Projet Long TOB

Rapport de l'itération 1

Équipe KL1 : Baptiste CEBULA - Cyrian RAGOT - Julien GAUTHIER - Léandre LE NULZEC - Marwane KARAOUI - Sami MOSTFA - Nell TRUONG - Victor BARILLY

26 avril 2024

Objectif du document :

Ce document a pour but de présenter l'avancement de notre équipe au cours de la première itération de notre projet long de Technologie Objet.

Nous détaillerons au passage les difficultés que nous avons rencontrées, comment nous les avons surmontées et notre plan pour la deuxième itération.

Sommaire

1 Introduction

Lors de cette première itération, nous avons, dans un premier temps, mis en place nos moyens de communication et de travail en équipe. Nous avons aussi initié le projet avec des outils tels que Gradle et GitHub puis nous avons choisit les librairies que nous allions utiliser.

Il nous a fallu un temps de mise en route pour lire la documentation des librairies, suivre des tutoriel et comprendre comment se répartir le travail car il nous a été difficile de ne pas se "marcher dessus" pour débuter le projet.

Pour ce qui est du développement concret, nous avions comme objectif d'avoir une première fenêtre visuelle avec quelques éléments de la barre d'outils mais aussi de développer la vue de composition avec, en particulier, un piano roll fonctionnel.

La suite de ce rapport explique en détail l'avancement actuel de ces objectifs.

2 Mise en place des méthodes agiles et outils de développement

2.1 Communication

Nous avons choisit de communiquer nos informations par un serveur discord qui nous permet d'avoir un salon global que toute l'équipe peut suivre et un salon dans lequel nous postons des liens de ressources ou fichiers partagés. Discord nous permet aussi de communiquer en messages privés, cela est utile lorsque deux personnes travaillent sur le même composant.

Nous essayons aussi d'utiliser CryptPad pour créer des diagrammes SysML de manière collaborative et Overleaf pour écrire nos documents partagés.

De plus, le README écrit sur GiHub permet aux développeurs d'avoir des informations sur le projet.

La répartition des tâches s'est faite à la fois via discord mais aussi par l'utilisation de Trello où nous nous désignons une tâche précise et nous indiquons si la tâche est "A faire", "En cours" ou "Finie".

2.2 Repartition des tâches

Comme dit précédemment, la répartition des rôles s'est faite via discord et Trello. Nous avons fait le choix de séparer le travail selon le visuel et le traitement des données. En effet, en suivant le patron de conception Modèle-Vue-Controlleur, il est plus simple de définir des séparer des tâches qui correspondent plus à la Vue et des tâches qui correspondent plus au Modèle. Ainsi, certains membres du groupe se sont plutôt penchés sur le visuel et donc l'utilisation de JavaFX tandis que d'autres se sont penchés sur le traitement des données en arrière plan avec notamment l'utilisation de JavaSound. Il était ainsi plus simple de constituer des sous-groupe qui ont travaillés sur des points similaires en communiquant par messages privés.

2.3 Environnement de développement

En premier lieu, pour un meilleur contrôle sur notre projet et l'utilisation de git nous avons choisit de créer un repository GitHub [1] sur lequel nous avons mis en place des règles proposées par GitHub qui permettent de maintenir une certaine stabilité du projet malgrè la présence de plusieurs développeurs. Nous travaillons alors tous sur une branche autre que la branche principale, cette dernière devant rester "propre". Afin de le garantir, nous avons créé une GitHub Action qui s'occupe de compiler le projet et lancer les tests lorsqu'une demande de merge est faite vers la branche principale. GitHub empêche alors toute demande de merge si la branche ne compile pas ou si des tests unitaires ne passent pas.

Aussi, nous pourrions utiliser la demande de review pour qu'une branche puisse être merge mais pour le début du projet cette option a été désactivée pour permettre une avancée rapide.

Pour la gestion des dépendances et faciliter le développement de l'application, nous avons fait le choix d'utiliser l'outil Gradle. Pour faciliter la portabilité d'un ordinateur à un autre et d'un IDE à un autre, nous utilisons une version de Gradle interne au projet grâce à Gradle Wrapper.

