6 Petit tutoriel à l'usage de notre panoplie dans Max

Cette section a pour but de présenter une mise en œuvre de notre panoplie dans l'environnement de programmation graphique Max. Nous avons structuré ce chapitre en différentes parties qui traitent de l'installation, de l'incorporation des modules dans l'environnement Max, de la connexion entre les différents modules et de la mise en relation de notre panoplie instrumentale avec des données externes quelconques. Nous terminons ce chapitre par deux exercices d'implémentation de notre panoplie. Le premier consiste à réaliser un dispositif d'accompagnement par imitation et le second est un exercice de génération musicale automatique sous contraintes.

6.1 Installation de la panoplie

Afin que Max incorpore les modules de notre panoplie à sa base d'outils, il est nécessaire de lui indiquer l'endroit où les fichiers de la panoplie sont enregistrés. Une option de Max permet de désigner un emplacement particulier dans l'arborescence d'un système de fichiers (cf. figure 50).

1. Ajoutez une entrée dans vos préférences de fichier

2. Sélectionnez le répertoire qui contient la panoplie

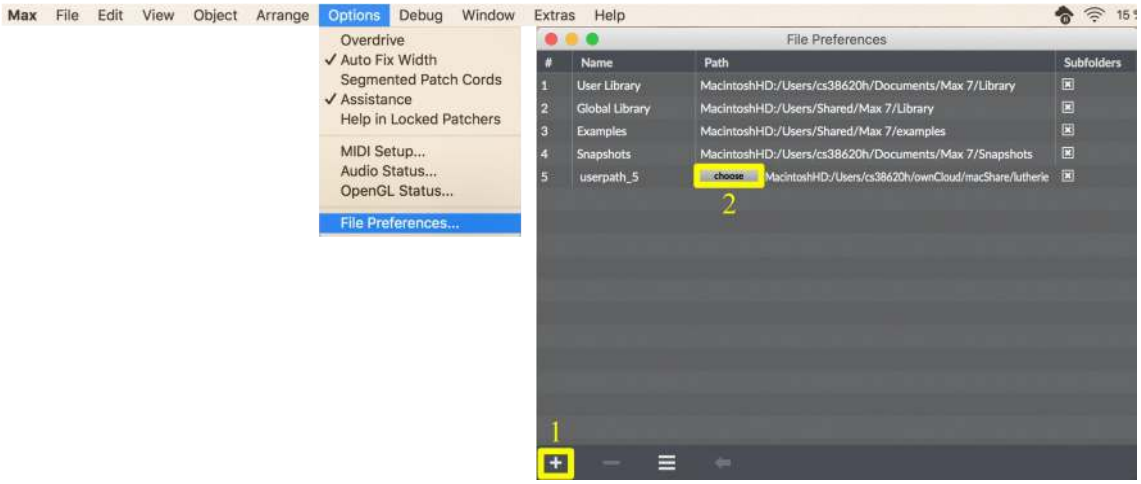

Figure 50: Localisation de la panoplie dans l'arborescence d'un système de fichiers

6.2 Incorporation d'un module à l'environnement Max

L'objet Max *bpatcher* est l'élément qui permet d'ajouter un module, de le visualiser et de le contrôler graphiquement. Il convient dans un premier temps d'invoquer l'objet Max (cf. figure 51), puis de le configurer pour incorporer un module.

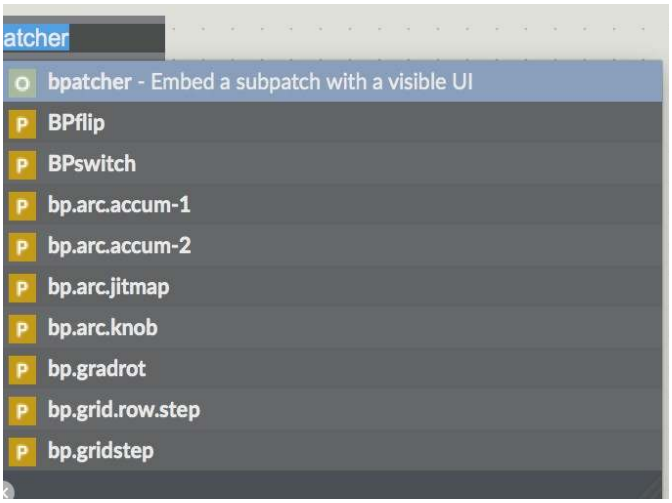

Figure 51: Invocation de l'objet Max bpatcher

L'inspecteur de Max est l'interface graphique pour configurer les options d'un objet et celles de l'objet *bpatcher* nous permettent de sélectionner un module et de lui attribuer un identifiant (cf. figure 52). Cet identifiant nous sert à différencier les modules entre eux et permet de les contrôler à l'aide des messages Max. Nous donnons plus de précision sur ce sujet à la section IV.6.5.

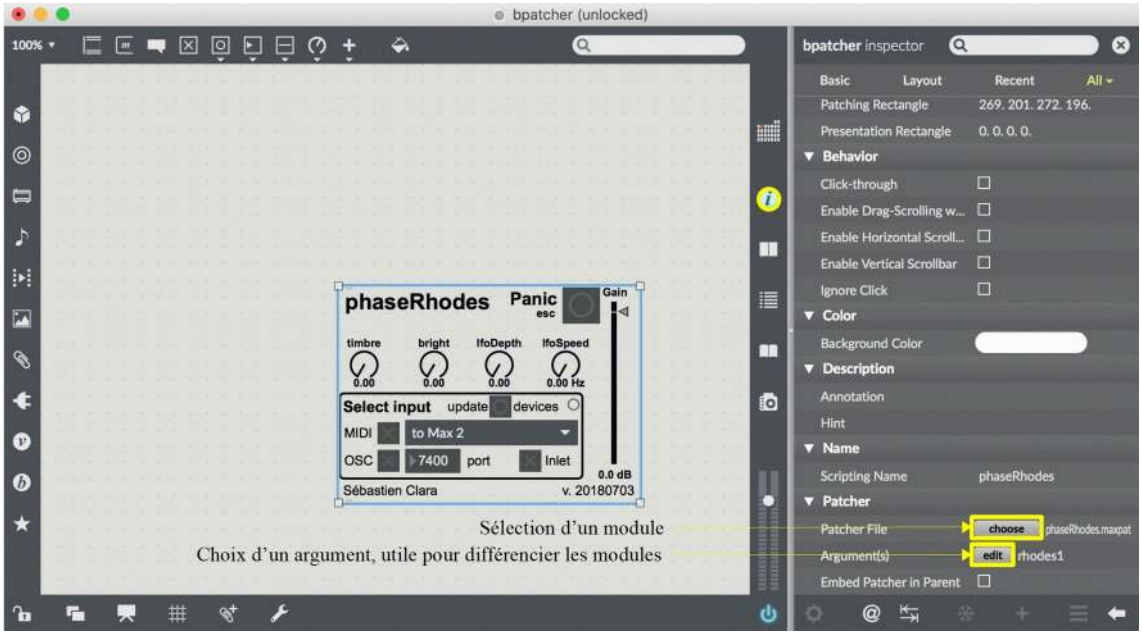

Figure 52: Configuration d'un module

6.3 Mise en relation des modules dans l'environnement Max

À la fin de notre chaîne de modules, le signal audio doit être connecté au convertisseur numérique/analogique de notre ordinateur. Pour ce faire, nous utilisons l'objet Max *ezdac~* qui se charge d'effectuer cette connexion (cf. figures 53 et 54).

