INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES HAUTS-DE-FRANCE

Université Polytechnique HAUTS-DE-FRANCE

dream AUDIOVISUEL & MULTIMÉDIA ©

Rapport technique projet type 2

Création d'un synthétiseur virtuel

à l'aide du langage Pure data

Raphaël Gosset

2020-2021

3ème année licence audiovisuelle

Table des matières :

I – Résumé :

Codage d'un synthétiseur virtuel semi-modulaire à l'aide du langage Pure Data vanilla.

II – Présentation :

Ce projet est un synthétiseur virtuel « semi-modulaire » créé à l'aide du langage Pure Data vanilla (sans rajout de fonctions absentes du langage de base). Il peut être utilisé à l'aide d'un clavier MIDI ou d'un clavier d'ordinateur.

Ce patch permet de jouer en polyphonie ou en monophonie, et de créer ses propres sons, à l'aide de différentes formes d'onde et de plusieurs effets.

Objectifs à atteindre :
- Avoir un synthétiseur fonctionnel, le plus optimisé possible, quitte à retirer des fonctionnalités pour le rendre plus efficace.
- Explorer au maximum les possibilités qu'offre Pure Data, dans la limite de ce qui est cohérent vis-à-vis du projet.
- N'utiliser que des fonctions de base, ne rajouter aucune extensions ou objets extérieurs pour permettre le partage public du projet et faciliter son utilisation.

Descriptions des chapitres :

III – Introduction au sujet

Introduction brève à Pure Data et au MIDI.

IV – Description du projet

Description du fonctionnement du synthétiseur d'un point de vue utilisateur : Quelles sont les fonctionnalités, à quoi correspond l'interface ?
En quelque sorte le manuel d'utilisation.

V – Description technique et méthodes utilisées

Explication des objets utilisés, des principes mathématiques, des problèmes rencontrés, et des solutions appliquées.

VI – Résultats et perspectives

Exposition des possibles perspectives futures, des fonctionnalités qui aurait pû être ajoutées.

III – Introduction au sujet :

1 – Protocole de communication MIDI

MIDI est l'acronyme de Musical Instrument Digital Interface. Il s'agit du standard le plus répandu pour les protocoles de communication utilisés en musique, bien qu'il soit parfois utilisé en dehors de ce domaine.
De nombreux outils utilisent le MIDI, que ce soit des synthétiseurs, des tables de mixages, ou plus simplement toute la gamme da matériel sous la dénomination « MIDI controller ».

Un « contrôle MIDI » peut correspondre à une touche de clavier, un potentiomètre, un bouton pressoir, etc.

Exemple de clavier MIDI

Exemple de controller MIDI

Pour un clavier MIDI, chaque touche a un identifiant propre, une « note MIDI » qui lui est attribué. Une touche peut être pressée avoir plus ou moins de force, et cette information de force s'appelle « vélocité ». La vélocité est comprise entre 0 et 127.
Donc lorsqu'une touche va être actionnée, le controller MIDI va envoyer un message compotant la note MIDI actionnée ainsi que la vélocité à laquelle elle a été actionnée.
Quand la touche est relâchée, le controller envoie un message avec cet note MIDI ainsi qu'une vélocité égale à 0.

Le fonctionnement diffère un peu pour les autres types de contrôle MIDI, mais utilise néanmoins ces mêmes types de données.

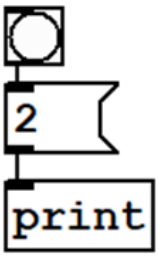
Il y a également d'autre partie au protocole MIDI, mais ils ne sont pas nécessaires pour comprendre le fonctionnement du projet.

2 – Pure data

Dans le langage Pure Data, le terme « programme » est généralement remplacé par le terme « patch », qui est le terme canonique. De même pour le terme « fonction », auquel on préfère l'utilisation du terme « objet ».

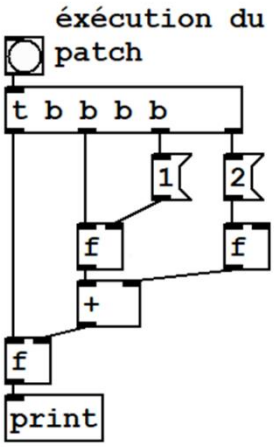
Pure Data est un logiciel de programmation graphique pour la création multimédia et musicale en temps réel (principalement).

Son fonctionnement est un peu différent d'un langage classique en ligne de codes, donc pour donner un aperçu des différences de fonctionnements voici 2 exemples de programmes simples :



Ce patch correspond au programme :

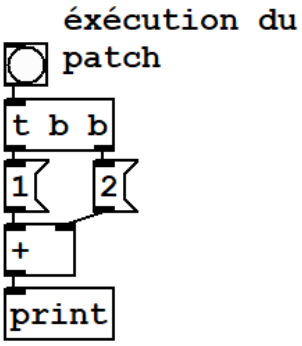
Afficher 2



Ce patch correspond au programme :

A = 2
B = 1
C = B + A
Afficher C

Ce patch est ici mis en forme pour correspondre aux lignes de codes, mais il ressemblerait normalement à ça :



La raison est que dans Pure Data il n'y a pas de variable générale, uniquement des variables locales. L'objet « f » que l'on voit au-dessus garde en mémoire de façon locale la valeur qu'il a reçu, mais cette valeur ne peut pas être appelé par d'autres objets. Il faudrait au contraire connecter cet objet aux autres pour que cette variable locale puisse être utilisée par d'autres parties du patch.

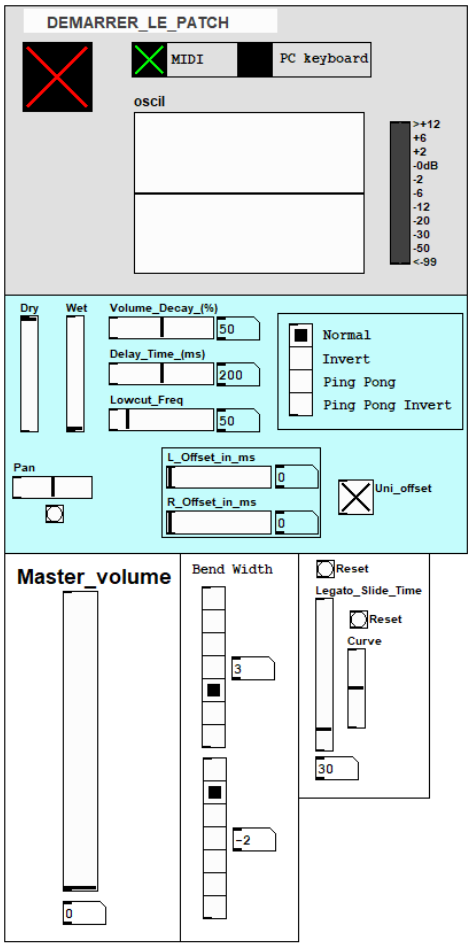
Le fonctionnement diffère beaucoup des autres langages, mais est néanmoins très efficace pour certains usages.

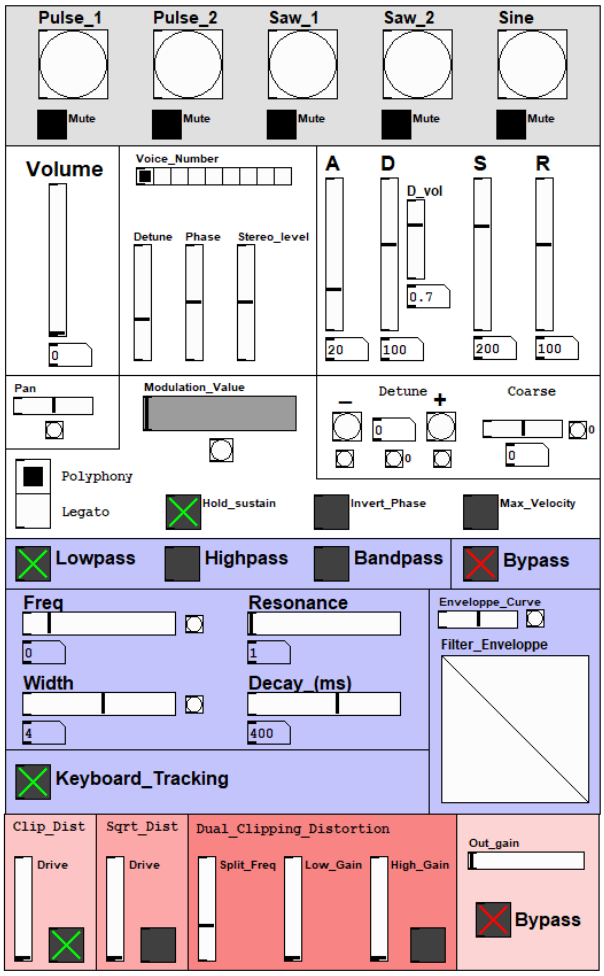
Pure Data possède aussi une panoplie d'objets spécifique à la création sonores.
Les objets nécessaires à la compréhension du projet seront présentés progressivement.

IV – Description du projet :

Cette partie sert de manuel d'utilisation. Elle est consacrée uniquement aux fonctionnalités et à l'utilisation d'un point de vue de l'utilisateur.

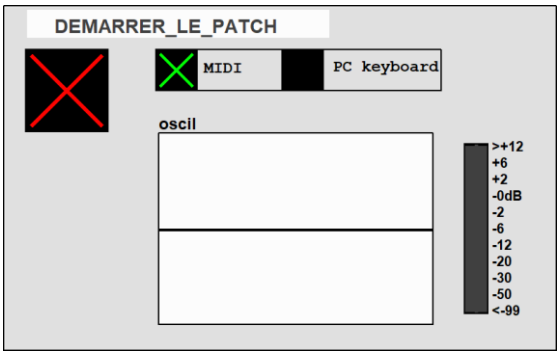
Voici l'interface visuelle avec laquelle commence l'utilisateur :





1 – Interface gauche

A – Patch de démarrage :

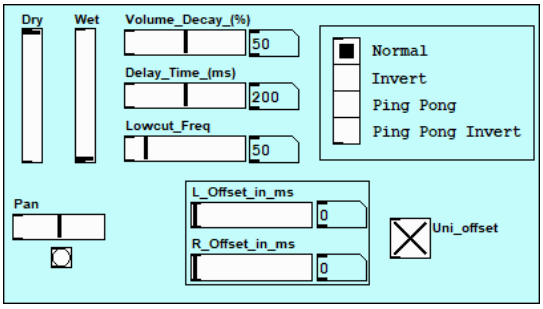


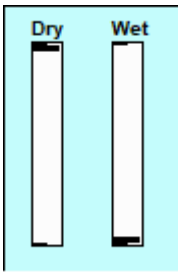
Ce patch comporte un oscilloscope (« oscil ») et un dB-mètre.

