Construction 2
Génération aléatoire de musique à base d'apprentissage profond

Destinataires:

Clients : Olivier Bodini et Nadi Tomeh, LIPN, USPN

Equipe de suivi: Sophie Toulouse et Thierry Hamon

INFO2 - Groupe 1:

Raphaël Garnier : *raphael.garnier@edu.univ-paris13.fr*

Antoine Escriva

Clément Guerin

Florian Bossard

Clément Bruschini

12 avril 2021

Version 5.0

Table des matières

Introduction

Dans le cadre du cours de conduite et gestion de projet, des groupes d'étudiants doivent mener un projet tout au long de sa vie. De sa phase d'avant-vente aux 5 itérations de production. Ce document est un rendu final reprenant l'ensemble du projet.

Ce projet nous a été adressé par Olivier Bodini et Nadi Tomeh appartenant respectivement au LIPN et USPN.

Notre projet a pour but de créer une application permettant à un utilisateur d'appliquer des algorithmes d'apprentissage profonds (de type RNN) sur une base de fichiers MIDI afin de générer de nouveaux morceaux de musique dans le style de cette base.

L'utilisateur aura accès à une interface simple permettant de choisir divers paramètres concernant la génération de la musique ainsi que de l'écouter et de l'enregistrer.

Ce rapport présente le résultat de l'ensemble des phases de ce projet, de l'initialisation jusqu'à la phase de construction 2.

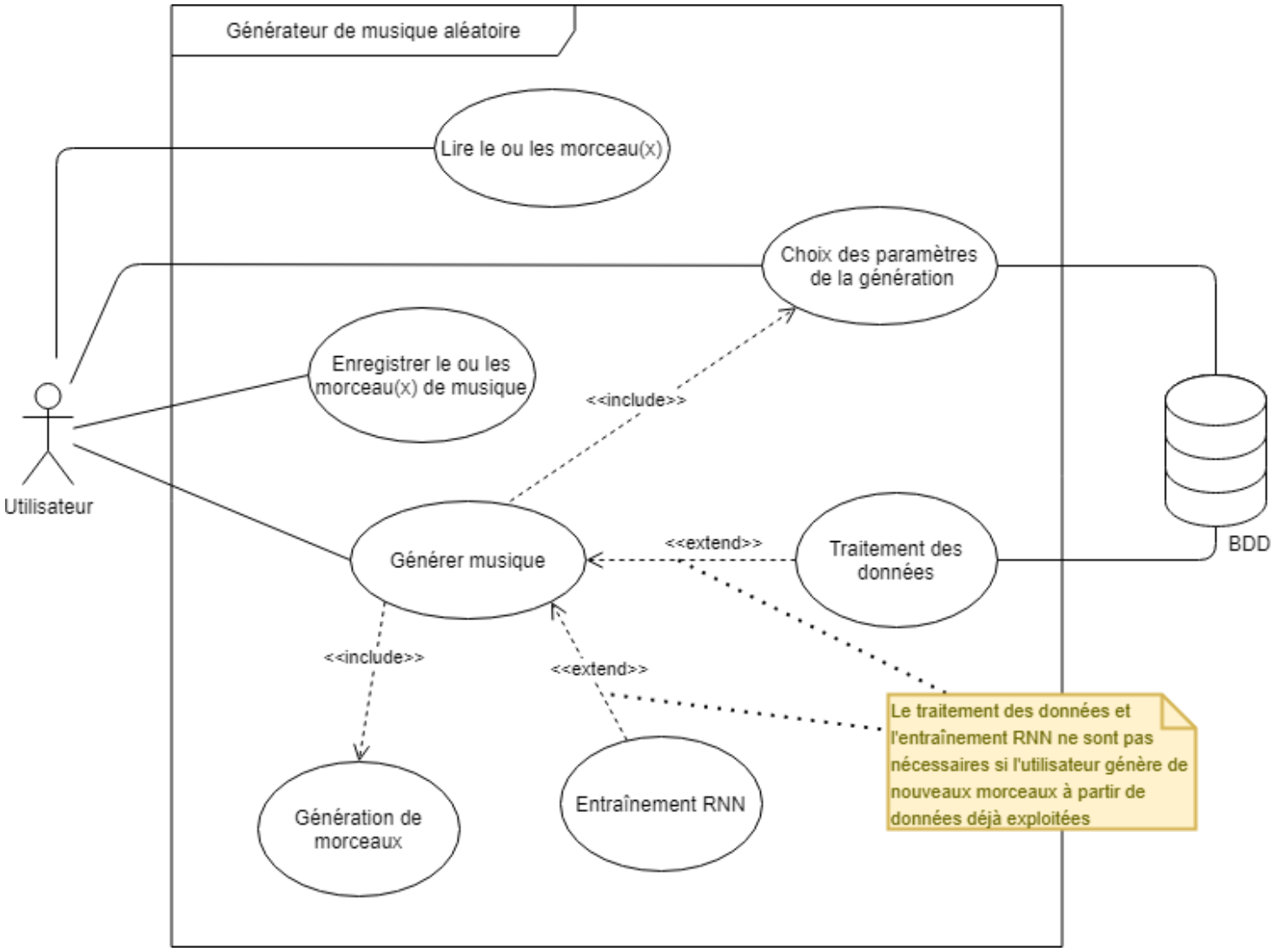
Plus précisément, ce document traite d'une part de l'analyse fonctionnelle du logiciel et propose l'ensemble de ses cas d'utilisation. Cela se présente par un diagramme des cas d'utilisations, leur hiérarchie ainsi que la description détaillée des principaux cas.

D'autre part, nous présentons l'ensemble de nos choix techniques et la manière dont cela a été implémenté.

Modifications effectuées depuis la version 4.0 de ce document :
Architecture (II pages 9-11) Base de données (IV page 14) Interface (VI pages 16-18) Sauvegarde des paramètres (VII page 19) RNN (VIII pages 20-22) Avancée (IX page 23) Amélioration futures (X page 24) De manière générale, les captures d'écran de l'application ont été mises à jour

I - Cas d'utilisation

1 - Diagramme de cas d'utilisation



Les acteurs :

- L'utilisateur représente un utilisateur "standard" qui pourrait être amené à utiliser notre logiciel.
- La base de données quant à elle,représente l'endroit où seront stockés les jeux de données (par exemple : un disque dur, une clé USB, un serveur NAS etc ..).

1.1 - Liste des cas d'utilisation

Voici la liste des cas ainsi qu'un résumé de leur scénario :

- Lire le ou les morceau(x) : L'utilisateur écoute la musique qui a été générée
- Choix des paramètres de la génération : L'utilisateur sélectionne les différents paramètres nécessaires pour la génération
- Enregistrer le ou les morceau(x) : L'utilisateur sauvegarde les morceaux de musique qui ont été générés
- Générer musique : L'utilisateur lance la génération de la musique
- Traitement des données : Les fichiers MIDI fournis par la base de données sont traités par le système afin de générer de la musique
- Génération de morceaux : Le système utilise le réseau de neurones entraîné pour générer de la musique
- Entrainement RNN : Le système entraîne le réseau de neurones à l'aide d'un jeu de données pour pouvoir générer de nouveaux morceaux

1.2 - Hiérarchisation des cas d'utilisation

Les cas d'utilisation du diagramme précédent peuvent être hiérarchisés selon les trois niveaux de priorité suivants :

1. Génération des morceaux, Traitement des données, Entraînement RNN, Lire les morceaux
2. Générer musique, Choix des paramètres de la génération
3. Enregistrer les morceaux

Le premier niveau est constitué du cœur du logiciel, les fonctionnalités principales.

Le deuxième niveau est constitué des différentes parties de l'interface utilisateur.

Le dernier niveau est composé des cas dont l'utilisation n'est pas indispensable au bon fonctionnement du logiciel mais permettent à l'utilisateur d'avoir un meilleur confort d'utilisation.

2 - Description détaillée des cas de niveau 1

2.1 - Cas d'utilisation : Génération des morceaux

Niveau : Sous-fonction

Résumé : Le système utilise le réseau de neurones entraîné pour générer de la musique

Précondition : La génération de musique a été lancée. Si les données sont nouvelles, les cas d'utilisations Traitement des données et Entraînement RNN ont été lancés et se sont terminés.

Scénario principal :

1. le système utilise le réseau RNN pour générer de la musique
2. le système met en forme les musiques créées par le réseau RNN pour qu'elles soient au format MIDI et wav
3. Le système supprime les résultats du réseau RNN autre que les fichiers MIDI et wav.

