àudio en directe en mode estèreo, no podràs canviar-lo a mono sense parar i tornar a començar. De la mateixa manera, si l'inicies en el mode mono per defecte, no pots canviar a estèreo sense iniciar i aturar el flux.

13.3 Sound Out

Fins ara en aquesta secció hem vist com obtenir múltiples fluxos d'àudio a Sonic Pi - ja sigui mitjançant l'ús del sintetitzador `:sound_in` o a través del potent sistema `live_audio`. A més de treballar amb múltiples fluxos d'àudio d'entrada, Sonic Pi també pot emetre múltiples fluxos d'àudio. Això s'aconsegueix mitjançant el FX `:sound_out`

Contextos de sortida

Recapitulem ràpidament com els sintetitzadors i efectes de Sonic Pi emeten el seu àudio al seu context FX actual.

Per exemple, considera el següent:

```
with_fx :reverb do    # C
  with_fx :echo do    # B
    sample :bd_haus   # A
  end
end
```

La manera més senzilla d'entendre el que passa amb el flux d'àudio és començar pel context d'àudio més intern i treballar cap a fora. En aquest cas, el context més intern està etiquetat com a `A` i és la mostra `:bd_haus` que s'està disparant. L'àudio va directament al seu context que és `B` - el FX `:echo`. Aquest afegeix ressò a l'àudio entrant i l'envia al seu context que és `C` - el FX `:reverb`. Aquest afegeix reverberació a l'àudio entrant i l'envia al seu context, que és l'últim nivell - els altaveus esquerre i dret (sortides 1 i 2 de la targeta d'àudio). L'àudio flueix cap a l'exterior amb una senyal estèreo en tot moment.

Sound Out FX

El comportament anterior és vàlid per a tots els sintetitzadors (incloent-hi `live_audio`) i la majoria dels FX amb l'excepció de `:sound_out`. El FX `:sound_out` fa dues coses. En primer lloc, emet el seu àudio al context extern com s'ha descrit anteriorment. En segon lloc, també emet el seu àudio directament a una sortida de la targeta de so. Fem-hi una ullada:

```
with_fx :reverb do      # C
  with_fx :sound_out, output: 3 do # B
    sample :bd_haus     # A
  end
end
```

En aquest exemple, la nostra mostra `:bd_haus` envia el teu àudio al seu context extern que és el FX `:sound_out`. Aquest, al seu torn, emet el seu àudio al seu context extern, el FX `:reverb` (com era d'esperar). Tanmateix, també envia una mescla mono a la tercera sortida de la targeta de so del sistema. L'àudio generat dins de `:sound_out` té, per tant, dues destinacions: el FX `:reverb` i la sortida 3 de la targeta de so.

Mono i Stereo out

Com hem vist, per defecte, el FX `:sound_out` dóna sortida a una mescla mono de l'entrada estèreo a un canal específic, a més de passar l'alimentació estèreo al seu context exterior (com cal esperar). Si la sortida d'una mescla mono no és precisament el que vols, hi ha diverses opcions alternatives. En primer lloc, utilitzant l'*opt* `mode:` pots triar que el senyal només de l'entrada esquerra o només de la dreta a la targeta d'àudio. O pots utilitzar la funció `:sound_out_stereo` FX per donar sortida a dues sortides consecutives de la targeta de so. Consulta la documentació de la funció per obtenir més informació i exemples.

Direct Out

Com també hem vist, el comportament per defecte de `:sound_out` i `:sound_out_stereo` és enviar l'àudio tant al seu context extern (com és típic de tots els FX) com a la sortida especificada a la targeta de so. No obstant això, ocasionalment pots voler enviar-lo només a la sortida de la targeta de so i no al context extern (i per tant no tenir cap possibilitat que el so sigui mesclat i enviat als canals de sortida estàndard 1 i 2). Això és possible utilitzant l'opció FX estàndard :amp que opera sobre l'àudio després que FX hagi pogut manipular l'àudio:

```
with_fx :sound_out, output: 3, amp: 0 do # B
  sample :loop_amen                      # A
end
```

A l'exemple anterior, la mostra `:loop_amen` s'envia al context extern, el FX `:sound_out`. Aquest envia una mescla mono a la sortida 3 de la targeta d'àudio i multiplica l'àudio per 0, cosa que essencialment el silencia. És aquest senyal silenciat el que s'envia al context extern de `:sound_out`, que és la sortida estàndard. Per tant, amb aquest codi, els canals de sortida per defecte no rebran cap àudio, i el canal 3 rebrà una mescla mono del trencament del tambor amen.

14. CONCLUSIONS

Amb això acaba el tutorial d'introducció a Sonic Pi. Espero que hagis après alguna cosa pel camí. No et preocupis si sents que no ho has entès tot - simplement juga i diverteix-te i aprendràs les coses al teu temps. No dubtis a tornar a consultar quan tinguis una pregunta que pugui estar inclosa en alguna de les seccions.

