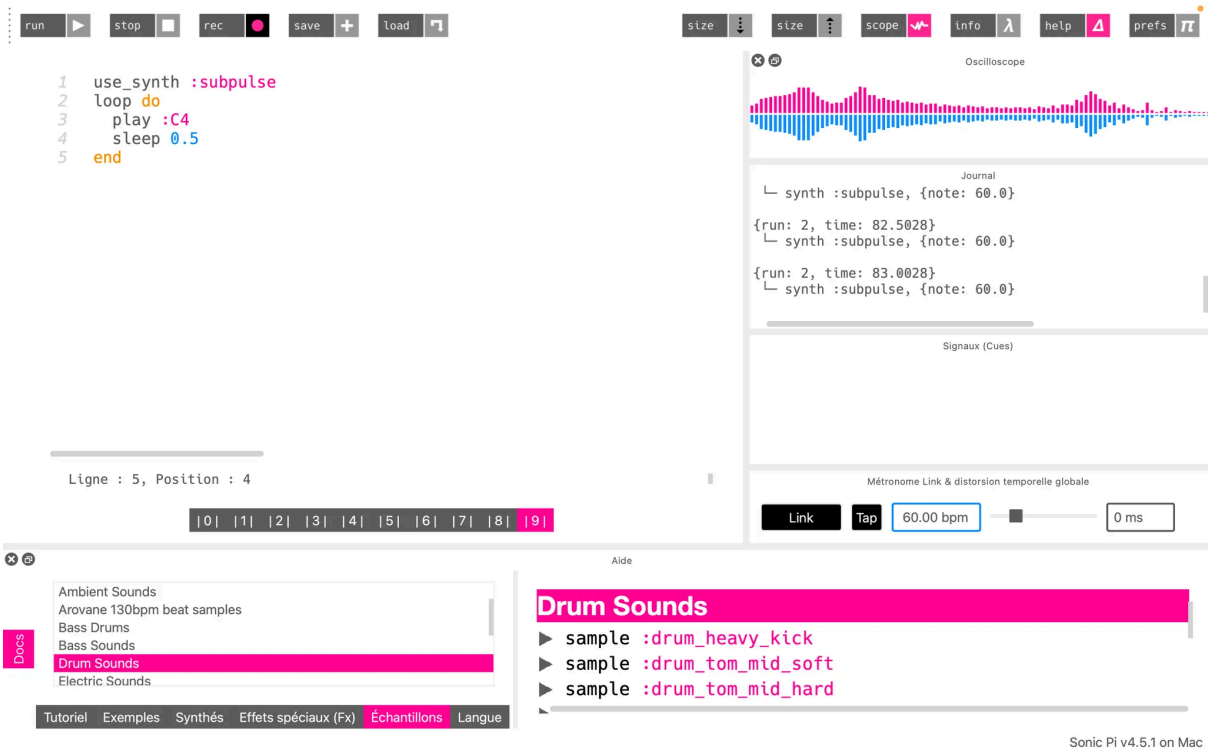
Sonic-Pi

Sonic-pi est un logiciel qui permet de faire de la musique en codant. En quelques lignes, on peut créer un rythme et accorder des sons pour en faire des chansons. C'est un moyen amusant pour apprendre à programmer.

Dans cet atelier, on va passer en revu ce qu'il est possible de faire avec sonic-pi, faire une reprise du titre "Harder Better Faster Stronger" des Daft Punk et pourquoi pas bidouiller et en créer d'autre. Ce qui est cool avec sonic-pi, c'est qu'on ne peut pas se planter, juste avoir un résultat totalement inattendu et qui pourrait bien sonner.

L'interface ressemble à ça.



Ya beaucoup de texte de partout mais pas de panique ! Je t'explique :

- En haut à gauche, on a l'éditeur de code. C'est la dessus qu'on va composer.
- En haut à droite c'est un oscilloscope, il se met à bouger dès que des sons sont joués. Ça te permet de voir les ondes sonores que tu entends. C'est marrant à regarder.
- En dessous de l'oscilloscope, on retrouve un journal. Il décrit tout ce qu'il se passe lorsqu'un son est joué.
- En dessous du journal, ce sont les signaux. Ils indiquent les événements qui surviennent lorsque l'on joue la musique.
- En dessous des signaux, il y a un métronome. C'est utile pour tout synchroniser.

- Enfin tout en bas, il y a l'aide. Tu y trouveras tout plein d'informations. N'hésite pas à y jeter un œil si tu veux en apprendre plus sur l'outil.

Synthétiseur

C'est peut être la première fois que tu utilises un langage de programmation textuelle. Mais tu verras, c'est très simple. On va commencer simplement par une note. Écrit cette petite ligne dans l'éditeur de code.

```
play 60
```

Et maintenant, clique sur le bouton **run**. Il ressemble à ça :

Cool c'est ton premier son ! Mais qu'est-ce qu'il s'est passé au juste ?

Le mot `play` demande à l'ordinateur de jouer une note. Tout simplement. Quant à `60`, c'est la fréquence.

Essai de remplacer le `60` par `75` et re-clique sur le bouton **run**. Le son est plus aigu !

