Télécom SudParis
Année scolaire 2015/2016

Projet Informatique 1ère Année
CSC 3502

MusiCore

Aurélien BETTINI
Constantin VURLI
Anais VILLEDIEU
Victor PLICHART

Enseignant responsable : **Francois Trahay**

Sommaire

1 Introduction

Ce projet informatique consiste à réaliser une application permettant la création de listes de lecture de pistes musicales à partir d'une sélection de musiques fournie par l'utilisateur. Nous voulons faire en sorte que notre logiciel fournisse des playlists dites « intelligentes ».

En d'autres termes, notre logiciel devra être capable à partir de la bibliothèque musicale de l'utilisateur de sélectionner un ensemble de titres avec des caractéristiques musicales similaires afin de rendre l'écoute homogène.

Actuellement, les plateformes d'écoute musicale en streaming proposent à l'utilisateur des titres musicaux relatifs à ceux que l'utilisateur a déjà écouté. En revanche, la création de playlists automatiques est inexistante, au mieux il s'agit d'une répartition aléatoire des titres proposés par l'utilisateur.

2 Cahier des charges

Il s'agit d'écrire une application qui sera capable, à partir d'une bibliothèque de musique fournit par l'utilisateur de créer une playlist dites « intelligentes » de toutes ses musiques afin que l'écoute soit homogène.

Fonctions du logiciel :

- A partir d'une sélection de morceaux musicaux fournis par l'utilisateur, le logiciel doit être capable d'extraire les propriétés musicales de chacun d'entre eux. On retrouve notamment comme caractéristiques principales le BPM (Beats par minute) du morceau et son harmonique principale.

Le logiciel doit être capable de stocker les données de chaque morceau et de les organiser.

L'utilisateur pourra choisir indépendamment les critères d'analyse utilisé par l'algorithme de création de la playlist musicale : Tempo (Beat per minutes, harmoniques).

A minima notre logiciel doit être capable de retourner un fichier texte contenant dans l'ordre les titres musicaux de la playlist créée ou de créer un fichier playlist qui puisse être lu par VLC par exemple.

Dans une optique de développement optimal, notre logiciel sera capable de lire la playlist créée via un lecteur audio intégré. Ce lecteur audio contiendra toutes les fonctionnalités de base d'un lecteur audio (lecture, pause, changement de morceau…).

Ce lecteur audio devra supporter les transitions en tempo entre les musiques. Le tempo des deux morceaux devra s'accorder au moment de la transition.

Toutes ces fonctionnalités doivent sont à implantés avec une interface graphique.

2.1.1 Contraintes techniques

Les contraintes techniques sont les suivantes :
- Notre logiciel doit être capable de gérer différents formats audio (.mp3, .wav)
- Notre logiciel doit être capable d'analyser les morceaux et de générer la playlist dans un temps acceptable. Comme critère de mesure on pourra utiliser le rapport N/T avec N le nombre de morceaux analysés et T le temps de calcul (on peut aussi raisonner avec le poids en Mb de chaque morceau).
- Le langage de développement choisi est Python 3
- Le logiciel final doit être utilisable sous tous les systèmes d'exploitation possédant Python3 (Linux, MacOS, Windows).
- Interface graphique réalisée avec GTK 3+.

2.1.2 Recette

Le logiciel doit comporter un menu permettant de quitter l'application et de lancer autant d'analyse que possible. La simulation doit pouvoir permettre à l'utilisateur de sélectionner les fichiers audio qu'il souhaite et de lancer une analyse de ses fichiers (durée de la musique, rythme de la musique ou bpm ainsi que la tonalité).

3 Développement

3.1 Analyse du problème et spécification fonctionnelle

3.1.1 Cas d'utilisation

L'application permet d'analyser des musiques et de créer une playlist de musiques à partir de musiques que fournira l'utilisateur.

3.1.2 Fonctionnalités de l'application

L'application permet d'interagir avec un menu pour quitter ou importer des musiques à partir d'un affichage graphique des répertoires et des fichiers de l'utilisateur. Lorsque l'utilisateur importe des musiques, il peut réaliser plusieurs opérations. Il peut lancer une analyse des musiques qui donnera la durée de la musique, le rythme de la musique (ou bpm) et estimera la tonalité de la musique. Il peut de même, à partir de musiques déjà analysées, lancer un tri des musiques qui va les ordonner afin de pouvoir exporter une playlist qui est la plus harmonieuse possible, telle que l'écart entre le rythme et la tonalité de deux musiques successives soit le plus faible possible. Enfin, l'utilisateur peut ordonner la playlist qui vient de lui être suggéré et l'exporter en format W3U (playlist qui peut être lu dans vlc) ou mp3.

3.1.3 Etude Technique

Nous utilisons le langage de programmation Python3.5 ainsi que les bibliothèques suivantes :
- Matplotlib afin de gérer les FFT (Fast Fourier Transform) sur les musiques importées
- librosa, étant utilisée pour la détection du rythme de la musique
- gtk3+ afin de gérer l'interface graphique de l'application
- numpy pour l'utilisation de matrices
- pydub pour l'exportation de fichiers mp3

3.1.4 Données manipulées

Les données manipulées sont : l'emplacement des musiques analysées, le titre des musiques, le rythme des musiques, la tonalité de la musique, une TreeView qui permet l'affichage dans un tableau des données reçues par l'analyse, un fichier csv servant de base de données ou sont mis les musiques ayant déjà étés analysées.

3.1.5 Liens existants entre les données

Il existe 4 modules dans cette application :

-L'interface graphique, qui est aussi la fonction principale de l'application

- Le module d'analyse, appelé par l'interface graphique afin que le module renvoie les données des musiques ayant étés analysées.
- Le module de tri, appelé par l'interface graphique afin de trier les morceaux importés dans l'application
- Le module d'exportation, appelé par l'interface graphique afin d'exporter les musiques importées dans l'application en format mp3 ou dans un fichier m3u.

3.1.6 Caractéristiques des données

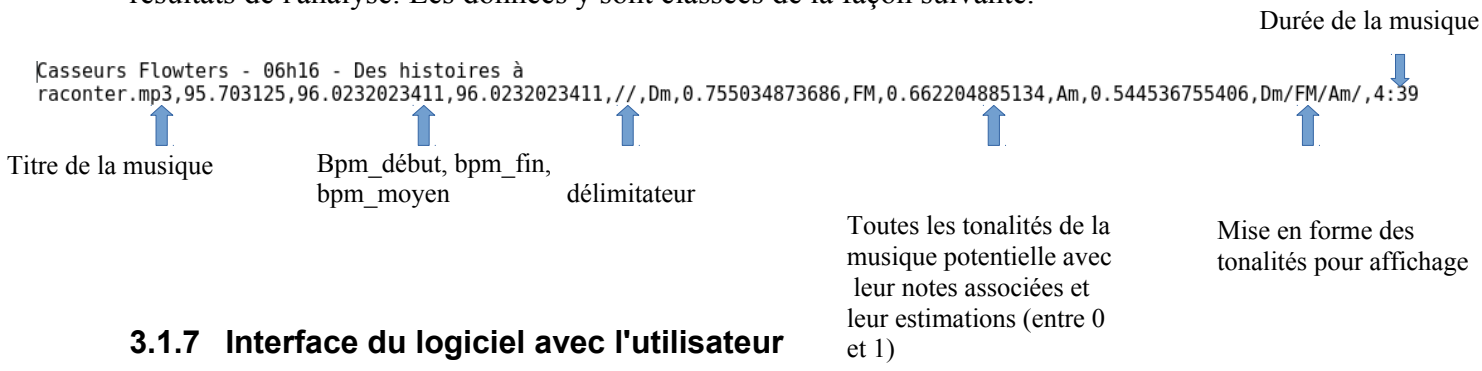
- Le rythme de la musique : float comprit entre 0 et 200
  - cette donnée est calculée dans le module d'analyse
- La tonalité de la musique : string pouvant avoir les valeurs suivantes : A, A#, B, C, C#, D, D#, E, F, F#, G, G#, AM, A#M, BM, CM, C#M, DM, D#M, EM, FM, F#M, GM, G#M, Am, A#m, Bm, Cm, C#m, Dm, D#m, Em, Fm, F#m, Gm.
- (M : Majeur, m: Mineur)
  - cette donnée est calculée dans le module d'analyse
  - cette donnée est transformée en entier allant de 0 à 24 dans le module de tri
- La durée de la musique : étant de la forme suivante mm:ss ou mm sont les minutes et ss les secondes.
- La TreeView : est une matrice de taille n x 4. Les données y sont classées de la façon suivante :

Title	Time	BPM	Harmonic
Deus Ex- Mankind Divided Soundtrack - Trailer Music	3:29	138	G#

La TreeView est actualisée à chaque fois qu'une musique est analysée.

