Document résultant de conception

1. Introduction

- Objectif du document : Décrire l'architecture et les choix techniques du générateur de musique interactif développé pour MusicaLau.
- Portée : L'application couvre l'affichage et l'interaction avec des instruments virtuels, la lecture/enregistrement de fichiers de partitions, et des fonctionnalités d'innovation musicale.

2. Architecture générale

- Modèle architectural : MVC (Modèle – Vue – Contrôleur)

[Utilisateur]

|

v

[Vue] <--> [Contrôleur] <--> [Modèle]

- Organisation du projet : Maven
- Librairie utilisée : Java MIDI (javax.sound.midi), javax.swing (JFrame, JButton, JPanel, etc.), java.awt (layouts, couleurs, graphiques), javax.sound.sampled.* avec génération de formes d'onde (ex. sinusoïdales) pour l'instrument « video game »
- Diagrammes d'architecture (ex : diagramme de composants, diagramme de déploiement, etc.)

3. Description des Composants

3.1 Modèle

- Gère les données musicales (notes, manière de générer une notes, mapping)
- Gère les données d'enregistrement (manière d'enregistrer)
- Gère les données de lecture (manière de lire)
- Interface avec les sons MIDI

3.2 Vue

- Interface graphique
- Permet :
    - Affichage dynamique d'un instrument à la fois
    - Interaction utilisateur (clics, sélections, touches, nombre d'octaves)

3.3 Contrôleur

- Fait le lien entre les actions utilisateur et le modèle
- Gère :
    - Sélection d'instrument
    - Jeu d'un instrument
    - Lecture de partitions
    - Enregistrement des notes
    - Navigation (Retour, Quitter)

