Université de Lille, Cité Scientifique, 59650 Villeneuve-d'Ascq

Licence 3 Informatique parcours Info

Module de Musique Live en Pharo

Rapport de stage

par

Antoine Delaby

Encadrant entreprise : Santiago Bragagnolo

Encadrant universitaire : Clément Quinton

11 avril 2022 – 3 juillet 2022

INRIA, Institut national de recherche en sciences et technologies du numérique

Parc scientifique de la Haute-Borne, 40 Av. Halley Bât A,

59650 Villeneuve-d'Ascq

Remerciements

Je remercie tout d'abord l'équipe de recherche RMOD de l'INRIA de m'avoir accueilli dans leurs locaux au cours de mon stage, de m'avoir accompagné tout au long de celui-ci et de m'avoir apporté leur aide lorsque j'en avais besoin.

Je remercie notamment Monsieur Domenico Cipriani pour le projet musical qui a fait l'objet de mon stage et pour les notions musicales que j'ai pu apprendre et que j'ai pu mettre en œuvre au cours de celui-ci.

Je remercie tout particulièrement mon encadrant professionnel Monsieur Santiago Bragagnolo pour son soutien, son aide et ses propositions tout au long de mon stage. Cela m'a permis d'avancer tout en faisant des découvertes sur le plan musical comme sur le plan informatique.

Je remercie aussi Monsieur Santiago Bragagnolo et mon encadrant de stage universitaire Monsieur Clément Quinton pour leur suivi tout au long du stage.

De même, je remercie l'ensemble des autres stagiaires présents dans l'équipe RMOD lors de mon stage, Gaylord Delporte, Remi Dufloer, Aboubacar Diawara, Abir Bezzazi, Ibrahima-Sambegou Diallo, Adrien Vanegue et Romain Degrave pour les échanges et réflexions que nous avons pu avoir en rapport avec mon sujet de stage tout comme avec le leur, me permettant d'avancer et d'apprendre sur le langage Pharo.

Enfin, je remercie mes parents pour leur écoute, leur soutien et leurs conseils au cours de cette période de stage.

Résumé de stage

Dans le cadre de ma formation en Licence 3 Informatique, j'ai effectué un stage d'une durée de douze semaines au sein de l'équipe RMOD des laboratoires nationaux de recherche en informatique INRIA à Villeneuve-d'Ascq. L'équipe RMOD travaille essentiellement sur la partie Génie Logiciel de l'informatique et programme en langage Pharo, un langage entièrement orienté objet.

Au cours de ce stage, j'ai participé à un tout nouveau projet visant à concevoir un nouvel environnement musical en Pharo pour le DJ Domenico Cipriani. Le but de ce projet était dans un premier temps d'implémenter le protocole MIDI dans Pharo. C'est un protocole de communication qui modélise la musique pour les ordinateurs et instruments électroniques sous forme de messages contenant des informations telles que la note jouée et avec quelle intensité, ou le type d'action musicale effectuée.

Pour le protocole MIDI, nous avons décidé d'utiliser une librairie déjà existante en C nommée PortMidi et d'y faire appel dans Pharo. J'ai donc dans un premier temps développé un exemple simple en langage C qui utilise cette librairie afin d'en comprendre le fonctionnement. Une fois cet exemple fonctionnel, je l'ai réimplémenté en Pharo. Pour utiliser la librairie C, j'ai repris un paquet Pharo nommé uFFI qui permet de faire appel à des fonctions externes en langage C depuis le code Pharo.

À cette étape du projet, j'avais un exemple simple d'implémentation des connexions MIDI en Pharo avec un envoi de messages (un ou plusieurs messages) et une réception de messages. Les prochaines étapes ont été de faire l'envoi des suites de messages, donc des suites de notes musicales que j'ai appelé « partition d'instrument » (il faut ici imaginer un instrument de musique qui joue des notes les unes à la suite des autres), de permettre de jouer ces partitions simultanément (comme un orchestre), de jouer ces partitions en continu et enfin de pouvoir modifier ces partitions à n'importe quel moment pendant qu'elles sont jouées.

Ce stage aura été pour moi une belle occasion de me perfectionner en développement informatique tout en travaillant sur un sujet qui me passionne, la musique.

Internship resume

As part of my third year in computer science course at the University of Lille, I did an internship in the RMOD team in the national laboratories for computer science researches INRIA in Villeneuve-d'Ascq. The RMOD team works essentially on software engineering with the language Pharo, a full object language.

During this internship, I took part to a whole new project which was to create a new musical environment in Pharo for a DJ named Domenico Cipriani. The goal of this project was in a first part to implement MIDI protocol. This is a protocol that models music for computers or electronic instruments with messages that contain musical information such as the note that is played and with which intensity or the type of the musical action that is done.

To implement the MIDI protocol, we decided to use an already existing C library named PortMidi and call its functions in Pharo. As a first step, I coded a simple example in the C language that uses this library to understand how it works and how to use it. Then, based on this example, I implemented a second version but this time in Pharo. To do this, I used a package in Pharo named uFFI that can call external C functions from the Pharo code.

At this step, I had a simple example of MIDI connections implementation in Pharo with the possibility to send messages (one or more messages) and to receive messages. The next steps of this project were to implement sending message sequences, so sequences of musical notes that I called "instrument score" (we need here to imagine that an instrument is playing a sequence of notes one after another), to be able to play these scores simultaneously (like an orchestra), to play these scores continuously and finally to be able to modify them while they are being played, like for example adding some notes, changing the speed of the melody.

Thanks to this internship, I acquired professional experience and skills in the computer science domain while combining it with another one of my passions, music.

Détermination du Projet Professionnel

Dans le cadre de la découverte du monde socio-professionnel, l'Université de Lille a organisé une option en licence 3 Informatique nommée Détermination du Projet Professionnel (DPP) afin de sensibiliser les étudiants en informatique sur des sujets divers et variés, de la recherche de stage à la gestion de projet. Pour ma part, j'ai choisi de participer à l'option enjeux environnementaux du numérique.

Le but de cette option est de sensibiliser les étudiants sur l'impact du numérique sur l'environnement, d'exposer les dangers et les solutions qui existent. Pour cela, j'ai participé à une activité de groupe consistant à la création d'une fresque contenant des cartes qui traitent de plusieurs sujets sur l'impact du numérique sur l'environnement. Ces cartes parlent toutes d'un sujet concernant de près ou de loin cet impact du numérique. Par exemple, des cartes parlaient des déchets massifs d'appareils électroniques et du fait qu'ils sont engendrés par la société de consommation actuelle. C'est cette société qui incite les gens à acheter toujours plus, comme les téléphones portables avec la sortie régulière de nouveaux smartphones pour que les gens jettent leur téléphone et en achètent un nouveau, bien que celui ne soit pas usé. Le principe de l'activité proposée pour cette option était le suivant : chaque groupe reçoit au fur et à mesure un paquet de cartes distribué aux membres du groupe. Chaque membre du groupe lit sa carte et montre les photos ou graphiques présents dessus, après quoi une discussion autour de ce sujet a lieu. Ensuite, les membres du groupe organisent sur la fresque les cartes en fonction des points communs entre leurs informations.

Une fois la fresque terminée, nous avons pu échanger autour du résultat final dans un premier temps au sein du groupe, notamment sur les questions suivantes :

- Est ce que vous connaissiez cet aspect de l'impact numérique sur l'environnement ?
- Pensez-vous que cet aspect est important ?
- Pensez-vous que cet aspect a un impact fort sur l'environnement ?

Nous avons ensuite échangé sur tous ces sujets par groupe de deux groupes afin de mettre en commun nos réflexions et la manière dont chaque groupe a agencé sa fresque. De mon côté, mon groupe et son duo avaient un agencement similaire notamment pour la consommation d'énergie pour produire des appareils électroniques, les matériaux utilisés pour leur fabrication, l'extraction des minerais pour produire ces matériaux et surtout l'impact de ces actions sur l'environnement avec l'assèchement des lacs et rivières et la détérioration majeure de la faune et la flore. Nous avions aussi un avis commun concernant la société de consommation d'aujourd'hui avec l'achat en masse et pas forcément utile de nouveaux appareils électroniques desquels découlent une grande quantité de déchets numériques presque tous non recyclables à 100 %.

Ce sont finalement ces sujets qui m'ont le plus marqué. Le plus inattendu a été pour moi la quantité considérable d'énergie nécessaire pour produire de l'équipement numérique et aussi le fait qu'aucun équipement ne soit finalement réellement recyclable entièrement. En effet, certaines parties d'un appareil électronique sont recyclables, mais ne le sont plus au bout d'un certain nombre de recyclages.

Dans un second temps, nous avons réitéré le principe de fresque en groupe, mais cette fois-ci avec des cartes parlant des solutions existantes pour remédier à l'impact du numérique sur l'environnement. Pour cette partie, nous avons dessiné un plan à deux axes qui représentent l'efficacité de la solution et sa difficulté de mise en œuvre. Dans les solutions disponibles, nous avions par exemple « se déconnecter du numérique et faire d'autres activités », « acheter moins d'équipement numérique », « réduire le temps que l'on passe sur un ordinateur ou un smartphone » ou encore « sensibiliser sur l'impact du numérique sur l'environnement ». Le but était donc de placer ces solutions en fonction de la difficulté d'appliquer ces solutions et leur efficacité dans le combat contre l'impact du numérique sur l'environnement.

