Projet : Py-Reconote

ISN Terminales S

Balthazar CRAVATTE
Mattéo RATET
Gilles Tuffet

Sommaire

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 avions 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 la 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* utiliser 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

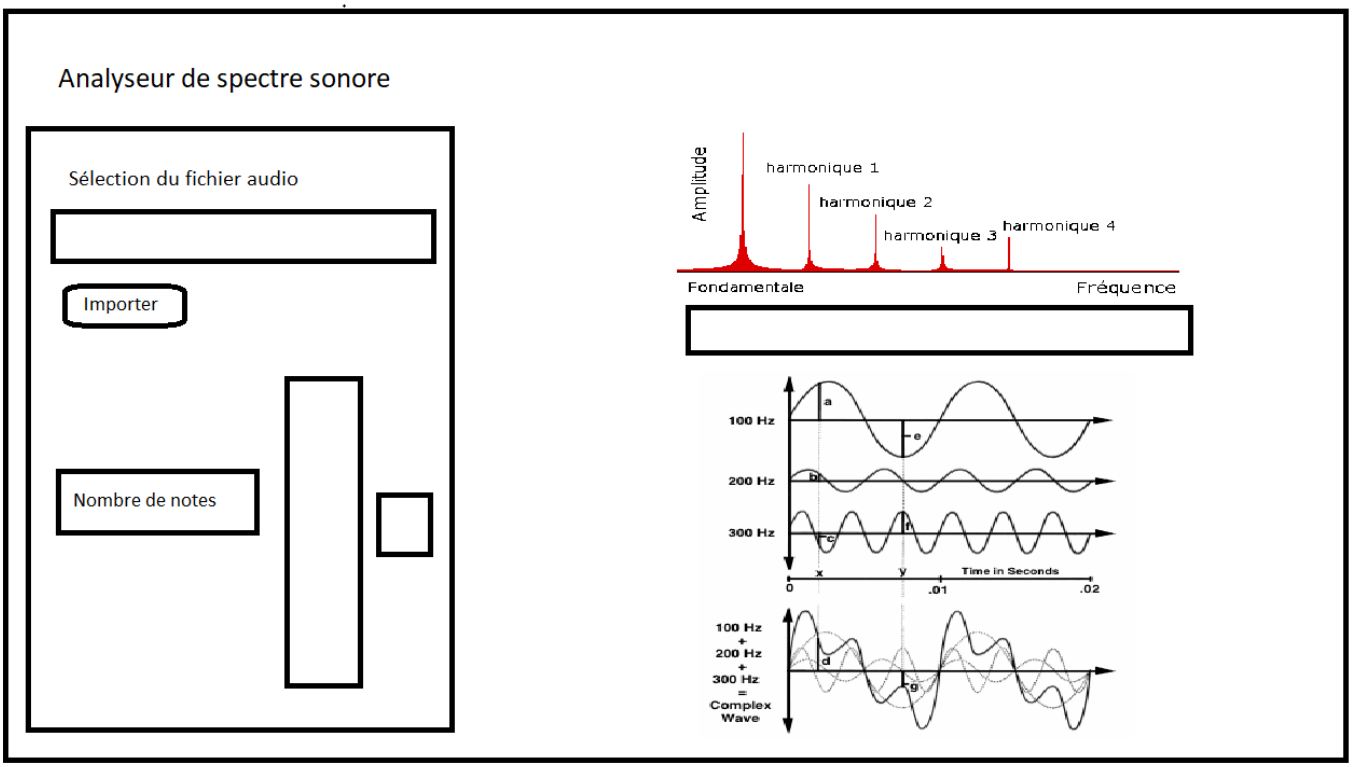
ses modifications à l'équipe du projet, GitHub repose sur le principe du fork (embranchement). Chaque personne qui change le code devient contributeur à part entière. Nous, nous en somme servit pour s'échanger les différentes versions du code. Cependant le fait que ce n'est pas une plateforme collaborative instantané 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 projet informatique nous nous tournerons vers d'autres alternatives.

Répartition des tâches :

Tâche	planning	Noms
Théorie mathématique, fonctions fourriers → 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

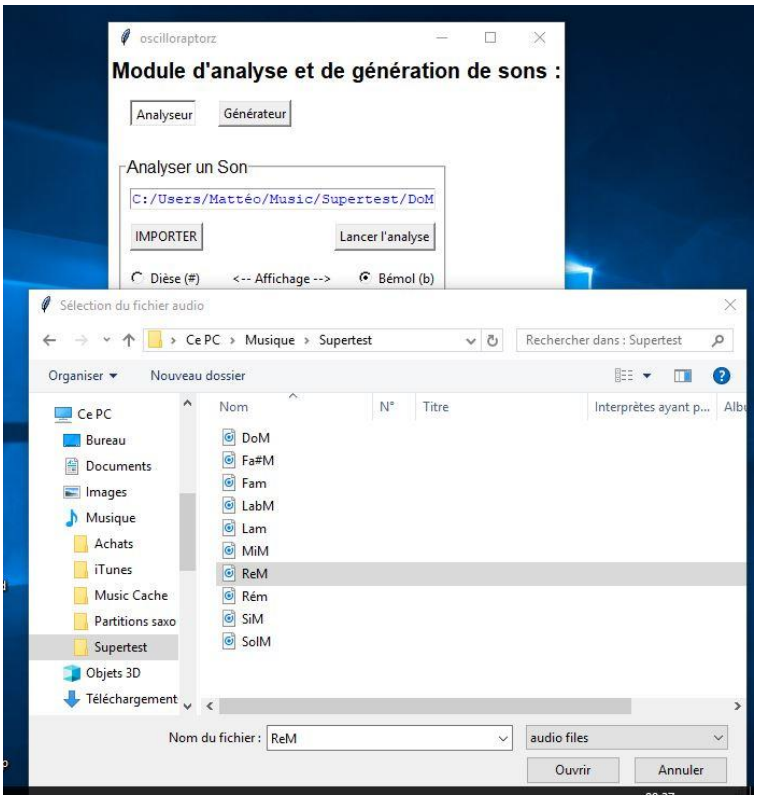
Voilà un premier croquis d'organisation (fait sur Paint) que l'on avait fait avant de commencer à coder :



