ENSI CAEN

ÉCOLE PUBLIQUE D'INGÉNIEURS
CENTRE DE RECHERCHE

Automix Software RC 1.0

Rapport de projet

Auteurs : M. Stevenot ; G. Hannes ; J. Ernult ; L. Carlier ; P. Gabon

Tuteurs : Baptiste Hemery, Estelle Cherrier

ENSICAEN
6, boulevard Maréchal Juin – CS 45 053 – F- 14050 Caen Cedex 4
Tél. +33 (0)2 31 45 27 50
Fax +33 (0)2 31 45 27 60

Table des matières

Introduction

Dans le cadre du projet de deuxième année, nous avons voulu proposer un projet innovant, que nous avons élaboré à partir d'une idée et surtout du constat suivant : dans les établissements publics ou dans un contexte privé, les options pour diffuser de la musique en continue sont souvent limitées :

- La radio : elle inclut des publicités et on n'a pas le choix des morceaux qui sont diffusés.
- Un DJ : souvent cher, cela demande une logistique importante, cependant la rendu sonore est agréable.
- Une playlist de fichiers musicaux : l'enchainement des titres n'est pas toujours automatisé et les transitions sont franches et désagréables.

La solution que nous apportons, *Automix Software*, est un logiciel de création de mix intelligent qui permet de tirer avantage des options précédentes sans subir leurs inconvénients. Il permet au grand public de créer des mix ininterrompus composés de musiques choisies par l'utilisateur. Les différentes musiques sont ordonnées selon plusieurs critères à la manière d'un DJ et s'enchainent avec des transitions agréables.

Pour créer un mix, l'utilisateur doit entrer dans le logiciel un ensemble de fichiers musicaux au format mp3, il pourra ensuite demander au logiciel de les trier et de générer un fichier mp3 résultant contenant le mix, ce processus est illustré par la Figure 1.

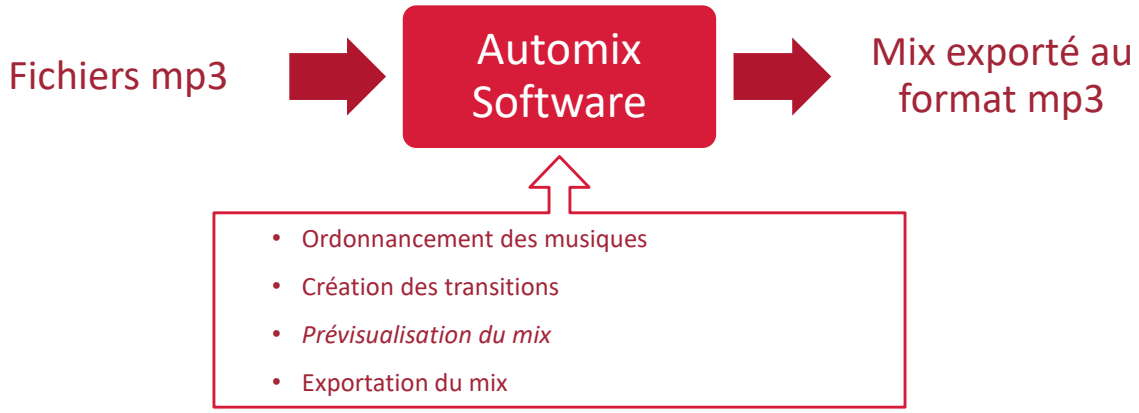

Figure 1 Processus de génération d'un mix audio

La version actuellement distribuée d'*Automix Software* est la *Release Candidate 1.0,* cette version embarque la majeure partie des fonctionnalités prévues au début du projet. Disponible en Français et en Anglais, cette application est destinée au grand public mais également aux professionnels qui voudraient compléter leur savoir-faire. *Automix Software* est compatible avec l'ensemble des versions *Windows*[1] les plus récentes, cela nous assure qu'un grand nombre de personnes sera en mesure d'exécuter le logiciel.

