BLIP

The Bizarre Language for Intermodulation Programming

Rapport de Projet (v0)

UQAC

Sujet et réalisation :	Breval Ferrari
Tuteur de projet :	Abdenour Bouzouane
Date de début :	1/05/25
Durée :	2 mois

CONTEXTE DU PROJET

BLIP est un langage de création musicale inspiré d'anciennes technologies similaires.

IL Y A TRÈS LONGTEMPS, AU JAPON…

En 1978, les japonais constatent leur premier boom dans le tout nouveau secteur du jeu vidéo. C'est aussi l'année de sortie de *Space Invaders* que vous connaissez peut-être[1]. Ces jeux vidéos bénéficiaient déjà beaucoup à l'époque de la capacité aux supports d'émettre du son. Cela était rendu possible précédemment par des solutions « maison » très rudimentaires[2] avant que des puces électroniques spécialisées ne soient commercialisées, comme la Intel 8253, la première *Programmable Interval Timer* compatible IBM[3] sortie peu après 1975[4]. Cette puce a été utilisée ensuite sur presque tous les ordinateurs x86, notamment et directement pour générer du son[5].

Avec ce genre de puce, la musique pouvait être écrite directement dans le code des jeux vidéos pour qu'elle arrive jusqu'aux enceintes. Par exemple, sur la puce 8253, il suffit de régler un *timer* pour déclencher une onde carrée de la fréquence souhaitée. Cette onde est ensuite envoyée aux enceintes, sous forme analogique (en tant que signal électrique) ou numérique (en tant qu'échantillons d'un signal discrétisé, donc des valeurs, souvent des nombres réels), qui contractent leur membrane en fonction du signal d'entrée. Cette membrane, qui vibre alors à la même fréquence que le signal périodique d'entrée, pousse l'air autour de la membrane. Si cette onde, faite de variations de pression de l'air, arrivait jusqu'à nos oreilles internes, elles étaient alors perçues comme du son.

Ce système demandait assez peu d'effort à programmer puisqu'il était possible de demander directement à ces puces des fréquences à jouer et elle se chargeait du reste (même s'il était nécessaire d'utiliser des techniques un peu plus poussées pour avoir autre chose qu'une onde carrée, comme des bruitages par exemple). Ainsi, quand ces puces ont commencé à intégrer les cartes mères d'ordinateurs personnels un peu plus puissants, et que les premiers langages interprétés ont vu le jour, des interfaces ont été élaborées pour contrôler directement les puces et produire du son soi-même.

LE LANGAGE COMME INTERFACE PRIMAIRE

C'est le cas pour les ordinateurs SHARP MZ[6], qui intégraient dans leurs OS (SP-1002 MONITOR IOCS comme SP-5001 BASIC) des commandes pour interagir avec leur puce Intel 8253. Afin d'alléger la syntaxe et comme les nouvelles capacités computationnelles le permettaient, un nouveau micro-langage a vu le jour : le *Music Macro Language*. La variante utilisée chez les ordinateurs SHARP MZ est documentée sur le site de la Fondation de Préservation de la Musique de Jeu-Vidéo (*VGMPF, Video Game Music Preservation Foundation*)[7]. La variante la plus standard et populaire est probablement celle utilisée dans le langage interprété Microsoft BASIC ; elle est décrite en détails sur le même site. En résumé, chaque note est représentée par une lettre (à l'américaine). Les octaves sont réglées par la commande « ON » pour passer à

1 https://carterjmrn.com/blog/the-evolution-of-gaming-culture-in-japan-from-arcades-to-mobile-gaming/
2 https://www.reddit.com/r/gamemusic/comments/1c4huqh/the_early_evolution_of_sound_and_music_in_video/
3 https://en.wikipedia.org/wiki/Programmable_interval_timer
4 https://bitsavers.org/components/intel/MCS80/98-153B_Intel_8080_Microcomputer_Systems_Users_Manual_197509.pdf
5 http://brokenthorn.com/Resources/OSDevPit.html
6 https://en.wikipedia.org/wiki/Music_Macro_Language
7 https://vgmpf.com/Wiki/index.php/Music_Macro_Language#Sharp_S-BASIC

l'octave N. La vitesse relative des notes peut être écrite juste après chaque note ou modifiée globalement par la commande « L » et ressemble aux nom des types de notes en anglais (4 = *quarter*, 2 = *half*, 1 = *whole*…). La vitesse du morceau est fixée par un tempo avec la commande « TEMPO ».