Suite à cet atelier, j'ai repensé à mon utilisation du numérique. J'ai remarqué avec les débats autour des solutions possibles que je pourrais en appliquer certaines dans mon quotidien pour participer à la réduction de l'impact du numérique sur l'environnement. Par exemple, je compte réduire le temps que je passe sur mes appareils électroniques et notamment sur mon ordinateur, je vais éteindre mon ordinateur au lieu de le laisser en veille lorsque je ne l'utilise pas. Je compte aussi réduire le temps que je passe sur mon téléphone.

Finalement, grâce à cet atelier, j'ai pu en apprendre davantage sur l'impact du numérique sur l'environnement et les solutions pour le réduire. En tant que fervent utilisateur d'appareils électroniques, je suis directement concerné par ce problème qui menace l'environnement et compte réduire ma part de cet impact.

Sommaire

Remerciements....2
Résumé de stage....3
Internship resume....4
Détermination du Projet Professionnel....5
Sommaire....7
Introduction....9
Contexte....10
Le laboratoire....10
L'équipe RMOD....11
Le sujet de stage....11
Ma contribution....12
Notions de musique....12
1. Notions de base....12
2. *Le protocole MIDI*....13
PortMidi....15
1. La librairie....15
2. CMake....17
Outils dans Pharo....18
L'environnement musical....21
1. L'outil musical....21
1.1 PortMidi dans Pharo....21
1.2 LiveCoding Music....22
1.2.1 Les notes....22
1.2.2 Les partitions....23
1.2.3 L'orchestre MIDI....24
1.2.4 Tests et Vérifications....26
2. Sound....29
2.1 Le piano MIDI....29
2.2 Code refactoring....30
Conclusion....31
Bilan....32
Glossaire....33
Bibliographie....34
Annexes....35
Annexe 1 – Composition de l'équipe RMOD....35
Annexe 2 – Schéma de la structure du module MIDI....36

Introduction

Il existe aujourd'hui un grand nombre d'artistes qui nous font rêver et nous transportent à travers leurs prestations musicales, du concert d'un orchestre ou d'un groupe de rock, à une playlist de chansons qu'on écoute sur notre smartphone. Le sujet principal de ce rapport concerne une autre forme de prestation musicale qui se répand de plus en plus grâce aux nombreuses possibilités qu'elle offre, la musique en informatique.

Dans la continuité de ma formation en Licence 3, j'ai été amené à faire un stage de fin d'études en entreprise ou en laboratoire informatique du 11 avril au 3 juillet 2022. J'ai effectué ce stage dans les laboratoires nationaux de recherche informatique INRIA à Villeneuve d'Ascq au sein de l'équipe de recherche RMOD. Cette équipe effectue des recherches sur le plan Génie Logiciel du domaine informatique en langage Pharo, notamment dans l'amélioration et l'optimisation d'applications de grande envergure.

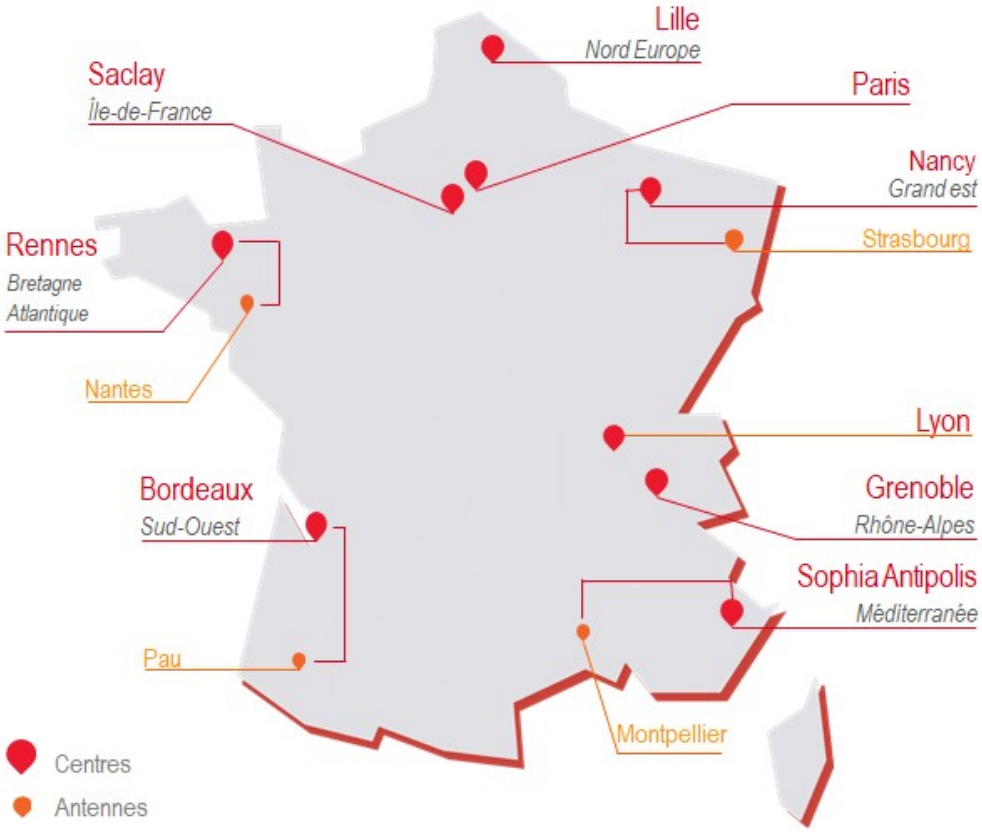
Étant musicien passionné, j'ai choisi de faire mon stage au sein de l'INRIA car l'expérience d'allier deux de mes passions informatique et musique m'attirait beaucoup. En effet, le sujet de ce stage consistait en la création d'un environnement musical dans Pharo. Le projet auquel j'ai participé est un nouveau projet qui a vu le jour suite à la visite du DJ Domenico Cipriani, dans les locaux de l'INRIA. Ce dernier effectue des prestations musicales en programmant de la musique en direct, c'est à dire en construisant sa musique au fur et à mesure qu'il la joue. Cet environnement musical constitue notamment une base solide pour son travail. La partie de ce nouveau projet me concernant fut d'implémenter le protocole MIDI dans Pharo.

Dans un premier temps, je présenterai les laboratoires INRIA, leur histoire et leurs activités. Je présenterai notamment l'équipe de recherche RMOD dans laquelle j'ai effectué mon stage et aussi les contributeurs au projet d'environnement musical en Pharo. Ensuite, dans une seconde partie, je développerai sur mon sujet de stage et sur les différents outils qui m'ont permis de le réaliser. Enfin, dans une dernière partie, je ferai un bilan sur l'état du projet à la fin de mon stage et sur ses apports en terme de connaissances et de compétences personnelles et professionnelles.

Contexte

Le laboratoire

Il existe aujourd'hui de nombreux domaines de recherche dans lesquels beaucoup de chercheurs exercent leur métier. Parmi les onze domaines scientifiques de recherche existants, l'Institut National de Recherche en sciences et technologies du numérique, ou INRIA, est comme l'indique son nom un laboratoire de recherche dans le domaine du numérique et de la technologie. INRIA c'est 200 équipes de recherche en collaboration avec des universités rassemblant au total plus de 3900 chercheurs et ingénieurs sur 10 centres de recherche. L'objectif du laboratoire est la découverte technologique tout en étant à la fois dans le monde académique et dans le domaine industriel.



Implantation de l'INRIA en France

L'INRIA a été créé en 1967 et était nommé à l'époque IRIA (Institut de Recherche en Informatique et en Automatique). Son but principal : placer la France à la pointe de la technologie et la rendre autonome. Le laboratoire se voit chargé de plusieurs missions sur des projets scientifiques ambitieux dont la recherche assistée par ordinateur. En 1979, l'IRIA

devient un institut national et devient l'INRIA et peut s'étendre sur toute la France afin d'attirer les jeunes chercheurs français et étrangers. De plus, à partir de 1984, l'INRIA s'engage dans l'accompagnement de nouvelles startups. C'est au cours des 30 années qui suivent que plus de 160 nouvelles startups seront accompagnées par l'institut.

L'équipe RMOD

Dans ce grand réseau de chercheurs que représente l'INRIA, j'ai intégré au cours de mon stage l'équipe de recherche RMOD située sur le site de Lille/Villeneuve-d'Ascq, dont le responsable est Monsieur Stéphane Ducasse. La composition de l'équipe est disponible en annexe 1. Cette équipe de recherche a pour mission principale l'aide à la réévaluation des applications orientées objet afin de les optimiser en terme d'efficacité, de fonctionnement et de maintenance. RMOD travaille notamment en collaboration avec l'Université de Lille et le Centre de Recherche en Informatique, Signal et Automatique (CRIStAL). Les membres de l'équipe RMOD développent avec le langage Pharo. C'est un langage entièrement orienté objet. Cela signifie que les classes, les méthodes et les variables sont tous des objets. Il en est de même pour l'environnement utilisé pour coder en Pharo. Dans ce langage, un objet est un élément qui définit le comportement minimal de toute chose dans Pharo. Chaque élément ajouté à Pharo découle de cette définition et y apporte des propriétés supplémentaires.

