PROJET 2A IRIS

Synthèse de signaux en temps réel et programmation évènementielle

Membres de l'équipe : AOUICH Abderrahmane
BOUAYSS Fatima
FILLALI Ayoub
HÉBERT Gaston
HIMMICH Hamza
LMOUDENE Mohamed-Amine

Enseignant encadrant : Mr. DERRIEN Olivier

Table des matières

Dans le cadre de notre formation à SeaTech et en tant que projet de fin de deuxième année du parcours IRIS, nous avons choisi de travailler sur le projet :

Synthèse de signaux en temps réel et programmation évènementielle

Le but est de réaliser un logiciel de synthèse musical fonctionnant en temps réel. Ce logiciel propose à L'utilisateur une interface graphique pour le paramétrage des fonctionnalités et le branchement d'un clavier MIDI permet de jouer les différentes notes.

Aspects historiques de la synthèse FM

Il existe deux grandes catégories de synthèses musicales : la synthèse analogique et la synthèse numérique.

La première est apparue dans les années 1950 avec l'avènement des circuits électroniques. Bien que fonctionnels, ils restent lourds, chers et fragiles, c'est-à-dire qu'ils sont sujets au dérèglement principalement induit par la surchauffe des composants. Les problèmes de la synthèse analogique seront réglés par la synthèse numérique apparue un petit peu plus tard dans les années 1970 dès l'apparition des premiers ordinateurs. Cette méthode a l'avantage d'être particulièrement légère et peu onéreuse en comparaison à la méthode analogique. En effet, les algorithmes de synthèses sont directement implémentés dans des calculateurs comme les processeurs.

La synthèse numérique peut être déclinée en deux sous-méthodes : la synthèse par échantillon et la synthèse par algorithme.

La synthèse par échantillon utilisée dans la plupart des pianos numériques fait appel à une bibliothèque de sons réels préenregistrés et lus à l'action des touches correspondantes. C'est une méthode qui consiste donc à imiter des instruments réels. Du fait de la bonne qualité d'enregistrement en amont, on assure un rendu très réaliste. Cependant, cette synthèse fait également intervenir une partie logicielle permettant de traiter les sons préenregistrés afin de répondre parfaitement au jeu de l'utilisateur. Mais c'est en synthèse par algorithme que la masse de calcul est la plus importante. Avec cette méthode, les sons sont entièrement calculés par des formules mathématiques à partir des numéros de note et de leur fréquence fondamentale. De nos jours, on implémente la synthèse numérique par algorithme dans des composants *hardware* ou dans des VST – *Virtual Studio Technology*. Ce sont des plug-ins intégrables à un logiciel de synthèse musicale permettant d'obtenir une grande variété de sons.

La synthèse FM – en anglais *Frequency Modulation* – est une méthode de synthèse de sons harmoniques par modulation de fréquence, en réalité par modulation de phase. Elle a été découverte par John Chowning, au début des années 1970 par expérimentation sur un synthétiseur analogique de son université. Il remarque que les fréquences aiguës peuvent modifier le timbre de sa forme d'onde de base. Par la suite, le brevet a été racheté par Yamaha qui a utilisé la technique dans son synthétiseur DX7, un modèle qui changera le monde de la musique.

Figure 1 – Yamaha DX7

Pour ce projet, nous avons fait le choix de programmer le logiciel sur MATLAB à la fois car l'environnement de développement intègre déjà une multitude de fonctions dédiées au traitement des signaux et à la création d'interface graphique, mais aussi car nous étions familiarisés avec le langage.

Synthèse analogique	Synthèse numérique	
	Synthèse par échantillons	Synthèse par algorithme Synthèse FM

Schéma récapitulatif de la place de la synthèse FM

Traitement des évènements par l'application

Synthèse de signaux en temps réel

Une multitude d'applications fait appel aux signaux générés en temps réel. En effet, dans certaines situations, on ne peut pas se satisfaire de signaux ou d'informations connus à l'avance, il faut savoir traiter l'information « en direct ». On peut par exemple citer le fonctionnement d'un avion. De nos jours et avec l'essor de l'électronique moderne, on est en mesure de diriger, en temps réel, les ailes de l'appareil à l'aide d'un joystick situé dans le cockpit et relié électroniquement aux calculateurs, eux-mêmes reliés aux servomoteurs. Ou autre exemple plus parlant, les synthétiseurs en musique : le son sortant des amplificateurs résulte d'une interprétation directe des messages en provenance du clavier. De fait, beaucoup d'applications industrielles s'appuient sur la notion de temps réel.

Programmation évènementielle

La programmation évènementielle est une technique d'organisation logicielle. Une boucle principale détecte en continue les évènements en provenance de divers périphériques. Ils sont ensuite traités dans des sous-programmes spécifiques pour produire les résultats souhaités. Le terme programmation nous rapproche du domaine informatique que l'on traite dans ce projet mais qui est traité dans une multitude d'exemples. Ainsi, la réalisation d'une Interface Homme-Machine (IHM) comme la plupart des logiciels est basée sur la programmation

évènementielle. Dans notre application, il y a deux grandes catégories d'évènements à traiter : la modification des paramètres dans l'IHM et la réponse en temps réel aux appuis sur les touches. Parmi, les paramètres modifiables, on retrouve le paramétrage des oscillateurs, le paramétrage des algorithmes et l'activation d'effets (voir *Figure 9*).

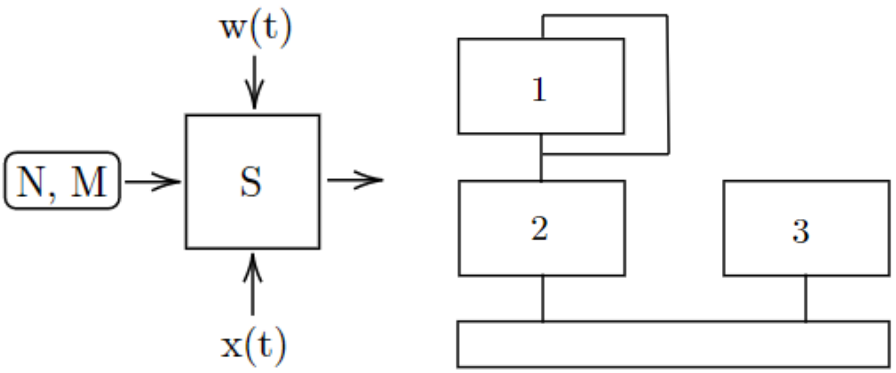
A propos de la synthèse FM

La synthèse FM a proprement parlé peut se définir très simplement par un « vibrato rapide ». Elle implique l'utilisation d'oscillateurs qui se combinent d'une manière particulière pour former ce que l'on appelle des algorithmes. Comme tout est programmé en logiciel, les oscillateurs sont obtenus en sortie de fonctions mathématiques périodiques et ont pour fonction de produire un signal périodique sinusoïdal utilisé pour moduler le signal entrant. Théoriquement, les oscillateurs peuvent être combinés d'une multitude de manières différentes, ce qui assure une variété toute aussi grande d'algorithmes. Le Yamaha DX7, constitué de six oscillateurs, propose trente-deux algorithmes.

