

# Projet : Py-Reconote

ISN Terminales S

Balthazar CRAVATTE

Mattéo RATET

Gilles Tuffet



# **Sommaire**

## **I - Introduction et présentation du projet**

## **II - Analyse du besoin, recherche d'idées & structuration du projet**

**A** – Recherche d'idées préliminaires

**B** – Analyse du besoin et définition des objectifs

**C** – Positionnement par rapport aux solutions existantes

## **III – Structure du projet & Collaboration**

**A** – Moyens mis en œuvre

**B** – Schéma d'organisation

**C** – Quelques captures montrant le projet final

## **IV – Réalisation personnelle**

## **V – Intégration & Validation**

**A** – Intégration du projet

**B** – Tests, Corrections et Validation

## **VI – Bilan & Perspectives**

## **VII – Diffusion du Projet**

## **VIII - Index**

# I - Introduction et présentation du projet

---

D'après Victor Hugo "La musique, c'est du bruit qui pense". C'est à dire que la musique est une suite d'onde mécanique audible qui évoque aux humains des émotions, des souvenirs, des histoires.

Aujourd'hui, de par son caractère universel, la musique est un élément culturel majeur. De nombreux imprudents se lancent dans l'apprentissage de cet art et découvrent vite son potentiel infini. C'est pourquoi, l'apprentissage de la théorie et la formation de l'oreille musicale est chronophage.

Cependant, l'explosion de l'internet et du numérique a changer notre manière d'apprendre et d'appréhender la musique. De nombreux adolescent ne voulant, ou n'ayant, pas les moyens de se payer des cours, s'achètent une guitare et apprennent de manière autodidacte la musique.

Mais Les outils musicaux numériques ne s'arrêtent pas au répertoire de tablature, telle que *Ultimate-Guitare* et compagnie. En effet, des outils performant, comme *Meludia*, permet de s'entraîner à forger son oreille musicale. Un rêve pour tout musicien que d'acquérir l'oreille absolue, cette faculté de pouvoir identifier n'importe quelle note juste en l'entendant. Un rêve pratiquement inatteignable, car c'est une faculté innée. Si on ne la possède pas dès la naissance il est impossible de l'avoir durant sa vie.

Or, avoir ce don en poche peut être un atout majeur pour toute pratique instrumentale ou de composition. C'est alors que l'outils numérique peut venir aider à la quête du graal.

En effet, puisque l'onde sonore est une onde mécanique progressive, elle peut être résumé par deux paramètres : la pression acoustique en fonction du temps. En analysant un fichier sonore numérique et en extrayant des informations physiques de l'onde, nous pourrons en déduire sa note (c'est-à-dire sa fréquence).

Ainsi, l'outil du numérique pourrait permettre d'ouvrir au plus grande nombre des facultés hors du commun. Nous allons donc nous ne demander, de quelle manière l'outils numérique peut-simplifier l'analyse musicale ?

## II - Analyse du besoin, recherche d'idées & structuration du projet

---

### A – Recherche d'idées préliminaires

La première chose à faire fut de s'accorder tous les trois sur un projet à mener. Deux idées principales émergèrent :

- Un moteur physique pour simuler des phénomènes mécaniques ou thermodynamiques
- Et un projet axé sur le son et la musique.

Au bout de quelques semaines nous nous sommes tournés vers le son, en espérant avoir également l'occasion d'étudier la physique des ondes ainsi que les transformés de Fourier dont nous avons déjà entendu parlé.

Balthazar étant musicien et ayant souvent recours à des logiciels de MAO (musique assisté par ordinateur), il souhaitait créer un programme qui lui serait utile et qu'il pourrait maîtriser facilement.

Nous nous sommes donc orientés vers de la génération d'ondes sonores. A partir de là le projet est resté assez vague pendant quelques temps. L'objectif premier semblait selon Balthazar être la création de l'oscilloscope qui générerait les signaux périodiques auxquels on appliquerait des filtres pour modifier le son.

Cependant, à ce moment-là, le projet et les objectifs étaient mal compris, mal définis et changeaient régulièrement et nous ne savions pas par où commencer. Nous stagnions. De plus Matteo et Gilles n'avaient pas d'ordinateur portable et nous ne pouvions pas véritablement mettre à profit les heures d'ISN car nous ne pouvions rien installer sur les ordinateurs de lycée.

## B – Analyse du besoin et définition des objectifs

Quelques semaines avant la deadline, nous avons finalement réussi à définir nos attentes et nos objectifs.

Mettre au point un programme en langage Python 3.6 capable d'une part d'analyser une séquence de notes à partir d'un fichier audio brut et de retourner leurs noms (en isolant leurs fréquences à l'aide de transformés de Fourier, une solution mathématique existante donc, mais qui nécessite une certaine compréhension du sujet), les intervalles les séparant ainsi que la gamme à laquelle elles appartiennent, et d'autre part de générer et d'encoder un fichier sonore correspondant à une gamme sélectionnée.

## C – Positionnement par rapport aux solutions existantes

Des alternatives existent pour lire et analyser les fréquences d'un fichier son. En effet, Notre projet peut s'apparenter à des applications telle que Shazam.

Néanmoins, *Py-reconote* répond de manière original à cette problématique. Si nous prenons l'exemple de Shazam, une application qui permet de reconnaître les morceaux. Son principe est simple : assigner aux 11 000 morceaux qu'il a dans sa base de données une empreinte unique. Lorsque le programme est lancé, il est capable de reconnaître cette empreinte et affiche donc en sortie le morceau correspondant.

La grande différence entre le *Py-reconote* et Shazam, est que notre programme reconnaît non pas un morceau mais une suite de notes et qu'il ne se base pas sur l'assignation d'empreintes pour analyser la fréquence.

En effet, le *Py-reconote* utilise les transformations de Fourier pour calculer les fréquences caractéristiques des notes du fichier en question.

Le *Py-reconote* n'est donc pas un reconnaisseur de morceaux mais un analyseur musical et scientifique. Cet outil permet donc de reconnaître musicalement une note par le calcul de sa fréquence. Ce calcul se basant sur sa la forme spatiale de l'onde.

Le projet *Py-Reconote* répond donc à une problématique de simplification de la théorie musicale. En effet, Le but principal de cet outil numérique est de pouvoir d'après l'analyse d'une fichier sonore y extraire les propriétés musicales et physiques. Les fichiers traités par *Py-reconote* sont des suites de notes croissantes.

## III – Structure du projet & Collaboration

---

### A – Présentation des outils collaboratifs et organisation du travail de groupe.

Étant habitués à faire des travaux collectifs ensemble. Nous avons des méthodes et habitudes de travailles bien rodées.

Nous avons principalement utilisé la plateforme Curse Voice. C'est un logiciel de messagerie instantané et de chat vocal qui a la particularité de marcher sous la forme de serveur, ou chacun peut se connecter et se déconnecter à sa guise sur la conversation. Cette plateforme permet une flexibilité d'utilisation et une réduction des contraintes techniques liées à la synchronisation de nos emplois du temps respectifs. Nous avons codé la plupart du temps tous ensemble, de manière à améliorer notre efficacité.