[1] *Windows 7* (2009) et plus récent

1 Organisation

1.1 Méthodes agiles

Nous suivons une méthodologie de développement agile reposant sur des sprints de deux semaines avec présentation d'un prototype à chaque fin de sprint.

Nous avons subdivisé le projet en un ensemble (non exhaustif) de *user stories*, c'est-à-dire tous les scénarios d'utilisation possibles. Les *user stories* sont par exemple : « En tant qu'utilisateur je peux redimensionner la fenêtre du logiciel comme je le souhaite » ou encore « En tant qu'utilisateur je peux choisir un dossier contenant des musiques ».

À partir de la liste des *user stories*, nous pouvons définir les produits suivants :

- Le produit minimal présente les fonctionnalités basiques indispensables exigées par le client. Dans notre cas, cela correspond à la réalisation d'une interface intuitive permettant la création d'une concaténation de musiques triées de façon rudimentaire.

- Le produit intermédiaire présente, en plus du produit minimal, la plupart des fonctionnalités jugées importantes. Il s'agit pour nous d'affiner le tri en utilisant une intelligence artificielle se basant sur différents critères musicaux et d'appliquer des fondus entre les pistes.

- Le produit idéal présente, en plus des produits précédents, des fonctionnalités augmentant le confort d'utilisation et la qualité du mp3 en sortie. Cela se traduit par le fait de réaliser des transitions variées, synchronisées et adaptées aux musiques ainsi que de donner la capacité à l'utilisateur d'influer sur les paramètres du mix.

À ces *user stories* nous associons des fonctionnalités et des tâches, qui nous permettent de répartir le travail. Pour contrôler l'avancement du projet une traçons une courbe *Burn-Up*[2] dont le principe est expliqué dans la partie 1.2. Cette méthodologie nous permet d'avancer avec le plus de constance et de rigueur possible et d'identifier rapidement et clairement les problèmes que nous devons résoudre.

Lors de la deuxième phase du projet, les *sprints* ont parfois duré plus de deux semaines. En effet il n'était pas toujours évident de fixer une date de réunion qui concordait avec les emplois du temps de tous les protagonistes du projet.

1.2 Présentation des outils

Nous développons notre logiciel en C++ avec l'environnement de *développement Microsoft Visual Studio* à destination de la plateforme *Microsoft .NET*. Le *.NET Framework* est un ensemble de composants logiciels structurels, permettant le développement d'applications. Il est distribué par Microsoft, à destination, principalement, des systèmes d'exploitation *Windows*. Il inclut une machine virtuelle gérant l'exécution d'applications *.NET*, d'outils de développement comme une bibliothèque de classes (*Framework Class Library*) et offre une inter opérabilité des langages.

[2] Voir Annexe 1: Courbe Burn-Up

Pour organiser nos *user stories* et nos tâches, nous utilisons *Trello* (Figure 2), un outil en ligne, que l'on peut décrire comme étant un tableau contenant des cartes. Ces cartes décrivent des tâches ou des *user stories* et peuvent être affectées à des membres, être ornées d'une description et de commentaires.

