Music Animations

(Animation de portée)

Cette application permet de faire des animations musicales (écrites), à des fins pédagogiques, pour les insérer dans des screencasts.

Animation

Table des matières

Réinitialiser les préférences

Introduction

Une animation est composée de `pas` (step) qui vont être exécutés l'un après l'autre, produisant des choses aussi diverses que :

- L'apparition d'une portée ;
- L'écriture d'une note ou d'un accord sur la portée ;
- La création d'un fond, avec dégradé ;
- Le déplacement de notes ;
- L'écriture et l'animation de textes avec des effets d'opacité ;
- L'aide à l'audio avec l'inscription de textes de doublage ;
- L'animation de flèches ;
- etc.

Chaque pas (chaque “step”) exécute une action. Ces pas se définissent dans la console située à gauche de l'écran (le bloc noir). Chaque pas se trouve sur

une ligne distincte. Donc chaque ligne est un nouveau pas, une nouvelle étape de l'animation.

Bien noter que chaque ligne représente un pas, il est impossible de définir plusieurs actions sur la même ligne (en réalité c'est faux, mais c'est dangereux ;-)).

Ces pas peuvent :

- Construire une portée ;
- Construire une note (et l'afficher) ;
- Construire un accord ;
- Construire une gamme ;
- Écrire un texte associé à un objet (note, portée, barre, etc.) ;
- Déplacer un objet quelconque (des notes, des textes, etc.) ;
- Supprimer une note ;
- Mettre une note en exergue (entourée d'un cercle de couleur) ;
- etc.

Définir le cadre de l'animation

Par défaut, le cadre réservé à l'animation est en mode `adapt`, c'est-à-dire qu'il se règle en fonction de la taille de la fenêtre. Plus la fenêtre est large et plus le cadre est large lui aussi, en respectant une proportion de 16/9 (utilisée notamment par les projections YouTube ou Viméo).

Mais dans le menu “Options”, on peut choisir une taille fixe en activant les items “Dimension 480p” et “Dimension 720p” qui correspondent aux deux formats traditionnels d'affichage sur le web (les valeurs `480` et `720` correspondent à la hauteur du cadre de l'image).

Ces deux formats peuvent être choisis par défaut pour une animation particulière en utilisant la commande :

DEFAULT('screensize', '<480p ou 720p>')

Définition précise de la taille

Comme mentionné plus haut, en mode `adapt` (“Dimension adaptée à

l'écran”), la taille du cadre est calculé par rapport à la taille de la fenêtre courante dans laquelle est chargée l'application.

Mais ce qu'il faut comprendre, c'est que les éléments de l'animation, eux, sont en valeur absolues, ce qui signifie que lorsqu'ils sont placés et dimensionnés, leur dimension et leur placement ne changent pas avec le redimensionnement de la fenêtre (hors zoom du navigateur, évidemment).

Donc, lorsqu'on travaille une animation très précisément, et qu'on veut ensuite l'enregistrer en conservant les mêmes positionnements, il est impératif d'enregistrer dans le code de l'animation la dimension actuelle du cadre dans lequel l'animation a été composée.

Grâce au menu “Options > Geler taille du cadre”, on peut obtenir le code à placer en début d'animation pour retrouver toujours cette taille, quel que soit la taille de la fenêtre. Ce code ressemble à :

```
DEFAULT('screensize', <{Number} Hauteur du cadre>)
```

… où `<hauteur du cadre>` est la hauteur à donner au cadre, en sachant que la largeur sera calculée pour respecter le format de 16/9e.

Composer le code de l'animation

Écrire un commentaire

Les commentaires s'écrivent avec `#` en début de ligne.

Noter que les commentaires ne peuvent pas se trouver sur un pas proprement, même après le code. Il faut absolument qu'ils soient sur une ligne seule, qui sera passée.

Faire une pause dans l'animation.

On utilise la méthode `WAIT` pour faire une pause, avec en argument le nombre de secondes (qui peut être un flottant, pour des temps très courts) :

```
WAIT(<nombre de secondes>)
```

Par exemple, pour attendre 4 secondes :

```
maNote=NOTE(a3)
WAIT(4)
# => 4 secondes avant de construire l'accord
monAccord('c3 eb3 g3')
```

Note : C'est un “pas” comme les autres, donc il doit être mis sur une ligne seule comme toute étape.

Noter qu'on aurait pu aussi utiliser le paramètre spécial d'attente en créant la note pour obtenir le même résultat :

```
maNote = NOTE(a3, {wait:4})
# => Attente de 4 secondes avant de passer à la suite
```

Se déplacer sur la portée

Pour tous les déplacements sur la portée, voir Déplacement du curseur de position

Écrire un texte général

Pour écrire un texte en rapport avec l'animation (en haut à gauche), on utilise la méthode `WRITE` :

WRITE()

Par exemple :

WRITE("Ce qu'il faut remarquer à ce moment-là.")

Noter que les objets tels que les notes ont également leur propre méthode textuelle, qui permet d'aligner le textes directement à ces notes, accords, etc.

Reset(-ter) l'animation

La commande `RESET` (sans parenthèses ni arguments) permet de repartir de rien en cours d'animation, c'est-à-dire :

- Effacer tous les objets SAUF les portées et les clés ;
- Remettre le "curseur" tout à gauche (endroit où seront donc affichés les nouveaux objets).
- Remettre toutes les valeurs par défaut.

Usage :

```
RESET
```

Voir aussi la commande `CLEAR` ci-dessous qui permet la même chose mais de façon plus ciblée.

Nettoyer l'animation

“Nettoyer l'animation” signifie supprimer tous les éléments affiché. On peut ou non demander que les portées restent en place, though.

Le pas à utiliser est :

```
CLEAR(<avec les portées>)
```

… `<avec les portées>` est FALSE par défaut, donc il faut ajouter `true` pour effacer aussi les portées :

```
CLEAR(true) // efface aussi les portées
```

Mettre un point d'arrêt avant la fin

Il peut être intéressant parfois de mettre un point d'arrêt à l'intérieur du code de l'animation, qui l'interrompra comme le bouton “Stop” (par exemple pour modifier les coordonnées d'un élément de l'animation).

Pour ce faire, il suffit d'utiliser la commande :

```
STOP
```

… à l'endroit où l'animation doit s'interrompre.

Jouer rapidement jusqu'au passage travaillé

Lorsqu'on travaille un passage, il peut être intéressant de jouer rapidement l'animation jusqu'à ce passage (surtout lorsque des animations lentes sont utilisées).

Pour ce faire, on utilise le mode Flash.

Le paramètre spécial d'attente (`wait`)

Dans toutes les méthodes d'animation, le dernier argument définit les paramètres optionnels (ou parfois obligatoires) de la méthode.

Parmi ces paramètres, on trouve le paramètre spécial `wait` très utile pour définir le comportement qu'aura l'animation sur l'étape.

Par défaut, en effet, l'animation attend toujours la fin de l'étape courante (la ligne de code courante) avant de passer à la suite. Le paramètre `wait` permet de modifier ce comportement :

- Si `wait` est mis à `false`, l'animation lance l'étape courante, mais passe immédiatement à la suite. Utilisé par exemple pour donner l'impression de simultanéité d'effets divers.
- Si `wait` est un nombre (entier ou flottant), l'animation lance l'étape courante puis attend le nombre de secondes défini avant de passer à la suite. Noter que deux effets peuvent en résulter : si le nombre de secondes est inférieur à la durée que prend l'étape, l'animation de l'étape est lancée, puis au bout du laps de temps stipulé on passe à l'étape suivante. Si, en revanche, le nombre de secondes est supérieur à la durée que prend l'étape, l'animation de l'étape est lancée, puis l'animation attend la fin de la durée avant de lancer l'étape suivante.

Exemples d'utilisation du paramètre `wait` :

```
maNote = NOTE('a4', {wait:10})
# => La note LA 4 est créée, mais l'animation attend 10 sec
ondes avant
#    de passer à la suite.

maNote.move('2:f2', {wait:false, duree:11})
# => L'animation va déplacer la note LA 4 vers la note FA 2
, un déplacement
#    qui va durer 11 secondes, mais l'étape suivante sera a
ppelée dès le début
#    du déplacement.
```

Le paramètre spécial de durée (`duree`)

Dans toutes les méthodes d'animation, le dernier argument définit les paramètres optionnels (ou parfois obligatoires) de la méthode.

Parmi ces paramètres, on trouve le paramètre spécial `duree` qui peut définir, en nombre de secondes, la durée que doit prendre l'animation quelconque.

Lorsque la durée doit être inférieur à la seconde, on utilise un flottant. Par exemple `0.5` pour une demi-seconde. *Noter qu'il est aussi possible d'utiliser une fraction, ce qui est peut-être plus clair. Par exemple `1/4` pour un quart de secondes.*

Exemples d'utilisation du paramètre spécial `duree` :

```
monTexte = TBOX("Ceci est un texte caché au départ", {hidde
n:true})
# => Un texte caché (hidden:true) est créé
monTexte.show({duree:5})
# => Le texte va apparaitre en 5 secondes
monTexte.move({x:10, y:10, duree:15})
# => Le texte va mettre 15 secondes pour rejoindre la posit
ion 10/10 de
#    l'écran de l'animation.
```

Vitesse de l'animation

On peut régler la vitesse de l'animation de façon interactive avec le “slider” se trouvant dans le contrôleur (sous le bouton “Play”).

Pour régler la vitesse de façon très fine, cf. aussi Réglage de la vitesse dans les préférences.

Mais on peut aussi définir dans le code cette vitesse à l'aide de la commande :

```
SPEED(<valeur>)
```

… avec `<valeur>` pouvant être un nombre de 0 à 20, `10` étant la vitesse normale, 0 étant la vitesse la plus lente et 20 la plus rapide.

Pour remettre la vitesse à la vitesse normale :

```
SPEED() // pas d'argument
```

Jouer l'animation

Pour jouer l'animation, cliquer simplement sur le bouton “Play” ou sur la barre espace.

Si un autre choix que l'animation entière avait été fait, sélectionner l'item « Jouer tout » dans le menu « Animation » ou presser le raccourci clavier CMD+0.

Jouer une portion de l'animation

En composant l'animation, on peut vouloir ne tester qu'une partie de code qu'on est en train de programmer sans avoir à rejouer tout ce qui précède.

Il existe plusieurs moyens de le faire avec “STAVES” :

- Sélectionner simplement le code à jouer (version rapide, mais non modifiable — cf. Jouer la sélection) ;
- Jouer l'animation à partir de la position du curseur. Pratique pour les animations longues lorsque l'on code la fin (cf. Jouer l'animation depuis le curseur) ;
- Placer des repères pour définir la sélection à jouer. Pratique lorsque le

code à modifier ne se trouve pas à la fin (cf. Jouer une portion par repère) ;

Jouer l'animation depuis le curseur

1. Choisir l'item “Jouer depuis curseur” dans le menu « Animation » (ou le raccourci clavier CMD+3) ;
2. Placer le curseur dans la première ligne de code à jouer (inutile ici de le placer en début de ligne, il suffit de le mettre dans la ligne de code par laquelle commencer) ;
3. Presser le bouton Play.

Noter cependant une chose importante : si le code à partir du curseur fait référence à des éléments précédemment créés, le jeu de l'animation échouera (elle n'exécute le code qu'à partir du curseur).

Jouer une partie de l'animation entre repères

Cette méthode est la plus pratique si on compte modifier le code au cours de différents essais et modifications.

1. Dans le code, placer **au-dessus** de la première ligne de code à interpréter (sur une seule ligne ne contenant QUE ce repère) :

```
#!START
```

2. Placer **en dessous** de la dernière ligne à interpréter (sur une seule ligne ne contenant QUE ce repère) :

```
#!END
```

3. Dans le menu « Animation », choisir l'item “Jouer entre #!START et #!END (ou jouer le raccourci CMD + 2)” ;
4. Cliquer sur le bouton “Play”

Jouer une partie de l'animation par sélection

Pour ce faire :

1. Sélectionner soigneusement la portion de code à tester (“soigneusement” signifie : en étant sûr de bien placer la sélection du début d'une ligne au début d'une autre ou à la toute fin du code. Cf. la procédure la plus simple ci-dessous)
2. Choisir l'item “Jouer la sélection” dans le menu « Animation » (ou presser le raccourci CMD+1).
3. Cliquer sur le bouton “Play”

Pour rejouer la même sélection (mais seulement si le code n'a pas été modifié), cliquer à nouveau sur “Play”.

Méthode pour sélectionner correctement la portion de code

Le plus simple, pour sélectionner correctement la portion de code à tester, est de procéder comme suit :

1. Placer le curseur au début de la première ligne de code
2. Relâcher la souris
3. Faire défiler le code de la console jusqu'à afficher la dernière ligne de code à tester.
4. Presser la touche Majuscule
5. Tout en tenant la touche Majuscule, cliquer au bout de la dernière ligne du code à tester.

Jouer seulement la sélection (bouton “Point”)

Grâce au bouton “Point” (le bouton se trouvant dans le contrôleur à côté du bouton Play), on peut ne jouer que la ligne dans laquelle se trouve le curseur ou les lignes sélectionnées (même si elles ne sont que partiellement sélectionnées).

Noter un point important ici : contrairement à l'option “Jouer seulement la sélection” (menu “Options”) qui va jouer ce qui précède la sélection en mode flash, ie très vite (cf. [Jouer une portion de l'animation][#run_only_selection]), ici, c'est vraiment uniquement la ligne ou les lignes sélectionnées qui seront interprétées.

Cela permet de faire des tests très ponctuels de commandes.

