Guide Strudel : Live Coding Musical

*Une rapide introduction par Florent Bertiaux, via **JamaisLeVendredi.com**.*

On parle de quoi ?

Strudel est un environnement de live coding musical basé sur le navigateur web, qui porte le langage de patterns TidalCycles en JavaScript. Avec Strudel, tu peux créer de la musique en écrivant du code de manière expressive et dynamique, directement dans ton navigateur, sans installation requise.

Qu'est-ce que le live coding ?

Le live coding musical consiste à créer et modifier de la musique en temps réel en écrivant du code. Tu peux changer ton code pendant que la musique joue, et les modifications prennent effet immédiatement.

Accéder à Strudel

Rends-toi sur **https://strudel.cc/** pour accéder au REPL (Read-Eval-Print Loop) de Strudel.

Raccourcis clavier essentiels

- `Ctrl + Enter` : Lancer/Actualiser le code
- `Ctrl + .` : Arrêter la lecture

---

1. Premiers Sons

Jouer un son simple

```
sound("casio")
```

Explication : La fonction `sound()` joue le son spécifié entre guillemets. Ici, "casio" est un des nombreux échantillons sonores disponibles par défaut.

Sons disponibles par défaut

Essaie ces sons :

- `insect`, `wind`, `jazz`, `metal`, `east`, `crow`, `casio`, `space`, `numbers`

Exemple :

```
sound("metal")
```

Sélectionner un échantillon spécifique avec `:`

Chaque son peut contenir plusieurs échantillons. Tu peux sélectionner un échantillon spécifique avec `:` suivi d'un numéro.

```
sound("casio:0")  // Premier échantillon (par défaut)

sound("casio:1")  // Deuxième échantillon

sound("casio:2")  // Troisième échantillon
```

Sons de batterie

Strudel inclut une large sélection de sons de batterie :

```
sound("bd hh sd oh")
```

Abréviations courantes :

- `bd` = **B**ass **D**rum (grosse caisse)
- `sd` = **S**nare **D**rum (caisse claire)
- `rim` = **Rim**shot
- `hh` = **H**i **H**at (charleston fermé)
- `oh` = **O**pen **H**ihat (charleston ouvert)
- `lt` = **L**ow **T**om
- `mt` = **M**iddle **T**om
- `ht` = **H**igh **T**om
- `rd` = **Rid**e cymbal
- `cr` = **Cr**ash cymbal
- `cp` = **C**la**p** (clap)

Changer de banque de sons avec `bank()`

```
sound("bd hh sd oh").bank("RolandTR909")
```

Banques disponibles :

- `RolandTR909` (House/Techno)
- `RolandTR808` (Hip-hop classique)
- `RolandTR707`
- `AkaiLinn`
- `RhythmAce`
- `ViscoSpaceDrum`

Exemple :

```
sound("bd sd cp hh").bank("RolandTR808")
```

---

2. Mini-Notation : Le langage rythmique

La **Mini-Notation** est un langage spécial conçu pour écrire des patterns rythmiques de manière concise.

Séquences simples (espace)

Les sons séparés par des espaces forment une séquence :

```
sound("bd hh sd hh")
```

Principe clé : Plus tu ajoutes de sons, plus la séquence devient rapide, car tous les sons sont compressés dans un **cycle** (durée de 2 secondes par défaut).

Exemple :

```
sound("bd bd hh bd rim bd hh bd")  // 8 sons = plus rapide
```

Silences avec ~ ou –

```
sound("bd hh ~ rim ~ bd hh rim")
```

Les symboles ~ et – représentent des silences.

Sous-séquences avec `[ ]`

Les crochets permettent de créer des subdivisions rythmiques :

```
sound("bd [hh hh] sd [hh bd]")
```

Explication : `[hh hh]` joue deux hi-hats dans l'espace d'un seul événement.

Sous-sous-séquences :

```
sound("bd [[rim rim] hh] sd cp")
```

Tu peux imbriquer autant de niveaux que nécessaire !

Parallélisme avec `,`

La virgule joue plusieurs patterns simultanément :

```
sound("bd bd bd bd, hh hh hh hh hh hh hh hh")
```

Exemple de batterie complète :

```
sound("bd sd, hh*4")
```

Multiplication avec `*`

Accélère un élément :

```
sound("bd hh*2 sd hh*3")
```

`hh*2` joue le hi-hat 2 fois plus vite.

Avec des sous-séquences :

```
sound("bd [hh rim]*2 sd [hh rim]*3")
```

Division avec `/`

Ralentit un élément :

```
sound("bd/2 sd hh")
```

`bd/2` joue la grosse caisse sur 2 cycles.