2.2 - Cas d'utilisation : Traitement des données

Niveau : Sous-fonction

Résumé : Les fichiers MIDI fournis par la base de données sont traités par le système afin de générer de la musique

Acteur secondaire : La base de données

Précondition : Les paramètres ont été renseignés

Scénario principal :

1. Le système convertit les fichiers MIDI en format CSV
2. Les fichiers CSV sont convertis en format exploitable pour l'entraînement RNN

2.3 - Cas d'utilisation : Entraînement RNN

Niveau : Sous-fonction

Résumé : Le système entraîne le réseau de neurones à l'aide d'un jeu de données pour pouvoir générer de nouveaux morceaux

Précondition : Les données ont été formatées pour être utilisées par le réseau de neurones

Acteur secondaire : La base de données

Scénario principal :

1. Le système envoie les données au bon format au réseau de neurones
2. Le réseau de neurones identifie les similitudes entre les morceaux

2.4 - Cas d'utilisation : Lire le ou les morceau(x)

Niveau : But utilisateur
Résumé : L'utilisateur écoute la musique qui a été générée
Acteur primaire : L'utilisateur
Précondition : L'utilisateur a lancé la génération de musique
Scénario principal :

1. L'utilisateur utilise le lecteur de musique pour écouter les morceaux
2. Le système gère la lecture du morceau

3 - Description détaillée des cas de niveau 2

3.1 - Cas d'utilisation : Générer musique

Niveau : But utilisateur
Résumé : L'utilisateur lance la génération de la musique
Acteur primaire : L'utilisateur
Acteur secondaire : la base de données
Scénario principal :

1. L'utilisateur lance la génération
2. Le choix des paramètres est appelé
3. Le système lance le traitement des données
4. Le système récupère les données traitées
5. Le système lance l'entraînement RNN
6. Le système lance la génération de morceaux
7. Le système renvoie à l'utilisateur la musique générée

Extensions :

4.a. Ces données ont déjà été traitées

4.a.1. Le système passe directement à l'entraînement RNN

6.a. L'entraînement RNN a déjà été effectué pour ces données

6.a.1. Le système passe directement à la génération de morceaux

3.2 - Cas d'utilisation : Choix des paramètres de la génération

Niveau : Sous-fonction
Résumé : L'utilisateur sélectionne les différents paramètres de la génération
Acteurs secondaires : L'utilisateur, la base de donnée
Scénario principal :

1. L'utilisateur sélectionne le nombre de morceaux à générer, des morceaux de référence, indique la durée d'un morceau,la tonalité et

choisit entre “Rythme uniquement”, “Rythme et mélodie” ou “Polyphonie”.

2. Le système valide les paramètres de l’utilisateur

Extension(s) :

2. a) Un des paramètres est incorrect

2. a) 1. Le système informe l’utilisateur qu’un ou plusieurs paramètres n’est pas valide et on repasse à l’étape 1.

4 - Description détaillée des cas de niveau 3

4.1 - Cas d’utilisation : Enregistrer le ou les morceau(x)

Niveau : But utilisateur

Résumé : L’utilisateur sauvegarde le morceau de musique qui a été généré

Acteur primaire : L’utilisateur

Précondition : La musique a été générée

Scénario principal :

1. L’utilisateur renseigne un répertoire où il souhaite stocker les morceaux et valide
2. Le système valide l’information et indique que le téléchargement a été effectué avec succès

II - Architecture MVC

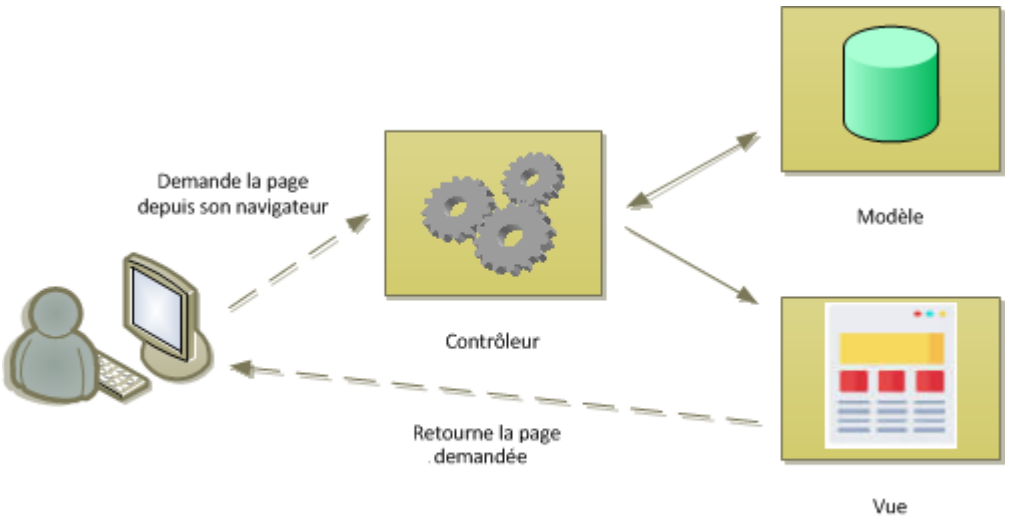
Pour ce projet, nous avons décidé d'utiliser une architecture Modèle-Vue-Contrôleur (MVC). Cette architecture peut être utilisée dès lors que le projet possède une interface graphique avec l'utilisateur.

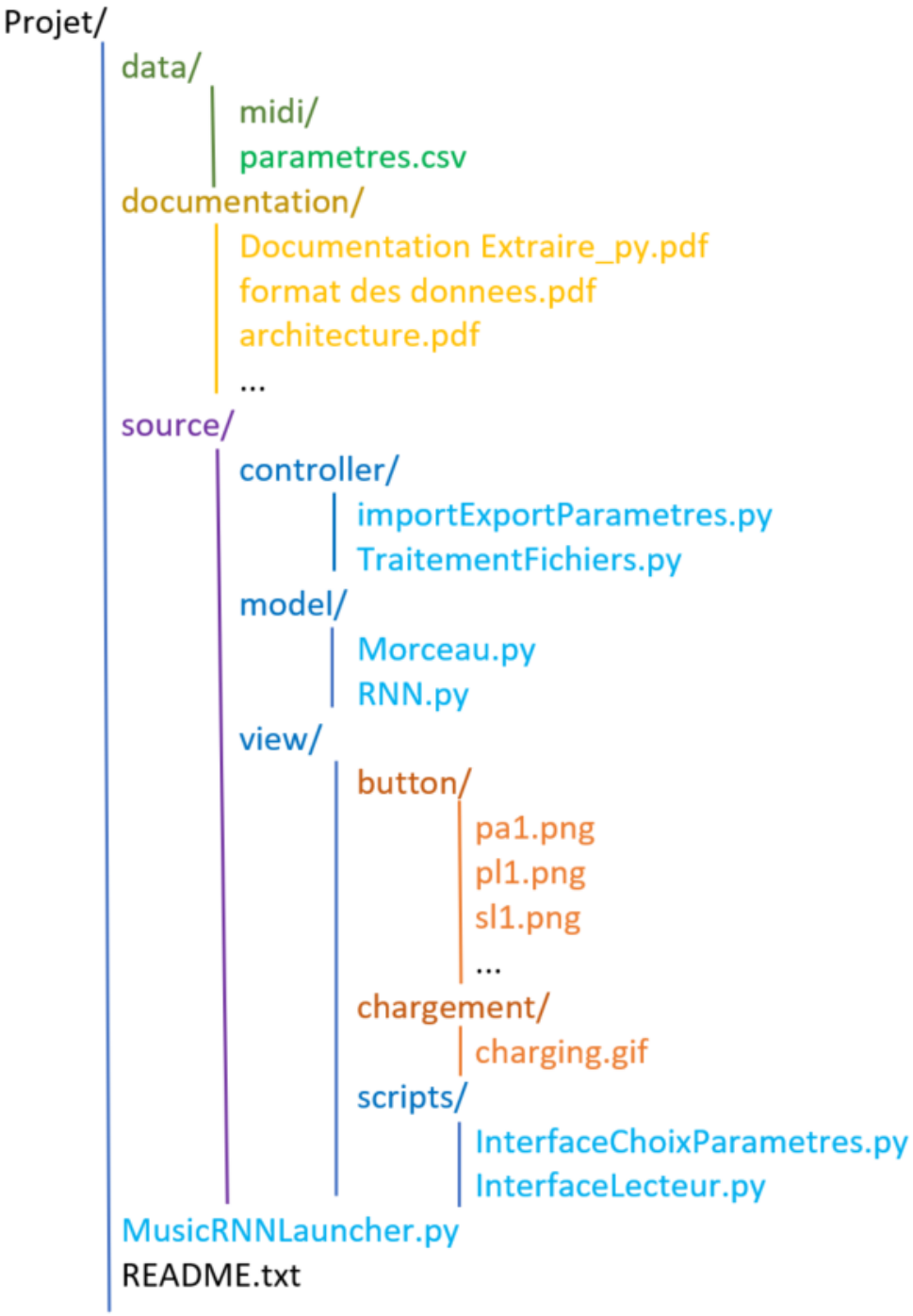
Le Modèle sert à gérer les données. Son rôle est d'aller récupérer les informations «brutes» dans la base de données, de les organiser et de les assembler pour qu'elles puissent ensuite être traitées par le contrôleur

La Vue se concentre sur l'affichage. Elle ne fait presque aucun calcul et se contente de récupérer des variables pour savoir ce qu'elle doit afficher.

Enfin, le Contrôleur gère la logique du code qui prend des décisions. C'est en quelque sorte l'intermédiaire entre le modèle et la vue : le contrôleur va demander au modèle les données, les analyser, prendre des décisions et renvoyer le texte à afficher à la vue.

