Université du Québec
à Chicoutimi

Projet : Création d’un langage de composition algorithmique

Cahier des charges et présentation du projet

v0.0.0

0. Résumé du projet

Le but de ce projet est la création d’un langage de composition musicale textuel, inspiré du langage MML, accompagné de son interpréteur, pour composer rapidement, designer un son ou fabriquer une onde précise, à l’aide d’une syntaxe claire et concise, de maths et de macros. Il suffira simplement d’écrire une partition sous forme de texte (avec l’éditeur que l’on veut), d’écrire l’expression mathématique d’un signal (sinusoïde, onde carrée voire fractale, peu importe), puis de lancer l’interpréteur qui lira la partition et jouera la musique écrite en la générant à partir de l’expression du signal comme s’il s’agissait d’un instrument de musique (un peu comme on peut faire jouer des notes de musique par un synthétiseur MIDI en choisissant l’instrument que l’on veut).

Contexte

La musique assistée par ordinateur est née de l'association entre l'ordinateur et le synthétiseur vers les années 1970[1]. Avec les évolutions de l'ordinateur au cours du temps, il a très vite été possible de créer de la musique sans synthétiseur, en utilisant les fonctionnalités des composants embarqués. En 1996, Steinberg a publié une nouvelle version de Cubase, une DAW (station de travail audionumérique), permettant cette fois de créer de la musique sans synthétiseur. Parallèlement, les premiers trackers voient le jour en 1987 avec Ultimate Soundtracker, NoiseTracker, jusqu'au plus connu aujourd'hui, MilkyTracker. Ces logiciels présentaient un nouveau style d'interface dirigée par le clavier, permettant une composition facile et efficace sans clavier MIDI.

D'autres méthodes de création musicale par ordinateur ont aussi été développées, appartenant à un groupe de méthodes appelé aujourd'hui musique algorithmique ou composition algorithmique. Ces méthodes utilisent le potentiel de calcul de l'ordinateur et souvent aussi la génération pseudo-aléatoire pour générer de la musique à l'aide de réseaux de neurones ou de formules mathématiques. Alors que certaines de ces méthodes explorent le non-déterminisme à travers la génération complète ou guidée, la popularisation de la programmation a entraîné la création de langages de programmation spécialisés dans la création musicale seulement assistée et automatisée, redonnant le contrôle aux artistes.

La musique est un langage. Comme beaucoup de langages, elle peut être écrite. La musique manuscrite a beaucoup évoluée et ses variations dans l'histoire et dans les différentes cultures sont toutes très intéressantes, reflétant des styles musicaux particuliers pour lesquels elles sont adaptées. La partition occidentale, complexe mais standardisée, a évolué pour permettre la représentation de la plupart des variations stylistiques musicales des différentes cultures à travers de nouveaux symboles tout comme d'autres standards comme UTF-8. Cependant, l'évolution de l'écriture a subit un changement important récemment avec l'invention de l'ordinateur. De nouvelles grammaires sont nées, le « langage SMS » par exemple ou aussi tous les différents langages de programmation. Il est beaucoup plus facile pour un ingénieur logiciel de permettre à un ordinateur d'interpréter du texte qu'un dessin et cela rend la partition occidentale classique inadaptée à l'écriture musicale sur ordinateur. Certains logiciels existent pour ce genre de tâche, comme Musescore ou Sibelius, mais ils sont très complexes et difficiles à prendre en main. L'approche plus commune pour un logiciel est d'utiliser le langage MIDI mais il s'écrit en binaire, ce qui nécessite encore des interfaces graphiques complexes. Il est aussi un peu laxe car ses implémentations diffèrent en pratique (certains programmes encodent les silences par des évènements de touches enfoncées avec une vélocité de 0 par exemple alors que d'autres utilisent des évènements de touches relâchées). Ainsi, quitte à repousser des utilisateurs moins expérimentés, des langages plus spécialisés ont vu le jour pour mettre le contrôle minutieux mais flexible des langages de programmation au service de la création musicale. Vous pouvez en trouver une liste non-exhaustive sur Wikipédia comprenant par exemple Kyma, spécialisé dans le design sonore et utilisé pour les films

1 https://fr.wikipedia.org/wiki/Musique_assist%C3%A9e_par_ordinateur

Wall-E, Nemo et quelques Star Wars. Chaque langage a sa propre vision de la musique et ses spécialités : certains peuvent être transformés parfaitement en MIDI voire pouvoir recréer une partition occidentale en PDF comme LilyPond.

