programer un synthétiseur à modulation de fréquence avec Pure data

François (a.k.a. Chon)

Nous allons programmer un synthétiseur polyphonique à modulation de fréquence.

Ce didacticiel suppose que vous savez utiliser les fonctions de base de Pure Data (ajouter, connecter des objets, lancer un patch, gérer les paths...)

Je ne suis moi-même pas un professionnel de Pure Data. Il se peut donc qu'il y ait des mauvais choix de ma part, je vous remercie de ne pas trop en tenir rigueur...

Vous n'avez besoin d'aucune librairie externe pour réaliser ce patch.

La notation entre crochets [bidule][1] *se réfère par défaut à l'objet "bidule" dans Pure Data*

Si vous avez un doute sur un objet, vous pouvez "cliquer-droit" dessus et mettre help. Pour les non-anglophones, une traduction des fichiers d'aide est en cours... Un peu de patience !!

1 Un peu de théorie (trois fois rien ! !)

La hauteur d'un son est déterminée par sa fréquence (par exemple le la3 a une fréquence de 440 Hz) : plus la fréquence est élevée, plus le son est aigu. Le principe de la modulation de fréquence est (comme son nom l'indique) de faire varier la fréquence d'un son au cours du temps.Si la modulation est lente, on obtient un effet de vibrato. Par contre, si la modulation est rapide, l'oreille entend un changement de timbre[2].Nous allons donc appliquer ce principe dans un synthétiseur polyphonique avec une petite enveloppe Attack-Release.

2 D'abord un synthé monophonique

Bon, hé bien il est temps de lancer Pure data !

Ouvrez un nouveau patch, ajoutez-y un objet [pd generateur] : une nouvelle fenêtre s'ouvre.

2.1 Le générateur de sons

C'est ici qu'aura lieu la synthèse du son proprement dite.On va moduler une sinusoïde avec une sinusoïde : le but est donc d'avor une fréquence qui varie autour d'une valeur de base (qui est la hauteur de la note). Et on veut pouvoir agir sur la fréquence et l'amplitude de cette modulation.

[1] et [bidule~] à un objet bidule de signal.

[2] la sonorité peut être radicalement transformée, mais il est très dur de prévoir le résultat...

1. Créez 2 objet [osc~] :l'un sera notre modulateur, l'autre notre générateur. Nous allons d'abord nous occuper du modulateur : on veut agir sur son amplitude et sa fréquence.
2. Créez un atom dans l'inlet de gauche de [osc~](il réglera la fréquence de modulation), un objet [*] avec dans son inlet droit un slider[3] entre 0 et 1, et dans son inlet gauche un atome (la fréquence du son) : pour avoir l'amplitude de la modulation on multiplie la fréquence du signal[4] par un nombre compris entre 0 et 1.Réinjectez la sortie sur un [*~]pour le multiplier avec [osc~].

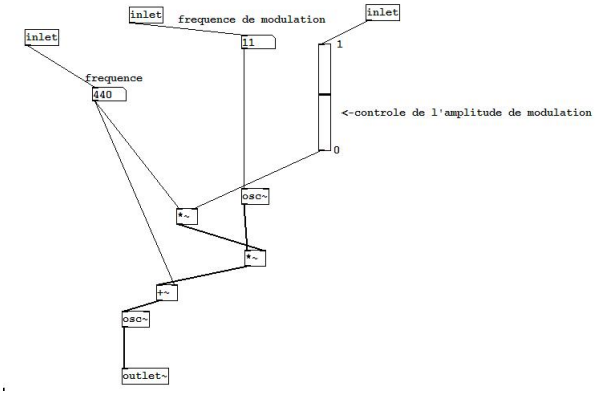
   A la sortie de l'[osc~] nous avons un signal qui oscille à la fréquence voulue.
3. Maintenant, on le fait "attaquer" le générateur [osc~], et le tour est joué!! vous avez un générateur dont la fréquence oscille au cours du temps.

Branchez la sortie du générateur sur un [dac~] et jouez avec les paramètres (essentiellement la fréquence et l'amplitude de modulation) en ouvrant bien les oreilles. Ca vous plaît ?

Alors on continue!

Ce subpatch prend en compte 3 variables : la fréquence de la note, la fréquence de modulation et l'amplitude de modulation. Créez donc trois [inlet] et branchez les aux endroits appropriés. De même, ajoutez un [outlet~] en sortie du générateur (enlevez le dac~)

voici une image de ce que vous devriez avoir si j'ai été assez clair...