Mais revenons à `60`. Il se trouve que ce nombre correspond à un Do du quatrième octave. Si tu as un piano, ce serait cette touche là :

Ensuite `61` serait un Do dièse, `62` un Ré et ainsi de suite

Tu peux d'ailleurs appeler les notes par leur nom. Remplace la ligne par ceci :

```
play :C4
```

Et maintenant, clique sur le bouton **run**. C'est le même son qu'avec `play 60` !

Mais pourquoi `:C4` ?

En fait c'est parce qu'il s'agit de la notation anglaise. Voila un petit tableau de correspondance pour s'y retrouver.

A	La
B	Si
C	Do
D	Ré
E	Mi
F	Fa
G	Sol
s (sharp)	# (dièse)
b (bemol)	♭ (bémol)

Et le ':' alors ? C'est ce qu'utilise le langage de sonic-pi (tiré du ruby) pour lui dire d'aller chercher une valeur dans son dictionnaire. Donc en mettant `:C4`, sonic-pi va voir dans son registre que ça signifie `60`.

Sample

Voilà pour l'utilisation du synthétiseur mais ça n'est pas la seule manière de faire de la musique avec Sonic-pi, on peut aussi jouer un morceau de musique et jouer avec ce morceau de musique. Ici on appelle cela des samples et pour cela, il y a la commande `sample`.

Remplace tout le texte par cette ligne :

```
sample :loop_amen
```

Clique sur **run** pour écouter le résultat. On entend un sample de batterie !

`:loop_amen` est, comme on l'a déjà vu avec le synthétiseur plus haut, un raccourcie vers le sample de batterie à jouer.

Par contre, on n'entend le morceau de batterie qu'une seule fois. Si on voulait utiliser le sample pour rythmer une musique, il faudrait pouvoir l'écouter en boucle. Si tu ajoutes ceci. (Je surligne en vert ce qu'il faut ajouter ;))

```
live_loop :rythme do
  sample :loop_amen
end
```

Quand tu cliqueras sur **run**, il joue le morceau qu'une seule fois encore mais en plus il y a un gros message d'erreur qui apparait.

```
Runtime Error: [workspace_one] - Thread death +-->
:live_loop_rythme
loop did not sleep or sync!
```

Qu'est ce que ça veut dire ? Déjà que même s'il y a une erreur, show must go on !
Mais surtout qu'il n'est pas content parce que la boucle n'a pas dormi ? Mais qu'est-ce que ça veut dire ?

Ça veut dire, il faut laisser un peu de temps passer avant de faire un nouveau tour de boucle. Pour cela, il y a une commande pratique : `sleep`. Ajoute la juste avant le `end`

```
live_loop :rythme do
  sample :loop_amen
  sleep 2
end
```

Maintenant si tu cliques sur **run**, le morceau devrait être joué en boucle, mais on entend comme un silence entre deux lectures, ça n'est pas très agréable. On pourrait pour améliorer ça lui demander d'attendre le temps du sample. C'est ce que va faire `sample_duration`. Remplace le `sleep 2` par ceci. (Je barre et surligne en rouge ce qu'il faut retirer. Et donc en vert ce qui va le remplacer ;))

```
live_loop :rythme do
  sample :loop_amen
  sleep ~~2~~ sample_duration :loop_amen
end
```

Et sans arrêter la musique, clique sur **run**. Tu as remarqué ? Les petits silences entre deux morceaux joués ont disparu ! Dans les `live_loop`, on peut modifier les commandes

dedans et quand tu cliques sur **run**, elles ne seront prises en compte que lors du prochain tour de boucle. Tu peux même changer les samples si tu veux, remplace les `:loop_amen` par des `:loop_industrial` puis clique sur **run**. Des quel sample est fini, il passe à l'autre.

Un autre sample

On peut utiliser toute sorte de son comme sample, pas seulement ceux inclus dans sonic-pi. Et si on en essayait un justement ! Dans la clé usb, il y a un répertoire tp_sonic_pi que tu peux copier sur ton ordinateur. Il contient d'autres samples. Quand ça sera copié. Modifie le `live_loop :rythme` pour qu'il l'utilise. Ton code devrait ressembler à ceci.

```
chemin = "<chemin du sample>/"
live_loop :rythme do
  sample :loop_amen
  monSample = chemin + "cola_bottle_baby_intro_1.wav"
  sample monSample
  sleep sample_duration :loop_amen monSample
end
```

`chemin =` et `monSample =` sont des déclarations de variable. On vient donc de créer deux variables qui s'appellent `chemin` et `monSample`. À droite du `=`, on met la valeur que l'on souhaite mettre dans la variable. Par exemple pour `chemin`, on va mettre l'emplacement du répertoire que je t'ai demandé de copier. Tu remarqueras qu'il est entouré de `"`, c'est pour indiquer qu'il s'agit de texte. Ça servira ensuite pour la variable `monSample` qui sera composé de la variable `chemin` plus (qu'on indique avec le symbole `+`) le nom du fichier contenant le sample.

