Rapport Projet Info 2014 DJ'OZ

Charlotte Poncette 9458-12-00 et Timothée Malengreau 6618-13-00

December 4, 2014

1 Fonctionnement du programme

Mix et Interprete servent à envoyer les partitions et musiques prisent en arguments vers deux fonctions, "Lire" et "Final" qui servent à gérer les différents cas (et donc fonctions) des éléments des listes reçues en entrée.

Mix reçoit la fonction Interprete et une musique en entrée. Elle commence par aplatir la liste afin de pouvoir la parcourir aisément. La musique est ensuite envoyée vers la fonction "Final" qui va déterminer le type du premier élément de la musique. Si c'est une fonction comme "Couper", "Merge" ou autre fonction de "Mix", "Final" va appeler la fonction correspondante, transformer le résultat envoyé en vecteur audio si nécessaire, puis positionner le vecteur audio dans la liste et faire un appel récursif de "Final" avec la suite de la musique. Si l'élément examiné est une partition, "Final" va appeler la fonction Interprete qui va a son tour aplatir la liste, puis appeler la fonction "Lire" qui va appeler la fonction correspondant à l'élément envoyé, puis renvoyer le résultat sous forme d'échantillon et rappeler "Lire" afin d'examiner les autres éléments de la partition.

Le programme devient vite très lent dès que le nombre de transformations complexes augmente. Le programme effectue Flatten de nombreuses fois car nous avons eu des problèmes de listes qui restaient imbriquées et du coup, le code ne fonctionnait plus.

2 Complexité

La complexité totale maximale est de N^2 car la fonction "Mix" fait appel N fois à des transformations de complexité maximale N. Cette complexité est importante et certainement due à une implémentation naive du code.

3 Difficultés rencontrées

3.1 Difficultés de compréhension

Du à la longueur de l'énoncé et à la quantité d'éléments différents à gérer (notes, partitions, filtres, musiques, etc) il a été difficile de se lancer dans l'écriture du programme car nous ne comprenions pas l'objectif final dans sa globalité. Nous avons compris au fur et à mesure de l'écriture des fonctions les interactions entre celles-ci, ce qui explique en partie notre implémentation naïve.

3.2 Oz

Nous avons utilisé que très peu l'émulateur Mozart et avons donc rencontré quelques difficultés lors de son utilisation, surtout au niveau de la compréhension et de la résolution des erreurs. De plus, étant plus habitués à d'autres langages tel que Java, nous avons rencontrés des difficultés lors de l'implémentation de notre code (pas d'autoboxing, impossibilité de modifier des variables, difficulté d'appeler des fonctions déjà existantes de Oz car pas d'api et informations disparates sur internet). Nous avons également eu des problèmes car l'ordre d'apparition des fonctions avait, parfois, une influence sur le bon fonctionnement du programme, et nous avons également été limités lors de la création de notre exemple car l'émulateur n'était pas autorisé à utiliser plus de mémoire RAM.