32-Traitement de liste

introduction

Ce tutoriel est volumineux, car il couvre un sujet important: ***le traitement de liste***. Nous couvrons plusieurs des modes disponibles de l'objet *zl*, qui fournit un centre d'échange de fonctions de traitement de listes. Nous verrons également comment effectuer des expressions mathématiques sur des listes avec l'objet *vexpr* et comment utiliser l'objet *prob* pour créer une table de probabilités pour la création musicale.

Dans Max, la ***liste*** est l'une des structures de données les plus puissantes qui soient. Vous pouvez définir n'importe quelle combinaison de valeurs dans une liste, puis les faire envoyer sous forme de messages, les faire traiter par des objets ou les traiter comme de petites tables. Cela est en grande partie rendu possible par l'objet *zl*, qui vous donne la possibilité d'interroger, de réorganiser et d'accéder à n'importe quel élément d'une liste. L'ajout de l'objet *vexpr* au traitement des listes vous permet de traiter les éléments d'une liste de manière mathématique sans avoir à les décomposer en leurs composants individuels.

L'utilisation de l'objet *prob* fait partie intégrante de nombreux programmes génératifs. L'objet *prob* vous permet de créer facilement une table de probabilité pondérée, avec seulement un **bang** nécessaire pour générer les données attendues. Dans notre tutoriel, nous verrons comment l'objet *prob* est utilisé pour initialiser une application de séquençage de base.

Pour ouvrir le patch du didacticiel, cliquez sur le bouton vert **Ouvrir didacticiel** dans le coin supérieur droit de la fenêtre de documentation.

Exécuter le patch de séquençage

Jetez un coup d'œil à notre tutoriel. C'est le patch Max par excellence - le séquenceur pas à pas. Lorsque vous ouvrez le patch, les deux premiers objets *multislider* (étiquetés **A** et **B**) sont ensemencés de valeurs, tandis que le troisième (étiqueté **C**) est vide. Ces objets contiennent des séquences pour un système de lecture MIDI alimenté par les deux objets *noteout* situés à leur droite. Double-cliquez sur les objets *noteout* pour sélectionner un synthétiseur disponible. Activez le *metro* en haut à gauche en cliquant sur le *toggle* et la séquence commencera à être lue, comme pilotée par le contenu des *multisliders* A et B. Cliquez sur le *multislider* C et faites glisser votre souris pour définir les valeurs de la ligne mélodique. Le résultat sera lu à l'aide du synthétiseur choisi.

Si vous utilisez un synthétiseur compatible General MIDI, vous devez entendre les deux séquences supérieures du *multislider* sous forme de batterie et la séquence en bas (**C**) comme une mélodie jouée sur un piano. En effet, selon les règles General MIDI, le canal MIDI **10** (l'argument de la sortie *noteout* supérieure) est le canal de ***batterie***, tandis que les quinze autres canaux jouent des instruments mélodiques et utilisent généralement des pianos par défaut. De nombreux synthétiseurs nécessitent que vous configuriez ce fonctionnement, mais les synthétiseurs intégrés sur les ordinateurs Macintosh et Windows se comportent de cette manière par défaut.

À droite du patch, se trouvent un certain nombre de routines d'édition qui utilisent l'objet *zl*, que nous examinerons en détail dans une minute. Vous pouvez faire tourner la séquence stockée dans le *multislider* **A**, inverser le contenu du *multislider B* et effectuer plusieurs fonctions différentes sur la mélodie contenue dans le *multislider* **C**. Cliquez sur les objets *button* qui activent les routines d'édition et vérifiez que les modifications sont immédiatement appliquées au contenu du *multislider*.

La fonction de base du séquenceur pas à pas doit vous être familière: le *metro* pilote un *counter* à **32** positions dont la sortie est transformée en un message **fetch** pour les trois objets *multislider*. Les sorties des *multisliders* sont envoyées à deux combinaisons différentes de *makenote* / *noteout ;* les deux *multisliders* de batterie sont envoyés à un objet *noteout* assigné au canal MIDI **10**, tandis que la mélodie est envoyée à un objet *noteout* affecté au canal MIDI **1**.

Les deux canaux de batterie présentent une méthode simplifiée de sélection de batterie: leurs objets *multisliders* sont limités à trois valeurs chacun. Les objets de *sélection* utilisés avec ces objets sont un moyen de remapper le contenu de *multislider* (**0**, **1** ou **2**) sur un instrument (aucun, kick, / hi-hat fermé et hi-hat caisse claire / ouverte, respectivement). De cette manière, la gamme complète du *multislider* est significative et la valeur de sortie est facile à utiliser.

