

# Make Sheet

---

## Projet MPA

---

**Etudiants :**  
Sorin Alexia  
Couedor Léo

**Enseignant référent :**  
Guillou Goulven

Introduction .....	3
1. Gestion de projet.....	4
2. Réalisation .....	5
2.1 Présentation générale.....	5
2.1.1 Frontend .....	5
2.2 Page Accueil .....	6
2.3 Page Accordeur .....	7
2.3.1 Frontend .....	7
2.3.2 Backend.....	7
2.4 Page Génération de partition .....	10
2.4.1 Frontend .....	10
2.4.2 Backend.....	11
2.5 Page Paramètres.....	13
2.5.1 Frontend .....	13
2.5.2 Backend.....	13
Conclusion .....	14
Annexes.....	15
Annexe 1 : Maquettes.....	15
Maquette page d'accueil.....	15
Maquette page de l'accordeur .....	15
Maquette page de génération de partition .....	16
Maquette page de paramètres.....	16
Annexe 2 : Réalisations.....	17
Réalisation page d'accueil .....	17
Réalisation page de l'accordeur.....	17
Réalisation page de génération de partition.....	18
Réalisation erreur page de génération de partition .....	18
Réalisation page de paramètres .....	19
Réalisation sélection des emplacements par défaut .....	19

# Introduction

Dans le cadre du projet MPA (mise en pratique accompagnée) de notre 3ème année de licence d'ingénierie informatique, nous avons eu pour objectif de réaliser une application comprenant un accordeur chromatique ainsi qu'un générateur de partition. Les limites et spécificités du projet étant relativement imprécises, c'est un projet durant lequel nous avons eu à faire par nous-même de nombreux choix, à la fois techniques et esthétiques.

Nous avons été accompagnés lors de ce projet par M. Guilloux, que nous avons pu solliciter afin de valider nos choix, principalement sur le plan esthétique, mais aussi sur le choix du nom de notre application : MakeSheet, issu du mélange entre make (fabriquer en anglais), et score sheet (partition en anglais).

Afin d'obtenir l'image la plus précise possible du travail à réaliser, nous avons commencé par rechercher les solutions déjà existantes. Les accordeurs chromatiques étant très nombreux sur internet, il nous a été facile de définir un fonctionnement et une esthétique qui nous satisfaisait nous ainsi que notre encadrant. En ce qui concerne le générateur de partition en revanche, la quantité de solutions est bien plus limitée. Aussi, ces solutions étant toutes payantes, et majoritairement limitées à certains instruments, nous n'avons pas trouvé d'application déjà existante convenable pour la transcription de musiques jouées à la clarinette, qui était notre instrument cible. Enfin, des maigres informations que nous avons pu obtenir sur le fonctionnement de ces divers sites et applications, nous nous sommes aperçus que l'utilisation de l'intelligence artificielle était un élément central de leur fonctionnement. Mais n'étant pas formé à son utilisation, cela nous a tout d'abord posé certaines inquiétudes concernant notre capacité à mener le projet à terme.

# 1. Gestion de projet

De par sa nature “expérimentale”, et à cause du fait que l’ensemble des points à traiter n’ai pas pu être défini dès le début du projet, nous avons décidé de ne pas définir de plan précis quant aux échéances à respecter, et ainsi d’adopter une méthodologie plus agile. Cela nous a permis d’être plus variable vis-à-vis des imprévus, ainsi que de la quantité de travail apportée qui n’a pas pu être linéaire en fonction de nos différentes obligations relatives aux autres matières de notre formation.

En ce qui concerne la répartition du travail au sein du binôme, nous avons décidé de séparer le travail entre la partie backend et la partie frontend. L’application devant comporter deux fonctionnalités, nous aurions également pu opter pour une répartition par fonctionnalité, mais nous avons fait ce choix pour optimiser la qualité de travail en fonction de nos connaissances et compétences respectives. En effet, puisque Léo possédait déjà les connaissances théoriques, particulièrement sur l’aspect des partitions, qui peut être complexe pour les non-initiés, nous avons décidé qu’il s’occuperait de la partie backend, pour nous concentrer sur la réalisation plutôt que sur l’apprentissage du solfège (Bien qu’Alexia en ai également appris certains points fondamentaux). A l’inverse, Alexia, relativement à l’aise à l’idée de concevoir et réaliser des interfaces graphiques, s’est occupée de la partie frontend de notre programme. Chacun a donc pu travailler indépendamment sur sa partie respective, mais nous avons tout de même travaillé conjointement, particulièrement pour valider nos différents choix.

Pour nos communications, ayant uniquement travaillé en distanciel sur ce projet, nous avons très largement utilisé Discord pour réaliser des appels vocaux et nous tenir au courant de l’avancée du projet. En ce qui concerne le partage de code, nous avons utilisé l’outil de versionnage de code git. Nos parties pouvant être très indépendantes, nous avons eu relativement peu de conflits, et ceux-ci ont pu être résolus très simplement.