4. Modèle de données et diagramme de classe

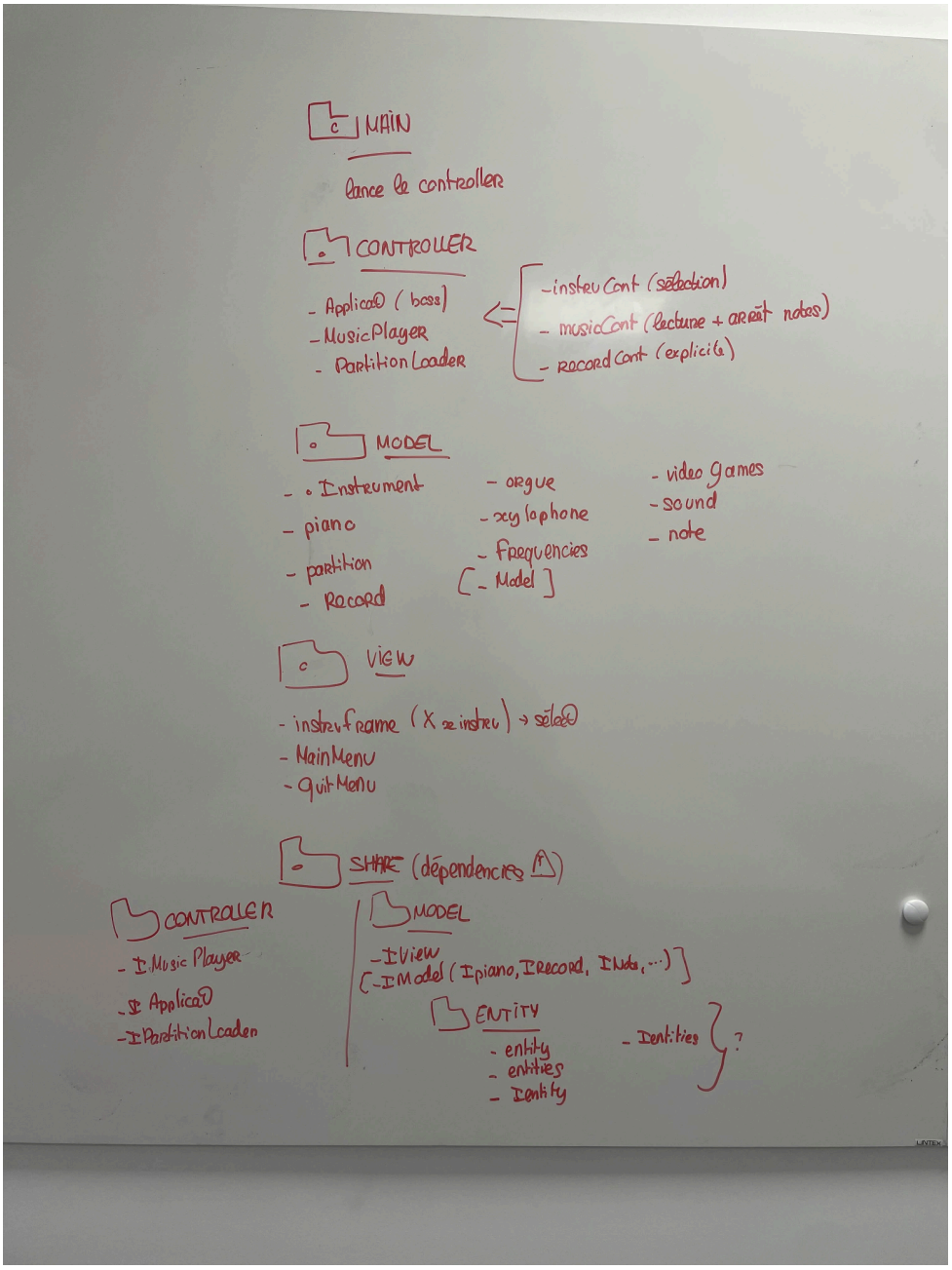

Diagramme architectural de départ

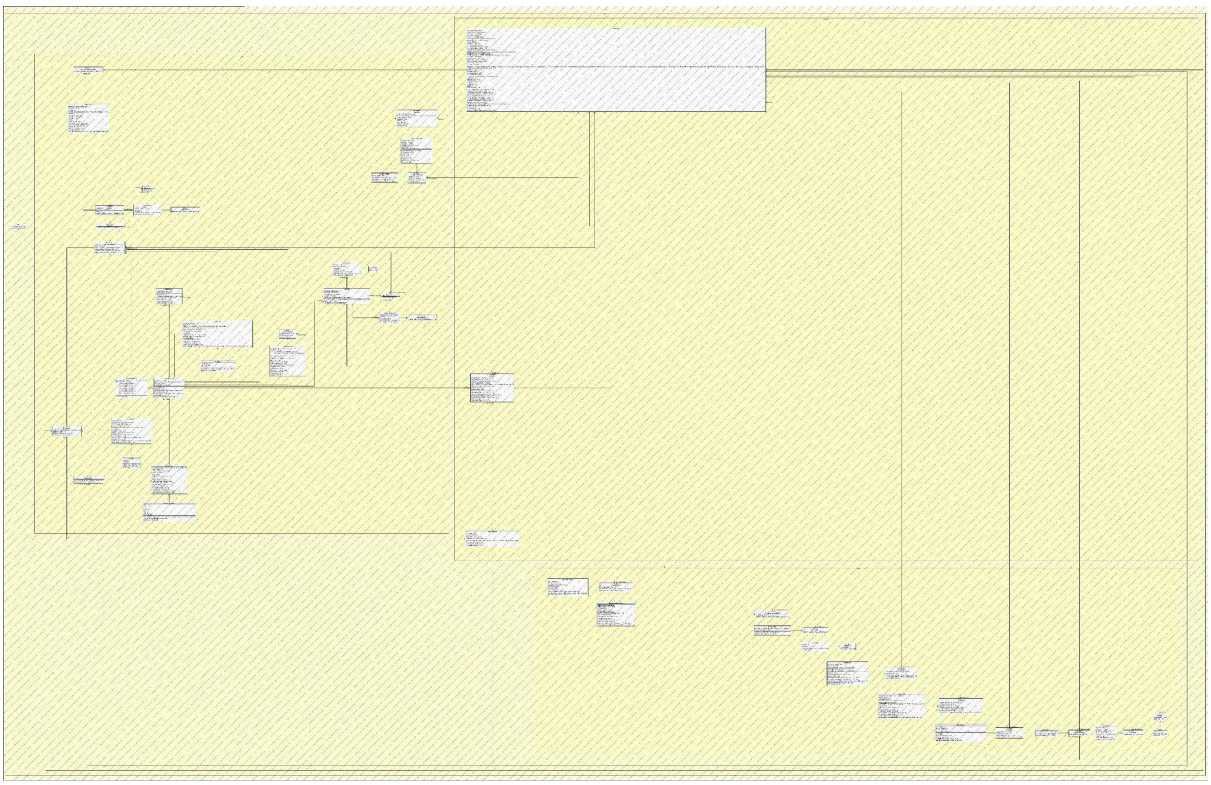

Diagramme de classes final

5. Choix techniques

5.1 Architecture : MVC (Modèle – Vue – Contrôleur)

L'architecture MVC a été choisie pour plusieurs raisons :

