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

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 du projet, et nous a permis de repartir sur de bonnes bases.

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

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 continu 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 le 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 mixes intelligent qui permet de tirer avantage des options précédentes sans subir leurs inconvénients. Il permet au grand public de créer des mixes 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.

La version actuelle d'*Automix Software* est la *Release Candidate 1.0.* Disponible en français et en anglais, elle 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.

Dans une première partie nous vous expliquerons les objectifs et le fonctionnement de notre logiciel, ensuite nous détaillerons notre organisation puis le déroulement du projet. Enfin, nous ferons un bilan du projet et du fonctionnement de l'équipe.

[1] *Windows 7* (2009) et plus récent

1 Présentation du projet

1.1 Objectifs

Nous voulions créer un logiciel qui allie la simplicité d'une playlist musicale et le travail d'un DJ permettant aux musiques de s'enchainer sans interruption et de manière agréable. Ce logiciel devait être à la portée du grand public, c'est à dire que les utilisateurs de notre logiciel ne doivent pas avoir besoin de connaissances particulières dans le domaine de la musique ou de l'informatique.

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 contenant le mix. Ce processus est illustré par la Figure 1.

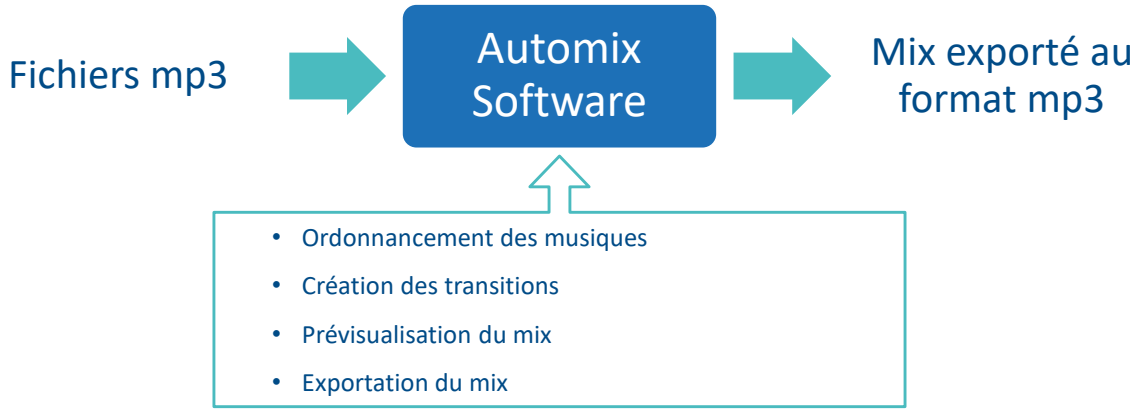

Figure 1 Processus de génération d'un mix audio

Pour répondre à nos exigences, nous avons été amenés à développer une interface simple et intuitive, une intelligence artificielle permettant de déterminer dans quel ordre jouer les musiques ainsi qu'un module d'exportation permettant de créer des transitions entre les musiques et de les exporter au format mp3.

1.2 Fonctionnement

Au moment de l'importation, les musiques sont analysées à l'aide d'*Essentia*[2], une bibliothèque d'extraction et d'analyse de données audio. Nous disposons alors de plusieurs caractéristiques musicales telles que le BPM[3] ou la clé[4].

Ces caractéristiques sont ensuite utilisées par l'intelligence artificielle que nous avons développée afin de trier les musiques précédemment importées. Pour ce faire, nous avons élaboré une distance entre deux pistes, calculée à partir des différentes données normalisées de chacune des pistes. Cette distance représente alors la capacité qu'ont deux musiques à s'enchainer de manière agréable.

[2] Distribuée sous licence GNU Affero General Public License
[3] Le battement par minute (BPM) est une unité pour mesurer le rythme d'une musique
[4] La clé représente la tonalité de la musique

Une fois la liste triée, l'utilisateur peut encore changer l'ordre des musiques, ce qui lui permet d'intégrer ses propres préférences au mix. Un lecteur intégré à l'interface permet à l'utilisateur d'avoir un aperçu du mix.

