Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя
(повне найменування вищого навчального закладу)
Факультет комп'ютерно-інформаційних систем і програмної інженерії
(назва факультету)
Кафедра програмної інженерії
(повна назва кафедри)

ПОЯСНЮВАЛЬНА ЗАПИСКА

до дипломного проекту (роботи)

магістра
(освітній ступінь (освітньо-кваліфікаційний рівень))

на тему: «Розробка середовища графічного візуалізатора MIDI-файлів з використанням мови програмування C# та інструментів системи Image-Line FL Studio»

Виконав: студент (~~ка~~) VI курсу, групи СПм-61
спеціальності (напряму підготовки)
121 – «Інженерія програмного забезпечення»
(шифр і назва спеціальності (напряму підготовки))

	(підпис)	Гладій М. Б. (прізвище та ініціали)
Керівник	(підпис)	Бойко І. В. (прізвище та ініціали)
Нормоконтроль	(підпис)	Бойко І. В. (прізвище та ініціали)
Рецензент	(підпис)	Мацюк О. В. (прізвище та ініціали)

м. Тернопіль – 2019

АНОТАЦІЯ

Дипломна робота на тему «Розробка середовища графічного візуалізатора MIDI-файлів з використанням мови програмування C# та інструментів системи Image-Line FL Studio» Гладія Максима Богдановича. – Тернопільський національний технічний університет імені Івана Пулюя, Факультет комп'ютерно-інформаційних систем і програмної інженерії, Кафедра програмної інженерії, група СПм–61 // Тернопіль, 2019.

С. – 101, рис. – 29, табл. – 7, дод. – 4, посил. – 25, слайдів – 15.

Метою дипломної роботи є дослідження інженерії музичного програмного забезпечення, розвитку методів створення програмних систем даної галузі та розробка графічного візуалізатора MIDI-файлів з використанням світлових ефектів частинок.

Методи та програмні засоби, використані під час розробки системи: мова програмування C# та її бібліотеки, специфікація та програмний інтерфейс OpenGL, водоспадна модель розробки програмного забезпечення.

Результатом роботи є реалізоване та протестоване програмне забезпечення, що відмінно виконує поставлене на нього завдання, а саме графічну візуалізацію MIDI-файлів з використанням світлових ефектів частинок, та має значний потенціал до вдосконалення.

Ключові слова: ІНЖЕНЕРІЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ, МОВА ПРОГРАМУВАННЯ C#, OPENGL, РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ, ПРОТОКОЛ MIDI, СЕКВЕНСЕР, FL STUDIO, SYNTHESIA, MIDI-ВІЗУАЛІЗАТОР.

ABSTRACT

Diploma work on the topic «MIDI-files graphic visualizer environment development with the use of C# programming language and Image-Line FL Studio system tools» by student Hladii Maksym Bohdanovych. – Ternopil Ivan Pul'uj National Technical University, Faculty of Computer Information Systems and Software Engineering, Software Engineering Department, group SPm-61 // Ternopil, 2019.

Pages – 101, pictures – 29, tables – 7, add. – 4, ref. – 25, slides – 15.

Diploma work is designed to do research about musical software engineering, evolution of methods of this branch's program systems development and MIDI-files graphic visualizer with the use of light particle effects development.

Methods and software used in performing of system development: C# programming language and its libraries, OpenGL specification and programming interface, waterfall model of software development.

The result of work is implemented and tested software, which does it's task well, namely graphic visualization of MIDI-files with the use of light particle effects, and has great potential for improvement.

Keywords: SOFTWARE ENGINEERING, C# PROGRAMMING LANGUAGE, OPENGL, SOFTWARE DEVELOPMENT, MIDI PROTOCOL, SEQUENCER, FL STUDIO, SYNTHESIA, MIDI-VISUALIZER.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, СКОРОЧЕНЬ ТА ТЕРМІНІВ

C# — об'єктно-орієнтована мова програмування, розроблена компанією Microsoft для платформи .NET.

Microsoft Visual Studio — інтегроване середовище розробки програмного забезпечення.

OpenGL (англ. Open Graphics Library – відкрита графічна бібліотека) — специфікація, що визначає незалежний від мови програмування програмний інтерфейс для написання програм, що використовують двовимірну та тривимірну комп'ютерну графіку.

MIDI (англ. Musical Instrument Digital Interface – цифровий інтерфейс музичних інструментів) — стандарт передачі інформації між електронними музичними інструментами, що дає їм можливість взаємодіяти між собою чи з комп'ютером та іншим MIDI-сумісним обладнанням, також здійснювати з одного інструменту управління іншими.

Секвенсер (англ. Sequencer) — програмне забезпечення, що дозволяє записувати, зберігати, відтворювати та редагувати музичні дані різних типів (у тому числі і MIDI-файли).

MIDI-файл — файл з розширенням *.mid, збережена послідовність MIDI-повідомлень.

FL Studio — редактор-секвенсер для написання музики.

MIDI-візуалізатор — програма, що здійснює графічну візуалізацію MIDI-інформації.

Synthesia — музична тренувальна програма, спрямована на допомогу у вивченні гри на фортепіано за допомогою графічної візуалізації MIDI-файлів.

YouTube — відеохостинг, що надає послуги розміщення відеоматеріалів, є підрозділом компанії Google.

UML (англ. Unified Modeling Language) — уніфікована мова моделювання, використовується у парадигмі об'єктно-орієнтованого програмування.

Діаграма послідовності — UML-діаграма, що відображає об'єкти та класи, що беруть участь у конкретному сценарії, та послідовність повідомлень, якими вони обмінюються між собою, щоб здійснювати виконання сценарію.

Діаграма класів — UML-діаграма, що демонструє загальну структуру ієрархії класів системи, їх атрибутів (полів), методів, інтерфейсів та взаємозв’язків.

Відеодисплейний термінал (ВДТ) — частина електронно-обчислювальної машини, що містить пристрій для візуального відображення інформації, тобто монітор.

Клавішний синтезатор — електронний музичний інструмент, що синтезує звук за допомогою одного чи кількох електричних генераторів коливань, виконаний у вигляді корпуса із клавіатурою.

ЗМІСТ

ВСТУП

Інженерія музичного програмного забезпечення є однією з найбільш динамічних галузей в історії інструментів та технологій музики. Використання цифрових засобів у її створенні не є новою ідеєю. Починаючи з 1950-х, музичне програмне забезпечення не лише почало охоплювати все більше й більше аспектів її створення та відтворення, а й досягло неабиякого рівня розповсюдження, коли компанії, що займаються розробкою такого програмного забезпечення, та автори проектів з відкритим кодом почали працювати відкрито та ділитись своїми напрацюваннями. Це посприяло демократизації засобів музичного виробництва та затвердило нові способи роботи з музикою, а практичне застосування музичних технологій суттєво розвинулось в зв'язку з прогресом в розробках програмного та апаратного забезпечення [1].

Стартовою метою цієї роботи є дослідження появи та розвитку музичного програмного забезпечення, інтеграції та модифікації основних його парадигм. Протягом останнього десятиліття компанії з розробки таких програм досягли значних успіхів у створенні віртуальних музичних інструментів та цифрових аудіо робочих станцій. Враховуючи швидкість розвитку музичних технологій стає важко усвідомити весь наявний обсяг існуючих програмних пакетів. Тим не менш, незважаючи на те, що деякі продукти потребують тривалого часу їх вивчення, більшість є зручними для користувачів та інтуїтивно зрозумілими. Нові розробки даної галузі дають можливість композиторам створювати набагато складніші музичні твори та відкрити для себе цілком новий рівень креативності. Сучасний музикант має доступ до технологій запису та розповсюдження музики, яке може бути як локальним, так і глобальним завдяки доступу до мережі Інтернет [2].

Варто також відзначити, що портативні зовнішні аудіоінтерфейси (зовнішні звукові карти) пройшли значне вдосконалення у порівнянні з їх

попередниками, що суттєво покращило якість звуку для композитора. Навіть звукові системи найпростішого персонального комп'ютера сучасності відповідають мінімальним вимогам для генерування звуку. Програмне забезпечення, що доступне будь-якому користувачу сучасного світу, охоплює значний спектр інструментів, що різними способами стимулюють процес створення музики, і незважаючи на те, що це дуже складні програми та алгоритми, вони все ще лише інструменти, якими керує користувач. Навіть при використанні найсучасніших музичних студій, за усі рішення процесу композиції досі відповідає музикант [3].

Метою даної дипломної роботи магістра є розробка середовища графічного візуалізатора MIDI-файлів, проте її було обрано не лише через постійний ріст популярності музичного програмного забезпечення. Дана тема є актуальною для автора роботи, в зв'язку з його безпосередньою діяльністю у музичній сфері (чому буде приділено увагу пізніше), та в недавній стрімкій появі попиту у музикантів на програмне забезпечення такого типу. Також супутнім завданням проекту є детальне вивчення особливостей створення музичного програмного забезпечення, огляд існуючих музичних програмних систем та дослідження методів їх створення.

Актуальність даної теми сягає коренем до XX ст., коли стали надзвичайно популярними електронні інструменти, а поява одного із них — клавішного синтезатора — поєднала із музичною галузь інженерії програмного забезпечення. Лише за кілька років стало настільки багато музичних інструментів із комп'ютерною складовою, що їх численним виробникам довелось узгодити між собою єдиний стандарт передачі музичних даних. Створений ними протокол MIDI став незмінним стандартом як і для музичних виконавців, так і для інженерів музичного програмного забезпечення, вивівши відповідну галузь на абсолютно новий рівень.

Розробка у 2006 році Ніколасом П'єґдоном музичної тренувальної програми «Synthesia» ввела в обіг поняття MIDI-візуалізатора. Його проект

був спрямований на те, щоб зробити процес навчання гри на фортепіано простішим та комфортнішим для початківців. Програма використовує MIDI- або комп'ютерну клавіатуру, щоб надати можливість її користувачу повторювати за нею графічно візуалізований на екранному віртуальному фортепіано MIDI-файл, даючи можливість бажаючим стати музикантами тренуватись необмежену кількість часу на різних рівнях складності.

Із стрімким розвитком відеохостингу YouTube програма отримала нову хвилю популярності, оскільки частина музикантів почала записувати процес відтворення композиції у «Synthesia», публікуючи отриманий файл як відео-урок на своєму каналі. До таких музикантів належить і автор даної дипломної роботи, чий YouTube-канал «Gamber Piano Tutorials» за 6 років свого існування отримав більше 32 тисяч підписників та 14 мільйонів переглядів.

В середині 10-х років XXI ст. з'явились YouTube-виконавці, що вирішили створити свої аналоги програми «Synthesia», відтворення MIDI-файлів у яких було б візуально привабливішим, із чим прямо пропорційні ріст популярності та комерційної вигоди YouTube-каналу. У свої версії MIDI-візуалізаторів вони впроваджують світлові ефекти та ефекти частинок, фонові зображення та графічні маніпуляції із ними, нові текстури клавіатури та кольорові схеми. Ще одним експериментом є створення 3D-візуалізатора MIDI-файлів, який покищо не отримав такої популярності, як його 2D-колеги. Дані проекти або залишаються на стадії розробки, або успішно проходять життєвий цикл їх створення, після чого залишаються приватною власністю конкретного виконавця.

Слідуючи тенденціям розвитку відеоконтенту вказаного жанру та прислуховуючись до вподобань аудиторії, автором роботи було вирішено проблему розробки власного MIDI-візуалізатора, що об'єднає в собі нещодавно народжені нові ідеї та сприятиме підвищенню росту популярності та комерційної вигоди його YouTube-каналу.

1 ІСТОРІЯ РОЗРОБКИ МУЗИЧНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

1.1 Синтезатор та зародження електронної музики

Друга половина XX ст. була часом появи та швидкого розвитку електронних музичних інструментів. В 60-х роках попереднього століття до уже існуючих електроінструментів долучився новий музичний інструмент — клавішний синтезатор. Перші з них були надзвичайно складними в транспортуванні та налаштуванні, проте вони надали музикантам фундамент для абсолютно нових напрямків розвитку музики.

