Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)
Кафедра програмної інженерії
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

бакалавр
(назва освітнього ступеня)

на тему: Розробка користувацького застосунку для зовнішнього MIDI синтезатора з використанням мови програмування C#

Виконав: студент IV курсу, групи СП-41
спеціальності 121 Інженерія програмного забезпечення
(шифр і назва спеціальності)

(підпис) Долінський І. С. (прізвище та ініціали)

Керівник (підпис) Стоянов Ю. М. (прізвище та ініціали)

Нормоконтроль (підпис) Стоянов Ю. М (прізвище та ініціали)

Завідувач кафедри (підпис) Петрик М. Р. (прізвище та ініціали)

Рецензент (підпис) (прізвище та ініціали)

Тернопіль
2024

Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет Комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)

Кафедра Програмної інженерії
(повна назва кафедри)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Петрик М. Р.
(підпис) (прізвище та ініціали)
«__» _____ 2024 р.

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня бакалавр
(назва освітнього ступеня)

за спеціальністю 121 Інженерія програмного забезпечення
(шифр і назва спеціальності)

студенту Долінському Івану Степановичу
(прізвище, ім'я, по батькові)

1. Тема роботи Розробка користувацького застосунку для зовнішнього MIDI синтезатора з використанням мови програмування C#

Керівник роботи к. т. н. ст. викладач Стоянов Ю. М.
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом ректора від «*15*» *квітня* 2024 року № *4/7-341*

2. Термін подання студентом завершеної роботи

3. Вихідні дані до роботи Предметна область, технічне завдання, вимоги та специфікація, програмне рішення

4. Зміст роботи (перелік питань, які потрібно розробити): Вступ. Розділ 1. Аналіз предметної області та формування вимог. Розділ 2. Проєктування програмного рішення. Розділ 3. Тестування системи. Розділ 4.Безпека життєдіяльності та основи охорони праці. Висновок. Список використаних джерел. Додатки

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, слайдів): Ілюстративні зображення, інформативні зображення для доповнення тексту, діаграми, знімки екрану з проробленою роботою.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Безпека життєдіяльності, основи охорони праці	Лазарюк В. В., доц. каф. МП		

7. Дата видачі завдання 14 лютого 2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1.	Ознайомлення з завданням до кваліфікаційної роботи	14.02.2024	Виконано
2.	Підбір джерел по темі кваліфікаційної роботи	15.02.2024-25.02.2024	Виконано
3.	Опрацювання джерел по темі кваліфікаційної роботи	26.02.2024-01.03.2024	Виконано
4.	Аналіз та підготовка технічної специфікації	04.03.2024-08.03.2024	Виконано
5.	Розробка архітектури програмного рішення	11.03.2024-15.03.2024	Виконано
6.	Аналіз сутностей програмного рішення	28.03.2024-02.04.2024	Виконано
7.	Розробка архітектури програмного рішення	03.04.2024-07.04.2024	Виконано
8.	Розробка користувацького інтерфейсу програмного рішення	09.04.2024-18.04.2024	Виконано
9.	Виконання звіту по проробленій роботі	09.04.2024-08.05.2024	Виконано
10.	Оформлення кваліфікаційної роботи	09.04.2024-11.05.2024	Виконано
11.	Виконання завдання до підрозділу «Безпека життєдіяльності»	06.06.2024-06.06.2024	Виконано
12.	Виконання завдання до підрозділу «Основи охорони праці»	06.06.2024-06.06.2024	Виконано
13.	Нормоконтроль		
14.	Перевірка на плагіат		
15.	Попередній захист кваліфікаційної роботи		
16.	Захист кваліфікаційної роботи		

Студент _____ Долінський І. С.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Стоянов Ю. М.
(підпис) (прізвище та ініціали)

РЕФЕРАТ

Розробка користувацького застосунку для зовнішнього MIDI синтезатора з використанням мови програмування C# // Кваліфікаційна робота освітнього рівня «Бакалавр» // Долінський Іван Степанович // Тернопільський національний технічний університет імені Івана Пулюя, факультет комп'ютерно інформаційних систем і програмної інженерії, кафедра програмної інженерії, група СП-41 // Тернопіль, 2024 // С. – 86, рис. – 25, табл. – 1, кресл. – 0, додат. – 2, бібліогр. – 12.

Ключові слова: індивідуальне програмне рішення, синтез музики, MIDI контролери, обробка MIDI, відтворення звуків, C#, WPF, Digital Audio Workstation, Sanford Multimedia, Newtonsoft, SOLID, GIT, Microsoft Visual Studio.

Кваліфікаційна робота досліджує процеси аналізу, проєктування, розробку та тестування програмного рішення для керування зовнішнім MIDI синтезатором за допомогою мови програмування C# та технологій WPF. В роботі розглянуто сучасні інструменти і підходи, що використовуються для оптимізації музичного виробництва.

У першому розділі проведено аналіз предметної області та визначено вимоги до розробки програмного забезпечення. Цей розділ надає огляд актуальності проблематики, встановлює мету та завдання роботи, зокрема розробку користувацького застосунку для керування MIDI синтезатором.

В другому розділі кваліфікаційної роботи описано процеси проектування та реалізації програмного рішення. У ньому детально описано процеси розробки інтерфейсу, обробки MIDI повідомлень, створення та управління пресетами, а також реалізацію візуальних інструментів для редагування музичних ефектів.

В третьому розділі кваліфікаційної роботи описано тестування та верифікацію програмного рішення. Розглянуто інструменти і методи тестування графічного інтерфейсу користувача, перевірки функціональності обробки MIDI сигналів та забезпечення стабільної роботи програми.

ANNOTATION

Development of a user application for an external MIDI synthesizer using the programming language C # // Qualification work of the educational level "Bachelor " // Dolinsky Ivan Stepanovich // Ternopil National Technical University named after Ivan Puluy, Faculty of Computer Information Systems and Software Engineering, Department of Software Engineering, group SP-41 // Ternopil, 2024 // P. - 86, Fig. - 25, table - 1, drawing - 0, addate. - 2, bibliography. - 12.

Keywords: individual software solution, music synthesis, MIDI controllers, MIDI processing, sound reproduction, C #, WPF, Digital Audio Workstation, Sanford Multimedia, Newtonsoft, SOLID, GIT, Microsoft Visual Studio io.

The qualifying work explores the processes of analyzing, designing, developing and testing a software solution for controlling an external MIDI synthesizer using the C # programming language and WPF technologies. The paper considers modern instruments and approaches used to optimize music production.

The first section analyzes the subject area and defines the requirements for software development. This section provides an overview of the relevance of the problem, sets the purpose and objectives of the work, in particular the development of a custom application for controlling a MIDI synthesizer.

The second section of the qualification work describes the processes of designing and implementing a software solution. It details the processes of developing an interface, processing MIDI messages, creating and managing presets, and implementing visual tools for editing musical effects.

The third section of the qualification work describes the testing and verification of the software solution. The tools and methods for testing the graphical user interface, checking the functionality of processing MIDI signals and ensuring stable operation of the program are considered.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

DAW (Digital Audio Workstation) – програмне забезпечення для запису, редагування та відтворення аудіо.

MIDI (Musical Instrument Digital Interface) – цифровий інтерфейс для передачі музичної інформації між електронними інструментами, комп'ютерами та іншими пристроями [6].

WPF (Windows Presentation Foundation) – технологія від Microsoft для створення настільних застосунків з багатими візуальними ефектами та інтерактивним користувацьким інтерфейсом.SOLID – принципи об'єктно-орієнтованого дизайну програмного забезпечення [2].

GUI (Graphical User Interface) – графічний інтерфейс користувача.

SOLID – принципи об’єктно-орієнтованого програмування, які забезпечують високу якість коду та підтримку легкої масштабованості й обслуговування проєкту [7].

IoT (Internet of Things) – концепція мережі фізичних об'єктів, оснащених технологіями для взаємодії один з одним або з зовнішнім середовищем через інтернет.

GUI (Graphical User Interface) – графічний інтерфейс користувача, що дозволяє взаємодіяти з програмним забезпеченням за допомогою графічних елементів.

MVVM – Model-View-ViewModel, шаблон проєктування архітектури додатку. В його основі відділення коду призначеного для користувача інтерфейсу (UI) від решти, і створення слабкого зв'язку для синхронізації[8].

Data Binding – прив'язка даних, метод в програмуванні, що дозволяє автоматично синхронізувати дані між моделлю та представленням.

Команди – спосіб обробки дій користувача в архітектурному шаблоні MVVM, що дозволяє централізовано керувати логікою взаємодії.

Сутність – абстрактне уявлення об'єкта предметної області, що включає його властивості та поведінку.

Бізнес логіка – частина програмного забезпечення, що реалізує правила та процеси предметної області.

Секвенція – послідовність нот або звуків, створена в музичному редакторі.

Хорус – музичний ефект, що створює відчуття присутності кількох виконавців.

Модуляція – зміна певного параметра звуку (наприклад, частоти чи амплітуди) для досягнення бажаного ефекту.

Дилей – ефект затримки звуку, що створює ехо або реверберацію.

Система – сукупність апаратних та програмних засобів, призначених для виконання певних функцій.

Користувач – особа, яка взаємодіє з програмним забезпеченням для виконання певних завдань.

Сигнал – електричний імпульс, що передає інформацію між елементами системи та керується певним протоколом.

Тест-кейс – документ, що містить набір умов, кроків та очікуваних результатів для перевірки певної функціональності програмного забезпечення [9].

ЗМІСТ

ВСТУП

В епоху, визначену швидкими технологічними досягненнями, необхідність індивідуальних програмних рішень має першорядне значення, особливо в спеціалізованих областях, таких як написання та синтез музики.

Ця робота спрямована на задовільнення цієї потреби, описуючи розробку користувацької програми, спеціально спроектованої для роботи із зовнішніми музичними контролерами MIDI, використовуючи універсальні можливості мови програмування C # та її фреймворку WPF.

За своєю суттю, робота має на меті забезпечити інтуїтивно зрозумілий і ефективний інструмент, який дозволяє користувачам легко взаємодіяти зі своїми MIDI синтезаторами, тим самим підвищуючи свої творчі можливості у написанні музики. Надаючи користувачам інтерфейс, який є одночасно зручним і багатофункціональним, додаток прагне оптимізувати процес синтезу і розблокувати нові сфери творчості для музикантів.

Прийнятий підхід є систематичним та ретельним, що охоплює різні етапи, включаючи детальний аналіз, моделювання системи, розробка дизайну, процеси налагодження та протоколи тестування. Приймаючи цей цілісний підхід, робота спрямована не тільки на розробку програми, але і на те, щоби забезпечити нюанси розуміння всього життєвого циклу розробки.

Використовуючи сучасні інструменти та методології, ця робота намагається вникнути в тонкощі розробки такого спеціалізованого програмного забезпечення, від його початкової концептуалізації до його остаточної реалізації.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ФОРМУВАННЯ ВИМОГ

1.1 Формування вимог та постановка задачі

У сучасному світі музичного виробництва, ефективне використання MIDI синтезаторів є важливим аспектом творчого процесу для багатьох музикантів і продюсерів. Враховуючи постійний розвиток технологій, важливо мати інструменти, які дозволяють зручно і ефективно керувати зовнішніми MIDI пристроями. Такі інструменти допомагають музикантам досягати нових висот у своїй творчості, спрощуючи процеси створення та редагування музики.

Проте, наявні методи керування зовнішніми MIDI синтезаторами часто є складними і неінтуїтивними, що створює додаткові бар'єри для користувачів. Як показано на рисунку 1.1, традиційні способи взаємодії з MIDI пристроями можуть бути обтяжливими та вимагати значних технічних знань.

Рисунок 1.1 – Зразок традиційної програми для обробки MIDI сигналів

Це викликає необхідність у створенні простих та ефективних інструментів, які зможуть спростити та оптимізувати роботу з MIDI синтезаторами.

Таким чином, основною метою і завданням даної кваліфікаційної роботи є розробка користувацького застосунку для керування зовнішнім MIDI синтезатором з використанням мови програмування C#. Запропоноване програмне рішення має забезпечити надійний та функціональний інтерфейс для взаємодії з MIDI пристроями, вирішуючи наступні задачі користувача:

1. Відправка та отримання MIDI повідомлень між комп'ютером і зовнішнім синтезатором;
2. Обробка MIDI повідомлень для відтворення звуків;
3. Можливість вибору наборів звуків із бази інструментів;
4. Створення, збереження та управління пресетами і налаштуваннями синтезатора;
5. Реалізація інтерфейсу для візуального редагування музичних ефектів;
6. Можливість створення послідовностей та патернів звуків;
7. Функціонал відтворення файлів формату MIDI;
8. Забезпечення стабільної роботи з різними моделями MIDI синтезаторів;
9. Забезпечення безпечного доступу до налаштувань та даних користувача;
10. Ефективна логіка для обробки MIDI повідомлень та забезпечення їх точності;
11. Зручний та інтуїтивно зрозумілий інтерфейс програми.

Фактична ціль цієї роботи полягає в розробці програмного рішення для керування зовнішніми MIDI синтезаторами з використанням сучасних технологій програмування. Це дозволить створити функціональне програмне забезпечення зі структурованим, доступним та стабільним кодом.

Додатковою ціллю виконання цієї роботи є дослідження та розгляд процесу розробки власного готового програмного рішення, для спеціалізованих областей як написання та продюсування музики.

1.2 Аналіз предметної області

Предметною областю даної кваліфікаційної роботи є сфера музичного виробництва, а точніше процеси керування зовнішніми MIDI синтезаторами.

Сфера музичного виробництва має довгу історію, що починається з перших музичних інструментів і продовжується до сучасних електронних синтезаторів. MIDI (Musical Instrument Digital Interface) – це протокол, який дозволяє музичним інструментам та комп'ютерам спілкуватися між собою [6]. Він є невід'ємною частиною сучасного музичного виробництва, забезпечуючи зв'язок між різними пристроями, такими як клавіатури, синтезатори та комп'ютери. На рисунку 1.2 зображено схему зв'язку між різними MIDI пристроями.

Рисунок 1.2 – Вигляд підключення зовнішніх пристроїв

MIDI технології є одним із ключових компонентів сучасного музичного виробництва. Вони надають музикантам і продюсерам інструменти для створення,

редагування та відтворення музики з високою точністю і гнучкістю. Ці технології дозволяють керувати звуками різних інструментів, зберігати і відтворювати музичні композиції, а також інтегрувати різноманітні звукові ефекти.

MIDI забезпечує стандартний протокол для передачі музичної інформації між електронними музичними інструментами, комп'ютерами та іншими пристроями. Це дозволяє музикантам створювати складні музичні композиції, використовуючи різні інструменти, та з легкістю інтегрувати їх у цифрове середовище. З появою MIDI музиканти отримали можливість значно розширити свої творчі можливості, поєднуючи різні звукові інструменти і технології у єдину екосистему.

Основні переваги MIDI технологій:

Точність і Гнучкість: MIDI дозволяє передавати детальні інструкції щодо музичних параметрів, таких як ноти, швидкість, тривалість і динаміка, що забезпечує високий рівень точності у відтворенні музичних творів.

Інтеграція Інструментів: MIDI стандарти забезпечують сумісність між різними музичними пристроями, що дозволяє музикантам легко комбінувати звуки різних інструментів у своїх композиціях.

Збереження і Відтворення: MIDI дані можуть бути збережені у вигляді файлів, що дозволяє зберігати музичні твори для подальшого редагування та відтворення.

Ефективність і Простота: MIDI файли мають невеликий розмір у порівнянні з аудіофайлами, що робить їх зручними для зберігання та передачі.

Виклики у використанні MIDI технологій:

Складність налаштувань: Налаштування MIDI синтезаторів та інших пристроїв може бути складним процесом, що вимагає значних технічних знань і досвіду. Це може стати бар'єром для новачків або менш технічно підкованих музикантів.

Потреба у спеціальних знаннях: Використання MIDI технологій вимагає розуміння музичної теорії та технічних аспектів MIDI протоколу, що може бути складним для деяких користувачів.

Відсутність зручного інтерфейсу: Багато існуючих інтерфейсів для роботи з MIDI синтезаторами є неінтуїтивними і складними у використанні. Це може значно ускладнити процес створення та редагування музики.

Отже, незважаючи на численні переваги, існують значні виклики, які необхідно подолати для ефективного використання MIDI технологій. Розробка користувацького застосунку для зовнішнього MIDI синтезатора з використанням мови програмування C# спрямована на вирішення цих викликів, забезпечуючи інтуїтивно зрозумілий інтерфейс та спрощення процесу налаштування і керування MIDI пристроями.

1.3 Аналіз ринку конкурентних рішень

На ринку програмного забезпечення для роботи з MIDI існують численні конкурентні рішення, що надають користувачам різноманітні можливості для створення та редагування музики. Одним з найпопулярніших рішень є FL Studio.

FL Studio [4] (раніше відомий як FruityLoops) від Image-Line[3] є одним з провідних DAW (Digital Audio Workstation) на ринку.