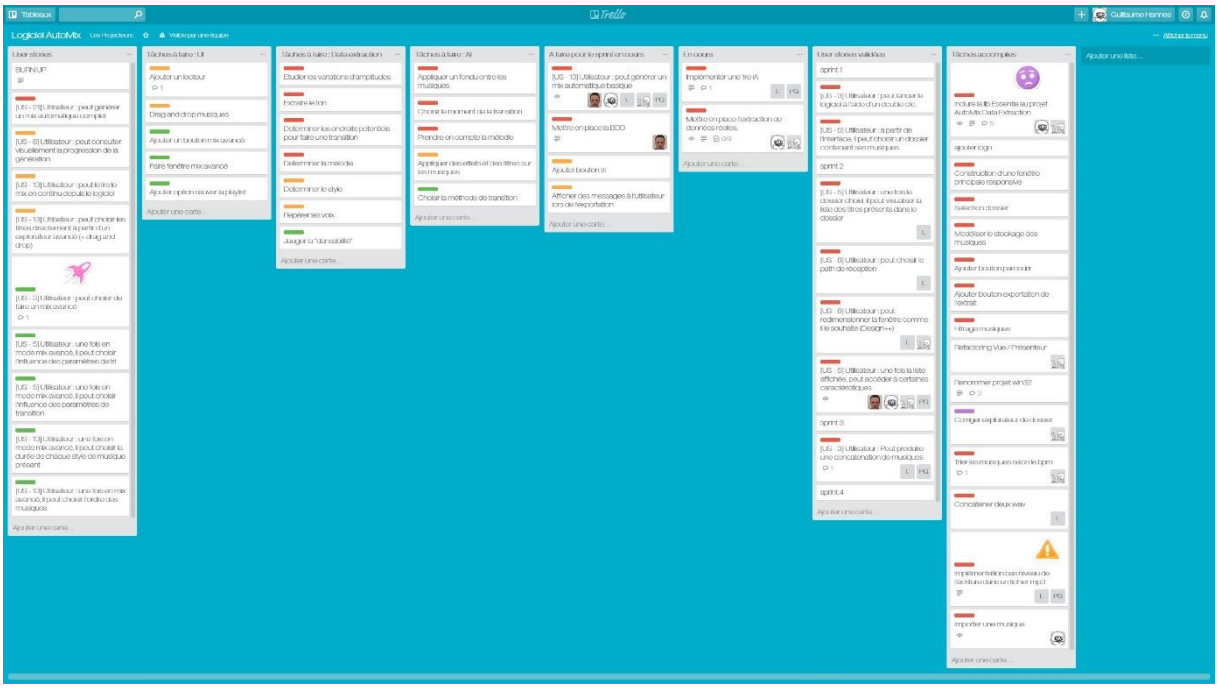

Figure 2 Capture d'écran de notre Trello

Étroitement lié au *Trello*, nous mettons à jour à chaque sprint le *Burn-Up*. C'est un graphique qui nous aide à évaluer notre progression par rapport aux objectifs fixés et dont le fonctionnement est le suivant :

- On affecte à chaque *user story* un certain nombre de points, attribués de manière croissante en fonction de l'importance vis-à-vis du client et de l'utilisateur.
- Ensuite, on trace trois courbes théoriques entre le début du projet, i.e. aucune *user story* de faite, jusqu'au produit minimal, au produit intermédiaire et au très bon produit.
- À chaque réunion de fin de *sprint*, on calcule la somme des *user stories* effectuées puis on place un point sur le graphique.

Lors de la deuxième phase du projet, nous avons édité des statistiques d'utilisation de notre Git afin d'avoir quelques indicateurs supplémentaires de l'avancée du projet. Cependant, ces chiffres sont en partie biaisés car certains membres de l'équipe ont beaucoup travaillé en binôme et ont utilisé la machine la plus performante pour travailler.

Ces outils nous apportent un retour visuel sur nos sprints pour évaluer notre avancement et remédier à d'éventuels problèmes. Nous pouvons également préciser que la plupart des membres de l'équipe utilisent *SourceTree*, distribué par Atlassian, afin de gérer leur dépôt git local et les échanges avec le dépôt distant situé sur le serveur *GitLab* de l'ENSICAEN.

1.3 Répartition du travail

Nous avons consacré beaucoup de temps au projet *Automix Software*, et cela sur toute la durée de l'année. Lors du kick-off meeting nous avions donné une estimation de 315 heures de travail. Aujourd'hui, nous ne sommes pas vraiment capables de donner une estimation du temps passé à la réalisation du projet, mais nous pouvons affirmer que nous avons dépassé la durée initialement prévue.