Vous l'aurez compris : ce langage ressemble comme deux gouttes d'eau à une vraie partition. Pour cette raison, il a pris très vite en popularité, d'où l'émergence de nombreuses variantes. Dès les années 80, des interfaces graphiques sont apparues pour créer de la musique avec un système proche du MML dans des éditeurs augmentés par des oscilloscopes et des instruments à base d'une onde customisable et de sons externes : il s'agit des *trackers*. Sur MilkyTracker par exemple, il est possible de dessiner un signal à la main et de l'utiliser comme instrument[8] (démo YouTube).

ET AUJOURD'HUI ?

Le MML existe toujours !

L'industrie musicale s'est cependant tourné vers le MIDI, un protocole de communication d'évènements (note appuyée, note relâchée…) standardisé en 1983[9]. Ce protocole a permis de généraliser et d'augmenter les technologies existantes par un système de paramètres codifiés. Aujourd'hui, les logiciels les plus utilisés pour composer sur ordinateur sont les DAWs (*Digital Audio Workstations*), des « stations » de travail centralisant plusieurs outils comme des synthétiseurs, des effets logiciels, des mixeurs etc (démo YouTube de FL Studio). Ils se distinguent par leur interface graphique très complexe et plus ou moins difficiles à apprivoiser mais très puissants. Le MML a été recalé chez les derniers fans de *chiptune*, les hackers de SNES / NES / Famicom et les nostalgiques. Pourtant, on ne peut pas nier l'influence du MML sur la culture musicale internationale.

PLUS D'OUTILS, PLUS DE FUN

Chaque outil artistique influe directement l'art qu'il permet de produire. Comme la peinture à huile et l'aquarelle ne permettent pas de produire les mêmes effets, chaque outil implique des possibilités et des contraintes, mais aussi une influence. Par exemple, les *trackers* comme les DAWs ont permis aux artistes de construire facilement des boucles, et beaucoup d'artistes ont alors appliqué la technique ancestrale de la répétition à leurs compositions : ce concept alors favorisé par ces outils a permis l'émergence de styles et de genres très répétitifs, ancêtres d'une grande partie des genres électroniques actuels. La vision du son a aussi changé : sous DAW, on travaille plus sur les textures que sur les ondes physiques comme sur un *tracker*.

De nouveaux outils complètement différents sont nés depuis, inspirés par le MML, ses dérivés et les langages de programmation existants. C'est le cas notamment de Sonic Pi, un éditeur pour un langage fait pour le *live coding* avec des boucles asynchrones, ou encore Faust qui ressemble à du C++. Tous ces outils contribuent à notre diversité culturelle, ce qui est indispensable à la qualité de l'art international et donc à notre bien-être sociétal[10]. (la sérendipité permise par nos outils de recherche a aussi une grande influence sur cette diversité ![11])