<u>Cette croix doit être activée pour permettre à l'ensemble de fonctionner</u>. Elle active le DSP du patch principal (Voir V – 6).

MIDI et **PC Keyboard** indique avec quel outil on joue.

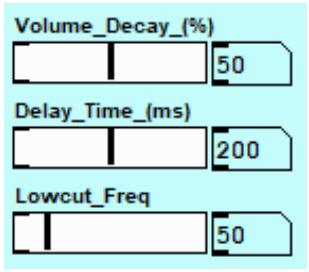
<u>B – Patch de delay</u> :





- **Dry** modifie le volume du signal d'origine qui est envoyé en sortie

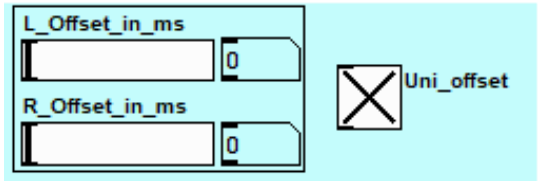
- **Wet** modifie le volume du signal créé envoyé en sortie.



- **Volume Decay** correspond à l'amplitude en % du signal renvoyé dans la boucle de delay.

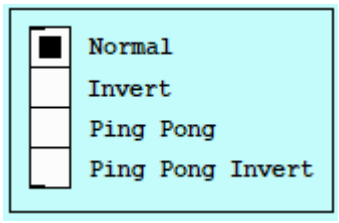
- **Delay Time** est le temps en milliseconde que prend le signal avant de revenir dans la boucle de delay.

- **Lowcut Freq** régle la fréquence de coupure du coupe-bas filtrant le signal en sortie du delay.



- **L_Offset** et **R_Offset** correspondent à un retard (en ms) constant du canal L ou R.

- **Uni_offset** lie les offsets des deux canaux. Modifier un modifie l'autre de la même façon



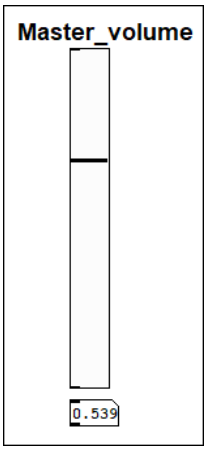
- **Normal** correspond à un delay classique, ou le son de chaque canal est répété.

- **Invert** fonctionne de la même façon, mais les canaux L et R du signal Wet sont inversés

- **Ping Pong** correspond à un delay balancier. Les canaux d'entrée L et R sont sommés, puis delayé à droite, puis à gauche, puis à droite, etc.

- **Ping Pong Invert** fonctionne de la même façon, mais les canaux de sortie du signal créé sont inversés.

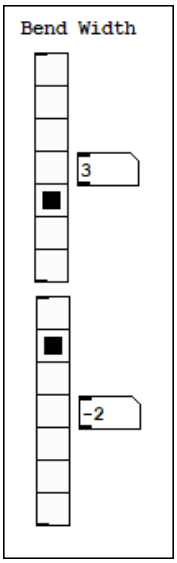
C – Patch du Volume Master :



- **Master_Volume** contrôle le volume de sortie du programme.

Cela n'affecte pas le volume entrant dans l'oscilloscope et le dB-mètre exposés en IV – 1 – A.

D – Patch de Pitch Bend :



Le pitch bend est le nom donné à l'action d'utiliser la pitch wheel, une molette présente sur la quasi-totalité des claviers synthétiseur.

Les informations envoyées par cette molette utilisent un canal dédié au sein du protocole MIDI, etvarient entre 0 et 16 384. Généralement, cela est paramétré pour faire varier une note de ± 2 demi-tons.

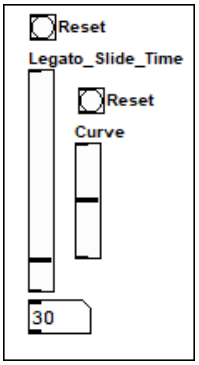
Ce patch permet de paramétrer l'écart de notes qu'accomplit la molette de pitch bend vers le bas et vers le haut de façon indépendante.

E – Paramètres de legato :

- **Legato_Slide_Time** affecte le « Slide time » du legato.

- **Curve** permet de modifier la forme de la courbe du legato.

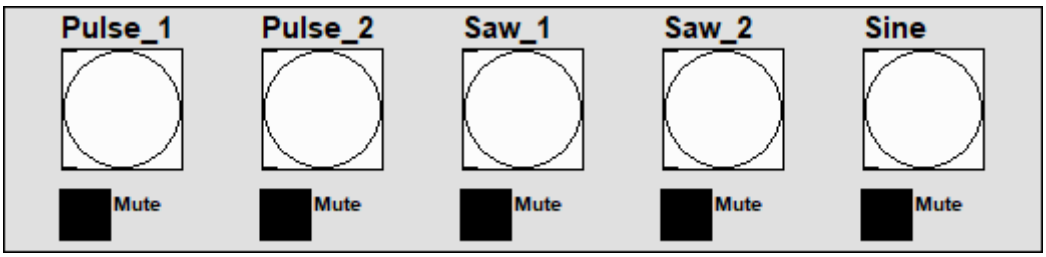
(Voir V – 4 – B)



2 – Partie droite

Le terme « opérateur » est ici utilisé pour désigner un oscillateur, muni de tous les paramètres de base (volume, pan, enveloppe ADSR, unisson, etc), d'un paramètre de modulation, ainsi que d'une suite d'effets. Chaque opérateur, s'il y en a plusieurs, est indépendant dans son paramétrage. Chacun d'eux produit donc son propre son.
Dans ce patch, il y a 5 opérateurs.

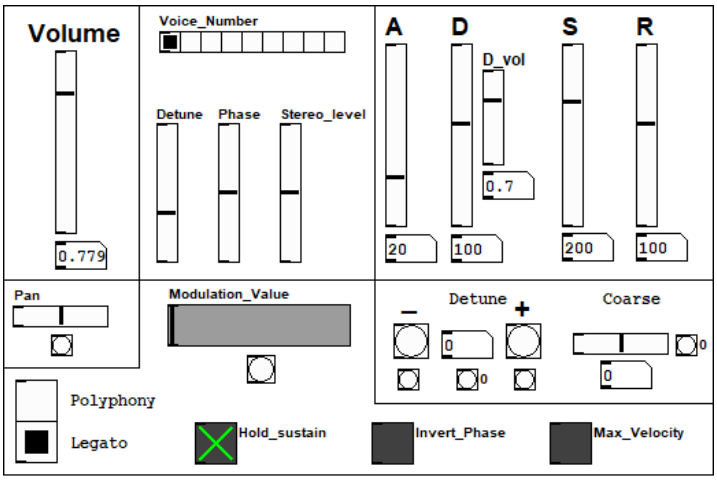
F – Liste des opérateurs :

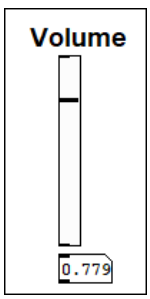


Cliquer sur un opérateur affichera dans tous les patchs de la partie droites les informations lui correspondant. Pour modifier l'opérateur Sine, il faut cliquer dessus. Toutes les modifications effectués par la suite ne seront envoyées qu'à lui.

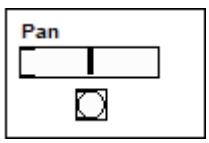
- **Mute** permet de couper le son venant d'un opérateur.

G – Paramétrage des opérateurs :

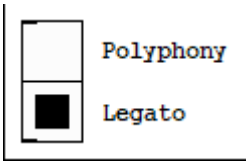




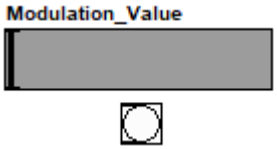
- Volume de sortie de l'opérateur



- Panning du signal de sortie

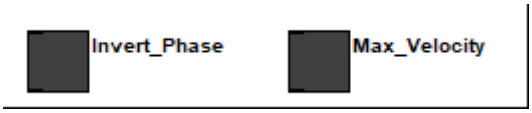


- Choix du mode de jeu (Voir V – 5)



- Taux de modulation des oscillateurs (Voir V – 1)

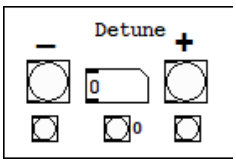
- Le bouton réinitialise la valeur au centre.



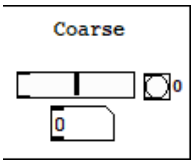
- **Invert Phase** inverse la phase.

- **Max Velocity** remplace toute vélocité non nulle envoyée à l'opérateur par 127, effaçant les dynamiques de jeu.

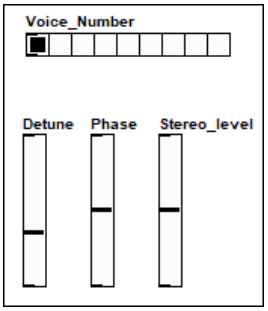
L'offset tuning désigne le fait de décaler la hauteur de la note reçue par l'opérateur. Il s'agit de la somme du Detune et du coarse pit :



- Paramétrage de l'offset à la note près. Les grands boutons ajoutent ± une octave, et les plus petits ajoutent ± un demi-ton.

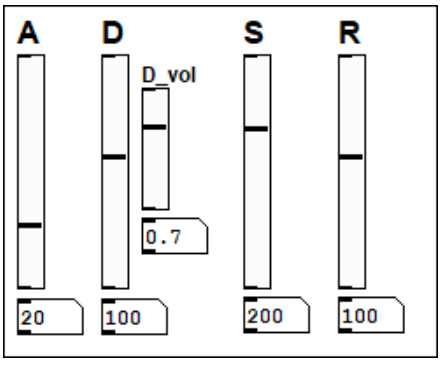


- **Coarse pit**, ou « fine tuning », permet d'ajouter ± 0,5 demi-ton.



- **Voice_Number** modifie le nombre de voix de l'unisson.

- Permet de paramétrer le taux d'écart de Detune, de phase et l'étalement stéréo (Voir V – 2).