En tant que compositeur du dimanche, je préfère une syntaxe la plus minimaliste possible ; j'ai appris à utiliser BeepComp il y a quelques années, un interpréteur et éditeur de MML, et j'ai eu la chance de discuter avec son créateur. Après avoir conçu quelques outils pour faciliter son utilisation, j'ai commencé à créer moi-même des variations du MML accompagnées d'interpréteurs dans le but de personnaliser la syntaxe et de l'améliorer mais aussi de rendre ce type de langage plus accessible aux malvoyants qui se plaignaient du manque d'accessibilité dans BeepComp. Dans le cadre de ce projet, je compte créer un langage accessible, minimaliste mais plus complet encore et qui donne plus de liberté aux compositeurs par l'utilisation de variables et expressions mathématiques tout en restant simple à prendre en main.

Objectif

Une partition simple mais personnalisée

La partition, de forme textuelle et inspirée du MML, ne contiendra que les notes (un nombre variable défini en argument mais que des lettres uniques de l'alphabet, minuscules et majuscules), des indications de tempo et des changements de variables données en paramètres de l'interpréteur. Tout effet musical qui peut être représenté par des changements de variables ne doit pas faire l'objet de symboles imposés. En revanche, comme il est très fréquent en MML de changer d'octave ou de volume, des symboles peuvent être définis en paramètre de l'interpréteur comme des macros en C, changeant une variable de la même manière à chaque fois pour faciliter la composition (on pourra par exemple définir '<' comme baissant d'une octave, c'est-à-dire diminuant une variable arbitraire 'o' de 1).

Des maths

Pour libérer la créativité au maximum, le langage à créer ne doit pas imposer ses propres instruments mais permettre d'en créer à partir d'expressions mathématiques de signal. Il pourra par contre proposer des exemples d'expressions valides comme des expressions de sinusoïdes ou d'ondes carrées. Cependant, la définition des instruments ne doit pas alourdir la syntaxe alors elle sera annexée par les arguments de l'interpréteur. Dans cette définition d'instrument, des variables prédéfinies seront disponibles telles que 't' pour le temps en secondes, 'n' pour la note (prendra la valeur de l'indice de la note dans les notes disponibles) mais aussi 'T' pour le temps total du morceau.

Des pentes

Il sera possible d'inclure des pentes dans la partition pour modifier des variables de manière linéaire par des expressions mathématiques définies en arguments et qui peuvent aussi accéder aux autres variables de l'instrument en plus de la variable 't' (temps en seconde) et 'T' (temps total de la pente). Ainsi, chaque pente aura un nom utilisé dans la partition pour modifier une variable, et cette pente sera délimitée par des symboles pour contenir une sous-partition qu'elle modifiera. Le tempo de la partition sera à 60 par défaut, mais il peut être modifié comme toute variable et sa variable associée en partition sera T.

Cette fonctionnalité permettra notamment de représenter des changements linéaires communs en musique comme les crescendos / decrescendos (augmentation / diminution progressive du volume), accelerando / ritardando (accélération, ralentissement) ou encore les glissando / meend hindoustanie (glissement d'une note à l'autre) mais de manière complètement contrôlée avec des fonctions mathématiques guidant la variation de ces paramètres en fonction du temps.