## 2. Réalisation

A la fois pour l'accordeur comme pour le générateur de partition, nous savions que le traitement de signaux, plus particulièrement de signaux audio, serait l'aspect central de notre projet. Pour cela, nous avons décidé de réaliser notre application avec le langage Python, très fourni en libraires, nous permettant d'utiliser diverses fonctions sans avoir à les réaliser nous-même, et ainsi faire un gain de temps considérable. De plus, ce langage étant relativement simple, proposant également des solutions pour la réalisation d'interfaces graphiques, bien que n'ayant précédemment jamais réalisé de projet avec ce langage, nous savions qu'il serait à coup sûr le meilleur choix pour nous donner les meilleures chances de mener notre projet à bien.

Cette partie, dédiée à la réalisation du projet, permettra d'aborder dans un premier temps de manière générale, puis page par page, la présentation du travail réalisé pour la partie backend mais également frontend, avec également quelques éléments d'explication du code produit ainsi que son organisation.

### 2.1 Présentation générale

#### 2.1.1 Frontend

L'interface de l'application MakeSheet peut être décomposée en quatre pages distinctes. Afin de réaliser l'interface, nous avons d'abord commencé par la création des liens entre les différentes pages avant d'ajouter les éléments permettant l'interaction avec le backend. Suite à cela, nous avons conçu une maquette pour chacune des pages, maquettes que vous pourrez trouver en annexe. La dernière étape a été le codage de l'IHM, en veillant à respecter au mieux les maquettes produites.

Nous avons opté pour une charte graphique composée de couleurs claires, avec du gris clair pour la couleur de fond des pages, et du blanc pour la barre de navigation. Certaines touches de doré sont également présentes, à l'image de notre logo. Le choix de ces couleurs garantit une bonne lisibilité pour les utilisateurs, et donne un côté sophistiqué et moderne à l'application.

La barre de navigation est commune à toutes les pages de l'application. Celle-ci est composée tout à gauche du logo puis plus à droite des quatre boutons représentés par une icône et le nom de la page. L'utilisateur peut également aisément repérer la page sur laquelle il se situe grâce à une mise en évidence du nom dans la barre de navigation, qui passe alors en couleur dorée.

L'ensemble de l'interface graphique a été réalisé avec la librairie Tkinter de python. Bien qu'il existe d'autres librairies permettant de concevoir des interfaces graphiques, elle reste la plus connue, la plus simple d'utilisation et surtout la mieux documentée. Bien qu'ayant déjà réalisé des IHM dans d'autres langages de programmation, nous n'en avons jamais fait en python. De par la structure légèrement différente de ce que nous connaissions pour la création d'interfaces graphiques, cela constituait pour nous un grand challenge pour obtenir un résultat convaincant, et a donc nécessité un certain apprentissage dans les premières heures du projet.

En ce qui concerne l'organisation du code de la partie frontend, nous avons choisi de le diviser en cinq classes différentes, avec en plus le fichier index qui permet de lancer l'application. Dans ce fichier d'index, nous créons dans un premier temps la fenêtre principale, fenêtre à laquelle nous attribuons diverses caractéristiques. Nous attribuons ensuite cinq frames (conteneurs), à savoir une pour la barre de navigation et une pour chaque page de l'application.

Les boutons présents sur la barre de navigation permettent de lancer une fonction qui permet le changement de page courante. Tkinter propose des barres de navigation pré-faites, mais qui soit n'acceptent pas les images, soit ne correspondent pas à nos attentes esthétiques, nous avons donc décidé de la réaliser nous-mêmes.

Les quatre autres frames sont utilisées pour lancer les cinq différentes classes correspondant à chaque page. Au lancement des différentes classes, une initialisation est effectuée dans laquelle on crée différentes frames, labels, images qu'on ajoute en les plaçant comme souhaité au frame concerné par la classe. Les frames sont ajoutées à la fenêtre uniquement lors du clic sur le bouton de la barre de navigation correspondant. La fonction appelée par le bouton permet l'ajout de cette frame à la fenêtre et la suppression des autres. Au lancement de l'application, la page d'accueil est la première page visible, c'est donc cette frame qui est ajoutée à la fenêtre lors de l'initialisation.

## 2.2 Page Accueil

La page d'accueil, relativement simple, est divisée en deux colonnes. Celle de gauche accueille quatre paragraphes placés les uns en dessous des autres, le premier expliquant l'utilité générale de l'application, et les suivants présentant l'ensemble des pages. La colonne de droite, bien plus légère, accueille le logo de l'application.

Comme l'atteste le rendu final présent dans l'annexe, la réalisation correspond parfaitement à la maquette initialement produite.

## 2.3 Page Accordeur

### 2.3.1 Frontend

La page de l'accordeur est composée du compteur, de sa légende, ainsi que d'un bouton pour lancer et arrêter l'écoute. Sur cette page, nous retrouvons davantage de couleurs. En effet, l'accordeur est divisé en trois zones représentées par leurs couleurs. La couleur verte indique la zone correspondant à l'intervalle de la note détectée, la couleur jaune à l'intervalle de la note suivante et le orange à l'intervalle de la note précédente. De plus, il est composé d'un trait violet indiquant la note parfaitement accordée et d'une aiguille rouge qui correspond à la fréquence trouvée. Plus l'aiguille se rapproche du segment violet, plus la fréquence trouvée est proche de la fréquence de la note juste.