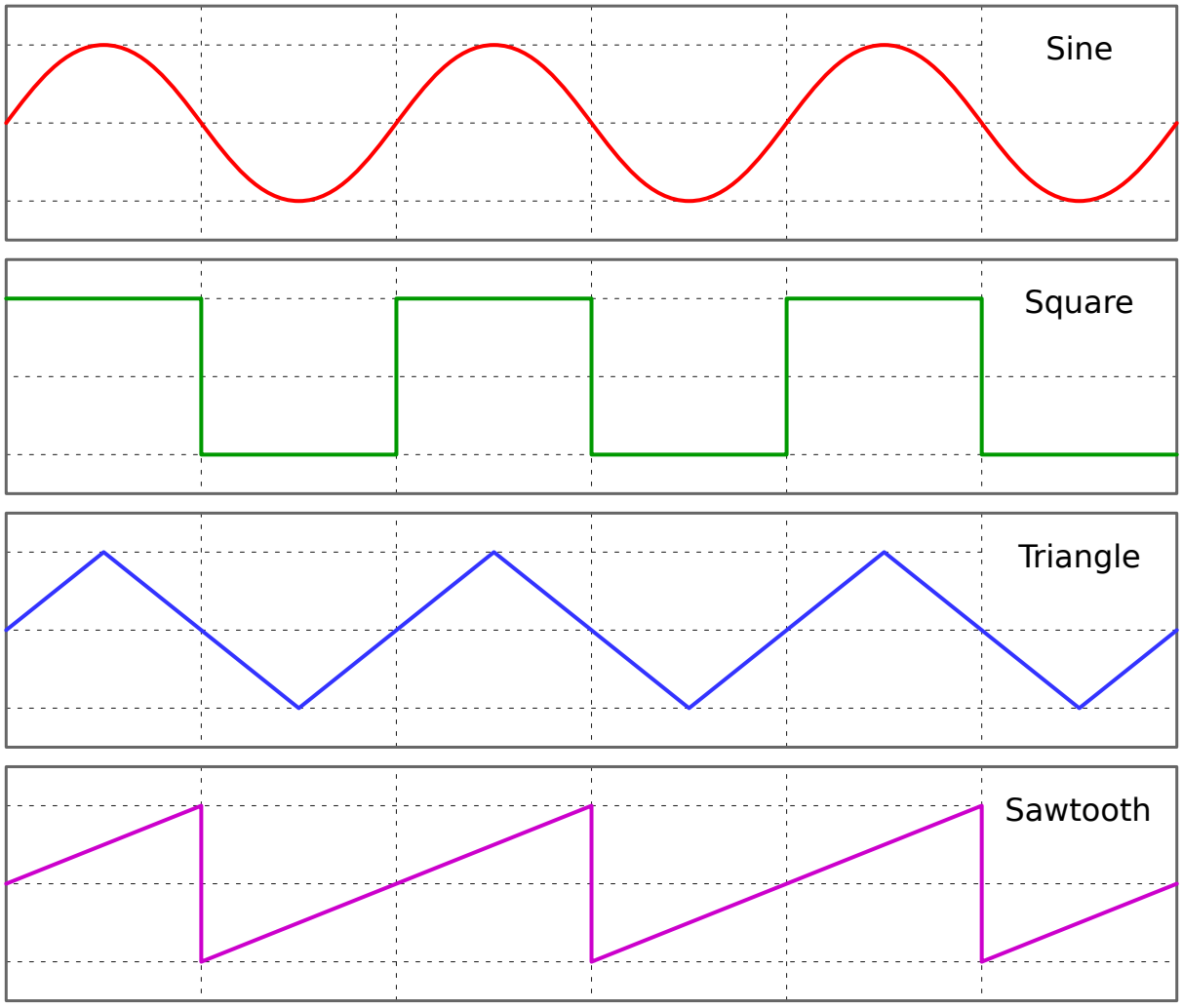
8 https://milkytracker.org/docs/manual/MilkyTracker.html
9 https://en.wikipedia.org/wiki/MIDI
10 https://sociologyinc.com/why-cultural-diversity-matters-more-than-ever-a-sociological-deep-dive/
11 https://ceur-ws.org/Vol-3448/paper-03.pdf

RECYCLER LE MML POUR LIBÉRER L'ARTISTE

Voilà donc le but du projet : moderniser et évoluer le MML vers un langage plus permissif sans perdre ses qualités. Il s'agit en fait d'une série de projets : j'ai commencé par partir d'un logiciel existant, BeepComp, et j'ai écrit des outils pour l'étendre, avant de me mettre à des prototypes de langages, LOV et enfin Bleeperpreter, mon premier pseudo-MML vraiment achevé. Bleeperpreter a permis de moderniser le variant MML de BeepComp mais il restait des idées à mettre en place, et c'est là qu'est né BLIP, inspiré de Bleeperpreter mais réécrit complètement.

