Musique Assistée par Ordinateur

Faire vibrer les octets !

Linux et la Musique

Quels outils pour libérer les ondes ?

Configurer Jack sous Linux

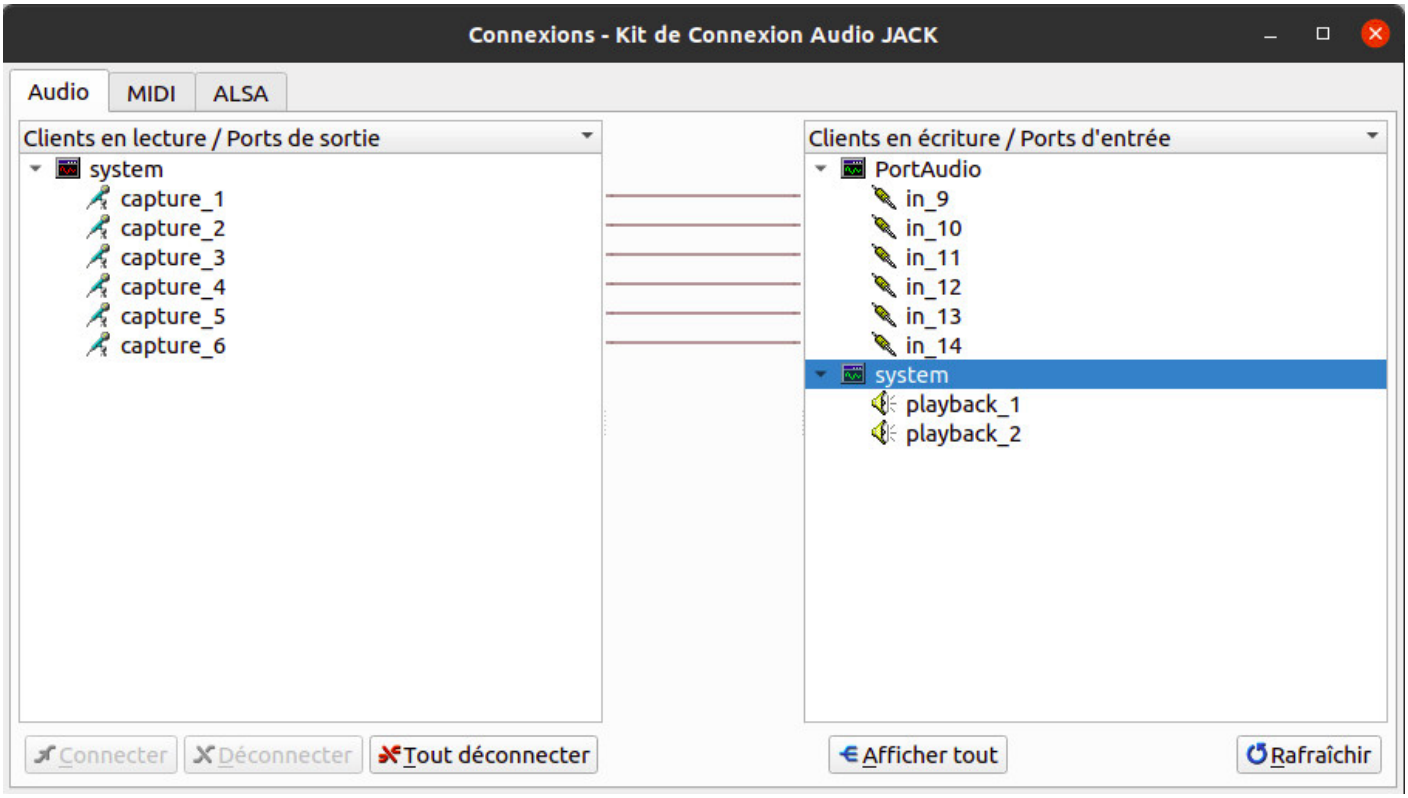
Avez-vous essayé d'**enregistrer du son avec Linux** ?

J'utilise pour cela mon **zoom H6**, que je connecte à **JACK** via la commande :

```
jackd -d alsa -d hw:H6
```

JACK Audio Connection Kit est un système qui centralise les bus audio au sein de Linux. Pour le manipuler, outre la ligne de commande pour les fonctions avancées, une petit utilitaire vous permet de le configurer à la souris : **qjackctl** .

Il permet par exemple de **rerouter les bus** (*via connecter*) :



Ici, **les 6 entrées de ma carte son** sont listées dans *system*, à gauche. À droite, en entrée, **elles sont envoyés** vers **Audacity**, ici dénommé « PortAudio ». Celui-ci a été réglé pour fonctionner avec JACK. C'est également possible pour de nombreux autres logiciels audio : **Ardour5**, **LMMS**, **Hydrogen**, **SuperCollider**, etc.

Rien ne sera joué sur mes enceintes car la sortie jack (le vrai câble, cette fois), dénommée ***system***, à droite, **n'est pas relié**.

Certains logiciels vous demandrons d'**activer la gestion de processus en temps réel** pour fonctionner. *Attention, cela implique des risques de sécurité*.

Pour ce faire, **si vous avez les droits d'administration**, commencez par **ajouter votre nom d'utilisateur au groupe audio**, à l'aide de la commande :

`sudo usermod -a -G dialout utilisateur`

Ensuite, il faudra **modifier le fichier */etc/security/limits.conf*** :

`sudo nano /etc/security/limits.conf`

Et **rajouter ces paramètres** avant *# End of file* :

```
@audio - rtprio 90
@audio - nice -10
@audio - memlock 500000


# End of file
```

Il vous faudra **ensuite quitter votre profil et vous reconnecter**. Redémarrez le PC tout simplement ou utilisez *exit*.

Pour **vérifier l'inscription dans un groupe** :

`nano /etc/group`

https://jackaudio.org/

JAPA : le spectrogramme sous Linux

Japa est un petit utilitaire pour Linux qui vous permet de **visualiser le spectrogramme** de la musique que vous créez ou écoutez.

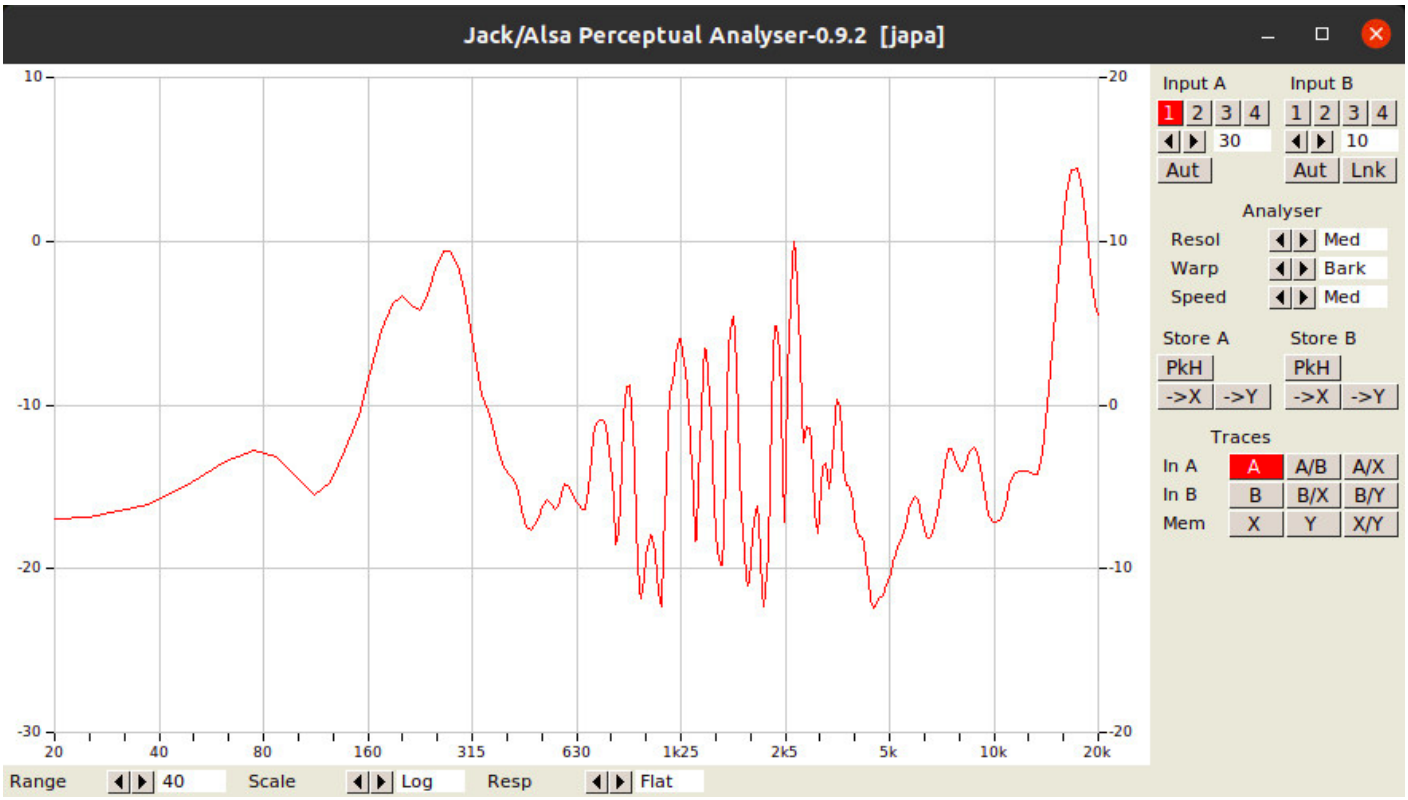
Vous pouvez l'installer à l'aide de :

```
sudo apt-get install japa
```

Pour le lancer, il faudra lui spécifier le serveur sonore que vous utilisez, avec la balise -A ou -J, pour Alsa ou Jack. J'utilise personnellement plutôt Jack pour travailler l'audio :

```
japa -J
```

Avec Alsa, la connexion est automatique, mais **pour Jack, il vous faudra rerouter l'audio vers Japa**, par exemple avec *QjackCtl*. C'est en fait assez pratique pour monitorer les applications individuellement.


Jack/Alsa Perceptual Analyser-0.9.2 [japa]
10
0
-10
-20
-30
20
40
80
160
315
630
1k25
2k5
5k
10k
20k
Input A
1 2 3 4
30
Aut
Input B
1 2 3 4
10
Aut Lnk
Analyser
Resol Med
Warp Bark
Speed Med
Store A
PkH
->X ->Y
Store B
PkH
->X ->Y
Traces
In A A A/B A/X
In B B B/X B/Y
Mem X Y X/Y
Range 40
Scale Log
Resp Flat

La Théorie Musicale

Tout plein d'idées pour parler d'un plaisir ineffable !

Qui a inventé la musique ?

La question peut paraître étrange, mais la réponse pourrait vous surprendre. Un jour, on m'en a donné **trois réponses...**

La première, et peut être la plus sensée, pourrait être de répondre que **personne n'a vraiment inventé la musique**. C'est un art qui date de la préhistoire, et dont les origines n'ont donc pas été documentées. Nous sommes dans la supposition. Il semble probable que l'art, lié au plaisir, soit hérité des principes de séduction de la sélection naturelle. Avez-vous déjà vu une araignée danser ? On peut aussi supposer que la musique est liée au langage, et peut-être que nos ancêtres ont communiqué à distance à l'aide de tambours, bien avant l'invention des sms ?

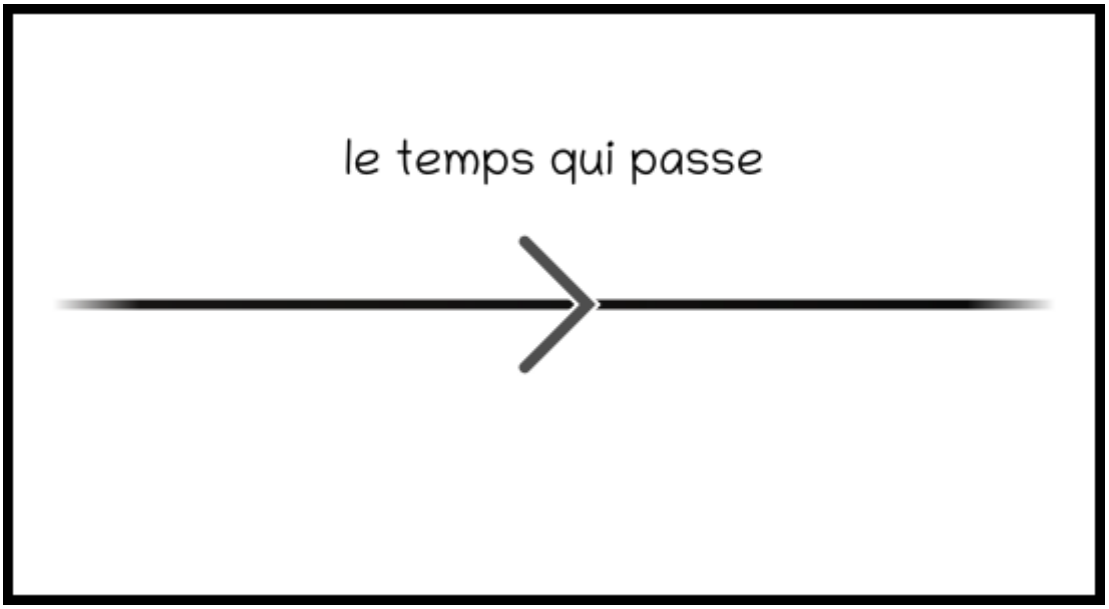
La deuxième réponse est mythologique. À une époque lointaine, il n'existait pas d'historien pour lever le voile des origines humaines, et on palliait à ce manque de connaissance par la transmission d'histoire orales, dîtes mythologies, qui expliquaient à leur manière l'origine du monde. Dans la mythologie grecque, **ce serait le dieu Apollon qui aurait transmis la musique, entre autres, à Orphée**. Ce dernier, à l'aide de sa lyre, était réputé pouvoir charmer les animaux et animer les objets à l'aide de la musique. C'est de cette manière qu'il put ensuite passer Cerbère pour rencontrer Hadès, dieu des enfers, afin de retrouver sa dulcinée, Eurydice...

La troisième réponse est plus étrange. On raconte que **Pythagore**, se promenant un jour, entendit des sons harmonieux. Il s'agissait en fait de forgerons dont les marteaux battaient enclumes. Ayant pesé les outils, il se rendit compte que les marteaux dont les sons allaient bien ensemble avaient en fait des relations de poids mathématiques particulières. Notamment, les deux sons les plus harmonieux concernaient deux marteaux dont le poids de l'un était le double de l'autre. Pythagore se rendit compte que l'on pouvait étendre ce principe à une corde vibrante. En ce sens, Pythagore est vu comme le premier à avoir décrit la musique comme un système dont on pourrait tirer des connaissances. C'est donc l'inventeur, bien que le terme soit anachronique, de la musicologie.

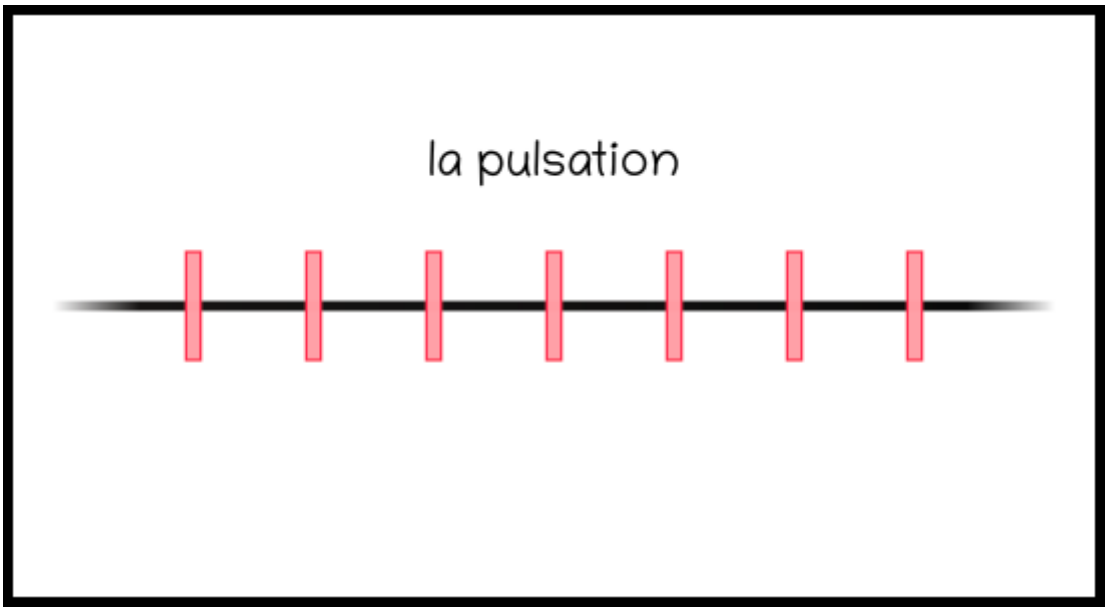
Qu'est-ce que le rythme ?

La grande majorité des musiques que nous écoutons sont rythmées. Même si nous avons **une compréhension intuitive** de cette notion, celle-ci est en fait assez complexe à bien comprendre.

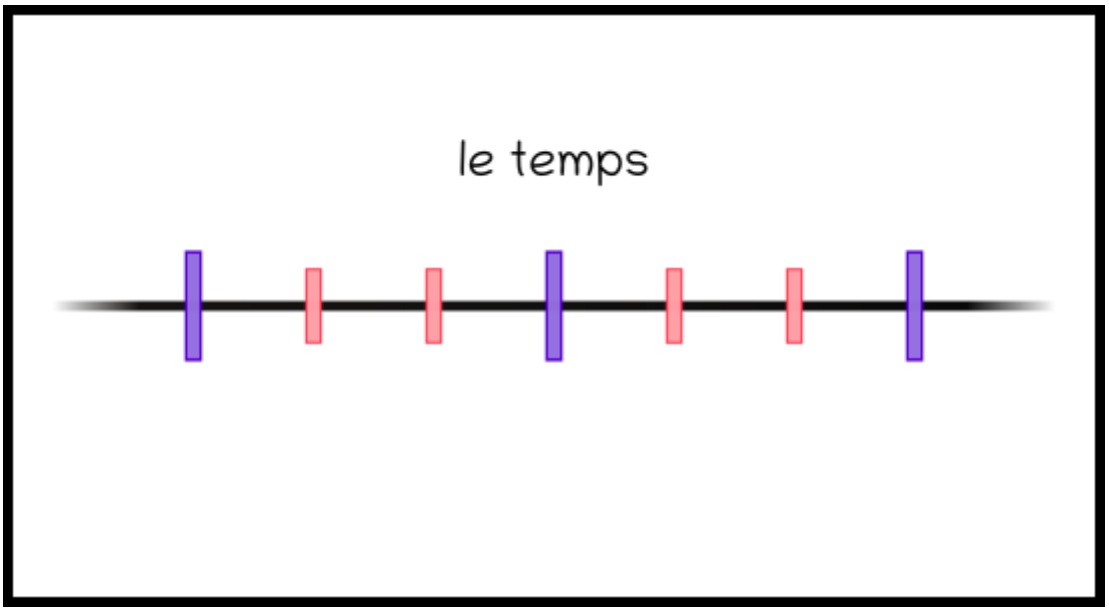
Pour pouvoir parler de rythme, il faut bien entendu commencer par penser **au temps qui passe**. Commençons par le *représenter*, comme un axe, avec une flèche qui indique le sens dans lequel celui-ci avance :



