19: Chronométrage

introduction

Dans ce didacticiel, nous allons travailler avec le ***temps*** - des objets qui traitent du temps, et un ensemble d'objets qui fonctionnent avec un système de chronométrage métrique intégré à Max. De plus, nous examinerons certains objets de décision qui utilisent des opérations logiques pour déterminer le déroulement du programme.

Comme nous l'avons vu dans les précédents tutoriels, l'utilisation du temps est une partie importante du travail avec les programmes Max; les objets temporels tels que *metro* peuvent aider à créer de la musique générative et des actions de dessin. Ils peuvent également être utilisés en conjonction avec les actions de l'utilisateur pour créer des programmes exécutables. Les objets de *transport* dans Max fournissent un système de chronométrage musical centralisé qui peut fournir des commandes de lecture / arrêt, des rétroactions de mesure / battement et peut même déclencher des événements à des moments précis. Tout ceci est construit sur la capacité de définir des temps spécifiques et des durées de temps de manière métriquement significative.

En plus des fonctions temporelles, nous abordons également certains des objets de comparaison booléens - des objets qui créent des résultats logiques basés sur la comparaison et le test de valeurs d'entrée. Ces objets sont particulièrement utiles lorsque vous travaillez avec des objets temporels, car vous voudrez souvent prendre des décisions opérationnelles basées sur des comparaisons entre le temps actuel et une fonction temporelle ou sous-temporelle souhaitée.

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

Quelques nouveaux objets ...

La partie gauche du didacticiel contient cinq petits patchs qui montrent le fonctionnement d'une poignée de nouveaux objets qui traitent du temps dans Max. Le patch **1** fournit un exemple d'objet *pipe*, qui peut être utilisé pour retarder le passage d'un message ou d'une liste pendant un certain temps. L'argument fournit le délai par défaut - dans ce cas, **1000** millisecondes (1 seconde). Si vous modifiez la valeur de la boîte de nombre de ***gauche***, vos modifications seront reflétées dans la sortie 1 seconde plus tard. Le temps de retard peut être modifié avec une valeur envoyée à l'entrée droite - une valeur entière est utilisée pour définir le nombre de millisecondes du temps de retard. Changez la boîte de nombre de droite en **2000**, puis changez l'entrée de gauche - vous verrez maintenant que les changements se produisent après un délai de 2 secondes.

Le petit patch suivant, intitulé **2**, est un exemple d'objet *delay*. Ceci est très similaire à l'objet *pipe*, à la différence qu'il est spécifiquement conçu pour retarder les messages **bang**. Comme pour l'objet *pipe*, le temps de retard peut être modifié en envoyant une valeur entière dans l'entrée droite. Les objets *delay* et *pipe* répondent à un message **stop** pour interrompre la sortie en cours.

Le patch **3** montre l'objet *clocker*, qui est étroitement lié à l'objet *metro*. La principale différence entre *clocker* et *metro* est la sortie: alors que l'objet *metro* produit des messages **bang** à intervalles réguliers, *clocker* génère le temps total écoulé (en millisecondes) depuis le début du **bang**. En cliquant sur le message **stop**, l'horloge s'arrête, tandis que l'envoi d'un nouveau message **bang** à *clocker* redémarre l'horloge et la mesure du temps écoulé.

Le patch étiqueté **4** montre un autre objet qui compte le temps écoulé : l'objet *timer*. Cet objet reçoit des messages **bang** dans les deux entrées. l'entrée ***gauche*** démarre l'intervalle à chronométrer et l'entrée ***droite*** arrête l'intervalle. L'objet *timer* est donc équivalent à un ***chronomètre***; cliquez sur le *button* gauche pour démarrer l'horloge, attendez quelques secondes, puis cliquez sur le *button* droit pour voir combien de temps s'est écoulé. Notez que *timer* est un objet Max légèrement inhabituel dans la mesure où son entrée droite est chaude (produit une sortie), et non son entrée gauche.

Le cinquième patch montre quelques-uns des objets de ***comparaison*** les plus courants en action. L'objet > (plus grand que) génère un **1** si le nombre entrant est plus grand que le nombre utilisé en tant qu'argument, ou un **0** s'il ne l'est pas; l'objet effectue donc un test en indiquant TRUE (**1**) ou FALSE (**0**). L'objet < (inférieur à) est l'inverse: indiquant true ou false (**1** ou **0**), selon que l'entrée est inférieure à l'argument de l'objet. L'utilisation de **1** ou de **0** pour indiquer true ou false est d'usage courant dans les langages de programmation, et constitue la base des opérations ***booléennes***. Les autres objets logiques seront probablement plus familiers aux programmeurs. L'objet == (égal) rapporte un **1** uniquement lorsque le nombre entrant est égal à l'argument, tandis que le *=!* (non égal) est son inverse. Il renvoie un **1** lorsque le nombre entrant n'est pas égal à l'argument. Comme avec beaucoup d'objets Max, un nouvel opérateur de comparaison peut être fourni par un nombre envoyé dans l'entrée droite de l'objet.