Усі синтезатори того часу могли виконувати лише одну ноту в конкретний момент часу, тобто були монофонічними. Доводилось використовувати кілька синтезаторів або записувати партію кожного звукового потоку на стрічковий магнітофон з багатьма доріжками, щоб отримати кілька звуків одночасно. Тодішні синтезатори були повністю аналоговими і керувались виключно напругою. При подачі напруги 1 В генератор інструменту повертав висоту тону 100 Гц, 2 В — 200 Гц тощо. Незважаючи на винайдення різних аналогових інтерфейсів, можливість отримання лише однієї ноти в конкретний момент часу все ж залишилась недоліком, з яким було неможливо змиритись, що і спричинило подальший розвиток електронних музичних інструментів. В 70-х роках XX ст. від компанії Oberheim поступив у продаж перший у світі поліфонічний синтезатор «Two Voice». Інструмент мав вбудовану клавіатуру, двоголосу поліфонію та обмежені можливості керування звуками. «Two Voice» мав невеликі розміри та був запрограмований з використанням нескладного алгоритму. Інші музичні компанії не відставали за конкурентом та створювали свої варіанти поліфонічних синтезаторів. Інструменти

виробників Yamaha, Roland, ARP, Moog швидко завойовували популярність електронних музикантів.

Наступним необхідним нововведенням для синтезатора стало надання йому пам’яті. З цією метою в синтезатор вбудували портативний комп’ютер, який мав змогу запам’ятати налаштування інструмента, натискання клавіш та передавати висоту взятих нот на звукові генератори. Після появи пам’яті у електронних інструментах їх асортимент стрімко зріс: одні синтезатори виконували звуки духових інструментів, другі — струнних, треті — зберігали в собі спеціальні сценічні ефекти. Це привело до того, що на концертах деяких виконавців можна було побачити численні клавіатури, кожна з яких виконувала конкретний необхідний музиканту звук.

Здешевлення мікропроцесорів та масове виготовлення інтегральних схем привело до появи у синтезаторах цифрової електроніки вкінці 70-х років. Виготовляти складові інструментів із компактних та дешевих цифрових компонентів було набагато вигідніше, тому аналогові інтерфейси керування швидко вийшли з обігу, а їм на заміну на початку 80-х з’явились перші цифрові інтерфейси.

Незважаючи на великий успіх цифрових синтезаторів, продовжувала існувати невирішена проблема: кожна компанія створювала свій цифровий інтерфейс, що швидко привело до численних звернень музикантів з проханнями створити для них персональний інструмент. Уже в 60-х роках відбувались перші спроби підключення синтезатора до комп’ютера, що не привели до суттєвих успіхів через велику ціну тодішньої техніки. Вкінці 70-х – на початку 80-х існувало вже чимало несумісних між собою інтерфейсів, що виготовлялись різноманітними компаніями. Лише їх розробники могли написати програмне забезпечення для їх об’єднання. Зазвичай такі системи створювались шляхом додавання в комп’ютер спеціальних схем, що генерували звук самі або створювали кілька каналів керуючої напруги для модульних синтезаторів. Багато провідних виробників

швидко усвідомили необхідність створення єдиного універсального інтерфейсу [4].

1.2 Протокол MIDI

1.2.1 Розробка стандарту MIDI

У червні 1981 року на виставці NAMM відбулась зустріч Ікутаро Какехаші (президента компанії Roland), Тома Оберхейма (президента компанії Oberheim) та Дейва Сміта (президента компанії Sequential Circuits). Виробники музичних інструментів поставили перед собою задачу: розробити стандарт передачі дій виконавця в цифровій формі між усіма типами електронних музичних інструментів.

Дейв Сміт зайнявся вивченням комп'ютерних мереж. При розробці мережевих протоколів складаються дві специфікації: апаратного з'єднання пристроїв та формату повідомлень, що передаються по мережі. Внутрішня робота комп'ютера при цьому залишається ізольованою, він реагує на повідомлення дотримуючись заданого стандарту. Такий підхід було використано і для з'єднання музичних інструментів, він став основним принципом протоколу Musical Instrument Digital Interface (MIDI), що залишився незмінним з того часу. В жовтні 1982 року була представлена перша версія специфікації MIDI, того ж року з'явився перший синтезатор з MIDI-інтерфейсом — «Sequential Circuits Prophet 600». Наступного року протокол MIDI було представлено інструментами інших двох компаній — в березні 1983 року з'явився «Roland JX 3 P», а в червні — «Yamaha DX 7».

До появи MIDI синтезатори складалися з двох компонентів в одному контейнері. Першим компонентом була система, що фактично створювала звук. Другим компонентом був контролер, зазвичай клавіатура, чиїм призначенням було перетворення дій виконавця в напругу і струм, тобто в мову, зрозумілу першому компоненту. Цей процес було названо

«захопленням виконавських штрихів». Протокол MIDI розірвав взаємозв'язок цих компонентів: тепер будь-який контролер міг керувати будь-яким звуковим генератором. Це мало величезне значення — тепер музикант міг вільно підбирати необхідне йому обладнання, без страху, що воно застаріє через півроку, як це тоді постійно відбувалось з більшістю електронних пристроїв.

Роботу над MIDI-інструментами потрібно було скоординувати, тому в 1983 році в Японії був створений комітет з MIDI-стандартів (JMSC). В серпні цього ж року було офіційно представлено специфікацію MIDI 1.0, а в червні 1984 року було сформовано асоціацію MIDI-виробників (MMA — MIDI Manufacturers Association). Організації MMA та JMSC спільно займаються діяльністю стандартизації та розширення протоколу MIDI. Будь-який зареєстрований член цих організацій може запропонувати своє доповнення до протоколу, після чого воно буде винесено на загальне голосування.

Протокол MIDI відкрив чимало можливостей для комп’ютерного синтезу та керування звуком. Комп’ютери почали використовуватись в ролі засобу керування синтезаторами — секвенсера, тобто програми-композитора. В 1984 році Джим Міллер створив програму «Personal Composer» для IBM PC — MIDI-секвенсер, що дозволяв роздруковувати ноти. З цього розпочався подальший стрімкий розвиток технологій, пов’язаних з новоствореним протоколом. Через деякий час було створено програми-секвенсери для комп’ютерів Apple II та Commodore 64, компанія Roland створила гітарний MIDI-контролер, компанія Yamaha — перший процесор ефектів, пресети якого можна було змінювати MIDI-повідомленнями, а 1985 рік став знаменитим через захоплення європейського ринку комп’ютерами Atari, першими що мали вбудовані MIDI-порти.

З появою Microsoft Windows 3.1 у користувачів персональних комп’ютерів з’явилась підтримка MIDI на рівні операційної системи. Було створено програму Cakewalk для Windows, програма Cubase, що раніше

випускалась для Atari та Macintosh, також стає доступною для персонального комп’ютера. В 1993 році у комп’ютерів з’являються перші звукові карти з вбудованим MIDI-інтерфейсом, а MIDI-технологія починає активно експлуатуватись в обох секторах ринку музики: професійному та любительському. Незважаючи на появу віртуальних студій, віртуальних синтезаторів, процесорів ефектів та інших численних програм, що використовують MIDI, специфікація досі має версію 1.0 [4].

1.2.2 Основи MIDI-протоколу

Musical Instrument Digital Interface — це цифровий інтерфейс та протокол зв’язку між музичними інструментами. Кожного разу коли музикант виконує яку-небудь дію пристроями керування (натискання/відпускання клавіш, педалей, зміну положення регуляторів тощо), інструмент формує відповідне MIDI-повідомлення, що миттєво передається по інтерфейсу. Інструменти, що отримують повідомлення, опрацьовують їх так, наче дія проводилась над їх власними пристроями керування. Таким чином, потік MIDI-повідомлень — це в деякій мірі «зліпок» дій виконавця, що зберігає ще й властивий йому стиль виконання — динаміку гри, технічні прийоми тощо. Під час запису на пристрої зберігання інформації MIDI-повідомлення отримують часові маркери, перетворюючись в своєрідний спосіб представлення музичної партитури. При відтворенні цих маркерів повністю відтворюється вихідний MIDI-потік.

Специфікація MIDI складається з апаратної специфікації самого інтерфейсу і специфікації формату даних, або протоколу — опису системи повідомлень, що передаються. Відповідно розрізняють апаратний MIDI-інтерфейс і формат MIDI-даних (тобто уже згадану MIDI-партитуру). Інтерфейс використовується для фізичного з’єднання джерела та приймача повідомлень, а формат даних — для створення, зберігання та передачі MIDI-

повідомлень. В сучасному світі ці поняття стали самостійними та тепер можуть використовуватись і окремо — MIDI-інтерфейсом можуть передаватись дані будь-якого іншого формату, а MIDI-формат може використовуватись лише для обробки партитур, без їх подальшого виводу на пристрій синтезу.

MIDI-інформація являє собою повідомлення або події (events), кожна з яких є командою для музичного інструменту. Стандарт MIDI передбачує 16 незалежних та рівноцінних логічних каналів, кожен з яких працює згідно власного режиму роботи. Спочатку це було призначено для інструментів, що здатні в конкретний момент часу виконувати звук лише одного тембру — кожному інструменту присвоювався свій номер каналу, що давало можливість багатотембрового виконання. Пізніше з’явились мультитемброві інструменти, тому зараз кожному каналу зазвичай призначають тембр, що за традицією називається інструментом, проте може насправді бути комбінацією кількох у одному каналі. Також існує традиція використовувати канал 10 для ударних інструментів — різні ноти у ньому відповідають різним ударним звукам фіксованої висоти, інші канали ж використовуються для інструментів іншого частотного діапазону, де різні ноти, зазвичай, відповідають різній висоті тону одного і того ж інструменту.

Оскільки MIDI-повідомлення — це потік даних реального часу, у їх кодуванні було приділено увагу полегшенню синхронізації в випадку втрати з’єднання. Для цього перший байт кожного повідомлення (байт стану), містить «1» в старшому розряді, а інші байти містять «0» та називаються байтами даних. Якщо після отримання усіх байтів даних останнього повідомлення на вхід приймача поступає байт, що не містить «1» в старшому розряді — це визначається як повторення інформаційної частини повідомлення. Такий метод передачі даних отримав назву «Running Status» і став досить популярним через зменшення об’єму даних, що передаються.

Для зберігання MIDI-партитур на носіях даних розроблено формати SMF (Standard MIDI File) трьох типів: 0 — власне MIDI-потік в тому вигляді, в якому він передається інтерфейсом; 1 — сукупність паралельних доріжок, кожна з яких є окремою партією композиції та виконується на одним MIDI-каналом; 2 — сукупність кількох композицій, кожна з яких складається з кількох доріжок. Найчастіше використовується формат 1, який дозволяє зберігати одну композицію в одному файлі, що виявилось найзручнішим способом для музикантів.

Крім MIDI-повідомлень, музичний файл містить також «несправжні події» (Meta Events), що використовуються для оформлення файлу і не передаються інтерфейсом — тобто інформацію про метрику та темп, опис композиції, назви партій, слова пісні тощо [5].

1.3 Секвенсер, його призначення та функціонал

В сучасному світі секвенсер вважається основним інструментом композитора. Він може бути вбудованим в музичний інструмент, може бути окремим пристроєм чи комп’ютерною програмою. Термін «секвенсер» тривалий час використовувався для позначення пристрою запису та відтворення звукової інформації для електронного музичного інструменту. Поступово термін змінив своє основне значення — тепер так називають функцію програмного забезпечення, що дозволяє зберігати, відтворювати та редагувати MIDI-інформацію. Незалежно від варіанту виконання робота з секвенсером займає більшу частину часу створення музичної композиції.

Секвенсер — це апаратний або програмний пристрій. Сьогодні важко знайти студію, в якій не було б комп’ютера. А ще важче знайти студійний комп’ютер без програми-секвенсера. Більше того, як правило комп’ютер в студії працює якраз виключно в якості секвенсера.

Апаратними секвенсерами активно користувалися в 90-х роках XX ст. Секвенсер на базі комп'ютера суттєво випереджує апаратні та вбудовані в музичні інструменти практично за усіма факторами. Основним способом запису інформації в секвенсер є відтворення необхідних партій MIDI-клавіатурою або іншим інструментом, що має MIDI-вихід. Це можуть бути електронні ударні, гітара або будь-який інструмент з наявним в ньому MIDI-інтерфейсом.

