
# Rapport de soutenance n°2

*Chef d'equipe* : Mayes Tidjedam
Camille Poingt
Alexandre Voinier

Promo 2024

# Contents

# 1 Introduction

Nous sommes à présent aux deux tiers de la création de Tmac-Studio. Le projet avance bien et nous sommes plus que satisfaits du résultat actuel. Nous sommes toujours aussi motivés pour faire de ce projet une réussite.
Pour cette deuxième étape, les bases de l'application étant solides, nous avons pu nous concentrer sur la création de contenu formant l'application. Nous avons implémenter de nouvelles fonctionnalités telles que des effets sur le son. De nouveaux outils sont également présents et viennent enrichir l'expérience utilisateur. Vous allez donc pouvoir observer ce changement sur les deux tableaux présents dans la cinquième section de ce rapport. De plus, nous vous présenterons tout ce que nous avons fait jusque là.

# 2 Présentation du groupe

TMAC a été créé fin janvier 2021 dans le cadre du projet de S4. Il est composé de quatre étudiants de l'EPITA qui ont pour ambition de révolutionner le monde de l'informatique. Vous trouverez ci_dessous une brève présentation de chaque membre du groupe afin d'en apprendre un peu plus sur ce groupe très peu connu du grand public.

## 2.1 Les membres du groupe

### 2.1.1 Mayes Tidjedam

Chef du projet TMAC studio, je suis aussi un grand féru de musique.
La création a toujours fait parti de ma vie et c'est pourquoi je me réjouis de pouvoir faire partie intégrante de ce groupe de travail. Je perçois ce projet également comme une occasion de poursuivre le développement de compétences telles que : l'écoute, la communication et le management.

### 2.1.2 Camille Poingt

J'ai toujours aimé travailler sur un ordinateur sans trop m'intéresser à l'informatique.
Ce n'est qu'en entrant à l'EPITA que j'ai commencé à coder, à m'intéresser à ce domaine. Je ne suis sûrement pas la meilleure, mais je compte faire tout mon possible pour parvenir à mener à bien ce projet.

### 2.1.3 Alexandre Voinier

Futur Directeur de la sécurité chez Google, je viens me former dans cette école réputé du sud de Paris, EPITA. Je viens de la banlieue parisienne à côté de Disneyland. Avec ce projet j'espère découvrir de nouvelles facettes de l'informatique et ainsi agrandir ma palette de compétences.

## 2.2 Les logiciels d'enregistrement vidéo

Il existe deux grandes familles de logiciels permettant d'enregistrer les flux vidéos d'un ordinateur. La première permet une diffusion en direct tandis que la seconde permet de sauvegarder un fichier vidéo pour le diffuser en différé. Ces logiciels permettent également d'enregistrer et/ou de retransmettre le son provenant du micro et/ou du support.

OBS Studio


Aucun de ces services ne permet d'avoir une visualisation du son précise et aucune ne permet de faire des modifications sur ce dernier.

## 2.3 Qu'en avons nous tiré?

Le monde de l'audio sur pc n'est composé que d'extrêmes. D'un côté nous avons des outils difficiles à prendre en main mais permettant d'avoir un rendu optimal.
De l'autre nous avons un outil facile d'utilisation ne permettant pas toujours d'avoir le meilleur résultat et commence à se faire vieux (Audacity créé en 2000).
Nous souhaitons apporté de la nuance et fournir une solution alternative à l'utilisateur. Allier la facilité d'utilisation de l'un avec la qualité de rendu de l'autre. C'est donc avec un objectif clair que TMAC s'est formé et à commencer à concevoir TMAC Studio

# 3 Présentation des technologies utilisées

Afin de mener à bien notre projet, nous avons eu besoin de certaines librairies.

## 3.1 GTK+ et Glade

Dans un premier temps, nous avons utiliser la bibliothèque gtk+ et le logiciel glade pour construire une interface graphique facile à utiliser. Nous utilisons Glade car c'est un logiciel qui nous permet de facilement créer notre interface et d'effectuer des modifications sans trop de difficultés.



## 3.2 FMOD

Par ailleurs, nous avons trouvé une librairie pour réaliser toutes nos opérations sur le son : la librairie FMOD. Nous n'avons jamais utilisé cette librairie mais elle nous semble pour le moment être la plus adaptée à notre projet.