Рисунок 1.3 – Логотип розробника додатку

Це комплексний інструмент для музичної творчості, який надає широкий набір функцій і інструментів для роботи з музикою. FL Studio відомий своєю простотою використання, багатошаровою підтримкою MIDI, величезною бібліотекою звуків та плагінів, а також можливістю інтеграції з різними аудіо та MIDI пристроями.

FL Studio надає користувачам інтуїтивний і потужний інтерфейс для роботи з MIDI данними. Воно включає в себе різноманітні інструменти та ефекти, які дозволяють музикантам та продюсерам створювати складні музичні композиції з великою кількістю шарів та звукових доріжок.

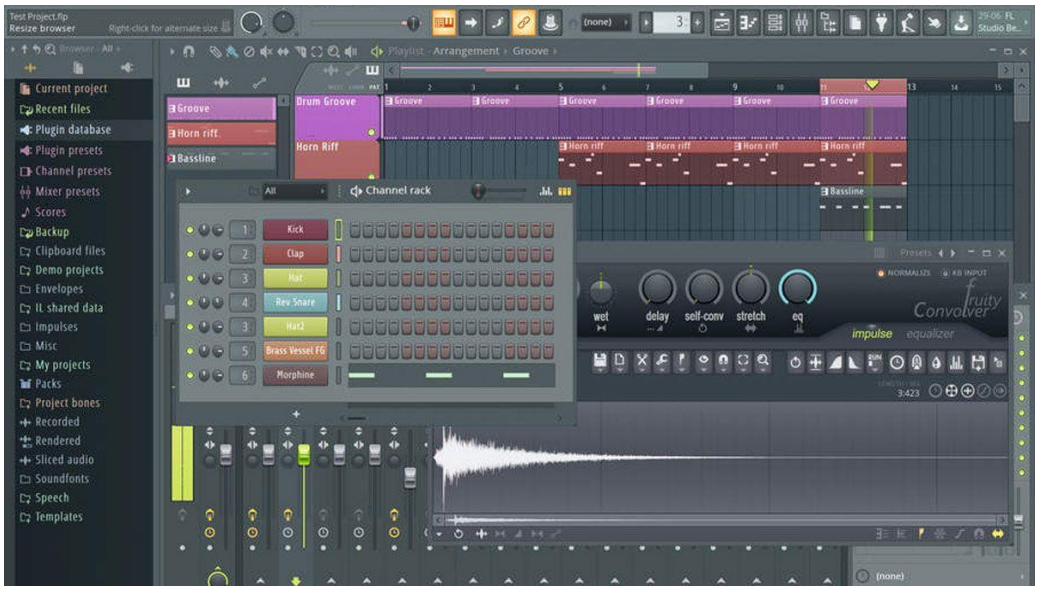

Рисунок 1.4 – Інтерфейс конкурентного додатку

FL Studio відрізняється від інших програм своєю широкою функціональністю, гнучкістю та легкістю використання. Інтерфейс програми надає користувачам можливість швидко та ефективно редагувати та аранжувати MIDI дані, а також використовувати різноманітні інструменти та ефекти для створення унікальних музичних композицій. Крім того, FL Studio підтримує використання різних плагінів, що розширює його можливості і дозволяє користувачам налаштовувати свій робочий процес.

Проте, існують деякі виклики у використанні FL Studio. По-перше, програма може бути дорогим продуктом, що може стати обмежуючим фактором для деяких користувачів. Крім того, для новачків FL Studio може здаватися складним та перенасиченим функціями, що може викликати неефективне використання програми.

Отже, FL Studio визнаний світовим лідером у галузі програмного забезпечення для роботи з MIDI технологіями, проте він не єдиний гравець на цьому ринку.

1.4 Опис обраних технологій

C# – сучасна мова програмування з великою базою користувачів, яка використовує систему безпечної типізації та підтримує об’єктно-орієнтований підхід програмування. C# є зручним засобом для створення цього проєкту, адже забезпечує необхідну інтеграцію з базами даних, реалізацію інтерфейсу та широкий спектр технологій. Також важливою перевагою C# є можливість роботи над проєктом у Microsoft Visual Studio, зручному та функціональному середовищі.

Головні переваги середовища Microsoft Visual Studio над іншими це:

- Інтеграція з графічним інтерфейсом редактора WPF.
- Глибокий аналіз коду для мов .NET, зручні інструменти редагування.
- Інтеграція з базами даних, підтримка виконання SQL запитів.
- Інтеграція менеджера пакетів NuGet, розширене управління залежностями проєкту.
- Комфортний інтерфейс усіх основних функціональних можливостей.

Мова програмування має величезну громаду прихильників і займає перші позиції в списку найбільш популярних інструментів. Це через те, що її різноманітні технології, принципи та парадигми дозволяють вирішувати різні

завдання, починаючи від створення програм та веб-додатків і закінчуючи розробкою ігор та систем штучного інтелекту для "розумних" пристроїв IoT.

Рисунок 1.5 – Логотип мови програмування [1]

Windows Presentation Foundation (WPF) – це сучасна технологія від Microsoft для створення настільних застосунків з багатими візуальними ефектами та інтерактивним користувацьким інтерфейсом. WPF використовує XAML для опису візуальних елементів, що дозволяє розробникам і дизайнерам працювати над одним і тим самим проєктом, використовуючи знайомі їм інструменти.

Переваги WPF:

- Підтримка апаратного прискорення для графіки.
- Можливість створення складних анімацій та візуальних ефектів.
- Гнучка система шаблонів та стилів для налаштування зовнішнього вигляду застосунків.

Рисунок 1.6 – Логотип технології [2]

Sanford.Multimedia.Midi – це бібліотека для роботи з MIDI-сигналами в .NET. Вона забезпечує необхідні засоби для обробки MIDI-подій, що робить її ідеальною для використання в застосунках, які потребують взаємодії з MIDI-пристроями.

Переваги Sanford.Multimedia.Midi:

- Підтримка різних MIDI-пристроїв.
- Зручний API для обробки MIDI-подій.
- Можливість розширення та налаштування під конкретні потреби проєкту.

Newtonsoft.Json – це популярна бібліотека для роботи з JSON у .NET. Вона дозволяє легко серіалізувати та десеріалізувати об’єкти в JSON та з нього, що є важливим для зберігання налаштувань застосунку та обміну даними.

Переваги Newtonsoft.Json:

- Висока продуктивність та гнучкість.
- Простий у використанні API.
- Підтримка багатьох сценаріїв серіалізації.

TestStack.White – це фреймворк для автоматизованого тестування GUI-застосунків у .NET. Він дозволяє створювати тести для перевірки функціональності інтерфейсу користувача, що допомагає забезпечити якість та стабільність застосунку [5].

Переваги TestStack.White:

- Підтримка різних типів GUI-застосунків.
- Інтуїтивно зрозумілий API для створення тестів.
- Можливість інтеграції з іншими інструментами тестування.

При розробці застосунку використовуються принципи SOLID для забезпечення високої якості коду та підтримки легкої масштабованості й обслуговування проєкту. SOLID включає:

- Single Responsibility Principle (Принцип єдиної відповідальності)
- Open/Closed Principle (Принцип відкритості/закритості)

- Liskov Substitution Principle (Принцип підстановки Лісков)
- Interface Segregation Principle (Принцип поділу інтерфейсів)
- Dependency Inversion Principle (Принцип інверсії залежностей)

Ці технології та підходи забезпечують надійність, ефективність та зручність користування застосунком, що є важливим для успішної реалізації проєкту.

Розглянуті технології, будуть використовуватися для розробки користувацького застосунку. Використання C# та WPF забезпечує сучасний підхід до створення інтуїтивно зрозумілого та функціонального інтерфейсу. Бібліотека Sanford.Multimedia.Midi надає необхідні засоби для роботи з MIDI сигналами, тоді як Newtonsoft.Json забезпечує зручну та ефективну роботу з JSON даними. Для забезпечення високої якості застосунку використовується фреймворк TestStack.White для автоматизованого тестування. Дотримання принципів SOLID гарантує, що код буде підтримуваним, масштабованим та легко адаптованим до змін.

Вибір цих технологій та підходів обґрунтований їх надійністю, ефективністю та широкими можливостями, що дозволяє створити високоякісний застосунок, здатний задовольнити потреби користувачів та забезпечити стабільну роботу з зовнішніми MIDI синтезаторами.

1.5 Огляд вимог

Вимоги до програмного забезпечення для роботи з MIDI синтезатором включають такі функціональні можливості:

1) Відправка та отримання MIDI повідомлень між комп'ютером і зовнішнім синтезатором:
   - Збір даних від синтезатора;
   - Відправка MIDI сигналів для керування синтезатором.

2) Обробка MIDI повідомлень для відтворення звуків:
   - Перетворення MIDI сигналів на аудіо;
   - Управління параметрами звуку на основі MIDI даних.
3) Можливість вибору наборів звуків із бази інструментів:
   - Перегляд доступних баз інструментів;
   - Вибір наборів звуків.
4) Створення, збереження та управління пресетами і налаштуваннями синтезатора:
   - Збереження користувацьких налаштувань;
   - Управління різними пресетами для швидкого виклику.
5) Реалізація інтерфейсу для візуального редагування музичних ефектів:
   - Інтуїтивний графічний інтерфейс для редагування звукових параметрів;
   - Можливість редагування ефектів в реальному часі.
6) Можливість створення послідовностей та секвенцій звуків:
   - Створення та збереження музичних послідовностей;
   - Редагування та комбінування секвенцій.
7) Функціонал відтворення файлів формату MIDI:
   - Програвання MIDI файлів з можливістю паузи та перемотування;
   - Підтримка різних MIDI форматів.
8) Забезпечення стабільної роботи з різними моделями MIDI синтезаторів:
   - Підтримка стандартів MIDI для сумісності з різними пристроями.
9) Забезпечення зручного доступу до налаштувань:
   - Можливість вибору MIDI in та MIDI out пристроїв;
   - Зміна внутрішніх параметрів зовнішніх контролерів.
10) Ефективна логіка для обробки MIDI повідомлень та забезпечення їх точності:

- Мінімізація затримок в обробці MIDI сигналів;
- Точне відтворення нот та ефектів.

Нефункціональні вимоги до теми "Розробка користувацького застосунку для зовнішнього MIDI синтезатора з використанням мови програмування C# та засобами WPF":

1) Продуктивність
   - Застосунок повинен забезпечувати мінімальну затримку при відправці та обробці MIDI-сигналів (не більше 5 мс).
   - Час завантаження застосунку не повинен перевищувати 3 секунд на стандартному обладнанні.
2) Масштабованість
   - Застосунок повинен підтримувати можливість додавання нових MIDI-пристроїв без значних змін у коді.
   - Застосунок має бути спроектований так, щоб легко інтегрувати нові функції та модулі.
3) Надійність
   - Застосунок повинен мати механізми відновлення після збоїв без втрати даних.
   - Програма повинна коректно обробляти винятки та несправності зовнішніх MIDI-пристроїв.
4) Сумісність
   - Застосунок повинен працювати на всіх сучасних версіях операційної системи Windows (від Windows 10 і новіших).
   - Програма повинна підтримувати різні моделі зовнішніх MIDI-синтезаторів через стандартні MIDI-протоколи.
5) Безпека
   - Застосунок повинен забезпечувати безпечне зберігання та передачу даних користувача.

- Використання MIDI-пристроїв не повинно створювати вразливостей для системи користувача.

6) Інтерфейс користувача

- Інтерфейс має бути інтуїтивно зрозумілим і зручним для використання, з дотриманням принципів UX/UI.
- Застосунок повинен підтримувати масштабування інтерфейсу для різних розмірів екранів.

7) Підтримка та обслуговування

- Код застосунку має бути добре документованим для полегшення підтримки та подальшого розвитку.
- Застосунок повинен мати автоматизовані тести для основних функцій.

8) Енергоспоживання

- Застосунок повинен оптимально використовувати ресурси системи, щоб не спричиняти значне навантаження на процесор та пам'ять.
- Програма має бути енергоефективною, особливо при роботі на ноутбуках та інших мобільних пристроях.

9) Відповідність стандартам

- Застосунок повинен відповідати стандартам розробки програмного забезпечення для MIDI-пристроїв.
- Використання бібліотек і фреймворків має відповідати ліцензійним вимогам та бути юридично коректним.

Ці нефункціональні вимоги забезпечать високий рівень якості, зручності та надійності застосунку для користувачів.

РОЗДІЛ 2. ПРОЄКТУВАННЯ ПРОГРАМНОГО РІШЕННЯ

2.1 Вибір архітектури для побудови системи

Розробка користувацьких застосунків на платформі WPF, вимагає особливого підходу через специфіку взаємодії з апаратним забезпеченням та необхідність динамічної зміни даних як з сторони інтерфейсу так і з сторони коду використовуючи вбудовану в технологію прив'язку даних, через це використання вже класичного шаблону MVC, що розділяє модель даних, вигляд та контролер, стає складним і недоречним.

В цьому випадку все більш привабливим стає ідея використання шаблону Model-View-ViewModel (MVVM) [8] як більш пристосованого до роботи з WPF та технології прив'язки даних. Графічне зображення взаємозв'язків компонентів програми, написаної по такому шаблону, можна побачити на рисунку 2.1.

Рисунок 2.1 – Структура MVVM

Основними перевагами використання шаблону MVVM є:

1.)Відокремлення логіки від представлення

В модель-представлення включено всю логіку роботи з MIDI синтезатором: обробка сигналів, управління параметрами звуку тощо. Це дозволяє зосередитись на реалізації специфічних функціях, не турбуючись про деталі користувацького

інтерфейсу. Таким чином, можна легко тестувати та оновлювати логіку без внесення змін у представлення.

2) Підтримка прив'язки даних (Data Binding)

WPF забезпечує потужні можливості для прив'язки даних, що є ключовим аспектом MVVM. Це дозволяє автоматично синхронізувати дані між моделлю та представленням, мінімізуючи код для обробки оновлень вручну.

3) Полегшене тестування

Завдяки чіткому розділенню відповідальностей у MVVM, тестування бізнес-логіки стає значно простішим. Оскільки логіка представлена у вигляді моделей та модель-представлень, її можна тестувати незалежно від інтерфейсу користувача.

4) Підтримка команд (Commands)

MVVM використовує команди для обробки дій користувача, що дозволяє розробникам керувати логікою взаємодії в централізованому місці. Це робить код більш організованим і легко підтримуваним. Для програмного продукту це означає, що всі команди, пов'язані з управлінням, можуть бути визначені у ViewModel, забезпечуючи таким чином чітку структуру і легко тестовану реалізацію.

5) Покращення структури проекту

MVVM сприяє чіткій організації проекту, розділяючи код на логічні частини: модель, представлення та модель-представлення. Це покращує читабельність коду та спрощує його підтримку.

Обраний архітектурний шаблон є доцільним доповненням до стеку обраних технологій, дозволить створити гнучкий, підтримуваний та розширюваний застосунок, який може легко адаптуватися до змін та вдосконалень. MVVM робить процес розробки більш структурованим та організованим, що є ключовим фактором для успішної реалізації складних проектів.

2.2 Виявлення класів сутностей та розробка бізнес логіки

Проаналізувавши опис даної системи ми можемо виявити декілька основних сутностей предметної області.

Сутність MidiEngine – призначена для адаптування інтерфейсу бібліотеки Sanford.Multimedia.Midi. Функція цієї сутності полягає в забезпеченні широкого функціоналу для роботи з MIDI-сигналами та пристроями. Цей клас дозволяє завантажувати, відтворювати, записувати, зупиняти MIDI-файли, керувати MIDI-пристроями, а також обробляти різні MIDI-команди, такі як початок ноти, кінець ноти, застосування ефектів чи вибір інструментів.

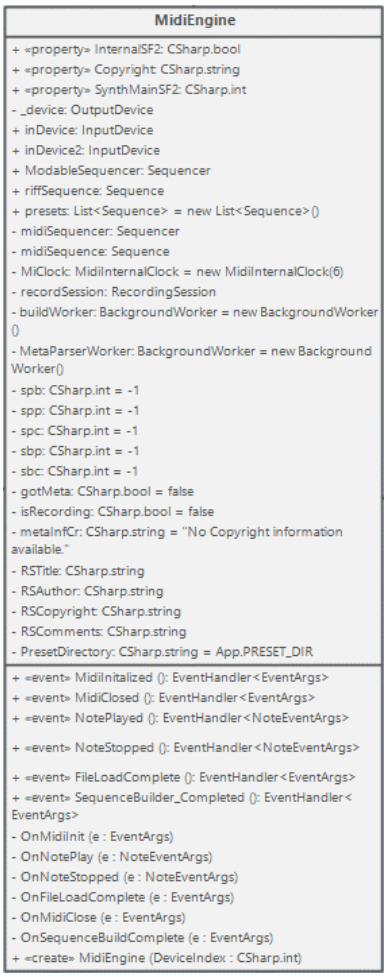

Рисунок 2.2 – Структура MidiEngine

Сутність StudioView – є частковим класом, що реалізує сторінку WPF (Page) і інтерфейс ISynthView. Він призначений для управління MIDI синтезатором через графічний інтерфейс користувача. Основні функціональні можливості включають налаштування MIDI пристроїв, вибір інструментів, управління параметрами каналів MIDI та інтеграцію з основним додатком. Таким чином, StudioView є центральним компонентом для взаємодії з MIDI синтезатором через графічний інтерфейс, забезпечуючи широкий спектр налаштувань і контрольних можливостей для користувача.