Найчастіше запис відбувається в основному вікні секвенсера, що зветься вікном аранжування. Таке вікно складається із двох частин: у лівій стороні екрану розміщуються назви доріжок та їх параметри (наприклад, номер MIDI-каналу), у правій — записана на кожній доріжці музична інформація (у вигляді прямокутників різною довжини). Також у вікні присутній локатор позиції — вертикальна лінія, що показує у якому місці запису зараз знаходиться користувач програми. Крім роботи у звичайному (лінійному) режимі більшість секвенсерів можуть записувати і відтворювати інформацію також у циклічному режимі, а однією із найголовніших можливостей секвенсера є можливість проведення квантизації — процесу переміщення нот до найближчих ритмічних долей.

Малоймовірно, що записана партія одразу стане такою, як було задумано композитором, тому секвенсер надає надзвичайно корисну можливість редагування виключно всіх MIDI-подій. Секвенсер містить багато редакторів для різних аспектів композиції музики: редактор MIDI-подій, що являє собою довгу таблицю або список; партитурний редактор, що призначений для тих, кому зручно визначати ноти за їх положеннями на нотному стані (а також тих, кому зручно користуватись мишею замість MIDI-клавіатури) тощо.

Основою секвенсера є клавішний редактор. В одному із популярних секвенсерів Cubase (компанія Steinberg) партитурного редактора немає взагалі. В класичному клавішному редакторі кожна нота відображається у

вигляді полоси, довжина якої відповідає довжині відповідної ноти, а розташування полос напроти відповідних клавіш звичайної музичної клавіатури, що розташована вертикально в лівій частині вікна, свідчить про висоту ноти і її приналежність до конкретної октави. Клавішний редактор є найбільш популярним серед користувачів програм-секвенсерів і зазвичай зветься «Piano Roll».

Також існують графічний редактор (призначений ля таких подій, як гучність чи зміна висоти тону), барабанний редактор (призначений для редагування ударних подій) та MIDI-мікшер (на якому зручно проводити фінальне зведення композиції — на кожний канал можна призначити MIDI-інструмент і регулювати його гучність, панораму тощо). В останніх десятиліттях виробники програм-секвенсерів почали надавати користувачам можливість запису не лише MIDI-, а й аудіодоріжок, тому в сучасному секвенсері можна керувати і вокалом, і живими інструментами [6].

1.4 Секвенсер FL Studio та VST-інструменти

FL Studio (раніше Fruity Loops) — це цифрова аудіо робоча станція-секвенсер для запису, редагування, зберігання та відтворення музики, створена програмістом Дідьє Дембреном у 1997 році. FL Studio — потужний патерновий секвенсер, доріжки якого містять не неперервні музичні партії, а послідовності з окремих фрагментів MIDI-секвенцій — патернів, що являються складовими для побудови аранжування. Патерни можуть бути і барабанними фрагментами, і басовими партіями, і сольними вставками, і спецефектами, і різноманітними комбінаціями переліченого раніше.

Для створення та редагування патернів FL Studio пропонує два MIDI-редактори: кроковий секвенсер (Step Sequencer) та клавішний редактор (Piano Roll). Перший з них надзвичайно зручний для створення ударних або нескладних мелодичних патернів (див. рис. 1.1, аркуш 22), а другий дозволяє

працювати з MIDI-партіями будь-якої складності та пропонує повний комплект MIDI-інструментів: індивідуальне вікно властивостей кожної ноти, квантизатор, арпеджіатор, рандомайзер тощо (див. рис. 1.2, аркуш 22).

Фінальне аранжування зводиться з складових патернів у вікні Playlist, яке також містить багатоканальну аудіочастину (Audio Tracks), на доріжках якої можна розміщувати аудіофрагменти (див. рис. 1.3, аркуш 22).

Рис. 1.1 — Робота з Step Sequencer у FL Studio

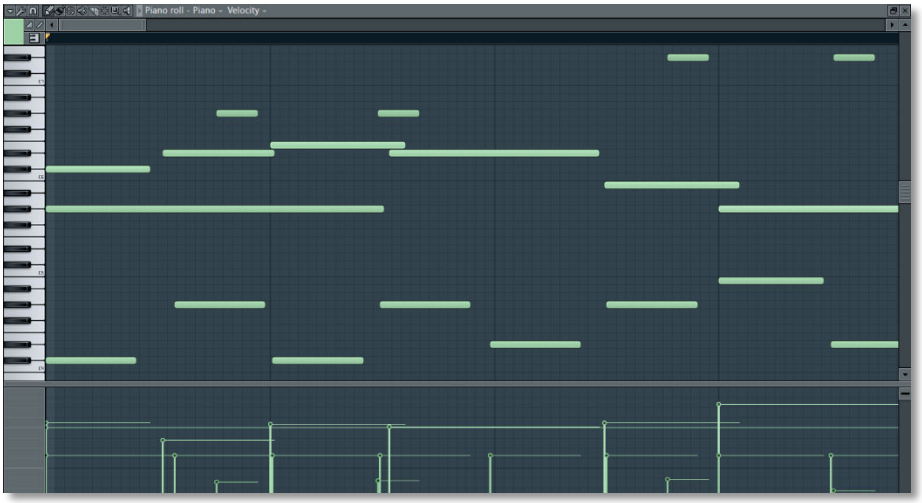

Рис. 1.2 — Робота з Piano Roll у FL Studio

Рис. 1.3 — Зведення композиції у вікні Playlist (FL Studio)

Для того, щоб відтворити MIDI-команди, потрібно мати генератор звуку. У цифрових аудіо робочих станціях такі генератори називають VSTi (Virtual Studio Technology Instrument) або VST-інструментами. MIDI-партії можуть програватися власними генераторами (синтезаторами, ритм-машинами, інструментами та семплерами FL Studio) чи спеціалізованими VST-інструментами інших виробників. Також MIDI-команди можуть виводитись через MIDI-інтерфейс комп’ютера на зовнішні MIDI-пристрої.

VST-інструментів у проекті може бути необмежена кількість. Кожен генератор звуку має свої налаштування та унікальне звучання, що імітує конкретний музичний інструмент. Звук кожного генератора може бути оброблений за допомогою безлічі ефектів (див. рис. 1.4) [7].

Рис. 1.4 — Генератори у FL Studio

Узагальнену блок-схему процесу створення музики у секвенсері на прикладі FL Studio зображено на рис. 1.5, аркуш 24.

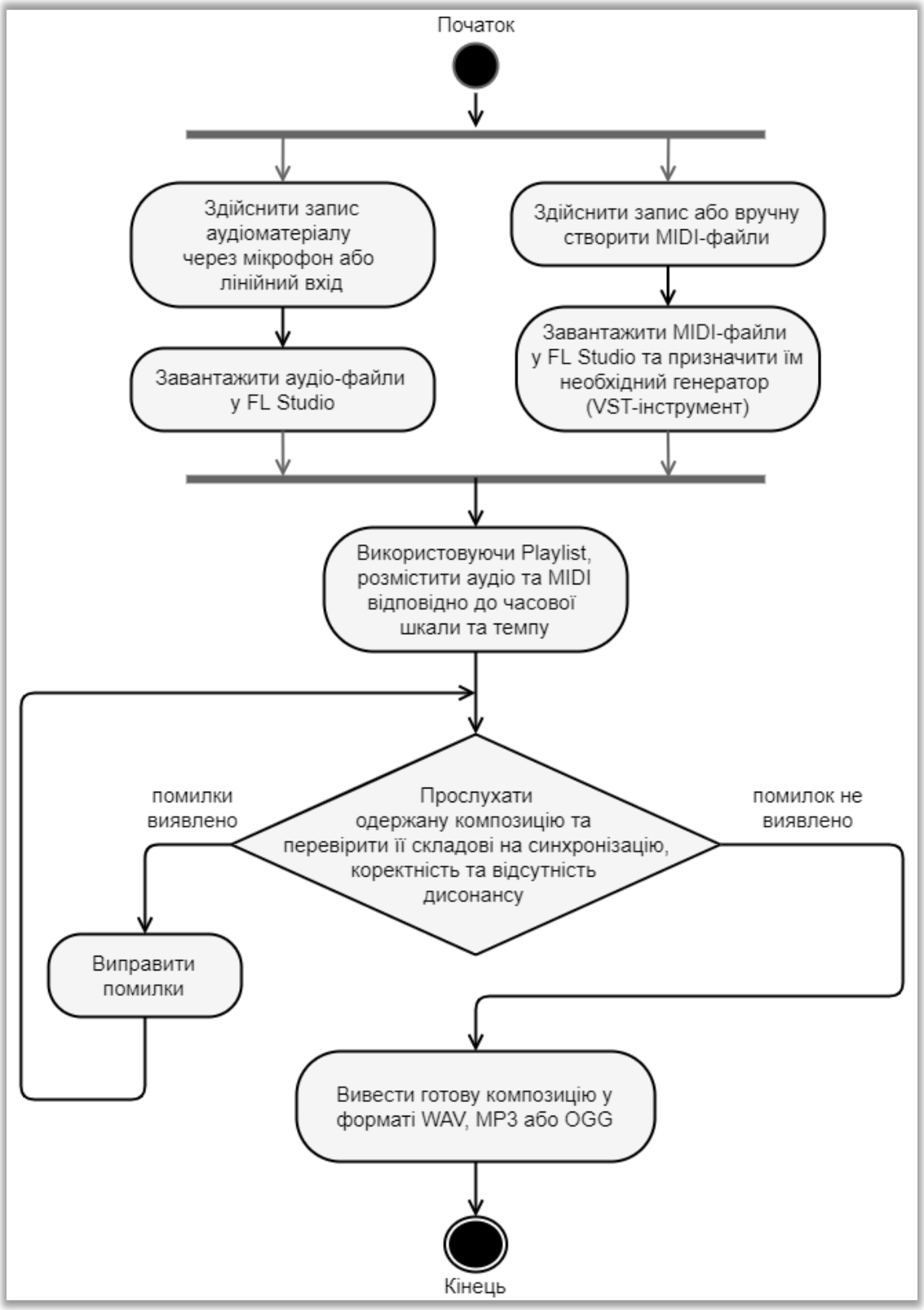

Рис. 1.5 — Блок-схема алгоритму створення музики у секвенсері на прикладі FL Studio

1.5 MIDI-візуалізатор. Тренувальна музична програма «Synthesia»

В 2006 році американський розробник Ніколас П’єґдон створив музичну гру «Synthesia» — тренувальну програму, спрямовану на вивчення гри на фортепіано, що дозволяє користувачам використовувати MIDI- або комп’ютерну клавіатуру, щоб зіграти графічно візуалізований на екранному віртуальному фортепіано MIDI-файл.

За основу для візуалізації взято кроковий редактор Piano Roll, описаний у підрозділі 1.3, розвернутий на 90 градусів проти годинникової стрілки: у нижній частині вікна «Synthesia» розміщено клавіатуру фортепіано, на яку зверху «падають» звуки MIDI-файлу різної тривалості, представлені прямокутниками різної довжини, а їх розташування по вертикалі та горизонталі визначається відповідно часовою позицією ноти та її висотою (див. рис. 1.6) [8].