# 4 Répartition et planning des tâches

## 4.1 Répartition des tâches

| | Mayes | Alexandre | Camille |
|---|---|---|---|
| Enregistrement du son | | | |
| Affichage du spectre du son | | | |
| Jouer la piste audio | | | |
| Sauvegarde des fichiers audios | | | |
| Importation de fichiers audios | | | |
| Interface graphique | | | |
| Découpage de la piste audio | | | |
| Réglage de la réverbe | | | |
| Réglage de la hauteur | | | |
| Site Web | | | |

Légende :
très impliqué(e)
moyennement impliqué(e)
très peu impliqué(e)

## 4.2 Planning des tâches

| | Première | Deuxième | Troisième |
|---|---|---|---|
| Enregistrement du son | ✓ | | |
| Affichage du spectre du son | 40% | ✓ | |
| Jouer la piste audio | ✓ | | |
| Sauvegarde des fichiers audios | 30% | ✓ | |
| Importation de fichiers audios | ✓ | | |
| Interface graphique | 33% | 66% | ✓ |
| Découpage de la piste audio | | | ✓ |
| Réglage de la réverbe | | | ✓ |
| Réglage de la hauteur | | ✓ | |

## 4.3 Description des tâches

Enregistrement du son: C'est la principale fonctionnalité de TMAC Studio. Le son pourra provenir de l'orinateur même ou d'une source extérieur. L'objectif est de le conserver pour pouvoir ensuite l'exploiter.

Affichage du spectre du son: Ce dernier permet d'avoir une vision global du son enregistré. Le volume et/ou la hauteur de ce dernier à chaque instant sont des données facilement déchiffrables sur un spectre de son.

Jouer un son enregistré: Cela permettra à l'utilisateur de savoir où en est son traitement du son sans avoir à sauvegarder puis jouer le fichier audio lui même.

Sauvegarde des fichiers audios: Le rendu final apres traitement devra pouvoir être stocker sur la machine et être jouée.

Importation de fichiers audios: L'utilisateur pourra ainsi réutiliser des fichiers audios pré-enregistrer et/ou revenir sur un travail laissé en suspend.

Interface graphique: C'est cette dernière qui permettra à l'utilisateur de se servir confortablement de TMAC studio. Elle devra être sobre et ergonomique tout en ayant une certaine élégance.

Découpage de la piste audio: Cette fonctionnalité permettra de réorganiser la piste audio. Nous pourrons retirer une partie de cette dernière et supprimer les silences.

Réglage de la réverbe: Nous pourrons ajouter ou créer de la réverbe. Cette dernière sera modulable selon plusieurs critères(la taille de la pièce ).

Réglage de la hauteur: Cet effet permettra de rendre le son plus grave ou plus aigu.

## 4.4 Ressentis concernant le projet et son avancement

Le groupe comptait à l'origine 4 membres. Nous ne sommes aujourd'hui plus que 3. Malgré ce changement, notre groupe avions su rebondir et redoubler d'efficacité. Nous avions pris cet évènement comme une expérience à prendre.
L'équipe est satisfaite du travail réalisé tant par sa quantité que sa qualité. C'est riche de cette sensation de satisfaction que nous commençons la troisième étape du la conception de notre projet.

# 5 Nouveautés

## 5.1 Spectre

Le spectre d'un son est le tableau ou la représentation graphique des fréquences de ce son a un instant t.
Le spectre a commencé dès la première période du projet. Nous avons commence par des recherches sur ce qu'est le spectre et sur comment nous allions le récupérer. Dès la première soutenance nous avons tenté de l'implémenter mais nous nous sommes heurté à de nombreux problèmes.

En effet l'implémentation du spectre dans la libraire a totalement change il y a peine trois ans. Entraînant un déficite de documentation, d'exemples et de forums sur le sujet. La documentation étant assez complique si ce n'est flou nous nous sommes tournes vers les forums. Malheureusement la majorité d'entre eux ne fournissent pas de réponse ou alors avec des explications inexistantes ne nous permettant pas comprendre et de réadapter a notre projet leur solution.