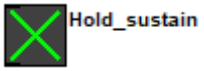


Enveloppe ADSR (Voir V – 3 - C) :

- Permet de modifier les paramètres A, D, S et R dans l'intervalle [5 ; 1000] millisecondes.

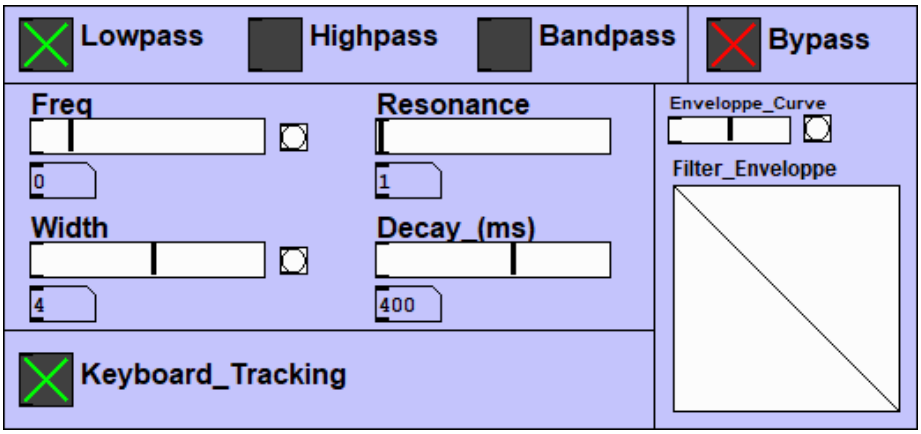
- Le volume de **Decay** est paramétrable entre 0 et 1.

- En mode legato, l'enveloppe ne se réinitialise pas quand une nouvelle note est actionnée.



- Indique si l'enveloppe est jouée d'une traite ou s'il faut que la note soit relâchée pour effectuer la Release. (Voir V – 3 – C)

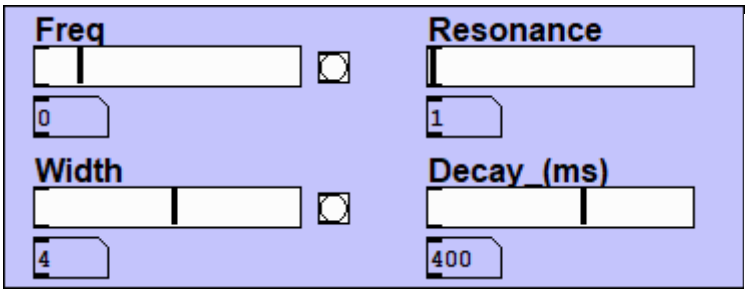
H – Patch Filtre (Voir V – 3 – A) :



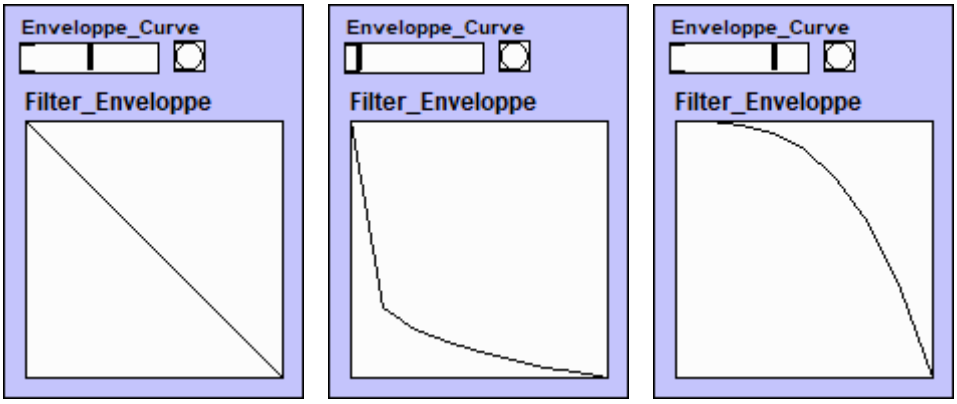
- Choix du type de filtre

- Désactivation du patch

- Activation du mode « suivi de note »



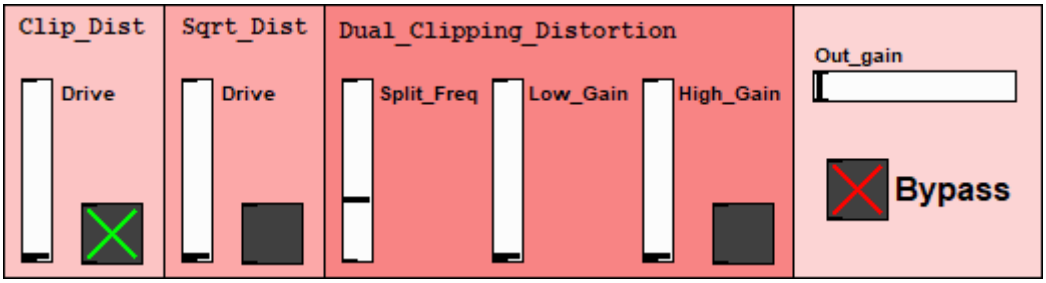
- **Freq** pour fréquence de coupure
- **Width** modifie la largeur de l'enveloppe
- **Decay** correspond à la durée de l'enveloppe

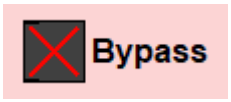


- **Curve** permet de modifier la forme de l'enveloppe du filtre.

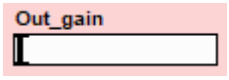
Le bouton réinitialise la valeur au centre.

I – Patch de distorsion (Voir V – 3 – B) :

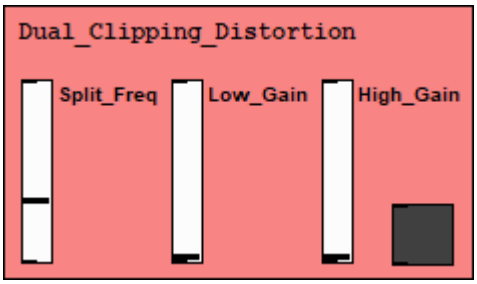




- Désactivation du patch



- Gain de sortie supplémentaire



- **Split Freq** règle la fréquence de coupure des deux bandes de fréquences
- **Low Gain** correspond au drive des basses fréquences.
- **High Gain** correspond au drive des hautes fréquences.

V – Description technique et méthodes utilisées

Le projet se sépare en 2 grandes parties.

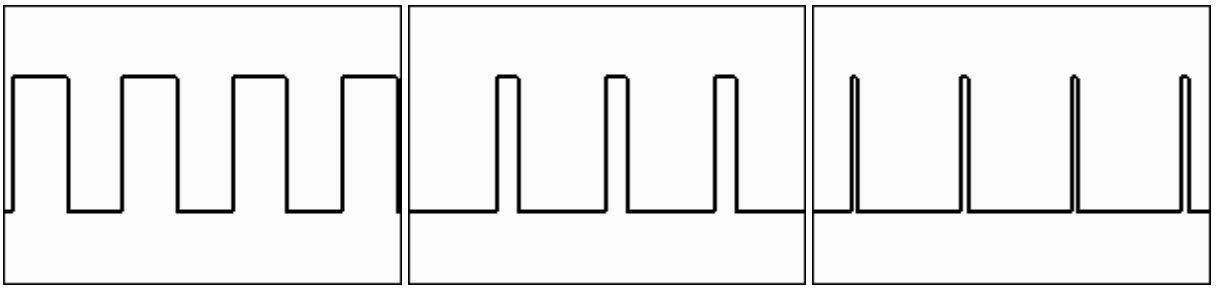
La première partie est l'ensemble de patchs décrits en IV – 1, et l'interface graphique. Celle-ci contient un oscilloscope et un dB-mètre, qui sont des fonctions de base de Pure Data.
Un sous-patch permet de recevoir et de comprendre des information MIDI envoyée depuis un clavier MIDI, ou bien d'interpréter les touches d'un clavier d'ordinateur comme des notes.
Quand un utilisateur appuie sur une touche d'un controller MIDI, ce patch interprète va recevoir la note MIDI ainsi que sa vélocité. Le patch assemble ces deux informations sous la forme d'une liste de 2 éléments, et l'envoie aux autres patchs concernés.
Dans le cas où un clavier d'ordinateur est utilisé, le même processus est appliqué, à l'exception d'une étape en plus, faisant une conversion entre l'identifiant du clavier et une note MIDI.

La 2ème partie de ce projet, et la plus complexe, est l'arborescence de patchs produisant et/ou modifiant du signal. Pour expliquer son fonctionnement, on va remonter son arborescence :

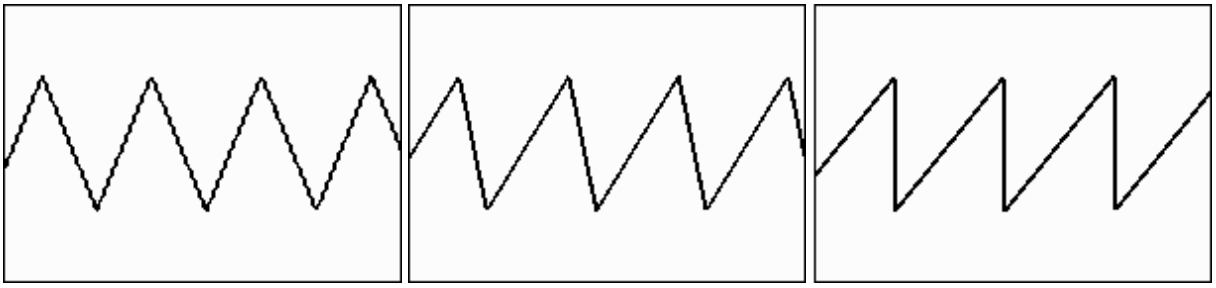
1 – Oscillateurs, formes d'onde

Il y a en tout 3 oscillateurs différents :

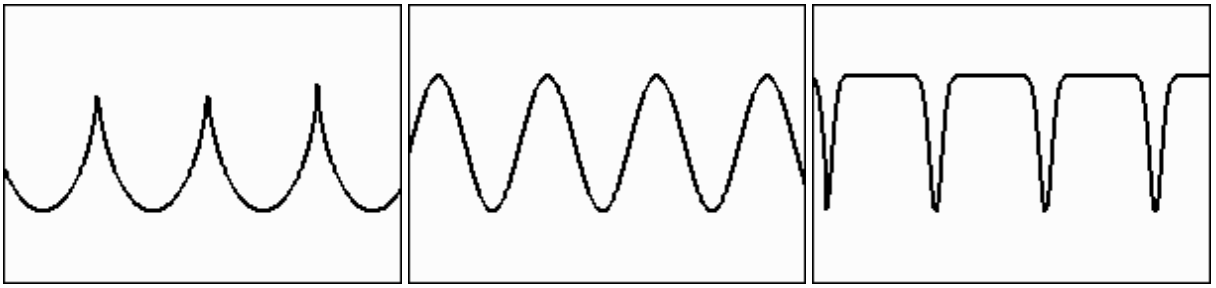
- Un oscillateur PWM (Pulse Width Modulation) :



- Un oscillateur triangle/dent de scie (sawtooth) :



- Un oscillateur sinusoïdal modulable :



Deux des opérateurs ont un oscillateur triangulaire, deux ont un PWM, et le dernier a le sinusoïdal.