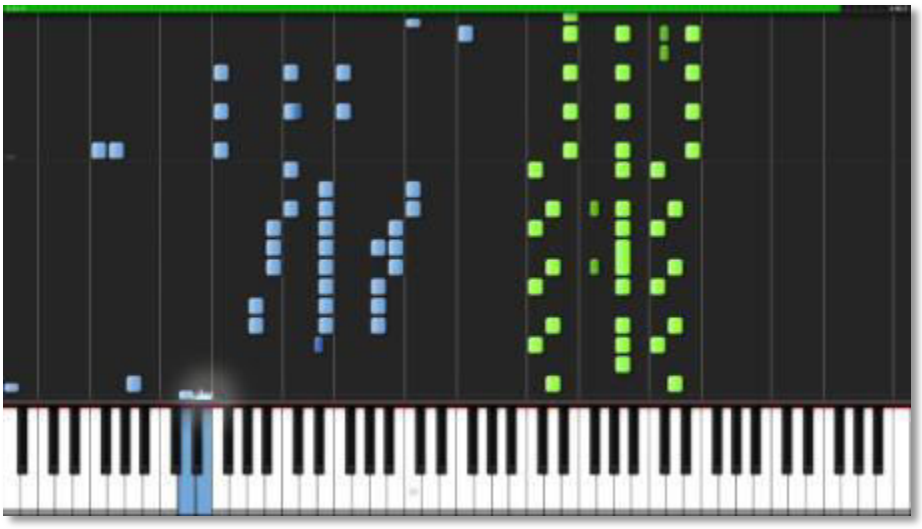

Рис. 1.6 — Музична тренувальна програма «Synthesia»

«Synthesia» отримала велику популярність, адже суттєво понизила вимоги щодо початку навчання гри на фортепіано для початківців. З появою даного MIDI-візуалізатора усі бажаючі змогли розпочати навчання негайно, не знаючи нотної нотації чи теорії музики. Програма стала популярною і серед досвідчених гравців, ставши доступною платформою для швидкого перегляду нового музичного твору, оскільки практично будь-яку композицію

можна знайти у вигляді того ж самого, відомого уже майже 30 років, стандартного формату MIDI-файлів, який «Synthesia» чудово розуміє.

Найбільшу популярність «Synthesia» отримала у музикантів, що є користувачами відеохостингу YouTube. Вони використовують її для графічного представлення MIDI-файлів, створення навчальних відео або ж просто для приємного супроводу їх гри на справжніх інструментах.

Незважаючи на велику цінність «Synthesia», програма повністю не розкриває потенціалу графічної візуалізації MIDI-файлів, що дає підстави до перегляду її концепції та створення нового програмного забезпечення, що поєднуватиме в собі графічну візуалізацію MIDI-файлів з використанням світлових ефектів частинок та функції секвенсера: створення нових та редагування існуючих MIDI-файлів у нотному редакторі. В якості успіху нова концепція може бути розширена до використання VST-інструментів, що може зробити розроблене програмне забезпечення першим представником нового покоління цифрових звукових робочих станцій.

2 РОЗРОБКА ПРОГРАМНОЇ СИСТЕМИ

2.1 Аналіз вимог до програмної системи

2.1.1 Огляд існуючих програм та аналіз предметної області

«Synthesia» стала водночас першим та найпопулярнішим MIDI-візуалізатором у своєму роді (див. рис. 2.1). Програма відображає MIDI-файл як падаючі ноти та дозволяє користувачу легко повторяти композицію за нею. Єдиною суттєвою системною вимогою програми є підтримка персональним комп'ютером стандарту OpenGL 2.0 (що випущений у 2004 році).

Уже більше як десятиліття «Synthesia» постійно розвивається. У 2008 році було задоволено прохання користувачів про створення нотного режиму перегляду та додано «Learning Pack» — платне розширення, що включає у себе додаткові навчальні функції та постійно розширюється. Програма дозволяє повторення фрагментів за допомогою створення циклу для тривалого тренування, дає можливість слідкувати за ритмом гри завдяки вбудованому метроному та містить більше 100 музичних інструментів на вибір [9].

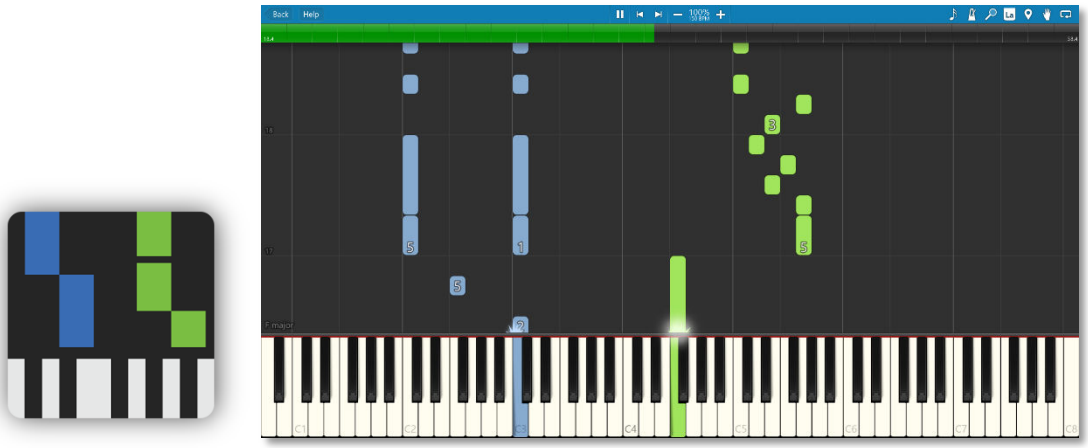

Рис. 2.1 — Логотип та зовнішній вигляд актуальної версії

тренувальної музичної програми «Synthesia»

Поява ідеї графічної візуалізації MIDI-файлів спровокувала спроби програмістів створити свій аналог «Synthesia». Одним із таких є програма з відкритим кодом «Piano From Above».

Її функціонал майже повністю повторяє основні функції «Synthesia», дозволяє налаштовувати швидкість відтворення, відображати та приховувати MIDI-треки, також містить метроном тощо [10]. Логотип та зовнішній вигляд «Piano From Above» зображені на рис. 2.2.

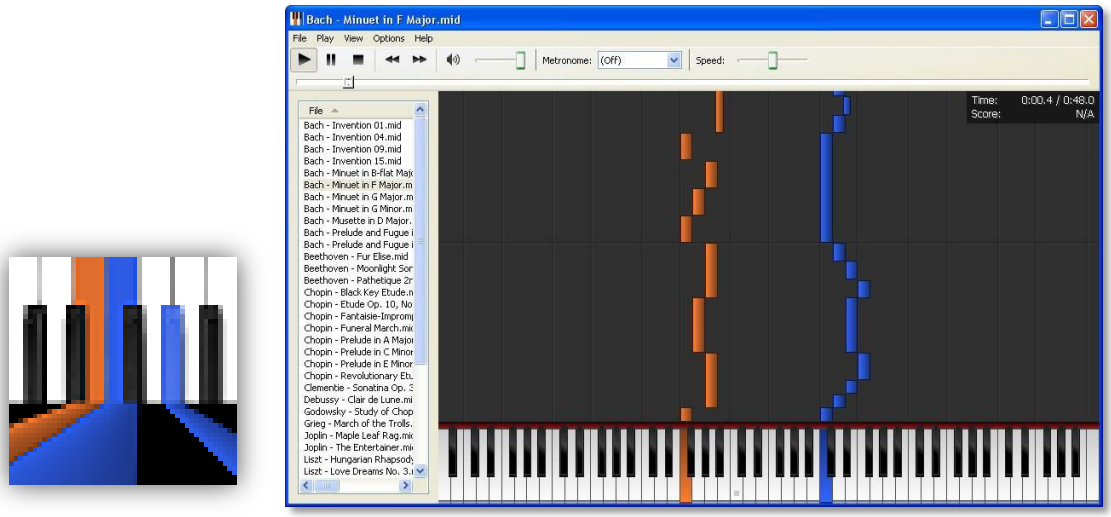

Рис. 2.2 — Логотип та інтерфейс тренувальної музичної програми з відкритим кодом «Piano From Above»

MIDI-візуалізатор було створено і для інших платформ: компанією «Revontulet Soft Inc.» було створено тренувальну музичну програму для мобільних пристроїв — «Perfect Piano».

У додатку реалізовано 88-клавішну фортепіанну клавіатуру, кілька режимів гри, в тому числі і спеціальний режим для двох виконавців, метроном, великий вибір вбудованих музичних інструментів тощо. Додаток підтримує multi-touch-екрани, розпізнає силу дотику та дозволяє запис аудіо у форматі MIDI та AAC [11].

Логотип та інтерфейс мобільного додатку «Perfect Piano» представлено на рис. 2.2, аркуш 29. Режим для двох виконавців представлено на рис. 2.3, аркуш 29.

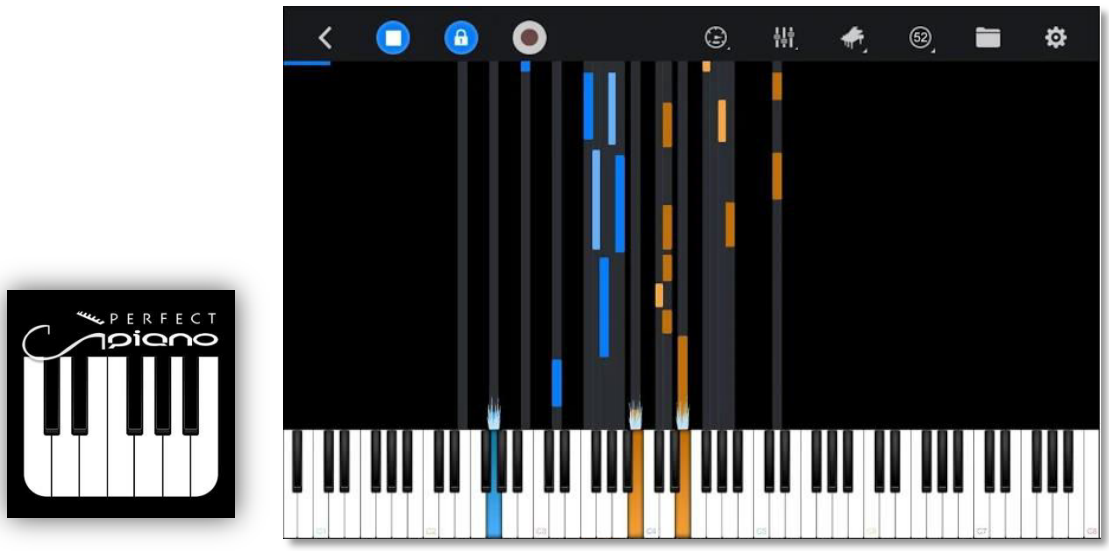

Рис. 2.2 — Логотип та інтерфейс мобільного додатку «Perfect Piano»

Рис. 2.3 — Спеціальний режим для двох виконавців у «Perfect Piano»

Найуспішнішим послідовником «Synthesia» в сфері створення навчальних музичного програмного забезпечення можна сміливо назвати мобільний комерційний додаток «Flowkey». Замість графічної візуалізації

додаток демонструє користувачу відео виконання музичного твору з підсвіткою клавіш, що натискаються, водночас демонструючи ноти даної композиції. Безплатна версія «Flowkey» містить лише 5 MIDI-творів для відтворення, а необмежена кількість MIDI-творів стає доступною після купівлі Premium-версії [12].

Логотип та інтерфейс мобільного додатку «Flowkey» представлено на рис. 2.4.

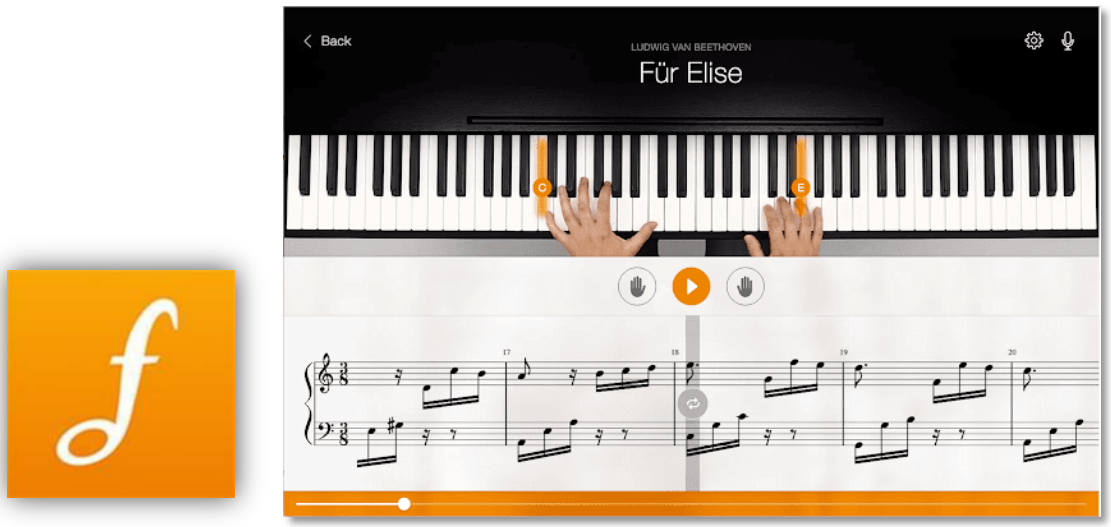

Рис. 2.4 — Логотип та інтерфейс мобільного додатку «Flowkey»

Описані вище додатки чудово виконують завдання, на них покладене, проте їх концепції досі мають потенціал до вдосконалення. Недоліком програми з відкритим кодом «Piano From Above» та мобільного додатку «Perfect Piano» можна назвати їх обмежену функціональність, а додаток «Flowkey», хоч він і не використовує графічну візуалізацію музичних творів, отримує скарги на надто високу вартість покупки Premium-версії.