La deuxième plateforme collaborative que nous avons utilisé est Google Docs. C'est un traitement de texte collaboratif en ligne. Nous nous en servons très régulièrement dans les autres matières dès que nous devons faire un projet en groupe. L'avantage de cet outil, c'est que nous y sommes très familiarisés avec.

Et enfin, la plateforme en ligne Github. C'est un service web d'hébergement et de gestion de développement de logiciels. L'avantage de Github sur L'open source est qu'il amène chaque contributeur à télécharger les sources du projet et à proposer ensuite ses modifications à l'équipe du projet, GitHub repose sur le principe du fork

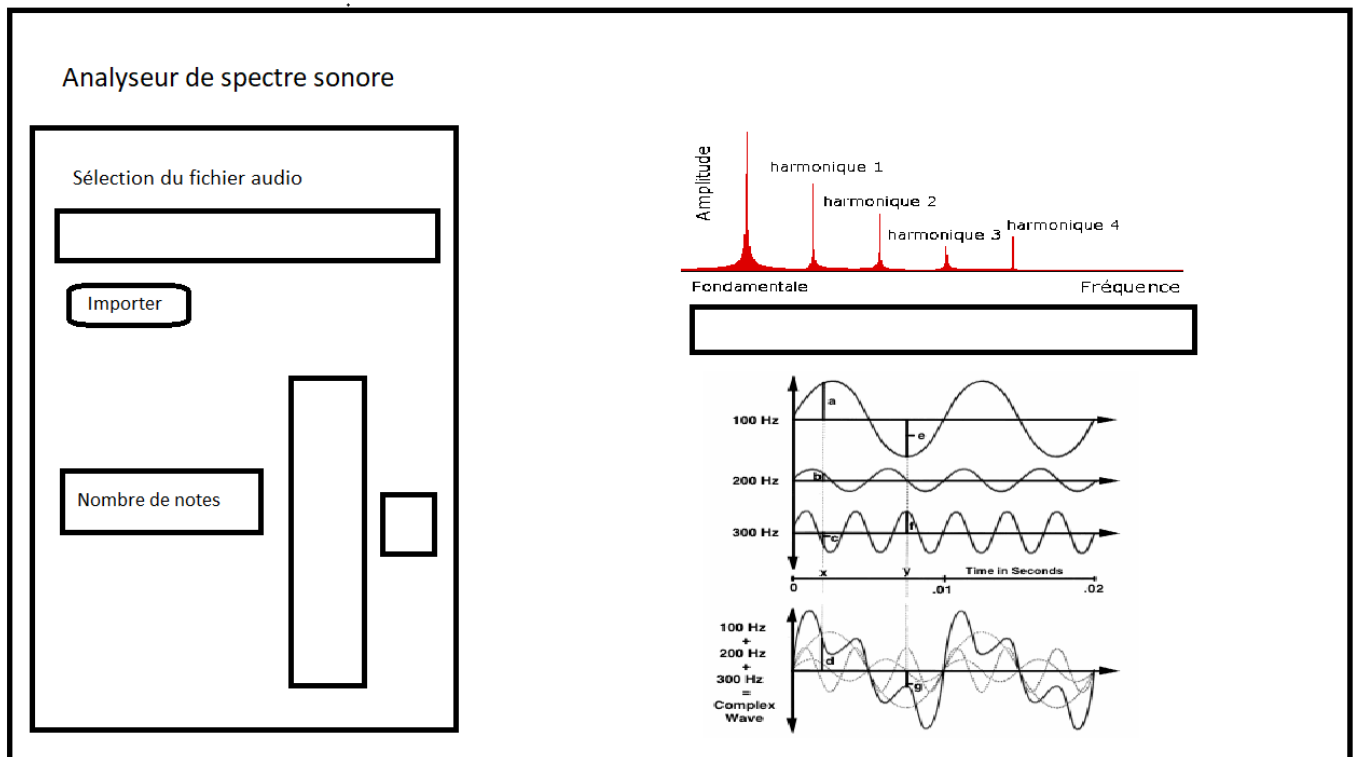
(embranchement). Chaque personne qui change le code devient contributeur à part entière. Nous, nous en sommes servis pour s'échanger les différentes versions du code. Cependant le fait que ce n'est pas une plateforme collaborative instantanée nous a posé problème à plusieurs reprises. En effet, deux personnes travaillant au même moment, peuvent ne pas travailler sur la même version du programme. Ainsi des erreurs persistaient. Nous pensons que pour les prochains projets informatiques nous nous tournerons vers d'autres alternatives.

## Répartition des tâches :

Tâche	planning	Noms
Théorie mathématique, fonctions fourrières → traductions en langage pythons	1 séances	Mattéo, Gilles
Théorie Musicale, relation intervalles fréquences	1 séance	Baltazar, Gilles
Codage d'un graphique statique et dynamique	1 séance	Gilles, Mattéo
Coder Échantillonneur d'un fichier sonore et en extraire la fréquence	1 séance	Mattéo
Codage relations musicale de la suite de note (intervalles, accord, gammes)	1 séance	Balthazar
Coder le générateur de gammes	1 séances	Mattéo, Balthazar
Gestion d'une interface graphique sous Tkinter	1 séances	Gilles

## B – Schéma d'organisation

Voilà un premier croquis d'organisation (fait sur Paint) que l'on avait fait avant de commencer à coder :

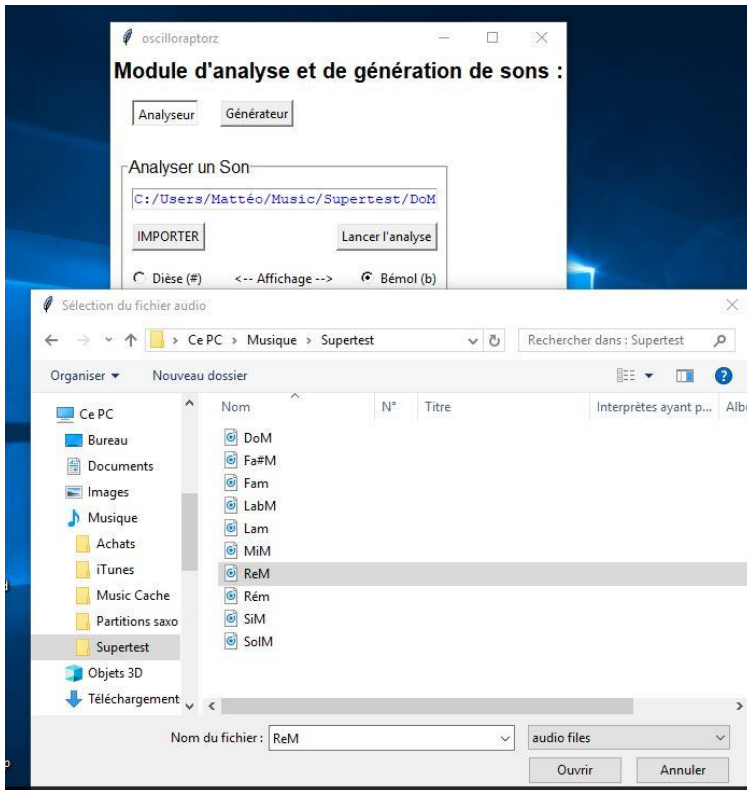


On peut déjà voir la partie de gauche dédiée aux paramètres. On remarque que les sorties textuelles ainsi que l'onglet « Générateur » sont absents.

