Beats basteln wie die Großen

Mit Sonic Pi eine Rhythmusspur programmieren

Wie man der freien Musik-Software Sonic Pi erste Klänge entlockt, haben Sie in c't 12/17 erfahren. Im zweiten Teil geht es ans Eingemachte: Sie coden einen elektronischen Beat, der mit Plattenknistern und einer satten Bassdrum punktet.

Von Pit Noack

Das Musizieren mit Sonic Pi ist kinderleicht – erste Rhythmen, Melodien und Soundexperimente lassen sich im Handumdrehen programmieren. Wer aber ganze Songs schreiben oder als Live-Coder im Club einheizen möchte, muss viel ausprobieren und üben – da unterscheidet sich die Software nicht von handgespielten Musikinstrumenten.

Der erste Teil dieser Einführung in c't 12/17 hat einzelne Funktionen in kurzen, abgeschlossenen Beispielen vorgeführt [1]. Wenn aber der Ehrgeiz erwacht und das Ziel ein vollständig selbst gestalteter Rhythmus ist, braucht man schon deutlich mehr Durchhaltevermögen. Genau hier steigt Teil 2 tiefer in die Materie ein und beschreibt, wie man auf Grundlage des Gelernten einen elektronischen Beat programmiert.

Das vorgestellte Beispielprogramm besteht aus vier Elementen: einem Plattenknistern, einer Hihat, einer Snare- und einer Bassdrum. Hihat und Bassdrum basieren nicht auf Samples und erfordern etwas mehr Programmieraufwand. Sie werden in mehreren Schritten von Grund auf neu entwickelt.

Den fertigen Code finden Sie im Listing `Beat` ganz am Ende des Artikels. Schreibt man diesen in den Buffer-Bereich von Sonic Pi und drückt anschließend den Run-Knopf, spielt die Software den Beat ab.

Der vollständige Code steht – ergänzt um zwei zusätzliche Elemente – auch als Download zur Verfügung (ct.de/yuz5).

In den Buffer-Bereich von Sonic Pi schreibt man den Code. Wenn es mal hakt, öffnen Sie per Druck auf den Help-Knopf den Hilfebereich.

Der c't-Tipp für Kinder und Eltern

Beat-Programmierung mit Sonic Pi

- Sonic Pi, PC mit Windows, macOS oder Linux sowie Kopfhörer oder Lautsprecher
- sicherer Umgang mit Tastatur und Maus, Englischkenntnisse von Vorteil, Lektüre von Teil 1 dieser Anleitung empfohlen
- Das komplette Programm tippt man in rund einer Stunde ab.
- Jugendliche ab circa 16 Jahren
- keine

Hinter diesem Link finden Sie zudem eine Übersicht aller Einzelschritte der Programmentwicklung.

Das Code-Beispiel dürfen Sie nach Belieben weiterentwickeln und veröffentlichen. Die Redaktion freut sich über eingesandte Remixes.

Takt und Tempo

Im ersten Schritt geben Sie das Tempo vor. Der Beispiel-Beat schlägt im 4/4-Takt bei moderaten 100 Schlägen pro Minute (bpm = beats per minute) – er soll also weder zu hektisch, noch zu schleppend sein. Das Tempo legen Sie fest, indem Sie in die erste Zeile `use_bpm: 100` schreiben.

Knisternde Atmosphäre

Es ist in modernen Musikproduktionen üblich, einem Mix dezente Störgeräusche und Unregelmäßigkeiten wie Rauschen oder Knistern beizufügen. So wirkt ein sonst zu perfektes digitales Klangbild organischer. Auch Sonic Pi bringt solche Geräusche mit: Unter dem Namen `vinyl_hiss` finden Sie in der Klangbibliothek das Laufgeräusch einer verkratzten Schallplatte.

Im Live Loop `knister_schleife` (siehe Listing `Beat` Zeile 5–8) spielt `sample :vinyl_hiss, amp: 2` das Knistern mit Lautstärke 2 ab. Damit der Loop ein lückenloses Geräusch produziert, füttern Sie `sleep` mit der exakten Dauer des Samples. Diese Zeitspanne ermittelt die Funktion `sample_duration`. Sie erwartet als Argument den Namen des Samples, dessen Länge gesucht wird. Das Ergebnis wird an `sleep` übergeben. Jetzt läuft der Loop in Endlosschleife – das erste Element des Beats ist komplett.

