MusicBot - Rapport final

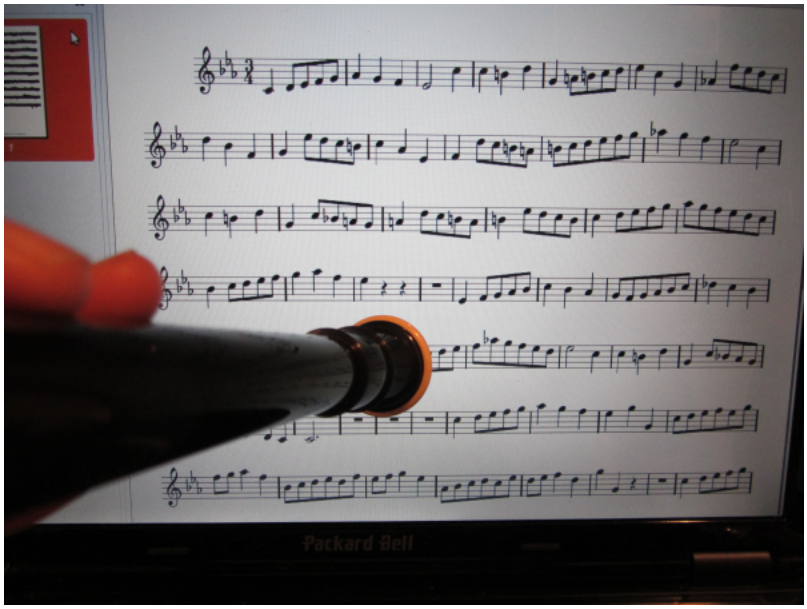
Alexandre Talon Alice Pellet-Mary Antoine Grospellier
Baptiste Rozière Benjamin Farinier Diego Nava Saucedo
François Pirot Pierre Macherel Louis Gal Valentin Gledel

20 janvier 2014

MusicBot

ENS DE LYON

Table des matières



1 Introduction

Pour l'instant l'utilisation des outils informatiques ou des supports numériques dans le monde de la musique est assez réduite. Si des recherches existent notamment chez les compositeurs contemporains sur les relations possibles entre les musiciens et la machine en terme de création et de dialogue, la plupart des instrumentistes utilisent des partitions papier et travaillent leurs partitions sans support électronique.

Or, ces outils et supports pourraient se révéler très utiles et quelques rares initiatives existent comme le montre cet article :
`http://actualites.ch.msn.com/musique-un-orchestre-belge-passe-des-partitions-aux-tablettes`.
Le premier but recherché est la dématérialisation des partitions, ce qui, à l'instar des liseuses numériques dans le monde du livre, permettrait de faciliter et le transport, et l'accès à une très large bibliothèque musicale pour le musicien. Cette dématérialisation passant par une numérisation, il est par la suite possible d'exploiter les partitions numériques obtenues directement via un outil informatique, dont on pourra tirer de nombreux outils, notamment dans un soucis de remodelage de la partition. Une première démarche est alors de proposer au musicien un outil d'affichage intelligent de la partition, qui lui permette de jouer un morceau sans avoir à se soucier du défilement de la partition, tâche toujours périlleuse dans un contexte de concert. Cela passe par l'écoute et le traitement informatique de l'interprétation du musicien, dont il est au passage possible de renvoyer un compte-rendu en fin de morceau, afin d'aider à orienter le travail du musicien. Nous espérons de la sorte réunir des utilisateurs de tous niveaux musicaux, depuis les débutants jusqu'aux musiciens confirmés. Pour convenir au public visé, notre programme se veut le plus complet et simple d'utilisation possible. Il est également important qu'il se retrouve sur un nombre de plates-formes conséquent.

MusicBot se présente sous la forme d'un assistant musical, dont le but est d'aider les musiciens de tous niveaux dans la phase de déchiffrage puis de travail d'un morceau. Il fourni donc des outils basiques permettant de faciliter la découverte et le travail de partitions, tels qu'une gestion intelligente du défilement de la partition, ou encore un outil permettant la reconnaissance automatique de scans combiné avec un éditeur permettant de corriger les erreurs du logiciel.

En outre, il propose une analyse critique de l'interprétation du musicien si ce dernier fournit une partition et renvoi la partition correspondant au morceau joué dans le cas contraire. Cela permettra aux débutants de chiffrer leur progression et sera très utile aux compositeurs.

Ce logiciel n'existe pour l'instant que sur ordinateur, mais une application androïd devrait être développée à terme. Ainsi, il pourra aisément suivre les musiciens lors de leurs déplacements. Son but est d'agglomérer des options disponibles dans d'autres logiciels déjà existants, mais souvent payants ou incomplets.

MusicBot accepte l'importation de partitions scannées, ce qui lui permet d'avoir accès à une base de données libre sans limite, notamment par le biais de sites de partitions libres de droits tels que **IMSLP**. Ceci le différencie de son principal concurrent **Weezic**. L'objectif est de créer une communauté active autour de cette application qui permettrait de l'enrichir après sa création.

2 Membres du projet

Le projet réunit les membres suivants :
- **François Pirot** : Chef de projet
- Alexandre Talon
- Alice Pellet-Mary
- Antoine Grospellier
- Baptiste Rozière
- Benjamin Farinier

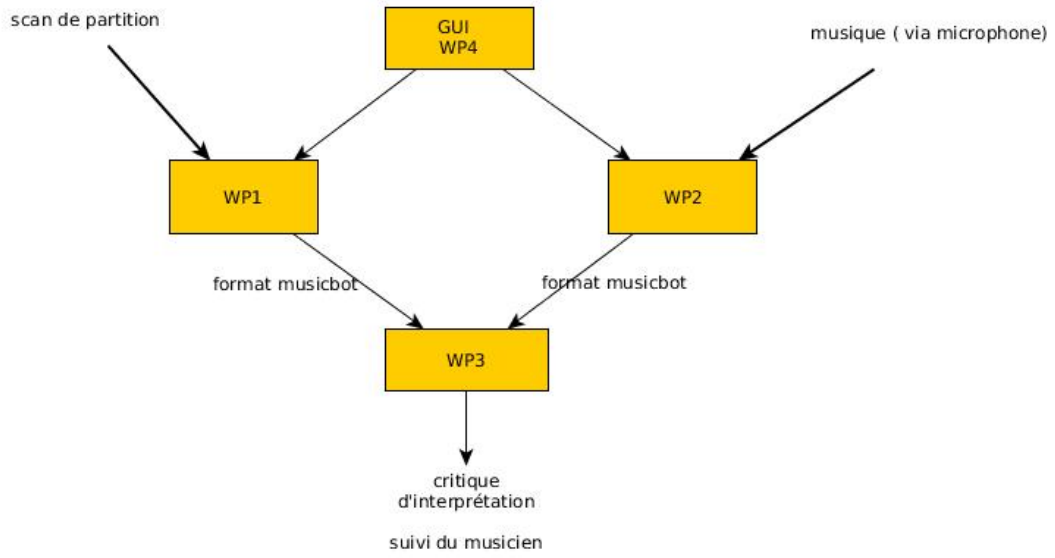


FIGURE 1 – Organisation du projet

- Diego Nava Saucedo
- Pierre Macherel
- Louis Gal
- Valentin Gledel

3 Répartition des tâches

Le projet se décompose en quatre modules principaux (work packages abrégés WP), en plus du work package communication. Le nom de chaque chef de work package est en gras.
Le WP1 est chargé de convertir une partition scannée en une suite de notes au format musicbot (qui consiste à écrire les notes et leur durée les unes après les autres). Le WP2, lui, convertit un flux audio en flux de notes au même format que précédemment. Et le WP3 assemble ces deux flux de notes pour essayer de suivre le musicien dans la partition.