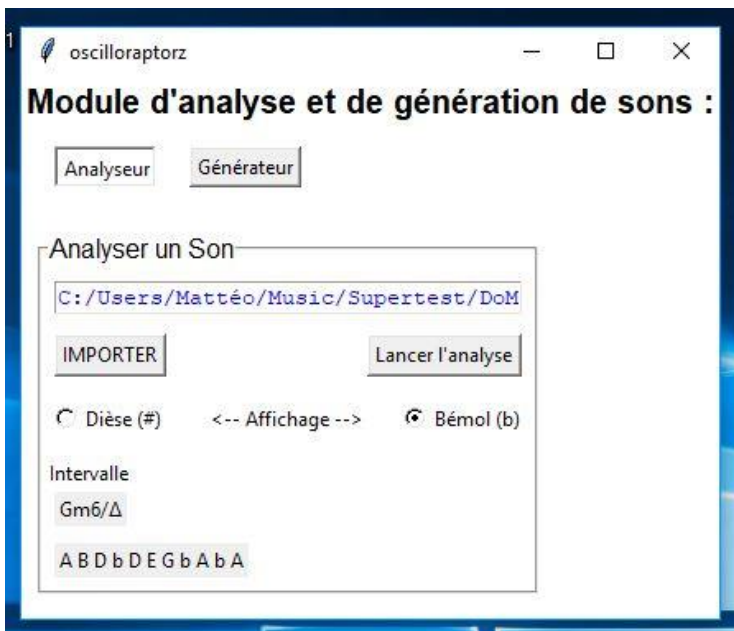
Les seules sorties envisagées au début étaient des graphiques : le spectre sonore (le cadre noir en dessous représentait les notes en format texte avec leurs fréquences associées que l'on voulait afficher sous chaque pic), ainsi que la décomposition en fonctions sinus du signal périodique original. Bien sûr, nous envisagions au départ de rendre ce deuxième graphique dynamique, ce qui s'est avéré quasi impossible sans interférer avec Tkinter.



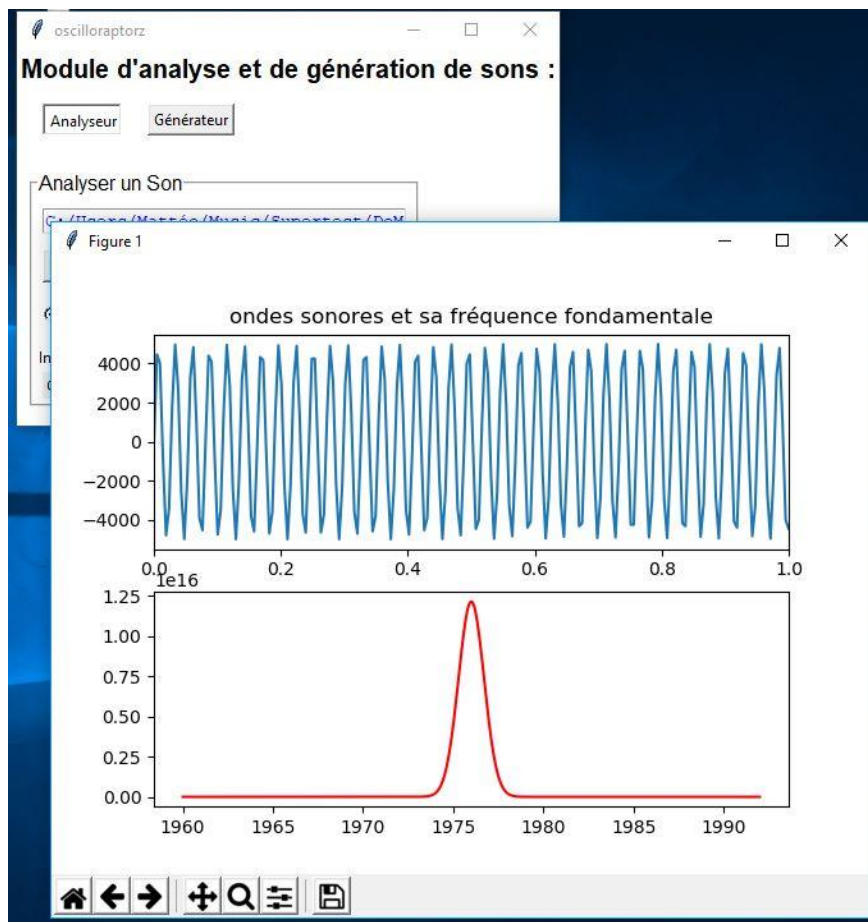
## C – Quelques captures montrant le projet final



Sur cette capture on peut voir l'importation d'un fichier sonore test après avoir cliqué sur le bouton « importer » (qui ouvre l'uploader de fichier). Le fichier sélectionné, on peut voir apparaître son chemin d'accès dans le champ texte de la fenêtre en arrière-plan.



Sur cette deuxième capture on peut voir le programme, en mode « Analyse » après importation et analyse d'un fichier (dont on peut voir le chemin d'accès apparu dans le premier champ). L'affichage est en mode « bémol » (il est en « Dièse » par défaut), et les notes obtenues en sortie sont effectivement en bémol.



Sur cette troisième capture du module d'analyse, on peut voir les deux graphiques. Le premier (en bleu, représente) l'intensité du son en fonction du temps pour une seconde. Le deuxième donne la fréquence fondamentale de la première note analysée).

oscilloraptorz

**Module d'analyse et de génération de sons :**

Analyseur   Générateur

Générer un Son

Amplitude

6500

Durée de la Note

2

Gamme :

CM

Nom du Fichier :

"gamme\_CM.wav"

Lancer la génération

Cette dernière capture montre le programme en mode générateur et toutes les entrées prises en comptes : l'amplitude du son (de 0 à 15000) la durée de la note (de 1 à 5 secondes) sont définies à l'aide de scalebox. La gamme (ici Do Majeur) choisie à l'aide d'une spinbox. Enfin le nom du fichier que l'on entre au clavier.

## IV – Réalisation personnelle

---

Je me suis personnellement occupé de l'étude du fichier sonore importé au format .wav (qui contenait une suite de notes jouée une, deux ou 3 seconde chacune par exemple) à partir duquel, je devais calculer les fréquences de chaque note afin de les identifier, tout en traçant l'allure globale de l'onde sonore ainsi que celle de sa transformée de fourier. J'ai également écrit une fonction permettant d'écrire un fichier au format .wav jouant les notes de n'importe quelles gammes, majeures ou mineurs au choix.