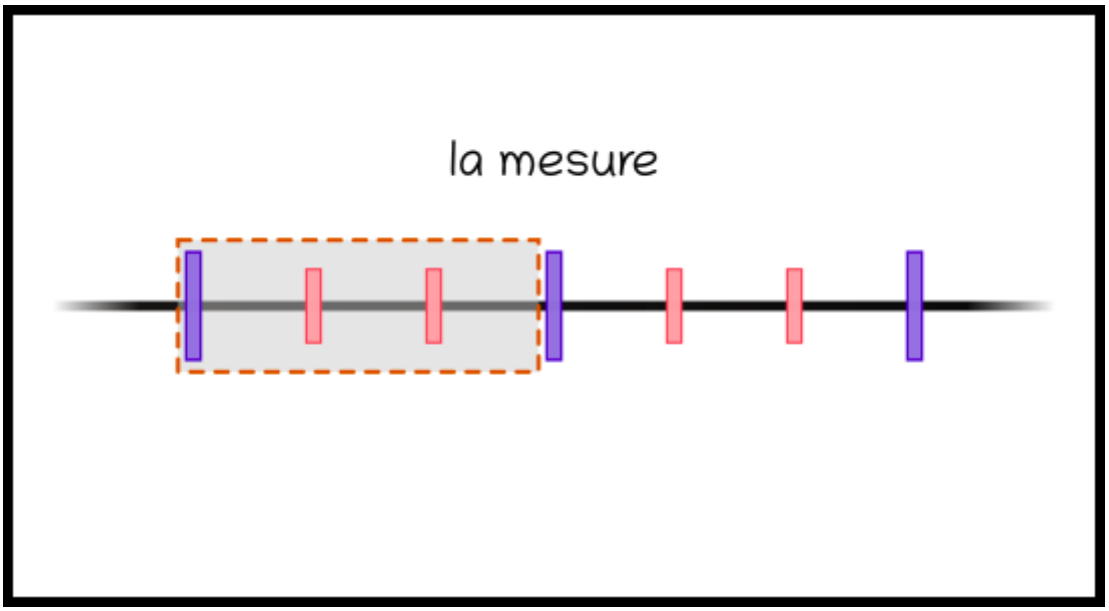
Le principe premier une fois le temps posé est que **celui-ci peut-être divisé en durées égales**. C'est le principe du métronome, qui émet un *clic* ou un *bip* à intervalles réguliers :



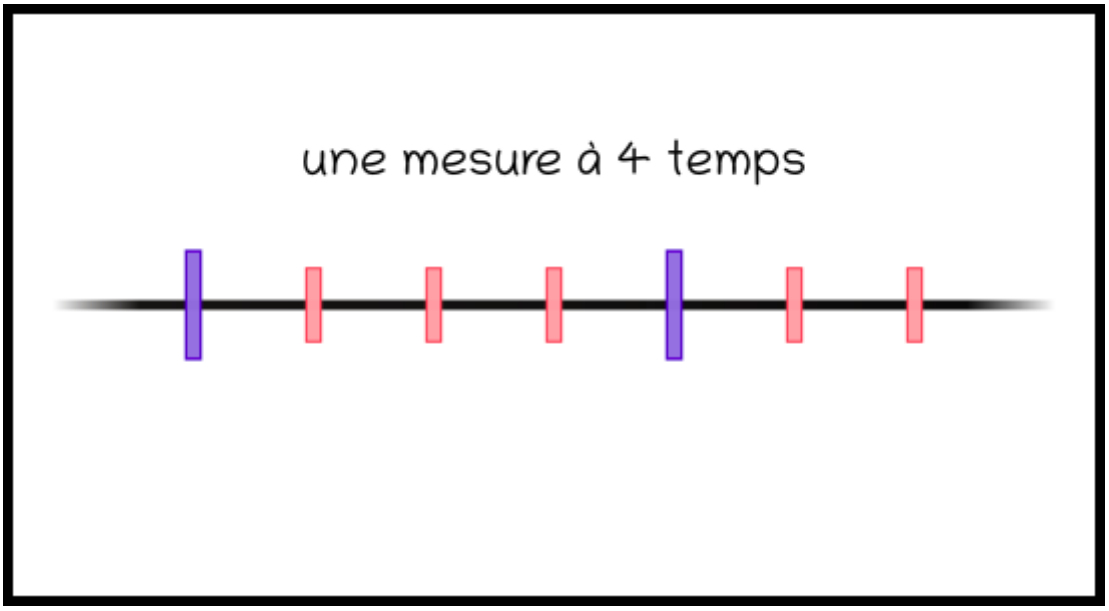
C'est ce que l'on nomme la *pulsation*, que le musicien bat souvent du pied. Pour l'instant, il n'y a pas de hiérarchie entre les notes. Que se passe-t'il si nous en rajoutons ?



Nous avons là un concept crucial, celui de **temps**. En général, **il y a un temps fort suivi d'un ou plusieurs temps faibles**. Dans l'exemple ci-dessus, j'ai marqué les temps forts en bleu. Si nous devions donner le rythme avant de commencer, nous dirions " 1 - 2 - 3 - 1 - 2 -3 " . Le temps fort nous donne le début de la *mesure* :

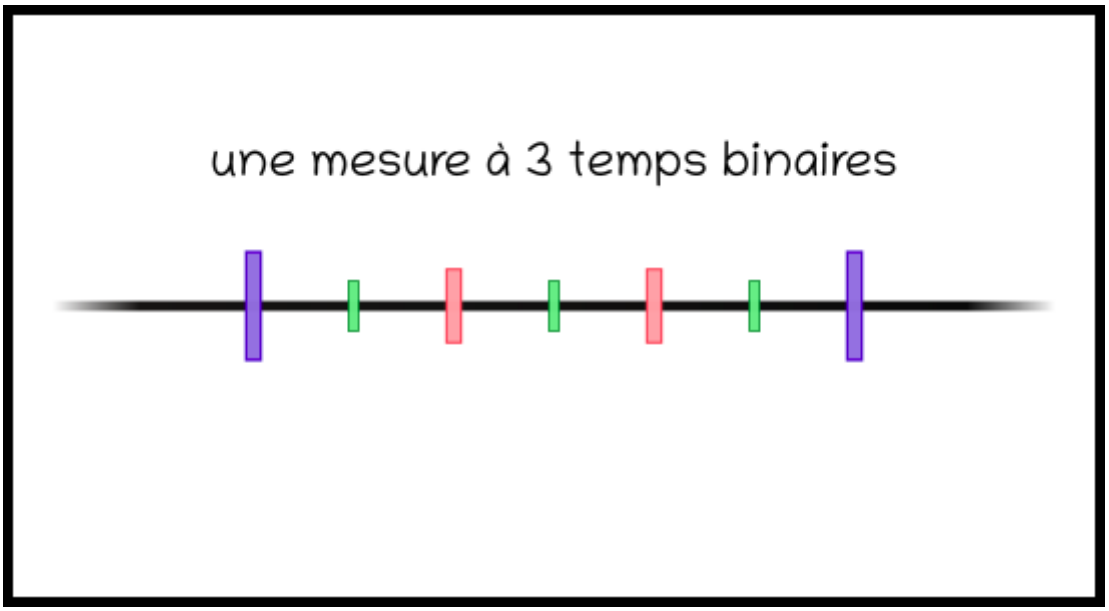


Une musique sur mesure à 4 temps ressemblera donc à ceci :

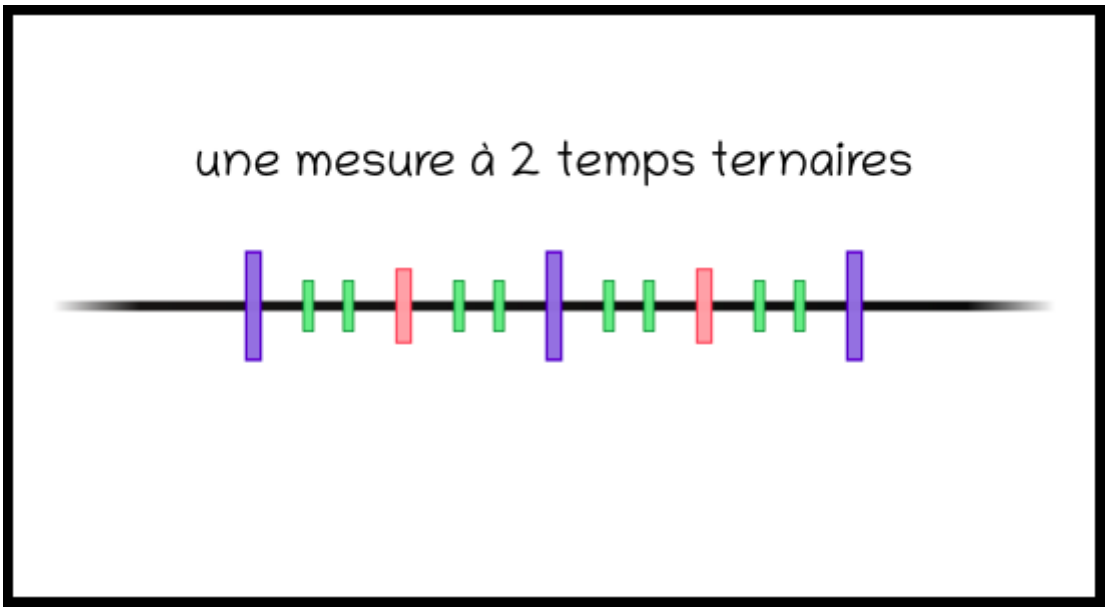


Qu'est-ce qui distingue le temps fort en musique ? Il est souvent accentué, et on a tendance à y change d'accord. Il relance la musique, c'est là que le danseur pose le pied.

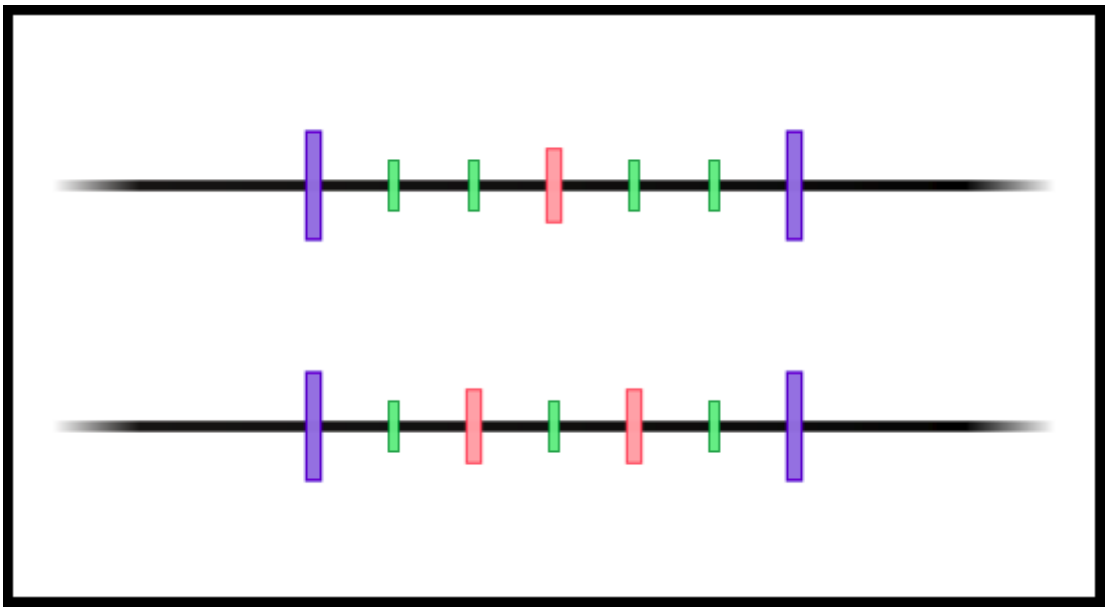
La prochaine étape consiste maintenant à **redécouper nos temps en parties égales !**



Comme vous le voyez, nous avons maintenant 1 mesure divisée en 3 temps qui sont divisés en 2 chacune ! Il est également possible de diviser en trois :



Une mesure a 3 temps binaires contient le même nombre de subdivisions qu'une mesure à 2 temps ternaires !



C'est ce qui amène beaucoup de gens à croire que **les valses** sont à trois temps ! En réalité, **elles sont à deux temps, mais *ternaires* !**

En général, **ce sont surtout des divisions en 2, 3 et 4 qui sont utilisées** par les musiciens. Les nombres impairs, comme 5, 7 ou 13 sont assez rarement utilisés, sauf dans certaines traditions musicales comme la musique des balkans, par exemple.

Quelles sont les caractéristiques physique du son ?

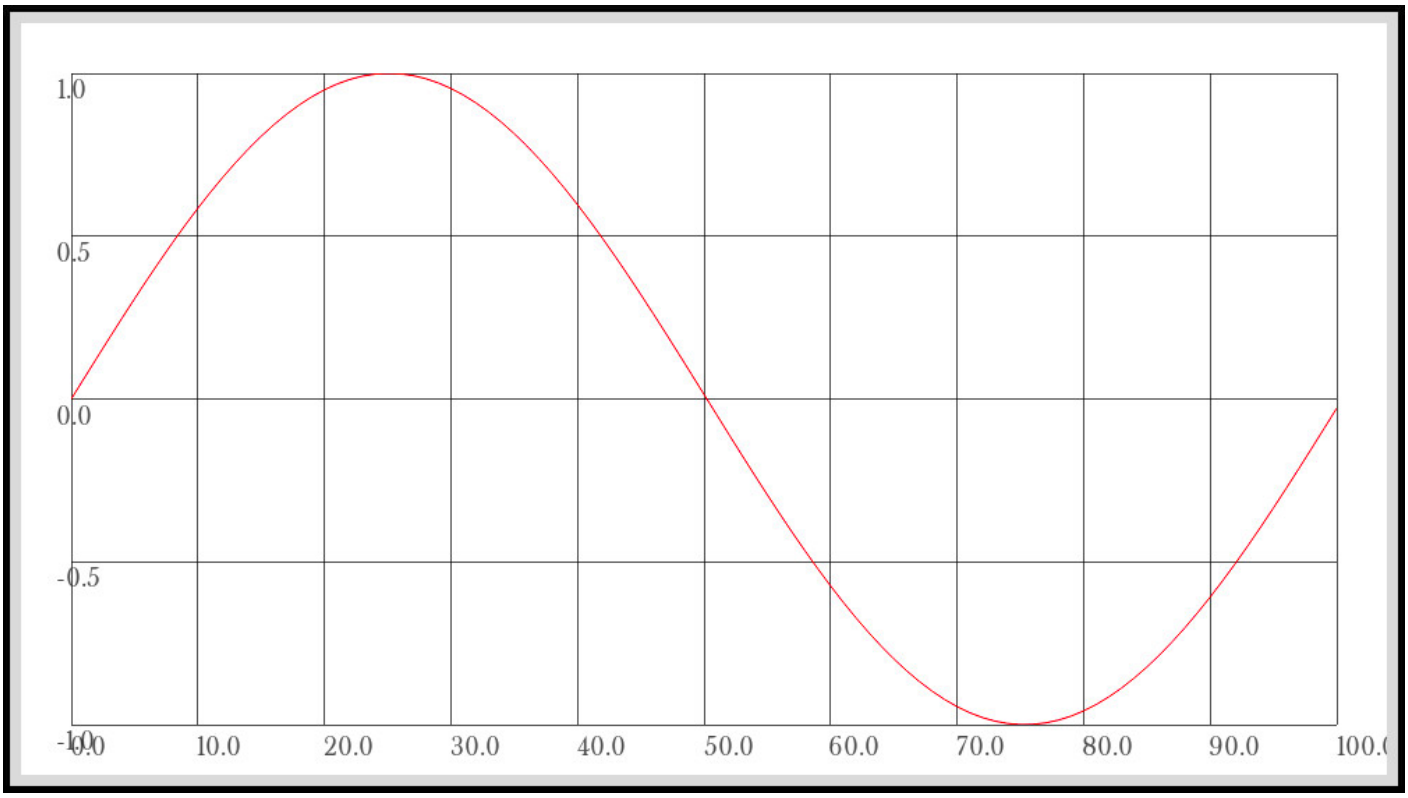
Domaine délaissé par beaucoup de musiciens, **la physique du son est pourtant une discipline qui permet de mieux comprendre la musique**, et d'ouvrir ses recherches à de nombreuses possibilités. Plus avant, l'omniprésence de l'amplification implique qu'aujourd'hui, à de rares exceptions près, **même les musiciens 'acoustiques' sont impactés par le travail de sonorisation**. Le but de cette page est de présenter les concepts fondamentaux liés à cette physique.

Fait plutôt connu, **le son est par nature une onde**. Contrairement au photon, qui est une onde en quelque sorte par lui-même et peut donc se déplacer dans le vide, **le son est une onde qui s'inscrit en fait en creux au sein d'un milieu donné**, en général dans l'air qui nous entoure. Cela explique que, contrairement aux batailles tonitruantes de vaisseaux spatiaux que l'on peut voir dans les films, le son ne se propage pas dans l'espace (l'espace dont on parle étant ici ce qui se trouve hors de l'atmosphère, et qui est en fait le vide).

La propagation de cette onde nécessite une origine, et, traditionnellement, **c'est l'action du musicien de faire vibrer une corde ou une membrane** qui la déclenche. Aujourd'hui, **ce peut également être un signal électrique qui fait vibrer des enceintes**. Lors de cette vibration, la membrane va commencer par pousser l'air qui se trouve directement à côté d'elle, puis revenir en arrière. L'air déplacé de cette manière va lui-même pousser la masse d'air qui le suit, puis revenir en arrière du fait de la dépression d'air créée derrière lui par le retour de la membrane. Cette action va alors se répéter en s'amenuisant et, si l'oreille d'un auditeur se trouve sur le chemin de ce mouvement d'air, cela fera vibrer son tympan, provoquant chez lui l'impression subjective d'un son. **Le son est donc un mouvement de surpressions et de dépressions qui se transmet dans un milieu ambiant**, à une vitesse donnée, dite vitesse du son. Au niveau de la mer, celle-ci est égale à 340,29 m / s.

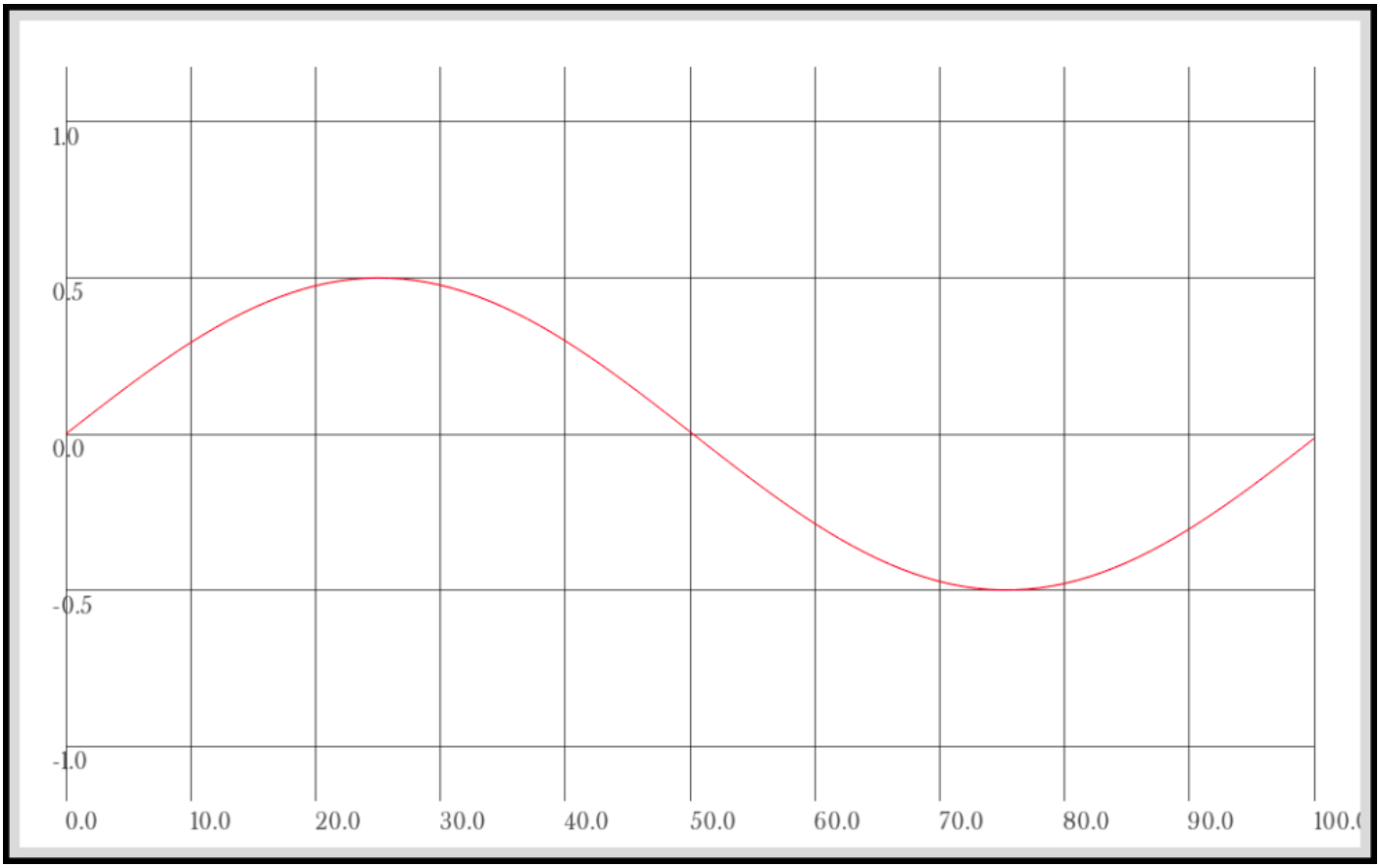
Fig a : musicien faisant vibrer l'air ambiant.

