Staff Animation

(Animation de portée)

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

Création d'une 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'apparitioin d'une portée ;
- L'écriture d'une note ou d'un accord sur la portée ;
- Le déplacement de notes ;
- L'écriture de textes explicatifs ;
- 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.

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.

Se déplacer sur la portée

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

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

Par défaut (sans argument), le déplacement sera de 40px (`Anim.defaut.hoffset`). Sinon, le déplacement sera de la valeur précisée.

Par exemple :

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

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

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
```

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 “Start”).

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
```

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` .

Les Notes

Table des matières

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*).

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

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)
- Une telle séquence (un pas) ne doit pas comporter d'espaces.

Déplacer une note

Pour déplacer une note, on utilise le pas :

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

Exemple :

```
maNote=NOTE(a4) // crée la note LA 4
maNote.moveTo(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
WAIT(4)
maNote.unexergue()  # => etat normal
```

Cf. aussi Entourer une note ci-dessous.

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()
```

Détruire une note

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

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

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 2e 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>)
```

… où `<indice note>` est l'indice 1-start (1 = première note fournie).

Par exemple :

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

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)
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.

Destruction d'un accord

Pour détruire l'accord, utiliser :

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

Les portées

Table des matières

Créer une portée

Pour afficher une portée, utiliser le pas :

```
NEW_STAFF(<clé>)
```

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

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)
```

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.

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.moveTo(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.

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 pour commencer la gamme. Par défaut le décalage courant (la position du “pointeur”).

asc : {Boolean}

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

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>)
```

Cet `indice` est “1-start”, c'est-à-dire que la première note porte l'indice 1, la seconde note porte l'indice 2, etc.

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

Les textes peuvent exister pour l'animation en général (ils sont alors écrits en haut à gauche et chaque nouveau texte remplace l'ancien), mais ils peuvent être associés aussi à tout objet de l'animation, note, portée, mesure, barre, etc.

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 :

harmony

Écrit le texte sous la portée, sous forme d'une marque d'harmonie cadencielle.
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.

chord

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

Par exemple, pour indiquer que l'accord est un premier degré sous son deuxième renversement :

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

Il existe aussi le raccourci :

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

Noter que 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:harmony, width:100})
OU
monAccord.harmony("I**", {width:100})
```

Pour indiquer une **CADENCE** (ajout de barre en bas et sur le bord droit du texte) :

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

Pour indiquer le **NOM DE L'ACCORD** au-dessus des notes :

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

… ou le raccourci `chord_mark` :

```
monAccord.chord_mark("C")
```

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")
```

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) du texte de l'objet :

```
<note>.texte.hide()
```

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.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.show()
```

… qui ré-affichera ce texte.

Définir les valeurs par défaut

Noter que toutes ces valeurs peuvent être redéfinies “à la volée“ en cours d'animation.

Décalage des marques de l'harmonie par rapport à la portée

Une valeur positive fera descendre les marques, les éloignant de la portée, une valeur négative rapprochera les marques de la portée.

```
OFFSET_HARMONY(<decalage par rapport a valeur actuelle>)
```

Décalage des marques des accords au-dessus de la portée

Une valeur positive fera MONTER l'accord, l'éloignant de la portée, une valeur négative rapprochera la marque de la portée.

```
OFFSET_CHORD_MARK(<decalage par rapport à la valeur actuell
e>)
```

Ré-initialiser toutes les valeurs de préférence

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

```
RESET_PREFERENCES
```

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.

Associer une flèche à un objet

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

```
arrow(<parameters>)
```

Par exemple, pour faire partir une flèche d'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).

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 :

```
maNote.arrow({
  width:  {Number} longueur fleche en pixels
  angle:  {Number} Angle en degres
  color:  {String} La couleur (constante ou string)
  top:    {Number} Placement vertical de la fleche (pixels)
  left:   {Number} Placement horizontal de la fleche (pixel
s)
  height: {Number} Hauteur de la fleche
})
```

Les valeurs **top** et **left** sont calculées automatiquement pour que la flèche soit placée correctement à droite de l'objet.

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).

La **color** (couleur) est noire par défaut. On peut utiliser toutes les constantes de couleur pré-définies, c'est-à-dire (les valeurs françaises sont des valeurs valides aussi) :

```
black   noir
red     rouge
blue    bleu
green   vert
orange
```

Méthodes d'animation des flèches

- Faire tourner la flèche
- Changer la taille de la flèche
- Déplacer 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({color:bleu})
maNote.arrow.size(100)
```

Produira une animation qui fera s'allonger 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 :

```
<fleche>.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()
WAIT(2)
maNote.arrow.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`.

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 `left` 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]