

UPMC/Licence/Info/2I013

Musidroid

BL/PM

Janvier 2018

Le but de l'application est de réaliser une interface graphique de création de morceaux de musiques qui nous a été inspirée par <http://www.google.com/logos/doodles/2017/fischinger/fischinger17.9.html>

Des grilles de notes Pour pouvoir créer un morceau de musique, on dispose d'un ensemble de grilles de points. Chaque ligne d'une grille correspond à la hauteur d'une note (l'écart entre deux lignes est d'un demi ton). Chaque colonne d'une grille correspond au moment où les notes sont jouées : chaque colonne représente un *instant*. Chaque point d'une grille peut être activé ou désactivé. Par exemple, la grille

(si#)	○	○	○	○	○	○	○
(si)	○	○	○	○	○	○	●
(la#)	○	○	○	○	○	○	○
(la)	○	○	○	○	○	●	○
(sol#)	○	○	○	○	○	○	○
(sol)	○	○	○	○	●	○	○
(fa#)	○	○	○	○	○	○	○
(fa)	○	○	○	●	○	○	○
(mi)	○	○	●	○	○	○	○
(re#)	○	○	○	○	○	○	○
(re)	○	●	○	○	○	○	○
(do#)	○	○	○	○	○	○	○
(do)	●	○	○	○	○	○	○

où les points noirs (●) sont activés, joue la gamme *do, ré, mi, fa, sol, la, si*.

Plus précisément, *do* est joué à l'instant 0, *ré* est joué à l'instant 1, *mi* est joué à l'instant 2, etc. et chacune de ces notes a une *durée* de 1.

Plusieurs notes d'une même grille peuvent être jouées ensemble (au même instant) ce qui permet d'obtenir des *accords*, et, un note peut s'étendre sur plusieurs instants, ce qui permet d'obtenir des durées différentes pour les notes. Exemple de grille présentant des accords et des durées de notes différentes :

(si#)	○	○	○	○	○	○	○
(si)	○	○	●	○	○	○	○
(la#)	○	○	○	○	○	○	○
(la)	○	○	○	■	○	○	○
(sol#)	○	○	○	○	○	●	○
(sol)	■	○	●	○	○	○	○
(fa#)	○	○	○	○	○	○	○
(fa)	○	○	○	○	○	●	○
(mi)	■	○	■	○	○	○	○
(re#)	○	○	○	○	○	○	○
(re)	○	○	●	○	○	○	○
(do#)	○	○	○	○	○	○	○
(do)	■	○	■	○	●	○	○

Ici, *do*, *mi* et *sol* sont joués à l’instant 0 et ont une durée de 2 ; *ré*, *sol* et *si* jouées à l’instant 3 et on une durée de 1 ; etc.

Toutes les grilles ont le même nombre de lignes et le même nombre de colonnes.

On attribue à chaque grille un *instrument* et une *octave* qui lui sont propres, ce qui permet d’obtenir des *polyphonies*. Les octaves sont numérotées de -1 à 10.

Toutes les grilles sont jouées ensemble selon une *cadence* commune. Une cadence est un entiers strictement positif. Plus l’entier est grand, plus la cadence est rapide.

Un modèle Avant de pouvoir jouer un morceau de musique, on en construit un *modèle* à partir des données de l’interface graphique.

Le modèle d’un morceau de musique est basé sur les éléments suivants :

- les notes qui sont caractérisées par leur hauteur, l’instant où elle sont jouées et leur durée ;
- les grilles qui sont caractérisées par l’instrument qu’elles jouent, l’octave et l’ensemble des notes jouées. Une grille de l’interface correspond à une *partie instrumentale* du morceau ; on emploiera de manière plus concise les termes d’*instrument* ou de *partie* pour désigner ces objets ;
- enfin, le morceau en son entier, qui est constitué de l’ensemble des grilles jouées ainsi que leur cadence commune ; on emploiera le terme de *partition* pour désigner cet objet.

Chacun de ces éléments devra correspondre à une classe :

- les notes : classe `Note` ;
- les parties instrumentales : classe `InstrumentPart` ;
- le morceau : classe `Partition`.

Les constructeurs et méthodes requises pour ces classes sont spécifiées dans la suite de ce document.

Pour définir ces classes on utilise deux types énumérés :

- `enum NoteName` qui définit les (noms des) notes ;
- `enum InstrumentName` qui définit les (noms des) instruments disponibles.

Ces énumérations sont fournies dans le *package* `12i013.musidroid.util`. Elles fournissent toutes deux une méthode `getNum()` qui donne le numéro (*ordinal*) des éléments de l’énumération. L’énumération `NoteName` fournit de surcroît la méthode statique `public static NoteName ofNum(int i)` qui donne la note correspondant au numéro passé en argument.

