DJ'Oz

LFSAB1402 – Projet 2014

Le projet de cette année consiste à écrire deux fonctions. L'une sera capable de transformer une partition musicale en un fichier audio et l'autre de mixer et transformer une ou plusieurs musiques.

Consignes

Ce projet est à réaliser par groupe de deux étudiants (pas nécessairement du même local) et à rendre sur INGInious [1] pour le jeudi 4 décembre à 18h00 (heure belge) au plus tard.

Le serveur *ne survivra pas* à une soumission massive juste avant la limite. Pensez à vous y prendre à temps en soumettant des versions intermédiaires.

Délivrables

Votre projet doit prendre la forme d'une archive nommée NOMA1-NOMA2.zip ou NOMA1-NOMA2.tgz contenant au moins trois fichiers. Un fichier (code.oz) avec votre code, un fichier (example.dj.oz) avec un exemple d'input pour votre programme et un fichier (rapport.pdf) avec votre rapport.

Le fichier example.dj.oz doit être différent du fichier d'exemple joie.dj.oz.

Votre code *doit* faire appel exactement une fois à la fonction `Projet.run` avec le contenu de example.dj.oz.

*Mentionnez vos **noms**, **prénoms** et **noma** dans votre **code** et dans votre **rapport** !*

Rapport

Dans le rapport, vous devez expliquer la structure de votre programme et les décisions de conceptions ainsi que les difficultés que vous avez rencontrées, les limitations et les problèmes connus de votre programme. Vous devez aussi y justifier toutes les constructions non-déclaratives

1. `https://inginious.info.ucl.ac.be/course/FSAB1402/project`

que vous utilisez (cellules, par exemple). Comment pourrait-on s'en passer en quoi votre programme serait-il pire dans ce cas, etc. Vous y dériverez aussi la complexité de votre/vos fonction(s) le plus précisément possible. Finalement vous devez aussi décrire toute extension que vous auriez apportée au projet. Quelques unes sont suggérées plus bas, mais il vous est libre d'en inventer d'autres.

Évaluation

Vous serez évalués sur la qualité de votre code et de votre rapport. Ceci implique que votre code doit non-seulement être correct et raisonnablement efficace, mais il doit aussi être documenté, structuré, lisible, compréhensible, réutilisable et modifiable. Votre rapport doit être clair, structuré et dans un français (ou anglais) raisonnablement correct. Pensez à utiliser un correcteur orthographique.

Les extensions que vous pouvez apporter au projet peuvent faire la différence entre un 18 et un 20 mais ne pourront pas sauver un projet mal fait. Concentrez-vous d'abord sur ce que l'on vous demande.

Contact

Pour toute questions, n'hésitez pas à me contacter (guillaume.maudoux@uclouvain.be) ou à venir me trouver dans mon bureau au Réaumur, local a.151 (premier étage, en face du ficus benjamini).

Objectif

Le projet se décompose en deux parties fortement liées. La première consiste à mixer des fragments de musique, et la deuxième à générer de tels fragments sur base d'une partition.

Ne commencez pas l'implémentation sans avoir lu tout l'énoncé. Il n'est pas forcément pertinent d'implémenter le projet dans l'ordre utilisé dans ce document.

Interpréter une partition

Une partition est donnée sous la forme de listes de listes de listes de ... de listes de notes et/ou de transformations de partitions.

⟨*partition*⟩ ::= ⟨*note*⟩
| ⟨*transformation*⟩
| ⟨*suite de partitions*⟩

⟨*suite de partitions*⟩ ::= `nil`
| ⟨*partition*⟩ `'|'` ⟨*suite de partitions*⟩

Les notes sont données en notation scientifique [2]. Ces notes représentent un son de fréquence pure à volume maximal durant exactement une seconde. On admet par ailleurs une notation raccourcie pour les notes de la quatrième octave. Si l'octave n'est pas spécifiée, on supposera donc par défaut qu'il s'agit de la quatrième.

⟨*note*⟩ ::= `silence` | ⟨*nom*⟩ | ⟨*nom*⟩⟨*octave*⟩ | ⟨*nom*⟩`#`⟨*octave*⟩

⟨*nom*⟩ ::= `a` | `b` | `c` | `d` | `e` | `f` | `g`

⟨*octave*⟩ ::= `0` | `1` | `2` | `3` | `4`

2. `http://en.wikipedia.org/wiki/Scientific_pitch_notation`

Cette notation permet d'écrire de manière succincte une partition mais n'est pas très pratique à manipuler puisque `a2` est un atome tandis que `a#2` est un tuple. Pour cette raison, nous vous fournissons la fonction `ToNote` qui transforme une note en note étendue. Vous êtes libres de modifier et d'adapter cette fonction à vos besoins.