3.1 WP0 : Communication

Membres du WP0 : Baptiste Rozière, François Pirot et Louis Gal
Le but de ce work package est de fournir des informations sur le projet. Il s'agit de documentation et de le faire connaître à son public potentiel. Ceci passe notamment par la création du site web du projet (http ://graal.ens-lyon.fr/musicbot/) et d'une page facebook. Il s'est aussi chargé de contacter des musiciens afin de mieux cibler leurs attentes.

3.2 WP1 : Numérisation de partition

Membres du WP1 : Diego Nava Saucedo, François Pirot, **Pierre Macherel** et Valentin Gledel

L'objectif de ce work package est de passer d'une partition sous un format d'image à une partition exploitable par notre logiciel, dans un format numérique adapté. Il se charge aussi du stockage des partitions.

Tâches

- Première tâche : Numériser la partition.

- Deuxième tâche : Transposition de partition.
  À partir d'une partition, l'objectif est d'obtenir une nouvelle partition avec le morceau transposé dans une autre tonalité.
- Troisième tâche : Conserver les partitions formatées dans une base de données pouvant être enrichie par les utilisateurs.

3.3 WP2 : Reconnaissance auditive

Membres : Alice Pellet-Mary, Antoine Grospellier, Baptiste Rozière, Louis Gal
L'objectif de ce WP est de passer d'un flux audio, obtenu en enregistrant le musicien, à un format exploitable par notre programme. À partir de ce format, on peut suivre le musicien dans la partition traitée par le WP1, ou bien créer une partition de ce que le musicien a joué.

Tâches

- Première tâche : Reconnaître les notes jouées en temps réel et les envoyer au WP3.
  Le format utilisé pour stocker les informations reconnues est proche de celui de lilypond (pour chaque note on écrit simplement son nom, son octave et sa durée en secondes). L'objectif est, pour chaque note jouée, de reconnaître la note et d'obtenir sa durée.

- Deuxième tâche : Obtenir une vraie partition à partir d'un flux audio.
  L'objectif de cette tâche est de créer une partition à partir d'un enregistrement du musicien. Cela pourrait permettre notamment aux musiciens composant de la musique, de jouer simplement ce qu'ils ont composé pour en obtenir une partition.

3.4 WP3 : Synchronisation musicale

Membres du WP3 : Alexandre Talon, Alice Pellet-Mary, Benjamin Farinier et François Pirot
L'objectif de ce work package est de parvenir à superposer deux flux musicaux correspondant à un même morceau. Il s'agit de réussir à mettre en correspondance une interprétation d'un morceau avec son interprétation théorique selon la lecture de la partition, et ainsi de suivre le musicien lorsqu'il joue. Cela se fait malgré les potentielles erreurs du musicien, qu'il faut par la suite lui signaler.
Ce work package utilise les partitions scannées par le work package 1 ainsi que le morceau enregistré et transformé par le work package 2.

Tâches

- Première tâche : Synchronisation musicien-partition.
  Il s'agit de suivre le musicien qui joue en temps réel, c'est-à-dire de savoir à tout moment sa position dans la partition.
- Deuxième tâche : correction des erreurs.
  La deuxième tâche consiste à faire un retour à l'utilisateur sur son interprétation du morceau : lui dire les endroits où il s'est trompé.

3.5 WP4 : GUI

Membres du WP4 : Alexandre Talon, **Diego Nava Saucedo**, Louis Gal et Pierre Macherel
Ce work package a pour but la création de l'interface graphique de Musicbot pour ses diverses déclinaisons. Il s'agit notamment de trouver une manière intuitive d'afficher et faire défiler une partition sur un écran. Il a aussi pour tâche de regrouper et faire communiquer les différents outils entre eux, ainsi que de proposer des fonctionnalités pour aider le musicien.

Tâches

- Première tâche : création d'une interface graphique pour ordinateur.
  Ce logiciel permet à l'utilisateur d'importer une partition au format pdf. Il se charge ensuite de la

	WP0 : Communication	WP1 : Numéritation de partition	WP2 : Reconnaissance auditive	WP3 : Synchronisation musicale	WP4 : GUI
Alexandre Talon				X	X
Alice Pellet-Mary			X	X	
Antoine Grospellier			X		
Baptiste Rozière	X		X		
Benjamin Farinier			X	X	X
Diego Nava Saucedo		X			X
François Pirot	X	X		X	
Louis Gal	X		X		X
Pierre Macherel		X			X
Valentin Gledel		X			

FIGURE 2 – Récapitulatif des différentes affectations. Une case grisée indique que la personne est chef du work package.

convertir, puis il affiche la partition reconstituée. Le musicien voit sa progression sur la partition. Cette dernière défile automatiquement lorsqu'il arrive en fin de page. Les éventuelles erreurs d'interprétation sont signalées, et un score est attribué à la fin du morceau. Il inclut aussi l'ajout de fonctionnalités supplémentaires tel que la recherche de partition en ligne ou encore l'édition de partition

- Deuxième tâche : communication entre les différents outils.
  Il s'agit de faire fonctionner les différents éléments servant à la reconnaissance et analyse sonore de manière parallèle à l'aide de thread, de les faire communiquer entre eux, pour faire en sorte que la partition défile à chaque fois qu'une note est jouée et reconnue.

3.6 Calendrier des work packages

- **Numérisation de partition** : 2 ou 3 mois à partir du début du projet (fin septembre)
- **Reconnaissance auditive** : 2 ou 3 mois à partir du début du projet
- **Synchronisation musicale** : 1 ou 2 mois à partir du début du projet
- **GUI** : Tout au long de l'évolution des autres WP
- **Préparation de la démonstration** : environ 2 semaines avant la démonstration (à partir de début janvier)