La relation entrée-sortie d'un oscillateur peut être écrite sous la forme :

$$x(t) = A.sin\big(2\pi F_P t + M\omega(t)\big)$$

Avec	$\omega(t)$	le signal en entrée de l'oscillateur de fréquence $F_m = Nf_P$
	N	le ratio
	$F_P = Nf_0$	la fréquence de la porteuse assimilée à la fréquence fondamentale
	$A(t)$	l'amplitude
	M	l'indice de modulation
	f_0	la fréquence fondamentale



Figures 2 & 3 – Entrées et sorties d'un oscillateur et agencement d'un algorithme

Les équations correspondantes à cet algorithme sont :

$$x_1(t) = sin\big(2\pi N_1 f_0 + M_1.x_1(t)\big)$$

$$x_{2(t)} = sin\big(2\pi N_2 f_0 + M_2.x_1(t)\big)$$

Enveloppe sonore

L'enveloppe sonore est une courbe décrivant l'évolution de la fonction de modulation de l'amplitude $A(t)$ en fonction du temps. Graphiquement, elle se présente sous la forme de droites qui caractérisent le son et son évolution dans un temps proche. On compte quatre caractéristiques principales :

• L'attaque A décrit la durée nécessaire pour atteindre le niveau maximal après l'appui sur le clavier. Les claviers prenant en compte la vélocité, c'est-à-dire la force d'appui de la touche, voient leur courbe d'attaque modifiée en conséquence.

• La chute D indique la durée nécessaire pour redescendre au niveau d'entretien. On peut comparer le niveau d'entretien à la tension nominale d'un appareil électrique.

• L'entretien S décrit le niveau stable et est conservé tant que la touche est activée.

• L'extinction R indique le temps nécessaire pour que le niveau revienne à zéro dès le relâchement de la touche.

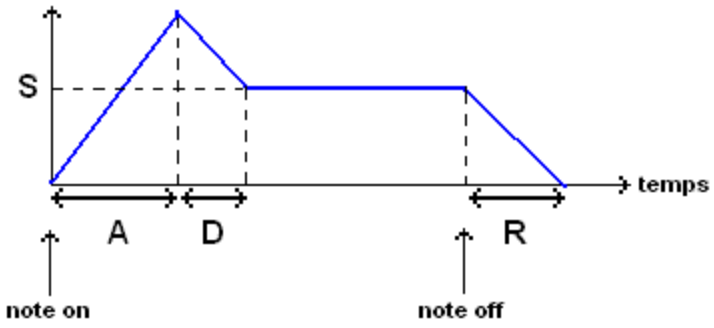


Figure 4 – Enveloppe ADSR

Pour modifier l'enveloppe sonore, on peut déplacer graphiquement les portions de courbes, ce qui est prévu sur notre interface, ou bien on peut spécifier les valeurs limites et les seuils de chaque zone. Une modification graphique de la courbe implique une modification effective et audible de l'oscillateur en question.

Cependant, il est nécessaire de prêter une attention particulière à l'enveloppe sonore. D'une part afin d'éviter un « clic » à l'action d'une touche. C'est un phénomène qui peut survenir si l'attaque est brutale et que l'intensité sonore passe subitement de zéro à une valeur plus élevée. D'autre part pour permettre au spectre d'évoluer dans le temps.

Flux d'information vers l'application

L'application reçoit des informations à la fois du clavier mais aussi depuis l'interfaces pour ce qui est des réglages. Toutes ces données sont des paramètres d'entrée pour les algorithmes de calcul réalisant une synthèse FM pour produire le signal qui sera envoyé aux haut-parleurs.

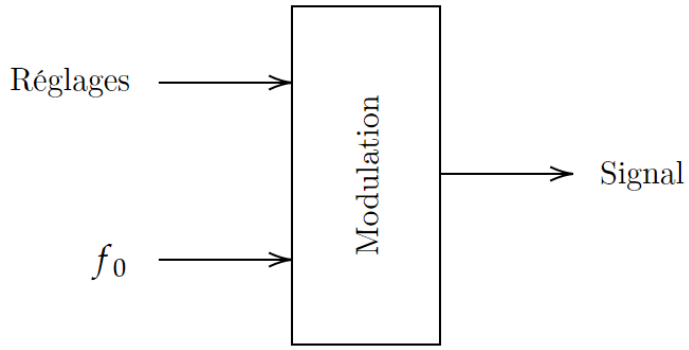


Figure 5 – Flux d'information

La récupération des données du clavier se fait par liaison USB selon le protocole de communication MIDI – *Musical Instrument Digital Interface* – introduite en 1991. Lors de l'activation d'une touche du clavier, on reçoit une information comprenant le statut d'activation de la note ainsi que son numéro. Lors du relâchement, les informations sont identiques sauf le statut passant en mode désactivé. Toutes ces informations permettent de connaître en temps réel les notes jouées ou non. Étant donné que l'on ne reçoit que le numéro de la touche, nous travaillons avec un registre associant les numéros à une fréquence calculée par une formule.

$$f = 440.2^{\left(\frac{n-69}{12}\right)}$$

Formule liant le numéro de la touche à la fréquence en Hz

où *f* représente la fréquence de la note *n*.

Le choix des réglages est ce qui a longtemps posé le plus de problème. Par exemple, sur le Yamaha DX7 possédant une interface très simple, la programmation des algorithmes était très rudimentaire, si bien que les utilisateurs utilisaient des modules préréglés au lieu de modifier directement les paramètres des oscillateurs. C'est ce qui explique aussi le faible nombre d'algorithmes nativement présents : il n'était pas possible de proposer plus de choix sur le seul petit écran de l'instrument. C'est là que réside tout l'intérêt du logiciel construit lors de ce projet.

Il permet de régler différents paramètres utilisés dans les algorithmes de calcul. Cependant, nous avons tenu à ce que le réglage depuis l'IHM soit également en temps réel, c'est-à-dire que l'utilisateur à la possibilité de modifier « en cours de route » les paramètres complets des

oscillateurs. C'est tout le cœur de la programmation évènementielle : répondre en temps réel aux évènements.

