МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Чорноморський національний університет
імені Петра Могили
Факультет комп'ютерних наук
Кафедра інтелектуальних інформаційних систем

ДОПУЩЕНО ДО ЗАХИСТУ
Завідувач кафедри інтелектуальних
інформаційних систем, д-р техн. наук, проф.
_____Ю. П. Кондратенко
«____»_____2024 р.

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

ІНТЕЛЕКТУАЛЬНА СИСТЕМА ГЕНЕРУВАННЯ МУЗИЧНИХ МЕЛОДІЙ НА ОСНОВІ VST SDK

Спеціальність 122 «Комп'ютерні науки»

122 – КРБ – 402.22010209

Виконав студент 4-го курсу, групи 402
_____ І. Ю. Дирда
«17» червня 2024 р.

Керівник: канд. техн. наук, доцент
_____Є. В. Сіденко
«17» червня 2024 р.

Миколаїв – 2024

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Чорноморський національний університет ім. Петра Могили
Факультет комп'ютерних наук
Кафедра інтелектуальних інформаційних систем

Рівень вищої освіти **бакалавр**
Спеціальність **122 «Комп'ютерні науки»**
(шифр і назва)
Галузь знань **12 «Інформаційні технології»**
(шифр і назва)

ЗАТВЕРДЖУЮ

Завідувач кафедри інтелектуальних інформаційних систем, д-р техн. наук, проф.
_____ Ю. П. Кондратенко
«____» _____2024 р.

З А В Д А Н Н Я
на виконання кваліфікаційної роботи

Видано студенту групи 402 факультету комп'ютерних наук Дирді Іллі Юрійовичу.

1. Тема кваліфікаційної роботи «Інтелектуальна система генерування музичних мелодій на основі VST SDK».

Керівник роботи Сіденко Євген Вікторович, канд. техн. наук, доцент.

Затв. наказом Ректора ЧНУ ім. Петра Могили від «28» грудня 2023 р. № 271

2. Строк представлення кваліфікаційної роботи студентом «17» червня 2024 р.

3. Вхідні (початкові) дані до роботи: база даних музичних композицій у форматі MIDI.

Очікуваний результат: інтелектуальна система генерування музичних мелодій на основі VST SDK.

4. Перелік питань, що підлягають розробці (зміст пояснювальної записки):

- аналіз сучасного стану задачі створення системи генерування музичних мелодій на основі VST SDK;
- порівняння видів аудіо обробляючих систем;
- розгляд і порівняння альтернативних рішень програмного забезпечення з генерування музичних композицій;

– розробка програмного забезпечення для генерування музичних мелодій.

5. Перелік графічного матеріалу: 35 рисунків, 2 таблиці та презентація.

6. Завдання до спеціальної частини: «Захист фізичного та психічного здоров'я під час створення музичних композицій»

7. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис
Спеціальна частина з охорони праці	Алексєєва А. О., доцент кафедри екології	

Керівник роботи __канд. техн. наук, доц. Сіденко Є. В._____
(наук. ступінь, вчене звання, прізвище та ініціали)

_____
(підпис)

Завдання прийнято до виконання ___Дирда І. Ю._____
(прізвище та ініціали)

_____
(підпис)

Дата видачі завдання «_6_» _____лютого____ 2024 р.

КАЛЕНДАРНИЙ ПЛАН
виконання кваліфікаційної роботи

Тема: Інтелектуальна система генерування музичних мелодій на основі VST SDK.

№	Найменування роботи	Початок	Закінчення	Примітки
1	Подання заяви на затвердження теми та керівників КРБ	10.11.2023	15.11.2023	Виконано
2	Отримання завдання на виконання КРБ	10.01.2024	15.01.2024	Виконано
3	Складання календарного плану роботи на весь період виконання КРБ	16.01.2024	30.01.2024	Виконано
4	Отримання завдання на переддипломну практику	15.04.2024	29.04.2024	Виконано
5	Проходження переддипломної практики, збір та аналіз матеріалів до КРБ	29.04.2024	11.05.2024	Виконано
6	Розробка звіту з переддипломної практики	12.05.2024	15.05.2024	Виконано
7	Виконання КРБ: аналіз сучасного стану задачі генерування музичних мелодій, огляд існуючих технологій, розробка ПЗ	13.05.2024	22.06.2024	Виконано
8	Перший попередній захист КРБ на засіданні комісії кафедри	27.05.2024	27.05.2024	Виконано
9	Доробка та остаточне оформлення КРБ	28.05.2024	09.06.2024	Виконано
10	Другий попередній захист КРБ на засіданні комісії кафедри	10.06.2024	10.06.2024	Виконано
11	Подання КРБ рецензенту	13.06.2024	13.06.2024	Виконано
11	Подання КРБ, її електронної копії та інших документів (відгуку, рецензії) до захисту	17.06.2024	21.06.2024	
12	Захист БКР перед екзаменаційною комісією (ЕК)	24.06.2024	28.06.2024	

Розробив студент __Дирда І. Ю._____ _____
(прізвище, ім'я, по батькові студента) (*підпис*)

Керівник роботи __канд. техн. наук, доц. Сіденко Є. В. _____
(*посада, прізвище, ім'я, по батькові)* *(підпис)*

«_29_» _____01_____ 2024 р.

АНОТАЦІЯ

кваліфікаційної роботи студента групи 402 ЧНУ ім. Петра Могили

Дирди Іллі Юрійовича

Тема: «Інтелектуальна система генерування музичних мелодій на основі VST SDK»

Вхід в музичну індустрію стає дедалі відкритішим з кожним роком. Інтелектуальні системи на кшталт плагіну для генерування музичних мелодій полегшать та покращать можливість створення власних музичних композицій.

Об'єкт роботи – процеси генерування музичних мелодій.

Предмет роботи – програмні засоби та бібліотеки на основі VST SDK для генерування музичних мелодій.

Метою кваліфікаційної роботи є спрощення процесу створення музичної композиції для різних типів користувачів цифрових звукових робочих станцій. Одним із завдань було дослідження сучасного стану задачі генерування музичних мелодій.

Пояснювальна записка складається зі вступу, чотирьох розділів, висновків та додатків.

У першому розділі було розглянуто середу розробки плагіну, проаналізовано обрану для тестування DAW, а також досліджено аналогічні системи.

У другому розділі було описано процес налаштування середи розробки та досліджено структуру плагіну.

У третьому розділі розглянуто зовнішній вигляд плагіну, його програмну реалізацію, а також наведено концепцію роботи плагіну.

У четвертому розділі описано деталі програмної реалізації системи та процесу генерування музичних мелодій.

В результаті розроблено інтелектуальну систему генерування музичних мелодій на основі VST SDK.

Кваліфікаційна робота містить 59 сторінок, 35 рисунків, 2 таблиці, 13 використаних джерел та 2 додатки.

Ключові слова: генерування, плагін, мелодія, VST.

ABSTRACT

To the bachelor’s qualification work by student of group 402 of

Petro Mohyla Black Sea National University

Dyrda Illia

"Intelligent system for generating musical melodies based on VST SDK"

Entry into the music industry is becoming more and more open every year. Intelligent systems like the plug-in for generating musical melodies will facilitate and improve the ability to create your own musical compositions.

The object of the work is the processes of the intelligent system of generating musical melodies based on the VST SDK.

The subject of the work is an intelligent system based on the VST SDK, which will allow the generation of musical melodies.

The purpose of the qualification work is to facilitate the creation of a musical composition for pioneers and already professional users of digital sound workstations. One of the tasks was to study the current state of the task of generating musical melodies.

The explanatory note consists of an introduction, four chapters, conclusions and appendices.

In the first chapter, the plug-in development environment was considered, the DAW selected for testing was analyzed, and similar systems were investigated.

The second chapter described the process of setting up the development environment and explored the structure of the plugin.

The third chapter examines the appearance of the plug-in, its software implementation, and also provides the concept of the plug-in's work.

The fourth chapter describes the details of the software implementation of the system and the process of generating musical melodies.

As a result, an intelligent system for generating musical melodies based on the VST SDK was developed.

The qualification work contains 59 pages, 35 figures, 2 tables, 13 used sources and 2 appendices.

Keywords: generation, plugin, melody, VST.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

ПЗ – програмне забезпечення

DAW – Digital Audio Workstations

MIDI – Musical Instrument Digital Interface

SDK – Software Development Kit

VST – Virtual Studiio Technology

ВСТУП

Актуальність теми. Музична індустрія щороку приваблює дедалі більше ентузіастів, які готові робити свій внесок і додавати нововведення як в творчому, так і в технічному аспекті. Використовуючи різноманітні сучасні цифрові звукові робочі станції (Digital Audio Workstations, DAW), формувати музичні композиції стало набагато легше. Ці платформи пропонують широкий спектр інструментів та можливостей для створення, редагування та мікшування музики, що робить процес виробництва доступнішим навіть для новачків.

До цих застосунків великі компанії та досвідчені користувачі винаходять додаткове програмне забезпечення - плагіни. Плагіни дають можливість покращувати звучання та простіше писати інструментальні мелодії. Вони можуть симулювати звучання різноманітних музичних інструментів, додаючи реалізму та глибини композиціям. Крім того, плагіни дозволяють використовувати ефекти, що раніше були доступні лише у професійних студіях, такі як реверберація, еквалізація та компресія.

Сучасні плагіни, як правило, підтримують різні формати, такі як VST, AU та AAX, що забезпечує їхню сумісність з більшістю популярних DAW. Це дозволяє музикантам і продюсерам мати доступ до величезного арсеналу інструментів та ефектів, незалежно від того, яку програму вони використовують. Більше того, розвиток технологій штучного інтелекту та машинного навчання відкриває нові горизонти для автоматизації процесів створення музики, таких як генерація мелодій, гармоній та ритмів.

Завдяки цим інноваціям, процес створення музики став не лише більш доступним, але й більш захоплюючим. Музиканти можуть експериментувати зі звуками, шукати нові стилі та підходи, що сприяє розвитку музичної культури в цілому. Таким чином, сучасні технології в музичній індустрії не тільки спрощують процес створення музики, але й надихають нове покоління митців на пошук унікальних та креативних рішень.

Об’єкт: процеси генерування музичних мелодій.

Предмет: програмні засоби та бібліотеки на основі VST SDK для генерування музичних мелодій.

Мета: спрощення процесу створення музичної композиції для різних типів користувачів цифрових звукових робочих станцій.

Для досягнення поставленої мети необхідно вирішити такі **завдання:**

- проаналізувати сучасний стан задачі генерування музичних мелодій;
- проаналізувати існуючі аналоги систем;
- обрати необхідний тип плагіну;
- розробити систему генерування музичних мелодій;
- розробити користувацький графічний інтерфейс;
- провести тестування застосунку.

Практичне значення: генерування музичних мелодій на основі VST SDK полягає у створенні програмного забезпечення, яке автоматично компонує музику в режимі реального часу, адаптуючись до настрою, стилю та вимог користувача. Використовуючи можливості VST SDK, така система інтегрується з популярними цифровими аудіо робочими станціями (DAW), дозволяючи музикантам і продюсерам легко додавати нові мелодійні лінії у свої треки, експериментувати з різними жанрами та настроями, а також значно скорочувати час на створення музичних композицій.

1 ОГЛЯД DAW ТА СЕРЕДИ РОЗРОБКИ СИСТЕМИ

1.1 Вибір цифрової звукової робочої станції

Користувачі DAW - цифрових звукових робочих станцій, дуже прискіпливі та обирають для себе продукт, який підійде їм за всіми поставленими критеріями. Більшість з сучасних бітмейкерів – люди в музичній індустрії, які займаються безпосередньо створенням музичного супровіду, обирають в якості програмного забезпечення застосунок FL Studio частіше, ніж інші його альтернативи.

Рисунок 1.1 – Логотип програми FL Studio

FL Studio [8] – є редактором-секвенсором для написання музики. DAW була створена в 1997 році програмістом Дідьє Дембреном, програма написана мовою програмування Delphi. В даний момент програмою займається компанія Image-Line Software.

Рисунок 1.2 – Логотип компанії Image-Line Software

FL Studio пропонує більше 100 різноманітних інструментів та ефектів, величезну звукову бібліотеку та зручні для новачків інструменти, щоб створювати музчні композиції. Унікальною характеристикою FL Studio є те, що використовуючи цей програмний продукт, є можливим написання музики різного рівня складності, при цьому, мати музичну освіту та спеціальні навички не є необхідним. Завдяки цьому, FL Studio можна вважати однією з найпопулярніших звукових робочих станцій, яким користуються велика кількість професійних музикантів та аматорів.

Музика створюється в процесі запису та зведення (звукозапису) аудіо-, або MIDI-матеріалу.

Musical Instrument Digital Interface, тобто MIDI – є стандартом передачі інформації між електронними музичними інструментами, який був розроблений 1983 року. MIDI дає можливість електронним музичним інструментам взаємодіяти між собою чи комп'ютером та іншим сумісним обладнанням, здійснювати з одного інструменту управління іншими. Визначення значення MIDI для кожної ноти зображення на рисунку 1.3.