Dans la partie 1.2 nous avons évoqué les statistiques de l'utilisation de notre Git. Ceux-ci soulignent notamment le fait qu'il y a eu un pic d'activité entre janvier et mars. Cette période correspond à la fin des examens du premier semestre, mais aussi au moment où nous avons contourné les difficultés d'intégrations d'une bibliothèque d'analyse audio[3] qui nous ont empêché d'avancer sur d'autres fonctionnalités telles que l'intelligence artificielle ou l'exportation du mix.

Tous les membres de l'équipe se sont investis pour obtenir le résultat que nous avons aujourd'hui. Les tâches ont été attribuées selon les connaissances et les préférences de chacun, certains développeurs se sont spécialisés dans un domaine, mais les méthodes agiles nous ont permis d'être assez souples vis-à-vis de la répartition des tâches.

Le tableau ci-après représente la réparation des différentes thématiques du projet en fonction des membres de l'équipe. Ce ne sont cependant que des tendances, certaines personnes ont pu se charger du développement de certaines fonctionnalités sortant de leur spécialisation.

	Maxime	Guillaume	Jordan	Louis	Pierre
1re phase	Caractéristiques audio	Caractéristiques audio / Sérialisation	Caractéristiques audio / Stockage de données	Interface graphique	Interface graphique / Intelligence artificielle
2e phase	Interface graphique / Exportation audio	Lecture audio / Sérialisation	Exportation audio	Intelligence artificielle	Intelligence artificielle

Tableau 1 Répartitions des différents sujets

On peut ajouter également les tâches annexes comme le dépôt Git, géré par Guillaume et Maxime, ou encore le site web, développé par Pierre, Guillaume et Maxime, ainsi qu'un installateur mis au point par Guillaume.

[3] Voir partie 2.1

2 Déroulement du projet

2.1 Choix techniques et stratégiques

Notre premier choix portait sur le langage à utiliser pour développer notre logiciel, comme nous l'avons présenté dans la partie 1.2, nous avons opté pour le C++ CLR. Nous avons fait ce choix car nous voulions allier la facilité de développement d'une interface graphique pour le système d'exploitation de Microsoft, tout en utilisant des bibliothèques développées en C++. Cependant la prise en main de l'environnement *Microsoft .NET* n'a pas été évidente. Nous n'avions en effet pas les idées très claires sur les différences entre le C++ managé et le C++ natif, et donc par conséquent, de l'importation de bibliothèques natives dans notre logiciel qui lui tourne sur la machine virtuelle *.NET*.

Nous avons essayé d'intégrer plusieurs bibliothèques audio à notre projet, majoritairement *Essentia*[4] et *Marsyas*[5] mais dans chaque cas nous avons rencontré des problèmes similaires, comme l'impossibilité totale de recompiler les sources nous même pour pouvoir ajouter des fonctions utiles dans la table des fonctions exportées ou corriger de nombreuses erreurs. Pour venir à terme de ces problèmes, nous avons dans un premier temps mis en place différents *mocks*[6] simulant une extraction de données audio. Nous avons ensuite décidé d'utiliser les exécutables en ligne de commandes distribués d'*Essentia* pour récupérer les informations qui nous intéressaient et de le stocker dans une base de données pour accélérer les futures analyses.

Cette nouvelle stratégie à fait disparaitre les contraintes liées au langage de programmation précédemment exposées, d'autre part nous avons appris rapidement que l'interopérabilité des langages offerte par le *.NET Framework* nous aurait permis d'utiliser le C# dès le début du projet. De plus, ce langage est moins verbeux que le C++ managé et offre la possibilité de réaliser des interfaces graphiques qui vont beaucoup plus loin en terme de design et de personnalisation.