Pour manipuler les fichiers sonores, j'ai opté pour le module wave car ils nous permettaient de récupérer les principales données du fichier (nb de canaux, fréquence d'échantillonnage et taille de l'échantillon) et d'obtenir sans trop de difficultés un tableau contenant les données à exploiter. De plus, grâce à ce module, nous avons pu ensuite générer nos propres fichiers .wav, bien qu'il ne s'agisse que d'une suite de sons très simple (sons composés d'une seul sinusoïde).

Pour écrire les différentes gammes, je n'ai pas réussi à trouver un moyen efficace de générer n'importe quelles gammes majeures sur commande (on aurait peut-être pu se baser sur les écarts entre les tons et les demi-tons qui sont toujours identiques pour les gammes majeures et mineurs). J'ai donc décidé d'entrer chaque gamme manuellement, néanmoins l'utilisation d'un dictionnaire permet de rendre plus claire la composition de chaque gamme pour peu que l'on connaisse l'ordre des dièses et bémols et que l'on soit habitué à l'écriture des notes avec des lettres (du moins, cela me paraît plus compréhensif qu'une simple suite de fréquence).

Après avoir choisi sa gamme, le code génère un fichier .wav dans le même dossier que celui où est enregistré le code python, le fichier obtenu est un fichier composé de 8 notes au temps d'exécution et aux fréquence d'échantillonnage identique. L'amplitude, le temps joué par la note et le choix de la gamme est laissé à l'utilisateur mais l'interaction reste néanmoins extrêmement limitée. Le but principal était avant tout de permettre à quelqu'un de tester le logiciel bien qu'il ne dispose pas de fichier sonore adéquate à tester.

Vient ensuite la partie la plus compliqué de mon travail, l'interprétation des données du fichier dans le but d'exploiter sa transformée de Fourier discrète afin d'en déduire les notes jouées.

Pour faire simple, la transformée de Fourier de l'échantillon d'une note est la reconstitution sur un graphe des principales fréquences contenant le son, elle va en quelque sorte décomposer le signal d'origine.

Pour obtenir la transformée de fourier de mon signal, deux choix s'offraient à nous, l'algorithme de numpy et celui de scipy, nous avons finalement optés pour numpy. Cet algorithme s'appelle "Transformation de Fourier Rapide" (TFR ou FFT en anglais). Il permet d'obtenir, à partir d'un échantillon de valeurs représentant une note, un grand tableau contenant des valeurs complexes. Une fois convertis en valeur réel (en multipliant les valeurs complexe par leurs conjugués) et après changés leurs types, on se retrouve avec un grand tableau à partir duquel nous pouvons savoir quelles sont les fréquences prédominantes de l'extrait sonore. On relève ensuite la fréquence la plus haute et on part du postulat qu'il s'agit de la fondamentale et donc de la note jouée par l'instrument (cela n'est pas forcément vrai mais nous avons choisi cette option par facilité, l'analyse complète de ce tableau n'étant pas vraiment à notre portée).

De plus, après avoir fait plusieurs tests, je me suis rendu compte que les notes identifiées était parfois fausses de quelques dizaines de Hertz. En cherchant une aide sur Internet, j'ai réussi à résoudre ce problème en utilisant une courbe gaussienne. Malgré cela un problème persiste, les notes sont juste à presque 100% des cas mais les octaves relevées sont parfois totalement improbables (jusqu'à 2 octaves de différence entre le signal relevé et le signal d'origine). A ce jour, nous n'avons pas réussi à résoudre ce problème.

Enfin, je me suis occupé de l'aspect graphique du programme. En utilisant matplotlib (un module scientifique spécialisés dans les graphiques), à part quelque difficultés sur l'actualisation du graphique et le choix des points à tracer, je n'ai pas eu tant de mal que ça à récupérer les infos dont j'avais besoin sur le wiki pour tracer les graphiques.

# V – Validation & Intégration

---

## A – Intégration du Projet

Notre réalisation finale est un programme ayant 2 fonctions principales : analyser un son et générer une gamme. A partir d'un fichier audio contenant une suite de note, il parvient à retrouver les notes jouées et identifier les écarts entre les notes voire la gamme jouée.

Les fonctions que j'ai écrites permettent de faire la transition entre le fichier audio brute et l'étude des notes jouées. Le programme affiche également dans une autre fenêtre 2 graphiques représentant le signal d'origine et sa transformée de Fourier.

Enfin, on peut changer de fenêtre principale pour afficher le générateur de gammes, il suffit alors de rentrer différentes données afin d'obtenir un fichier .wav contenant toute les notes d'une gamme choisie.

## B – tests, Correction et Validation

Pour valider le générateur de gammes, je n'ai pas effectué de test particulier, après avoir écrits l'ensemble du code et corriger toutes les petites erreurs de syntaxe, j'ai généré de nombreuses gammes pour être certains que cela fonctionnait comme je le souhaitais.

Par contre, lors du travail concernant l'analyseur, j'ai d'abord travaillé avec des enregistrements très simples que j'avais moi-même écrit (avec le générateur). Puis, pour tester ce code sur des enregistrement plus compliqués, j'ai demandé à Balthazar de me fournir de nombreux extraits audio (différentes gammes jouées aux pianos et à la basse notamment).

## VI – Bilans & Perspectives

---

### Développer le projet :

Je suis content d'avoir réussi à identifier des notes de musique à partir d'un enregistrement sonore en utilisant l'algorithme de la transformée de Fourier. Néanmoins, cette identification ne fonctionne toujours pas de manière parfaite, les notes sont bien identifiées mais les octaves sont parfois totalement différentes avec celle des instruments réelles (jusqu'à 2 octaves de différences), j'aimerais pouvoir régler ce problème à l'avenir.

En ce qui concerne l'aspect plus globale du projet, j'aurai aimé pouvoir intégrer un lecteur de son afin de voir s'afficher en temps réel, le nom des notes en bas à gauche et les graphiques correspondant sur la droite le tout de manière synchronisés. L'écriture d'une partition avec les notes correspondantes aurait aussi été sympa à faire bien que la réalisation soit compliqués.

Enfin, j'aurai aimé intégré les graphiques de matplotlib directement dans la page Tkinter, cela me semble tout à fait possible mais nous n'avons pu le faire par manque de temps.

### Ce que le projet m'a apporté :

Grâce à ce projet, j'ai pu découvrir de manière plus approfondie le langage python et la programmation en groupe. En ce qui concerne python, j'ai dû me familiariser avec différents modules comme wave, struct et surtout matplotlib et numpy.

J'ai eu beaucoup de mal à véritablement comprendre le fonctionnement de fonctions comme la fft (Fast Fourier Transformation) mais au final à force de m'informer sur différents forums et de lire les wikis des modules, je pense avoir réussi à cerner l'essentiel trouve que cela fut enrichissant.