Un interpréteur simple

L’interpréteur de ce langage (le « logiciel ») doit se limiter à lire la musique par défaut ou à exporter des fichiers audio dont le format est indiqué en argument. Pour accélérer la compilation, il sera mis à disposition de l’utilisateur final un système de compilation conditionnelle variant les dépendances en fonction des fonctionnalités demandées : librairies de lecture audio, codecs et formats de fichier audio… En plus de ces fonctionnalités de base, l’interpréteur pourrait imprimer un mémo sur les éléments de syntaxe disponibles et des exemples pour la partition et pour les expressions mathématiques.

Codé par un *nerd*, pour les *nerds*

Vous l’aurez peut-être deviné : le public visé possédera des connaissances basiques aussi bien en programmation qu’en mathématiques et en musique. Le minimalisme de ce langage permettra aux utilisateurs d’utiliser l’éditeur qu’ils souhaitent, avec peut-être des extensions Visual Studio Code ou Zed si j’ai du temps en plus, et ils peuvent aussi se servir de scripts pour faciliter l’édition des paramètres de l’interpréteur dans leur langage préféré (bash, zsh, Python...). Cela facilite l’utilisation du langage par des utilisateurs malvoyants qui n’auront donc pas d’interface graphique imposée (contrairement aux utilisateurs de DAW ou de certains langages comme Sonic Pi ou BeepComp qui ont leur propre éditeur). L’intégration d’expressions mathématiques est aussi un moyen ludique d’approfondir ses connaissances mathématiques. Pour rebondir sur les valeurs transmises par ce projet (volonté d’accessibilité et de créativité sans limite), son code sera disponible librement sous la licence MIT.

Enfin, ce projet pourrait servir à des fins de design sonore voire de génération de signal avancé pour des ingénieurs en télécommunications par exemple.

Ce qui a déjà été fait

Le projet décrit dans ce document a déjà débuté récemment et est disponible sur GitDab sous le nom de bng. Les premières étapes ont déjà été réalisées mais certains problèmes le rendent difficile à mener à bout.

Pour l'instant, une syntaxe a été trouvée. Plusieurs fichiers YAML ont été conçus pour démontrer cette syntaxe, dont celui-ci. Un système de *parsing* a été adapté, avec des solutions pour gérer les sous-partitions (boucles, tuplets…). Une interface en ligne de commande est disponible bien qu'incomplète, et des *features* ont été imaginées et implémentées en surface pour une compilation conditionnelle en Rust.

Cependant, la structure du projet manque de cohérence. Sans spécification précise, ce projet a évolué sans vrai fil conducteur : le prototype de fichier à parser en YAML contient à la fois la partition et ses métadonnées comme les instruments et les variables, ce qui rend le format complexe, surtout que j'ai tenté d'y ajouter un système de microtonalité par curiosité, complexifiant davantage le développement de l'interpréteur, alors que le système de génération audio n'est toujours pas en place. Dans le cadre de mon stage / projet à l'UQAC, j'ai réalisé que j'ai l'opportunité de recadrer et restructurer le projet à temps plein et de le mener à bout.