Au départ nous avons réfléchi à plusieurs idées esthétiques pour l'accordeur, comme par exemple une barre de progression. Mais notre choix s'est vite porté sur le compteur, qui est beaucoup plus visuel et esthétique.

Sur la maquette nous pouvons constater l'existence de deux boutons : un pour arrêter et un pour lancer l'écoute. Mais lors de la réalisation, nous avons décidé de changer cela afin d'obtenir un seul et unique bouton pour que l'interface soit la plus ergonomique possible. Une deuxième modification qui a été faite entre la maquette et la réalisation concerne l'emplacement de la légende. En effet, celle-ci était sur la maquette située à droite de l'accordeur, mais afin d'optimiser l'occupation de l'espace, nous avons finalement décidé de la placer en dessous de l'accordeur.

Lors de l'initialisation de l'index, le thread de la classe de l'accordeur est lancé. Celui-ci reste en pause tant que l'utilisateur n'a pas appuyé sur le bouton lancer. Le thread permet l'accès aux différentes fonctions du backend, tout en modifiant en temps réel les éléments de l'accordeur en fonction des retours de ces fonctions. Le compteur de l'accordeur est créé dans un canvas. Un canvas est une zone rectangulaire de dessin et de figures avec Tkinter.

Ce canvas est composé d'un demi-cercle qui sert de base, de 3 arcs de cercle qui forment les zones de couleurs, et d'un segment violet. Pour créer ou modifier les arcs dans un canvas, nous utilisons la fonction "create\_arc", pour les segments, la fonction "create\_line", et enfin pour les labels, "create\_text". A l'image du compteur, la légende fait partie intégrante du canvas, et est composée de texte mais aussi de rectangles créés avec "create\_rectangle".

### 2.3.2 Backend

Le fichier nommé "back.py" contient l'ensemble des fonctions que nous avons définies permettant le fonctionnement à la fois de l'accordeur, mais également du générateur de partition, qui possèdent d'ailleurs des fonctions communes dans leur utilisation.

Tout d'abord, nous retrouvons deux fonctions. La première permettant de générer un tableau de type dictionnaire associant un nom de note (les nom de l'ensemble des notes étant définies dans un tableau séparé) avec son octave et sa fréquence. Le passage d'une fréquence à une autre se faisant par un multiplicateur constant, il est donc aisé de construire les fréquences d'une octave donnée. La seconde fonction définie permet de construire un tableau contenant les fréquences des notes sur autant d'octaves que souhaité, avec de multiples appels à la fonction présentée précédemment. Enfin, la fonction principale au fonctionnement de l'accordeur, qui pour une fréquence donnée va parcourir le tableau défini plus tôt pour trouver la fréquence la plus proche, qu'elle soit inférieure ou supérieure à la valeur de fréquence donnée.

Une fois cette fréquence trouvée, la fonction retourne le nom de la note correspondante ainsi que son octave, mais également les notes précédentes et suivantes, pour l'affichage dans l'interface graphique, ainsi que l'écart entre la fréquence de la note jouée, et la fréquence de cette même note parfaitement accordée, permettant ainsi d'accorder son instrument le plus précisément possible.

A noter également qu'un contrôle a été appliqué dans le cas où la fréquence donnée serait très inférieure ou supérieure à la première ou la dernière des valeurs possibles dans le tableau des fréquences. Toujours en utilisant le multiplicateur constant, dans ce cas, la note n'est attribuée à la première ou la dernière note du tableau que dans le cas où elle se situe à moins de la moitié de la différence de fréquence avec la note précédente ou suivante, bien que ces notes ne soient pas dans le tableau. Par exemple, la fréquence de la première note de notre tableau étant de 32.7Hz, le Do0, et la note précédente (n'étant pas dans le tableau) ayant une fréquence de 30.87Hz, le Si-1, si la fréquence donnée est de 32.0Hz, la fréquence sera attribuée au Do0. A l'inverse, si la fréquence donnée est de 31.0Hz, étant plus proche du Si-1 n'étant pas dans notre tableau, aucune note ne lui sera attribuée.

Notre accordeur ayant pour but de donner en temps réel la note jouée, il nous a ensuite fallu trouver le moyen de calculer en temps réel la fréquence jouée, afin de la fournir à la fonction présentée plus haut, et ainsi obtenir la précision de cette note. Nous avons donc testé plusieurs solutions sur les librairies à utiliser pour réaliser cela, avant d'obtenir un résultat fonctionnel.