Le sujet de stage

L'INRIA accueille régulièrement des acteurs du monde de Pharo pour favoriser la découverte et le partage de connaissances. Lors de ces événements, les intervenants et leurs projets sont variés, tout comme les domaines qui les concernent. L'un de ces intervenants fut un DJ nommé Domenico Cipriani, un expert en création de musique en LiveCoding, c'est à dire qu'il crée de la musique au fur et à mesure qu'il la joue. Pour pousser ses projets plus loin encore, il a créé avec Pharo un nouveau module qui lui permet de créer de la musique à l'aide de méthodes et d'objets. Lors de sa prestation musicale dans les locaux de l'INRIA, les membres de l'équipe RMOD lui ont proposé d'apporter à son projet une base solide et optimisée. C'est pour cette proposition que mon sujet de stage a été créé.

Au cours de mon stage, j'ai donc commencé un tout nouveau projet consistant en la création d'un module pour Pharo grâce auquel il est possible de créer de la musique. Il existe en musique plusieurs manières de jouer une mélodie : soit en audio, soit avec le protocole MIDI. C'est sur cette deuxième possibilité que je me suis concentré. Je vais maintenant expliquer plus en détail de quelle façon j'ai effectué cette mission.

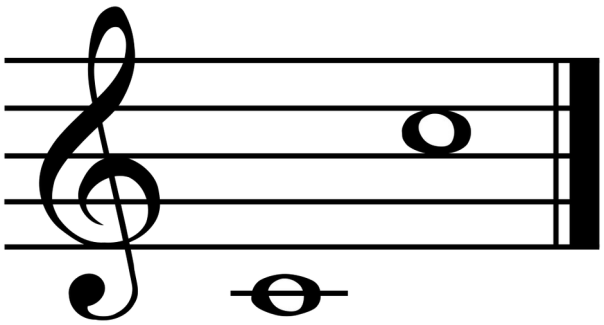
Ma contribution

Notions de musique

La musique est un domaine aussi fascinant que complexe avec un vocabulaire qui lui est dédié. En voici quelques notions de base importantes pour la compréhension du projet auquel j'ai participé.

1. Notions de base

La musique est avant tout une suite de notes jouées les unes à la suite des autres. Une note de musique résulte d'un son joué d'une certaine manière avec une intensité plus ou moins forte. De plus, une même note de musique peut sonner différemment en fonction de l'octave sur laquelle elle se trouve. Une octave représente l'intervalle contenant toutes les notes entre une note et cette même note en plus aiguë ou en plus grave, comme l'indique le schéma suivant pour la note Do.



Une octave de la note Do

Les octaves sont la base des morceaux musicaux, c'est ce qui différencie par exemple une basse qui va jouer des notes graves d'une guitare qui jouera des notes plus aiguës.

Cette suite de notes que les instruments nous font le plaisir de jouer sont des partitions. Les notes sont jouées à un certain rythme que l'on appelle tempo (souvent indiqué en battements par minute, BPM), pendant une certaine durée qui est comptée en temps par définition mais qui peut aussi être représentée en secondes ou millisecondes. Il est d'ailleurs possible que plusieurs partitions soient jouées en même temps pour jouer une mélodie que l'on nomme plus couramment une musique ou une chanson.

2. *Le protocole MIDI*

La musique est toujours associée à des sons joués par des instruments que l'on prend plaisir à écouter. Pourtant, ces instruments ne sont pas les seuls à pouvoir créer de la musique. Les machines peuvent en faire de même, grâce au protocole MIDI.

MIDI signifie Musical Instrument Digital Interface et désigne un protocole de communication de musique pour machines telles que des instruments électroniques ou séquenceurs comme les claviers de piano synthétiques. Le but de ce protocole est de normaliser les échanges d'informations musicales entre tous les appareils musicaux. Les connexions MIDI ont été créées grâce à un travail commun entre les concepteurs de synthétiseurs Dave Smith, Ikutaro Kakehashi et Tom Oberheim qui ont présenté leurs travaux sur le MIDI à une conférence professionnelle dans le monde de la musique. Ces travaux ayant plu au public, C'est à partir de 1985 que le MIDI a été produit à grande échelle dans des ordinateurs pour le grand public.

Les machines ne peuvent pas réellement entendre des sons et les traduire. Elles ne comprennent que les zéros et les uns, soit uniquement des bits. C'est pour cela que pour modéliser de la musique, le protocole MIDI utilise des messages qui ne contiennent qu'une suite de bits représentant les actions musicales effectuées et les notes jouées. De cette façon, les ordinateurs ou autres appareils musicaux traduisent ces suites de bits, agissent en conséquence ou jouent des sons ou des bruits.

Le schéma classique d'une connexion MIDI entre deux appareils et instruments se matérialise par deux câbles appelés connecteurs DIN les reliant. En effet, de par leur conception, les ports MIDI sont unidirectionnels. Il existe trois ports MIDI différents : le port MIDI In pour recevoir des informations, le port MIDI Out pour envoyer des informations, et enfin le port MIDI Thru dont le but est de retransmettre les informations reçues. Aujourd'hui, les câbles USB possèdent des propriétés MIDI In et Out directement intégrées, ce qui permet de n'avoir qu'un seul câble.

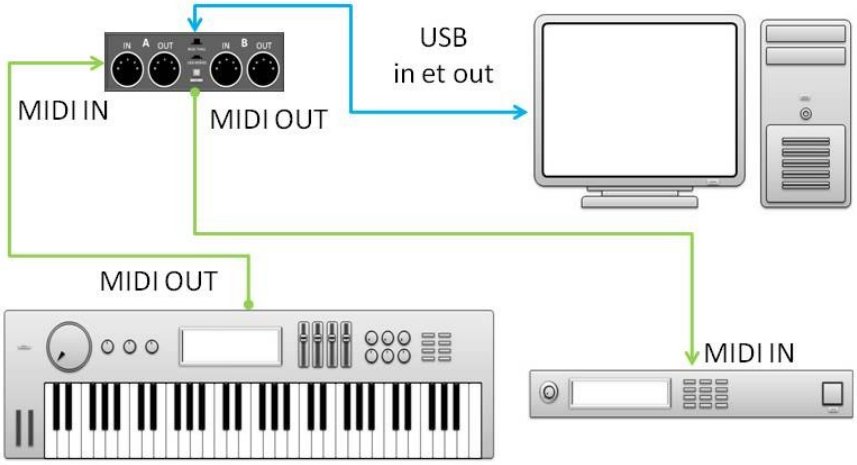


Schéma de connexions MIDI

Aujourd'hui, le protocole MIDI est utilisé par beaucoup d'instruments et logiciels permettant de faire de la musique aussi bien avec des instruments de musique que des logiciels ou appareils électroniques, donnant ainsi accès à un très grand nombre de possibilités. L'une d'entre elle a été implémentée par Domenico Cipriani avec le langage Pharo, à laquelle j'ai apporté ma contribution.

PortMidi

L'objectif de mon stage était d'implémenter les connexions MIDI pour concevoir un nouvel environnement musical dans Pharo. Toutefois, aucun outil n'existe actuellement dans Pharo concernant le MIDI et implémenter l'intégralité du protocole prendrait trop de temps. Pour remédier à ce problème, j'ai utilisé une librairie C externe nommée PortMidi.

1. La librairie

PortMidi est une librairie écrite en langage C dont le principal fondateur est Roger B. Dannenberg. Le dépôt de ce projet est disponible sur plusieurs sites dont GitHub. Cette librairie est un moyen « cross-platform » d'implémenter les connexions MIDI entrée et sortie. Elle contient toutes les fonctions permettant de gérer ces flux de données et d'y envoyer des informations musicales : ouverture et fermeture de flux de données, créations de messages musicaux à partir de données externes, envoi et réception de messages MIDI avec déchiffrage de leurs données. Afin de maitriser la librairie, j'ai réalisé dans un premier temps plusieurs exemples simples.

D'abord, j'ai implémenté en langage C un premier exemple qui utilise cette librairie pour voir ce qu'il était possible de faire et surtout pour comprendre le fonctionnement de la librairie, son comportement. Le dépôt GitHub de PortMidi est très riche en exemples et fichiers annexes permettant d'aider les utilisateurs à prendre en main la librairie. Je me suis donc aidé d'un exemple d'implémentation de la librairie pour développer le mien. Ce premier exemple contient deux fonctions, chacune permettant respectivement d'envoyer et de recevoir des messages MIDI, plus une fonction principale qui implémente le protocole MIDI. En observant l'exemple trouvé sur GitHub et en me renseignant sur le protocole MIDI en général, j'ai conclu que la procédure à suivre pour utiliser les connexions MIDI était la suivante :

- 1. Récupérer l'identifiant du dispositif à utiliser pour les envois ou réception de messages. Cela peut être un logiciel ou sur l'ordinateur utilisé, ou encore un appareil externe comme un synthétiseur.
- 2. Ouvrir le ou les flux nécessaires pour la transmission de messages MIDI (entrée/input pour la réception de messages et sortie/output pour l'envoi de messages).
- 3. Générer les messages MIDI qui seront à transmettre si tel est le cas. En m'aidant de documentation et de la librairie PortMidi, j'ai compris qu'un message MIDI contient en général trois champs de données : le premier contient l'action musicale effectuée comme par

exemple jouer une note. Les second et troisième champs contiennent des données qui varient en fonction de l'action effectuée. Si je reprends l'exemple de la note jouée, le second champ contiendra la valeur de la note jouée et le troisième contiendra son intensité (aussi appelé velocity).