Enchainer des animations

Le code d'une animation pouvant très vite devenir conséquent, on peut enchainer très facilement des animations. Il suffit pour cela d'utiliser la commande :

```
SUITE('<nom ou chemin de l'animation sans extension>)
```

On peut mettre cette commande où l'on veut dans le code, puisqu'elle ne sera interprétée qu'une fois l'animation courante exécutée. Il est donc possible de la mettre au début du code comme à la fin.

Pour faire référence à une animation se trouvant le dossier de l'animation courante (ie qui utilise la commande `SUITE`), on peut indiquer le chemin complet bien sûr, mais aussi le raccourci-point :

```
SUITE('./anim_dans_meme_dossier')
```

Noter que sans ce `./` l'animation sera cherchée à la racine du dossier des animations.

Astuce : on peut remplacer les traits plats (“__”) par des espaces, ils seront automatiquement corrigés.

Activer/désactiver le mode “Flash”

Le mode “Flash” permet de jouer très rapidement une partie de l'animation. C'est utile lorsqu'une partie est calée, mais qu'on ne veut pas la rejouer systématiquement au même rythme pour atteindre la portion en cours d'élaboration.

On peut aussi l'utiliser pour accélérer un ensemble de modifications qui pourraient se faire en même temps.

Pour passer en mode flash :

```
MODE_FLASH        ou      FLASH
```

Pour stopper le mode flash et revenir à la vitesse normale

```
STOP_MODE_FLASH   ou      STOP_FLASH
```

Dans ce cas, on met le code à passer entre :

```
FLASH
.................
...  code    ...
... à jouer  ...
... en flash ...
.................
STOP_FLASH
```

NB : On peut utiliser l'autocomplétion `fla[TABULATION]` pour placer la commande `FLASH` et l'autocomplétion `flas[TABULATION]` pour placer la commande `STOP_FLASH`.

On peut bien évidemment passer plusieurs passages en mode flash. Chaque partie entre `FLASH` et `STOP_FLASH` sera accélérée.

Note : Si les commandes se multiplient trop dans le code au point de s'y perdre, on peut utiliser l'outil “Outils > Nettoyer les commandes FLASH” qui permet de supprimer toutes ces commandes du code de l'animation.

Régler le décompte

Par défaut, l'animation attend deux secondes avant de démarrer.

On peut régler sa valeur par :

DEFAULT('decomte',)

Mettre `0` pour supprimer tout décompte.

Sauvegarde automatique

En activant le menu “Options > Activer Auto-Save”, tous les modifications apportées au code seront automatiquement enregistrées.

Ouvrir une animation récemment ouverte

Le menu “Fichier > Ouvrir animation récente >” permet de rapidement retrouver une animation récemment ouverte ou créée.

Ce menu contient jusqu'à 10 animations récentes.

Ré-initialiser toutes les valeurs de préférence

Pour remettre toutes les valeurs de décalages aux valeurs de départ, utiliser la commande (SANS PARENTHÈSES) :

```
RESET_PREFERENCES
```

Noter que ça ne ré-initialise que les décalages des éléments, tels que les textes de marque de l'harmonie ou des accords, etc. Pour une ré-initialisation complète, utiliser la commande `RESET` *.*

Définir l'animation courante comme animation par défaut

Pour définir l'animation courante comme l'animation par défaut — ie celle qui s'ouvrira au prochain chargement de l'application —, activer le menu “Fichier > Définir comme anim par défaut”.

Déplacement du curseur de position

Le “curseur de position” est la ligne verticale bleue qui traverse verticalement l'animation (invisible quand on la joue) qui indique où seront positionnés les prochains éléments.

Noter que ce “curseur de position” n'a rien à voir avec la position courante de l'animation, entendu qu'une animation n'est pas, sauf rare exception, un déplacement continu le long d'une portée. Pour l'explication d'une gamme, par exemple, il peut y avoir un affichage fixe le long duquel on se déplace indifféremment.

Table des matières

Déplacement du curseur vers la droite

La commande pour écrire à la suite des dernières notes sur la portée, on utilise la commande :

```
NEXT([<nombre pixels>])
```

Cela déplacera le curseur de position d'une valeur par défaut de 40px. Cette valeur peut être modifiée grâce à `DEFAULT('next', <nouveau nombre de pixels>)` (cf. Réglage de l'avancée à chaque NEXT (curseur)).

Par exemple :

```
NEXT()
// => les notes suivantes s'écriront 40px plus à droite
```

On peut également effectuer un déplacement précis en indiquant le nombre de pixels en argument.

```
NEXT([<nombre pixels>])
```

Par exemple, pour déplacer le curseur de position de 100px vers la gauche (donc un retour en arrière) :

```
NEXT(-100)
// => les notes suivantes s'écriront 100px plus à gauche
```

Déplacer le curseur sur un pas

À chaque déplacement (`NEXT` ou `SET_CURSOR`), l'animation mémorise la position comme un “pas” (*note : si la grille est active, ils sont représentés par des marqueurs tout en haut de l'animation.*). Il est très simple de revenir à un

de ces “pas”.

Première formule

```
SET_CURSOR({pas:<indice du pas>})
```

Deuxième formule

```
SET_CURSOR('pas', <indice du pas>)
```

Troisième formule

```
SET_CURSOR(<indice du pas>)
```

Noter que dans cette dernière formule, on utilise la même syntaxe que pour placer le curseur de position à une position précise. Mais l'animation par du principe que si son premier argument est un nombre inférieur à 15 (donc qui serait positionné sur une clé de portée), c'est un pas et non pas (sic) une position en pixels. Il y a rarement plus de 15 pas dans une animation traditionnelle.

Dans le cas contraire, s'il n'y avait pas de portée et qu'on voulait vraiment positionner un élément à moins de 15 pixels, on devrait alors utiliser la formule explicite :

```
SET_CURSOR({offset:<nombre de pixels>})
```

Régler le pas entre chaque NEXT()

Cf. Réglage de la position next

Revenir au début des portées (RESET_CURSOR)

Pour revenir au début des portées, on peut bien entendu utiliser la commande précédente avec un nombre négatif. Si l'on se trouve à la position 500, il suffit d'indiquer :

```
NEXT(-500)
```

… pour se replacer au début.

Mais une commande permet d'effecter ce retour plus simplement, sans connaitre la position courante :

```
RESET_CURSOR
```

Cela remet automatiquement le curseur au début, donc à gauche, en tenant compte des altérations et métriques que peuvent contenir les portées.

Pour modifier cette position de début, utiliser la commande `DEFAULT('x_start', <nouvelle valeur>)` *(cf. Réglage de la position horizontale initiale (curseur)).*

Placer le curseur à une position précise (SET_CURSOR)

Par indication de la position exacte

Pour placer la curseur à une position précise, utiliser la commande :

```
SET_CURSOR(<position left en pixel>)
```

Par exemple, pour se positionner à 100px exactement :

```
SET_CURSOR(100)
```

Les prochains `NEXT()` partiront de cette nouvelle position.

Par indication du “pas”

Chaque fois qu'un `NEXT` est invoqué, le curseur de position se déplace, créant un nouveau “pas”. On peut utiliser ces pas avec la commande `SET_CURSOR` en lui fournissant l'indice (1-start) du pas à utiliser :

```
SET_CURSOR({pas:<indice du pas>})

ou

SET_CURSOR('pas', <indice du pas>)
```

Note : On peut voir ces pas en affichant la grille, ils sont représentés par des petits rectangles tout en haut de l'animation.

Par exemple :

```
NEXT()
# => Place le curseur disons à 200. C'est le 1er pas
NEXT()
# => Place le curseur disons à 250. C'est le 2e pas
NEXT()
# => Place le curseur à 300. C'est le troisième pas.
SET_CURSOR({pas:1}) ou SET_CURSOR('pas', 1)
# => Place le curseur au premier pas, donc à 200
```

Par indication du décalage de “pas”

Au lieu de fournir l'indice du pas comme dans l'utilisation précédente, on peut fournir aussi le décalage de pas par rapport à la position courante.

```
SET_CURSOR({offset:<nombre de pas du déplacement>})

ou

SET_CURSOR('offset', <nombre de pas du déplacement>)
```

Par exemple, pour revenir de deux pas en arrière :

```
SET_CURSOR({offset:-2})

ou

SET_CURSOR('offset', -2)
```

Outils pratiques

Table des matières

Point de repère avec la grille

Lorsque la grille est active (pour l'activer : “Options > Afficher la grille”), lorsque l'on clique à un endroit de l'animation, un point apparait et les coordonnées de ce points sont donnés en bas de l'écran.

Outil “Coordonnées”

Cet outil permet d'obtenir les coordonnées précises à l'écran, pour pouvoir positionner un élément par exemple.

On l'active par le menu “Outil > Activer coordonnées”.

Ensuite, il suffit de déplacer et de re-tailler le rectangle rouge pour obtenir les coordonnées des lignes de fuite bleues.

Noter qu'après un redimensionnement du cadre rouge, il est parfois nécessaire de cliquer au centre du cadre pour obtenir les nouvelles coordonnées.

Les Notes

Table des matières

- Détruire d'une note
- Référence aux notes d'un objet pluri-notes

Désignation des notes

Les notes doivent être désignées par :

```
<nom note><alteration><octave>
```

- On désigne le `nom des notes` par une seule lettre, anglaise, de "a" (la) à "g" (sol).
- L'altération est soit rien (note naturelle), soit un signe (cf. ci-dessous "b", "d", "x", ou "t").
- Vient ensuite l'octave, un nombre, négatif si nécessaire (*mais pour le moment, on va seulement jusqu'à l'octave 0, les autres ne sont pas gérés*). Noter que l'octave peut être omis s'il est défini dans les paramètres de la création de la note. Par exemple :

  ```
  maNote = NOTE('e', {octave:4})
  ```

Marque des altérations

La valeur `<alteration>` ci-dessus peut être :

- `b` pour bémol ;
- `t` (comme "ton") pour double bémol ;
- `d` pour dièse ;
- `x` pour double-dièse ;
- `z` pour bécarre (comme le Pomme/Ctrl+"Z" qui annule la dernière action, ce "z" annule la dernière altération).

Constantes notes

De l'octave 0 (qui n'existe pas en français) à l'octave 7 il existe des constantes pour chaque note avec l'altération bémol ("b") et dièze ("d").

Les deux formules suivantes sont donc possible :

```
no=NOTE(ad5)
no=NOTE('ad5')
```

Note: Attention à ne pas donner à une variable note le nom d'une de ces constantes. Par exemple, si on fait :

```
a5=NOTE(a5)
```

Cela génèrera une erreur de constante déjà définie.

On peut utiliser plutôt :

```
na5=NOTE(a5)
```

Créer une note

```
<variable name> = NOTE(<note>)
```

- `<variable>` peut avoir le nom qu'on veut, HORMIS un nom de constante, comme `a5`.
- La `<note>` peut être soit un string soit une constante (cf. Désignation des notes)

On peut aussi, si l'on est certain que la note ne sera pas utilisée après (déplacée, supprimée, mise en exergue, etc.) ne pas l'affecter à une variable et composer immédiatement son aspect.

Par exemple, pour placer une note LA 4 avec une marque de cadence et un accord :

```
NEW_STAFF(SOL)
NOTE(a4).cadence('V I').chord('A Maj')
```

Déplacer une note

Pour déplacer une note, on utilise le pas :

```
<nom variable note>.move(<nouvelle note>[,<params>])
```

Exemple :

```
maNote=NOTE(a4) // crée la note LA 4
maNote.move(g4) // descend la note vers SOL4
```

Ne mettre aucune espace dans ce code.

Noter que la note de destination devra vraiment la nouvelle valeur de la note. Si la note "a4" se déplace vers "a3", cette note deviendra "a3" dans ses données.

Placer une note sur une portée précise

Si on veut placer une note sur une portée hors de la portée active, on indique l'indice de cette portée avant la note, puis ":" :

```
<indice portée>:<note>
```

Par exemple :

```
note_autre_staff=NOTE('2:a4')
OU
note_autre_staff=NOTE('2:'+a4)
```

… placera un LA4 sur la deuxième portée, même si elle n'est pas active.

Noter que cela ne rend pas la portée active.

Mettre une note en exergue (ou la retirer de l'exergue)

Utiliser la méthode `exergue()` (pour mettre en exergue, en couleur) et `unexergue()` (pour la sortir de l'exergue).

Exemple :

```
maNote=NOTE('cx5')
maNote.exergue()   # => note en couleur (bleu par defaut)
WAIT(4)
maNote.unexergue()  # => etat normal
```

On peut définir la couleur, entre `red` (rouge) `green` (vert) ou `blue` (bleu, par défaut) :

```
maNote.exergue({color:<couleur>})
```

Ou même simplement :

```
maNote.exergue(green)
```

Cf. aussi Entourer une note ci-dessous.

Note fantôme

On peut rendre la note à peine visible (fantôme), en lui appliquant la méthode :

```
<note>.fantomize()
```

Par exemple :

```
maNote = NOTE('2:c3').fantomize()
```

Pour sortir la note de cet état et la remettre visible, utiliser :

```
<note>.defantomize()
```

Par exemple :

```
maNote = NOTE(c4)
maNote.fantomize()
WAIT(4)
maNote.defantomize()
```

Entourer une note

Pour mettre en exergue une note de façon plus forte que la méthode exergue, on peut utiliser la méthode `surround()` qui entoure la note d'un cercle de couleur.

Syntaxe :

```
<note>.surround([<parameters>])
```

… où `<note>` est une instance de Note, et `<parameters>` sont les paramètres optionnels envoyés.