Le schéma suivant décrit le fonctionnement de cette architecture :





Code couleur :

bleu : les scripts pythons
jaune : la documentation
orange : les images/gif pour l'interface
vert : les données utiles au bon fonctionnement du programme

Description des quelques scripts importants (dans l'ordre du schéma) :

ImportExportParametres.py : gère l'import et l'export des paramètres

TraitementFichiers.py : gestion du traitement des fichier midi jusqu'à leur format utilisable par le RNN et écriture des résultats dans des fichiers MIDI

Morceau.py : classe qui permet d'envelopper un fichier midicsv dans un objet
RNN.py : tous les codes relatifs au RNN

InterfaceChoixParametres.py : la partie principale de l'interface (selection des parametres)

InterfaceLecteur.py : la partie de l'interface qui gère l'affichage et la lecture des morceaux

MusicRNNLauncher.py : le fichier pour lancer l'application

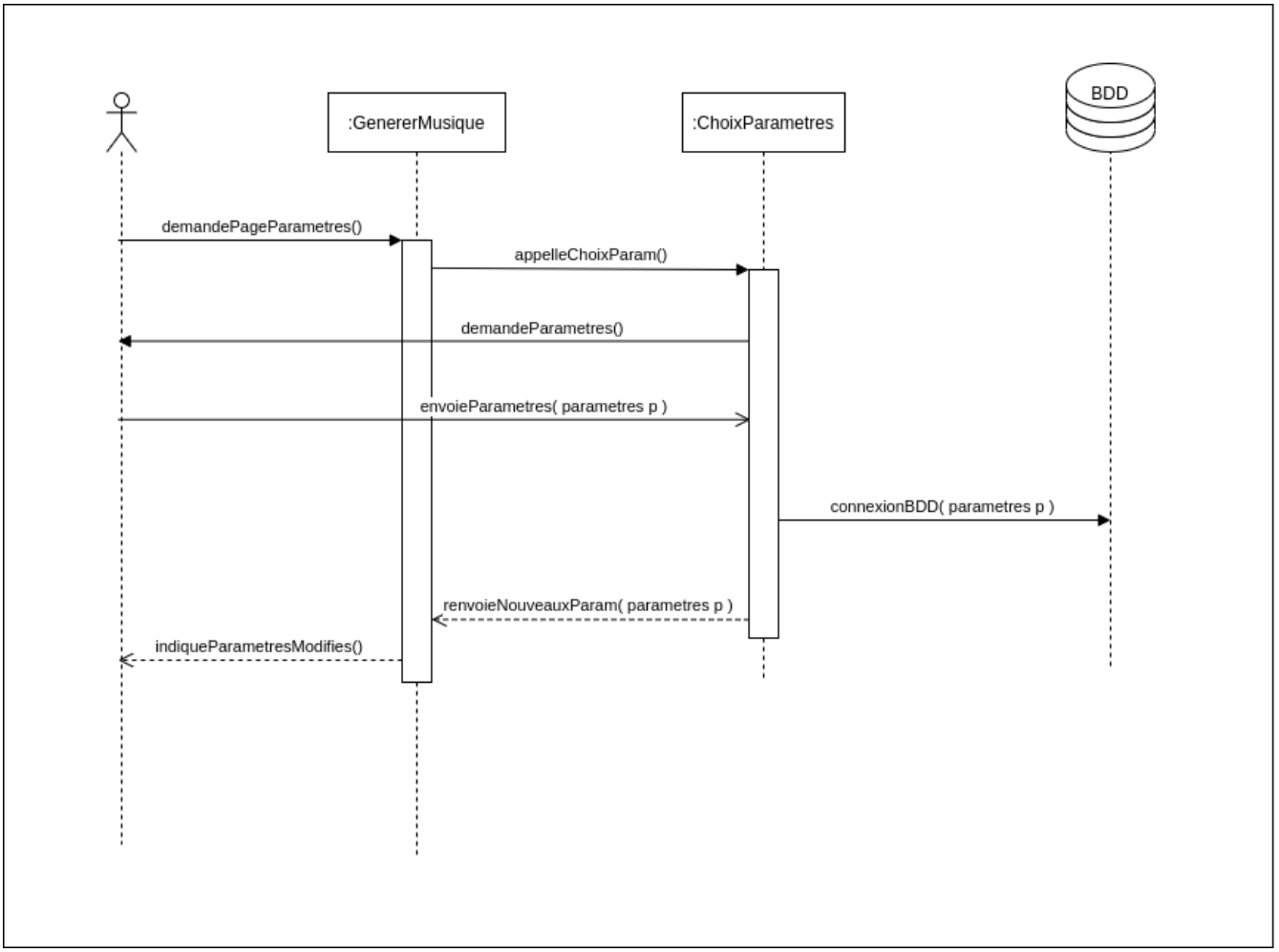
Dans data/ on stocke le fichier de sauvegarde des paramètres de l'utilisateur ainsi que l'ensemble des transformations effectuées sur les fichiers .mid de la base de données.
La structure du dossier midi est la suivante :

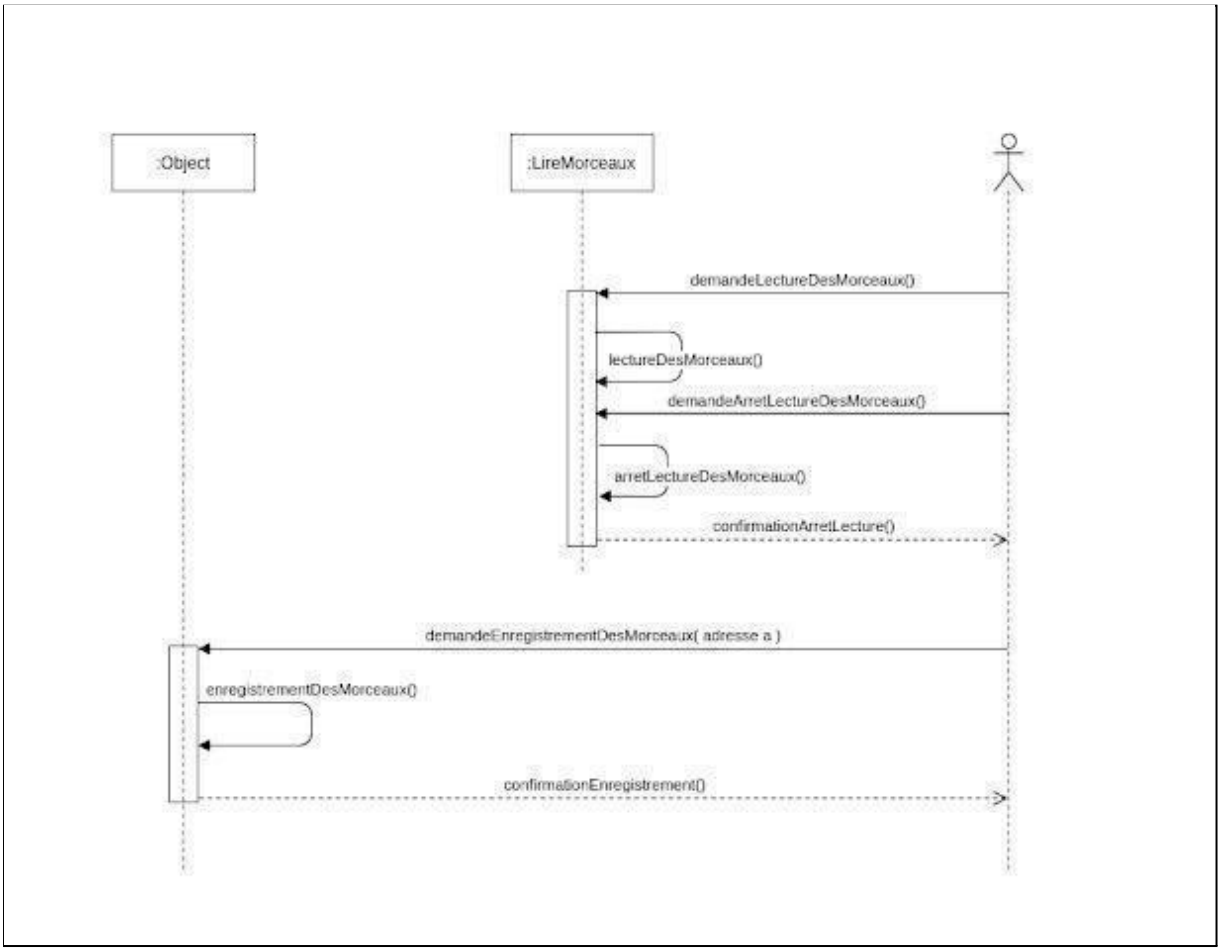
midi/
    CSV/
    Conversion_rythme/
    Resultat/
    x.mid
    y.mid
    …

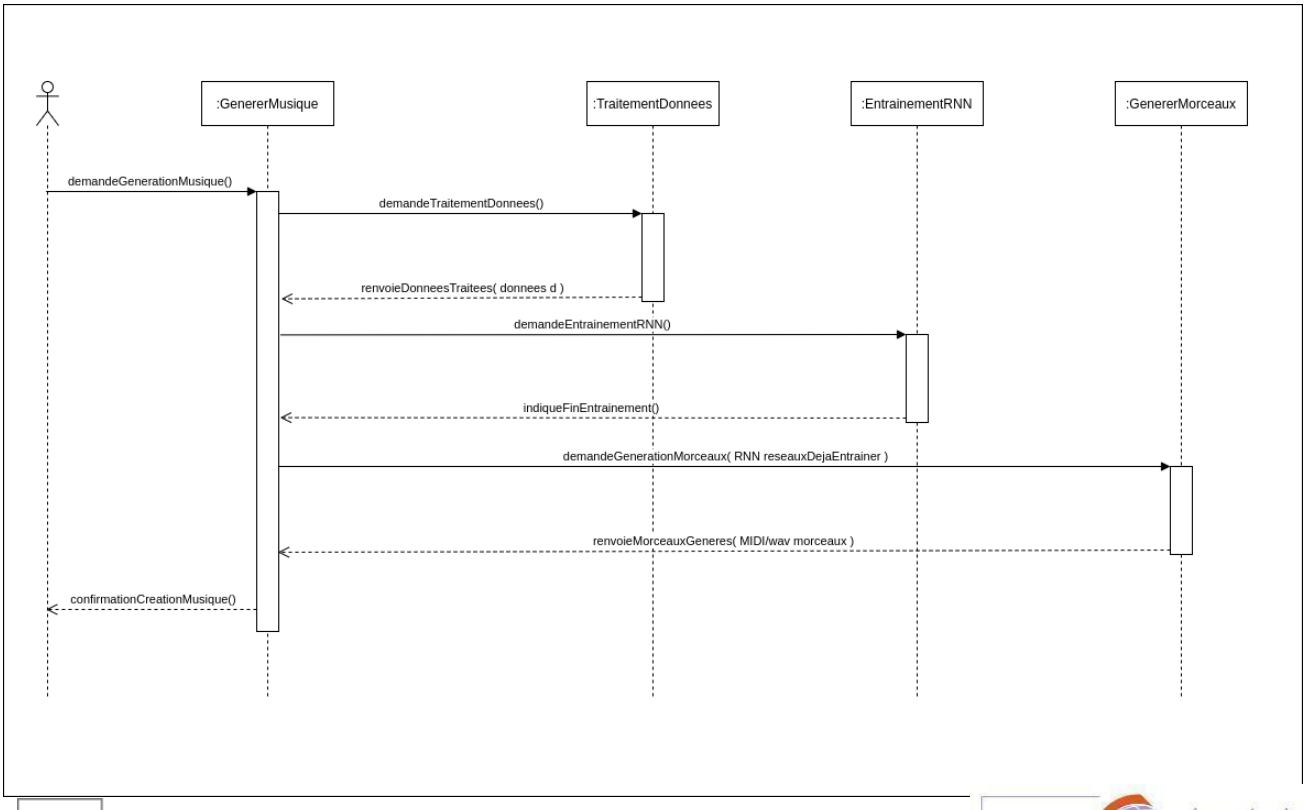
En orange, on retrouve les fichiers de la base de données, dans CSV on trouve ces mêmes fichiers convertis en .csv, dans conversion rythme c'est leur version utilisable par le RNN et enfin dans Resultat on place les fichiers renvoyés en sortie du RNN. Cela permettra, si l'entraînement est stoppé pour une quelconque raison, de parcourir les sous dossiers au prochain entraînement et de ne pas refaire tous les traitements des fichiers de la BDD.