A – Oscillateur PWM :

La première étape a été de créer une forme d'onde carrée. Pour cela, le calcul était simple :

carré(t) = sin(t) / |sin(t)|

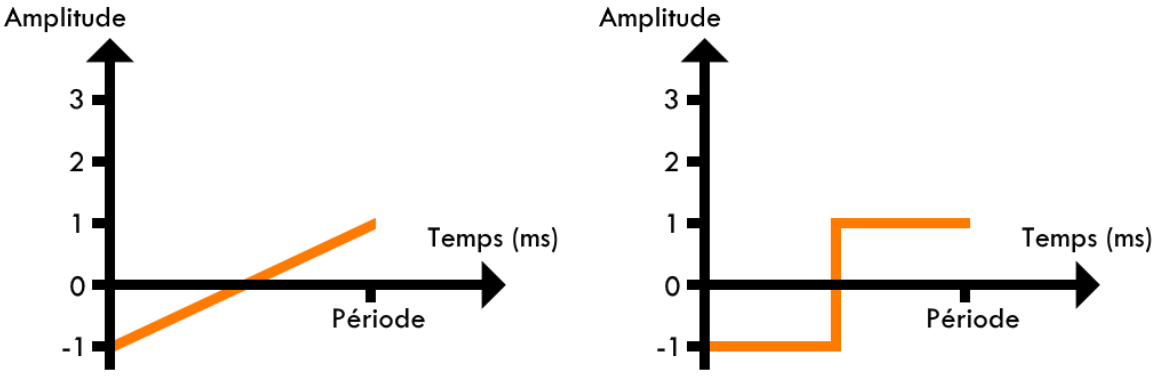
Une fonction divisée par elle-même donne toujours 1, mais divisée par sa valeur absolue, elle donnera 1 ou -1 dépendant de son signe. Ainsi, sqr(t), alterne périodiquement entre 1 et -1.

Cependant, il est assez compliqué de généraliser le processus dans le but d'obtenir une PWM, à partir d'une sinusoïde. Dans Pure Data, les 3 oscillateurs par défaut sont les objets « osc~ », produisant une sinusoïde, « phasor~ » produisant une dent de scie (sawtooth), et « noise~ ».
La solution choisie a été d'utiliser « phasor~ » :

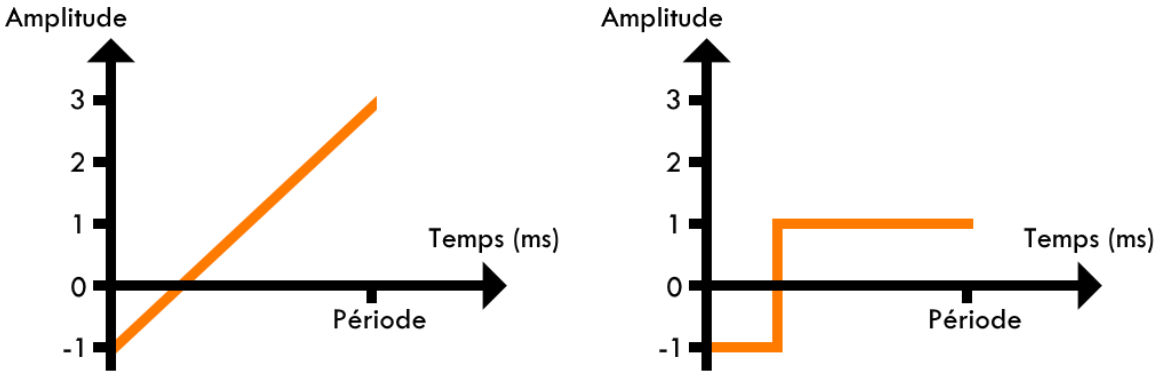
- On fait varier la sawtooth dans l'intervalle [0 ; 2]
- On fait varier l'amplitude de ce signal.
- On lui soustrait 1.
- On le divise par sa valeur absolue.

Voici quelques exemples :

- Rapport cyclique = 0,5



- Rapport cyclique = 0,75



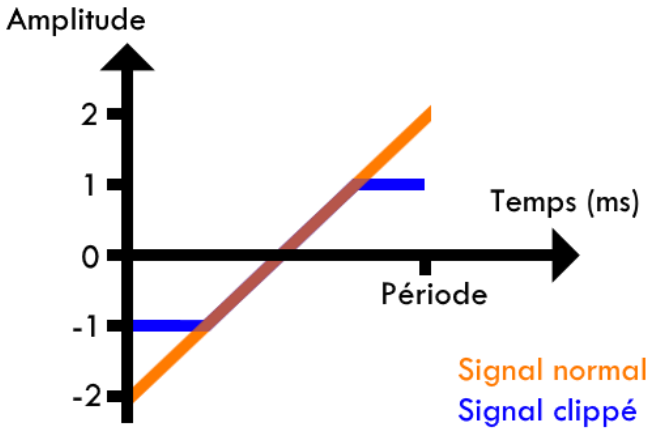
Ainsi, en faisant varier le nombre par lequel on multiplie la sawtooth, le processus effectué par la suite résulte en une variation du rapport cyclique.

B – Oscillateur triangle/sawtooth :

Il existe déjà l'objet « phasor~ » dans Pure Data permettant de produire une sawtooth sur l'intervalle [0 ; 1]. La méthode appliquée pour pouvoir moduler cette forme d'onde se compose en 2 parties :

Le nombre que l'on fait varier s'appelle X, et est sur l'intervalle [0 ; 0,5]. Le signal S(t) est séparé en un signal S1 et S2

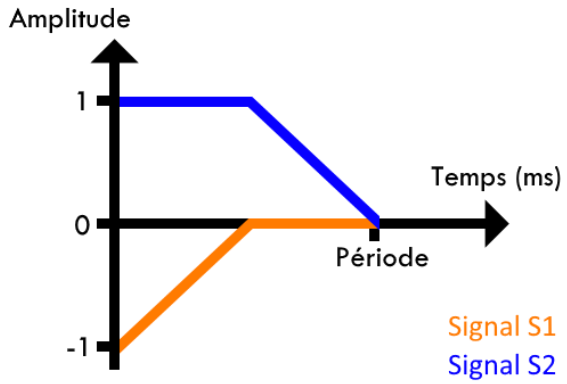
- S1 = S1 * 1/X
- S1 = S1 - 1
- On clip S1 dans l'intervalle [-1 , 0] grâce à l'objet « clip~ »

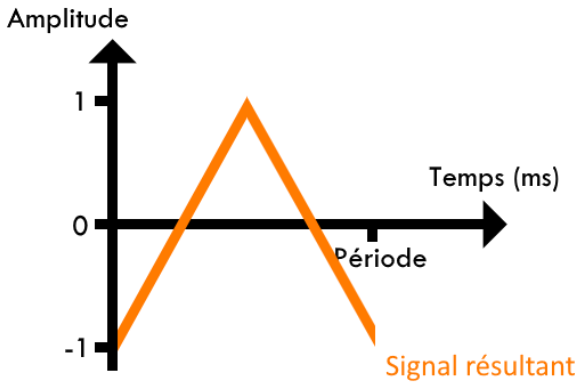


Exemple d'un signal clippé entre [-1 ; 1]

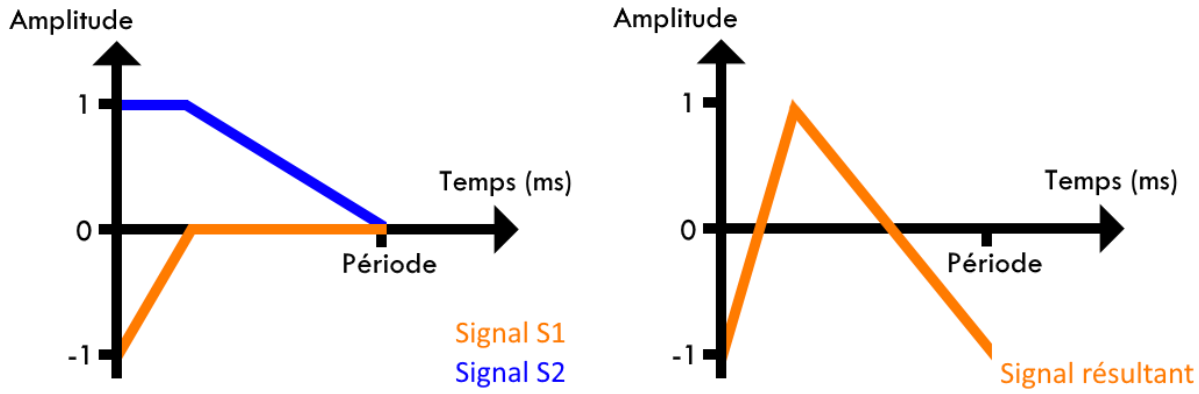
- S2 = (S2 – 1) * (-1)
- S2 = S2 * 1/(1-x)
- On clip S2 dans l'intervalle [0 ; 1]

Ensuite on somme S1 et S2, on multiplie le signal résultant par 2, puis on soustrait 1.
Pour X = 0,5 on obtient :

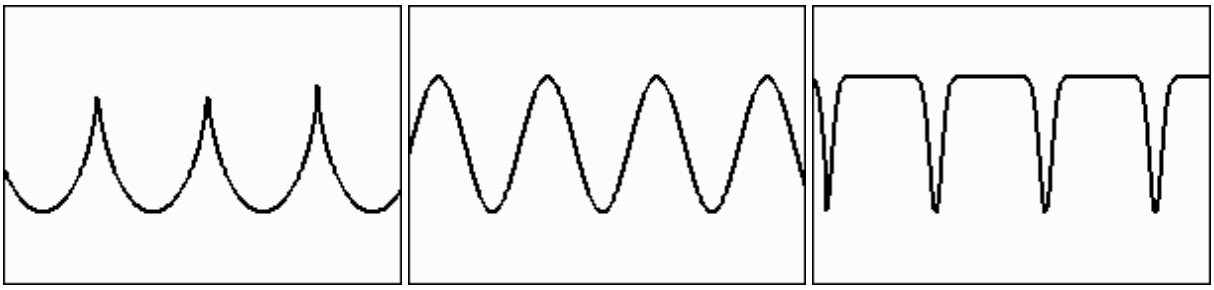




Pour X = 0,25 on a :



C – Oscillateur sinusoïdal :



On part de la fontion « osc~ », qui produit une sinusoïde, et on la fait varier dans l'intervalle [0 ; 1]. On lui applique ensuite une puissance, contenu entre [0,625 ; 16]. Ensuite on multiplie par deux et on soustrait 1 pour établir le signal sur [-1 ; 1]. Exemples ci-dessus.