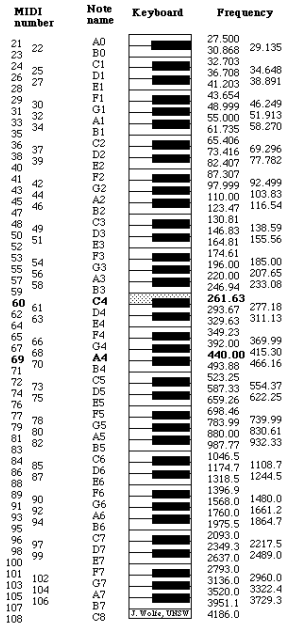

Рисунок 1.3 – Параметри клавіатури, яка використовується для створення музики в MIDI

Готова композиція в FL Studio може бути записана у файл з розширенням WAV, MP3 або OGG. Створення музики відбувається в панелях Piano Roll (рис. 1.4), Step Sequencer, пізніше здійснюється збірка у вікні Playlist та кінцева обробка у вікні Mixer. Є великий набір вже готових інструментів і безліч ефектів, які можуть бути задіяні в режимі реального часу.

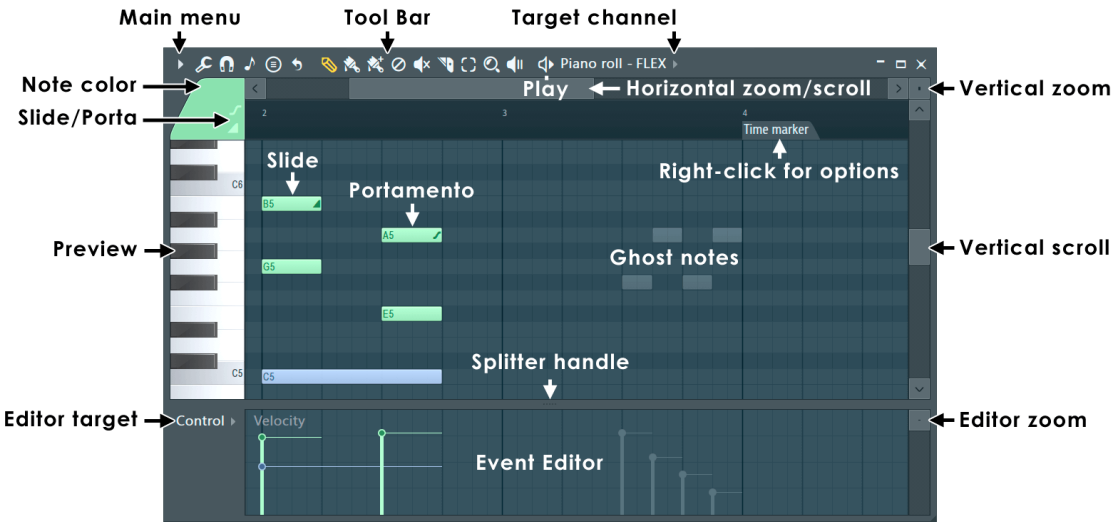

Рисунок 1.4 – Складник Piano Roll в FL Studio

Редактор Piano Roll призначений для створення складних багатоголосних партій. Він фактично виконує функцію нотного стану, але в більш зручній формі, дозволяючи писати партитуру, спираючись лише на слух. Інтерфейс редактора нагадує віртуальну клавіатуру піаніно, яка має 128 клавіш, кожна з яких відповідає певній ноті. Використовуючи цей тип візуального редактора, можна уникнути необхідності у додаткових позначеннях знаків альтерації та пауз, що значно полегшує процес створення музичних творів.

Плейлист цього редактору розмежований розділами. Ці розділи позначають часовий період звучання ноти. Тобто клавіатура — це вісь Y, а часовий розмежовувач — це вісь X.

Головна складова проекту композиції — генератор (канал). Приклад генератору зображено на рисунку 1.5.

Рисунок 1.5 – Приклад генератору Groove Machine в FL Studio

Генератор синтезує або відтворює звук. Генераторів в проекті композиції може бути необмежена кількість. Кожен з них володіє своїми налаштуваннями, унікальним звуком, що імітує будь-який інструмент. Для плагінів програмуються нотні партитури, записувані в Piano Roll. Партитури у FL Studio мають кінцеву довжину. Шматочки партитур (патерни) складаються в послідовності (розташовуються в потрібному порядку в списку відтворення) у вікні Playlist (рис. 1.6).

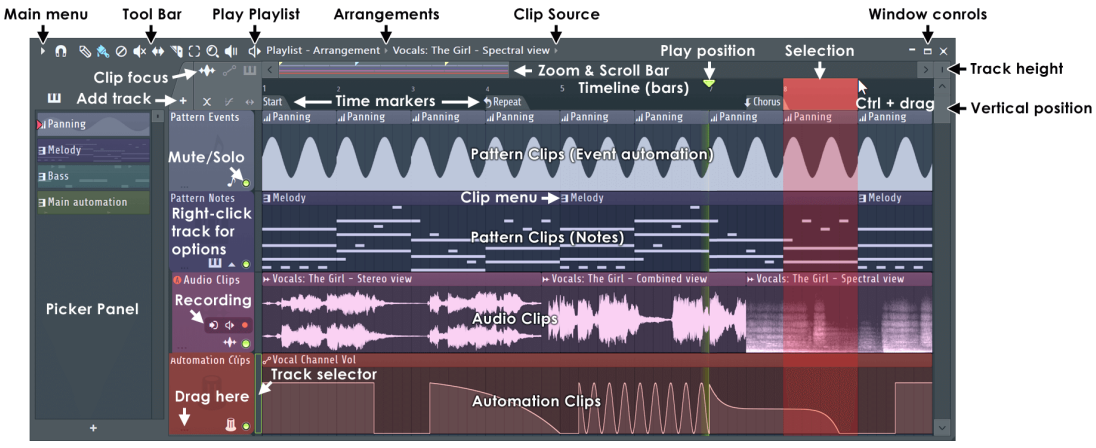

Рисунок 1.6 – Складник Playlist в FL Studio

Звук кожного генератора може бути оброблений за допомогою безлічі ефектів.

Як генератор можна підключити будь-який VST або DXi плагін. Окрім того, FL Studio можна використовувати в інших музичних редакторах як VST або DXi плагін.

1.2 Визначення типу плагіна

Технологія віртуальної студії (Virtual Studio Technology) [1] — це програмний інтерфейс плагіна аудіо, який полегшує інтеграцію програмних синтезаторів і ефектів у цифрові аудіоробочі станції (DAW), розроблений компанією Steinberg в 1996 році.

Рисунок 1.7 – Логотип Virtual Studio Technology

Рисунок 1.8 – Логотип компанії Steinberg

Плагін VST – це компонент обробки аудіо, який використовується в головній програмі.

Ця головна програма надає аудіо та/або потоки подій, які обробляються кодом плагіна. Загалом, плагін VST може отримати потік аудіоданих, застосувати процес до аудіо та повернути результат головній програмі.

Плагін працює у звичайному режимі за допомогою процесора комп'ютера. Аудіопотік ділиться на серію блоків, які послідовно подає хост, контролюючи розмір цих блоків відповідно до поточного середовища.

VST додаток зберігає стан усіх своїх параметрів, пов'язаних із запущеним процесом, тоді як хост не зберігає жодної інформації про те, що плагін зробив з останнім обробленим блоком даних.

Для головної програми плагін VST є чорним ящиком із довільною кількістю входів і виходів (події MIDI або аудіо) та відповідних параметрів. Хосту не потрібно знати внутрішні процеси плагіна, щоб використовувати його.

Процес плагіна може використовувати будь-які внутрішні параметри, а також, залежно від можливостей хоста, дозволяти автоматизацію змін параметрів користувача.Існує декілька різновидів VST-плагінів, такі як:

- VST інструменти - VST-плаґіни, що генерують звук;
- VST ефекти - VST-плаґіни, що здійснюють обробку звукового сигналу, наприклад ефекти реверберації або фейзеру;
- VST MIDI ефекти - VST-плаґіни, що обробляють MIDI-повідомлення перед тим, як MIDI-дані потрапляють на інші VST інструменти або апаратні пристрої. Вони дозволяють. наприклад транспонувати музику або генерувати арпеджіо.

1.3 Вибір середовища розробки плагіна

Обираючи середовище розробки програмного коду, я зупинився на Microsoft Visual Studio 2019. З продуктом я познайомився в університеті, це дало змогу адаптуватися до нього, тим більш він є дуже зручним для звичайного користувача та програмний код VST плагінів рекомендується створювати саме в цій середі.

Рисунок 1.9 – Логотип Microsoft Visual Studio 2019

Окрім цього написання плагіну VST зумовлює використання мови C++. У мові C++ упор робиться на об'єктно-орієнтований підхід до структури програми у вигляді використання класів уявлення об'єктів.

Рисунок 1.10 – Логотип мови програмування C++

Дані, пов'язані з об'єктом, інкапсулюються ним і невидимі поза об'єктом. Зв'язок з об'єктом здійснюється за допомогою функцій-членів, які працюють з інкапсульованими даними.

З іншого боку, існує сильна підтримка ієрархії об'єктів. Нові типи об'єктів можуть бути похідними від базового об'єкта, а функції-члени базового об'єкта можуть бути замінені аналогічними функціями, що надаються похідними об'єктами.

Таке розташування дає явну перевагу з погляду модульності програми. Наприклад, основній програмі може знадобитися доступ до безлічі різних різновидів похідних об'єктів у багатьох різних частинах коду, але необхідно посилатися лише на базові об'єкти, а не на похідні об'єкти.

1.4 Використання допоміжних засобів

Створення VST-плагіну є досить складним процесом. Steinberg пропонує власний продукт для створення плагіну, під назвою VST 3 SDK та VST 4 GUI, але новачку, який досконало не знає структуру цих програм, не впоратися. Гарну альтернативу запропонувала компанія Raw Material Software ltd, яка розробила фреймворк JUCE [20], ним зараз користується більшість розробників аудіо-програмного забезпечення.

Рисунок 1.10 – Логотип фреймворку JUCE

Інакше кажучи, JUCE – це кодова база C++ з відкритим кодом, яку можна використовувати для створення автономного програмного забезпечення на Windows, macOS, Linux, iOS і Android, а також плагінів VST, VST3, AU, AUv3, AAX і LV2.

JUCE дозволяє розробникам зосередитися на найцінніших частинах свого програмного забезпечення, дбаючи про відмінності між операційними системами (як настільними, так і мобільними) і форматами плагінів. Завдяки бібліотеці блоків обробки цифрового аудіо (DSP) JUCE можна швидко створювати прототипи, розгортати різні звукові ефекти, фільтри, інструменти та генератори та випускати різні програми та плагіни з узгодженим користувацьким досвідом на всіх підтримуваних платформах. Використання фреймворку також захищає продукти від оновлень операційної системи та плагінів. JUCE надає абстракцію для обробки зразків аудіо та MIDI з різних аудіопристроїв на кожній платформі або хост-DAW.

JUCE надає універсальну абстракцію інтерфейсу користувача, яка може працювати на будь-якій платформі, з можливістю апаратного прискорення через OpenGL. Фреймворк обробляє візуалізацію двовимірної та тривимірної графіки, а також вибір форматів зображень і шрифтів. Усі віджети інтерфейсу користувача JUCE можна тематизувати, що дозволяє мати узгоджену взаємодію з користуванням різними продуктами та платформами.

1.5 Аналіз аналогічних систем генерування музичних мелодій

Одним із тих музичних застосунків, який дає змогу генерувати власні музичні мелодії є «THE_INSTRUMENT» (рис. 1.11) [17], розроблений Філом Спайсером. Його фраза «Якщо я можу зробити пісні, які накопичили сотні мільйонів прослуховувань, використовуючи мої звуки та інструменти... ти ви теж можете це зробити!» напряму вказує на мету та ціль створеного їм плагіну – полегшити та допомогти зі створенням власних пісень та мелодій неосвіченим або навпроти компетентним у музичній сфері людям.

Рисунок 1.11 – Вигляд плагіну THE_INSTRUMENT

Окрім режиму генерування мелодій, плагін також має інші опції його застосування. Плагін має певну базу даних звуків, які відповідають музичним

інструментам, таким як: піаніно, гітара, духові інструменти і т.д. Сам процес створення мелодії відбувається наступним чином: користувач має можливість обрати один з деякої кількості пресетів (рис. 1.12) прогресії акордів, після чого власноруч побудувати акорди в своїй DAW чи просто перетягнути пресет на аудіодоріжку.