- Le fichier csv servant de base de données à l'analyse permet de retrouver une musique ayant déjà été analysé auparavant afin de réduire de manière significative le temps d'exécution de l'application.

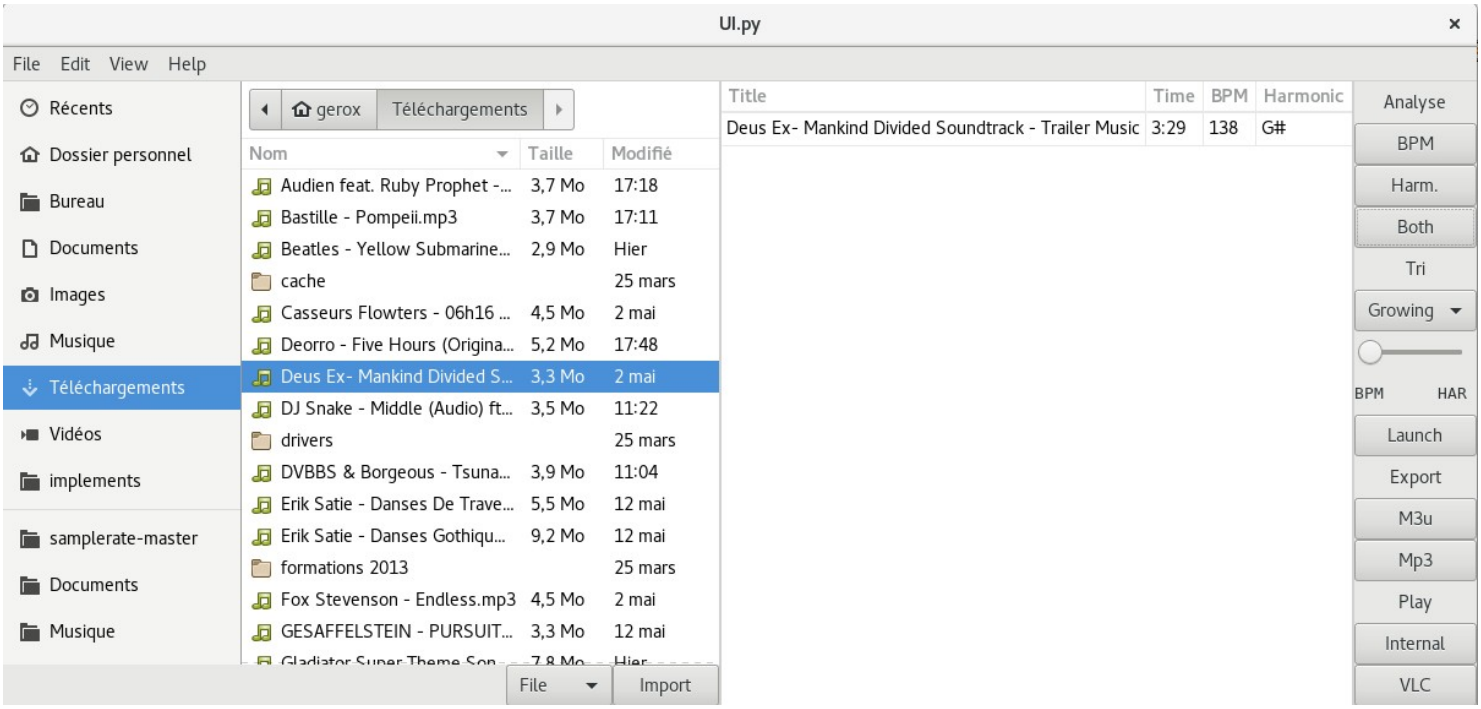
Le fichier csv contient plus de données que nécessaire afin de pouvoir analyser les résultats de l'analyse. Les données y sont classées de la façon suivante.



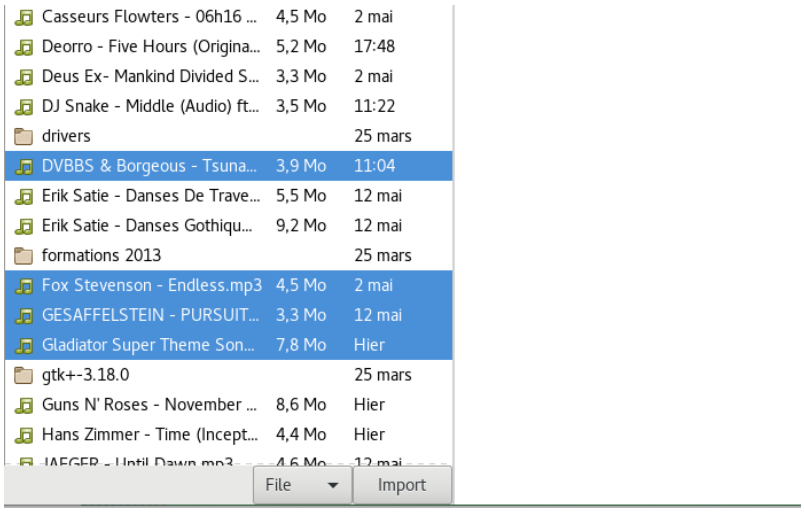
3.1.7 Interface du logiciel avec l'utilisateur

Le logiciel propose à l'utilisateur un menu principal avec diverses options : importer, l'analyse du bpm, l'analyse de la tonalité (ou Harmonique), l'analyse de ces deux données (Both), le fait de choisir et adapter les données à prendre en compte pour trier les musiques (privilégier les harmoniques, le rythme(bpm)) à l'aide d'un curseur, le fait de lancer l'analyse,

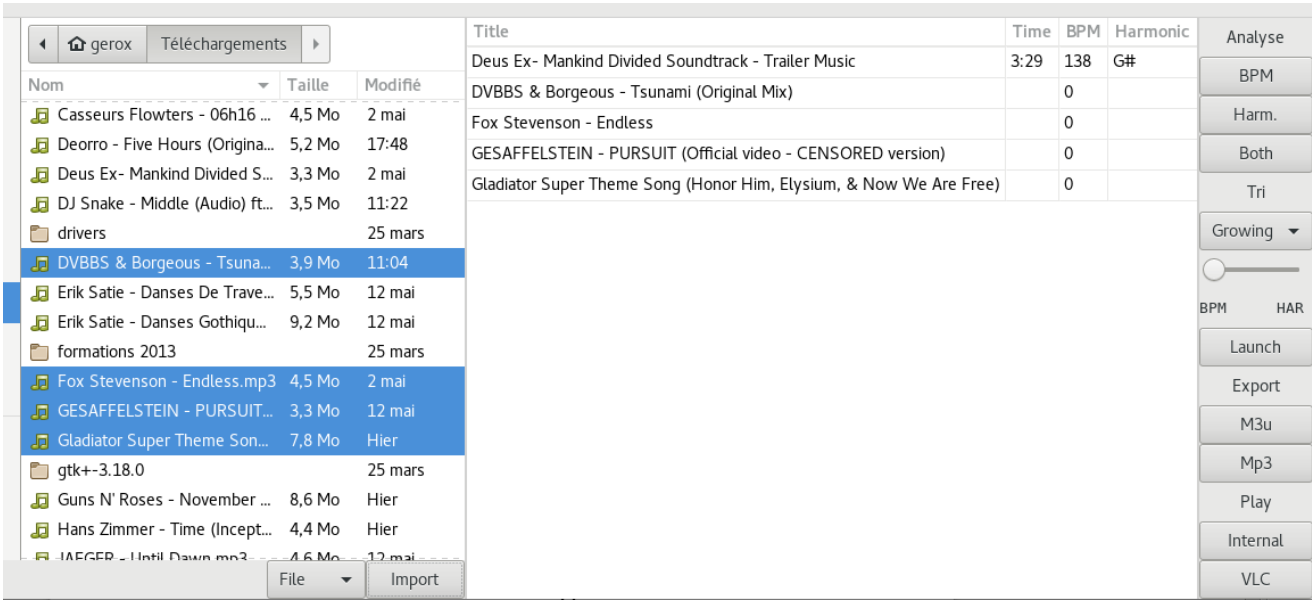
exporter les musiques importées en mp3 ou m3u ainsi que de lancer une lecture des musiques dans un lecteur interne ou externe avec vlc.



