Programmation Fonctionnelle I, Printemps 2017 – TD AUDIO

http://deptinfo.unice.fr/~roy

Mettez vos casques et attention au volume sonore : toujours le volume à zéro avant de lancer l'écoute d'un son, et on monte progressivement... Téléchargez et installez le logiciel gratuit Audacity.

1. Restriction au format WAVE PCM 16 bits

Pour votre première rencontre avec le traitement du son en Racket avec **rsound**, nous commençons par nous intéresser à la fréquence d'échantillonnage d'un son. Normalement elle est de 44.1 kHz, soit **44100** échantillons de 16 bits (*frames*) par seconde. Bien entendu, dans l'industrie musicale, elle pourra être meilleure, par exemple 48 kHz sur 24 bits. **Rsound** utilise pour l'instant le format **WAVE PCM 16 bits à 44.1 kHz**. Il est capable de lire des fichiers audio échantillonnés par exemple à 8000 Hz lors de l'enregistrement de voix humaine par exemple, mais il reste strict sur le format PCM[1].

Exercice 1 **a)** Avec le navigateur conseillé ***Chrome***, allons sur le Web rechercher un fichier **.wav**. Il contiendra des nombres lus, par exemple le fichier `27.wav` contiendra : *twenty seven*. L'enregistrement n'est pas très bon, peu importe. Allez le trouver sur Voxeo :

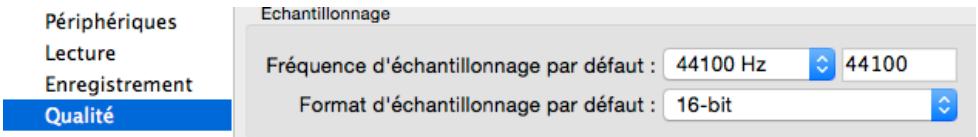
`https://evolution.voxeo.com/library/audio/prompts/numbers/index.jsp`

et récupérez le fichier `27.wav` dans votre répertoire de travail. Ecoutez-le (avec **Audacity** par ex).

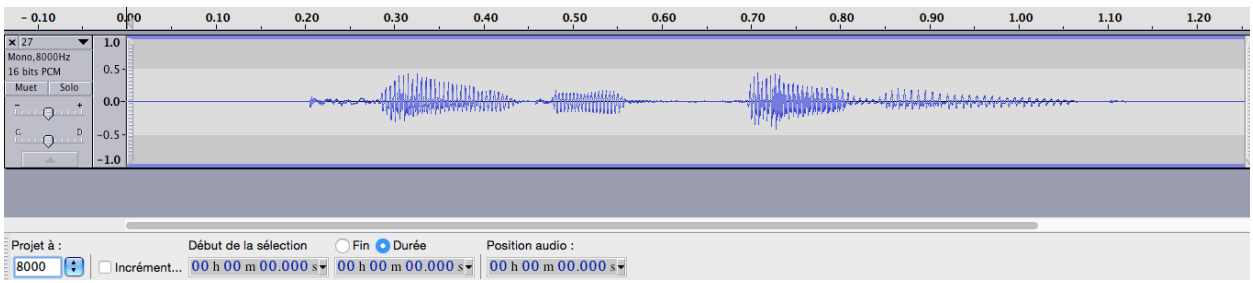
b) Maintenant vous êtes dans Racket, ouvrez le fichier `TP7.rkt` et définissez une variable `n27` dont la valeur est le son obtenu en chargeant `27.wav` dans la mémoire Scheme avec `rs-read`. Comme vous pouvez le constater, ceci aboutit à une ***erreur*** dont le message est clair : ce n'était pas un fichier PCM[2]. On ne peut rien faire du côté de Racket, nous utiliserons un programme externe **Audacity** déjà installé :

`http://audacity.sourceforge.net/?lang=fr`

Dans les *Préférences* d'Audacity, réglez la qualité par défaut :



Avec Audacity, ouvrez le fichier `27.wav`. Les 8000 Hz (coin en bas à gauche) ne sont pas un problème pour Racket qui saura les modifier (on pourrait le faire dans Audacity, mais ne le faisons pas maintenant).