⟨*note étendue*⟩ ::= `note(nom:`⟨*nom*⟩ `octave:`⟨*octave*⟩ `alteration:`⟨*alteration*⟩`)`

⟨*alteration*⟩ ::= `none` | `'#'`

```
fun {ToNote Note}
   case Note
   of Nom#Octave then note(nom:Nom octave:Octave alteration:'#')
   [] Atom then
      case {AtomToString Atom}
      of [N] then note(nom:Atom octave:4 alteration:none)
      [] [N O] then note(nom:{StringToAtom [N]}
                         octave:{StringToInt [O]}
                         alteration:none)
      end
   end
end
```

Enfin, une transformation de partition consiste à en modifier la durée, la hauteur ou le volume. Ces paramètres peuvent être soit fixés, soit ajustés proportionnellement à leur valeur courante.

⟨*transformation*⟩ ::= `muet(` ⟨*partition*⟩ `)`
| `duree(` `secondes:`⟨*float*⟩ ⟨*partition*⟩ `)`
| `etirer(` `facteur:`⟨*float*⟩ ⟨*partition*⟩ `)`
| `bourdon(` `note:`⟨*note*⟩ ⟨*partition*⟩ `)`
| `transpose(` `demitons:`⟨*entier*⟩ ⟨*partition*⟩ `)`

`fun {Interprete Partition} ... end` Vous devez construire une fonction `Interprete` qui prend en argument une partition et renvoie une voix, c'est à dire une liste d'échantillons de musique. Un échantillon se caractérise par une durée en secondes (un nombre réel), une hauteur en nombre de demi-tons par rapport à `a4` (un entier), et un nom d'instrument dont la valeur sera toujours `none`, sauf si vous implémentez l'extension associée.

⟨*voix*⟩ ::= `nil`
| ⟨*échantillon*⟩ `'|'` ⟨*voix*⟩

⟨*échantillon*⟩ ::= `silence(duree:`⟨*secondes*⟩ `)`
| `echantillon(hauteur:`⟨*entier*⟩ `duree:`⟨*secondes*⟩ `instrument:`⟨*instrument*⟩`)`

⟨*instrument*⟩ ::= `none` | ⟨*oz atom*⟩

Contrairement à une partition, une voix ne peut pas contenir de listes imbriquées. De ce point de vue, `Interprete` ressemble à `Flatten`.

`echantillon(...)` Un échantillon se définit par une durée (un réel), un instrument (`none` par défaut) et une hauteur. Pour la hauteur (et donc la fréquence) des note, nous nous baserons sur l'échelle chromatique [3] [4]. Dans ce modèle la gamme complète est composée de 12 demi-tons. Une

3. `http://fr.wikipedia.org/wiki/Échelle_chromatique`
4. `http://fr.wikipedia.org/wiki/Fréquences_des_touches_du_piano`

note située douze demi-tons plus haut qu'une autre a une fréquence deux fois plus grande. `a4` a donc une fréquence deux fois plus élevée que `a3`. Nous définissons la hauteur d'un échantillon comme le nombre de demi-tons qui le séparent de `a4`, le "la" de référence à 440 Hz.

`duree()`, `etirer()` Bien qu'elles puissent paraître simples, ces transformations impliquent certaines subtilités. Par exemple, pour etirer une partition il suffit d'augmenter la durée de chacune de ses notes. Par contre, pour fixer la durée d'une partition, il faut modifier la durée de chaque note en fonction de la durée totale de la partition.

`muet()`, `bourdon()`, `transpose()` Un bourdon est une partition dans laquelle toutes les notes sont identiques. En particulier

```
{Interprete muet(P)} == {Interprete bourdon(note:silence P)}
```

pour toute partition `P`. Transposer une partition revient à décaler toutes les notes d'un nombre donné de demi-tons. Dès lors,

```
{Interprete transpose(demitons:~12 a4)} == {Interprete a3}
```

Mixer la musique

La deuxième partie du projet consiste à mixer des morceaux de musique pour obtenir un unique vecteur audio.

Une musique est composée de voix et de morceaux issus de fichiers audio au format WAVE sur lesquels sont appliqués des filtres. Plusieurs musiques peuvent être jouées en même temps.