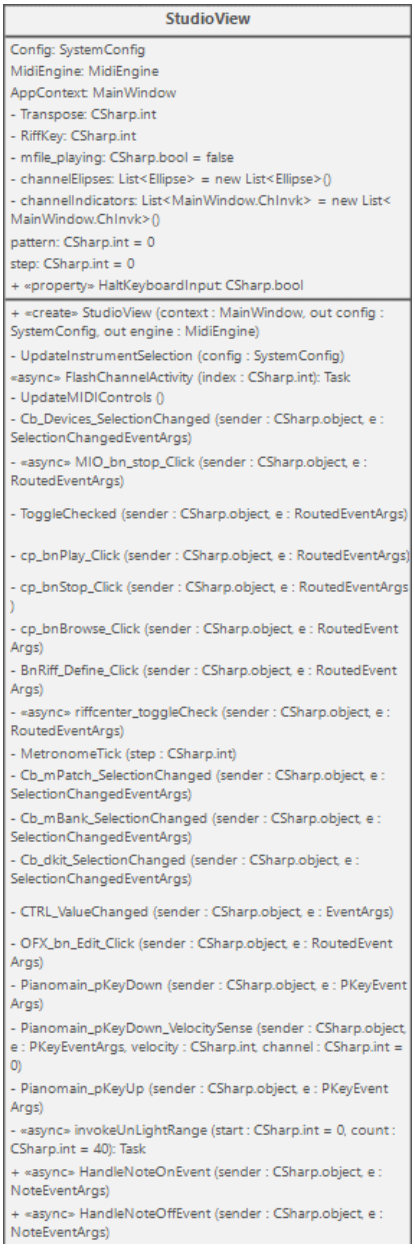

Рисунок 2.3 – Структура StudioView

Сутність SeqData – це компонент, який забезпечує збереження та створення послідовностей нот. Вона зберігає всю необхідну інформацію про ноти в послідовності, включаючи ефекти, рівень гучності та налаштування інструментів для кожної окремої ноти. Ця сутність дозволятиме користувачам зберігати послідовності нот та змінювати параметри кожної ноти.

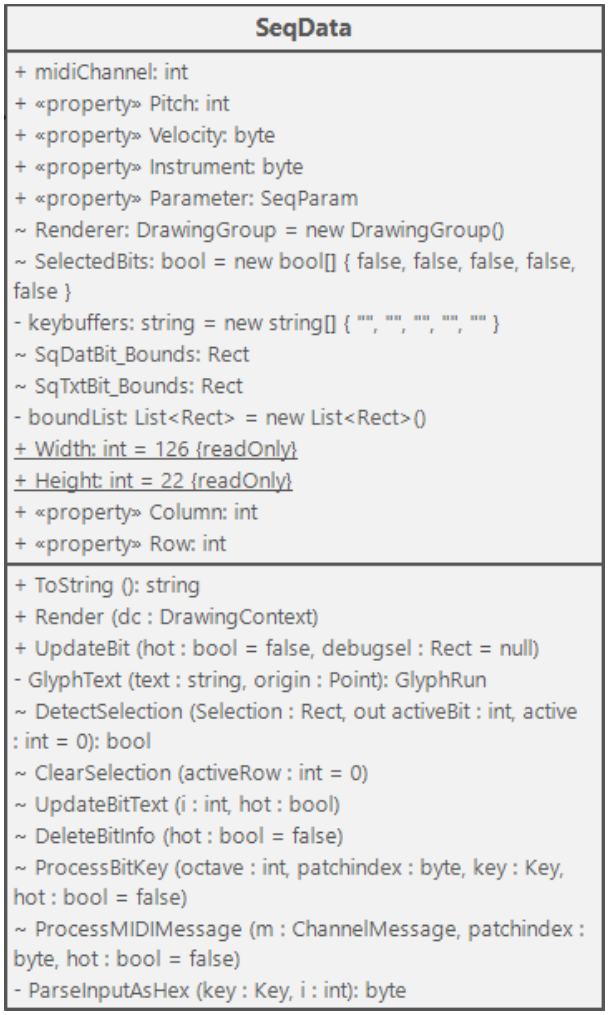

Рисунок 2.4 – Структура SeqData

Сутність InstrumentDefinition – грає визначну роль у визначенні та збереженні списків інструментів, які доступні в бібліотеці Sanford.Multimedia.Midi. Ця сутність відповідає за управління інформацією про інструменти, забезпечуючи доступ до них і їхню коректну інтеграцію з іншими

частинами системи. Завдяки чому користувачі можуть легко обирати та налаштовувати інструменти для своїх музичних проектів

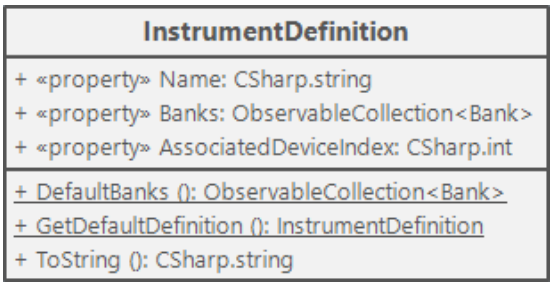

Рисунок 2.5 – Структура InstrumentDefinition

Інтерфейс ISynthKey напряму відповідає за обробку натискань клавіш, Реалізація цього інтерфейсу в програмі дозволяє обробляти натискання як фізичних клавіш контролера, так і віртуальних клавіш на екрані. Це забезпечує користувачам можливість взаємодії з програмою через різні пристрої вводу, що робить процес створення музики більш інтуїтивним та доступним. На рисунку 2.6 зображено сутність цього інтерфейсу та діаграму класів, які реалізовуватимуть його.

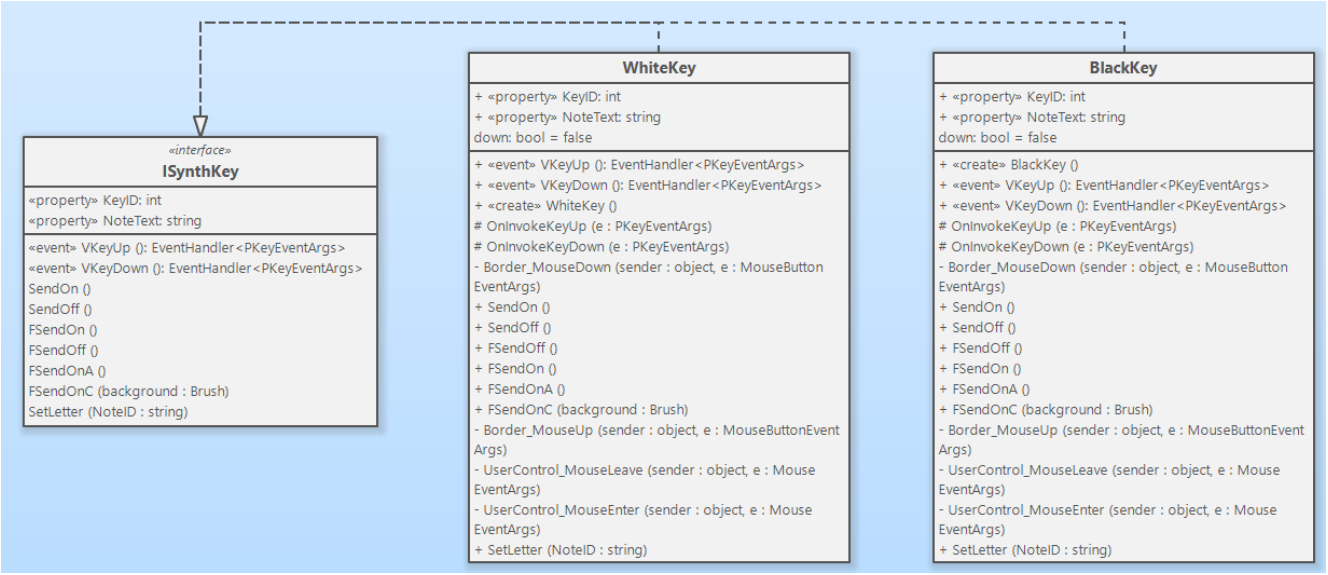

Рисунок 2.6 – Структура ISynthKey та його реалізації

Завдяки створенню діаграм класів для всіх сутностей вдалося чітко окреслити бажану структуру системи та опрацювання користувацького вводу. Ці діаграми, у поєднанні з інструментами фреймворку, були доповнені, адаптовані та використані для формування остаточного програмного рішення.

В результаті було отримано програмну реалізацію моделі у вигляді діаграми сутностей. Ця діаграма представлена на рисунку 2.7.

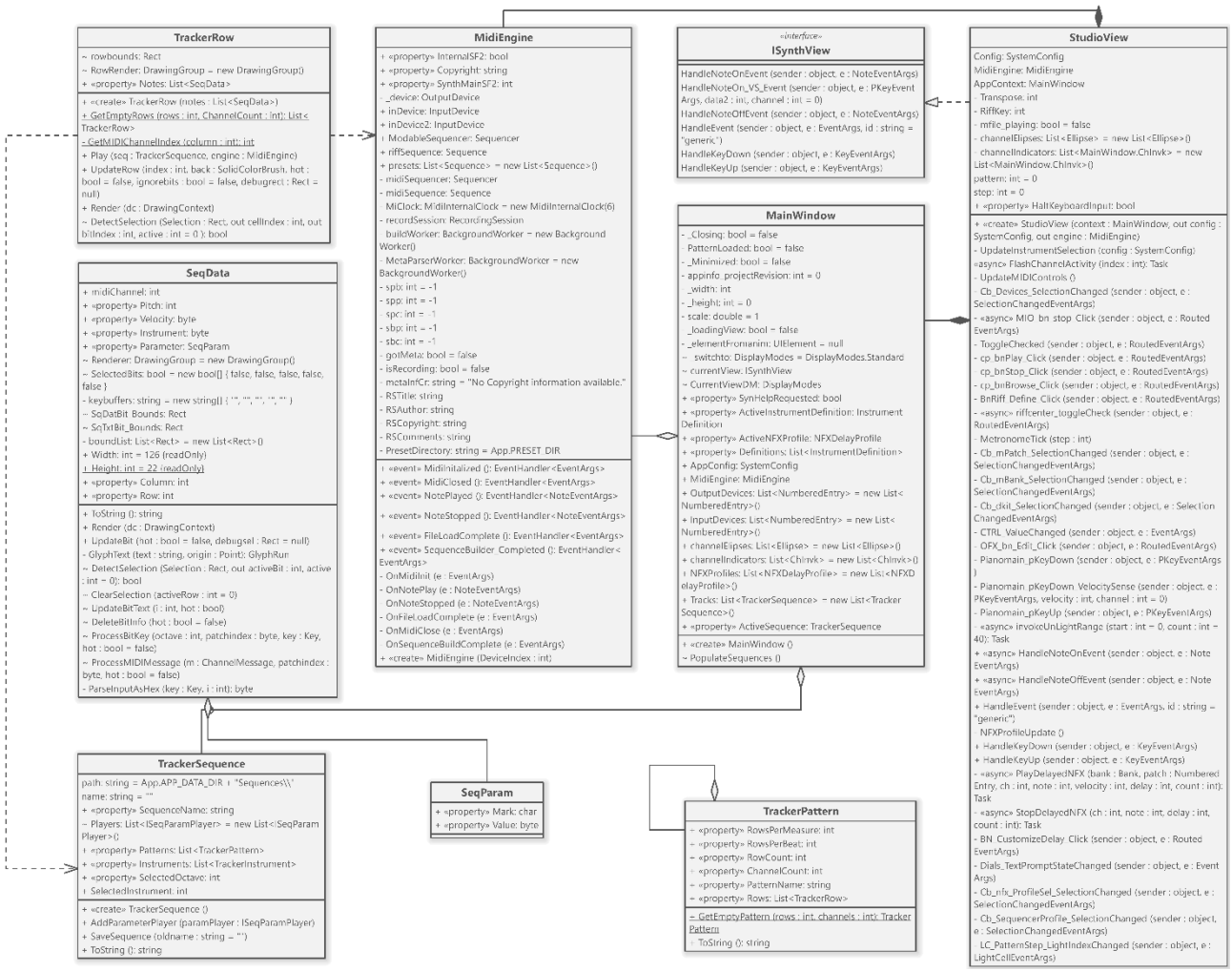

Рисунок 2.7 – Загальна структура системи

В цілому, система, описана діаграмою, надає міцну основу для створення користувацького застосунку для роботи з зовнішнім MIDI-синтезатором.

Завдяки її модульній конструкції та можливості розширення, система може бути легко адаптована до потреб конкретних користувачів та проектів.

2.3 Розробка інтерфейсу

Інтерфейс користувача (UI) відіграє ключову роль у забезпеченні зручного та інтуїтивно зрозумілого доступу до функціональних можливостей програми.

Для розробки ефективного UI буде використано WPF фреймворк, що пропонує широкий спектр інструментів для створення багатих та динамічних інтерфейсів.

2.3.1 Аналіз варіантів використання системи

Інтерфейс програми повинен реалізувати всі функціональні вимоги згадані в пункті опису вимог та забезпечити користувачу зручний доступ до функціональностей системи.

Для кращого розуміння та моделювання взаємодії користувачів з системою та її інтерфейсом, було створено діаграму варіантів використання, яка зображена на рисунку 2.8.

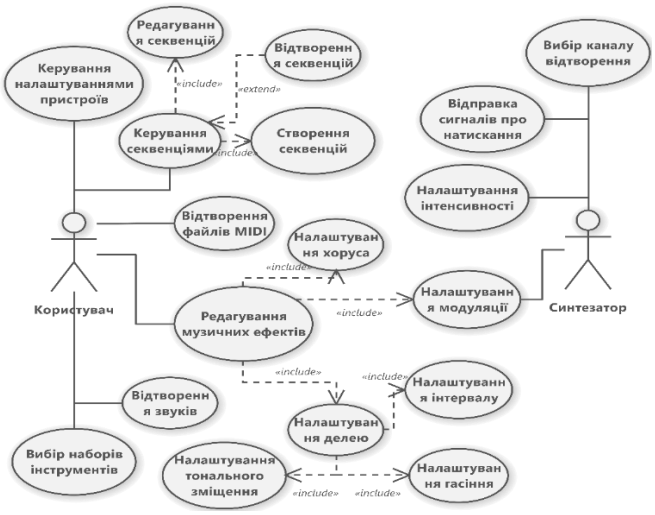

Рисунок 2.8 – Діаграма варіантів використання системи

Для цієї системи було виділено два основних актори, які і будуть взаємодіяти з програмою:

- Користувач: Основний актор, який взаємодіє з програмою для створення, редагування та відтворення музичних ефектів і послідовностей.
- Синтезатор: Зовнішній пристрій, з яким програма обмінюється MIDI сигналами.

Таблиця 2.1 містить всі основні елементи, що відображають функціональні можливості системи для роботи з MIDI сигналами.

Таблиця 2.1 – Варіанти використання системи

Код	Актор	Найменування	Формування
К1	Користувач	Налаштування пристроїв	Налаштування MIDI in та MIDI out пристроїв
К2	Користувач	Відтворення файлів MIDI	Відтворення та пауза файлів формату MIDI
К3	Користувач	Редагування музичних ефектів	Налаштування основних ефектів хорусу, модуляції та дилею
К4	Користувач	Вибір набору інструменту	Перегляд та вибір різних наборів звуків
К5	Користувач	Відтворення звуків	Відтворення звуків за допомогою екранної клавіатури
К6	Користувач	Керування секвенціями	Включає створення, відтворення та редагування секвенцій звуків

Продовження таблиці 2.1

Код	Актор	Найменування	Формування
К7	Користувач	Налаштування дилею	Налаштування інтервалу повторень, степені гасіння та тонального зміщення звуку
К8	Користувач	Налаштування модуляції	Налаштування модуляційних ефектів
С1	Синтезатор	Відправка сигналів про натискання	Відправка MIDI сигналів від контролера в систему
С2	Синтезатор	Налаштування кривої інтенсивності	Налаштування кривої гучності відносно швидкості натискання клавіші
С3	Синтезатор	Вибір каналу відтворення	Вибір відповідного каналу для передавання сигналів
С4	Синтезатор	Налаштування модуляції	Налаштування модуляційних ефектів за допомогою джойстика

Отже, проаналізувавши виявлені варіанти використання системи було вирішено розділити функціональність системи на 4 вікна, одного основного та трьох додаткових, які дозволять відділити самостійні частини системи та розгрузити основний робочий простір. Далі буде розглянуто кожне з 4 виділених вікон системи.

2.3.2 Головне вікно програми

StudioView – це головне вікно програми, яке містить, віртуальну клавіатуру синтезатора, елементи керування для відправки та отримання MIDI повідомлень, вибору наборів звуків, редагування музичних ефектів та відтворення MIDI файлів. Шаблон зовнішнього вигляду цього вікна можна побачити на рисунку 2.9.