Par exemple :

```
maNote=NOTE(c5)
maNote.surround() # => entoure d'un cercle rouge

oNote=NOTE(a4)
oNote.surround({color:blue, rectangle:true})
# => entoure d'un rectangle bleu
```

Les paramètres peuvent être :

color:{String\|Constante}

La couleur, parmi 'blue', 'red', 'orange', 'green', 'black' (les valeurs peuvent se définir avec ou sans guillemets).

rectangle:{Boolean}

Si true, un rectangle au lieu d'un rond.

margin:{Number}

Joue sur le diamètre du cercle/rectangle pour laisser plus ou moins de place. Utile lorsque plusieurs notes assez proches sont entourées

Retirer le cercle

Deux commandes pour retirer le cercle de la note :

```
maNote.circle.remove()
```

Ou :

```
maNote.unsurround()
```

Forcer l'actualisation d'une note

Parfois, il peut survenir qu'une note reste déplacée sur le côté à cause d'une note inférieure conjointe alors que la note inférieure n'est plus là.

Par exemple :

```
nosi = NOTE(b3)
nodo = NOTE(c4)
# => c4 sera décalé à droite
nosi.move(b2)
# => Le si est déplacé mais le do
#    reste décalé à droite
```

Dans ce cas (avant que ce bug #49 ne soit corrigé), on peut utiliser la méthode `update` sur la note pour la remettre bien en place :

```
...
nosi.move(b2)
nodo.update()
```

Détruire une note

Pour détruire la note (la supprimer de l'affichage, utiliser :

```
<nom variable note>.remove()
```

Les motifs mélodiques

Bien entendu, un motif mélodiques (une suite de notes) peut s'écrire avec la commande `NOTE` (cf. Les notes). Mais il peut être plus rapide d'utiliser plutôt la commande `MOTIF` qui crée rapidement un motif mélodique et permet de le manipuler comme un ensemble.

Table des matières

- Régler la distance entre les notes
- Modifier la vitesse d'affichage du motif

Créer un motif

monMotif = MOTIF(''[,])

Par exemple :

```
unMotif = MOTIF('c4 c c d e d c5 e d d c')
```

Noter que dans un motif, l'octave n'a besoin d'être stipulé avec la note que s'il est différent de l'octave de la note précédente. Par défaut (si aucune note ne possède de définition d'octave), c'est l'octave 4 qui est choisi.

Noter qu'il est tout à fait possible de déterminer des positionnements sur des portées différentes, en préfixant les notes de la portée où elles doivent être écrites. Par exemple :

```
unMotifSurDeuxPortees = MOTIF('c4 2:c c')
```

Si la portée active est la 3e portée, la première et la dernière note seront affichées sur cette portée, tandis que la deuxième note sera affichée sur la 2e (`2:`).

Manipuler les notes du motif

On peut faire référence et manipuler les notes d'un motif à l'aide de la méthode `note` .

Par exemple :

```
monMotif.note(5).remove()
# => Détruit la 5e notes du motif
```

Comme `remove` ci-dessus, toutes les méthodes applicables aux notes seules sont applicables à une note d'un groupe de notes.

Pour connaitre les différents moyens de faire référence aux notes, cf.

Référence aux notes d'un objet pluri-notes

Paramètres optionnels

staff

La portée à laquelle sera associée le motif. Noter que l'appartenance des notes est indépendante de cette valeur ; si le motif est long et que l'animation doit passer à une portée suivante, les notes suivantes seront associées à cette portée. De la même manière, une note définie par `:` sera associée à la portée correspondante (indice portée).

speed

Vitesse d'affichage du motif au cours de l'animation (cf. Régler la vitesse d'affichage du motif).

offset_x

Définition de la distance horizontale entre les notes (cf. Régler la distance entre les notes).

Régler la distance entre les notes

Régler la vitesse d'affichage du motif

La vitesse d'affichage des notes du motif se règle avec la propriété `speed` envoyé dans les paramètres optionnelles de la création du motif :

```
maMelodie = MOTIF('<mes notes>', {speed:<vitesse>})
```

Avec `<vitesse>` qui correspond à peu près au nombre de notes affichées par secondes. Donc une vitesse de `1` affiche une note par secondes, une vitesse de `2` affiche 2 notes par seconde et une vitesse de `0.5` affiche 1 note en 2 secondes.

Les Accords

Table des matières

Création d'un accord

On crée un accord avec :

```
monAccord=CHORD('<note1> <note2>... <noteN>')
```

… où chaque note doit correspondre à la définition normale.

Par exemple :

```
accDom=CHORD('c3 eb3 g3')
```

Création d'un accord sur plusieurs portées

On peut poser l'accord sur différentes portées en ajoutant `<indice portée>:` devant la note (*rappel : l'indice portée est "1-start", donc "1" pour la première portée en comptant depuis le haut*).

Noter qu'il est inutile d'indiquer l'indice portée de la portée active.

Par exemple (en imaginant que la portée 1 est la portée active) :

```
acc=CHORD('2:c3 2:g3 e4 g4')
```

… placera "c3" et "g3" sur la 2^{e} portée (certainement la clé de fa) et les notes "e4" et "g4" sur la 1ère portée qui est la portée active.

Référence aux notes de l'accord

Comme tous les types d'objets possédant plusieurs notes (Accords, Gammes, Motifs), on fait appel aux différentes notes à l'aide de la méthode `note`

appelée sur l'objet :

```
<nom accord>.note(<indice note(s)>)
```

Par exemple :

```
accDom=CHORD('c3 eb3 g3')
accDom.note(1).move('c4')
// Prends la première note (c3) et la déplace en c4.
```

Voir la valeur que peut prendre `<indice note(s)>` dans Référence aux notes d'un objet pluri-notes.

Opération sur plusieurs notes

La méthode `note` de l'objet peut être également appelé avec une liste d'indice de notes. Toutes les notes subiront alors le même traitement.

Par exemple, pour masquer un ensemble de notes :

```
monAccord = CHORD('b3 d4 f4 a4')
WAIT(1)
monAccord.note([1,2,4]).hide()
# => masque la 1ère, 2e et 4e note
WAIT(2)
monAccord.note([1,2,4]).show()
# => ré-affiche les notes
```

Affecter une note à une variable

Comme pour tout “groupe de notes”, si l'on veut affecter une note de l'accord à une variable (pour l'utiliser plus facilement ensuite), on ne peut pas procéder ainsi :

```
maNote=monAccord.note(2)      # => # ERREUR #
maNote.write("Une seconde !") # => # ERREUR #
```

Pour ce faire, il faut impérativement utiliser :

```
maNote=Anim.Objects.monAccord.note(2)
maNote.write("Une seconde !")
```

Donc ajouter `Anim.Objects` devant `monAcccord.note(2)`, `Anim.Objects` étant la propriété qui contient tous les objets de l'animation.

On peut aussi utiliser la méthode pratique `objet(...)` :

```
maNote = objet('monAccord.note(2)')
```

Mais dans ce cas, noter qu'il faut absolument mettre le paramètre (la référence de l'objet) entre guillemets ou apostrophes.

Écrire le nom de l'accord

Une marque d'accord est un texte. Cf. Écrire un accord.

Écrire l'harmonie

Une marque d'harmonie est un texte. Cf.Écrire l'harmonie.

Écrire une cadence

Une cadence est un texte. Cf. Écrire une cadence.

Écrire une modulation

Une marque de modulation est un texte. Cf. Écrire une modulation.

Destruction d'un accord

Pour détruire l'accord, utiliser :

```
<nom variable accord>.remove()
```

Référence aux notes d'un objet pluri-notes

Un objet “pluri-notes” est un objet de l'animation qui possède et manipule plusieurs notes. C'est le cas par exemple des accords, des gammes ou les motifs.

Tous ces objets possèdent une méthode `note` qui permet de récupérer et de manipuler une ou plusieurs des notes de l'objet. Par exemple :

```
monAccord = CHORD('c4 e4 g4')
monAccord.note(2)
# => Retourne la deuxième note, le MI 4
# (noter que cette étape de l'animation ne fera rien puisqu
'on ne dit pas
#  ce qu'il faut faire avec cette note)
```

Les valeurs possibles de l'argument passé à la méthode `note` peuvent être :

Un indice unique

C'est alors l'indice de la note dans l'objet, la première note ayant l'indice 1

Une liste d'indices

Une liste des indices de notes à manipuler. Syntaxe : [``, `` etc].

Un rang de notes

Un range de notes où l'on définit la première et la dernière des notes à prendre. Syntaxe : `..`.

Par exemple :

```
monMotif = MOTIF('c4 d e f g a')

monMotif.note(5).surround()
# => Entoure la 5e note, donc le SOL 4
monMotif.note([1,3,5]).colorize(blue)
# => Met en bleu les notes 1 (DO 4), 3 (MI 4) et 5 (SOL 4)
monMotif.note('2..4').fantomize()
# => "Fantomise" les notes 2 à 4, donc du RÉ 4 au FA 4
```

Les portées

Table des matières

Créer une portée

Pour créer une portée, utiliser le pas :

```
NEW_STAFF(<cle>[, <params>])
```

… où `<cle>` peut être `SOL`, `FA`, `UT3`, `UT4`.

… et `<params>` peut contenir :

```
NEW_STAFF(<cle>, {
  (*)   offset    : decalage vertical par rapport à précéde
nte (pixels),
        y         : la position verticale précise (au lieu
de `offset`)
        x         : la position horizontale précise (bord g
auche par défaut)
        width     : la largeur de la portée (tout l'écran p
ar défaut)
  (**)  metrique  : la métrique à utiliser
  (***) armure    : l'armure à placer (tonalité)
})
```

(*) Noter que par défaut, il y aura toujours un décalage entre deux portées créées, ce `offset` ne fait qu'aggrandir l'espacement (s'il est positif) ou le rétrécir (s'il est négatif)*

(*) *Une valeur de type "4/4" ou "3/8" etc. Cf. Définir la métrique de la portée*

(*) *La tonalité, par exemple "A", "Bb" ou "C#". Cf. Définir l'armure de la portée*

Par exemple :

```
NEW_STAFF(SOL)
```

… qui affichera une portée en clé de sol en dessous de la dernière portée.

Noter que cette portée deviendra la portée active, c'est-à-dire celle où seront placées les objets définis par la suite.

Activer une portée

Activer une portée signifie que tous les pas suivants la viseront. Par exemple, les notes se déposent toujours sur la portée active.

```
ACTIVE_STAFF(<indice de la portee>)
```

… où `<indice de la portee>` est son rang dans l'affichage, en partant de 1 et du haut. Donc la portée la plus en haut s'active par :

```
ACTIVE_STAFF(1)
```

Définir l'armure de la portée

On définit l'armure de la portée avec le paramètre `armure` qu'on renseigne avec le nom de la tonalité exprimée au format de l'application, c'est-à-dire une lettre minuscule pour la note (de "a" à "g"), la lettre "b" pour "bémol" ou "d" pour "dièse".

Par exemple, pour une armure de FA# majeur avec une clé d'UT3 :

```
NEW_STAFF(UT3, {armure:'fd'})
```

Noter que l'animation en fonctionne pas comme un logiciel de PAO : il faut indiquer cette armure pour chaque portée sur laquelle on veut voir l'armure en question.

Définir la métrique de la portée

On définit la métrique avec le paramètre `metrique` qu'on renseigne avec les deux nombres qui composent la métrique séparés par une balance.

Par exemple pour une mesure en 6/8 :

```
NEW_STAFF(SOL, {metrique:"6/8"})
```

Récupérer une portée

On peut récupérer une portée (pour la “travailler” dessus) à l'aide de la commande :

```
STAFF(<indice 1-start>)
```

Par exemple, pour supprimer la deuxième porée construite :

```
STAFF(2).remove()
```

Noter que pour mettre la portée dans une variable, il faut impérativement utiliser :

maPortee=Anim.Objects.STAFF(x)

Supprimer des lignes supplémentaires

Pour le moment, la suppression de lignes supplémentaires n'est pas automatique, afin de laisser toute liberté à la programmation de l'animation.

Utiliser la commande :

```
STAFF(<indice 1-start>).remove_suplines({
  top     : <indice ou liste d’indice a supprimer au-dessus
>,
  bottom  : <indice ou liste d’indices a supprimer en desso
us>,
  xoffset : <decalage horizontal>
})
```

- N'utiliser que `top` et `bottom` au besoin ;
- Les indices des lignes se comptent À PARTIR DE LA PORTÉE (donc en montant pour `top` et en descendant pour `bottom`) ;
- `xoffset` n'est à préciser que si les lignes à supprimer ne se trouvent pas sur le décalage horizontal courant (suppression arrière, rare).

Autre commande pour supprimer des lignes supplémentaires

On détruit ces lignes à l'aide de la commande :

```
REMOVE_SUPLINE(<parameters>)
```

Une ligne supplémentaire est caractéristée par :

- La portée qui la porte ;
- Son indice à partir de la portée ;
- Sa position supérieure ou inférieure ;
- Son décalage à gauche (“frame” de l'animation).

Cela détermine les paramètres de `<parameters>`.

```
{
  staff: indice 1-start de la portée (depuis le haut),
  bottom: liste d'indices ou indice de la ligne à supprimer
 en bas,
  top: liste d'indices ou indice de la ligne à supprimer en
 haut,
  xoffset: décalage gauche (frame)
}
```

Toutes les valeurs à part `bottom` xou `top` sont optionnelles :

Si `staff` n'est pas précisé, on prendra la portée active.

Si `xoffset` n'est pas précisé, on prendra le décalage courant (ce qui représente le cas le plus fréquent, entendu qu'on va rarement supprimer une ligne supplémentaire "en arrière").