En ce qui concerne le travail par équipe, je pense qu'il a bien fonctionnés dans notre groupe, chacun a codés la partie du code qui l'intéressait avant de nous retrouver tous ensemble pour l'intégrer au code principale. La principale difficulté dans notre groupe fut plus la décision d'un sujet que le travail ensemble

## VII – Diffusion du projet

---

<https://github.com/lasource2018/oscilloraptor>

Nous avons choisi la version 4.0 de Creative Commons comme licence de diffusion pour plusieurs raisons.

Tout d'abord, possible d'utiliser cette licence dans le cadre de projets *Open data*. Il permet de mieux contrôler juridiquement l'accessibilité des différentes parties d'un projet. Par exemple de choisir d'attribuer une même licence Creative Commons pour une publication et les jeux de données liés, ou une licence pour la publication et une autre pour les données.

De plus Créative Common ne nécessite pas d'adaptation de juridique nationale pour être applicable. L'aspect modulable de Creative Common nous largement séduite comparé aux autres licences. On pense que la licence répondra précisément à nos attentes.



# VIII – Index (code)

```
1 from tkinter import *
2 from tkinter import filedialog
3 from tkinter import messagebox
4 import tkinter.font as tkFont
5 import wave, struct, math
6 from struct import *
7 import numpy as np
8 import matplotlib.pyplot as plt
9 from scipy.optimize import curve_fit
10
11 #Définition des fonctions
12
13 #Importation du fichier et vérification du format (.wav only)
14 def selec_fichier():
15     pathfilename = filedialog.askopenfilename(initialdir = "/",title = "Sélection du fichier audio",filetypes = (("audio
16     files","*.*"),("all files","*.*")))
17     longueur = len(pathfilename)
18     ext = []
19     for boucle in range(1,5):
20         ext.append(pathfilename[-boucle])
21     ext.reverse()
22     Extension = "".join(ext)
23     if Extension != ".wav":
24         messagebox.showerror("Erreur", "Mauvais Format : Veuillez sélectionner un fichier .wav")
25     else:
26         Filenamebox.delete(0, END)
27         Filenamebox.insert(END, pathfilename)
28         return pathfilename
29
30 #switch entre les onglets générer et analyser
31 def switchMode():
32     SM = AnaGen.get()
33     if SM == 0:
34         Generateur.grid_forget()
35         Analyseur.grid(row=3, column=1, sticky=NW, pady=15, padx=10)
36     elif SM == 1:
37         Analyseur.grid_forget()
38         Generateur.grid(row=3, column=1, sticky=NW, pady=15, padx=10)
39
40 # Ecris un fichier audio .wav à partir d'une onde sinusoïde et d'une fréquence d'échantillonnage
41 def Ecriture_Son(nomFichier, listeEch):
42
43     nbreOctets = 2
44     nbreCanaux = 1
45     fech = 44100
46
47     nbreEchant = len(listeEch)
48     parametres = (nbreCanaux, nbreOctets, fech, nbreEchant, 'NONE', 'notcompressed')
49
50     Liste = []
51     for i in range(nbreEchant):
52         b = struct.pack('h',listeEch[i])
53         Liste.append(b[0])
54         Liste.append(b[1])
55
56     data = bytes(Liste)
57
58     f = wave.open(nomFichier, 'wb')
59     f.setparams(parametres)
60     f.writeframes(data)
61     f.close()
62     messagebox.showinfo("Succès", "Le fichier son a bien été généré avec succès.")
63
64 # 2 dictionnaire classant les notes de D0 131 à D0 523 soit 2 octaves
65 oc = { "C": 131, "C#": 139, "Db": 139, "D": 147, "D#": 156, "Eb": 156, "E": 165, "F": 175, "F#": 185, "Gb": 185, "G": 196, "G#": 208,
66 "Ab": 208, "A": 220, "A#": 233, "Bb": 233, "B": 247 }
67
68 oc2 = { "C": 262, "C#": 277.18, "Db": 277.18, "D": 293.66, "D#": 311.13, "Eb": 311.13, "E": 329.63, "F": 349.23, "F#": 369.99, "Gb":
69 369.99, "G": 392.00, "G#": 415.30, "Ab": 415.30, "A": 440.00, "A#": 466.16, "Bb": 466.16, "B": 493.88, "C2": 523 }
70
71 # ensemble des gammes majeures et mineures
72 CM = [oc["C"],oc["D"],oc["E"],oc["F"],oc["G"],oc["A"],oc["B"],oc2["C"]]
73 Am = [oc["A"],oc["B"],oc2["C"],oc2["D"],oc2["E"],oc2["F"],oc2["G#"],oc2["A"]]
74
75 #Gammes Majeures avec #
76 GM = [oc["G"],oc["A"],oc["B"],oc2["C"],oc2["D"],oc2["E"],oc2["F#"],oc2["G"]]
77 DM = [oc["D"],oc["E"],oc["F#"],oc["G"],oc["A"],oc["B"],oc2["C#"],oc2["D"]]
78 AM = [oc["A"],oc["B"],oc2["C#"],oc2["D"],oc2["E"],oc2["F#"],oc2["G#"],oc2["A"]]
79 EM = [oc["E"],oc["F#"],oc["G#"],oc["A"],oc["B"],oc2["C#"],oc2["D#"],oc2["E"]]
80 BM = [oc["B"],oc2["C#"],oc2["D#"],oc2["E"],oc2["F#"],oc2["G#"],oc2["A#"],oc2["B"]]
81 FM = [oc["F#"],oc["G#"],oc["A#"],oc["B"],oc2["C#"],oc2["D#"],oc2["F"],oc2["F#"]]
82 CM = [oc["C#"],oc["D#"],oc["F"],oc["F#"],oc["G#"],oc["A#"],oc2["C"],oc2["C#"]]
83
84 #Gammes Mineurs avec #
85 Em = [oc["E"],oc["F#"],oc["G"],oc["A"],oc["B"],oc2["C"],oc2["D#"],oc2["E"]]
```



```

86 Bm= [oc["B"],oc2["C#"],oc2["D"],oc2["E"],oc2["F#"],oc2["G"],oc2["A#"],oc2["B"]]
87 F_m = [oc["F#"],oc["G#"],oc["A"],oc["B"],oc2["C#"],oc2["D"],oc2["F"],oc2["F#"]]
88 C_m = [oc["C#"],oc["D#"],oc["E"],oc["F#"],oc["G#"],oc["A"],oc["C"],oc2["C#"]]
89 G_m = [oc["G#"],oc["A#"],oc["B"],oc2["C#"],oc2["D#"],oc2["E"],oc2["G"],oc2["G#"]]
90 D_m = [oc["D#"],oc["F"],oc["F#"],oc["G#"],oc["A#"],oc["C"],oc2["D"],oc2["D#"]]
91 A_m = [oc["A#"],oc["C"],oc2["C#"],oc2["D#"],oc2["F"],oc2["F#"],oc2["A"],oc2["A#"]]
92
93 #Gammes Majeures avec bémol
94 FM = [oc["F"],oc["G"],oc["A"],oc["Bb"],oc2["C"],oc2["D"],oc2["E"],oc2["F"]]
95 Bbm= [oc["Bb"],oc2["C"],oc2["D"],oc2["Eb"],oc2["F"],oc2["G"],oc2["A"],oc2["Bb"]]
96 EbM = [oc["Eb"],oc["F"],oc["G"],oc["Ab"],oc["Bb"],oc2["C"],oc2["D"],oc2["Eb"]]
97 AbM = [oc["Ab"],oc["Bb"],oc2["C"],oc2["Db"],oc2["Eb"],oc2["F"],oc2["G"],oc2["Ab"]]
98 DbM = [oc["Db"],oc["Eb"],oc["F"],oc["Gb"],oc["Ab"],oc["Bb"],oc2["C"],oc2["Db"]]
99 GbM = [oc["Gb"],oc["Ab"],oc["Bb"],oc2["B"],oc2["Db"],oc2["Eb"],oc2["F"],oc2["Gb"]]
100 Cbm= [oc["B"],oc["Db"],oc["Eb"],oc["E"],oc["Gb"],oc["Ab"],oc["Bb"],oc2["B"]]
101
102 # Gammes Mineurs avec bémol
103 Dm = [oc["D"],oc["E"],oc["F#"],oc["G"],oc["A"],oc["Bb"],oc2["C#"],oc2["D"]]
104 Gm = [oc["G"],oc["A"],oc["Bb"],oc2["C"],oc2["D"],oc2["Eb"],oc2["F#"],oc2["G"]]
105 Cm = [oc["C"],oc["D"],oc["Eb"],oc["F"],oc["G"],oc["Ab"],oc["B"],oc2["C2"]]
106 Fm = [oc["F"],oc["G"],oc["Ab"],oc["Bb"],oc2["C"],oc2["Db"],oc2["E"],oc2["F"]]
107 Bbm = [oc["Bb"],oc2["C"],oc2["Db"],oc2["Eb"],oc2["F"],oc2["Gb"],oc2["A"],oc2["Bb"]]
108 EbM = [oc["Eb"],oc["F"],oc["Gb"],oc["Ab"],oc["Bb"],oc2["B"],oc2["D"],oc2["Eb"]]
109 AbM = [oc["Ab"],oc["Bb"],oc2["B"],oc2["Db"],oc2["Eb"],oc2["F"],oc2["G"],oc2["Ab"]]
110
111
112 # génère une gamme composé d'une suite de sinus, le fichier est ensuite enregistré dans le dossier où est enregistré le code python
113 def ecrire_gamme(nom_fichier, gamme, amplitude, tps_note):
114     E = []
115     n = gamme
116     for i in range(len(n)):
117         O = [int(round(amplitude*math.sin(2*math.pi*n[i]*t*(1/44100)))) for t in range(44100*tps_note)]
118         E.extend(O)
119     Ecriture_Son(nom_fichier, E)
120
121
122 #La fonction reliée au bouton Lancer la génération, qui récupère toutes les valeurs et lance "ecrire_gamme()"
123
124 def Launch_Generation():
125     nom_fichier = SaisieNom.get()
126     gamme = GammeEntry.get()
127     amplitude = Amp.get()
128     tps_note = DNote.get()
129     ecrire_gamme(nom_fichier, gamme, amplitude, tps_note)
130
131
132
133
134 # extrait plusieurs données(nombre de canaux, fréquence d'échantillonnage, nombre d'échantillon) ainsi que tout le contenu d'un fichier
135 d'un fichier audio .wav
136 def lecture_Son(nomFichier) :
137     data = []
138     f = wave.open(nomFichier, 'rb')
139
140     nbCanaux = f.getnchannels()
141     nbreEchant = f.getnframes()
142     tailleEchant = f.getsampwidth()
143     fech = f.getframerate()
144
145     if tailleEchant == 2 :
146         for i in range(nbreEchant) :
147             b = f.readframes(1)
148             try:
149                 val = struct.unpack('h', b)
150             except:
151                 val = struct.unpack('2h',b)
152             data.append(val[0])
153
154     elif tailleEchant == 1 :
155         for i in range(nbreEchant) :
156             b = f.readframes(1)
157             val = struct.unpack('b', b)
158             data.append(val[0])
159     else :
160         print("Format de fichier non reconnu")
161
162     return f, data, nbCanaux, nbreEchant, fech
163
164
165 #définition des intervalles
166
167 def IntervalleBasique(L, Intervalle, PositionListe):
168     TailleListe = len(L)
169     for loop in range(TailleListe):
170         quotient = L[loop]/L[0]
171         if 2.2 < quotient < 1.8 and Intervalle == "Octave":
172             return True
173         if 1.49 < quotient < 1.51 and Intervalle == "Quinte":
174             if PositionListe:
175                 return loop