LES MATHS SONT VOTRE SEULE LIMITE

La première amélioration se fut dans le choix d'instrument. En effet, depuis les débuts du MML, il est commun de proposer des instruments prédéfinis. Seulement, j'ai remarqué que souvent ces instruments sont très simples, reflétant la primalité des instruments disponibles dans les années 70 à 80 : sinusoïdes, ondes carrées, triangulaires ou à dents de scie.



Par Omegatron — Travail personnel, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=343520

La visualisation en domaine temporel de ces signaux témoigne de la simplicité de leur expression mathématique. Il serait presque possible d'écrire directement l'expression du signal dans le script MML ou en argument d'entrée à l'interpréteur / compilateur[12]. Heureusement, on n'est plus en 1978 et nos ordinateurs sont largement assez puissants pour interpréter une expression à la volée et l'utiliser efficacement pour générer énormément d'échantillons et en faire du son. J'utilise pour cela une librairie Rust spécialisée dans la performance, *fasteval*. J'ai accès à de nombreuses fonctions et opérateurs mathématiques ainsi qu'aux constantes de base (π et le nombre d'Euler au moment d'écrire ce paragraphe). Bleeperpreter accepte donc un signal en entrée et utilise sa variable « f » et « t » (si elles sont disponibles) pour les remplacer par la fréquence de la note jouée et le temps en seconde depuis le début de la note (respectivement).

> Petite anecdote : pendant que je testais le logiciel, j'ai voulu recréer un vibrato sur une sinusoïde.
>
> J'ai donc pensé qu'il suffirait de faire varier la variable de fréquence de la note (« f ») elle-même comme une sinusoïde. J'ai donc glissé une fonction « sin » dans une autre et le résultat est le suivant : la fréquence variait mais tendait à chaque itération vers une variation plus ample. J'ai enregistré le résultat intrigant mais pas inintéressant (filtré pour ne pas être inaudible) sur ma page BandCamp ; on peut y remarquer un effet stroboscopique en lien avec le théorème de Nyquist-Shannon.

PLUS DE NOTES

La seconde amélioration qu'amène Bleeperpreter, c'est une gamme de taille d'octave dynamique. Pour cela, j'ai trouvé une formule en ligne permettant de calculer la fréquence d'une note par rapport à la première note d'une octave et le nombre de notes dans l'octave. Plus tard, je me suis aperçu que la gamme « sonnait » différemment d'une gamme « normale ». Je m'étais en fait construit un système de tempérament égal, aussi appelé tempérament à division égale de l'octave, différent du tempérament Pythagorien qui était utilisé souvent pour accorder les pianos (je me suis emmêlé les pinceaux en faisant la vidéo). Comme ce calcul prend en compte le nombre de notes dans une octave, il est possible de faire varier ce nombre de notes tout en respectant les règles établies par le tempérament (une note aura le double de la fréquence de la même note une octave en dessous).

Dans Bleeperpreter, il s'agit encore d'un paramètre passé à l'interpréteur / compilateur au moment de l'appeler en terminal. On lui donne le nom des notes qu'on compte utiliser et il les reconnaît dans le script MML avant de calculer leur fréquence par rapport à leur nombre et à l'octave correspondante.

La possibilité d'intégrer des noms de notes variables a été rendue possible en grande partie par la librairie nom, un combinateur de parseurs.

PLUS DE TEMPÉRAMENTS

Bleeperpreter se restreint donc au tempérament égal pour calculer la fréquence des notes. Mais cette expression mathématique peut aussi être donnée au runtime, comme celle de génération de son ! Dans BLIP, une expression permettant de passer d'un index de note à une fréquence est donnée en paramètre de l'interface en ligne de commande. Par contre, on a toujours les commandes MML classiques qu'on ne peut pas modifier : « o » pour l'octave, « l » pour la longueur des notes… Et si le tempérament choisi ne

12 Je considère que mes projets sont des interpréteurs puisqu'ils peuvent lire le MML directement jusqu'aux enceintes et aussi des compilateurs quand ils transforment le script en échantillons sonores (puis encodage MP3 etc).

dépendait pas des octaves mais d'autres concepts différents ? Il faudrait alors modifier le langage.