Teilzeitjob: die Hihat

Das Knistern ist fertig, nun geht es an die eigentliche Drum-Programmierung. Aus einem Rauschsignal und einem Filter bastelt man flugs eine eigene Hihat (siehe `Beat` Zeile 11–16). `use_synth :pnoise` wählt einen Rauschgenerator als Synth und `play release: 0.01` spielt eine Note mit kurzer Ausklingzeit. Normalerweise erwartet `play` einen Wert für die Tonhöhe. Der Rauschgenerator `pnoise` braucht diese Information aber nicht, da er unabhängig von der Tonhöhe ein ganzes Rauschspektrum erzeugt. Dieses enthält tiefe, mittlere und hohe Klangbestandteile. Damit die Hihat am Ende das typische „Tschik" produziert, bearbeiten Sie den Rauschimpuls mit der Funktion `with_fx` mit einem Hochpassfilter (`hpf` = high pass filter). Dieser dämpft die Frequenzen unterhalb einer Trennfrequenz (`cutoff`). Je höher der Wert von `cutoff`, desto dünner und spitzer der Klang. Die Frequenz wird indirekt über Notenwerte bestimmt: Die 120 verweist auf die hundertzwanzigste Taste einer Klaviatur. Zuletzt verpacken Sie diesen Code mit `define :hihat` in eine Funktion.

```
01  # Hihat-Spieler 1
02  4.times do
03    hihat
04    sleep 1.0 / 4
05  end
```

Dieser Code spielt vier Hihat-Schläge, die zusammen einen Vierteltakt dauern.

Im folgenden Schritt besteht das Etappenziel darin, eine Zeiteinheit (in diesem Fall ein Vierteltakt) in vier Hihat-Schläge aufzuteilen. Die 4 in `4.times` des Code-Beispiels `Hihat-Spieler 1` steht für die Anzahl der Wiederholungen des Codes zwischen `do` und `end` und damit auch für die Anzahl der Hihat-Schläge. `sleep` setzt die Pause zwischen den einzelnen Hihat-Schlägen auf `1.0/4`, das ergibt `0.25`, also einen Sechzehnteltakt pro Schlag. Im Ergebnis dauert das Ausführen dieses Codes genau einen Vierteltakt, weil $4 \times 0.25 = 1$. Probieren Sie, was passiert, wenn Sie die 4 in der zweiten und in der vierten Zeile durch einen anderen Wert ersetzen.

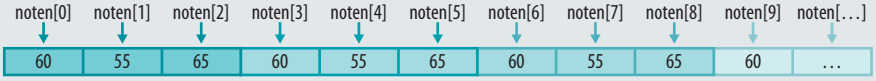
Der Code `Hihat-Spieler 2` spielt die Hihat in einem Live Loop. So können Sie den Code ändern, während er läuft; ein Druck auf den Run-Knopf sorgt für die Ausführung des modifizierten Codes, ohne dass der Beat unterbrochen werden muss. Für eine abwechslungsreiche Aufteilung trägt man den Wert für `sleep` nicht

```
01  # Hihat-Spieler 2
02  live_loop :hihat_schleife do
03    aufteilung = 4
04    aufteilung.times do
05      hihat
06      sleep 1.0 / aufteilung
07    end
08  end
```

Die Hihat läuft in einem Live Loop, den man verändern kann, während er spielt.

Eine Wertefolge im Ring

Dieser Ring enthält nur drei Werte, kann aber so behandelt werden, als sei er endlos lang.

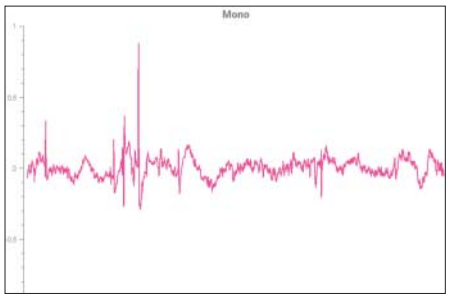


mehr manuell ein, sondern lässt ihn berechnen: Die Variable `aufteilung` in Zeile 3 legt die Anzahl der Hihat-Schläge pro Schleifendurchgang fest. Gleichzeitig steuert `aufteilung` in Zeile 6 indirekt die Dauer von `sleep`, indem 1.0 durch `aufteilung` dividiert wird.