Si l'utilisateur souhaite conserver le mix créé, il peut l'exporter. Pour cela, nous avons développé un module qui va créer des transitions entre les pistes comme exposé sur la Figure 2. Le volume de la première piste descend progressivement pendant que le volume de la seconde augmente, les deux pistes étant superposées avec leur battement aligné, nous avons un fondu agréable entre ces dernières.

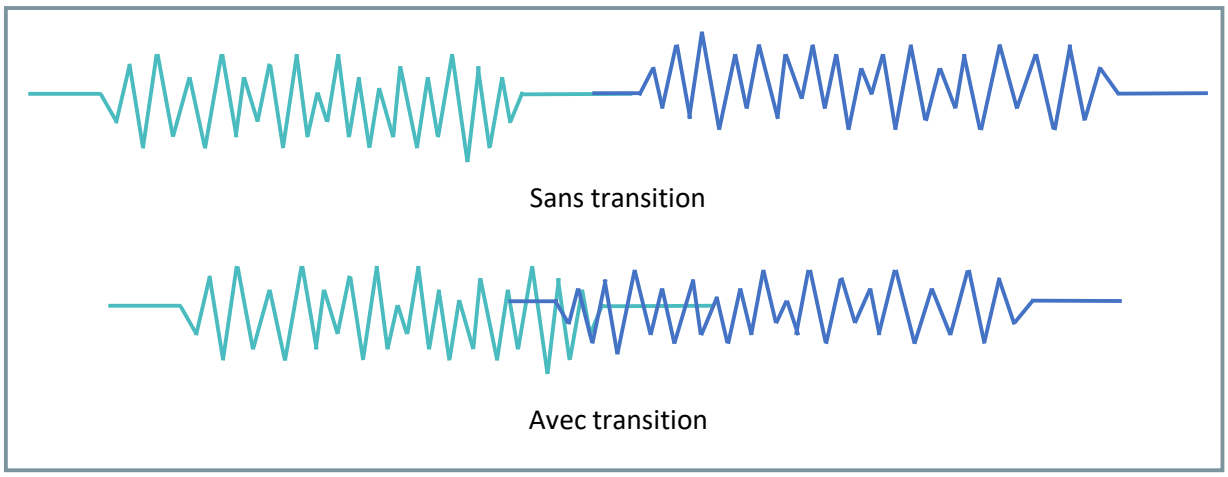

Figure 2 Transitions musicales

1.3 L'intelligence artificielle en détail

Durant la conception de notre intelligence artificielle, nous avons établi un parallèle entre le problème du voyageur de commerce et le tri des musiques. En informatique, le problème du voyageur de commerce est un problème d'optimisation qui, étant donné une liste de villes, et les distances entre toutes les paires de villes, détermine un plus court chemin qui visite chaque ville une et une seule fois en terminant dans la ville de départ.

Nous nous sommes d'abord tournés vers un algorithme génétique ainsi qu'une distance rudimentaire entre deux musiques qui correspond simplement à la différence du BPM entre deux musiques. Mais ces deux choses, l'une comme l'autre, n'ont pas eu les résultats attendus.

Nous avons donc commencé à réfléchir à une distance plus précise. Nous avions extrait un certain nombre de caractéristiques musicales grâce aux exécutables de la bibliothèque audio Essentia, c'est pourquoi nous avons développé une distance qui conserve la différence de BPM mais qui cette fois est pondérée et couplée à d'autres paramètres comme la proximité des clés musicales.