Велика кількість користувачів бажає отримати розширення «Synthesia» або повністю новий MIDI-візуалізатор з використанням світлових ефектів частинок та можливістю створення нових та редагування існуючих MIDI-файлів у вбудованому нотному редакторі.

Нова концепція також може бути розширена до використання VST-інструментів замість вбудованих інструментів операційної системи, що може

спрямувати таку розробку на шлях перетворення у першого представника нового покоління цифрових звукових робочих станцій.

2.1.2 Постановка задачі дипломної роботи магістра

Завданням даної дипломної роботи є створення програмного забезпечення, що здійснюватиме графічну візуалізацію MIDI-файлів з використанням світлових ефектів частинок. Можливе розширення системи: включення нотного редактора для створення нових та редагування існуючих MIDI-файлів, а також реалізація можливості використання VST-інструментів.

Перш за все, необхідно виділити головний перелік задач, які необхідно виконати в рамках дипломного проекту:

– об’єднавши результати огляду існуючих програмних систем та аналізу предметної області визначити точні вимоги до програмного забезпечення, що розробляється;

– визначити акторів та варіанти використання програмної системи;

– обрати процес розробки, мову програмування та середовище розробки програмного забезпечення, інтерфейс для роботи з графікою;

– здійснити моделювання архітектури системи: побудувати UML-діаграми послідовності та класів;

– успішно виконати конструювання програмної системи, її класів та методів, реалізувавши усі вимоги, що були затверджені в рамках проекту;

– здійснити розгортання програмного забезпечення;

– провести тестування отриманої програмної системи та переконатись у коректності роботи створеного програмного забезпечення.

В результаті виконання процесу визначення вимог було складено перелік функцій, що повинні бути реалізовані у програмній системі:

– відтворення та графічна візуалізація MIDI-файлів на екранному віртуальному фортепіано;

– можливість зміни масштабу та якості графічної візуалізації за шкалою «half – low – medium – high – double»;

– можливість зміни кольору об’єктів, що відображають звуки MIDI-файлу, та світлових ефектів частинок; зміни розміру, швидкості та кількості частинок, що генеруються;

– можливість задання фонового зображення під час візуалізації MIDI-файлу та регулювання його прозорості.

Підсумковою та головною вимогою є стабільність та ефективність програмного забезпечення, що розробляється, цілісність його архітектури та відсутність програмних помилок, функціональна придатність та відповідність поставленим вимогам.

2.1.3 Пошук акторів та варіантів використання

Пошук акторів є першим кроком у моделюванні програмної системи. Кожен тип зовнішніх сутностей, з якими повинна взаємодіяти система, повинен бути представлений відповідним актором. Операційне середовище програмної системи складається з користувачів, пристроїв та програм — акторів, з якими система взаємодіє. Вони не обов’язково представляють конкретні фізичні сутності, а лише окремі ролі деяких з них, що мають відношення до відповідних варіантів використання. Одна фізична сутність може перейти в роль кількох акторів, один актор може відігравати ролі кількох різних фізичних сутностей.

Аналіз предметної області та вимог до програмного забезпечення, що розробляється, показав, що для його функціонування достатньо одного актора: власне користувача програмної системи (див. рис. 2.5).

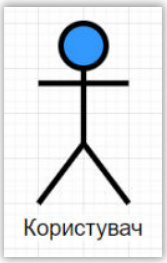

Рис. 2.5 — Єдиний актор програмної системи

Короткий опис знайденого актора представлено у таблиці 2.1.

Табл. 2.1 — Короткий опис актора програмної системи

Актор	*Короткий опис*
Користувач	Має змогу відкривати MIDI-файли, запускати та зупиняти їх відтворення та графічну візуалізацію, змінювати її масштаб та якість, змінювати колір звуків та характеристики ефектів світлових частинок, встановлювати фонове зображення

Аналіз вимог до програмного забезпечення, що розробляється, та пошук акторів програмної системи привели до визначення варіантів використання, що представлені у таблиці 2.2.

Табл. 2.2 — Короткий опис варіантів використання програмної системи

Актори	*Найменування*	*Формулювання*
Користувач	Відкрити MIDI-файл	Користувач виконує відкриття MIDI-файлу, вказуючи директорію його місцезнаходження; система відкриває вказаний користувачем MIDI-файл
Користувач	Розпочати / зупинити відтворення MIDI-файлу	Користувач вмикає / вимикає відтворення обраного MIDI-файлу, система розпочинає / призупиняє графічну візуалізацію

Користувач	Змінити масштаб та якість графічної візуалізації	Користувач змінює масштаб та якість графічної візуалізації за шкалою «half – low – medium – high – double»
Користувач	Змінити колір звуків MIDI-файлу та ефектів світлових частинок	Користувач змінює колір звуків MIDI-файлу та ефектів частинок, обираючи колір з кольорової гами операційної системи
Користувач	Змінити швидкість, розмір та кількість світлових частинок	Користувач змінює швидкість, розмір та кількість світлових частинок
Користувач	Встановити фонове зображення для графічної візуалізації	Користувач встановлює фонове зображення, вказуючи директорію його місцезнаходження
Користувач	Відрегулювати прозорість фонового зображення	Користувач регулює прозорість встановленого фонового зображення

Підсумовуючи пошук акторів та визначення варіантів використання програмної системи, створимо діаграму варіантів використання (див. рис. 2.6).

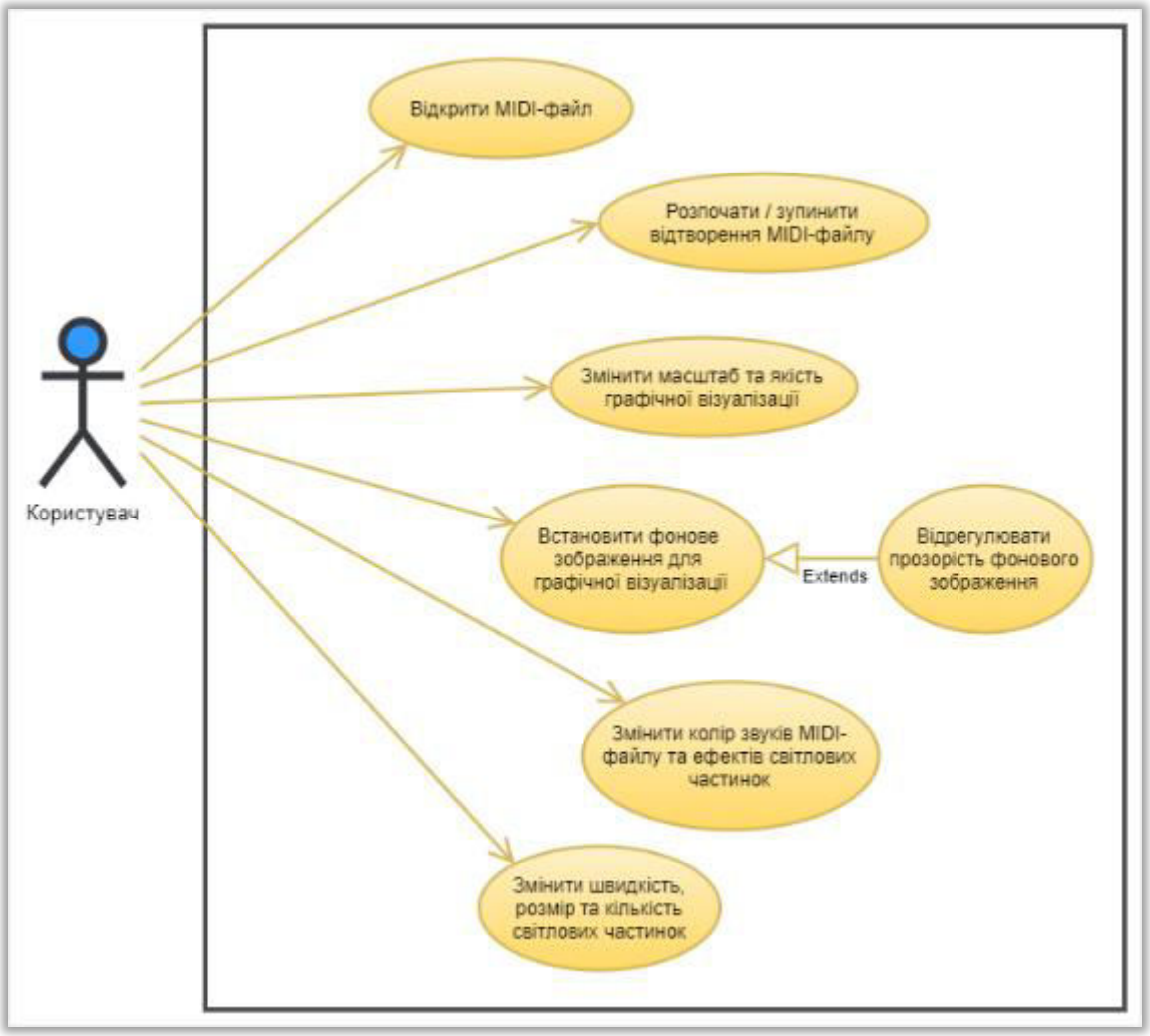

Рис. 2.6 — Діаграма варіантів використання програмної системи

2.2 Проектування та конструювання програмної системи

2.2.1 Вибір процесу розробки

Проаналізувавши існуючі методології розробки програмного забезпечення та особливості програмної системи, що розробляється, в якості процесу розробки було обрано водоспадну модель.

Водоспадна модель (Waterfall model) була першою моделюю розробки програмного забезпечення, що була представлена світові. У водоспадній моделі кожна фаза повинна бути завершена до того, як розпочнеться наступна, і дані фази не повинні перекриватись. Дана модель вважається

найпростішою для розуміння та виконання: вона є найбільш раннім підходом, що застосовувався для розробки програмних систем.

У моделі «Водоспад» процес розробки програмного забезпечення є поділеним на окремі фази. Результати кожної фази стають вхідними даними для відповідних наступних. Це означає, що будь-яка фаза процесу розробки може розпочатись лише за умови завершення попередньої фази, тобто водоспадна модель — це послідовний процес проектування, в якому процес протікає (як водоспад) через фази створення концепції, ініціації, аналізу, проектування, конструювання, тестування, розгортання та обслуговування.

Водоспадну модель використовують тоді, коли вимоги до програмного забезпечення змінюються не часто або ж не змінюються зовсім; коли програмне забезпечення, що розробляється, не є складним та тривалим; коли усі необхідні ресурси доступні та налаштовані; коли проект є невеликим, усі вимоги зрозумілі, а середовище розробки — стабільне. Так як водоспадна модель ілюструє процес розробки програмного забезпечення як лінійний послідовний потік, її також називають лінійно-послідовною моделлю життєвого циклу. Зазвичай виділяють шість основних послідовних фаз даної моделі (див. рис. 2.7, аркуш 36).

Рис. 2.7 — Шість основних фаз водоспадної моделі

Requirements (Вимоги). Перша фаза вимагає отримання розуміння програмного забезпечення, яке потрібно розробити та якими повинна бути його функціональність, призначення тощо. Саме на цьому етапі фіксуються специфікації вхідних та вихідних даних, а також вивчається бажаний фінальний продукт. Бізнес-аналітик документує вимоги до програмного забезпечення, а команда розробників їх аналізує.