Experimentieren Sie mit unterschiedlichen Werten für die Aufteilung. Probieren Sie zum Beispiel die Zahlen 2, 4, 6, 8. Ersetzt man Zeile 3 durch `aufteilung = [2, 4, 6, 8].choose`, wird bei jedem Schleifendurchgang einer dieser vier Werte zufällig ausgewählt.

Der Code `Hihat-Spieler 2` ist schon recht musikalisch: Ein festgelegter Zeitabschnitt (ein Vierteltakt) wird dynamisch (also abhängig von der Variablen `aufteilung`) zerlegt. Die Dauer dieses Zeitabschnitts und die Dauer eines Schleifendurchgangs bleiben gleich, egal, welche Zahl für `aufteilung` eingesetzt wird. Mathefreaks sehen sofort: Ganz gleich, welchen Wert (außer 0) `aufteilung` annimmt, `aufteilung` × 1 / `aufteilung` ergibt in jedem Fall 1.

Wäre es nicht toll, wenn man im Live Loop diese Aufteilung wie ein Schlagzeuger nach einem bestimmten Muster wechseln könnte? Das geht in Sonic Pi mit einem sogenannten Ring. Ein Ring ist ein Behälter für Wertefolgen, der für musikalische Zwecke eine besonders nützliche Eigenschaft hat: Er verhält sich so, als sei er unbegrenzt. Angenommen, ein Ring `noten` enthält die Wertefolge 60, 55 und 65. dann liefert noten[0] den Wert 60, noten[1] gibt



Sonic Pi zeigt auch Wellenformen an, etwa vom Plattenknistern.

55 zurück, noten[2] 65. Was passiert, wenn man mit noten[3] den nicht-existenten vierten Wert abfragt? Man landet wieder bei dem ersten Wert, also 60.

Im fertigen Code für die `hihat_schleife` (siehe `Beat` Zeile 18–24) wird in Zeile 19 ein Ring erzeugt und unter dem Namen `aufteilungen` gespeichert. Er enthält acht Werte. Der Ausdruck `aufteilungen[tick]` in Zeile 20 sorgt dafür, dass bei jedem Schleifendurchgang der nächste Wert des Ringes abgerufen wird. `tick` arbeitet wie ein Zähler: Beim ersten Aufruf gibt die Funktion 0 zurück, beim zweiten die 1, beim dritten die 2 und so weiter. `ring` und `tick` greifen hier ineinander wie zwei Zahnräder in einem gut geölten Getriebe.

In Zeile 22 benötigt man noch einmal den aktuellen Teiler. Da in Zeile 20 der Wert des `tick`-Zählers bereits erhöht wurde, nutzen Sie `look`, um den Wert des Zählers abzurufen, ohne ihn zu erhöhen.

Leises Klopfen: die Snare

Die Hihat ist fertig, es folgt eine sparsam eingesetzte Snaredrum – siehe `Beat` Zeile 27–33. Der musikalische Sinn der Trommel erschließt sich erst im Zusammenspiel mit den anderen Elementen des Beats. In Zeile 30 spielt die Funktion `sample` einen unter dem Namen `sn_dub` gespeicherten Snare-Schlag ab. `sustain: 0` und `release: 0.05` bedingen einen sehr kurzen Schlag. Für ein ausgewogenes Klangbild wird die Snare ähnlich der Hihat durch einen Filter geschickt. Diesmal ist es ein Tiefpassfilter (`lpf` = low pass filter), der den Klang oberhalb der Tonhöhe 100 dämpft.

Das Timing der Snare-Schläge steuern zwei Ringe. Übrigens hat jeder Live Loop seine eigene, unabhängige Tick-Variable. Beim ersten Schleifendurchlauf passiert Folgendes: Vor dem Snare-Schlag wartet Sonic Pi 2,5 Vierteltakte, danach 1,5. Ein Schleifendurchlauf dauert also insgesamt einen ganzen Takt. Beim zweiten Durchgang beträgt die Wartedauer vorher drei, hinterher einen Vierteltakt. Im Ergebnis dauert also auch der zweite Durchgang einen Takt. Der dritte Durchgang gleicht wieder dem ersten, und so geht es immer weiter.