III - Diagramme de séquence

Les diagrammes suivants représentent le diagramme de séquence du système. Il a été divisé en 3 parties pour le rendre plus lisible.




:Object
:LireMorceaux
demandeLectureDesMorceaux()
lectureDesMorceaux()
demandeArretLectureDesMorceaux()
arretLectureDesMorceaux()
confirmationArretLecture()
demandeEnregistrementDesMorceaux(adresse a)
enregistrementDesMorceaux()
confirmationEnregistrement()


:GenererMusique
:TraitementDonnees
:EntrainementRNN
:GenererMorceaux
demandeGenerationMusique()
demandeTraitementDonnees()
renvoieDonneesTraitees(donnees d)
demandeEntrainementRNN()
indiqueFinEntrainement()
demandeGenerationMorceaux(RNN reseauxDejaEntrainer)
renvoieMorceauxGeneres(MIDI/wav morceaux)
confirmationCreationMusique()

IV - Base de données

Nous avons remarqué que certaines propriétés sont nécessaires à notre base de données pour que l'entraînement du RNN soit possible et efficace. Le traitement de la base de données n'étant pas un point central du projet, nous n'avons pas eu le temps de travailler suffisamment dessus pour que tout soit fonctionnel dans 100% des cas.

Notre base de données doit être au format midi. Les fichiers sur lesquels on veut entraîner le RNN doivent être tous dans le même dossier (les sous dossiers ne seront pas pris en compte).

Les fichiers midi doivent correspondre à des morceaux proches musicalement (éviter de mélanger les modes majeur et mineur,...) et il faut éviter au maximum les changements de tonalité dans les morceaux, le RNN ne comprend pas encore cette particularité.

De plus si le fichier midi correspond à un morceau polyphonique le RNN ne récupérera que la piste 2 du fichier (si cette piste est vide on entraînera pas le RNN et si cette piste contient la ligne de basse alors on entraînera le RNN sur une basse). Solution, faire passer manuellement au préalable les fichiers de la base de données dans un script python qui pour chaque ligne du fichier CSV remplace le premier élément par 2 (ainsi il n'y aura qu'une seule piste dans le fichier et celle-ci sera reconnue par le RNN). Le script python qui fait cette manipulation sera fourni, mais il faut noter que dans ce cas la les morceaux polyphoniques seront traités comme des mélodies simples (le RNN ne saura pas que ce sont des morceaux polyphoniques) et les résultats en sortie du RNN seront de même nature.

Pour entraîner le RNN sur des mélodies simples il faut que la base de données contienne des fichiers midi avec uniquement des mélodies simples dedans.

Enfin nous supposons que la base de données a été pré traitée avant d'être fourni à l'application, dans le cas contraire la faute revient à l'utilisateur si le programme ne fonctionne pas correctement.

V - Traitement des fichiers MIDI

Nous avons décidé que, pour traiter les fichiers MIDI, nous allions les convertir en .csv. C'est un format qui nous permet d'extraire les informations du fichier MIDI plus facilement et surtout qui le rend plus lisible. Pour cela, nous avons utilisé la bibliothèque python py_midicsv. Après traitement, le fichier de sortie se présente de la façon suivante :

Les premières lignes du fichier nous donnent des informations sur la musique notamment le compositeur, la date de réalisation, le copyright, la durée etc..

Ici, on remarque que le titre s'appelle "Espana Op. 165" et qu'il date de 2001.

```
0, 0, Header, 1, 8, 480
1, 0, Start_track
1, 0, Title_t, "Espana Op. 165"
1, 0, Title_t, "Tango"
1, 0, Copyright_t, "Copyright © 2001 by Bernd Krueger"
1, 0, Text_t, "Isaac Albeniz"
1, 0, Text_t, "Andantino"
1, 0, Text_t, "Fertiggestellt 28.01.2001\012"
1, 0, Text_t, "Normierung: 23.12.2002\012"
1, 0, Text_t, "Update am 29.8.2010\012"
```

Puis, dans la suite du fichier, nous accédons aux données musicales. Suivant l'information, une ligne possède un certain format. Par exemple, "Track, Time, Type, **Instrument_name_t"** lorsqu'il est spécifié le nom de l'instrument. Dans le cas ci-dessous, on voit que c'est du piano joué de la main gauche.

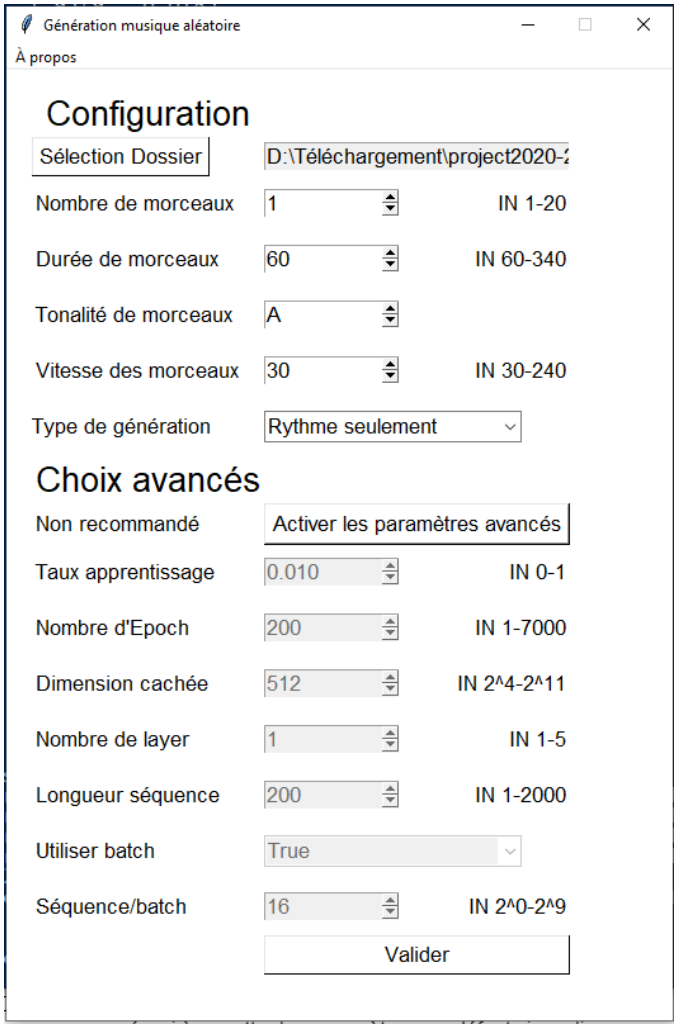
Pour les notes, on aura une ligne de ce type : Track, Time, Type, Note_on_c, Channel, **Note**, Velocity. Les notes dans un fichier MIDI sont codées par un entier compris entre 0 et 127.