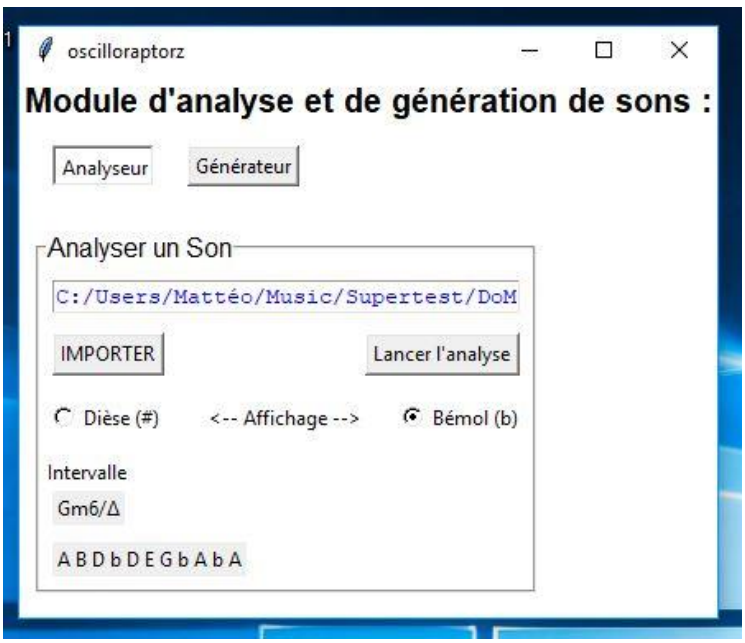
On peut déjà voir la partie de gauche dédiée aux paramètres. On remarque que les sorties textuelles ainsi que l'onglet « Générateur » sont absents.

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.

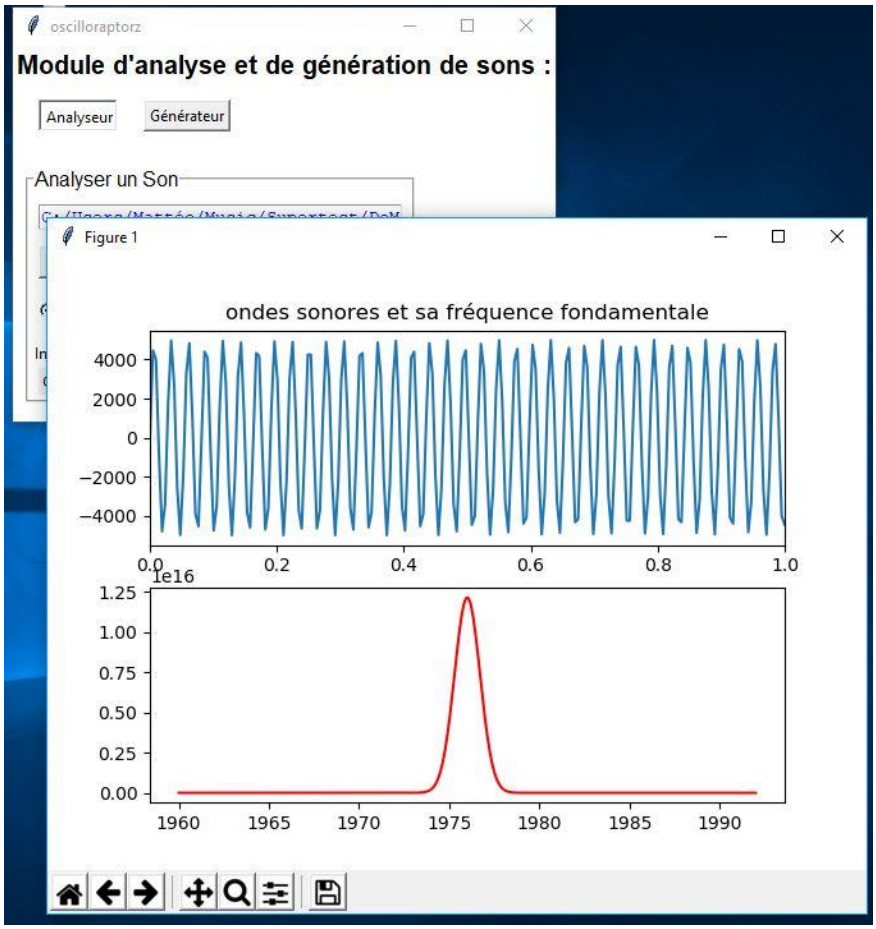
C – Quelques captures montrant le projet final