- 4. Envoyer les messages créés sur le flux MIDI output si le but est d'envoyer des messages.

- 5. Recevoir les messages envoyés sur le flux MIDI input si le but est de recevoir des messages. Il faut se mettre en attente de la réception d'un ou plusieurs messages avant d'enclencher la lecture des messages.

- 6. Toujours fermer les flux MIDI input et/ou output à la fin de toute activité MIDI. Ne pas fermer les flux de données peut entraîner des dysfonctionnements de la librairie.

- 7. Avant toute action, initialiser la librairie et après toute action, arrêter la librairie. En effet, PortMidi possède deux méthodes nécessaires à son bon fonctionnement et qui doivent être exécutées avant et après chaque utilisation de celle-ci. Ne pas avoir respecté cette étape a été une source d'erreurs et de comportements étranges au cours du développement de mon projet.

```c
int main(){
    //init PortMidi at beginning
    Pm_Initialize();

    //getting first MIDI-input available
    int input = Pm_GetDefaultInputDeviceID();
    //verifying that API found an input
    const PmDeviceInfo *info;
    info = Pm_GetDeviceInfo(input);
    if (info == NULL) {
        printf("Could not open input device (%d).\n", input);
        exit(EXIT_FAILURE);
    }
    //initializing MIDI-input
    Pm_OpenInput(&midi_in, input, NULL, INPUT_BUFFER_SIZE, NULL, NULL);

    //getting first available MIDI-output
    int output = Pm_GetDefaultOutputDeviceID();
    //verifying that API found an output
    info = Pm_GetDeviceInfo(output);
    if (info == NULL) {
        printf("Could not open output device (%d).\n", output);
        exit(EXIT_FAILURE);
    }
    //initializing MIDI-output for the API
    Pm_OpenOutput(&midi_out, output, NULL, OUTPUT_BUFFER_SIZE, NULL, NULL, LATENCY);

    fprintf(stderr, "midiIn = %d\nmidiOut = %d\n", midi_in, midi_out);

    //testing connections by sending a message to MIDI-in
    sendMessage();
    testReceivingMessages();

    //closing MIDI-in and -out
    Pm_Close(midi_in);
    Pm_Close(midi_out);
```

Premier exemple d'implémentation de la librairie PortMidi

Une fois cet exemple fonctionnel, je l'ai modifié en séparant le code utilisé pour l'envoi de messages et celui pour la réception de messages en créant un second fichier. De cette manière, j'ai pu exécuter les deux exemples dans deux terminaux différents et ainsi obtenir une version plus proche de l'objectif à atteindre.

Néanmoins, pour pouvoir utiliser du code en langage C, il faut d'abord le compiler. Pour exécuter l'exemple précédent, j'ai donc du compiler la librairie PortMidi.

2. CMake

Lorsque je code en langage C, je dois compiler le code que je veux exécuter et j'utilise pour cela un fichier nommé Makefile. L'utilisation de ce type de fichier est simple lorsque je dois compiler peu de fichiers. Mais lorsqu'il faut compiler un grand nombre de fichiers comme pour la librairie PortMidi, écrire le fichier Makefile devient rapidement long et fastidieux. Mais grâce aux indications laissées sur le dépôt GitHub de PortMidi, j'ai découvert une autre manière de compiler des projets en C, CMake.

CMake fonctionne sur la base de fichiers textes nommés CMakeLists. Ces fichiers ont une syntaxe particulière qui vont permettre d'y écrire des instructions que le compilateur C va analyser pour compiler le code C. En analysant un fichier CMakeLists, le compilateur va créer des fichiers compilés aux emplacements voulus qui pourront être utilisés par la suite. De plus, CMake compile pour toutes les versions de système d'exploitation.

```
cmake_minimum_required (VERSION 3.9)

project(portmididemo)

set(CMAKE_CXX_STANDARD 11)
set(CMAKE_C_STANDARD 99)

add_subdirectory(portmidi)

add_executable(PortMidiDemo src/main.c)
add_executable(SendMessages src/send/SendMessages.c)
add_executable(ReceiveMessages src/receive/receiveMessages.c)
target_link_libraries(PortMidiDemo PRIVATE portmidi)
target_link_libraries(SendMessages PRIVATE portmidi)
target_link_libraries(ReceiveMessages PRIVATE portmidi)
```

Fichier CMakeLists utilisé pour mon projet

Pour utiliser la librairie PortMidi dans Pharo et pour des questions de simplicité, je l'ai compilé directement dans mon dépôt GitHub. De cette façon, le fichier compilé est directement disponible pour toute personne souhaitant utiliser mon dépôt.

Outils dans Pharo

La librairie PortMidi est totuefois une librairie externe à Pharo et en langage C. Il n'est donc pas possible de l'utiliser dans Pharo sans intermédiaire. Pour avoir accès aux fonctions que PortMidi propose, j'ai utilisé un paquet déjà intégré à Pharo nommé uFFI.

Unified Foreign Function Interface, ou uFFI, est un paquet présent dans Pharo développé en 2020 par l'équipe RMOD de l'INRIA et qui permet de faire appel à des fonctions externes à Pharo. Ce paquet contient plusieurs classes et plusieurs fonctions qui permettent de modéliser les fonctions, structures et classes externes à Pharo. Dans le cas de la librairie PortMidi, j'ai créé un objet de type FFILibrary (type qui fait partie du paquet Pharo de uFFI) qui contiendra toutes les fonctions liées à celles de la librairie.

```
FFILibrary subclass: #PortMidiLibrary
    instanceVariableNames: ''
    classVariableNames: ''
    package: 'PortMidiDemo-PortMidi'
```

Objet modélisant la librairie PortMidi dans Pharo

Je prends maintenant l'exemple d'une fonction simple de la librairie PortMidi « Pm_Initialize() » qui doit être exécutée avant toute utilisation de la librairie. Le prototype de cette fonction est : PmError Pm_Initialize(void). PmError est un type créé par les développeurs de la librairie pour une gestion des retours des fonctions plus simple. Ce type est en fait un entier déguisé dont la valeur indiquera si une erreur est survenue ou non. Le type void indiqué en paramètre de la fonction signifie qu'aucun paramètre n'est nécessaire pour cette fonction. Avec ces informations, il est maintenant possible de lier cette fonction de la librairie avec une fonction du paquet uFFI. Pour cela, il faut utiliser la fonction « ffiCall: » du paquet uFFI et lui indiquer en paramètre le prototype de Pm_Initialize.

```
portMidiInitialize
    "Calls the PortMidi C library method that initializes the library."
    ^ self ffiCall: #( int Pm_Initialize() )
```

Fonction liée à la fonction Pm_Initialize de PortMidi

Ici j'ai utilisé en paramètre un objet « #( ) » qui en Pharo représente un tableau de données dans lequel j'ai renseigné le type de la fonction. Ici j'ai inscrit « int » en type de retour de la fonction pour indiquer un entier et non le type PmError. En effet, avec uFFI, il est possible de ne pas renseigner le type précis des variables ou des retours des fonctions et à la place indiquer un type relativement similaire. Si je souhaite à l'inverse renseigner le type exact et que ce type ne figure pas dans les types par défaut du langage C, il faut le modéliser de la même manière qu'avec les classes. Pour le type PmError, il faudrait utiliser une « FFIEnumeration » afin de lister toutes les valeurs de retour possibles.