Note : lors d'un déplacement, une suppression ou tout autre effet qui doit rendre obsolète la ligne supplémentaire, il est préférable de déclencher la

suppression des lignes supplémentaires AVANT la commande sur la note. Par exemple, pour un déplacement :

```
note=NOTE(c4) // ajoute une ligne supplément en bas
WAIT(2)
REMOVE_SUPLINE({bottom:1})
note.move(c5)
```

Précision des indices

Les indices peuvent être une simple valeur numérique :

```
bottom:1 / top:1
```

… ou une liste d'indices

```
top:[1,2] / bottom:[1,2]
```

Noter que ces indices sont "1-start" et se comptent toujours À PARTIR DE LA portée, donc en descendant pour `bottom` et en montant pour `top` .

Noter aussi que `bottom` et `top` sont complètement indépendants, pour `bottom` on ne tient compte QUE des lignes supplémentaires inférieures et pour `top` on ne tient compte QUE des lignes supplémentaires supérieures.

Écrire un texte sur la portée

- Écrire le texte au curseur
- Régler l'écart entre texte et portée
- Décaler le texte à droite ou à gauche
- Définir la largeur du texte
- Utiliser un style de texte
- Positionner le texte en dessous de la portée

Écrire un texte au curseur

Pour écrire un texte à la position actuelle du curseur, on utilise la commande :

```
STAFF(<indice portée>).write("<le texte>"[, <paramètres optionnels>])
```

Par défaut, ce texte se place au-dessus de la portée.

Régler l'écart entre texte et portée

Pour le régler à partir de maintenant et pour tous les textes de portée qui suivront, utiliser la commande :

```
DEFAULT('staff_top_text', <nombre de pixels>)
```

Pour revenir à la valeur par défaut :

```
DEFAULT('staff_top_text')
```

Pour régler seulement un texte en particulier, l'écrire en défissant la propriété `offset_y` des paramètres (2e argument). Par exemple :

```
STAFF(1).write("Mon texte 20 pixels plus haut", {offset_y: 20})
```

Décale le texte à droite ou à gauche

Par défaut, le texte s'aligne par le centre à la position courante du curseur. On peut afficher ce décalage à l'aide de la propriété `offset_x` envoyé aux paramètres optionnels. Par exemple :

```
STAFF(2).write("Mon texte 30 pixels plus à droite", {offset_x: 30})
```

Bien entendu, on peut combiner tous les éléments :

```
STAFF(2).write("Mon texte modifié", {offset_y:10, offset_x: 20, style:'cadre'})
```

Définir la largeur du texte

On peut définir la largeur de la boite contenant le texte à l'aide de la propriété `width` :

```
STAFF(1).write("Texte à largeur", {width:<nombre de pixels>
})
```

Utiliser un style de texte

Pour la cohérence de l'affichage, on peut utiliser quelques styles de texte prédéfinis en réglant la propriété `style` des paramètres optionnels :

```
STAFF(1).write("Mon texte stylisé", {style:'<style(s) à uti
liser>'})
```

Si plusieurs styles doivent être combinés, les séparer par des espaces. Par exemple :

```
STAFF(2).write("Plusieurs styles compatibles", {style:'cadr
e exergue italic'})
```

Liste des styles :

```
italic    Place le texte en italique
cadre     Place un cadre autour du texte
exergue   Met le texte en exergue (gras)
```

Positionner le texte en dessous de la portée

DEFAULT('staff_text_up', false)

Pour le replacer en haut :

DEFAUT('staff_text_up')

Supprimer une portée

Comme pour tout objet de l'animation, on supprime une portée à l'aide de la méthode `remove` :

```
NEW_STAFF(SOL)
# => Crée une premiere portée
STAFF(1).remove()
# => Détruit la première portée
```

Attention cependant à un certain point : chaque fois qu'une portée supérieure est supprimée, l'indice des portées suivante est modifiée, pour tenir compte SEULEMENT des portées de l'animation courante.

Donc, si l'on a deux portées et que l'on veut les détruire les deux, le code suivant produira une erreur :

```
NEW_STAFF(SOL)
NEW_STAFF(FA)
....
STAFF(1).remove()
STAFF(2).remove()
=> !ERREUR : la première portée ayant été détruite, la port
ée d'indice `2` n'existe pas !
```

Il convient donc écrire :

```
STAFF(1).remove()
STAFF(1).remove()
```

Ou pour la clarté du code :

```
STAFF(2).remove()
STAFF(1).remove()
# => OK
```

Il en va de même, évidemment pour l'écriture de tout autre élément sur une portée après la suppression de portée supérieures.

Les gammes

Table des matières

Introduction

On peut produire en un seul pas une gamme à l'aide de la commande :

```
<var>=SCALE(<tonalité>[, <paramters>])
```

- `<var>` est un nom de variable quelconque
- `<tonalité>` est la tonalité exprimée par une seule lettre (anglaise) de "a" (la) à "g" (sol). On peut ajouter toutes les altérations voulues (cf. Les altérations). Noter que par défaut, suivant la portée active, l'animation affiche ses notes à la hauteur où elles produiront le moins de lignes supplémentaires.
- `<parameters>` est une liste de paramètres optionnels. Cf. ci-dessous.

Paramètres de définition des gammes

Ils constituent le second argument de la commande `SCALE`, après la note de la tonalité.

C'est un objet de propriétés :

```
maGamme=SCALE('a',{
  octave : 2,
  for    : 5,
  etc.
})
```

Liste des propriétés

type : {String}

Le type de gamme, parmi : 'MAJ' : Gamme majeure (défaut), 'min_h' : Mineure harmonique, 'min_ma' : MINeure Mélodique Ascendante, 'min_md' : MINeure Mélodique Descendante.

octave : {Number}

L'octave à laquelle il faut affiche la gamme. Par défaut, celui qui produira le moins de ligne supplémentaires, donc celui dont le plus grand nombre de notes se trouve *dans* la portée.

staff : {Number}

L'indice de la portée sur laquelle il faut écrire la gamme. Par défaut, la portée active.

offset : {Number}

Le décalage horizontal entre chaque note de la gamme.

asc : {Boolean}

Si TRUE (défaut), la gamme sera ascendante, sinon, elle descendra.

speed

La vitesse d'affichage de la gamme. La valeur correspond au nombre de notes qui seront afficher par seconde. Donc utiliser une valeur entre 0.01 et 1 pour afficher moins d'une note par seconde. Par exemple, la valeur `0.5` affichera 1 note toutes les deux secondes.

for : {Number}

Le nombre de notes de la gamme à afficher. Par défaut, 8 pour pouvoir les afficher toutes, de la tonique à la tonique.

from : {Number}

La première note de la gamme de laquelle partir (1-start)

Utilisation des notes de la gamme

Comme pour tout “groupe de notes” (accord, motif, etc.) les notes de la gamme peuvent être ensuite traitées séparément grâce à la méthode `note` appelé sur l'objet (ici appelée sur la gamme)

Soit une gamme :

```
maGamme=SCALE('d')
```

On récupère ses notes par :

```
maGamme.note(<indice note>)
```

Pour connaitre les différents moyens de faire référence aux notes, cf. Référence aux notes d'un objet pluri-notes

Par exemple, si je veux poser un texte sur la deuxième note :

```
maGamme.note(2).write("Une seconde !")
```

Comme pour tout “groupe de notes”, si l'on veut mettre un élément (note) du groupe dans une variable, ce code n'est pas possible :

```
maNote=maGamme.note(2)
maNote.write("Une seconde !") # => # ERREUR #
```

Pour ce faire, il faut impérativement utiliser :

```
maNote=Anim.Objects.maGamme.note(2)
maNote.write("Une seconde !")
```

Les Textes

Table des matières

Introduction

Il existe trois sortes de textes dans Staves :

- Les textes “portés” par les notes, les portées, les accords, tous les objets musicaux en somme, et comme les harmonies, les accords. De façon générale, ils sont créés à l'aide de la méthode `write` appelée sur l'objet (p.e. `maNote.write(....)`) ainsi que tous les méthodes-raccourcis comme la méthode `chord` qui permet d'écrire le nom de l'accord (p.e. `monAccord.chord("Do min.")`) ;
- Les textes de sous-titre ou de doublage qui permettent, comme leur nom l'indique, de créer des sous-titres ou des doublages à dire sur l'animation ;
- les “TBox(es)”, les boites de texte, indépendantes des objets musicaux, qui permettent d'afficher des textes à l'écran de façon indépendante des objets musicaux, avec un style propre.

Créer une boite de texte

Une “TBox”, une boite de texte, est un texte indépendant de l'animation, qui permet d'afficher un texte à n'importe quel endroit de l'animation, avec le style défini.

- Créer (instancier) la boite de texte
- Définir les valeurs par défaut des boites de texte
- Définir l'aspect de la boite de texte
- Styles de textes des boites de texte
- Définir les dimensions de la boite de texte
- Régler l'alignement du texte dans la boite
- Animer les boites de texte

Instancier la boite de texte

Une boite de texte (une “TBox”) se créer avec la commande `TBOX`

```
<variable> = TBOX("<Le texte>"[, <parametres>])
```

Par défaut, une boite de texte apparaitra toujours au milieu de l'écran, sur un fond translucide, avec un caractère lisible.

`<parametres>` peut être un “Hash” (`{<propriété>:<valeur>, <propriété>:<valeur> etc.}`) ou un “String” (une chaine de caractère). Si c'est un String, c'est le nom d'un style prédéfini de texte.

Par exemple :

```
monGrandTitre = TBOX("Mon grand titre en petites capitales"
, grand_titre)
```

Voir tous les Styles de texte des boites de texte.

Pour créer le texte sans l'afficher, ajouter le paramètre `hidden` à true :

```
txt = TBOX("Mon texte masqué", {hidden:true})
```

Cela permet par exemple de créer tous les textes au début du code de l'animation pour pouvoir en disposer ensuite grâce à la méthode `show` :

```
txt.show()
```

Ou de faire apparaitre le texte lentement :

```
txt = TBOX("Mon texte apparait lentement", {hidden:true}).s
how({duree:4})
# => Le texte apparait en 4 secondes
```

Un autre paramètre intéressant à la création est le paramètre `wait` (cf. Le paramètre spécial `wait`). Voir aussi le paramètre spécial `duree` .

Définir les valeurs par défaut des boites de texte

Toutes les valeurs par défaut se règlent à l'aide de la commande `DEFAULT` .

```
DEFAULT('tbox_font_family', "<police à utiliser>")
DEFAULT('tbox_font_size', <nombre de point de la taille de police>)
DEFAULT('tbox_padding', <nombre de pixel de marge intérieure>)
DEFAULT('text_color', '<couleur par défaut du texte>')
DEFAULT('tbox_background', "<couleur de background par défaut>")
DEFAULT('tbox_border', "<bord par défaut des boites de texte>")
```

Définir le style de la boite de texte

...

Définir la couleur de fond translucide

Pour définir la couleur de fond translucide (noire par défaut), on utilise le paramètre `background`.

```
tbox1 = TBOX("Mon texte", {background:<couleur #RRVVBB ou constante>})
```

Si la boite est déjà créée :

```
tbox1.set({background:<couleur>})
```

On peut jouer aussi sur l'opacité avec le paramètre `opacity`. Une opacité de 1 rendra complètement opaque le fond (aucune transparence), une opacité de 0.xx où `xx` est un nombre créera la transparence maximale.

```
tbox1 = TBOX("mon texte", {opacity:<nombre de 0 à 1>})
```

Si la boite est déjà créée :

```
tbox1.set({opacity:<nombre de 0 à 1>})
```

Définir les dimensions de la boite de texte

On définit les dimensions de la TBox à l'aide des paramètres `width` et `height`. Noter que par défaut la hauteur s'adaptera au texte contenu, mais pas la largeur, qui occupera la moitié de la largeur de l'écran.

```
boiteDimensionnee = TBOX("Une boite dimensionnée", {width:6
00, height:200})
# Une boite de texte qui fera 600 par 200 pixels.
```

Redéfinir les dimensions en cours d'animation

C'est la méthode `set` qui permet de modifier tous les paramètres de la TBox, à commencer par les dimensions.

On peut redéfinir ces dimensions de deux manières : de façon absolue ou de façon relative.

On redéfinit les dimensions de façon absolue avec les paramètres `width` et `height` :

```
maBoite.set({width:<nouvelle largeur>, height:<nouvelle hau
teur>})
```

On redéfinit les dimensions de façon relative avec les paramètres `offset_width` et `offset_heigth` :

```
<tbox>.set({
  offset_width  : <différence avec largeur courante>,
  offset_height : <différence avec hauteur courante>
})
```

Par exemple :

```
maboite.set({offset_width:20, offset_height:-11})
# => La boite sera allongée de 20 pixels et la hauteur sera
 raccourcie de 11 pixels
```

Définir l'alignement du texte dans la boite

Par défaut, l'alignement du texte est centré. On peut modifier cette alignement en utilisant le paramètre `align` en lui donnant la valeur “left” (alignement à

gauche), “right” (alignement à droite), “justify” (justification) et “center” (pour revenir au center).

Par exemple :

```
monTexte = TBOX("Mon texte aligné à gauche.", {align:left})
```

Styles de textes des boites de textes

Ces valeurs peuvent définir la propriété `style` dans les paramètres envoyés à TBOX (2e argument) ou servir de 2e argument si aucune autre propriété ne doit être définie.

Par exemple :

```
monTexte = TBOX("Mon petit texte", {style:small})
```

Ou :

```
monTexte = TBOX("Mon autre petit texte", small)
```

Liste des styles pré-définis

```
grand_titre       Un grand titre pour l'animation
chapitre          Un titre de chapitre
titre             Un autre titre dans un chapitre.
small             Un petit texte
tiny              Un texte minuscule
copyright         La marque du copyright
```

Animer les boites de texte

On peut animer les boites de texte avec la méthode `set` qui peut en redéfinir tous les paramètres.