2 – Unisson

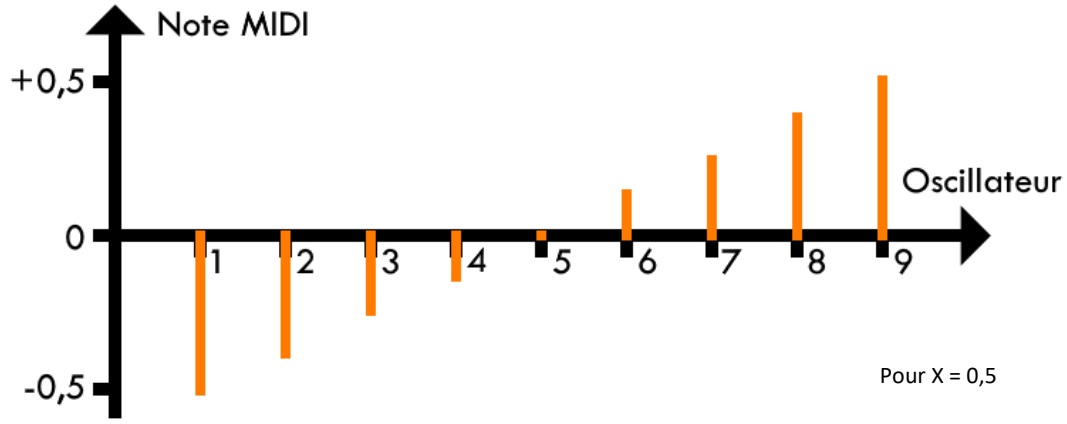
L'unisson consiste en la multiplication d'un même signal. Cependant, additionner le même signal ne fait qu'augmenter son amplitude. De fait, chaque itération de ce signal (voix d'unisson) doit être légérement différente pour créer un réél effet.
Dans ce patch, on peut jouer sur 4 paramètres :

- Le nombres de voix
- Le detune
- La phase
- L'étalement stéréo

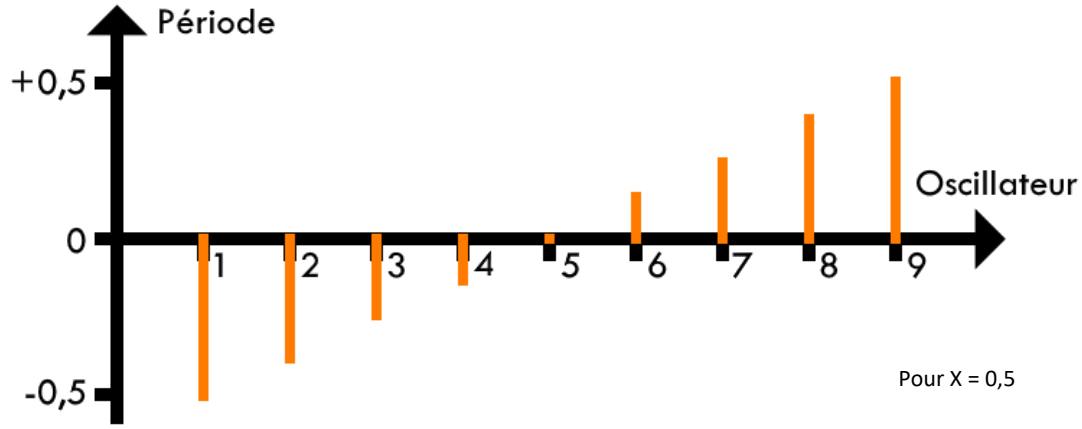
L'utilisateur peut ici utiliser jusqu'à 9 voix, ce qui signifie que le patch d'unisson contient 9 patchs d'oscillateur, qui seront activés ou désactivés en fonction du nombre choisi.

Le principe du detune est de désaccorder chaque oscillateur un peu plus que l'autre, pour que chacun ai sa propre fréquence.
Même chose pour la phase, chaque oscillateur est un peu plus déphasé que le précédent.

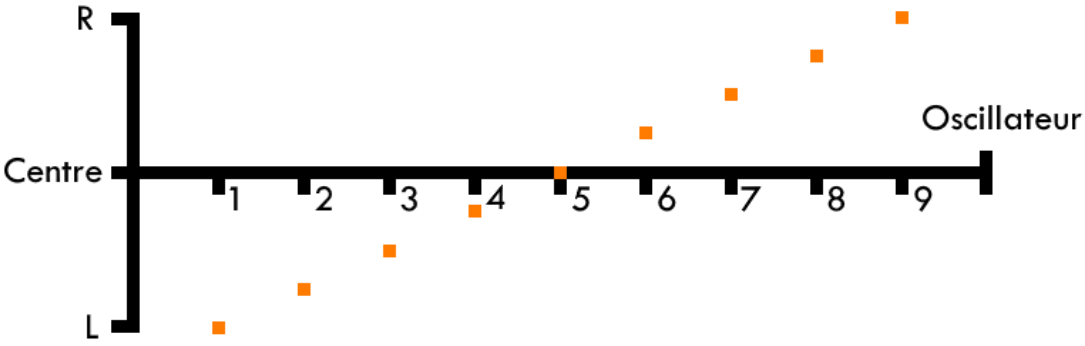
L'utilisateur précise une valeur X comprise dans [0 ; 0,5], valeur qui sera « étalée » du négatif au positif entre tous les oscillateurs :



Le fonctionnement est le même pour la phase.



L'étalement stéréo correspond à la position de chacun des oscillateurs dans le champs stéréo. À 0%, toutes les voix sont « empilées » au centre, créant un son mono. À 100%, les voix correspondent au schéma ci-dessous :



3 – Filtre, distorsion, enveloppe ADSR

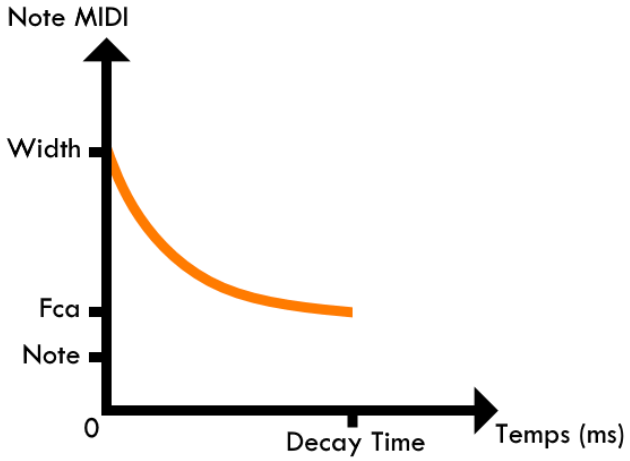
A – Patch filtre :

Il y a trois modes de fonctionnement au filtre, passe-haut, passe-bas, et passe-bande. L'enveloppe de ce patch contrôle la fréquence de coupure de ces trois filtres. Appelons la « Fréquence de coupure d'arrivée » de l'enveloppe Fca.

Si l'option de suivi de note est activée, la Fca est calculée relativement à la note jouée, tel que Fca = Note + Freq. Sinon la fréquence de coupure est fixe, tel que Fca = Freq.

La largeur de l'enveloppe (Width) est l'écart entre la fréquence de coupure de départ et la Fca. L'enveloppe part de (Fca + Width) jusqu'à (Fca).

La forme de la courbe peut être modifiée par le paramètre de « Curve ».



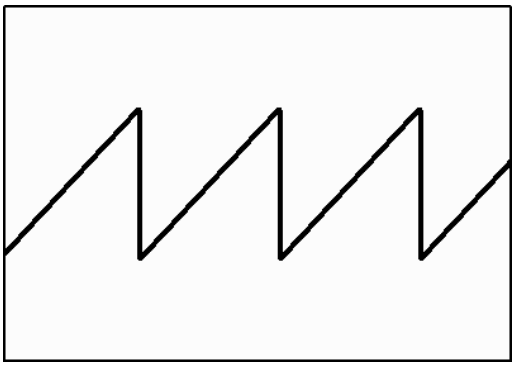
B – Distorsion :

Le patch de distorsion comporte 3 modes différents.

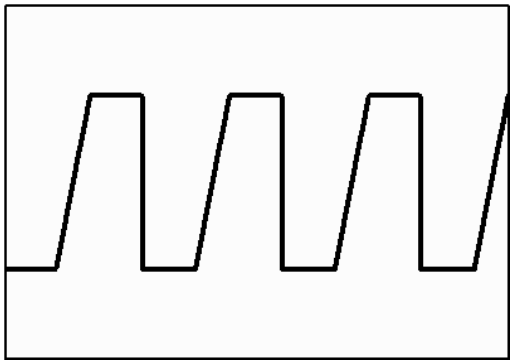
Le 1er mode de distorsion utilise l'objet « clip~ ». (Voir V – 1 – B)
Augmenter l'amplitude du signal pour ensuite le clipper permet de simuler une carréification analogique, qui rajoute des harmoniques.
Le « drive » fait varier cette amplification, modifiant ainsi le taux de distortion.
Cette distorsion est accompagnée d'une filtre coupe haut en sortie autour de 17 kHz, pour éviter de surcharger les aiguës.