Tiefergelegt: die Bassdrum

Zu guter Letzt kommt der komplexeste Bestandteil unseres Beats, die Bassdrum. Ihr Sound orientiert sich an aktuellen Produktionen der Stilrichtungen Trap oder Drum and Bass: Ein dezentes Knacken sorgt für den nötigen Kick und die rhythmische Präzision, ein sehr tiefer, gefilterter Sinuston mit einer langen Ausklingphase macht den Sound voluminös und raumfüllend. Eine leichte Verzerrung reichert den tieffrequenten Bereich der Bassdrum um Obertöne an. Sie merken, auch hier greift das Beispiel nicht auf ein vorgefertigtes Sample zurück – die große Trommel wird vielmehr in mehreren kleinen Schritten programmiert. Kleiner Tipp: Alle Nuancen der programmierten Bassdrum lassen sich am besten mit guten Boxen oder Kopfhörern wahrnehmen.

Jeder Schlag der Bassdrum hat eine eigene Tonhöhe, sodass sie nicht nur ein

```
01  # Bassdrum-Funktion 1
02  define :bassdrum do |note_1|
03    use_synth :sine
04    with_fx :hpf, cutoff: 100 do
05      play note_1 + 24, amp: 40, ↵
↳release: 0.01
06    end
07  end
```

Schritt eins beim Bassdrum-basteln: ein hohes Knacken sorgt für den nötigen Kick und rhythmische Präzision.

```
01  # Bassdrum-Funktion 2
02  define :bassdrum do ↵
↳|note_1, dauer|
03    use_synth :sine
04    with_fx :hpf, cutoff: 100 do
05      play note_1 + 24, amp: 40, ↵
↳release: 0.01
06    end
07    play note_1, amp: 10, ↵
↳release: dauer
08    sleep dauer
09  end
```

In Zeile 7 wird der tieffrequente Anteil der Bassdrum hinzugefügt.

rhythmisches Element ist, sondern auch als harmonischer Bestandteil fungiert. Darüber hinaus kann die Bassdrum von der anfänglichen Tonhöhe zu einer zweiten gleiten – das macht beeindruckende Klangeffekte möglich.

Im ersten Schritt steht das Knacken auf dem Plan – siehe `Bassdrum-Funktion 1` auf Seite 134. Dafür statten Sie die Bassdrum mit dem Argument `note_1` aus. Das ist die anfängliche Tonhöhe. `use_synth :sine` wählt einen Sinusgenerator als Synth. Der Synth wird mit einer Tonhöhe gespielt, die zwei Oktaven (+ 24 Halbtöne) über dem Wert von `note_1` liegt. `release: 0.01` sorgt für eine kurze Ausklingzeit. Um das gewünschte hohe Knacken zu erreichen, wird dieser kurze Impuls noch durch einen Hochpassfilter geschickt.

Im nächsten Schritt wird der tieffrequente Anteil hinzugefügt, siehe `Bassdrum-Funktion 2`. Das zweite Argument bestimmt mit `release: dauer` die Länge der Ausklingphase und in Zeile 8 mit `sleep dauer` auch die Zeit, die nach dem Auslösen eines Schlages bis zur weiteren Ausführung des Codes gewartet wird. Das verhindert unbeabsichtigtes Überlappen von einzelnen Bassnoten, was zu ungewollten Störgeräuschen und Übersteuerungen führen kann.

```
01  # Bassdrum-Test 1
02  16.times do
03    bassdrum 32, 0.5
04    bassdrum 34, 0.5
05    bassdrum 37, 1
06    bassdrum 32, 2
07  end
```

Der Bassdrum-Test 1 spielt 16 Takte.