Рисунок 2.9 – Шаблон інтерфейсу головного вікна

Після завершення моделювання вікно міститиме наступні елементи керування:

- Панель екранної клавіатури – ця панель використовується для відтворення звуків емулюючи MIDI сигнали фізичного контролера. Вона містить набір кнопок вводу з підписами нот за які вони відповідають.
- Меню вибору інструментів – це меню використовується для вибору наборів звуків з бази інструментів. Воно містить два випадаючих списки один з вибором набору інструментів, інший з вибором конкретного інструменту.
- Панель редагування ефектів – ця панель використовується для візуального редагування музичних ефектів. Вона містить користувацькі

елементи керування для налаштування параметрів різних ефектів, таких як гучність, зміщення тональності, баланс, модуляція, реверберація та хорус.

- Панель відтворення MIDI – ця панель використовується для відтворення MIDI файлів. Вона містить кнопки для вибору, відтворення та паузи MIDI файлів.
- Панель налаштування MIDIOut – це панель для можливості скрізьної передачі сигналу, містить випадаючий список із доступними пристроями.
- Панель секвенсора – панель для візуалізації звукових послідовностей, та входу в вікно секвенсора, містить індикатори вибраного патерну, кнопку входу в секвенсор та чекбокс активаціїї його роботи.

2.3.3 Додаткові вікна програми

Отже, щоби не перенавантажувати основне робоче вікно у віддільні вікна було виділено функціонал роботи з секвенціями, налаштуванням ефекту затримки (делею), та загальні налаштування роботи з контролерами.

Вікно SequenceTracker – це вікно використовується для створення, редагування та комбінування музичних секвенсій. Вікно SequenceTracker містить наступні елементи керування:

Панель послідовностей – панель використовується для створення та редагування музичних послідовностей. Вона містить поля вводу для введення назви та опису послідовності, а також кнопки для додавання, видалення та редагування послідовностей. Реалізована у вигляді таблиці де стовпцями виступають канали для програвання, а стрічками часові відмітки

Панель секвенцій – панель використовується для вибору інструментів, керування програванням послідовностей, вибору паттернів та, неймингу послідовностей та патернів. Вона містить поля вводу для введення назви секвенції,

кнопки для додавання, видалення та редагування секвенцій, модифіковані чекбокси для вибору патеру для зміни, та випадаючий список для вибору інструментів.

Шаблон зовнішнього вигляду цього вікна для роботи з секвенціями звуків можна побачити на рисунку 2.10.

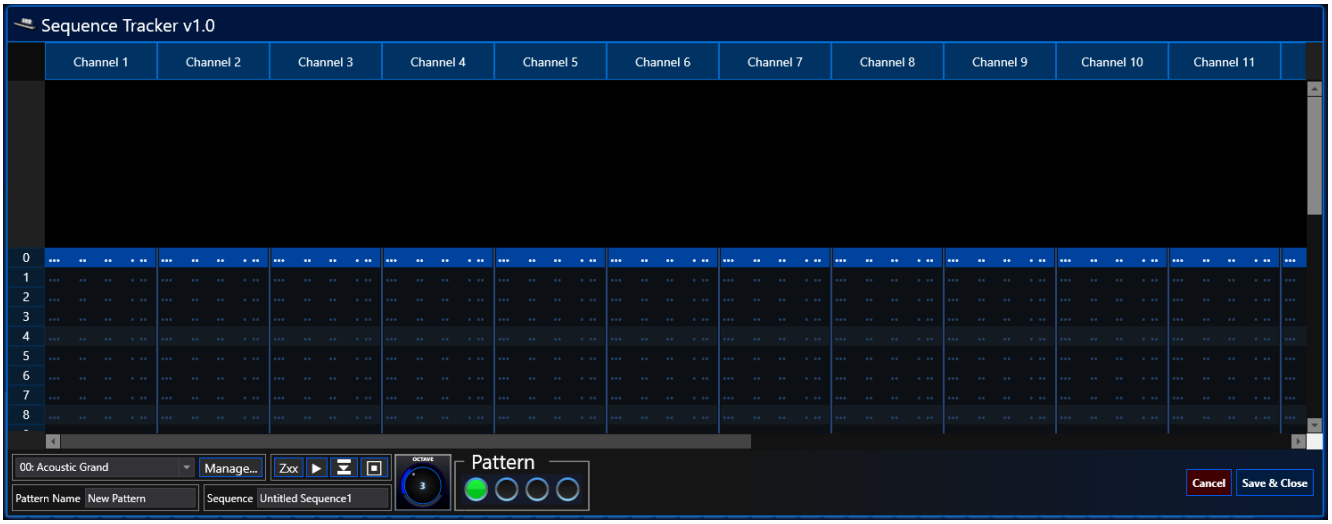

Рисунок 2.10 – Шаблон інтерфейсу вікна SequenceTracker

Вікно CustomizeNoteFX Delay – це вікно використовується для налаштування параметрів ефекту затримки, таких як кількість повторень, інтервал, зміщення ноти та гучність. Вікно CustomizeNoteFX Delay містить наступні елементи керування:

Поле кількості повторень – поле використовується для введення кількості разів, що ефект затримки буде повторюватися.

Поле інтервалу – поле використовується для введення інтервалу між повтореннями ефекту затримки.

Поле зміщення ноти – поле використовується для введення значення, на яке буде зміщена нота після застосування ефекту затримки.

Поле змінення гучності – повзунок використовується для налаштування гучності після ефекту затримки.

Поле роботи з профілями – поле зі списком профілів, кнопкою збереження та кнопкою видалення.

Шаблон зовнішнього вигляду цього вікна для роботи з налаштування затримки можна побачити на рисунку 2.11.

Рисунок 2.11 – Шаблон інтерфейсу вікна CustomizeNoteFX Delay

Вікно Configuration – це вікно використовується для керування пристроями MIDI in та MIDI out, а також для підключення ефектів контролерів. Вікно Configuration містить наступні елементи керування:

Поле вибору пристрою 1 – поле використовується для вибору першого MIDI-пристрою вводу.

Relay Mode для пристрою 1 – опція, що дозволяє увімкнути або вимкнути режим реле для першого MIDI-пристрою вводу.

Поле вибору пристрою 2 – поле використовується для вибору другого MIDI-пристрою вводу.

Relay Mode для пристрою 2 – опція, що дозволяє увімкнути або вимкнути режим реле для другого MIDI-пристрою вводу.

Група Allowed Device Control Parameters – містить чекбокси для налаштування параметрів контролю пристрою.

Шаблон зовнішнього вигляду цього вікна для налаштування контролерів можна побачити на рисунку 2.12.

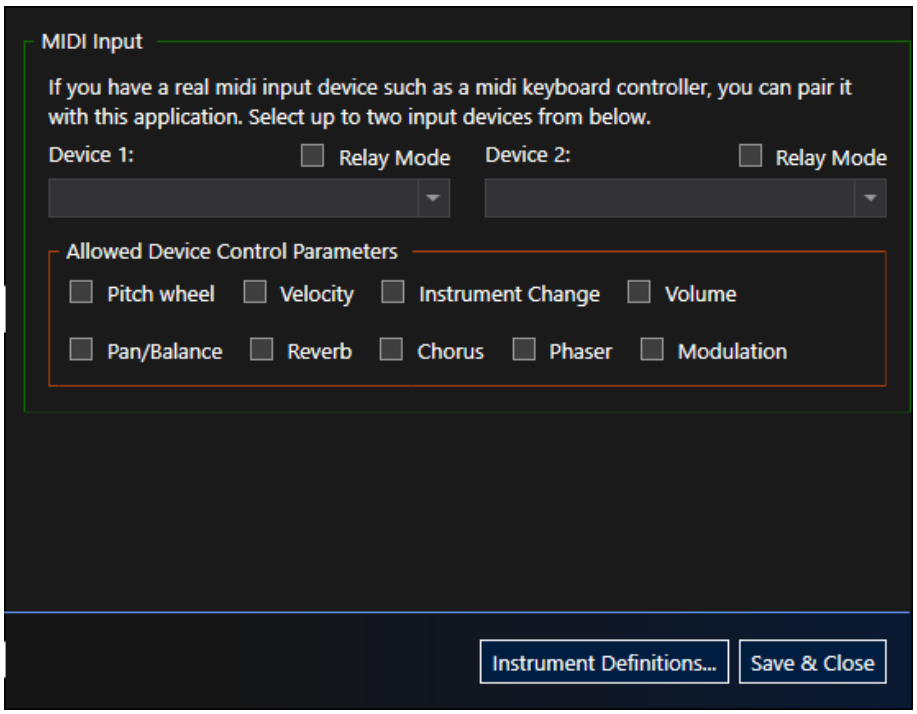

Рисунок 2.12 – Шаблон інтерфейсу вікна Configuration

Отже, було спроектовано інтерфейс користувача з використанням WPF фреймворку дозволило створити багатий та динамічний інтерфейс, який реалізує всі функціональні вимоги системи та забезпечує користувачам зручний доступ до функціональних можливостей програми. Окремий аналіз варіантів використання системи дав можливість детально пропрацювати кожен сценарій взаємодії користувача з системою, що допомогло створити ефективний та зрозумілий інтерфейс.

РОЗДІЛ 3. ТЕСТУВАННЯ СИСТЕМИ

3.1 Мета та обсяг тестування

Метою тестування системи є забезпечення її коректної роботи відповідно до заданих вимог та специфікацій. Тестування охоплює перевірку всіх основних функціональних можливостей системи, включаючи роботу з MIDI-сигналами, управління інтерфейсом користувача, налаштування синтезатора та збереження музичних послідовностей.

Функціональне тестування передбачає перевірку всіх ключових функцій системи, таких як відтворення зупинка MIDI-файлів, а також керування MIDI-пристроями та обробка різних MIDI-команд. Це тестування дозволяє виявити будь-які проблеми з основними операціями системи та забезпечити їх безперебійну роботу.

Тестування інтерфейсу користувача є критичним аспектом, оскільки інтерфейс повинен бути інтуїтивно зрозумілим і зручним для користувачів. Це тестування включає перевірку всіх вікон та їх елементів керування, таких як панель екранної клавіатури, меню вибору інструментів, панель редагування ефектів, панель відтворення MIDI та панель налаштування MIDIOut. Важливо переконатися, що всі ці елементи функціонують коректно і відповідають очікуванням користувачів, надаючи їм зручний доступ до всіх необхідних функцій.

Загалом, тестування системи є важливим етапом розробки, який дозволяє виявити та виправити помилки, забезпечуючи високу якість кінцевого продукту. Виконання комплексного тестування допомагає гарантувати, що система буде надійною, зручною у використанні та відповідатиме всім вимогам користувачів, забезпечуючи високий рівень користувацького досвіду.

3.2 Функціональне тестування

Функціональне тестування охоплює перевірку всіх функціональних можливостей системи, описаних в таблиці варіантів використання. Основна увага приділялася наступним аспектам:

Налаштування пристроїв (К1) – перевірка можливості налаштування MIDI in та MIDI out пристроїв. Тестування правильності підключення та передачі сигналів між програмою та пристроями.

Відтворення файлів MIDI (К2) – перевірка функцій відтворення та паузи MIDI файлів. Тестування коректності завантаження та відтворення різних MIDI файлів.

Редагування музичних ефектів (К3, К7, К8) – перевірка налаштувань ефектів хорусу, модуляції та дилею. Тестування візуального редагування та збереження параметрів ефектів.

Вибір набору інструменту (К4) – перевірка можливості перегляду та вибору різних наборів звуків. Тестування коректності зміни інструментів під час відтворення.

Відтворення звуків (К5) – перевірка роботи екранної клавіатури для відтворення звуків. Тестування відповідності звуків нотам на клавіатурі.

Керування секвенціями (К6) – перевірка створення, відтворення та редагування секвенцій звуків. Тестування функціональності збереження та завантаження секвенцій.

Відправка сигналів про натискання (С1) – перевірка коректності передачі MIDI сигналів від контролера до системи.

Налаштування кривої інтенсивності (С2) – перевірка налаштувань кривої гучності відносно швидкості натискання клавіші.

3.3 Тестування інтерфейсу користувача

Тестування інтерфейсу користувача включає перевірку всіх елементів управління та їх взаємодію. Основні аспекти тестування:

1. Панель екранної клавіатури
   - Перевірка коректного відображення клавіш та їх відповідність нотам.
   - Тестування взаємодії користувача з віртуальною клавіатурою.
2. Меню вибору інструментів
   - Перевірка роботи випадаючих списків для вибору набору та конкретного інструменту.
   - Тестування правильності відображення та зміни інструментів.
3. Панель редагування ефектів
   - Перевірка візуального редагування параметрів ефектів.
   - Тестування взаємодії користувача з елементами керування параметрами ефектів.
4. Панель відтворення MIDI
   - Перевірка роботи кнопок вибору, відтворення та паузи MIDI файлів.
   - Тестування коректності відображення стану відтворення.
5. Панель налаштування MIDIOut
   - Перевірка можливості вибору пристроїв та передачі сигналу.
   - Тестування випадаючого списку з доступними пристроями.
6. Панель секвенсора
   - Перевірка візуалізації звукових послідовностей та входу в вікно секвенсора.
   - Тестування взаємодії користувача з індикаторами вибраного патерну та іншими елементами.

3.4 Тестові сценарії

Для ретельного тестування програмного забезпечення застосовуватиметься формат тест-кейсів. Тест-кейс являє собою детальний опис послідовних кроків, які необхідно виконати для перевірки певного тестувального сценарію [9]. Використання тест-кейсів дозволяє зробити процес тестування програмного забезпечення максимально ефективним та точним. Кожен тест-кейс забезпечує систематичний і структурований підхід до тестування, що дає змогу виявити можливі помилки та недоліки в програмі на ранніх етапах. Це, своєю чергою, сприяє підвищенню якості кінцевого продукту та забезпеченню його відповідності встановленим вимогам і стандартам. Тест-кейси поділяються на два типи:

- Позитивні: Перевіряють коректне виконання функцій програми.
- Негативні: Перевіряють некоректне виконання функцій програми.

Результати тестування, отримані за допомогою тест-кейсів, дозволяють виявити помилки в програмі та гарантувати її високу якість.

Структура тест-кейсів:

Summary (Заголовок): Короткий та точний опис модуля чи функції, що тестується.

Preconditions (Умови запуску): Стан системи, необхідний для початку тестування.

Steps to reproduce (Кроки виконання): Детальний опис дій, які потрібно виконати.

Expected Result (Очікуваний результат): Як система повинна поводитися під час тестування.

Postconditions (Умови завершення): Стан системи після завершення тестування.

Отже, з урахуванням вказаних умов було написано тест-кейси та проведено тестування основних функціональних можливостей системи:

Тест-кейс 1 – відтворення звуків за допомогою екранної клавіатури

Summary: Перевірка можливості відтворення звуків за допомогою екранної клавіатури.

Preconditions: Програма запущена.

Steps to reproduce:

1) Відкрити програму.

2) Перейти до панелі екранної клавіатури.

3) Натиснути на клавіші екранної клавіатури.

Expected Result: Відтворюються відповідні звуки.

Postconditions: Звуки відтворюються.

Рисунок 3.1 – Виконання тест-кейсу 1

Тест-кейс 2 – налаштування інструменту

Summary: Перевірка можливості вибору та налаштування інструменту з існуючої бази.

Preconditions: Програма запущена.

Steps to reproduce:

1) Відкрити програму.

2) Перейти до меню вибору інструментів.

3) Вибрати набір інструментів.

4) Вибрати конкретний інструмент.

Expected Result: Інструмент вибрано з існуючої бази і налаштовано без помилок.

Postconditions: Інструмент налаштований.