Comment peut-on décrire cette onde ? La représentation commune de l'onde sonore est la suivante :



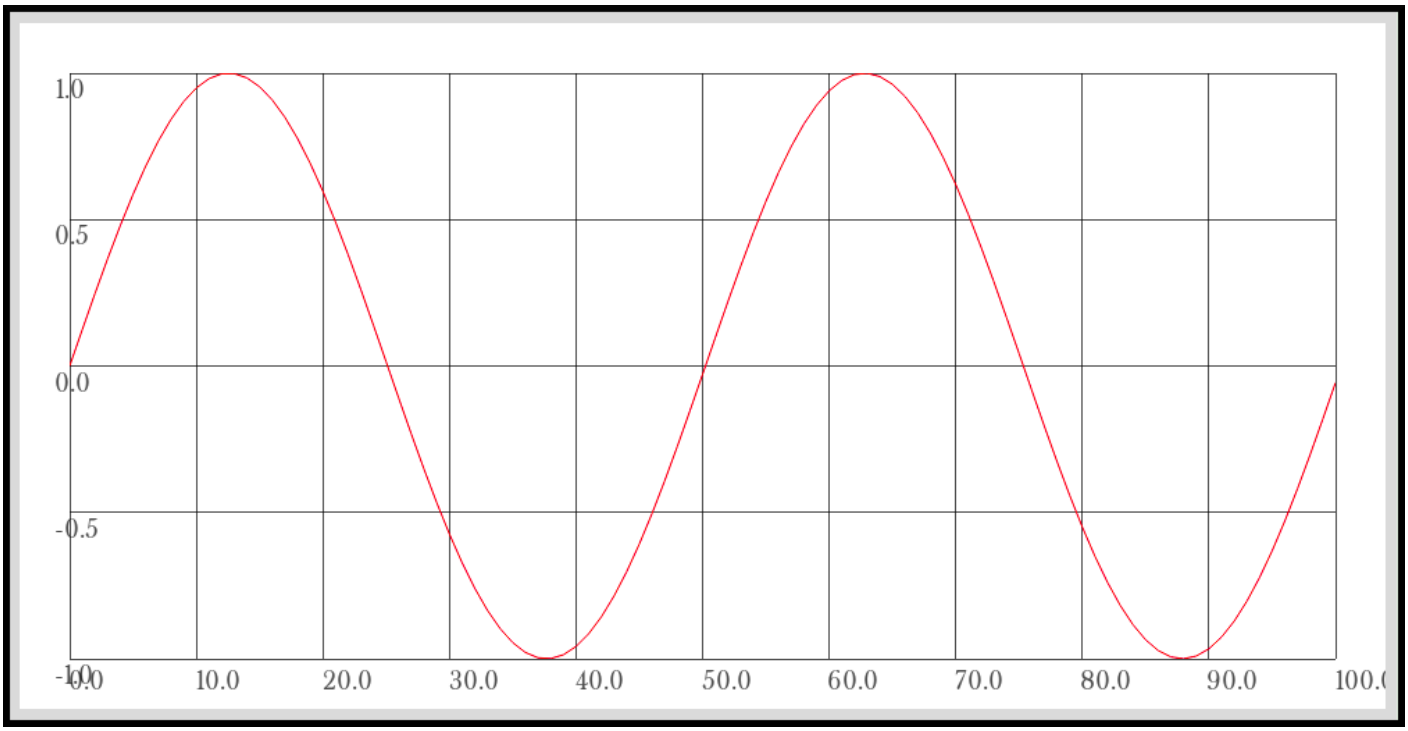
Il s'agit d'un tableau en deux dimensions, et donc définit par deux axes : **en Y**, verticalement, il s'agit de **l'amplitude de l'onde**, c'est-à-dire le déplacement de l'air par rapport à l'endroit où il se trouvait à l'origine. Horizontalement, **en X**, il s'agit **du temps qui passe**, dont nous mentionnerons les échelles un peu plus tard. Le tableau montre donc **la variation de la position de l'air dans le temps par rapport à sa position initiale**.

Deux paramètres peuvent varier si nous gardons cette forme d'onde :



Ici, l'onde met autant de temps que précédemment à revenir à sa position initiale, mais **l'amplitude est deux fois moins grande**. De la manière dont nous percevons un son, **cette amplitude correspond au volume sonore : le son que nous avons ici serait donc perçu deux fois moins fort que le son précédent**.

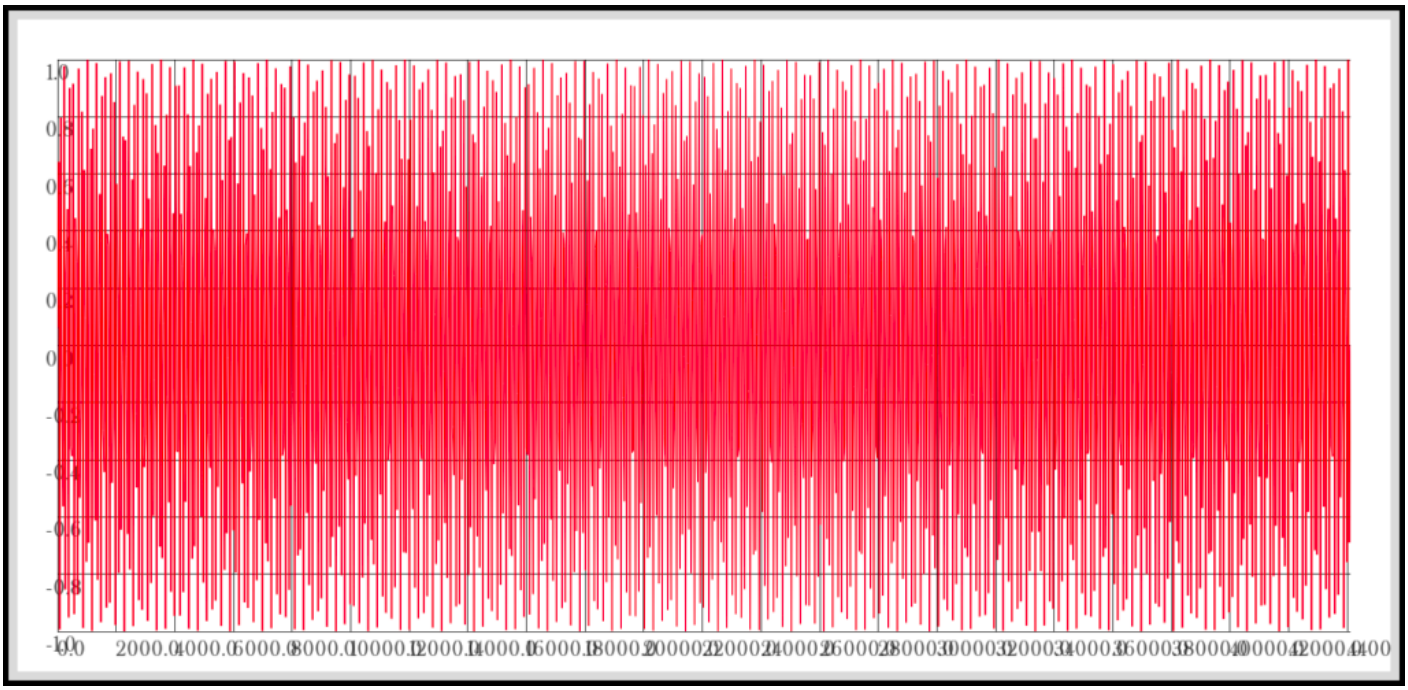
Le second paramètre qui peut varier est la fréquence à laquelle cette onde oscille :



Ici, **dans le même temps qu'au premier exemple, l'onde a effectué deux oscillations**. Il s'agit du paramètre nommé **fréquence** et **qui correspond à notre perception du grave et de l'aigu**. Le son ci-dessus, ayant une fréquence deux fois plus rapide que le premier exemple, **il nous paraîtrait deux fois plus aigu**.

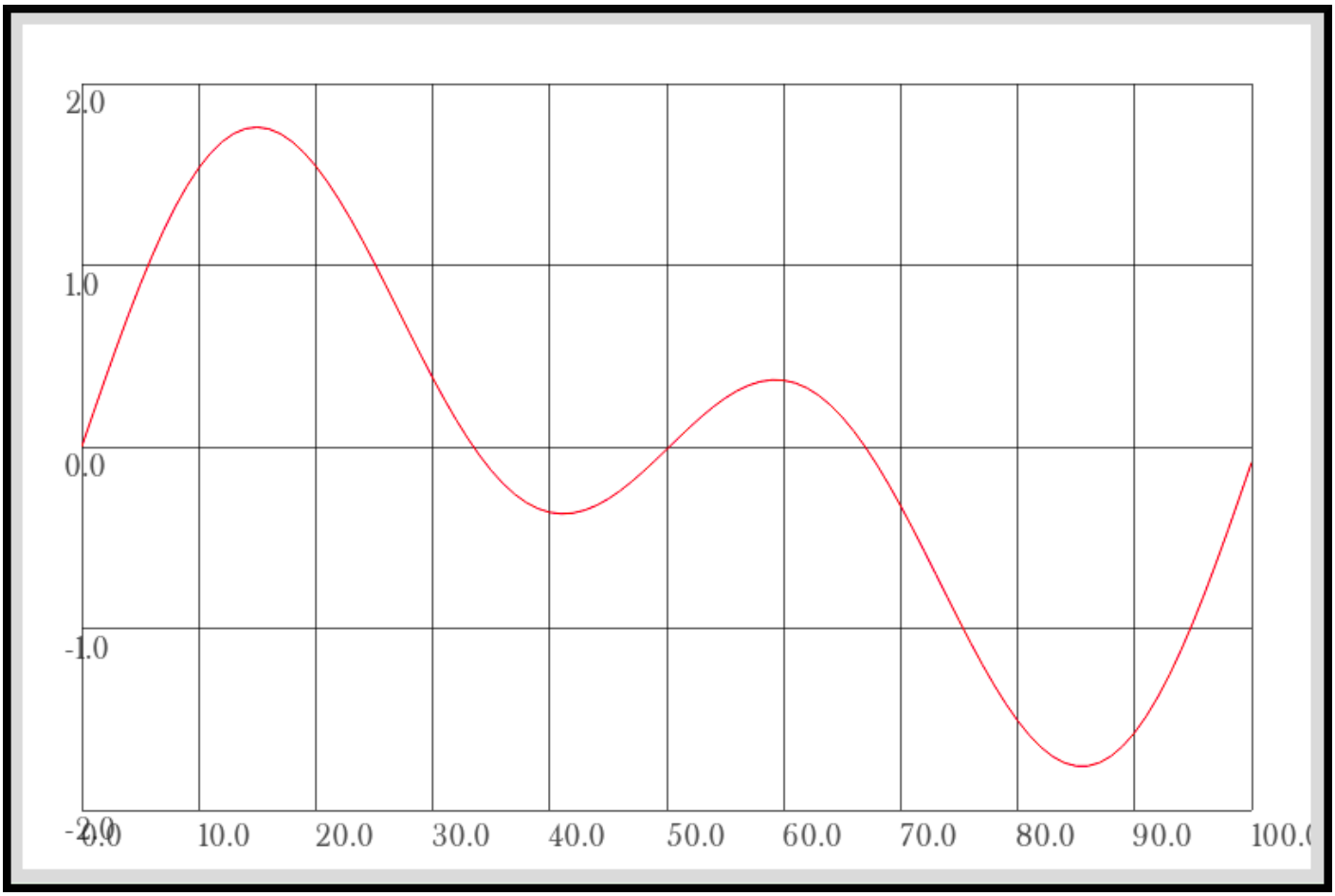
En terme de sémantique, nous avons vu que **la variation verticale de l'onde s'appelle l'amplitude**. Elle peut être négative ou positive. **Horizontalement, nous parlons de la fréquence** de l'onde, qui est **mesurée en Hertz**, c'est-à-dire en nombre d'oscillations par seconde. Une seule oscillation, c'est à dire le mouvement que fait l'onde pour monter, puis descendre et revenir à son point de départ se nomme une **phase**. **La fréquence est donc égale au nombre de phases de l'onde par seconde.**

Le premier exemple que j'ai pris oscille à une fréquence de 440 Hz, c'est-à-dire qu'il dure environ 0.002 seconde ! Le même tableau sur une seconde entière ne permet plus de discerner les ondes, puisqu'il y en a 440 :



Vous noterez par ailleurs que l’échelle en bas n’est en fait pas indiquée en secondes. Le tableau est une représentation numérique d’un signal sonore, qui ne peut être continue, comme dans le monde réel. Le son est donc constitué d’un certain nombre de points par seconde, ici 44 100. On appelle ce nombre **la fréquence d’échantillonnage**, donnée en mesures par seconde, c’est donc également en **Hertz** que cette mesure est exprimée.

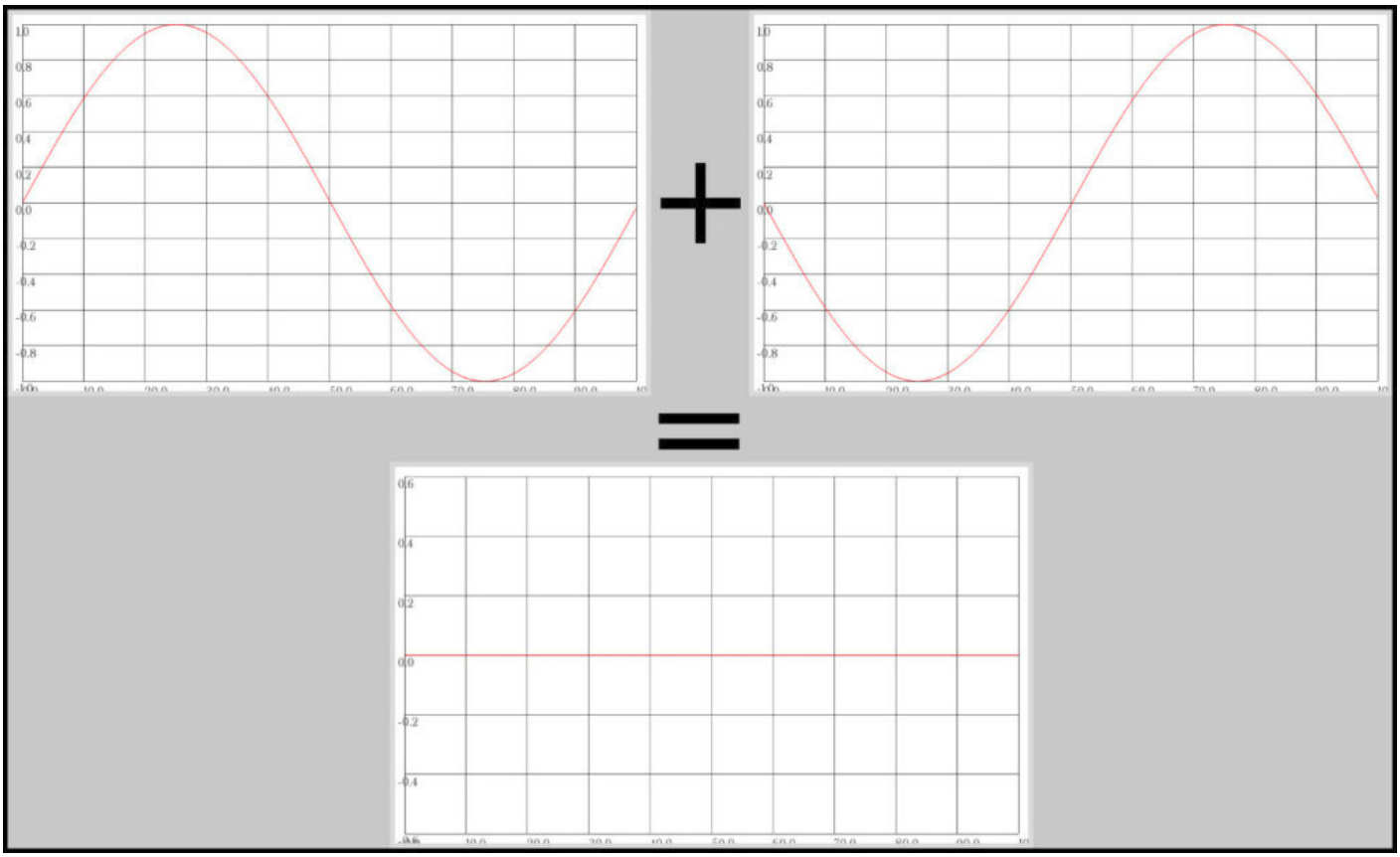
Que se passe-t-il lorsque deux musiciens jouent ensemble ? L’air qui vibre à endroit donné est à la fois déplacé par la vibration qu’émet l’un, et par la vibration qu’émet l’autre. En d’autres termes, **lorsqu’il y en a plusieurs, les ondes s’additionnent**.



Ci-dessus l'onde résultante de **l'addition d'une onde à 440 Hz et d'une onde à 880 Hz**. Les amplitudes s'étant additionnées, on dépasse maintenant l'amplitude de 1 que nous avions avant. En d'autres termes, ajouter des sons augmente le volume sonore. **Les deux fréquences**, par contre, **restent audibles**, car l'addition à fait émerger une crête qui correspond à chaque onde que nous avions au départ. **Créer de nouveaux sons en en additionnant s'appelle la synthèse sonore additive**.

Fig b : principe de synthèse sonore additive.

Une conséquence directe de ce principe, assez déroutante, est la suivante :



Émettre une onde en même temps qu'une onde de même fréquence dont la phase est décalée de 180°, c'est-à-dire commençant vers le bas (vers une amplitude négative) **ne produit aucun son** ! Ce qui est logique puisque les ondes étant les même excepté le signe de leur amplitude, **la somme de leurs valeurs est toujours égale à zéro** !

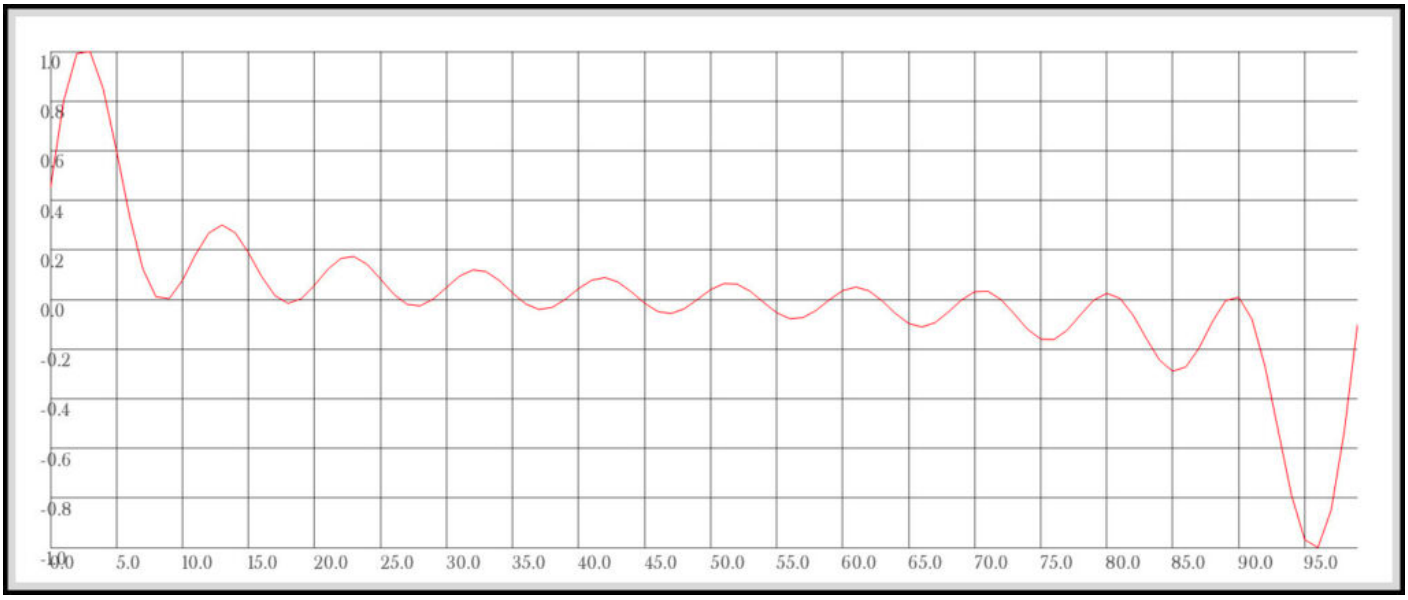
On appelle ce principe **l'opposition de phase**, qui est utilisé dans certains milieux professionnels pour réduire le bruit des machines, par certains casques audio pour réduire le bruit ambiant, ou encore en *design sonore* pour éliminer des bruits parasites.