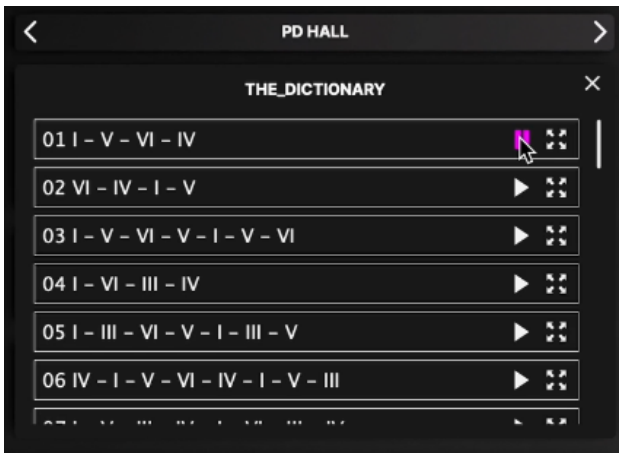

Рисунок 1.12 – Вибір пресету в плагіні THE_INSTRUMENT

Після цього, користуючись можливостями та варіантами налаштування запропонованої плагіном мелодії, людина формує власне звучання. Натискаючи кнопку «Record», мелодія буде записана та збережена в форматі аудіофайлу – mp3, ogg або wav. Також застосунок пропонує зберігти її в MIDI-форматі.

Перевагою цього плагіну можна назвати наявність великої базі музичних інструментів та простоту в використанні, але в дійсності генерація мелодій відбувається через здобуття самих музичних акордів з вже прописаного переліку пресетів і полягає лише у формуванні власного унікального звучанння.

Справжнім показником роботи штучного інтелекту зі звуком є застосунок «MUSIA» (рис. 1.13) [11].

Плагін був створений компанією Creative Mind і пропонує шість головних функції. Однією з них і є створення музичної прогресії акордів під назвою «Auto Chord».

Рисунок 1.13 – Зовнішній вигляд застосунку MUSIA

Концепція роботи дуже проста: користувач налаштовує тональність (рис. 1.14) для своєї прогресії, після чого натискає компонент-кнопку «Auto Chord» (рис. 1.15).

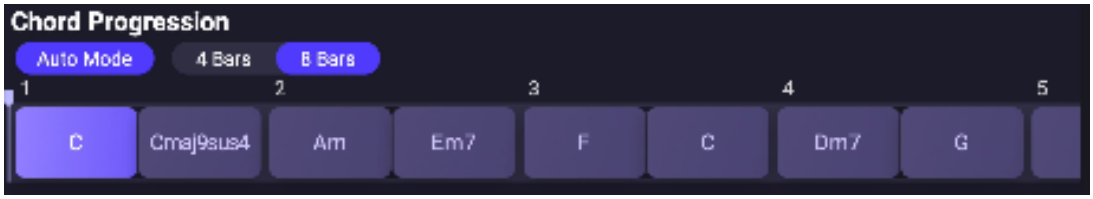

Рисунок 1.14 – Зовнішній вигляд елементу вибору тональності

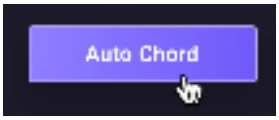

Рисунок 1.15 – Зовнішній вигляд елементу вибору тональності

По завершенню операції створення акордів, вони будуть виведені в формат даних MIDI та користувач отримає змогу завантажити їх у вигляді файлу, або напряму вивести їх у свою DAW.

Висновки до розділу 1

Створення VST SDK плагіну є дуже складним та комплексним процесом. Програмну реалізацію застосунку було вирішено розробляти у середовищі Microsoft Visual Studio 2019, оскільки автор є досить адаптованим до неї. В якості

цифрової звукової робочої станції була обрана FL Studio, за рахунок своєї оптимізованості до різного класу користувачів. Типом розробляємого плагіну VST SDK проєкт. Також, задля покращення варіативності його методів, було використано фреймворк JUCE.

Розглянуті музичні застосунки «THE_INSTRUMENT» та «MUSIA» демонструють значні можливості сучасних технологій у створенні музики.

«THE_INSTRUMENT», розроблений Філом Спайсером, має на меті полегшити процес створення музичних мелодій для користувачів з різним рівнем музичної підготовки. Завдяки інтуїтивно зрозумілому інтерфейсу та великій базі даних звуків, користувачі можуть створювати унікальні мелодії шляхом вибору пресетів прогресій акордів та налаштування власного звучання. Збереження мелодій у форматах mp3, ogg, wav або MIDI додає гнучкості у використанні створених композицій.

«MUSIA», розроблений компанією Creative Mind, є прикладом використання штучного інтелекту у музичній сфері. Застосунок пропонує шість основних функцій, серед яких особливо виділяється «Auto Chord». Ця функція автоматично генерує музичні прогресії акордів на основі заданої тональності, що значно спрощує процес створення музики. Генеровані акорди можуть бути збережені у форматі MIDI, що дозволяє легко інтегрувати їх у будь-яку цифрову аудіо робочу станцію (DAW).

Обидва плагіни надають широкі можливості для створення музики, поєднуючи простоту використання з потужними функціями. Це робить їх доступними як для новачків, так і для досвідчених музикантів, сприяючи креативності та інноваціям у музичному виробництві.

2 НАЛАШТУВАННЯ ПРОЄКТУ ТА ЙОГО СТРУКТУРА

2.1 Бібліотеки

Оскільки створення логіки генерування мелодії було виконано з використанням фреймворку JUCE, то всі необхідні методи та функції спираються на бібліотеки даного інструментарію.

Бібліотека **juce_audio_devices** - містить оголошення класів і функцій, що стосуються роботи з аудіо пристроями.

```
#include <juce_audio_devices/juce_audio_devices.h>
```

Ця бібліотека надає розробникам інтерфейси для взаємодії з аудіо пристроями, такими як мікрофони, аудіо інтерфейси, аудіо відтворювачі та інші аудіо пристрої, які можуть бути доступні на пристрої.

Деякі з класів і функцій, які можуть бути оголошені у бібліотеці **juce_audio_devices**, включають:

- клас **AudioIODevice**: Цей клас представляє аудіо пристрій і надає методи для взаємодії з ним, такі як відкриття та закриття пристрою, встановлення параметрів відтворення та запису тощо;
- клас **AudioIODeviceCallback**: Цей клас є базовим класом для зворотних викликів аудіо пристроїв, які використовуються для отримання або передачі аудіо даних між програмою і аудіо пристроєм;
- функції для отримання списку доступних аудіо пристроїв, їх властивостей та параметрів;
- класи і функції для роботи з MIDI-протоколом, який дозволяє взаємодіяти з MIDI-контролерами і пристроями;
- функції для керування аудіо потоками, включаючи відтворення, запис та обробку аудіо даних.

Juce_audio_formats входить до складу бібліотеки JUCE і містить оголошення класів і функцій, які дозволяють робити роботу з різними аудіо форматами.

```
#include <juce_audio_formats/juce_audio_formats.h>
```

Ця бібліотека дозволяє вам завантажувати, зберігати та маніпулювати аудіофайлами у різних форматах, таких як WAV, AIFF, FLAC, Ogg Vorbis і багатьох інших. Ось кілька класів і функцій, які можна знайти в цьому файлі:

- клас **AudioFormat**: Цей клас представляє аудіоформат і має методи для зчитування та запису аудіоданих у файл цього формату;
- клас **AudioFormatReader** та **AudioFormatReaderSource**: Ці класи використовуються для зчитування аудіоданих з файлу в об'єкт **AudioFormatReader**, який потім може бути використаний для відтворення аудіо чи отримання аудіоданих;
- клас **AudioFormatWriter** та **AudioFormatWriterSource**: Ці класи дозволяють записувати аудіодані у файл в заданому аудіоформаті;
- різноманітні класи, що відповідають різним аудіоформатам, наприклад, **WavAudioFormat**, **FlacAudioFormat**, **OggVorbisAudioFormat** тощо. Кожен з цих класів містить код для зчитування та запису даних у відповідному форматі;
- функції для визначення доступних аудіоформатів, їх властивостей та параметрів.

J**uce_core** - це один із модулів мови програмування C++, який надається фреймворком JUCE (Jules' Utility Class Extensions).

```
#include <juce_core/juce_core.h>
```

JUCE є великим інструментом для розробки аудіо-плагінів, аудіо-додатків та мультимедійного програмного забезпечення. Модуль **juce_core** є одним з основних компонентів бібліотеки JUCE і пропонує базові утиліти та класи для виконання різних завдань, таких як робота з рядками, файлами, потоками та основними структурами даних. Він надає абстракційний рівень над платформо-залежною функціональністю, що полегшує розробку крос-платформеного програмного забезпечення. Розробники часто використовують **juce_core** як основу для створення складніших додатків або плагінів з використанням JUCE.

2.2 Необхідні приготування для створення проєкту

Під час першого запуску Projucer (рис. 2.1) з'являється нове вікно проекту. Також можна запустити це пізніше, вибравши «Новий проект...» у головному меню Projucer.

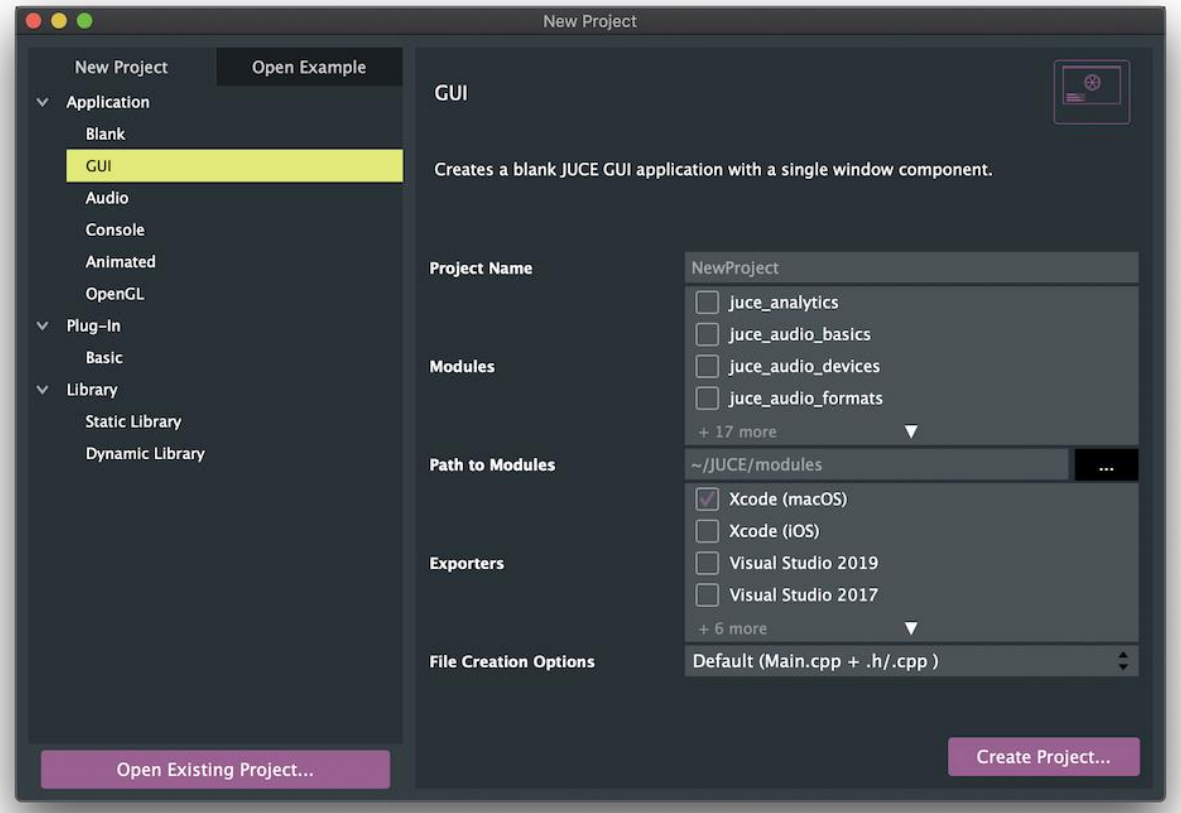

Рисунок 2.1 - Projucer - нове вікно проекту

У лівій частині вікна був вибраний тип проекту, який необхідно було створити.

Для кожного типу проекту Projucer генерує всі файли проекту та додає відповідний мінімальний код, який все налаштовує. Таким чином, після того, як проєкт був створений, можна негайно розпочати розробку фактичної функціональності.

Якщо ви вперше досліджуєте JUCE і не знаєте, з чого почати, виберіть програму з графічним інтерфейсом користувача — це налаштує усі основи робочої програми як для настільних, так і для мобільних платформ.