2.2 Une enveloppe AR

Le problème avec le générateur de sons que l'on a programmé, c'est qu'il ne s'arrête jamais!! On va donc lui greffer une enveloppe AR[5]. Revenez dans la fenêtre principale de votre patch, et créez un deuxième subpatch : [pd Enveloppe].

A priori on voudrait que notre synthé fonctionne avec des données MIDI[6], on va donc utiliser les propriétés de ces dernières.Une note MIDI est définie par 2 valeurs :lorsque la note est jouée ce sont le numéro MIDI de la note[7] ainsi que

[3]cliquez droit dessus et allez dans propriétés pour régler les bornes du slider.

[4]bien sûr on peut mettre une amplitude indépendante de la fréquence, mais en limitant l'amplitude de modulation de la sorte on s'assure de n'avoir que des fréquences positives

[5]'Attack-release' :attaque,relâchement.

[6]on implémentera le necessaire en temps voulu.

[7]son "pitch".

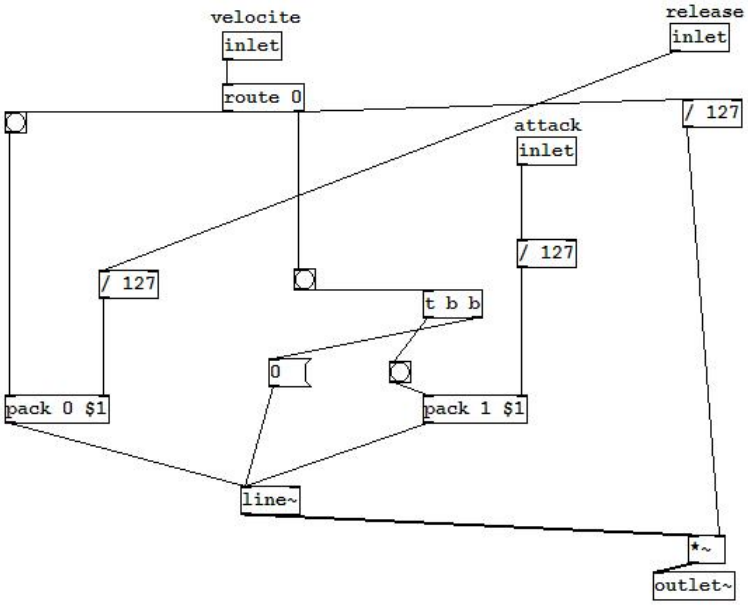
sa vélocité, lorsque la note est relachée, ce sont re-le numéro de la note et une vélocité nulle[8].

Notre enveloppe acceptera donc 3 paramètres en entrée : la vélocité, et une valeur d'attaque ainsi que de relâchement. Le principe est le suivant :

- lorsqu'une note est jouée l'amplitude augmente progressivement (pendant la durée d'attaque)de zéro à une valeur donnée par la vélocité,
- lorsque le message 'note off' (velocité=0) est reçu, l'amplitude retombe à zéro progressivement(pendant la durée de relâchement).

L'objet [line˜] est tout désigné pour générer l'enveloppe. Reste donc à savoir comment déterminer les paramètres à lui envoyer.

Finalement on doit distinguer 2 cas : soit la vélocité est nulle, soit la vélocité est non nulle. L'objet [route 0] va nous permettre cette distinction. On ne va donner que 2 valeurs possibles pour l'enveloppe :soit 1 (vélocité $\neq$ 0) soit 0 (note off), et remultiplier par la vélocité derrière. Bon, un schéma vaut souvent beaucoup mieux qu'un long discours :



Ici [route 0] envoie un bang dans son outlet gauche lorsqu'il reçoit 0 en entrée[9] : le [pack 0 $][10] envoie alors dans le [line˜] le couple (0,"temps de relâchement")[11].

Lorsque qu'il reçoit autre chose que 0, le route envoie la vélocité sur son outlet droit : on multiplie alors le signal de sortie par la vélocité (normalisée à l'unité), et on procède de la même façon que précédemment pour créer l'enveloppe (à ceci près qu'on envoie une remise à zéro juste avant de relancer la rampe au cas où la note précédement jouée n'ai pas eu le temps de se mettre à zéro, mais ce n'est pas obligatoire).

Hé bien on vient de programmer les deux éléments les plus importants de

[8] ou 'note off'.