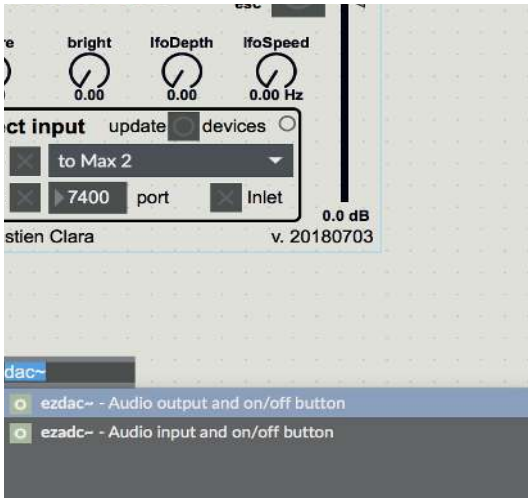

Figure 53: Connexion à la sortie audio de l'ordinateur

En cochant dans l'interface graphique les entrées et les sorties appropriées, nous pouvons connecter nos modules entre eux avec les liens Max standards et élaborer des montages séries ou/et parallèles pour nos modules d'augmentations, de synthèses et de traitements (cf. figure 54).

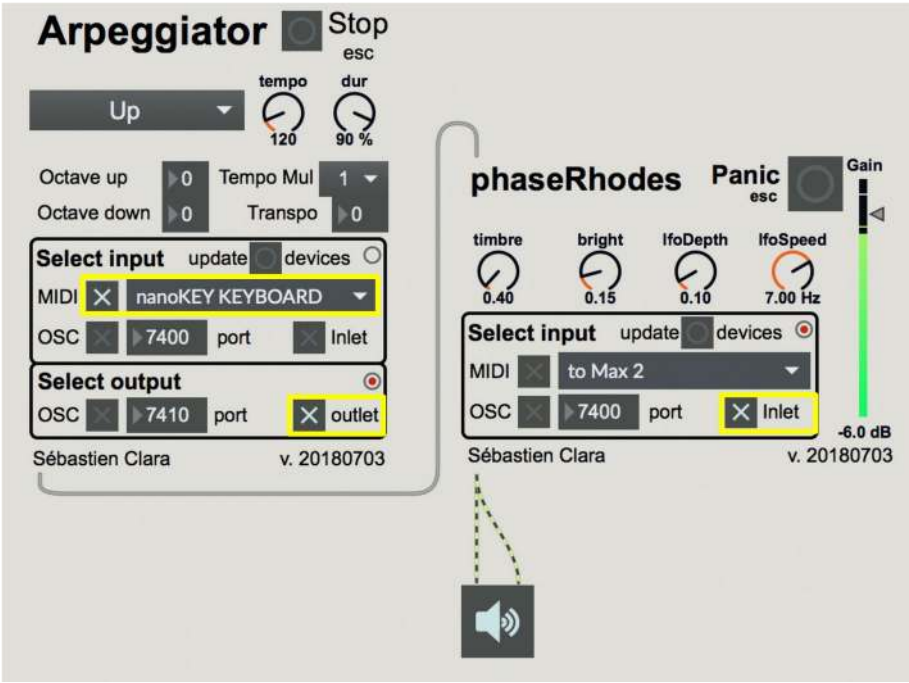

Figure 54: Mise en relation de deux modules dans l'environnement Max

6.4 Mise en relation des modules par OSC

Le standard de communication OSC utilise le réseau et plus particulièrement le protocole TCP/IP pour transmettre des données. Un dispositif complexe, distribué sur différentes machines et connectées au même réseau, peut communiquer par ce canal. Toutefois, un usage de Max est de répartir la charge des modules d'un dispositif sur les différents cœurs d'un processeur d'une machine. Les modules d'un dispositif sont empaquetés dans des applications autonomes Max[299] et ces dernières peuvent utiliser ce canal de communication pour échanger des informations. De plus, on peut tester facilement une séquence particulière de modules par ce canal. Pour cela, il est nécessaire de choisir un numéro de port de communication commun à la sortie d'un module et à l'entrée d'un autre (cf. figure 55). On peut ainsi séquencer ou/et paralléliser plusieurs modules. Pour un montage un peu plus complexe, on prendra garde de ne pas créer une boucle dans la transmission des informations.

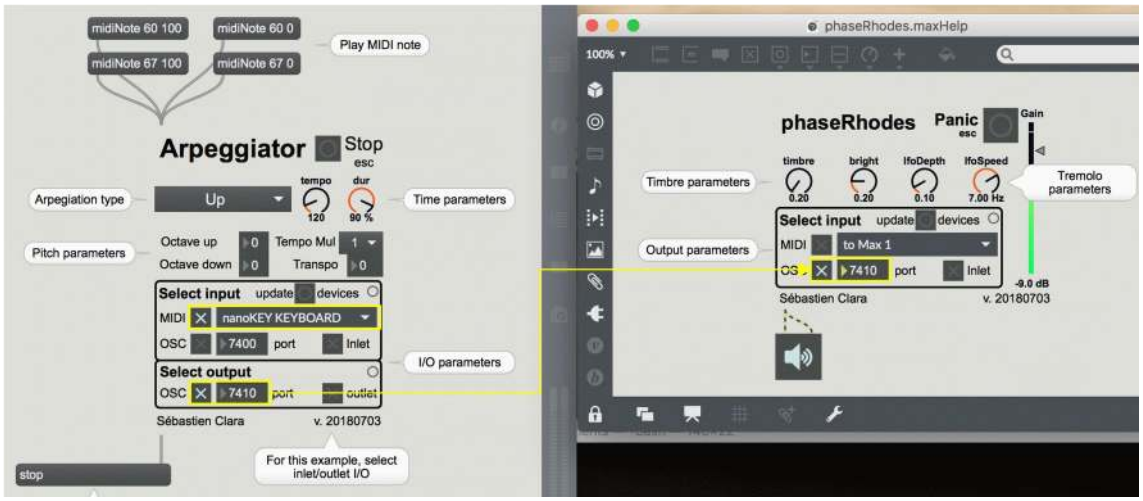

Figure 55: Les données générées par le module d'arpégiation sont transmises au module de synthèse par OSC par le port de communication 7410

6.5 Contrôle des modules par l'émission de message Max

Nous disposons de trois méthodes pour contrôler les paramètres d'un module. La première méthode consiste à utiliser l'interface graphique de Max. De plus, Max dispose d'objets spécifiques pour gérer les préréglages d'un programme et l'usage de ces éléments est la deuxième méthode de contrôle des modules. La dernière méthode

299 Nous avons constaté cette répartition de la charge applicative sur un processeur multicœur par l'usage de différentes applications autonomes Max dans la partie applicative de la pièce *Partita II* (2012) de Philippe Manoury.

consiste à utiliser les mécanismes de communication de Max. Dans cette section, nous traitons uniquement cette dernière méthode.