Revenons à notre onde à 440 Hz. Cet exemple n'a pas été choisi au hasard. **440 Hz est la fréquence de référence du diapason**, utilisée pour s'accorder, **et correspond au La** du milieu du piano. En fait, ce diapason a évolué au fil du temps. À l'époque baroque, il était fixé à 435 Hz, et, aujourd'hui, la plupart des musiciens professionnels s'accordent à 442 Hz. Il est important de noter que **si la fréquence du diapason change, il reste cependant *toujours* un La**. Le nom des notes ne renvoie donc pas forcément à la même fréquence en fonction des époques. Nous laisserons la question des notes de côtés pour le reste de cet article, mais notons que *le La étant à*

440 Hz, la fréquence 880 Hz correspond au La à l'octave supérieure (et donc 220 Hz correspond au La à l'octave inférieure).

À quoi ressemble le La du premier exemple lorsque nous l'écoutons ? Pour ceux qui ont connu le téléphone fixe, il s'agit de la tonalité que nous avions avant d'avoir composé un numéro. *C'est un son dit « pur » car c'est le plus simple que l'on puisse trouver*, et il correspond à la description d'une fonction mathématique simple qui s'appelle Sine. **Cette onde s'appelle une onde sinusoïdale**. Par ailleurs, *tout son plus complexe peut lui-même être décrit comme une somme d'ondes sinusoïdales* de fréquences différentes, selon le principe des *transformées de Fourrier*.

En réalité, les sons du monde qui nous entoure sont très riches et ne contiennent pas qu'une seule onde : en fonction de la matière qui les composent, **ils contiennent également des ondes qui sont les multiples entiers de l'onde la plus basse** qu'il émettent en vibrant. **On appelle ces ondes des *harmoniques*** :



Ici, un exemple (théorique) d'une onde et de 9 de ses harmoniques. La première oscille à 440 Hz, la deuxième à 440 *2 = 880 Hz, la troisième à 440 * 3 = 1320 Hz, etc…

En fait, **certaines harmoniques ressortent plus ou moins**, en fonction du matériau de l'objet ou de la présence d'une caisse de résonance. **C'est ce qui explique la différence de son** entre une trompette et un saxophone, même lorsqu'ils jouent la même note. J'ai pris cet exemple car le mode d'émission du son joue aussi un rôle dans les fréquences émises. On entend également le souffle du saxophoniste lorsqu'il joue, ou le bruit des maillets sur un xylophone en plus de la résonance des lamelles. **Cette différence entre les types d'ondes émises est appelé le timbre, c'est la texture qui distingue un son d'un autre**.

Nous finirons cet article en énonçant que **du fait de la nature même d'un son, celui-ci a nécessairement quatre paramètres sans quoi il ne pourrait advenir** :

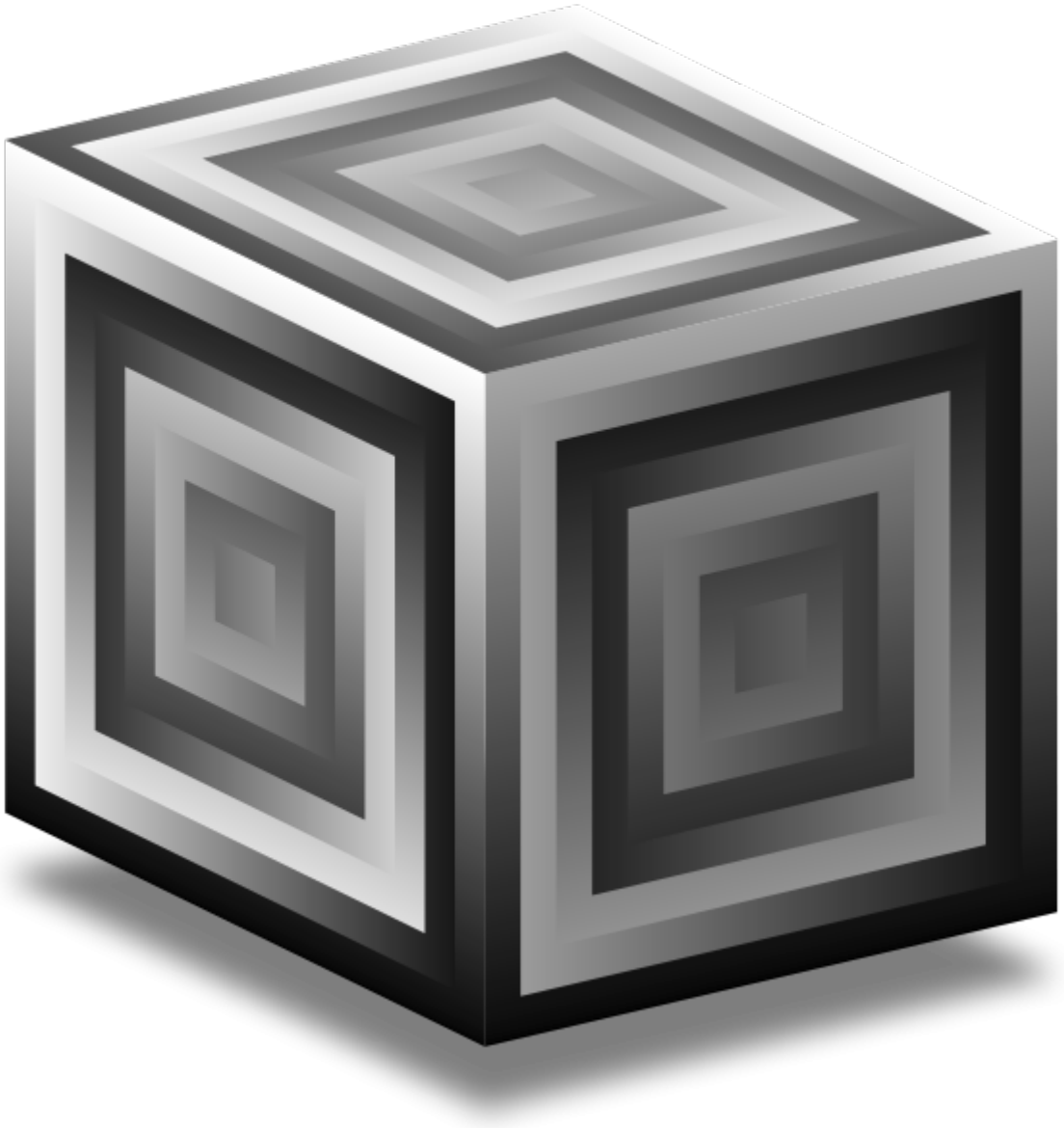
- Le premier est l'**amplitude**, c'est-à-dire son volume sonore.
- Le second est sa **durée**, jusqu'à ce que celui-ci cesse d'être émis.
- Le troisième est sa **fréquence**. Elle est au maximum égale à la durée du son.
- Le quatrième est le **timbre**, c'est-à-dire la forme de l'onde.

Découvrir SuperCollider

Faire de la musique avec du code ?

Qu'est-ce que SuperCollider ?

SuperCollider est un langage de programmation qui permet de coder des instructions musicales.



En fait, il s'agit d'un projet un peu plus complexe : **celui-ci embarque un IDE, qui permet de piloter un serveur dédié à faire du son**. Le logiciel utilise extensivement le protocole de

communication OSC.

Vous pourrez trouver un sacré paquet d'exemples tous prêts ici !

Page suivante : Comment bien démarrer avec SuperCollider ?

(*) Comment bien démarrer avec SuperCollider ?

> (*) : Il manque des informations. Cette page est en cours de construction et n'est donc pas finalisée.

Avant toute chose, SuperCollider est un outil puissant, mais **qui nécessite des bases solides en programmation et en acoustique** pour être utilisé. Si vous ne savez pas encore déclarer une fonction, ou que vous ne savez pas pourquoi émettre deux sons simultanés à 440 et 447 HZ est un choix esthétique fort, je vous conseille de vous documenter sur ces deux sujets, et de commencer par composer sur Sonic Pi, qui est une excellente porte d'entrée à SuperCollider.

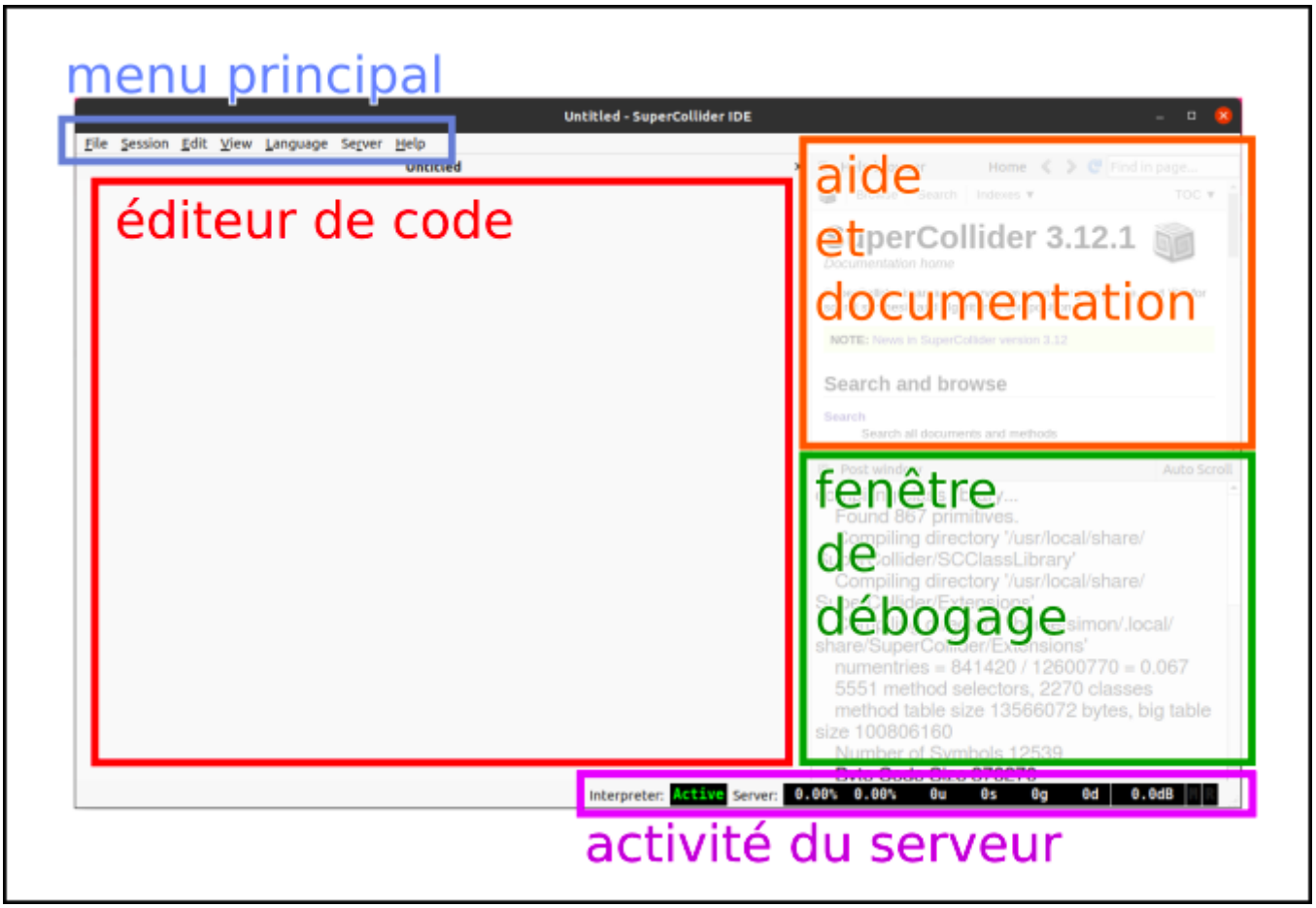
Si vous êtes d'attaque, il faut d'abord commencer par l'installer !

Pour les utilisateurs de Linux, vous pouvez le trouver depuis le gestionnaire de paquets :

```
sudo apt-get install scide
```

Lançons maintenant le logiciel !

Voici l'interface qui apparaît au lancement :



Avant de détailler les différents espaces, revenons sur un point assez important. **SuperCollider n'est pas un logiciel construit que d'un seul bloc.** Vous pouvez l'utiliser sans ouvrir la fenêtre ci-dessus. Pour simplifier, il y a **deux briques fondamentales** : le **serveur**, qu'on appelle communément *SuperCollider*, et l'**IDE**, ici *scide*, qui correspond à la fenêtre. Le serveur s'occupe de générer la musique, et l'IDE permet de piloter le serveur. Vous pourriez en fait piloter le serveur depuis n'importe quel langage de programmation, simplement en utilisant les bonnes commandes. De manière plus approfondie, le serveur s'appelle *scsynth* et le langage de l'IDE s'appelle *sclang*. Mais cela n'a pas d'importance pour débuter. Ce qu'il faut retenir pour l'instant, c'est surtout la présence de ce fameux **serveur**.

L'IDE de base, et le langage de programmation associé, est développé spécialement pour manipuler le serveur, ce qui en fait un environnement idéal pour débuter.

Voyons rapidement les différents espaces de l'IDE, qui apparaissent au lancement, et que l'on peut voir sur l'image ci-dessus :

- le *menu principal* vous permet les opérations communes : manipulation de fichiers, paramétrages, infos...
- l'*éditeur de code* vous permet... d'éditer du code !
- la *fenêtre d'aide et de documentation* intègre la documentation de SuperCollider.

- la *fenêtre de débogage* est un terminal qui affiche les messages internes, comme les erreurs, et les sorties que vous souhaitez afficher.
- la *fenêtre d'activité du serveur* indique un certain nombre d'information à propos du serveur.

Nous allons maintenant **cheminer tranquillement à travers les opérations basiques que vous pouvez effectuer avec l'IDE**. La plupart d'entre elles sont liées à **des raccourcis claviers**, que vous devrez sans doute mémoriser tant vous aurez à vous en servir.

Premièrement, vous serez souvent amené à **vous servir de la *fenêtre de débogage***, qui vous permettra d'obtenir des informations intéressantes sur ce qui se passe, notamment sur ce qui se passe mal...

Il y a déjà des informations qui y sont affichées : **lorsque l'IDE démarre, il vous donne des informations sur la manière dont il a chargé SuperCollider**. Par exemple la dernière ligne :

```
SCDoc: Indexed 1925 documents in 1.21 seconds
```

m'indique qu'il a trouvé 1925 pages de documentation, et qu'il les a chargées en 1 secondes 21.

Pour **remettre à zéro la *fenêtre de débogage***, on utilise la commande **CTRL + MAJ + P** . C'est pratique pour éviter que les informations s'accumulent !

Voyons maintenant **comment afficher des informations dans la fenêtre de débogage**.

Pour ce faire, il vous faudra vous servir de l'*éditeur de code*. Commençons par écrire un programme "Hello World", dont le but est simplement d'afficher "Hello World" dans la fenêtre de débogage. Voici la commande qui le permet, qu'il faudra taper dans l'*éditeur de code* :

```
"Hello World".postln;
```

Ensuite, pour lancer le code, il faudra utiliser la commande **CTRL + ENTRÉE** .

Si vous avez eu de la chance, le texte s'est mis temporairement en surbrillance, et "Hello World" s'est affiché deux fois dans la *fenêtre de* débogage. Sinon...

Il est nécessaire de donner quelques explications sur ce que nous venons de faire.

Premièrement, **le langage que nous utilisons est un langage dit "orienté objet"**. Cela indique qu'**il est constitué uniquement d'*objets* qui répondent à des *messages*.**

Dans notre exemple, l'objet "Hello World", qui est de type *String*, c'est-à-dire une suite de caractères, peut répondre au message "postln" en s'affichant dans la fenêtre de débogage.

À noter, deux syntaxes sont équivalentes dans SuperCollider pour envoyer un message à un objet :

`objet.message();`

`message( objet );`

Autrement dit, nous aurions également pu écrire

`postln( "Hello World" );`

Comme vous l'aurez remarqué, **on doit mettre des ; à la fin des phrases pour indiquer à SuperCollider la fin d'une commande**. Si cela fait sens pour vous, vous pouvez noter qu'il est possible d'omettre le dernier **;** d'un bloc.

Voyons maintenant pourquoi il est possible que "Hello World" ne se soit pas affiché lorsque que vous avez appuyé sur **CTRL + ENTRÉE**. Ce sera notamment le cas si vous êtes revenu à la ligne après la dernière commande.

Contrairement à d'autres langages de programmation, **SuperCollider ne va pas lire l'ensemble du code écrit dans l'éditeur** lorsque que vous utilisez **CTRL + ENTRÉE**. Par défaut, **il ne lira que le code situé sur la même ligne que votre curseur de texte.** C'est parce qu'il s'agît d'un langage dit interprété.

Considérons le code suivant :

```
"Je suis la première ligne".postln;
"Je suis la deuxième ligne".postln;
```

On pourrait s'attendre qu'en appuyant sur **CTRL + ENTRÉE** s'affichent successivement "Je suis la première ligne", puis "Je suis la deuxième ligne".

Il n'en est rien. Si votre curseur de texte est à la deuxième ligne, alors seule la deuxième ligne sera affichée dans *la fenêtre de débogage*.

Quel intérêt me direz-vous ? Eh bien dans le contexte musical, c'est assez pratique ma foi ! Nous pouvons avoir une ligne par instrument et les faire jouer indépendamment les uns des autres !

Mais **comment faire pour exécuter plusieurs lignes à la fois ?**

Deux solutions sont possibles : soit **sélectionner nos deux lignes et appuyer sur CTRL + ENTRÉE**, ce qui implique qu'elles soit adjacentes.

Soit, **nous utilisons des parenthèses pour regrouper du code en un seul bloc**. Dans ce cas là, **CTRL + ENTRÉE** exécutera l'ensemble du code entre parenthèse si le curseur de texte est à

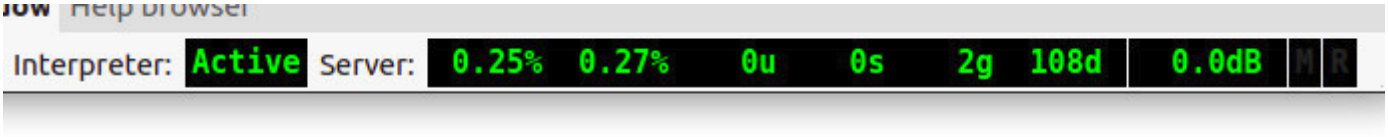
l'intérieur :

```
(
"Je suis la première ligne".postln;
"Je suis la deuxième ligne".postln;
)
```

Il est temps de faire du son ! Presque !

D'abord, il faut que **le serveur soit allumé** pour qu'il puisse produire du son. La commande qui permet de l'allumer est **CTRL + B** .

Cela est visible dans la fenêtre d'activité du serveur, qui affichera ses informations en vert lorsque le serveur est allumé :



Maintenant... **Quelques règles de sécurité. Comme tous les logiciels audio, l'utilisation de SuperCollider peut être dangereuse.** C'est principalement le cas lorsque que le volume est trop fort. Cela peut endommager votre matériel, et surtout, vos oreilles... Vous apprendrez avec le temps quels réglages sont les plus adaptés.