Par exemple, si la boite possède un fond opaque noir (background:black, opacity:1) et qu'on veut le changer en fond rouge d'opacité 0.5, on utilise :

```
# -- Définition/création de la boite --
maBoite = TBOX("Ma boite", {background:black, opacity:1})
...
...
maBoite.set({background:red, opacity:0.5})
# Anime la boite en modifiant la couleur de fond (en vrai f
ondu) et son opacité
```

Déplacer les boites de texte

On peut utiliser la [méthode universelle `move`] pour déplacer les boites, mais on peut également utiliser là aussi la méthode `set` :

Déplacement vers des coordonnées absolues :

```
maBox.set({x:<valeur horizontale>, y:<valeur verticale>, du
ree:<en x secondes>})
```

… ou en coordonnées relatives :

```
maBox.set({
  offset_x    : <nombre de pixels de déplacement horizontal
>,
  offset_y    : <nombre de pixels de déplacement vertical>
})
```

Créer un texte

Types spéciaux de texte

Par défaut, un texte est "normal", il s'écrit tel qu'il est défini.

Mais il existe des types qui peuvent être définis grâce à la propriété `type` envoyée en paramètres :

chord

Le type `chord` permet d'écrire un accord au-dessus de l'élément porteur du texte. Il est stylisé en conséquence. Cf. Écrire un accord.

harmony

Écrit le texte sous la portée, sous forme d'une marque d'harmonie.
Si le texte se finit par un certain nombre de "*" ou de "•", ils sont considérés comme des renversements de l'accord et traités visuellement comme tels. Cf. Écrire l'harmonie.

cadence

Écrit le texte sous la portée, à la position courante, sous forme de marque cadentielle (donc avec des traits "__|" pour marquer la fin de la partie). Cf. Écrire une cadence.

modulation

Écrit le texte pour une modulation, au-dessus de la portée et de travers. Cf. Écrire une modulation.

part

Pour la marque d'une partie. Cf. Écrire une marque de partie

Écrire l'harmonie

Pour indiquer une harmonie, par exemple pour indiquer que l'accord est un premier degré sous la forme de son deuxième renversement :

```
monAccord=CHORD('g4 c5 e5')
monAccord.write("I**",{type:harmony})
```

Il existe aussi le raccourci :

```
monAccord.harmony("I**")
```

Pour le positionnement de la marque, cf. Position des textes d'harmonie et de cadence.

Écrire une cadence

Pour indiquer une **CADENCE** avec la méthode `write` :

```
monAccord.write("I", {type:cadence, type_cadence:<type de l
a cadence>})
```

Mais cette méthode peut être grandement simplifiée en utilisant la méthode `cadence` :

```
monAccord.cadence("I", {type:<type de la cadence>})
```

Noter qu'ici le `type_cadence` de la méthode `write` a été remplacé par `type`.

Les cadences possibles sont :

```
parfaite        V I
italienne       II|IV|VI V I
imparfaite      I V
plagale         IV I
picarde         V IMaj
demie           II V
rompue          V VI
faureenne       I IV V
```

Noter que ce sont des constantes, on peut les utiliser sans guillemets. Par exemple :

```
monAccord.cadence("I*", {type:imparfaite})
```

Pour une explication détaillée des cadences, cf. Page wiki sur les cadences)

Pour le positionnement de la marque, cf. Position des textes d'harmonie et de cadence.

Allonger la barre inférieure

Pour allonger la barre inférieur de la marque d'harmonie, on peut ajouter le paramètre `width` qui sera le nombre de pixels désiré. Par exemple :

```
monAccord.write("I**", {type:cadence, width:100})
OU
monAccord.cadence("I**", {width:100})
```

Marque à droite de la barre

Pour placer une marque à droite de la barre verticale, donc un texte concernant la suite, mettre ce texte entre crochets dans le texte de la cadence :

monAccord.cadence("I* [texte à droite]")

Ça peut être par exemple l'index de l'accord dans l'harmonie suivante.

Écrire un accord

Pour indiquer le **NOM DE L'ACCORD** au-dessus des notes (ou d'un autre objet) :

```
monAccord.write("C", {type:chord})
```

… ou le raccourci `chord` :

```
monAccord.chord("C")
```

Pour le positionnement de la marque de l'accord cf. Réglage de la position de l'accord.

Écrire une modulation

Pour écrire une modulation (au-dessus de la portée, nom de l'accord de travers), utiliser :

```
<objet>.modulation(<nom>[, <parameters>])
```

Par exemple :

```
monAccord = CHORD('c4 d4 fd4 a4')
monAccord.modulation('Sol')
```

Ou :

```
monAccord.write('Sol', {type:modulation})
```

Texte sous la barre

Le ton de la modulation est indiqué au-dessus de la barre transversal, mais un autre texte peut être écrit SOUS la barre. Pour ce faire, il suffit de le mettre entre crochets dans la définition de la modulation :

```
monAccord.modulation('Sol [Dom de Dom]')
```

… produira quelque chose comme (mais incliné) :

```
| SOL
|---------
| Dom de Dom
|
|
```

Pour le positionnement de la marque de modulation cf. Réglage de la position de la marque de modulation.

Écrire une marque de partie

Utiliser la méthode `part` sur un note, un accord etc. pour placer une marque de partie, avec en premier argument le nom de la partie et en second argument les paramètres optionnels.

```
<porteur>.part(<nom de partie>[, <parametres>])
```

Par exemple :

```
maMelodie = MOTIF('c3 c3 e3 d3 c3')
maMelodie.note(3).part(coda)
```

… écrira la partie “CODA” sur la troisième note de la mélodie.

Autre exemple :

```
STAFF(1).part(pont,{offset_x:10})
```

… écrira la marque de partie “PONT” sur la première portée (`STAFF(1)`) en la déplaçant de 10px vers la droite par rapport à la position courante du curseur de position.

Paramètres

Comme pour la plupart des méthodes, les paramètres se mettent entre crochets. Ces paramètres sont les suivants :

PART("Refrain",{ offset_x : décalage horizontal de la marque, offset_y : décalage vertical de la marque, })

Constantes partie

On peut utiliser ces constantes comme premier argument :

```
La marque…            … écrira :
--------------------------------
exposition            EXPOSITION
expo                  EXPO.
developpement         DÉVELOPPEMENT
development           DEVELOPMENT
dev                   DEV.
pont                  PONT
coda                  CODA
themea                THÈME A
tha                   TH. A
themeb                THÈME B
thb                   TH. B
themec                THÈME C
thc                   TH. C
refrain               REFRAIN
couplet               COUPLET
```

Si le nom de la partie n'est pas une constante, il faut la mettre entre

guillemets.

Réglage de la position de la marque de partie

On peut régler de façon absolue la marque de partie grâce à :

```
DEFAULT('part_y', <position verticale>)
DEFAULT('part_x', <position horizontale>)
```

On peut la définir par rapport au décalage actuel avec :

```
DEFAULT('offset_part_y', <decalage vertical>)
DEFAULT('offset_part_x', <decalage horizontal>)
```

On peut bien entendu, très ponctuellement, ajuster la position de la marque dans les paramètres envoyés à la méthode `part` (second argument), avec `offset_x` et `offset_y`.

Comme pour tout décalage vertical, une valeur positive éloigne de la portée tandis qu'une valeur négative rapproche de la portée.

Créer un texte pour l'animation

Utiliser la commande :

```
WRITE("<le texte>")
```

Créer un texte pour un objet

Pour associer un texte à un objet, il faut bien sûr créer l'objet puis ensuite appeler sa méthode `write` (écrire) :

```
maNote=NOTE(a4)
maNote.write("C'est un LA 4")
```

Créer des sous-titres

- La commande `CAPTION`
- Utilisation de sous-titre au lieu de doublage

- Désactiver les doublages en cours d'élaboration de l'animation
- Affichage temporisé du doublage
- Désactiver le doublage temporisé]
- Effacer le sous-titre ou le doublage

La commande `CAPTION`

On peut vouloir créer des sous-titres pour deux raisons principales :

1. Afficher des explications en même temps que l'animation joue (vrais sous-titres qui seront lisibles dans l'animation) ;
2. Comme pour un doublage, afficher un texte qui sera lu au cours de l'animation, donc affiché hors animation (qui ne sera pas visible dans l'animation finale).

Pour ces deux utilisations, on utilise la commande :

```
CAPTION(<texte>[, <parameters>])
```

Cette commande affiche le texte `<texte>` à l'écran en respectant les paramètres optionnels `<parameters>`.

Utilisation de sous-titres au lieu de doublages

Par défaut, le texte s'affichera comme un **texte de doublage**, donc hors du cadre de l'animation. Pour utiliser vraiment, PENDANT TOUTE L'ANIMATION, le texte en sous-titre (à l'intérieur de l'animation), alors définir :

```
DEFAULT('caption', true)
```

On peut également définir localement qu'un texte doit être un doublage ou non grâce à la propriété `caption` dans les paramètres :

```
CAPTION("Mon doublage", {caption:false})
ou
CAPTION("Mon doublage", {doublage:true})

CAPTION("Mon sous-titre", {caption:true})
```

De façon encore plus simple, on peut envoyer un deuxième argument `true` pour dire que c'est un sous-titre ou `false` pour dire que c'est un texte de doublage. Mais dans ce cas, bien entendu, aucun autre paramètre ne peut être transmis contrairement aux syntaxes précédentes.

Par exemple :

```
CAPTION("Mon doublage", false)

CAPTION("Mon sous-titre", true)
```

Affichage temporisé du doublage

Il est possible, au lieu d'afficher le texte des doublages comme un bloc, de les faire apparaitre petit à petit, au rythme de la parole.

Pour ce faire, ajouter au début de l'animation (ou à l'endroit où devra être activé cette fonctionnalité) :

```
DEFAULT('caption_timer', true)
```

On peut régler le débit de parole avec :

```
DEFAULT('caption_debit', <valeur>)
```

… où `<valeur>` est 1 par défaut, et le débit augmente à mesure que la valeur augmente et diminue quand la valeur se trouve entre 0 et 1 (p.e. `0.5` pour aller deux fois moins vite).

Quand le doublage est temporisé, on peut ajouter le paramètres `wait` au 2e argument de la commande `CAPTION`, ce qui aura pour effet de ne faire à l'étape suivante que lorsque tout le texte aura été dit.

```
CAPTION("Mon doublage temporisé", {wait:true})
```

Dans le cas contraire, l'animation passera à l'étape suivante tout en écrivant le texte de façon temporisé.

Affichage temporisé sur étapes suivantes

Plutôt que d'attendre la fin de l'écriture du doublage, on peut vouloir que l'animation joue certaines étapes sur l'écriture du doublage (donc sur le texte qui sera dit). Mais dans ce cas, si les étapes prennent moins de temps que l'écriture du doublage, et qu'un autre doublage est appelé, le premier doublage sera avorté, coupé avant la fin.

On peut remédier à cela grâce à la commande :

```
WAIT_CAPTION
```

… *(qu'on peut obtenir en autocomplétion par* `waic[TAB]` *)* qui va attendre, après les étapes d'animation, la fin de l'écriture du doublage avant de passer à la suite.

Par exemple :

```
# Il faut que les doublages soient "temporisés"
DEFAULT('caption_timer', true)

# ....
CAPTION("Ceci est un doublage qui sera dit sur les étapes s
uivantes.")

# Puisque CAPTION ci-dessus ne définit pas le paramètre {wa
it:true}, les
# étapes suivantes seront jouées pendant que le doublage s'
affichera
maGamme = SCALE('c')
maGamme.note([1,3,5]).surround()

# Même si les deux étapes ci-dessus se terminent avant le d
oublage, on
# attendra ici la fin de l'écriture du doublage avant de de
 passer à la suite
WAIT_CAPTION
```

Désactivation des doublages en cours d'élaboration de l'animation

Lorsqu'on est en train de mettre sur pied l'animation, il peut être intéressant de

déactiver les doublages lorsqu'ils sont temporisés avec un paramètre `wait` à `true` qui fait attendre la fin de l'affichage.

Pour les désactiver, actionner le menu “Options > Omettre les doublages”.

Pour remettre cette fonction en route, actionner le menu devenu “Options > Jouer les doublages”.

Désactiver le doublage temporisé

Pour désactiver le double temporisé à un moment de l'animation, écrire à l'endroit où la fonctionnalité doit être désactivée :

```
DEFAULT('caption_timer', false)
```

Noter aussi qu'on peut demander, pour se concentrer sur l'animation, demander à l'application d'omettre les doublages (cf. Omettre les doublages)

Effacer le sous-titre ou le doublage

Pour effacer le sous-titre ou le doublage, appeler la méthode sans argument ou utiliser un texte vide en premier argument. Par exemple, pour effacer le sous-titre actif :

```
CAPTION("", {caption:true})
CAPTION("", true)
```

Pour effacer le doublage :

```
CAPTION()
```

Définition des position des textes

On peut définir les positions de façon générale, dans les préférences. Cf. Réglage des valeurs par défaut. Par défaut, les réglages sont pensés pour être efficace dans la plupart des situations, mais on peut avoir à les affiner dans les cas spéciaux.

On peut également modifier la position d'un texte en particulier. Pour cela

voir : * Modifier la position verticale ; * Modifier la position horizontale) ; * Modifier la portée du texte.

Modification de la position verticale

La position verticale peut se régler avec la propriété `offset_y` définie dans le second argument des méthodes de texte.

Par exemple :

```
STAFF(1).write("Mon texte décalé", {offset_y: 50})
```

Cela aura pour effet de monter le texte de 50px par rapport à sa position "naturelle".

Noter pour tous ces réglages qu'une valeur positive ÉLOIGNE toujours le texte de la portée tandis qu'une valeur négative le RAPPROCHE.