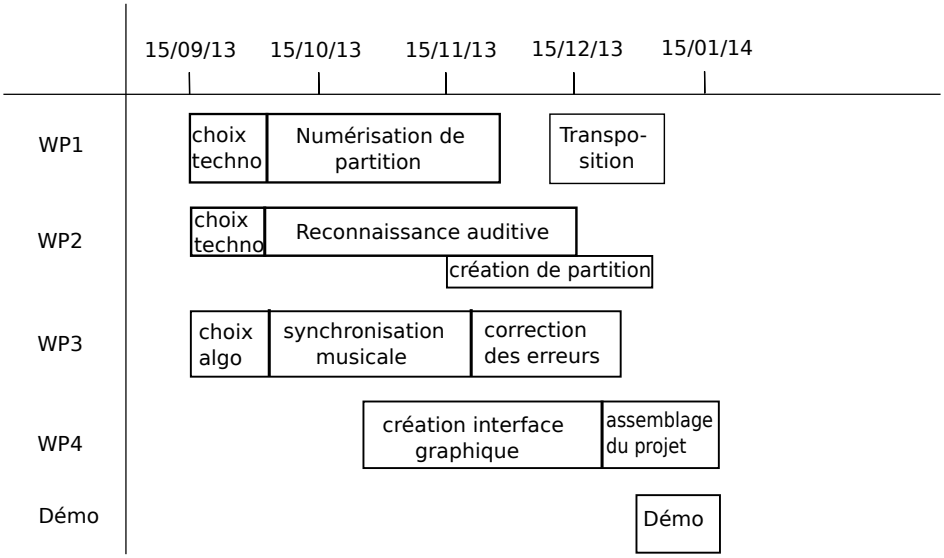


FIGURE 3 – Diagramme de réalisation des work packages

4 Description précise du travail de chaque WP

4.1 WP1 (Numérisation de partition)

Le logiciel Audiveris nous permet de numériser des partitions de musique au format pdf, pour obtenir en sortie un fichier dans le format musicXML. Ce fichier sera ensuite converti dans le format lilypond (grâce au logiciel musicxml2ly) et finalement, un dernier parseur transformera la sortie dans notre version allégée de lilypond. Le code du WP1 est en C++.

4.1.1 Choix du format numérique

Compte-tenu des formats préexistants et des bibliothèques disponibles pour chacun de ces formats, nous avons choisi d'exporter les partitions sous format lilypond. Un format simplifié découlant du format lilypond est utilisé au sein même du projet, qui se résume à la séquence des notes et de leur durée.

4.1.2 Base de données

Ce work package repose donc sur une base de données lilypond, hébergée par les serveurs de l'ENS de Lyon, et ayant pour ambition de grossir avec le nombre d'utilisateurs de MusicBot. En effet, l'utilisateur a la possibilité de rajouter à la base de données ses propres partitions au format lilypond, qu'il aurait déjà à sa disposition, ou bien qu'il aurait obtenues grâce à MusicBot. Ces partitions pourront alors être réutilisées de manière directe par les futurs utilisateurs, sans passer par l'étape de formatage.

4.1.3 Formatage d'une partition sous forme d'image

Parce que la base de données lilypond disponible sur internet est limitée, et parce qu'elle est très importante au format pdf, notamment via des sites comme imslp, il était naturel de chercher à proposer à l'utilisateur d'importer des partitions sous ce format d'image.

Cette étape se fait à l'aide du logiciel Audiveris. Ce dernier permet de faire de la reconnaissance d'image sur une partition dans des formats d'image divers (pdf, png, jpg, ...), et de la renvoyer au format XML, qui est un format descriptif exhaustif du contenu de la partition. Cette opération est très satisfaisante pour des images de départ de très bonne qualité, mais se dégrade rapidement avec la qualité du fichier d'entrée. De prochaines mises à jour devraient tendre à améliorer cette tendance.

Enfin, nous utilisons les outils fournis par lilypond pour transformer cette sortie XML en une sortie au format lilypond, exploitable par nos propres algorithmes.

4.1.4 Parsage du fichier lilypond - transposition

Un parseur des fichiers lilypond a été implémenté, afin de transformer des partitions complexes au format lilypond vers le format simple utilisé au sein même du projet - qui est donc invisible à l'utilisateur.

Mis en relation avec un module de transposition, ce parseur permet également de modifier le fichier lilypond afin d'y effectuer une opération de transposition selon un décalage défini par l'utilisateur. Le module de transposition en question est implémenté de sorte qu'il respecte, autant que possible, les règles de solfège respectées par la partition de départ, notamment en ce qui concerne les changements de tonalité. En conséquence, la sortie se veut logique, afin d'en faciliter le déchiffrage par un musicien ayant de bonnes connaissances en solfège.

4.1.5 Récapitulatif des choix technologiques

- Langage de programmation : C++
- Format de sortie des partitions : lilypond
- Logiciel de reconnaissance d'image vers une partition XML : Audiveris
- Conversion du format XML au format lilypond : outil musicxml2ly de la bibliothèque lilypond

4.1.6 Difficultés rencontrées et solutions

Les fichiers reconnus par Audiveris le sont de manière satisfaisante uniquement lorsque l'entrée est d'une très bonne qualité, de l'ordre de la sortie d'une partition compilée. Une mise à jour future d'Audiveris devrait proposer une amélioration de la reconnaissance dans le cas d'une entrée de qualité moyenne, de l'ordre d'une numérisation de basse résolution.

Lorsque la reconnaissance par Audiveris s'est mal déroulée, on obtient une partition qui comporte des trous, et qui ne respecte plus la bonne structure du solfège. Cela pose problème au moment du passage du format XML au format lilypond, à cause du côté beaucoup plus restrictif du format d'arrivée sur celui de départ. Nous n'avons malheureusement pas trouvé de solution autre que de recommencer la reconnaissance avec un fichier de meilleure qualité.

Même dans le cas d'une partition d'entrée de très bonne qualité, la sortie d'Audiveris n'est jamais parfaite. C'est pourquoi nous proposons à l'utilisateur des outils pour modifier une partition lilypond via une interface graphique. Une fois que l'utilisateur aura entièrement corrigé la partition, il pourra alors l'envoyer sur le serveur, pour garder une trace de son travail et ainsi en faire profiter les futurs utilisateurs.

Audiveris est très gourmand en ressources, et son installation très complexe et peu portable. C'est pourquoi nous avons décidé de l'installer sur un serveur distant, et proposons donc l'étape de reconnaissance de partition par l'intermédiaire de requêtes à ce serveur. Pour optimiser le tout, nous conservons les résultats de ces requêtes afin de pouvoir les proposer directement aux futurs utilisateurs, qui auront la liberté d'en corriger les potentielles erreurs.

4.2 WP2 (Reconnaissance auditive)

Le travail principal de ce work package consiste à convertir un fichier audio en un fichier texte facilement exploitable par notre programme. Cette conversion est difficile à réaliser. Heureusement, il existe déjà des outils permettant de la faire de manière plus ou moins satisfaisante. Pour ce projet, nous avons décidé de travailler avec une bibliothèque appelée Aubio.
À partir de cette bibliothèque nous avons développé deux programmes, un pour envoyer en temps réel les notes au WP3 dans le cadre de la synchronisation entre le musicien et la partition, et un autre pour créer une partition à partir d'un fichier son `.wav`.