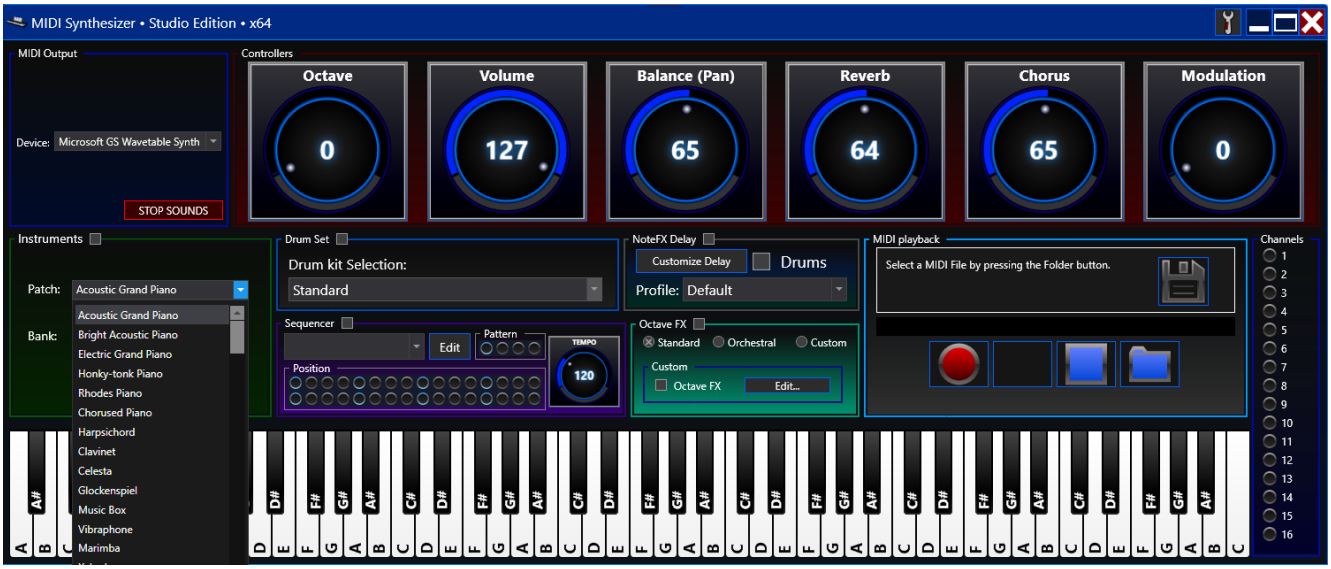

Рисунок 3.2 – Виконання тест-кейсу 2

Тест-кейс 3 – відправлення сигналів за допомогою підключеного контролера MIDI

Summary: Перевірка можливості відтворення звуків за допомогою контролера MIDI.

Preconditions: Програма запущена, підключено контролер.

Steps to reproduce:

1) Відкрити програму.

2) Підключити контролер.

3) Надіслати сигнал з контролера.

Expected Result: Система отримує та обробляє MIDI сигнали без затримок

Postconditions: MIDI сигнали від контролера оброблені системою.

Рисунок 3.3 – Виконання тест-кейсу 3

Тест-кейс 4 – відтворення MIDI файлу

Summary: Перевірка можливості відтворення існуючого файлу MIDI формату.

Preconditions: Завантажено MIDI файл.

Steps to reproduce:

1) Відкрити програму.

2) Завантажити MIDI файл.

3) Натиснути кнопку "Відтворення".

Expected Result: MIDI файл починає відтворюватися без затримок або помилок.

Postconditions: MIDI файл відтворюється.

Рисунок 3.4 – Виконання тест-кейсу 4

Тест-кейс 5 – тестування роботи ефектів

Summary: Перевірка коректності роботи ефектів (хорус, модуляція, дилей) у системі.

Preconditions: Система запущена, користувач знаходиться у вікні StudioView.

Steps to reproduce:

1) Перейти до панелі редагування ефектів у вікні StudioView.

2) Налаштувати параметри ефекту хорус.

3) Повторити кроки 1-2 для ефектів модуляції та дилей.

4) Відтворити звуки за допомогою клавіатури та перевірити, чи застосовані ефекти відображаються у звуковому відтворенні.

Expected Result: Ефекти хорус, модуляція та дилей правильно застосовуються, чутні зміни в звуці відповідно до налаштувань.

Postconditions: Налаштування ефектів зберігаються, система готова до подальшого використання.

Рисунок 3.5 – Виконання тест-кейсу 5

Тест-кейс 6 – тестування вікна створення секвенцій звуків

Summary: Перевірка коректної роботи вікна SequenceTracker для створення та редагування секвенцій звуків.

Preconditions: Система запущена, користувач увійшов у вікно SequenceTracker.

Steps to reproduce:

1) Відкрити вікно SequenceTracker.

2) Ввести назву нової секвенції в полі "Назва".

3) Додати нову послідовність нот за допомогою панелі послідовностей.

4) Вибрати інструмент для послідовності за допомогою випадаючого списку.

5) Зберегти послідовність, натиснувши кнопку "Зберегти".

6) Відтворити секвенцію та перевірити коректність відтворення нот.

Expected Result: Нова секвенція успішно створюється та зберігається, ноти відтворюються правильно згідно з налаштуваннями.

Postconditions: Створена секвенція зберігається в системі, інтерфейс готовий до створення нових секвенцій.

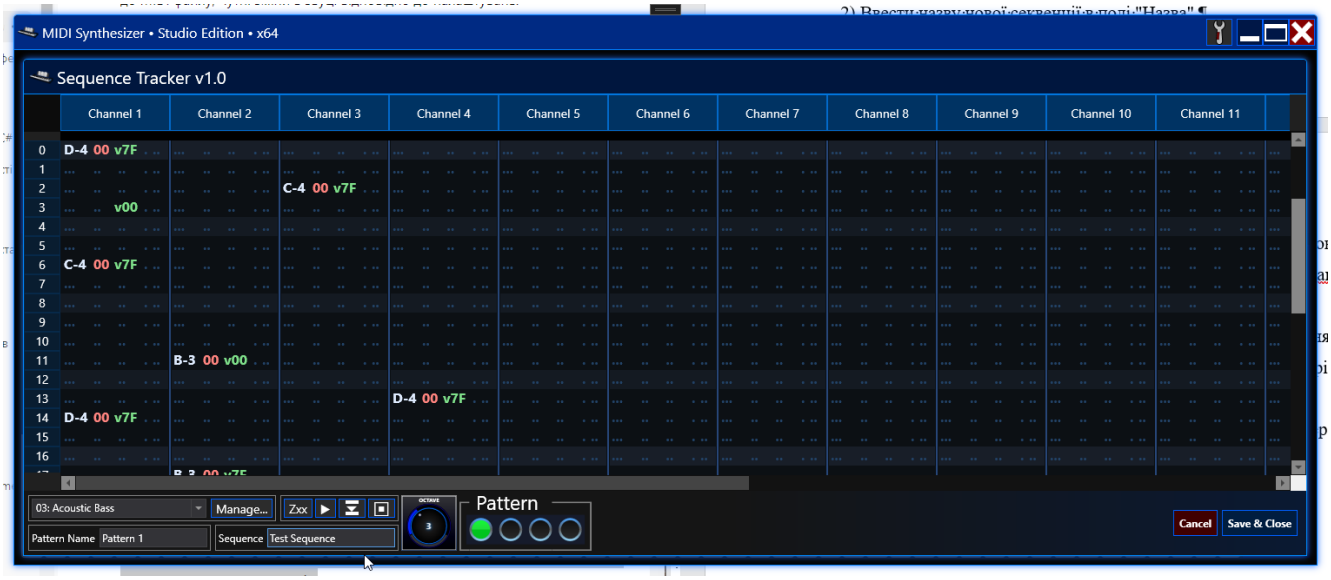

Рисунок 3.6 – Виконання тест-кейсу 6

Тест-кейс 7 – тестування вікна налаштування MIDI in та MIDI out

Summary: Перевірка коректної роботи вікна Configuration для налаштування пристроїв MIDI in та MIDI out.

Preconditions: Система запущена, користувач увійшов у вікно Configuration.

Steps to reproduce:

1) Відкрити вікно Configuration.

2) Вибрати перший MIDI-пристрій вводу з випадаючого списку.

3) Увімкнути режим реле для першого MIDI-пристрою.

4) Повторити кроки 2-3 для другого MIDI-пристрою вводу.

5) Налаштувати Allowed Device Control Parameters, вибравши відповідні параметри.

6) Натиснути кнопку "Зберегти налаштування" для збереження конфігурації.

7) Перевірити коректність роботи обраних MIDI пристроїв при введенні сигналів.

Expected Result: MIDI-пристрої налаштовуються правильно, сигнали вводяться та виводяться коректно, параметри контролю працюють відповідно до налаштувань.

Postconditions: Налаштування MIDI-пристроїв зберігаються, система готова до використання з обраними пристроями.

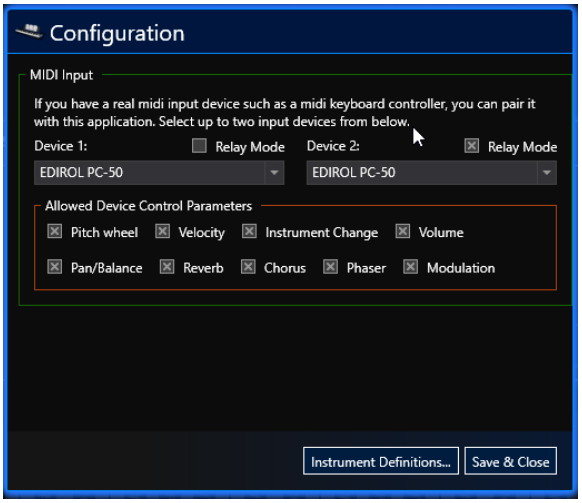

Рисунок 3.7 – Виконання тест-кейсу 7

Підсумовуючи, можна сказати, що тестування системи допомогло забезпечити її безперебійну та коректну роботу відповідно до специфікацій. Було виконано функціональне тестування, яке включало перевірку основних можливостей системи дозволило виявити потенційні проблеми та виправити їх на ранніх етапах розробки. Особлива увага приділялась тестуванню інтерфейсу користувача, перевірялись всі основні елементи інтерфейсу, включаючи екранну клавіатуру, меню вибору інструментів, панель редагування ефектів, відтворення MIDI та налаштування MIDIOut, щоб забезпечити їх правильне функціонування.

РОЗДІЛ 4. БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ ТА ОСНОВИ ОХОРОНИ ПРАЦІ

4.1 Долікарська медична допомога при захворюваннях, травмах та в умовах надзвичайних ситуацій.

Ціла низка чинників навколишнього середовища може негативно впливати на здоров’я та життя людини. Нещасні випадки виникають удома, на роботі, на дорогах, у місцях відпочинку, тобто далеко від лікувальних закладів. Вони завжди трапляються несподівано і вимагають невідкладної допомоги протягом перших 4–5 хвилин після нещасного випадку [11], коли ще не відбулися необоротні зміни у клітинах організму. Перша долікарська допомога – це комплекс простих термінових дій, спрямованих на збереження здоров’я і життя потерпілого.

Перша долікарська допомога може і повинна бути надана очевидцями трагедії, для чого потрібні знання й уміння. Кожній людині необхідно знати методи первинної реанімації, тобто комплекс заходів щодо надання допомоги на місці події, попередження летального наслідку й оживлення людини. За даними ВООЗ, близько 30% осіб [11], які загинули внаслідок нещасних випадків та надзвичайних ситуацій, могли б жити, якби їм своєчасно і правильно надали першу долікарську допомогу та вжили заходів щодо оживлення або своєчасно забезпечили транспортування до медичного закладу.

Під час надання першої долікарської допомоги необхідно керуватися такими принципами:

- виконання дій повинно бути правильним та відповідати характеру травми або захворювання.
- необхідно обирати найефективніші методи допомоги.
- швидке реагування є критичним для успішного результату.
- дії повинні бути логічними і послідовними.
- необхідно діяти впевнено, без сумнівів.
- потрібно зберігати спокій, щоб не посилювати паніку у потерпілого.

Слідуючи вказаним принципам надання першої медичної допомоги забеспечить те що надана допомога буде якісною та не нашкодить постраждалому.

Основні етапи надання долікарської допомоги включають декілька ключових кроків, які повинні бути виконані з урахуванням специфіки травми або захворювання:

1) Усунення впливу небезпечних факторів: Необхідно звільнити потерпілого від дії електричного струму, винести з приміщення, що горить, або з води, погасити одяг, який палає тощо.

2) Оцінка стану потерпілого: Визначення характеру і тяжкості травми, що становить найбільшу загрозу для життя потерпілого.

3) Відновлення життєво важливих функцій організму: Відновлення прохідності дихальних шляхів, здійснення штучного дихання, зовнішній масаж серця, зупинка кровотечі, знеболення, іммобілізація місця перелому, накладення пов’язки тощо.

4) Виклик медичної допомоги: Виклик швидкої медичної допомоги або лікаря, або вжиття заходів для транспортування потерпілого до найближчого лікувального закладу.

5) Підтримка основних життєвих функцій до прибуття медичного працівника: Збереження життєдіяльності потерпілого до моменту надання професійної медичної допомоги.

Кожна людина повинна володіти базовими знаннями і навичками з надання першої допомоги, адже від цього може залежати життя оточуючих у критичній ситуації.

Своєчасно надана та правильно здійснена перша долікарська допомога не лише рятує життя потерпілому, а й забезпечує подальше успішне лікування, запобігає розвитку важких ускладнень, знижує втрату працездатності або ступінь каліцтва. Кожній людині необхідно володіти знаннями і навичками з надання першої долікарської допомоги, адже від цього може залежати життя потерпілого в умовах надзвичайної ситуації.

4.2 Методи оцінки соціальної та соціально-економічної ефективності заходів щодо покращення умов та охорони праці.

Україна активно інтегрується у світове співтовариство, де концепції формування та розвитку безпеки й охорони праці пройшли тривалий еволюційний шлях і показали свою спроможність в економічно розвинених країнах. Законодавство Євросоюзу у сфері охорони праці можна умовно розділити на дві групи: директиви ЄС щодо захисту працівників та директиви ЄС щодо випуску товарів на ринок (включаючи обладнання, устаткування, машини, засоби колективного та індивідуального захисту, які використовують працівники на робочому місці). До управління охороною праці у західних країнах застосовуються такі підходи, як ISRS (International Safety Rating System), OHSAS (Occupetional Health and Safety Assessment), управління ризиком на підприємстві та інтеграція системи управління охороною праці з управлінням якістю (ISO 9001), охороною навколишнього середовища (ISO 14001) і безпекою (OHSAS 18001) [12].

Одним з основних методів оцінки соціальної ефективності заходів щодо покращення умов та охорони праці є опитування працівників щодо їх задоволеності робочими умовами та безпекою на робочому місці. Цей метод дозволяє отримати безпосередню зворотний зв'язок від працівників та виявити проблеми, які можуть залишатися непоміченими при виключно формальному підході до оцінки.

Аналіз статистики нещасних випадків на виробництві є важливим показником соціальної ефективності заходів з охорони праці. Зменшення кількості та тяжкості нещасних випадків свідчить про покращення умов праці та ефективність впроваджених заходів.

Метод економічної оцінки втрат полягає у визначенні прямих та непрямих втрат від нещасних випадків та професійних захворювань. Прямі втрати

включають витрати на лікування та компенсації, непрямі - втрати продуктивності, простої обладнання та зниження якості продукції. Порівняння цих втрат з витратами на заходи з покращення умов праці дозволяє оцінити їх економічну доцільність.

Використання міжнародних стандартів, таких як OHSAS 18001, ISO 9001 та ISO 14001, дозволяє підприємствам систематично підходити до управління охороною праці та оцінки її ефективності [12]. Цикл PDCA (плануй, роби, перевіряй, впливай) є основою цих стандартів і забезпечує безперервне покращення умов праці.

Методи оцінки соціальної та соціально-економічної ефективності заходів щодо покращення умов та охорони праці є ключовим елементом успішного управління охороною праці на підприємстві. Вони дозволяють не лише підвищити безпеку та задоволеність працівників, але й забезпечують економічну ефективність підприємства в цілому. Інтеграція цих методів з міжнародними стандартами сприяє розвитку системного підходу до охорони праці та створює основу для стійкого розвитку підприємства.

ВИСНОВКИ

У результаті виконання даної кваліфікаційної роботи було створено програмне забезпечення, яке вирішує поставлені задачі та досягає визначених цілей. Зокрема, було розроблено програму для обробки MIDI повідомлень та управління музичними контролерами з використанням мови програмування C# та фреймворку WPF.

Програмний продукт надає користувачам можливість ефективно управляти MIDI пристроями, створювати та редагувати пресети, а також налаштовувати музичні ефекти. Інтуїтивно зрозумілий інтерфейс забезпечує зручне користування програмою для широкого кола користувачів.

Розробка програмного забезпечення включала кілька етапів: аналіз вимог, проектування системи, реалізацію та тестування. Детальний аналіз вимог користувачів дозволив уникнути помилок проєктування та інших ресурсоємних затрат, забезпечивши якість розробленого продукту.

Основою розробленого програмного забезпечення стали мова програмування C# та фреймворк WPF, які забезпечують створення сучасного та функціонального інтерфейсу. Вагомими перевагами цього технологічного стеку є: підтримка високої продуктивності та стабільності роботи програмного забезпечення, можливість створення візуально привабливих та зручних інтерфейсів та легкість інтеграції з різними MIDI пристроями.

Вибір багаторівневої архітектури дозволив створити якісне та масштабоване програмне рішення. Це забезпечило ефективну комунікацію між модулями, уникнення дублювань коду та зниження зайвих залежностей.

Побудова UML діаграм допомогла структурувати компоненти системи, визначити їх характеристики та шляхи взаємодії з іншими частинами системи. Це дозволило сформувати чітке уявлення про програмне рішення на етапі проєктування, що спростило розробку логіки та користувацького інтерфейсу.

Для перевірки правильності роботи програми було проведено тестування основних функціональних можливостей за допомогою різних методів тестування. Особлива увага приділялася обробці MIDI сигналів, що є критично важливим для правильного функціонування програми. Всі виявлені недоліки були усунуті, що дозволило забезпечити стабільну роботу програмного забезпечення.