Autre exemple pour place plus bas un texte d'harmonie d'un accord :

```
monAccord = CHORD('2:g3 2:b3 d3 g3')
monAccord.harmony("I", {offset_y:20})
```

La marque "I" de l'harmonie sera placé 20 pixels sous sa position naturelle.

Modifier la position horizontale

Définir la propriété `offset_x` dans les paramètres (2e argument) des méthodes de texte (`write` , `harmony` , etc.)

Par exemple pour écrire le texte 100px plus à gauche sur la deuxième portée :

```
STAFF(2).write("Mon texte décalé", {offset_x:-100})
```

Pour placer le texte d'une cadence 20 pixels plus loin (plus à droite) :

```
monAccord = CHORD('g3 b3 d3')
monAccord.cadence("I", {type:parfaite, offset_x:20})
```

Bien entendu, on peut combiner la décalage horizontal et vertical :

```
monAccord.chord("Cm7", {offset_y:10, offset_x:20})
```

… en évitant toutefois de trop modifier localement les valeurs, ce qui donnerait un moins bon aspect à l'animation. Il vaut mieux trouver des réglages généraux qui conviennent à la situation.

Modifier la portée du texte

Par défaut, la portée du texte est celle de son “possesseur” (l'accord, la note, etc.) ou le cas échéant la portée courante.

On peut néanmoins définir explicitement la portée de référence en définissant la propriété `staff` dans les paramètres envoyés à la méthode de texte (2e argument), avec en valeur l'indice (à partir de 1 et du haut) de la portée.

Par exemple, imaginons deux portées, avec comme portée active la portée 1, c'est-à-dire celle du haut. Sur cette portée, on écrit un accord, et on veut écrire en dessous la cadence. Mais on aimerait que cette cadence, au lieu d'apparaitre sous la première portée (qui est la portée de l'accord qui porte le texte), devra apparaitre sous la deuxième. Voici le code :

```
NEW_STAFF(SOL)
NEW_STAFF(FA)
ACTIVE_STAFF(1)
monAccord = CHORD('c4 g4 e4')
monAccord.cadence("I", {staff:2, type:plagale})
```

Supprimer un texte

Supprimer un texte d'objet

Pour supprimer le texte de l'objet, c'est-à-dire de le faire disparaitre de l'affichage, utiliser la méthode `hide` (cacher) ou `remove` du texte de l'objet :

```
<note>.texte[<type>].hide() / remove()
```

Où `<type>` est le type de texte :

- 'regular' pour un texte normal (sans type)
- 'chord' pour un accord
- 'cadence' pour une cadence
- 'harmony' pour une harmonie
- 'modulation' pour une modulation.

Par exemple :

```
maNote=NOTE(a4)
# Écrire le texte
maNote.write("C'est un LA 4")
# Attendre 2 secondes
WAIT(2)
# Supprimer le texte
maNote.texte['regular'].hide()
```

Noter que cette méthode supprime l'affichage du texte, mais l'objet `texte` existe toujours pour l'objet et on peut le ré-utiliser plus tard, par exemple avec :

```
maNote.texte['regular'].show()
```

… qui ré-affichera ce texte.

En revanche, si on utilise :

```
maNote.texte['regular'].remove()
```

… alors l'objet sera vraiment détruit.

Les images

Table des matières

Ajouter une image

On peut insérer n'importe quelle image dans l'animation, par exemple une partition, grâce à la commande :

```
IMAGE(<parametres>)
```

Par exemple :

```
monImage = IMAGE({url:'path/to/monimage.png'})
```

Les paramètres peuvent être les suivants :

```
monImg=IMAGE({
  url           : {String} <chemin vers image> ATTRIBUT OBL
IGATOIRE
  x             : {Number} <position horizontale dans l'ani
mation>
  y             : {Number} <position verticale dans l'anima
tion>
  width         : {Number} <taille de l'image|auto>
  height        : {Number} <hauteur de l'image|auto>
  zoom          : {Number} Le pourcentage de zoom (surclass
e width et height)
  // Propriétés pour le fond
  padding       : {Number} La marge autour de l'image (bord
ure)
  bg_color      : {String} La couleur de fond de l'image et
 de la bordure
  bg_opacity    : {Float}  Opacité de la marge (0 -> 1)
  bg_image      : {Boolean} Si false, le fond de l'image n'
est pas opaque.
  // Propriétés pour le cadrage
  cadre_width   : {Number} <largeur de la portion à prendre
 dans l'image>
  cadre_height  : {Number} <hauteur de la portion à prendre
 dans l'image>
  inner_x       : {Number} <décalage horizontal de la porti
on d'image>
  inner_y       : {Number} <décalage vertical de la portion
 d'image
})
```

- *Note : Toutes les mesures s'expriment en pixels (mais sans 'px', juste le nombre de pixels).*

- Noter que si `width` et `height` ne sont pas fournis, c'est la taille original de l'image qui sera prise en référence). Si une seule des deux valeurs est fournie, l'autre sera calculée en conséquence par rapport à la dimension originale (sans déformation).

Pour des informations concernant le “recadrage de l'image”, cf. Modifier le cadrage de l'image.

Noter que l'image est aussitôt construite est insérée dans l'animation avec les

paramètres fournis, sauf si `build:false` est ajouté aux paramètres définissant l'image (cf. Ne pas construire l'image).

Modifier la taille de l'image

Pour modifier la taille de l'image, on peut jouer sur les paramètres `width` et `height` ou le paramètre `zoom`.

Lorsque seule une des deux valeurs `width` ou `height` est fournie, la seconde est calculée en fonction de la taille originale de l'image pour n'obtenir aucune déformation.

Lorsque `zoom` est fourni, c'est une valeur proportionnelle qui détermine le grossissement ou la dimunie de l'image. La valeur `1` met l'image à sa taille normale, la valeur `2` doublera la taille de l'image tandis que `0.5` l'affichera deux fois plus petite.

Modifier la position de l'image

Pour modifier la position de l'image, il faut jouer sur les paramètres `y` (haut) et `x` (gauche) de ses paramètres optionnels. Par exemple :

```
monImage=IMAGE({url:'path/to/image.png', y:100, x:200})
```

… placera l'image 100 pixels plus bas que le haut du cadre de l'animation et à 200 pixels du bord gauche.

Noter qu'il est extrêmement simple de connaitre les coordonnées de l'image : il suffit de la déplacer dans l'animation une fois qu'elle est affichée. Ses coordonnées s'affichent en bas de l'écran. Donc :

- Définir la commande d'affichage de l'image sans définir `y` et `x` ;
- Faire jouer l'animation jusqu'au moment où l'image s'affiche ;
- Se mettre en pause ;
- Déplacer l'image au bon endroit ;
- Relever les coordonnées qui s'affichent et définir `y` et `x` dans les paramètres de la commande IMAGE.

Déplacer l'image

Pour déplacer l'image (pas la positionner mais l'animer en la changeant de place), utiliser sa méthode `move` :

```
monImage = IMAGE({<parametres de l'image>})
monImage.move({paramètres du déplacement})
```

On peut déplacer l'image en déterminant sa position finale à l'aide des paramètres `x` (position horizontale) et `y` (position verticale). La valeur est en pixels.

```
monImage.move({x:300, y:100})
```

On peut déplacer l'image en déterminant le nombre de pixels de déplacement horizontal à l'aide de `for_x` ou vertical à l'aide de `for_y` :

```
monImage.move({for_x:10, for_y:100})
```

Ou en combinant les deux formules :

```
monImage.move({x:300, for_y:10})
```

On détermine le temps (durée) de déplacement de l'image à l'aide de la propriété `duree` qui détermine le nombre de secondes que mettra l'image pour atteindre sa nouvelle position (2 secondes par défaut) :

```
monImage.move({for_x:10, for_y:100, duree:1})
```

Zoom dans l'image

On peut faire un zoom dans l'image à l'aide de la méthode `zoom` :

```
monImage = IMAGE({... définition ...})
...
monImage.zoom(<paramètres du zoom>)
# => Zoom dans l'image
```

Pour le zoom le plus simple, un argument unique, de type nombre, va indiquer le taux de grossissement ou de diminution de l'image (par rapport à la taille

initiale). La valeur `1` fait revenir l'image à sa taille initiale, la valeur `2` double la taille de l'image, la valeur `0.5` diminue de moitié la taille de l'image.

Avec cet argument unique, le grossissement se fait avec un centre au centre de l'image.

Par exemple :

```
monImage.zoom(3)
# => Grossis 3 fois la taille originale de l'image
```

Pour exécuter un zoom plus complexe, on doit définir la largeur de portion d'image à voir au final, ainsi que le positionnement de l'image à l'aide des paramètres :

```
width     Largeur d'image (en taille réelle)
inner_x   Décalage horizontal par rapport au 0 de l'image
inner_y   Décalage vertical par rapport au 0 de l'image
```

Par exemple :

```
monImage.zoom({width:100, inner_x:10, inner_y:10})
```

Pour obtenir très facilement ces valeurs, il suffit d'éditer l'image en double-cliquant dessus (après avoir jouer l'animation jusqu'à la faire apparaitre), puis de grossir ou diminuer l'image à l'aide des boutons “+” et “-” (et les touches modificatrices), et de déplacer le cadre rouge (sans le redimensionner puisqu'il représente la taille actuelle du cadre) jusqu'à obtenir l'effet voulu.

Cliquer ensuite sur “code” et copier-coller les valeurs du champ “Données pour ZOOM”.

Modifier le cadrage de l'image

Plutôt que d'insérer tout un tas d'images, par exemple une image par portée de la partition, il est plus intéressant d'importer comme image la partition entière puis ensuite de “zoomer” (ou “cadrer”) sur certaines parties de l'image pour n'en faire apparaitre qu'une partie.

C'est ici que le cadrage entre en jeu.

Pour recadrer ou cadrer une image :

Si l'image est affichée (par exemple en fin d'animation ou après une pause), il suffit de double-cliquer dessus pour l'éditer (*Noter qu'il ne faut pas que la grille soit affichée pour pouvoir double-cliquer sur l'image. Pour masquer la grille : "Options > Masquer la grille"*).

Dans le cas contraire :

- Activer le menu "Outils > Cadrage image..." ;
- Si l'image n'apparait pas, cliquer le bouton "Chercher toutes les images dans le code" ;
- Cliquer sur l'aperçu de l'image à recadrer.
  => La fenêtre de recadrage s'ouvre.
- Utiliser le cadre rouge pour choisir un portion de l'image, celle qui sera visible. On peut le déplacer en cliquant à l'intérieur du cadre rouge puis en glissant la souris et on peut changer sa taille à l'aide du coin en bas à droite ;
- Cliquer ensuite sur le bouton outil (à droite) correspondant à la commande désirée (puisque plusieurs options sont possibles à ce niveau là, depuis le simple code pour instancier l'image avec ce cadrage jusqu'à la commande pour faire un travelling sur l'image).

Effectuer un travelling dans l'image

Si le cadrage de l'image le permet (ie si l'image réelle est plus grande que le cadrage qui est fait dedans), on peut exécuter un travelling grâce à la méthode `travelling` appliquée à l'image créée.

Noter que le "travelling" n'est pas à confondre avec le déplacement de l'image. Dans un déplacement de l'image (cf. Déplacer l'image), l'image bouge dans le cadre de l'animation, change de position. Dans un "travelling", l'image reste à la même position dans le cadre de l'animation, c'est son contenu qui se déplace, comme lors d'un travelling de cinéma.

Par exemple :

```
monImage = IMAGE({<parametres pour définir l'url et le cadr
age>})
...
monImage.travelling({<parametres du travelling>})
```

Cette méthode attend des paramètres qui vont définir le travelling. Ces paramètres sont :

```
<image>.travelling({
  x       : {Number} La nouvelle position horizontale du ca
drage
  OU
  for_x   : {Number} Le nombre de pixels de déplacement hor
izontal

  y       : {Number} La nouvelle position verticale du cadr
age
  OU
  for_y   : {Number} Le nombre de pixels de déplacement ver
tical

  width   : {Number} La largeur du nouveau cadre (optionnel
)
  height  : {Number} La hauteur du nouveau cadre (optionnel
)
  zoom    : {Number} Peut remplacer `width` et `height` (po
urcentage)

  duree : {Number} La durée du travelling en secondes (2 pa
r défaut).
})
```

Noter que pour obtenir les nouvelles coordonnées du cadrage, il suffit :

- d'éditer l'image (en double-cliquant dessus ou en utilisant le menu “Outils > Cadrage image...”) ;
- De déterminer le cadrage de fin du travelling ;
- De demander le code (bouton “-> Code” de l'édition) ;
- De copier-coller le code donné dans le cadre “Paramètres travelling”.

Bordure de couleur et flou autour de l'image

On peut définir une border de couleur et une opacité de cette bordure pour créer un effet de flou (transparence) autour de l'image.

Pour cela, on joue sur les paramètres :

```
padding         Nombre de pixels de la bordure
bg_color        La couleur de la bordure
bg_opacity      L'opacité de cette bordure
```

Par exemple, pour créer une transparence blanche bien visible autour de l'image :

```
monImage = IMAGE({url:'path/de/mon/image.png', padding:40,
bg_color:40, bg_opacity:0.7})
```

Noter que par défaut, le fond de l'image sera lui aussi de la couleur `bg_color`, mais complètement opaque.

Pour empêcher ce comportement, ajouter le paramètre :

```
bg_image:false
```

Modifier la source de l'image

Plutôt que de détruire une image pour la remplacer par une autre, on peut simplement changer la source de l'image courante.