Avec uFFI, il est aussi possible de lier à Pharo des fonctions plus complexes avec notamment un type de retour très fréquent en langage C, les pointeurs. Je prends maintenant un autre exemple de fonction de la librairie PortMidi, une fonction qui permet d'ouvrir un flux de données pour l'envoi de messages MIDI (il existe aussi une fonction jumelle pour un flux de réception de messages MIDI).

```
openOutputForStream: aStream withDevice: outputDeviceID withOutputDriverInfos: infos
withBufferSize: outputBufferSize withTimeProcedure: timeProcedure withTimeInfos: timeInfos
andLatency: latency
   "Calls the PortMidi C library method that open an MIDI output where aStream is a pointer
that receives a pointer to the newly opened stream, outputDeviceID is the device to open.
Parameters infos, timeProcedure and timeInfos are optional."
   ^ self ffiCall: #( int Pm_OpenOutput( void** aStream, int outputDeviceID, void* infos,
uint outputBufferSize, void* timeProcedure, void* timeInfos, uint latency ) )
```

Fonction d'ouverture de flux MIDI output

Cette fois-ci, l'appel « ffiCall : » est plus dense et complexe. Cette fonction contient sept paramètres dont certains sont des pointeurs. Le type de retour est un entier PmError comme pour la fonction précédente. Les paramètres « outputDeviceID », « outputBufferSize » et « latency » sont des entiers dont le type est présent par défaut dans le langage C. Il y a aussi un autre point important dans cet exemple, les paramètres de type « void* ». Le type void* indique en C que cela peut être n'importe quel type, ce qui permet d'y renseigner des valeurs nulles. Les paramètres ayant un type void* dans la fonction ci-dessus peuvent avoir des valeurs nulles car elles sont optionnelles au bon fonctionnement de son exécution. C'est pourquoi le type void* est ici le plus adapté. Enfin, il reste le type « void** ». Ce type indique que la variable aStream est un pointeur de pointeur. C'est à dire que la valeur de cette variable est un pointeur vers une zone de la mémoire de l'ordinateur qui contient un autre pointeur vers une autre zone de la mémoire contenant le flux de données dans lequel seront envoyés les messages MIDI. Comme c'est un pointeur, il ne

peut être renseigné comme un type par défaut du langage C. En effet, le langage C et le langage Pharo ne gèrent pas la mémoire de l'ordinateur de la même manière. Le langage Pharo alloue des espaces dans la mémoire dynamiquement. Cela signifie que les adresses des objets Pharo en mémoire peuvent changer, tandis que d'un autre côté, le langage C alloue des espaces en mémoire de manière statique. De ce fait, si je renseigne en paramètre un pointeur dont l'adresse et la valeur peuvent changer, le code en C ne pourra pas retrouver cette variable et cela peut détériorer l'image Pharo. Pour remédier à ce problème, j'ai utilisé une autre classe du paquet uFFI nommée « FFIExternalValueHolder ».

```
createNewPointerForStreamWithType: pointerType
   "Create a new instance of FFIExternalValueHolder and an instance of this new class to
get a pointer for poiterType."
   | midiStream valueHolderClass |
   valueHolderClass := FFIExternalValueHolder ofType: pointerType.
   midiStream := valueHolderClass new.
   ^ midiStream tfPointerAddress.
```

Exemple d'utilisation d'une classe à adresse en mémoire fixe

Les instances de cette classe sont des objets dont l'adresse en mémoire ne changera pas tout au long de leur utilisation. J'ai donc créé une nouvelle instance de cette classe en précisant « void** » pour indiquer le type qu'elle représente puis j'ai passé l'adresse de cette nouvelle instance en paramètre à la fonction d'ouverture de flux MIDI output.

Tous les outils que j'ai mentionné précédemment m'ont permis de mettre en œuvre le projet auquel j'ai été assigné pour mon stage. J'ai pu apprendre les fondamentaux sur le protocole MIDI, poser des bases solides avec la librairie PortMidi en C puis dans Pharo avec le paquet uFFI. Je vais maintenant parler plus en détails du projet auquel j'ai participé pendant mon stage.

L'environnement musical

Lors de ma première réunion avec mon tuteur de stage Monsieur Santiago Bragagnolo, il a présenté mon sujet de stage en détail. Nous en avons retenu deux parties majeures : l'implémentation MIDI et l'implémentation sonore. Implémenter l'un permettrait d'implémenter l'autre plus facilement car les procédures pour produire de la musique restent les mêmes. Nous avons donc décidé que je me concentrerai sur l'implémentation de musique live avec des connexions MIDI.

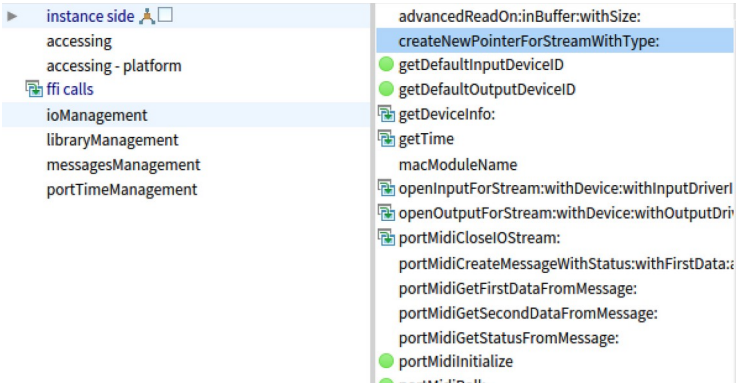
1. L'outil musical

Le paquet que j'ai développé se décompose en trois parties : une première partie qui concerne toutes les classes et méthodes provenant de la librairie C PortMidi, une seconde partie contenant tout le code que j'ai écrit pour implémenter les prestations musicales en live à l'aide de PortMidi, et enfin une troisième partie pour tous les tests que j'ai écrit.

1.1 PortMidi dans Pharo

Après avoir compris le fonctionnement de la librairie PortMidi en langage C, j'ai redéveloppé cet exemple en langage Pharo. Comme décrit ci-dessus dans la partie sur le paquet uFFI, j'ai créé un nouvel objet dans Pharo dont la superclasse est « FFILibrary ». J'ai écrit une méthode effectuant un appel externe « ffiCall : » aux méthodes importantes pour utiliser le protocole MIDI, à savoir :

- Les méthodes d'initialisation et d'arrêt de PortMidi
- Les méthodes permettant de récupérer les informations sur les dispositifs MIDI disponibles
- Les méthodes permettant l'ouverture et la fermeture des flux MIDI input et output
- Les méthodes de création, d'envoi et de réception de messages MIDI

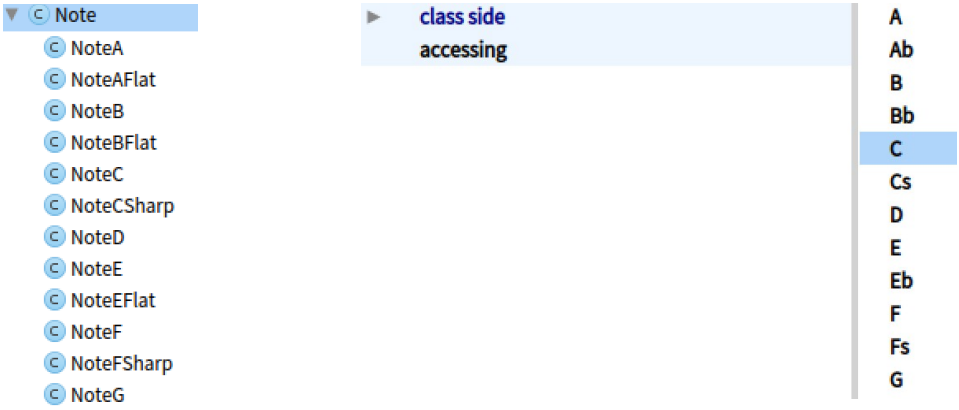


Méthodes de PortMidi dans Pharo classées par catégorie

1.2 LiveCoding Music

1.2.1 Les notes

Pour implémenter de la musique, il faut d'abord partir de la caractéristique la plus simple en musique et qui est à la base de toute mélodie, les notes de musique. En MIDI, les notes ne sont que des valeurs. Le protocole MIDI est capable de couvrir environ dix octaves de musique, soit 128 notes. Dans un premier temps, j'ai créé une classe Note qui contient simplement le nom et la valeur d'une note. Par exemple, un Do à la cinquième octave aura comme valeur 60 et comme nom 'C' (C correspond au Do en anglais). Toutefois, renseigner soi-même le nom et la valeur d'une note devient vite long lorsqu'il s'agit d'écrire une partition entière. Pour optimiser cette partie de la prestation musicale, j'ai donc ajouté une méthode de classe qui créer une instance de cette classe pour chacune des 12 notes qui existent. Chacune de ces instances correspond à une sous classe de la classe Note qui, à l'initialisation d'une instance, vont directement avoir les valeurs correspondant à la note qu'elles représentent.



Sous classes de la classe Note et les méthodes de classes leur correspondant

De plus, j'ai créé une seconde classe de notes plus élaborée que j'ai appelé MIDINote et qui cette fois-ci contient une instance de la classe Note plus une octave et une durée. La durée de la note dépend du tempo utilisé pour la prestation musicale mais aussi du rythme musical de la note. Par exemple, une note noire Do avec un tempo de 60 BPM durera une seconde. Cette encapsulation combinée aux méthodes permet d'automatiser la création d'une note tout en personnalisant son octave et obtenir au final sa valeur numérique correcte.

```
Object subclass: #MIDINote
    instanceVariableNames: 'note octave duration velocity'
    classVariableNames: 'DefaultDuration DefaultOctave DefaultVelocity'
    package: 'PortMidiDemo-Source'
```

Définition de la classe MIDINote

Ainsi, pour créer une note Do de valeur 60, il suffit d'écrire le code « MIDINote new note: Note C ; octave : 5 ».

1.2.2 Les partitions

Les notes sont importantes pour la musique mais c'est en créant des suites de notes que l'on obtient des mélodies ou des prestations musicales. De mon point de vue de musicien, je considère ces suites de notes comme des partitions que l'on joue. D'un autre côté, d'autres musiciens plus techniques comme Domenico Cipriani représentent ces suites de notes plutôt comme des pistes musicales. J'ai donc créé sur cette base une classe nommée « InstrumentScore », soit une partition d'instrument.

```
Object subclass: #InstrumentScore
    instanceVariableNames: 'track instrument positionInScore timerCounter channel muted'
    classVariableNames: 'DefaultVelocity'
    package: 'PortMidiDemo-Source'
```

Définition de la classe InstrumentScore

Cette classe contient une liste de notes qui constituent la mélodie à jouer, des variables d'instance et un ensemble de méthodes utiles pour la lecture de ces partitions. Je reviendrai un peu plus tard sur ce point.