⟨*musique*⟩ `::= nil`
| ⟨*morceau*⟩ `'|'` ⟨*musique*⟩

⟨*morceau*⟩ `::= voix(`⟨*voix*⟩`)`
| `partition(`⟨*partition*⟩`)`
| `wave(`⟨*nom de fichier*⟩`)`
| ⟨*filtre*⟩
| `merge(` ⟨*musiques avec intensités*⟩ `)`

⟨*musiques avec intensités*⟩ `::= nil`
| (⟨*float*⟩ `#` ⟨*musique*⟩) `'|'` ⟨*musiques avec intensités*⟩

⟨*filtre*⟩ `::= renverser(` ⟨*musique*⟩ `)`
| `repetition(nombre:`⟨*naturel*⟩ ⟨*musique*⟩ `)`
| `repetition(duree:`⟨*secondes*⟩ ⟨*musique*⟩ `)`
| `clip(bas:`⟨*float*⟩ `haut:`⟨*float*⟩ ⟨*musique*⟩ `)`
| `echo( delai:`⟨*secondes*⟩ ⟨*musique*⟩ `)`
| `echo( delai:`⟨*secondes*⟩ `decadence:`⟨*float*⟩ ⟨*musique*⟩ `)`
| `echo( delai:`⟨*secondes*⟩ `decadence:`⟨*float*⟩ `repetition:`⟨*naturel*⟩ ⟨*musique*⟩ `)`
| `fondu(ouverture:`⟨*seconds*⟩ `fermeture:`⟨*secondes*⟩ ⟨*musique*⟩ `)`
| `fondu_enchaine(duree:`⟨*secondes*⟩ ⟨*musique*⟩$_1$ ⟨*musique*⟩$_2$ `)`
| `couper(debut:`⟨*secondes*⟩ `fin:`⟨*secondes*⟩ ⟨*musique*⟩ `)`

`fun {Mix Interprete Music} ... end` Vous devez construire une fonction `Mix` qui prend deux arguments et renvoie un vecteur audio. Le premier (`Interprete`) est une fonction permettant d'interpréter des partitions et le second (`Music`) est une musique respectant le format ci-dessus. Un vecteur audio est une liste de flottants compris dans l'intervalle $[-1{,}0; 1{,}0]$. Ils représentent l'échantillonnage [5] du signal sonore à une fréquence de 44 100 Hz. Votre fonction va donc passer d'une représentation symbolique de la musique à une liste de valeurs susceptibles d'être jouées par votre ordinateur.

`partition()` Pour mixer une partition, il faut utiliser la fonction `Interprete` passée comme premier argument à la fonction `Mix` afin d'obtenir une voix. Ensuite, il faut appliquer la règle ci-dessous qui permet de mixer une voix. Ceci revient à dire que

```
{Mix Interprete partition(P)} == {Mix Interprete voix({Interprete P})}.
```

`voix([echantillon(...) ...])` Pour mixer une voix provenant de l'interprétation d'une partition, il faut convertir chaque échantillon en un vecteur audio. Pour jouer plusieurs échantillons de suite, il suffit de concaténer leurs vecteurs audio respectifs.

`echantillon(hauteur:`h `duree:`s `instrument:none)`, `silence(duree:`s`)` Si un échantillon n'a pas d'instrument, il faut générer un son pur de la fréquence indiquée. Dans notre cas, la fréquence f est donnée en fonction de la hauteur h de la note par la formule

$$f = 2^{h/12} * 440\,\text{Hz}$$

Une fois la fréquence connue, on obtient le vecteur audio en échantillonnant un sinus de cette fréquence à 44 100 Hz. On obtient une suite $(a)_i$ de valeurs respectant la formule

$$a_i = \frac{1}{2} \sin\left(\frac{2\pi * f * i}{44100}\right) \tag{1}$$

Si le son dure une seconde, le vecteur audio correspondant doit contenir exactement 44 100 valeurs. Il va de soi que toutes les valeurs d'un silence sont à zéro.

`wave()` Nous vous fournissons une méthode pour obtenir un vecteur audio à partir d'un fichier WAVE. Ce format étant assez complexe, nous ne supportons que l'encodage PCM à 8, 16, 24 ou 32 bits. La fréquence d'échantillonnage doit toujours être de 44 100 Hz.

La bibliothèque fournie pour ce projet contient la fonction `Projet.readFile` qui permet de lire des fichiers WAVE. Le dossier `wave/animaux` contient par ailleurs plusieurs fichiers audio avec lesquels vous pouvez faire des tests.