On modifie la source (ie le fichier de l'image) à l'aide de sa méthode `src` :

```
monImage = IMAGE({url:'path/to/image_1.png'})
WAIT(2)
monImage.src('path/to/image_2.png')
```

Ne pas construire l'image

Pour ne pas construire l'image lors de son instanciation (par exemple pour pouvoir mettre la liste des images en début de code), ajouter `build:false` aux paramètres :

```
monImage = IMAGE({url:'path/to/monimage.png', build:false})
```

Puis ensuite, à l'endroit où l'image doit être construite, utiliser :

```
monImage.build()
```

Les flèches

Table des matières

Introduction

On peut créer des flèches indépendantes (cf. Flèches indépendantes) mais le plus judicieux est de les associer à des objets, à commencer par des notes ou des accords.

Associer des flèches à un objet

Pour associer une flèche à un objet quelconque (p.e. une note) on utilise la méthode (de l'objet) :

```
<objet>.arrow([<identifiant fleche>, ]<parameters>)
```

Par exemple, pour faire partir une flèche vers le bas depuis une note :

```
maNote=NOTE(a4)
maNote.arrow({color:red, angle:90})
```

… ce qui produira une flèche rouge, partant de la note avec un angle de 90 degré (cf. ci-dessous les angles).

On peut créer autant de flèches pour l'objet que l'on veut. Il suffit pour cela d'ajouter un identifiant (sans premier argument comme identifiant, c'est la flèche d'identifiant `0` qui est prise en compte).

Par exemple :

```
maNote = NOTE(c4)
maNote.arrow('droite rouge', {color:red, angle:0})
maNote.arrow('bas verte', {color:green, angle:45})
```

On fait ensuite référence à ces différentes flèches en faisant référence à leur identifiant :

```
maNote.arrow('bas verte').rotate(20)
# => Tourne la flèche 'bas verte' à l'angle 20
maNote.arrow('droite rouge').colorize(blue)
# => Colore la flèche 'droite rouge' en bleu.
```

Définition de la flèche

Lors de la création de la flèche avec la méthode `arrow` (ou ARROW pour une flèche indépendante) on peut envoyer ces paramètres optionnels à la méthode :

```
<objet>.arrow(['<id flèche>',]{
  width     : {Number} longueur fleche en pixels
  angle     : {Number} Angle en degres
  color     : {String} La couleur (constante ou string)
  offset_x  : {Number} Décalage horizontal par rapport à po
sition normale
  offset_y  : {Number} Décalage vertical par rapport à posi
tion normale
  y         : {Number} Placement vertical de la fleche (pix
els)
  x         : {Number} Placement horizontal de la fleche (p
ixels)
  height    : {Number} Hauteur de la fleche
})
```

`<objet>` est “l'objet porteur” de la flèche, c'est par exemple une note.**

```
maNote = NOTE(a4)
maNote.arrow('fleche vers droite', {angle:0})
```

`<id flèche>` est l'identifiant optionnel de la flèche, utile si l'objet porteur doit porter plusieurs flèches.

Les valeurs **y** et **x** sont calculées automatiquement pour que la flèche soit placée correctement suivant l'objet porteur. On peut ajuster ponctuellement la valeur avec **offset_x** et **offset_y**.

L'**angle** est de 0 degré par défaut, c'est-à-dire que la flèche sera horizontale et pointera à droite (pour une autre valeur cf. Angle des flèches).

Pour la définition de **color** (la couleur) cf. les constantes couleurs.

Méthodes d'animation des flèches

- Faire tourner la flèche
- Changer la taille de la flèche
- Déplacer la flèche
- Modifier la couleur de la flèche

Faire tourner la flèche

```
<fleche>.rotate(<angle>)
```

Par exemple, pour une flèche associée à une note :

```
maNote=NOTE(c4)
maNote.arrow()
maNote.arrow().rotate(45)
```

Changer la taille (longueur) de la flèche

```
<fleche>.size(<longueur de la fleche>)
```

Par exemple :

```
maNote=NOTE(b4)
maNote.arrow('growup', {color:bleu})
maNote.arrow('growup').size(100)
```

Produira une animation qui fera s'allonger (ou se rétrécir) la flèche de sa longueur actuelle à la longueur `100px` .

Noter que cette méthode **crée réellement une animation** c'est-à-dire fait varier sous nos yeux la taille de la flèche. Si on veut définir la taille de la flèche au départ, utiliser plutôt le paramètre `width` dans les paramètres envoyés à la création de la flèche (cf. définition de la flèche).

Déplacer la flèche

Une flèche se déplace à l'aide de la méthode :

```
<objet>.<fleche>([<id>]).move(<parameters>)
```

… où les paramètres peuvent être :

```
move({
  x : {Number} Déplacement horizontal (en pixels)
  y : {Number} Déplacement vertical (en pixels)
})
```

Par exemple, si une flèche associée à une note doit se déplacer en descendant et en se déplaçant vers la droite :

```
maNote=NOTE(g4)
maNote.arrow('bouge')
WAIT(2)
maNote.arrow('bouge').move({x:50, y:50}) <--
```

Une valeur positives produira toujours un déplacement vers la droite (->) pour `x` et un déplacement vers le bas pour `y`, un valeur négative produira un déplacement vers la gauche (<-) pour `x` et un déplacement vers le haut pour `y`.

Modifier la couleur de la flèche

Pour modifier la couleur de la flèche, utiliser la méthode `colorize` :

```
<objet porteur>.arrow([<id>]).colorize(<couleur>)
```

Par exemple :

```
monAccord = CHORD('c4 e4 g4')
monAccord.arrow({color:green})
# => Une première flèche verte
monAccord.arrow().colorize(blue)
# => La colorize en blue
```

Pour les constantes couleur utilisable cf. les constantes couleurs.

Angle des flèches

Repère pour la définition de l'angle d'une flèche :

```
angle = 0   => flèche horizontale pointant à droite
angle = 90  => flèche verticale pointant en bas
angle = -90 => flèche verticale pointant en haut
angle = 180 => flèche horizontale pointant à gauche
```

Noter que pour les valeurs entre 90 et -90 (donc pointant vers la gauche), il

faut modifier le `x` de la flèche pour qu'elle ne traverse pas la note.

Détruire la flèche

Pour détruire la flèche (la retirer de l'affichage), utiliser sa méthode `remove`.

Par exemple :

```
maNote=NOTE(c3)
maNote.arrow()
WAIT(2)
maNote.arrow().remove() <--
```

Flèches indépendantes

[TODO]

Les Boites et les cadre

Les boites (box) permettent de dessiner des boites et des cadres (de forme, de couleur et de tailles diverses) dans l'animation.

Table des matières

Créer une boite

On crée une boite à l'aide de la commande :

```
BOX(<parameters>)
```

`BOX` “hérite” des méthodes et des propriétés universelles de boites, donc peut être manipulé et défini avec ces méthodes et propriétés.

Types de boites

type:plain

Pour une boite pleine, qui recouvrira une portion de l'animation.

type:cadre

Pour dessiner un cadre dans l'animation. cf. Créer un cadre.

type:segment

Pour dessiner une forme de segment, c'est-à-dire une sorte de "U". cf. Créer un segment.

Créer un cadre

On crée un cadre à l'aide de la commande :

```
BOX({type:cadre[, <autres parametres>]})
```

On peut définir la largeur de bordure du cadre (3 pixels par défaut) à l'aide du paramètres `border` définissant le nombre de pixels.

Par exemple

```
monCadre = BOX({type:cadre, border:10})
# Pour un cadre de 10 pixels de large
```

Pour définir la couleur du cadre, définir le paramètre `color`.

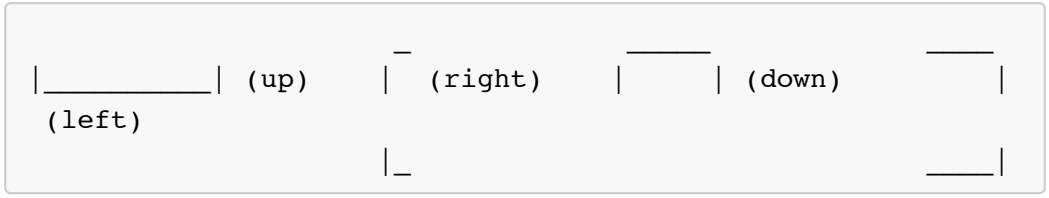
Par exemple :

```
cadreBleu = BOX({type:cadre, color:blue})
```

Pour les autres méthodes et propriétés cf. Méthodes et des propriétés universelles de boites.

Créer un segment "U"

Un “segment en U” est une forme qui ressemble à :



C'est grâce au paramètre `dir` qu'on détermine l'orientation du “U” :

```
seg = BOX({type:segment, dir:down})
```

Par défaut, un segment se présente avec les “fourches” vers le haut, comme un “U”, donc `dir:up` correspond à ne rien mettre du tout.

On peut décider de l'épaisser des branches à l'aide de `border` (nombre de pixels). Par défaut, l'épaisseur et de 1 pixel.

Noter que le paramètre pour régler la longueur des fourches varie en fonction de l'orientation. Pour un "U" vers le haut ou vers le bas, c'est la hauteur du segment qui importe, donc le paramètre `height` :

```
seg = BOX({type:segment, dir:down, height:20 /* fourches de
 20 pixels */})
```

Pour un "U" à droite ou à gauche, c'est le paramètre `width` qui déterminera les longueurs de fourche.

Pour les autres méthodes et propriétés cf. Méthodes et des propriétés universelles de boites.

Fond de l'animation

On peut définir le fond de l'animation avec la commande :

```
BACKGROUND([<couleur>][, <parametres>])
```

… où **couleur** est une constante couleur CSS (p.e. 'blue' ou 'red') et `<parametrs>` sont les paramètres optionnels. La couleur peut être omise (un seul argument paramètres) ou être mise dans les paramètres avec :

```
BACKGROUND({background:'<couleur>'})
```

`BACKGROUND` est un raccourci de `BOX` pour faire une boite qui occupe toute la surface de l'animation, avec un z-index de bas niveau. On peut donc utiliser les mêmes paramètres et les mêmes méthodes que les boites.

Méthodes et propriétés universelles de boites

Tous les objets qui héritent de ces méthodes et propriétés peuvent les utiliser. Par exemples les boites (Box) et les fond d'animation.

Table des matières

Résumé de toutes les propriétés

```
<var> = <OBJET>({
  x           : {Number} Position horizontale,
  y           : {Number} Position verticale,
  width       : {Number} Largeur,
  height      : {Number} Hauteur,
  z           : {Number} Placement devant/derrière,
  background  : {String} Couleur du fond,
  gradient    : {String} Couleur alternative pour le dégrad
é,
  opacity     : {Float}  Opacité,
  color       : {String} Couleur de la police ou du cadre,
  # Pour la méthode `set`
  offset_x    : {Number} Décalage horizontal
  offset_y    : {Number} Décalage vertical
  offset_w    : {Number} Delta largeur (modification relati
ve de la largeur)
  offset_h    : {Number} Delta hauteur (modification relati
ve de la hauteur)
})
```

Résumé de toutes les méthodes

set

Modifie une propriété quelconque ou plusieurs propriétés en même temps. cf. Modifier l'objet

move

Déplacement de l'objet. Cf. Déplacer l'objet. Noter qu'on peut objet la même chose avec `set`, avec la possibilité de modifier en même temps d'autres valeurs.

fade

Crée un fondu. cf. Fondu de l'objet

Effet de simultanéité

Un autre paramètre intéressant à la création d'un objet héritant des méthodes

et propriétés universelles de boite est le paramètre `wait`. S'il est faux (false) cela permet de passer à l'étape suivante sans attendre la fin de l'apparition de l'objet.

Cela permet des effets pour l'apparition quasi-simultanée de plusieurs boites ou animations diverses.

Par exemple :

```
boite1 = TBOX("Un premier texte", {wait:false, duree:4})
# La boite1 apparaitra en 4 secondes, mais on passe à la su
ite avant
# la fin de son apparition
boite2 = TBOX("Un autre texte", {wait:false, duree:3})
# La boite2 apparaitra en 3 secondes mais on passe à la sui
te immédiatement
boite3 = TBOX("Un troisième", {duree:2})
# La boite3 apparaitra en 2 secondes, mais on attendra la f
in de son apparition
# pour passer à la suite.
```

Régler la position horizontale

On la règle avec le paramètre `x` en fournissant un nombre (l'unité sera le pixel).

Par exemple :

```
maboite = BOX({x:100})
```

Pour obtenir cette position x, il suffit de déplacer l'objet (les coordonnées apparaissent dans la feedback en bas à gauche de l'écran). On peut également utiliser l'Outil “Coordonnées”.

Régler la position verticale

On la règle avec le paramètre `y` en fournissant un nombre (l'unité sera le pixel).

Par exemple :

```
maboite = BOX({y:100})
```

Pour obtenir cette position `y`, il suffit de déplacer l'objet (les coordonnées apparaissent dans la feedback en bas à gauche de l'écran). On peut également utiliser l'Outil “Coordonnées”.

Régler la position du calque

On peut régler la position de l'objet par rapport aux autres objets, dans l'axe `z` (ie. devant ou derrière) à l'aide du paramètre `z`.

Par exemple :

```
boitedevant   = BOX({z: 200})
boitederriere = BOX({z: 10})
# Même si elle a été créée après, `boitederriere` se retrou
vera
# derrière `boitedevant`.
```

Régler la largeur de l'objet

On la règle avec le paramètre `width` en fournissant un nombre (l'unité sera le pixel).

Par exemple :

```
maboite = BOX({width:100})
```

On peut obtenir cette valeur `width` à l'aide de l'Outil “Coordonnées”.

Régler la hauteur de l'objet

On la règle avec le paramètre `height` en fournissant un nombre (l'unité sera le pixel).