Cahier des charges du projet

Le sujet que nous avons choisi propose quelques étapes à suivre auxquelles nous avons choisi d'ajouter des directions supplémentaires. En particulier :

- La synthèse de sinusoïdes pures en monophonique
- La synthèse de sinusoïdes en polyphonique.
- L'implémentation de la synthèse FM à deux oscillateurs couplés, que nous avons choisi d'étendre directement aux six oscillateurs et aux 32 algorithmes initiaux comme prévu sur le Yamaha DX7.
- La généralisation à des oscillateurs non-sinusoïdaux tels des créneaux, des dents de scie.
- L'ajout de courbes de modulation de l'enveloppe temporelle (ADSR).
- L'affichage de la forme d'onde et du spectre du signal modulé en temps réel.
- L'ajout d'une fenêtre affichant le clavier MIDI et mettant en évidence les touches activées.
- L'ajout d'un oscillateur à basse fréquence ou LFO – Low Frequency Oscillator – un générateur d'oscillation utilisé en musique pour commander des modulations lentes.
- L'ajout de plusieurs effets additionnels paramétrables par l'utilisateur.

Déroulement du projet

Organisation du projet et répartition des tâches

La gestion de projet est une partie indispensable pour mener à le projet à bien le plus efficacement possible.

Afin de garantir le bon déroulement du projet, il nous a fallu déterminer les différentes tâches et les répartir équitablement entre nous. Pour cela, nous avons premièrement pris connaissance du sujet, établi les fonctions et la structure de programmation nécessaires pour remplir les besoins, effectué des recherches sur les technologies que nous pouvions utiliser et déterminé les tâches à accomplir.

Nous avons travaillé sur quelques tâches de manière collective comme d'autres ont été effectuées en autonomie, en expliquant les différentes modifications. Ainsi, pour faire le point sur nos tâches effectuées et contribuer efficacement à l'amélioration du projet, nous discutons de l'état d'avancement et des contraintes trouvées lors des séances allouées au projet avec Mr. DERRIEN qui nous a confirmé à chaque fois le travail réalisé et les solutions proposées. Notre but était d'avancer le maximum ce qui pouvait l'être afin d'ajouter de nouvelles fonctionnalités.

Echéancier de réalisation

La réalisation de notre projet est passée par 4 étapes : l'élaboration, la planification, l'exécution et la vérification.

Elaboration

Première lecture du cahier des charges et identification de l'idée principale qui consiste à réaliser une *interface graphique de synthèse FM commandée par un clavier MIDI* et analyse du contexte et des contraintes liées au projet.

Planification

Lors de cette phase, nous avons mis en place une structure pour la réalisation du projet. Elle formalise la décision de répartition des tâches, les fonctions et classes à implémenter, les choix de programmation et d'organisation du code, la partie concernant l'interface graphique et la partie matériel notamment la connexion du clavier MIDI.

Exécution

Il s'agit de la phase opérationnelle. Elle permet de développer une solution pour l'accomplissement du projet. Cette phase commence par le développement des différentes fonctions, leur implémentation dans l'interface ainsi la connexion au clavier.

Vérification

Cette phase comprend l'évaluation globale du projet, la relecture du code, les différents tests pour vérifier la performance de chaque fonctionnalité, la correction des erreurs et la réalisation du rapport en y évoquant les tâches réalisées, le résultat final, les problèmes rencontrés et les points à améliorer.

	Ayoub	Amine	Gaston	Fatima	Hamza	Abderrahmane
Tâche 1	■	■				
Tâche 2		■			■	
Tâche 3	■		■			■
Tâche 4	■	■	■	■	■	■
Tâche 5	■					
Tâche 6		■		■		
Tâche 7	■	■		■	■	
Tâche 8	■	■	■	■	■	■
Tâche 9			■	■		■
Tâche 10	■	■				

Tableau 1 – Répartition des tâches du projet

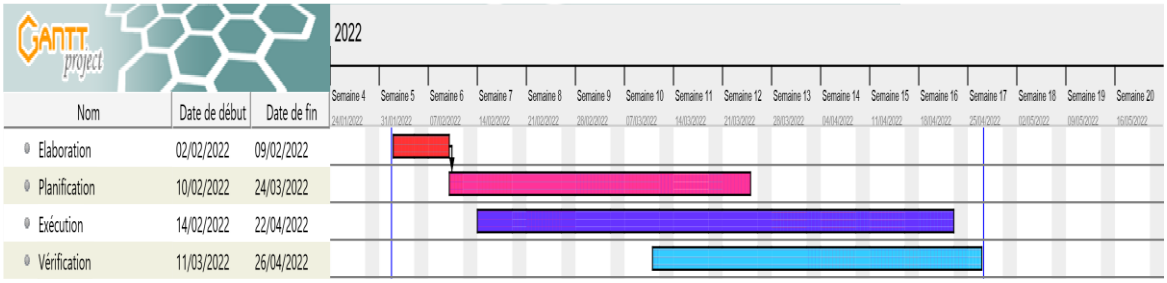


Tableau 2 – Diagramme de GANTT

Matériel utilisé

Le logiciel intègre nécessairement une fonctionnalité de branchement d'un clavier MIDI pour permettre le jeu au clavier en temps réel. Avec le clavier utilisé lors du développement, un *KORG nanoKEY2*, il est nécessaire d'installer un driver propre à ce clavier pour permettre de récupérer les messages *via* USB.

C'est un clavier de 25 touches couvrant deux octaves que l'on peut décaler *via* les boutons de gauche. La récupération des messages est automatisée et la fonction `midireceive` de MATLAB permet d'avoir une information sur les messages reçus à chaque action d'une combinaison de touches. Nous avons fait le choix de paramétrer le nombre de messages simultanés à 16, bien qu'il soit théoriquement possible d'accepter autant de messages que voulu. Les informations transmises sont le numéro de la note jouée et sa vélocité, c'est-à-dire la vitesse d'action de la touche.

Choix de programmation

L'ensemble des programmes ont été réalisés sous MATLAB et le projet est scindé en différents fichiers réalisant chacun une fonction du logiciel final. En particulier, nous avons programmé :

- Quatre classes
  - Oscillator
  - Algorithm
  - Modulation
  - Paramètres généraux de l'interface
- Une fonction
  - Réinitialisation de la liste des périphériques connectés
  - Génération du spectre en temps réel
- Trois fichiers de configuration d'interface pour *App Designer*
  - Le logiciel principal

- La fenêtre de modification de l'enveloppe ADSR
- La fenêtre de modification et d'édition d'algorithmes

La classe `DX7Modulation` est la classe mère qui construit le logiciel à chaque lancement en fonction des paramètres d'enveloppe et d'oscillateurs en mémoire ou enregistrés précédemment.

Synthèse des oscillateurs

Comme le nombre d'oscillateur est fini et que chaque oscillateur possède une structure similaire en termes de fréquence ou pour d'autres valeurs, nous avons choisi de construire une classe `oscillator` grâce à laquelle sont instanciés les six oscillateurs utilisés dans le logiciel.