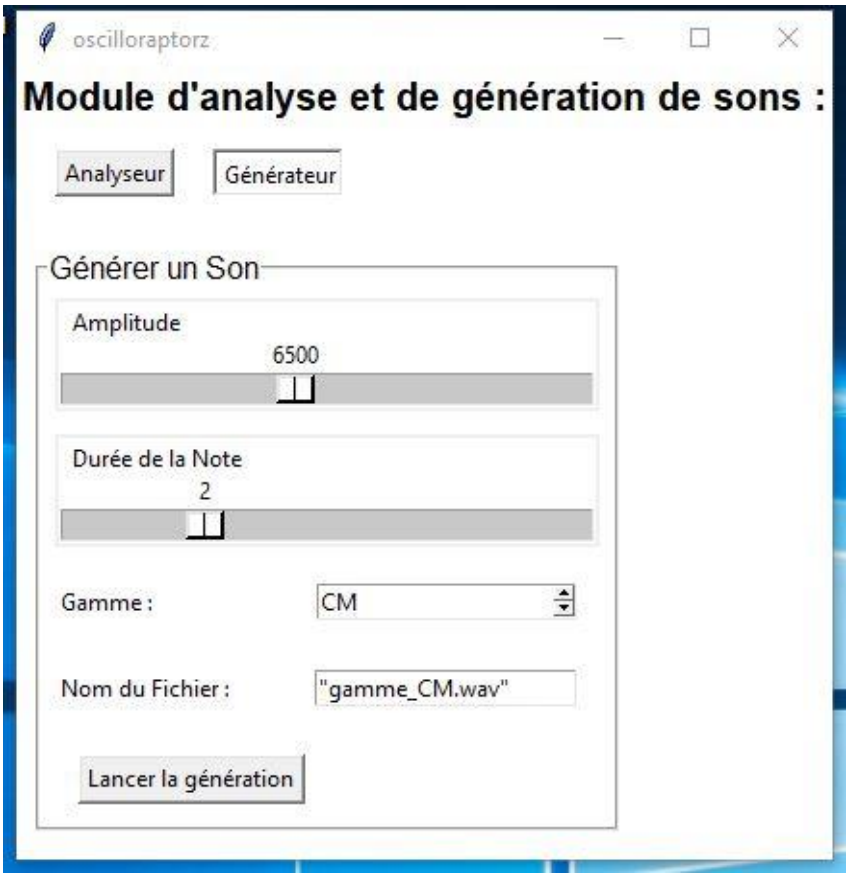
Sur cette capture on peut voir l'importation d'un fichier sonore test après avoir cliqué sur le bouton « importer » (qui ouvre l'uploader de fichier). Le fichier sélectionné, on peut voir apparaître son chemin d'accès dans le champ texte de la fenêtre en arrière-plan.



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).



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

Mon travail a consisté en la réalisation du GUI (Graphical User Interface, ou interface graphique), c'est-à-dire à penser et élaborer un dispositif de dialogue simple et intuitif entre l'utilisateur et le programme. Cela passe notamment par la mise en place de différents champs (scalebox, entrées textuelles, boutons et des différentes variables de contrôles associées) permettant à l'utilisateur d'entrer des données nécessaires au bon fonctionnement du programme, dans une fenêtre principale organisée (éviter l'invite de commande) et logique.

La première étape fut donc de trouver un module qui permettait de simplifier la mise en place d'une interface graphique. 3 choix principaux s'offraient alors à nous : pygame, wxWidgets (et son jumeau wxPython spécifique au langage Python) ainsi que Tkinter.

Pygame a vite été écarté car il est spécialisé dans la gestion des évènements et n'offre que peu de possibilités pour proposer des entrées. Bref trop orienté jeu vidéo et application en temps réel.

Nous nous sommes ensuite orienté vers wxWidgets qui nous paraissait plus complet et semblait avoir un fonctionnement similaire au HTML que nous avions déjà utilisé pour le projet de 2ème trimestre, mais il était en réalité beaucoup plus compliqué que ce que nous pensions et surtout il n'était pas compatible avec le module pyo que nous avions choisis pour gérer le son.

J'ai donc commencé à bricoler sur Tkinter pour me familiariser avec les différentes commandes et fonctions.

Lorsque nous avons abandonné Pyo qui faisait littéralement tout le travail à notre place en une petite trentaine de lignes de code, nous avons redéfini notre projet et cela nous as bien débloqué.

J'ai mis du temps à comprendre le fonctionnement des différentes méthodes de placement des éléments et ai passé de longues heures à m'arracher les cheveux, ne comprenant pas que la méthode de placement par grille (le .grid()) et par paquets (le .pack()) n'étaient pas compatibles et ne devaient pas être utilisées ensembles. J'ai opté pour la méthode .grid() en l'associant aux Frames afin de me rapprocher de la méthode des classes et des d'HTML qui m'est assez familière.

Le problème de cette méthode c'est qu'à chaque fois que l'on rajoute un élément (ou widget), il faut absolument restructurer complètement le programme afin que les frames interagissent bien entre elles.

Le programme final devant répondre à deux fonctionnalité bien distinctes : l'analyse et la génération de fichiers sonores, j'ai vite décidé de mettre en place deux onglets qui ne pouvaient coexister. Si l'utilisateur est dans le mode « Analyse », le GUI n'affiche plus le mode « Génération » et écrase les données de ce dernier et inversement. Cela permet d'une part de simplifier l'interface et d'éviter d'avoir des variables qui interfèrent. Cela se manifeste par la fonction « switchMode() » qui récupère la variable

Un autre challenge fut de comprendre comment importer un fichier audio et vérifier qu'il s'agit bien d'un .wav, c'est-à-dire un son brut non compressé (que l'on pouvait manipuler à l'aide du module wave.py).

C'est le rôle de la fonction « selec_fichier() » qui utilise le chemin d'accès pour récupérer les 4 derniers caractères (c'est-à-dire l'extension du fichier) pour les placer dans une liste (que l'on doit ensuite inverser, car on a commencé par le dernier caractère). On transforme alors cette liste en string() pour la comparer à la chaîne « .wav ». Il suffisait ensuite de mettre en place les messages d'erreurs et les réponses adaptées avec le module tkinter.messagebox. Malheureusement cette fonction multitâche souffre d'un énorme problème qui sera détaillé plus tard (partie Intégration).

Un des gros obstacles lors de la mise en commun du programme et de l'interface graphique fut de résoudre le problème des variables globales et locales ainsi que les variables de contrôle des boutons auxquels nous n'avions pas vraiment fait attention et de rajouter des fonctions « de structures » qui permettent simplement de regrouper d'autres fonctions et de les exécuter dans un ordre précis et en réponse à un input (comme un clic sur un bouton). C'est le cas de la fonction « Launch_Generation() » par exemple.

Au final, ma contribution au groupe fut la partie graphique (le GUI) et le review du code de Mattéo et de Balthazar (restructuration et corrections plus que création) afin de permettre son intégration.