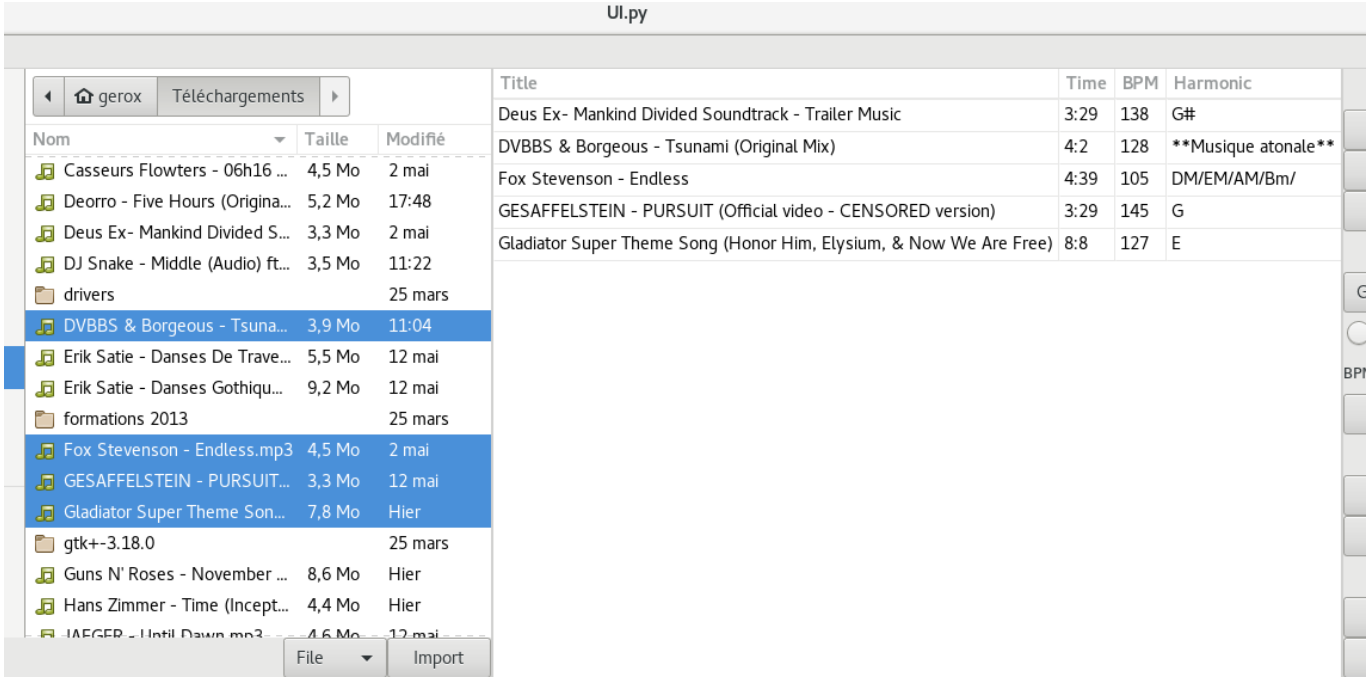
Prenons le cas d'utilisation ou l'utilisateur souhaite analyser des musiques afin de connaître la tonalité de la musique pour pouvoir faire un mashup ou connaître les accords qu'il doit jouer à la guitare. Dans un premier temps l'utilisateur lance le programme et sélectionne les musiques à l'aide de l'interface graphique



L'utilisateur peut sélectionner plusieurs musiques à la fois. A noter que seuls les fichiers avec une extension audio ('.mp3') sont visibles par l'utilisateur. L'utilisateur importe les musiques en appuyant sur le bouton importer.



Les musiques sont alors importées dans la TreeView.
L'utilisateur peut maintenant lancer l'analyse des musiques en cliquant sur 'Both' afin d'avoir toutes les données.
Une fois les musiques analysées, le TreeView se met à jour automatiquement afin d'afficher les résultats.



L'utilisateur peut ensuite utiliser ses données afin de faire un mashup ou connaître les accords à jouer pour reproduire le morceau. Il peut ensuite lancer la fonction de tri pour ordonner ses musiques afin de créer une playlist 'intelligente'.

3.2 Conception préliminaire

3.2.1 Structures de données utilisées dans l'application

Dans l'application, nous utilisons des structures de données pour le fichier csv servant de base de données des musiques déjà analysées (Voir 3.1.6).

Titre, bpm_debut, bpm_fin, bpm_moyen, //, harmo1, coefficient correlation harmo1,…,durée

La TreeView est structurée la façon suivante :

Title	Time	BPM	Harmonic
Deus Ex- Mankind Divided Soundtrack - Trailer Music	3:29	138	G#

Les CSV utilisés dans l'application sont des objets ayant les caractéristiques suivantes :

- le chemin jusqu'au fichier csv instancié (le nom du fichier est donné par l'utilisateur)
- le chemin vers la base de données contenant les informations des musiques déjà analysées
- le chemin du répertoire root afin de naviguer plus facilement entre les données

Les analyses sont des objets ayant les caractéristiques suivantes :

- le chemin vers le fichier audio
- Le nom de l'analyse

Les musiques sont des objets définis par leur:

- Titre
- Durée
- BPM
- Tonalité

3.2.2 Stockage de l'information

Les données sont stockées dans un fichier csv ('.csv') contenant les résultats des analyses précédentes.

3.2.3 Structuration de l'application en modules

- Module UI (interface graphique) : ce menu initialise l'affichage graphique sous gtk3+ de l'application et communique avec les autres modules de l'application.

- Module Analyse : Ce module permet de de charger les musiques importé par l'interface graphique à l'aide la bibliothèque librosa. Il permet d'analyser le bpm et la tonalité de la musique. Il gère de même l'enregistrement des informations des musiques analysées, la manipulation d'objets csv ainsi que la recherche dans le fichier csv afin que l'utilisateur n'analyse pas une musique plusieurs fois.

- Module Exportation : Permet l'exportation des musiques importées en format mp3 ou m3u

- Module Tri : Permet de trier les musiques importées dans l'interface graphique afin de trouver le plus court chemin entre les musiques importées vis à vis de deux paramètres : la tonalité et le rythme (bpm).

3.2.4 Fonctions nécessaires au fonctionnement de l'application

Module Analyse

Classe csv :

- `def is_file(self, path_to_file):`

  fonction qui permet de savoir si un fichier csv est présent

- `def add_list(self, csv_file, liste):`

  fonction qui permet de rajouter une list à un fichier csv existant ou non

- `def find_title_in_database(self, titre):`

  permet de tester si le titre de la musique est dans la base de donné et renvoie les informations necessaire afin d'extraire ses informations

- `def add_column(self, path_to_csv_file, list_a_rajouter, col=None):`

  permet d'ajouter une colonne dans un fichier csv à un numéro de colonne donné

- `def get_column(self, num_col):`

  permet d'obtenir la colonne num_col du fichier csv qui a été instancié

- `def get_row(self, num_row):`

  permet d'obtenir la ligne num_row du fichier csv qui a été instancié

- `def get_column_database(self, num_col):`

  permet d'obtenir la colone num_col du csv de la base de données

- `def get_row_database(self, num_row) :`

  permet d'obtenir la ligne num_ligne du csv de la base de données

- `def delete_row(self, num_row):`

permet de supprimer la ligne num_row du fichier csv initialement instancié

- `def delete_column(self, num_col):`

  permet de supprimer la colonne num_col du fichier csv initialement instancié

- `def delete_row_database(self, num_row):`

  permet de supprimer la ligne num_row de la base de données

- `def delete_column_database(self, num_col):`

  permet de supprimer la column num_col du fichier csv

- `def nombre_ligne_csv(self):`

  renvoie le nombre de ligne du fichier csv initialement instancié

- `def clear(self):`

  supprime le fichier csv initialement instancié

- `def safe_state(self):`

  permet de supprimer les sauts de lignes dans le fichier csv initialement instancié afin de ne pas créer d'erreur lors de la manipulation di fichier csv

- `def safe_state_database(self):`

  permet de supprimer les sauts de lignes dans la fichier csv de la base de données afin de ne pas créer d'erreur lors de la manipulation de fichiers csv

Classe analyse

- `def extraire_path(self):`
  la fonction extrait le titre d'une musique à partir de son chemin d'acces '/home/bob/Musique/music.mp3' renvoie 'music.mp3'

- `def clean_analyses(self):`
  permet de supprimer le fichier d'analyse créer par l'utilisateur

- `def extrairedatamusic(self):`
  permet d'extraire les données d'un fichier audio en faisant appel à la bibliothèque librosa

- `def analyse_bpm(self, y, sr):`

la fonction prend en entrée les données d'un fichier audio sous forme d'une liste et le taux d'échantillonage et renvoie le bpm moyen, bpm de fin, bpm de début d'un fichier audio

- `def analysefft(self, y=None, Fs=None, k=None, afficher=False):`
  fonction qui prend en entrée les données d'un fichier audio, la fréquence d'échantillonage, le nombre de sample que l'on souhaite réaliser.
  La fonction renvoie une liste contenant pour les fréquences maximum des fft effectuées réalisées sur chaque échantillons

- `def rechercheaccords(self, freq):`
  fonction qui prend en entrée la sortie de la fonction précèdente. Il va estimer les notes présentes dans la musique et va ensuite estimer la tonalité de la musique à l'aide de l'algotithme de **Krumhansl-Schmuckler**

- `def is_music_harmonic(self, tonalite):`
  fonction qui permet d'estimer si la musique est tonale ou atonale

- `def major_plus_minor(self, tonalite):`
  fonction qui permet de renvoyer la tonalité G si l'analyse de tonalité trouve GM et Gm

Module Sortie :

- **extraction**(resultat) :
  Cette fonction lit la liste résultat, contenant les objects musique triés, et créer un document .m3u permettant découter la playlist finale.