Des fichiers MIDI Les (modèles des) morceaux ne sont pas joués directement. Ils sont enregistrés dans un fichier au format *MIDI*, puis ce fichier est chargé pour être joué.

Pour engendrer les fichiers MIDI à partir du modèle, on utilise les ressources de la classe `MidiFile2I013` qui est fournie dans le *package* `12i013.musidroid.util`. Cette classe utilise elle-même le *package* `com.leff.midi`

dû à Alex Leffelman.

La classe `MidiFile2I013` fournit

- La méthode statique `public static void write(File f, Partition p)` qui crée le codage *MIDI* du modèle de la partition `p` et l'écrit dans le fichier `f`.

Spécification des classes du modèle

Les classes du modèle sont à ranger dans le *package* `l2i013.musidroid.model`.

Classe `Note`

- Constructeur : `public Note(int t, NoteName n, int d)` où `t` est l'instant, `n` le nom de la note et `d` sa durée. Les instants sont des entiers positifs ou nul (le premier instant est 0). Les durées sont des entiers strictement positifs.
- Méthode d'accès : `public int getInstant()` qui donne l'instant de la note ;
- Méthode d'accès : `public NoteName getName()` qui donne le nom de la note ;
- Méthode d'accès : `public int getDuration()` qui donne la durée de la note.

Classe `InstrumentPart`

- Constructeur : `public InstrumentPart(InstrumentName n, int o)` où `n` est le nom de l'instrument et `o` son octave ; une octave est comprise entre 0 et 10 (inclus) ;
- Méthode d'accès : `public InstrumentName getInstrument()` qui donne le nom de l'instrument ;
- Méthode d'accès : `public int getInstrumentNum()` qui donne le numéro de l'instrument ;
- Méthode d'accès : `public int getOctave()` qui donne la valeur de l'octave ;
- Méthode d'accès : `public ArrayList<Note> getNotes()` qui donne l'ensemble des notes jouées par l'instrument. Les notes doivent être classées selon l'ordre croissant des instants et, pour un même instant, selon la hauteur des notes : c'est l'ordre *lexicographique* selon les instants et les hauteurs ;
- Méthode de modification : `public void setInstrument(InstrumentName n)` qui modifie la valeur du nom de l'instrument ;
- Méthode de modification : `public void setOctave(int o)` qui modifie la valeur de l'octave ;
- Méthode d'ajout : `public void addNote(int t, NoteName n, int d)` qui ajoute à l'ensemble des notes jouées la note de nom `n` à l'instant `t` avec une durée `d` ;
- Méthode de retrait : `public void removeNote(int t, NoteName n)` qui retire de l'ensemble des notes la note de nom `n` jouée à l'instant `t`. La méthode laisse l'ensemble de notes inchangé si n'y a pas de note `n` jouée à l'instant `t`.
- Méthode utilitaire : `public String toString()` donne une présentation des données de l'instance sous forme de chaîne de caractères.

Classe `Partition`

- Constructeur : `public Partition(int t)` où `t` est la cadence (*tempo*) du morceau ;
- Méthode d'accès : `public int getTempo()` qui donne la valeur de la cadence du morceau ;
- Méthode d'accès : `public InstrumentPart getPart(int i)` qui donne la partie de numéro `i` du morceau (voir *infra* : méthode d'ajout `addPart`). Si le morceau ne contient pas une telle partie, la valeur de retour est `null` ;
- Méthode d'accès : `public int getSize()` qui donne le nombre de parties instrumentales du morceau ;
- Méthode de modification : `public void setTempo(int t)` qui donne la valeur `t` à la cadence du morceau ;
- Méthode d'ajout : `public int addPart(InstrumentName n, int o)` qui ajoute à l'ensemble des parties du morceau une nouvelle partie pour l'instrument de nom `n` à l'octave `o`. La méthode donne

- en retour le numéro de la partie nouvellement créée. Si la partie ne peut être créée (par exemple, lorsque `o` est invalide), la valeur de retour de la méthode est `-1` ;
- Méthode d'ajout : `public void addNote(int i, int t, NoteName n, int d)` qui ajoute la note de nom `n` à l'instant `t` et de durée `d` à la partie numéro `i` du morceau. Si le morceau ne contient pas de partie numéro `i`, la méthode ne fait rien.
 - Méthode de retrait : `public void removePart(int i)` qui retire la partie numéro `i` de l'ensemble des parties du morceau. Si la partie de numéro `i` n'existe pas, la méthode ne fait rien.