Chaque oscillateur est repéré grâce aux attributs suivants :

- Identifiant
- Indicateur de marche de l'oscillateur
- Valeur de l'amplitude
- Forme d'onde utilisée (sinus, triangle, créneau)
- Offset de fréquence pour la transposition
- Amplitude
- Valeur du gain
- Valeur de l'enveloppe sonore
- Indicateur de marche de l'enveloppe sonore

En particulier, l'indicateur de marche de l'oscillateur indique si l'oscillateur est activé ou non. En clair, son expression n'apparaitra pas dans l'équation de modulation.

```
1.  properties (Access = public)
2.      id;
3.      switchOsc;
4.      envSwitch;
5.      ratio;
6.      offsetHz;
7.      waveForme;
8.      ADSR_;
9.      ADSR_p;
10.     level;
11.     amplitude;
12. end
```

Tous les attributs cités plus haut servent de paramètres pour l'instanciation des six oscillateurs, par exemple l'oscillateur n°1 présenté ci-dessous.

```
1.  oscillator1 = Oscillator(1,'On','On',1,0,"Sin",100,1,[0.35,0.05,0.4,0.2],0);
```

Par suite, ces informations sont récupérées dans la classe `DX7modulation` pour tous les oscillateurs si ils sont allumés afin d'effectuer le calcul de modulation en temps réel en suivant la combinaison d'oscillateurs : l'algorithme.

Combinaison des oscillateurs en algorithmes

Pour ce qui est de la sauvegarde des différents algorithmes, qu'ils fassent partis des 32 algorithmes originaux du DX7 de Yamaha ou qu'ils aient été créés par l'utilisateur, nous enregistrons la configuration au format hexadécimal en prenant pour base de codage la grille suivante :

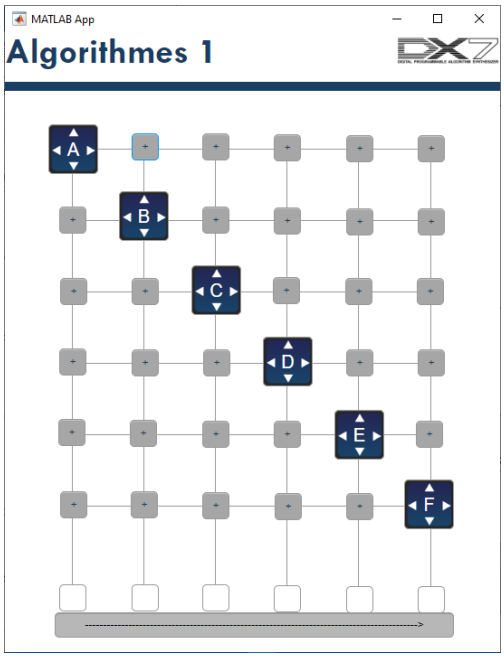


Figure 5 – Fenêtre de sélection et d'agencement des oscillateurs

Ce schéma permet de visualiser graphiquement les liaisons entre les oscillateurs. Une connexion entre deux oscillateurs indique que la sortie du premier est utilisée comme modulateur du second. L'opération est réitérée pour la connexion suivante avec un autre oscillateur. Un oscillateur en boucle sur lui-même indique qu'il est modulateur de sa propre sortie. Toutes les connexions arrivant sur la sortie – en bas – représentent le signal sortant de l'algorithme.

La configuration de chacun des six oscillateurs est enregistrée sur deux caractères hexadécimaux soit 8 caractères binaires à la valeur 0 ou 1. Pour le codage, il faut respecter la nomenclature suivante.

Premier caractère hexadécimal	Premier bit	Modulation de sa propre sortie
	Second bit	Oscillateur utilisée comme entrée pour l'oscillateur suivant
	Troisième bit	Oscillateur utilisée comme entrée pour l'oscillateur suivant
	Quatrième bit	Oscillateur utilisée comme entrée pour l'oscillateur suivant

Second caractère hexadécimal	Premier bit	Oscillateur utilisée comme entrée pour l'oscillateur suivant
	Second bit	Oscillateur utilisée comme entrée pour l'oscillateur suivant
	Troisième bit	Modulation avec la sortie
	Quatrième bit	Oscillateur éteint

Tableau 1 – Nomenclature de la notation hexadécimale pour l'enregistrement des algorithmes

Par exemple, la configuration correspondant à l'oscillateur n°1 modulant lui-même et la sortie sera écrit 82 = 10000010. Il faut faire de même pour obtenir la notation hexadécimale des 32 autres algorithmes originaux du DX7 bien qu'il soit possible d'enregistrer autant de configurations que souhaitées. Pour enregistrer la configuration courante, nous exportons la séquence des 12 caractères hexadécimaux caractéristique de l'algorithme créé ou modifié. Pour le chargement d'une configuration, nous décodons la valeur hexadécimale en binaire pour ouvrir l'algorithme choisi.

Modulation en fréquence du décodage des messages MIDI

La modulation des messages décodés est une des fonctionnalités les plus importantes du projet. Elle consiste à moduler les signaux entrant par les d'oscillateurs avant de les envoyer vers la carte son de l'ordinateur.

Comme la modulation est une imbrication de fonctions périodiques – sinus, cosinus, créneau, triangle – nous calculons successivement les échantillons pour chaque oscillateur, puis nous renvoyons en sortie le signal modulé. À chaque opération, nous récupérons la valeur de l'indice de modulation depuis l'interface utilisateur.

Pour l'agencement des fonctions dans le calcul final, nous nous référons à l'algorithme sélectionné et donc aux relations entre les oscillateurs. Le calcul reprend enfin la forme de l'équation de relation entrée-sortie des oscillateurs.

Au niveau de la programmation, nous utilisons la fonctionnalité `eval` permettant d'évaluer du code MATLAB passé en paramètre comme des caractères, ceci afin de pouvoir traiter tous les oscillateurs impliqués dans la modulation en s'affranchissant d'une boucle à indice. En effet, nous avons fait le choix de nommer les oscillateurs de la manière suivante :

oscillator***i***

où ***i*** est le numéro de l'oscillateur. Pour le traitement au sein d'une boucle afin de traiter chaque oscillateur séparément avant leur intégration dans la formule de modulation finale, le travail en chaîne de caractères – *string* – est la seule solution possible.

Modification de l'enveloppe ADSR

Dans la première version du logiciel, les modifications de l'enveloppe ADSR influent directement sur la sortie, c'est-à-dire sur le signal complétement modulé.

Dans la dernière version, nous prévoyons de pouvoir modifier l'enveloppe ADSR pour chacun des six oscillateurs, ce qui offre la possibilité de produire une diversité de sons encore plus grande.