Clique sur **run** pour écouter la mélodie avec ce nouveau sample. Si tu ne l'entend pas, c'est peut être parce que tu n'as pas mis le bon emplacement. Il doit y avoir un message d'erreur disant qu'il n'a pas trouvé le fichier.

```
Runtime Error: [workspace_one] - Thread death +-->
:live_loop_rythme
Error calling sample_duration: filter matched no sample
```

Modifier le sample

On va maintenant commencer à jouer avec le sample. Personnellement, je le trouve un peu lent, on va l'accélérer un peu. A la fin de la ligne `sample monSample` ajoute ceci `, beat_stretch: 16` (et n'oubli pas la virgule, c'est important). Et remplace la ligne `sleep sample_duration monSample` par `sleep 16`.

```
chemin = "<chemin du sample>/"
live_loop :rythme do
  monSample = chemin + "cola_bottle_baby_intro_1.wav",
beat_stretch: 16
  sample monSample
  sleep sample_duration monSample 16
end
```

Clique sur **run** pour voir ce qu'il change. La vache ce que c'est lent ! Encore plus lent qu'avant !!!

En fait ce qu'a fait `beat_stretch`, c'est de régler la vitesse du sample de manière à qu'elle dure un certain nombre de battement (ici 16). Ensuite on lui demande d'attendre 16 battements. Mais par défaut, le système est réglé sur 60 battement par seconde. Le sample s'étire donc sur 16 secondes !

Pendant que c'est en train de jouer, tu peux changer le sample et mettre celui-ci par exemple `"<chemin du sample>/cola_bottle_baby_intro_2.wav"` . Quand tu cliqueras sur run, tu verras que le changement est assez subtile. Ça peut être pas mal pour la chanson de varier un peu entre ces deux samples, c'est moins fatiguant à écouter.

Heureusement, on peut le changer. Insère donc la commande suivante juste en dessous de `live_loop :rythme do` et clique sur run. (Si tu en as marre de cliquer sur run, tu peux aussi utiliser le raccourcie CTRL+R, ou CMD+R sur macOs)

```
chemin = "<chemin du sample>/"
live_loop :rythme do
  use_bpm 150
  monSample = chemin + "cola_bottle_baby_intro_1.wav",
beat_stretch: 16
  sample monSample
  sleep 16
end
```

Non là, ça va trop vite ! abaisse le `use_bpm` à `123`.

Bon c'est déjà mieux mais ça manque de pêche. On va maintenant mettre des effets sur le sample. On utilise la commande `with_fx do … end` qui permet d'appliquer un effet sur tout ce qui se trouve entre `do` et `end`. Entoure la ligne avec `sample monSample` par une commande `with_fx`, ça doit ressembler à ça.

```
chemin = "<chemin du sample>/"
live_loop :rythme do
  use_bpm 150
  monSample = chemin + "cola_bottle_baby_intro_1.wav",
beat_stretch: 16
  with_fx :eq, low_shelf: 2, high_shelf: 2, mix: 1 do
    sample monSample
  end
  sleep 16
end
```

Ici, on utilise l'effet `:eq` qui est un equalizer, c'est un outil qui permet d'ajuster le volume sur certaines bandes de fréquence sonore. Là par exemple, on le fait uniquement sur les sons graves avec `low_shelf: 2` et sur les sons aiguës avec `high_shelf: 2`.

On va ajouter un deuxième effet ? Entoure encore une fois la ligne avec `sample monSample` par une commande `with_fx`.

```
chemin = "<chemin du sample>/"
live_loop :rythme do
  use_bpm 150
  monSample = chemin + "cola_bottle_baby_intro_1.wav",
beat_stretch: 16
  with_fx :eq, low_shelf: 2, high_shelf: 2, mix: 1 do
    with_fx :bpf, centre: 100, res: 0.3, mix: 0.6 do
      sample monSample
    end
  end
  sleep 16
end
```

A présent c'est l'effet `:bpm` pour Band Pass Filter (Filtre Passe Bande). Cet effet retire carrément certaines bandes de fréquence pour ne laisser que celle qui nous intéresse. On

indique la position de la bande à conserver avec `centre: 100` et la taille de la bande avec `res: 0.3`. `mix: 0.6`, quant à lui, permet de ne pas appliquer complètement l'effet. À `0`, il n'est pas appliqué du tout, à `1` il est appliqué complètement. Jouer avec ces valeurs peut donner des effets intéressants sur le son.

Mélodie

Pour notre atelier, on va avoir besoin d'une mélodie. On a vu tout au début qu'on pouvait jouer du synthé avec la commande `play`. Alors, ça peut être tentant d'enchaîner ces commandes pour créer une mélodie mais ça serait très lourd à écrire.

Heureusement, on peut procéder autrement : avec des listes.

Ajoute ceci juste tout à la fin

```
melodie1 = [:Fs1, :Fs1, :Cs3, :B2, :E2, :E2, :B2, :A2,
            :A1, :A1, :Gs2, :Fs2, :A2, :A2, :Cs2, :B1]
```

`melodie1` est une variable qui va contenir toutes les notes d'une première mélodie. Ces notes sont séparées par une virgule (`,`) et le tout est entre crochet (`[ ]`). De cette manière, on crée une liste.