System Design (Проектування системи). Протягом цієї фази одержані на попередній фазі технічні вимоги посилено вивчаються та розпочинається проектування системи. Архітектор та старші члени команди працюють над архітектурою програмного забезпечення, високим та низьким рівнями дизайну проекту. Дана фаза також допомагає у визначенні апаратних і системних вимог та визначає загальну архітектуру системи. Базовий програмний код, що повинен бути готовим на наступній фазі, створюється саме під час проектування системи.

Implementation (Реалізація). Команда розробки працює над створенням програмного коду проекту. Завдяки результатам фази проектування, система вперше розробляється у вигляді невеликих програм, які звуться блоками або юнітами, що будуть інтегровані у наступній фазі. Кожен юніт розробляється та проходить тестування на свій функціонал, а даний процес називають юніт-тестуванням.

Integration and Testing (Інтеграція та тестування). Усі юніти, що розроблені у попередній фазі, інтегруються у програмну систему після тестування кожного з них та виправлення знайдених помилок. Командою також проводиться регресивне тестування на випадок появи нових дефектів. Необхідно, щоб створене програмне забезпечення постійно проходило тестування для перевірки наявності будь-яких недоліків. Також тестування приділяє увагу тому, щоб майбутній користувач програмного забезпечення не стикався з проблемами і під час встановлення програмного забезпечення.

Deployment of System (Розгортання системи). Після завершення функціонального та нефункціонального тестування, готовий продукт переходить у фазу розгортання в середовищі його користувачів або випускається на ринок програмного забезпечення.

Maintenance (Обслуговування). Ця фаза проходить після встановлення створеного програмного забезпечення та включає в себе модифікації у системі в цілому або ж в окремому її компоненті для зміни атрибутів чи підвищення продуктивності. Потреба у таких модифікаціях з'являється або у відповідь на запити, отримані від замовника, або ж в процесі виявлення дефектів під час використання програмної системи. Клієнту надається регулярні обслуговування та підтримка програмного забезпечення, що було для нього розроблене.

Перевагою водоспадної моделі є те, що вона дозволяє здійснювати строгий контроль за процесом створення програмного забезпечення, адже графік розробки зазвичай містить терміни для завершення кожного етапу. Водоспадна модель піддається легкому керуванню через її жорсткість — кожна фаза повинна принести розробникам конкретні результати. Дана модель найкраще підходить для невеликих проектів, де вимоги добре вивчені, адже фази виконуються та завершуються одна за одною, не перекриваючись, етапи моделі прості та зрозумілі, а процеси та їх результати доступно документуються.

Головним недоліком водоспадної моделі є те, що вона не дозволяє значного відхилення від вимог або їх перегляду. Після того, як програмне забезпечення переходить на етап тестування, стає надзвичайно важко повернутись назад, якщо з'явилась потреба додати або змінити щось, що не було продумано на концептуальному етапі. Через це водоспадна модель не рекомендується до використання у складних об'єктно-орієнтованих проектах та проектах з помірним та високим ризиками у вимог в тому числі і тому, що

при її використанні важко оцінити час та витрати, що необхідні для кожної фази розробки програмного забезпечення [13].

2.2.2 Вибір мови програмування та середовища розробки

Наступним етапом проектування є визначення мови програмування для написання програмної системи. Цей вибір не є складним, проте має фундаментальну важливість для майбутнього проекту. Кожна мова програмування має свої множину ключових слів та синтаксис написання програм, рівень складності та специфіку застосування, переваги та недоліки. Спираючись на рейтинг мов програмування та специфіку проекту претендентами на використання стали мови C# та Java, що уже тривалий час змагаються за перше місце за популярністю у світі.

C# — це мова програмування загального призначення, що була створена в рамках ініціативи Microsoft .NET. Її було розроблено для загальної мовної інфраструктури (CLI) — відкритої специфікації, розробленої компанією Microsoft. Програми C# компілюються в байт-код, що може запускатись при реалізації CLI.

Java — це мова програмування загального призначення, що була створена із конкретною ціллю «write once, run anywhere», що означала можливість запуску єдиного коду на будь-якій платформі. Програми Java компілюються в байт-код, що може запускатись при реалізації віртуальної машини JAVA (JVM).

Поява і Java, і C# спричинена переходом від низькорівневих мов програмування, таких як C++, до мов більш високого рівня, що компілюються в байт-код. Спільними рисами двох мов вважаються безпека типів, одиночне успадкування, підтримка інтерфейсів та вбудоване «збирання сміття», тобто автоматичне керування пам’яттю. Java підтримує поліморфізм «за замовчуванням», а в C# для використання поліморфізму

буде потрібно використати ключове слово «virtual» в базовому класі та «override» в класі-нащадку.

Прийнято вважати, що C# зазвичай використовується для розробки програмного забезпечення для платформ Microsoft .NET Framework CLR і є найбільш використовуваною реалізацією CLI. З іншого боку Java має свою велику зону популярності, адже компанія Google використовує JVM для своєї операційної системи Android.

Зваживши усі переваги та недоліки обох мов програмування та взявши до уваги особливості та цільову платформу програмного забезпечення, що розробляється, для реалізації проекту було обрано мову програмування C#. Завдяки великій різноманітності синтаксичних конструкцій та можливості роботи з платформою .NET, дана мова дозволяє швидко впроваджувати програмні задачі, характеризується надійністю та елегантністю, в зв'язку з чим найкраще підходить для розробки графічного MIDI-візуалізатора [14].

В якості середовища розробки було обрано Visual Studio 2019 — інтегроване середовище розробки програмного забезпечення, що користується надзвичайно великими популярністю та довірою, і є найкращим вибором для реалізації проекту дипломної роботи.

2.2.3 Вибір інтерфейсу для роботи з графікою

В даний момент найбільш популярними інтерфейсами для роботи з графікою є OpenGL та Direct 3D.

Direct 3D — це окрема самостійна частина бібліотеки Microsoft DirectX (яка в основному призначена для створення ігрового програмного забезпечення), що відповідає за графіку та обробку графічної інформації. Direct 3D не має розширень: усі нові можливості та вдосконалення стають доступними лише з виходом нової версії інтерфейсу.

OpenGL (англ. Open Graphics Library – відкрита графічна бібліотека) — специфікація, що визначає незалежний від мови програмування програмний

інтерфейс для написання програм, що використовують двовимірну та тривимірну комп'ютерну графіку. Даний інтерфейс складається із приблизно 250 окремих команд (майже 200 команд в ядрі OpenGL та ще 50 команд в бібліотеці сервісів OpenGL), які використовуються для того, щоб визначити об’єкти та операції, необхідні для створення інтерактивних прикладних програм.

На базовому рівні, OpenGL — просто специфікація, документ, що описує набір функцій і їх точну поведінка. Виробники обладнання на основі цієї специфікації створюють реалізації — бібліотеки функцій, що відповідають набору функцій специфікації. Бібліотека OpenGL розроблена в якості низькорівневого, апаратно-незалежного інтерфейсу, що реалізується на більшості апаратних платформ. Якщо апаратна частина не дозволяє реалізувати якусь можливість, її потрібно емулювати програмно. Розробники такої емуляції повинні пройти спеціальні тести (тести на відповідність), перш ніж реалізація стане офіційно класифікована як OpenGL-реалізація.

Таким чином, розробникам програмного забезпечення досить навчитися використовувати функції, описані в специфікації, залишивши ефективну реалізацію останніх розробникам апаратного забезпечення. Для досягнення такої цілі до складу бібліотеки OpenGL не включені ніякі команди для виконання завдань роботи з вікнами або для отримання користувацького вводу; замість цього потрібно працювати через будь-яку систему керування вікнами, що працює з конкретними апаратними засобами. Також бібліотека OpenGL не надає команд високого рівня для опису моделей тривимірних об'єктів. Такі команди могли б дозволити визначати відносно складні форми, наприклад, автомобілі, частини тіла, літаки або молекули. Основним принципом роботи OpenGL є отримання наборів векторних графічних примітивів у вигляді точок, ліній та багатокутників з подальшою математичною обробкою отриманих даних і побудовою растрового зображення на екрані і / або в пам'яті [15].

Враховуючи невелику складність графічної складової програмного забезпечення, що розробляється, та простоту користування програмним інтерфейсом OpenGL при використанні Visual Studio, специфікацію OpenGL було обрано для реалізації усіх графічних складових даної дипломної роботи.

2.2.4 Побудова UML-діаграм послідовності

Діаграма послідовності — одна з найбільш використовуваних при розробці програмного забезпечення діаграм, що відображає об'єкти та класи, що беруть участь у конкретному сценарії, та послідовність повідомлень, якими обмінюються між собою об'єкти, щоб здійснювати виконання сценарію [16].

Діаграма послідовності містить паралельні вертикальні лінії (лінії часу) — різні процеси або об'єкти, що живуть одночасно, та горизонтальні стрілки — повідомлення, якими вони обмінюються, у коректному порядку. Дану діаграму корисно застосовувати для графічного відображення простих сценаріїв, коли необхідно дослідити поведінку кількох об’єктів в рамках одного варіанту використання. Тим не менш, діаграми послідовності є хорошим вибором для представлення взаємодії об’єктів будь-якого сценарію.

На рис. 2.8 зображено діаграму послідовності для варіанту використання «Розпочати / зупинити відтворення MIDI-файлу». Користувач дає запит на відтворення MIDI-файлу, програмна складова системи розпочинає відтворення та дає запит на ввімкнення графічної візуалізації MIDI-файлу. Графічна складова системи виконує запит та відправляє повідомлення про успішність виконання запиту програмній складовій системи, яка у свою чергу відправляє повідомлення про успішність виконання запиту користувачеві. Аналогічно відбувається припинення відтворення MIDI-файлу.

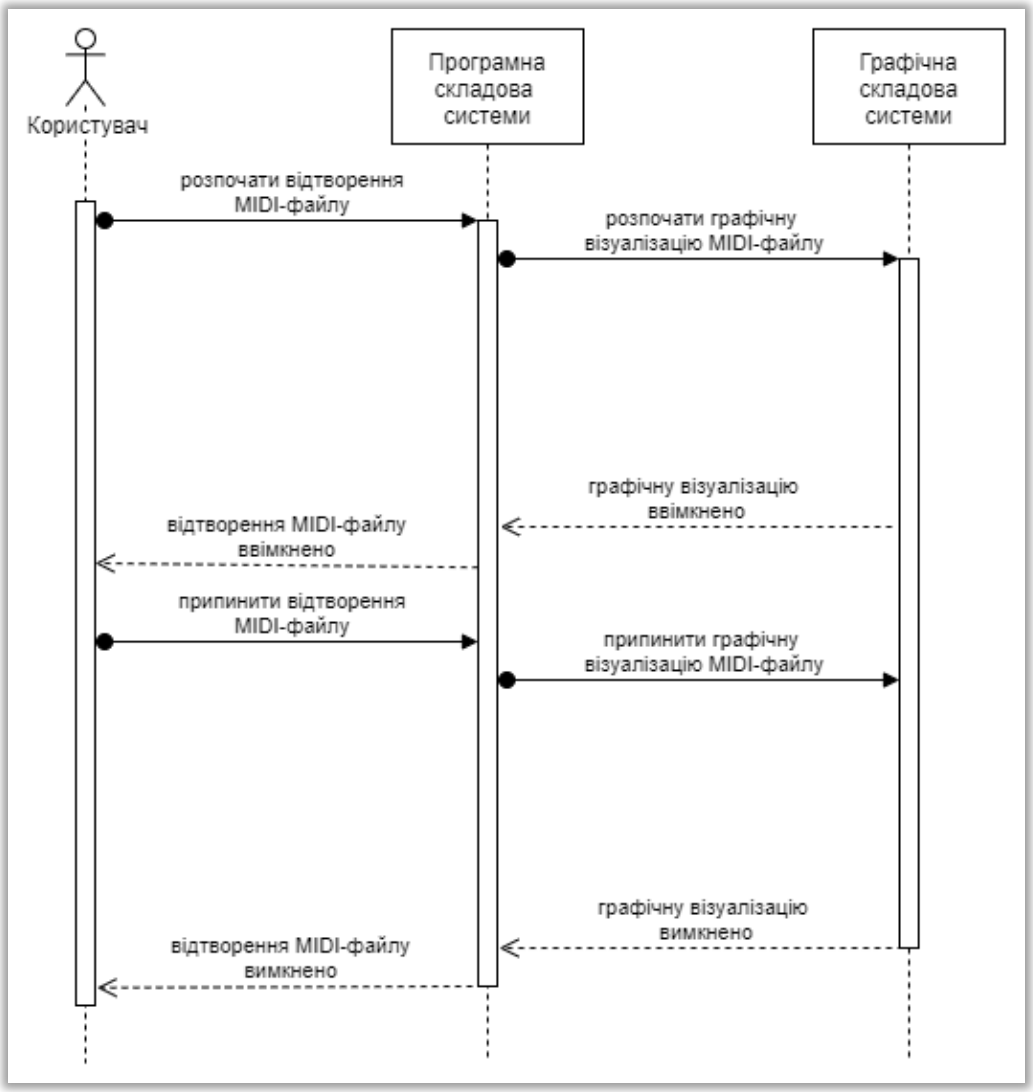

Рис. 2.8 — Діаграма послідовності варіанту використання «Розпочати / зупинити відтворення MIDI-файлу»

На рис. 2.9 зображено діаграму послідовності для варіанту використання «Змінити колір звуків MIDI-файлу та ефектів світлових частинок». Користувач дає запит на бажання зміни кольору звуків MIDI-файлу та ефектів світлових частинок, програмна складова системи повертає користувачеві повідомлення з палітрою кольорів. Здійснивши вибір нового кольору користувач надсилає нове повідомлення програмній складовій системи, яка в свою чергу дає запит на зміну кольору. Графічна складова системи виконує запит та відправляє повідомлення про успішність виконання запиту програмній складовій системи, яка у свою чергу відправляє повідомлення про успішність виконання запиту користувачеві.