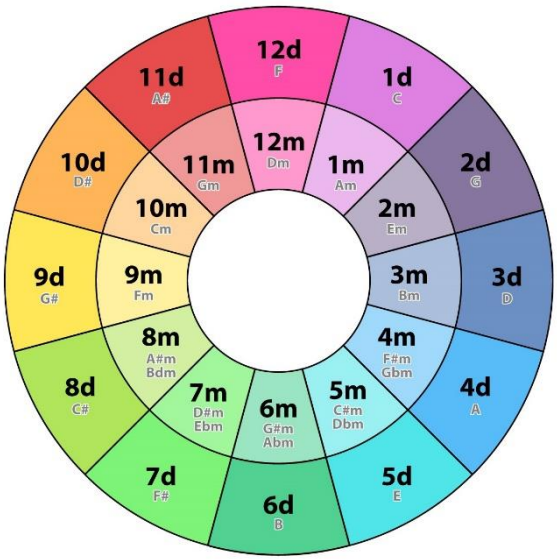

Figure 3 Notation Open Key

La Figure 3 présente le classement en clés des pistes musicales selon une notation particulière, la notation *Open Key.* La lettre *d* ou *m* correspond à la tonalité (respectivement majeure ou mineure) de la musique. Le chiffre associé correspond à une note musicale. Après quelques recherches auprès de sites renseignés, nous avons mis au point une distance fondée sur la théorie du *cycle des quintes* afin de réaliser des transitions agréables.

Le fonctionnement est le suivant : passer de mineur à majeur sans changer la note, ou encore de passer d'une note à une note adjacente sont les meilleurs cas possibles. Par exemple, passer de *5d* à *12m* est probablement inadapté à une transition. En revanche passer de *5d* à *4d* produira un meilleur résultat.

Nous avons également ajouté dans le calcul de la distance une donnée calculée par la bibliothèque audio et appelée *danceability*[5]. Le poids que nous lui associons est faible par rapport aux deux autres caractéristiques.

Aujourd'hui, notre intelligence artificielle exploite un algorithme de recuit simulé. Cet algorithme se base sur une notion de thermodynamique et plus précisément de métallurgie. En effet, le recuit est une opération consistant à laisser refroidir lentement un métal pour améliorer ses qualités. L'idée physique est qu'un refroidissement trop brutal peut bloquer le métal dans un état peu favorable alors qu'un refroidissement lent permettra aux molécules de s'agencer au mieux dans une configuration stable. C'est cette même idée qui est à la base de l'algorithme de recuit simulé. Pour éviter que l'algorithme ne reste piégé dans des minimums locaux, on fait en sorte que la température *T* (de type flottant) décroisse lentement en fonction du temps.

Dans cet algorithme, nous démarrons à une température T_0 et nous la faisons baisser jusqu'à une température T_{min} pas à pas. A chaque pas, nous intervertissons deux chansons au hasard :

- Si cela améliore le résultat alors nous confirmons l'inversion
- Sinon nous le faisons avec une probabilité valant $e^{\frac{-\Delta}{T}}$ où Δ est la variation apportée.

Nous faisons cela à plusieurs reprises (environ le nombre total de musiques dans la liste) à chaque pas. Quand la température atteint T_{min}, nous arrêtons l'algorithme et nous renvoyons la liste ainsi triée.

[5] Evaluée entre 0 et 3

2 Organisation

2.1 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 cible 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 produit. 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, nous traçons une courbe *Burn-Up*[6] dont le principe est expliqué dans la partie 2.2. 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.

2.2 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.

Pour organiser nos *user stories* et nos tâches, nous utilisons *Trello*, 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.

[6] Voir Annexe 1: Courbe Burn-Up

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 cible et au produit idéal.
- 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 dépôt 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*[7] afin de gérer leur dépôt Git local et les échanges avec le dépôt distant situé sur le serveur *GitLab* de l'ENSICAEN.

[7] GUI pour Git, distribué par *Atlassian* sous Windows.

2.3 Répartition du travail

Nous avions estimé le temps de travail à 315h. Aujourd'hui nous avons largement dépassé cette estimation.

Dans la partie 2.2, 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[8]. Ces difficultés 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 1 représente la répartition 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 en dehors de leur spécialisation.

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

Tableau 1 Répartition des différents thèmes

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

[8] Voir partie 3.1

3 Déroulement du projet

3.1 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 2.2, 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*[9] et *Marsyas*[10] 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*[11] 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 a fait disparaître 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.

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

3.2 Git et intégration continue

Nous avons utilisé GitLab comme outil gestionnaire de versions couplé à la stratégie du Git Flow (Figure 4). 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.