UN PEU DE VARIATION VOYONS !

Quasiment tous les langages de programmation permettent de donner des noms arbitraires à des tas d'objets par la définition de variables. Mais regardons d'un autre angle le MML parce qu'il en contient aussi !

Au final, les « commandes » « O », « L » etc sont des changements de variables. La variable O participe à la hauteur de la note tandis que la variable L à la longueur de la note. J'ai donc rendu ces variables optionnelles ; elles sont rajoutées au langage sur demande en paramètre de la CLI de BLIP. Elles peuvent être ensuite utilisées dans l'expression du tempérament. Et, au final, comme le tempérament permet de résoudre la variable « f » de l'expression de l'instrument, j'ai intégré l'expression de tempérament dans celle de l'instrument comme une seule expression. Afin de pouvoir s'aider du nombre total de notes, comme il est nécessaire pour le EDO, BLIP donne à disposition la variable N.

Il reste un problème. La variable L / l et T pour le tempo n'existent plus non plus. La durée des notes doit également être fixée par l'utilisateur.

MAÎTRE DU TEMPS

Pas trop de suspens là-dessus, c'est juste une autre expression mathématique qui calcule cette fois la durée d'une note en secondes par rapport à toutes les variables à disposition (par défaut + celles ajoutées par l'utilisateur), donnée en paramètres de BLIP.

Il reste encore un problème. En testant Bleeperpreter, je me suis souvenu à quel point, sous BeepComp, les pentes me manquaient : crescendos, accelerandos… Impossible de les représenter en MML à par par saccades, en modifiant les variables progressivement.

DES MATHS PARTOUT…

J'ai trouvé un moyen : comme il existait dans la syntaxe de BeepComp que j'utilisais des boucles et des tuples, j'ai rajouté un nouvel élément capable de contenir une sous-partition et d'y appliquer des effets : la pente. On lui donne en argument de la CLI un nom reconnu en MML puis une variable à modifier ainsi qu'une expression dont le résultat sera assigné à la variable cible à chaque frame (donc à chaque échantillon calculé). On peut ainsi faire un crescendo en formant une pente modifiant une variable impliquée dans l'amplitude du signal de l'instrument (qui sera ensuite traduit en volume), comme « a » dans « a * sin(2*pi()*(442*2^((n+1)/N))*t) », l'incrémentant à chaque frame de manière linéaire ou de n'importe quelle autre manière (exponentielle par exemple), et appliquer cette pente sur la partie du MML que l'ont veut en l'entourant d'accolades avec le nom de la pente.

RÉSULTAT SATISFAISANT (ET FONCTIONNEL)

Le résultat est publié sur GitDab sous ce lien : https://gitdab.com/breval/blip. La documentation est sur le site officiel des documentations des logiciels écrits en Rust (c'est juste mon langage préféré) : https://docs.rs/bliplib. L'interpréteur / compilateur est fonctionnel et peut être installé via cargo (fourni avec rustup) avec la commande « cargo install bliplib --features bin ». Ces deux mois n'ont pas été hyper faciles mais je suis content du résultat. Vous aurez un aperçu de mes difficultés sur mon dev log RSS.

REMERCIEMENTS

Merci à Abdenour Bouzouane pour m’avoir permis de mettre mon projet personnel au profit de mon succès universitaire et pour son conseil en tant que non-musicien.

Je remercie aussi chaleureusement tous les contributeurs du monde de l’informatique musicale qui m’inspirent tous les jours et nous permettent de diversifier notre art et notre culture, et surtout Hiro Morozumi pour tout l’amour qu’il a donné dans BeepComp, le précurseur de BLIP et Bleeperpreter, ainsi que pour m’avoir permis d’échanger sur le sujet.

Enfin, merci à toi, lecteur, pour ton intérêt dans mon projet.

J’espère que mes logiciels seront utiles aux musiciens amateurs, professionnels et aux scientifiques qui les découvrent. Bien que ce projet est officiellement terminé, il sera officieusement poursuivi tant que cela sera possible ; il y aura toujours de nouvelles idées à essayer.