Machen Sie den `Bassdrum-Test 1`. Gut klingende Werte für `note_1` liegen im

```
01  # Bassdrum-Funktion 3
02  define :bassdrum do |note1, dauer|
03    use_synth :sine
04    with_fx :hpf, cutoff: 100 do
05      play note1 + 24, amp: ↵
↳40, release: 0.01
06    end
07    with_fx :distortion, distort: ↵
↳0.1, mix: 0.3 do
08      with_fx :lpf, cutoff: 26 do
09        with_fx :hpf, cutoff: 55 do
10          play note1, amp: 85, ↵
↳release: dauer, note_slide: dauer
11        end
12      end
13    end
14    sleep dauer
15  end
```

Drei Effekte sorgen dafür, dass die Bassdrum nicht zu dick tönt und sich klar im Mix positioniert.

Bild: Hartmut Gieselmann

Die Bassdrum unseres Beats kommt dem Sound von Hardware-Drummachines wie der Roland TR-8 klanglich recht nah.

schmalen Bereich zwischen etwa 32 und 38.

Die Bassdrum klingt schon ganz schick, aber noch etwas zu üppig, was Lautsprecher überlasten kann. Um einen ausgewogeneren Klang zu erreichen, der auch bei hohen Lautstärken satt und rund ankommt, wird der tieffrequente Teil der Bassdrum in der `Bassdrum-Funktion 3` durch drei Effekte geschickt: Ein Hochpassfilter dünnt die tiefen Frequenzen aus, ein Tiefpassfilter nimmt auch Klanganteile im oberen Teil weg. Auf diese Weise spielt die Bassdrum in einem eng abgesteckten Frequenzbereich (Zeilen 8 und 9). Zuletzt wird das Signal noch durch einen dezent eingestellten Verzerrer geschickt, der den Sound der Bassdrum etwas knuspriger macht (Zeile 7). Da die beiden Filter viel Lautstärke schlucken, erhöhen Sie diese per `amp` auf 85. Machen Sie erneut den `Bassdrum-Test 1` und achten Sie auf die Veränderungen des Klangs.

```
01  # Bassdrum-Test 2
02  16.times do
03    bassdrum 32, 1
04    bassdrum 32, 1, 38
05    bassdrum 32, 2, 10
06  end
```

Die Tonhöhe des zweiten Bassdrum-Schlages gleitet nach oben, die dritte nach unten.

```
01  # Bassdrum-Spieler 1
02  live_loop :bassdrum_schleife do
03    bassdrum 36, 1.5
04    bassdrum 36, 1.5
05    bassdrum 36, 1.0, ring␣
↳(10, 10, 10, 40)[tick]
06  end
```

Die Bassdrum-Schleife spielt drei Schläge. Die Tonhöhe des dritten Schlages gleitet, durch einen Ring gesteuert, bei jedem vierten Durchlauf nach oben, sonst nach unten.

Der Code in der nun fertigen Bassdrum-Funktion (siehe `Beat` Zeile 36–50) fügt den angekündigten Tonhöhenverlauf hinzu. Das dritte Argument `note_2` legt die zweite Tonhöhe fest. `note_2 = note_1` bestimmt ein sogenanntes „optionales Argument“: Wenn man es bei der Verwendung der Funktion weglässt, wird an seiner Stelle automatisch `note_1` eingesetzt. Die Bassdrum verhält sich dann ganz genau so wie in `Bassdrum-Funktion 3`, gleitet also nicht zu einer zweiten Tonhöhe, sondern behält die mit `note_1` festgelegte Tonhöhe bei.

Besonderes Augenmerk verdienen die Zeilen 44 und 45: `bass = play note_1` löst das Klangereignis aus und gibt ihm zugleich einen Namen. `note_slide: dauer` legt die Dauer fest, die eine Veränderung der Tonhöhe während des Abspielens brauchen wird. Diese Möglichkeit bietet `control`, das als erstes Argument den Namen des zu kontrollierenden Klangereignisses (hier: `bass` aus der vorherigen Zeile) erwartet. `note: note_2` sorgt dann für den gewünschten Tonhöhenverlauf.

Testen Sie die fertige Bassdrum mit dem Code `Bassdrum-Test 2`. Beim zweiten Schlag gleitet die Tonhöhe von 32 hoch auf 38, beim dritten nach unten auf 10, in einen Bereich, den Menschen nicht mehr hören können, den man aber bei der entsprechenden Lautstärke körperlich spürt und der die Wände zum Wackeln bringt. Das macht Spaß und freut auch die Nachbarn.