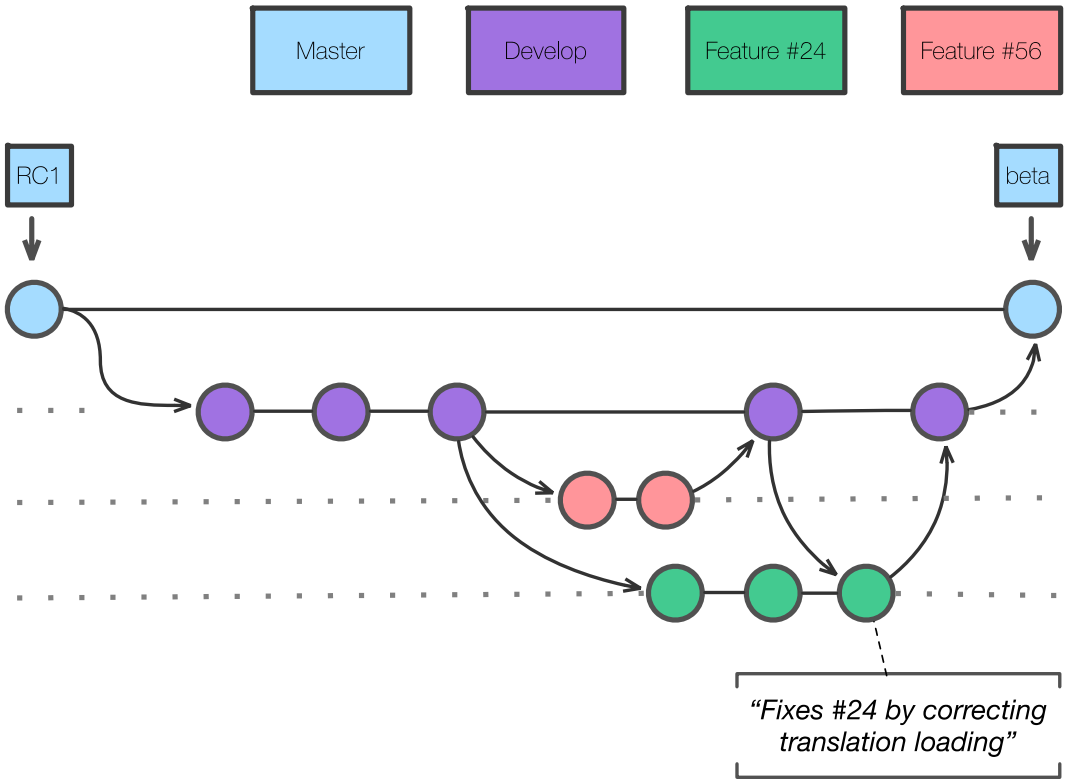

Figure 4 Illustration du Git Flow

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.

3.3 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 5), 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.

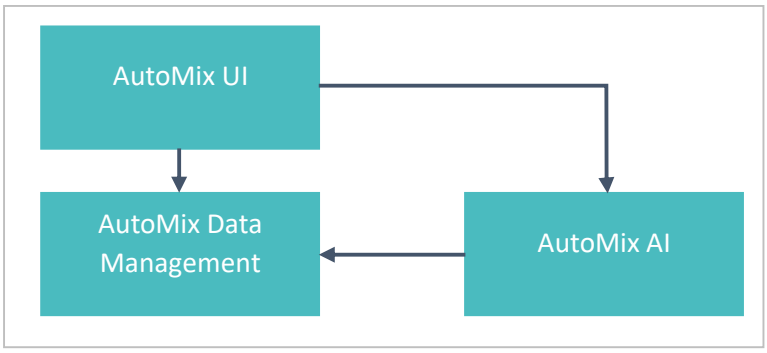

Figure 5 Diagramme de paquet

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.

4 Bilan du projet

4.1 Objectifs réalisés

La version actuelle de notre logiciel embarque la totalité des fonctionnalités du produit cible, seules quelques fonctionnalités correspondant au produit idéal non pas été réalisées.