Dans un soucis de satisfaire les besoins de tous les membres du groupe, notre projet fonctionne sous Gradle 5.6 et Java 11 qui sont compatibles et qui sont installés sur les ordinateurs de l'école.

Lors de cette première itération nous avons alors dû nous habituer à l'utilisation de tous ces outils.

1. https://github.com/cyrianR/poo-long-project-n7/tree/main

3 Traitement audio

3.1 Préambule

L'API de traitement audio Java Sound permet de contrôler l'entrée et la sortie de médias sonores, comprenant à la fois des données audio et des données MIDI (Musical Instrument Digital Interface). Dès lors, nous avons utilisé cette API pour ce deuxième aspect, afin de lire puis interpréter les fichiers MIDI provenant d'instruments électroniques par un séquenceur, comme indiqué par le schema ci-dessous.

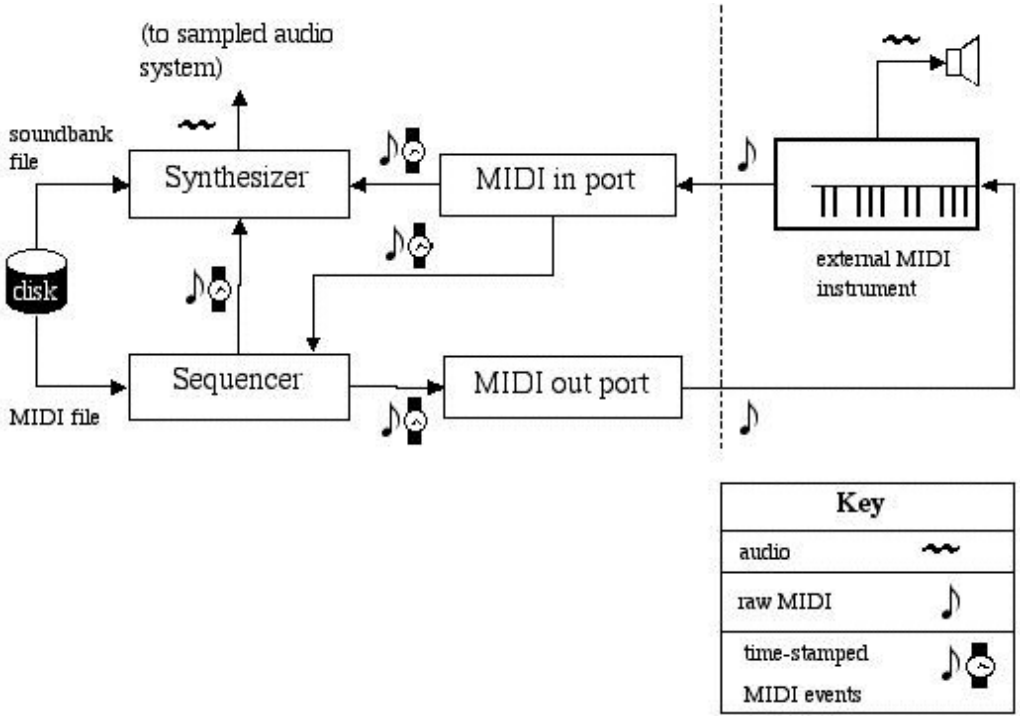

FIGURE 1 – Exemple de configuration globale

C'est ainsi que, si l'on souhaite reproduire la configuration ci-dessus, nous pourrons faire que le séquenceur envoie les messages MIDI à un appareil tel qu'un synthétiseur qui pourra émuler les sons de certains instruments de musique en lisant un fichier de banque de sons. L'API Java Sound nous permettra ainsi de contrôler la lecture du séquenceur, comme en la démarrant ou en la stoppant.