Nous avons donc songé sérieusement à changer de langage de programmation à mi-parcours du projet, sans mettre en application cette idée, bien trop coûteuse en temps et posant des questions quant à la stabilité du logiciel. Nous avons donc continué à exploiter l'interopérabilité de deux langages en utilisant notamment une bibliothèque C#, *NAudio*, nous apportant les fonctions nécessaires pour lire et écrire nos fichiers mp3.

[4] Distribuée sous licence GNU Affero General Public License
[5] Distribuée Sous licence GNU General Public License
[6] En programmation orientée objet, les mocks sont des objets simulés qui reproduisent le comportement d'objets réels.

2.2 Git et intégration continue

Nous avons utilisé GitLab comme outil gestionnaire de versions couplé à la stratégie du Git Flow (Figure 3). Il s'agit d'un *workflow* basé sur le développement de fonctionnalités et la structure de ses branches en fait sa particularité.

Git Flow fournit un modèle de ramification défini comme suit :

- La branche *master* contient le code stable délivré.
- La branche *develop* contient le code stable pour l'intégration de fonctionnalités.
- Les branches *feature/** sont utilisées pour coder une fonctionnalité spécifique sur le projet.

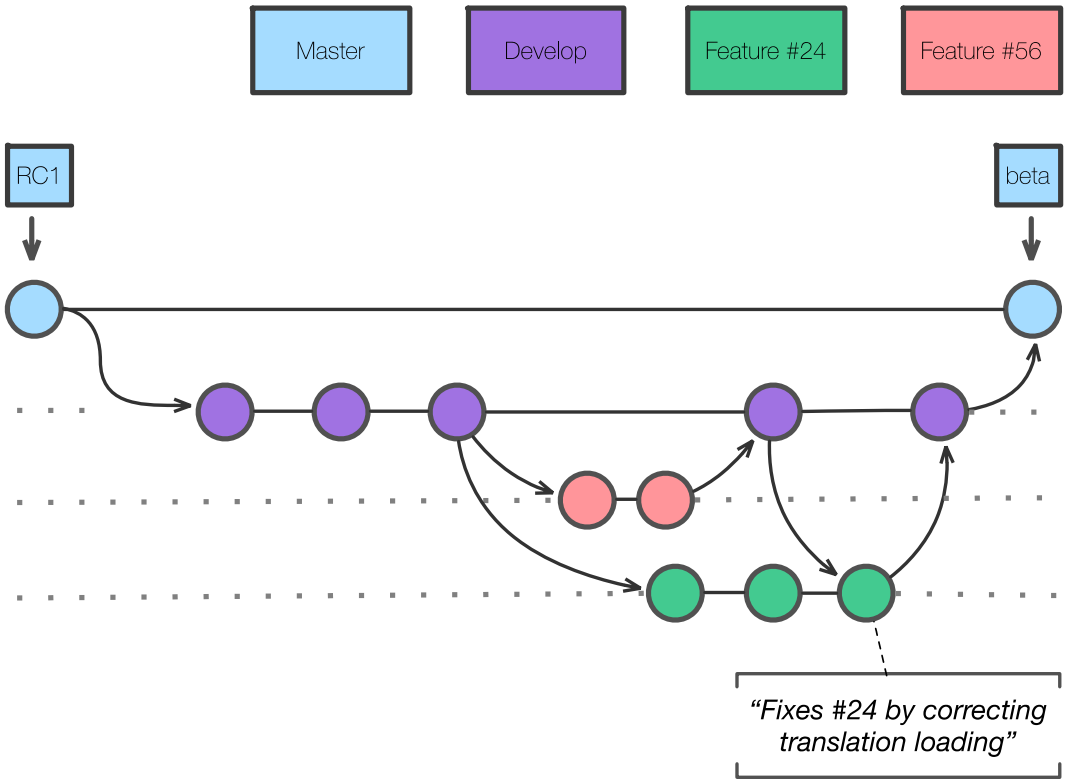

Figure 3 Illustration du Git Flow

Le flux de travail est défini comme suit : un développeur crée une branche de fonctionnalité basée sur le dernier développement, commit son travail, teste son code puis le fusionne dans *develop* s'il s'exécute correctement. Le développeur ne risque pas de faire régresser le développement puisque tout se passe dans sa branche de fonctionnalité aussi longtemps que le développement est dirigé par les tests. Après s'être assuré que sa branche de développement locale est à jour, que tous les conflits sont résolus et que les tests ont bien été exécutés, on peut fusionner dans *develop*.