Alternance avec `< >`

Les chevrons alternent entre les éléments à chaque cycle :

```
sound("<bd sd rim cp>")
```

Cycle 1 : bd
Cycle 2 : sd
Cycle 3 : rim
Cycle 4 : cp

Combiné avec multiplication :

```
sound("<bd sd rim cp>*8")
```

Joue 8 notes par cycle en alternant les sons.

Élongation avec `@`

Donne plus de poids temporel à un élément :

```
sound("bd@3 sd")
```

`bd@3` dure 3 fois plus longtemps que `sd`.

Réplication avec `!`

Répète un élément sans l'accélérer :

```
sound("bd!3 sd")
```

Équivalent à `sound("bd bd bd sd")`.

Aléatoire avec `?`

`?` donne 50% de chance de jouer le son :

```
sound("bd hh? sd hh?")
```

Avec probabilité personnalisée :

```
sound("bd hh?0.2 sd hh?0.8")
```

`?0.2` = 20% de chance, `?0.8` = 80% de chance.

Choix aléatoire avec `|`

```
sound("bd | sd | cp | hh")
```

Choisit aléatoirement entre les options.

Rythmes euclidiens avec `(beats, steps)`

Crée des rythmes basés sur l'algorithme euclidien :

```
sound("bd(3,8)")
```

Explication : Distribue 3 coups (beats) sur 8 pas (steps).

Avec offset (décalage) :

```
sound("bd(3,8,2)")
```

Le troisième paramètre décale le pattern.

Exemple musical :

```
sound("bd(5,8), hh*8, sd(3,8,2)")
```

---

3. Notes et Mélodies

Jouer des notes avec `note()`

```
note("c d e f g a b")
```

Notation des octaves :

```
note("c3 e3 g3 c4")
```

Les chiffres indiquent l'octave (c3 = do de la 3e octave).

Altérations

- `#` = dièse (monte d'un demi-ton)
- `b` = bémol (baisse d'un demi-ton)

```
note("c# d eb f# g ab b")
```

Sélectionner le son avec `.s()`

```
note("c e g b").s("piano")
```

Synthétiseurs disponibles :

- `sawtooth` (dent de scie)

- `square` (carré)
- `triangle` (triangle)
- `sine` (sinusoïde)

Exemple :

```
note("c3 e3 g3 c4").s("sawtooth")
```

Gammes avec `.scale()`

```
n("0 2 4 6").scale("C:minor").s("piano")
```

Explication :

- `n()` : numéros de degrés dans la gamme
- `.scale("C:minor")` : gamme de do mineur

Gammes populaires :

- `C:major` (do majeur)
- `C:minor` (do mineur)
- `D:dorian` (ré dorien)
- `E:phrygian` (mi phrygien)
- `F:lydian` (fa lydien)
- `G:mixolydian` (sol mixolydien)
- `C:pentatonic` (pentatonique)
- `C:blues` (blues)

Exemple :

```
n("0 1 2 3 4 5 6 7").scale("D:pentatonic").s("triangle")
```

Accords

```
note("c3,e3,g3")
```

La virgule joue les notes simultanément.

Avec progression d'accords :

```
note("<[c3,e3,g3] [d3,f3,a3] [e3,g3,b3]>*2").s("sawtooth")
```

---

4. Effets Audio

Filtre passe-bas (lowpass) avec `.lpf()`

```
sound("bd sd, hh*8").lpf(2000)
```

Explication : Coupe les fréquences au-dessus de 2000 Hz.

Avec pattern :

```
sound("bd sd, hh*8").lpf("<500 1000 2000 4000>")
```

Résonance du filtre avec `.lpq()`

```
sound("bd sd, hh*8").lpf(1000).lpq(10)
```

Valeurs entre 0 et 50. Plus c'est élevé, plus c'est agressif.

Filtre passe-haut avec `.hpf()`

```
sound("bd sd, hh*8").hpf(200)
```

Coupe les fréquences en dessous de 200 Hz.

Filtre voyelle avec `.vowel()`

```
note("c2 eb2 g2 bb2").s("sawtooth").vowel("<a e i o u>")
```

Simule les sons de voyelles.

Distorsion avec `.distort()`

```
note("c3 e3 g3").s("sawtooth").distort(2)
```

⚠️ **Attention** : peut devenir très fort !