```

```

176         else :
177             return True
178         break
179     elif 1.32 < quotient < 1.34 and Intervalle == "Quarte":
180         if PositionListe:
181             return loop
182         else :
183             return True
184         break
185     elif 1.18 < quotient < 1.23 and Intervalle == "TierceMin":
186         return True
187         break
188     elif 1.24 < quotient < 1.26 and Intervalle == "TierceMaj":
189         return True
190         break
191     elif 1.16 < quotient < 1.10 and Intervalle == "Seconde":
192         return True
193         break
194
195 def IntervalleComplexe(L, Intervalle, CalcVTheorique):
196     TailleListe = len(L)
197     if CalcVTheorique :
198         if Intervalle == "Sixte":
199             ValeurIntervalle = L[0] * (5/4) * (4/3)
200         elif Intervalle == "SeptiemeMaj":
201             ValeurIntervalle = L[0] * (3/2) * (5/4)
202         elif Intervalle == "SeptiemeMin":
203             ValeurIntervalle = L[0] * (3/2) * (3/2)
204         elif Intervalle == "Neuvieme":
205             ValeurIntervalle = L[0] * (3/2) * (4/3) / 2
206     else:
207         if Intervalle == "Sixte":
208             ValeurIntervalle = L[IntervalleBasique(L, "Quarte", True)] * (5/3)
209         elif Intervalle == "SeptiemeMaj":
210             ValeurIntervalle = L[IntervalleBasique(L, "Quinte", True)] * (5/4)
211         elif Intervalle == "SeptiemeMin":
212             ValeurIntervalle = L[IntervalleBasique(L, "Quinte", True)] * (3/2)
213         if Intervalle == "Neuvieme":
214             ValeurIntervalle = L[IntervalleBasique(L, "Quinte", True)] * (3/2) / 2
215     for loop in range(TailleListe):
216         quotient = L[loop] / ValeurIntervalle
217         if 0.9 < quotient <= 1.01:
218             return True
219         break
220
221 def GammeDefinisiteur(Liste):
222     try:
223         if IntervalleBasique(Liste, "TierceMaj", False):
224             if IntervalleBasique(Liste, "Quarte", False) :
225                 if IntervalleComplexe(Liste, "SeptiemeMaj", False):
226                     return "Ionien"
227                 else:
228                     return "Mixolydien"
229             else :
230                 return "Lydien"
231         else :
232             if IntervalleBasique(Liste, "Quinte", False):
233                 if IntervalleComplexe(Liste, "Sixte", True):
234                     if IntervalleComplexe(Liste, "Neuvieme", True):
235                         return "Aéolien"
236                     else:
237                         return "Phrygien"
238                 else:
239                     return "Dorien"
240             else:
241                 return "Locrien"
242     except:
243         return "Erreur"
244
245 #Détections d'accords
246 def AccordDefinisiteurTaille3(Liste):
247     try :
248         if IntervalleBasique(Liste, "TierceMaj", False):
249             if IntervalleBasique(Liste, "Quinte", False):
250                 return ""
251             elif IntervalleComplexe(Liste, "Sixte", True):
252                 return "6"
253         elif IntervalleBasique(Liste, "TierceMin", False):
254             if IntervalleBasique(Liste, "Quinte", False):
255                 return "m"
256             elif IntervalleComplexe(Liste, "Sixte", True):
257                 return "m6"
258         elif IntervalleBasique(Liste, "Quarte", False):
259             if IntervalleBasique(Liste, "Quinte", False):
260                 return "sus4"
261         elif IntervalleComplexe(Liste, "Neuvieme", False):