Pour lire ces notes, on va ajouter un nouveau `live_loop` tout à la fin comme ceci.

```
melodie1 = [:Fs1, :Fs1, :Cs3, :B2, :E2, :E2, :B2, :A2,
            :A1, :A1, :Gs2, :Fs2, :A2, :A2, :Cs2, :B1]
live_loop :synth do
  melodie1.size.times do
    play melodie1.tick
    sleep 0.5
  end
end
```

On retrouve notre variable contenant la liste de notes (`melodie1`). le `.size` est une méthode de la liste pour récupérer le nombre d'éléments qu'il y a dedans. Quant au `.times`, c'est une autre méthode pour effectuer des actions un certain nombre de fois. Donc dans notre cas, ça sera pour jouer chacune des notes que compose la liste.

Qu'est-ce qu'il se passe quand tu cliques sur **run** ? Déjà c'est pas trop jolie, c'est tout juste si on entend le synthé et puis il n'y est pas en rythme. On va arranger ça.

Petits ajustements

Actuellement, si tu as bien suivi toutes les instructions depuis le début, ton code devrait ressembler à ça.

```
chemin = "<chemin du sample>/"
live_loop :rythme do
  use_bpm 123
  monSample = chemin + "cola_bottle_baby_intro_1.wav"
  with_fx :eq, low_shelf: 2, high_shelf: 2, mix: 1 do
    with_fx :bpf, centre: 100, res: 0.3, mix: 0.6 do
      sample monSample, beat_stretch: 16
    end
  end
  sleep 16
end


melodie1 = [:Fs1, :Fs1, :Cs3, :B2, :E2, :E2, :B2, :A2, :A1,
:A1, :Gs2, :Fs2, :A2, :A2, :Cs2, :B1]
live_loop :synth do
  melodie1.size.times do
    play melodie1.tick
    sleep 0.5
  end
end
```

On va faire quelques petites modifications sur la mélodie pour que ce soit plus sympa à écouter.

Pour commencer, on va accélérer le rythme du synthé pour coller plus aux percussions. Ajoute la commande `use_bpm` juste en dessous du `live_loop :synth do`.

```
melodie1 = [:Fs1, :Fs1, :Cs3, :B2, :E2, :E2, :B2, :A2, :A1,
:A1, :Gs2, :Fs2, :A2, :A2, :Cs2, :B1]
live_loop :synth do
  use_bpm 123
  melodie1.size.times do
```

```
    play melodie1.tick
    sleep 0.5
  end
end
```

Normalement, en cliquant sur run. Tu devrais remarquer que le synthé va déjà plus vite et qu'on l'entend boucler 2 fois pendant que jouent les percussions. N'empêche que ne l'entend pas bien, on peut utiliser la commande `use_synth` pour changer de synthé. On va utiliser le synthé tb303. Ajoute-le en dessous du `use_bpm`.

```
melodie1 = [:Fs1, :Fs1, :Cs3, :B2, :E2, :E2, :B2, :A2, :A1,
:A1, :Gs2, :Fs2, :A2, :A2, :Cs2, :B1]
live_loop :synth do
  use_bpm 123
  use_synth :tb303
  melodie1.size.times do
    play melodie1.tick
    sleep 0.5
  end
end
```

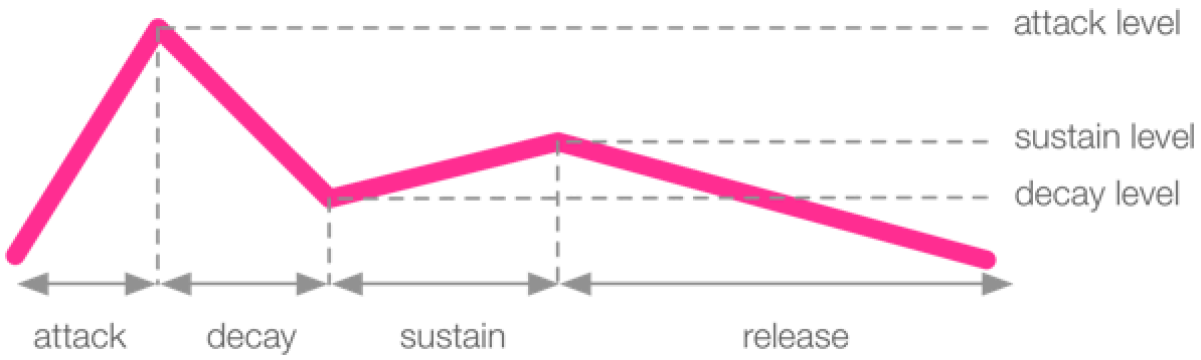
C'est différent mais ça fait mal aux oreilles. Cela-dit le `:tb303` peut être personnalisé. Ajouter ceci à la fin de la ligne `play melodie1.tick`.

```
melodie1 = [:Fs1, :Fs1, :Cs3, :B2, :E2, :E2, :B2, :A2, :A1,
:A1, :Gs2, :Fs2, :A2, :A2, :Cs2, :B1]
live_loop :synth do
  use_bpm 123
  use_synth :tb303
  melodie1.size.times do
    play melodie1.tick, cutoff: 100, res: 0.8, amp: 0.5,
release: 0.5
    sleep 0.5
  end
end
```