- **plfinale**(resultat) :
  Cette fonction permet du créer un unique fichier .mp3 contenant toutes les chansons après le tri final de l'utilisateur, ainsi que des transitions entre chaque morceaux de type crossfade (fade-in, fade-out). Le résultat du tri finale est dans la liste résultat contenant les objects musique ordonnés.

Module tri :

- **Init_solution** (self, tableaudobjets) : à partir du tableau d'objets répertoriant les musiques, détermine le nombre de solutions qui sera implémenté dans l'algorithme génétique puis crée une matrice de zéros où seront écrites les solutions.
- **Creation_tablBPM** (self ; tableaudobjets) : Récupère les BPMs présents dans le tableau d'objets répertoriant les musiques, et les place dans la liste tabl_BPM
- **Creation_tablHARMO**(self ;tableaudobjets) : Récupère les pitchs présents dans le tableau d'objets répertoriant les musiques, et les place dans la liste tabl_HARMO
- **Remplissage_matricesolution** (self) : A partir des tableaux tabl_BPM, tabl_HARMO, remplit les matrices solutions (Matrice_solutionBPM et Matrice_solutionHARMO) de façon aléatoire.
- **Ecart_BPM** (self, matrice_solutionsBPM) : A partir de la matrice solution BPM créée, effectue la somme de la différence entre chaque élément consécutif de chaque ligne afin de déterminer pour chaque solution l'écart BPM total .
- **Ecart_HARMO** (self, matrice_solutionsHARMO): A partir de la matrice solution HARMO créée, effectue la somme de la différence entre chaque élément consécutif de chaque ligne afin de déterminer pour chaque solution l'écart Harmonique total.
- **Mutation** (self, matrice_solutionsBPM, matrice_solutionsHARMO) : Création de deux nouvelles matrices solutions (matrice_solutionsMutationBPM et matrice_solutionsMutationHARMO) identiques aux matrices solutions précédentes à deux lignes près qui sont interverties aléatoirement.
- **Ponderation**(self, tabl_BPMsoustrait, tabl_HARMOsoustrait) : A partir des lignes contenant la somme des écarts en BPM et en harmonique, on crée une nouvelle table, tabl_BPMHARMOsoustrait qui combine les deux types d'écart et les pondère en fonction de l'importance de chaque écart.
- **CreationNouvelleSolution** (self, matrice_solutionsBPM, matrice_solutionsHARMO, matrice_solutionsMutationBPM, matrice_solutionsMutationHARMO) : Création d'une nouvelle matrice solution de la taille de la matrice solution précédente qui est composée de la moitié des meilleures solutions de chaque matrice (la mutée et celle initiale).
- **AlgoGenetique** (self, tableaudobjets) : Itère un certain nombre de fois les fonctions précédentes et renvoie la meilleure solution possible trouvable en un temps convenable.

Interface graphique :

- **UI2.glade** : fichier XML contenant le layout de la fenetre principale et faisant la connexion bouton – Signal
- **getTitle(path) :** extrait le titre d'un chemin.
- **exportplaylist()** : créé une copie de la playlist sous forme de matrice.
- **exportPath()** : créé une liste ordonnée des emplacements de chansons.
- **actualize(mat)** : importe une matrice pour actualiser l'affichage.
- **class Handler** : gere les signaux.
- **onBPM()** : lance l'analyse BPM.
- **onHarm() :** lance l'analyse harmonique.
- **onBoth()** : lance les deux analyses.
- **onDeleteWindow()** : gestion de la fermeture de la fenêtre.
- **onLaunch()** : lance le tri.
- **onM3u()** : lance l'exportation en M3u.
- **onMp3()** : lance l'exportation en MP3.
- **onVLC()** : lit la dernière exportation mp3/m3u dans VLC.
- **Gtk.main() :** boucle d'affichage.

3.3 Conception détaillée

Se conférer à la documentation réalisée avec le logiciel Sphinx en html. Dans l'archive contenant le projet, dirigez-vous dans le sous répertoire : /docs/_build.index.html

Les tests unitaires ont étés réalisés dans le dossier /tests/test_module

3.4 Codage

Se conférer à la documentation réalisée avec le logiciel Sphinx en html. Dans l'archive contenant le projet, dirigez-vous dans le sous répertoire : /docs/_build.index.html

3.5 Tests unitaires

Chaque module a été codé et testé séparément du reste du programme. Ainsi, nous avons pu tester les différentes fonctions d'analyse afin d'effectuer des analyses (Cf partie 4). Les différents tests se situe dans le répertoire /tests. Pour les tests effectués sur un panel de musique, la syntaxe d'analyse a été écrite ainsi que les résultats mais le chemin vers les fichiers audio ne sont pas valides car les musiques ne sont pas enregistrées sous le git.

3.6 Tests d'intégration

Après validation des différentes fonctions, procédez à l'intégration des différents modules et effectuez les tests d'intégration prévus.

3.7 Tests de validation

Nous avons testé le programme afin de vérifier que le module d'analyse et l'interface graphique communiquait bien entre eux et que les résultats étaient correctement renvoyés et affichés.

4 Interprétation des résultats

4.1 Analyse de la tonalité

4.1.1 Préambule

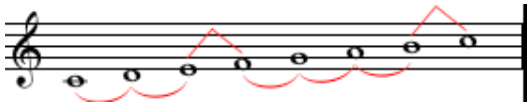
L'analyse du rythme est effectuée avec la bibliothèque librosa sous python et permet de détecter les pics d'énergie de la musique. Ces différents pics sont disposés sur un chronogramme. Il faut ensuite faire une moyenne des écarts de ce chronogramme afin d'avoir accès aux différentes informations sur rythme : bpm_début (on prend en compte les 15 premières secondes, bpm_fin (on prend en compte les 15 dernières decondes) et le bpm moyen (on prend en compte la moyenne de tous les écarts).

Cependant, la partie ayant été la plus chronophage est la partie sur l'estimation de la tonalité. En effet, afin d'appréhender la tonalité, il est nécessaire de comprendre plusieurs notions de musique. Je vais dans un premier temps décrire l'algorithme permettant d'arriver à des résultats sur l'analyse harmonique et commenter les différents tests réalisés pour valider et mettre en avant les limites de ce procédé.

4.1.2 Quelques notions de musiques

Les différentes notions de musiques qui suivent sont expliquées de façon sommaire :

Afin de composer une musique, les compositeurs ont (en occident) la chance de pouvoir se reposer sur un bon nombre d'outils musicaux leurs permettant de composer une musique qui « sonne juste » à l'oreille. Nous pouvons citer en premier lieu comme outil, l'octave, qui est l'intervalle séparant deux sons dont la fréquence fondamentale du plus aigu est le double de celle du plus grave.

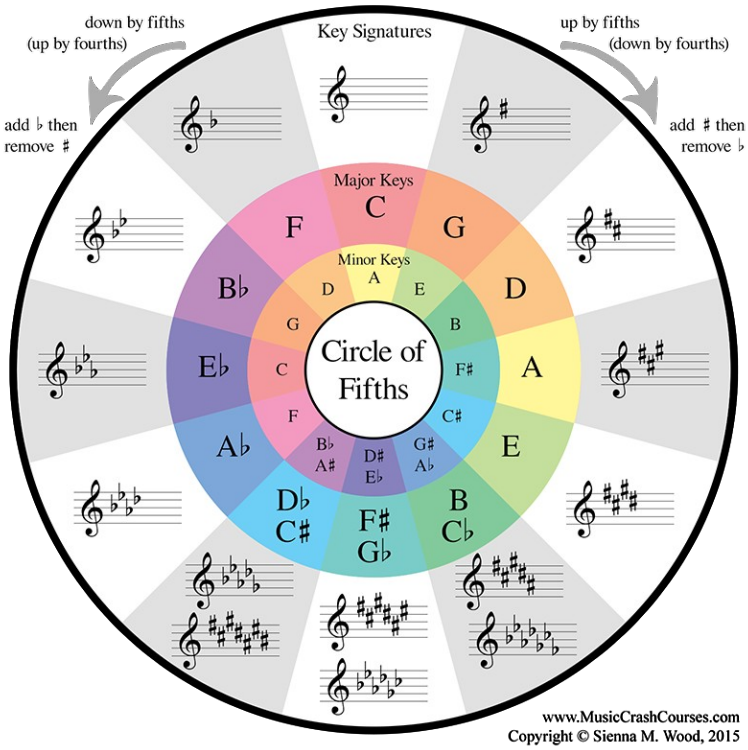


On définit ensuite les gammes à l'aide de ses octaves, qui est une « suite de notes »

Cependant, toutes les notes de l'octave ne vont pas être utilisées lorsque qu'une chanson est composée car la musique ne serait pas harmonieuse (comme si l'on appuyait sur plusieurs touches d'un piano en même temps).