Grâce à la librairie "pyAudio", notre programme commence donc par ouvrir le microphone par défaut de l'ordinateur, et range les données issues de l'enregistrement dans un tableau par blocs, qui permet ensuite la création d'un fichier externe au format wav qui est utilisé par la suite pour l'analyse du signal. La taille d'un bloc est définie selon un paramètre du microphone, le nombre de samples par morceau, dépendant d'un autre paramètre, le nombre de samples par seconde. Nous avons défini ce nombre de samples par morceau selon le théorème de Shannon, indiquant que pour qu'un signal puisse être reconstruit, il est nécessaire que l'échantillon soit deux fois plus grand que la plus grande fréquence à repérer. L'instrument cible de notre application, à savoir la clarinette accordée en Si bémol, ayant une fréquence maximale d'environ 2000 Hz, nous avons fixé la taille d'un morceau à un peu plus de deux fois cette valeur.



Une fois le morceau à analyser enregistré dans le fichier, nous utilisons ensuite la librairie “wave” pour ouvrir ce fichier, puis la librairie “soundfile” afin d’extraire les informations de signal de ce fichier. Enfin, une fois en possession de ces données, nous pouvons appliquer l’algorithme de FFT (Fast Fourier Transform), permettant de transformer les données issues d’un signal du domaine temporel au domaine fréquentiel, nous permettant ainsi d’obtenir la fréquence dominante d’un fichier donné, avant de l’envoyer à notre fonction de recherche de note.

Après divers tests de cette fonctionnalité, nous nous sommes cependant aperçus de deux problèmes. Le premier, concerne la vitesse de traitement. En effet, bien que le traitement soit relativement rapide, il ne l’est cependant pas suffisamment pour trouver plusieurs notes dans un laps de temps très limité. Nous avons pour cela essayé de faire le traitement du signal sans passer par un fichier externe, mais sans succès. Le second problème concerne la précision. Nous nous sommes en effet aperçu que la note détectée se trouvait de manière récurrente, voire parfois systématiquement, être cinq demi-tons au-dessus de la note réelle. Nous avons pour cela essayé diverses solutions, qui seront développées dans la partie consacrée au générateur de partition, avant d’en venir à la conclusion que cela devait en très grande partie venir du matériel utilisé. En effet, les tests ayant été produits avec le micro d’un ordinateur portable et le haut-parleur d’un téléphone faute de meilleur matériel, la fidélité de la note jouée, puis enregistrée, s’en est obligatoirement vu dégradée.

## 2.4 Page Génération de partition

### 2.4.1 Frontend

La classe de partition est composée d'une partie formulaire à gauche et d'une partie chronomètre à droite, qui est elle-même une classe à part entière. La partie formulaire est composée de cinq labels, quatre zones de saisie et de deux checkBox. Une fois toutes les informations recueillies, nous pouvons cliquer sur le bouton "lancer" qui va permettre de commencer l'enregistrement. Il y a également un label d'erreur, qui est initialement invisible, mais apparaît dans le cas où un champ n'est pas rempli ou s'il est invalide, indiquant alors la nature de l'erreur. Le chronomètre est composé d'un label qui affiche le temps écoulé, et d'une image de chronomètre. Celui-ci se lance dès que l'utilisateur lance l'enregistrement si aucune erreur n'est détectée.

La réalisation ressemble en grande partie à la maquette, à l'exception de l'ajout de nouveaux renseignements nécessaires à la création des partitions. Une légère différence est également présente pour le chronomètre, puisque l'image est maintenant placée à gauche du label au lieu d'être à droite comme initialement prévue sur la maquette, et ce pour des raisons purement esthétiques.

La structure du code est ici semblable à celle de l'accordeur présenté plus haut, avec cependant l'ajout d'un chronomètre et d'une gestion des erreurs, éléments non présents sur la page de l'accordeur.

La gestion des erreurs s'effectue dans la fonction appelée par le bouton "lancer". Nous vérifions que chaque zone de saisie soit remplie, et respecte les critères établis. Si elle ne respecte pas les critères, le label d'erreur est modifié en indiquant l'erreur précise. De plus, un premier contrôle effectué au moment de l'écriture dans certaines zones de texte pour lesquelles les valeurs sont limitées (Ce qui est le cas par exemple pour le champ, tempo dans lequel il est impossible d'écrire autre chose que des chiffres).

Le chronomètre, comme dit précédemment, est une classe à part entière. Celle-ci est composée d'une image, d'un label et d'une variable de type StringVar. Nous attribuons ensuite au label la valeur du StringVar, ce qui permet de modifier en temps réel l'affichage durant l'exécution de la méthode de calcul du temps.

## 2.4.2 Backend

Au tout début du projet, cette fonctionnalité était de loin celle qui nous posait le plus d'inquiétudes, n'ayant alors que peu de pistes quant à la manière de procéder. Notre première idée était d'utiliser une image de partition vierge, sur laquelle nous serions venus apposer les différentes notes entendues, dont leur type (ronde, croche, blanche,...) aurait été déterminée grâce à une approximation obtenue de sa durée. Cette solution nous semblait cependant extrêmement complexe à mettre en place, particulièrement concernant la hauteur des différentes notes, qui aurait alors dû être gérées par un système de coordonnées. Après de nombreuses recherches, nous avons finalement eu une idée différente, bien plus simple à mettre en place.