C'est déjà mieux ! Ce que tu as ajouté sont des paramètres qui personnalisent le synthé que l'on joue. `cutoff: 100` va filtrer les fréquences sonores un peu comme une tondeuse sur

des cheveux. `res: 0.8` va faire résonner le son, plus la valeur est élevée, plus on aura l'impression de jouer le son dans une cathédrale. `amp: 0.5` agit sur l'amplitude sonore du synthé, en dessous de 1, ça baisse le son, au-dessus, ça l'augmente. `release: 0.5` agit sur l'enveloppe sonore. Il y a 4 phases : attack, decay, sustain et release. Chaque phase à une amplitude sonore que l'on peut définir. De manière résumé, ça donne :

- `attack` - durée de passage de l'amplitude 0 jusqu'au niveau `attack_level`
- `decay` - durée pour passer l'amplitude du niveau `attack_level` jusqu'au niveau `decay_level`
- `sustain` - durée pour faire passer l'amplitude du niveau `decay_level` au niveau `sustain_level`
- `release` - durée pour passer l'amplitude du niveau `sustain_level` au niveau 0



Synchronisation

Maintenant, ajoutons une pause sur la mélodie pour qu'elle ne joue qu'au démarrage du sample. On pourrait très bien calculer le nombre de battements qu'il reste entre la fin de la mélodie et la fin du sample. Mettre le résultat comme valeur d'un `sleep`. Mais on peut tout aussi bien lui demander d'attendre que la lecture du sample soit terminée.

Te rappels-tu des signaux en dessous du journal. Si tu regardes bien lorsque la musique est en train de tourner, un message apparaît à chaque début d'un `live_loop` . Le signale ressemble à **/live_loop/synth** ou **/live_loop/rythme** selon qu'il s'agisse d'un nouveau passage de boucle de la mélodie ou du sample. En ajoutant `sync :rythme` , la boucle de mélodie va attendre un signale **/live_loop/rythme** avant de reprendre. Ajoute à la fin du `live_loop :synth` la ligne suivante :

```
melodie1 = [:Fs1, :Fs1, :Cs3, :B2, :E2, :E2, :B2, :A2, :A1,
:A1, :Gs2, :Fs2, :A2, :A2, :Cs2, :B1]
live_loop :synth do
  use_bpm 123
```

```
  use_synth :tb303
  melodie1.size.times do
    play melodie1.tick, cutoff: 100, res: 0.8, amp: 0.5,
release: 0.5
    sleep 0.5
  end
  sync :rythme
end
```

Ensuite, on va ajouter une seconde mélodie. Mais on alternera ces deux mélodies à chaque tour du `live_loop :synth`. Pour faire cela, on va ajouter une nouvelle ligne déclarant une autre mélodie qu'on appelera … `melodie2`. Cette ligne va contenir cette liste de note. Tu peux la mettre juste en dessous de la déclaration de `melodie1`.

```
melodie1 = [:Fs1, :Fs1, :Cs3, :B2, :E2, :E2, :B2, :A2, :A1,
:A1, :Gs2, :Fs2, :A2, :A2, :Cs2, :B1]
melodie2 = [:Fs1, :Fs1, :Fs2, :E2, :Fs2, :Fs2, :B2, :A2, :A1,
:A1, :Cs3, :B2, :A2, :A2, :Gs1, :Fs1]
live_loop :synth do
  use_bpm 123
  use_synth :tb303
  melodie1.size.times do
    play melodie1.tick, cutoff: 100, res: 0.8, amp: 0.5,
release: 0.5
    sleep 0.5
  end
  sync :rythme
end
```

Enfin on va ajouter ce qui va permettre d'alterner entre les deux mélodies. Tu remarqueras qu'il va y avoir une nouvelle variable `melodie` qui recevra la mélodie à jouer le temps de la boucle. Pense donc à retirer les **1** de `melodie1` quand ils sont joués

```
melodie1 = [:Fs1, :Fs1, :Cs3, :B2, :E2, :E2, :B2, :A2, :A1,
:A1, :Gs2, :Fs2, :A2, :A2, :Cs2, :B1]
melodie2 = [:Fs1, :Fs1, :Fs2, :E2, :Fs2, :Fs2, :B2, :A2, :A1,
:A1, :Cs3, :B2, :A2, :A2, :Gs1, :Fs1]
live_loop :synth do
```

```
  use_bpm 123
  use_synth :tb303
  melodie = [melodie1, melodie2].tick(:melodie)
  melodie1.size.times do
    play melodie1.tick, cutoff: 100, res: 0.8, amp: 0.5,
release: 0.5
    sleep 0.5
  end
  sync :rythme
end
```

C'est cette ligne qui va permettre d'alterner entre les deux mélodies grâce au `.tick`. Tu te souviens de ce que ça fait ? Par contre tu as dû remarquer qu'elle est un peu particulière : elle est suivie de `(:melodie)`.

Il faut savoir que `.tick` fonctionne avec un compteur. À chaque fois que `.tick` est appelé, il ajoute 1 à celui-ci. Le truc, c'est que c'est le même compteur partout dans le `live_loop`. Mais il est possible de gérer d'autres compteurs, il suffit pour cela d'indiquer à `.tick` lequel utiliser. Sans cela, il se passerait des trucs bizarres. Tu peux faire le test en retirant le `(:melodie)`.

Clique donc sur **run** pour écouter le résultat. C'est déjà plus sympa, non ? On sent que le sample et les mélodies vont bien ensemble.