La modification des quatre paramètres A, D, S et R s'effectue sur une fenêtre particulière avec des curseurs. Le début de l'attaque ainsi que la fin du relâchement sont fixes – points L4. En revanche, les autres pans de droite correspondant à la durée de l'attaque, de la chute, du relâchement et à la longueur du maintien sont personnalisables – sections R1, R2 et R3.

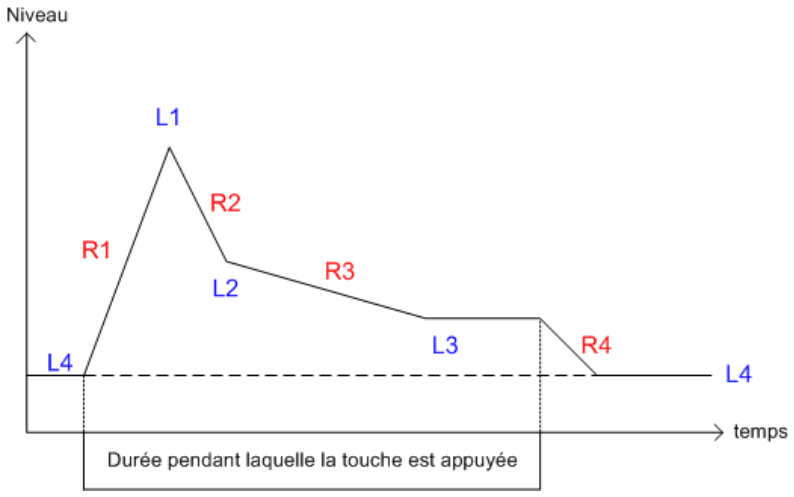


Figure 6 – Enveloppe ADSR et position des pics

Les paramètres de l'enveloppe sont directement enregistrés dans la classe principale du fichier `dx7Main`, ensuite récupérés dans la classe `DX7Modulation` pour le calcul.

Buffer de sortie vers la carte son

Le buffer est une mémoire tampon permettant le stockage temporaire des échantillons avant qu'ils soient envoyés aux haut-parleurs. Cette technique permet d'éviter des coupures car le traitement, bien que très rapide, ne se fait pas de manière instantanée.

Pour cela, le buffer de la carte son possède un buffer pouvant accueillir 128 échantillons. Une fois vide, elle est alimentée par 128 nouveaux échantillons venant d'être traités et ainsi de suite. Cependant, la lecture par la carte son est plus rapide que le traitement des messages MIDI. Dans ce cas, nous avons imaginé un second buffer de taille 512 venant s'interfacer entre le traitement des messages MIDI et la carte son. À la manière du buffer de la carte son, il est alimenté en échantillons dès qu'ils ont été traités.

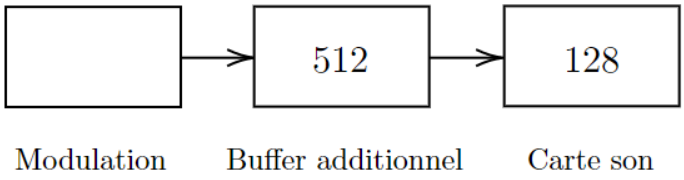


Figure 7 – Chaîne de traitement des échantillons

Dans notre cas, la carte son récupère les échantillons dans le buffer additionnel au lieu de les récupérer directement après le calcul, ce qui aurait pour effet de produire un son saccadé car le calcul est plus rapide que le traitement par la carte son. Il n'y alors aucune coupure lors de la lecture des échantillons.

La taille choisie pour le buffer dépend de nombreux paramètres matériels, notamment de la latence. En effet, un buffer trop grand évite les coupures induites par le ralentissement du programme, mais augmente nécessairement la latence.

Fichiers `.mlapp`

Sous MATLAB, les fichiers `mlapp` sont les fichiers de configuration des interfaces graphiques construites sous App Designer. Ce fichier intègre le code de la configuration et de l'emplacement des différents éléments de l'interface – boutons, axes, menus… – et il accueille les codes de calcul pour les fonctions principales du logiciel, à savoir la modulation en fréquence du décodage des messages MIDI et l'application d'effets additionnels. Techniquement, le projet comporte autant de fichiers `mlapp` que de fenêtres graphiques. Il y a donc deux fichiers similaires en structure pour l'interface de modification de l'enveloppe et pour la configuration des algorithmes. De plus, chacun de ces fichiers fait appel à des fonctions MATLAB « classiques » – fichier `.m` – pour effectuer les différents calculs.

Fonctionnalités additionnelles

En plus des fonctionnalités demandées par le cahier des charges du projet, nous avons fait le choix d'ajouter au logiciel quelques fonctionnalités supplémentaires communément retrouvées sur les synthétiseurs d'époque, d'une part afin de nous permettre d'élargir nos connaissances sur le sujet, et d'autre part pour améliorer l'expérience utilisateur.

Effet flanger

Le *flanger* est un effet obtenu en ajoutant au signal ce même signal dont le retard d 'une durée courte et variable a été modulé par un LFO.

Effet de vitesse (accélération et ralentissement)

L'effet de vitesse permet de modifier la vitesse de lecture des échantillons de sortie. Pour cela, nous utilisons la fonction `stretchAudio` de MATLAB qui permet de modifier l'échelle de temps de l'échantillon par le facteur passé en paramètre.

Filtrage passe-bande

Par définition, le filtrage de type passe-bande ne laisse passer qu'un intervalle de fréquences comprises entre une fréquence de coupure haute et une fréquence de coupure basse. Mathématiquement, ils présentent un gain plus élevé dans cet intervalle, ce qui est effectivement ajouté lors de l'application de cet effet. En outre, le filtrage passe-bande permet d'imiter les sonorités des instruments d'époque.

Effet de compression

L'effet de compression est un effet de dynamique qui permet de contrôler l'amplitude d'un signal, c'est à dire de gérer les différences entre les sons les plus faibles et les sons les plus forts d'une source sonore. Pour cela, le compresseur s'assure de réduire le gain du signal sonore lorsque celui-ci dépasse un certain seuil.

Effet de limiter

L'effet limiter est un type de compression dynamique qui répond très rapidement aux pics de la forme d'onde du signal. Cela consiste à couper le signal à partir d'un seuil en atténuant les pics, on parle alors d'écrêtement ou *clipping*.

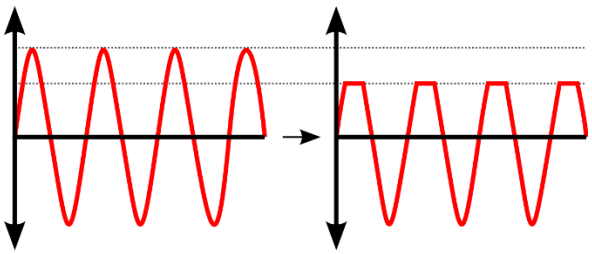


Figure 8 – Principe d'écrêtement du signal

Effet de distorsion

L'effet de distorsion est utilisé pour créer des sons saturés en surdosant le gain, on parle aussi d'*overdrive*. Mathématiquement, il suffit d'augmenter l'amplitude du signal pour obtenir une saturation à une valeur maximale fixée.