`merge()` L'opération `merge` permet de jouer plusieurs musiques en même temps. Chaque musique est associée avec une intensité, ce qui permet de combiner une voix principale avec un fond sonore plus doux. Concrètement, chaque vecteur audio doit d'abord être multiplié par son intensité avant d'être combiné avec les autres. Afin de ne pas saturer le vecteur final, la somme des intensités ne doit pas dépasser 1,0.

Pour combiner deux vecteurs audio, il suffit de les additionner. Le vecteur final a la longueur de la plus longue des musiques à combiner. Les musiques trop courtes sont complétées avec du silence. Voici un exemple de merge :

```
merge([ 0.5#Music1 0.2#Music2 0.3#Music3 ])
```

5. http://fr.wikipedia.org/wiki/Échantillonnage_(musique)

filtres

renverser Une musique est renversée quand elle est jouée à l'envers, en commençant par la fin. Cette technique hautement avancée permet aux spécialistes de découvrir ou de cacher des messages sataniques.[6][7] À utiliser avec modération...

repetition Cette opération répète une musique, soit un certain nombre de fois, soit jusqu'à atteindre une durée donnée. Si la durée fixée est plus petite que celle de la musique, la musique est tronquée.

clip Le clipping consiste à plafonner les valeurs possibles dans le vecteur audio. Si le vecteur [−1, −0,9, −0,8, −0,7, −0,6, −0,5, −0,4, −0,3, −0,2, −0,1, 0, 0,1, 0,2, 0,3, 0,4, 0,5, 0,6, 0,7, 0,8, 0,9, 1,0] est clippé entre −0,15 et 0,5, on obtient le vecteur [−0,15, −0,15, −0,15, −0,15, −0,15, −0,15, −0,15, −0,15, −0,15, −0,1, 0, 0,1, 0,2, 0,3, 0,4, 0,5, 0,5, 0,5, 0,5, 0,5, 0,5]

echo Vous pouvez introduire de l'écho sur base de trois paramètres. Tout d'abord le délai donne le temps entre la musique initiale et sa répétition. Ensuite, la décadence donne l'intensité de l'écho par rapport à la musique initiale. Une décadence plus grande que 1,0 donnerait un écho plus puissant que la musique initiale. Enfin, vous pouvez introduire plusieurs échos consécutifs avec l'option de répétition. Dans ce cas, la décadence d s'applique plusieurs fois (le premier écho a une intensité de d, le second de $d * d$ et ainsi de suite.

De même que pour `merge`, veillez à ce que la somme des intensités de la musique et de ses échos soit toujours de 1,0. Quelques exemples :

```
{Mix I [echo(delai:1.0 M)]} ==
  {Mix I [merge([0.5#M 0.5#[voix([silence(duree:1.0)]) M]])]}
{Mix I [echo(delai:1.0 decadence:0.5 repetition:2 M)]} ==
  {Mix I [merge([
    0.57#M
    0.29#[voix([silence(duree:1.0)]) M]
    0.14#[voix([silence(duree:2.0)]) M]
  ])]}
```

fondu, fondu_enchaine Le fondu est une technique qui vise à adoucir les transitions entre les morceaux de musique. L'option `ouverture:` donne la durée en seconde à partir du début du morceau pendant laquelle l'intensité de la musique va croître linéairement entre 0,0 et 1,0. L'option `fermeture:` force quant-à-elle une intensité décroissante à la fin de la musique.

Le fondu enchainé applique un fondu de fermeture sur la première musique et un fondu d'ouverture sur la deuxième. De plus, il avance le deuxième morceau de la durée du fondu pour que les deux fondus soient parfaitement superposés. Vous êtes libre d'utiliser le type de fondu[8] de votre choix pour le fondu enchainé. Le fondu linéaire (à gain égal) reste cependant le plus simple.

Il est interdit d'appliquer un fondu sur un vecteur audio trop court (moins long que la durée du fondu).

6. http://fr.wikipedia.org/wiki/Backmasking#Messages_sataniques
7. http://en.wikipedia.org/wiki/Backmasking#Satanic_backmasking
8. http://fr.wikipedia.org/wiki/Fondu

couper Cette opération permet d'obtenir une partie d'un morceau. Les paramètres donnent le début et la fin de l'intervalle désiré. Si l'intervalle tombe en dehors de la musique, le vecteur est complété par des zéros là où c'est nécessaire. Par exemple `{Mix [couper(debut:~1.0 fin:0.0 M)]}` donne toujours un silence d'une seconde.