Par exemple :

```
maboite = BOX({height:100})
```

On peut obtenir cette valeur `height` à l'aide de l'Outil “Coordonnées”.

Régler la couleur de fond de l'objet

On la règle avec le paramètre `background` en fournissant une couleur ({String}). Soit une constante couleur CSS, soit une valeur hexadécimale précédée de '#' (p.e. `#FF0000`).

Par exemple :

```
maboite = BOX({background:'blue'})
```

Créer un dégradé

On crée un dégradé en utilisant les propriétés `background` qui définit la couleur gauche de départ et `gradient` qui définit la couleur droite d'arrivée.

Par exemple :

```
boitedegrade = BOX({background:'blue', gradient:'red'})
```

Ou :

```
boitedegrade = BOX({background:'#00F', gradient:'#F00'})
```

Régler l'opacité de l'objet

On la règle à l'aide du paramètre `opacity` en donnant une valeur flottante de `0` (complètement translucide, donc invisible) à `1` (complètement opaque).

Par exemple :

```
maboitetransparente = BOX({opacity:0.05})
```

Modifier l'objet

La méthode `set` invoquée sur l'objet permet de modifier n'importe quelle propriété et même plusieurs propriétés en même temps.

Syntaxe 1

```
<objet>.set('<propriété>', <nouvelle valeur>[, <parametres optionnels>])
```

Syntaxe 2

```
<objet>.set({<hash des propriétés>}[, <paramètres optionnels>])
```

Cf. les Propriétés modifiable de l'objet.

Par exemple, pour déplacer une boite pleine verticalement en modifiant son opacité :

```
maPlainBox.set({y:200, opacity:0.08})
```

Paramètres optionnels

Les deux paramètres optionnels sont `duree` qui détermine en combien de secondes devra se produire le changement et `wait` qui détermine soit que l'on doit passer tout de suite à l'étape suivante sans attendre la fin du changement (si `false`) soit le temps d'attente (si un nombre flottant ou entier de secondes).

Par exemple, pour modifier la position horizontale de l'objet (`x`) en 10 secondes et passer à l'étape suivante après 2 secondes :

```
monObjet.set('x', 320, {duree:10, wait:2})
```

ou :

```
monObjet.set({x:320}, {duree:10, wait:2})
```

Déplacement de l'objet

L'objet héritant des méthodes universelles de boite peut être déplacé à l'aide de la méthode `move`.

Cette méthode reçoit principalement trois arguments : `x` qui détermine la nouvelle position horizontale, `y` qui définit la nouvelle position verticale et `duree` qui définit la durée que devra prendre le déplacement, en secondes.

Par exemple :

```
maboite = BOX({x:100, y:10})
WAIT(1)
maboite.move({x:200, y:10, duree:10})
# La boite se déplacera vers la droite en 10 secondes
```

Autres paramètres

On peut également utiliser les paramètres `for_x` et `for_y` pour indiquer le nombre de pixels de déplacement par rapport à la position courante.

Par exemple :

```
maboite = BOX({x:100, y:20})
WAIT(1)
maboite.move({for_x:200, for_y:-10})
# La boite se retrouvera à la position x = 300 (100 + 200)
et à la
# position y = 10 (20 - 10)
```

Astuce pour définir le mouvement

Pour définir de façon simple le mouvement, le plus aisé est de déterminer la position x/y d'arrivée (simplement en déplaçant l'objet pour en prendre les coordonnées), puis de retirer la valeur voulue pour définir le point de départ (entendu que souvent le point d'arrivée est plus important).

Par exemple :

Après déplacement, j'ai déterminé que mes coordonnées d'arrivée sont x=320 et y=144.

Donc je code =

```
maboite.move({x:320, y:144})
```

Ensuite, j'ajoute la création de la boite AU-DESSUS de cette ligne avec un décalage :

```
maboite = BOX({x:320 - 100, y: 144 - 60})
WAIT(1)
maboite.move({x:320, y:144})
```

Fondu de l'objet

La méthode `fade` permet de fondre l'objet (noter que cela revient à faire un show ou hide très lent).

La méthode peut s'appeler sans argument, puisqu'elle détecte automatiquement le type de fondu (d'ouverture ou de fermeture) qu'il faut opérer.

Par exemple :

```
maboite = BOX({hidden:true})
# La boite sera masquée à la création grâce à `hidden:true`
maboite.fade()
# La boite apparaitra
WAIT(2)
# Et après 2 secondes…
maboite.fade()
# … la boite disparaitra
```

Destruction de l'objet

On détruit l'objet comme les autres objets grâce à la méthode `remove`.

Par exemple :

```
maBoite.remove()
```

Noter que la méthode passe à l'étape suivante sans attendre. Pour faire disparaitre l'objet de façon plus fluide, il est préférable d'appeler avant la

méthode `hide` ou `fade`. Par exemple :

```
maBoite.hide({duree:5})
... quelque chose qui se passe ici
maBoite.remove()
# Détruit la boite une fois qu'elle n'est plus visible à l'
écran
```

Régler les valeurs par défaut

On peut régler à tout moment (et en particulier au début de l'animation) toutes les valeurs par défaut, particulièrement tout ce qui concerne les positionnements.

Table des matières

Réglage de la vitesse

```
DEFAULT({speed: <coefficiant>})
```

Ou :

```
DEFAULT('speed', <coefficiant>)
```

… où `<coefficiant>` est un nombre différent de zéro, qui accélère l'animation s'il est < 1 (p.e. `0.5`) et qui la ralentit s'il est supérieur à 1 (p.e. `5`)

Pour remettre les valeurs par défaut

```
DEFAULT('speed')
```

Astuce pour le réglage des positions

Ci-dessous, on peut découvrir le fonctionnement de la commande `DEFAULT` pour redéfinir des valeurs de l'animation, et notamment des positionnements d'élément.

Pour pouvoir trouver la valeur-pixel à affecter, le mieux est de procéder ainsi :

1. Trouver dans ce manuel le paramètre à modifier pour obtenir le positionnement recherché.
2. Une fois ce paramètre déterminé, regarder sa valeur actuelle dans le bloc des infos de l'animation (qui doit se trouver en bas à droite de la page). Au bout d'une ligne commençant par ce paramètre, vous pouvez trouver sa valeur.
3. Ensuite, déplacer la règle de mesure (en bas à gauche de l'animation — l'animation, pas la page) à l'endroit du changement pour estimer les nouvelles valeurs à donner (50, 100 et 150 sont en pixels).
4. Régler le paramètre avec la valeur estimée et essayer.

Noter qu'on peut aussi, grâce à l'outil “Coordonnées” obtenir les coordonnées exacts des éléments à l'écran (cf. Outil “Coordonnées”).

Réglage de la position des portées

Pour régler la position verticale de la première portée :

```
DEFAULT('staff_top', <nombre de pixels>)
```

Ou :

```
DEFAULT({staff_top: <nombre de pixels>})
```

Pour remettre la valeur par défaut :

```
DEFAULT('staff_top')
```

Noter que si cette valeur est modifiée en cours d'animation (à la volée), cela affectera la position des portées suivantes.

Pour régler l'espace vertical entre les portées :

```
DEFAULT('staff_offset', <nombre de pixels>)
```

Ou :

```
DEFAULT({staff_offset: <nombre de pixels>})
```

Pour remettre la valeur par défaut :

```
DEFAULT('staff_offset')
```

Réglage de la position horizontale initiale

Cette position correspond à la position du “curseur”, c'est-à-dire l'endroit où seront marqués les premiers éléments sur la portée (hors clé, armure et métrique).

```
DEFAULT('x_start', <nombre pixels>)
```

Ou :

```
DEFAULT({x_start: <nombre pixels>})
```

Pour remettre la valeur par défaut :

```
DEFAULT('x_start')
```

Pour déterminer facilement la valeur, cf. Astuces pour le réglage des positions.

Réglage de la position `next`

La position `next` détermine de combien de pixels le “curseur” se déplacera à droite lorsque la commande `NEXT` sera utilisée (déterminant la position où seront créés les prochains objets)

Pour la régler de façon absolue

```
DEFAULT({next: <nombre pixels>})
```

Ou :

```
DEFAULT('next', <nombre pixels>)
```

Pour la régler de façon relative

(par rapport au `next` courant)

```
DEFAULT({offset_next: <nombre de pixels>})
```

Ou :

```
DEFAULT('offset_next', <nombre pixels>)
```

Pour remettre les valeurs par défaut

```
DEFAULT('next')
DEFAULT('offset_next')
```

Pour déterminer facilement la valeur, cf. Astuces pour le réglage des positions.

Réglage de la position des textes d'harmonie et de cadence

(Pour les marques d'harmonie et de cadence, cf. Types spéciaux de texte)

Pour la régler de façon absolue

```
DEFAULT({harmony: <nombre de pixels>})
```

Ou :

```
DEFAULT('harmony', <nombre pixels>)
```

Pour la régler de façon relative (par rapport à décalage courant)

```
DEFAULT({offset_harmony: <nombre de pixels>})
```

Ou :

```
DEFAULT('offset_harmony', <nombre pixels>)
```

On peut utiliser aussi la commande spéciale :

```
OFFSET_HARMONY(<nombre de pixels>)
```

Pour remettre les valeurs par défaut

```
DEFAULT('harmony')
DEFAULT('offset_harmony')
```

Pour déterminer facilement la valeur, cf. Astuces pour le réglage des positions.

Placement des textes d'harmonie et de cadenece sur une portée précise

Par défaut, les marques d'harmonie et de cadence se place sous la portée de l'objet qui possède ces textes (l'accord, la note, etc.). On peut cependant forcer l'affichage de ce texte, de façon générale (c'est-à-dire pour tous les textes d'harmonie et de cadence à partir de cette définition) grâce au réglage de :

```
DEFAULT('staff_harmony', <indice portée>)
```

Par exemple, pour que ces textes s'affichent sur la 2e portée, même quand la première portée est active ou que l'objet qui porte le texte se trouve sur la première portée :

```
DEFAULT('staff_harmony', 2)
```

Après cette définition, toutes les marques d'harmonie et de cadence se placeront sous la deuxième portée.

Par rappel, on peut aussi définir cette position au moment de la définition de l'harmonie ou de la cadence grâce à la propriété `staff` :

```
monAccord.harmony("V**", {staff:2})
```

… mais ça ne modifie la position QUE pour cette marque d'harmonie.

Réglage de la position des marques d'accord

(Pour les marques d'accords, cf. Types spéciaux de texte)

Pour régler la position de façon absolue

```
DEFAULT({chord: <nombre de pixels>})
```

Ou :

```
DEFAULT('chord', <nombre pixels>)
```

Pour régler la position de façon relative (par rapport à décalage courant)

```
DEFAULT({offset_chord: <nombre de pixels>})
```

Ou :

```
DEFAULT('offset_chord', <nombre pixels>)
```

On peut utiliser aussi la commande spéciale :

```
OFFSET_CHORD_MARK(<nombre de pixels>)
```

Pour remettre les valeurs par défaut

```
DEFAULT('chord')
DEFAULT('offset_chord')
```

Pour déterminer facilement la valeur, cf. Astuces pour le réglage des positions.

Position de la marque de modulation

Pour placer la marque de modulation, on peut utiliser pour la position verticale et la position horizontale une valeur absolue ou une valeur relative (relative aux positions actuelles).

Régler la position horizontale absolue

```
DEFAULT('modulation_x', <nombre de pixels>)
```

Ce `<nombre de pixels>` sera retiré au `x`, donc si le nombre est positif, la marque recule vers la gauche.

Pour revenir à la position par défaut :

```
DEFAULT('modulation_x')
```

Pour déterminer facilement la valeur, cf. Astuces pour le réglage des positions.

Régler la position horizontale relative

Pour modifier la position relativement à la position courante :

```
DEFAULT('offset_modulation_x', <nombre de pixels de decalag
e>)
```

Régler la position verticale absolue

```
DEFAULT('modulation_y', <pixels par rapport au haut de la p
ortée>)
```

Régler la position verticale relative

```
DEFAULT('offset_modulation_y', <décalage par rapport à la p
osition courante>)
```

Un nombre positif baissera la marque, un nombre négative la remontera.

Réglage de la taille des notes

Note : C'est une donnée qu'il vaut mieux ne pas modifier puisqu'elle est calculée automatiquement par rapport à la taille de la portée.

```
DEFAULT('note_size', <hauteur en pixel (flottant)>)
```

Ou :

```
DEFAULT({note_size: <hauteur en pixel>})
```

Pour remettre la valeur par défaut :

```
DEFAULT('note_size')
```

Ré-initialiser toutes les valeurs par défaut

Pour remettre toutes les valeurs aux valeurs par défaut, utiliser la commande (SANS PARENTHÈSES) :

```
RESET_PREFERENCES
```

Exécuter du code

La commande `EXEC` permet d'exécuter du code. C'est une commande de débuggage.

Par exemple :

```
EXEC('now_start()')
# => Place un point de départ temporel
... animation ...
EXEC('now()')
# => Écrit en console le nombre de secondes et milliseconde
s écoulées
#    depuis `now_start`
```

Pour écrire un message en console depuis le code, on peut utiliser :

```
EXEC('dlog("<le message à écrire>")')
```

Annexe

Constantes couleurs

La **color** (couleur) d'un objet quelconque est noire par défaut (une note, une flèche, etc.).

Pour modifier sa couleur, on peut utiliser ces constantes couleur (les valeurs françaises sont des valeurs valides aussi) :

```
black   noir
white   blanc
red     rouge
blue    bleu
green   vert
grey    gris
orange

background_color    La couleur de fond de l'animation (blan
c cassé)
```