Notre compréhension augmentait quand même après chaque lecture, chaque essaye. A la première soutenance nous parvenions a récupérer un tableau qui devrait contenir les valeurs de nos fréquences, mais a chaque fois il était vide.

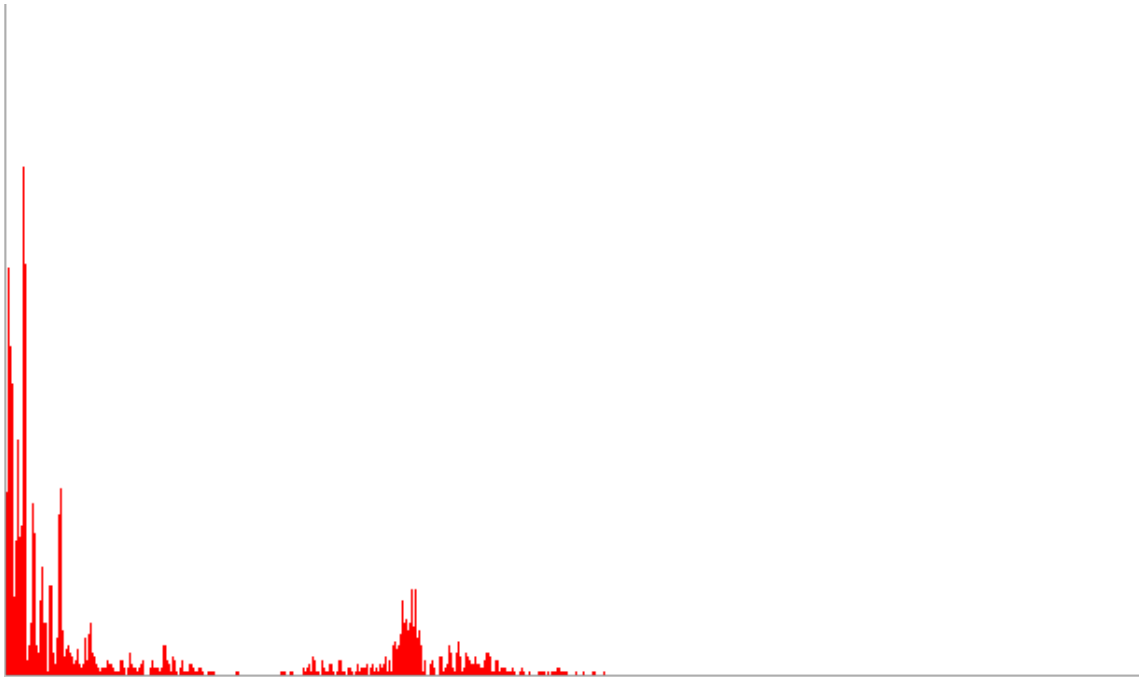
Enfin après une bonne semaine de vacances les essaient on repris. Pendant une semaine ce fut échec sur échec. Finalement ça marchait, il ne restait plus qu'à l'adapter a notre programme. Cela a été vite réussit en comparaison en une journée le spectre était adapter.

Il ne nous restait plus qu'à l'afficher sur notre interface graphique.
Nous avions fait un TP en S3 sur GTK et nous avions utilisé une zone de dessin. Nous nous sommes donc servi de ce TP comme base pour comprendre l'utilisation d'une zone de dessin avec GTK. Une fois son fonctionnement compris, il ne nous restait plus qu'à faire un peu de maths et le tour était joué.

Nous récupérons 512 valeurs de fréquences différentes. nous divisons alors la largeur de la zone de dessin par 512 pour obtenir la largeur des colonnes pour chaque fréquence. Ensuite les valeurs sont entre 0.001 et 5 nous multiplions ces valeurs par la hauteur de la zone de dessin divise par 5. Enfin nous affichons ces colonnes en rouge.