Le 2ème mode de distorsion utilise la fonction mathématique racine.
La fonction racine ne pouvant pas s'appliquer à des valeurs négatives, on sépare la partie négative du signal de sa partie positive avec l'objet « clip~ ».
On inverse le signal négatif pour le rammener dans les positifs.
On applique ensuite à ces deux signaux la fonction racine.
On inverse de nouveau le signal négatif on le somme au canal positif.
Dans ce type de distorsion, le drive modifie le X de la « racine de X ».

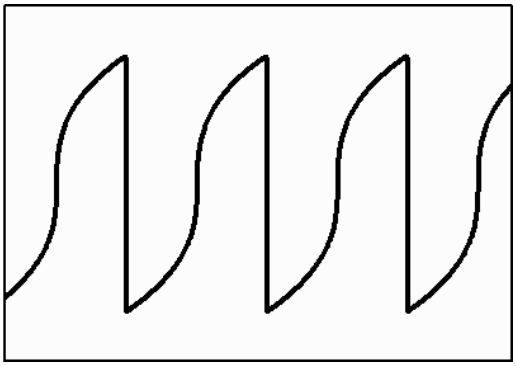
Le 3ème mode de distorsion est le plus aléatoire.
Le signal est envoyé en parallèle dans un passe-bas et dans un passe-haut.
Chacune de ces bandes de fréquences subit la la distortion mode 1.
Les deux signaux sont ensuie additionnés.



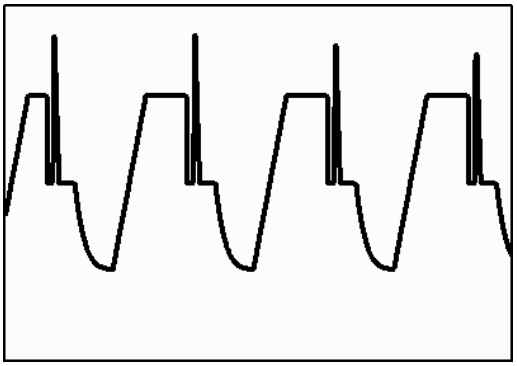
Signal témoin



Exemple de distorsion mode 1



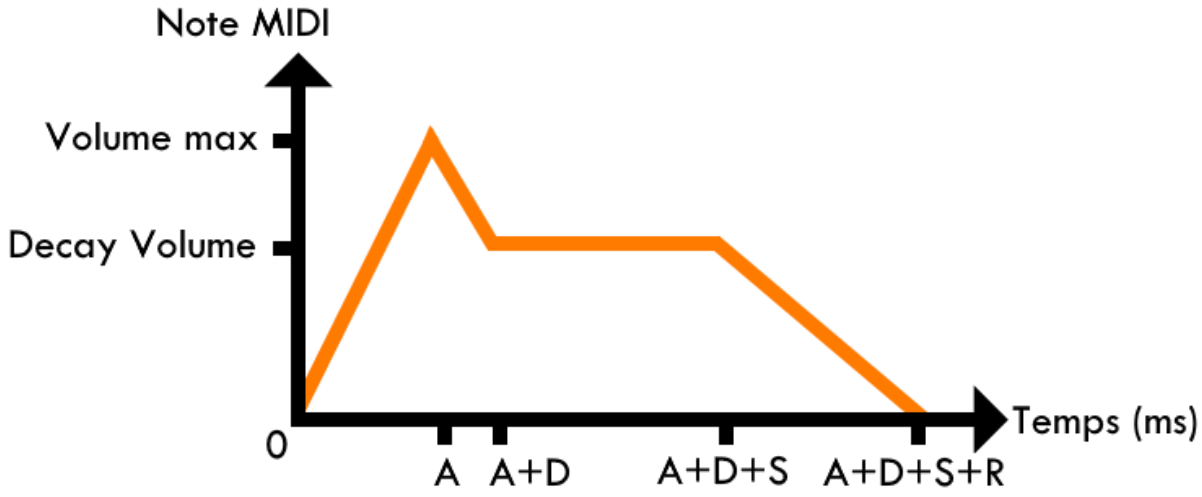
Exemple de distorsion mode 2



Exemple de distorsion mode 3

C – Enveloppe ADSR :

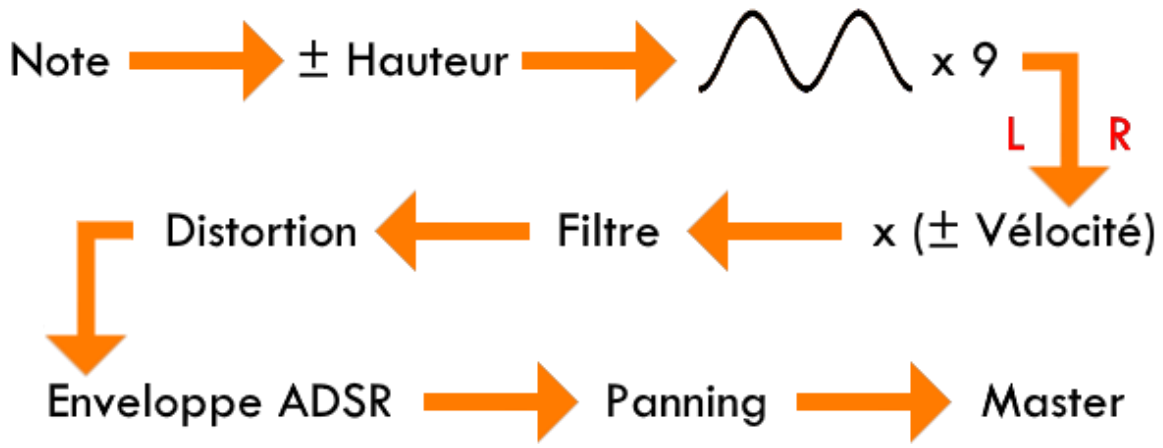
ADSR correspond à Attack, Decay, Dustain, Release, en français « attaque », « chute », « maintien », et « relâchement ». Un 5ème paramètre, le volume de decay, permet de décrire la courbe que suit par exemple le volume, ou la fréquence de coupure d'un filtre.



4 – Les opérateurs

Le texte qui suit décrit la suite d'opérations lorsqu'une note accompagné d'une vélocité non-nulle est reçue par un opérateur :

- Au moment où une note est reçue, un trigger est émis. Un trigger est un signal ponctuel.
- L'offset tuning et le pitch bend sont ajoutés à la note reçue. (Voir IV – 2 – F, IV – 1 – D)
- Cette information est envoyée au patch d'unisson. Le son se sépare ensuite en deux canaux stéréo et qui seront traités parallèlement.
- L'amplitude est modifiée selon la vélocité reçue, et est parfois inversée.
- Le signal est envoyé dans un filtre, dont l'enveloppe est activée à l'aide du trigger.
- Un patch de distorsion est appliqué.
- L'amplitude du signal est modelée par une enveloppe ADSR, qui est activée par le trigger.
- Les deux signaux sont ensuite pannés.
- Master Volume.



Tous les opérateurs utilisés dans ce projet suivent ce système. Les seules différences entre eux sont les oscillateurs.

5 – Modes de fonctionnements

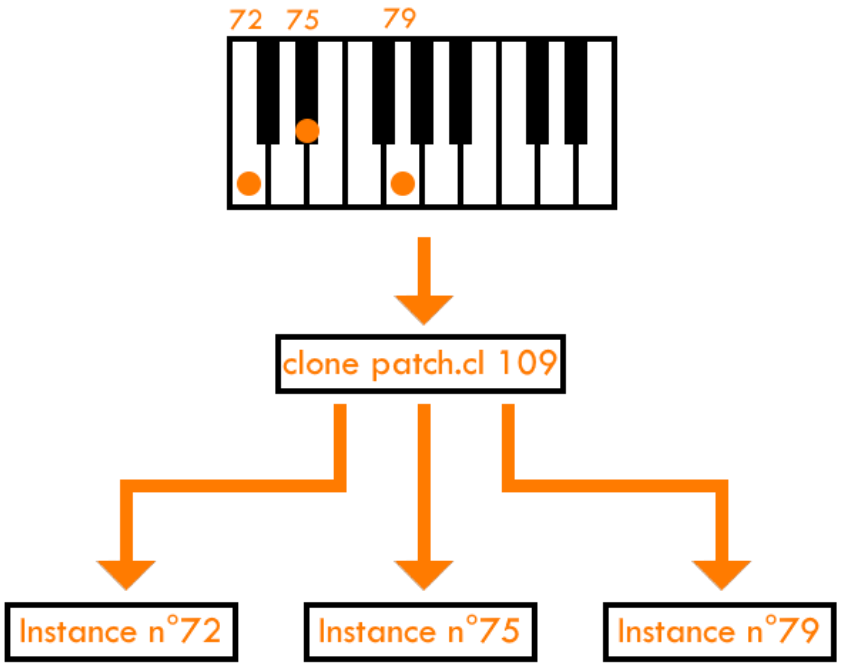
Les opérateurs sont englobés dans d'autres patchs, permettant de les faire fonctionner en polyphonie ou en monophonie :

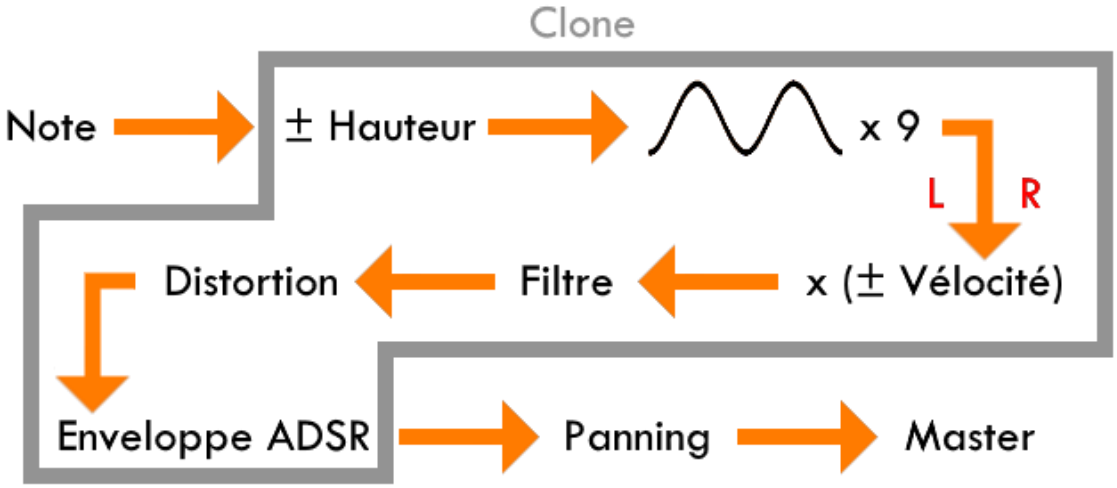
A – Polyphonie :

La polyphonie consiste en la possibilité de jouer plusieurs notes simultanément. Dans ce programme, la fonction « clone » est utilisé pour permettre cela.