Nous avons référencé les paramètres de contrôle des modules par des étiquettes. Pour modifier un paramètre avec un message Max, il est nécessaire de se référer à la liste des étiquettes des modules disponible à l'annexe VI.9 et à l'identifiant passé en argument à l'objet *bpatcher*. Puis, il suffit de remplacer le préfixe #1 identique à toutes les étiquettes par le nom de l'identifiant choisi durant la configuration de l'objet *bpatcher* (cf. figure 52).

Nous donnons un exemple du contrôle d'un module par l'émission de message Max dans la section suivante (cf. le troisième point de la figure 56). Dans cet exemple, nous commandons les paramètres d'un module avec un contrôleur MIDI. Mais le principe de communication mise en œuvre dans cet exemple est similaire pour mettre en relation nos modules avec une autre source de contrôle comme le résultat généré en temps réel par un algorithme quelconque.

6.6 Commander un module à l'aide des messages de contrôle MIDI

Pour contrôler un module à l'aide de messages de contrôle MIDI, la première étape consiste à incorporer un module avec l'objet Max *bpatcher* (cf. figure 51). Nous configurons l'objet Max *bpatcher* en sélectionnant un module et en lui attribuant un identifiant à l'aide de l'inspecteur Max (cf. figure 52). Nous connectons le module à l'environnement de Max, notamment pour accéder au convertisseur numérique/analogique de notre système (cf. figure 53).

Puis, trois étapes sont nécessaires pour mettre en relation les données émises par un contrôleur MIDI avec les paramètres d'un module.

1. La première consiste à réceptionner les données MIDI et à les aiguiller correctement pour mettre en œuvre un plan de connexion particulier (cf. chapitre III.3).

2. La deuxième étape sert à mettre en relation les données MIDI aux paramètres du module et cette tâche dépend des données saisies, des paramètres de contrôle d'un module et d'un plan de connexion. Dans l'exemple de la figure 56, le type

de l'interface mise en place est point à point, c'est-à-dire que quatre potentiomètres contrôlent chacun un paramètre de synthèse.

3. Enfin, on ajoute au nom de l'étiquette d'un paramètre l'identifiant spécifié dans l'objet Max *bpatcher*, pour que les messages Max atteignent la bonne cible comme nous l'avons spécifié dans la section précédente. Par exemple avec l'étiquette *#1-modIndex* couplée à l'identifiant *rhodes1*, nous obtenons l'adresse d'envoi : *rhodes1-modIndex* (cf. figure 56).

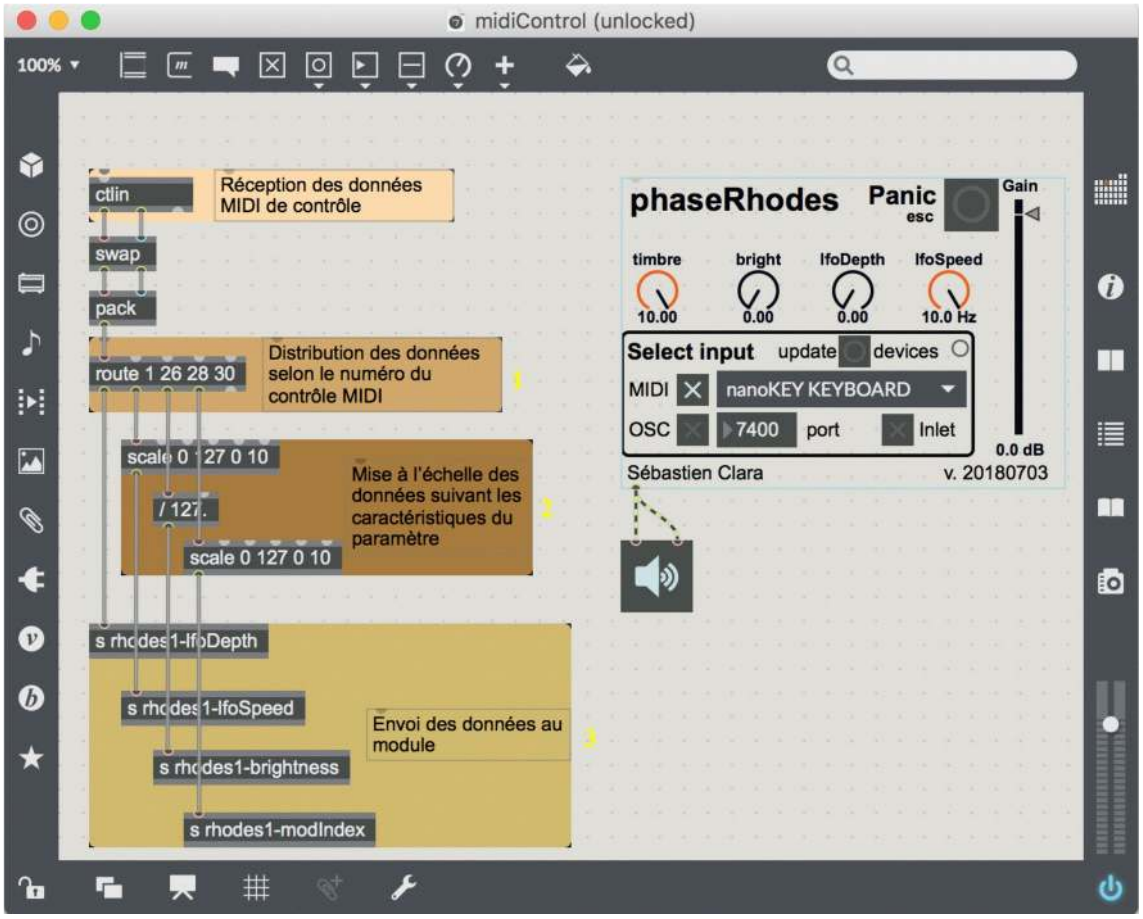

Figure 56: Mise en relation de données de contrôle MIDI à un module de synthèse

Dans les deux prochaines sections, nous réalisons deux exercices avec notre panoplie.

6.7 Réalisation d'un accompagnement par imitation

Dans cet exemple, nous souhaitons utiliser notre module de synthèse concaténative (cf. section IV.4.5) et lui ajouter un accompagnement par imitation. Pour ce faire, nous allons utiliser notre module d'imitation (cf. section IV.3.4), notre module d'ajustement automatique des notes MIDI (cf. section IV.3.3) et un synthétiseur de notre panoplie.

Le module *uc33concat.maxpat* est un sous-module de synthèse concaténative et il dispose de trois sorties. La première sortie est la sortie audio de la piste de sommation finale (*master*) du sous-module. La deuxième est une sortie audio, mais un potentiomètre rotatif permet de router une des neuf pistes du sous-module vers cette sortie (potentiomètre *channel*). La dernière sortie est une sortie de contrôle. Les différentes instances de la synthèse concaténative sont gérées par un tempo et cette information est transmise par cette dernière sortie.