Slicer

Il est possible d'ajouter un effet qui permet de modifier le volume sonore à la volée. Ça peut donner un résultat assez cool. On va essayer ça !

On doit utiliser un effet spécial (la commande `with_fx`) qui s'appelle `:slicer`. On va l'ajouter à la mélodie jouée, juste avant `16.times do`.

```
melodie1 = [:Fs1, :Fs1, :Cs3, :B2, :E2, :E2, :B2, :A2, :A1,
:A1, :Gs2, :Fs2, :A2, :A2, :Cs2, :B1]
melodie2 = [:Fs1, :Fs1, :Fs2, :E2, :Fs2, :Fs2, :B2, :A2, :A1,
:A1, :Cs3, :B2, :A2, :A2, :Gs1, :Fs1]
live_loop :synth do
  use_bpm 123
  use_synth :tb303
  melodie = [melodie1, melodie2].tick(:melodie)
```

```
  with_fx :slicer do
    melodie.size.times do
      play melodie.tick, cutoff: 100, res: 0.8, amp: 0.5,
release: 0.5
      sleep 0.5
    end
  end
  sync :rythme
end
```

Maintenant, clique sur **run**. Tu as vu comment ça a changer la mélodie ? C'est comme si quelqu'un bougeait frénétiquement le volume du synthé.

On peut ajouter des paramètres à l'effet :

- `wave:` va changer la manière dont le volume bougera. On a le choix entre 4 valeurs. (0 : en dent de scie, ça monte d'un coup et redescend progressivement; 1 : en carré, ça monte d'un coup puis redescend d'un coup ; 2 : en triangle , ça monte et redescend progressivement à la manière d'un triangle ; 3 : en vague : ça monte et redescend progressivement à la manière d'une vague)
- `phase:` va la longueur de la vague, ça se compte en battement. En mettant une valeur comme 0.125, ce sera encore plus hachuré que sans valeur. En mettant 1, on se rend bien compte du style de vague indiqué via le paramètre `wave:` .
- `invert_wave:` va quant à lui inverser la vague. On l'active en mettant la valeur à 1. Essai et tu verras ce que ça fait.

Il y a encore plein d'autres paramètres mais ce sont ces 3 là qui nous importent le plus. Dans notre cas, on va utiliser cet effet pour sélectionner les notes jouées et faire un échange entre le synthé et le sample. Utilise ces paramètres :

```
melodie1 = [:Fs1, :Fs1, :Cs3, :B2, :E2, :E2, :B2, :A2, :A1,
:A1, :Gs2, :Fs2, :A2, :A2, :Cs2, :B1]
melodie2 = [:Fs1, :Fs1, :Fs2, :E2, :Fs2, :Fs2, :B2, :A2, :A1,
:A1, :Cs3, :B2, :A2, :A2, :Gs1, :Fs1]
live_loop :synth do
  use_bpm 123
  use_synth :tb303
  melodie = [melodie1, melodie2].tick(:melodie)
  with_fx :slicer, phase: 2, wave: 1,
          invert_wave: [0, 1].tick(:phase) do
```

```
    melodie.size.times do
      play melodie.tick, cutoff: 100, res: 0.8, amp: 0.5,
release: 0.5
      sleep 0.5
    end
  end
  sync :rythme
end
```

Le `[0, 1].tick(:phase)` t'intrigue peut être ? C'est de manière condensé une liste où on indique ensuite qu'on prend la valeur suivante avec un compteur bien à lui comme on l'avait fait avec la sélection de la mélodie. Au final sur le paramètre `invert_wave:` ça va l'activer une fois sur deux à chaque passage de boucle.

Clique sur **run** pour écouter comment ça rend. On sent que certaines notes sont rendues muettes mais ce ne sont pas les mêmes à chaque passage de boucle. Par contre, on ne fait plus la différence entre les mélodies. Ça c'est parce qu'il n'y a que la première mélodie qui la vague dans un sens et la seconde qui à la vague dans l'autre sens. Aussi je te conseille de doubler la liste servant à sélectionner les mélodies comme ça

```
melodie1 = [:Fs1, :Fs1, :Cs3, :B2, :E2, :E2, :B2, :A2, :A1,
:A1, :Gs2, :Fs2, :A2, :A2, :Cs2, :B1]
melodie2 = [:Fs1, :Fs1, :Fs2, :E2, :Fs2, :Fs2, :B2, :A2, :A1,
:A1, :Cs3, :B2, :A2, :A2, :Gs1, :Fs1]
live_loop :synth do
  use_bpm 123
  use_synth :tb303
  melodie = [melodie1, melodie1, melodie2,
melodie2].tick(:melodie)
  with_fx :slicer, phase: 2, wave: 1,
        invert_wave: [0, 1].tick(:phase) do
    melodie.size.times do
      play melodie.tick, cutoff: 100, res: 0.8, amp: 0.5,
release: 0.5
      sleep 0.5
    end
  end
  sync :rythme
end
```

Maintenant on va essayer d'utiliser le même effet sur le sample. Avec une petite nuance, tu l'as remarqué ?

```
chemin = "<chemin du sample>/"
live_loop :rythme do
  use_bpm 123
  monSample = chemin + "cola_bottle_baby_intro_1.wav"
  with_fx :slicer, phase: 2, wave: 1,
          invert_wave: [1, 0].tick(:phase) do
    with_fx :eq, low_shelf: 2, high_shelf: 2, mix: 1 do
      with_fx :bpf, centre: 100, res: 0.3, mix: 0.6 do
        sample monSample, beat_stretch: 16
      end
    end
    sleep 16
  end
end
```

Ecoutons le résultat. Clique sur **run** !