Pour faire utiliser la fonction clone de Pure Data, il faut enregistrer dans le répertoire où se trouve le patch parent un autre patch, qui sera indiqué à la fonction clone. On rajoute

ensuite en argument de cette fonction un nombre, nombre qui sera la quantité d'instances du patch à cloner.
Par exemple, la fonction « clone bonjour.cl 12 » permet de créer 12 instances parallèles du patch « bonjour.cl ». Chacune de ces instances est indépendante et a un identifiant propre, allant de 0 à 11.
Dans notre cas, la fonction « clone » est utilisée pour créer de la polyphonie. Chaque note du clavier est assignée à une des instances du clone, permettant donc de jouer plusieurs notes à la fois.

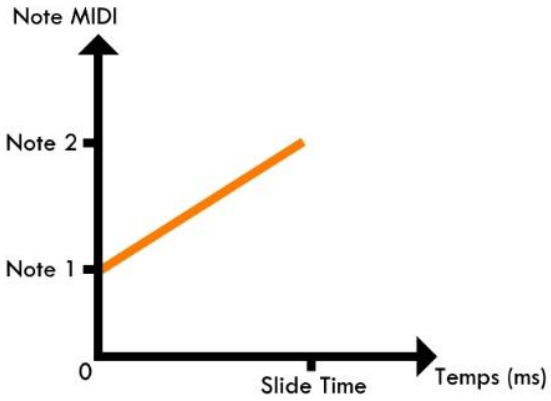




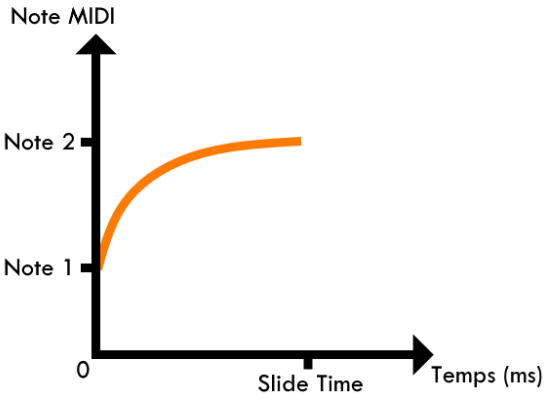
B – Monophonie :

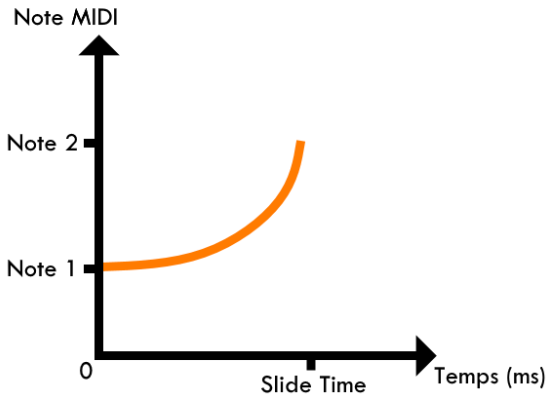
A l'inverse, pour jouer en monophonie, il n'y a besoin que d'un seul oscillateur dont on fait varier la fréquence en fonction de la note jouée. Il n'est donc pas nécessaire de cloner l'opérateur. L'appellation de ce type de jeu est « Legato ».

Le legato est une articulation de jeu, qui signifie « lié ». Selon la vitesse que l'on prend pour aller d'une note à l'autre, cela peut donner l'impression de « glisser » entre les notes. Ce temps est ici appelé « Slide Time ».



Il est aussi possible de modifier la courbure de legato pour obtenir le type de résultats ci-dessous :



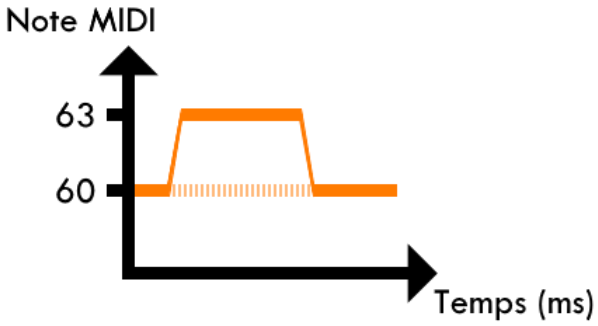


Voici le fonctionnement adopté :

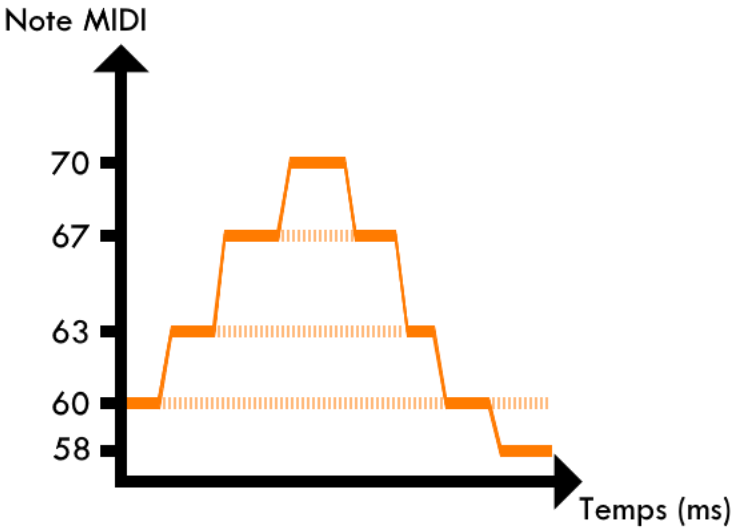
- Si aucune note n'est déjà pressée, au moment où l'utilisateur en presse une, le son part de la dernière note jouée jusqu'à la nouvelle, en une durée paramétrable (Voir IV – 1 – E), appelée « slide time ».

- Si l'utilisateur presse une note alors qu'une autre l'est déjà, un processus similaire s'applique. L'oscillateur part de la note en train d'être jouée, et va jusqu'à la nouvelle note.

Dans le cas où l'utilisateur appuierait par exemple sur la note 60, puis sur la note 63 tout en maintenant la touche 60, la note partirait de 60 à 63. Mais si cet utilisateur par la suite relâche la note 63, il faut que le son reparte de nouveau vers la note 60, car celle-ci est toujours enclenchée :



Dans un cas plus complexe, un utilisateur pourrait par exemple faire cet enchainement de note 60 – 63 – 67 – 70, et ensuite relâcher 70 puis 67 puis 63, pour ensuite appuyer sur la touche 58. Le graphique ressemblerait à ça :



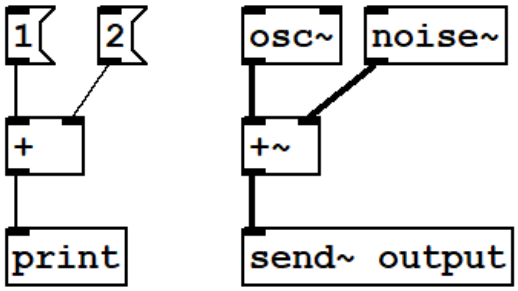
La méthode utilisée dans ce patch pour obtenir ce fonctionnement implique un tableau. Lorsqu'une note accompagnée d'une vélocité non nulle est reçue, elle est ajoutée à la fin du « tableau des notes ». La vélocité est ajoutée à la fin du « tableau des vélocités ». Cela permet de garder en bijection les vélocités et les notes, et ainsi de garder l'intention de jeu.
Lorsqu'une note accompagnée d'une vélocité nulle arrive, le tableau des notes est alors balayé pour trouver la position de la note. Les informations correspondant à cette position sont ensuite retirées des 2 tableaux.

A l'origine, le programme utilisait une liste et non un tableau, mais cela posait de nombreux problèmes. Dans un tableau, chaque élément est identifiable grâce à sa position dans celui-ci, permettant de lire directement ou de le remplacer, là où dans une liste, un élément n'est pas identifiable par sa position. Cela force à découper la liste en une myriade de liste de 1 élément, pour pouvoir lire chacun d'eux, pour ensuite recomposer ces listes en une seule. Ce processus peu optimisé et compliqué a amené à préférer l'utilisation de tableaux.

6 – Optimisation

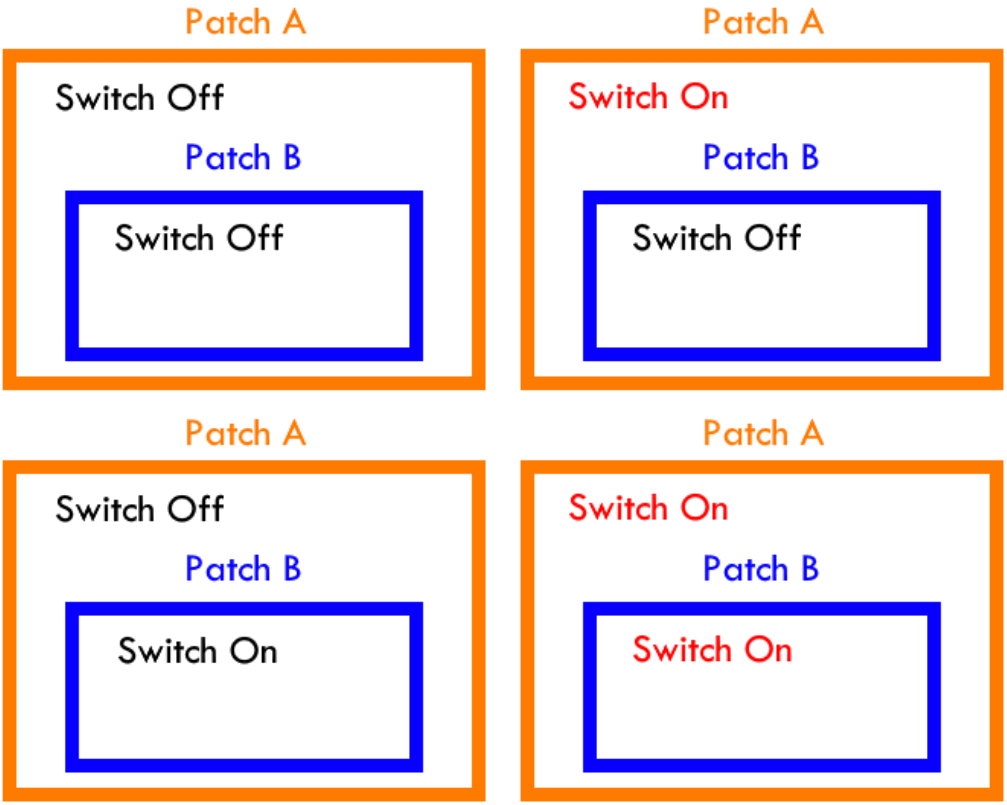
Dans Pure Data, il est possible de classifier les objets en 2 groupes, selon ce qu'il renvoie, selon leur type d'output. Un output « classique » correspond à un nombre, une chaîne de caractère, une liste, un trigger, etc. Le second type d'output est l'output signal.