4.2.1 Formats d'entrée et de sortie

Dans le cadre de la synchronisation entre le musicien et la partition, le WP2 prend en charge l'enregistrement du musicien et renvoie un flux de notes pris en charge par le WP3. Ce flux est au format `"nomDeLaNote octave duréeEnSeconde - noteSuivante octaveSuivante duréeSuivante - ..."` Où nomDeLaNote est un entier compris entre 0 (do) et 11 (si). On a donc
entrée : Un flux audio que l'on se charge d'enregistrer
sortie : Un flux de notes pour le WP3 au format `"note octave durée -"`

Dans le cadre de la création de partition, le WP2 prend en entrée un fichier audio au format `.wav`, et renvoie un fichier lilypond (http ://www.lilypond.org/) et un pdf avec la partition correspondant au fichier audio. Le WP4 doit également fournir au WP2 des informations nécéssaires à la création de la partition, comme par exemple le titre du morceau, la tonalité, la clé, etc...
entrée : Un fichier `entrée.wav` et les informations utiles à la création de la partition
sortie : Un fichier `sortie.ly` et sa partition compilée `sortie.pdf`

4.2.2 Récapitulatif des choix technologiques

Pour ce WP, nous avons utilisé la bibliothèque Aubio, dont la fonction `Aubionotes` permet d'extraire d'un fichier audio la séquence des notes reconnues, avec leur durée associée. Cette fonction peut fonctionner soit sur un fichier audio enregistré au format wav, soit sur un flux audio en temps réel. Dans ce dernier cas, c'est `Aubionotes` qui se charge de récupérer, à travers JACK, le flux audio. La fonction `Aubionotes` nous convenait bien pour l'utilisation d'Aubio sur un enregistrement (dans le but d'en faire une partition). En revanche, pour la synchronisation en temps réel du musicien et de la partition, la fonction `Aubionotes` utilisée en temps réel convertissait la sortie en midi, ce dont nous n'avions pas besoin. Nous avons donc adapté la fonction `Aubionotes` en une similaire, mais dont la sortie était adaptée à nos besoins, en utilisant les autres fonctions de la bibliothèque Aubiolib.
Le code d'Aubio est en C, et notre code est en C++.

4.2.3 Difficultés rencontrées et solutions

Aubio est un outil conçu pour fonctionner seulement sur un ordinateur : il prend en charge l'enregistrement du son et sa gestion, qui n'est pas forcément la même sur une tablette ou sur un smartphone. De plus son code est en C, avec des scripts en python, ce qui rend compliqué son portage sur Android. Pour une application Android de MusicBot, il faudrait donc soit porter le code d'Aubio pour Android, soit faire défiler la partition à une vitesse constante et se contenter de donner un score au musicien après avoir joué (en enregistrant le morceau avec le smartphone, puis en l'envoyant à un serveur qui ferait les calculs et utiliserait Aubio).
L'utilisation d'Aubio nous a aussi forcé à restreindre nos objectifs à des instruments monodiques. En effet, Aubio ne reconnaît pas les accords, notre programme ne pourra donc être utilisé que par des instruments monodiques (pas de guitare ou de piano).
Nous devons également nous restreindre à des instruments ne jouant que les notes de la gamme, car la fonction d'Aubio que nous utilisons ne renvoie pas la fréquence de la note, mais un arrondis au demi-ton le plus proche. Il n'est donc pas possible d'utiliser musicbot avec des instruments fretless, ou avec la voix, qui pourraient faire des notes de n'importe quelle fréquence. Il existe cependant la fonction d'Aubio `Aubiopitch` qui renvoie la fréquence de la note au lieu d'un arrondis, et on pourrait donc étendre musicbot aux instruments fretless ou à la voix si le projet se poursuit, mais il faudrait dans ce cas refaire tout le travail d'`Aubionotes` qui consiste à regrouper dans une même note les sons de fréquence proche.

4.2.4 Gestion des erreurs d'Aubio

Aubio fonctionne plutôt bien pour reconnaître les notes mais il y a quand même quelques petits problèmes : certaines notes sont renvoyées plusieurs fois alors qu'elles ne sont jouées qu'une fois (il renvoie régulièrement deux ou trois fois la même note courte au lieu d'une seule note longue), certaines notes dues au bruit peuvent apparaître, les notes n'ont pas toujours la bonne octave, ou encore certaines notes sont décalées d'un demi ton.
Nous avons donc créé un parseur qui transforme la sortie d'Aubio en une sortie au format voulu, en essayant de corriger ces erreurs. Nous nous sommes concentrés sur la fusion des notes dédoublées par Aubio et la suppression des notes dues aux bruits car ce sont les erreurs les plus courantes, et il semblait possible d'améliorer les choses (alors qu'il est plus dur de savoir si Aubio s'est trompé d'un demi ton ou pas).
Pour gérer ces erreurs nous avons fixé des seuils : hauteur maximum ou minimum d'une note, durée minimum d'une note, en dessous de laquelle on fusionne la note avec sa voisine s'il s'agit de la même note. Ces seuils sont modifiés lorsque l'utilisateur précise le tempo dans lequel il joue, et la clé qu'il utilise (clé de sol, clé de fa...).
Ce parseur est utilisé dans les deux tâches du WP2, la synchronisation et l'écriture de partition. Mais les seuils ne sont pas les mêmes dans les deux cas.

Pour la synchronisation, nous pouvons laisser passer plus d'erreurs car le WP3 est au courant de ces erreurs et essaye de ne pas les prendre en compte (le WP3 calcule la distance de Leveinstein entre les notes reconnues et la partition, il peut donc choisir de mettre un poids plus faible à certaines erreurs).
Pour l'écriture de partition, les notes dédoublées par Aubio ainsi que les notes dues au bruit sont problématiques, nous avons donc mis des seuils plus haut, pour laisser passer moins d'erreurs. Cependant, si ces seuils permettent de supprimer la plupart des notes dues aux erreurs d'Aubio, ils suppriment aussi parfois des notes qui n'étaient pas des erreurs. Mais le musicien peut ensuite aller corriger les erreurs dans le fichier lilypond, ou utiliser l'édition de partition proposé par la GUI.

4.2.5 Expériences

Pour trouver des seuils satisfaisants pour notre parseur, et vérifier que notre code fonctionnait, nous avons fait des expériences avec de nombreux enregistrements.
Nous avons testé Aubio avec des instruments différents, comme la flûte, le piano, ou encore le violon. Et nous avons trouvé qu'Aubio fonctionnait bien avec tous ces instruments, malgré leurs timbres assez différents. Aubio reconnaît même plutôt bien la voix humaine, à condition de ne chanter que des notes de la gamme. Nous avons également fait des tests avec des morceaux de tempo différents et avec plus ou moins de notes répétées, pour trouver des seuils permettant de supprimer la plupart des notes dédoublées par Aubio, sans pour autant fusionner des notes répétées volontairement.

4.2.6 Problèmes rencontrés dans l'utilisation d'Aubio en temps réel

Nous avons également rencontré un problème lors de l'utilisation d'Aubio en temps réel (pour la synchronisation). `Aubionotes` peut être utilisé soit à partir d'un enregistrement `.wav` (ce que nous utilisons pour la création de partition, et qui ne pose pas de problèmes), soit en direct (ce que nous utilisons pour la synchronisation entre le musicien et la partition). L'utilisation d'Aubio en direct nécessite l'utilisation de JACK (http ://jackaudio.org/) pour connecter le son entrant à Aubio, ainsi que pour récupérer la sortie d'Aubio. En effet, la sortie d'`Aubionotes`, lorsqu'on l'utilise en direct, est envoyé au format midi dans JACK, et il faut ensuite la récupérer pour notre programme (par exemple avec RtMidi : https ://ccrma.stanford.edu/software/stk/classRtMidi.html), et reconvertir le midi dans le format qui nous intéresse. Pour éviter ce deuxième passage par JACK et la conversion inutile en midi, nous avons donc modifié la fonction `Aubionotes` d'Aubio.