Pour mettre en relation le module de synthèse concaténative à d'autres modules de notre panoplie, nous utilisons le module d'interface *audio2midiPolyphonic.maxpat* pour convertir du son en notes MIDI. Nous pouvons relier la première ou la deuxième sortie du module *uc33concat.maxpat* à cette interface de conversion. Nous choisissons la deuxième, car nous pouvons transformer une piste audio en piste de contrôle. Si nous diminuons le volume sonore de cette piste, nous ne l'entendons plus, mais le son produit par cette piste est toujours actif sur la deuxième sortie du module *uc33concat.maxpat* et donc, nous pouvons l'employer pour alimenter notre module de conversion des sons en notes MIDI (cf. figure 57).

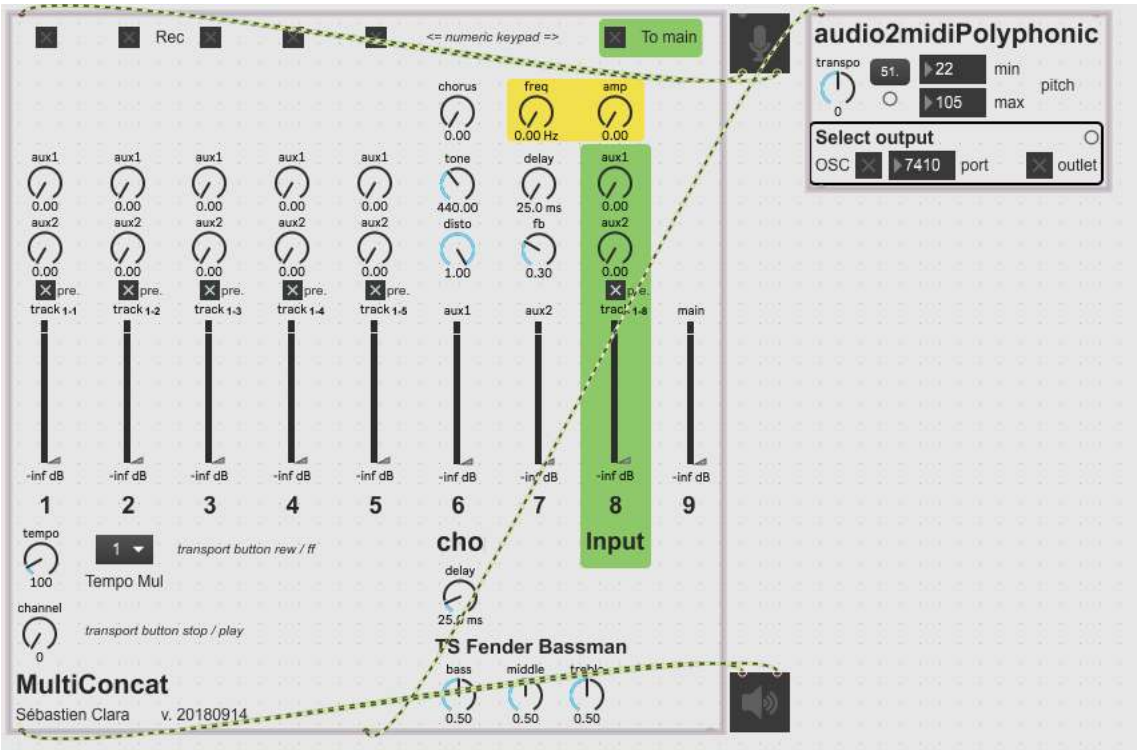

Figure 57: Accompagnement par imitation : conversion des sons en notes MIDI

À la suite du module d'interface *audio2midiPolyphonic.maxpat,* nous pouvons ajouter en série notre module d'imitation, notre module d'ajustement automatique des notes MIDI et un synthétiseur (cf. figure 58).

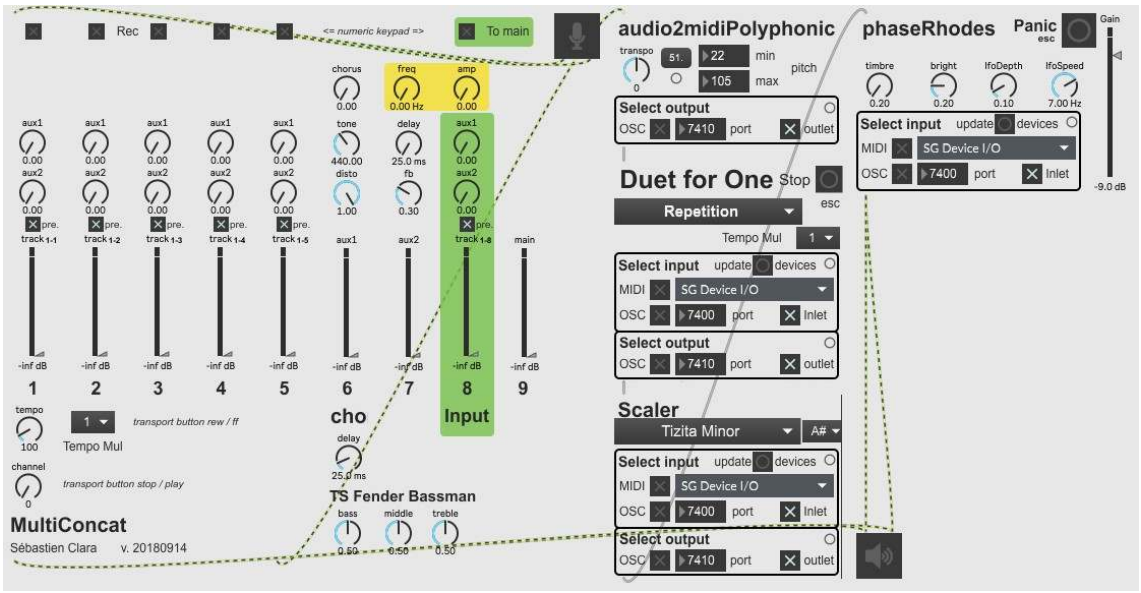

Figure 58: Accompagnement par imitation : implémentation du module d'imitation, du module d'ajustement automatique des notes MIDI et d'un synthétiseur

Nous avons atteint l'objectif de cet exercice en réalisant un dispositif d'accompagnement par imitation de notre module de synthèse concaténative, mais nous poursuivons cet exercice afin de lui apporter d'autres fonctionnalités. Dans cet exercice, nous ajustons les notes MIDI à un mode. Nous allons faire de même avec la première sortie audio du module de synthèse concaténative et aligner l'audio et le MIDI à un mode commun.

L'objet Max *retune~* peut être utilisé pour ajuster automatiquement les notes audio à un mode. Pour le contrôler et l'aligner avec notre module d'ajustement automatique des notes MIDI, nous utilisons le sous-module dictionnaire des modes. En ajoutant le mot clé *enablednotes* au résultat de la cinquième sortie du sous-module dictionnaire des modes, nous configurons l'objet *retune~* à un mode. La sixième sortie du sous-module dictionnaire retourne une liste. La première valeur de cette liste correspond à une échelle et la seconde valeur à une tonique. Nous utilisons l'objet Max *unpack* pour segmenter cette liste en deux valeurs. Nous pouvons alors adresser ces valeurs au module d'ajustement automatique des notes MIDI et aligner les deux objets d'ajustement des notes par une action. L'étiquette *#1-selectScale* permet de configurer

l'échelle et l'étiquette *#1-scaleRoot*, la tonique du mode (cf. section IV.6.5 et le figure 59).