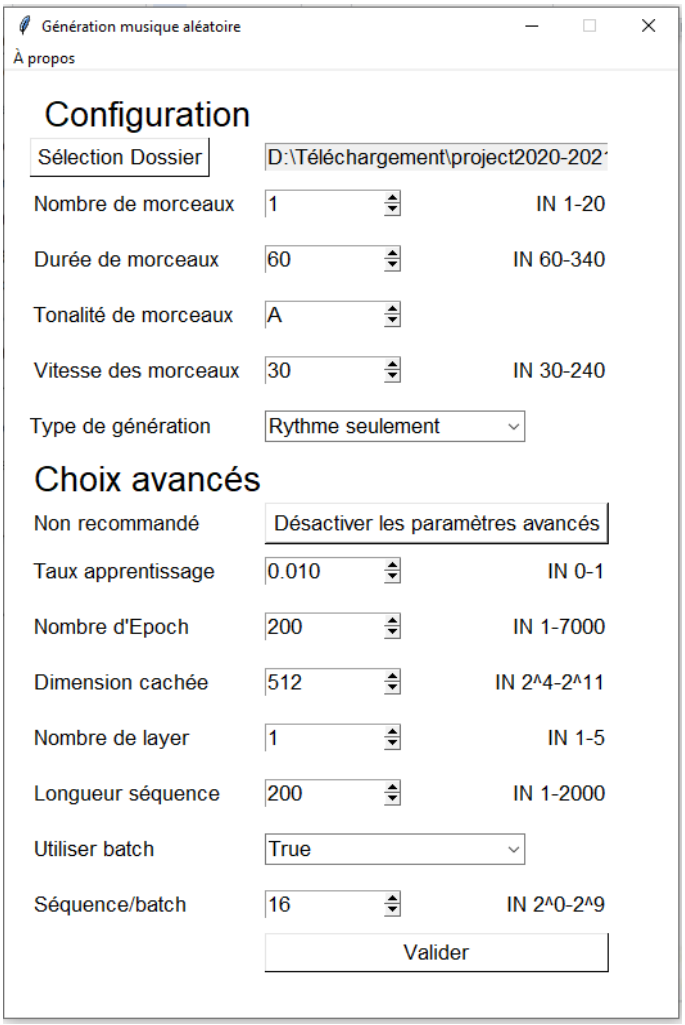
```
3, 0, Start_track
3, 0, Title_t, "Piano left"
3, 0, Program_c, 0, 0
3, 0, Note_on_c, 0, 38, 53
3, 0, Note_on_c, 0, 50, 53
3, 720, Note_on_c, 0, 50, 0
3, 720, Note_on_c, 0, 38, 0
```

VI - Interface Homme Machine (IHM)

1 - Sélection des paramètres

La première page permet à l'utilisateur de rentrer les paramètres de la génération.





Comme indiqué dans le cahier des charges, il peut :

- ➢ Sélectionner un dossier contenant les jeux de données au format .midi grâce au bouton "Choisir" qui ouvre une seconde fenêtre permettant de naviguer dans les répertoires de l'ordinateur.
- ➢ Indiquer le nombre de morceaux à générer qui est initialisé à 1.
- ➢ Spécifier la durée d'un morceau. Celle-ci est initialisée à 60 secondes.
- ➢ Spécifier la tonalité d'un morceau. A, A#, B, …

- Spécifier la vitesse d'un morceau. Exprimé en bpm entre 30 et 240.
- Préciser le type de la génération entre "Rythme seulement", "Rythme et mélodie","Polyphonie". (Polyphonie non fonctionnel)

Nous avons décidé d'ajouter une fonctionnalité qui n'était pas prévue dans le cahier des charges mais qui nous semblait pertinente pour un certain type d'utilisateur. Ainsi, cliquer sur le bouton "Activer les paramètres avancés" donne la main sur les paramètres de configuration du réseau de neurones. Cette partie est donc réservée pour des utilisateurs expérimentés qui veulent obtenir un meilleur résultat lors de la génération. Pour un utilisateur ayant peu d'expérience avec les réseaux de neurones, nous conseillons de conserver les paramètres par défaut qui permettent d'obtenir des résultats satisfaisants dans la plupart des cas.

De plus, nous avons laissé la possibilité à l'utilisateur d'accéder directement au lecteur de musique via le bouton "Accès au lecteur" sans pour autant devoir générer de nouveau morceaux

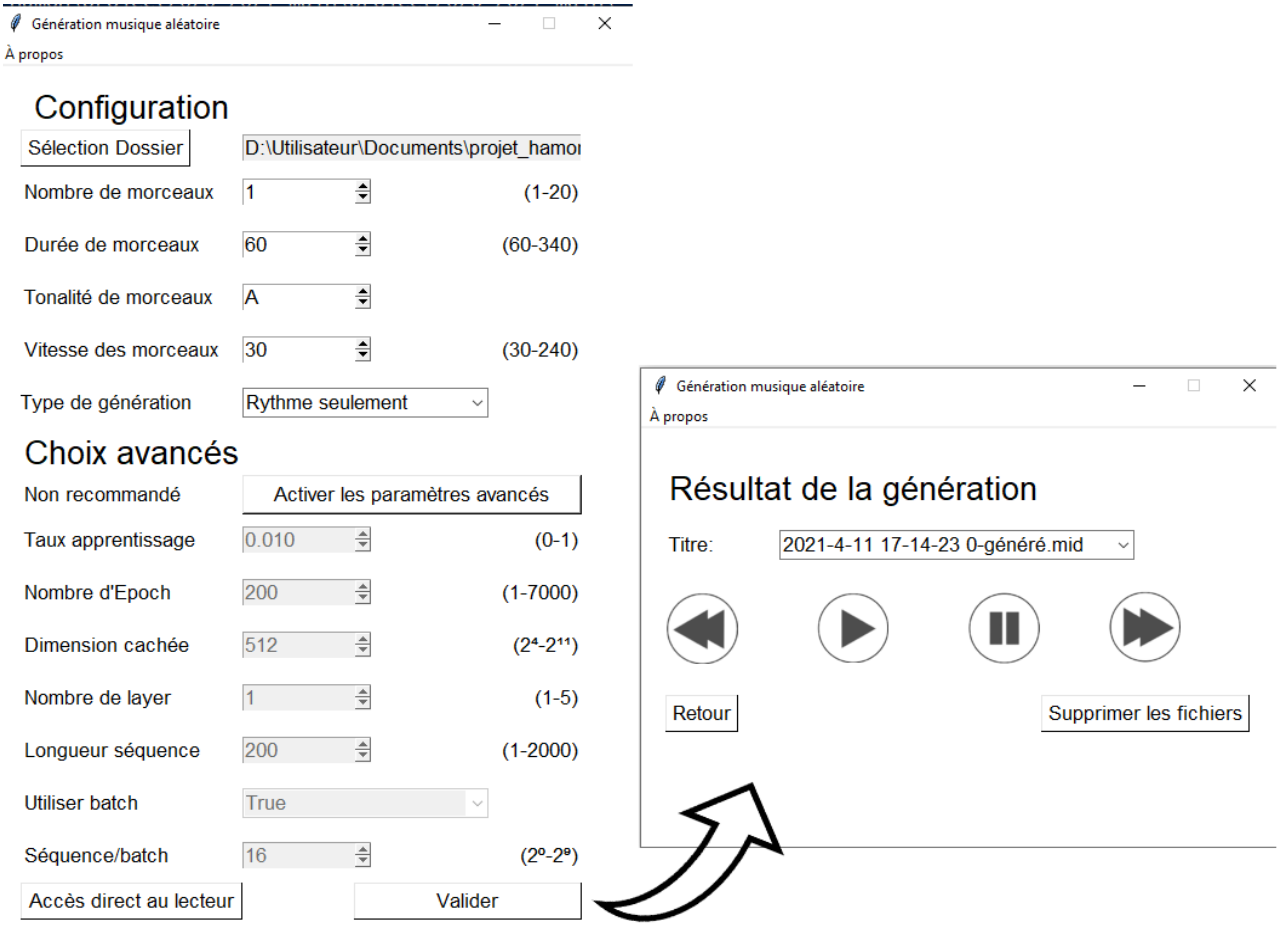
2 - Lecteur audio

Une fois que l'utilisateur a rentré tous les paramètres, il clique sur le bouton "Valider" pour lancer la génération. Du côté de l'interface, le bouton "Valider" devient "Chargement…" et de l'autre, l'état d'avancement de l'apprentissage du réseau de neurones est affiché dans la fenêtre du terminal dans laquelle l'application a été lancée. L'interface du lecteur s'ouvre une fois la génération terminée.

Comme nous voulions une interface très intuitive et "user friendly", le lecteur est très basique. En effet, l'utilisateur peut sélectionner une musique, mettre sur pause ou encore avancer/reculer d'un morceau.

En bas de la page, il peut soit retourner à la page des paramètres grâce au bouton "Retour" pour lancer une nouvelle génération de morceaux avec les mêmes paramètres ou en les modifiant, soit choisir de supprimer tous les fichiers issus du générateur via le bouton "supprimer les fichiers".

Le schéma suivant reprend la dernière version de l'interface :



VII - Sauvegarde des paramètres

Les paramètres entrés par l'utilisateur sont sauvegardés dans des fichiers .csv, leur format est le suivant :

	A	B	C	D	E	F
1	URL_Dossier	NombreMorceaux	DureeMorceaux	TonaliteMorceaux	VitesseMorceaux	TypeGeneration
2						
3	C:\monAdresse	1	60	A	30	Rythme et mélodie

	G	H	I	J	K	L	M
1	TauxApprentissage	NombreEpoch	NombreDimensionCachee	NombreLayer	LongeurSequence	BatchBool	NombreSequenceBatch
2							
3	0.010	20	512	1	200	TRUE	16

URL_Dossier : Spécifie le répertoire où est stockée la base de données

NombreMorceaux : Nombre de morceaux à générer

DureeMorceaux : Longueur des morceaux générés (en nombre de notes)

TonaliteMorceaux : Tonalité des morceaux générés

VitesseMorceaux : Vitesse des morceaux générés (en bpm)

TypeGeneration : Spécifie si l'utilisateur attend du rythme ou une mélodie

TauxApprentissage : Taux d'apprentissage du modèle

NombreEpoch : Nombre d'étapes d'entraînement

NombreDimensionsCachee : Nombre de dimensions cachées du RNN

NombreLayer : Nombre de couches du RNN

LongueurSequence : Longueur des séquences à passer au RNN

BatchBool : True si l'utilisateur veut entraîner le RNN avec des lots, False sinon

NombreSequenceBatch : Nombre de séquences passées au RNN

VIII - Entraînement et génération avec un RNN

1 - RNN “simple” et entraînement

Lors d’une étape précédente de ce projet, nous avions commencé à faire des premiers tests sur un RNN simple qui prenait en argument du texte en français et générait du texte en retour après s’être entraîné dessus pendant un certain temps.