C'est marrant comme échange entre la mélodie et le sample. Par contre, le sample continue à varier son volume après la fin de la mélodie. Faudrait trouver un moyen pour qu'après la mélodie, le sample ne varie plus. Et si je te disais que c'était possible ?

Il est en effet possible de contrôler les paramètres de l'effet spécial pendant qu'elle est en train de faire effet. On le fait avec la commande `control`. En lui passant l'effet spécial, on peut y apporter des modifications.

Pour cela, il faut ajouter à la fin de la ligne `with_fx :slicer` après le `do` ceci : `|fx|`. Mais qu'est-ce donc ? C'est une variable contenant l'effet spécial. Ici on l'appel `fx` mais on peut lui donner n'importe quel nom. Si son nom est entouré de `|`, c'est pour indiquer qu'elle n'est utilisable qu'entre le `do` et le `end` de l'effet spécial.

Ensuite on va réduire le sleep 16 à 8 pour qu'il n'attende que la moitié du temps. Cela nous permettra d'ajouter le nécessaire pour arrêter l'effet spécial en cours d'exécution. On va ajouter les lignes suivantes

```
chemin = "<chemin du sample>/"
live_loop :rythme do
  use_bpm 123
  monSample = chemin + "cola_bottle_baby_intro_1.wav"
```

```
      with_fx :slicer, phase: 2, wave: 1,
            invert_wave: [1, 0].tick(:phase) do |fx|
        with_fx :eq, low_shelf: 2, high_shelf: 2, mix: 1 do
          with_fx :bpf, centre: 100, res: 0.3, mix: 0.6 do
            sample monSample, beat_stretch: 16
          end
        end
        sleep 168
        control fx, mix: 0
        sleep 8
      end
    end
```

Le `control` va passer le paramètre `mix:` de l'effet `fx` à 0 ce qui revient à retirer l'effet spécial. Ensuite on lui demande d'attendre 8 battement jusqu'à la fin du sample.

Vérifie donc en cliquant sur **run**. On remarque que le sample joue normalement à la moitié.

Découpage

Un autre truc que l'on peut faire avec les samples, c'est de les découper et les recomposer. C'est un moyen marrant pour transformer un sample en un autre. Faisons un exemple, clique sur l'onglet de travail 2 pour repartir de zéro. Enfin pas tout à fait parce que nous allons reprendre le sample sans l'utilisation du slicer.

```
    chemin = "<chemin du sample>/"
    live_loop :rythme do
      use_bpm 123
      monSample = chemin + "cola_bottle_baby_intro_1.wav"
      with_fx :eq, low_shelf: 2, high_shelf: 2, mix: 1 do
        with_fx :bpf, centre: 100, res: 0.3, mix: 0.6 do
          sample monSample, beat_stretch: 16
        end
      end
      sleep 16
    end
```

Si on l'écoute en cliquant sur **run**. On aura juste le sample dans son entièreté. Jusque là tout va bien.

Mais sur le sample, on peut ajouter des paramètres pour sélectionner une partie à jouer :

- `num_slices:` permet de définir en combien de morceaux on souhaite découper le sample.
- `slice:` quant à lui sélectionne le morceau du sample à jouer. Attention, on commence par 0.
- `start:` indique à partir de quand jouer le morceau. On indique une valeur comprise entre 0 (le début) et 1 (la fin). On peut le combiner avec `slice:`. Dans ce cas, c'est le morceau qui sera encore plus découpé.
- `finish:` indique jusqu'à quand jouer le morceau. On indique une valeur comprise entre 0 (le début) et 1 (la fin). On peut le combiner avec `slice:`. Dans ce cas, c'est le morceau qui sera encore plus découpé. Si `finish:` est plus petit que `start:` le morceau sera joué à l'envers.

Dans notre cas, essai d'ajouter les paramètre `slice: 4, num_slices: 16` au sample. Et ensuite, abaisser le `sleep` à `1` tout en le plaçant juste sous la commande `sample`. Ça doit ressembler à ceci :

```
chemin = "<chemin du sample>/"
live_loop :rythme do
  use_bpm 123
  monSample = chemin + "cola_bottle_baby_intro_1.wav"
  with_fx :eq, low_shelf: 2, high_shelf: 2, mix: 1 do
    with_fx :bpf, centre: 100, res: 0.3, mix: 0.6 do
      sample monSample, beat_stretch: 16, slice: 4,
num_slices: 16
      sleep 1
    end
  end
end
```

Clique sur **run** maintenant. C'est court, hein ?

En découpant de cette manière, on pourrait prendre des morceaux et les replacer ailleurs et ainsi faire un autre rythme. Cela risque de demander beaucoup de copier/coller mais il est possible de faire un peu comme avec les mélodies au synthé plus haut.