Pour l'instant, je vous conseille deux choses :

Premièrement, utiliser **CTRL + M** pour afficher les niveaux de sortie de SuperCollider. Cela permet d'**avoir un indice visuel de la puissance sonore, qui permet de vérifier si le son est trop fort sans risquer de s'abîmer les oreilles.**

Deuxièmement, tant que vous n'êtes pas à l'aise avec les outils de production de son de SuperCollider : **si vous n'êtes pas sûr du son qui va sortir de vos enceintes ou de votre casque : baissez le volume au maximum, c'est-à-dire jusqu'au silence. Lancez votre son. Puis augmentez graduellement le volume. Répétez pour chaque nouveau son.**

Maintenant que les règles de sécurité sont posées, voici donc enfin la première ligne de code qui va nous permettre de faire du son :

```
{ SinOsc.ar( 440, mul: 0.1 ) }.play
```

Pour **arrêter tous les son du serveur**, vous pouvez utiliser la commande clavier ***ctrl + maj + .*** .

Cette ligne de code est en fait le raccourci d'une méthode plus complexe. Nous allons voir ses composants dans le détail.

Dans SC, **les éléments à l'intérieur d'accolades { } sont une fonction**. La classe *Function* a une méthode particulière, *play*, qui permet de l'exécuter et d'en produire le son si elle contient des éléments musicaux.

Dans notre cas, nous avons en effet utilisé une des briques de bases du son dans SC : SinOsc.

SinOsc est la classe permettant de générer des signaux sonores sinusoïdaux.

Voici une version plus modulable de la ligne précédente :

```
(
{
	SinOsc.ar(
		freq: [ 440, 440 ],
		phase: 0,
		mul: 0.1,
		add: 0
	)
}.play
)
```

SinOsc prend en compte quatre paramètres : **la fréquence d'oscillation**, **le décalage initial de la phase** (normalement compris entre 0 et 2pi), **l'amplitude** (désignée comme *mul*), et un paramètre nommé *add* qui s'additionne à *mul* .

En comparant les deux exemples, vous noterez qu'il est possible d'omettre le nom des paramètres si ceux-ci sont spécifiées dans le bon ordre.

Dans ce cas particulier, j'ai utilisé une array de deux valeurs pour la fréquence. Lorsque que nous utilisons ce type d'argument, **SC créé automatiquement un nombre de canaux sonores égal à la taille de notre array** : ici, nous obtenons donc une sortie stéréo. Cette méthodologie, très utilisée dans SC, s'appelle *Multichannel Expansion*.

Dans notre cas, les paramètres de *phase* et d'*add* sont superflus, je les omettrai pour l'instant.

Dans SC, **tous les objets répondent aux méthodes mathématiques**, ce qui est extrêmement pratique pour la synthèse sonore :

```
(
{
SinOsc.ar( [ 440, 440 ], mul: 0.1 ) +
SinOsc.ar( [ 440, 440 ] * 5 / 4, mul: 0.1 )
}.play
)
```

Vous remarquerez qu'il est possible d'appliquer directement une opération mathématique à une array par ailleurs.

Pour ceux qui auraient remarqué le mystérieux *.ar* qui suit l'appel à la classe : nous y reviendrons.

Cette manière de faire est très pratique pour prototyper ou créer des drones, mais elle est loin d'exploiter la manière dont SC a été pensé.

C'est maintenant que nous allons voir comment se servir du serveur de SC.

L'idée est de créer une sorte de moule pour notre instrument, dont nous pourrons nous servir ensuite pour créer autant d'instruments que nous souhaitons.

Dans SC, cela correspond à créer une ***SynthDef*** :

```
(
SynthDef( \sineLabomedia, {
| out = 0, freq = 440, amp = 0.1 |

var snd = SinOsc.ar( freq, mul: amp );

Out.ar( out, [ snd, snd ] );

} ).add
)
```

Ok ! Premièrement, si vous êtes perdu, ***scide* vous permet d'accéder à la documentation du mot sur lequel est positionné le curseur d'écriture en utilisant le raccourci *ctrl* + *d*** . C'est notamment pratique pour se documenter sur les classes et comprendre leurs arguments, ou pour le débogage.

La *SynthDef* commence par le nom du synthé, dans un format particulier : il s'agit d'un *symbole*. Celui-ci se repère car il commence par un anti-slash (\ ) . Vous pouvez spécifier celui que vous souhaitez, mais il devrait en général correspondre à un nom clair pour l'instrument que vous

créez.

Ensuite vient la fonction qui correspond au son de notre instrument. Vous pouvez voir que j'ai indiqué des arguments à celle-ci : vous pouvez les omettre, mais c'est en général une bonne idée d'en avoir car ce sont ces arguments auxquels vous accéderez lorsque vous créerez votre instrument pour spécifier ses particularités.

Dans notre cas, cette fonction accomplit deux actions :

Une variable qui stocke un son est créée, à l'aide *SinOsc.*

Le son est poussé vers la sortie de la carte son grâce à *Out* . Vous pourrez noter qu'ici, je créé la stéréo non pas dans *SinOsc*, mais dans *Out*.

Enfin, **j'utilise la méthode *.add* sur ma *SynthDef***. C'est là l'élément important : en faisant cela, j'ajoute un nouveau modèle d'instrument au sein du serveur, que je pourrai réutiliser à l'envie *a posteriori*.

Une fois cela fait, je peux utiliser la classe *Synth* pour appeler mon synthé, en indiquant son nom :

```
Synth( \sineLabomedia )
```

Il utilisera les arguments par défaut que j'ai assigné, mais il m'est également possible d'en spécifier d'autres :

```
Synth( \sineLabomedia, [ freq: 660 ] )
```

Pour le modifier après l'avoir créé, il faudra **référencer l'instrument dans une variable**.

L'exemple commun est celui-ci, nous lançons le synthé et le référençons dans la variable *x* :

```
x = Synth( \sineLabomedia, [ freq: 440 * 3 / 2 ] )
```

Il est ensuite possible de modifier ses paramètres. Attention, il faut accéder aux arguments de la fonction en les indiquant en tant que symboles, c'est-à-dire en leur ajoutant un anti-slash :

```
x.set( \freq, 330, \amp, 0.05 )
```

Pour l'arrêter :

```
x.free
```

Il est étrange que nous puissions effectuer ces actions dans l'ordre de notre choix. C'est parce que la variable *x* est une variable *globale*.

Les *variables globales* restent en mémoire à la fin de l'exécution d'un bloc. Elles sont communes à tous les fichiers SC actuellement ouverts. Sans ce mécanisme, le code ne pourrait pas être interprété mais seulement exécuté.

Par défaut, **chacune des lettres de l'alphabet est une *variable globale* dans SuperCollider**, qui vous permet de référencer des objets constamment modifiables.

Je vous déconseille néanmoins l'usage des lettres de l'alphabet comme variables globales. Premièrement, car la variable *s* référence par défaut le serveur de SC, et qu'il est commun de réserver la variable *t* à un objet particulier. Ensuite parce qu'une lettre unique n'indique pas vraiment ce que la variable contient.

Pour créer vos *variables globales*, il suffira d'indiquer un nom précédé d'un ~ :

```
~monInstrument = Synth( \sineLabomedia, [ freq: 660 ] )
```

[EN CONSTRUCTION]

Ajouter de nouveaux synthés à Sonic Pi

***Sonic Pi* est un logiciel parfait pour** débuter **la programmation** tout en apprenant à **composer**.

Celui-ci est en fait lié à *SuperCollider* : il en reprend le principe mais **le langage est beaucoup plus simple** à appréhender.

***Sonic Pi* intègre de base de nombreux synthés que l'on peut utiliser pour faire de la musique**, à l'image de l'émulation du célèbre TB303 :

```
synth :tb303
```

Malheureusement, on fait assez vite le tour des synthés proposés par défaut, et l'on peut **souhaiter en créer nous même** pour pouvoir les utiliser.

J'ai suivi ce tutoriel pour ce faire, et il est bluffant de voir à quel point c'est facile à faire !

Voici une variante de l'exemple donné, **corrigé** (un seul anti-slash dans le symbole de la *SynthDef*) et **amendé** pour être plus facile à utiliser. Il vous faudra **créer un dossier (par exemple nommé *SCDef*), et enregistrer ce script à l'intérieur**, par exemple sous le titre *SCDefGeneration.scd*

```
(
SynthDef(\piTest, { |freq = 200, amp = 1, out_bus = 0|

    var son =  SinOsc.ar([freq,freq], 0, 0.5) * Line.kr(1, 0, 5, amp, doneAction: 2);

    Out.ar(out_bus, son);

    ).writeDefFile(thisProcess.nowExecutingPath.dirname ++ "/");
)
```

Lorsque vous interprétez ce code dans *SuperCollider*, avec *ctrl + Entrée*, un fichier binaire est créé dans le dossier *SCDef*. Celui-ci contient le synthé à utiliser par *Sonic Pi*.

Pour enregistrer une autre *SynthDef*, il faudra **la substituer à celle de l'exemple**, autrement dit copier-coller la SynthDef, du mot SynthDef(, jusqu'à la parenthèse fermante, puis rajouter ".writeDefFile(thisProcess.nowExecutingPath.dirname ++ "/");" à la fin, avant d'interpréter le code pour créer le fichier binaire.

Pour charger ce synthé dans *Sonic Pi*, il faudra interpréter la commande suivante dans *Sonic Pi* après l'avoir lancé, et ce à chaque fois :

```
load_synthdefs "/chemin/vers/SCDef/"
```

Cette commande chargera dans *Sonic Pi* l'ensemble des *SynthDefs* présentes dans le dossier.

Après cela, on peut **utiliser le synthé dans *Sonic Pi* comme d'habitude**, mais celui-ci ne propose ni autocomplétion ni documentation. Attention, a priori il faut également **mettre son nom entre guillemets** pour qu'il soit reconnu :

```
synth 'piTest', freq: 330, amp: 0.5
```

Les différents paramètres accessibles depuis *Sonic Pi* pour moduler le synthé sont les arguments de la fonction associée à la *SynthDef*, c'est-à-dire ceux encadrés par des barres verticales, ou suivant le mot clef *arg*, tout en haut du code :

```
|freq = 200, amp = 1, out_bus = 0|
```

ou

```
arg freq = 200, amp = 1, out_bus = 0;
```

Attention, dans certains cas, **certaines *SynthDef* peuvent utiliser des commandes qui ne fonctionnent pas avec *Sonic Pi***. Le résultat n'est donc pas garanti.

Si vous avez lu jusque là, je pense qu'une des prochaines étapes pourrait être de farfouiller parmi ces liens :

https://github.com/SCLOrkHub/SCLOrkSynths/tree/master/SynthDefs

https://github.com/dakyri/synthdef_collection

https://github.com/everythingwillbetakenaway/Synthdefs

https://github.com/theseanco/awesome-synthdefs/blob/master/co34pt/SynthDefs/co34pt_synthdefs.scd

https://github.com/supercollider-quarks/SynthDefPool/tree/master/pool

https://github.com/loopier/synthdefs

...

---

https://github.com/search?q=synthdef

https://sccode.org/

https://scsynth.org/

Bon son !

Découvrir PureData

Des boîtes et des données pour faire du son et de la vidéo !

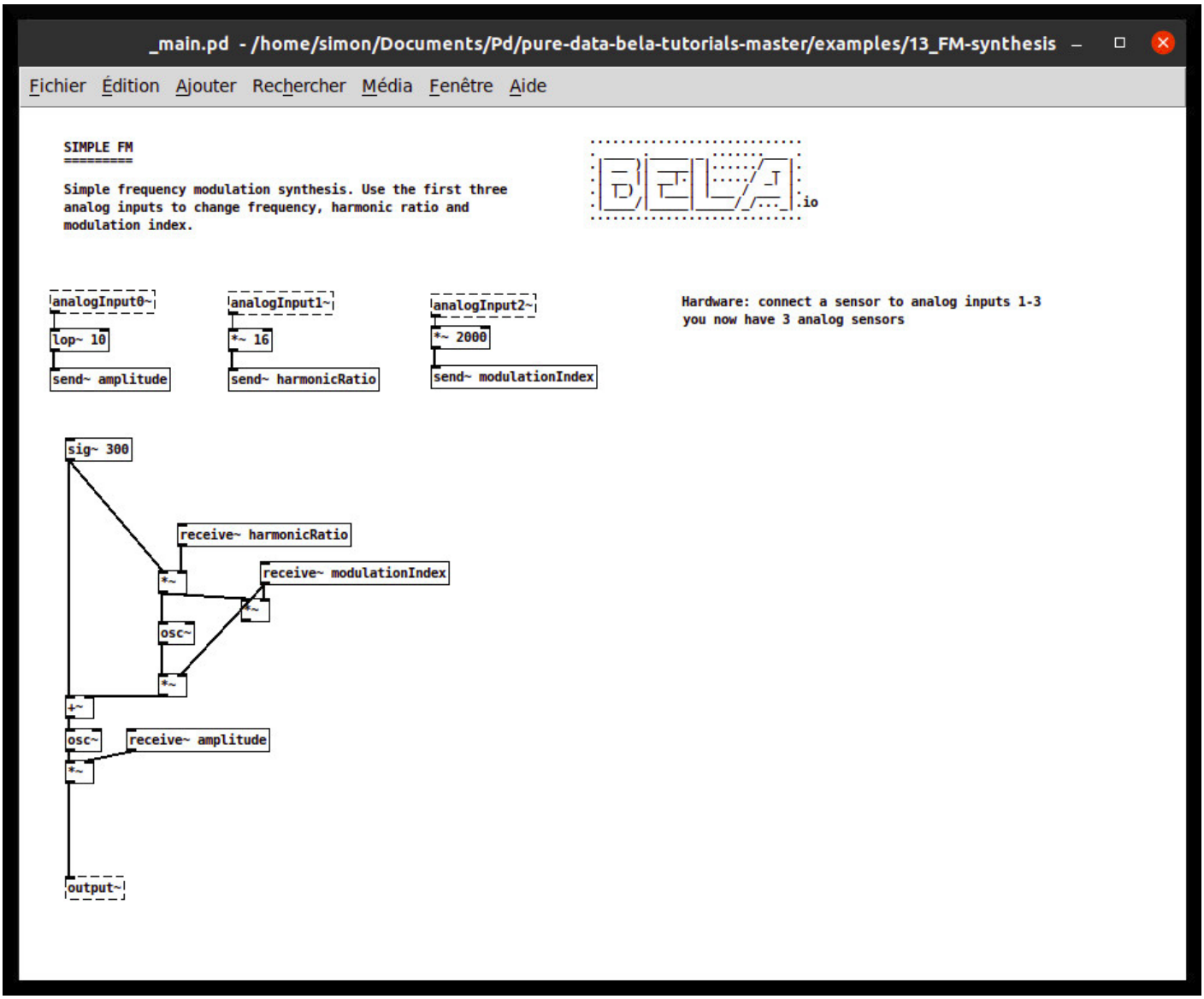
Qu'est-ce que PureData ?

PureData est un logiciel permettant de **créer et manipuler de l'audio et de la vidéo**.

Crée par Miller Puckette, il s'agit d'un logiciel libre.

PureData permet de **créer des boîtes ayant des fonctionnalités** particulières et de **les relier ensemble à l'aide de fils**.

Voici un exemple de fichier PureData. On appelle cela un *patch* :

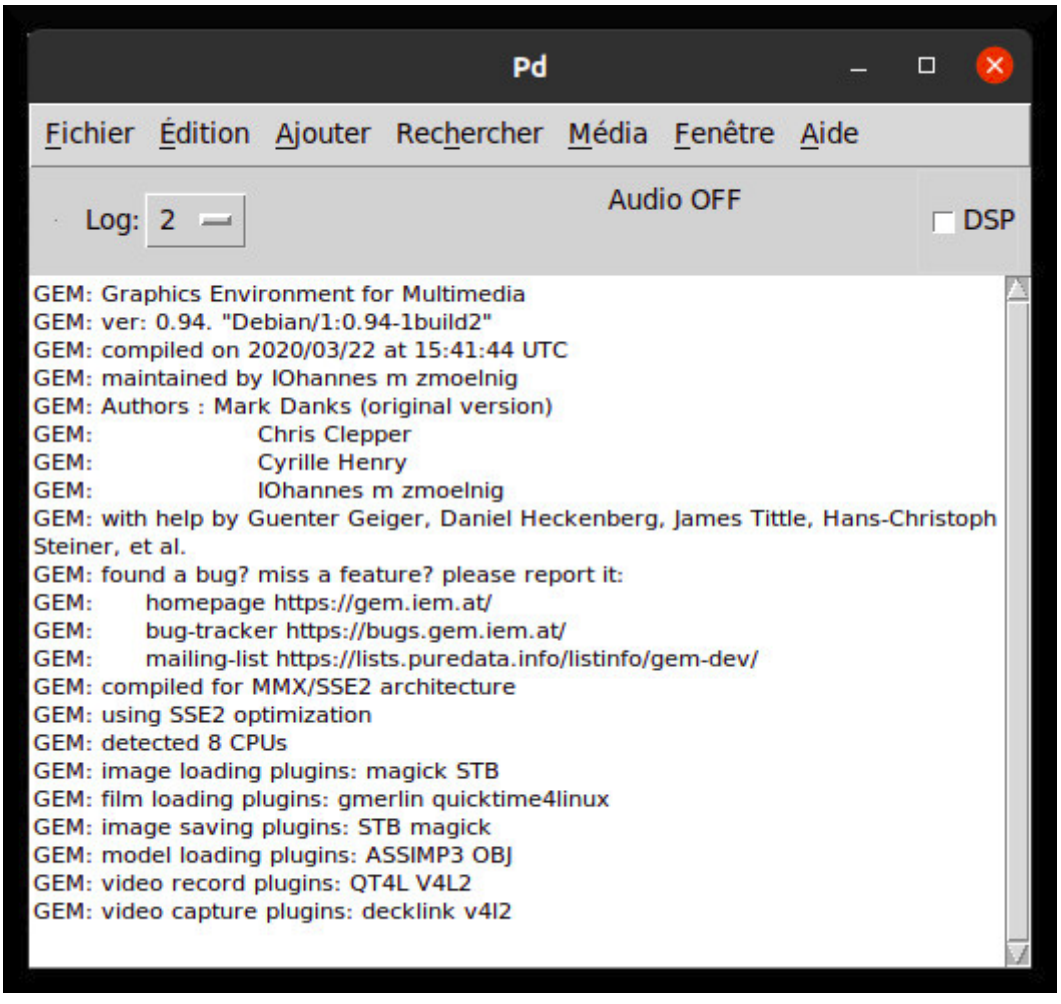


PureData est avant tout utilisé par **les artistes numériques** pour la réalisation d’œuvres.

(*) Comment bien débuter avec PureData ?