En occident, des théories ont étés élaborées afin de sélectionner les notes que l'on peut jouer dans une gamme afin que les accords formés par ses notes soient harmonieux et agréables à entendre (modes). Enfin, ces modes peuvent être majeurs ou mineurs.

Une tonalité se définit comme une gamme de sept notes, désignée par sa tonique (premier note de la gamme définie) et son mode (majeur ou mineur) : par exemple, la « tonalité de sol majeur ».
Le concept de tonalité se retrouve beaucoup dans la musique contemporaine occidentale (dans la majorité des musiques) car elle fournit un support au compositeur pour écrire un morceau.
Ce qui nous a poussés à étudier la tonalité est qu'il existe des règles permettant de passer d'une tonalité à une autre sans que cela « sonne faux ». Ceci est représenté dans la Circle of Fifth.



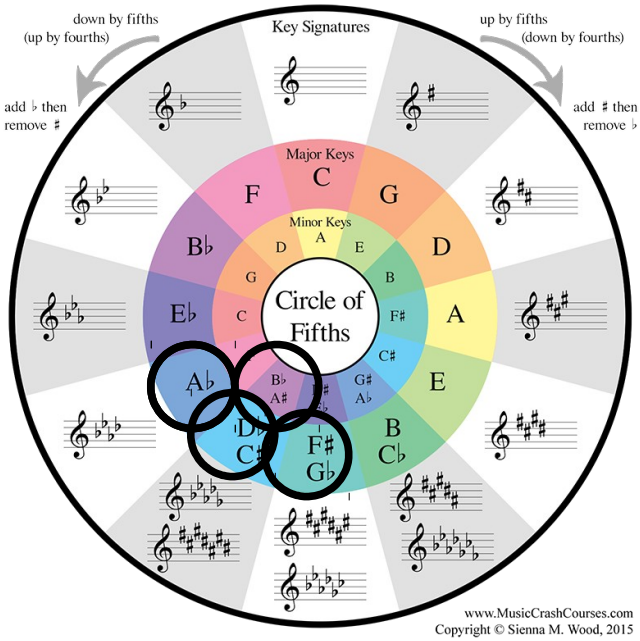
Pour une tonalité donnée, il est possible de passer à la tonalité située à côté sans que cela ne « sonne faux ». C'est ce que nous voulions implémenter dans notre application.

4.1.3 Detection de la tonalité d'une piste audio

Dans un premier temps, la musique à analyser est chargée dans le module analyse par une fonction de la bibliothèque librosa. Ensuite, l'algorithme découpe la musique en plusieurs échantillons (sample) à partir du milieu de la musique (pour ne pas tenir compte de certaines introductions de musique qui peuvent fausser l'analyse). Sur chacun de ces samples, on réalise des FFT. On écarte les basses fréquences qui ont, dans les morceaux contemporains une très grande amplitude, amenant des facteurs d'erreurs. On supprime donc les fréquences en dessous de 500Hz. Puis on détermine par intervalle de 200Hz les fréquences les plus élevées qui sont stockées dans une liste. Pour finir, on discrimine les résultats obtenus pour ne garder que les fréquences ayant une amplitude plus élevée que la moyenne des amplitudes trouvées.

Dans un second temps, on estime à partir de ses fréquences les accords correspondants. Pour ce faire, on supposera que les notes des musiques correspondent aux fréquences des notes tempérées. Pour chaque fréquence, on détermine la fréquence fondamentale et on en déduit la note correspondante.

Dans un troisième temps, on regarde si l'accord correspond au sample analysé. On se réfère pour cela à la « Cirlce of Fifth »



Un accord est agréable à entendre s'il est constitué des notes situé à proximité de lui. (Ex : l'accord C# / Ab / F# / Bb est un accord harmonieux. Il faut s'avoir que les musiques tonales contiennent en quasi-totalité ce genre d'accords.

Dans l'algorithme, on estime qu'un accord est joué si au moins 2 notes côte à côte sur la « Circle of Fifthé » sont trouvées.

Enfin pour la dernière partie, on applique l'algorithme de Krumhansl-Schmuckler. On peut en effet connaître la répartition des accords pour une tonalité majeure ou mineure selon cet algorithme.

major profile

do	do#	re	re#	mi	fa	fa#	so	so#	la	la#	ti
6.35	2.23	3.48	2.33	4.38	4.09	2.52	5.19	2.39	3.66	2.29	2.88

minor profile

la	la#	ti	do	do#	re	re#	mi	fa	fa#	so	so#
6.33	2.68	3.52	5.38	2.60	3.53	2.54	4.75	3.98	2.69	3.34	3.17

Il faut compter le nombre de fois qu'un accord apparaît dans le morceau analysé en se référant à l'algorithme nous permettant d'estimer les accords joués. Il faut répartir le nombre d'apparition d'un accord d'un certain mode en fonction du profil majeur ou mineur.

C major

(do , C)	(6.35, 432)
(do#, C#)	(2.23, 231)
(re , D)	(3.48, 0)
(re# , D#)	(2.33, 405)
(mi , E)	(4.38, 12)
(fa , F)	(4.09, 316)
(fa# , F#)	(2.52, 4)
(so , G)	(5.19, 126)
(so#, G#)	(2.39, 612)
(la , A)	(3.66, 0)
(la# , A#)	(2.29, 191)
(ti , B)	(2.88, 1)

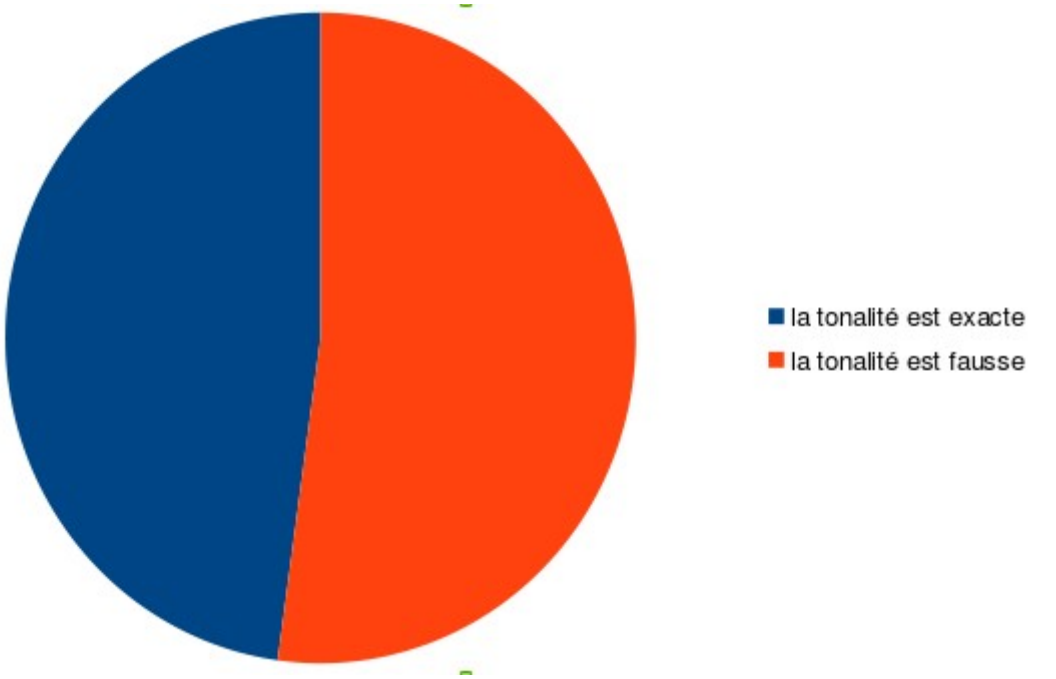
Exemple pour la tonalité CM (la gamme commence par C)