Toutefois, la plupart du temps, la musique ne se limite pas à un seul instrument. Ce qui fait aussi la musique, c'est la synergie entre les instruments, que ce soit pour un groupe de métal ou un orchestre. C'est pourquoi j'ai ajouté une autre classe de partition nommée « Score ». La différence entre la classe InstrumentScore et la classe Score réside dans le fait que InstrumentScore modélise un seul instrument, une seule mélodie à jouer tandis que la

classe Score représente un ensemble de ces pistes musicales. Ici aussi, cette classe contient les variables d'instances et un ensemble de méthodes pour la lecture des partitions.

```
Object subclass: #Score
    instanceVariableNames: 'tempo tracks modified'
    classVariableNames: 'DefaultTempo'
    package: 'PortMidiDemo-Source'
```

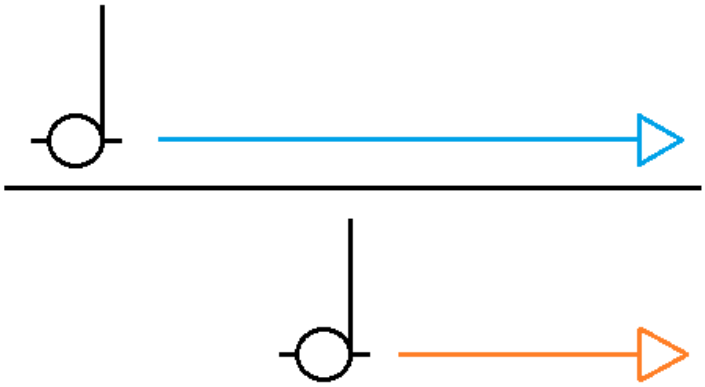
Définition de la classe Score

1.2.3 L'orchestre MIDI

L'objectif principal de mon sujet de stage est de pouvoir jouer de la musique dans Pharo. Après avoir développé les classes permettant de créer des partitions, j'ai créé les méthodes permettant de donner vie à ces suites de notes.

En MIDI, jouer une note se décompose en deux étapes : la première représente le moment où l'on commence à jouer la note (événement aussi appelé NoteOn). On peut représenter ce moment par une simple pression sur une touche d'un piano. La seconde étape représente le moment où l'on décide d'arrêter de jouer la note (que l'on appelle aussi NoteOff). On peut représenter ce moment par le relâchement d'une touche de piano. Du point de vue du protocole MIDI, ces deux événements correspondent à deux messages distincts qui doivent être envoyés l'un après l'autre. Le premier champ des deux messages est une valeur représentant les événements NoteOn et NoteOff, leur deuxième champ correspond à la note jouée et leur troisième champ est l'intensité de la note. Pour un message NoteOn, l'intensité est variable mais pour un message NoteOff, celle-ci sera égale à zéro. En réutilisant les méthodes de la librairie PortMidi que j'ai liées dans Pharo avec uFFI, j'ai codé des méthodes qui permettent de jouer une note donnée pendant le temps qui lui est propre. J'ai ensuite réutilisé ces méthodes pour jouer une mélodie. Pour cela, j'ai appliqué la méthode précédente sur chaque note de la suite de notes. Enfin, j'ai repris ces méthodes pour coder la possibilité de jouer une partition qui contient plusieurs partitions d'instruments en même temps, comme un orchestre MIDI.

Cette dernière partie fut complexe car jusqu'ici, il était possible de ne jouer qu'une seule piste et le fait d'attendre la fin d'une note jouée ne posait pas de problème. Avec la possibilité de jouer plusieurs pistes en même temps, il faut garder en tête que les notes ne sont presque jamais jouées en même temps et que certaines notes se terminent pendant que d'autres sont jouées. De ce fait, l'exécution de la prestation musicale doit toujours garder la main sans avoir à attendre une quelconque note.



Exemple d'une suite de notes jouées simultanément

Pour remédier à ce problème, j'ai instauré un système avec un timer qui joue le rôle de repère tout au long de la lecture d'une partition.

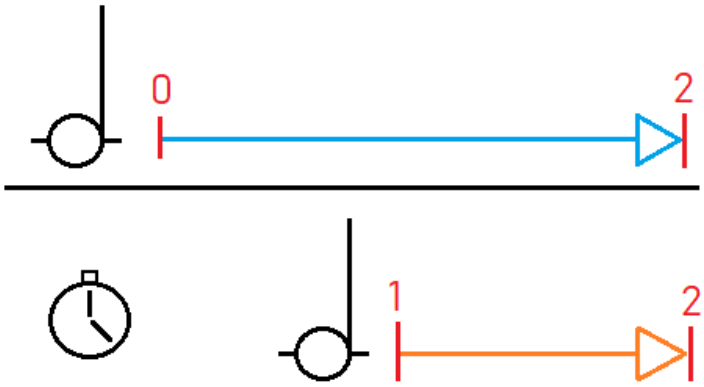


Schéma du comportement du timer

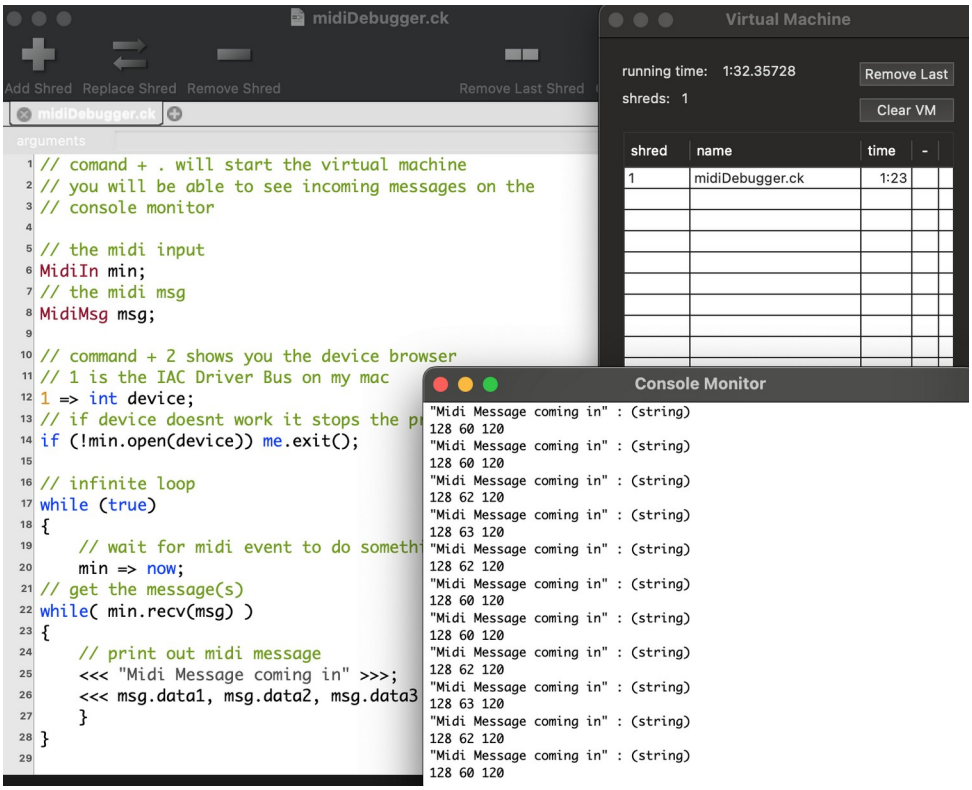
Lors du démarrage de la lecture, le timer est lancé. Au même moment, une boucle est lancée sur une méthode dont le but est de vérifier l'état de la partition en fonction du temps écoulé. On appelle cette technique « Polling ». Pour effectuer cette vérification, cette méthode va se servir de la variable d'instance « timerCounter » de chaque partition. Cette variable va contenir le temps que le timer doit atteindre avant la prochaine note à jouer. Par exemple, si on débute la lecture d'une partition d'instrument, la variable d'instance timerCounter vaudra zéro. Cela signifie qu'il faut sans plus attendre jouer la prochaine note de la partition, c'est à dire la première note. Dès lors que la méthode remarque un changement d'état, elle envoie un message NoteOff pour la note jouée si besoin et envoie un

message NoteOn pour la prochaine note à jouer. Ainsi, toutes les partitions d'instrument peuvent être jouées simultanément sans pour autant se bloquer les unes avec les autres.

1.2.4 Tests et Vérifications

J'ai développé ce projet MIDI avec la méthode TDD (Test Driven Development), en répétant la procédure suivante : écrire des tests pour mon code, puis coder les méthodes de façon à faire fonctionner les tests. Pour tester le bon fonctionnement des échanges de messages, j'ai donc dans un premier temps créé des tests pour les méthodes simples et créé une copie de mon image Pharo afin que, de la même manière que pour tester mon exemple simple en langage C, je puisse avoir une image qui envoie des messages MIDI et l'autre image qui se charge de recevoir les messages pour tester les méthodes plus complexes. D'un côté j'avais une boucle qui envoyait des messages à intervalle régulier, de l'autre côté une autre boucle qui récupérait les messages et les affichait dans une console dans Pharo que l'on appelle Transcript afin de vérifier que les données que je transmets soient correctes.