3.2 Traitement d'une note

Afin de représenter au mieux une note de musique, qui peut s'avérer être une information complexe, nous utilisons d'abord des caractéristiques telles que l'octave et l'indice de note. En effet, l'octave indique la plage de fréquences à laquelle une note appartient, tandis que l'indice de note, allant de 0 à 11 pour représenter les notes de Do à Si (ou de C à B dans le système Anglo-Saxon) avec les éventuels demi-tons (Do dièse/C# par exemple), spécifie la hauteur exacte de la note à l'intérieur de cette plage.

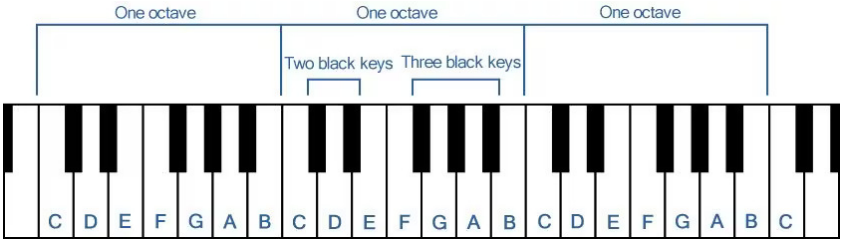

FIGURE 2 – Les notes sur un clavier

Pour traduire ces informations en une forme numérique utilisée couramment dans la musique numérique,

nous utilisons le numéro de note MIDI, calculé par la formule :

$$midiNoteNumber = 12 * octave + noteIndex \tag{1}$$

La représentation d'une note comprend également des aspects tels que la vélocité (l'intensité avec laquelle la note est jouée), et la durée (la longueur temporelle pendant laquelle la note est maintenue).

Nous avons choisi d'utiliser des impulsions par quart de note (PPQ) comme unité de durée. Plus ce nombre PPQ est élevé, plus la séquence sonore est réaliste. En effet, pour par exemple un tempo de 60 battements par minute : avec un PPQ de 10, il n'y aurait que 10 divisions par seconde, limitant la précision du timing de chaque note. C'est donc le nombre de ces impulsions par quart de note qui détermine la taille d'un tick (impulsion individuelle).

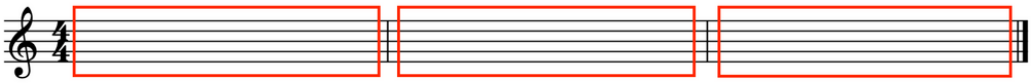

FIGURE 3 – Les mesures en rouge - signature temporelle de 4/4 (4 notes par barre)

Dès lors, pour tester le bon fonctionnement de la classe Note, il faut s'assurer que les cas limites ne sont pas dépassés. En effet une note doit respecter des conventions tels qu'une gamme de 11 octaves, qui couvre une suffisamment vaste étendue de fréquences audibles, et un indice de note compris entre 0 et 11 permettant qu'une note ne correspond qu'à un unique couple (indice, octave). C'est ainsi que l'on associe un une numéro de note MIDI comme expliqué précedemment avec (1).

Octave	Note Numbers											
	C	C#	D	D#	E	F	F#	G	G#	A	A#	B
-2	0	1	2	3	4	5	6	7	8	9	10	11
-1	12	13	14	15	16	17	18	19	20	21	22	23
0	24	25	26	27	28	29	30	31	32	33	34	35
1	36	37	38	39	40	41	42	43	44	45	46	47
2	48	49	50	51	52	53	54	55	56	57	58	59
3	60	61	62	63	64	65	66	67	68	69	70	71
4	72	73	74	75	76	77	78	79	80	81	82	83
5	84	85	86	87	88	89	90	91	92	93	94	95
6	96	97	98	99	100	101	102	103	104	105	106	107
7	108	109	110	111	112	113	114	115	116	117	118	119
8	120	121	122	123	124	125	126	127				

FIGURE 4 – Table des numéros de note MIDI

Une fois que nous savons comment implémenter une classe Note, la plupart des tests de validité consistent à vérifier que des notes ne vérifiant pas ces conditions aux limites lèvent l'exception *InvalidNoteException*.