Pour faciliter le développement de l'application, nous avons choisi de réduire le plus possible les tâches externes au logiciel en les automatisant. En effet, cela a considérablement réduit les retards notamment dus à l'intégration du code, ce qui laisse plus de temps au développement pour produire un code plus propre et plus de fonctionnalités.

Nous avons ainsi délégué les tâches répétitives que sont l'intégration, les *builds* et les tests à un serveur qui les exécute dès que du code est envoyé sur le dépôt : c'est l'intégration continue. Cela centralise le contrôle du code, assure une qualité et une stabilité constantes et moins de travail redondant. Les développeurs obtiennent ainsi un retour visuel rapide sur le déroulement des choses et si un problème survient, la version stable du logiciel sur *develop* reste protégée car le contenu ne pourra pas être fusionné.

Cependant, il ne s'agit là que d'un aperçu de la puissance de l'intégration continue car le serveur GitLab fournit par l'école n'implémente pas toutes les fonctionnalités de *GitLab-CI*, dont la gestion automatique des branches, la livraison et le déploiement de code en continu.

2.3 Développement logiciel

Tout au long du projet, nous avons veillé au respect des principes de génie logiciel et à la propreté du code. Nous avons découpé le logiciel en modules (Figure 4), ce qui est vital pour le développement en équipe, mais aussi pour isoler les composants externes. Nous avons notamment utilisé le patron *procuration* ce qui nous a permis d'employer des *mocks* et de changer l'implémentation de l'extraction de données au cours du projet.

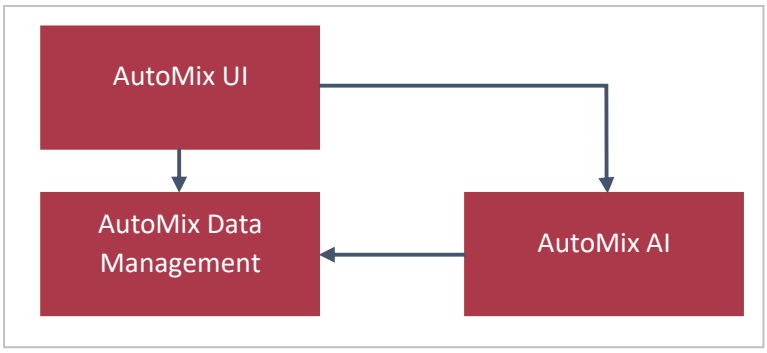

Figure 4 Diagramme de paquet

Nous avons également mis en place des patrons *stratégie* au niveau de l'implémentation des algorithmes d'intelligence artificielle pour pouvoir les manipuler et en changer plus facilement.

3 Rétrospective

3.1 Objectifs réalisés

Nous avons réalisé la plupart des objectifs initialement prévus mais nous avons aussi réalisé des objectifs complémentaires qui sont apparus au cours du projet.

Nous avons dans un premier temps consacré nos efforts sur les *user stories* définissant le produit *minimal*. De ce fait, un logiciel fonctionnel basique a été rapidement réalisé. Il a été ensuite amélioré par le développement des fonctionnalités concernant les produits *moyen* et *idéal*. La Figure 5 illustre l'ensemble des user stories validées durant le projet. On peut noter également que nous avons eu du mal à gérer les transitions, qui ont été terminées à la fin de la période de développement.