Mettez toute la séquence dans les objets des *multisliders* **A** et **B** à **0** (faites glisser la souris sur eux le long du bas). La batterie dans la séquence s'arrête. Essayez d'ajouter différents motifs pour avoir une idée de la manière dont les nombres dans l'objet correspondent aux différents sons que vous obtenez.

Travailler avec *prob* (probabilités)

- Double-cliquez sur le subpatch **Generate_ptns** pour l'ouvrir.

La configuration initiale des séquences de batterie est effectuée à l'aide de la section **generate new patterns** du patch, située sous les objets *multisliders*. Si vous voulez le voir en action, cliquez sur le *button* connecté à l'objet *uzi* des hi-hats. Vous verrez que le *multislider* va être brouillé, mais que le contenu tend à mettre l'accent sur la valeur **1** (qui correspond au charleston fermé). Cela se produit à cause de la table de probabilités mise en place et appliquée par l'objet *prob*.

La configuration de la table de probabilités se fait par le biais du *message* volumineux déclenché à l'ouverture du patcheur par le *loadbang*. La façon dont notre table de probabilités fonctionne est basé sur les transitions - celles-ci sont définies par trois listes de trois éléments contenant la valeur actuelle, une autre valeur et une valeur de pondération, utilisée pour déterminer la probabilité d'une transition de la valeur actuelle à une autre valeur. Ainsi, par exemple, **0 1 2** signifie que la pondération appliquée à une transition de **0** à **1** est de **2**. Que signifie **2** ? Il n'a pas vraiment de signification spécifique - il a seulement une valeur définie par rapport à toutes les autres pondérations pour cette transition.

La boîte de *messages* utilisée pour les tables de probabilités du charleston définie toutes les transitions possibles dans un seul ensemble de trois valeurs (**0,1,2**). Étant donné que vous ne pouvez vous trouver qu'à un seul endroit à la fois, les probabilités dépendent de la valeur actuelle. Examinons les calculs de probabilité si l'étape actuelle est **0**:
Étape suivante: **0**; Pondération: **3**; Probabilité: 3/8 (37.5%)
Étape suivante: **1**; Pondération: **3**; Probabilité: 3/8 (37.5%)
Étape suivante: **2**; Pondération: **2**; Probabilité: 2/8 (25,0%)

Afin de comprendre le calcul de la valeur de pondération, vous devez additionner toutes les pondérations de l'étape actuelle, puis utiliser cette somme pour diviser les valeurs individuelles d'un facteur de pondération. Dans cet exemple, le total des valeurs de pondération pour tous les cas où l'étape actuelle est **0** s'élève 8. Cela nous donne le dénominateur des calculs que nous avons effectués. Par conséquent, 25% du temps, nous passerons d'un son de charleston inexistant à un charley ouvert; 37,5% du temps, nous allons passer à un charley fermé et le reste du temps, il n'y aura pas de son du tout.

Si vous examinez les cas où le pas actuel est **2** (un charleston ouvert), vous verrez que les transitions vers **0** et **1** sont toutes deux pondérés à **4**, tandis que la transition vers **2** (aucun changement) se voit attribuer un poids de **0**. Cela signifie que la table de l'objet *prob* empêchera la répétition d'un charleston ouvert, car il y a 0% de chance qu'une valeur actuelle de **2** soit suivie d'un **2**.

L'objet *prob* est piloté par un objet *uzi*, configuré pour produire **32** messages **bang**. Cependant, les sorties de *prob* sont des valeurs individuelles et le *multislider* attend une liste de 32 valeurs pour la configuration. Comment combiner efficacement tous ces messages ?

Travailler avec *zl*

- Double-cliquez sur le subpatch **Process_Lists** pour l'ouvrir.

L'objet *zl* est un objet unique qui fonctionne dans plus de 20 *modes* différents, définis par le premier argument de l'objet. Les modes exécutent des fonctions de traitement de liste, telles que le tri, la suppression des éléments indésirables et l'assemblage des listes. Dans le cas des générateurs de séquence pour les batteries, l'objet *zl* est utilisé en mode **groupe**, avec un argument de **32**. Cela signifie que l'objet va collecter **32** valeurs, les regrouper dans une seule liste, puis sortir la liste par sa sortie gauche. IC'est un moyen rapide et très efficace de regrouper un flux de données dans une liste prédéfinie et de transformer une tâche autrement fastidieuse en une tâche pour un seul objet.

Le patch du didacticiel montre de nombreuses utilisations de l'objet *zl* - en particulier dans les fonctions d'édition du côté droit. Toutes ces routines d'édition commencent par l'objet *zl* en mode **reg**. Dans ce mode, l'objet *zl* acceptera une liste dans son entrée droite et le stockera jusqu'à ce qu'il reçoive un **bang** dans l'entrée gauche. L'objet *zl* **reg** est, en substance, l'équivalent des objets *int*, *float* ou *value*, mais est conçu pour le stockage de liste.