Lors de l'enregistrement, les différentes notes, d'abord obtenues à l'image de l'accordeur présenté plus tôt, puis converties selon leur code MIDI grâce à une fonction, sont enregistrées dans un tableau. Une fois l'enregistrement terminé, ce tableau est passé en paramètre d'une autre fonction ayant pour but d'effectuer une série de traitement sur ce tableau. Tout d'abord, deux sous-tableaux sont créés, le premier permettant de stocker les notes jouées, le second comptant pour chaque note son nombre d'occurrences consécutives. Ensuite, ce second tableau est parcouru pour repérer les notes n'ayant qu'une seule occurrence, c'est-à-dire n'étant présente que sur un seul morceau. Dans ce cas, cette note est qualifiée de "note parasite", puisque n'ayant pas une présence suffisante, et est ainsi retirée des tableaux. Pour les cas où une note parasite serait venue couper une série de note, un nouveau passage est effectué pour à nouveau comptabiliser le nombre d'occurrences consécutives des différentes notes, donnant ainsi les tableaux finaux.

Une fois le traitement de ces tableaux finalisé, un fichier MIDI est ensuite créé grâce à la librairie "Midiutile". À ce fichier MIDI nous ajoutons ensuite le nom, le tempo, ainsi que le numérateur et le dénominateur de la partition, informations obtenues à partir du formulaire.

Ce fichier MIDI est ensuite rempli par le parcours des tableaux des notes présenté plus haut. Les notes sont donc ajoutées une à la fois, avec leur durée calculée en fonction du nombre d'occurrences consécutives, et du tempo. Le fichier est ensuite sauvegardé dans le répertoire courant d'exécution de l'application.

L'application permet de choisir le type de fichier en sortie de l'enregistrement. Ainsi, si l'option de génération de partition a été cochée, ce fichier MIDI est ensuite utilisé, en étant converti d'abord au format XML, puis au format d'une partition en PDF, le tout grâce à la librairie "Music21", faisant appel à MuseScore pour effectuer ce traitement. Ce fichier PDF, est ensuite déplacé dans le répertoire spécifié dans la page de paramètres. Si l'option d'obtenir le fichier MIDI est cochée, ce fichier est également déplacé dans le répertoire correspondant. En revanche, si cette option n'est pas cochée, le fichier est supprimé immédiatement après son utilisation pour la conversion de la partition.

L'énorme avantage de passer par une conversion de fichier MIDI vers une partition, à l'inverse notre idée d'origine de placer les notes graphiquement est bien évidemment la qualité graphique de la partition produite. De cette manière, nous sommes sûr que les notes détectées sont placées au bon endroit, et cela nous libère d'un certain nombre de traitement, désormais géré automatiquement. Mais cette solution n'est cependant pas parfaite, et possède également ses inconvénients. Premièrement, certaines informations ne sont pas, ou pas facilement transcriptibles au format MIDI, tel que l'expressivité de certains passages par exemple (si certains passages doivent être joués plus fort, moins fort,...). Alors qu'il s'agit d'un point qui aurait éventuellement pu être traité plus simplement avec notre solution d'origine.

Ensuite, un second exemple pouvant être donné concerne l'édition des partitions. En effet, il aurait été plus facile d'intégrer directement à l'application des solutions d'édition de la partition générée avant son export au format PDF. Notre application proposant l'export de l'enregistrement au format MIDI, il est ensuite possible d'ouvrir ce fichier dans d'autres applications tel que MuseScore par exemple, pour ensuite faire une nouvelle conversion en partition, mais il sera obligatoire de passer par une application externe.

Enfin, comme évoqué dans la partie dédiée à l'accordeur, la fiabilité des partitions générées avec notre application est limitée. Pour tenter de résoudre le premier problème de temps de traitement légèrement trop long pour capter les notes plus courtes, nous avons tenté d'optimiser au mieux notre code, de jouer sur les différents paramètres à notre disposition, ou encore de faire usage de threads pour différencier l'enregistrement du traitement des notes, mais aucune solution n'a abouti à une solution réellement efficace. Pour le second problème de hauteur de notes, après beaucoup de réflexion, nous avons émis l'hypothèse que dans certains cas la note captée pourrait en réalité être une harmonique de la note fondamentale, les notes issues d'une clarinette n'étant pas purs. Nous avons pour cela tenté d'effectuer un traitement sur les harmoniques les plus présents dans l'enregistrement d'un morceau, mais cette hypothèse s'est malheureusement avérée fausse. En l'absence d'autres hypothèses, nous estimons donc que d'autres tests avec du matériel de qualité, permettrait très certainement de résoudre ce problème.

## 2.5 Page Paramètres

### 2.5.1 Frontend

La fenêtre de paramètres n'est composée que de deux boutons et de leurs labels associés. Les labels indiquent le chemin vers le dossier d'enregistrement des fichiers midi et PDF, ainsi que vers l'exécutable de MuseScore, nécessaire à la création des partitions. Les boutons permettent l'ouverture d'un explorateur de fichier afin de sélectionner un nouvel emplacement.