V – Validation & Intégration

A – Intégration du Projet

Notre réalisation est un programme composé d'une interface graphique et de deux fonctionnalités principales :

- un module d'analyse sonore permettant de déterminer les notes jouées à partir d'un fichier wave ainsi que les intervalles les séparant et même la gamme à laquelle elles appartiennent,
- et un module de génération de fichiers audio contenant les 8 notes de la gamme sélectionnée.

Une interface graphique permet de faire le pont entre l'utilisateur qui peut modifier les différents paramètres et importer des fichiers de manière intuitive.

B – Tests, Correction et Validation

Mis à part quelques problèmes insolubles que nous avons dû contourner (incompatibilité entre certaines bibliothèques et une fois un crash systématique du programme sans rapport d'erreur qui nous a forcé à nous passer d'une fonction pourtant basique et qui fonctionnait bien toute seule), la plupart des erreurs étaient des fautes de syntaxes, ou de définition de variables (on définit la variable après la fonction, on fait des renvois à des fonctions définies plus bas dans le code…).

Un énorme problème inhérent à notre façon de travailler et du manque de temps est malheureusement toujours très présent et empêche le programme d'être véritablement ergonomique et utilisable. Nous avons littéralement chacun travaillé en créant des fonctions, et parfois multitâches. Toutes prises individuellement fonctionnent parfaitement.

Le problème est que toutes les variables créées dans ces fonctions y restent et il a été très compliqué de combiner toutes ces fonctions pour que le programme marche.

Ainsi, bien que nous ayons réglé un certain nombre de problèmes, certains bouts de codes n'ont pas pu être complètement reliés à l'interface graphique et ne sont donc pas fonctionnels.

Pour ma part j'ai testé chacune de mes fonctions sur Python Tutor, un site qui permet de tester en live, ligne par ligne un code Python tout en suivant les variables, listes et autres paramètres engagés afin d'être sûr de n'avoir pas commis de faute.

Pour la partie graphique, je n'ai posté mon code sur Github qu'une fois terminé. En effet, je codais seul, ce qui permettait d'enregistrer et d'exécuter le code à chaque modification. Certains widgets sont placés au pixel près. J'ai donc été un peu triste quand j'ai vu que sur le Mac de Balthazar, les polices par défaut étaient différentes et cassaient ces petits ajustements. Mais rien de bien méchant ici.

VI – Bilans & Perspectives

Perspectives :

Nous avions pris beaucoup de retard du fait d'une mauvaise définition de nos objectifs et des fonctionnalités à mettre en place. Le travail accompli en un temps réduit est plus qu'honorable. Il nous aurait fallu une ou deux semaine de plus pour véritablement aboutir à une version propre et utilisable sans fonctionnalités non implémentés et avec une réactualisation systématique de l'interface graphique.

En plus de cela, nous aurions pu ajouter plusieurs fonctionnalités, en particulier au niveau des graphiques que l'on aurait pu largement améliorer, diversifier et intégrer à la fenêtre principale. De plus, le module d'analyse aurait pu permettre d'établir le profil spectral complet et donc d'identifier le timbre précis d'un instrument (afin de différencier un piano et une guitare par exemple). Dans cette optique, nous aurions pu ajouter une banque de profils spectraux qui augmente à chaque fois que le programme est confronté à un nouvel instrument (en quelque sorte le rendre intelligent).

Nous avons détecté un problème d'octave que nous aurions pu fixer. En effet, L'analyseur détecte la bonne note à coup sûr mais se trompe parfois d'octave.

Nous aurions pu également ajouter un lecteur sonore qui aurait permis de jouer le son analysé en temps réel (et affiché les notes correspondante une par une). Nous avions également eu l'idée de dessiner les notes sur une partition en temps réel.

Mais bon, le projet était déjà très ambitieux pour des lycéens, surtout avec une motivation moyenne et un retard démentiel accumulé.

Bilan personnel :

Personnellement, ce projet m'a permis de vraiment me rendre compte de ce qu'est la programmation en informatique et en particulier en python, et à quel point ce langage est infiniment plus complexe et richeque je l'imaginais. Le fait de devoir aller chercher des modules, des bibliothèques et leur documentation sur le web fut clairement un challenge. France IOI c'est bien pour comprendre la syntaxe et les fonctionnalités basiques de Python mais cela reste quand même extrêmement limitant.

Ce projet m'a donné une excuse pour regarder des cours de Licence en ligne sur la physique des ondes et à me renseigner sur les liens entre physique et musique :

- Science Etonnante : « Les mathématiques de la musique » : https://youtu.be/cTYvCpLRwao
- Richard Taillet : « Quelques liens entre physique et musique (cours de L3) » : https://www.youtube.com/watch?v=iz6yG961O2Y

J'ai également pu approfondir et comprendre le concept des transformations de Fourier :

- 3Blue1Brown : « What is the Fourier Transform? A Visual introduction. » : https://www.youtube.com/watch?v=spUNpyF58BY
- Science4All : « L'analyse de Fourier | Infini 9 » : https://www.youtube.com/watch?v=eOehH6H42Es