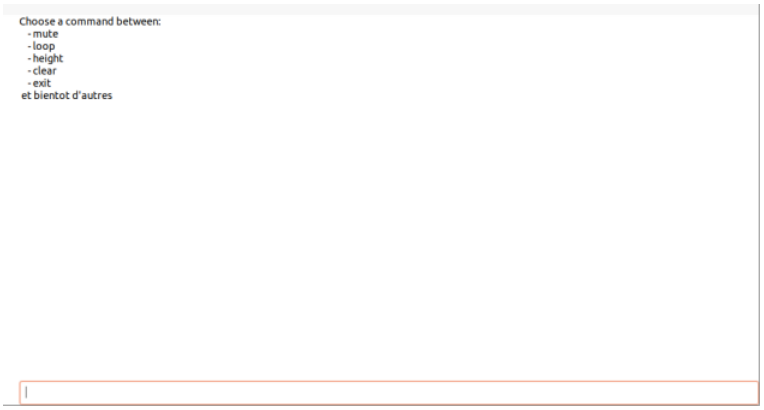
Le spectre est mis a jour toutes les 25ms avec FMOD, nous avons donc créer un événement qui va venir le récupérer les valeurs et mettre a jour l'interface graphique toutes les 25ms.

Quand tout marchait bien nous avons alors décide de merge le spectre dans la branche principale. Alors les problèmes commençait de nouveau. Les problèmes venait des effets sur le son comme le fait que le son soit joue en en boucle ou qu'on puisse mute le son. Mais comme nous avions finis tout le reste nous avons pu tous nous y mettre dessus le temps d'un après-midi et régler tous les problèmes.



## 5.2 Terminal de commande

Nous recherchions une méthode permettant à l'utilisateur d accéder à toutes les fonctionnalités de l'application de la façon la plus ergonomique possible. C'est alors que nous est venu l'idée d'implémenter un terminal de commande qui pourra récupérer la demande exprimée et la traiter correctement.

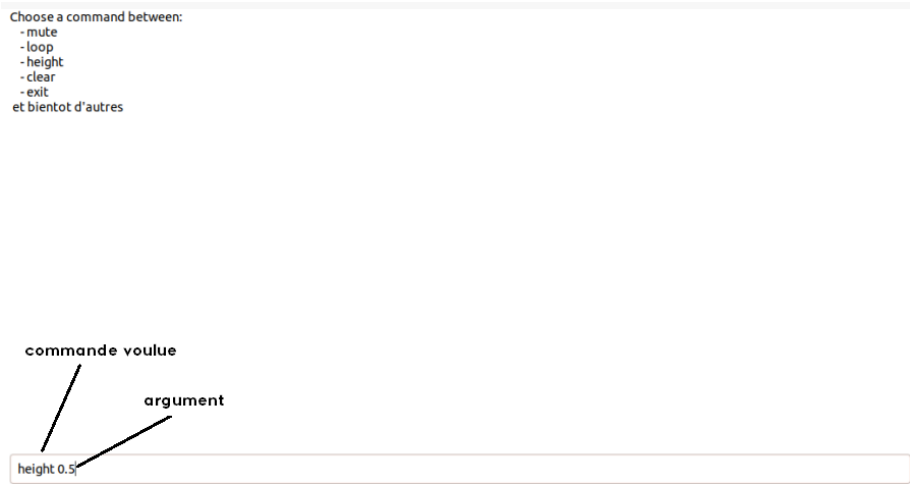



### 5.2.1 La conception

Avant tout, il nous fallait un moyen de reconnaître une requête qui prend la forme d'une chaîne de caractères entrés par l'utilisateur. Nous devions donc récupérer la chaîne pour ensuite la reconnaître.
De plus nous avons utilisé un système d'évènement qui permet de relevé le moment où la touche entrée a été pressée. A ce moment, tout ce qui a été entré dans la boite de dialogue est récupérer avec la fonction gtk_entry_get_text() et est stocké dans une variable de type "gchar" en lien avec GTK. C'est assez pratique car ce type est très proche du type "char*". Nous allons donc pouvoir l'utiliser pour reconnaître la commande demandée.

### 5.2.2 La fonction de reconnaissance

Cette fonction permet de sélectionner un nombre de caractères à comparer et renvoies 0 si les deux chaînes comparées sont les mêmes, sinon elle renvoie 1. L'intérêt principale d'avoir coder nous même cette fonction de reconnaissance est qu'elle nous permet de distinguer la commande de ses arguments et ainsi de les récupérer séparément.