Échéancier

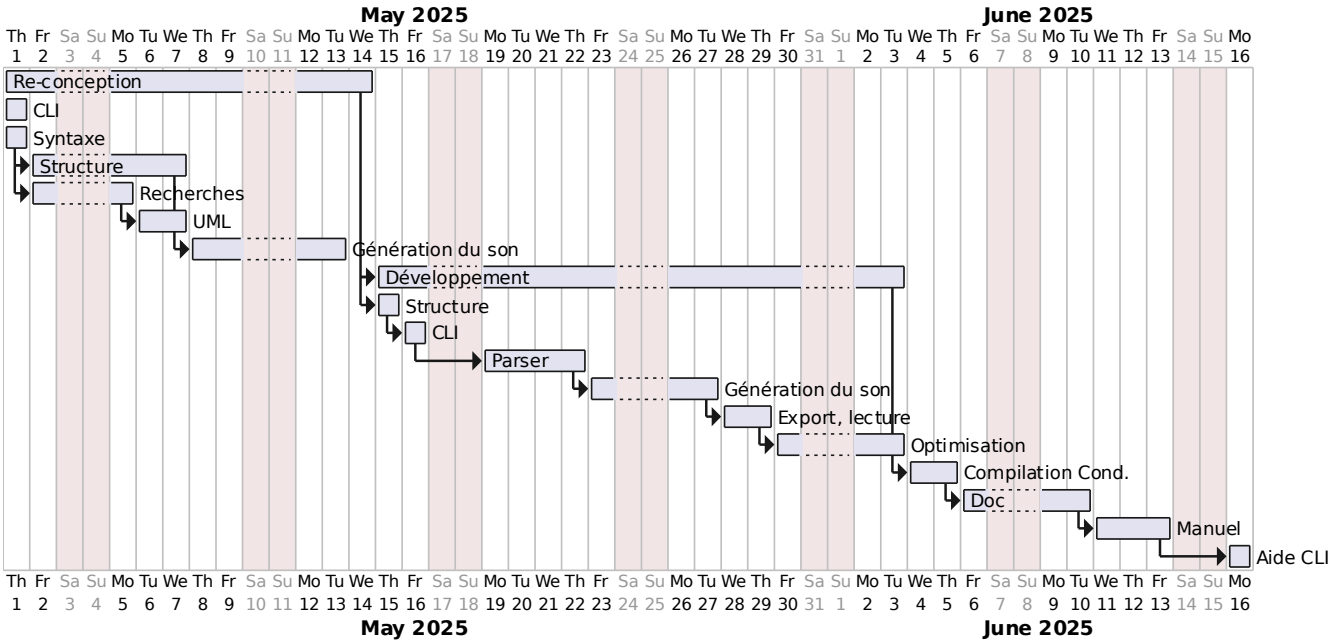
Tâches à effectuer

- Re-concevoir le projet, 9.5j
  - Ré-adapter l'interface en ligne de commande aux nouvelles spécifications, 0.5j
  - Revoir la syntaxe du langage, 1j
    - Retirer les éléments superflus
    - Ajouter un système de changement de variables
    - Ajouter un système de changement progressif de variables avec des noms d'expressions mathématiques arbitraires (définies à l'extérieur de la partition)
  - Revoir la structure du projet, 4j
    - Faire des recherches sur la structure des compilateurs et en tirer un modèle abstrait puis plus spécifique adapté du projet, 2j
    - Diviser le projet en deux : librairie de parsing + interface en ligne de commande (pour une éventuelle intégration à d'autres projets)
    - Faire un diagramme UML du projet, 2j
  - Concevoir un système de génération sonore basé sur la nouvelle syntaxe (à faire une fois le parser terminé), 4j
    - Produire des diagrammes de flux des algorithmes si possible
- Développer le projet, 14j
  - Implémenter la nouvelle structure du projet, 1j
  - Ré-implémenter l'interface en ligne de commande (fonctionnalités basiques), 1j
  - Ré-adapter le parser aux changements de syntaxe, 4j
  - Implémenter le système de génération de son, 3j
  - Développer la fonctionnalité d'export audio et de lecture en direct, 2j
  - Optimiser le code si besoin, 3j
- Implémenter la compilation conditionnelle, 2j
- Commenter tout le code et écrire la documentation de la librairie, 3j
- Écrire un manuel d'utilisation détaillé et un guide rapide, 3j
- Ajouter les mémos et exemples à l'interface en ligne de commande, 0.5j
- Éventuellement écrire une extension VSCode / Zed pour l'intégration du langage dans ces éditeurs
- Peut-être aussi concevoir et implémenter un IDE officiel pour atteindre plus d'utilisateurs potentiels

Gantt

On considérera que je travaillerai de lundi à vendredi inclus.

Voici un diagramme de Gantt des tâches requises comme échéancier du projet :



Ce diagramme a été réalisé en PlantUML et le code est disponible via cette URL : https://gist.github.com/brevalferrari/933e7126d9c48065e8513db5c6302838