VII – Diffusion du projet

https://github.com/lasource2018/oscilloraptor

Nous avons choisi la version 4.0 de Creative Common 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)

```
from tkinter import *
from tkinter import filedialog
from tkinter import messagebox
import tkinter.font as tkFont
import wave, struct, math
from struct import *
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit

#Définition des fonctions

#Importation du fichier et vérification du format (.wav only)
def selec_fichier():
    pathfilename =  filedialog.askopenfilename(initialdir = "/",title = "Sélection du fichier audio",filetypes = (("audio files","*.*"),("all files","*.*")))
    longueur = len(pathfilename)
    ext = []
    for boucle in range(1,5):
        ext.append(pathfilename[-boucle])
    ext.reverse()
    Extension = "".join(ext)
    if Extension != ".wav":
        messagebox.showerror("Erreur", "Mauvais Format : Veuillez sélectionner un fichier .wav")
    else:
        Filenamebox.delete(0.0, END)
        Filenamebox.insert(END, pathfilename)
        return pathfilename

#switch entre les onglets générer et analyser
def switchMode():
    SM = AnaGen.get()
    if SM == 0:
       Generateur.grid_forget()
       Analyseur.grid(row=3, column=1, sticky=NW, pady=15, padx=10)
    elif SM == 1:
        Analyseur.grid_forget()
        Generateur.grid(row=3, column=1, sticky=NW, pady=15, padx=10)


# Ecris un fichier audio .wav à partir d'une onde sinusoïde et d'une fréquence d'échantillonage
def Ecriture_Son(nomFichier, listeEch) :

    nbreOctets = 2
    nbreCanaux = 1
    fech = 44100

    nbreEchant = len(listeEch)
    parametres = (nbreCanaux, nbreOctets,fech, nbreEchant, 'NONE', 'notcompressed')


    Liste = []
    for i in range(nbreEchant) :
        b = struct.pack('h',listeEch[i])
        Liste.append(b[0])
        Liste.append(b[1])

    data = bytes(Liste)

    f = wave.open(nomFichier, 'wb')
    f.setparams(parametres)
    f.writeframes(data)
    f.close()
    messagebox.showinfo("Succès", "Le fichier son a bien été généré avec succès.")

# 2 dictionnaire classant les notes de D0 131 à DO 523 soit 2 octaves
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, "Ab":208,"A": 220, "A#" :233,"Bb" : 233, "B" : 247 }

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": 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 }

# ensemble des gammes majeures et mineures

CM = [oc["C"],oc["D"],oc["E"],oc["F"],oc["G"],oc["A"],oc["B"],oc2["C"] ]
Am = [oc["A"],oc["B"],oc2["C"],oc2["D"],oc2["E"],oc2["F"],oc2["G#"],oc2["A"]]

#Gammes Majeurs avec #
GM = [oc["G"],oc["A"],oc["B"],oc2["C"],oc2["D"],oc2["E"],oc2["F#"],oc2["G"]]
DM = [oc["D"],oc["E"],oc["F#"],oc["G"],oc["A"],oc["B"],oc2["C#"],oc2["D"]]
AM = [oc["A"],oc["B"],oc2["C#"],oc2["D"],oc2["E"],oc2["F#"],oc2["G#"],oc2["A"]]
EM = [oc["E"],oc["F#"],oc["G#"],oc["A"],oc["B"],oc2["C#"],oc2["D#"],oc2["E"]]
BM = [oc["B"],oc2["C#"],oc2["D#"],oc2["E"],oc2["F#"],oc2["G#"],oc2["A#"],oc2["B"]]
F_M = [oc["F#"],oc["G#"],oc["A#"],oc["B"],oc2["C#"],oc2["D#"],oc2["F"],oc2["F#"]]
C_M = [oc["C#"],oc["D#"],oc["F"],oc["F#"],oc["G#"],oc["A#"],oc2["C"],oc2["C#"]]

#Gammes Mineurs avec #
Em = [oc["E"],oc["F#"],oc["G"],oc["A"],oc["B"],oc2["C"],oc2["D#"],oc2["E"]]
```

```
Bm= [oc["B"],oc2["C#"],oc2["D"],oc2["E"],oc2["F#"],oc2["G"],oc2["A#"],oc2["B"]]
F_m = [oc["F#"],oc["G#"],oc["A"],oc["B"],oc2["C#"],oc2["D"],oc2["F"],oc2["F#"]]
C_m = [oc["C#"],oc["D#"],oc["E"],oc["F#"],oc["G#"],oc["A"],oc["C"],oc2["C#"]]
G_m = [oc["G#"],oc["A#"],oc["B"],oc2["C#"],oc2["D#"],oc2["E"],oc2["G"],oc2["G#"]]
D_m = [oc["D#"],oc["F"],oc["F#"],oc["G#"],oc["A#"],oc["C"],oc2["D"],oc2["D#"]]
A_m = [oc["A#"],oc["C"],oc2["C#"],oc2["D#"],oc2["F"],oc2["F#"],oc2["A"],oc2["A#"]]

#Gammes Majeures avec bémol
FM = [oc["F"],oc["G"],oc["A"],oc["Bb"],oc2["C"],oc2["D"],oc2["E"],oc2["F"]]
BbM= [oc["Bb"],oc2["C"],oc2["D"],oc2["Eb"],oc2["F"],oc2["G"],oc2["A"],oc2["Bb"]]
EbM = [oc["Eb"],oc["F"],oc["G"],oc["Ab"],oc["Bb"],oc2["C"],oc2["D"],oc2["Eb"]]
AbM = [oc["Ab"],oc["Bb"],oc2["C"],oc2["Db"],oc2["Eb"],oc2["F"],oc2["G"],oc2["Ab"]]
DbM = [oc["Db"],oc["Eb"],oc["F"],oc["Gb"],oc["Ab"],oc["Bb"],oc2["C"],oc2["Db"]]
GbM = [oc["Gb"],oc["Ab"],oc["Bb"],oc2["B"],oc2["Db"],oc2["Eb"],oc2["F"],oc2["Gb"]]
CbM= [oc["B"],oc["Db"],oc["Eb"],oc["E"],oc["Gb"],oc["Ab"],oc["Bb"],oc2["B"]]

# Gammes Mineurs avec bémol
Dm = [oc["D"],oc["E"],oc["F#"],oc["G"],oc["A"],oc["Bb"],oc2["C#"],oc2["D"]]
Gm = [oc["G"],oc["A"],oc["Bb"],oc2["C"],oc2["D"],oc2["Eb"],oc2["F#"],oc2["G"]]
Cm = [oc["C"],oc["D"],oc["Eb"],oc["F"],oc["G"],oc["Ab"],oc["B"],oc2["C2"] ]
Fm = [oc["F"],oc["G"],oc["Ab"],oc["Bb"],oc2["C"],oc2["Db"],oc2["E"],oc2["F"]]
Bbm = [oc["Bb"],oc2["C"],oc2["Db"],oc2["Eb"],oc2["F"],oc2["Gb"],oc2["A"],oc2["Bb"]]
Ebm = [oc["Eb"],oc["F"],oc["Gb"],oc["Ab"],oc["Bb"],oc2["B"],oc2["D"],oc2["Eb"]]
Abm = [oc["Ab"],oc["Bb"],oc2["B"],oc2["Db"],oc2["Eb"],oc2["E"],oc2["G"],oc2["Ab"]]


# génère une gamme composé d'une suite de sinus, le fichier est ensuite enregistré dans le dossier où est enregistré le code python
def ecrire_gamme(nom_fichier,gamme,amplitude,tps_note):