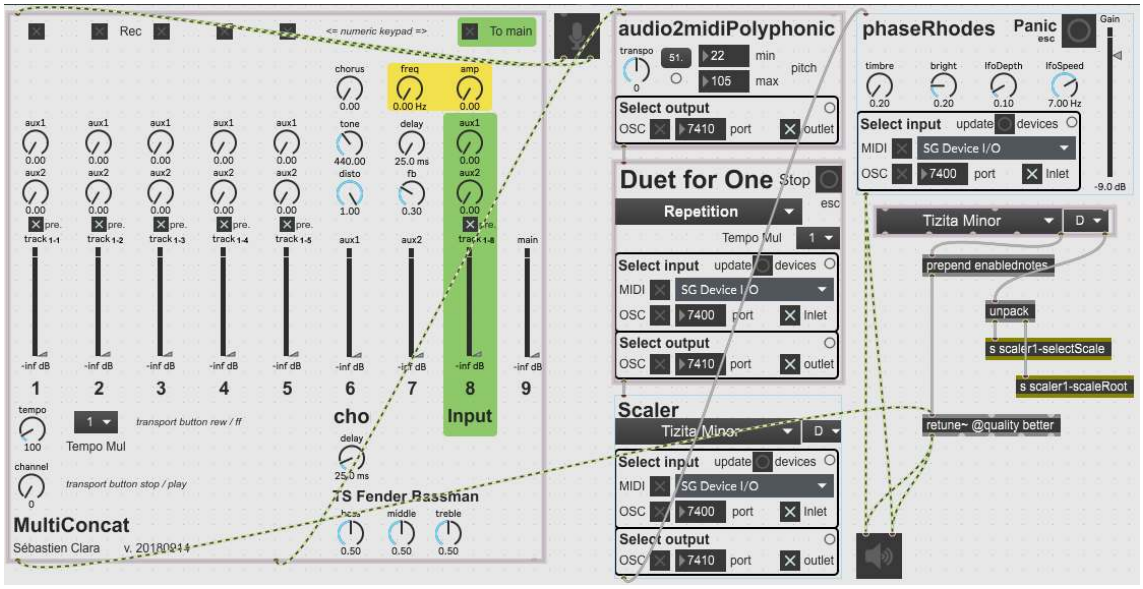

Figure 59: Accompagnement par imitation : alignement des modules d'ajustement automatique des notes MIDI et de l'audio

Après avoir homogénéisé les hauteurs audio et MIDI à un mode, nous allons faire de même avec le timbre des deux sources sonores. Pour ce faire, nous séquençons un compresseur pour homogénéiser la dynamique des deux sources sonores (cf. section III.4.3) et une réverbération pour donner un espace commun à ces dernières (cf. section III.4.2.1 et la figure 60).

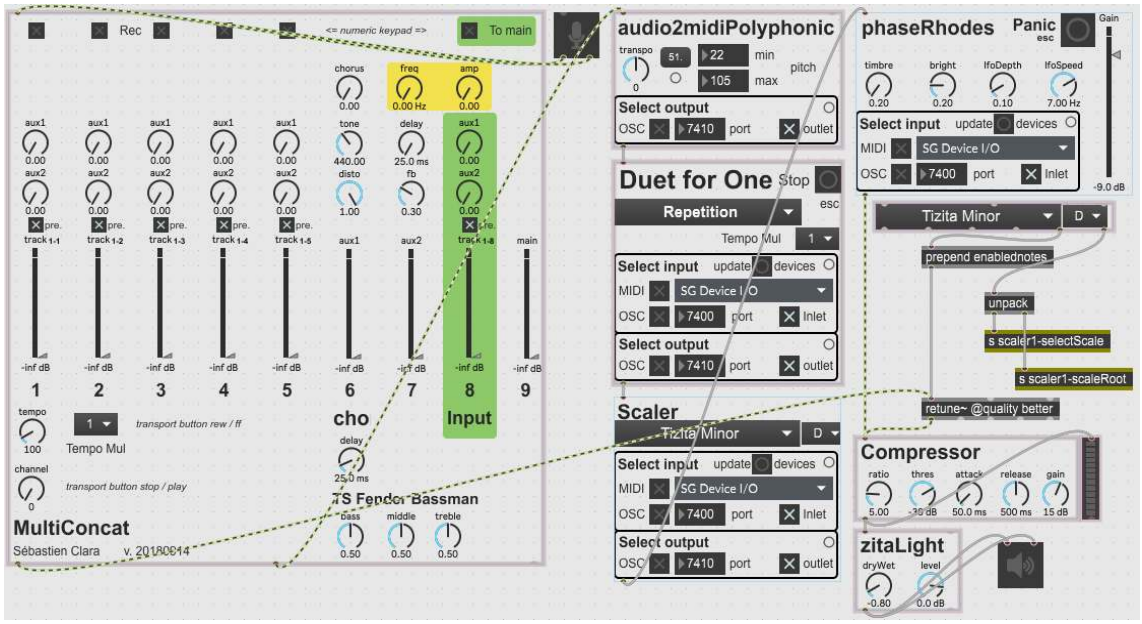

Figure 60: Accompagnement par imitation : ajout de traitements audio

Dans le répertoire des exemples de notre panoplie, nous avons ajouté le programme *voixUC33.maxpat* qui reprend le principe de l'exemple que nous venons de traiter dans cette section.

6.8 Musique générative sous contraintes

Pour cet exercice d'implémentation de notre panoplie, nous souhaitons générer des notes aléatoirement selon différentes contraintes. Pour faciliter les explications, nous nous contenterons d'utiliser comme capteur la souris de l'ordinateur. Toutefois, les programmes que nous nous apprêtons à développer peuvent être utilisés avec d'autres types de capteurs.

Pour développer cet exercice, nous choisissons les contraintes suivantes :

1. Les notes générées doivent dépendre d'une base que nous appelons tonique.
2. Selon la tonique choisie, les notes générées doivent être en correspondance avec la qualité d'un accord ou d'une échelle.
3. Les notes doivent être générées selon un tempo.
4. Un pourcentage de silence doit interrompre le flux mélodique.
5. Le programme que nous développons doit interagir avec le reste de notre panoplie.
6. La vélocité des notes doit être générée aléatoirement.

Pour réaliser cet exercice, nous employons les modules de notre panoplie suivants :

a) Dictionnaire d'accords (cf. IV.3.2)

b) Dictionnaire de modes (cf. IV.3.3)

c) Gestionnaire de sorties des modules (cf. IV.1.5)

Contrainte 1 : Sélection d'une tonique

Pour sélectionner une tonique, nous avons deux possibilités. Nous pouvons régler la tonique par sa note MIDI, ou bien sur une portée à l'aide de l'objet Max *nslider* (cf. figure 61).