Extensions

Les extensions visent à améliorer, étendre et mettre en œuvre des aspects spécifiques de votre implémentation. Elles ne sont pas obligatoires et ne permettent pas de sauver un projet mal réalisé. Combinées avec un bon projet, une extension peut apporter un bonus de deux points, et deux extensions (ou plus) un bonus de trois points.

Lissage

Comme vous le remarquerez surement lors de l'implémentation, la synthèse de sons sur base de notes crée des bruits désagréables entre chaque note. Pour palier ce problème, il faut adoucir le début et la fin des notes.

Dans cette extension, vous allez modéliser une enveloppe sonore [9] semblable à celles utilisées par les synthétiseurs que l'on trouve dans les instruments électriques.

L'enveloppe la plus simple est un trapèze. Le son augmente doucement jusqu'à sa puissance maximale, continue aussi longtemps que désiré, puis diminue doucement jusqu'à disparaître.

Cette enveloppe revient à utiliser le filtre de fondu sur toutes les notes. Vous êtes libres d'utiliser la durée de fondu qui vous convient, ou d'utiliser une autre enveloppe pour un son plus réaliste (voir le lien sur l'enveloppe sonore).

Instruments

Cette extension met à profit le champ `instrument:` des échantillons formant une voix. Dans ce cas, on étend la grammaire des transformations de partitions avec la transformation `instrument`

⟨*transformation*⟩ ::= ...
| `instrument(nom:`⟨*instrument*⟩ ⟨*partition*⟩`)`

instrument() Change l'instrument utilisé pour jouer la partition. L'instrument par défaut est `none`. C'est toujours l'instrument situé au plus près de la note qui est utilisé. Autrement dit

```
{Interprete instrument(nom:guitare instrument(nom:piano a4))} ==
            {Interprete instrument(nom:piano a4)}
```

Lors de la synthèse d'un échantillon dont l'instrument n'est pas `none`, il ne faut plus générer vous même le vecteur audio mais utiliser les sons de référence fournis dans l'archive instruments.tgz. Ces fichiers doivent être placés dans wave/instruments. Tous les fichiers d'instrument ont le format `<nom>_<note>.wav`. Pour générer le nom de fichier à lire en fonction de la note, allez lire la documentation de Mozart/Oz, et plus spécialement la fonction `VirtualString.toAtom`

À vous de définir ce qui se passe quand la durée de l'échantillon est différente de celle du fichier. Si vous deviez couper le vecteur audio, pensez à lisser les coupure par un fondu.

9. `http://fr.wikipedia.org/wiki/Enveloppe_sonore`

Créativité

Le but de cette extension est de mettre en œuvre les fonctionnalités du projet. Vous pouvez transcrire une partition à plusieurs voix, ou mixer quelques extraits en boucle à la Ramstein. Usez de votre créativité et produisez une musique originale d'une taille raisonnable.

La qualité de votre musique sera jugée sur base du fichier final et sur le nombre de transformations. Autrement dit, il faut que votre musique comporte un certain nombre de transformations et produise un résultat qui ne déchire pas les oreilles.

Vous pouvez bien entendu utiliser d'autres fichiers audio que ceux fournis avec le projet. Dans ce cas, n'oubliez pas de les remettre dans l'archive finale.

Vous devez remettre un fichier creation.dj.oz qui contient votre création et un fichier creation.wav qui contient le résultat fourni par votre programme.

Effets complexes

La plus libre et par conséquent la plus dure des extensions. Vous devez ajouter trois filtres sur les musiques, comme par exemple des filtre passe-bas, passe-haut ou de modulation d'amplitude. Si vous êtes motivés, ajoutez du phaser, de la distortion, du fuzz ou du wah-wah !

Trucs et astuces

— Un squelette de code est disponible dans l'archive. Celui-ci explique les quelques méthodes déjà fournies et donne le format attendu pour la soumission.
— Matlab et/ou Octave peuvent aider à visualiser le contenu d'un fichier WAVE. Utilisez simplement la commande `plot(wavread('nom_du_fichier.wav'));`.
— Commencez par de petits exemples. Implémentez une seule transformation à la fois et vérifiez à chaque étape si vos résultats sont valides.
— Pour épargner la bande passante, vous ne devez pas inclure les répertoires wave/instruments et wave/animaux dans votre soumission. Ces dossier seront ajoutés par nos soins lors des tests.