3.3 Traitement d'une séquence de notes

Afin de créer des mélodies, il faut passer d'une note à une séquence de notes, et pour cela, l'implantation de la notion de pattern est nécessaire. Un pattern est une structure de données représentant donc une séquence d'événements MIDI, tels que le début ou la fin d'une note. Dès lors, nos avons créé une classe Pattern afin de stocker et manipuler des informations MIDI, avec donc par exemple la possibilité d'y ajouter ou de supprimer une note, dans le but de produire des mélodies suivant un rythme cohérent.

Pour vérifier le bon fonctionnement de la classe Pattern, il fallait tout d'abord vérifier si elle permettait bien d'y ajouter des notes. Nous avons créé des fichiers de tests pour vérifier si ajouter une note a une influence sur la longueur du pattern. En effet ajouter simultanément deux notes à la fin du pattern devrait agrandir sa longueur

de la durée de la note la plus longue, de même pour une note de même durée après une autre, ou au contraire deux notes de même durée ne devraient pas changer la taille du pattern s'il se terminait déjà bien plus longtemps.

De plus, il faut vérifier si les tracks d'un pattern dans lequel on ajoute plusieurs notes contiennent uniquement les évènements MIDI de leur notes associées (ou non pour celles qui ont été supprimées). Pour ceci, nous avons crée un pattern dans lequel nous ajoutons plusieurs notes puis nous vérifions si les évènements MIDI de chaque track correspondent bien aux notes ajoutées et qu'il n'y a pas de perte ni d'ajout d'information.

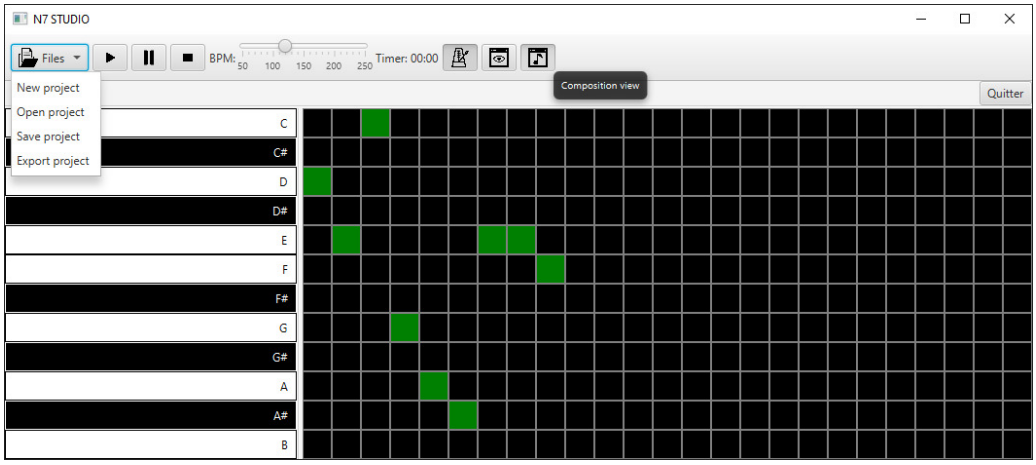

FIGURE 5 – Exemple de rendu d'un pattern

Enfin, il était nécessaire d'ajouter des tests vérifiables "manuellement" afin de vérifier si la classe Pattern permet bel et bien de sauvegarder et de lire des séquences de notes sauvegardées dans des fichiers MIDI. Pour ceci, nous avons reproduit la mélodie d'une musique connue que nous avons sauvegardé dans un fichier test.mid pour ensuite la charger et la lire. Nous avons donc initialisé un pattern vide dans lequel on a ajouté toutes les notes avec leur hauteur, durée, vélocité et date de commencement, ainsi que d'autres notes plus basses en même temps pour reproduire quelques accords. Nous avons ensuite sauvegardé puis écouté la mélodie en initialisant un second pattern avec le fichier MIDI créé afin d'affirmer le bon fonctionnement de la sauvegarde.