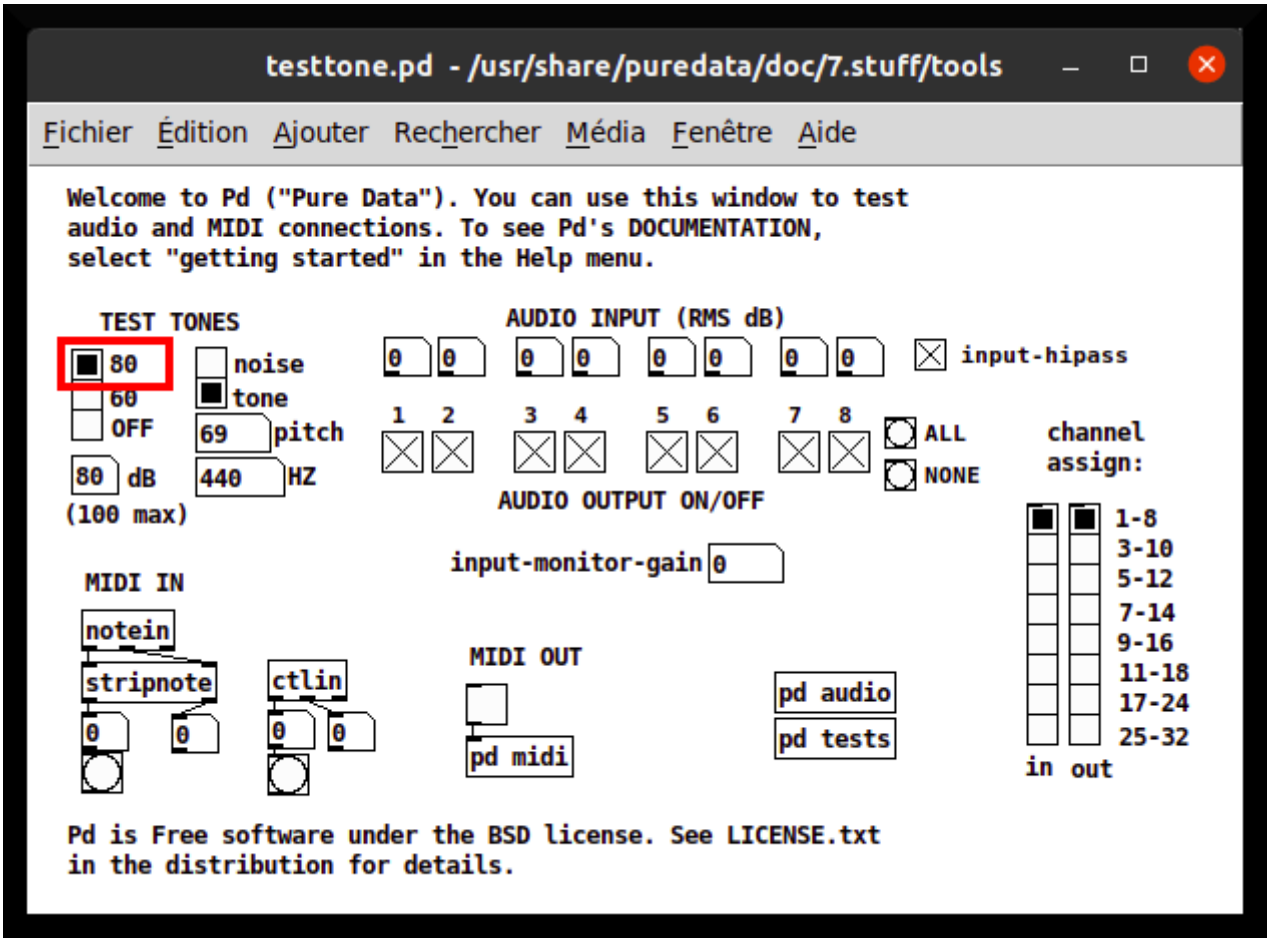
> (*) : Il manque des informations. Cette page est en cours de construction et n'est donc pas finalisée.

Lorsque vous lancez PureData, **le menu principal s'ouvre dans une nouvelle fenêtre**, qui ressemble à ceci :



Une bonne pratique est de commencer par utiliser le menu *Média > Tester l'audio et le MIDI*... afin de **voir si le son fonctionne**, en utilisant la case "80". **Attention à baisser le son de vos**

enceintes avant de commencer pour éviter d'endommager votre audition :

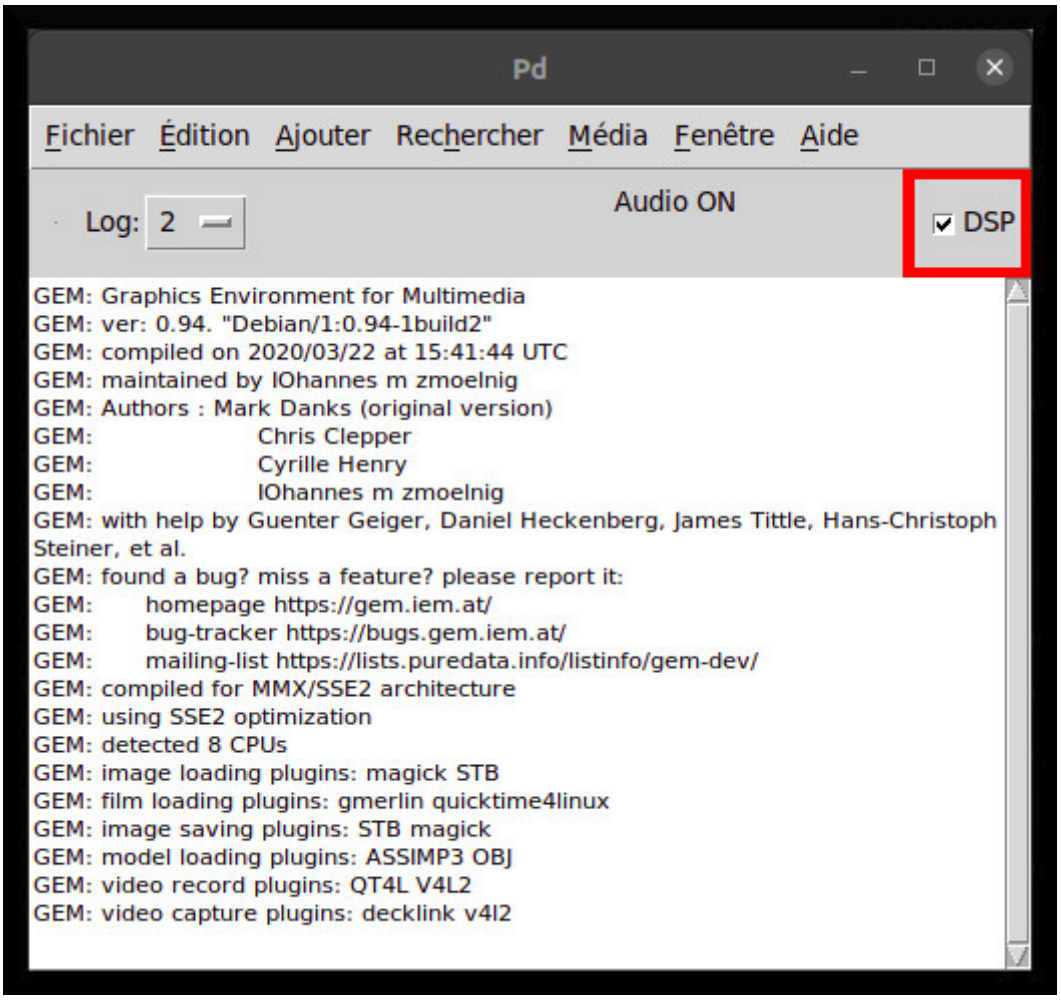


S'il n'y a pas de son, vérifier les messages d'erreurs qui apparaissent dans la fenêtre principale de PureData. La ligne

```
ALSA output error (snd_pcm_open): Device or resource busy
```

indique que **PureData est en conflit avec une application qui utilise actuellement la carte son**, ce qu'il n'aime pas vraiment, par exemple si une fenêtre internet contenant une vidéo est ouverte. Dans ce cas-là, fermez l'application qui accède à la carte son et redémarrez PureData. Si vous souhaitez utiliser plusieurs application audio en même temps, il faudra passer par *JACK* pour tout synchroniser, si vous utilisez Linux.

Si vous n'avez malgré tout pas de son, **vérifiez que l'audio est activé dans la case DSP de la fenêtre principale** :



[EN CONSTRUCTION]

Découvrir LMMS

Boum - tchak - boum - tchak !

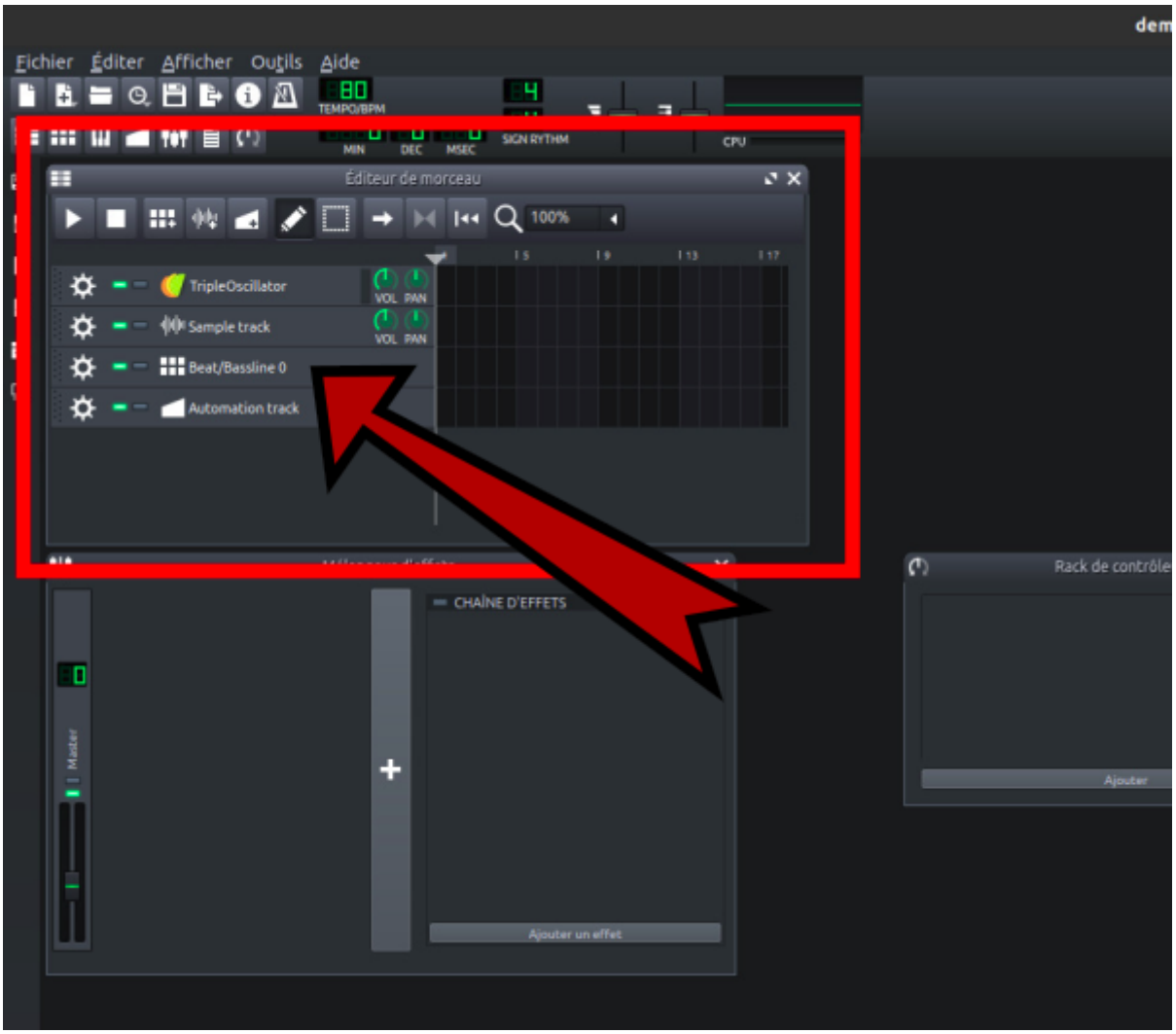
Utiliser le séquenceur de LMMS

Dans ce tutoriel, nous verrons comment **utiliser le séquenceur intégré à *LMMS* pour créer un rythme**.

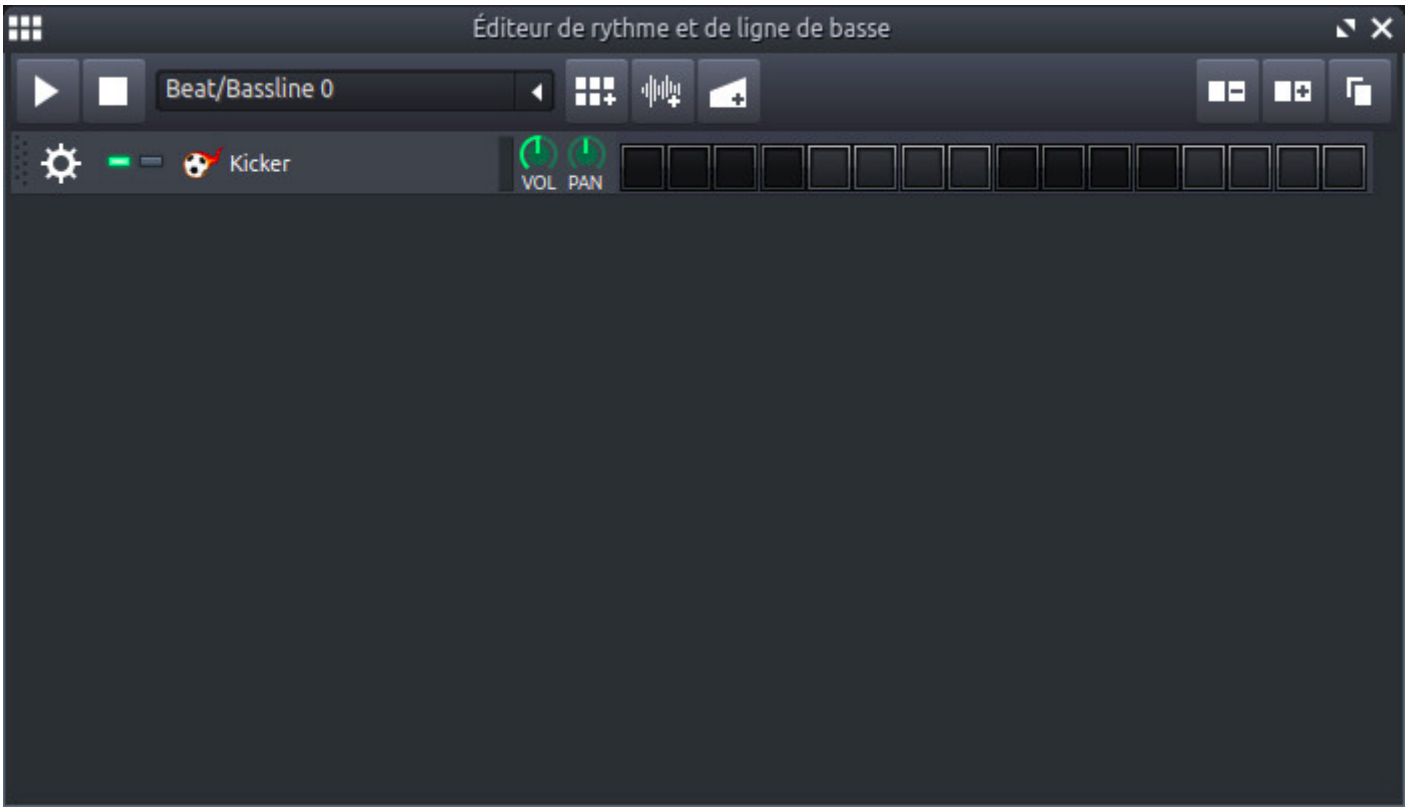
Si vous n'êtes pas tout-à-fait à l'aise avec la notion de rythme, vous pourrez trouver quelques indications à ce propos sur le tuto qui y est dédié.

Lorsque nous lançons **un nouveau projet dans *LMMS***, celui-ci ajoute automatiquement **une piste de rythme** au projet.

Celle-ci est située dans l'*Éditeur de Morceau* et s'appelle par défaut *Beat/Bassline 0* .



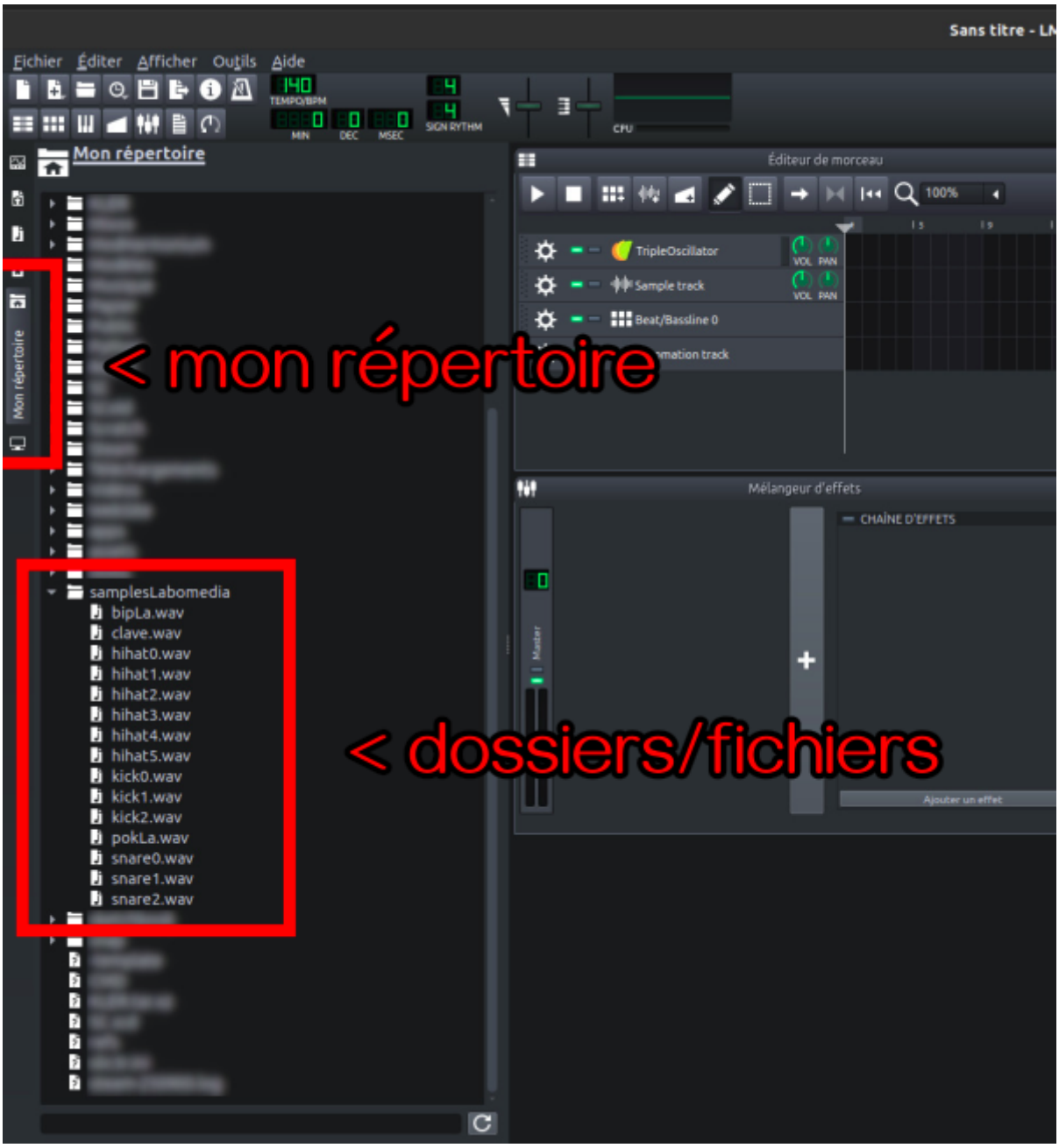
En cliquant sur son nom, on peut accéder à la **fenêtre de séquençage** de *LMMS*.