Demandez *Exporter...* depuis le menu *Fichier* et sauvez dans le répertoire courant sous le format **WAV (Microsoft) signé 16 bits PCM**, et zou ! Nommez le nouveau fichier `27-pcm.wav`.

c) A partir de maintenant nous travaillons dans Racket. Vous devriez pouvoir charger avec `rs-read` le nouveau fichier `27-pcm.wav` sans erreur dans la variable `n27` de type `rsound`. Ecoutez-le avec `play`.

d) Comment vérifiez-vous en Racket que le fichier `27-pcm.wav` est bien échantillonné à 8 kHz ?

---

[1] http://soundfile.sapp.org/doc/WaveFormat/

[2] Il y a des sites qui fournissent du format PCM 16 bit (mais peut-être à 8000 Hz) : https://www.freesound.org

e) Combien comporte-t-il d'échantillons (*frames*) ?

d) Programmez une fonction `(duration snd)` retournant la durée inexacte du son `snd` en secondes flottantes. Calculez la durée du son `n27` *[rép: 1.2545 sec]*

e) Avec `rs-draw`, faites afficher la courbe du son. Visualisez-vous les quatre syllabes ?

f) Avec `clip`, éliminez les segments horizontaux au début et à la fin qui sont inaudibles (trouvez les frontières à la souris). Nommez le nouveau son `n27-clipped` et enregistrez-le sur le disque sous le nom `27-pcm-clipped.wav`. Faites afficher la nouvelle courbe. Quelle est la durée du nouveau son ?

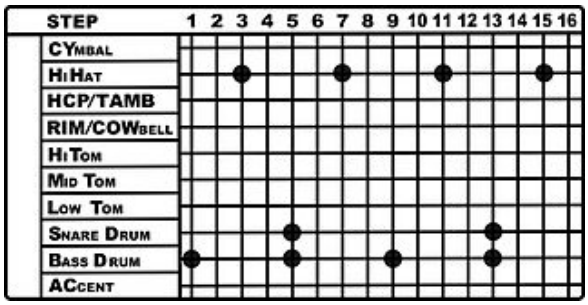
g) Faites jouer le nouveau son. Si vous n'avez pas commenté le `(play n27)` antérieur, il risque de se superposer au `play` actuel car `play` est **asynchrone**[3]. Dans ce cas, il faut faire suivre le premier d'un `(sleep n)` où `n` est au moins la durée en secondes du son joué, que l'on connaît par d). Oui, *c'est un peu désagréable, nous aurons mieux avec les pstreams...*

h) A partir de la variable `n27-clipped`, définissez une autre variable `n27b7` dont la valeur est le son *twenty seven seven*.

i) Utilisez la fonction `(resample-to-rate new-frame-rate sound)` et regardez sa doc en ligne. Comment l'utiliseriez-vous pour ré-échantillonner le son `n27b7` pour produire un nouveau son `n27b7-44100` puis un fichier `n27b7-44100.wav` ? Vérifiez avec Audacity que l'échantillonnage est bien 44.1Kz et surtout que le son est bien le même à l'oreille ! Comparez aussi les courbes des deux sons. Avec **Audacity**, transformez le fichier *wav* en fichier *mp3*.

2. Utilisation des PSTREAMS. Un SEQUENCEUR pas à pas

En musique électronique, un **séquenceur**[4] est un logiciel (voire un appareil MIDI) utilisé pour produire une boucle sonore constituée de plusieurs sons joués en séquence.



- L'ancienne boîte à rythmes *Roland-TR707* -

Méthode 1 : utilisation de l'horloge d'une animation

Exercice 2 Programmez une **animation** `(sequenceur L freq)` prenant une liste de sons `L` (par exemple des percussions) ainsi que la fréquence d'horloge `freq`. Elle fera jouer en boucle les sons de la liste `L`, un son à chaque top d'horloge. **Le monde sera une liste de sons**, mais qu'il faudra gérer de manière **circulaire** pour boucler ! Vous enverrez les sons dans une *pstream*. Exemple :

```
(define MYSOUNDS
  (list (silence 1) snare c-hi-hat-1 snare snare bassdrum bassdrum))
(sequenceur MYSOUNDS 1/5)          ; the audience is listening, 5 sounds / sec
```

[3] Une fonction est **asynchrone** lorsqu'elle termine immédiatement, son processus restant en effet en arrière-plan un certain temps. La fonction `play` rend la main tout de suite mais le son peut durer 5 minutes ! On dit aussi qu'elle n'est *pas bloquante*.
[4] Cherchez *step sequencer* sur YouTube.