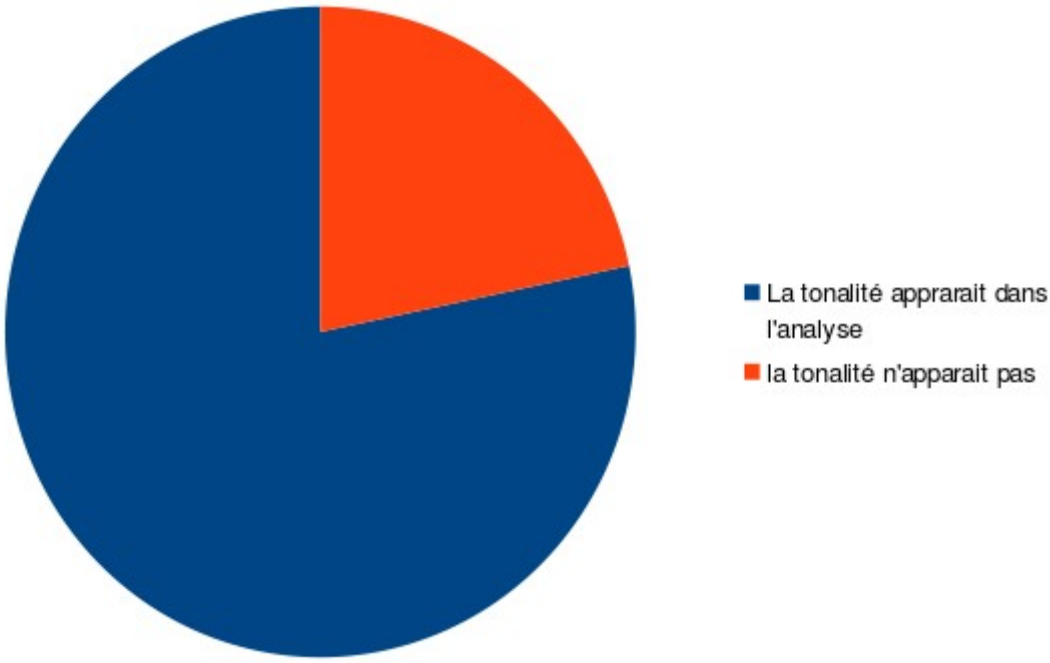
On calcule le coefficient de corrélation obtenu avec ses deux variables. Plus le coefficient de corrélation est proche de 1, plus il est probable que la tonalité soit trouvée. Dans l'algorithme implémenté, on estime que si aucun coefficient de corrélation n'est plus grand que 0,65, alors la musique est atonale. Sinon toutes les tonalités de plus de 0,5 sont renvoyées, ceci permettant d'analyser les résultats obtenus en pratique.

4.1.4 Résultats

On réalise la cherche de la tonalité de 23 musiques dont on connaît par avance le résultat. Les résultats sont présentés de deux façons :

- La tonalité qui obtient le plus haut coefficient de correlation est la tonalité exacte
- La tonalité est présente dans les coefficients de correlation de plus de 0,5 renvoyés par l'algorithme





Au final on trouve un taux de detection de tonalité de manière exact qui esr de 48 %.

On constate que l'on peut trouver d'autres tonalité par rapport à la tonalité cherchée (comme le montre le graphique plus haut) et pourtant trouver un coefficient de corrélation supérieur à 0,5 pour la tonalité exact. Dans point de vu théorique, cela est expliqué par le fait que chaque tonalité possède des tonalités relative, parallèle ou dominant qui ressemblent fortement à la tonalité cherchés. Cela peut expliquer pourquoi l'algorithme ne converge pas nécessairement vers la tonalité exacte.

Mashup[1]

On peut se demander si en prenant un mashup donné, l'analyse des musiques de cette musique va renvoyer des tonalités similaires. Car en pratiques, ces tonalités sont très proches. C'est en effet une des contrainte du mashup car plusieurs musiques doivent être superposées en veillant à ce que l'ensemble de la musique soit harmonieuse.

Premier mashup analysé : « Five Voices - 3LAU » qui regroupe les chansons suivantes (on effectue de même une analyse de la tonalité des musiques) :

- `Deorro - Five hour : tonalité obtenue → FM`
- `Clean Bandit - Rather Be ft. Jess Glynne : tonalité obtenue → G#m`
- `Tritonal feat. Phoebe Ryan - Now Or Never : tonalité obtenue → F#m`

Pour ses trois musiques, l'analyse de la tonalité à convergée vers la tonalité exacte pour 2 des trois musiques. 3 tonalités sur 3 apparaissent dans l'analyse avec des coefficient de corrélation de plus de 0,5.
On constate que les musiques possèdent des tonalités qui sont très proches (un ton d'écart). les résultats obtenues semblent donc être cohérents : les musiques peuvent être assemblées ensemble.

Deuxième mashup analysé :

- `Bastille - Pompeii.mp3 : tonalité obtenue → EM`
- `Audien feat. Ruby Prophet - Circles.mp3 : tonalité obtenue → EM`
- `Mako - Our Story [Free].mp3 : tonalité obtenue → Fm`

Pour ses trois musiques, l'analyse de la tonalité à convergée vers la tonalité exacte pour aucune des trois musiques. 3 tonalités sur 3 apparaissent dans l'analyse avec des coefficient de corrélation de plus de 0,5.
On constate que les tonalités trouvées sont similaires pour l'analyse de ses 3 musiques bien que la tonalité exacte ne soit trouvées dans aucun des trois cas.

Si il s'avère que pour des musiques ayant les mêmes tonalités, l'analyse converge pour chaque musique vers la même tonalité, on peut penser que l'algorithme de détection de tonaité possède un biais d'une tonalité expliquant l'ecart des résultats.

Musiques atonales

Certaines musiques ne sont pas composées dans le système de musique occidental et peuvent être décritent comme étant atonale. Dans l'application, on teste le fait qu'une musique

---

[1] Mashup : le mashup est une chanson créée à partir d'une ou deux autres chansons pré-enregistrées, habituellement en superposant la partie vocale d'une chanson sur la partie instrumentale d'une autre

soit tonale ou atonale si sont la tonalité possèdant le plus grand coefficient de correlation est supérieur à 0,65.

On peut réaliser plusieurs tests sur des musiques atonales :

- « Erik Satie - Danses De Travers »

L'analyse de la tonalité renvoie :

Dm avec un coef de corr:
0.563937415572
F#m avec un coef de corr:
0.513922796417
AM avec un coef de corr:
0.518642433964

L'algorithme estime donc que la musique est atonale et complète la case de la tonalité par **Musique Atonale**.

Title	Time	BPM	Harmonic
Daft Punk - Get Lucky (Official Audio) ft. Pharrell Williams	4:7	116	DM/EM/AM/Bm/
Erik Satie - Danses De Travers	5:42	166	**Musique atonale**

Cependant, certaines musiques peuvent être estimés comme étant atonale le passage analysé de la musique ne reflète pas la musique en entier ou si la musique effectue un changement de tonalité.

En effet, certaines musiques effetcue des changements de tonalités respectant les règles énoncées par la « Circle of Fifths ». Cependant, l'algorithme de détection de tonalité ne peut pas detécter se genre de changement. Dans la plupart des cas, l'algorithme renvoie **Musique atonale**.

Rôle du nombre d'échantillonnage

Il apparaît parfois que l'algorithme ne converge pas vers la bonne valeur. Cependant, en augmentant manuellement le nombre de sample analysé, il se peut que l'algorithme converge vers la bonne tonalité. Ceci est valable par exemple pour la musique `DVBBS & Borgeous - Tsunami (Original Mix).`

Le nombre d'échantillonage est par défaut placé à 50 échantillons. Mais le fait de passer à un nombre d'échantillon de 70 à permit à l'algorithme de converger vers la tonaité exacte. Cependant, il n'est pas possible d'appliquer un nombre d'échantillons de 70 pour chaque musique car cela correspond à une analyse sur une durée de 1m45 ce qui n'est pas possible pour des musiques de moins de 3 minutes car cela renverrai une erreur car l'analyse dépasserait.