Les tests nous ont montré que l’utilisation d’un RNN aussi simple ne donnait pas de bons résultats et avait tendance à répéter le texte passé en entrée. Nous avons donc modifié quelques paramètres, notamment en augmentant la taille de la couche cachée ou en modifiant le RNN en GRU. Dans notre cas, cela a permis un meilleur apprentissage et de meilleures performances en sortie (pour l’apprentissage sur un texte en français).

Nous allons donc conserver ce RNN et ses paramètres associés pour l’apprentissage sur le format d’encodage des fichiers MIDI (voir VII - 3 - Format des données).

Le texte d’entraînement est d’abord découpé en morceaux d’une taille variant entre 100 et 200 caractères. Chacun de ces morceaux est ensuite transformé en deux tenseurs : le premier (torseur d’entrée) contient les N-1 premiers caractères du morceau ; le second (tenseur de sortie attendu) contient les N-1 derniers caractères du même morceau. Lors de l’entraînement à proprement parler, on sélectionne au hasard un certain nombre de couples tenseur d’entrée et de sortie associés. Ce lot va ensuite être passé au RNN qui va s’entraîner dessus en générant un tenseur (basé sur le tenseur d’entrée) qui va ensuite être comparé au tenseur de sortie attendu. Après chaque entraînement sur un lot, le Réseau met à jour ses paramètres avant de s’entraîner sur le lot suivant.

2 - RNN final

Avec les expérimentations précédentes, nous avions une base pour complexifier et améliorer notre RNN. Avant l'entraînement, le RNN reçoit en paramètre les séquences sur lesquelles il va devoir s'entraîner. Il crée un dictionnaire qui relie chaque note unique (voir format des données ci-dessous) avec un entier, cela lui sera utile plus tard. Les données d'entrées sont ensuite découpées en données de test et en données d'entraînement. Chaque séquence obtenue est ensuite subdivisée en deux séquences : une séquence dite "input" et une autre dite "target". Ces séquences sont ensuite encodées grâce au dictionnaire créé plus tôt (c'est la méthode "one-hot-encode").
Vient ensuite l'entraînement : en fonction des choix de l'utilisateur, les séquences input et target sont passées au RNN soit toutes ensemble, soit par lots (ce sont des batchs). Le RNN génère des séquences qu'il compare ensuite aux séquences "target" et il calcule une erreur (la loss, ou perte). Il se sert ensuite de cette erreur pour corriger sa structure interne par la méthode de la rétro-propagation.

Le modèle est maintenant entraîné et il faut maintenant générer autant de morceaux que l'utilisateur a choisi. Pour cela, on commence par choisir une note aléatoirement dans le dictionnaire créé plus tôt et on demande au RNN de compléter cette séquence. Le RNN va alors évaluer cette note et générer un vecteur de probabilités associé à cette note de la taille du dictionnaire. Chaque note a donc une probabilité particulière d'arriver après celle que l'on a choisie. Le RNN en sélectionne une après un tirage prenant en compte de probabilités et continue d'évaluer la liste des notes de la même manière jusqu'à ce qu'il arrive à la longueur désirée.

Les morceaux sont ensuite renvoyés, convertis en midi et l'utilisateur peut les écouter.

3 - Format des données

Nous avons défini la façon dont seront représentées les données pour le RNN. Pour ce qui est du RNN pour le rythme simple et pour la mélodie simple, notre première approche est la suivante.
Chaque note sera représentée par un mot de deux à quatre lettres :

- La première lettre correspond au temps entre le début de la note précédente et le début de la note actuelle (pour ne pas conserver la valeur en milliseconde par rapport à l'axe du temps du début de la note, ce qui serait un entier qui ne ferait que grandir inutilement).
- La deuxième lettre correspond à la durée de la note actuelle (croche, blanche, …). L' affectation de chaque lettre à une valeur (noire, croche, …) est détaillée plus bas.

- La troisième (et quatrième) lettre correspond à la hauteur de la note actuelle (Do, Re, …, Silence).

Pour ce qui est de l'affectation des lettres:

- Nous supposerons que la note la plus courte que nous pouvons jouer est une quadruple croche. 64 quadruple croches durent aussi longtemps qu'une ronde (Ou 32 triples croches). Avec un alphabet de 64 lettres (A-Z, a-z, 0-9 et -,+) nous devrions pouvoir représenter toutes les valeurs nécessaires en supposant qu'aucune note ne dure plus longtemps qu'une ronde.

- Comme pour le point précédent on peut supposer qu'une note ne dure pas plus qu'une ronde. Avec 26 valeurs on peut représenter les longueurs suivantes (triées dans l'ordre décroissant, "a" ne serait donc pas très utile).

a	ronde pointée	n	croche triolet
b	ronde	o	double croche pointée
c	blanche pointée	p	double croche
d	blanche	q	croche quintolet
e	ronde quintolet	r	double croche triolet
f	blanche triolet	s	triple croche pointée
g	noire pointée	t	triple croche
h	noire	u	double croche quintolet
i	blanche quintolet	v	triple croche triolet
j	noire triolet	w	quadruple croche pointée
k	croche pointée	x	quadruple croche
l	croche	y	triple croche quintolet
m	noire quintolet	z	quadruple croche triolet

- On supposera dans un premier temps que pour la hauteur on n'a besoin que d'un octave (de Do à Do) et si on a besoin de plus on pourra toujours ajouter un caractère supplémentaire (un chiffre) qui nous dit quel est l'octave concerné (Do1 ou Do5 comme pour un piano).

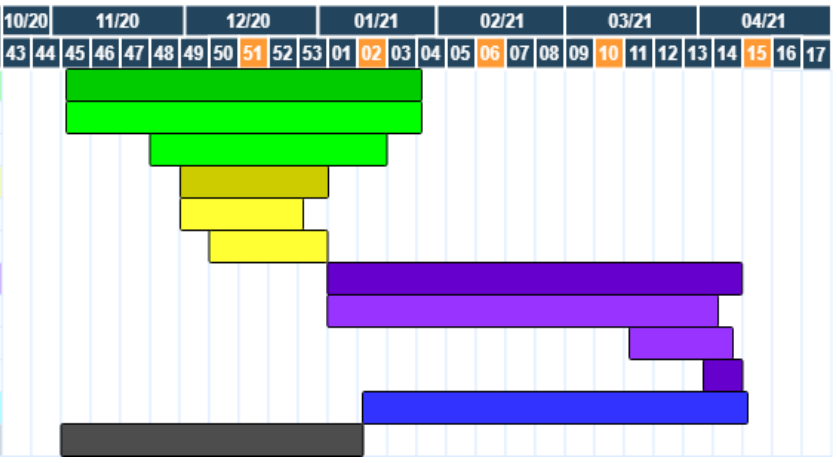
  a = Do, b = Do#, c = Re, …, l = Si, m = silence.

Remarques :

1. Pour le vocabulaire musical nous fournirons une annexe permettant de rappeler quelques notions de base de musique.
2. Nous représentons ici l'encodage par des lettres pour rendre la lecture plus simple pour nous mais utiliser un encodage sur un alphabet de 64 lettres ou utiliser des entiers codés sur 7 bits est équivalent.
3. Pour la représentation du rythme on n'a pas besoin de conserver la hauteur des notes donc les valeurs seront encodées sur deux caractères, par conséquent, on ne note pas les silences (il seront implicite dans l'encodage).

IX - Avancée

Tâche	Jours/Homme	Semaine début	Semaine Fin
Analyse fonctionnelle	15	45	03
Diagrammes de cas d'utilisation & de séquences	12	45	03
Maquette interface	8	48	02
Architecture logicielle	24	49	53
Conception	12	49	52
Prototypage	12	49	53
Développement	55	01	14
Implémentation	30	01	12
Intégration	12	05	13
Déploiement	13	10	13
Documentation	12	02	13
Formation	7	45	01



Concernant l'avancée de ce projet lors de cette dernière phase de construction, les principales fonctionnalités demandées dans le cahier des charges ont été implémentées et sont opérationnelles. De plus, la documentation concernant le code et son fonctionnement a été rédigée dans son intégralité.

Lors d'une réunion avec nos clients, nous avons même pu leur transmettre notre application afin de vérifier son bon fonctionnement dans leur environnement et ce avec succès.

Cependant, nous n'avons pas eu le temps d'implémenter le RNN qui gère les rythmes et mélodies polyphoniques ainsi que la cartographie des compositeurs en fonction des rythmes de leurs morceaux (demande des clients qui n'était initialement pas prévue).

Nous pouvons expliquer ce manque de temps par le fait que la formation à la compréhension et l'utilisation d'un réseau de neurones récurrent nous a pris plus de temps que prévu.

De plus, n'étant pas expérimenté dans ce domaine, certains bug ont été très difficile à résoudre d'une part par le fait de trouver d'où il venait mais surtout savoir comment le corriger.

X - Améliorations futures

Nous n'avons pas eu le temps d'implémenter convenablement l'entraînement du RNN pour générer des morceaux polyphoniques même si celui qui nous sert pour les mélodies simples semble capable dans une certaine mesure d'y arriver.

Nous n'avons pas non plus eu le temps de nous pencher sur la cartographie des compositeurs en fonction des rythmes de leurs morceaux.

Quelques pistes d'améliorations supplémentaires pour le future :

- Gérer le traitement/correction de la base de données directement dans l'application et non pas attendre que le RNN s'entraîne sur des données de mauvaise qualité pour corriger celles-ci manuellement.
- Ajouter plus de paramètres complexes pour l'entraînement du RNN.
- Proposer d'autres types de réseaux, pas uniquement des RNN.
- Effectuer un vrai traitement sur une grosse base de données pour pouvoir générer des résultats encore plus pertinents.
- Implémenter une fonction permettant de sauvegarder l'état du RNN afin de pouvoir prolonger plus tard son entraînement.
- Implémenter une fonction de parcours qui permette de récupérer les fichiers midi présents dans les sous-dossiers de notre base de données.