Méthode 2 : production d'un son par assemblage

*Une autre voie consiste à **construire le son** d'une seule boucle, puis de la répéter autant de fois qu'on veut. Mieux vaut en effet ne pas construire le son correspondant à 100 boucles, qui serait trop gros.*

Exercice 3 **a)** Programmez la fonction `(one-loop L freq)` prenant une liste de sons `L` et une fréquence `freq` en Herz, et retournant un son formé de l'**assemblage** des sons de la liste `L`, avec un espacement correspondant à la fréquence `freq`. Vérifiez avec `pstream-play` que le son obtenu est correct.,

b) Programmez par récurrence sur `n` une fonction `(play-loop snd n)` jouant en boucle `n` fois le son `snd` (par exemple celui obtenu par `one-loop`). Quel est l'inconvénient de cette fonction ?

c) Reprogrammez b) d'une autre manière, en utilisant `pstream-queue-callback`, qui permet de relancer le son dès qu'il est fini. Nommez `(async-play-loop snd n)` cette nouvelle fonction. Cela supprime-t-il l'inconvénient de b) ?

Programmation Fonctionnelle I, Printemps 2017 – TP AUDIO

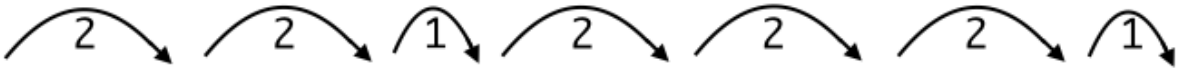
http://deptinfo.unice.fr/~roy

Vous pouvez lire en même temps la partie optionnelle du cours aux pages 27-30.

Rappel : les **notes de piano** (blanches et noires) sont espacées d'un *demi-ton*, ou *half-step*. Un octave comprend 12 demi-tons. Les demi-tons possèdent la propriété suivante :

On passe de la note MIDI numéro `m` à la note MIDI `m+1` située un demi-ton à droite en multipliant la fréquence de `m` par $\sqrt[12]{2}$. On passe donc de la note numéro `m` d'un octave à la note numéro `m+12` de l'octave suivant en multipliant sa fréquence par 2. Le DO4 est à 261.6 Hz tandis que le DO5 est à 523.2 Hz.

La note du milieu du clavier est le DO du 4ème octave, numéro MIDI 60. Les 12 demi-tons de cet octave sont toutes les touches blanches et noires numéros 60, 61, 62, 63, 64, 65...



DO	DO#	RE	MIb	MI	FA	FA#	SOL	LAb	LA	SIb	SI	DO
C4	C#	D	D#	E	F	F#	G	G#	A	A#	B	C5
60	61	62	63	64	65	66	67	68	69	70	71	72

La **gamme de DO majeur (*C major*)** est bien connue : DO-RE-MI-FA-SOL-LA-SI-DO (à l'anglo-saxonne C-D-E-F-G-A-B-C). Les intervalles séparant les notes de cette game majeure ne sont pas tous les mêmes, mais valent 1 ou 2 demi-tons. Par exemple on passe de RE/62 à MI/64 par 2 demi-tons mais de MI/64 à FA/65 par 1 seul demi-ton ! On voit ci-dessus que la suite des variations d'intervalles de la gamme majeure est donc: 2-2-1-2-2-2-1. Mais on peut jouer la gamme majeure à partir de n'importe quelle note (cf exo 3) !

Pour jouer des notes de piano en Racket, il faut charger un module : `(require rsound/piano-tones)`. La fonction `(piano-tone n)` retourne un son contenant la note de piano de numéro MIDI `n`. Les notes de piano n'ont pas la même durée et font *plus ou moins* 3 secondes.

Exercice 1 Programmez une fonction `(sec n)` prenant un nombre inexact de secondes et retournant un nombre entier exact d'échantillons équivalent, avec la fréquence usuelle d'échantillonnage de 44.1 kHz.