Розроблене програмне забезпечення є важливим інструментом для музикантів та звукоінженерів, які працюють з MIDI пристроями. Воно забезпечує зручний та ефективний спосіб управління музичними контролерами, дозволяючи користувачам зосередитися на творчому процесі. Програма також може бути корисною для навчальних закладів, де вона може використовуватися як інструмент для навчання студентів роботі з MIDI пристроями.

У майбутньому передбачається розширення функціональних можливостей програми, включаючи підтримку нових типів MIDI пристроїв, інтеграцію з іншими музичними програмами та платформами, а також розробку мобільної версії програмного забезпечення. Це дозволить розширити коло користувачів та забезпечити їм додаткові можливості для творчості.

Таким чином, розроблене програмне рішення відповідає сучасним вимогам до програмних продуктів, забезпечує високу ефективність та надійність у роботі з MIDI пристроями, а також є зручним у використанні. Виконана робота демонструє важливість комплексного підходу до розробки програмного забезпечення, що включає аналіз, проєктування, реалізацію та тестування.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Guide of the C# language [Електронний ресурс] – Режим доступу до ресурсу: https://learn.microsoft.com/en-US/dotnet/csharp/tour-of-csharp/.

2. Overview of WPF .NET [Електронний ресурс] – Режим доступу до ресурсу: https://learn.microsoft.com/en-gb/dotnet/desktop/wpf/overview/?view=netdesktop-8.0.

3. Image-Line [Електронний ресурс] – Режим доступу до ресурсу: https://www.image-line.com/.

4 FL Studio [Електронний ресурс] – Режим доступу до ресурсу: https://www.image-line.com/fl-studio/.

5. About TestStack White [Електронний ресурс] – Режим доступу до ресурсу: https://teststackwhite.readthedocs.io/en/latest/

6. Introduction to MIDI Standart [Електронний ресурс] – Режим доступу до ресурсу: https://cecm.indiana.edu/361/midi.html.

7. Принципи SOLID. [Електронний ресурс] – Режим доступу до ресурсу: https://training.epam.ua/ua/blog/602.

8. Overview of Model-View-ViewModel (MVVM) [Електронний ресурс] – Режим доступу до ресурсу: https://learn.microsoft.com/en-gb/dotnet/architecture/maui/mvvm.

9. Test Scenario: Definition, Purpose, and How to Create [Електронний ресурс] – Режим доступу до ресурсу: http://surl.li/unxsj.

10. Git [Електронний ресурс] – Режим доступу до ресурсу: https://gitscm.com/.

11. БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ ТА ЦИВІЛЬНИЙ ЗАХИСТ / О. Г. Левченко, О. В. Землянська, Н. А. Праховнік, В. В. Зацарний. – Київ: КПІ ім. Ігоря Сікорського, 2019. – 53-56 с.

12. Безпека життєдіяльності / Т. Є.Стиценко, Г. В. Пронюк, Н. М. Сердюк, І. І. Хондак. – Харків: ХНУРЕ, 2018. – 67-69 с.

ДОДАТКИ

Лістинг 1 – файл MidiEngine

```
using Sanford.Multimedia.Midi;
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Linq;
using System.Text;
using System.Windows.Threading;


namespace MidiSynth7.components
{
    public class ErrorHandler
    {
        public static string ShowError(Exception error)
        {

            return error.ToString();
        }
    }

    public class MidiEngine
    {

        #region Declarations

        /// <summary>
        /// Gets whether or not the engine will use the internal
synthesizer (by fluidSynth)
        /// Deprecated until further notice
        /// </summary>
        public bool InternalSF2 { get; private set; }

        /// <summary>
        /// This returns the copyright information that is present in
the track.
        /// </summary>
        public string Copyright { get { return metaInfCr; } }

        public int SynthMainSF2 { get; set; }

        public    event    EventHandler<EventArgs>    MidiInitalized;
// When the midi was started.
        public     event     EventHandler<EventArgs>     MidiClosed;
// When the midi was stopped.
        public    event    EventHandler<NoteEventArgs>    NotePlayed;
// When a note has played.
        public    event    EventHandler<NoteEventArgs>    NoteStopped;
// when note stopped.
```

```
            public event EventHandler<EventArgs> FileLoadComplete;
            public              event              EventHandler<EventArgs>
SequenceBuilder_Completed;                              // The record build
worked and has completed.

            private                 OutputDevice                  _device;
// The output device of midi.
            public                  InputDevice                  inDevice;
// The input device (external keyboards etc)
            public                 InputDevice                 inDevice2;
// The input device (external keyboards etc)
            public               Sequencer               ModableSequencer;
// For Riff center
            public                  Sequence                 riffSequence;
// Loaded Riff Sequence
            public List<Sequence> presets = new List<Sequence>();

            private                Sequencer                midiSequencer;
// The sequencer that plays midi files.
            private                 Sequence                 midiSequence;
// Timer used for checking the loop status.
            private MidiInternalClock MiClock = new MidiInternalClock(6);
// The internal time keeper for midi recording.
            private            RecordingSession            recordSession;
// Recording session of the midi files.
            private BackgroundWorker buildWorker = new BackgroundWorker();
            private    BackgroundWorker    MetaParserWorker    =    new
BackgroundWorker();
            private List<(int AuxIndex, OutputDevice device)> OutDevices
= new List<(int AuxIndex, OutputDevice device)>();


            private int spb = -1;
            private int spp = -1;
            private int spc = -1;
            private int sbp = -1;
            private int sbc = -1;
            private bool gotMeta = false;
            private        bool        isRecording        =        false;
// used for the recording session.
            private   string   metaInfCr   =   "No   Copyright   information
available.";                        // meta information : Copyright
            private string RSTitle;
            private string RSAuthor;
            private string RSCopyright;
            private string RSComments;
            private string PresetDirectory = App.PRESET_DIR;

            #endregion

            #region EventHandles
            private void OnMidiInit(EventArgs e)
            {
                MidiInitalized?.Invoke(this, e);
            }
            private void OnNotePlay(NoteEventArgs e)
            {
```

```
            NotePlayed?.Invoke(this, e);
        }
        private void OnNoteStopped(NoteEventArgs e)
        {
            NoteStopped?.Invoke(this, e);
        }
        private void OnFileLoadComplete(EventArgs e)
        {
            FileLoadComplete?.Invoke(this, e);
        }
        private void OnMidiClose(EventArgs e)
        {
            MidiClosed?.Invoke(this, e);
        }
        private void OnSequenceBuildComplete(EventArgs e)
        {
            SequenceBuilder_Completed?.Invoke(this, e);
        }
        #endregion

        /// <summary>
        /// Initalizes the Musical Instrument Digital Interface for
use.
        /// </summary>
        ///    <param    name="DeviceIndex">The    midi    device    to
start.</param>
        public MidiEngine(int DeviceIndex)
        {
            try
            {
                OutDevices.Clear();
                Generatepresets();
                _device = new OutputDevice(DeviceIndex);
                OutDevices.Add((-1, _device));
                OnMidiInit(new EventArgs());
                GenerateSequencer();
                GenerateSequence();
                buildWorker.DoWork += BuildWorker_DoWork;
                buildWorker.RunWorkerCompleted                      +=
BuildWorker_RunWorkerCompleted;
                buildWorker.WorkerSupportsCancellation = false;
                MetaParserWorker.WorkerSupportsCancellation = false;
                MetaParserWorker.DoWork += MetaParserWorker_DoWork;
                MetaParserWorker.RunWorkerCompleted                 +=
MetaParserWorker_RunWorkerCompleted;
            }
            catch (Exception ex)
            {
                throw ex;
            }
        }

        #region Methods
        /// <summary>
        /// Export connected output device map
        /// </summary>
        /// <returns></returns>
```

```
            public List<(int aux,int device)> ExportOutDeviceMap()
            {
                List<(int aux, int device)> z = new List<(int aux, int
device)>();
                foreach (var item in OutDevices)
                {
                    if(item.AuxIndex != -1)
                    {
                        z.Add((item.AuxIndex,item.device.DeviceID));
                    }
                }
                return z;
            }

            /// <summary>
            /// Load in a device and assign it to aux. If the device is
already loaded, it's aux index is returned.
            /// If the aux id is in use, an exception is thrown.
            /// </summary>
            public int OpenOutputDevice(int aux,int deviceIndex)
            {
                if (aux == -1) throw new InvalidOperationException("-1 is
reserved for the engine default.");

                var f = OutDevices.Find(x => x.AuxIndex == aux);
                if(f.device != null)
                {
                    throw new InvalidOperationException("This index is
already in use");
                }

                OutputDevice device = OutDevices.FirstOrDefault(x =>
x.device.DeviceID == deviceIndex).device;

                if (device != null)
                {
                    return OutDevices.FirstOrDefault(x =>
x.device.DeviceID == deviceIndex).AuxIndex;
                }

                device = new OutputDevice(deviceIndex);
                int index = OutDevices.Count + 1;
                OutDevices.Add((OutDevices.Count + 1, device));
                return index;

            }

            /// <summary>
            /// returns a list of the available MIDI output devices
installed on your system.
            /// </summary>
            public static List<string> GetOutputDevices()
            {
                List<string> devlist = new List<string>();
                for (int i = 0; i < OutputDevice.DeviceCount; i++)
                {
```

```
devlist.Add(OutputDevice.GetDeviceCapabilities(i).name);
                }

                //ERROR
                if (devlist.Count() == 0)
                {
                    throw new Exception("No usable MIDI devices found.");
                }
                return devlist;
            }

            /// <summary>
            ///  Returns  a  list  of  the  avilable  MIDI  input  devices
installed on your system.
            /// </summary>
            public static List<string> GetInputDevices()
            {
                List<string> devlist = new List<string>();
                for (int i = 0; i < InputDevice.DeviceCount; i++)
                {

devlist.Add(InputDevice.GetDeviceCapabilities(i).name);
                }
                return devlist;
            }

            public static (string noteLabel, int octave) GetNote(int note,
string spacer = "")
            {
                string[]  noteString  =  new  string[]  {  "C"+spacer,  "C#",
"D" + spacer, "D#", "E" + spacer, "F" + spacer, "F#", "G" + spacer, "G#",
"A" + spacer, "A#", "B" + spacer };
                if (note < 0) note = 0;
                //if (note > 127) note = 127;
                int octave = (note / 12) - 1;
                if (octave < 0) octave = 0;
                if (octave > 9) octave = 9;
                int noteIndex = note % 12;
                string label = noteString[noteIndex];
                return (label, octave);
            }

            /// <summary>
            /// Returns the length of the sequence.
            /// </summary>
            public int GetLen() => midiSequence.GetLength();

            /// <summary>
            /// Returns the length of the sequence played so far.
            /// </summary>
            public int SeqPos() => midiSequencer.Position;

            /// <summary>
            /// Returns the number of tracks in the sequence.
            /// </summary>
            public int GetTrackCount() => midiSequence.Count();
```

```
            /// <summary>
            /// Save sequence as midi file.
            /// </summary>
            public void SaveSequence(string fileName)
            {
                midiSequencer.Sequence[0].Insert(0,                    new
MetaMessage(MetaType.TrackName, Encoding.Default.GetBytes(RSTitle)));
                midiSequencer.Sequence[0].Insert(1,                    new
MetaMessage(MetaType.Text, Encoding.Default.GetBytes(RSAuthor)));
                midiSequencer.Sequence[0].Insert(2,                    new
MetaMessage(MetaType.Copyright, Encoding.Default.GetBytes(RSCopyright)));
                midiSequencer.Sequence[0].Insert(3,                    new
MetaMessage(MetaType.Text, Encoding.Default.GetBytes(RSComments)));
                midiSequencer.Sequence.SaveAsync(fileName);
            }

            /// <summary>
            /// Plays a midi note with specific parameters.
            /// </summary>
            public void MidiNote_Play(int channel, int pitch, int volume,
bool send_event = true, int output=-1)
            {
                OutputDevice  device  =  OutDevices.FirstOrDefault(x  =>
x.AuxIndex == output).device;
                if (pitch > 127)
                {
                    return;
                }
                if (pitch < 0)
                {
                    return;
                }
                if (!InternalSF2)
                {
                    device.Send(new  ChannelMessage(ChannelCommand.NoteOn,
channel, pitch, volume));

                }
                if (isRecording)
                {
                    recordSession.Record(new
ChannelMessage(ChannelCommand.Controller,                          channel,
(int)ControllerType.Volume, volume));
                    recordSession.Record(new
ChannelMessage(ChannelCommand.NoteOn, channel, pitch, volume));
                }
                if(send_event)      OnNotePlay(new      NoteEventArgs(new
ChannelMessage(ChannelCommand.NoteOn, channel, pitch, volume)));
            }

            /// <summary>
            /// Stop the specific note.
            /// </summary>
            public  void  MidiNote_Stop(int  channel,  int  pitch,  bool
send_event = true, int output = -1)
            {
```

```
                OutputDevice device = OutDevices.FirstOrDefault(x =>
x.AuxIndex == output).device;
                if (pitch > 127)
                {
                    return;
                }
                if (pitch < 0)
                {
                    return;
                }
                if (!InternalSF2)
                {
                    device.Send(new ChannelMessage(ChannelCommand.NoteOff,
channel, pitch, 0));
                }

                if (isRecording)
                {
                    recordSession.Record(new
ChannelMessage(ChannelCommand.NoteOff, channel, pitch));
                }
                if(send_event)     OnNoteStopped(new     NoteEventArgs(new
ChannelMessage(ChannelCommand.NoteOff, channel, pitch, 0)));
            }

            /// <summary>
            /// Set the instrument(Patch)
            /// </summary>
            public void MidiNote_SetProgram(int bank, int patch, int ch,
int output=-1)
            {
                OutputDevice device = OutDevices.FirstOrDefault(x =>
x.AuxIndex == output).device;
                if (!InternalSF2)
                {
                    device.Send(new
ChannelMessage(ChannelCommand.Controller,                                ch,
(int)ControllerType.BankSelect, bank));
                    device.Send(new
ChannelMessage(ChannelCommand.ProgramChange, ch, patch));
                }
                if ((isRecording))
                {
                    if (spb != bank && spc != ch && spp != patch)//case 1
- none are equal.
                    {
                        recordSession.Record(new
ChannelMessage(ChannelCommand.Controller,                                ch,
(int)ControllerType.BankSelect, bank));
                        recordSession.Record(new
ChannelMessage(ChannelCommand.ProgramChange, ch, patch));
                    }
                    if ((spb == bank && spc != ch && spp != patch) ||
(spb != bank && spc == ch && spp != patch) || (spb != bank && spc != ch &&
spp == patch))//case 2 - none but one are equal
                    {
```

```
                        recordSession.Record(new
ChannelMessage(ChannelCommand.Controller,                                   ch,
(int)ControllerType.BankSelect, bank));
                        recordSession.Record(new
ChannelMessage(ChannelCommand.ProgramChange, ch, patch));
                    }


                    if ((spb == bank && spc == ch && spp != patch) ||
(spb == bank && spc != ch && spp == patch) || (spb != bank && spc == ch &&
spp == patch))//case 3 - any two are equal
                    {
                        recordSession.Record(new
ChannelMessage(ChannelCommand.Controller,                                   ch,
(int)ControllerType.BankSelect, bank));
                        recordSession.Record(new
ChannelMessage(ChannelCommand.ProgramChange, ch, patch));
                    }


                    spb = bank;
                    spp = patch;
                    spc = ch;
                }
            }


            #endregion

            #endregion
        }


        #region Custom event args
        public class NoteEventArgs : EventArgs
        {
            ChannelMessage cm;
            bool sq = false;
            public NoteEventArgs(ChannelMessage chMsg)
            {
                cm = chMsg;
            }
            public NoteEventArgs(ChannelMessage chMsg, bool isSequence)
            {
                cm = chMsg;
                sq = isSequence;
            }
            public ChannelMessage ChannelMssge
            {
                get
                {
                    return cm;
                }
            }
            public bool Sequence
            {
                get
```

```
                {
                    return sq;
                }
            }
        }

        #endregion
    }
```