### 5.2.3 Les fonctionnalités

A l aide du terminal de commande, vous pouvez effectuer toute sorte d action dans l'application telles que :

- rec : Commencer un enregistrement
- recstop : Arrêter un enregistrement
- play : Jouer un son déjà chargé
- pause : Mettre le son qui est en train d'être joué en pause
- clear : Nettoie la boite de dialogue
- exit : Permet de quitter l'application

Nous pouvons aussi appliquer des modifications au son chargé via des commandes telles que:

- height x : Change la hauteur du son en fonction du paramètre flottant x
- mute : Rend muet la chaîne utilisée ce qui nullifie le volume du son quel que soit le volume de l application
- loop x : permet au son d'être joué en boucle si x vaut 1 et arrête cette fonctionnalité

L'objectif de ce terminal est (à terme) de permettre de tout faire via des commandes tapées afin d'accélérer le travail.

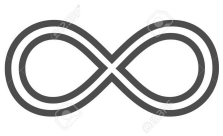
## 5.3 Les différents outils

Comme énoncé lors de la dernière soutenance, un de nos objectifs pour cette soutenance était de commencer à implémenter des effets sur le son. Nous avons réussi à implémenter certains effets qui pourront être activés/modifiés via le terminal de commande. Ainsi nous avons réussi à implémenter les fonctionnalités boucle, muet et hauteur.

### 5.3.1 Le mode boucle

Premièrement l'option boucle est la seule que l'on a été obligé de modifier dans le but de faire marcher le spectre. En effet, nous avions dans un premier temps explorer la solution de changer le mode du son avec la fonction de la librairie "FMOD FMOD_Sound_SetMode()".
Ainsi cette fonction aurait été utilisée tout simplement avec un booléen. Lorsque ce booléen était à vrai, le mode devenait "FMOD_LOOP_NORMAL" et au contraire lorsque ce booléen était à faux, le mode devenait "FMOD_LOOP_OFF".
Cette solution semblait parfaite et efficace mais non cela ne pouvais pas être aussi facile !

Ainsi lors de l'introduction du code construisant le spectre dans notre projet, un problème est survenu. En effet, lorsque le son redémarre avec la bouche activée, nous avons remarqué un petit moment où le son s'arrêtait. Pendant ce court moment, le spectre ne pouvait pas récupérer de valeur ce qui provoqua une "segmentation fault".



Une première solution était d'utiliser la fonction "FMOD_Channel_IsPlaying()" qui indique comme son nom l'indique si un canal est en train de jouer un son ou non. Après l'introduction de cette fonction dans notre code, nous nous sommes vite rendu compte que cela n'allait pas être possible. En effet, le temps d'exécution de cette fonction est assez grand pour faire planter l'évènement construisant le spectre.

Nous avons donc décider de créer nous même notre mode boucle. Cela n'est pas très compliqué et d'ailleurs nous avons même utilisé finalement la fonction "FMOD_Channel_IsPlaying()". En effet le principe est tel qu'on modifie un booléen nouvellement introduit dans la structure "MusStruct" selon l'activation du mode boucle.

Ainsi on vérifie souvent si le canal affichant le spectre joue un morceau, si ce n'est pas le cas et que le mode boucle est activé, on rejoue la musique avec la fonction "FMOD_System_PlaySound()". Dans l'autre cas où il n'y a pas de son joué mais que le mode boucle n'est pas activé, on stoppe tout simplement l'évènement qui va récupérer le spectre à l'aide de la fonction "g_source_remove()".

### 5.3.2 Le mode mute

Ce mode fut le plus simple à coder dans le sens où les fonctions existent déjà dans la librairie FMOD. Mais bon après tout comment pourrait-on réaliser un logiciel ressemblant à Audacity sans un mode mute.

Ainsi cette fonction est plutôt simple. En effet dans un premier temps on va récupérer un booléen correspondant au statut du mode mute avec la fonction "FMOD_ChannelGroup_GetMute()". Ainsi si ce booléen résultant est à vrai, on enlève le mode mute et dans le cas contraire, on active le mode mute. Ces deux actions se feront avec la fonction "FMOD_ChannelGroup_SetMute()".