Exercice 2 La **gamme chromatique** comprend 12 notes séparées par un demi-ton. Ce sont donc les notes MIDI consécutives, par exemple 60,61,62,63,64,65,... Programmez une fonction `(chromatique num dt)` retournant un son contenant les 12 notes de la gamme chromatique à partir de la note de numéro MIDI `num`, et espacées d'un temps `dt` (exprimé en *frames*). Exemple :

```
(pstream-play ps (chromatique 60 (sec 1)))
```

Exercice 3 Programmez la fonction `(majeure num dt)` retournant un son contenant les 7 notes de la gamme majeure obtenue en partant de la note MIDI numéro `num`, les notes étant espacées d'un temps `dt` (exprimé en *frames*). Exemples :

```
(pstream-play ps (majeure 60 (sec 1)))        ; gamme majeure en DO
(pstream-play ps (majeure 67 (sec 1)))        ; gamme majeure en SOL
```

Tournez la page....

Exercice 4 La fonction `(build-sound n f)` permet de construire des sons avec des formules mathématiques (trigo) particulières. Ici `n` représente le nombre d'échantillons (*frames*) du son désiré et `f` est la *génératrice* : une fonction prenant un numéro `i` d'échantillon et retournant la valeur de l'amplitude *f(i)* de cet échantillon dans [-1,+1].

a) Produisez et écoutez un son pur `foo` de 5 secondes à 440 Hz en définissant ainsi un oscillateur :

```
(define (osc freq i)    ; fréquence = freq
  (local [(define t (/ i 44100))]
    (sin (* 2 pi freq t))))
```

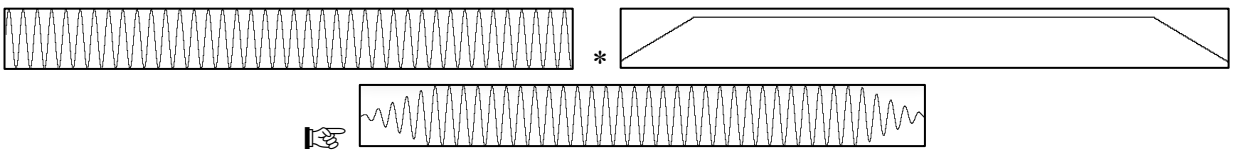
b) En audio, la **modulation d'amplitude** (AM) consiste à ne pas avoir une amplitude constante mais à la faire varier avec un oscillateur sinusoïdal à basse fréquence (LFO = *Low Frequency Oscillator*). Programmez un son `foo-am` de 10 secondes à 440 Hz dont l'amplitude est modulée par un LFO à 2.5 Hz.

c) On peut aussi jouer sur des fréquences variables. Essayez d'introduire t^2 au lieu de `t` dans une nouvelle définition `osc-t2` de `osc` et produisez un son `foo-chirp` de 10 sec.

d) A propos, jusqu'à combien de Hz percevez-vous un son ? *En théorie jusqu'à 20000 Hz mais testez...* Votre audition baissera graduellement au long de votre vie, surtout si vous réglez vos écouteurs trop forts.

Exercice 5 Le ***fading***. Il s'agit d'**atténuer** un son en un temps très court, plutôt que de le couper brutalement : les variations brutales de fréquences entraînent souvent des *clics* désagréables dans les sons. Idem au début pour le lancer. Que le son naisse et meure en douceur :-)

a) En utilisant `build-sound`, programmez la fonction `(rampe len r-in r-out)` retournant un son de longueur `len` *frames*, valant 1 partout sauf sur les `r-in` premières *frames* montantes de 0 à 1, et sur les `r-out` dernières *frames* descendantes de 1 à 0. Lequel son sera bien entendu inaudible, son seul but est d'être multiplié par un autre son pour en **modifier l'enveloppe** (cf cours page 3) ! On peut multiplier deux sons *frame à frame* avec la fonction `(rs-mult snd1 snd2)`. Avec `rs-draw`, faites dessiner cette rampe comme ci-dessous.



b) En déduire la fonction `(fade-in-out snd r-in r-out)` retournant une version du son `snd` atténuée aux deux extrémités, avec des rampes comportant `r-in` et `r-out` *frames*. Testez sur l'un des sons obtenus dans les exercices précédents. *Toujours insérer dans une animation un son avec fading aux deux bouts !*

*N.B. Dans les **DAW** (Digital Audio Workstations : logiciels de musique numérique), on peut souvent modifier l'enveloppe d'un morceau audio à la souris, comme le montre le synthé Harmor ci-dessous.*