4.2.7 Écriture de partition

Pour la synchronisation, Aubio suivi du parseur suffisent pour obtenir le flux de notes dont a besoin le WP3. Mais pour l'écriture de partition, il faut réutiliser ce flux de notes pour en faire un fichier lilypond.
L'écriture d'une partition de musique à partir d'un flux de notes peut être compliquée car il y a plusieurs possibilités lors de l'écriture d'une partition : choix de la clé, de la tonalité, des rythmes, de la mesure, etc. Nous nous sommes donc restreints à des cas simples, qui pourront être améliorés si le projet se poursuit.

Altérations accidentelles : Sol dièse ou la bémol ?
Pour gérer les altération accidentelles dans la partition, nous ne prenons en compte que l'armure : s'il y a des bémols à la clé, nous choisissons en priorité une altération bémol, et une altération dièse s'il y a des dièses à la clé. Et s'il n'y a rien à l'armure, nous avons fixé un gagnant parmi chaque couple, en fonction de sa probabilité d'apparition (par exemple fa dièse gagne sur sol bémol). Cette solution n'est pas parfaite, mais elle couvre déjà un grand nombre de cas.

Rythme et mesure : Trouver le rythme d'une note est essentiel pour la création de la partition et également très compliqué à cause de la quantité de rythmes possibles. Pour simplifier le problème, nous nous sommes limités à des partitions binaires, dont la pulsation est la noire (les mesures autorisées sont 1/4, 2/4, 3/4 et 4/4), et la plus petite unité de temps est la double croche. Cette restriction couvre beaucoup de morceaux classiques, mais élimine aussi un grand nombre de morceaux.
La deuxième difficulté du rythme est que le musicien ne joue jamais parfaitement en rythme. Nous voulions donc proposer 2 modes d'enregistrements pour la partition. Un premier mode avec un métronome, pour que le musicien soit le plus en rythme possible, et un deuxième mode sans métronome. Cependant, pour des raisons techniques, nous n'avons pas eu le temps d'implémenter l'enregistrement audio du morceau par

l'interface graphique, donc le mode avec métronome n'est actuellement pas disponible. Mais la fonction du WP2 permettant sa mise en place est déjà écrite.
Dans le mode sans métronome, le tempo est fixé avec la première note (qui doit être une noire), et adapté au cours du morceau. Si le musicien change de tempo lentement, la partition s'adaptera. En revanche, pour pouvoir faire cela, nous avons interdit tous les rythmes peu fréquents (par exemple une noire liée à une double croche), et nous n'autorisons pas les notes liées d'une mesure à une autre.
Pour les musiciens voulant un peu plus de choix dans les rythmes, il faut utiliser le mode avec métronome (c'est un métronome juste visuel pour ne pas gêner la reconnaissance auditive). Dans ce mode, on autorise certains rythmes en plus, et les notes liées d'une mesure à une autre, en se basant sur le fait que le musicien suit bien le métronome.

Correction des erreurs : Nous avons essayé de faire le moins d'erreurs possible dans la création de la partition mais il peut toujours y en avoir quelques une de présentes. Les musiciens connaissant Lilypond peuvent ensuite aller corriger ces erreurs dans le fichier Lilypond. Pour faciliter cela, la partition est écrite avec des retours à la ligne à la fin de chaque mesure, ce qui rend le fichier Lilypond plus lisible. Pour les musiciens ne voulant pas revenir au fichier Lilypond, il est possible de modifier la partition avec l'édition de partition proposée par le WP4.

4.3 WP3 (Synchronisation musicale)

Le WP3 reçoit du WP1, avant le début du morceau, la suite des notes de la partition au format `nomDeLaNote octave duréeEnSeconde -` (une durée de une seconde est fixée par défaut pour les noires, et les durées des autres rythmes découlent de cette durée : 0.5 pour les croches, 2 pour les blanches, etc). Lorsque le musicien joue, il reçoit les notes du WP2 au même format, mais en temps réel.
Le WP3 s'occupe d'abord de repérer à tout moment à quel endroit de la partition le musicien se trouve. Cette donnée est ensuite envoyée en temps réel au WP4 pour l'affichage de la partition. Dans un second temps, il indique au musicien les erreurs qu'il a effectuées.

4.3.1 Récapitulatif des choix technologiques

Le code du WP3 est en C++ et ce WP n'utilise pas de bibliothèque ou d'outil particulier.

4.3.2 Difficultés rencontrées et solutions

Dans le WP3, il s'agit de chercher dans un texte un motif qui varie au cours du temps : les dernières notes jouées par le musicien. Divers algorithmes existent pour rechercher, à la volée, un motif dans un texte. Cependant, dans ces algorithmes c'est le texte que l'on traite à la volée après avoir fait des pré-calculs sur le motif, alors que dans notre cas il s'agit de l'inverse. Pour palier à ce problème, il a été décidé d'effectuer des calculs de distance d'édition entre le motif joué et chaque sous-chaîne de notes de la partition : quand le musicien fait trop d'erreurs d'affilée, on recherche la sous-séquence de la partition la plus proche de la séquence d'erreur, en terme de distance d'édition. Pour réduire la taille de l'alphabet, nous avons considéré que les lettres étaient juste le nom de la note (do, do#, ré...), et pas son octave. Ainsi, un la 440 et un la 880 seront considérés comme la même note. Nous ne prenons pas non plus en compte la durée des notes lors de la recherche du motif. Cette recherche de motif se fait via l'algorithme de Levenstein, qui utilise le paradigme de la programmation dynamique.

4.3.3 Suivi du musicien

Le suivi du musicien se fait en deux temps. Dans un premier temps, le programme essaye de suivre le musicien en considérant qu'il joue bien, sans rajouter des notes, supprimer des notes ou revenir en arrière. À chaque nouvelle note, le programme avance d'une note. Cependant, si le programme observe 3 erreurs de suite, il considère qu'il a perdu le musicien, et passe à la recherche de motif. Pour cela, il regarde les dernières notes jouées par le musicien, et calcule la distance d'édition de ce motif à toutes les sous-chaînes du texte de longueur à peu près la même. Il choisit ensuite la sous-chaîne minimisant la distance d'édition, et s'il y a égalité entre plusieurs sous-chaînes, il choisit la plus proche de la dernière position connue du musicien dans la partition. Malgré le fait de choisir la chaîne la plus proche, nous avons observé expérimentalement que le programme sautait parfois à des endroits très éloignés dans la partition. Nous avons donc rajouté

un paramètre de confiance, pour pouvoir favoriser plus ou moins les chaînes proches. Ainsi, si le musicien connaît bien le morceau et ne se reprend pas trop loin dans la partition, il peut modifier ce paramètre pour que le programme ne saute pas n'importe où dans la partition lorsqu'il joue.
Nous avons également prévu de ne faire la recherche du motif que dans la partition affichée au musicien, en supposant qu'il ne connaît pas le morceau par cœur et joue toujours quelque chose d'affiché à l'écran.
Une fois la recherche de motif terminée, le programme recommence à essayer de suivre le musicien, en remettant le nombre d'erreurs à zéro.