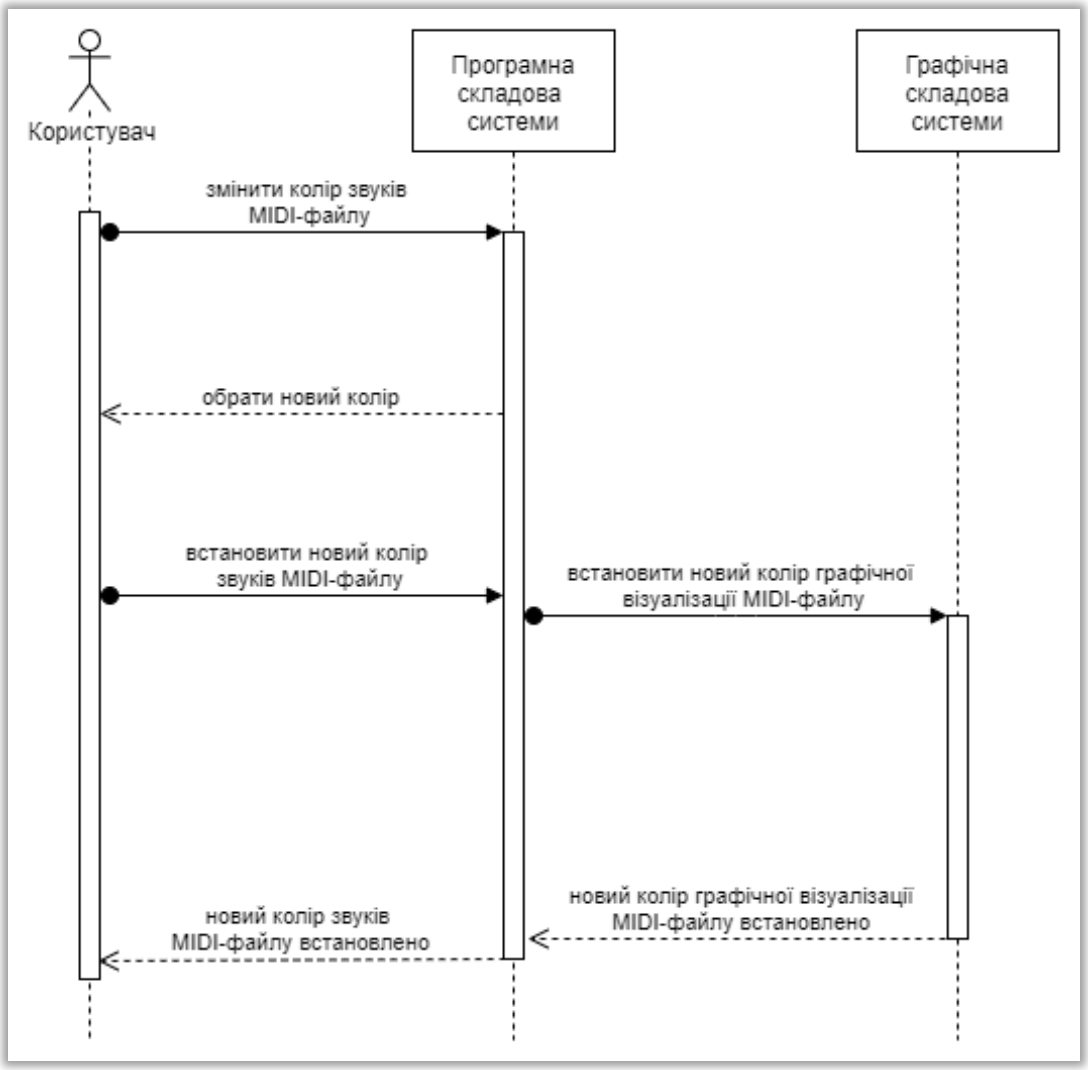

Рис. 2.9 — Діаграма послідовності варіанту використання «Змінити колір звуків MIDI-файлу та ефектів світлових частинок»

2.2.5 Побудова UML-діаграм класів

Діаграма класів — структурна діаграма, що демонструє загальну структуру ієрархії класів системи, їх атрибутів (полів), методів, інтерфейсів та взаємозв’язків. Діаграма широко застосовується не лише для документації та візуалізації, а й для конструювання методами прямого та оберненого проектування. Основною метою створення діаграми класів є графічне представлення статичної структури декларативних елементів системи (класів, типів тощо) [16].

Основними класами системи, що забезпечують її функціонал, є PianoMaster та PianoControl (див. рис. 2.10). Клас PianoMaster завантажує основні бібліотеки проекту, включає в себе методи для ініціації та побудови інтерфейсу програми, а також містить метод Main, що власне і розпочинає роботу усієї системи. Клас PianoControl відповідає за взаємодію користувача з інтерфейсом програми, завантажує та регулює основні його компоненти.

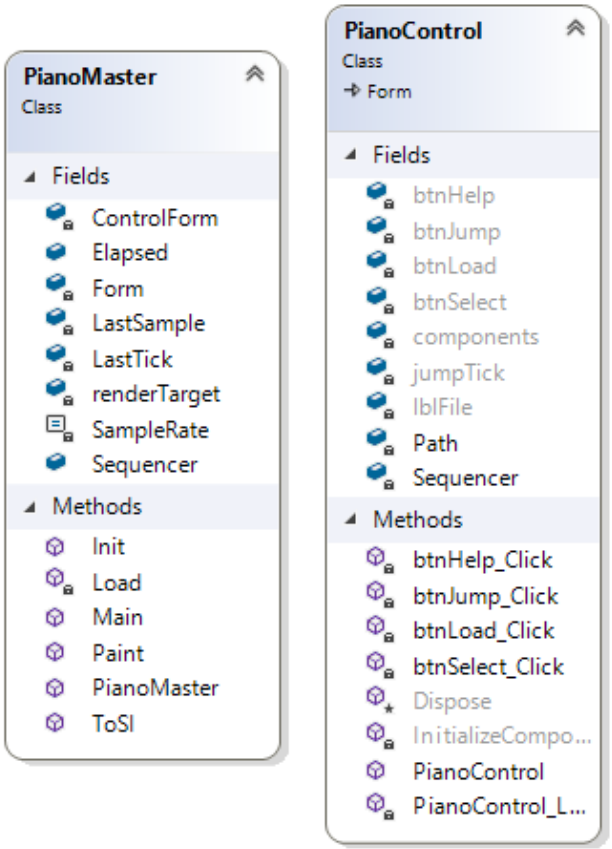

Рис. 2.10 — Діаграма основних класів проекту

Класи MIDISequencer, MIDIKeyboard, Note та NoteManager (див. рис. 2.11) відповідають за взаємодію з MIDI-файлами. Клас MIDISequencer займається відкриттям, обробкою та відтворенням MIDI-файлу, клас MIDIKeyboard — відповідає за роботу віртуальної клавіатури. Клас Note містить характеристики звуків композиції (Key, Length, Time, Position, Channel, Velocity та Playing), які зберігаються у музичному файлі, а клас NoteManager відповідає за роботу із ними та їх організацію у потік для відтворення.

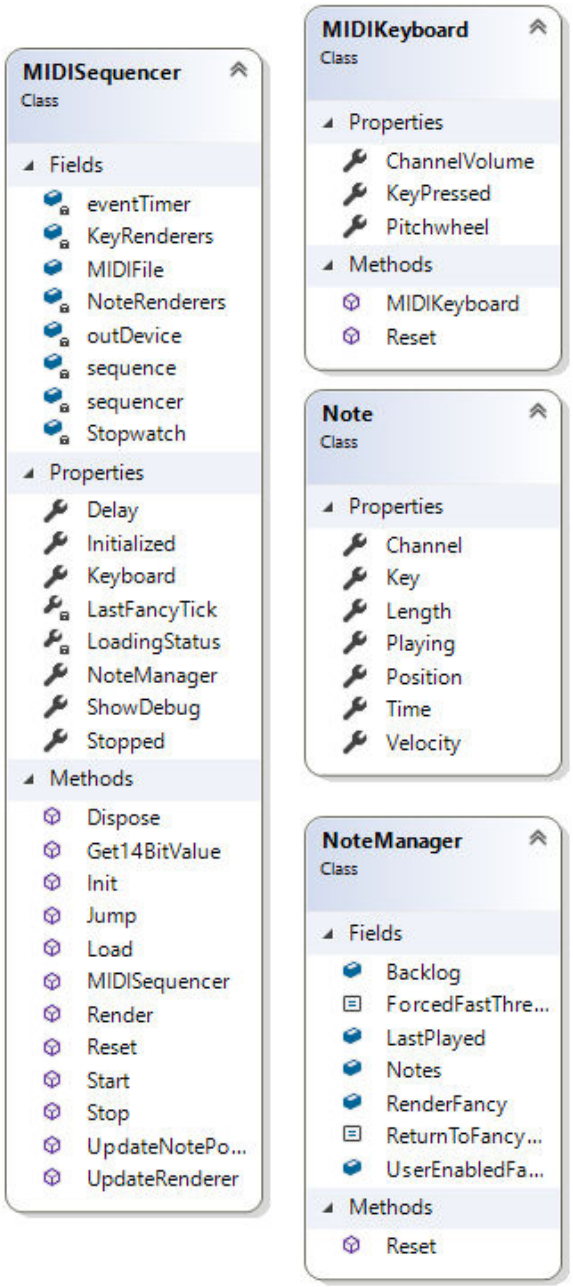

Рис. 2.11 — Діаграма класів для роботи з MIDI-файлами

Абстрактні класи NoteRenderer та KeyRenderer створені для реалізації під час відтворення MIDI-файлу графічних ефектів падаючих нот та натиснутих клавіш відповідно: класи NoteBodyRenderer та KeyBodyRenderer відповідають за графіку нот та клавіш, а класи NoteEffectRenderer та KeyEffectRenderer — за реалізацію їх світлових ефектів частинок (див. рис. 2.12).