### 2.5.2 Backend

Lors de son lancement, l'application commence par exécuter une fonction permettant d'initialiser les emplacements par défaut. Dans le répertoire de lancement de l'application, un dossier nommé "serial" est créé s'il n'existe pas déjà. A l'intérieur de ce dossier, deux nouveaux fichiers sont créés, permettant de conserver les chemins d'enregistrement des fichiers ou de l'exécutable de MuseScore. Lors du tout premier lancement de l'application, ces fichiers sont initialisés au dossier "téléchargements" de l'utilisateur pour l'enregistrement des partitions et fichiers MIDI, et au dossier par défaut lors de l'installation de MuseScore pour l'exécutable de ce dernier. Lorsque l'utilisateur sélectionne un nouveau dossier par le biais de l'explorateur de fichier évoqué plus haut, le contenu de ces deux fichiers est mis à jour. Si, lors du lancement de l'application, ces deux fichiers existent déjà, ils ne sont pas modifiés, ce qui permet de garder en mémoire les choix de l'utilisateur. Le contenu de ces fichiers est ensuite utilisé aux différents endroits ayant besoin de ces informations.

# Conclusion

Bien que notre application finale possède trop de lacunes en ce qui concerne sa fiabilité pour l'envisager en utilisation réelle, nous pensons cependant qu'elle pourrait être utilisée en outil d'appoint, afin de repérer sommairement les intervalles majeurs d'une musique, c'est-à-dire la taille des écarts entre les notes, mais également pour éventuellement repérer la gamme utilisée dans un morceau.

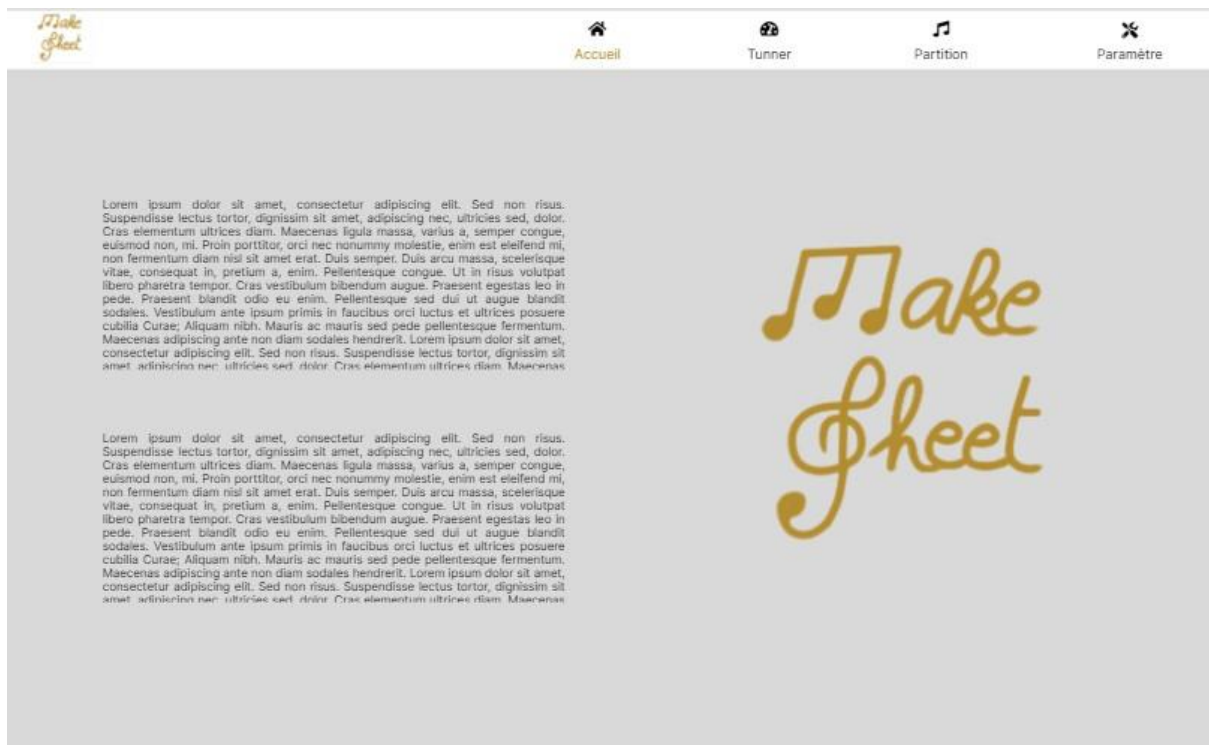
Le résultat n'est malheureusement pas à la hauteur de nos attentes, mais de manière assez objective, il semblait assez difficile d'obtenir un résultat suffisamment bon pour une utilisation réelle sans faire appel à l'intelligence artificielle. Cependant, considérant que nous n'y sommes pas du tout formé, nous n'aurions jamais eu le temps de produire un projet à un tel niveau d'avancement dans le temps imparti, bien qu'avec plus de temps, si nous avions choisi la voie de l'intelligence artificielle, nous aurions probablement abouti à un résultat d'autant plus convaincant.