Conclusion

Ce projet nous a permis d'aborder pour la première fois les différentes phases permettant de mener à terme un projet. Ainsi, nous avons conduit ce projet de la rédaction du cahier des charges en fonction des besoins de nos clients jusqu'à la réalisation et l'installation de l'application sur les machines des clients.

Réaliser ce projet fut l'occasion pour nous d'utiliser certaines compétences que nous avions acquises pendant notre cursus et plus particulièrement celles liées à la modélisation et la représentation de notre application.

Nous avons dû apprendre à travailler en équipe ce qui implique notamment l'attribution à chacun d'un rôle spécifique à ce projet et le partage des tâches. Nous avons mis en place pour la première fois un système de gestion de version (Git) et réalisé au total plus de 160 commits.

Le sujet du projet nous a beaucoup intéressé et nous a permis d'avoir une première approche des réseaux de neurones à travers un exemple concret. Nous avons aussi pu obtenir quelques résultats intéressants montrant que notre application est bien capable de fonctionner efficacement dans les cas pour lesquels elle a été conçue.

Annexe

1 - Documentation de la bibliothèque MIDICSV

RECORD STRUCTURE

Each record in the CSV representation of a MIDI contains at least three fields:

Track
Numeric field identifying the track to which this record belongs. Tracks of MIDI data are numbered starting at 1. Track 0 is reserved for file header, information, and end of file records.
Time
Absolute time, in terms of MIDI clocks, at which this event occurs. Meta-events for which time is not meaningful (for example, song title, copyright information, etc.) have an absolute time of 0.
Type
Name identifying the type of the record. Record types are text consisting of upper and lower case letters and the underscore ("_"), contain no embedded spaces, and are not enclosed in quotes. **csvmidi** ignores upper/lower case in the **Type** field; the specifications "**Note_on_c**", "**Note_On_C**", and "**NOTE_ON_C**" are considered identical.

Records in the CSV file are sorted first by the track number, then by time. Out of order records will be discarded with an error message from **csvmidi**. Following the three required fields are parameter fields which depend upon the **Type**; some **Type**s take no parameters. Each **Type** and its parameter fields is discussed below.

Any line with an initial nonblank character of "**#**" or "**;**" is ignored; either delimiter may be used to introduce comments in a CSV file. Only full-line comments are permitted; you cannot use these delimiters to terminate scanning of a regular data record. Completely blank lines are ignored.

File Structure Records

0, 0, Header*, format, nTracks, division*
The first record of a CSV MIDI file is always the **Header** record. Parameters are *format*: the MIDI file type (0, 1, or 2), *nTracks*: the number of tracks in the file, and *division*: the number of clock pulses per quarter note. The **Track** and **Time** fields are always zero.
0, 0, End_of_file
The last record in a CSV MIDI file is always an **End_of_file** record. Its **Track** and **Time** fields are always zero.
Track, **0, Start_track**
A **Start_track** record marks the start of a new track, with the *Track* field giving the track number. All records between the **Start_track** record and the matching **End_track** will have the same *Track* field.
Track, Time, **End_track**
An **End_track** marks the end of events for the specified *Track*. The *Time* field gives the total duration of the track, which will be identical to the *Time* in the last event before the **End_track**.

File Meta-Events

The following events occur within MIDI tracks and specify various kinds of information and actions. They may appear at any time within the track. Those which provide general information for which time is not relevant usually appear at the start of the track with **Time** zero, but this is not a requirement.

Many of these meta-events include a text string argument. Text strings are output in CSV records enclosed in ASCII double quote (") characters. Quote characters embedded within strings are represented by two consecutive quotes. Non-graphic characters in the ISO 8859-1 Latin-1 set are output as a backslash followed by their three digit octal character code. Two consecutive backslashes denote a literal backslash in the string. Strings in MIDI files can be extremely long, theoretically as many as $2^{28}-1$ characters; programs which process MIDI CSV files should take care to avoid buffer overflows or truncation resulting from lines containing long string items. All meta-events which take a text argument are identified by a suffix of "**_t**".

Track, Time, **Title_t,** *Text*
The *Text* specifies the title of the track or sequence. The first **Title** meta-event in a type 0 MIDI file, or in the first track of a type 1 file gives the name of the work. Subsequent **Title** meta-events in other tracks give the names of those tracks.
Track, Time, **Copyright_t,** *Text*
The *Text* specifies copyright information for the sequence. This is usually placed at time 0 of the first track in the sequence.
Track, Time, **Instrument_name_t,** *Text*
The *Text* names the instrument intended to play the contents of this track, This is usually placed at time 0 of the track. Note that this meta-event is simply a description; MIDI synthesisers are not required (and rarely if ever) respond to it. This meta-event is particularly useful in sequences prepared for synthesisers which do not conform to the General MIDI patch set, as it documents the intended instrument for the track when the sequence is used on a synthesiser with a different patch set.
Track, Time, **Marker_t,** *Text*
The *Text* marks a point in the sequence which occurs at the given *Time*, for example "Third Movement".
Track, Time, **Cue_point_t,** *Text*
The *Text* identifies synchronisation point which occurs at the specified *Time*, for example, "Door slams".
Track, Time, **Lyric_t,** *Text*
The *Text* gives a lyric intended to be sung at the given *Time*. Lyrics are often broken down into separate syllables to time-align them more precisely with the sequence.
Track, Time, **Text_t,** *Text*
This meta-event supplies an arbitrary *Text* string tagged to the *Track* and *Time*. It can be used for textual information which doesn't fall into one of the more specific categories given above.
Track, **0, Sequence_number,** *Number*
This meta-event specifies a sequence *Number* between 0 and 65535, used to arrange multiple tracks in a type 2 MIDI file, or to identify the sequence in which a collection of type 0 or 1 MIDI files should be played. The **Sequence_number** meta-event should occur at **Time** zero, at the start of the track.
Track, Time, **MIDI_port,** *Number*
This meta-event specifies that subsequent events in the **Track** should be sent to MIDI port (bus) *Number*, between 0 and 255. This meta-event usually appears at the start of a track with **Time** zero, but may appear within a track should the need arise to change the port while the track is being played.
Track, Time, **Channel_prefix,** *Number*

This meta-event specifies the MIDI channel that subsequent meta-events and **System_exclusive** events pertain to. The channel *Number* specifies a MIDI channel from 0 to 15. In fact, the *Number* may be as large as 255, but the consequences of specifying a channel number greater than 15 are undefined.

Track, Time, **Time_signature,** *Num, Denom, Click, NotesQ*

The time signature, metronome click rate, and number of 32nd notes per MIDI quarter note (24 MIDI clock times) are given by the numeric arguments. *Num* gives the numerator of the time signature as specified on sheet music. *Denom* specifies the denominator as a negative power of two, for example 2 for a quarter note, 3 for an eighth note, etc. *Click* gives the number of MIDI clocks per metronome click, and *NotesQ* the number of 32nd notes in the nominal MIDI quarter note time of 24 clocks (8 for the default MIDI quarter note definition).

Track, Time, **Key_signature,** *Key, Major/Minor*

The key signature is specified by the numeric *Key* value, which is 0 for the key of C, a positive value for each sharp above C, or a negative value for each flat below C, thus in the inclusive range −7 to 7. The *Major/Minor* field is a quoted string which will be **major** for a major key and **minor** for a minor key.

Track, Time, **Tempo,** *Number*

The tempo is specified as the *Number* of microseconds per quarter note, between 1 and 16777215. A value of 500000 corresponds to 120 quarter notes ("beats") per minute. To convert beats per minute to a **Tempo** *value*, take the quotient from dividing 60,000,000 by the beats per minute.

Track, **0, SMPTE_offset,** *Hour, Minute, Second, Frame, FracFrame*

This meta-event, which must occur with a zero **Time** at the start of a track, specifies the SMPTE time code at which it should start playing. The *FracFrame* field gives the fractional frame time (0 to 99).

Track, Time, **Sequencer_specific,** *Length, Data, …*

The **Sequencer_specific** meta-event is used to store vendor-proprietary data in a MIDI file. The *Length* can be any value between 0 and $2^{28}-1$, specifying the number of *Data* bytes (between 0 and 255) which follow. **Sequencer_specific** records may be very long; programs which process MIDI CSV files should be careful to protect against buffer overflows and truncation of these records.

Track, Time, **Unknown_meta_event,** *Type, Length, Data, …*

If **midicsv** encounters a meta-event with a code not defined by the standard MIDI file specification, it outputs an unknown meta-event record in which *Type* gives the numeric meta-event type code, *Length* the number of data bytes in the meta-event, which can be any value between 0 and $2^{28}-1$, followed by the *Data* bytes. Since meta-events include their own length, it is possible to parse them even if their type and meaning are unknown. **csvmidi** will reconstruct unknown meta-events with the same type code and content as in the original MIDI file.

Channel Events

These events are the "meat and potatoes" of MIDI files: the actual notes and modifiers that command the instruments to play the music. Each has a MIDI channel number as its first argument, followed by event-specific parameters. To permit programs which process CSV files to easily distinguish them from meta-events, names of channel events all have a suffix of "**_c**".