[9] c'est un cas particulier, on verra son fonctionnement plus loin

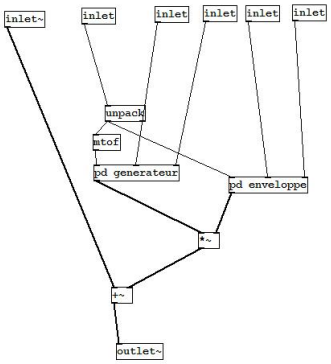
[10] le $ signifie "variable locale", mais en fait on aurait pu se contenter d'un zéro, qui aurait de toute façon été remplacé par ce qui rentre dans l'inlet droit.

[11] on remarquera que j'ai divisé les temps d'attaque et de relâchement par 127 dans un souci de normalisation de données MIDI.

notre synthétiseur : on vient de faire un synthé FM monophonique ! On va le sauvegarder pour en faire un objet indépendant, mais d'abord il faut lui mettre ce qu'il faut comme inlets et outlets. Attention à leurs positions relatives, Pure Data s'en sert pour placer les inlet sur l'objet résultant.

Faites attention : dans notre synthé polyphonique les notes arriveront "empaquetées" (pitch et vélocité), on met donc un objet [unpack] sur l'inlet correspondant. De plus le générateur recevra un pitch (un numéro de note MIDI) et on veut avoir une fréquence : l'objet [mtof][12] est fait pour ça.

Au passage remarquez l'ajout d'un [inlet~] tout à gauche qui s'additionne au signal de notre synthé mono : on s'en servira pour la polyphonie en mettant nos synthés en série.



Sauvez donc votre patch sous le nom de MonoSynthAR~ (attention à la casse !) en vérifiant bien que vous le sauvez dans un de vos path (sinon vous ne pourrez jamais le rappeler dans un autre patch).

3 Polyphonie...

Allez ! On a presque fini ! On va faire un synthé à 4 notes de polyphonie (mais la procédure est la même quelquesoit la polyphonie voulue).

3.1 ...Grace à l'objet [poly] ! !

L'objet [poly] prend comme argument le nombre de voies de polyphonie (n), en entrée des paires (Pitch et vélocité) et les ressort avec en plus un nombre compris entre 1 et n dans son inlet tout à gauche : ça a l'air compliqué, mais ça ne l'est pas du tout ! (Mais je vous conseil de regarder l'aide pour bien en comprendre le fonctionnement).

3.2 L'objet [route]

L'objet [route] prend en entrée une liste et possède des arguments lors de sa création (autant que l'on veut).Il compare le premier terme de la liste à ses argument.

– Si le premier terme de la liste correspond à un des argument, [route] envoie **le reste de la liste** dans l'outlet[13] correspondant. Mais si la liste ne

[12] MIDI to frequency. Il existe également l'objet inverse : [ftom]

[13] l'ordre des outlets est le même que l'ordre des arguments

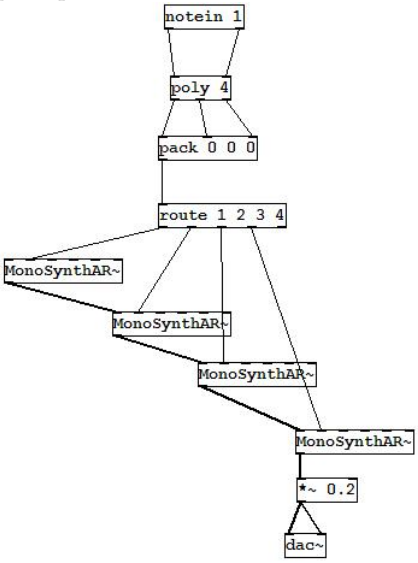
contient qu'un seul argument, comme dans le cas de notre enveloppe, route envoie un bang.
- si aucun argument ne correspond au premier terme de la liste **la liste entière** est envoyée dans l'outlet tout à droite.

Et voila!! On a tous les outils en main pour "polyphoniser" le synthé.

3.3 D'abord les signaux

On va d'abord monter "la chaîne du son" : On place d'abord un objet [notein 1](1 est le numéro de canal) qui donne le pitch (à gauche) et la vélocité (à droite) , et les envoie dans un [poly 4].

On empaquette les 3 parametres et envoie le paquet dans un [route 1 2 3 4], qui va nous distribuer les notes dans quatre synthés monophoniques. On place donc 4 objets [MonoSynth~] dont les deuxièmes inlets sont branchés sur les outlets du [route 1 2 3 4] (ne pas brancher sur l'outlet tout à droite!). Les 3 premiers [MonoSynth~] ont leurs sorties branchées sur l'inlet tout à gauche du [MonoSynthAR~] suivant : seul le dernier [MonoSynthAR~] a son outlet branché sur le [dac~] (mettre un [*~ 0.25] car on somme 4 sinusoïdes, donc il y a des risques de saturation). Jouez sur votre clavier (ou logiciel) favori, et normalement vous devriez avoir du son (pour le moment ce ne sont que des sons purs). Vous pouvez tester la polyphonie en jouant des accords[14]. Si ça marche, on peut passer à la suite.