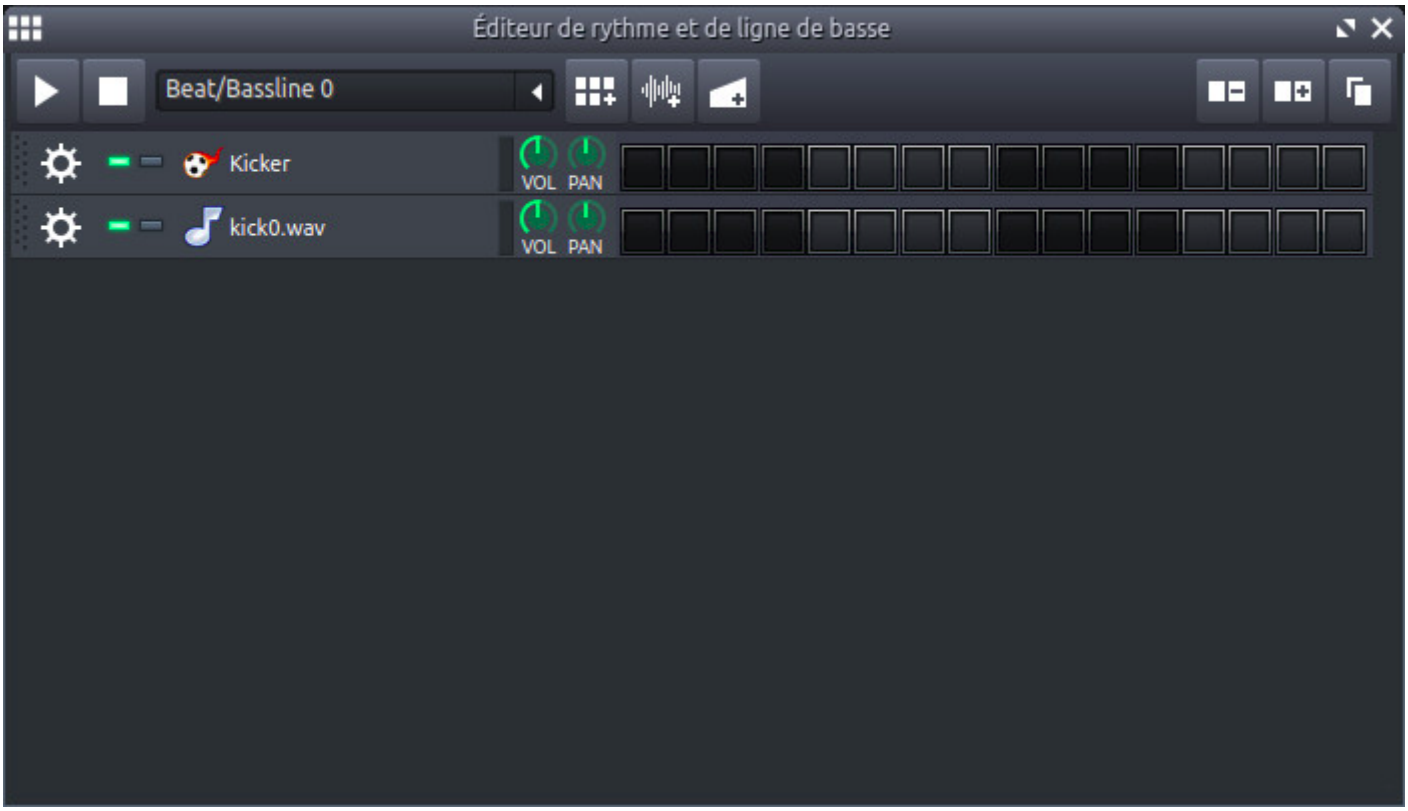
Un instrument nommé *Kicker* est ajouté automatiquement. Il permet de synthétiser des sons de grosse caisse, et vous pouvez le supprimer si vous ne l'utilisez pas.

En général, on utilise des *samples* (des enregistrements de son) afin de créer la rythmique qui nous convient.

La barre de menus située à gauche du logiciel permet d'accéder à son répertoire de fichier et de glisser les *samples* directement dans le séquenceur. *Attention*, quand on clique sur un *sample* du répertoire, celui-ci est joué en guise de préécoute : si votre volume audio est élevé il peut arriver que vous vous abîmiez les oreilles. Prudence donc.

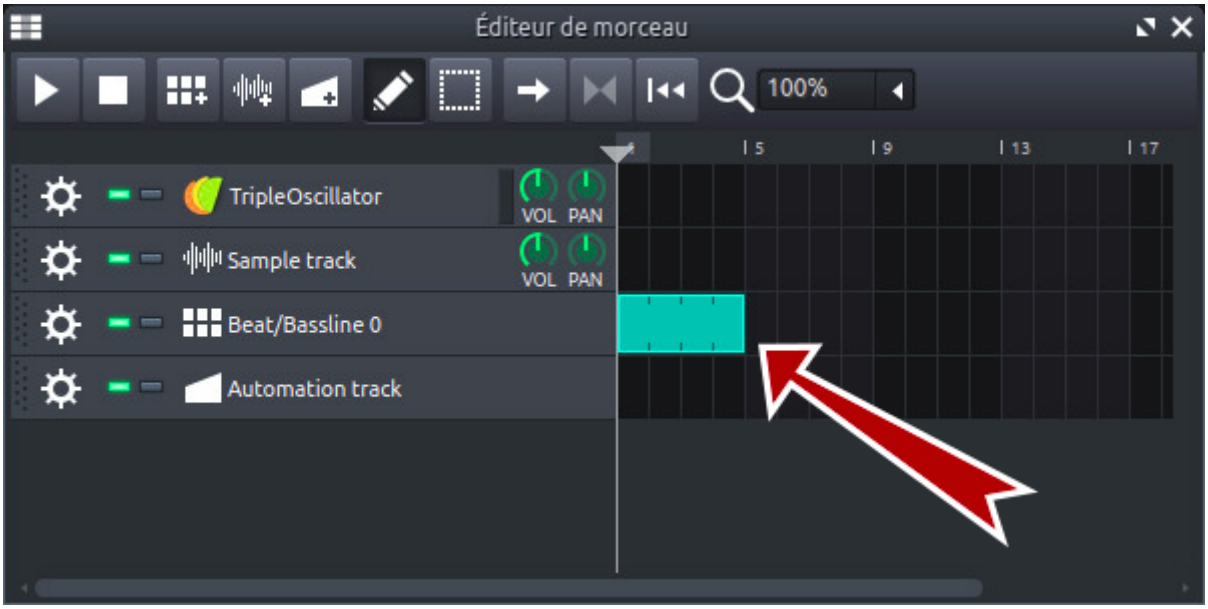


Une fois le *sample* glissé dans le séquenceur, celui-ci apparaît, et on peut le modifier en cliquant sur son nom.



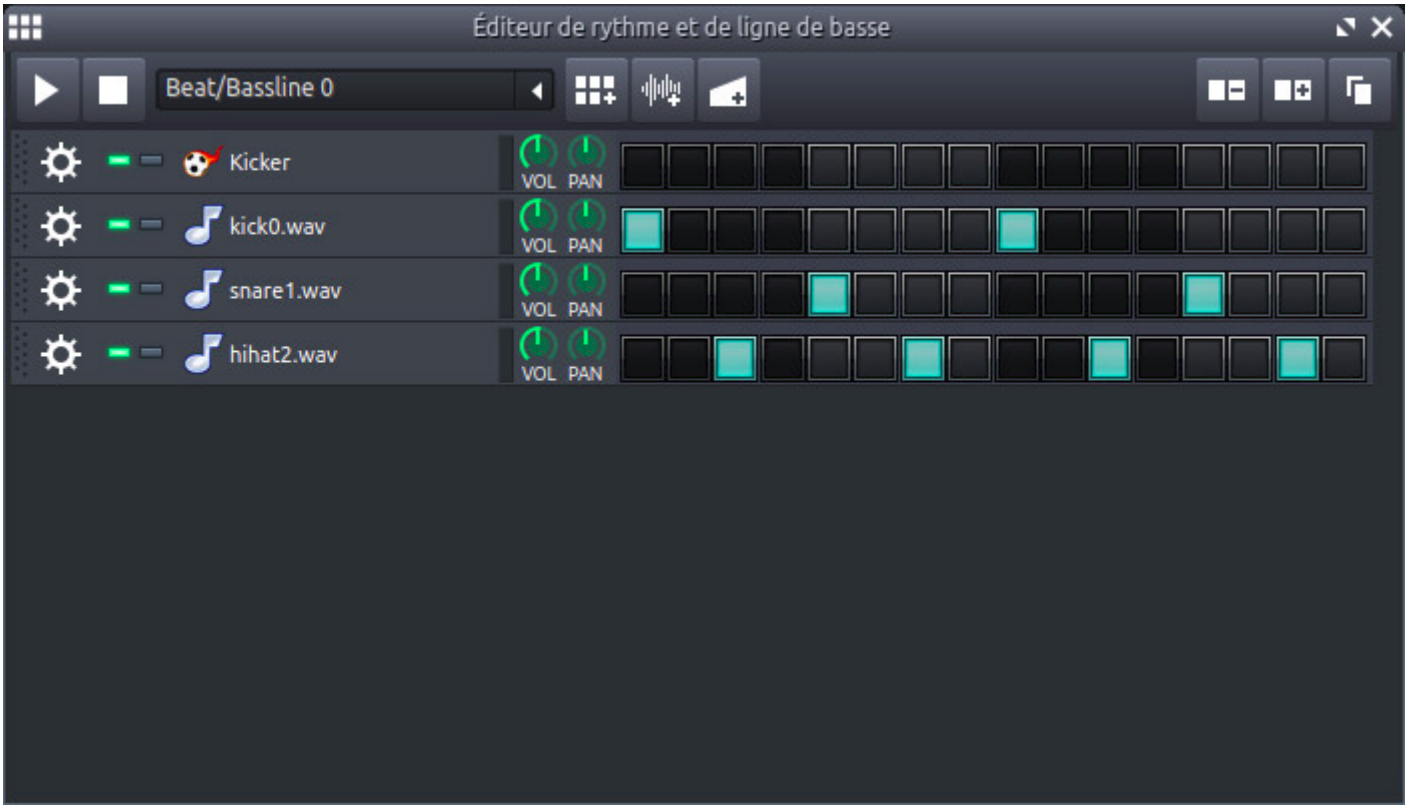
La grille à droite des *samples* est **le séquenceur lui-même**. **Celui-ci peut être mis en route à l'aide de la barre espace**. Lorsque que nous faisons ceci, seuls les éléments du séquenceur sont joués, ce qui permet d'écouter uniquement la boucle de rythme sur laquelle nous travaillons.

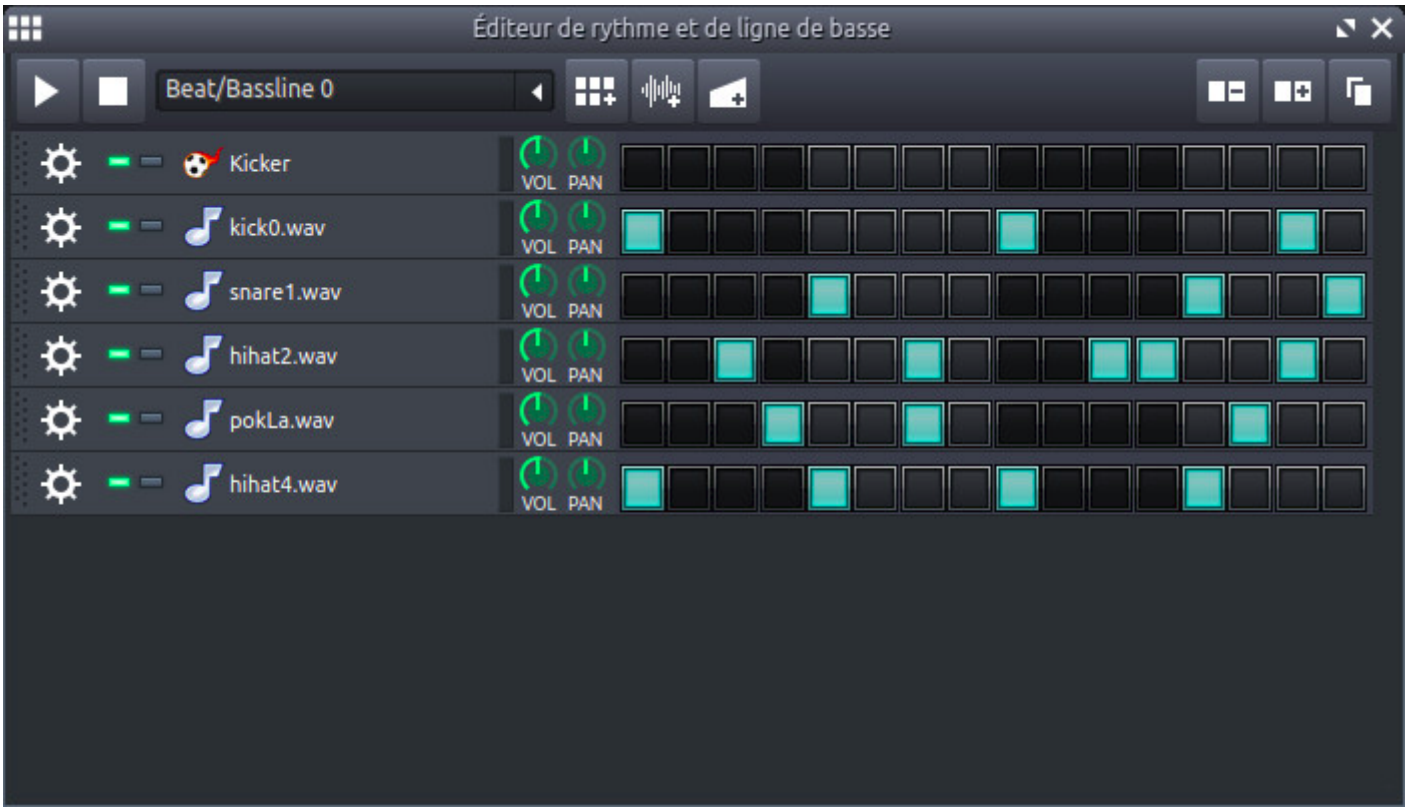
Pour écouter l'ensemble des instruments, il faut ajouter une piste pour le séquenceur dans l'*Éditeur de morceau*, et jouer le morceau dans l'*Éditeur*.



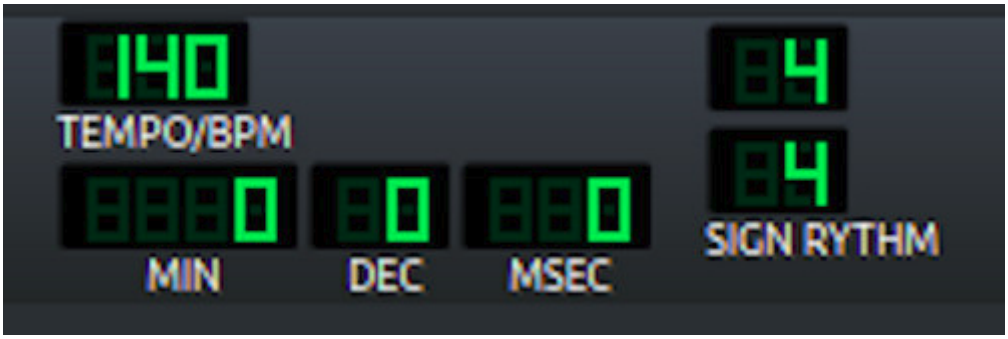
Afin de **construire le rythme** il faut ensuite **cocher les temps où l'ont souhaite que le *sample* soit joué**.

Le rythme le plus simple, en 4/4, consiste à alterner un *kick* et un *snare* (grosse caisse et caisse claire) sur les temps forts, et à intercaler un coup de *hihat* (charleston) entre chaque temps. En général, on rajoute ensuite quelques coups de *kick*, *snare* et *hihat* pour 'teinter' le rythme, et on agrémente avec des percussions additionnelles (toms, cloche, cymbales, etc).





Le tempo de base, 140, est assez élevé (sauf si vous travaillez à demi-tempo). **Pour modifier le tempo et la métrique, on utilise le menu dédié, en haut de l'interface.**



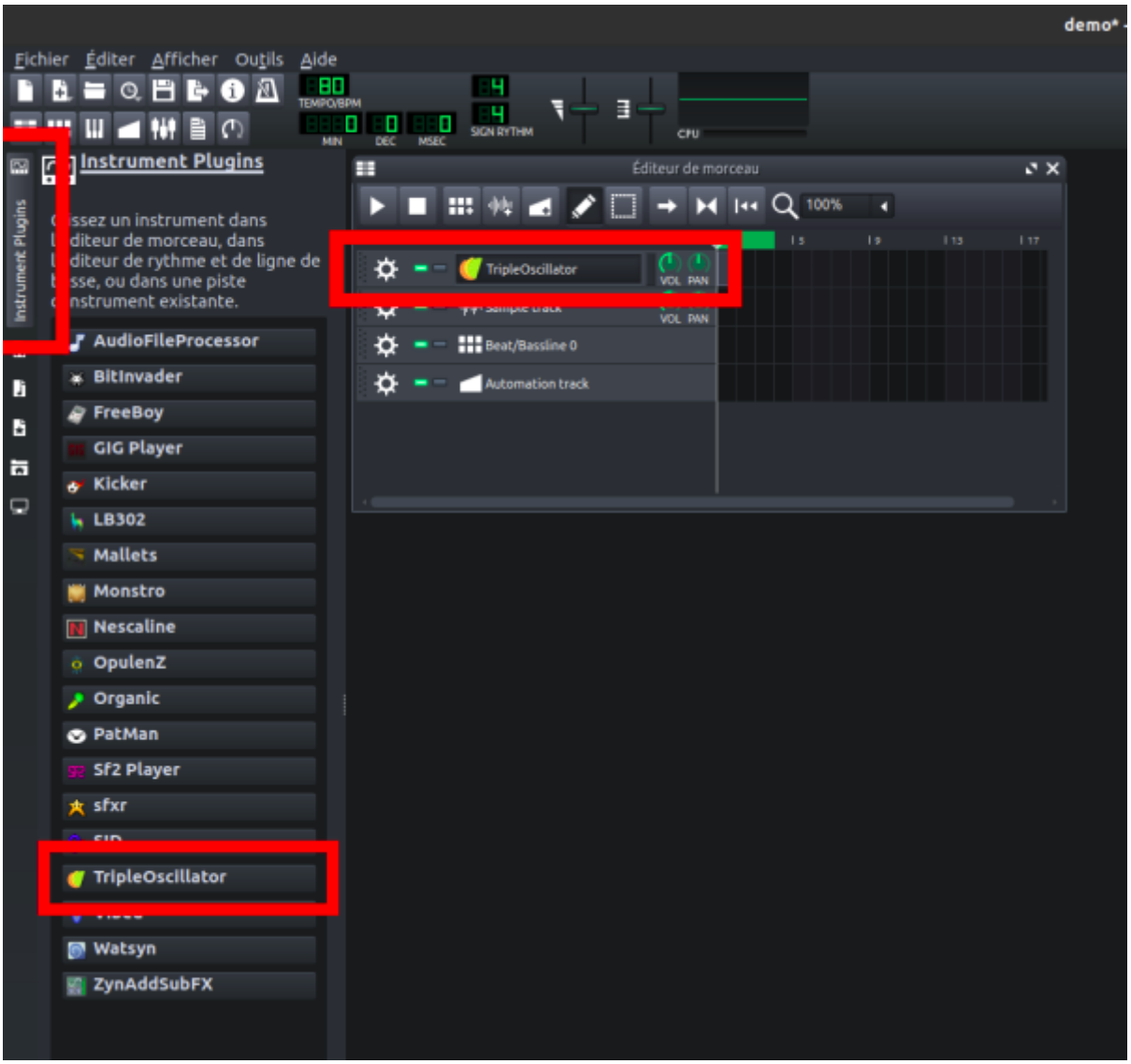
(*) Utiliser le TripleOscillator de LMMS pour créer des mélodies

> (*) : Il manque des informations. Cette page est en cours de construction et n'est donc pas finalisée.

Sous ce titre aguicheur, nous allons découvrir comment **synthétiser des sons à l'aide du module *TripleOscillator* de *LMMS***.

Celui-ci permet de **créer des mélodies, des accompagnements, des percussions, des nappes...** Théoriquement, il peut presque recréer n'importe quel son !

Par défaut, *LMMS* ajoute un *TripleOscillator* (TO) à l'*Éditeur de Morceau* lors du chargement d'un nouveau projet. Autrement, on peut utiliser le menu à gauche du logiciel pour en rajouter (ainsi que d'autres modules de synthèse). C'est l'onglet *Instrument Plugins* qui permet d'y accéder.



Pour commencer à configurer le TO, **on clique sur son nom, ce qui ouvre sa fenêtre de configuration**. Malheureusement, celle-ci est assez petite et n'est pas redimensionnable...

Premièrement, je vous recommande de **réduire son amplitude (son volume sonore)** car celui-ci est très bruyant par défaut et *risque de vous abîmer les tympans*.

Vous pouvez **écouter à tout moment le son que celui-ci produit à l'aide du petit piano** en bas de la fenêtre. On peut également déclencher ces touches à l'aide du clavier (à partir de la touche "a").

À ce stade, en fait, **nous avons actuellement trois sons qui sont joués**. Ceux-ci correspondent aux trois bandes centrales de la fenêtre : ce sont des **oscillateurs, qui produisent des ondes sonores**. L'un est réglé à la bonne fréquence, un autre est une octave en-dessous, le dernier deux octaves en-dessous.

Commençons par couper deux oscillateurs pour se concentrer sur un seul.

Pour régler le volume d'un oscillateur, on utilise le potentiomètre ***VOL*** . En double-cliquant dessus, on peut entrer la valeur à l'aide du clavier.

Les petits boutons en haut à droite de l'oscillateur permettent de **définir sa forme d'onde**. Essayez les pour expérimenter les différents sons que ces ondes produisent.