Effet d'amplification (*enhancer effect*)

L'amplification consiste à multiplier le signal par un coefficient afin d'en augmenter son intensité sonore.

Panoramique (*Panning effect*)

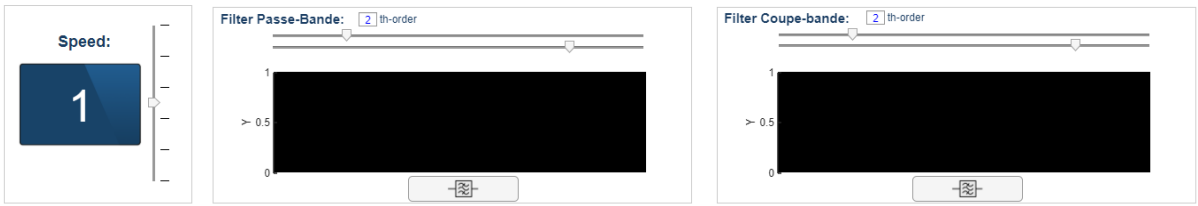
Cet effet se réfère à la distribution d'un échantillon mono sur les channels gauche et droite d'une piste stéréo. Par suite, il est possible de modifier la « position » spatiale du son par un slider.

Effet de modulation en anneau (*Ring Modulator*)

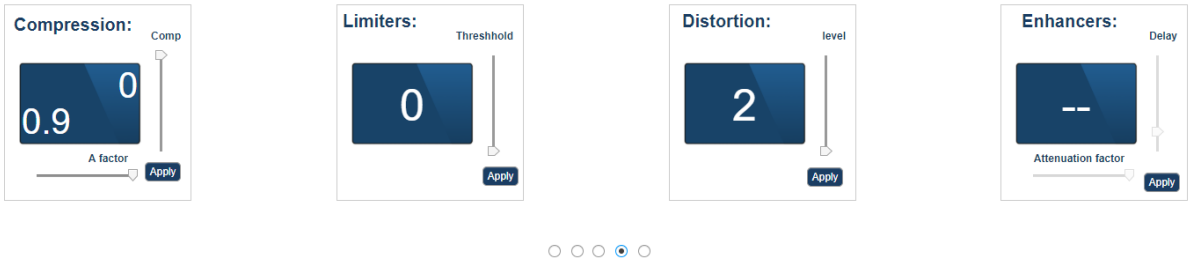
C'est un effet audio de modulation utilisant un oscillateur pour créer une onde sinusoïdale, qui est ensuite multipliée avec le signal de départ (celui d'une guitare par exemple) pour produire de nouvelles harmoniques.

Effets d'écho, flanger, de vibrato et de chorus

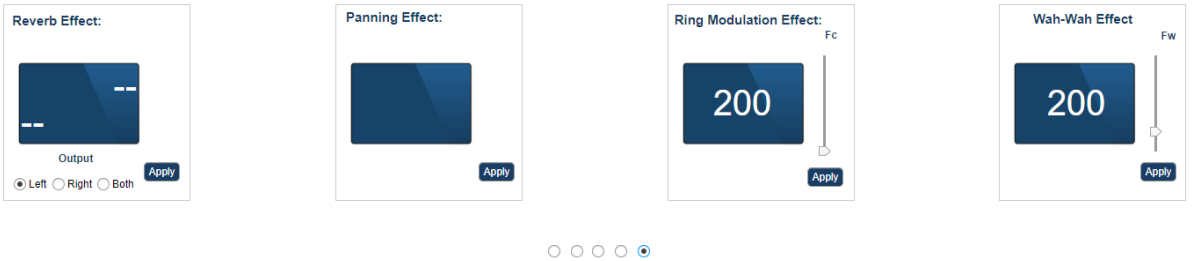
Effets d'application d'un gain, filtres passe-haut et passe-bas



Effets de vitesse, filtrages passe-bande et coupe-bande



Effets de compression, de limite, de distorsion et d'amplification



Effets de réverbération, de panoramique, de modulation en anneau et wha-wha

Interface graphique

Pour l'interface graphique, nous avons utilisé *App Designer*, la dernière fonctionnalité de MATLAB qui permet de réaliser des Interfaces Homme-Machine (IHM) répondant en temps réel aux actions de l'utilisateur. La communication entre les actions de l'interface et les variables du programme est faite par le biais de fonctions de rappel ou callbacks.

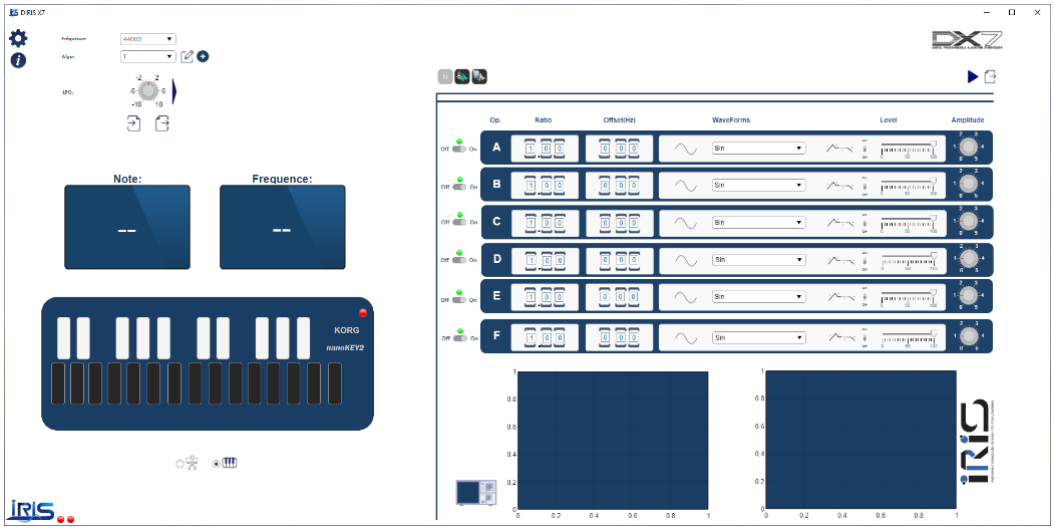


Figure 9 – Fenêtre principale de l'application

Choix de la fréquence d'échantillonnage et des algorithmes

Un petit menu permet de modifier la fréquence d'échantillonnage des échantillons, de choisir l'algorithme voulu, de créer des algorithmes et de les enregistrer, de modifier la fréquence, la forme et la vitesse du LFO ainsi que d'importer et exporter la configuration courante.