4.3.4 Gestion des erreurs d'Aubio

Pour le suivi du musicien, nous sommes partis du principe que le fichier contenant la partition renvoyé par le WP1 était parfait (le musicien peut le corriger s'il y a des erreurs). En revanche, nous avons essayé de tenir compte des erreurs d'Aubio que le WP2 n'avait pas réussi à supprimer.
Nous avons adapté la distance d'édition pour que la recherche du motif dans le texte tienne compte des erreurs d'Aubio. Au lieu de mettre un coût de 1 pour l'ajout, la suppression ou la modification d'un caractère, nous avons mis des coûts adaptés. Pour la modification d'une note, le coût varie selon la distance entre les deux notes. Ainsi, la modification d'un do en do# ou en si coûtera moins cher que la modification d'un do en sol. Cela permet de gérer les cas où Aubio se trompe d'un demi ton dans la reconnaissance des notes. Nous avons également mis un coût plus faible aux insertions de notes qu'aux suppressions, car malgré le traitement du WP2, il y a encore régulièrement des notes dédoublées ou des notes ajoutées.
Enfin, dans la partie où le programme suit le musicien (quand il n'a pas encore atteint le nombre maximum d'erreurs et lancé la recherche de motif), si nous recevons une fausse note, mais que la note précédente était la même, nous considérons que c'est une note dédoublée par Aubio, et nous restons à la dernière position, sans augmenter le nombre d'erreurs.

4.3.5 Affichage des erreurs du musicien

Nous voulions afficher au musicien ses erreurs à la fin du morceau pour lui permettre de s'améliorer. Ce travail est en fait plus difficile que prévu car la plupart des erreurs de notes que nous observons sont dues aux erreurs de reconnaissance d'Aubio et non au musicien. Même si nous avons réussi à réduire ces erreurs et à suivre le musicien, le calcul du nombre de notes justes jouées par le musicien n'est pas du tout représentatif. Par exemple, pour un morceau joué presque parfaitement, on compte 40% de notes fausses, ou non reconnues dans la partition lors de la phase de recherche de motif (lors de la recherche du motif, certaines notes sont considérées comme des ajouts, et ne correspondent donc à aucune note de la partition). Cependant, malgré le nombre important d'erreurs qui ne sont pas dues au musicien, on observe quand même qu'entre un morceau bien joué et un morceau moins bien joué, le second morceau obtient un pourcentage de notes fausses bien plus important. Nous pensons donc afficher à l'utilisateur une note relative, sa progression, plutôt que le nombre de notes justes, qui n'est pas représentatif. Mais nous n'avons pas encore eu le temps d'ajouter cette fonctionnalité à l'interface graphique.
Nous avons également une fonction qui permet de calculer le tempo du musicien au cours du morceau, et il pourrait être intéressant d'afficher la courbe du tempo en fonction du temps, pour que l'utilisateur puisse voir s'il accélère ou ralentit par endroits. Cette fonctionnalité n'est pas non plus disponible dans l'interface graphique pour l'instant.
Il existe aussi une fonction permettant de calculer le nombre de fois que le musicien s'est repris, ou a sauté des notes, mais là encore ce n'est pas forcément très représentatif de ce que le musicien a vraiment fait, et ce n'est pas implémenté dans l'interface graphique.

4.4 WP4 (GUI)

Le WP4 (GUI) se charge d'agglomérer les différents works packages en un logiciel cohérent. Il propose à l'utilisateur de charger une partition de musique en pdf depuis une banque de données (IMSPL). Ce pdf est ensuite converti par le WP1. L'interface utilisateur envoie au WP3 le nom du fichier contenant la partition puis récupère les données du WP3 afin d'afficher la partition (cet affichage suit la progression du musicien), et d'indiquer à l'utilisateur son score relatif.

4.4.1 Récapitulatif des choix technologiques

L'application est développée en C++, à l'aide de la bibliothèque Qt. En plus de cela nous utilisons une bibliothèque tierce nommée poppler-qt5. Qt a l'avantage de fournir un ensemble assez complet d'outils permettant de créer une interface utilisateur de manière intuitive. Qt est en plus multi-plateforme, c'est-à-dire que l'application créée peut fonctionner aussi bien sur Linux que sur Windows ou Mac OS X. Toutefois, à l'heure actuelle, l'application n'a été pensée que pour fonctionner sur Linux et Mac OS. La bibliothèque poppler-qt5, quand à elle, nous permet de convertir un pdf en une QImage qui est une classe de Qt permettant de stocker et manipuler des images. Cette dernière est aussi multi-plateforme.

4.4.2 Défilement de Partition

Le but principal de musicbot étant d'offrir aux musiciens un outil de défilement intelligent permettant de remplacer les partitions, nous avons beaucoup réfléchi afin de trouver un mode de défilement permettant une lecture à la fois fluide et agréable. Plusieurs possibilités s'offraient à nous, la première, et la plus commune dans le monde de l'informatique, était de faire un défilement continu de la partition, les portées allant du bas vers le haut, comme si l'utilisateur avait un lecteur de pdf et faisait défiler la partition lui même, tout en faisant en sorte d'afficher à la fois les dernière portées jouées, celle courante et les suivantes. De cette manière le musicien avait la possibilité de revenir en arrière s'il jugeait avoir mal interprété une partie du morceau afin de la rejouer et pouvait aussi anticiper la suite du morceau à tout moment, ce qui n'est pas possible avec une partition physique. Cependant, ce mode de défilement présente un défaut, en effet le défilement continu forcerait le musicien à constamment déplacer son regard pour pouvoir suivre la portée courante. De plus bien que continu le défilement n'est pas forcément à vitesse constante, d'autant que dans notre cas le défilement dépend du musicien car le logiciel le suit et non l'inverse, ce qui rend une lecture prolongé assez inconfortable.

Nous avons alors choisi un défilement portée par portée. Là encore plusieurs solutions se présentaient. Une solution naturelle était de modifier toutes les portées affichées par les suivantes une fois arrivé à la fin, mais ceci revenait a simuler le comportement d'une partition classique, ce qui n'apporte pas de réelle plus-value. L'idée est alors de coupler le défilement en continu et portée par portée. Ainsi, une fois arrivé à la fin d'une portée, la portée la plus ancienne est remplacée par une nouvelle portée. Nous aurions pu simplement remonter chaque portée d'un cran afin que le musicien lise toujours sur la portée du milieu mais l'irrégularité du défilement aurait posé problème. C'est pour cela que nous avons choisi un système qui laisse les portées fixes et remplace simplement les portées déjà lues par des portées à lire. Cela se rapproche beaucoup d'une partition traditionnel sans l'inconvénient du changement de page, même si les portées ne sont plus forcément ordonnées de haut en bas.

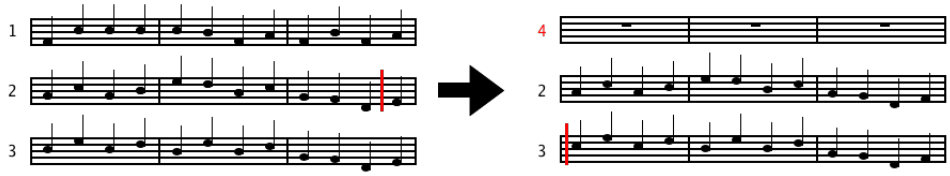


FIGURE 4 – Illustration du défilement, lorsque l'on a fini de lire la deuxième portée, la quatrième remplace la première.

