LFSAB1402
Informatique 2

2018-2019

Projet - Oz Player

AMINE AIT SAID - 64751400
NIKITA TYUNYAYEV - 19761500

6 décembre 2018

Introduction

Le présent rapport reprend les points importants de notre code Oz Player. Il nous était demandé d'implémenter deux fonctions en Oz. La premiere, {PartitionToTimedList}, qui devra transformer une partition musicale en une liste de notes et d'accords, chacun associe a une duree. La seconde, {Mix}, mixera plusieurs formats d'entree afin d'emettre un fichier audio en format WAV.

Limitations et problèmes du programme

La principale difficulté fut de bien saisir les tâches que doivent atteindre les fonctions {PartitionToTimedList} et {Mix}. Élaborer un code composé d'une centaine de lignes demande une bonne connaissance de l'ensemble du programme ainsi qu'une grande concentration car de simples erreurs de syntaxe peuvent vite arriver.
Tout d'abord, nous avons commencé à utiliser des constructions non-déclaratives telle que les cellules dans notre programme. En pensant que cela serait plus simple au niveau de l'implémentation ainsi que de la compréhension du code mais cela a entraîné des erreurs dont nous ne connaissions pas la cause.
D'autre part, dans la deuxième partie du projet, nous pensions qu'une "music" est une liste composée de listes d'échantillons. Ce qui est complètement absurde en observant les filtres telles que "cut", "fade" ou encore "loop". Nous avons également eu certaines difficultés à implémenter les fonctions {Fade} et {Cut}. Nous sommes quand même parvenus à créer une boone fonction {Cut}. Par contre, le code de la fonction {Fade} a été implémenté mais il ne fonctionne pas correctement pour des problèmes aux bornes ou extrêmes.
Finalement, en assemblant les différentes parties de codes, nous avons eu certains problèmes de compliation dû à des erreurs de syntaxe.

Justification des constructions non-déclaratives

Nous n'avons utilisé aucune construction non-déclarative car après l'avoir expérimenté, de nombreux problèmes sont apparus dans notre programme. C'est pour cette raison que nous nous sommes tournés vers les fonctions récursives.

Choix d'implémentation

Tout d'abord, nous avons décidé de nous baser uniquement sur la programmation fonctionnelle. Dans la première partie, nous avons voulu introduire deux nouvelles fonctions du nom de {ChordToExtended} et {ChordOrNoteToExtended} pour apporter plus de clarté dans notre raisonnement et aérer notre fonction {PartitionToTimedList}. Dans la fonction {NoteToExtended}, nous avons également jugé pratique de considérer le cas où des notes pourrait être des silence. Cette décision est justifié car la forme de celle-ci diffère de celle d'une note.
Dans la globalité du code, nous avons décidé l'organiser en sous-fonctions pour une meilleure structuration du projet. Cela permettant bien sûr de mieux le comprendre.

Complexités

1) La transformation "**duration**"
   Soit N le nombre d'éléments constituant la liste Partition et soit M le nombre d'élément contenu dans la liste Partition aplatit.
   On commence par appliquer {PartitionToTimedList} sur la partition, ce qui nous coûte O(M). Pour calculer la durée totale, il nous suffit de parcourir chaque élément de notre liste qu'il soit

note ou accord puisque les notes contenus dans un accord ont tous la même durée. Nous pouvons alors dire que la complexité est en O(N). Enfin, il nous reste à appliquer la fonction {Stretch} sur notre partition, qui possède une complexité en O(M) puisque nous devons appliquée cette fonction à chaque élément. La complexité total est donc de O(N + M)

2) Le "**merge**"
Soit N la taille de la liste, mise à plat, la plus grande prise en argument.
Puisque nous devons parcourir les deux listes prises en argument pour être sommé membre à membre, les multiplier par une intensité de par l'application de la fonction {Map} ou encore pour l'appel récursive de la sous-fonction {MergeHelper}.

3) Le filtre "**loop**"
Soit N le nombre de secondes indiquées.
Nous appliquons simplement la sous-fonction {LoopHelper} de manière récursive jusqu'à ce que la durée dépasse le seuil de 0.0. Ainsi, la complexité de la fonction {Merge} est en O(N).

Extensions

Nous n'avons pas utilisé d'extension.