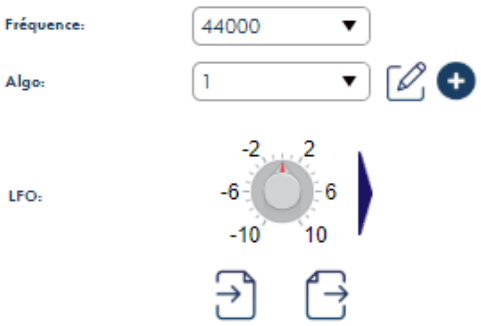


Figure 10 – Choix des paramètres principaux

Modification des paramètres de l'enveloppe

Depuis l'espace de paramétrage des oscillateurs, on peut modifier l'enveloppe ADSR en ajustant chacune des valeurs A, D, S et R avec des slidders. La courbe est modifiée en temps réel et l'enregistrement de la configuration est appliquée instantanément.

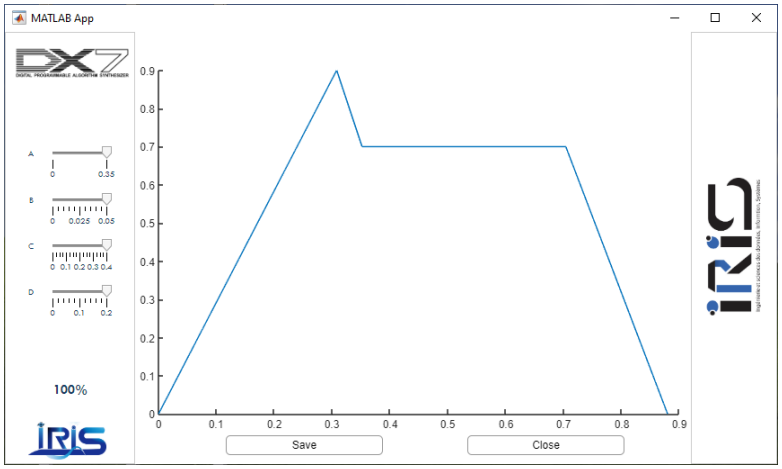


Figure 11 – Fenêtre de modification des paramètres de l'enveloppe ADSR

Édition & création des algorithmes

Pour éditer les algorithmes, il suffit de cliquer sur les boutons + entre deux algorithmes pour les + relier.

Si l'on souhaite qu'un oscillateur soit modulé par sa propre sortie, il faut cliquer dessus, il apparaît alors avec une flèche (voir l'oscillateur *F* ci-contre).

De toute manière, il faut qu'au moins un opérateur soit relié à la sortie – rectangle du bas – pour obtenir un son.

Enfin, le bouton « OK » permet d'enregistrer la configuration ayant été créée ou modifiée.

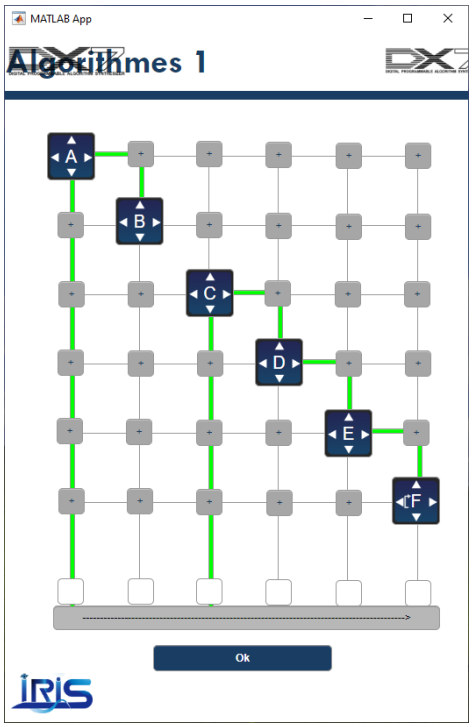


Figure 12 – Fenêtre de configuration des algorithmes

Paramétrage des oscillateurs (modification des valeurs et activation/désactivation)

Cette zone de l'interface permet de paramétrer les oscillateurs dont les informations sont enregistrées dans l'instance correspondante de la classe *Oscillator*.

Pour allumer un oscillateur, on clique sur l'interrupteur à bascule tout à gauche. Par la suite, on peut modifier de gauche à droite :

- Le ratio *M*
- Le décalage
- La forme d'onde
- L'enveloppe ADSR et son activation
- L'amplitude
- Le niveau de l'amplitude entre 0 et 100 %

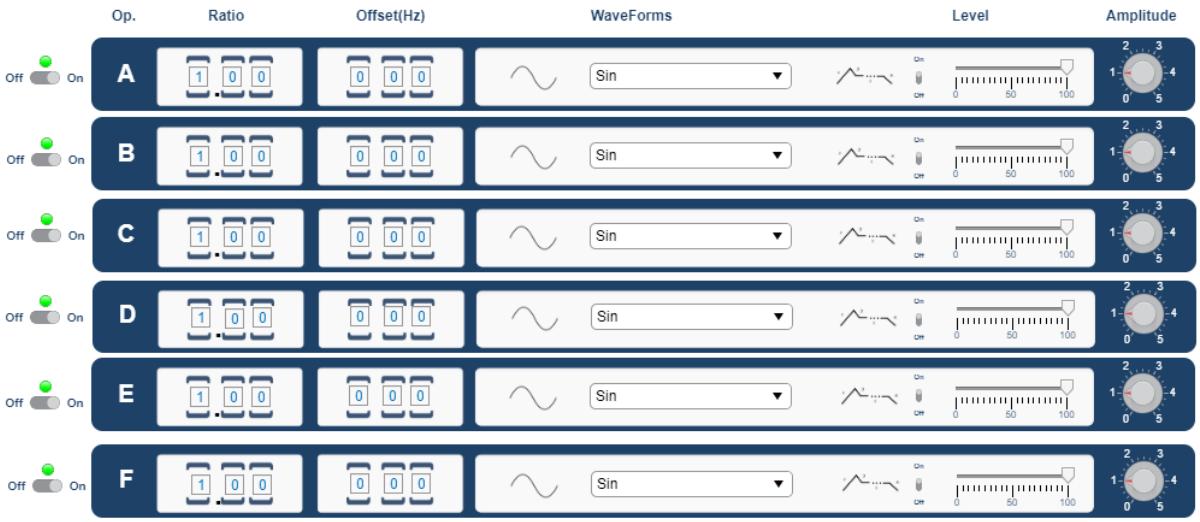


Figure 13 - Zone de paramétrage des oscillateurs

Affichage du spectre et de la forme d'onde du signal sortant en temps réel

Deux axes permettent de suivre en temps réel la forme d'onde ainsi que le spectre du signal de sortie ayant été modulé.