### 5.3.3 Réglage de la hauteur

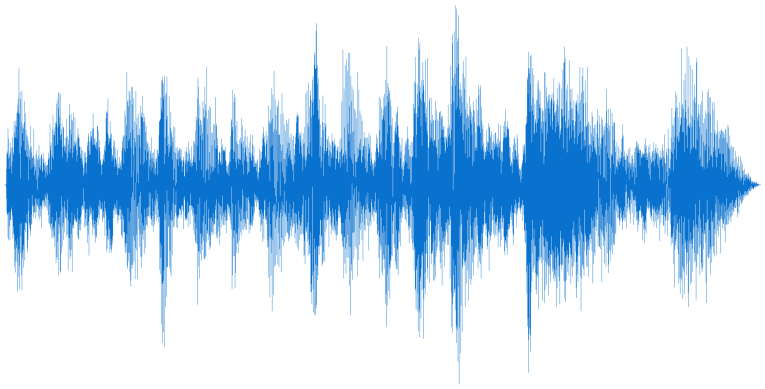
De manière générale la hauteur fait référence à la fréquence fondamentale d'un son. Ainsi comme on peut le voir ci-dessous les sons aigus vont avoir une fréquence plus élevée que la normale et au contraire les sons graves auront une fréquence plus basse.


Alors que nous cherchions un moyen d'accélérer et de diminuer la vitesse de lecture de la musique, nous avons eut l'idée de cet effet. Après mainte recherches, nous avions trouvé la fonction "FMOD_Channel_Set_Pitch()" mais celle-ci accélérait la musique au fur et à mesure qu'elle se jouait.
Il faut savoir que cette fonction augmente également la hauteur du son car de manière générale, augmenter la vitesse du son augmente sa hauteur. Par contre augmenter la hauteur d'un son n'entraîne pas forcément l'augmentation de sa vitesse de lecture et d'ailleurs ce fait est bien visible dans le résultat final.

Par la suite nous avons donc chercher un autre moyen de changer cette vitesse et nous avons trouver la méthode qui consiste à ajouter un "DSP" de type "PITCHSHIFT" au canal lisant le son. Ainsi le "pitch" du son pouvait être modifier dans cette structure à l'aide de la fonction "FMOD_DSP_SetParametersFloat()" sachant que le "pitch" de base est égal à 1 et que un pitch de 0.5 fait descendre le son à un octave en dessous et un octave au dessus correspond à un "pitch" égal 2.
Finalement en testant cette méthode, nous nous sommes rendu compte qu'elle ne modifiait non pas la vitesse du son mais bel et bien sa hauteur.

## 5.4 Courbe des amplitudes

La courbe des amplitudes est l'une des représentation du son la plus connue comme on peut le voir ci-dessous. En effet on peut la retrouver dans la plupart des logiciels de traitement de son tel que Audacity, principal inspiration de notre projet.



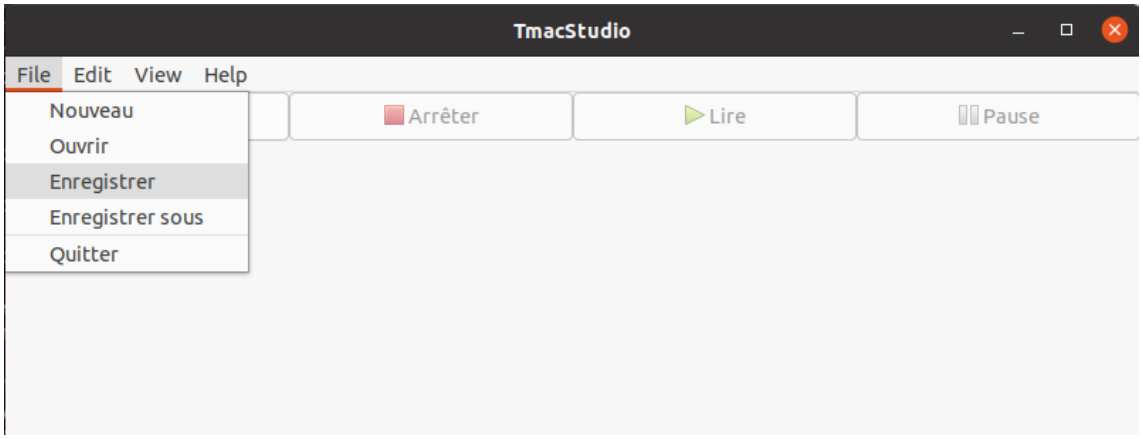
Cependant représenter le son est selon nous l'une des tâches les plus compliquées à réaliser pour ce projet notamment à cause de la librairie utilisée. Si le spectre a pu être finalement réalisé pour cette soutenance, cela n'a pas été le cas pour la courbe des amplitudes. Cette absence peut être expliqué principalement par un manque de temps mais aussi par un manque de compréhension en la librairie utilisée. Malgré cela des recherches ont été effectuées sur le sujet et elles seront présentées dans les paragraphes qui suivent.