4.1.5 Conclusion

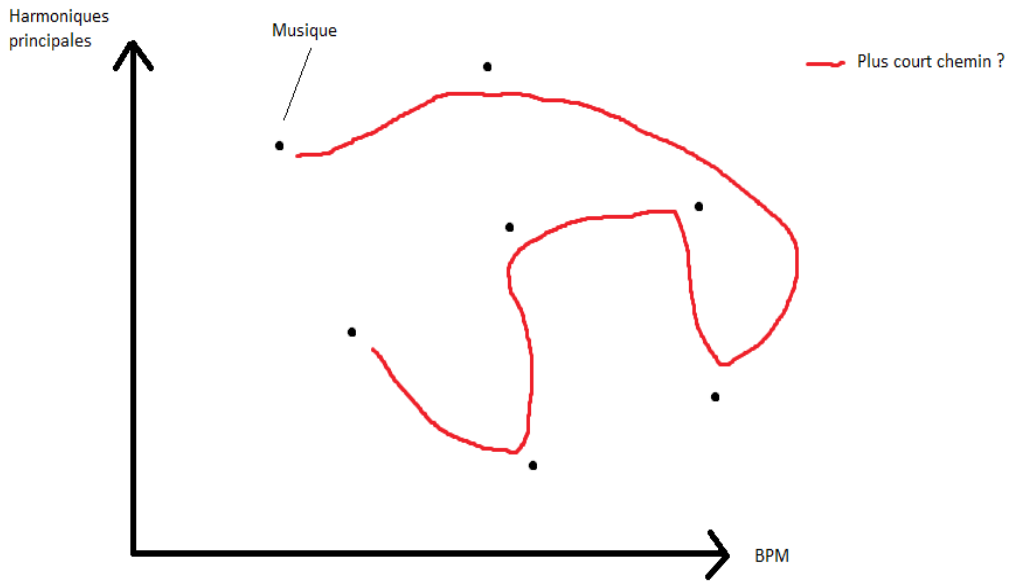
Très peu de logiciel propose des algorithmes de détection de tonalité et sont tous non open source. De même ses logiciels ne sont pas parfait et ceux-ci possède des taux d'erreur allant de 30 % à 70 %. Vis à vis des résultats obtenues (moins de 48 % de réussite), nous pouvons constater que bien que l'algorithme soit loin d'être exacte, l'algorithme de Krumhansl-Schmuckler se révèle assez efficace vis à vis de sa compléxité. De plus, il serait

nécessaire d'effectuer beaucoup plus de testes pour valider et mettre en lumières toutes les limites de l'algorithme (en particulier la détection des notes et accords).

4.2 Algorithme de tri

4.2.1 Définition de notre problème

Suite à la détermination des données à analyser, il est très vite apparu que nous devions composer entre l'harmonique principale du morceau et son BPM. Deux musiques seront alors appréciables à l'oreille au niveau de l'enchaînement si l'écart entre leur BPM et l'écart entre leur harmonique principale sont le plus faibles possibles. Il apparaît alors un problème d'optimisation à deux dimensions qui peut être résumé de la façon suivante :



Comment trouver le meilleur chemin entre les points afin de globalement minimiser le plus possible l'écart Bpm et l'écart harmonique entre chaque musique ? Ici on voit bien qu'un tri « simple » ne suffit pas puisqu'il ne prend qu'un seul paramètre en entrée (ex : tri fusion). La résolution de notre problème de plus court chemin est finalement similaire à celui du voyageur de commerce.

4.2.2 Algorithme choisi

Le voyageur de commerce est un problème très complexe qui se résolve de façon exacte avec une complexité en factorielle. Dans notre cas, la résolution exacte n'est pas envisageable car l'utilisateur est susceptible de proposer une centaine de musiques à l'algorithme, et les temps de calcul seraient infiniment longs.

Nous nous sommes donc tournés vers un algorithme qui résout le problème du voyageur de commerce de façon approchée. Arbitrairement, nous avons choisi de gérer notre tri complexe avec un algorithme génétique.

4.2.3 Principe de l'algorithme génétique

On part d'une population de n solutions initiales. Pour chacune de ces solutions, on calcule l'écart BPM et l'écart harmonique successif entre chaque musique. On obtient alors deux nombres correspondant aux écarts respectifs, que l'on peut ajouter entre eux tout en les pondérant. On obtient alors un « classement » de nos solutions en fonction de leur qualité. Plus l'écart total pondéré est faible, meilleure est la qualité de la solution.

Nous allons alors procéder à la phase de mutation, qui consiste à modifier légèrement chaque solution afin de créer une nouvelle population de solutions. Dans notre cas, la mutation choisie a été simple, pour chaque solution initiale, nous intervertissons deux BPMs et deux harmoniques principales. On évalue la qualité de ces solutions mutées.

A présent, nous choisissons parmi les solutions mutées et les initiales, les n meilleures solutions. Celles-ci formeront notre génération suivante.

On réitère ce système un certain nombre de fois jusqu'à obtenir k-générations. Plus le nombre de générations augmente et plus on va tendre vers la solution exacte.

4.2.4 Intérêt de l'algorithme génétique

L'algorithme génétique a pour intérêt d'être extrêmement polyvalent. Le nombre de générations est réglable facilement. On peut également agir sur d'autres facteurs tels que le nombre de solutions initiales, ou la façon dont les solutions mutent. Cela offre un outil très versatile qui peut être modifié en fonction du nombre de musiques à analyser par exemple, afin d'offrir un bon rapport qualité/temps d'exécution.

4.2.5 De l'algorithme au code

Pour mettre en place l'algorithme sur Python, nous avons tout d'abord créé un objet avec les attributs « titre, emplcement, duree, bpm_moy, pitch » (en réalité, il y a aussi bpm_min et bpm_max mais on ne s'en sert pas).

A partir de cet objet, on a pu construire deux matrices de type (nombre_solution * nombre_musique) contenant pour l'une les BPMs et pour l'autre les harmoniques. A partir de ces matrices, il est facile de calculer les écarts Bpm et les écarts harmoniques entre chaque musique (une petite astuce a été utilisée pour l'écart harmonique car les écarts sont répartis sur une « roue », voir la partie du programme associée à ce calcul).

Ces écarts sont stockés dans deux tableaux, l'un pour les écarts harmoniques et l'un pour les écarts BPM qui sont ensuite réunis en un seul lorsque la pondération est appliquée. Une entrée dans chaque tableau correspond à la somme des écarts BPM (respectivement harmoniques) d'une solution. On identifie alors les meilleures solutions : celles dont la valeur de la somme des écarts pondérés sont les plus faibles.

On procède alors à la construction des matrices mutées et on identifie de même les meilleures solutions. Reste alors à sélectionner les meilleures solutions de chacun des matrices pour former la matrice de la nouvelle génération, et on réitère le processus.

Remarque : le nombre de solutions initiales dépend du nombre de musiques. En dessous de 100 musiques il y a autant de solutions qu'il y a de musiques, et au dessus le nombre de solutions est fixé à 100.

4.2.6 Perspectives d'amélioration

- L'algorithme ne gère pas les musiques atonales.

- La mutation choisie est assez simpliste.

- Le principe de sélection pourrait être amélioré via un système de « joutes » entre solution par exemple.

5 Manuel utilisateur

5.1 Installation (Linux)

5.1.1 Dépendances

Plusieurs modules sont requis pour faire fonctionner MusiCore :

- cairocffi
- numpy
- scipy
- librosa

Vous pouvez les installer avec python-pip, dans le terminal faire :

sudo apt-get install **python3-pip**
sudo pip install **cairocffi numpy scipy librosa audiosegment pydub audioread**

5.1.2 Téléchargement :

La dernière version du projet est disponible sur : https://github.com/gerosix/MusiCore.git

Vous pouvez utiliser git, dans le terminal faire :
sudo apt-get install git
mkdir MusiCore
cd MusiCore
git clone https://github.com/gerosix/MusiCore.git
cd ..

5.1.3 Lancement :

cd MusiCore
./Lauch.py

5.2 Mode d'emploi du logiciel

Comment lancer une analyse ?

Si vous souhaitez lancer l'analyse de plusieurs musiques, sélectionnez à l'aide de l'interface graphique les musiques à sélectionner. Cliquez ensuite sur le bouton importer.
Une fois les musiques importées, cliquer sur le type d'analyse que vous souhaitez effectuer (Harm, BPM, Both). Il faudra attendre quelques instants en fonction du nombre de musiques à analyser. Le tableau se met automatiquement à jour

Je souhaite accéder à plus d'information concernant les musiques analysées, comment faire ?

Pour avoir accès à quelques informations complémentaires, vous pouvez vous rendre dans /database/database.csv.
Vous aurez accès en plus des analyses classiques au bpm de début, bpm de fin ainsi que toutes les tonalités trouvées par l'application ayant un coefficient de corrélation supérieur à 0,5.

```
Adventure Club - Thunderclap (Original Mix).mp3,172.265625,166.708669353,171.238193825,**Musique atonale**,GM/,3:30
Audien feat. Ruby Prophet - Circles.mp3,129.19921875,127.667212204,127.983376672,//,C#m,0.726874815079,EM,0.939131036696,AM,0.649298623007,BM,
Bastille - Pompeii.mp3,129.19921875,128.237437965,128.556436567,//,C#m,0.602936539708,EM,0.784240192995,Em,0.616204876788,AM,0.607478773856,E,
Beatles - Yellow Submarine.mp3,107.666015625,106.074892241,105.987874282,**Musique atonale**,CM/FM/,3:2
```

Je lance l'analyse mais l'application me renvoie de mauvaises valeurs à chaque fois, comment résoudre le problème ?