- Séparation des responsabilités : facilite la maintenance, la lisibilité du code et l'ajout de fonctionnalités.
- Réutilisabilité : permet de réutiliser le modèle et les contrôleurs indépendamment de l'interface.
- Testabilité : chaque composant peut être testé séparément.

5.2 Outils de construction : Maven

Le projet est structuré comme une application Maven Java :

- Organisation modulaire avec gestion des dépendances.
- Dossier src/main/java structuré par packages : model, view, controller, share.

5.3 Librairie MIDI pour instruments standards (piano, xylophone)

Pour le rendu sonore des instruments classiques (piano, xylophone), nous utilisons l'API Java MIDI :

- Package utilisé : javax.sound.midi.*
- Avantages :
    - Facilité d'utilisation pour jouer des notes.
    - Bonne compatibilité avec les touches clavier et événements GUI (*Graphical User Interface)*.
    - Support intégré pour plusieurs canaux, octaves, instruments.

Spécificité du piano :

- Permet à l'utilisateur de sélectionner le nombre d'octaves (minimum 2, jusqu'à 5 dans notre implémentation).
- Les touches sont générées dynamiquement selon ce paramètre.

5.4 Synthèse sonore personnalisée pour l'instrument "video games"

Pour simuler les sons typiques des jeux, nous avons opté pour la génération manuelle d'ondes audio à l'aide de :

- Package utilisé : javax.sound.sampled.*
- Approche : au lieu d'utiliser des sons préenregistrés ou MIDI, nous générons le son en temps réel via des fonctions mathématiques.
- Types d'ondes supportées :
    - Sinusoïdale (pure)
    - Carrée (binaire, type Game Boy)
    - Triangulaire (plus douce)
    - Bruit (pour effets percussifs ou aléatoires)

· Paramètres contrôlables :

    - Fréquence (Hz)
    - Durée (ms)
    - Volume (amplitude)
    - Type d'onde

Avantages :

- Contrôle total sur le rendu sonore
- Possibilités d'innovation : génération aléatoire, synthés personnalisés, effets dynamiques

Contraintes :

- Gestion manuelle des buffers audio
- Plus bas niveau → nécessite des précautions sur le threading et la synchronisation

5.5 Interface graphique : Swing

- Bibliothèque utilisée : javax.swing.*, avec java.awt.* pour les éléments graphiques.
- Choisie pour sa compatibilité avec Java standard, et sa flexibilité graphique.
- Composants utilisés :
    - JFrame : fenêtre principale
    - JPanel : zones personnalisées (dessin d'instruments)
    - JButton : contrôles (jouer, ouvrir, enregistrer, retour, quitter)
    - JFileChooser : sélection de fichiers pour ouvrir/enregistrer

6. Gestion des erreurs fonctionnalités Enregistrement, Lecture de partition

- Mauvais format de partition : gestion via try/catch
- Fichier d'enregistrement déjà existant : gestion via try/catch + message d'erreur et possiblité de recommencer

7. Scénarios d'utilisation (Use cases techniques)

Exemples :

- "L'utilisateur choisit le xylophone, ouvre bella_ciao.txt, écoute le morceau"
- "L'utilisateur joue une mélodie au piano avec 3 octaves, en utiliser le clavier de son ordinateur ou pas, l'enregistre sous le_morceau_qui_me_fera_devenir_celebre.txt "

8. Innovations apportées

Choix d'instrument à touches ou frappés, car difficulté d'implémentation et sons des instruments à vent non ressemblant

- Ajout d'un tambour comme nouvel instrument
- Ajout d'un orgue
- Ajout d'un lecteur d'un fichier mp3 (fonctionnalité chat)
- Animation des touches de tous les instruments, des bouton READ, MEOW (chat) et du bouton REC lors d'un enregistrement
- Enregistrement de partition sous format txt
- Ajout de partitions (au clair de la lune et fur elise)