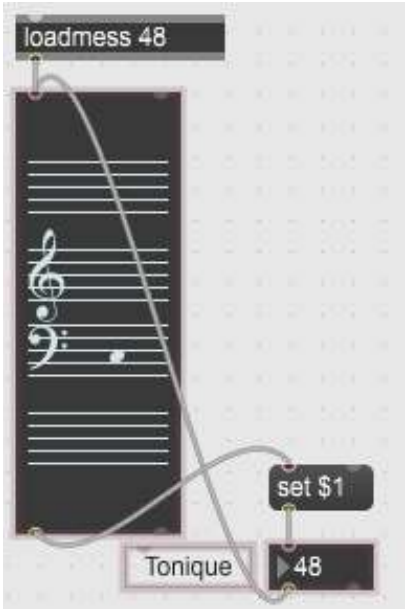

Figure 61: Musique générative : sélection d'une tonique

Contrainte 2 : Sélection d'une échelle ou d'une qualité d'accord

Pour contraindre les notes générées à un accord, nous utilisons le sous-module *chord.maxpat* et pour contraindre les notes générées à une échelle, nous utilisons le sous-module *scale.maxpat*. Ces deux sous-modules sont disponibles dans notre panoplie et nous les incluons dans notre programme avec l'objet Max *bpatcher* (cf. section IV.6.2).

Pour choisir une note aléatoirement dans la contrainte que nous nous sommes fixés, nous utilisons l'objet Max *random*. Cet objet Max génère un nombre aléatoire compris entre 0 et un nombre spécifié en argument moins un. Par exemple, quand l'objet *random 7* est stimulé, il retourne un nombre entier aléatoire compris entre 0 et 6.

Cependant, les sous-modules *chord.maxpat* et *scale.maxpat* retournent des listes. Par exemple, lorsque nous sélectionnons un accord majeur, le sous-module *chord.maxpat* retourne la liste [0, 4, 7]. Pour sélectionner un élément aléatoirement de cette liste avec l'objet Max *random*, nous devons connaître la taille de la liste, puis sélectionner l'élément de cette liste choisi aléatoirement. L'objet Max *zl* initialisé avec l'argument

len retourne la taille d'une liste et l'objet Max *zl* initialisé avec l'argument *lookup* sélectionne un élément d'une liste.

Pour générer une note aléatoire contrainte selon la qualité d'un accord ou d'une échelle, il nous reste à additionner l'élément précédent sélectionné aléatoirement à la tonique que nous avons fixée dans notre première contrainte (cf. figure 62).

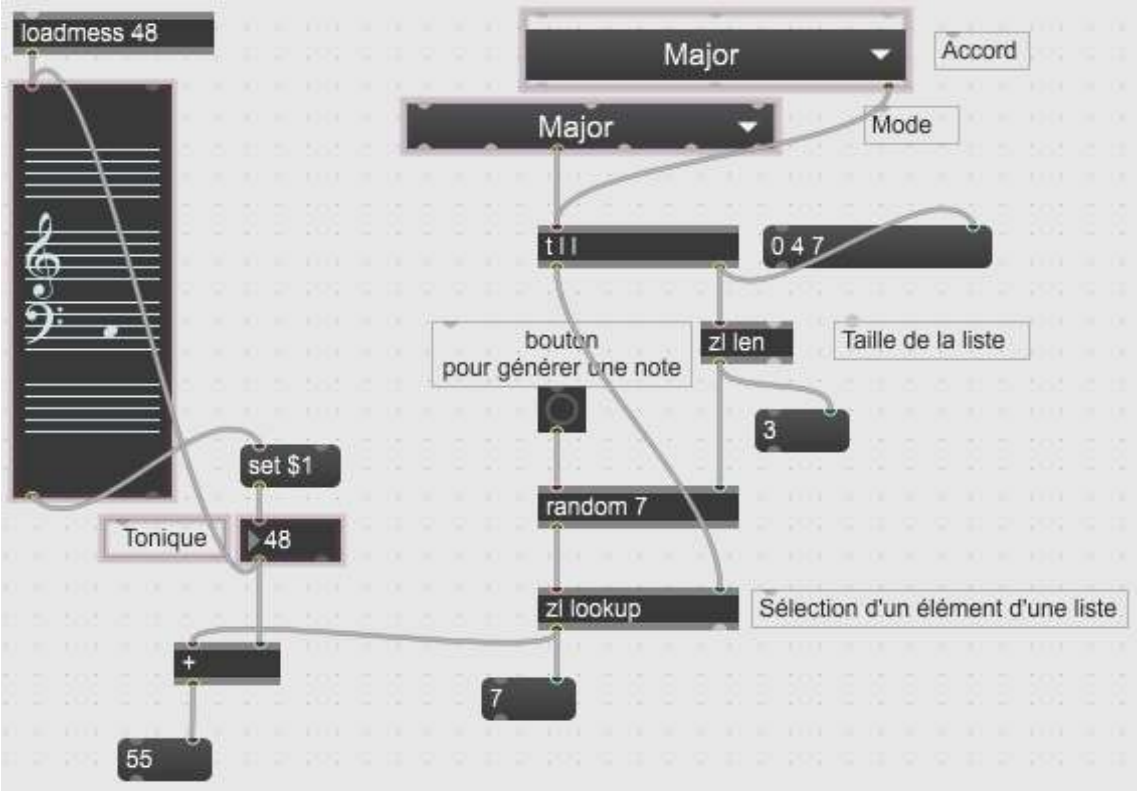

Figure 62: Musique générative : génération d'une note selon une contrainte

Le programme que nous avons développé dans cette section, permet de générer une note manuellement. Dans la section suivante, nous implantons une fonctionnalité pour générer des notes automatiquement.

Contrainte 3 : Génération de notes selon un tempo

L'objet Max *metro* génère des impulsions à intervalle régulier. Avec cet objet, l'intervalle entre chaque impulsion est déterminé en milliseconde. En musique, nous utilisons généralement la notion de tempo ou de bpm pour déterminer des impulsions à intervalle régulier. Pour obtenir une durée en milliseconde, nous devons donc diviser à 60000 un tempo que nous déterminons avec un potentiomètre rotatif.

Le déclenchement ou l'arrêt de l'objet *metro* est contrôlable. Pour cela, nous ajoutons une boîte à cocher (cf. figure 63).