    E = []
    n = gamme
    for i in range(len(n)):
        O = [int(round(amplitude*math.sin(2*math.pi*n[i]*t*(1/44100)))) for t in  range(44100*tps_note)]
        E.extend(O)
    Ecriture_Son(nom_fichier, E)


#La fonction reliée au bouton Lancer la génération, qui récupère toutes les valeurs et lance "ecrire_gamme()"

def Launch_Generation():
    nom_fichier = SaisieNom.get()
    gamme = GammeEntry.get()
    amplitude = Amp.get()
    tps_note = DNote.get()
    ecrire_gamme(nom_fichier,gamme,amplitude,tps_note)
```

```
# extrait plusieurs données(nombre de canaux, fréquence d'échantillonage, nombre d'échantillon) ainsi que tout le contenu d'un fichier
d'un fichier audio .wav
def lecture_Son(nomFichier) :

    data = []
    f = wave.open(nomFichier, 'rb')

    nbCanaux = f.getnchannels()
    nbreEchant = f.getnframes()
    tailleEchant = f.getsampwidth()
    fech = f.getframerate()

    if tailleEchant == 2 :
        for i in range(nbreEchant) :
            b = f.readframes(1)
            try:
                val = struct.unpack('h', b)
            except:
                val = struct.unpack('2h',b)
            data.append(val[0])

    elif tailleEchant == 1 :
        for i in range(nbreEchant) :
            b = f.readframes(1)
            val = struct.unpack('b', b)
            data.append(val[0])
    else :
        print("Format de fichier non reconnu")


    return f,data,nbCanaux,nbreEchant,fech

#définission des intervalles

def IntervalleBasique(L,Intervalle, PositionListe):
    TailleListe = len(L)
    for loop in range(TailleListe):
        quotient = L[loop]/L[0]
        if 2.2 < quotient < 1.8 and Intervalle == "Octave":
            return True
        if 1.49 < quotient < 1.51 and Intervalle == "Quinte":
            if PositionListe:
                return loop
```

```
            else :
                return True
            break
        elif 1.32 < quotient < 1.34  and Intervalle =="Quarte":
            if PositionListe:
                return loop
            else :
                return True
            break
        elif 1.18 < quotient <1.23 and Intervalle == "TierceMin":
            return True
            break
        elif 1.24 <quotient <1.26 and Intervalle == "TierceMaj":
            return True
            break
        elif 1.16 < quotient < 1.10 and Intervalle == "Seconde":
            return True
            break

def IntervalleComplexe(L, Intervalle, CalcVTheorique):
    TailleListe = len(L)
    if CalcVTheorique :
        if Intervalle == "Sixte":
            ValeurIntervalle = L[0] * (5/4) *(4/3)
        elif Intervalle == "SeptiemeMaj":
            ValeurIntervalle = L[0] * (3/2) * (5/4)
        elif Intervalle == "SeptiemeMin":
            ValeurIntervalle = L[0] * (3/2) * (3/2)
        elif Intervalle == "Neuvieme":
            ValeurIntervalle = (L[0] * (3/2)*(4/3))/2
    else:
        if Intervalle == "Sixte":
            ValeurIntervalle = L[IntervalleBasique(L,"Quarte",True)] * (5/3)
        elif Intervalle == "SeptiemeMaj":
           ValeurIntervalle = L[IntervalleBasique(L,"Quinte", True)]*(5/4)
        elif Intervalle == "SeptiemeMin":
            ValeurIntervalle = L[IntervalleBasique(L, "Quinte",True)]*(3/2)
        if Intervalle == "Neuvieme":
            ValeurIntervalle = L[IntervalleBasique(L,"Quinte", True)] * (3/2)/2
    for loop in range(TailleListe):
        quotient = L[loop]/ValeurIntervalle
        if 0.9 < quotient <= 1.01:
                return True
                break

def GammeDeffinisseur(Liste):
    try:
        if IntervalleBasique(Liste, "TierceMaj",False):
            if IntervalleBasique(Liste,"Quarte", False) :
                if IntervalleComplexe(Liste, "SeptiemeMaj", False):
                    return "Ionien"
                else:
                    return "Mixolydien"
            else :
                return "Lydien"
        else :
            if IntervalleBasique(Liste, "Quinte", False):
                if IntervalleComplexe(Liste,"Sixte", True):
                    if Intervallecomplexe(Liste,"Neuvieme", True):
                        return "Aéolien"
                    else:
                        return "Phrygien"
                else:
                    return "Dorien"
            else:
                return "Locrien"
    except:
        return "Erreur"

#Détections d'accords
def AccordDeffinisseurTaille3(Liste):
    try :
        if IntervalleBasique(Liste, "TierceMaj", False):
            if IntervalleBasique(Liste,"Quinte", False):
                return ""
            elif IntervalleComplexe(Liste, "Sixte",True):
                 return "6"
        elif IntervalleBasique(Liste,"TierceMin", False):
            if IntervalleBasique(Liste,"Quinte", False):
                return "m"
            elif IntervalleComplexe(Liste,"Sixte", True):
                return "m6"
        elif IntervalleBasique(Liste,"Quarte", False):
            if IntervalleBasique(Liste,"Quinte", False):