Нижче наведено огляд усіх підтримуваних типів проектів:

Таблиця 2.1 – Типи проєктів

Тип проєкту	**Опис**
Заява/Бланк	Це створює пусту програму JUCE.
Програма/графічний інтерфейс	Це створює мінімальну програму JUCE з порожнім вікном програми. Можна почати тут і додати більше функціональних можливостей, наприклад більше компонентів GUI, використовуючи різні класи, які пропонує JUCE.
Додаток/Аудіо	Це створює мінімальну програму JUCE, як-от Application/GUI, але автоматично додає весь код налаштування, потрібний для легкого введення та виведення звуку. Можна використовувати це для ігор, мультимедійних програм та багато іншого.
Додаток/консоль	JUCE також дуже корисний для розробки програм командного рядка, які взагалі не мають графічного інтерфейсу. Можна використати цей тип проекту для створення такої програми.
Аплікація/Анімація	Це створює програму, яка малює анімований графічний дисплей. Можна почати тут, наприклад, створити анімований мобільний додаток.
Програма/OpenGL	Це створює порожню програму JUCE, як Application/GUI, але додає підтримку OpenGL для рисування функцій, включаючи імпорт 3D-моделі та шейдери GLSL.

Закінчення таблиці 2.1

Плагін/Базовий	Це створює базовий плагін аудіо. Весь код для підтримки форматів плагінів VST, AudioUnit і AAX додається автоматично. Залежно від налаштувань цей тип проекту може потребувати деяких додаткових підготовчих кроків для правильної роботи.
Бібліотека/Статичний, Бібліотека/Динамічний	Цей тип проекту корисний для створення багаторазових бібліотек програмного забезпечення, які створюються на основі JUCE. Projucer підтримує створення бібліотек як для статичного, так і для динамічного зв’язування.

При створенні проєкту необхідно вказати назву – мій застосунок має ім’я «GENER», що одночасно є співзвучним зі словом «генерувати», розповідаючи про мету плагіну, а також має схоже звучання зі словом «жанр».

Рисунок 2.2 – Логотип проєкту GENER

Після надання застосунку назви, необхідно закінчити ряд ще деяких налаштувань. Наступним і дуже важливим кроком буде вибір формату плагіна, а також, в моєму випадку, визначення місцеположення інструментарію VST3 SDK (рис. 2.3) на комп’ютері, оскільки я визначив для розробки саме цей формат.

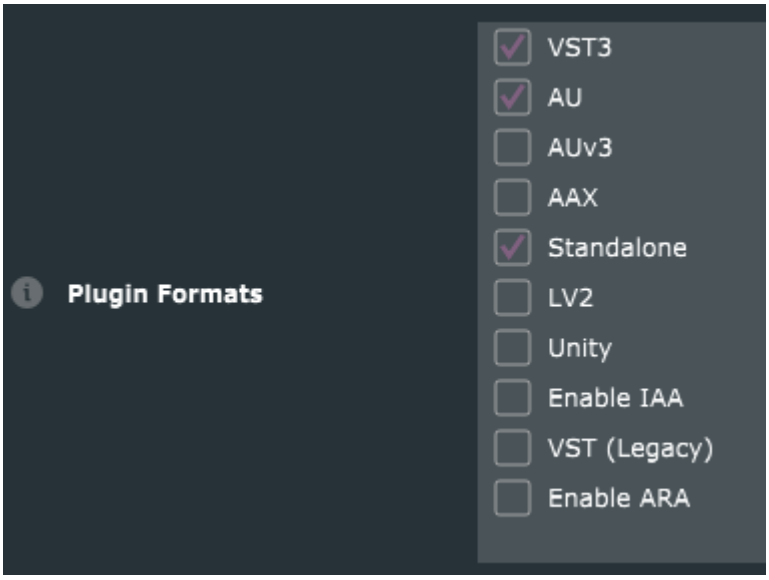

Рисунок 2.3 – Визначення формату плагіна

Далі необхідно було визначити характеристику застосунку (рис. 2.4). Оскільки моя програма зумовлює написання мелодії, яка по завершенню процесу буде виведена у формат MIDI даних, то відповідним варіантом буде «Plugin MIDI Output».

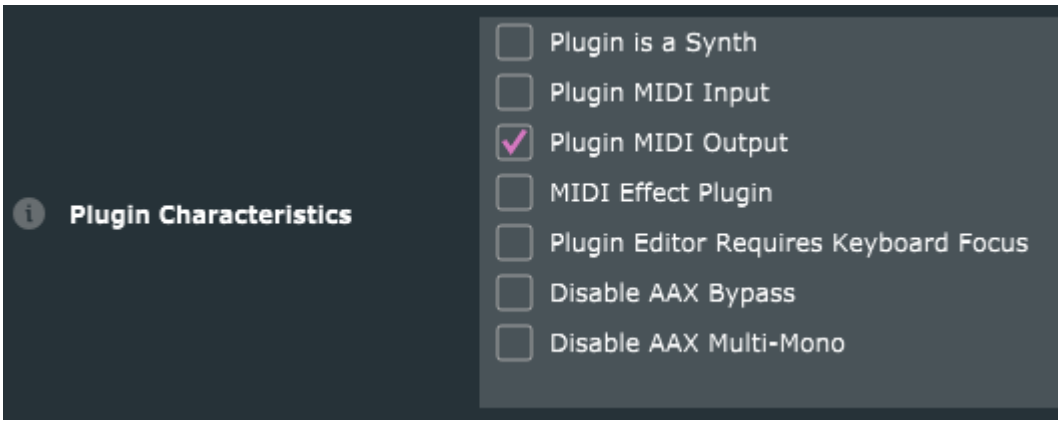

Рисунок 2.4 – Визначення характеристики плагіна

2.3 Структура проєкту

Варто ближче розглянути анатомію проекту Projucer, зображену на рисунку 2.5. Нижче наведено структуру папок проекту JUCE, який Projucer створює для користувача:

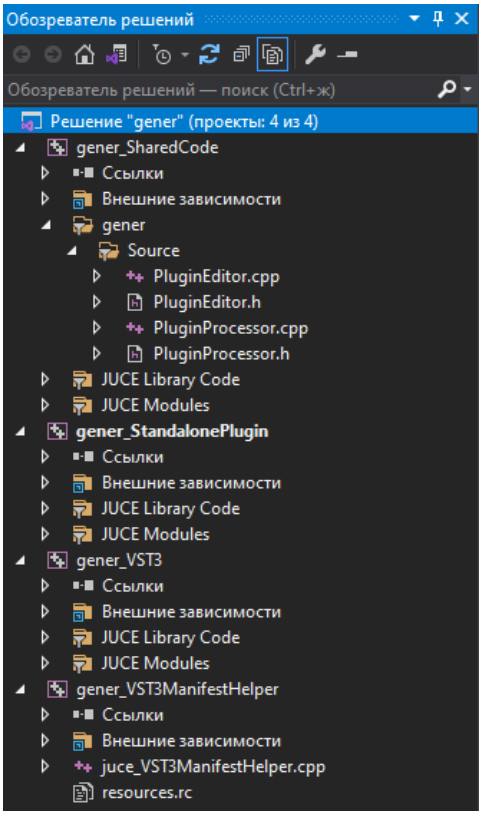

Рисунок 2.5 – Анатомія проекту GENER

На верхньому рівні знаходиться виділене рішення gener, яке містить усі налаштування проекту. Рішення вміщує в себе чотири каталоги, одним із них є gener_SharedCode, який налічує три найголовніші папки:

Таблиця 2.2 – Найголовніші папки проєкту

Папка	**Вміст**
Source	Вихідний код C++ програми JUCE.
JuceLibraryCode	Тут кілька автоматично згенерованих файлів заголовків, які включають код бібліотеки JUCE через модулі JUCE. Важливо, що фактичний код бібліотеки JUCE знаходиться не тут, а у глобальній папці JUCE, яка була інстальована раніше.
JUCE Modules	Зазвичай містить класи, які представляють дані та логіку програми. Ці класи часто структуровані

Закінчення таблиці 2.2

	відповідно до шаблону проектування Model-View-Controller (MVC) або подібних архітектур.

В папці Source можна знайти чотири найважливіших файли застосунку.

Клас **PluginEditor** є одним із ключових елементів у розробці аудіо плагінів за допомогою JUCE. Він відповідає за створення та управління графічним інтерфейсом користувача (GUI) плагіну. Основним завданням **PluginEditor** є візуалізація параметрів плагіну та надання можливості користувачу взаємодіяти з ними шляхом взаємодії з елементами GUI.

Інтерфейс користувача, що створюється в **PluginEditor**, дозволяє користувачам візуалізувати параметри плагіну, такі як рівень гучності, частоту, коефіцієнти обробки звуку тощо. Це забезпечує зручний та інтуїтивно зрозумілий інтерфейс для користувачів плагіну, що дозволяє їм з легкістю налаштовувати звукові параметри під свої потреби.

Один із ключових методів у класі **PluginEditor** - це метод **resized()**. Цей метод викликається при зміні розміру вікна плагіну та дозволяє адаптувати розміщення та розмір елементів GUI до нового розміру вікна. Використовуючи цей метод, розробники можуть забезпечити коректне відображення GUI навіть при зміні розміру вікна плагіну.

Крім того, в **PluginEditor** можуть бути додаткові методи для ініціалізації та оновлення GUI, а також для обробки подій, які виникають внаслідок взаємодії користувача з елементами GUI. Наприклад, вони можуть містити методи для обробки натискання кнопок, переміщення регуляторів або зміни значень полів введення.

Загалом, клас **PluginEditor** є важливим елементом у створенні аудіо плагінів з візуальним інтерфейсом користувача, що надає можливість ефективно взаємодіяти з плагіном та налаштовувати його параметри у зручний спосіб. Він

допомагає покращити користувацький досвід користувачів плагіну та забезпечити йому більшу функціональність та зручність використання.

Клас **PluginProcessor** є одним із найважливіших компонентів у розробці аудіо плагінів з використанням бібліотеки JUCE. Він відповідає за обробку аудіо сигналів та управління їхніми параметрами в межах плагіну. Головною метою **PluginProcessor** є обробка вхідного аудіо сигналу, застосування аудіо ефектів або обробки до нього та генерація вихідного аудіо сигналу.

Основний метод у класі **PluginProcessor** - це метод **processBlock()**. Цей метод викликається при обробці нового блоку аудіо даних, який надходить від вхідних аудіо каналів плагіну. У цьому методі реалізується основний алгоритм обробки аудіо, включаючи застосування аудіо ефектів, обробки сигналу та інші аудіо операції, необхідні для досягнення бажаного звукового ефекту.

Крім того, у класі **PluginProcessor** можуть бути реалізовані додаткові методи для ініціалізації та налаштування параметрів плагіну, таких як зміна рівня гучності, частоти, регулювання фільтрів тощо. Ці методи дозволяють підтримувати та налаштовувати функціонал плагіну в залежності від потреб користувача.

Однією з важливих характеристик **PluginProcessor** є його платформонезалежність. Завдяки бібліотеці JUCE, **PluginProcessor** може бути розроблений один раз і запущений на різних операційних системах, таких як Windows, macOS та Linux, забезпечуючи єдинообразну роботу плагіну на різних платформах.

Узагальнюючи, клас **PluginProcessor** відіграє важливу роль у розробці аудіо плагінів, забезпечуючи обробку аудіо сигналів та управління їхніми параметрами. Він є ключовим елементом у досягненні бажаних звукових ефектів та забезпечує зручний та ефективний інтерфейс для взаємодії з аудіо даними у межах плагіну.

Висновки до розділу 2

Отже, реалізація логіки генерації мелодії в даному проекті базується на використанні фреймворку JUCE, який забезпечує багатий набір інструментів для

роботи з аудіо. Зокрема, використання бібліотек juce_audio_devices, juce_audio_formats та juce_core відіграє ключову роль у забезпеченні функціональності проекту.

Модуль juce_core є основним компонентом фреймворку JUCE і забезпечує базові утиліти та класи для роботи з рядками, файлами, потоками та основними структурами даних. Він полегшує розробку крос-платформеного програмного забезпечення, забезпечуючи абстракцію над платформо-залежною функціональністю.

Завдяки цим бібліотекам фреймворку JUCE, розробники можуть ефективно взаємодіяти з аудіо пристроями, працювати з різними аудіо форматами та використовувати основні утиліти для створення складних аудіо додатків і плагінів. Це робить JUCE потужним інструментом для розробки аудіо програмного забезпечення.

3 ЗОВНІШНІЙ ВИГЛЯД ТА КОНЦЕПЦІЯ РОБОТИ ПЛАГІНУ

3.1 Візуальна складова

Плагін «GENER» є результатом роботи, спрямованої на створення інноваційного засобу для музичної творчості. Усі елементи дизайну були виконані самостійно завдяки графічному редактору Adobe Photoshop. Зовнішній вигляд плагіну «GENER» зображений на рисунку 3.1.

Рисунок 3.1 – Зовнішній вигляд плагіну «GENER»

Застосовуючи інструменти графічного дизайну у програмі Adobe Photoshop, було створено вигляду плагіну, який би відображав його унікальний характер та функціональність. Дизайн «GENER» виконаний в досить популярному нині стилі Sci-Fi, відзначається мінімалістичним підходом, відтінками наукової фантастики та лаконічними формами, що сприяють зручному користуванню. Така зовнішня оболочка надає плагіну ретро-футуристичного вигляду.