Suite à une réunion avec Domenico Cipriani au cours de laquelle nous avons examiné l'avancement de mon projet et du sien, il m'a proposé une autre alternative pour tester mon code appelé miniAudicle.



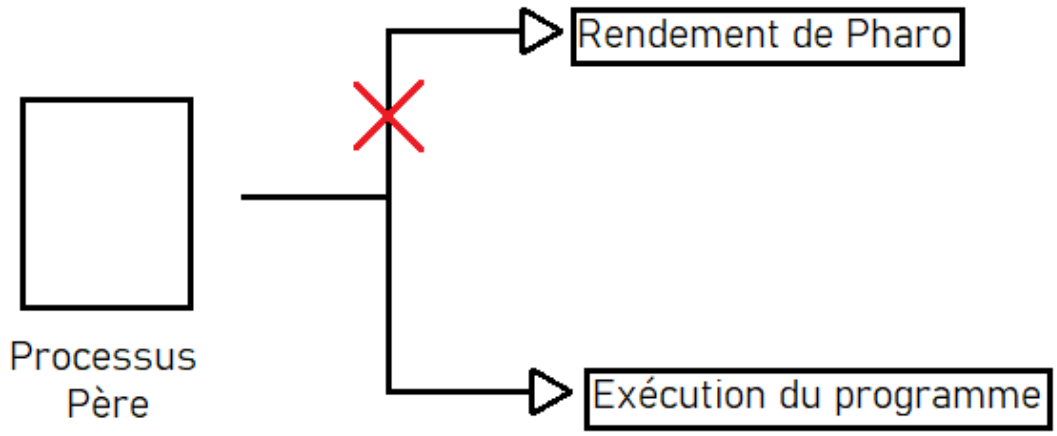
Logiciel MiniAudicle

MiniAudicle est un logiciel qui va de paire avec le langage Chuck, un langage spécialement conçu pour la création de musique. Il permet notamment de tester les support musicaux que l'on utilise. Pour ce projet, Domenico Cipriani m'a proposé un fichier écrit en langage Chuck et qui se comporte comme un debugger pour les supports musicaux. Comme présenté sur la capture d'écran ci-dessus, miniAudicle possède un éditeur pour écrire du code, un lanceur de machine virtuelle pour tester le code, et une console pour afficher toutes les informations importantes lors de l'exécution du code. Grâce au fichier que Domenico Cipriani m'a transmis, j'ai pu contrôler la réception des messages MIDI que j'envoyais depuis Pharo et surtout de vérifier le temps entre chacun de ces messages.

Bien sûr, hormis la copie de mon code ou l'utilisation d'un autre logiciel pour tester mes échanges de données, il est possible de faire ces vérifications avec des synthétiseurs ou autres machines pouvant recevoir et traduire des messages MIDI.

1.2.5 La musique live

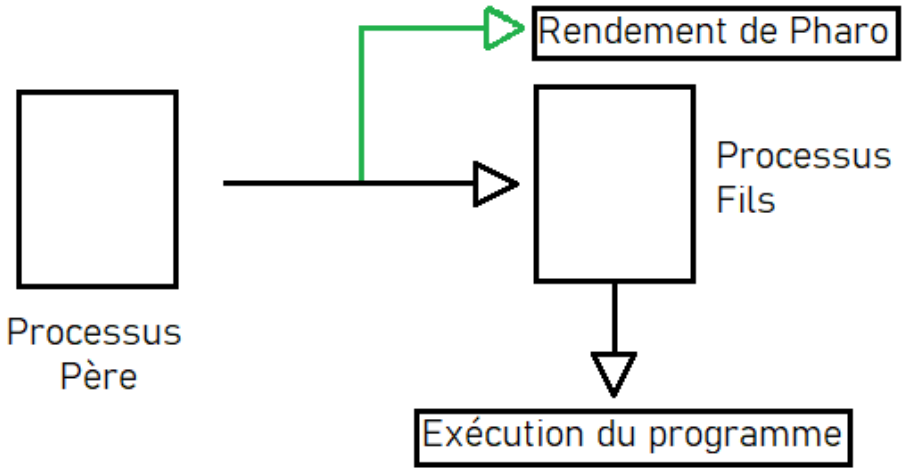
Afin que mon projet permette réellement de coder de la musique live, j'ai ajouté au lecteur de partitions la possibilité de les modifier en même temps que de les jouer. Le problème auquel je me suis heurté fut la lecture des partitions. Il y a un point important à connaître sur Pharo : lorsque j'exécute du code dans le playground pour jouer une partition, l'image Pharo se gèle jusqu'à la fin de l'exécution du code.



Comportement du processus du rendement de Pharo

Cela est dû au fait que le processus dans lequel est exécuté ce code est le même processus qui s'occupe du rendement visuel de l'image de Pharo. De ce fait, lorsque je joue une partition, je suis en incapacité de faire autre chose en même temps dans l'image Pharo. Pour résoudre ce problème, j'ai utilisé un paquet de Pharo qui permet de créer des

processus indépendants afin d'exécuter la lecture des partitions à l'intérieur d'un nouveau processus. Ainsi, je peux effectuer des modifications sur les partitions tout en les jouant.



Utilisation de processus indépendant

Lors de réunions avec Domenico Cipriani, Santiago Bragagnolo et Stéphane Ducasse, plusieurs ajouts de fonctionnalités ont été proposées. Nous en avons retenu une en particulière qui consiste en l'enregistrement des prestations musicales. Cela signifie que chaque modification de la partition doit être prise en compte et enregistrée pour pouvoir rejouer avec exactitude la prestation musicale. Pour implémenter cet aspect de la musique live, j'ai décidé de fonctionner avec un système de versions. Chaque modification entraine une nouvelle version de la partition à un temps donné dans la prestation. J'ai donc ajouté une nouvelle variable d'instance au joueur de partition qui contient chacune des versions dans un dictionnaire ordonné : c'est une structure de donnés dans laquelle il est possible d'ajouter des associations (clé, valeur) tout en conservant l'ordre d'ajout des couples de donnés. Ainsi, lors de chaque modification de la partition, la version actuelle de la partition est stockée dans le dictionnaire avec en clé la valeur actuelle du timer (pour pouvoir retrouver lors de la relecture de la prestation à quel moment cette version a été modifiée), et un clone de cette version est créé puis modifié pour devenir enfin la nouvelle version de la partition.

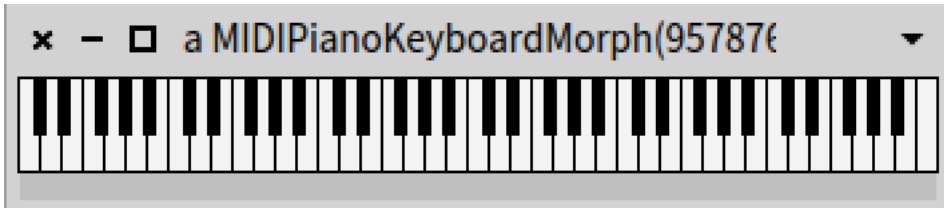
En plus des classes et fonctions que j'ai développé pour permettre les prestations musicales avec le protocole MIDI dans Pharo, j'ai ajouté à mon dépôt GitHub le code d'un ancien paquet écrit sur l'ancêtre de Pharo nommé Squeak.

2. Sound

2.1 Le piano MIDI

Le paquet Sound a été développé en SmallTalk en 2002 et contient un large choix de classes et fonctions qui permettent de faire de la musique.

Ce paquet contient des interfaces (Morph) avec des graphiques modifiables comme un égaliseur ou encore un enregistreur de voix. L'interface qui m'a le plus intéressé fut celle d'un clavier de piano.



Interface du paquet Sound d'un clavier de piano

En appuyant sur les touches de ce clavier, on peut jouer de la musique comme sur un synthétiseur ou sur un vrai piano, que ce soit en mode MIDI ou en mode audio. Dû à l'ancienneté du paquet, j'ai utilisé un vieil ordinateur pour le faire fonctionner. De plus, le paquet ne fonctionne pas sur Pharo, j'ai donc adapté le paquet en ajoutant les classes manquantes dans la dernière version de Pharo mais présentes dans d'anciennes versions et en modifiant certaines parties du code qui étaient codées avec d'anciennes méthodes qui ne sont pas adaptées à Pharo. Par exemple, l'une des erreurs était causée par une fonction dont le nom était identique à d'autres fonctions déjà présentes. Pharo considérait donc cette fonction comme une réécriture des fonctions originales et causait des dysfonctionnements.

Une fois le paquet Sound adapté et réparé pour la dernière version de Pharo, j'ai parcouru le code concernant l'interface du clavier de piano et y ai apporté des modifications afin de relier les touches de ce clavier à mon code pour envoyer les messages MIDI correspondant aux notes jouées. Pour vérifier le bon fonctionnement de ce clavier avec mon code, j'ai effectué mes tests avec miniAudicle et le fichier Chuck que Domenico Cipriani m'avait transmis pour confirmer la bonne réception des messages envoyés avec les touches.