Délai (delay) avec `.delay()`

```
sound("bd rim sd rim").delay(0.5)
```

Paramètres supplémentaires :

```
sound("bd rim").delay(0.5).delaytime(0.25).delayfeedback(0.7)
```

- `.delaytime()` : temps du délai
- `.delayfeedback()` : niveau de feedback (⚠️ <1 sinon ça explose)

Réverbération avec `.room()`

```
sound("bd sd rim cp").room(0.8)
```

Avec taille de pièce :

```
sound("bd sd rim cp").room(0.8).roomsize(4)
```

Panoramique avec `.pan()`

```
sound("bd rim sd rim").pan("0 0.5 1 0.5")
```

0 = gauche, 0.5 = centre, 1 = droite

Exemple avec oscillation :

```
sound("bd rim sd rim").pan(sine.slow(2))
```

Gain (volume) avec `.gain()`

```
sound("hh*8").gain("0.4 0.6 1 0.8")
```

Enveloppe ADSR

Contrôle l'évolution temporelle du son :

```
note("c3 e3 g3").s("sawtooth")

  .attack(0.1)   // Temps de montée

  .decay(0.2)    // Temps de descente
```

```
  .sustain(0.5)  // Niveau soutenu

  .release(0.3)  // Temps de relâchement
```

Forme courte :

```
note("c3 e3 g3").s("sawtooth").adsr("0.1:0.2:0.5:0.3")
```

---

5. Manipulation de Patterns

Vitesse avec `.fast()` et `.slow()`

```
sound("bd sd rim cp").fast(2)  // 2x plus rapide

sound("bd sd rim cp").slow(2)  // 2x plus lent
```

Inverser avec `.rev()`

```
note("c d e f").rev()  // Joue f e d c
```

Effets stéréo avec `.jux()`

Applique une fonction uniquement sur le canal droit :

```
sound("bd sd rim cp").jux(rev)
```

Avec intensité variable :

```
sound("bd sd rim cp").juxBy(0.5, rev)
```

0 = mono, 1 = stéréo complet

Ajouter avec `.add()`

```
note("c e g").add("<0 2 4 7>")
```

Ajoute des intervalles aux notes.

Répétition avec `.ply()`

```
sound("bd sd").ply("<1 2 3 4>")
```

Répète chaque événement n fois.

Décalage avec `.off()`

```
sound("bd sd, hh*4").off(1/8, x => x.speed(2))
```

Explication : Copie le pattern, le décale de 1/8 de cycle et applique la transformation.

Variations aléatoires avec `.sometimes()`

```
sound("hh*8").sometimes(x => x.speed(0.5))
```

Applique la fonction 50% du temps.

Avec probabilité personnalisée :

```
sound("hh*8").sometimesBy(0.3, x => x.speed(0.5))
```

30% de chance d'application.

Tempo avec `setcpm()`

```
setcpm(120/4)  // 120 BPM en 4/4

sound("bd sd, hh*8")
```

CPM = Cycles Par Minute (par défaut = 30)

---

6. Exemples Complets Testables

Beat House Classique

```
sound("bd*4, [~ cp]*2, [~ hh]*4").bank("RolandTR909")
```

Beat Hip-Hop

```
setcpm(90/4)

sound(`

  [bd - - -] [- - bd -] [bd - - bd] [- - - -],

  [- - - -] [cp - - -] [- - - -] [cp - - -],

  [hh hh hh hh] [hh hh hh hh] [hh hh hh hh] [hh hh - -]

`).bank("RolandTR808")
```

Ligne de basse techno

```
note("<a1 b1*2 a1(3,8) e2>")
  .off(1/8, x => x.add(12).degradeBy(0.5))
  .add(perlin.range(0, 0.5))
  .superimpose(add(0.05))
  .s("sawtooth")
  .decay(0.15)
  .sustain(0)
  .gain(0.4)
  .cutoff(sine.slow(7).range(300, 5000))
```

Progression d'accords ambient

```
note("<[c3,e3,g3] [a2,c3,e3] [f2,a2,c3] [g2,b2,d3]>*2")
  .s("sawtooth")
  .lpf(800)
  .room(0.9)
  .roomsize(5)
```

```
  .attack(1)

  .release(2)

  .gain(0.3)
```

Mélodie pentatonique avec effets

```
n("0 2 4 5 7 9 11 12")

  .scale("C:pentatonic")

  .s("triangle")

  .delay(0.5)

  .delaytime(0.125)

  .room(0.5)

  .lpf(2000)

  .jux(rev)
```

Pattern euclidien multi-couches

```
stack(

  sound("bd(5,8)").bank("RolandTR909"),

  sound("sd(3,8,2)").gain(0.8),
```

```
    sound("hh(7,8)").gain(0.6),
    sound("cp(2,8,3)").gain(0.7)
  )
```

Expérimentation sonore

```
  n(run(8))
    .scale("C:minor")
    .s("sawtooth")
    .lpf(sine.range(200, 4000).slow(4))
    .lpq(10)
    .phaser(4)
    .room(0.7)
    .juxBy(0.8, rev)
```

---

7. Fonctions de Pattern Avancées

`stack()` - *Empiler des patterns*

```
stack(
  sound("bd*4"),
  sound("~ cp"),
  sound("hh*8")
)
```

`cat()` - *Concaténer séquentiellement*

```
cat(
  sound("bd sd"),
  sound("hh cp"),
  sound("rim oh")
)
```

Chaque pattern joue pendant un cycle complet.