                    return "sus4"
        elif IntervalleComplexe(Liste,"Neuvevieme", False):
```

```
            if IntervalleBasique(Liste,"Quinte", False):
                return "sus2"
        elif IntervalleBasique(Liste, "Octave", False):
            if IntervalleBasique(Liste,"Quinte", False):
                return "PowerChord HEEEELL YEAH"
    except:
        return "Unknown Chord"
def AccordDeffinisseurTaille4(Liste):
    try:
        if IntervalleComplexe(Liste,"SeptiemeMin", True):
            if AccordDeffinisseurTaille3(Liste, False) == "6" or "m6" or "sus4" or "sus2":
                return AccordDeffinisseurTaille3(Liste),"/7"
            else:
                return AccordDeffinisseurTaille3(Liste),"7"
        if IntervalleComplexe(Liste,"SeptiemeMaj",True):
            if AccordDeffinisseurTaille3(Liste) == "6" or "m6" or "sus4" or "sus2":
                return AccordDeffinisseurTaille3(Liste), "/Δ"
            else :
                return AccordDeffinisseurTaille3(Liste), "Δ"
    except:
        return AccordDeffinisseurTaille3(Liste)

#détection d'intervalles
def IntervalleDeffinisseur(Liste):
    try:
        Intervalle = ["Seconde","TierceMaj","TierceMin","Quarte","Quinte","Sixt","SeptiemeMin","septiemeMaj"]
        n = 0
        while n <= len(Intervalle):
            if IntervalleBasique(Liste, Intervalle[n], False) or IntervalleComplexe(Liste, Intervalle[n], True):
                return Intervalle[n]
            n = n + 1
    except:
        return "Erreur"

#fonction général qui réunie toute les fonctions qui viennent d'être défini
def sortie(ListeFreq,ListeNote):
    Taille = len(ListeFreq)
    if Taille == 1:
        return ListeNote[0]
    elif Taille == 2:
        return IntervalleDeffinisseur(ListeFreq)
    elif Taille == 3 or 4:
        return ListeNote[0], AccordDeffinisseurTaille4(ListeFreq)
    elif Taille > 5:
        return ListeNote[0], GammeDeffinisseur(ListeFreq)


# 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
l'octave d'origine
def octave(freq):
    n = 0
    while freq < 255 or freq> 500 :
        if freq < 255:
            freq = freq * 2
            n = n - 1
        elif freq > 500:
            freq = freq/2
            n = n + 1
    return freq, n


# Identifie une note à partir de sa fréquence, Diese == 1 ou 0 selon affichage souhaités # ou b
def notefreqjuste (frequence, Diese):
    a, n = octave(frequence)
    if  259 < a < 263:
        return "C",n

    elif 274 < a < 279:
        if Diese == 1:
            return "C#",n
        else:
            return "Db",n

    elif 290 < a < 295:
        return "D",n

    elif 309 < a < 313:
        if Diese == 1:
            return "D#",n
        else:
            return "Eb",n

    elif 326 < a <332:
        return "E",n

    elif 346 < a < 351:
        return "F",n

    elif 366 < a < 371:
```

```
        if Diese == 1:
            return "F#",n
        else :
            return "Gb",n

    elif 389 < a < 395:
        return "G",n

    elif 412 < a < 418:
        if Diese == 1:
            return "G#",n
        else:
            return "Ab",n

    elif 437 < a < 443:
        return "A",n

    elif  463 < a < 469:
        if Diese == 1:
            return "A#",n
        else:
            return "Bb",n

    elif 490 < a < 496:
        return "B",n



#fonction d'affinage qui sera utilisé plus tard
def func(x,a,b,c):
    return c * np.exp( -np.power( (x-a)/b, 2) )


#fonction qui va lire le fichier, séparés les données pour chaque note, faire la transformée de fourier pour chaque note et enfin afficher un graphique représentant les 2 courbes


def analyser_son(nomFichier):
    plt.ion     #on utilise pyplot en interactif

    # Ouverture du fichier wav a decrypter
    f, x, nbCanaux, nbFrames, fech = lecture_Son(nomFichier)

    #Découper le fichier pour analyser chaque note après l'autre
    frequences, freq_gauss, FreqNoteJuste, ListeNote,  = [], [], [], [],
    larg_frame = 44100
    for posi in range(0,nbFrames,larg_frame):

        # Sequence contenant une note
        f.setpos(posi)
        donnee = f.readframes(larg_frame)
        data = struct.unpack('%sh' % (larg_frame*nbCanaux ), donnee)

        # Transformee de Fourier
        fourier  = np.fft.fft(data)
        valeur_réelle   = np.real(fourier * fourier.conjugate())
        freqs = np.fft.fftfreq(len(fourier)) * nbFrames

        # Estimation de la frequence
        idx = np.argmax(valeur_réelle)
        f0, maxi = np.abs(freqs[idx]), valeur_réelle[idx]
        frequences.append( f0 )

        #Ajustement par une gaussienne (permet l'ajustement des valeurs de f0)
        ind = np.where( np.abs(freqs - f0) < 20 )
        popt, pcov = curve_fit( func, freqs[ind], valeur_réelle[ind]/maxi, p0=[f0,1,1] )
        a, b, c = popt
        freq_gauss.append(a)

        note, n = notefreqjuste (a, Diese)
        ListeNote += note

        FreqNoteJuste.append(oc2[note]*(n-1))

        #Affichage
        signal = []
        for i in range(0,len(data),210):