Figure 5 User stories réalisées

Afin de distribuer notre produit, nous avons développé un installateur qui permet l'installation très simple du logiciel par n'importe quel utilisateur. De plus, nous avons créé un site web[7] mettant en avant le logiciel et permettant de le télécharger. Ces objectifs supplémentaires sont venus naturellement vers la fin du projet car nous disposions d'une version distribuable.

Au terme de ces deux phases du projet, nous avons réussi à développer un logiciel grand public simple à prendre en main, nous avons également soigné le design de l'interface (Figure 6) en nous appuyant sur les commentaires de quelques personnes.

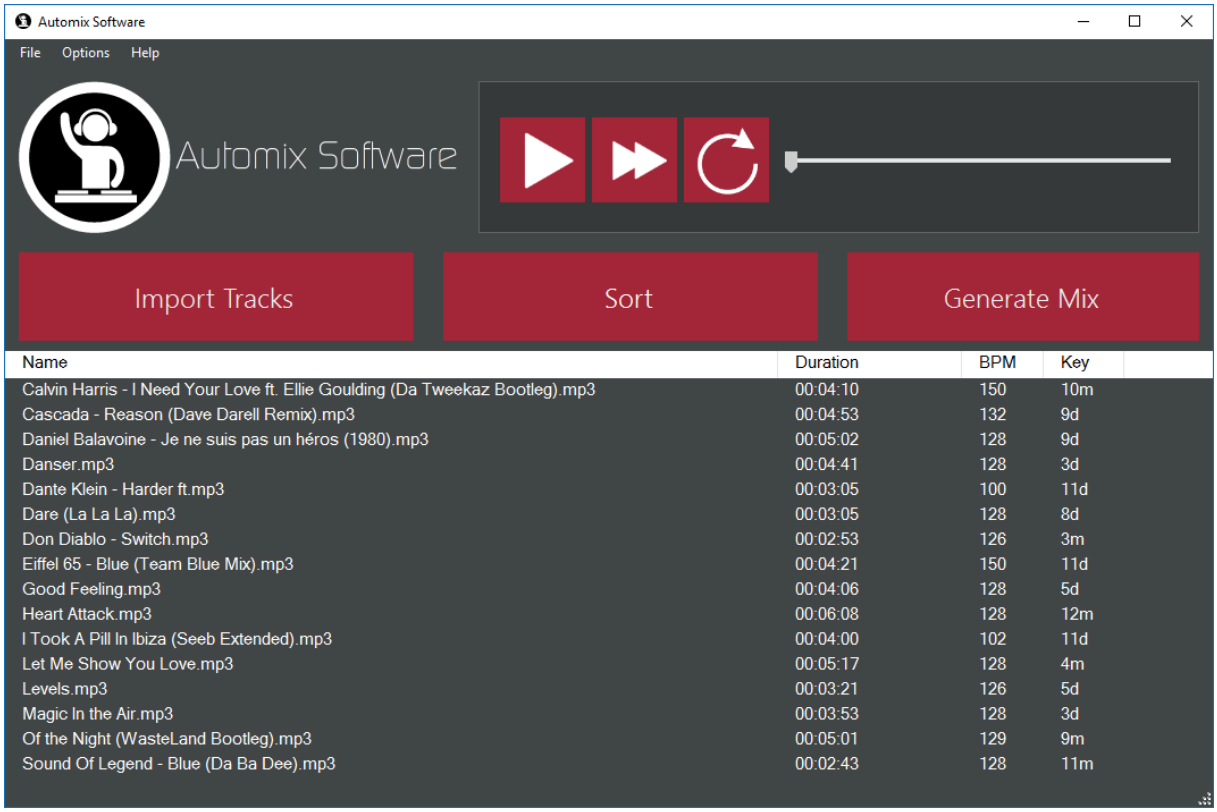

Figure 6 Capture d'écran d'Automix Software RC 1.0

3.2 Objectifs non réalisés

Quelques objectifs du projet n'ont pas été atteints dans la version du logiciel que nous déployons aujourd'hui. En effet, nous n'avons pas réalisé le *mode mix avancé*, ainsi que les fonctionnalités qui en découlent, permettant à l'utilisateur davantage de personnalisation du mix, comme par exemple la possibilité de fixer l'ordre des musiques.