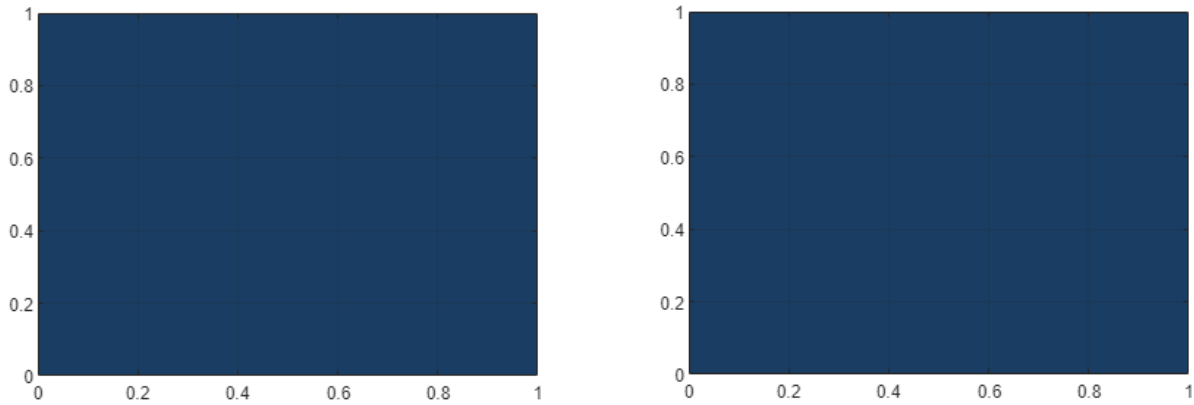


Figure 14 – Fenêtres d'affichage du spectre et de la forme d'onde du signal sortant

Affichage en temps réel des notes jouées sur le piano

Cette zone de l'application ne permet pas de jouer les notes mais uniquement de visualiser les touches jouées. Ce clavier vient en complément

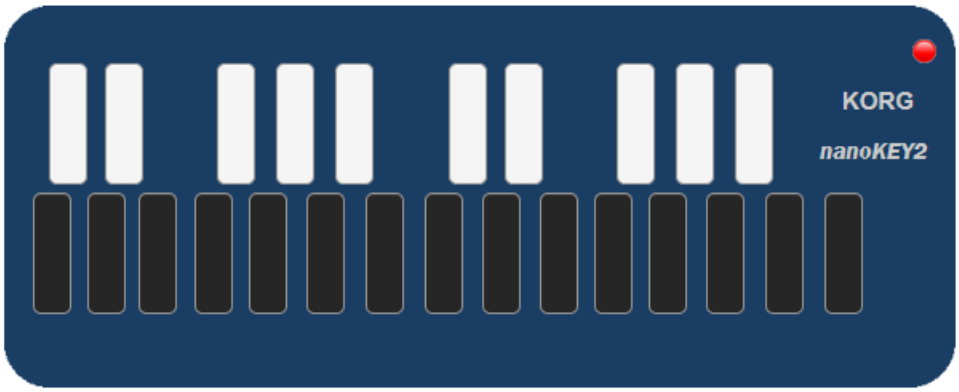


Figure 15 – Clavier virtuel

Fenêtre d'édition, d'application et de modification d'effets

Sur cette fenêtre, l'utilisateur a la possibilité d'appliquer un ou plusieurs effets à un signal préalablement enregistré. Il peut provenir d'une séquence jouée sur le clavier MIDI ou d'un fichier ouvert depuis les fichiers de l'ordinateur.

Sur ce fichier audio, l'utilisateur a la possibilité de sélectionner un échantillon au moyen de slidders. L'échantillon sélectionné apparait en rouge. Pour appliquer des effets à cette sélection, il faut se référer aux cinq volets situés plus bas. On a ensuite la possibilité de lire l'échantillon modifié seul ou bien le signal dans sa globalité.

Pour finir, la séquence modifiée peut être exportée au format wav.

Figure 16 – Interface de modification de fichiers audio

Fenêtre didacticiel lors du premier lancement

Nous avons imaginé une fonctionnalité de didacticiel lors du premier lancement de l'application. Cela consiste en un fenêtrage des zones d'intérêt pour mettre en évidence les différentes fonctionnalités de l'application. Ces fenêtres se déplacent automatiquement

pendant quelques secondes sur les points d'intérêt afin de laisser le temps à l'utilisateur de découvrir l'interface graphique. S'il le souhaite, l'utilisateur peut cliquer sur le bouton « skip » pour passer le tutoriel.

Conclusion

Le projet de deuxième année traitant de la synthèse de signaux en temps réel et de la programmation de l'interface graphique associée sous *App Designer* nous a permis de découvrir certains aspects du traitement des signaux appliqué à la musique. En effet, depuis l'essor de la musique électronique il y a quelques décennies, la modulation a pris une grande importance dans la synthèse d'une variété de sons toujours grandissante. Le traitement en temps réel implique plusieurs considérations en comparaison au traitement en mémoire, en particulier le dimensionnement de buffers permettant le stockage temporaire d'information permettant de pallier le problème de la vitesse de calcul, très élevée pour ce genre d'application même sur des machines anciennes.

Le choix de programmation est justifié par la redondance d'information

L'application ainsi créée permet aux utilisateurs de pouvoir concevoir des sons personnalisés à partir d'une combinaison d'oscillateurs et d'effets.

De plus, en vue de l'avancée du projet à quelques semaines de l'échéance, il nous a été proposé d'ajouter une fonctionnalité de réverbération artificielle à l'application. Il s'agit d'une technique de modification du son de sortie ayant pour but de donner un effet de réverbération au signal modifié en donnant l'impression qu'il est joué dans un lieu plus ou moins vaste.

Enfin, comme le driver du clavier *Korg NANO Key 2* n'est disponible que sous Windows, nous avons imaginé une solution pour les utilisateurs de Linux. En connectant un smartphone en local sur une page web disposant d'un clavier réalisé avec CSS, l'application récupère les messages comme s'ils provenaient d'un clavier MIDI et l'utilisateur peut jouer.

Bibliographie & Médiagraphie

- **Documentation officielle de MATLAB**, *mathworks.com*, MATLAB Documentation - MathWorks France
- **Principe de fonctionnement de l'effet *flanger***, *samplecraze.com*, consulté le 23 mars 2022, Flanger Effect - what is it and how does it work - Samplecraze
- **Plug-in de connexion du clavier KORG nanoKEY2**, *korg.com*, consulté le 06 avril 2022, Downloads | nanoKEY2 | KORG (USA)