3.4 Les Control Change

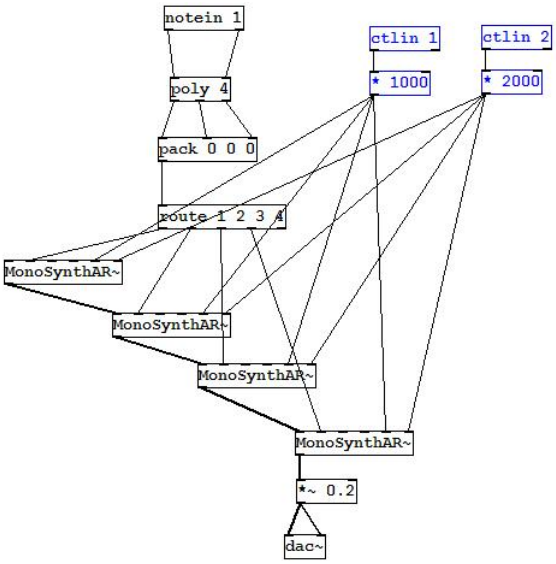
Il nous reste 4 paramètres à contrôler : les couples Attaque/Relâchement, et Fréquence/Amplitude de modulation. On va se servir des objets [ctlin] pour les manipuler.

En gros on va se servire de message de Control Change (de 0 à 127). On a deja normalisé les temps d'ataque/relâchement du générateur d'enveloppe, et

[14] pas plus de 4 notes simultanées.

il reste donc à choisir sur quelles valeurs (en millisecondes) on veut que ces durées s'échelonnent. On place donc un objet [ctlin 1][15] qui contrôlera la durée d'attaque, un objet [ctlin 2] pour le relâchement, puis on les multiplie chacun par la durée maximale que l'on veut (qui peut être controlée aussi par control change, pourquoi pas...) : ici, 1 seconde pour l'attaque et 2 secondes pour le release.

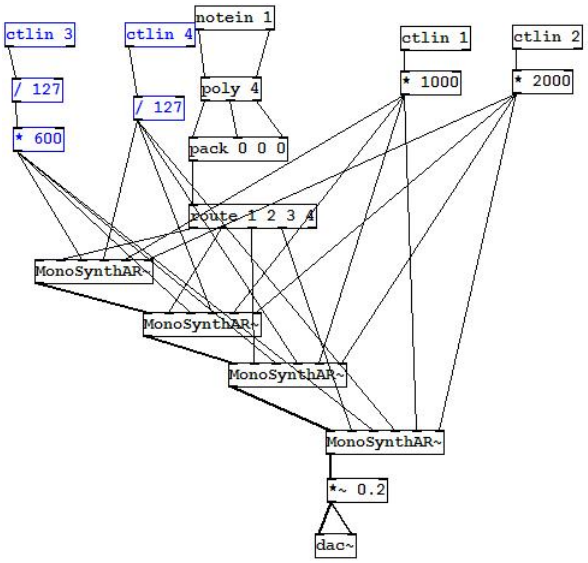
Enfin on n'a plus qu'à les envoyer sur les inlets appropriés de chaque [MonoSynth~](ici les deux plus à droite).



Il ne nous reste plus qu'à repeter la procédure pour la fréquence et l'amplitude de modulation (ici avec [ctlin 3] et [ctlin 4], mais attention ! on n'a pas normalisé les entrées. Il faut donc le faire soit dans l'objet[MonoSynth~](ne pas oublier de sauvegarder), sois sur les sorties des control change. De plus le contrôle de l'amplitude de modulation étant comprise entre 0 et 1 on n'a pas besoin de le remultiplier derrière. J'ai choisi une fréquence de modulation maximale de 600Hz.

Après avoir branché les contrôles de modulation votre patch devrait ressembler à ça :

[15] le 1 est le numéro de control change, vous pouvez en choisir un autre, pourvu qu'il corresponde au numéro associé au potentiomètre (ou à la courbe d'automation) choisi.



Voila, c'est terminé, le patch ressemble à présent à un amas de lignes dont on a du mal à trouver le début et la fin, mais que voulez-vous, ça fait partie des joies de Pure Data!! Amusez vous bien avec votre synthétiseur, et je vous rappelle qu'il y a une comunauté prête à vous aider dans la découverte de Pure Data sur le site **www.idecibel.com**.