A l'heure d'aujourd'hui deux options s'offrent à nous et nous ne savons pas encore laquelle choisir. Dans un premier temps, il est possible de récupérer les données du son dans un tableau avec la fonction FMOD_Sound_ReadData(). Cette méthode permettrait selon certain forum de construire une courbe statique du son. Cependant il pourrait y avoir quelque problèmes liée à cette méthode. En effet, premièrement rien ne nous dit que les données récupérées via cette fonction correspondront bien aux amplitudes progressives du son. Sur la documentation de la librairie FMOD, il est indiqué que ces données récupérées sont dans un codex propre à FMOD.

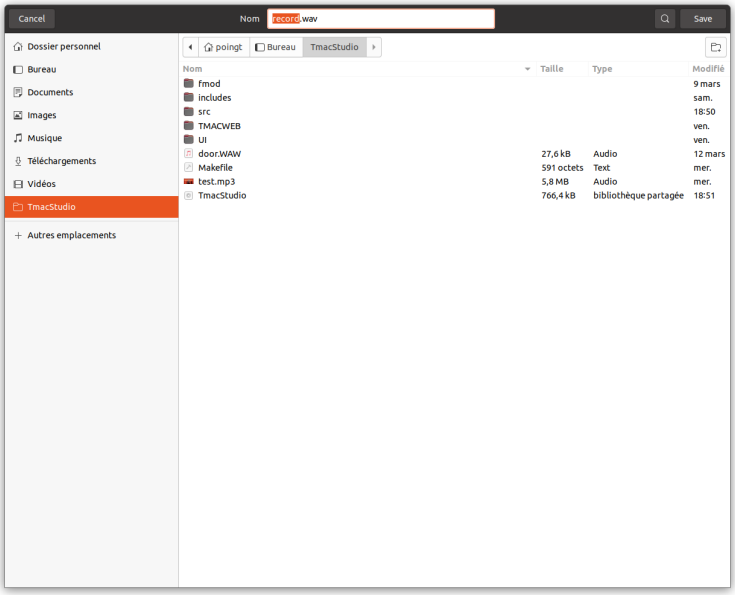
D'autre part une autre méthode s'offre à nous : récupérer les données du son non pas de manière statique comme énoncé précédemment mais de manière dynamique lorsque le son est en train d'être joué. Cette méthode présente des avantages puisque comme pour le spectre elle nous permettrait de récupérer les données en temps réel et de plus avec un calcul assez simple, on pourrait récupérer les décibels évolutifs du son joué. Ainsi comme pour le spectre, il s'agirait d'ajouter un DSP sur la canal joué et de récupérer au fur et à mesure les données FFT en leur appliquant à chaque fois la formule décibels = 20*log($FTT\_data$).

Pour finir cette tâche figure parmi les plus importantes à réaliser pour la prochaine soutenance. Elle sera sans doute plus facile à réalisée maintenant que nous avons le spectre.

## 5.5 Sauvegarde




Si cette étape avait déjà été commencé lors de la dernière soutenance, elle est aujourd'hui bel et bien terminée. A la dernière soutenance, nous avions tout de même réussi à récupérer les données du son telles que la fréquence, le type. Ces données pouvaient déjà à l'époque être stocker dans un fichier. Mais alors un problème persistait : nous n'arrivions pas à récupérer la longueur d'un son. Nous avions donc en sortie un fichier avec des informations à propos du son mais qui ne pouvait pas jouer celui-ci. Par ailleurs, nous avions à ce moment-là déjà récupérer également le code gtk correspondant à l'ouverture de la fenêtre de choix (ci-dessous) permettant de se balader dans les fichier et d'en créer un nouveau, il ne nous manquait plus que la partie de la fonction permettant de récupérer la longueur du son.