У плагіні реалізовані три основні функції, кожна з яких відображена окремою кнопкою. Перша кнопка (рис. 3.2) дозволяє генерувати інструментальні мелодії в якості акордів записаних у MIDI-дані, надаючи користувачеві можливість

створювати унікальні музичні композиції. Після завершення процесу дані зберігаються у файл в визначеній користувачем папці.

Рисунок 3.2 – Кнопка генерації в плагіні «GENER»

Друга кнопка (рис. 3.3) відповідає за генерацію басової партій, що додає глибину та ритмічність створюваним супровідам.

Рисунок 3.3 – Кнопка генерації басової партії в плагіні «GENER»

Третя кнопка (рис. 3.4) виконує функцію збереження згенерованих даних у форматі MIDI, забезпечуючи можливість подальшого редагування та використання у різних музичних програмах.

Рисунок 3.4 – Кнопка генерації барабанної партії в плагіні «GENER»

Однією з ключових особливостей "GENER" є присутність ComboBox (рис. 3.5), що дозволяє користувачеві обирати тоновість для генерованих мелодій.

Рисунок 3.5 – ComboBox компонент вибіру тональності «GENER»

Ця функція розширює можливості плагіну, дозволяючи адаптувати його до конкретних музичних потреб користувача.

3.2 Реалізація кнопок

В середовищі розробки аудіо-плагінів за допомогою JUCE (Jules' Utility Class Extensions) кнопка generateButton є важливим компонентом користувацького інтерфейсу, що дозволяє користувачеві генерувати музичні мелодії. Ця кнопка створює інтерактивний елемент, який активує процес генерації MIDI файлів при натисканні. У цьому рефераті розглядається детальна програмна реалізація generateButton, включаючи її ініціалізацію, налаштування, візуальне оформлення та функціональність.

У конструкторі класу GenerAudioProcessorEditor кнопка generateButton проходить процес ініціалізації. Спершу визначається розмір головного вікна редактора за допомогою функції setSize.

Позиція та розміри кнопки generateButton встановлюються за допомогою методу setBounds. Ці координати визначають розташування кнопки у вікні інтерфейсу.

```
generateButton.setBounds(572, 70, 420, 420);
```

Для надання кнопці графічного оформлення використовуються кілька зображень, які завантажуються з локальних файлів за допомогою методу juce::ImageFileFormat::loadFrom.

Кнопка generateButton налаштовується для використання цих зображень у різних станах (звичайний, активний) за допомогою методу setImages.

```
generateButton.setImages(false, true, true,
   normalImage, 1.0f, juce::Colours::transparentBlack,
   activeImage, 1.0f, juce::Colours::transparentBlack,
   activeImage, 1.0f, juce::Colours::transparentBlack);
```

Кнопка generateButton додається до користувацького інтерфейсу з допомогою методу addAndMakeVisible.

```
addAndMakeVisible(generateButton);
```

Для визначення дії, що виконується при натисканні кнопки, використовується лямбда-функція, яка встановлюється через властивість onClick кнопки generateButton. Ця функція викликає метод generateMIDIFile класу audioProcessor, що відповідає за генерацію MIDI файлу.

```
generateButton.onClick = [this] {
   audioProcessor.generateMIDIFile();
};
```

У методі paint класу GenerAudioProcessorEditor завантажується та відображається фонове зображення для інтерфейсу плагіна.

Крім кнопки generateButton, у інтерфейсі додаються й інші елементи, такі як bassButton, drumButton, та випадаючий список keySelector.

```
addAndMakeVisible(bassButton);
addAndMakeVisible(keySelector);
addAndMakeVisible(drumButton);
```

Реалізація кнопки generateButton у середовищі JUCE демонструє комплексний підхід до створення інтерактивних елементів користувацького інтерфейсу аудіо-плагіна. Процес включає налаштування візуального оформлення, розміщення на екрані та визначення функціональних дій, що дозволяють користувачеві генерувати музичні мелодії у форматі MIDI. Такий підхід забезпечує зручність та інтуїтивність роботи з плагіном.

3.3 Реалізація зміни музичної тональності

Зміна музичної тональності є ключовою функціональністю в багатьох аудіо-додатках і плагінах. В середовищі розробки аудіо-плагінів за допомогою JUCE ця функціональність може бути реалізована за допомогою випадаючого списку (ComboBox), який дозволяє користувачам вибирати бажану тональність. У цьому рефераті розглядається детальна програмна реалізація компонента для зміни музичної тональності, включаючи його ініціалізацію, налаштування, обробку подій та інтеграцію з основним аудіопроцесором.

У конструкторі класу GenerAudioProcessorEditor створюється і налаштовується випадаючий список keySelector. Цей компонент дозволяє користувачеві вибирати музичну тональність з наданого списку.

```
keySelector.addItemList(keyNames, 1);
keySelector.setSelectedId(1);
```

Випадаючий список додається до інтерфейсу користувача за допомогою методу addAndMakeVisible.

```
addAndMakeVisible(keySelector);
```

Список тональностей завантажується у ComboBox за допомогою методу addItemList. keyNames містить перелік назв тональностей, які можуть бути обрані.

```
juce::StringArray keyNames = { "C", "C#", "D", "D#", "E", "F", "F#", "G", "G#", "A", "A#", "B" };
```

Коли користувач вибирає нову тональність зі списку, викликається лямбда-функція, встановлена через властивість onChange компонента keySelector. Ця функція зберігає індекс вибраного елемента та виконує необхідні дії.

```
keySelector.onChange = [this] {
    selectedKeyIndex = keySelector.getSelectedItemIndex();
    repaint();
    audioProcessor.setSelectedKey(selectedKeyIndex);
};
```

У методі paint класу GenerAudioProcessorEditor відображається графічне зображення вибраної тональності. Це забезпечує візуальний зворотний зв'язок для користувача.

Для досягнення естетичного вигляду випадаючий список робиться прозорим.

```
keySelector.setAlpha(0);
keySelector.setBounds(155, 380, 430, 126);
```

Коли користувач обирає нову тональність, цей вибір передається до основного аудіопроцесора для обробки. Метод setSelectedKey аудіопроцесора відповідає за збереження та обробку вибраної тональності.

```
audioProcessor.setSelectedKey(selectedKeyIndex);
```

Реалізація компонента для зміни музичної тональності в середовищі JUCE є важливим аспектом створення інтерактивного аудіо-плагіна. Цей компонент надає користувачам можливість легко вибирати тональність та отримувати візуальний зворотний зв'язок про свій вибір. Завдяки інтеграції з аудіопроцесором, вибрана тональність може бути використана для генерації музичних мелодій у правильній тональності. Такий підхід забезпечує зручність використання та функціональність аудіо-плагіна.

3.4 Опис процесу генерування MIDI-даних

Процес генерації мелодії плагіну (якого типу вона не була б), полягає в тому, що коли потенційний користувач, є він звичайною людиною або професійним музикантом, обирає музичний ключ, враховуючи його жанрові потреби і цілі, після чого натискає на відповідну кнопку, яка в свою чергу створює MIDI файл, відштовхуючись від критеріїв, виводячи його в файл формату mid.

Дизайн плагіну створений таким чином, що в першу чергу людина помічає велику кнопку з написом "g" - компонент генерації інструментальних акордів. Дані вміщують числові змінні, які відповідають довжині та позиції певної ноти. Наприклад, створена нота D, тобто нота «Ре», буде мати позицію 62, оскільки так

вказано параметрами клавіатури, яка використовується для створення музики в MIDI, які було зазначено вище в звіті.

Програма надає ще декілька опцій для продовження насичення власної музичної композиції. Однією з них є можливість надання треку басової партії. Важливо помітити необхідну ремарку, що більшість музикантів, які використовують звукові станції, є представниками тих жанрів музичної індустрії, що не потребують об'ємних музичних партій.

Мелодії зазвичай налічують нотну базу у вигляді одного акорду, що по своїй суті дорівнює трьом нотам. Це стосується як і ведучого музичного інструменту, так і басового. Тому задля простоти та враховуючи обмеженість часу, було прийнято рішення для плагіну "GENER" складати басову партію лише по трьом нотам, що насправді і так дуже насичено, оскільки в піснях жанру хіп-хоп, наприклад, зазвичай налічується тільки одна. Якщо пісня знаходиться у тональності "До", то й бас буде "лежати" тільки на цій ноті.

Висновки до розділу 3

Плагін «GENER» був створений як інноваційний інструмент для музичної творчості. Дизайн плагіну, виконаний у популярному стилі Sci-Fi за допомогою Adobe Photoshop, підкреслює його унікальність та функціональність. Плагін має три основні функції, представлені окремими кнопками для генерації інструментальних мелодій, басових партій та барабанних партій, які зберігаються у форматі MIDI. Важливою особливістю «GENER» є наявність ComboBox для вибору тональності, що дозволяє користувачам адаптувати плагін під свої музичні потреби. Процес генерації MIDI-даних забезпечує створення файлів з інформацією про позиції та довжину нот, що робить його зручним для використання як професійними музикантами, так і аматорами.

У цьому розділі було детально розглянуто реалізацію ключових компонентів користувацького інтерфейсу аудіо-плагіна в середовищі розробки JUCE: кнопки generateButton та випадаючого списку keySelector. Обидва компоненти відіграють

важливу роль у функціональності плагіна, забезпечуючи користувачам зручний та інтуїтивно зрозумілий інтерфейс для взаємодії з програмою.

Реалізація кнопки generateButton показує важливість детального налаштування елементів інтерфейсу для досягнення інтерактивності та функціональності. Процес включає ініціалізацію, налаштування розмірів та позиції, а також встановлення графічного оформлення для різних станів кнопки. Лямбда-функція, прив'язана до події натискання, дозволяє кнопці безпосередньо викликати метод generateMIDIFile класу audioProcessor, забезпечуючи ефективну генерацію MIDI файлів. Такий підхід забезпечує користувачам зрозумілий та зручний спосіб взаємодії з основною функцією плагіна.

Реалізація випадаючого списку keySelector підкреслює важливість надання користувачам можливості вибору музичної тональності. Ініціалізація списку, завантаження тональностей, обробка подій зміни вибору та візуальний зворотний зв'язок є критичними аспектами цього компонента. Використання лямбда-функції для обробки подій дозволяє зберігати вибрану тональність та передавати її до аудіопроцесора, що забезпечує правильну генерацію музичних мелодій відповідно до вибраної тональності. Візуальне відображення вибраної тональності через метод paint покращує користувацький досвід, роблячи інтерфейс більш інформативним та привабливим.

Загалом, реалізація кнопки generateButton та випадаючого списку keySelector в середовищі JUCE демонструє важливість комплексного підходу до створення інтерфейсу аудіо-плагіна. Ці компоненти не лише забезпечують необхідну функціональність, але й роблять програму зручною та інтуїтивно зрозумілою для користувачів. Інтеграція з аудіопроцесором дозволяє ефективно виконувати основні завдання плагіна, такі як генерація мелодій та вибір тональності, що в кінцевому результаті підвищує загальну ефективність та привабливість плагіна.

4 ПРОГРАМНА РЕАЛІЗАЦІЯ МЕХАНІК СТВОРЕННЯ МУЗИЧНОЇ МЕЛОДІЇ

4.1 Реалізація логіки генерування музичних мелодій

Одним з найголовніших факторів при створенні правильного програмного коду було врахування критеріїв та характеристик самої DAW в обличчі FL Studio. Необхідно було врахувати, що ця звукова робоча станція відрізняється від інших тим, що використовує дещо іншу систему будування мелодії загалом. Наприклад, в альтернативних варіантах систем, які пропонує музична індустрія, зазвичай використовується інша кількість «тіків». У FL Studio "тік" - це одиниця часу, яка використовується для послідовного розташування та синхронізації подій у програмі. Роздільна здатність внутрішнього таймінгу FL Studio вимірюється в тиках на чвертьну ноту (PPQ). За замовчуванням FL Studio працює з роздільною здатністю 96 тиків на чвертьну ноту, коли в тому ж Ableton, або Cubase, за стандартом використовується лише 480. Але в FL Studio це можна налаштувати у налаштуваннях програми на вищі значення для більш точного та детального синхронізації. Логіка генерації музичних мелодій у методі generateMIDIFile() JUCE аудіо плагіна базується на використанні MIDI файлів для створення випадкових мелодій у певній музичній тональності. Було наведено детальний опис кроків, що використовуються у цьому методі.

