Séquenceur décembre

Le planning prévisionnel établi dans le cahier des charges fixait pour ce mois de décembre deux objectifs majeurs : l'implémentation de plusieurs prises et l'introduction des patterns Il prévoyait également une première gestion de la lecture en boucle.

Grâce à l'avance prise en novembre sur la gestion multi-pistes (kick, Snare, Hi-Hat étant déjà intégrés et fonctionnels), j'ai pu consacrer l'intégralité de ce mois à la logique pure du séquenceur : la transformation d'une grille statique en un instrument musical capable de jouer un rythme en temps reel.

Conception de la grille

L'objectif « introduction des patterns » a été traité simultanément sur l'interface et le cœur logique. Au niveau de l'interface (*Frontend*), j'ai fait évoluer le fichier fenetre_principale.py pour remplacer le bouton de test unique par une grille de 16 cases à cocher (*steps*) pour chaque piste. Cette modification offre une représentation visuelle conforme aux standards des boîtes à rythmes classiques (type TR-808). Sur le plan logique (*Backend*), la classe Piste du fichier coeur.py a été modifiée pour accueillir une mémoire interne : elle ne stocke plus seulement un chemin de fichier audio, mais contient désormais une liste de 16 booléens (self.pattern = [False] * 16) représentant l'état actif ou inactif de chaque temps. Enfin, la synchronisation est assurée par une connexion signal-slot via la méthode update_step, permettant de mettre à jour instantanément la mémoire du programme dès que l'utilisateur coche une case.

Implémentation de la lecture en boucle

Le défi technique principal de ce mois était de passer d'une lecture manuelle à une lecture automatique. Pour répondre à l'exigence de « lecture en boucle », j'ai implémenté un système de minuterie basé sur QTimer. J'ai intégré cet objet dans la fenêtre principale en le réglant sur un intervalle de 125 ms, un choix technique qui correspond mathématiquement à une double-croche pour un tempo standard de 120 BPM ($60s / 120 / 4$). Concernant l'algorithme de lecture, à chaque déclenchement du Timer, la méthode boucle_de_lecture pilote le SequenceurCore : celui-ci exécute la fonction jouer_step_actuel pour vérifier l'état des patterns du pas en cours et déclencher les sons via le moteur audio, puis incrémente le compteur via pas_suivant pour boucler de 0 à 15 indéfiniment.

Mise à jour de architecture

L'implémentation réelle a conduit à une simplification du diagramme de classes initial. La classe Pattern, initialement prévue comme une entité distincte, a été fusionnée directement dans la classe Piste sous forme de liste pour optimiser les performances et simplifier l'accès aux données lors de la lecture temps réel. De même, la responsabilité du Timer a été attribuée à la fenêtre principale pour tirer parti de la boucle d'événements native de PySide6.

Conclusion

Les objectifs de décembre sont pleinement validés. Le logiciel permet désormais de composer des rythmes sur 16 temps et de les écouter en boucle, transformant le prototype en un réel séquenceur (tout de même assez rudimentaire)

Les tests de lecture ont cependant mis en évidence une limitation technique attendue : la gestion de la polyphonie. Actuellement, si deux sons sont joués simultanément (ex: Kick + Snare sur le même temps), le moteur audio basique peut couper le son précédent. Conformément au planning, le mois de janvier sera donc dédié à la « synchronisation entre les pistes ». Cela impliquera le développement d'un mixage audio mathématique (addition de signaux) et l'ajout d'un retour visuel dynamique (curseur de lecture) sur la grille