4.4.3 Affichage de Partition

Le défilement des partitions étant particulier il nous fallait un moyen de manipuler la partition et chacune des portées de manière indépendante. Une possibilité était de créer la partition nous même, mais cela demande une très grande quantité de travail. Il nous fallait donc utiliser la partition générée par lilypond à partir du fichier obtenu en numérisant la partition. Le problème était de pouvoir obtenir les informations nécessaires à partir du pdf obtenu. À partir de l'image brut uniquement ceci revenait à refaire une numérisation mais nous tournerions alors en rond. Fort heureusement lilypond lors de la génération du pdf rajoute des hyperliens pour chacune des notes et silences de la partition, nous pouvons alors grâce à cela récupérer toutes les informations nécessaire, à savoir les coordonnées des notes à l'intérieur du pdf

ainsi que leur position a l'intérieur du fichier lilypond d'origine. À l'aide de ces informations il nous est alors possible de séparer chacune des portées. En effet la position des note nous donne déjà un sous-intervalle plus ou moins précis de la position de chacune des portées, puis pour avoir plus de précision on analyse l'image en recherchant une ligne totalement vide entre deux portées, ceci nous permet d'avoir un découpage par portée de la partition.

4.4.4 Édition de partition

Les technologies utilisées pour la numérisation de partition ainsi que celle pour l'écriture de partition à partir d'un flux audio n'étant pas parfaites, il était nécessaire de proposer à l'utilisateur de pouvoir modifier et corriger la partition de manière simple et sans avoir besoin de connaissances en lilypond. C'est pourquoi nous avons ajouté à l'application un éditeur de partition. L'éditeur prend en entrée des fichiers lilypond en notation relative et retourne le fichier lilypond corrigé dans le même format.
De la même manière que pour le défilement de partition nous avons choisi de travailler directement à partir du pdf généré par lilypond. Encore une fois les informations données par lilypond sont suffisantes pour pouvoir modifier le fichier lilypond avec une interaction graphique. Grâce aux coordonnées données par les hyperliens du pdf nous savons sur quelle note l'utilisateur a cliqué et la position de la note dans le fichier lilypond nous permet de récupérer les informations de cette note, à savoir la note en elle même, l'octave, la durée, etc à l'aide d'un parsage préalable. Étant donné que nous nous trouvons en notation relative (l'octave, la durée de la note entre autre dépendent de la note précédente) en plus de modifier la note sélectionnée par l'utilisateur, il nous faut modifier les note adjacentes pour compenser les modifications non désirées. Les modifications nécessitant de recompiler le fichier lilypond pour avoir la partition modifiée en pdf demandent à l'utilisateur de confirmer les changements effectués sur la note courante. S'il sélectionne une autre note sans avoir appliqué les changements ceci seront annulés. En plus de cela l'éditeur permet d'ajouter et de retirer des notes dans la partition ce qui permet à l'utilisateur d'avoir des outil complet pour la création de partition de base.

4.4.5 Difficultés rencontrées et solutions

La bibliothèque de Qt ne propose pas de classe pour la gestion des pdf. Nous avons donc cherché des bibliothèques permettant de convertir des pdf en fichier image. Deux bibliothèque nous proposaient ces fonctionnalités poppler-qt5 et Magick++, sachant que poppler nous permettait d'obtenir une QImage, qui est la classe des images pour Qt, à partir d'un pdf. Bien que ces bibliothèques soient multi-support et que leur intégration dans les distributions linux ne pose pas de problèmes, leur utilisation dans un environnement Mac est plus compliquée. En effet l'installation de poppler-qt5 a posé de nombreux problème, surtout pour la compilation. Alors que Magick++ ne posait pas de problème d'installation mais la bibliothèque montrait des dysfonctionnement lorsqu'elle était utilisée avec Qt, empêchant la conversion des pdf en format images (tandis que lorsque Qt n'était pas utilisé ce problème n'apparaissait pas). Finalement les problèmes de l'installation de poppler-qt5 venaient du fait que l'installateur ne regardait pas aux bons endroit et demandait la redéfinition de nombreuse variables de chemins. Le problème de Magick++ n'étant toujours pas résolu à cette heure justifie l'utilisation de poppler-qt5.

4.5 Mise en place du serveur

Le choix d'utiliser un serveur distant pour Musicbot résulte de plusieurs considérations. La première, pratique, est de proposer une base de donnée afin de permettre aux utilisateurs de partager leurs partitions avec la communauté. La seconde, technique, provient à la fois de la volonté d'éviter à l'utilisateur l'installation complexe de certains paquets comme de la nécessité de fournir une puissance de calcul non disponible sur tous les supports.

Le serveur devait donc être capable de faire tourner Aubio et Audiveris. Dans le but d'en faciliter l'installation, une distribution contenant directement les paquets correspondant devait être choisie. De plus, l'installation canonique de Audiveris se fait via "javaws", un lanceur Java net requérant la présence d'une interface graphique. Ces deux critères ont donc orienté le choix vers une distribution Ubuntu serveur.

Coté logiciel, Nginx a été choisi comme serveur Web, à la fois pour sa légèreté et pour la facilité de sa configuration. Pour le langage de script, c'est PHP qui a été utilisé, car déjà maîtrisé par plusieurs membres du projet. Enfin pour la base de donnée, c'est SQLite qui avait été envisagé, pour son intégration aux deux

technologies précédemment citées et pour sa simplicité. Cependant cette partie n'a pas été concrétisée faute de temps.

5 Améliorations futures possibles

5.1 WP1 (Numérisation de partition)

Le logiciel Audiveris sur lequel nous nous reposons devrait bientôt connaître une mise à jour majeure, qui lui permettrait d'améliorer sensiblement sa méthode de reconnaissance, notamment dans le cas de partitions d'entrée de qualité médiocre. De plus, cette nouvelle version d'Audiveris sera uniquement accessible via serveur du développer de ce logiciel. Il s'agira alors de substituer ce nouveau serveur au nôtre afin que notre programme fonctionnement de nouveau.

Toujours dans un but d'améliorer le rendu, il serait envisageable d'implémenter un noyau de solfège sur lequel s'appuie la reconnaissance, afin d'en repérer un grand nombre d'erreurs qui seraient directement signalées à l'utilisateur, afin de lui faciliter la tâche de correction de la partition renvoyée. La meilleure option imaginable serait de faire dialoguer Audiveris avec ce noyau de sorte que les erreurs soient corrigées automatiquement, mais cela demande encore un travail considérable.

5.2 WP2 (Reconnaissance auditive)

L'écriture de partition pourrait être améliorée afin de gérer des rythmes plus compliqués, des changement de tonalité ou de tempo dans le morceau, ou encore permettre d'écrire des morceaux ternaires (pour l'instant, les seules mesures acceptées sont 1/4, 2/4, 3/4 et 4/4). Il pourrait aussi être intéressant de rendre l'écriture interactive, en la reliant à l'éditeur de partition du WP1, afin de remplacer l'insertion à la main de chacune des notes par la saisie sonore que nous proposons actuellement.

Un second objectif plus ambitieux serait de parvenir à traiter le cas des instruments polyphoniques. Il y a deux idées possibles pour y parvenir, qui s'exécutent à des niveaux différents. La première serait d'adapter Aubio afin qu'il ait la possibilité de reconnaître des notes jouées simultanément. La seconde serait de continuer à ne reconnaître qu'une seule note à la fois, potentiellement arbitrairement parmi un ensemble de notes jouées, et de travailler au niveau de la mise en relation avec la partition, afin de trouver une combinaison de sélections de notes de la partition polyphonique qui corresponde à l'ensemble des notes entendues. Cette étape serait grandement facilitée si l'on avait une propriété assurée sur la sélection de la note reconnue à l'intérieur d'un accord, par exemple celle de fréquence la plus élevée.