Метод починається з ініціалізації об'єкту MIDI файлу та послідовності MIDI повідомлень:

```
juce::MidiFile midiFile;
midiFile.setTicksPerQuarterNote(960);
juce::MidiMessageSequence sequence;
midiFile.addTrack(sequence);
```

В даному відрізку коду:

- MidiFile – створює порожній об’єкт MidiFile;

- setTicksPerQuarterNote – встановлює формат часу для використання, коли цей файл записується в потік;
- MidiMessageSequence – створює порожній об’єкт міді-послідовності;
- addTrack – додає міді-доріжку до файлу (Це створить власну внутрішню копію переданої послідовності).

Встановлення поділів на четвертину ноти (ticks per quarter note) визначає часовий розподіл MIDI подій. Значення 960 є стандартним для цифрової звукової робочої станції FL Studio і забезпечує достатню точність.

Мапа keyNotes асоціює індекси тональностей з відповідними наборами нот:

```
std::map<int, std::vector<int>> keyNotes = {
{0,  {60, 62, 64, 65, 67, 69, 71}},
};
```

Вище наведено відривок коду, який демонструє набір нот для тональності «До-мажор». Кожний набір нот відповідає певній мажорній тональності, представленій у MIDI значеннях. Конкретно для наведеної тональності доцільно і правильно використовувати наступний набір нот:

- C – нота «До», відповідає значенню 60 згідно до MIDI-стандарту;
- D – нота «Ре», відповідає значенню 62 згідно до MIDI-стандарту;
- E – нота «Ми», відповідає значенню 64 згідно до MIDI-стандарту;
- F – нота «Фа», відповідає значенню 65 згідно до MIDI-стандарту;
- G – нота «Соль», відповідає значенню 67 згідно до MIDI-стандарту;
- A – нота «Ля», відповідає значенню 69 згідно до MIDI-стандарту;
- B – нота «Сі», відповідає значенню 71 згідно до MIDI-стандарту.

Індекс обраної тональності отримується з редактора, посилаючись на компонент ComboBox під назвою keySelector та його метод selectedKeyIndex:

```
int selectedKeyIndex =
dynamic_cast<GenerAudioProcessorEditor*>(getActiveEditor())-
>getSelectedKeyIndex();
```

Цей індекс використовується для вибору відповідного набору нот з мапи keyNotes.

Також використовується генератор випадкових чисел для створення мелодії, спираючись на ноти обраної тональності:

```
std::random_device rd;
```

Цей код створює екземпляр std::random_device, який є недетермінованим генератором випадкових чисел, він використовується для заповнення інших генераторів.

```
std::mt19937 gen(rd());
```

Тут std::mt19937 — це генератор випадкових чисел Mersenne Twister. Його заповнюють значенням, наданим rd(). Mersenne Twister відомий своєю високоякісною генерацією випадкових чисел.

```
std::uniform_int_distribution<> dis(0, keyNotes[selectedKeyIndex].size() - 1);
```

Це створює рівномірний розподіл цілих чисел у діапазоні [0, size -1], де size(розмір) — це кількість елементів у keyNotes[selectedKeyIndex]. Об’єкт dis гарантує, що кожен індекс у цьому діапазоні має однакову ймовірність бути обраним.

```
int previousNote = keyNotes[selectedKeyIndex][dis(gen)];
```

Цей рядок генерує випадкове ціле число в межах зазначеного діапазону за допомогою dis(gen), який потім отримує доступ до елемента з keyNotes[selectedKeyIndex] за згенерованим випадковим індексом. Вибрана нота призначається попередній ноті.

```
for (int i = 0; i < 8; ++i) {
```

Цикл виконується 8 разів, що означає, що буде згенеровано 8 нот і додано до послідовності MIDI.

```
int note = keyNotes[selectedKeyIndex][dis(gen)];
```

Випадкова нота вибирається з масиву keyNotes за допомогою розподілу, деініціалізованого раніше.

```
int duration = 500;
```

Тривалість кожної ноти становить 500 мілісекунд.

```
sequence.addEvent(juce::MidiMessage::noteOn(1, note, (uint8_t)127), i * duration);
```

Подія примітки додається до послідовності в час початку i * duration. Швидкість встановлена на 127 (максимальна).

```
sequence.addEvent(juce::MidiMessage::noteOff(1, previousNote), (i + 1) * duration);
```

Подія вимкнення ноти для попередньої ноти додається до послідовності на (i + 1) * тривалості.

```
previousNote = note;
```

Поточна нота зберігається як попередня для використання в наступній ітерації. Кожний крок генерує нову випадкову ноту з обраної тональності і створює MIDI події для ввімкнення та вимкнення ноти.

Згенеровані MIDI дані записуються у файл з унікальним іменем:

```
juce::File directory("C:/Users/dadya/Documents/");
juce::String baseName = "GENER";
juce::String extension = ".mid";
```

Створюється об'єкт directory класу juce::File, який представляє каталог (папку) за вказаним шляхом, шлях до каталогу: "C:/Users/dadya/Documents/". Наступним кроком є задіяння об'єкту baseName класу juce::String, який містить строку "GENER" і об'єкт extension класу juce::String, який містить строку ".mid". Це ім'я, яке може бути використане як базове ім'я для файлів.

Далі дуже важливим етапом формування файлу є надання йому змоги мати власний індекс імені задля уникнення повторів при створенні.

```
int fileIndex = 1;
```

Оголошується змінна fileIndex і встановлюється її початкове значення 1. Ця змінна використовується для створення унікальних імен файлів.

```
while (fileToSave.existsAsFile())
{
```

```
    fileName = baseName + "_" + juce::String(fileIndex++) + extension;
    fileToSave = directory.getChildFile(fileName);
}
```

Починається цикл while, який виконується доти, поки файл з поточним ім'ям (яке зберігається в об'єкті fileToSave) існує в файловій системі. Метод existsAsFile() повертає true, якщо файл існує. У циклі формується нове ім'я файлу шляхом конкатенації базового імені (baseName), символу підкреслення ("_"), поточного значення fileIndex (яке інкрементується після використання) та розширення файлу (extension). Об'єкту fileToSave присвоюється новий об'єкт juce::File, який представляє файл з новим іменем у вказаній директорії. Метод getChildFile(fileName) повертає файл з вказаним ім'ям у заданій директорії.

Цей блок коду перевіряє наявність файлів з однаковими іменами та створює нове унікальне ім'я, якщо файл вже існує. Дані зберігаються у визначену директорію.

Метод generateMIDIFile() генерує випадкову мелодію у певній тональності, використовуючи випадкові числа для вибору нот з визначеного набору. Згенеровані MIDI події записуються у файл, забезпечуючи унікальність імені файлу. Цей підхід дозволяє швидко створювати варіації мелодій з заданими параметрами, що може бути корисним для музичних експериментів та автоматичного створення музики.

Цей фрагмент коду гарантує, що новий файл буде мати унікальне ім'я, щоб уникнути перезапису існуючих файлів. Імена файлів (рис. 4.1) будуть виглядати наступним чином:

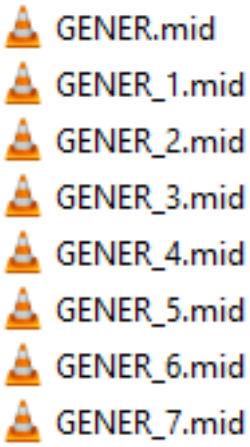

Рисунок 4.1 – Вигляд побудованих імен файлів

```
juce::FileOutputStream outputStream(fileToSave);
```

Об'єкт outputStream класу juce::FileOutputStream відповідає за запис даних у файл. Файл, в який будуть записані дані, задається об'єктом fileToSave, був визначений раніше і має унікальне ім'я.

```
if (outputStream.openedOk())
```

Перевіряється, чи вдалося успішно відкрити потік для запису. Метод openedOk() повертає true, якщо потік був успішно відкритий.

```
midiFile.writeTo(outputStream);
```

Якщо потік був успішно відкритий, викликається метод writeTo(outputStream) для об'єкта midiFile. Об'єкт midiFile представляє MIDI-файл і має метод writeTo, який записує MIDI-дані у вказаний потік outputStream.

4.2 Реалізація логіки генерування басових та барабанних партій

Логіка генерування басових партій у методі generateBassMIDIFile() є схожою до логіки генерування мелодій у generateMIDIFile(), але з деякими відмінностями, що відображають особливості басових партій. Найголовнішою відмінністю є масштаб створеної мелодії. Справа в тому, що більшість жанрів сучасної музичної індустрії, які користуються великою популярністю у теперішньої аудиторії та на які роблять опір сьогоденні музиканти, зумовлюють під собою дуже прості басові

партії. Зазвичай вони не складають великого різноманіття і демонструють від трьох до п’яти різних нот певної тональності. Тому було вирішено створити схожу на генерування основної мелодії логіку, але врахувати відмінність кількості нот.

Загалом програмний код відрізняється лише фрагментом набіру значень в мапі keyNotes. Як і в попередньо зазначеній логіці, keyNotes асоціює індекси тональностей з відповідними наборами нот, але в цьому випадку басових, відповідно, числові значення будуть менше, адже буде використана нижча октава:

```
std::map<int, std::vector<int>> keyNotes = {
{0,  {36, 38, 40}}
```

Даний фрагмент коду демонструє набір числових значень для тональності «До-мажор» і складає лише три ноти:

- C – нота «До», відповідає значенню 36 згідно до MIDI-стандарту;
- D – нота «Ре», відповідає значенню 38 згідно до MIDI-стандарту;
- E – нота «Ми», відповідає значенню 40 згідно до MIDI-стандарту.

Також однією з відмінностей є задання іншої тривалості для ноти, замість 500 було визначено 1000 в якості максимальної довжини.

```
int duration = 1000;
```

Файлу та надання йому індексу створюються ідентично до головної логіки, з єдиною відмінністю у назві файлу, що зображено на рисунку 4.2:

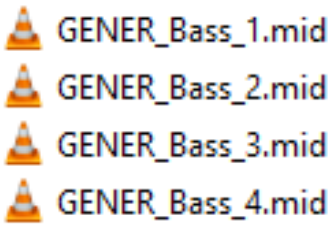

Рисунок 4.2 – Вигляд побудованих імен файлів басової партії

Ситуація з реалізацією генерування барабанної партії є дещо іншої. По-перше початок програмного коду є ідентичним до інших, але сама логіка створення дуже різна. Барабанна партія уявляє собою декілька складових.

Kick у музиці, особливо в контексті електронної музики та виробництва музики, є терміном, який зазвичай використовується для опису басового барабану або основного ударного звуку, що задає ритм і структуру композиції.

Основні характеристики **Kick**:

- низькочастотний звук: **Kick** зазвичай має низькі частоти, які добре відчуваються фізично і створюють основу ритму;
- чітка атака: це швидкий і різкий звук, який відразу привертає увагу слухача і задає пульс композиції;
- ритмічна функція: Kick часто використовується для підкреслення ритмічної структури музичного твору, граючи на сильних долях такту.

У студійній роботі створення і налаштування Kick є важливим аспектом, оскільки він повинен добре поєднуватися з іншими елементами композиції, такими як басові лінії, мелодії та інші ударні.

Snare (снейр) у музиці є іншим важливим елементом ударної установки. Це барабан, який відрізняється своїм характерним чітким і різким звуком.

Основні характеристики Snare:

- різкий звук: Snare має чіткий, різкий і дещо дзвінкий звук, що виділяється серед інших ударних інструментів;
- позиція в ритмі: Snare часто використовується для підкреслення слабких долей такту, таких як друга і четверта долі у чотиридольному такті, створюючи ритмічний контраст з бас-барабаном (Kick).

Hat (hi-hat, хай-хет) є важливим компонентом ударної установки та електронної музики. Він відіграє ключову роль у створенні ритмічної основи та надає композиції динаміки і руху.

Основні характеристики hi-hat:

- ритмічний елемент: hi-hat часто використовується для підтримки ритму, граючи на кожну восьму або шістнадцяту долю такту, що додає текстури і руху композиції;

- динаміка: hi-hat може бути зіграний з різною інтенсивністю, від м'якого тикання до гучного тріску, що дозволяє додавати динамічні зміни в музиці.

Враховуючи усі вищенаведені фактори, була визначено, що використання барабанної партії у проєкті буде логічним при застосуванні так званого барабанного секвенсора. Одним з таких є плагін Nexus (рис. 4.3).

Рисунок 4.3 – Вигляд плагіну Nexus і вже ввімкнений барабанний секвенсор «DR 90s HipHop Kit»

Кожна з MIDI-клавіш та нот цього секвенсора відповідає певному складнику барабанної установки і при застосуванні згенерованої плагіном «GENER» партії буде правильно накладатися і відтворювати необхідну композицію.

Звісно ж інтерпретувати MIDI-дані, створені плагіном можна і до інших інструментів DAW, виділяючи певні фрагменти партії і застосовуючи їх окремо.

Реалізація логіки генерування барабанної партії у методі drumMIDIFile() складається з кількох основних кроків.

Перший крок – це створення об'єкту MIDI файлу та послідовності MIDI повідомлень для барабанів так само, як це реалізовано для інших інструментів.