Лістинг 2 – файл TrackerSequence та Seq data

```
    using System;
    using System.Collections.Generic;
    using Sanford.Multimedia.Midi;
    using System.IO;
    using System.Linq;
    using Newtonsoft.Json;
    using System.Windows.Media;
    using System.Windows;
    using System.Windows.Input;
    using System.Diagnostics;

    namespace MidiSynth7.components
    {

        public class TrackerSequence
        {
            string path = App.APP_DATA_DIR + "Sequences\\";
            string name = "";

            public string SequenceName
            {
                get=>name;
                set
                {
                    if (string.IsNullOrWhiteSpace(value))
                    {
                        name = "Untitled Sequence";
                    }
                    string                    safeSeqName                    =
Path.GetInvalidFileNameChars().Aggregate(value,     (current,     c)     =>
current.Replace(c, '-'));
                    string                 safePrvSeqName                 =
Path.GetInvalidFileNameChars().Aggregate(name,     (current,     c)     =>
current.Replace(c, '-'));

                    if   (File.Exists(path   +   safeSeqName+".mton")   &&
value       !=       name       &&       !string.IsNullOrWhiteSpace(name)
&& !string.IsNullOrWhiteSpace(value))
                    {
                        throw  new  Exception("A  sequence  with  this  name
already exists.");

                    }
```

```
                    if(File.Exists(path + safePrvSeqName + ".mton"))
                    {
                        File.Move(path + safePrvSeqName + ".mton", path +
safeSeqName + ".mton");
                    }
                    name = value;
                }
            }

            internal     List<ISeqParamPlayer>     Players     =     new
List<ISeqParamPlayer>();

            public List<TrackerPattern> Patterns { get; set; }

            public List<TrackerInstrument> Instruments { get; set; }

            public int SelectedOctave { get; set; }

            public int SelectedInstrument{ get; set; }

            public TrackerSequence()
            {
                AddParameterPlayer(new seqparams.Sxx());
            }

            public void AddParameterPlayer(ISeqParamPlayer paramPlayer)
            {
                ISeqParamPlayer   check   =   Players.FirstOrDefault(x   =>
x.Mark == paramPlayer.Mark);
                if (check != null)
                {
                    throw   new   InvalidOperationException($"A   parameter
player with the designated mark '{check.Mark}' already exists.");
                }
                Players.Add(paramPlayer);
            }

            public void SaveSequence(string oldname = "")
            {
                foreach (TrackerPattern item in Patterns)
                {
                    int r = 0;
                    foreach (TrackerRow row in item.Rows)
                    {
                        foreach (SeqData data in row.Notes)
                        {
                            if(data.Row != r)
                            {
                                data.Row = r;
                            }
                        }
                        r++;
                    }
                }
                if(!Directory.Exists(path))
                {
                    Directory.CreateDirectory(path);
```

```
                }
                string                      safeSeqName                      =
Path.GetInvalidFileNameChars().Aggregate(SequenceName,   (current,   c)   =>
current.Replace(c, '-'));

                if (!string.IsNullOrWhiteSpace(oldname))
                {
                    string                safeoldSeqName                =
Path.GetInvalidFileNameChars().Aggregate(oldname,    (current,    c)    =>
current.Replace(c, '-'));
                    if (File.Exists(path + safeoldSeqName + ".mton"))
                    {
                        File.Delete(path + safeoldSeqName + ".mton");
                    }
                }
                if (File.Exists(path + safeSeqName+".mton"))
                {
                    File.Delete(path + safeSeqName + ".mton");
                }
                using        (StreamWriter        sw        =        new
StreamWriter(path+safeSeqName + ".mton"))
                {
                    sw.Write(JsonConvert.SerializeObject(this,
Formatting.Indented));
                }
            }

            public override string ToString()
            {
                return SequenceName;
            }
        }

        public class TrackerPattern
        {
            public int RowsPerMeasure { get; set; }
            public int RowsPerBeat { get; set; }
            public int RowCount { get; set; }
            public int ChannelCount { get; set; }
            public string PatternName { get; set; }
            public List<TrackerRow> Rows { get; set; }

            public  static  TrackerPattern  GetEmptyPattern(int  rows,  int
channels)
            {
                return new TrackerPattern()
                {
                    ChannelCount = channels,
                    RowCount = rows,
                    PatternName = "New Pattern",
                    Rows = TrackerRow.GetEmptyRows(rows, channels),
                    RowsPerBeat = 4,
                    RowsPerMeasure = 16
                };
            }

            public override string ToString()
```

```
            {
                return $"RowCount: {RowCount} | Pattern Name:
{PatternName} | Actual row count: {Rows.Count}";
            }
        }

        public class TrackerRow
        {
            internal Rect rowbounds;
            internal DrawingGroup RowRender = new DrawingGroup();

            public List<SeqData> Notes { get; set; }

            public TrackerRow(List<SeqData> notes)
            {
                Notes = notes;
                rowbounds = new Rect(0, 22, 0, 22);
            }

            public static List<TrackerRow> GetEmptyRows(int rows, int
ChannelCount)
            {
                List<TrackerRow> Rows = new List<TrackerRow>();
                for (int i = 0; i < rows; i++)
                {
                    TrackerRow row = new TrackerRow(new List<SeqData>());

                    for (int ii = 0; ii < ChannelCount; ii++)
                    {
                        int midiChannel = GetMIDIChannelIndex(ii);
                        row.Notes.Add(new SeqData()
                        {
                            Row = i,
                            Column = ii,
                            Instrument = null,
                            midiChannel = midiChannel,
                            Parameter = null,
                            Pitch = null,
                            Velocity = null

                        });
                    }
                    row.rowbounds = new Rect(0, row.Notes[0].Row * 22,
SeqData.Width * row.Notes.Count, 22);
                    Rows.Add(row);
                }
                return Rows;
            }

            private static int GetMIDIChannelIndex(int column)
            {
                int midiChannel = column;
                if (column >= 9)
                {
                    midiChannel = column + 1;
                }
                if (column + 1 > 15)
```

```
                {
                    midiChannel = 9;
                }

                return midiChannel;
            }



            public void Play(TrackerSequence seq, MidiEngine engine)
            {

                foreach (SeqData item in Notes)
                {
                    if (item.Parameter != null)
                    {
                        ISeqParamPlayer              Player              =
seq.Players.FirstOrDefault(x => x.Mark == item.Parameter.Mark);
                        if (Player != null)
                        {
                            Player.Play(item, engine, seq);
                            continue;
                        }
                    }
                    TrackerInstrument                 ti                 =
seq.Instruments.FirstOrDefault(x => x.Index == item.Instrument);
                    int output = ti?.DeviceIndex ?? -1;

                    if(ti != null)
                    {
                        engine.MidiNote_SetProgram(ti.Bank,  ti.Instrument,
item.midiChannel, output);
                        if(item.Pitch.HasValue)
                        {
                            if(item.midiChannel != 9)
                            {

engine.MidiNote_SetControl(ControllerType.AllNotesOff,    item.midiChannel,
127, output);//in theory just for the channel?
                            }
                            if(item.Pitch > -1)
                            {
                                engine.MidiNote_Play(item.midiChannel,
item.Pitch.Value, item.Velocity ?? 127, false, output);
                            }

                        }
                    }
                    if (item.Pitch.HasValue)
                    {

                        if (item.Pitch == -2)
                        {

engine.MidiNote_SetControl(ControllerType.AllSoundOff,    item.midiChannel,
127, output);//in theory just for the channel?
                        }
```

```
                        if (item.Pitch == -1)
                        {

engine.MidiNote_SetControl(ControllerType.AllNotesOff,    item.midiChannel,
127, output);//in theory just for the channel?
                        }
                    }
                }
            }

            public void UpdateRow(int index, SolidColorBrush back, bool
hot=false, bool ignorebits = false, Rect? debugrect=null)
            {
                var dc = RowRender.Open();
                dc.DrawRectangle(back, null, new Rect(0, index * 22, 126
* Notes.Count, 22));

                dc.Close();
                for (int i = 0; i < Notes.Count; i++)
                {
                    if (!ignorebits) Notes[i].UpdateBit(hot);
                }
            }

            public void Render(DrawingContext dc)
            {
                dc.DrawDrawing(RowRender);
                foreach (var item in Notes)
                {
                    item.Render(dc);
                }
            }

            internal   bool   DetectSelection(Rect   Selection,   out   int
cellIndex, out int bitIndex, int active=0 )
            {
                cellIndex = 0;
                bitIndex = 0;
                var         sel         =         Notes.Where(x         =>
x.SqDatBit_Bounds.IntersectsWith(Selection));
                foreach (SeqData item in sel)
                {
                    item.DetectSelection(Selection,out bitIndex,active);
                }

                cellIndex = sel.ToList()[0].Column;

                return Selection.Width > 2 || Selection.Height > 2;
            }
        }

        public class SeqData
        {
            #region Data Properties
            public int midiChannel;
            public int? Pitch { get; set; }
            public byte? Velocity { get; set; }
```

```
            public byte? Instrument { get; set; }
            public SeqParam Parameter { get; set; }
            #endregion

            #region Rendering Properties
            internal DrawingGroup Renderer = new DrawingGroup();
            internal bool[] SelectedBits = new bool[] { false, false,
false, false, false };
            private string[] keybuffers = new string[] { "", "", "", "",
"" };

            public static readonly double[] BitWidths = new double[]
            {
                34,24,34,10,20
            };
            public static readonly int[] BitOffsets = new int[]
            {
                0,34,58,92,102
            };
            internal Rect SqDatBit_Bounds;  // overall dimensions
            internal Rect SqTxtBit_Bounds;  // text dimensions
            private List<Rect> boundList = new List<Rect>();
            private   (string   text,   Brush   FG,   GlyphRun   cached)?[]
cachedRuns = new (string text, Brush FG, GlyphRun cached)?[5];
            private DrawingGroup[] txtrender = new DrawingGroup[5]
            {
                new DrawingGroup(),
                new DrawingGroup(),
                new DrawingGroup(),
                new DrawingGroup(),
                new DrawingGroup()
            };

            public static readonly int Width = 126;
            public static readonly int Height = 22;

            public     static     SolidColorBrush     BR_Empty     =     new
SolidColorBrush(Color.FromArgb(255, 035, 067, 103));     // blank and some
params
            public     static     SolidColorBrush     BR_HotFG     =     new
SolidColorBrush(Color.FromArgb(255, 223, 236, 255));    // Active row text
            public     static     SolidColorBrush     BR_SelBG     =     new
SolidColorBrush(Color.FromArgb(255,  032,  057,  097));       //  Selection
Background
            public     static     SolidColorBrush     BR_SelFG     =     new
SolidColorBrush(Color.FromArgb(255,  089,  149,  231));       //  Selection
Foreground
            public     static     SolidColorBrush     BR_Pitch     =     new
SolidColorBrush(Color.FromArgb(255, 223, 236, 255));    // Unselected Pitch
            public     static     SolidColorBrush     BR_Patch     =     new
SolidColorBrush(Color.FromArgb(255, 255, 133, 128));    // Unselected Patch
            public     static     SolidColorBrush     BR_Veloc     =     new
SolidColorBrush(Color.FromArgb(255,  128,  225,  139));       //  Unselected
Velocity
            public     static     SolidColorBrush     BR_AParV     =     new
SolidColorBrush(Color.FromArgb(255, 137, 185, 247));    // 'A' parameter
```

```
            public    static    SolidColorBrush    BR_BdrEn    =    new
SolidColorBrush(Color.FromArgb(255, 012, 016, 020));    // Dark border
            public    static    SolidColorBrush    BR_BdrEx    =    new
SolidColorBrush(Color.FromArgb(255, 039, 070, 120));    // Light border

            public int Column { get; set; }
            public int Row { get; set; }

            #endregion

            public override string ToString()
            {
                string note = "...";
                if(Pitch.HasValue)
                {

                    var n = MidiEngine.GetNote(Pitch.Value, "-");
                    note = n.noteLabel + n.octave;
                    if(Pitch.Value == -1)
                    {
                        note = "===";
                    }
                    if (Pitch.Value == -2)
                    {
                        note = "~~~";
                    }
                    if (Pitch.Value == -3)
                    {
                        note = "^^^";
                    }
                }
                string instindex = "..";
                if(Instrument != null)
                {
                    instindex = Instrument.ToString();
                }
                string velocity = "...";
                if (Velocity.HasValue)
                {
                    velocity = "v" + (Velocity.Value/2);
                }
                string param = "...";
                if(Parameter != null)
                {
                    param       =       Parameter.Mark.ToString()       +
Parameter.Value.ToString("X:2");
                }
                return "|"+note + instindex + velocity + param;
            }

            public void Render(DrawingContext dc)
            {
                dc.DrawDrawing(Renderer);
            }

            public void UpdateBit(bool hot = false, Rect? debugsel = null)
            {
```

```
                boundList.Clear();
                double offset = 0;
                SqDatBit_Bounds = new Rect(Column * Width, Row * Height,
Width, Height);
                SqTxtBit_Bounds = new Rect((Column * Width) + 2, (Row *
Height) + 1, Width - 6, Height - 2);
                Rect PitchBit_Bounds = new Rect(SqTxtBit_Bounds.X,
SqDatBit_Bounds.Y, BitWidths[0], Height);
                offset += PitchBit_Bounds.Width;
                Rect InstrBit_Bounds = new Rect(SqTxtBit_Bounds.X +
offset, SqDatBit_Bounds.Y, BitWidths[1], Height);
                offset += InstrBit_Bounds.Width;
                Rect VelocBit_Bounds = new Rect(SqTxtBit_Bounds.X +
offset, SqDatBit_Bounds.Y, BitWidths[2], Height);
                offset += VelocBit_Bounds.Width;
                Rect SqParBit_Bounds = new Rect(SqTxtBit_Bounds.X +
offset, SqDatBit_Bounds.Y, BitWidths[3], Height);
                offset += SqParBit_Bounds.Width;
                Rect SqValBit_Bounds = new Rect(SqTxtBit_Bounds.X +
offset, SqDatBit_Bounds.Y, BitWidths[4], Height);
                boundList.Add(PitchBit_Bounds);
                boundList.Add(InstrBit_Bounds);
                boundList.Add(VelocBit_Bounds);
                boundList.Add(SqParBit_Bounds);
                boundList.Add(SqValBit_Bounds);
                var rc = Renderer.Open();

                for (int i = 0; i < SelectedBits.Length; i++)
                {
                    Brush bg = SelectedBits[i] ? BR_SelBG : null;
                    if (bg != null)
                    {
                        rc.DrawRectangle(bg, null, boundList[i]);
                    }

                    UpdateBitText(i,hot);
                    rc.DrawDrawing(txtrender[i]);
                }
                Point tr2 = SqDatBit_Bounds.TopRight;
                Point br2 = SqDatBit_Bounds.BottomRight;

                rc.DrawLine(new Pen(BR_BdrEn, 4), tr2, br2);
                rc.DrawLine(new             Pen(BR_BdrEx,             2),
SqDatBit_Bounds.TopRight, SqDatBit_Bounds.BottomRight);
                if (debugsel.HasValue)
                {
                    rc.DrawRectangle(Brushes.Red, null, debugsel.Value);
                }
                rc.Close();
            }


            private GlyphRun GlyphText(string text, Point origin)
            {
                if(SystemComponent.MPTGlyphTypeFace == null)
                {
```

```
                    bool                          v                          =
SystemComponent.MPTEditorFont.TryGetGlyphTypeface(out
SystemComponent.MPTGlyphTypeFace);
                    if (!v)
                    {
                        throw  new  InvalidOperationException("Cannot  get
typeface!");
                    }
                }

                double fontSize = 16;
                ushort[] glyphIndexes = new ushort[text.Length];
                double[] advanceWidths = new double[text.Length];

                double totalWidth = 0;
                for (int n = 0; n < text.Length; n++)
                {
                    ushort glyphIndex;

SystemComponent.MPTGlyphTypeFace.CharacterToGlyphMap.TryGetValue(text[n],
out glyphIndex);
                    glyphIndexes[n] = glyphIndex;
                    double                      width                      =
SystemComponent.MPTGlyphTypeFace.AdvanceWidths[glyphIndex] * fontSize;
                    advanceWidths[n] = width;
                    totalWidth += width;
                }

                GlyphRun              run              =              new
GlyphRun(SystemComponent.MPTGlyphTypeFace,
                                        bidiLevel: 0,
                                        isSideways: false,
                                        renderingEmSize: fontSize,
                                        pixelsPerDip: 1,
                                        glyphIndices: glyphIndexes,
                                        baselineOrigin: origin,
                                        advanceWidths: advanceWidths,
                                        glyphOffsets: null,
                                        characters: null,
                                        deviceFontName: null,
                                        clusterMap: null,
                                        caretStops: null,
                                        language: null);

                return run;
            }
            internal   bool   DetectSelection(Rect   Selection,   out   int
activeBit, int active=0)
            {
                activeBit = 0;
                var lb = new bool[5];
                SelectedBits.CopyTo(lb, 0);
                for (int i = 0; i < SelectedBits.Length; i++)
                {
                    SelectedBits[i]                                        =
boundList[i].IntersectsWith(Selection);
                }
```

```
                if (!Enumerable.SequenceEqual(lb, SelectedBits))
                {
                    UpdateBit(Row == active);
                }
                //scroll to active corner

                activeBit = SelectedBits.ToList().IndexOf(true);
                return Selection.Width > 2 || Selection.Height > 2;
            }

            internal void ClearSelection(int activeRow=0)
            {
                var lb = new bool[5];
                SelectedBits.CopyTo(lb, 0);
                SelectedBits = new bool[] { false, false, false, false,
false };
                if (!Enumerable.SequenceEqual(lb, SelectedBits))
                {

                    UpdateBit(Row == activeRow);
                }
            }

            internal void UpdateBitText(int i, bool hot)
            {

                Brush fg = BR_Empty;
                string text = "...";
                switch (i)
                {
                    case 0:
                        fg    =    SelectedBits[i]    ?    BR_SelFG    :
(Pitch.HasValue ? BR_Pitch : BR_Empty);
                        if (Pitch.HasValue)
                        {
                            var note = MidiEngine.GetNote(Pitch.Value, "-
");
                            text = note.noteLabel + note.octave;
                            if (Pitch == -1)
                            {
                                fg   =   SelectedBits[i]   ?   BR_SelFG   :
BR_Empty;
                                text = "== ";
                            }
                            if (Pitch == -2)
                            {
                                fg   =   SelectedBits[i]   ?   BR_SelFG   :
BR_Empty;
                                text = "~~ ";
                            }
                            if (Pitch == -2)
                            {
                                fg   =   SelectedBits[i]   ?   BR_SelFG   :
BR_Empty;
                                text = "^^ ";
                            }
                        }
```

```
                        break;
                    case 1:
                        fg    =    SelectedBits[i]    ?    BR_SelFG    :
(Instrument.HasValue ? BR_Patch : BR_Empty);
                        text = "..";
                        if (Instrument.HasValue)
                        {
                            text = Instrument.Value.ToString("X2");

                        }
                        break;
                    case 2:
                        fg    =    SelectedBits[i]    ?    BR_SelFG    :
(Velocity.HasValue ? BR_Veloc : BR_Empty);
                        text = " ..";
                        if (Velocity.HasValue)
                        {
                            text = "v" + Velocity.Value.ToString("X2");
                        }
                        break;
                    case 3:
                        text = ".";
                        fg    =    SelectedBits[i]    ?    BR_SelFG    :
(Parameter?.Mark == 'A' ? BR_AParV : BR_Empty);
                        if (Parameter != null)
                        {
                            text = Parameter.Mark.ToString();
                        }
                        break;
                    case 4:
                        text = "..";

                        fg    =    SelectedBits[i]    ?    BR_SelFG    :
(Parameter?.Mark == 'A' ? BR_AParV : BR_Empty);
                        if (Parameter != null)
                        {
                            text = Parameter.Value.ToString("X2");
                        }
                        break;
                    default: break;
                }
                if (hot && fg == BR_Empty)
                {
                    fg = BR_Pitch;
                }
                if(cachedRuns[i] == null)
                {
                    cachedRuns[i]   =   (text,fg,   GlyphText(text,   new
Point(boundList[i].TopLeft.X + 2, boundList[i].TopLeft.Y + 16)));
                    var rc = txtrender[i].Open();
                    rc.DrawGlyphRun(fg, cachedRuns[i].Value.cached);
                    rc.Close();
                }
                if (cachedRuns[i]?.text != text)
                {
                    cachedRuns[i]   =   (text,fg,   GlyphText(text,   new
Point(boundList[i].TopLeft.X + 2, boundList[i].TopLeft.Y + 16)));
```

```
                    var rc = txtrender[i].Open();
                    rc.DrawGlyphRun(fg,cachedRuns[i].Value.cached);
                    rc.Close();
                }

                if (cachedRuns[i]?.FG != fg)
                {
                    cachedRuns[i]  =  (text,  fg,  GlyphText(text,  new
Point(boundList[i].TopLeft.X + 2, boundList[i].TopLeft.Y + 16)));
                    var rc = txtrender[i].Open();
                    rc.DrawGlyphRun(fg, cachedRuns[i].Value.cached);
                    rc.Close();
                }

                //rc.DrawText(Text(text,             fg),             new
Point(boundList[i].TopLeft.X + 2, boundList[i].TopLeft.Y));

                //rc.Close();

            }

            internal void DeleteBitInfo(bool hot = false)
            {
                for (int i = 0; i < SelectedBits.Length; i++)
                {
                    if (SelectedBits[i])
                    {
                        switch (i)
                        {
                            case 0://note
                                Pitch = null;
                                break;
                            case 1://instrument
                                Instrument = null;
                                break;
                            case 2://velocity
                                if(SelectedBits[1]  ||  Instrument  ==  null
|| Pitch == null)
                                {
                                    Velocity = null;
                                }
                                else
                                {
                                    Velocity = 127;
                                }
                                break;
                            case 3://parammark
                                Parameter = null;
                                break;
                            case 4://paramvalue
                                if (SelectedBits[3])
                                {
                                    Parameter = null;
                                } else
                                {
                                    Parameter.Value = 0;
                                }
```

```
                            break;
                        default:
                            break;
                    }
                }
            }

            UpdateBit(hot);
        }

        internal void ProcessBitKey(int octave, byte patchindex, Key
key, bool hot = false)
        {
            int i = Array.IndexOf(SelectedBits, true);
            switch (i)
            {
                case 0:
                    int                    index                    =
Array.IndexOf(SystemComponent.MPTKeysTable, key);
                    if(index > -1)
                    {
                        Pitch = index + 21 + (12 * octave);
                        Instrument = patchindex;
                        if (Velocity == null)
                        {
                            Velocity = 127;
                        }
                    }
                    if(key == KeyInterop.KeyFromVirtualKey(187))
                    {
                        Pitch = -1;
                        Instrument = null;
                        Velocity = null;
                    }
                    break;
                case 1:
                    //instrument editor
                    if (key == KeyInterop.KeyFromVirtualKey(187))
                    {
                        Pitch = -1;
                        Instrument = null;
                        Velocity = null;
                        break;
                    }
                    Instrument = ParseInputAsHex(key, i);
                    break;
                case 2:
                    Velocity = ParseInputAsHex(key, i);
                    if(Velocity > 127)
                    {
                        Velocity = 127;
                    }
                    break;
                case 3:
                    break;
                case 4:
                    break;
```

```
                    default:
                        break;
                }
                UpdateBit(hot);
            }

            internal void ProcessMIDIMessage(ChannelMessage m, byte
patchindex, bool hot=false)
            {
                if(m.Command == ChannelCommand.NoteOn && m.Data2 > 0)
                {
                    Pitch = m.Data1;
                    Instrument = patchindex;
                    Velocity = (byte)m.Data2;
                    UpdateBit(hot);
                }
                if ((m.Command == ChannelCommand.NoteOff || (m.Command ==
ChannelCommand.NoteOn && m.Data2 == 0)) && Pitch == null)
                {
                    Pitch = -1;
                    Instrument = null;
                    Velocity = null;
                    UpdateBit(hot);
                }


            }

            private byte ParseInputAsHex(Key key, int i)
            {
                if (keybuffers.Length < 2)
                {
                    keybuffers[i] = keybuffers[i].PadLeft(2, '0');
                }
                string prv = keybuffers[i];
                keybuffers[i] += SystemComponent.GetCharFromKey(key);
                if (keybuffers[i].Length > 2)
                {
                    keybuffers[i] = keybuffers[i].Remove(0, 1);
                }
                //try parse hex

                if                          (byte.TryParse(keybuffers[i],
System.Globalization.NumberStyles.HexNumber, null, out byte res))
                {
                    return res;
                }
                else
                {
                    keybuffers[i] = prv;
                    if                      (byte.TryParse(keybuffers[i],
System.Globalization.NumberStyles.HexNumber, null, out byte fallback))
                    {
                        return fallback;
                    }
                }
                return 0;
```

```
            }
        }

        public class TrackerInstrument
        {
            public byte Index { get; set; }
            public int DeviceIndex { get; set; }
            public int Bank { get; set; }
            public int Instrument { get; set; }
            public string DisplayName { get; set; }

            public TrackerInstrument(byte index,int device,int bank, int
instrument, string name)
            {
                Index = index;
                DeviceIndex = device;
                Bank = bank;
                Instrument = instrument;
                DisplayName = name;

            }

            public override string ToString()
            {
                return $"{(int)Index:00}: {DisplayName}";
            }
        }

        public class SeqParam
        {
            public char Mark { get; set; }

            public byte Value { get; set; }
        }
    }
```