Les *user stories* suivantes n'ont pas été validées par manque de temps de développement :

- En tant qu'utilisateur je peux, une fois en mode mix avancé, choisir la durée de chaque style de musique présent
- En tant qu'utilisateur je peux, une fois en mode mix avancé, influer sur la durée du mix
- En tant qu'utilisateur je peux, une fois en mode mix avancé, paramétrer le tri
- En tant qu'utilisateur je peux, fixer l'ordre des musiques

[7] Disponible à l'adresse suivante : http://www.ecole.ensicaen.fr/~stevenot/automix-software

Ces *user stories* correspondent au produit *idéal* et sont donc moins importantes pour une utilisation satisfaisante du logiciel. Deux à trois sprints supplémentaires auraient permis de terminer l'ensemble des fonctionnalités. Nous avons préféré ne pas entamer cet ensemble qui n'aurait pas pu être abouti dans les temps mais plutôt de développer les deux points annexes liés à la distribution de notre application.

3.3 Bilan technique et personnel du projet

Ce projet nous a apporté énormément tant sur le plan technique qu'organisationnel. Nous avons pris en main le *.NET Framework* et les outils associés, qui ne sont pas enseignés à l'école mais qui sont très utilisés dans le monde professionnel. Cela nous rend aujourd'hui capable de développer complétement une application graphique exploitant l'environnement *.NET*.

Nous avons également été initié à la gestion de projet agile, que nous avons appliqué durant tout le déroulement du projet. Cela nous a appris à avoir une vision beaucoup plus orientée vers les utilisateurs et le produit. Enfin, nous avons appris beaucoup sur l'utilisation d'un serveur d'intégration continue associé à un dépôt Git suivant un *workflow* efficace.

Conclusion

Nous sommes satisfaits d'avoir pu produire un logiciel qui répond aux problématiques initiales et qui satisfait les contraintes que nous avions fixés ainsi qu'une grande partie des objectifs de ce projet. Nous avons rencontré, lors de la première phase du projet, de nombreuses difficultés qui auraient pu avoir un impact négatif encore plus fort sur le logiciel. Nous avons pris des mesures qui ont permis de continuer notre progression, mais on réduit les performances du logiciel en influant sur le temps d'analyse des fichiers mp3. Les cours d'architectures parallèles du second semestre ont apporté de nouvelles pistes pour réduire les temps d'exécutions, et de belles perspectives d'amélioration de notre application.

Les fonctionnalités qui ne sont pas présentes dans la version actuelle du logiciel, ainsi que des idées qui n'ont pas été formalisées et réellement envisagées cette année, offrent-elles aussi des perspectives futures pour ce projet.

Nous avons l'intention de continuer à travailler sur ce logiciel afin de le rendre encore plus robuste et plus rapide tout en envisageant de confier le développement de nouvelles fonctionnalités à une équipe de deuxième année dès l'an prochain.

Remerciements

Tout d'abord nous tenons à remercier nos tuteurs, Estelle Cherrier et Baptiste Hemery, pour leur réactivité et leurs conseils ainsi que pour l'intérêt qu'ils ont porté à notre projet tout au long de cette année. Nous remercions tout particulièrement Sylvain Vernois, qui a apporté son expertise à un moment critique et nous a permis de repartir sur de bonnes bases.

Nous remercions aussi les développeurs des différentes bibliothèques open-source dont nous nous sommes servi à de nombreuses reprises pour développer notre logiciel.

Annexes

Annexe 1: Courbe Burn-Up