4 Interface utilisateur

4.1 Préambule

Nous avons choisi de réaliser l'interface de notre application à l'aide de la bibliothèque JavaFX et non Swing comme nous avons pu faire en cours et en TP. Nous avons fait ce choix car JavaFX nous semblait plus facile à prendre en main et nous offrait plus de possibilités de développement (même si nous ne les avons pas encore utilisées sur cette itération).

4.2 Barre d'outils

Afin de réaliser un tel logiciel, il est essentiel que l'utilisateur dispose d'un interface graphique afin de pouvoir de pouvoir facilement composer de la musique sur son ordinateur. La barre d'outils est la base de cette interface graphique. Elle a pour but de mettre à disposition de l'utilisateur tous les boutons permettant de réaliser les actions de base de notre logiciel. Ici, il ne s'agit pas de boutons permettant directement la composition d'un morceau de musique, mais plutôt de boutons permattant de gérer le projet dans sa globalité, la lecture d'un morceau ou le type d'affichage de l'application. Voici la liste exacte de ces boutons et leurs utilités respectives :

— *File* : menu déroulant dont l'utilité sera précisée plus tard
— *Play* : permet de lancer la lecture d'un morceau
— *Pause* : permet de pauser la lecture d'un morceau
— *Stop* : permet d'arrêter la lecture d'un morceau
— *BPM* : permet d'ajuster le nombre de battements par minute du morceau
— *Timer* : affiche le temps de la lecture du morceau

— *Metronome* : permet d'activer ou de désactiver le métronome sur le morceau
— *Overview* : permet d'accéder à la vue générale de l'application
— *Composition view* : permet d'accéder au piano roll de l'application.

Le menu déroulant du bouton *File* contient les entrées suivantes :
— Create a new project : permet de créer un nouveau projet musical
— Open an existing project : permet d'ouvrir un projet déjà créé
— Save the current project : permet de sauvegarder le projet courant
— Export the current project : permet d'exporter le projet courant

Voici une capture d'écran de la barre d'outils que nous avons réussi à créer :

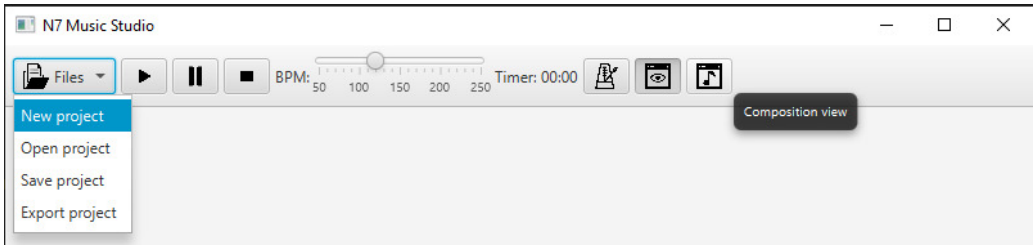

FIGURE 6 – Aperçu de la barre d'outils

Nous avons décidé d'utiliser des icônes pour nos boutons pour des raisons esthétiques. Les boutons ne sont actuellement pas tous fonctionnels. Nous programmerons plus tard les actions correspondant aux différents boutons. Nous avons cependant choisi d'adopter une architercture en Modèle-Vue-Contrôleur (MVC) et améliorer la clarté de notre code.

4.3 Piano roll

Le piano roll est un élément de base de la compostion musicale. Il permet à l'utilisateur de l'application de définir des suites de notes et/ou d'accords composant le morceau. L'utilisateur choisit également l'octave des notes (pour les rendre plus aigues ou plus graves), la vélocité des notes (la vitesse à laquelle elles sont jouées) et l'instrument qui les joue.

Afin de réaliser le piano roll de notre application nous nous sommes inspirés de d'autres applications de composition musicale. Cependant, nous n'avons pas complètement fini l'interface du piano roll au cours de cette itération. Les fonctionnalités de changement d'octave, d'instrument et de vélocité n'ont pas encore été implémentées. De même, une partie des fonctionalités de manipulation des notes que nous souhaitions implémenter ne sont pour l'instant pas disponibles.