Ce projet reste néanmoins un projet particulièrement intéressant, à la fois de par son sujet, mais également à la vue du travail réalisé et des compétences apportées, particulièrement concernant le traitement de signaux audio et la construction d'interfaces graphiques sous Python, deux aspects sur lesquels nous n'avions jamais travaillé avant le début de ce projet.

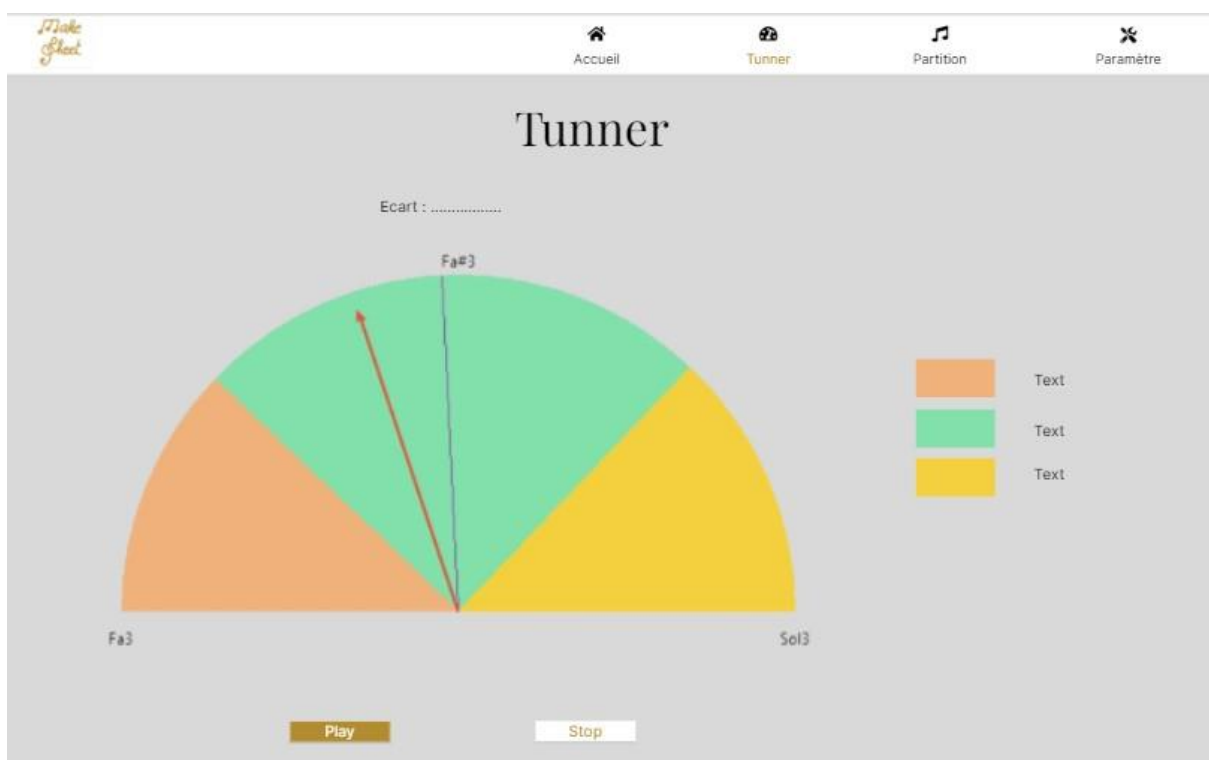
# Annexes

## Annexe 1 : Maquettes


### Maquette page d'accueil





### Maquette page de l'accordeur





## Maquette page de génération de partition



 Accueil

 Tunner


 Partition

 Paramètre

# Partition

Titre :


Tempo :


00:00 


Play


Stop


## Maquette page de paramètres



 Accueil

 Tunner

 Partition

 Paramètres

# Paramètres

Chemin du fichier : C:\Users\asorin\Documents

Selection...