Track, Time, **Note_on_c,** *Channel, Note, Velocity*

Send a command to play the specified *Note* (Middle C is defined as *Note* number 60; all other notes are relative in the MIDI specification, but most instruments conform to the well-tempered scale) on the given *Channel* with *Velocity* (0 to 127). A **Note_on_c** event with *Velocity* zero is equivalent to a **Note_off_c**.

https://www.inspiredacoustics.com/en/MIDI_note_numbers_and_center_frequencies

-> Midi note number to note name

Track, Time, **Note_off_c,** *Channel, Note, Velocity*
Stop playing the specified *Note* on the given *Channel*. The *Velocity* should be zero, but you never know what you'll find in a MIDI file.
Track, Time, **Pitch_bend_c,** *Channel, Value*
Send a pitch bend command of the specified *Value* to the given *Channel*. The pitch bend *Value* is a 14 bit unsigned integer and hence must be in the inclusive range from 0 to 16383. The value 8192 indicates no pitch bend; 0 the lowest pitch bend, and 16383 the highest. The actual change in pitch these values produce is unspecified.
Track, Time, **Control_c,** *Channel, Control_num, Value*
Set the controller *Control_num* on the given *Channel* to the specified *Value*. *Control_num* and *Value* must be in the inclusive range 0 to 127. The assignment of *Control_num* values to effects differs from instrument to instrument. The General MIDI specification defines the meaning of controllers 1 (modulation), 7 (volume), 10 (pan), 11 (expression), and 64 (sustain), but not all instruments and patches respond to these controllers. Instruments which support those capabilities usually assign reverberation to controller 91 and chorus to controller 93.
Track, Time, **Program_c,** *Channel, Program_num*
Switch the specified *Channel* to program (patch) *Program_num*, which must be between 0 and 127. The program or patch selects which instrument and associated settings that channel will emulate. The General MIDI specification provides a standard set of instruments, but synthesisers are free to implement other sets of instruments and many permit the user to create custom patches and assign them to program numbers.

Apparently, due to instrument manufacturers' skepticism about musicians' ability to cope with the number zero, many instruments number patches from 1 to 128 rather than the 0 to 127 used within MIDI files. When interpreting *Program_num* values, note that they may be one less than the patch numbers given in an instrument's documentation.
Track, Time, **Channel_aftertouch_c,** *Channel, Value*
When a key is held down after being pressed, some synthesisers send the pressure, repeatedly if it varies, until the key is released, but do not distinguish pressure on different keys played simultaneously and held down. This is referred to as “monophonic” or “channel” aftertouch (the latter indicating it applies to the *Channel* as a whole, not individual note numbers on that channel). The pressure *Value* (0 to 127) is typically taken to apply to the last note played, but instruments are not guaranteed to behave in this manner.
Track, Time, **Poly_aftertouch_c,** *Channel, Note, Value*
Polyphonic synthesisers (those capable of playing multiple notes simultaneously on a single channel), often provide independent aftertouch for each note. This event specifies the aftertouch pressure *Value* (0 to 127) for the specified *Note* on the given *Channel*.

System Exclusive Events

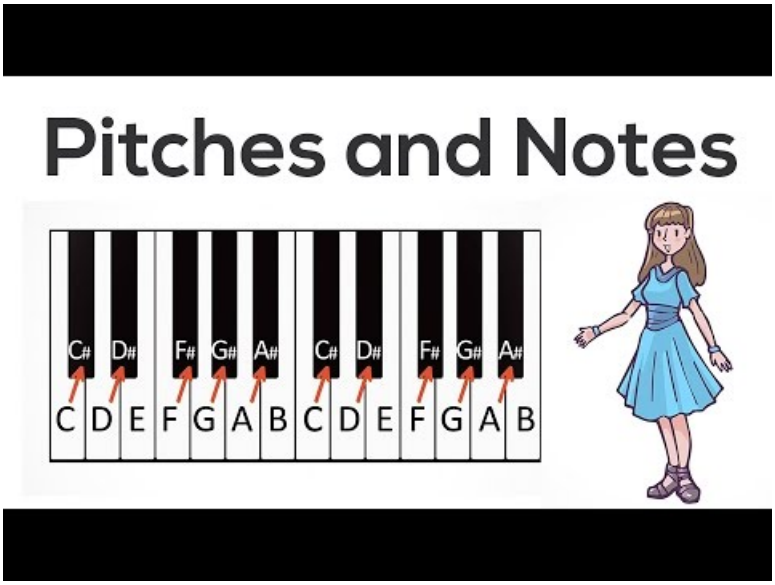
System Exclusive events permit storing vendor-specific information to be transmitted to that vendor's products.

Track, Time, **System_exclusive,** *Length, Data, …*
The *Length* bytes of *Data* (0 to 255) are sent at the specified *Time* to the MIDI channel defined by the most recent **Channel_prefix** event on the *Track*, as a System Exclusive message. Note that *Length* can be any value between 0 and $2^{28}-1$. Programs which process MIDI CSV files should be careful to protect against buffer overflows and truncation of these records.
Track, Time, **System_exclusive_packet,** *Length, Data, …*

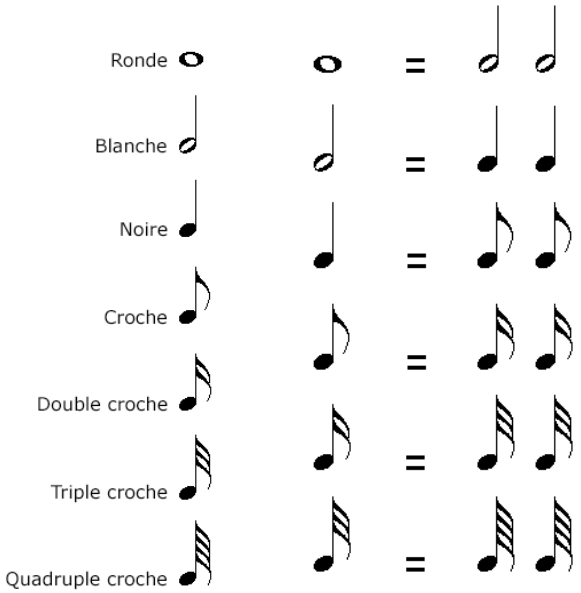
The *Length* bytes of *Data* (0 to 255) are sent at the specified *Time* to the MIDI channel defined by the most recent **Channel_prefix** event on the *Track*. The *Data* bytes are simply blasted out to the MIDI bus without any prefix. This message is used by MIDI devices which break up long system exclusive message into small packets, spaced out in time to avoid overdriving their modest microcontrollers. Note that *Length* can be any value between 0 and $2^{28}-1$. Programs which process MIDI CSV files should be careful to protect against buffer overflows and truncation of these records.

2 - Notions de théorie musicale

Vous trouverez ci dessous quelques illustration et explications des connaissances musicales de base nécessaire à la compréhension du projet.

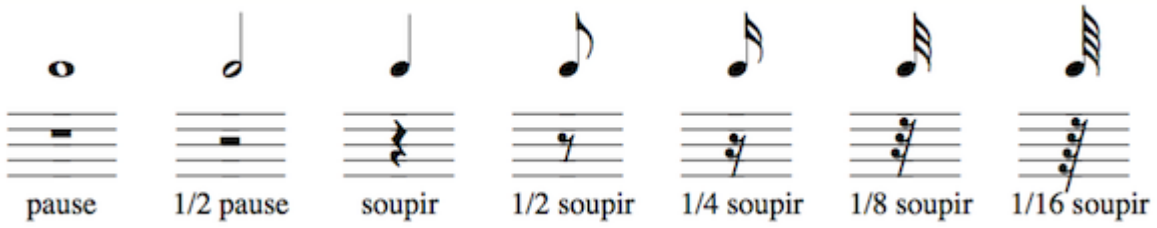


C = Do, D = Ré, …



Unité de temps : [1]	division naturelle : [1/2 + 1/2]	division en triolet : [1/3 + 1/3 + 1/3]
𝅝	𝅗𝅥 𝅗𝅥	3 𝅗𝅥 𝅗𝅥 𝅗𝅥
𝅗𝅥	♩ ♩	3 ♩ ♩ ♩
♩	♪ ♪	3 ♪ ♪ ♪
♪	𝅘𝅥𝅯 𝅘𝅥𝅯	3 𝅘𝅥𝅯 𝅘𝅥𝅯 𝅘𝅥𝅯
etc.	*etc.*	*etc.*

De manière similaire au triolets on définit les quintolets :
division en quintolet [⅕ + ⅕ + ⅕ + ⅕ + ⅕]
Les notes pointées sont définies de la manière suivante :
blanche pointée = 3/2 * blanche, représentée comme une blanche suivie d'un point
Les silences (absences de notes) sont représentés de la manière suivante et suivent une logique similaire.



A tout cela on peut ajouter rapidement qu'en théorie musicale il existe des modes, qui ne nous servent pas dans ce projet mais auxquels nous devons faire attention. En effet mélanger différents modes dans la base de données à de forte chance de créer des résultats peu agréable à écouter.

Mode	***Majeur ou mineur?***	***Intervalles***	***Utilisation***
Ionien	Majeur	**1 ton - 1 ton - 1/2 ton - 1 ton - 1 ton – 1 ton - 1/2 ton**	Joie
Dorien	Mineur	**1 ton - 1/2 ton - 1 ton - 1 ton - 1 ton - 1/2 ton - 1 ton**	Jazz et blues
Phrygien	Mineur	**1/2 ton - 1 ton - 1 ton - 1 ton - 1/2 ton - 1 ton - 1 ton**	Espagnole et flamenco
Lydien	Majeur	**1 ton - 1 ton - 1 ton - 1/2 ton - 1 ton - 1 ton - 1/2 ton**	Prondeur et msytère
Mixolydien	Majeur	**1 ton - 1 ton - 1/2 ton- 1 ton - 1 ton - 1/2 ton - 1 ton**	Blues
Éolien	Mineur	**1 ton - 1/2 ton - 1 ton - 1 ton - 1/2 ton- 1 ton - 1 ton**	Doux et mielleux
Locrien	Mineur	**1/2 ton- 1 ton - 1 ton- 1/2 ton - 1 ton - 1 ton - 1 ton**	Tension et stress