Voici un aperçu du piano roll :

5 Conclusion

5.1 Difficultés

Le première difficulté que nous avons rencontrée a été de devoir lire la documentation de l'API JavaSound. En effet, le traitement du son est un domaine très particulier que nous n'avions pas vu en TP. Cela a eu pour effet de nous faire perdre du temps, surtout que nous ne comprenions pas forcment cette documentation tous de la même manière.

La principale difficulté que notre groupe a rencontrée durant cette itération est la communication interne. En effet, les deux semaines de travail se déroulant pendant les vacances ont été moins productives que la dernière semaine que nous avons passée à l'ENSEEIHT. Nous avons eu des difficultés à mettre en place notre environnement de travail et à nous répartir correctement les tâches entre nous. Le fait de ne pas pouvoir nous retrouver a grandement complexifié l'entraide entre les membres de l'équipe et la communication sur ce que nous étions en

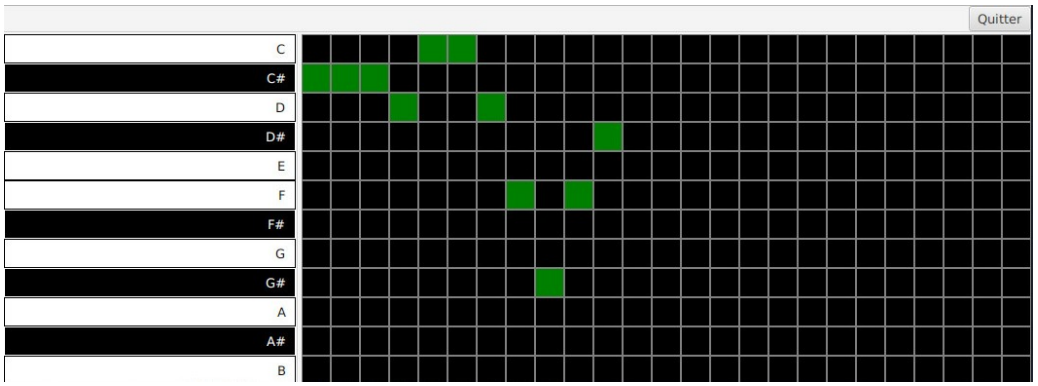

FIGURE 7 – Aperçu du piano roll

train de développer. Ainsi nous avons développé certaines fonctionalités deux fois, car nous ne savions pas que plusieurs personnes s'en occupaient. Cependant, ces problèmes se sont progressivement réglés à notre retour à l'ENSEEIHT et nous sommes donc plus optimistes à ce sujet pour les itérations suivantes.

Nous avons également eu du mal à tous comprendre de manière précise ce que nous devions développer. En effet, n'étant pas tous familiers avec les applications de composition musicales nous ne connaissions pas tous le principe de certains composants comme le piano roll par exemple. Nous avons essayé de traiter ce problèmes en rédigeant des spécifications (sur le modèle de ce qui peut être fait en entreprise) qui guident de la manière la plus précise possible les personnes qui développent. Actuellement, nous n'avons pas encore pu évaluer l'efficacité de cette solution mais nous le ferons au cours de la prochaine itération.

Enfin, nous avons eu quelques difficultés dans la manipulation de l'outil Git. En effet, même si nous avions tous l'habitude d'utiliser celui-ci nous ne nous étions jamais retrouvés à développer à autant de personnes sur un unique projet. Nous avons donc dû apprendre à nous servir de système de branche afin de pouvoir travailler de manière efficace.

5.2 Projections concernant l'itération 2

Nous souhaiterions au cours de la deuxième itération du projet long terminer la conception du piano roll. En effet, nous avons déjà bien avancé celle-ci. Il nous reste donc à terminer les derniers détails et s'assurer d'un intégration correcte de tous les éléments du piano roll. Nous souhaiterions également débuter deux autres parties imporante de notre application que sont la gestion de projet et de fichiers ainsi que le système de playlist.