Nous avons dans un premier temps concentré 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 *cible* et *idéal*. La Figure 6 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 6 User stories réalisées

4.2 Résultats obtenus

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 7) en nous appuyant sur les commentaires de quelques personnes. Les caractéristiques les plus simples apparaissent une fois la musique importée, le tri automatique modifie l'ordre des musiques, l'exportation au format mp3 prend en compte les modifications de l'utilisateur.

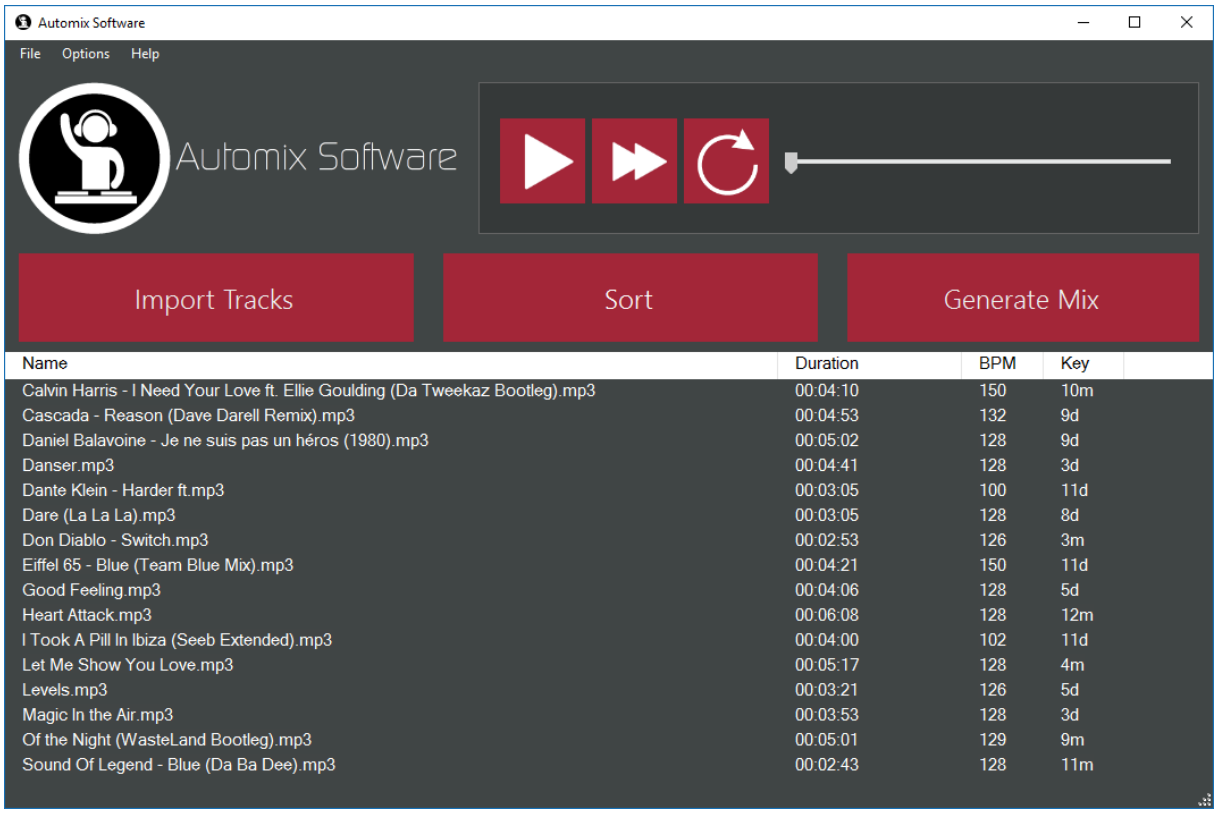

Figure 7 Capture d'écran d'Automix Software RC 1.0

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[12] (Figure 8) mettant en avant le logiciel et permettant de le télécharger. Ces travaux supplémentaires sont venus naturellement vers la fin du projet car nous disposions d'une version distribuable.