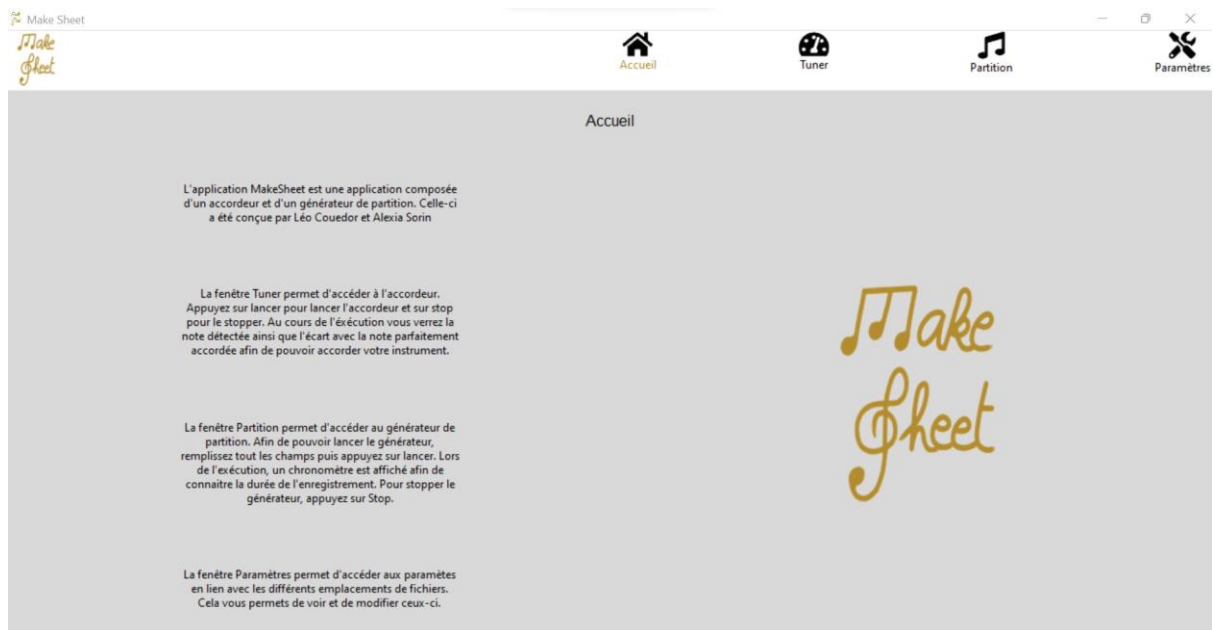
Chemin du MuseScore : C:\Users\asorin\Documents\MuseScore3

Selection...



## Annexe 2 : Réalisations

### Réalisation page d'accueil



### Réalisation page de l'accordeur



## Réalisation page de génération de partition

Make Sheet

Accueil Tuner Partition Paramètres

Partition

Titre

Tempo

Numérateur

Dénominateur

Le ou les types de fichier(s) : ☒ MIDI ☒ PARTITION

00:07

Stop

## Réalisation erreur page de génération de partition

Make Sheet

Accueil Tuner Partition Paramètres

Partition

Titre

Tempo

Numérateur

Dénominateur

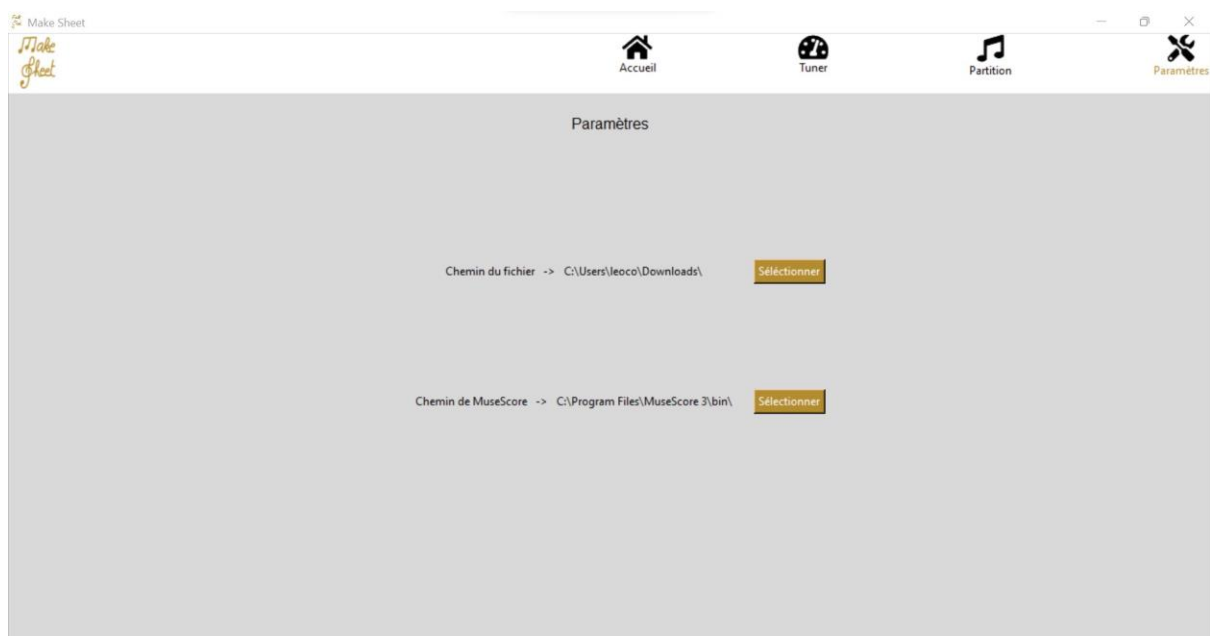
Le ou les types de fichier(s) : ☐ MIDI ☐ PARTITION

Erreur : le tempo est manquant

00:00

Lancer

## Réalisation page de paramètres



## Réalisation sélection des emplacements par défaut