Après de longs moments de recherche et d'incompréhension, nous avons enfin réussi à trouver la solution. Afin de pouvoir récupérer absolument toutes les données du son, nous avions besoin d'un moment continu où nous pourrions écrire à répétition dans le fichier sortie et ainsi sauvegarder le son. Cette analyse du son pouvait avoir lieu pendant que le son était en train d'être joué mais des problèmes auraient pu survenir si une pause ou un recommencement est effectué. Cette analyse a donc finalement lieu pendant que le son est enregistré à l'aide du microphone. En effet toutes les dix secondes, le son va être verrouillé avec la fonction de la librairie FMOD FMOD_Sound_Lock(), les informations nécessaires à la sauvegarde vont être récupérées. De plus une nouvelle variable présente dans la structure MusStrut et correspondant à la longueur du son va être incrémentée. Puis on va déverrouiller le son avec la fonction FMOD_Sound_UnLock(), on va également mettre le système à jour avec FMOD_System_Update() et on va répéter ces étapes tant que l'utilisateur est en train d'enregistrer un son.

Une fois cette méthode comprise un autre problème se présentait à nous, la boucle. En effet ces étapes étaient nécessaires tant que l'utilisateur était en train d'enregistrer un son. Nous avions donc dans un premier temps chercher une fonction dans la librairie FMOD qui pourrait indiquer si le record avait lieu ou pas. Il en existe bien une appelée FMOD_System_IsRecording() mais après plusieurs tests nous avions remarqué qu'elle ne s'effectuait pas tout le temps. Nous avions donc cherché à implémenter un booléen dans la structure MusStruct qui serait mis à jour lorsque l'on commence et finit d'enregistrer un son.

Ce nouveau système était alors fonctionnel dans le sens où il permettait bien de savoir si on était en train d'enregistrer ou pas mais un autre problème se posait alors : une boucle dans gtk. En effet gtk_main étant à lui-même une boucle infinie avec un système assez particulier, c'était impossible de faire tourner une boucle à l'intérieur. Nous avons donc enfin eu l'idée d'une autre méthode qui sera, je vous rassure, la dernière énoncée dans ce récit pour ce point. Nous avons décider d'utiliser la fonction g_timeout_add_seconds() qui permet d'ajouter un évènement qui répètera à l'infini la fonction mise en paramètre selon le temps donné en paramètre. Ainsi lorsque l'utilisateur débutera son enregistrement cette évènement, composé des étapes citées ci-dessus, commencera et se répètera toutes les dix secondes. De plus cet évènement a un certain identifiant qui est récupérer au début dans une variable de la structure MusStruct et qui sera utilisé, lorsque l'utilisateur arrêtera d'enregistrer le son, dans la fonction g_source_remove().

# 6 Le site web

Le site web est proche de sa version finale. Il manque uniquement un bouton permettant de télécharger de l'application. Nous voulions un site à l'image de notre application: sobre, épuré, intuitif et élégant.
La page a été codé en "HTML5", les animation et autres visuels, quant à eux, ont été réalisés en CSS.
Il a pour but de présenter l'application et l'équipe en quelque mots.

# 7 Perspectives et objectifs d'avenir

Nous entrons dans la phase finale de la réalisation de notre projet. C'est lors de cette dernière que nous allons concrétiser tout le travail accompli jusqu'à présent. Nous allons donc ajouter tout type de modificateurs de son et continuer d affiner l'interface graphique.

# 8 Conclusion

Le groupe avance bien et la cohésion d'équipe est toujours aussi forte. Ce projet permet à chacun d'entre nous de développer de nouvelles compétences. Maintenant que nous entrons dans la phase finale de réalisation/conception, le groupe peut laisser libre court à sa créativité. Nous entrons dans la partie la plus agréable du projet. Dans cette dernière, nous allons apporter les idées que nous avions et la satisfaction du travail accompli nous accompagnera lors de cette dernière ligne droite.