Лістинг 3 – файл StudioView

```
    using Microsoft.Win32;
    using MidiSynth7.entities.controls;
    using System;
    using System.Collections.Generic;
    using System.Linq;
    using System.Threading.Tasks;
    using System.Windows;
    using System.Windows.Controls;
    using System.Windows.Input;
    using System.Windows.Media;
    using System.Windows.Shapes;

    namespace MidiSynth7.components.views
    {
        /// <summary>
        /// Interaction logic for StudioView.xaml
```

```
        /// </summary>

        public partial class StudioView : Page, ISynthView
        {
            SystemConfig Config;
            MidiEngine MidiEngine;
            MainWindow AppContext;
            private int Transpose;
            private int RiffKey;
            private bool mfile_playing = false;
            private List<Ellipse> channelElipses = new List<Ellipse>();
            private   List<MainWindow.ChInvk>   channelIndicators   =   new
List<MainWindow.ChInvk>();
            int pattern = 0;
            int step = 0;
            public bool HaltKeyboardInput { get; private set; }

            public StudioView(MainWindow context, ref SystemConfig config,
ref MidiEngine engine)
            {
                InitializeComponent();
                Config = config;
                MidiEngine = engine;
                AppContext = context;
                cb_Devices.Items.Clear();


                if (Config.ActiveOutputDeviceIndex == -1)
                {
                    Config.ActiveOutputDeviceIndex = 0;
                }

                int midiOutIndex = 0;
                AppContext.OutputDevices.Clear();
                foreach (string item in MidiEngine.GetOutputDevices())
                {
                    AppContext.OutputDevices.Add(new
NumberedEntry(midiOutIndex, item));
                    midiOutIndex++;
                }
                foreach (NumberedEntry item in AppContext.OutputDevices)
                {
                    cb_Devices.Items.Add(item);
                }

                if            (Config.ActiveOutputDeviceIndex            <
cb_Devices.Items.Count)
                {
                    cb_Devices.SelectedIndex                                =
Config.ActiveOutputDeviceIndex;
                }

                Transpose = Config.PitchOffsets[1];
                CB_EnforceInstruments.IsChecked                             =
config.EnforceInstruments;
                pianomain.SetNoteText(Transpose);
                #region Controls
```

```
                CTRL_Chorus.Value = Config.ChannelChoruses[0];
                CTRL_Modulation.Value = Config.ChannelModulations[0];
                CTRL_Balance.Value = Config.ChannelPans[0];
                CTRL_Octave.Value = Config.PitchOffsets[0];
                CTRL_Reverb.Value = Config.ChannelReverbs[0];
                CTRL_Volume.Value = Config.ChannelVolumes[0];
                UpdateMIDIControls();
                #endregion

                #region Patch & banks

                UpdateInstrumentSelection(config);
                #endregion

                NFXProfileUpdate();

                #region ADD ellipses
                AppContext.channelIndicators.Clear();
                AppContext.channelElipses.Clear();
                AppContext.channelElipses.Add(ch1);
                AppContext.channelElipses.Add(ch2);
                AppContext.channelElipses.Add(ch3);
                AppContext.channelElipses.Add(ch4);
                AppContext.channelElipses.Add(ch5);
                AppContext.channelElipses.Add(ch6);
                AppContext.channelElipses.Add(ch7);
                AppContext.channelElipses.Add(ch8);
                AppContext.channelElipses.Add(ch9);
                AppContext.channelElipses.Add(ch10);
                AppContext.channelElipses.Add(ch11);
                AppContext.channelElipses.Add(ch12);
                AppContext.channelElipses.Add(ch13);
                AppContext.channelElipses.Add(ch14);
                AppContext.channelElipses.Add(ch15);
                AppContext.channelElipses.Add(ch16);
                foreach (Ellipse item in AppContext.channelElipses)
                {
                    AppContext.channelIndicators.Add(new
MainWindow.ChInvk(item, AppContext));
                }
                #endregion


                Cb_SequencerProfile.ItemsSource = AppContext.Tracks;
                Cb_SequencerProfile.SelectedIndex = 0;
            }

            private void UpdateInstrumentSelection(SystemConfig config)
            {
                cb_mBank.Items.Clear();
                cb_dkitlist.Items.Clear();
                foreach             (Bank             item             in
AppContext.ActiveInstrumentDefinition.Banks.Where(xb => xb.Index != 127))
                {
                    cb_mBank.Items.Add(item);
                 }
```

```
                foreach (NumberedEntry item in
AppContext.ActiveInstrumentDefinition.Banks.FirstOrDefault(xb => xb.Index
== 127)?.Instruments)
                {
                    cb_dkitlist.Items.Add(item);
                }
                cb_mBank.SelectedIndex = 0;
                cb_dkitlist.SelectedIndex = 0;
                config.ChannelBanks[0] = (cb_mBank.Items.Count >=
config.ChannelBanks[0]) ? config.ChannelBanks[0] : 0;
                config.ChannelInstruments[0] = (cb_mPatch.Items.Count >=
config.ChannelInstruments[0]) ? config.ChannelInstruments[0] : 0;
                config.ChannelInstruments[9] =
(cb_dkitlist.Items.Count >= config.ChannelInstruments[9]) ?
config.ChannelInstruments[9] : 0;

                cb_mBank.SelectedIndex = config.ChannelBanks[0];

                cb_mPatch.SelectedIndex = config.ChannelInstruments[0];

                cb_dkitlist.SelectedIndex = config.ChannelInstruments[9];
            }

            async Task FlashChannelActivity(int index)
            {
                Action invoker = delegate ()
                {
                    AppContext.channelElipses[index].Fill =
(Brush)FindResource("CH_IND_ON");
                    AppContext.channelIndicators[index].CounterReset();
                };
                await Dispatcher.BeginInvoke(invoker);
            }

            private void UpdateMIDIControls()
            {

                if (MidiEngine != null)
                {
                    for (int i = 0; i < 16; i++)
                    {
                        MidiEngine.MidiNote_SetReverb(i,
CTRL_Reverb.Value);
                        MidiEngine.MidiNote_SetChorus(i,
CTRL_Chorus.Value);
                        MidiEngine.MidiNote_SetModulation(i,
CTRL_Modulation.Value);
                        MidiEngine.MidiNote_SetPan(i, CTRL_Balance.Value);

MidiEngine.MidiNote_SetControl(Sanford.Multimedia.Midi.ControllerType.Volum
e, i, CTRL_Volume.Value);
                    }

                }

            }
```

```
        private void Cb_Devices_SelectionChanged(object sender,
SelectionChangedEventArgs e)
        {

            Config.ActiveOutputDeviceIndex =
cb_Devices.SelectedIndex;
            AppContext.GenerateMIDIEngine(this,
((NumberedEntry)cb_Devices.SelectedItem).Index);

            AppContext.ActiveInstrumentDefinition =
AppContext.Definitions.FirstOrDefault(x => x.AssociatedDeviceIndex ==
(((NumberedEntry)cb_Devices.SelectedItem)?.Index ?? -1)) ??
AppContext.Definitions[0];//associated or default
            UpdateInstrumentSelection(AppContext.AppConfig);

        }

        private async void MIO_bn_stop_Click(object sender,
RoutedEventArgs e)
        {
            MidiEngine?.MidiEngine_Panic();
            await invokeUnLightRange(0, pianomain.KeyCount + 21);
        }

        private void ToggleChecked(object sender, RoutedEventArgs e)
        {
            gb_ds.IsEnabled = cb_DS_Enable.IsChecked.Value;
            if (Config != null) Config.EnforceInstruments =
CB_EnforceInstruments?.IsChecked ?? true;
        }



        private void CTRL_ValueChanged(object sender, EventArgs e)
        {
            UpdateMIDIControls();
            if (Config != null)
            {
                Config.ChannelReverbs[0] = CTRL_Reverb.Value;
                Config.ChannelVolumes[0] = CTRL_Volume.Value;
                Config.ChannelModulations[0] = CTRL_Modulation.Value;
                Config.ChannelChoruses[0] = CTRL_Chorus.Value;
                Config.ChannelPans[0] = CTRL_Balance.Value;
                Config.PitchOffsets[0] = CTRL_Octave.Value;//global
octave

            }
        }

        private void OFX_bn_Edit_Click(object sender, RoutedEventArgs
e)
        {

        }
```

```
            private    void    BN_CustomizeDelay_Click(object    sender,
RoutedEventArgs e)
            {
                AppContext.ShowNFX();
            }

            private   void   Dials_TextPromptStateChanged(object   sender,
EventArgs e)
            {
                HaltKeyboardInput = ((DialControl)sender).InputCaptured;
            }

            private void Cb_nfx_ProfileSel_SelectionChanged(object sender,
SelectionChangedEventArgs e)
            {
                if (AppContext == null) return;
                if (cb_NFX_Dropdown.SelectedItem == null) return;
                var item = cb_NFX_Dropdown.SelectedItem;

                AppContext.ActiveNFXProfile = (NFXDelayProfile)item;
                if(AppContext.ActiveNFXProfile == null)
                {
                    AppContext.ActiveNFXProfile                              =
AppContext.NFXProfiles[0];
                }
            }

            private    void    Cb_SequencerProfile_SelectionChanged(object
sender, SelectionChangedEventArgs e)
            {
                if(AppContext != null)
                {
                    AppContext.ActiveSequence                                =
Cb_SequencerProfile.SelectedItem as TrackerSequence;
                }
            }

            private  void  LC_PatternStep_LightIndexChanged(object  sender,
LightCellEventArgs e)
            {
                step = e.LightIndex;
            }
        }
    }
```

Додаток Б – Диск