Les deux derniers opérateurs logiques sont *&&* (ET logique) et || (OU logique). Dans le premier cas, l'objet *&&* n'émettra un **1** que si les deux nombres envoyés aux entrées gauche et droite sont différents de zéro; sinon, il produit un **0**. Dans le second cas, l'objet || produit un **1** lorsque les entrées gauche ou droite recevront des valeurs différentes de zéro. Si les deux sont à **0**, cet objet émettra un **0**.

Changez la boîte de *nombres* en haut du patch contenant ces objets et vous verrez comment ils se comportent. Notez que la seule fois où les six opérateurs produisent un **1**, c'est lorsque la valeur en entrée est **10**.

Chronométrage métrique, partie 1

Le grand patch à droite (étiqueté **6**) est un patch de performance qui utilise tous ces nouveaux objets (et certains anciens) dans le contexte du système de ***temps métrique***. Lancez le patch (à l'aide du grand *toggle* jaune en haut); vous entendrez un battement régulier de votre synthétiseur MIDI par défaut, avec des variations qui se produisent à différents moments de la performance. La performance est bouclée, se répétant toutes les 16 mesures. Afin de comprendre ce patch plutôt complexe, nous devons d'abord comprendre le système de temps métrique.

Le chronométrage métrique est basé sur un objet central de chronométrage - *transport* - et sur sa capacité à diffuser des informations de temps à d'autres objets qui attendent des informations de temps. Beaucoup de ces objets "d'écoute" nous sont déjà familiers (*metro*) ou ont été introduits dans ce tutoriel (*clocker*). Nous avons utilisé ces objets en mode auto-horloge, où ils utilisent une base de temps exprimée en millisecondes pour produire une sortie à des moments régulièrs. Cependant, lorsqu'ils sont utilisés en conjonction avec l'objet *transport*, le temps est exprimé en notation métrique: des valeurs telles que **4n** (pour une noire) peuvent être utilisées à la place de la notation temporelle en millisecondes.

L'objet *transport* est le chronomètre central et fournit le temps en format ***bar*** / ***beat*** / ***tick*** - tous pilotés par un tempo et une signature temporelle. Par conséquent, si *transport* est configuré pour fonctionner avec une signature temporelle de **4/4** et à un tempo de 120 battements par minute, chaque battement prendra 500 millisecondes et une mesure prendra 2 secondes (2000 millisecondes). Afin de voir toutes les manières dont *transport* contrôle ce patch, nous devons le

décomposer en sections.

Notre "grosse case à cocher" lance le processus - principalement parce qu'elle lance le *transport*. Le *transport* accepte un message **1/0** pour le démarrage et l'arrêt, ce qui est parfait pour le contrôle par un *toggle*. Le *toggle* lance également le *metro* à gauche (partie de la section «rouge» du patch), qui utilise 4n comme argument de synchronisation. Dans le passé, nous utilisions des millisecondes pour les minutages de l'objet *metro*, mais dans ce cas, la notation 4n signifie qu'il enverra un **bang** toutes les noires, sur la base de la durée diffusé par *transport*. La sortie de ce *metro* va à deux endroits: vers une boîte de *message* qui crée une note MIDI 70 (via *makenote*) et vers le *transport*. Pourquoi est-il acheminé vers le *transport* ? Parce qu'un message **bang** reçu par le *transport* fera afficher le temps actuel, que nous affichons dans trois boîtes de *nombres* connectées aux trois premières sorties de l'objet. Ce temps est affiché en bar, beat et ticks, respectivement. Il y a **480** ticks par temps (480 impulsions par noire ou ***ppq*** par défaut).

Ensuite, examinons la section «verte» du patch. Elle utilise le nombre de mesures (tel qu'indiqué par le *transport*), et utilise les objets > et < pour déterminer si nous sommes entre les mesures 4 et 8. Si nous sommes dans cette plage, il active un autre *metro* dont la cadence est à ***16n*** (double croche). Cela génère un nombre aléatoire utilisé pour créer l'une des quatre valeurs (en utilisant les objets *random* et *select*); ces nombres sont envoyés à l'objet *makenote* pour générer des notes MIDI. Cette valeur est également envoyée à un objet *pipe* avec un argument de 4nd, ce qui retarde les valeurs d'une noire pointée. Cette sortie se voit ajouter **7**, ce qui donne une sortie qui, lorsqu'elle est envoyée au synthétiseur MIDI, sonne une quinte parfaite au-dessus de l'original. Le résultat est un jeu de notes rapides pour un segment de boucle composé de quatre mesures.

Passons maintenant à la section «bleue», qui est la partie du patch qui force la séquence à ***boucler***. La tête de cette section du patch est un nouvel objet : *timepoint*. L'objet *timepoint* prend une durée métrique en tant qu'argument; son seul but est de générer un message **bang** lorsque ce temps métrique est atteint par le *transport*. Dans ce cas, lorsque nous atteignons le début de la mesure **17** (c'est-à-dire après seize mesures), un accord de quatre notes est envoyé à la *makenote*, puis un message **bang** est envoyé jusqu'à un *message* connecté à l'entrée droite de *transport*. Ce message (**0.**) indique à *transport* de passer immédiatement au tick **0.** de la séquence, c'est-à-dire à la mesure 1, 1 et 0. C'est ainsi que nous forçons la séquence à tourner en boucle: chaque fois que nous atteignons la 17e mesure, nous sommes immédiatement renvoyé au début.