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

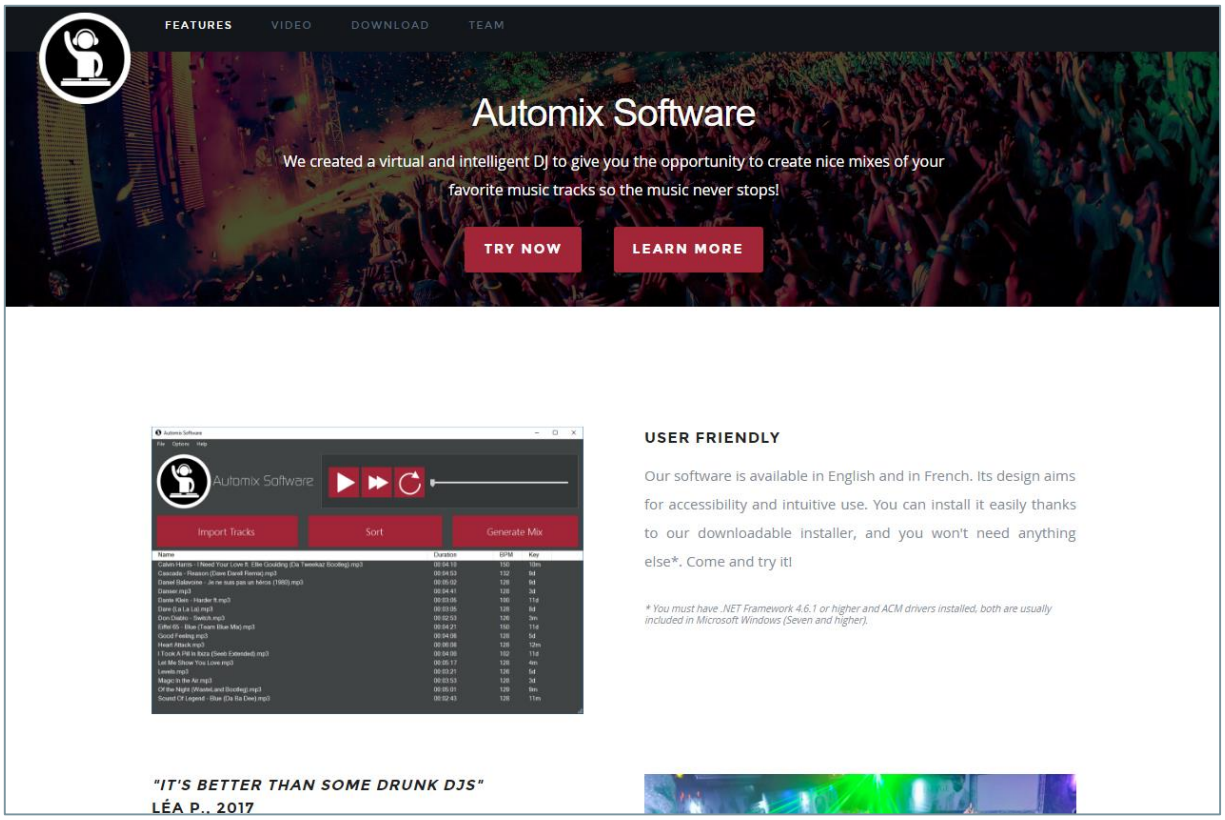

Figure 8 Capture du site web d'Automix Software

4.3 Perspectives

L'objectif du produit idéal correspondant au mode *mix avancé* n'a pas été atteint, en effet, nous n'avons pas réalisé les user stories et les fonctionnalités qui en découlent. Ce mode *mix avancé* pourrait permettre à l'utilisateur de personnaliser davantage ses mixes, par exemple en offrant la possibilité de fixer l'ordre des musiques.

Nous avons choisi de nous consacrer à la distribution de notre logiciel en réalisant le site web et l'installateur, qui rendent le logiciel facile d'accès pour le grand public, plutôt qu'à des fonctionnalités optionnelles. En effet, dans sa version actuelle, le logiciel fonctionne bien et les fonctionnalités qui ne sont pas présentes, ainsi que des idées qui n'ont pas été envisagées cette année, offrent des perspectives intéressantes et des pistes d'amélioration pour ce projet.