Utiliser plusieurs canaux avec `$:`

```
$: sound("bd sd, hh*8")

$: note("c e g").s("piano")
```

Chaque ligne `$:` crée un canal séparé qui joue en parallèle.

---

8. Charger des Échantillons Personnalisés

Depuis une URL

```
samples({
  mySound: "https://example.com/sound.wav"
})



sound("mySound")
```

Depuis GitHub

```
samples('github:tidalcycles/Dirt-Samples')

sound("ade*4")
```

Définir une base d'URL

```
samples({
  _base: "https://example.com/samples/",
  kick: "kick.wav",
  snare: "snare.wav"
})

sound("kick snare")
```

---

9. Astuces et Bonnes Pratiques

Utiliser les backticks pour le multi-ligne

```
sound(`
  bd sd bd cp,
  hh*8,
  ~ rim ~ rim
`)
```

Créer des variations progressives

```
sound("bd sd, hh*<4 8 16 32>")
```

Le hi-hat accélère à chaque cycle.

Combiner effets

```
note("c e g")
  .s("sawtooth")
  .lpf(1000)
  .lpq(5)
  .delay(0.5)
  .room(0.3)
  .pan(sine)
  .gain(0.7)
```

Les effets se chaînent avec `.`

Utiliser des signaux continus

```
sound("bd*4").cutoff(sine.range(200, 2000).slow(4))
```

`sine`, `saw`, `square`, `tri` créent des oscillations.

Variations aléatoires

```
note("c e g").sometimes(x => x.add(7))
```

Ajoute parfois une quinte.

---

10. Ressources et Documentation

Sites officiels

- **REPL Strudel** : https://strudel.cc/
- **Workshop interactif** : https://strudel.cc/workshop/
- **Documentation complète** : https://strudel.cc/learn/
- **Exemples** : https://strudel.cc/examples/

Tutoriels vidéo

- Workshop Strudel sur YouTube
- Creative Code Art - Coding Music with Strudel

Communauté

- **Tidal Club** : https://club.tidalcycles.org/
- **Discord Strudel**
- Showcase et partage de patterns

Récapitulatif des Symboles Mini-Notation

Symbole	Fonction	Exemple
espace	Séquence	`sound("bd sd hh")`
`:`	Numéro d'échantillon	`sound("hh:0 hh:1")`
`~` ou `-`	Silence	`sound("bd ~ sd ~")`
`[ ]`	Sous-séquence	`sound("bd [hh sd]")`
`,`	Parallèle	`sound("bd, hh*4")`
`*`	Multiplication	`sound("hh*4")`
`/`	Division	`sound("bd/2")`
`< >`	Alternance	`sound("<bd sd>")`
`@`	Élongation	`sound("bd@3 sd")`
`!`	Réplication	`sound("bd!3")`
`?`	Aléatoire	`sound("bd?")`
`|`	Choix	`sound("bd|sd")`
`(x,y)`	Euclidien	`sound("bd(3,8)")`

Récapitulatif des Fonctions Principales

Fonction	Description	Exemple
`sound()`	Jouer un son	`sound("bd sd")`
`note()`	Jouer une note	`note("c e g")`
`s()`	Sélectionner synthé/son	`.s("sawtooth")`
`bank()`	Banque de sons	`.bank("RolandTR909")`
`scale()`	Gamme musicale	`.scale("C:minor")`
`lpf()`	Filtre passe-bas	`.lpf(1000)`
`hpf()`	Filtre passe-haut	`.hpf(200)`
`delay()`	Délai	`.delay(0.5)`
`room()`	Réverbération	`.room(0.8)`
`pan()`	Panoramique	`.pan(0.5)`
`gain()`	Volume	`.gain(0.7)`
`fast()`	Accélérer	`.fast(2)`
`slow()`	Ralentir	`.slow(2)`
`rev()`	Inverser	`.rev()`
`jux()`	Effet stéréo	`.jux(rev)`
`stack()`	Empiler patterns	`stack(p1, p2)`
`setcpm()`	Tempo	`setcpm(120/4)`

Conseils pour Débuter

1. **Commence simple** : Un seul pattern à la fois
2. **Expérimente** : Change les valeurs pour voir l'effet
3. **Écoute** : Laisse tourner le pattern pour bien l'entendre
4. **Construis progressivement** : Ajoute des couches petit à petit
5. **Sauvegarde** : Note tes patterns préférés
6. **Explore** : Teste les exemples de la documentation

Bon live coding ! 🎵