```

```

262         if IntervalleBasique(Liste,"Quinte", False):
263             return "sus2"
264     elif IntervalleBasique(Liste, "Octave", False):
265         if IntervalleBasique(Liste,"Quinte", False):
266             return "PowerChord HEEEEELL YEAH"
267 except:
268     return "Unknown Chord"
269 def AccordDeffinisieurTaille4(Liste):
270     try:
271         if IntervalleComplexe(Liste,"SeptiemeMin", True):
272             if AccordDeffinisieurTaille3(Liste, False) == "6" or "m6" or "sus4" or "sus2":
273                 return AccordDeffinisieurTaille3(Liste), "/"
274             else:
275                 return AccordDeffinisieurTaille3(Liste), "7"
276         if IntervalleComplexe(Liste,"SeptiemeMaj", True):
277             if AccordDeffinisieurTaille3(Liste) == "6" or "m6" or "sus4" or "sus2":
278                 return AccordDeffinisieurTaille3(Liste), "/"
279             else:
280                 return AccordDeffinisieurTaille3(Liste), "Δ"
281     except:
282         return AccordDeffinisieurTaille3(Liste)
283
284 #détection d'intervalles
285 def IntervalleDeffinisieur(Liste):
286     try:
287         Intervalle = ["Seconde", "TierceMaj", "TierceMin", "Quarte", "Quinte", "Sixte", "SeptiemeMin", "septiemeMaj"]
288         n = 0
289         while n <= len(Intervalle):
290             if IntervalleBasique(Liste, Intervalle[n], False) or IntervalleComplexe(Liste, Intervalle[n], True):
291                 return Intervalle[n]
292             n = n + 1
293     except:
294         return "Erreur"
295
296 #fonction général qui réunie toute les fonctions qui viennent d'être défini
297 def sortie(ListeFreq, ListeNote):
298     Taille = len(ListeFreq)
299     if Taille == 1:
300         return ListeNote[0]
301     elif Taille == 2:
302         return IntervalleDeffinisieur(ListeFreq)
303     elif Taille == 3 or 4:
304         return ListeNote[0], AccordDeffinisieurTaille4(ListeFreq)
305     elif Taille > 5:
306         return ListeNote[0], GammeDeffinisieur(ListeFreq)
307
308 # A partir d'une note à une octave quelconque, transpose cette note à l'octave 3 (entre 260 et 495) et conserve dans une variable
309 l'octave d'origine
310 def octave(freq):
311     n = 0
312     while freq < 255 or freq > 500 :
313         if freq < 255:
314             freq = freq * 2
315             n = n - 1
316         elif freq > 500:
317             freq = freq/2
318             n = n + 1
319     return freq, n
320
321 # Identifie une note à partir de sa fréquence, Diese == 1 ou 0 selon affichage souhaités # ou b
322 def notefreqjuste (frequence, Diese):
323     a, n = octave(frequence)
324     if 259 < a < 263:
325         return "C", n
326
327     elif 274 < a < 279:
328         if Diese == 1:
329             return "C#", n
330         else:
331             return "Cb", n
332
333     elif 290 < a < 295:
334         return "D", n
335
336     elif 309 < a < 313:
337         if Diese == 1:
338             return "D#", n
339         else:
340             return "Eb", n
341
342     elif 326 < a < 332:
343         return "E", n
344
345     elif 346 < a < 351:
346         return "F", n
347
348     elif 366 < a < 371:

```

```

350     if Diese == 1:
351         return "F#",n
352     else:
353         return "Gb",n
354
355 elif 389 < a < 395:
356     return "G",n
357
358 elif 412 < a < 418:
359     if Diese == 1:
360         return "G#",n
361     else:
362         return "Ab",n
363
364 elif 437 < a < 443:
365     return "A",n
366
367 elif 463 < a < 469:
368     if Diese == 1:
369         return "A#",n
370     else:
371         return "Bb",n
372
373 elif 490 < a < 496:
374     return "B",n
375
376
377
378 #fonction d'affinage qui sera utilisé plus tard
379 def func(x,a,b,c):
380     return c * np.exp( -np.power( (x-a)/b, 2) )
381
382
383 #fonction qui va lire le fichier, séparés les données pour chaque note, faire la transformée de fourier pour chaque note et enfin
384 #afficher un graphique représentant les 2 courbes
385
386 def analyser_son(nomFichier):
387     plt.ion #on utilise pyplot en interactif
388
389     # Ouverture du fichier wav a decrypter
390     f, x, nbCanaux, nbFrames, fech = lecture_Son(nomFichier)
391
392     #Découper le fichier pour analyser chaque note après l'autre
393     frequences, freq_gauss, FreqNoteJuste, ListeNote, = [], [], [], []
394     larg_frame = 44100
395     for posi in range(0,nbFrames,larg_frame):
396
397         # Sequence contenant une note
398         f.setpos(posi)
399         donnee = f.readframes(larg_frame)
400         data = struct.unpack('sh'*(larg_frame*nbCanaux), donnee)
401
402         # Transformée de Fourier
403         fourier = np.fft.fft(data)
404         valeur_réelle = np.real(fourier * fourier.conjugate())
405         freqs = np.fft.fftfreq(len(fourier)) * nbFrames
406
407         # Estimation de la fréquence
408         idx = np.argmax(valeur_réelle)
409         f0, maxi = np.abs(freqs[idx]), valeur_réelle[idx]
410         frequences.append( f0 )
411
412         #Ajustement par une gaussienne (permet l'ajustement des valeurs de f0)
413         ind = np.where( np.abs(freqs - f0) < 20 )
414         popt, pcov = curve_fit( func, freqs[ind], valeur_réelle[ind]/maxi, p0=[f0,1,1] )
415         a, b, c = popt
416         freq_gauss.append(a)
417
418         note, n = notefreqjuste (a, Diese)
419         ListeNote += note
420
421         FreqNoteJuste.append((oc2[note]*(n-1)))
422
423     #Affichage
424     signal = []
425     for i in range(0,len(data),210):
426         signal.append(data[i])
427     fs = 44100/210
428
429     Time=np.linspace(0, len(signal)/fs, num=len(signal))
430
431     plt.subplot(211)
432     plt.plot(Time,signal)
433     plt.title("ondes sonores et sa fréquence fondamentale")
434     plt.xlim(0,1)
435