De plus, les problèmes majeurs que nous avons rencontrés lors de la première phase du projet auraient pu avoir un impact négatif bien plus fort sur le logiciel. Nous avons pris des mesures palliatives qui ont permis de continuer notre progression, mais ont réduit les performances du logiciel. Les cours d'architectures parallèles du second semestre ont apporté de nouvelles pistes pour réduire les temps d'exécutions, et apporter une réponse plus pérenne à ces problématiques.

4.4 Rétrospective

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.

Les séances de travail intensif que nous avons effectuées, toute l'équipe réunie, nous ont permis de débloquer certains points et de faire de grands pas dans le logiciel et nous souhaiterions par conséquent rendre ce genre de sessions plus fréquentes.

Outre ces deux points, nous pouvons faire une rétrospective sur l'équipe et le projet en suivant la technique agile de *Start, Stop, Continue* comme schématisée sur la Figure 9.

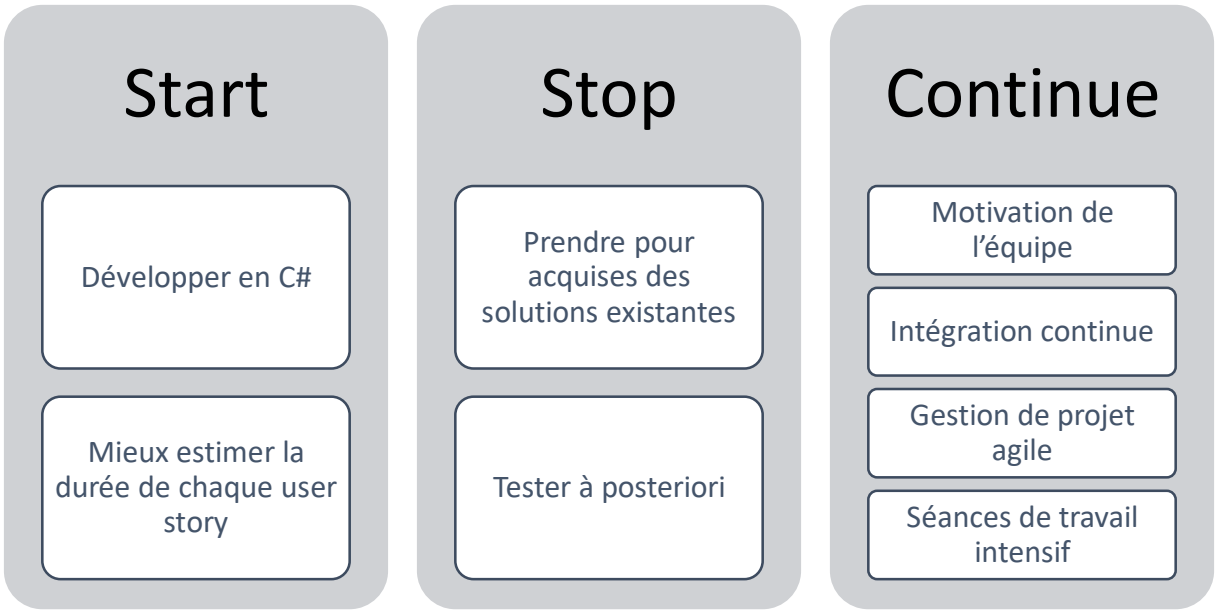

Figure 9 Schéma Start, Stop, Continue

Conclusion

Nous sommes satisfaits d'avoir pu produire un logiciel dans sa totalité, qui répond aux problématiques initiales et aux contraintes que nous avions fixées par la réalisation d'une grande partie des objectifs. Nous sommes fiers de pouvoir proposer au public le téléchargement de notre logiciel dans une version stable et fonctionnelle sur notre site web.

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 capables de développer complétement une application graphique exploitant l'environnement *.NET*.

Nous avons l'intention de continuer à travailler sur ce projet afin de le rendre encore plus robuste et plus rapide. Il nous reste beaucoup d'idées pour faire évoluer Automix Software et nous envisageons de confier le développement de nouvelles fonctionnalités à une équipe d'élèves de deuxième année dès l'an prochain.

Annexes

Annexe 1: Courbe Burn-Up