Remplace donc le contenu entre le `do` et `end` du `live_loop` par ceci

```
chemin = "<chemin du sample>/"
live_loop :rythme do
```

```
  use_bpm 123
  monSample = chemin + "cola_bottle_baby_intro_1.wav"
  with_fx :eq, low_shelf: 2, high_shelf: 2, mix: 1 do
    with_fx :bpf, centre: 100, res: 0.3, mix: 0.6 do
      4.times do
        numSlice=[16, 2, 4, 6].tick()
        4.times do
          sample monSample, beat_stretch: 16, slice:
4numSlice, num_slices: 16
          sleep 1
        end
      end
    end
  end
end
```

Et clique à nouveau sur **run**. C'est déjà plus sympa.

On va maintenant ajouter un autre `live_loop` qui va jouer un autre sample pour rythmer. Ajoute ceci à la toute fin.

```
live_loop :rythme2 do
  use_bpm 123
  sample :bd_haus, amp: 0.5, beat_stretch: 1, cutoff: 80
  sleep 1
  sample :bd_haus, amp: 0.7, beat_stretch: 1, lpf: 120
  sleep 1
end
```

Clique maintenant sur **stop** puis sur **run** pour relancer la musique. C'est important ici d'enchaîner les deux de manière à ce que ces boucles soit bien synchro.

Encore du synthé ?

Aller si t'insiste. Pour accompagner ce sample modifié, on va reprendre la boucle de synthé précédente

```
melodie1 = [:Fs1, :Fs1, :Cs3, :B2, :E2, :E2, :B2, :A2, :A1,
:A1, :Gs2, :Fs2, :A2, :A2, :Cs2, :B1]
melodie2 = [:Fs1, :Fs1, :Fs2, :E2, :Fs2, :Fs2, :B2, :A2, :A1,
:A1, :Cs3, :B2, :A2, :A2, :Gs1, :Fs1]
live_loop :synth do
  use_bpm 123
  use_synth :tb303
  melodie = [melodie1, melodie1, melodie2,
melodie2].tick(:melodie)
  with_fx :slicer, phase: 2, wave: 1,
        invert_wave: [0, 1].tick(:phase) do
    melodie.size.times do
      play melodie.tick, cutoff: 100, res: 0.8, amp: 0.5,
release: 0.5
      sleep 0.5
    end
  end
  sync :rythme
end
```

Et juste modifier les mélodies

```
melodie1 = [:Fs1, :Fs1, :Cs3, :B2, :E2, :E2, :B2, :A2, :A1,
:A1, :Gs2, :Fs2, :A2, :A2, :Cs2, :B1]
melodie2 = [:Fs1, :Fs1, :Fs2, :E2, :Fs2, :Fs2, :B2, :A2, :A1,
:A1, :Cs3, :B2, :A2, :A2, :Gs1, :Fs1]
melodie3 = [:Fs1, :A1, :Fs2, :A2, :Cs3, :A2, :Fs2, :A2, :E1,
:E1, :A2, :E2, :B2, :A2, :Gs2, :A2]
melodie4 = [:Ds1, :Ds1, :Fs3, :D3, :B3, :A3, :Fs2, :Ds3, :D2,
:D1, :Fs2, :A2, :Fs1, :Fs1, :Fs1, :Fs1]
melodie5 = [:E2, :Fs2, :A2, :Fs2, :Cs3, :B2, :A2, :Fs2, :E2,
:Fs2, :A2, :Fs2, :Cs3, :B2, :A2, :Fs2]
melodie6 = [:A2, :A2, :Fs2, :Fs2, :A1, :A1, :Fs1, :Fs1, :A2,
:A2, :Fs2, :Fs2, :A2, :A2, :Fs2, :Fs2]
melodie7 = [:E2, :Fs2, :A2, :Fs2, :Cs3, :B2, :A2, :Fs2, :E2,
:Fs2, :A2, :Fs2, :Cs3, :B2, :A2, :Fs2]
melodie8 = [:E3, :Fs3, :A3, :Fs3, :Cs4, :B3, :A3, :Fs3 , :E2,
:Fs2, :A2, :B2, :Fs1, :Fs1, :Fs1, :Fs1]
live_loop :synth do
```

```
  use_bpm 123
  use_synth :tb303
  melodie = [melodie3, melodie4, melodie5, melodie6,
melodie7, melodie8].tick(:melodie)
  with_fx :slicer, phase: 2, wave: 1,
        invert_wave: [0, 1].tick(:phase) do
    melodie.size.times do
      play melodie.tick, cutoff: 100, res: 0.8, amp: 0.5,
release: 0.5
      sleep 0.5
    end
  end
  sync :rythme
end
```

Clique maintenant sur run pour écouter le résultat. Alors, ça sonne comment ?

Et ensuite ?

L'un des intérêts de Sonic-pi, c'est que l'on peut faire des des modifications en live lorsque l'on produit de la musique. Aussi dans notre code, on peut s'y prendre de deux manières différentes : Soit on code absolument tout et on a juste à dérouler la chanson comme une partition. Soit on fait des ajustement sur le code pendant qu'il est en train de jouer.

C'est à présent le moment idéal pour essayer de faire une petite présentation et de jouer avec le code en live !