```



```

435     plt.subplot(212)
436     fnew = np.linspace( fregs[ind][0], fregs[ind][1], 512)
437     plt.plot( fnew, maxi * func(fnew,a,b,c), 'r')
438
439
440     plt.draw()
441     plt.pause(3)
442     plt.clf()
443
444
445     f.close()
446     intervale = sortie(FreqNoteJuste, ListeNote)
447     return ListeNote, intervale
448
449 #La fonction reliée au bouton Lancer l'analyse, qui sélectionne et lance la fonction "analyser_son"
450 def Launch_Analyser():
451     nomFichier = selec_fichier()
452     a, b = analyser_son(nomFichier)
453     AllNotes = " ".join(a)
454     return AllNotes
455
456
457 #Début de l'interface graphique
458 fenetre = Tk()
459 fenetre.title("oscilloraptorz")
460 fenetre.configure(background="white")
461 Label(fenetre, text="Module d'analyse et de génération de sons :", bg="white", fg="black", font="none 15 bold").grid(row=1, column=1,
462 sticky=NW)
463 FontSS = tkFont.Font(size=18)
464
465
466
467 #fenêtre de droite
468 CadreMode = LabelFrame(fenetre, bg="white", bd=0, height=50)
469 CadreMode.grid(row=2, column=1, sticky=NW, pady=7, padx=10)
470 AnaGen = IntVar()
471 AnaGen.set(0)
472 Ana = Radiobutton(CadreMode, text="Analyseur", value=0, variable=AnaGen, indicatoron=0, command=switchMode)
473 Ana.grid(row=2, column=1, sticky=NW, pady=5, padx=10)
474 Gen = Radiobutton(CadreMode, text="Générateur", value=1, variable=AnaGen, indicatoron=0, command = switchMode)
475 Gen.grid(row=2, column=2, sticky=NW, pady=5, padx=10)
476
477
478
479 #Fenêtre analyseur
480 Analyseur = LabelFrame(fenetre, bg="white", bd=2, height=400, text="Analyser un Son", font="FontSS")
481 Analyseur.grid(row=3, column=1, sticky=NW, pady=15, padx=10)
482
483 #Importation du fichier
484 Filenamebox = Text(Analyseur, width = 35, height=1, wrap=WORD, bg="white", fg="blue")
485 Filenamebox.grid(row=3, column=1, sticky=NW, pady=10, padx=8)
486 Import = Button(Analyseur, text="IMPORTER", command=selec_fichier)
487 Import.grid(row=4, column=1, sticky=NW, padx=8, pady=2)
488 AnaLaunch = Button(Analyseur, text="Lancer l'analyse", command=Launch_Analyser)
489 AnaLaunch.grid(row=4, column=1, padx=8, pady=2, sticky=E)
490
491 #Affichage en Dièse ou en bémol
492 Diese = IntVar()
493 Diese.set(1)
494 D1 = Radiobutton(Analyseur, text='Dièse (#)', value=1, variable=Diese, bg = "white")
495 D1.grid(row=6, column=1, sticky=W, padx=4, pady=11)
496 Label(Analyseur, text="<- Affichage ->", bg="white").grid(row=6, column=1, padx=4, pady=11)
497 B1 = Radiobutton(Analyseur, text='Bémol (b)', value=0, variable=Diese, bg = "white")
498 B1.grid(row=6, column=1, sticky=E, padx=4, pady=11)
499
500 #Affichage des intervalles et des notes
501 InterNote = LabelFrame(Analyseur, bg="white", text="Intervalle", bd=0)
502 InterNote.grid(row=7, column=1, sticky=W, padx=4, pady=1)
503 PrintInter = "Gm6/A" #relier avec la fonction "sortie"
504 Label(InterNote, text=PrintInter).grid(row=1, column=1, sticky=W, padx=4, pady=3)
505 NomNotes = LabelFrame(Analyseur, bg="white", text="Notes détectées", bd=0)
506 Label(InterNote, text=Launch_Analyser()).grid(row=2, column=1, sticky=W, padx=4, pady=7)
507
508
509
510 #Fenêtre Générateur
511 Generateur = LabelFrame(fenetre, bg="white", bd=2, height=400, text="Générer un Son", font="FontSS")
512 Generateur.columnconfigure(1, minsize=35)
513
514 #Amplitude
515 Amp = Scale(Generateur, orient="horizontal", from_=0, to=15000, resolution=100, sliderlength=20, label="Amplitude", length=277,
516 bg="white")
517 Amp.grid(row=1, column=1, sticky=NW, padx=8, pady=6)
518
519 #Durée de la note
520 DNote = Scale(Generateur, orient="horizontal", from_=1, to=5, resolution=1, sliderlength=20, label="Durée de la Note", length=277,
521 bg="white")
522 DNote.grid(row=2, column=1, sticky=NW, padx=8, pady=6)
523
524 #Gamme
525 Gammetext = Label(Generateur, text="Gamme :", bg="white")
526 Gammetext.grid(row=3, column=1, sticky=NW, padx=8, pady=12)
527 GammeEntry = Spinbox(Generateur, bg="white", values=["CM", "DM", "AM", "EM", "FM", "BbM", "EbM", "AbM", "Am", "Em", "Bm", "Dm", "Gm"],
528 wrap=True)
529 GammeEntry.grid(row=3, column=1, padx=20, pady=12, sticky=E)
530
531 #saisie du nom du fichier
532 NameFileEntry = Label(Generateur, text="Nom du Fichier :", bg="white")
533 NameFileEntry.grid(row=4, column=1, sticky=NW, padx=8, pady=12)
534 SaisieNom = Entry(Generateur, bg="white", width = 22)
535 SaisieNom.grid(row=4, column=1, padx=20, pady=12, sticky=E)
536
537 #Bouton générer
538 GenLaunch = Button(Generateur, text="Lancer la génération", command=Launch_Generation)
539 GenLaunch.grid(row=5, column=1, padx=20, pady=12, sticky=W)
540
541
542 fenetre.mainloop()
543

```