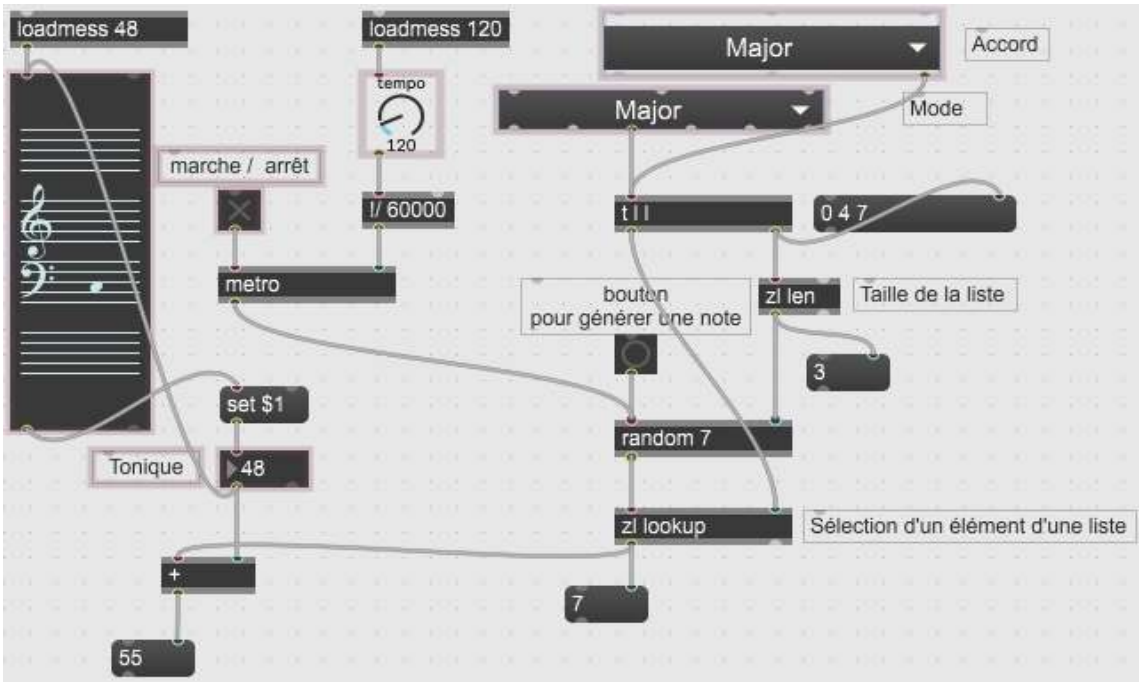

Figure 63: Musique générative : génération des notes selon un tempo

Le programme que nous avons développé dans cette section, génère un flux mélodique constant à intervalle régulier. Pour un auditeur, ce flux mélodique peut devenir monotone. Dans la section suivante, nous implantons une fonctionnalité pour introduire des silences.

Contrainte 4 : Génération aléatoire de silences

Pour introduire des silences dans le flux mélodique généré avec le programme de la section précédente, nous utilisons l'objet Max *gate*. Avec cet objet, nous créons une rétention d'information qui aboutit dans notre cas à la création d'un silence.

Pour générer des silences aléatoirement, nous utilisons l'objet Max *random*. Pour déterminer le taux de silence par un pourcentage, nous initialisons l'objet *random* avec l'argument 101. Chaque stimulation de l'objet *random* nous retourne un nombre entier compris entre 0 et 100. Nous devons comparer ce résultat généré aléatoirement au pourcentage de silence que nous sélectionnons avec un potentiomètre rotatif. Le résultat de cette comparaison nous sert à contrôler l'objet Max *gate* et donc à générer des silences selon un pourcentage.

Nous installons cet algorithme entre l'objet Max *metro* qui génère des impulsions à intervalle régulier et l'objet *random* qui nous permet de sélection un élément d'une liste d'un accord ou d'une échelle (cf. figure 64).

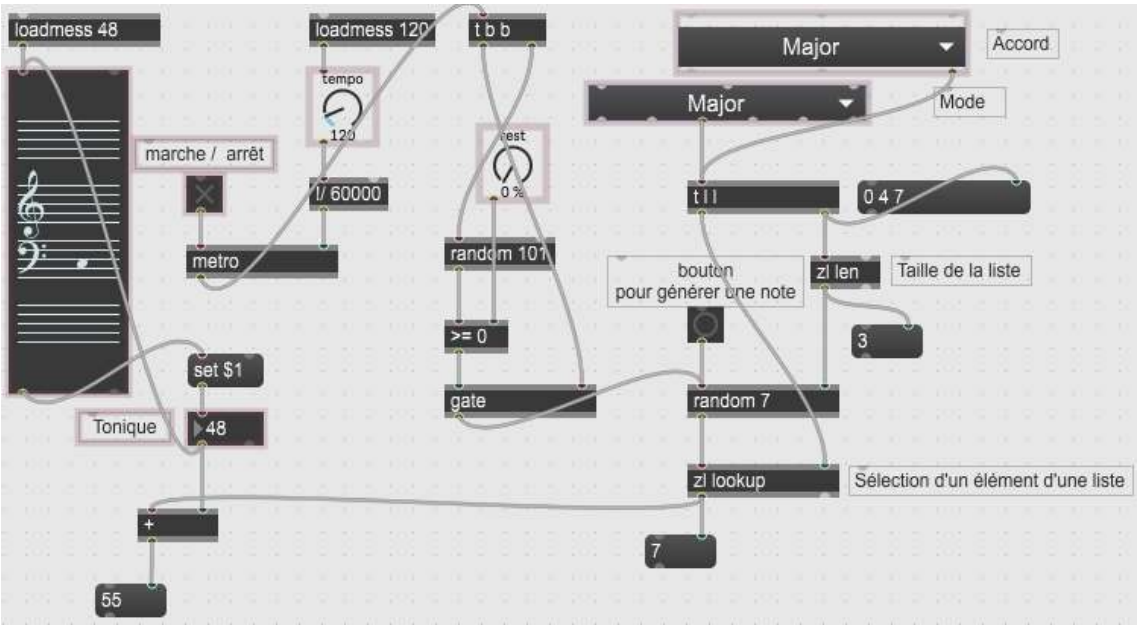

Figure 64: Musique générative : génération aléatoire de silences

Contrainte 5 : Intégration du flot mélodique à notre panoplie

À la section IV.1.5, nous avons traité la communication entre les modules de notre panoplie et notamment, du gestionnaire des sorties des modules et du mot clé *midiNote*.

Nous incluons le gestionnaire des sorties, le sous-module *selOutput.maxpat*, à notre programme avec l'objet Max *bpatcher* (cf. section IV.6.2).

Pour simplifier la gestion des notes MIDI, nous utilisons l'objet Max *makenote*. Ce dernier objet demande une hauteur, une vélocité et une durée pour déclencher une note MIDI et pour l'arrêter. Durant les précédentes sections de cet exercice, nous avons généré des notes aléatoirement à un tempo donné. Nous utilisons ces deux données pour paramétrer l'objet *makenote* et pour le moment, nous fixons la vélocité des notes à 100.

Pour formater un message pour un autre module de notre panoplie, nous concaténons le résultat des deux sorties de l'objet *makenote* avec l'objet Max *pack* et nous ajoutons le mot clé *midiNote* à ce dernier message avec l'objet Max *prepend* (cf. figure 65).