Der Code `Bassdrum-Spieler 1` erzeugt die `bassdrum_schleife` und spielt drei Schläge. Ein Schleifendurchgang dauert vier Vierteltakte, also einen ganzen Takt. Der letzte Schlag gleitet, durch einen Ring gesteuert, im ersten, zweiten und dritten Takt nach unten auf 10, beim vierten nach oben auf 40.

Als letzte Finesse wird bei jedem zehnten aus sechzehn Takten eine kleine Verzierung eingefügt. Zeile 54 im fertigen Code der `bassdrum_schleife` (siehe Listing Zeile 52–61) erzeugt mit `bools` einen Ring, der aus sogenannten „Wahrheitswerten“ besteht: 0 steht für `false` (falsch, unwahr), 1 für `true` (wahr). `tick` durchläuft diesen Ring, und liefert jeden sechszehnten Takt `true`, sonst `false`. Nur wenn `true` geliefert wird, werden die Zeilen 55 und 56 ausgeführt, sonst Zeile 58. Da Zeile 54 `tick` bereits einmal aufruft, muss es in Zeile 60 durch `look` ersetzt werden. Damit erhöht sich der tick-Zähler pro Schleifendurchgang nur einmal.

Wenn Ihnen die Entwicklung des Beats Spaß gemacht hat, haben Sie nun die Möglichkeit, ihn beliebig zu verändern und zu erweitern. Modifizieren Sie den Ring für die Hihat-Aufteilungen, indem Sie Werte ändern, hinzufügen oder löschen (siehe `Beat` in Zeile 19). Ersetzen Sie für die Ausklingzeit der Snaredrum und der Hihat die festen Werte mit `rand(0.01..0.05)` durch zufallsgenerierte Werte, um mehr Abwechslung ins Klanggeschehen zu bringen (Zeile 14 und 30). Geübte Sonic-Pi-Programmierer lassen Hihat, Snaredrum und Bassdrum nacheinander starten und zwischendurch pausieren – weder der Fantasie noch den Möglichkeiten sind hier Grenzen gesetzt. *(mre@ct.de)* c't

Literatur

[1] Pit Noack, Programmier deinen Song, Erste Schritte mit der Musik-Software Sonic Pi, c't 12/17, S. 144

Code-Beispiel, Sonic Pi, Maschinennah Creative Coding, Beispiel-MP3:
ct.de/yuz5

```
# Beat
use_bpm 100

# KNISTERN
live_loop :knister_schleife do
  sample :vinyl_hiss, amp: 2
  sleep sample_duration :vinyl_hiss
end

# HIHAT
define :hihat do
  use_synth :pnoise
  with_fx :hpf, cutoff: 120 do
    play release: 0.01, amp: 13
  end
end

live_loop :hihat_schleife do
  aufteilungen = ring 2, 4, 2, 2, 2, 2, 2, 6
  aufteilungen[tick].times do
    hihat
    sleep 1.0 / aufteilungen[look]
  end
end

# SNARE
live_loop :snare_schleife do
  sleep ring(2.5, 3)[tick]
  with_fx :lpf, cutoff: 100 do
    sample :sn_dub, sustain: 0, release: 0.05, amp: 3
  end
  sleep ring(1.5, 1)[look]
end

# BASSDRUM
define :bassdrum do |note1, dauer, note2 = note1|
  use_synth :sine
  with_fx :hpf, cutoff: 100 do
    play note1 + 24, amp: 40, release: 0.01
  end
  with_fx :distortion, distort: 0.1, mix: 0.3 do
    with_fx :lpf, cutoff: 26 do
      with_fx :hpf, cutoff: 55 do
        bass = play note1, amp: 85, ↵
↳release: dauer, note_slide: dauer
        control bass, note: note2
      end
    end
  end
  sleep dauer
end

live_loop :bassdrum_schleife do
  bassdrum 36, 1.5
  if bools(0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0)[tick]
    bassdrum 36, 0.5, 40
    bassdrum 38, 1, 10
  else
    bassdrum 36, 1.5
  end
  bassdrum 36, 1.0, ring(10, 10, 10, 40)[look]
end
```

Dies ist das Listing des kompletten Beats.