Для спрощення, барабанна партія використовує три основні ноти:

- kick (бас-барабан) – A3 (MIDI номер 48);
- snare (малий барабан) – G3 (MIDI номер 50);
- hi-hat (закритий хай-хет) – F#4 (MIDI номер 54.

Барабанна партія генерується на чотири такти, де кожний такт складається з чотирьох долей. Використовується внутрішній цикл для кожного такту та кожної долі:

```
for (int bar = 0; bar < 4; ++bar) {
   for (int beat = 0; beat < 4; ++beat) {
```

Була також обрана інша тривалість нот, оскільки барабанні звуки не мають настільки подовженого звучання як наприклад піаніно.

```
int duration = 121;
```

Була також обрана інша тривалість нот, оскільки барабанні звуки не мають настільки подовженого звучання як наприклад піаніно.

```
drumSequence.addEvent(juce::MidiMessage::noteOn(1, kickNote, (uint8_t)127), (bar * 4 + beat) * duration);
if (beat == 1 || beat == 3)
drumSequence.addEvent(juce::MidiMessage::noteOn(1, snareNote, (uint8_t)127), (bar * 4 + beat) * duration);
if (beat == 1 || beat == 3)
drumSequence.addEvent(juce::MidiMessage::noteOn(1, hatNote, (uint8_t)127), (bar * 4 + beat) * duration);
```

Тут кожна доля такту обробляється окремо, причому бас-барабан звучить на кожну долю, а малий барабан і хай-хет на другу і четверту долю.

Для кожної ноти також додаються події "нота вимкнена" після відповідної тривалості:

```
for (int i = 0; i < 16; ++i) {
int duration = 121;
drumSequence.addEvent(juce::MidiMessage::noteOff(1, kickNote), (i + 1) * duration);
   drumSequence.addEvent(juce::MidiMessage::noteOff(1, snareNote), (i + 1) *
duration);
drumSequence.addEvent(juce::MidiMessage::noteOff(1, hatNote), (i + 1) * duration);
```

Це гарантує, що кожна нота буде вимкнена після її програвання, що необхідно для правильного відтворення MIDI подій.

Згенеровані MIDI дані записуються у файл з унікальним іменем:

```
juce::File drumFileToSave("C:/Users/dadya/Documents/DRUMNEW.mid");
```

Цей блок коду записує MIDI дані у файл DRUMNEW.mid у вказану директорію.

Цей підхід дозволяє швидко створити основу барабанного треку, який може бути використаний для подальшого редагування та додавання складніших ритмічних елементів.

4.3 Тестування системи

Після завершення процесу розробки системи, було проведено загальне її тестування, яке дає можливість свідчити про успішне виконання проєкту.

Першим етапом перевірки було тестування інтерфейсу плагіна. Окрім того, що кнопки правильно реагують на натискання, мають затверджену позицію на екрані, вони реагують зміною кольору літери, як це показано на рисунку (рис. 4.4):

Рисунок 4.4 – Зміна кольору літери під час натискання на кнопку

Як і було заплановано, за бажанням можна змінити музичну тональність. Було реалізовано як мажорну, так і мінорну гамму:

Рисунок 4.5 – Тональність «До мажор»

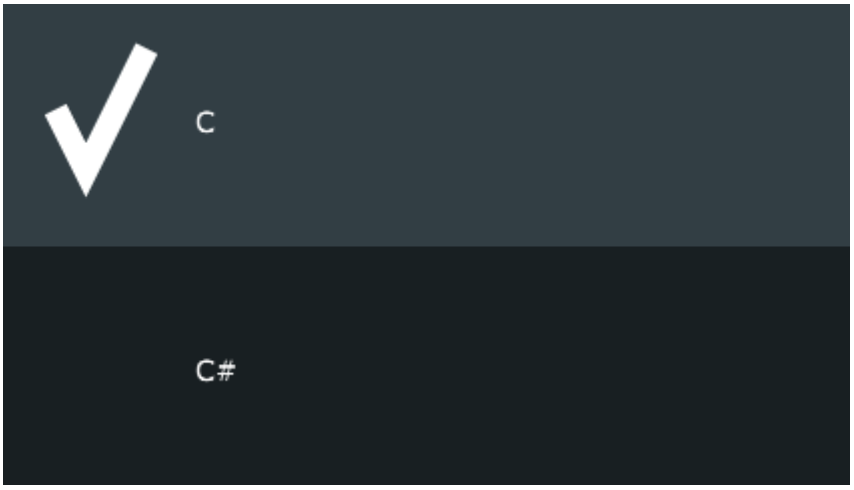

Рисунок 4.6 – Зміна музичної тональності

Рисунок 4.7 – Тональність змінена на «До-дієз мажор»

Після вибору музичного «ключа» користувач може згенерувати одну з трьх наведених партій: барабанну, басову чи партію основної мелодії. Кожна з них буде мати різну структуру.

В результаті таким чином буде виглядати основна мелодія (рис. 4.8) після переведення MIDI даних в DAW FL Studio.

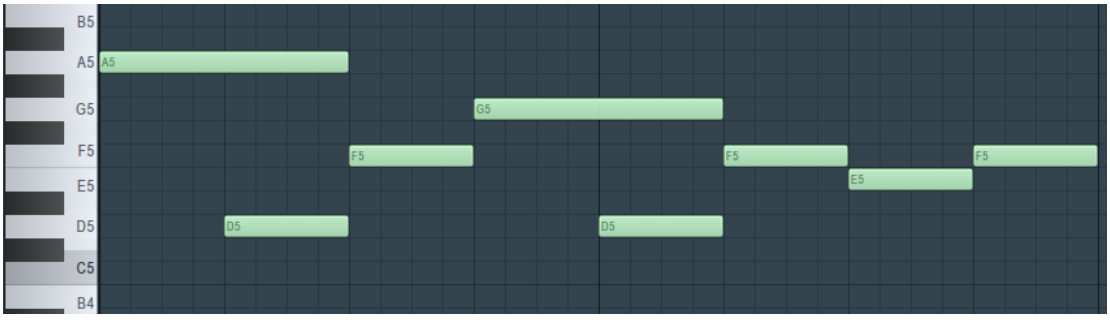

Рисунок 4.8 – Вигляд основної мелодії

Басова партія (рис. 4.9) буде мати наступний вигляд і складатися з трьох нот.

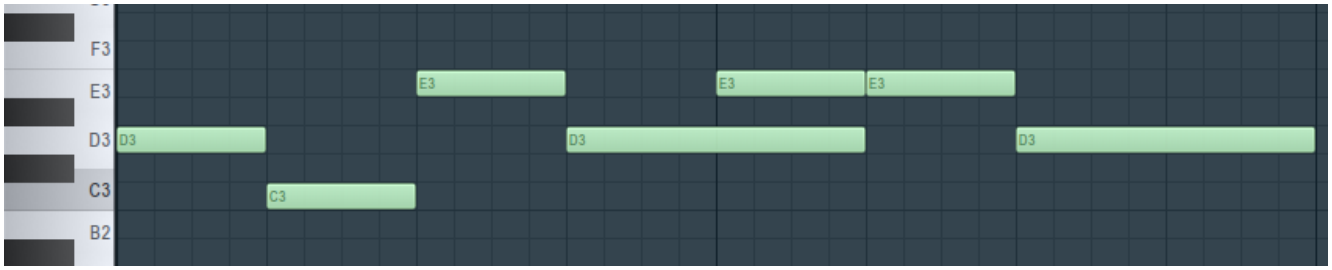

Рисунок 4.9 – Вигляд басової партії

Барабанна партія (рис. 4.10) буде сумісна з іншими плагінами, її ноти відповідають за три інструменти, якими є kick, snare і hat, де в цьому випадку E3 є hat, D3 – hat, а C3 відповідає за kick.

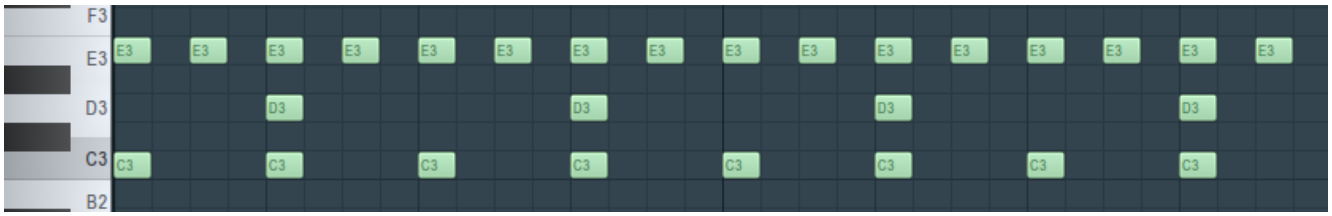

Рисунок 4.10 – Вигляд барабанної партії

Висновки до розділу 4

При створенні програмного коду для аудіо плагінів, одним із найважливіших аспектів є врахування специфічних характеристик та критеріїв звукової робочої станції, такої як FL Studio. Відмінною рисою FL Studio є її унікальна система побудови мелодій, що включає використання специфічної одиниці часу - "тіка". Роздільна здатність внутрішнього таймінгу FL Studio за замовчуванням складає 96 тиків на чвертьну ноту (PPQ), проте це значення може бути налаштоване для більш точної синхронізації.

Метод generateMIDIFile() у JUCE аудіо плагіні ілюструє процес створення випадкових мелодій у заданій тональності шляхом генерації MIDI файлів. Використання випадкових чисел для вибору нот з визначеного набору дозволяє

швидко створювати унікальні мелодії. Це підходить для музичних експериментів та автоматичного створення музики, забезпечуючи гнучкість і різноманіття варіантів.

Генерація басових партій у методі generateBassMIDIFile() подібна до логіки генерування мелодій у generateMIDIFile(), але з урахуванням специфіки басових партій. Зокрема, у сучасній музиці популярні прості басові партії, які зазвичай складаються з трьох-п'яти нот. Тому логіка генерування була адаптована, щоб відповідати цим вимогам, використовуючи меншу кількість нот та нижчу октаву.

Генерація барабанних партій вимагає іншого підходу, оскільки вони складаються з декількох елементів, таких як Kick, Snare та Hi-hat. Для реалізації генерування барабанних партій у проєкті було вирішено використовувати барабанний секвенсор, наприклад, плагін Nexus. Це дозволяє ефективно створювати складні барабанні партії, які доповнюють інші елементи композиції, забезпечуючи цілісність та ритмічну структуру музичного твору.

Таким чином, розробка програмного коду для генерації музичних композицій у FL Studio потребує врахування специфічних особливостей цієї звукової робочої станції та адаптації методів генерації для різних типів музичних партій. Це дозволяє створювати ефективні інструменти для музичних експериментів і автоматизованого створення музики, що відповідають потребам сучасної музичної індустрії.

ВИСНОВКИ

З кожним роком кількість починаючих музикантів збільшується за рахунок зниження рівню доступності входу в сферу музичної індустрії.

Актуальність створеного проєкту «GENER» полягає у тому, що генерування музичних мелодій завдяки такій системі дасть змогу початківцям розібратися та поліпшити свій музичні навички.

Метою проєкту було спрощення процесу створення музичної композиції для різних типів користувачів цифрових звукових робочих станцій.

Для досягнення поставленої мети було вирішено такі завдання:

- проаналізовано сучасний стан задачі генерування музичних мелодій;
- проаналізовано існуючі аналоги систем;
- обрано необхідний тип плагіну;
- розроблено систему генерування музичних мелодій;
- розроблено користувацький графічний інтерфейс;
- проведено тестування застосунку;

Створено зрозумілий та лаконічний графічний інтерфейс, який є оптимальним для будь-якого класу користувача – є він початківцем чи досвідченим музикантом.

Розроблено інтелектуальну систему генерування музичних мелодій. «GENER» вміє генерувати три типи мелодій: басова партія, барабанна партія та основна мелодія для буль якого виду музичної композиції.