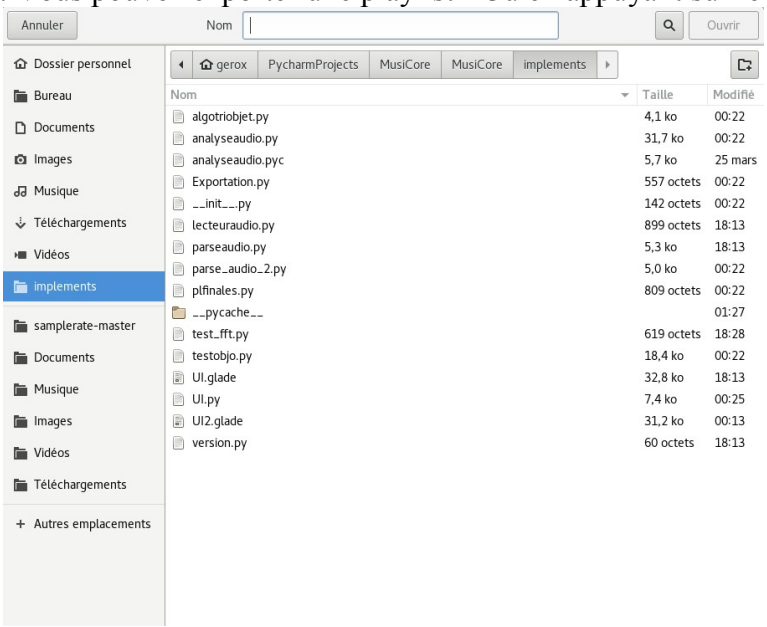
Il se peut que le fichier csv servant de base de données soit endommagé. Si une erreur s'est glissée dans ce fichier, l'interface graphique pourrait vous renvoyer cette valeur erronée (si elle est du bon type d'objet). Pour résoudre le problème, supprimer la ligne correspondant au titre ayant un problème dans le fichier csv de base de données. Relancez ensuite l'application.

Comment trier mes musiques ?

Appuyez sur le bouton de tri afin de lancer l'analyse
Si un enchainement de musique ne vous convient pas, vous pouvez le régler en effectuant un **drag and drop** de la sélection.

Comment exporter une playlist de musique ?

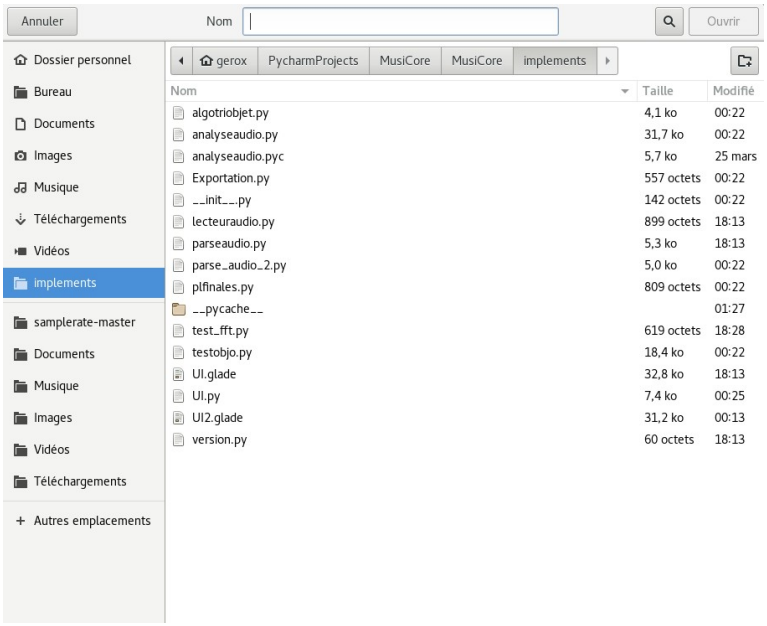
Importer les musiques que vous souhaitez mettre dans votre playlist. Analysez les, triez les si vous le souhaitez. Vous pouvez exporter une playlist m3u en appuyant sur le bouton m3u



Une fenêtre apparaît alors, rentrez le nom de la playlist que vous souhaitez puis cliquez sur « ouvrir ». La playlist m3u est créée.

Comment écoutez votre plaalist ?

Vous pouvez appuyer sur le bouton VLC afin de lire les musiques avec l'application vlc.
Sinon, vous pouvez choisir de lire la musique en interne en appuyant sur le bouton « Internal »



Une fenêtre apparaît alors, rentrez le nom de la playlist que vous souhaitez puis cliquez sur « ouvrir ». La playlist m3u est créée.

Comment écoutez votre playlist ?

Vous pouvez appuyer sur le bouton VLC afin de lire les musiques avec l'application vlc.
Sinon, vous pouvez choisir de lire la musique en interne en appuyant sur le bouton « Internal »

6 Conclusion

Notre projet de création de playlist intelligente a été mené à bien. Nous avons pu créer un module d'analyse musicale fine, qui nous permettait d'obtenir le profil rythmique et harmonique d'une chanson. Nous avons pu trier ces chansons selon ces critères grâce à des outils algorithmiques fournissant une solution approchés basés sur le problème du voyageur de commerce. Ainsi, nous avons créé une application qui permet à l'utilisateur d'avoir une playlist évolutive de ses chansons, qu'il peut paramétrer selon ses goûts.

Plusieurs améliorations sont envisageables, par exemple un player de musique pourrait être directement codé dans l'application, et ainsi de contourner l'exportation. Un tri considérant l'harmonie et/ou le rythme au début et à la fin du morceau permettrait une grande fluidité dans les transitions. A un plus haut niveau, avec des outils d'analyse plus poussés (qui n'existent actuellement pas), il serait possible d'avoir une analyse de meilleure qualité, qui puisse supporter les cas marginaux.

Ce travail nous a permis de nous intéresser à la théorie musicale, mais a aussi posé des problèmes théoriques intéressants, comme le problème du voyageur de commerce. L'application est actuellement assez limitée par la qualité de l'analyse, par manque d'outils et de recherche dans ce milieu, mais avec des morceaux « sympathiques » (c’est-à-dire dont la qualité d’analyse est bonne), on obtient une playlist de qualité.

7 Bibliographie

- Tollari Sabrina, Yves Coueque, Julien Ohler – Algorithmes génétiques pour résoudre le problème du commis voyageur – 2012 - http://sis.univ-tln.fr/~tollari/TER/AlgoGen1/
- Wikipédia – Problème du voyageur de commerce- https://fr.wikipedia.org/wiki/Probl%C3%A8me_du_voyageur_de_commerce
- Scipy – NumPy Manual - 2015 - http://docs.scipy.org/doc/numpy-1.10.1/reference/
- GNU Free Doc – Python GTK+ 3 Tutorial – 2016 - http://python-gtk-3-tutorial.readthedocs.io/en/latest/
- James Robert – Pydub Example Use – 2011 - https://github.com/jiaaro/pydub#bugs--questions
- Wikipédia – M3U - https://en.wikipedia.org/wiki/M3U
- Robert Hart – Key-finding algortihm – 2012 - http://rnhart.net/articles/key-finding/
- Jake Vanderplas – Understanding the FFT Algorithm – 2013 - https://jakevdp.github.io/blog/2013/08/28/understanding-the-fft/
- Wikipédia – Note de musique - https://fr.wikipedia.org/wiki/Note_de_musique
- Craig Stuart Sapp – Visualizing Harmony – 2001 - https://ccrma.stanford.edu/~craig/papers/01/icmc01-harmony-2up.pdf

8 Annexes

8.1 Annexe 1 : Gestion de projet

	SUIVI D'ACTIVITES					
			Charge en heures/participant			
Description de l'activité	Charge en %	Charge en h	Aurélien Bettini	Victor Plichart	Anaïs Villedieu	Constantin Vurli
Total	129,5	259	87	64	44	64
Gestion de projets	24	48	9	9	21	9
Conception préliminaire	9,5	19	5	5	3	6
Conception détaillée	43,5	87	32	20	12	23
Codage	45	90	36	30	8	16
Intégration	7,5	15	5	0	0	10
Soutenance	0	0	0	0	0	0
Gestion de projets						
Réunion de lancement	6	12	3	3	3	3
Réunions de suivi	12	24	6	6	6	6
Rédaction	6	12	0	0	12	0
Conception préliminaire						
Définition des fonctionnalités	5,5	11	3	4	1	3
Définition des différents modules	4	8	2	1	2	3
Conception détaillée						
Définition des structures de données	8,5	17	5	8	1	3
Définition des fonctions	9,5	19	3	10	1	5
Définition des tests unitaires	2	4	4	0	0	0
Auto-formation	23,5	47	20	2	10	15
Codage						
Ecriture de l'interface	5	10	0	0	0	10
Ecriture des fonctions	36,5	73	30	30	8	5
Tests unitaires	3,5	7	6	0	0	1
Intégration						
Intégration des différents modules	6,5	13	3	0	0	10
Tests d'intégration	1	2	2	0	0	0
Soutenance						
Préparation	0	0	0	0	0	0
Soutenance	0	0	0	0	0	0