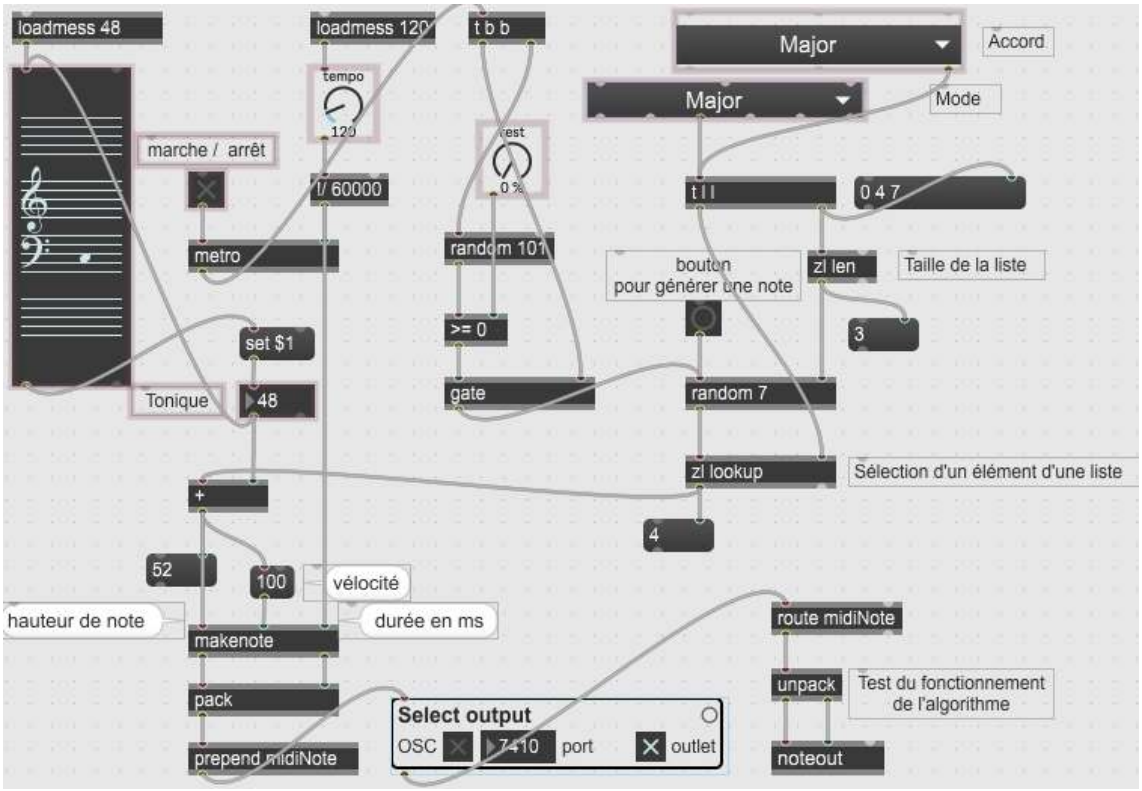

Figure 65: Musique générative : intégration du programme à notre panoplie

Dans la prochaine section, nous intégrons la génération aléatoire des vélocités des notes MIDI.

Contrainte 6 : Génération aléatoire de vélocités

Les données MIDI varient entre 0 et 127. Dans Max, une vélocité égale à 0 indique la fin d'une note. Par conséquent, nous pouvons faire varier la vélocité d'une note dans un intervalle compris entre 1 et 127. Pour ce faire, nous utilisons l'objet Max *random* et nous l'initialisons avec l'argument 127. Le résultat de ce dernier objet est compris entre 0 et 126. Nous devons ajouter un à ce dernier résultat afin d'obtenir un intervalle compris entre 1 et 127.

Avec ce dernier algorithme, la dynamique des notes générées est élevée. Si nous désirons réduire cette dynamique et fixer un seuil à l'aide d'un potentiomètre rotatif, en dessous duquel les vélocités générées aléatoirement ne descendront pas, nous reprenons l'algorithme précédent, mais nous réduisons la valeur de l'argument de l'objet *random* et nous additionnons au résultat de l'objet *random* une valeur proportionnelle à cette réduction (cf. figure 66).

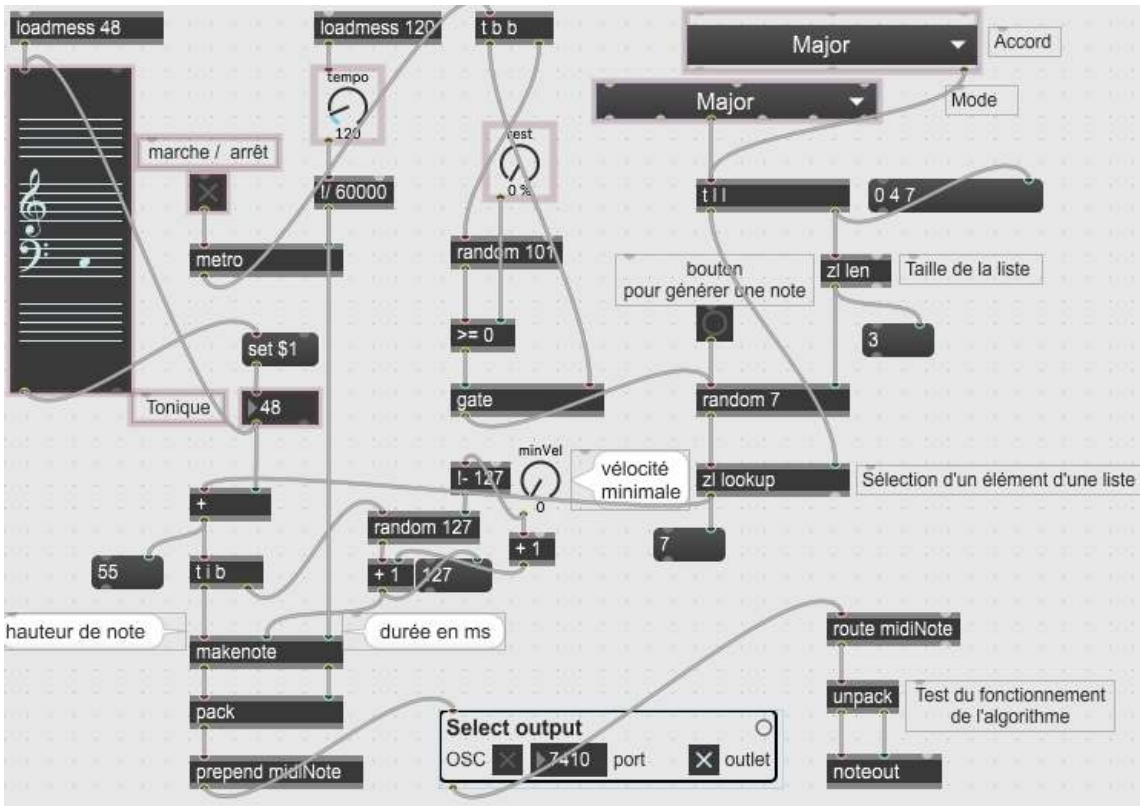

Figure 66: Musique générative : gestion des vélocités

Dans le répertoire des exemples de notre panoplie, nous avons intégré le sous-module *melodyFlow.maxpat* qui reprend le principe et les contraintes que nous venons de traiter dans cette section.

Après la présentation de différents types de modules et de la mise en œuvre pratique de notre panoplie instrumentale dans l'environnement de programmation graphique Max, nous décrivons dans le chapitre suivant la pièce et son dispositif instrumental que nous avons réalisés pour valider artistiquement nos développements.