5.3 WP3 (Synchronisation musicale)

Afin d'optimiser la pertinence du suivi du musicien sur le partition, il est possible de prendre en compte le rythme attendu par la partition, relié à un métronome variable qui se règle automatiquement et en direct sur la performance du musicien.

Dans le compte-rendu de l'interprétation, d'autres critères que simplement le suivi du texte de la partition peuvent être pris en compte : le respect du rythme le long du morceau, la stabilité de la pulsation, une bonne gestion des nuances demandées. Plus les critères seront nombreux et précis, plus ce compte-rendu sera intéressant pour les musiciens de bon niveau.

5.4 WP4 (GUI)

L'application Android pour tablette et smartphone est pour l'heure en suspend. Du fait de l'incompatibilité des outils utilisé avec Android, la dépendance avec le serveur est trop grande. Dans l'éventualité où ces outil puissent un jour être utilisé sur ces terminaux, le développement sur Android pourra être effectué. Il serait aussi possible pour l'éditeur de partition de supporter la notation lilypond absolu, notation qui est plus simple que la notation relative. Il serait également envisageable de rajouter des modes d'affichages supplémentaires et de permettre à l'utilisateur de choisir le mode qui lui convient le mieux. Et pour finir nous pouvons continuer à proposer des fonctionnalité supplémentaire pour rendre l'utilisation de l'application plus agréable pour l'utilisateur.

6 Unification des Work Packages

6.1 Conventions de codage

Nous avons fixé quelques conventions de codage pour que le code soit cohérent.
Langue : Nous avons choisi d'écrire nos noms de fonction, de variables et nos commentaires en anglais pour permettre à des gens ne parlant pas forcément français de contribuer au projet.
Noms de variables et de fonctions : Nous avons écrit nos nom de fonction et de variable de la forme `maFonction` et `maVariable`, avec parfois l'utilisation d'un `_` pour indiquer le type des variables, par exemple `maVariable_type`.

6.2 Threads et appels de fonction

Le programme global fait des appels à plusieurs fonctions devant tourner en parallèle, nous avons donc utilisé des threads, pour permettre par exemple à l'interface graphique et à Aubio de fonctionner en même temps.

Pour la partie synchronisation du musicien avec la partition, l'interface graphique est dans un thread, `Aubionotes` tourne dans un autre thread, et il en faut encore un dernier pour JACK qui se charge de récupérer l'entrée audio. La fonction `Aubionotes` que nous avons modifiée fait ensuite des appels aux fonctions des WP2 et WP3 pour supprimer les erreurs d'Aubio et calculer la position du musicien en temps réel dans la partition. Un message est envoyé à l'interface graphique pour pouvoir afficher la position du musicien en temps réel sur la partition. L'interface graphique envoie également un message à aubionotes pour lui indiquer les portées qui sont affichées à l'écran.

Pour les autres fonctionnalités de MusicBot, le principe des threads est le même : une pour l'interface graphique et une autre pour faire les calculs avec les fonctions des WPs. Les deux threads communiquent en s'envoyant des messages lorsque c'est nécessaire.

7 Retombées

7.1 Objectif principal

À terme, MusicBot aura pour ambition de se retrouver sous diverses plates-formes, allant du smartphone à l'ordinateur, et de proposer sous chacune de ses déclinaisons un assistant complet au travail musical adapté à son support. MusicBot a la volonté d'aider le musicien à rendre son travail efficace, d'une part en lui fournissant des outils rendant la phase de déchiffrage plus confortable, et d'autre part en cernant les parties du morceau dont la réalisation est imparfaite et nécessite un travail plus approfondi et précis.

Il se veut totalement libre afin de pouvoir profiter d'une communauté prête à le faire évoluer, et de compter un nombre d'utilisateurs conséquent. Ce n'est que de cette manière qu'il pourra proposer une bibliothèque de partitions numériques conséquente et grandissante, et que la qualité des services proposés s'améliorera. L'utilisateur n'a pas forcément envie de passer par une étape de préparation de la partition numérique coûteuse en temps, avant de commencer à travailler dessus. C'est pourquoi nous comptons sur certains d'entre eux pour contribuer à améliorer et agrandir la base de données de partitions proposée par MusicBot.

7.2 Public visé

Sont visés par cet assistant tous les musiciens qui n'auraient pas encore acquis une autonomie complète dans leur travail. Cela concerne notamment tous les élèves en conservatoire et écoles de musique, qu'ils soient débutants ou bien qu'ils soient plus avancés. Chacun pourra y trouver des outils qui sauront se démontrer d'une grande aide. Cependant, MusicBot n'a pas la prétention de pouvoir remplacer un professeur !

L'utilisation de MusicBot se limite au travail d'un seul instrument à la fois. L'application n'est donc pas adaptée à une utilisation au sein d'un orchestre, ou même d'un petit ensemble d'instrumentistes. Elle intervient lors du travail individuel de chacun des musiciens.

Le portage de l'application sur le plus de plates-formes possibles permettra d'élargir tout autant l'ensemble de ses utilisateurs.

8 Concurrents

Le principal concurrent de MusicBot est l'application pour Ipad Weezic, qui propose un outil d'accompagnement automatique des musiciens. MusicBot s'en différencie en se penchant sur l'analyse des erreurs d'interprétation du musicien et en l'aidant dans son travail. De plus, nous ne sommes limités ni par le support (exclusivement Ipad pour Weezic), ni par la base de données (seules des partitions pré-formatées sont utilisables par Weezic). En plus de cela, notre contenu est entièrement gratuit, contrairement à Weezic dont les partitions pré-formatées sont payantes.

Conclusion

Notre programme permet aux utilisateurs d'obtenir des partitions numériques à partir d'un scan ou même d'une entrée audio. Le serveur permet même d'effectuer l'interprétation de scan sans installer de logiciel. Il aide aussi les musiciens avec le suivi sur la partition et la critique d'interprétation. La possibilité d'obtenir un score à la fin du morceau permet aux débutants de chiffrer leurs performances de façon ludique. Le suivi sur la partition permet aussi de jouer sans avoir à tourner de pages, ce qui peut faire perdre en fluidité. Les autres partie du logiciel s'adressent plutôt à des musiciens plus expérimentés. L'interprétation de scans et l'obtention de partitions à partir d'un morceau joué intéressent de nombreux musiciens professionnels qui ont fréquemment besoin de créer des partitions au format numérique. Nous en avons contactés certains qui sont prêts à bêta tester le programme.

Nous avons plusieurs pistes pour améliorer le projet. Tout d'abord, une application pour tablettes et smart-phones rendrait le logiciel beaucoup plus pratique. Ensuite, la critique d'interprétation pourrait encore être améliorée au niveau de l'évaluation du rythme, et il serait intéressant de prendre aussi en compte le volume. Une méthode améliorant la précision du rythme pour l'obtention de partitions à partir d'un flux audio en utilisant un métronome est en cours de développement.

Dans un futur plus lointain, il serait intéressant d'étudier les moyens disponibles pour permettre de reconnaître des entrées audio polyphoniques.