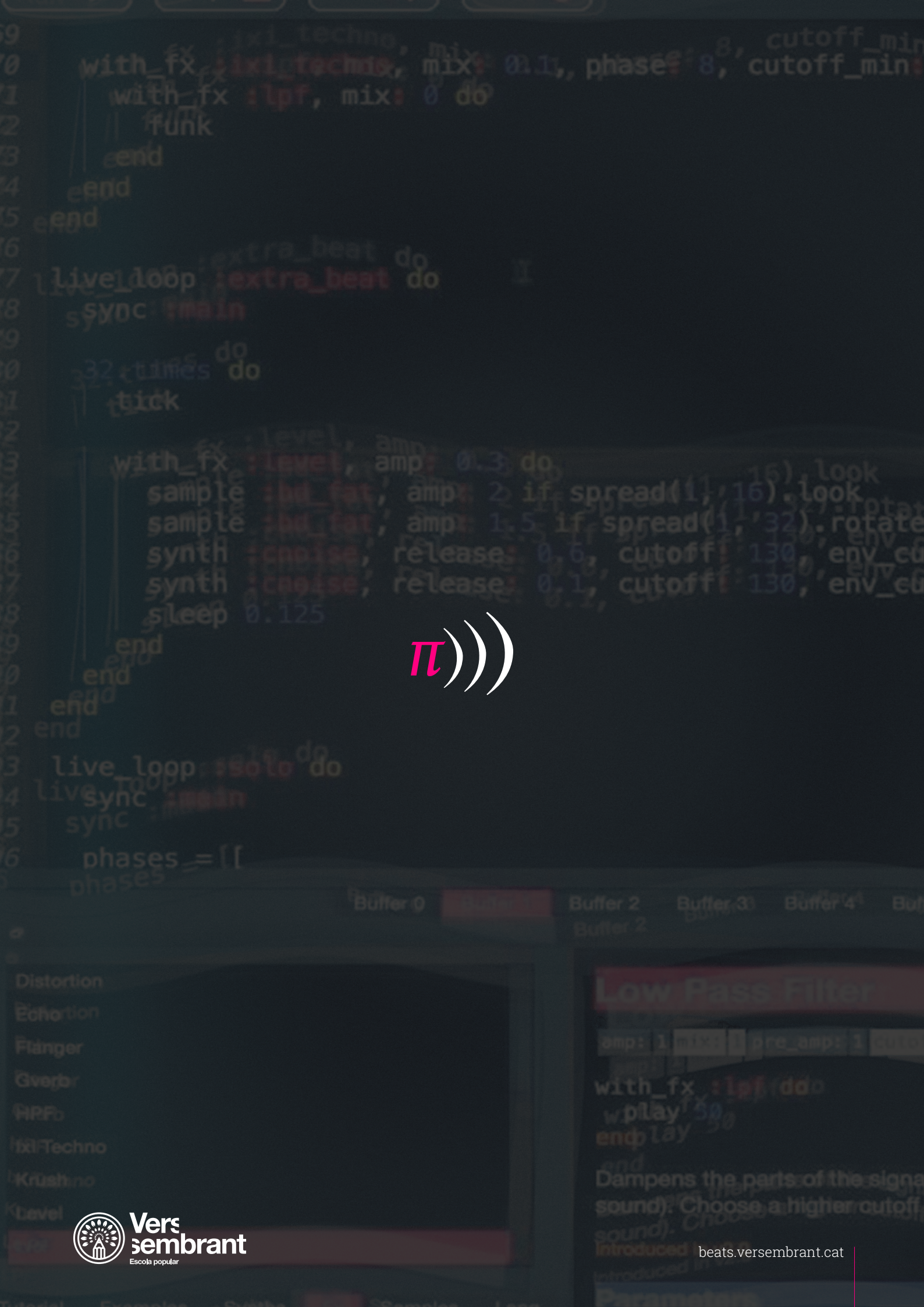
Si tens alguna pregunta que no s'hagi tractat al tutorial, entra als fòrums de la comunitat de Sonic Pi i fes-la allà. Trobaràs algú amable i disposat a donar-te un cop de mà.

Finalment, també et convido a fer una ullada en profunditat a la resta de la documentació d'aquest sistema d'ajuda. Hi ha una sèrie de característiques que no han estat explicades en aquest tutorial i que estan esperant que les descobreixis.

Així que juga, diverteix-te, comparteix el teu codi, actua per als teus amics, mostra les teves pantalles i recorda:

No hi ha errors, només oportunitats.

Sam Aaron



Versembrant
Escola popular

beats.versembrant.cat