Було проаналізовано і досліджено методи безпеки при роботі з середою розробки програмного забезпечення, описано та продемонстровано план робочого приміщення і визначено техніку запобігання фізичного і психологічного тиску при роботі з програмуванням аудіо-додатків.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Створення аудіоплагінів. 2014. URL: https://habr.com/ru/articles/224911/.
2. Boulanger R. The Audio Programming Book: навч. посіб. Вид. 1-ше. Массачусетс, 2010. 889 с.
3. Cipriani A. Electronic Music and Sound Design - Theory and Practice with Max 8 - Volume 1: навч. посіб. Вид. 4-тє. Лондон, 2019. 592 с.
4. Cipriani A. Electronic Music and Sound Design - Theory and Practice with Max 8 - Volume 2: навч. посіб. Вид. 3-тє. Лондон, 2019. 748 с.
5. Creasey D. Audio Processes: Musical Analysis, Modification, Synthesis, and Control: навч. посіб. Вид. 1-ше. Лондон, 2016. 752 с.
6. Farnell A. Designing Sound: навч. посіб. Вид. 1-ше. Массачусетс, 2010. 668 с.
7. Finke M. Making Audio Plugins. 2015. URL: https://www.martin-finke.de/articles/audio-plugins-001-introduction/.
8. FL Studio. Overview. 2024. URL: https://www.image-line.com/fl-studio/.
9. Hewitt M. Music Theory for Computer Musicians: навч. посіб. Вид. 1-ше. Нью-Йорк, 2008. 336 с.
10. Hillerson T. Programming Sound with Pure Data: Make Your Apps Come Alive with Dynamic Audio: Вид. 1-ше. Лондон, 2014. 196 с.
11. Kapur A. Programming for Musicians and Digital Artists: Creating music with ChucK: навч. посіб. Вид. 1-ше. Нью-Йорк, 2015. 344 с.
12. Langford S. Digital Audio Editing: Correcting and Enhancing Audio in Pro Too : навч. посіб. Вид. 1-ше. Лондон, 2017. 336 с.
13. Lazzarini V. Computer Music Instruments: Foundations, Design and Development: навч. посіб. Вид. 1-ше. Лондон, 2017. 381 с.
14. Loy G. Musimathics, Volume 2: The Mathematical Foundations of Music: Вид. 1-ше. Массачусетс, 2011. 584 с.

15. Manaris B. Making Music with Computers (Chapman & Hall/CRC Textbooks in Computing): навч. посіб. Вид. 1-ше. Лондон, 2024. 502 с.

16. MUSIA. What is the MUSIA Plugin. 2024. URL: https://musia.ai/support.

17. Phil Speiser. THE_INSTRUMENT. 2024. URL: https://philspeiser.com/product/the_instrument/.

18. Pirkle W. Designing Audio Effect Plugins in C++: For AAX, AU, and VST3 with DSP Theory : навч. посіб. Вид. 2-ге, переробл. і допов. Лондон, 2019. 656 с.

19. Pirkle W. Designing Software Synthesizer Plugins in C++: With Audio DSP : навч. посіб. Вид. 1-ше. Лондон, 2022. 276 с.

20. Raw Material Software Limited. JUCE. 2024. URL: https://juce.com/.

21. Ruggles D. Getting Started with Audio Plugin Development and JUCE: A Beginner's Guide. 2024. URL: https://www.linkedin.com/pulse/getting-started-audio-plugin-development-juce-guide-danny-ruggles-ukx8e/.

22. Steinberg Media Technologies GmbH. Знайомство з VST 3 SDK. 2024. URL: https://steinbergmedia.github.io/vst3_doc/vstsdk/index.html.

23. Steinberg Media Technologies GmbH. Портал розробника VST 3. 2024: URL: https://steinbergmedia.github.io/vst3_dev_portal/pages/index.html.

24. Stoic. P. Vst/Vsti Plugin Manager Book: Manage Your Favorite Vst Plugins And Virtual Instruments: навч. посіб. Вид. 1-ше. Лондон, 2022. 110 с.

25. Wakefield G. Generating Sound & Organizing Time: Thinking with gen~ Book 1: навч. посіб. Вид. 1-ше. Сан-Франциско, 2022. 381 с.

ДОДАТОК А

Код генерації музичних мелодій для різних інструментальних складових

```
#include "PluginProcessor.h"
#include "PluginEditor.h"
#include <fstream>
#include <cstring>
#include <Windows.h>
#include <JuceHeader.h>

GenerAudioProcessorEditor::GenerAudioProcessorEditor(GenerAudioProcessor& p)
    : AudioProcessorEditor(&p), audioProcessor(p)
{
    setSize (1024, 559);
    generateButton.setBounds(572, 70, 420, 420);
    bassButton.setBounds(348, 142, 210, 210);
    drumButton.setBounds(205, 193, 110, 110);
    keySelector.addItemList(keyNames, 1);
    keySelector.setSelectedId(1);
    saveToMidiButton.setButtonText("Save to MIDI");
    saveToMidiButton.setBounds(150, 200, 200, 50);
    juce::Image normalImage =
juce::ImageFileFormat::loadFrom(juce::File("D:/CHNU/Accembler/G_E_N_E_R_button.png"));
    juce::Image activeImage =
juce::ImageFileFormat::loadFrom(juce::File("D:/CHNU/Accembler/G_E_N_E_R_button_activated.png"));
    juce::Image normalbassImage =
juce::ImageFileFormat::loadFrom(juce::File("D:/CHNU/Accembler/G_E_N_E_R_bass_button.png"));
    juce::Image activebassImage =
juce::ImageFileFormat::loadFrom(juce::File("D:/CHNU/Accembler/G_E_N_E_R_bass_button_activated.png"));
    juce::Image normaldrumImage = juce::ImageFileFormat::loadFrom(juce::File("D:/G_E_N_E_R__drum_button.png"));
    juce::Image activedrumImage =
juce::ImageFileFormat::loadFrom(juce::File("D:/G_E_N_E_R__drum_button_activated.png"));
    generateButton.setImages(false, true, true,
        normalImage, 1.0f, juce::Colours::transparentBlack,
        activeImage, 1.0f, juce::Colours::transparentBlack,
        activeImage, 1.0f, juce::Colours::transparentBlack);
    bassButton.setImages(false, true, true,
        normalbassImage, 1.0f, juce::Colours::transparentBlack,
        activebassImage, 1.0f, juce::Colours::transparentBlack,
```

```
        activebassImage, 1.0f, juce::Colours::transparentBlack);
    drumButton.setImages(false, true, true,
        normaldrumImage, 1.0f, juce::Colours::transparentBlack,
        activedrumImage, 1.0f, juce::Colours::transparentBlack,
        activedrumImage, 1.0f, juce::Colours::transparentBlack);
    addAndMakeVisible(generateButton);
    addAndMakeVisible(bassButton);
    addAndMakeVisible(keySelector);
    addAndMakeVisible(drumButton);
    generateButton.onClick = [this] {
        audioProcessor.generateMIDIFile();
    };

    bassButton.onClick = [this] {
        audioProcessor.generateBassMIDIFile();
    };

    saveToMidiButton.onClick = [this] {

    };

    keySelector.onChange = [this] {
        selectedKeyIndex = keySelector.getSelectedItemIndex();
        repaint();
    };

    drumButton.onClick = [this] {
        audioProcessor.drumMIDIFile();
    };

    drumButton.setVisible(true);
}

GenerAudioProcessorEditor::~GenerAudioProcessorEditor()
{
}
void GenerAudioProcessorEditor::paint (juce::Graphics& g)
{
    juce::Image backgroundImage =
juce::ImageFileFormat::loadFrom(juce::File("D:/CHNU/Accembler/G_E_N_E_R.png"));
```

```
    juce::Image normalImage =
juce::ImageFileFormat::loadFrom(juce::File("D:/CHNU/Accembler/G_E_N_E_R_button.png"));
    juce::Image selectorImage =
juce::ImageFileFormat::loadFrom(juce::File("D:/CHNU/Accembler/G_E_N_E_R_selector.png"));
    g.drawImage(backgroundImage, getLocalBounds().toFloat());
    int x = 155;
    int y = 380;
    int width = 430;
    int height = 126;
    g.drawImage(selectorImage, x, y, width, height, 0, 0, selectorImage.getWidth(), selectorImage.getHeight());
    keySelector.setAlpha(0);
    keySelector.setBounds(155, 380, 430, 126);
    g.setColour (juce::Colours::white);
    g.setFont (15.0f);
    if (selectedKeyIndex >= 0 && selectedKeyIndex < keyNames.size()) {
        juce::String imageName = "D:/CHNU/Accembler/" + keyNames[selectedKeyIndex] + ".png";
        juce::Image keyImage = juce::ImageFileFormat::loadFrom(juce::File(imageName));
        g.drawImage(keyImage, 155, 380, 430, 126, 0, 0, keyImage.getWidth(), keyImage.getHeight());
    }
}
void GenerAudioProcessorEditor::resized()
{
    keySelector.setBounds(10, 10, 200, 50);
}
```

ДОДАТОК Б

Код графічного користувацького інтерфейсу

```
#include "PluginProcessor.h"
#include "PluginEditor.h"
#include <fstream>
#include <cstring>
#include <Windows.h>// Äëÿ èñïîëüçîâàíèÿ WinAPI
#include <JuceHeader.h>

GenerAudioProcessorEditor::GenerAudioProcessorEditor(GenerAudioProcessor& p)
    : AudioProcessorEditor(&p), audioProcessor(p)
{
    setSize (1024, 559);
    generateButton.setBounds(572, 70, 420, 420);
    bassButton.setBounds(348, 142, 210, 210);
    drumButton.setBounds(205, 193, 110, 110);
    keySelector.addItemList(keyNames, 1);
    keySelector.setSelectedId(1);
    saveToMidiButton.setButtonText("Save to MIDI");
    saveToMidiButton.setBounds(150, 200, 200, 50);
    juce::Image normalImage =
juce::ImageFileFormat::loadFrom(juce::File("D:/CHNU/Accembler/G_E_N_E_R_button.png"));
    juce::Image activeImage =
juce::ImageFileFormat::loadFrom(juce::File("D:/CHNU/Accembler/G_E_N_E_R_button_activated.png"));
    juce::Image normalbassImage =
juce::ImageFileFormat::loadFrom(juce::File("D:/CHNU/Accembler/G_E_N_E_R_bass_button.png"));
    juce::Image activebassImage =
juce::ImageFileFormat::loadFrom(juce::File("D:/CHNU/Accembler/G_E_N_E_R_bass_button_activated.png"));
    juce::Image normaldrumImage = juce::ImageFileFormat::loadFrom(juce::File("D:/G_E_N_E_R__drum_button.png"));
    juce::Image activedrumImage =
juce::ImageFileFormat::loadFrom(juce::File("D:/G_E_N_E_R__drum_button_activated.png"));
    generateButton.setImages(false, true, true,
        normalImage, 1.0f, juce::Colours::transparentBlack,
        activeImage, 1.0f, juce::Colours::transparentBlack,
        activeImage, 1.0f, juce::Colours::transparentBlack);
    bassButton.setImages(false, true, true,
        normalbassImage, 1.0f, juce::Colours::transparentBlack,
        activebassImage, 1.0f, juce::Colours::transparentBlack,
```

```
        activebassImage, 1.0f, juce::Colours::transparentBlack);
    drumButton.setImages(false, true, true,
        normaldrumImage, 1.0f, juce::Colours::transparentBlack,
        activedrumImage, 1.0f, juce::Colours::transparentBlack,
        activedrumImage, 1.0f, juce::Colours::transparentBlack);
    addAndMakeVisible(generateButton);
    addAndMakeVisible(bassButton);
    addAndMakeVisible(keySelector);
    addAndMakeVisible(drumButton);
    generateButton.onClick = [this] {
        audioProcessor.generateMIDIFile();
    };

    bassButton.onClick = [this] {
        audioProcessor.generateBassMIDIFile();
    };

    saveToMidiButton.onClick = [this] {

    };

    keySelector.onChange = [this] {
        selectedKeyIndex = keySelector.getSelectedItemIndex();
        repaint();
    };

    drumButton.onClick = [this] {
        audioProcessor.drumMIDIFile();
    };

    drumButton.setVisible(true);
}

GenerAudioProcessorEditor::~GenerAudioProcessorEditor()
{
}
void GenerAudioProcessorEditor::paint (juce::Graphics& g)
{
    juce::Image backgroundImage =
juce::ImageFileFormat::loadFrom(juce::File("D:/CHNU/Accembler/G_E_N_E_R.png"));
```

```
    juce::Image normalImage =
juce::ImageFileFormat::loadFrom(juce::File("D:/CHNU/Accembler/G_E_N_E_R_button.png"));
    juce::Image selectorImage =
juce::ImageFileFormat::loadFrom(juce::File("D:/CHNU/Accembler/G_E_N_E_R_selector.png"));
    g.drawImage(backgroundImage, getLocalBounds().toFloat());
    int x = 155;
    int y = 380;
    int width = 430;
    int height = 126;
    g.drawImage(selectorImage, x, y, width, height, 0, 0, selectorImage.getWidth(), selectorImage.getHeight());
    keySelector.setAlpha(0);
    keySelector.setBounds(155, 380, 430, 126);
    g.setColour (juce::Colours::white);
    g.setFont (15.0f);
    if (selectedKeyIndex >= 0 && selectedKeyIndex < keyNames.size()) {
        juce::String imageName = "D:/CHNU/Accembler/" + keyNames[selectedKeyIndex] + ".png";
        juce::Image keyImage = juce::ImageFileFormat::loadFrom(juce::File(imageName));
        g.drawImage(keyImage, 155, 380, 430, 126, 0, 0, keyImage.getWidth(), keyImage.getHeight());
    }
}
void GenerAudioProcessorEditor::resized()
{
    keySelector.setBounds(10, 10, 200, 50);
}
```