J'ai ajouté ma version du paquet Sound à mon dépôt GitHub (tout en partageant le lien du dépôt original de Sound) afin que les utilisateurs curieux puissent réutiliser ce clavier de piano. C'est d'ailleurs ce que Monsieur Domenico Cipriani a fait en reprenant son code pour

créer une nouvelle interface graphique permettant de jouer de la musique, mais cette fois-ci avec du son.

2.2 Code refactoring

Bien que le paquet Sound soit un ancien paquet, il représente plusieurs années de travail et possède une structure bien définie et une hiérarchie des classes bien fondée. De plus, ce paquet et celui que j'ai développé lors de mon stage possèdent beaucoup de points communs sur l'implémentation du MIDI. Au cours d'une réunion avec Santiago Bragagnolo et Stéphane Ducasse, nous avons relevé plusieurs confusions concernant les termes que j'ai employé pour définir certaines classes, méthodes et variables. De plus, certaines méthodes n'avaient pas leur place dans les classes où elles se situaient. Nous avons donc convenu que je ferais du refactoring sur tout le projet. Pour ce faire, j'ai d'abord posé la structure de mon projet et l'ai comparé à celle du paquet de Sound afin de voir quels points de mon projet sont à revoir. J'ai commencé par réorganiser les classes et à les renommer. Par exemple, j'ai renommé la classe « Score » qui modélise une partition d'instrument en « InstrumentScore », et la classe « Prestation » en « Score » pour bien différencier une partition d'instrument d'une partition qui contient plusieurs partitions d'instrument. De plus, j'ai créé une nouvelle classe que j'ai appelé « MIDIScorePlayer » qui se charge uniquement de jouer de la musique, que ce soit une partition, une partition d'instrument ou simplement une note.

```
Object subclass: #MIDIScorePlayer
   instanceVariableNames: 'midiOut scoreVolume score versions currentVersionIndex'
   classVariableNames: 'DefaultOutput DefaultVolume'
   package: 'PortMidiDemo-Source'
```

Définition de la classe MIDIScorePlayer

C'est cette classe qui se charge d'exécuter la lecture d'une partition dans un processus indépendant et qui permet de faire les modifications de cette partition dont je parle dans la section 1.2.5.

Conclusion

Dans le cadre du projet de concevoir un environnement musical dans Pharo pour soutenir le projet du DJ Domenico Cipriani de créer de la musique en direct, j'ai conçu au cours de ce stage au sein de l'INRIA un nouveau module répondant à ses attentes. J'ai pu réaliser ce projet avec l'aide de plusieurs outils comme la librairie C PortMidi ou le paquet uFFI dans Pharo, avec le protocole MIDI dans Pharo.

Aujourd'hui, ce module est disponible sur mon dépôt GitHub avec un fichier README.md qui contient les étapes à suivre pour installer les différents paquets qu'il contient mais aussi quelques explications afin de comprendre le sens et la fonction des différentes classes que j'ai écrites. Ce dépôt contient donc tous les éléments nécessaires pour réaliser ses propres prestations musicales en MIDI. Le dépôt étant public, tout le monde peut y accéder et l'intégrer à son projet personnel pour créer de la musique en MIDI dans Pharo. De plus, Domenico Cipriani utilise ce module pour consolider le sien. Tout comme lui, d'autres utilisateurs montrant de l'intérêt pour le MIDI dans Pharo notamment dans la communauté de Pharo sur le site officiel du langage peuvent réutiliser mon travail pour construire leurs propres projets.

Effectuer ce stage au sein de l'équipe RMOD m'a permis de perfectionner mes compétences en développement. En apprenant à utiliser la librairie externe C PortMidi, j'ai pu découvrir la programmation en langage C avec une librairie externe et comment compiler de gros projets grâce à l'outil CMake que je ne manquerai pas de réutiliser dans de futurs projets en C. De plus, développer en Pharo m'a permis de découvrir une autre manière de faire de la programmation orientée objet. J'ai pu exploiter au maximum le concept de simplification des méthodes (une action, une méthode) qui favorise la compréhension et la polyvalence de mon code. Ensuite, en étudiant le protocole MIDI, j'ai pu approfondir mes connaissances en musique et aussi revoir certaines notions. J'ai pu découvrir une autre façade de la musique que je n'ai pas l'habitude de voir en tant que bassiste. Enfin, intégrer l'équipe RMOD m'a permis de découvrir d'autres manières de développer des programmes grâce à des sprints organisés en fin de mois au cours desquels chacun travaille en duo sur un problème non résolu en Pharo, ou la découverte de nouvelles techniques de développement comme le kata, qui consiste à programmer en duo sur un sujet pendant que le reste de l'équipe observe et avec un changement régulier de duo de programmeurs.

Bilan

D'un point de vue professionnel, ce stage au sein de l'INRIA m'a permis de découvrir une autre façade du domaine professionnel informatique. J'ai pu découvrir le métier de chercheur en laboratoire et les projets auxquels ils participent. J'ai pu apprendre à utiliser de nouveaux outils comme CMake, voir de nouvelles techniques de développement et améliorer celles que j'avais déjà acquises. J'ai pu participer à un tout nouveau projet dont j'étais l'unique auteur, apprendre à définir les tâches importantes et à les organiser, communiquer sur mon avancée à mon tuteur de stage Santiago Bragagnolo et l'utilisateur final Domenico Cipriani en organisant et animant des réunions.

C'est pourquoi je suis d'autant plus déterminé à poursuivre mes études en développement informatique en alternance afin de continuer à mettre en pratique mes connaissances et compétences acquises lors de mes formations en informatique et lors de mon stage au sein de l'INRIA.

Glossaire

GitHub : GitHub est une entreprise de développement et services logiciels aux États-Unis. GitHub développe notamment la plateforme internet du même nom sur laquelle il est possible de sauvegarder des projets sous forme de dépôt.

Cross-Platform : Un logiciel cross-platform ou multiplateforme est un logiciel conçu pour fonctionner sur plusieurs plateformes informatiques telles que MAC OS ou Windows.

Thread : un thread est la traduction anglaise de processus. Un processus est un programme auquel on assigne une tâche à réaliser.

Multithreadé : Un programme multithreadé est un programme qui utilise plusieurs threads, c'est à dire plusieurs processus en même temps afin d'effectuer plusieurs tâches en même temps.

Timer : un timer est un chronomètre, souvent utilisé pour se repérer dans le temps.

Prototype : Le prototype d'une fonction contient le type de retour de cette fonction, son nom et ses paramètres.

Pointeur : En langage C, un pointeur est une valeur correspondant à un espace bien défini dans la mémoire d'une machine.

Égaliseur : En musique, un égaliseur permet de modifier ou de filtrer différentes bandes de fréquences sonores.

Encapsulation : C'est un principe qui consiste à ajouter un masque, une épaisseur de code afin notamment de cacher à l'utilisateur les données brutes.

Refactoring : En développement, le refactoring consiste à reprendre du code et de repenser sa structure, sa syntaxe, sa nomenclature, à diviser le code pour plus de clarté et d'efficacité.

Bibliographie

https://github.com/PortMidi/portmidi : Dépôt GitHub de la librairie externe en langage C PortMidi. Ce dépôt contient des exemples d'implémentation de la librairie.

« Le protocole MIDI », document synthèse par Jean Décarie et Simon-Pierre Gourd, 2001. C'est un document au format pdf qui m'a permis d'en savoir plus sur le protocole MIDI.

« Maximum MIDI Music application in C++ », Paul Messic, 1998. Adaptation numérique au format pdf du livre. C'est un autre document sur lequel je me suis appuyé pour comprendre et implémenter les connexions MIDI en C et dans Pharo.

https://github.com/pharo-contributions/Sound : Dépôt GitHub du projet Sound dont je me suis inspiré pour retravailler la structure de mon module.

« Concurrent Programming in Pharo », Stéphane Ducasse et Guillermo Polito, 29 septembre 2021. C'est un document au format pdf qui m'a permis d'implémenter les processus dans mon projet pour permettre la lecture et modification simultanée de partitions.

https://stackoverflow.com : Site internet StackOverflow. C'est un forum sur lequel j'ai pu trouver des réponses lorsque je programmais en langage C et notamment pour la compilation de la librairie PortMidi.

Annexes

Annexe 1 – Composition de l'équipe RMOD

Membres

Permanents

- **Professeur**
  - Anne Etien
- **Directeur de recherche**
  - Stéphane Ducasse (Responsable)
- **Maîtres de conférences**
  - Nicolas Anquetil (hdr)
  - Vincent Aranega
- **Chargés de recherche**
  - Steven Costiou
  - Marcus Denker
- **Ingénieurs**
  - Christophe Demarey
  - Guillermo Andres Polito
  - Pablo Adrian Tesone

Non permanents

- **Doctorants**
  - Nour jihene Agouf
  - Santiago Bragagnolo
  - Aless Hosry
  - Pierre Misse-Chanabier
  - Théo Rogliano
  - Maximilian Willembrinck
  - Oleksandr Zaitsev
- **Ingénieurs**
  - Soufyane Labsari
  - Clotilde Toullec

Annexe 2 – Schéma de la structure du module MIDI