Chronométrage métrique, partie 2

Ensuite, nous devons décoder la section «brune» du patch. Elle utilise les objets *key* et *select* pour surveiller si la barre d'espace (ASCII 32) est enfoncée. Lorsque cela se produit, un objet *timer* reçoit un **bang** sur les ***deux*** entrées. Etant donné que les objets Max transmettent les messages dans un ordre allant de droite à gauche, appuyer sur la barre d'espacement arrête simultanément le chronomètre avec un **bang** à ***droite*** (et affiche le temps écoulé) et le redémarre pour une autre mesure avec un **bang** à ***gauche***. L'objet affiche le temps écoulé en millisecondes; nous divisons **60000** (le nombre de millisecondes en une minute) par le temps nécessaire pour obtenir une valeur bpm (battements par minute).

Cependant, nous voulons que cette valeur de bpm soit comprise dans une fourchette raisonnable. Nous utilisons donc l'objet *split* et n'utilisons que des valeurs comprises entre **30** et **200 .**. Si nous nous situons dans cette plage, *split* affichera le nombre à partir de sa sortie gauche. Et la valeur sera utilisée pour créer un message de ***tempo*** qui sera envoyé au *transport*. Cela changera immédiatement le tempo de lecture de la séquence et modifiera les vitesses relatives de plusieurs des objets (mais pas tous) utilisés pour créer notre séquence. Si le *timer* émet une valeur en dehors de la plage de l'objet *split* (par exemple, si nous n'avons pas appuyé sur la barre d'espace depuis un

moment), il sera envoyé par la sortie droite de l'objet et écarté.

L'une des raisons pour lesquelles toute la séquence n'a pas été modifiée est que certaines des fonctions de patch utilisent le timing standard (millisecondes) plutôt qu'un temps métrique. La section la plus à droite («bleue et violette») du patch en est un bon exemple. Un objet *timepoint* est activé lorsque le temps de transport est à mesure **6**, temps 3. Ce qui déclenche un *clocker* qui fonctionne pendant **10** secondes (10 000 millisecondes), produisant un accord de deux notes tous les noires (**4n**). L'objet *timepoint* envoie également un **bang** à un objet *delay* **3000**, qui produit un accord de deux notes **3000** millisecondes après avoir reçu le message. À un tempo très lent, cela peut se produire seulement quelques battements après le déclenchement de *timepoint*, alors qu'à un tempo rapide, cela peut prendre plusieurs mesures. C'est un bon exemple de la façon dont le temps métrique et le temps standard peuvent être mélangés et assortis pour obtenir des résultats génératifs intéressants. Notez que le temps de fonctionnement de l'objet *clocker* est comparé à un objet > pour s'éteindre après 10 secondes ... un objet *timer* suivant l'intervalle entre le démarrage et l'arrêt de l'objet *clocker* le confirme.

L'importance de l'overdrive

Le paramètre **Overdrive**, proposé dans le menu **Options**, est l'un des facteurs susceptibles de provoquer des comportements différents dans ce patch. Si la case **Overdrive** est cochée (on), le traitement des événements et les calculs ont la priorité par rapport aux processus de dessin à l'écran et aux processus graphiques. Lorsque le paramètre **Overdrive** est désactivé (off), les processus de dessin ont la même priorité que le traitement des événements. Selon la nature de votre patch, **Overdrive** peut offrir de meilleures performances pour les tâches de traitement d'événements critiques en termes de temps.

Étant donné que notre patch de didacticiel est critique en terme de temps, il sera plus performant si **Overdrive** est activé. Si **Overdrive** est désactivé, vous remarquerez peut-être que l'affichage de la sortie de *transport* ne se produit pas toujours sur le tick **0** et vous pourrez constater que la lecture devient saccadée lorsque la densité des notes est élevée ou que vous effectuez d'autres tâches sur votre ordinateur (comme l'ouverture d'un autre patchuer dans Max). Les ordinateurs modernes sont incroyablement rapides, mais nos oreilles sont très sensibles aux discontinuités temporelles. Par conséquent, lorsque vous travaillez sur des patches qui séquencent les événements de manière critique, il est généralement conseillé d'activer **Overdrive**.

Résumé

Dans ce tutoriel, nous avons couvert un nouveau système complet au sein de Max: le système de ***chronométrage métrique***. Il fournit un contrôle centralisé du chronométrage pour votre patch Max, coordonnant la génération d'événements et facilitant la création de séquences musicales. Nous avons également appris à connaître plusieurs nouveaux objets temporels, tels que *pipe*, *delay*, *clocker* et *timer*, qui peuvent être utilisés (avec ou sans métrique) pour créer des actions séquencées. Enfin, nous avons vu l'utilisation des objets de comparaison booléene et d'opérateurs logiques pour faciliter la prise de décision dans un patch complexe.