Si un objet produit un output classique, par exemple un nombre, ce nombre va avoir une place mémoire allouée, et restera « inerte », à moins que cette mémoire allouée soit réécrite par une nouvelle info.



Au contraire un signal est un flux d'informations continu. De fait, les objets produisant du signal ne créent pas de l'information de façon ponctuelle, mais font des calculs sans s'arrêter. Cette différence se marque par l'ajout d'un « ~ » au nom de l'objet.

Cependant, ces objets produisant du signal ne peuvent entreprendre des calculs que si le DSP (Digital Signal Processing) est activé. Ce DSP est activable/désactivable au sein d'un patch à l'aide de l'objet « switch~ ». Le fonctionnement de cet objet est le suivant :



Si le DSP d'un patch est désactivé, les DSP de ses patchs enfant est désactivé. Si le DSP du parent devient actif, le DSP de l'enfant est réactivé ou non selon l'état de son Switch à lui. Par défaut, activer le DSP d'un patch parents activera celui de ses sous-patchs.

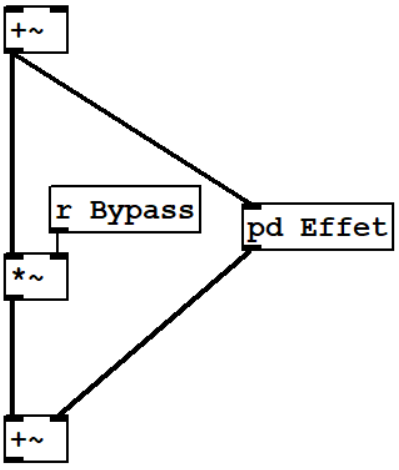
En reprenant le schéma de fonctionnement du synthétiseur, on a un patch d'unisson, contenant 9 patchs, donc 10 patchs DSP actifs. Les signaux sont ensuite traités. En comptant le patch opérateur en lui-même, cela fait 14 patchs DSP actifs.
Avec l'objet « clone », on multiplie par 109 instances, amenant à peu près à 1500 patchs, qui sont elles-mêmes dans 5 opérateurs, ce qui nous amène à peu près à 7500 instances DSP actives.
Le problème a donc été de trouver toutes les conditions possibles pour indiquer quels DSP activer et lesquels désactiver.

La première étape est de désactiver le DSP d'un opérateur lorsque son volume est à 0.

En mode polyphonie, le DSP d'une instance du clone n'est activé qu'à la condition que la note correspondante soit pressée (et que le DSP de son patch parent le soit aussi, mais cela sera sous-entendu à chaque fois par la suite). Il se désactive que une fois que l'enveloppe ADSR se conclue, et pas quand la note est relâchée.
Le mode legato, le DSP s'active au début de l'enveloppe ADSR et se conclue à sa fin.

Dans le patch d'unisson, seuls les patchs contenant un oscillateur utilisé a un DSP actif.

L'optimisation des DSP passe aussi par l'exécution des « bypass ». Bypass signifie « court-circuit ». Par exemple :



Lorsque l'on veut avoir l'effet, on coupe le signal passant en parallèle de celui-ci et on active son DSP.

Lorsque l'on veut bypass cet effet, on active le signal en parallèle, et on désactive son DSP.
Cela désactivera les objets présents dans le patch, ainsi que ses sous-patchs, épargnant de la puissance de calcul.

Dans des versions précédentes, le bypass était inclus dedans, ce qui forçait à désactiver le DSP des sous-patchs uniquement, et pas du patch en lui-même, laissant certains objets continuer à calculer.

Un choix qui a été fait pour optimiser le synthétiseur a été l'extraction du synthétiseur des opérateurs. Le fonctionnement du delay prend beaucoup de mémoire et de calcul, donc

avoir un delay sur le master au lieu d'un sur chaque opérateur permet d'améliorer les performances, mais au détriment de certaines possibilités sonores.

Une optimisation très importante était la gestion du pitch bend. (Voir IV – 1 – D)
Dans les versions précédentes, utiliser la molette de pitch bend envoyait l'informations à toutes les instances des opérateurs, donc à 545 instances, et chacun des 9 oscillateurs de ces 545 instances recalculer cette information à cause du detune de l'unisson.
Cela causait au programme de freeze quelques secondes et de ne plus produire de sons.

La solution à ce problème était similaire à celle du DSP, c'est-à-dire envoyer l'information uniquement aux instances actives du clone. Cela réduit la quantité de calcul de potentiellement 4 905 à quelques dizaines selon la façon dont est paramétrés le patch.

VI – Résultats et perspectives

Il y a plusieurs directions que peut prendre ce projet pour aller plus loin :

La première et la plus évidente, serait de rajouter des fonctionnalités. A l'origine du projet, il était question d'avoir une matrice de modulation FM. Cela a ensuite été retiré car trop peu pratique a implanté dans le projet, néanmoins cela pourrait bien finir par être rajouter plus tard.
Une autre fonctionnalité qui devrait voir le jour est l'ajout d'un Arpeggiateur. Dans les versions ultérieures du projet, les prototype, il y avait un Arpeggiateur, mais son fonctionnement était trop approximatif et incompatible avec la nouvelle architecture du programme.

Ce patch étant un instrument virtuel, il peut paraitre intuitif que l'on puisse l'utiliser au sein d'un DAW (Digital Audio Workstation). Cependant, la plupart des DAW utilise des formats de fichiers spécifique (VST, AU, AAX pour les plus courants). Or, les patchs Pure Data ne sont pas dans ces formats, ce qui les rend inutilisable dans ce genre d'environnement.
Il existe des programme « pont », tel que pdVST, ou encore Camomile, permettant de créer fichier interprétable en tant que VST par un DAW.
Néanmoins, cela ne résout pas le 2ème plus important problème : l'interface.

Bien qu'efficace quand il s'agit de créer un projet fonctionnel, l'interface n'est pas très malléable, et encore moins « user-friendly ». Créer une interface qui propose une expérience utilisateur claire et agréable est quasiment mission impossible. Pour pallier ce problème, il est possible de créer une couche supplémentaire de code dans un autre programme qui pourrait faire le lien entre une interface graphique et le Patch Pure Data.

Mais une autre solution, allant de pair avec le problème mentionné précédemment, serait de recoder entièrement le patch dans un langage adéquat. Cela permettrait d'être beaucoup plus souple dans la création de l'interface graphique, et de créer directement un fichier au format VST, sans avoir à passer par des étapes supplémentaires.

Une autre possibilité serait de créer un pont entre du matériel physique et le patch Pure Data, à l'aide d'un Arduino.
Cette idée est à l'origine du projet. Le but était de se servir d'un clavier MIDI, de créer un ensemble de potentiomètre, de rajouter quelques contrôles supplémentaires, de les connecter à un arduino, afin de les faire communiquer avec un patch Pure Data.

Compte-tenu de l'évolution qu'a connu ce patch, et de son expansion, construire ce synthétiseur serait maintenant particulièrement complexe, mais néanmoins intéressant.

Lexique :

Arpeggiateur : Un arpeggiateur est une fonctionnalité de certains claviers, qui est que lorsque qu'un accord est joué, les notes le composant sont jouées les unes après les autres en boucles, de sorte à en faire un arpège.

DAW : Digital Audio Workstation. Logiciel de création sonore, musicale, et autres.

Delay : Effet sonore proche de l'écho.

Demi-ton : Unité de base dans le langage musical, qui désigne une distance entre deux notes. Entre Do# et Do se trouve un demi-ton, entre Do et Ré deux demi-tons.

Drive : Synonyme pour amplification. La distorsion est parfois dû à une sur-amplification, en anglais overdrive. Le terme drive est donc parfois utilisé pour désigner le taux de distorsion, car plus il y a de drive plus il va y avoir de distortsion.

Modulation FM : Type de modulation consistant à faire varier la fréquence d'un oscillateur, souvent par le signal produit par un autre oscillateur.

Oscillateur : Objet créant un signal périodique.

Panning : Le panning désigne le fait de placer un élément dans le champ stéréo (plus vers la droite ou vers la gauche).

Synthétiseur modulaire : Catégorie regroupant les synthétiseurs où chaque élément est modifiable. L'ordre des effets est modifiable, les effets en eux-mêmes, les oscillateurs, tout est modifiable et paramétrable. On peut enlever ou rajouter des éléments au synthétiseur à volonté.

Synthétiseur modulaire : Contrairement au modulaire, tout n'est pas modifiable à souhait. On peut ne pas avoir la possibilité de rajouter des éléments, ou l'ordre d'application des effets n'est pas modulable, etc. Le caractère modulaire vient du fait que le son est néanmoins grandement paramétrable.

Trigger : En mathématique, un trigger peut être considéré comme une impulsion de Dirac, un signal passant de 0 à 1 puis de nouveau à 0 en un temps extrêmement court. Sur un synthétiseur analogique, un trigger sert souvent pour déclencher des événements, tel que l'activation de percussion, ou des flashs de lumière.

Vanilla : Indique l'absence d'objets importés, extérieurs au langage de base, qui obligerait une personne à installer ces extensions pour pouvoir faire fonctionner ce projet.