            signal.append(data[i])
        fs = 44100//210

        Time=np.linspace(0, len(signal)/fs, num=len(signal))

        plt.subplot(211)
        plt.plot(Time,signal)
        plt.title("ondes sonores et sa fréquence fondamentale")
        plt.xlim(0,1)
```

```
        plt.subplot(212)
        fnew = np.linspace( freqs[ind][0], freqs[ind][-1], 512)
        plt.plot( fnew, maxi * func(fnew,a,b,c), 'r')

        plt.draw()
        plt.pause(3)
        plt.clf()


    f.close()
    intervale = sortie(FreqNoteJuste, ListeNote)
    return ListeNote, intervale

#La fonction reliée au bouton Lancer l'analyse, qui sélectionne et lance la fonction "analyser_son"
def Launch_Analyser():
    nomFichier = selec_fichier()
    a, b = analyser_son(nomFichier)
    AllNotes = " ".join(a)
    return AllNotes


#Début de l'interface graphique
fenetre = Tk()
fenetre.title("oscilloraptorz")
fenetre.configure(background="white")
Label (fenetre, text="Module d'analyse et de génération de sons :", bg="white", fg="black", font="none 15 bold") .grid(row=1, column=1,
sticky=NW)
FontSS = tkFont.Font(size=18)


'_____________________________________________________________________________________________________________________'

#fenêtre de droite
CadreMode = LabelFrame(fenetre, bg="white", bd=0, height=50)
CadreMode.grid(row=2, column=1, sticky=NW, pady=7, padx=10)
AnaGen = IntVar()
AnaGen.set(0)
Ana = Radiobutton(CadreMode, text="Analyseur", value=0, variable=AnaGen, indicatoron=0, command=switchMode)
Ana.grid(row=2, column=1, sticky=NW, pady=5, padx=10)
Gen = Radiobutton(CadreMode, text="Générateur", value=1, variable=AnaGen, indicatoron=0, command = switchMode)
Gen.grid(row=2, column=2, sticky=NW, pady=5, padx=10)

'_____________________________________________________________________________________________________________________'

#Fenêtre analyseur
Analyseur = LabelFrame(fenetre, bg="white", bd=2, height=400, text="Analyser un Son", font="FontSS")
Analyseur.grid(row=3, column=1, sticky=NW, pady=15, padx=10)

#Importation du fichier
Filenamebox = Text(Analyseur, width = 35, height=1, wrap=WORD, bg="white", fg="blue")
Filenamebox.grid(row=3, column=1, sticky=NW, pady=10, padx=8)
Import = Button(Analyseur, text="IMPORTER", command=selec_fichier)
Import.grid(row=4, column=1, sticky=NW, padx=8, pady=2)
AnaLaunch = Button(Analyseur, text="Lancer l'analyse", command=Launch_Analyser)
AnaLaunch.grid(row=4, column=1, padx=8, pady=2, sticky=E)

#Affichage en Diese ou en bémol
Diese = IntVar()
Diese.set(1)
Di = Radiobutton(Analyseur, text='Dièse (#)', value=1, variable=Diese, bg ="white")
Di.grid(row=6, column=1, sticky=W, padx=4, pady=11)
Label (Analyseur, text="<-- Affichage -->", bg="white") .grid(row=6, column=1, padx=4, pady=11)
Be = Radiobutton(Analyseur, text='Bémol (b)', value=0, variable=Diese, bg ="white")
Be.grid(row=6, column=1, sticky=E, padx=4, pady=11)

#Affichage des intervalles et des notes
InterNote = LabelFrame(Analyseur, bg="white", text="Intervalle", bd=0)
InterNote.grid(row=7, column=1, sticky=W, padx=4, pady=1)
PrintInter = "Gm6/Δ" #relier avec la fonction "sortie"
Label(InterNote, text=PrintInter).grid(row=1, column=1, sticky=W, padx=4, pady=3)
NomNotes = LabelFrame(Analyseur, bg="white", text="Notes détectées", bd=0)
Label(InterNote, text=Launch_Analyser()).grid(row=2, column=1, sticky=W, padx=4, pady=7)

'_____________________________________________________________________________________________________________________'

#Fenêtre Générateur
Generateur = LabelFrame(fenetre, bg="white", bd=2, height=400, text="Générer un Son", font="FontSS")
Generateur.columnconfigure(1, minsize=35)

#Amplitude
Amp = Scale(Generateur, orient="horizontal", from_=0, to=15000, resolution=100, sliderlength=20, label="Amplitude", length=277,
bg="white")
Amp.grid(row=1, column=1, sticky=NW, padx=8, pady=6)

#Durée de la note
DNote = Scale(Generateur, orient="horizontal", from_=1, to=5, resolution=1, sliderlength=20, label="Durée de la Note", length=277,
bg="white")
DNote.grid(row=2, column=1, sticky=NW, padx=8, pady=6)

#Gamme
Gammetext = Label(Generateur, text="Gamme :", bg="white")
Gammetext.grid(row=3, column=1, sticky=NW, padx=8, pady=12)
GammeEntry = Spinbox(Generateur, bg="white", values=("CM", "GM", "DM", "AM", "EM", "FM", "BbM", "EbM", "AbM", "Am", "Em", "Bm","Dm","Gm"),
wrap=True)
GammeEntry.grid(row=3, column=1, padx=20, pady=12, sticky=E)

#saisie du nom du fichier
NameFileEntry = Label(Generateur, text="Nom du Fichier :", bg="white")
NameFileEntry.grid(row=4, column=1, sticky=NW, padx=8, pady=12)
SaisieNom = Entry(Generateur, bg="white", width = 22)
SaisieNom.grid(row=4, column=1, padx=20, pady=12, sticky=E)

#Bouton générer
GenLaunch = Button(Generateur, text="Lancer la génération", command=Launch_Generation)
GenLaunch.grid(row=5, column=1, padx=20, pady=12, sticky=W)

'_____________________________________________________________________________________________________________________'



fenetre.mainloop()
```