Le potentiomètre ***PAN*** règle le panoramique de l'oscillateur (sa position dans le champ stéréo).

Le potentiomètre ***CRS*** règle le décalage de fréquence de l'oscillateur par rapport à la note originale (+12 équivaut à une octave au-dessus par exemple).

Les potentiomètres ***FL*** et ***FR*** décalent légèrement les oscillations de l'oscillateur dans le canal gauche et le canal droit de la stéréo. Cela permet d'obtenir un son plus organique.

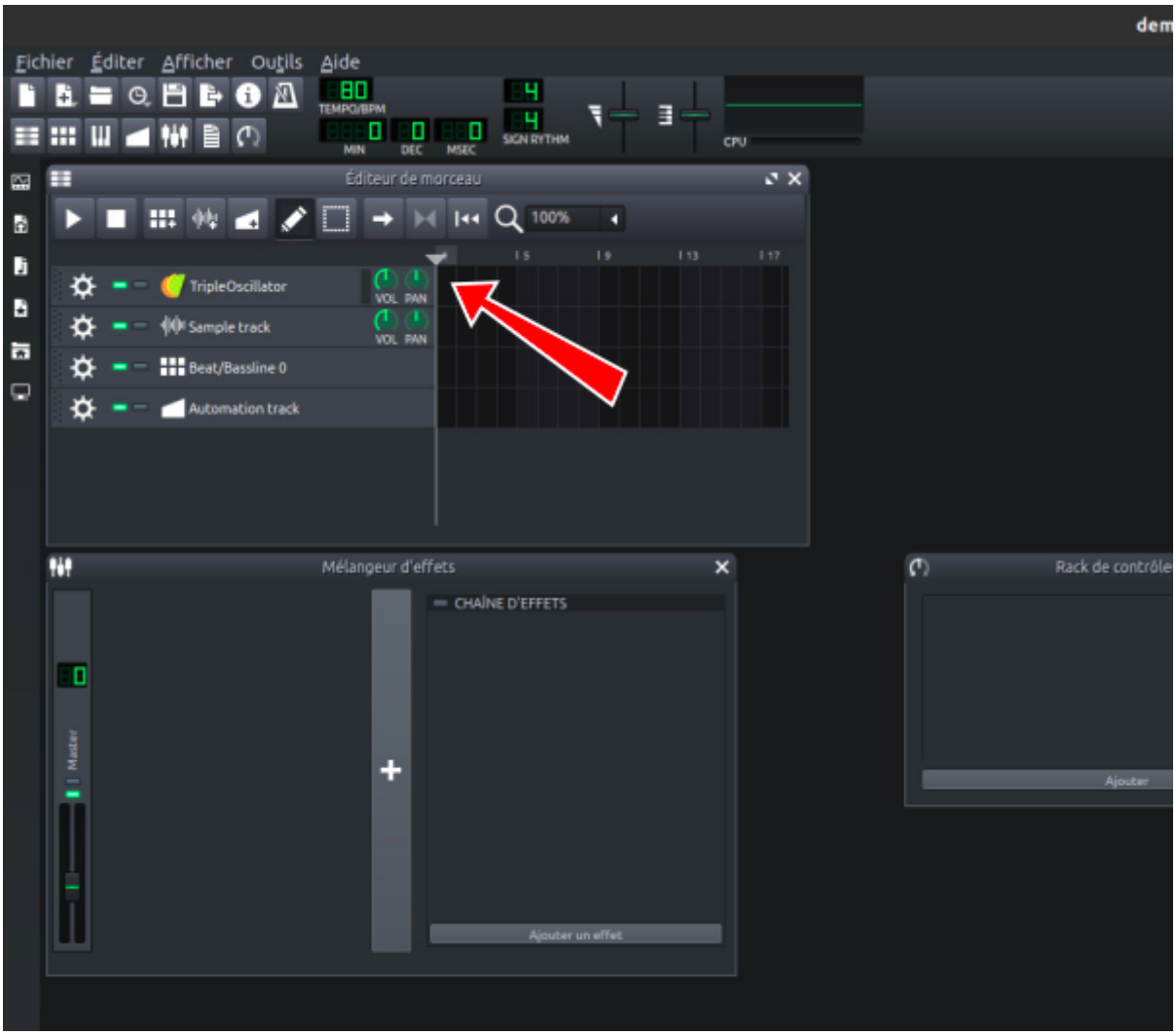
Le potentiomètre ***PO*** permet de décaler la phase de l'oscillateur.

Le potentiomètre ***SPD*** permet de décaler les phases de l'oscillateur entre les deux canaux stéréos.

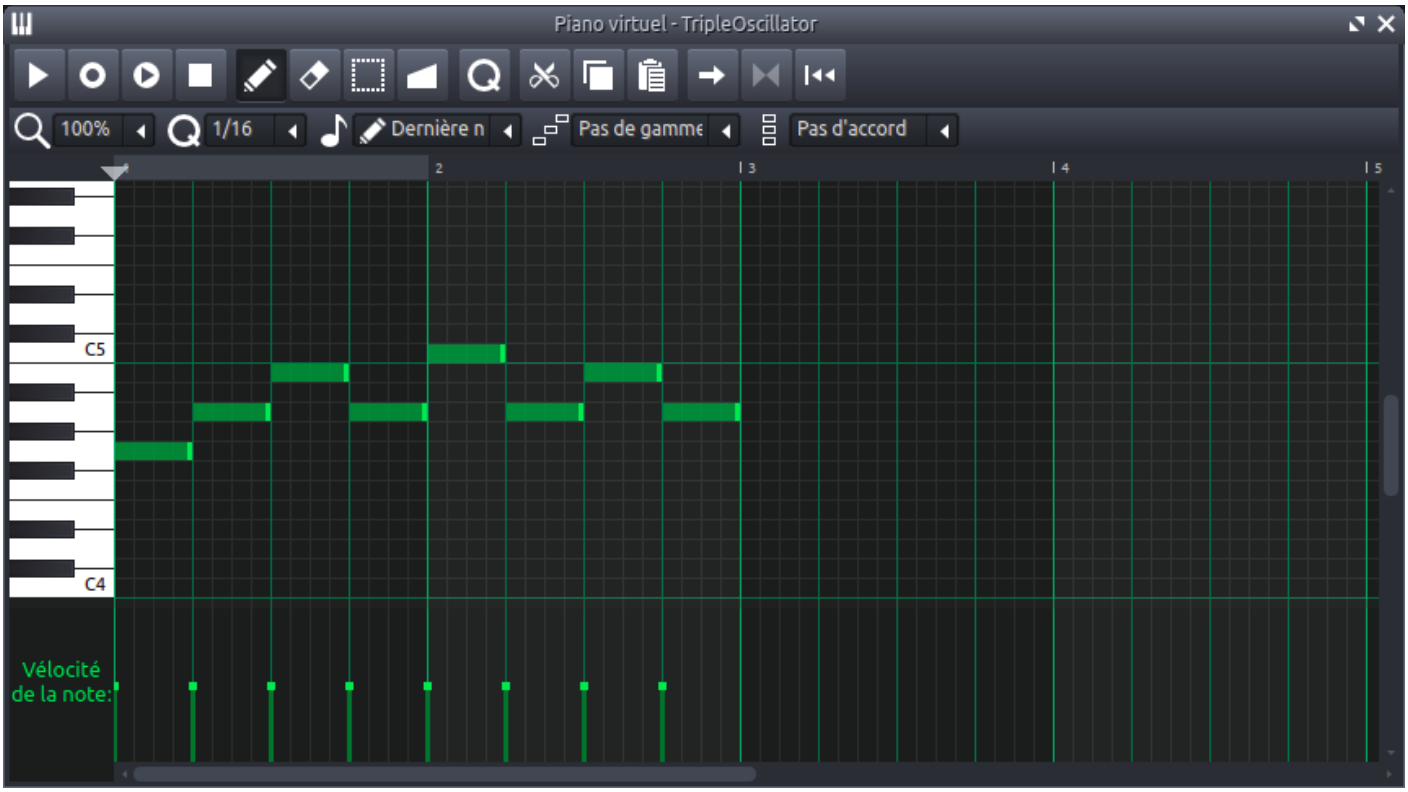
Remettez maintenant les deux autres oscillateurs en route et essayez différents réglages pour voir comme le son change, en utilisant le clavier pour produire du son facilement.

Une fois que l'on a trouvé un son qui nous plaît (on pourra le retoucher plus tard), nous allons **utiliser le *piano virtuel* pour jouer une boucle**.

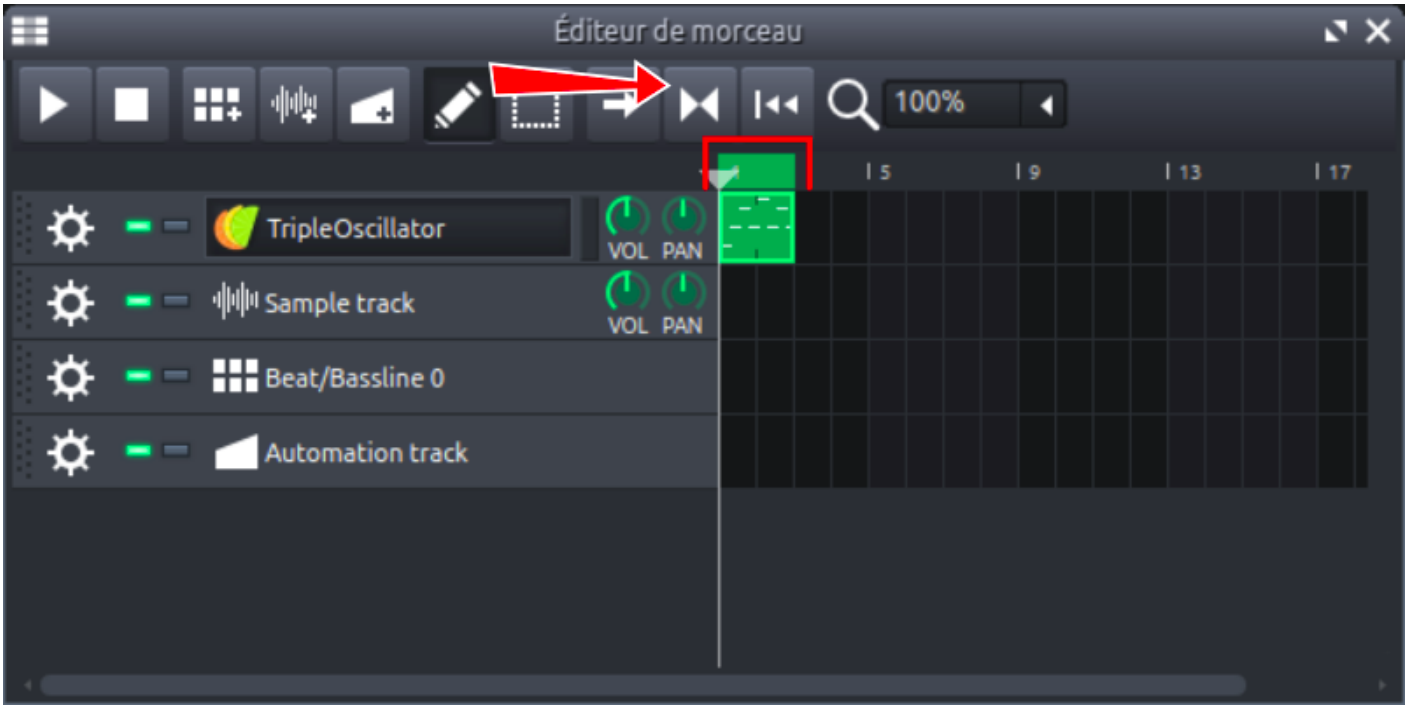
Pour l'ouvrir, il faut **double-cliquer sur le rail à droite** de la piste du *TripleOscillator*, dans l'*Éditeur de morceau*.



On peut ensuite **indiquer les notes que le synthé aura à jouer**. Pour commencer, si vous n'êtes pas familier avec la théorie musicale, vous pouvez essayer ce motif :



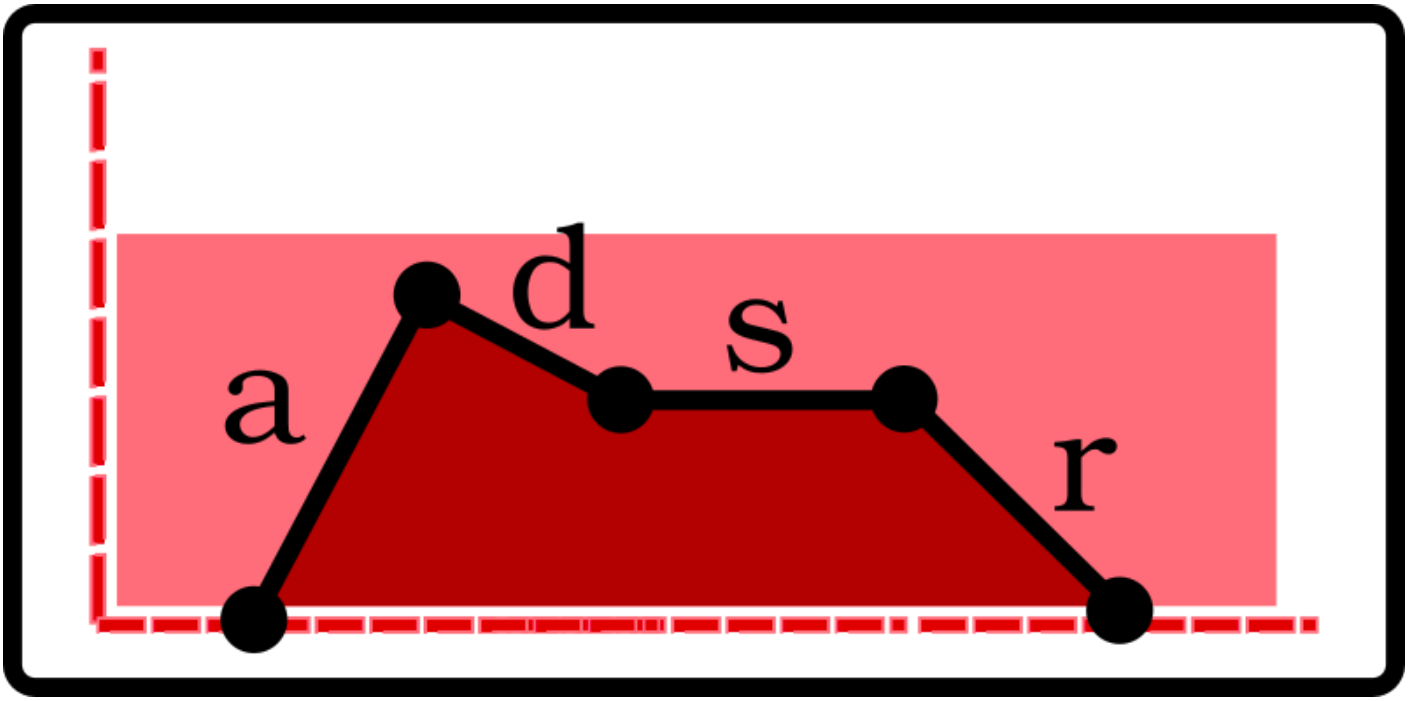
Ensuite, il faut **activer la lecture en boucle** à l'aide du bouton dédié, et utiliser le *clic droit* sur la barre de bouclage pour englober le motif musical que l'on vient de créer.



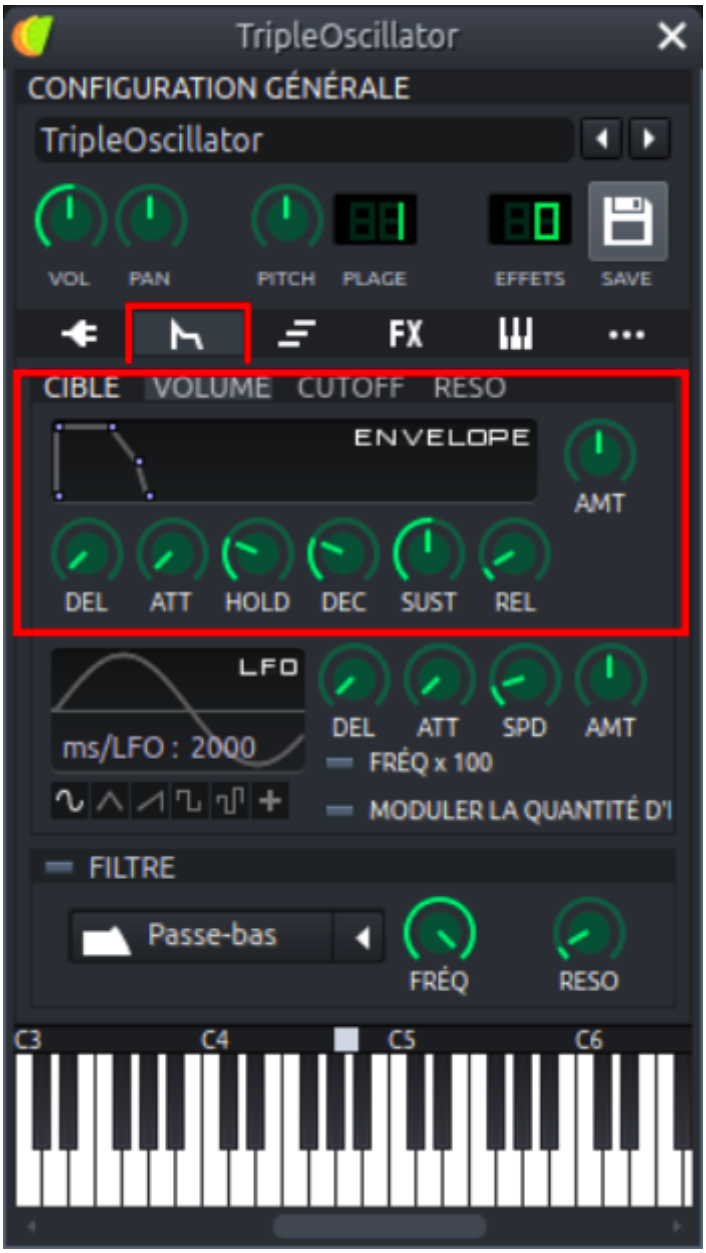
Après cela, **la touche espace lance le son et le joue automatiquement, ce qui nous permet de l'éditer pendant qu'il joue.**

Pour l'instant, **le son est assez indélicat, car il se comporte d'une manière étrange** : il se lance au volume maximal, reste constant, puis se coupe d'un coup.

Pour changer cela, on utilise une technique appelée *ADSR* qui permet de **spécifier à l'aide de quatre paramètres le comportement du volume du synthé dans le temps** pour chaque note qu'il doit jouer. La courbe qui en résulte s'appelle l'enveloppe.

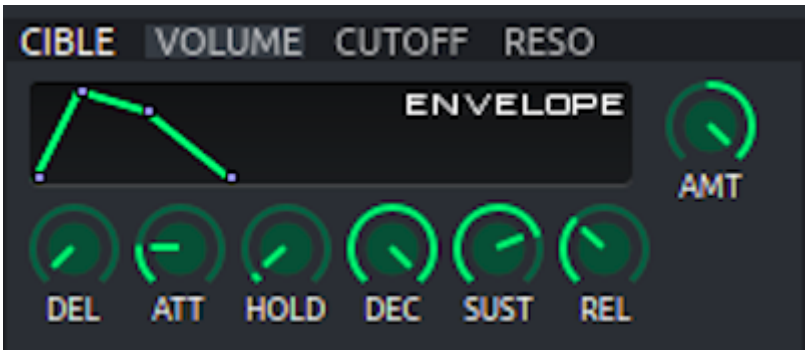


Pour le régler, **on change de menu dans la fenêtre** du *TripleOscillator*.



Pour que cela soit audible, il faut monter le potentiomètre ***AMT*** , qui définit le taux d'enveloppe qui sera appliqué au son. En général, on le règle au maximum.

Ensuite, **les potentiomètres à gauche permettent de régler la forme de l'enveloppe**. C'est assez facile car le parallèle visuel est intuitif.



À titre d'exercice, **essayez d'imaginer** à quoi ressemble l'enveloppe d'une corde de guitare quand le doigt la frappe ? Le son d'un violon quand on y frotte un archet ?

[EN CONSTRUCTION]