Lorsque vous cliquez sur le *button* pour chaque routine d'édition, la liste de *zl* **reg** est exportée vers un autre objet de traitement de liste. Plusieurs des routines d'édition utilisent *zl* dans un autre mode pour la manipulation de liste; par exemple, la routine "rotate A" utilise *zl* en mode **rot** avec un argument de **1** pour faire ***pivoter*** les valeurs d'une entrée - tout se déplace d'une position vers la droite et la dernière entrée devient la première. La routine «reverse B» utilise *zl* **rev** pour ***inverser*** l'ordre d'une liste entrante, tandis que la «sort parts of C» combine trois objets *zl* pour ***scinder*** la liste en morceaux (**iter**), trier chacun d'eux et les ré-assembler (**group**) en une liste. La plupart des modes *zl* prennent un deuxième argument ou permettent à un nombre situé dans l'entrée droite de changer un paramètre - par exemple, l'objet *zl* **iter** décompose une liste en sous-listes d'une taille définie par l'argument (dans notre cas, **4**) ce qui peut également être modifié en envoyant un nombre dans l'entrée droite. Si nous modifions la boîte de *nombres* connectée à l'objet *zl* **iter** à **32** (la longueur totale de la séquence), puis que nous cliquons ensuite sur le *button*, la mélodie stockée dans le *multislider* **C** sera triée des valeurs basses aux valeurs hautes.

La meilleure façon de voir tous les modes de fonctionnement *zl* est peut-être de consulter le fichier d'aide *zl*. Déverrouillez le patcheur, sélectionnez un objet *zl* et choisissez «Ouvrir l'aide de *zl*» dans le menu contextuel ou dans le menu d'aide de l'application. Le fichier d'aide est divisé en six onglets, car il existe de nombreux modes, mais vous pourrez voir toutes les façons dont *zl* peut être utilisé pour plier, replier et mutiler des listes Max.

Utilisation de vexpr

Lorsque vous travaillez avec des listes, vous souhaitez parfois utiliser des expressions mathématiques pour tous les éléments d'une liste. L'utilisation d'objets math standard de Max *(+,* etc.) nécessiterait de diviser, traiter et réassembler une liste - ce n'est pas la manière la plus efficace

de traiter le problème. Il existe une solution: l'objet *vexpr*.

Comme son nom l'indique, l'objet *vexpr* est très similaire à l'objet *expr* que nous avons appris précédemment, à l'exception du fait qu'il est conçu spécifiquement pour traiter des listes. Si nous examinons l'utilisation de *vexpr* dans la routine «transpose **C**», nous voyons que la syntaxe bien familière **$ i1** est utilisée pour référencer les valeurs entières individuelles de la liste, et que nous allons ajouter (ou soustraire) le nombre **1** à chaque entrée. Par conséquent, lorsque l'objet *zl* **reg** génère une liste de **32** valeurs (envoyée à l'origine par le *multislider* **C**), chacune des entrées est incrémentée ou décrémentée et sera produite comme une liste de 32 valeurs.

La routine "randomize **C**" montre l'utilisation de *vexpr* avec deux listes utilisées en entrée. La partie droite de la routine va générer une liste de 32 valeurs de nombres aléatoires variant de **-1** à **1** (le résultat de l'objet *uzi*, *random* **3** et des objets **-1** envoyés dans un **groupe** *zl*). C'est liste est entrée dans l'entrée droite de *vexpr*. L'entrée gauche reçoit la liste de 32 valeurs du *multislider* **C** qui est maintenue par un *zl* **reg**. L'objet *vexpr* itère à travers les deux listes, en ajoutant les éléments de l'entrée droite (identifiés par **$ i2**) aux éléments de l'entrée gauche (identifiés par **$ i1**). Cela signifie que chacun des éléments de la liste sera modifié par un élément aléatoire différent, ce qui donnera une petite randomisation au contenu du *multislider*. Notez que cette randomisation peut être effectuée plusieurs fois pour transformer progressivement la mélodie en un nouveau motif.

Résumé

Nous avons commencé ce tutoriel en découvrant les tables de probabilités et leur utilisation dans l'ensemencement du contenu d'un objet *multislider*. À partir de là, nous avons couvert plusieurs façon de traiter le contenu d'un objet *multislider* (fourni sous forme de liste) à l'aide des objets *zl* et *vexpr*. Comme les listes sont une structure de données intégrale dans l'environnement Max, ces objets deviendront une partie souvent utilisée de votre boîte à outils de programmation.