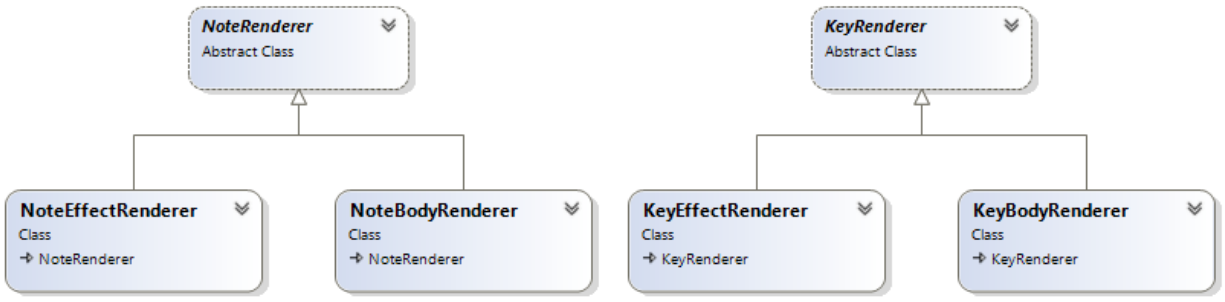

Рис. 2.12 — Діаграма класів для роботи з графікою

Загальну діаграму основних класів програмної системи зображено на рис. 2.13.

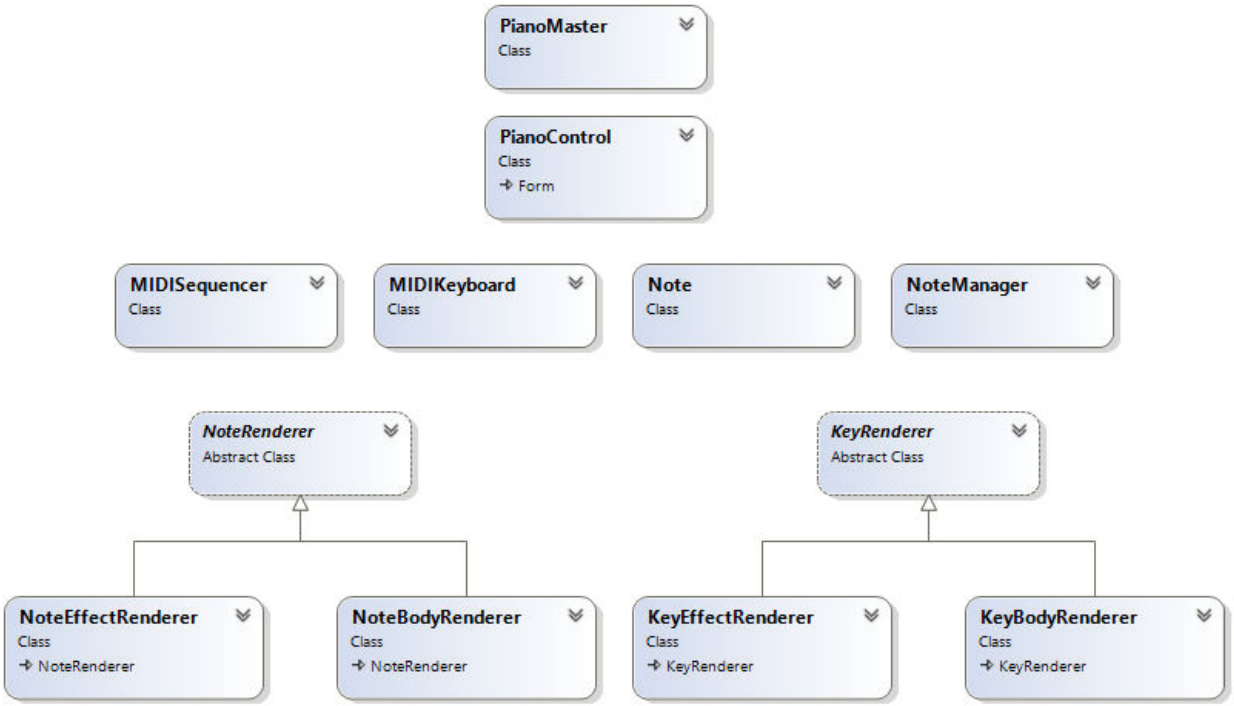

Рис. 2.13 — Діаграма основних класів програмної системи

2.3 Реалізація та тестування програмної системи

2.3.1 Реалізація та огляд програмної системи

Після запуску програми першим з’являється консольне вікно, що викликає вікно для вибору файлу, щоб завантажити MIDI-файл у програмну систему (див. рис. 2.14).

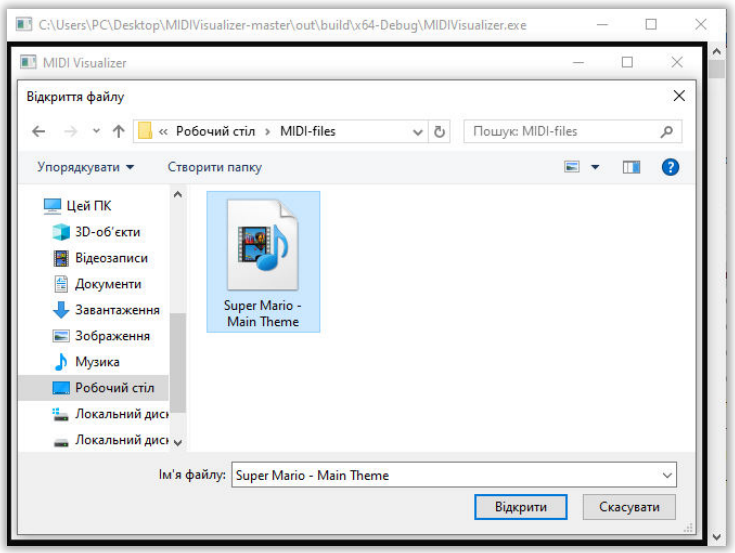

Рис. 2.14 — Вікно для вибору файлу

Після завантаження MIDI-файлу у систему графічний візуалізатор стає готовим до роботи, його початковий стан зображено на рис. 2.15.

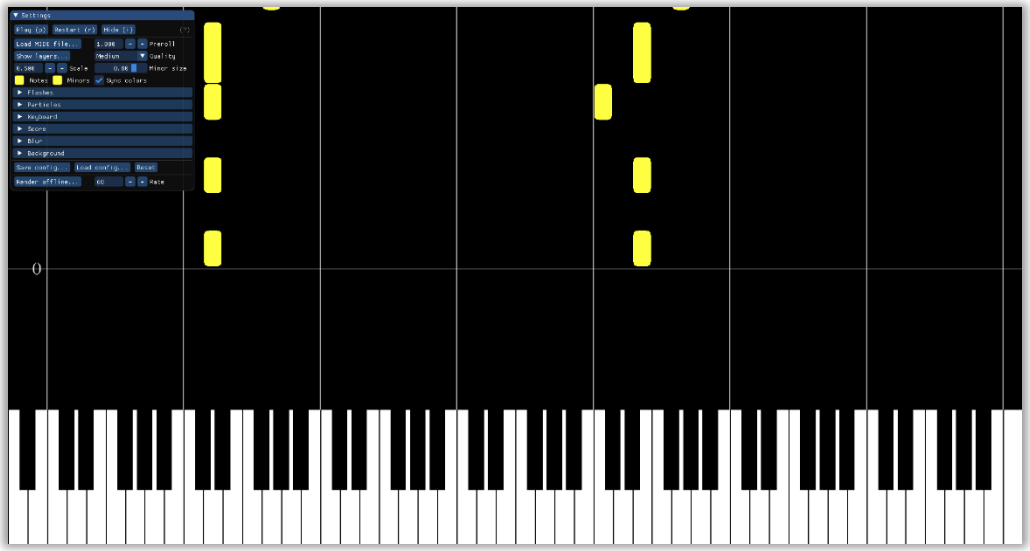

Рис. 2.15 — Головне вікно MIDI-візуалізатора

У верхньому лівому куті знаходиться прозоре вікно налаштувань (див. рис. 2.16), яке можна вільно переміщувати та приховувати під час відтворення MIDI-файлу.

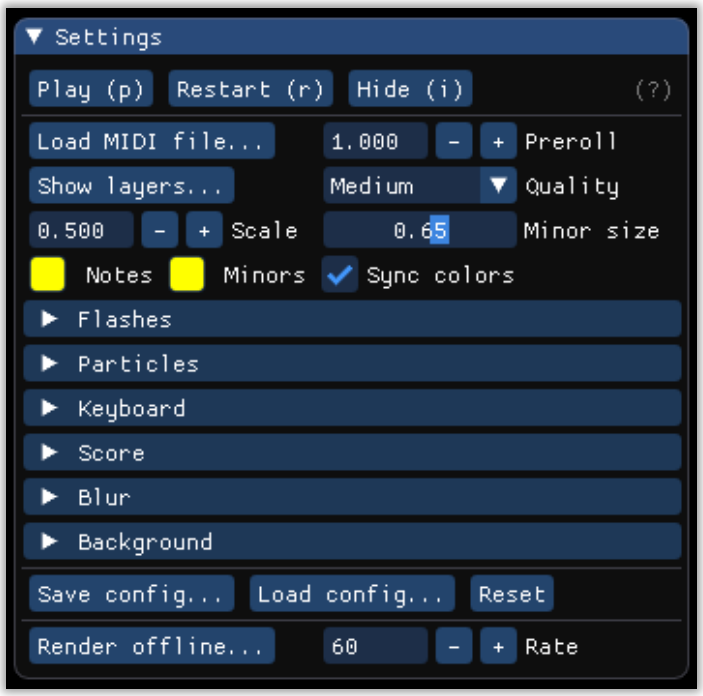

Рис. 2.16 — Вікно налаштувань MIDI-візуалізатора

Кнопки «Play» / «Pause», «Restart» та «Hide» відповідають за старт / припинення відтворення, запуск відтворення з початку та приховання вікна налаштувань. Їм відповідають клавіші клавіатури «p», «r» та «i».

Кнопка «Load MIDI file…» дозволяє завантажити новий MIDI-файл на зміну попередньому. Меню «Show layers…» призначене для зміни порядку шарів графічної візуалізації. Параметр «Preroll» задає початкову позицію відтворення MIDI-файлу, параметр «Quality» — якість світлових ефектів частинок. Параметр «Scale» призначений для зміни масштабу графічних елементів MIDI-файлу, а параметр «Minor size» — для налаштування співвідношення ширини білих та чорних клавіш віртуальної клавіатури.

Кнопки «Save config…» та «Load config…» дозволяють відповідно зберегти та завантажити уже збережені налаштування MIDI-візуалізатора,

кнопка «Reset» — повернути програму до налаштувань за замовчуванням. Параметр «Rate» задає частоту кадрів графічної візуалізації.

Параметри вибору кольору «Notes» та «Minors» призначені для вибору кольору падаючим нотам, що відповідають білим та чорним клавішам відповідно, параметр «Sync colors» — для встановлення одного кольору усім графічним елементам.

На рис. 2.17 відображено функціонал, що прихований у вкладках «Score» та «Background». Вкладка «Score» призначена для ввімкнення / вимкнення відображення на екрані горизонтальних ліній, що відображають поділ музичної композиції на такти, та вертикальних ліній, що відображають поділ клавіатури фортепіано на октави. Параметр вибору кольору «Lines» дозволяє обрати колір цих ліній. Вкладка «Background» дозволяє обрати фоновий колір або ж завантажити фонове зображення з можливістю встановити значення його прозорості.

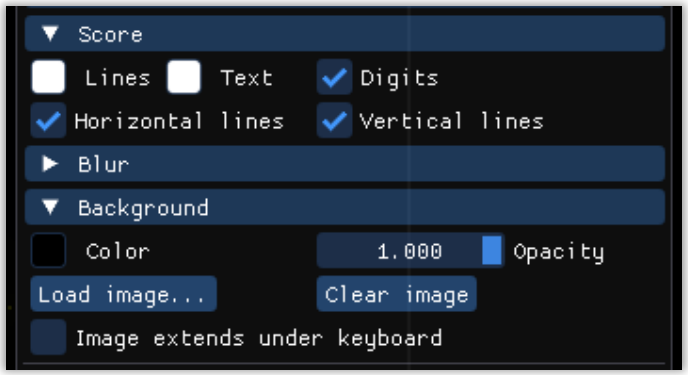

Рис. 2.17 — Функціонал вкладок налаштувань «Score» та «Background»

На рис. 2.18 відображено функціонал, прихований у вкладці «Keyboard». Вона призначена для встановлення іншого кольору чорним клавішам клавіатури, ввімкнення / вимкнення підсвітки натиснутих клавіш та встановлення її кольору.

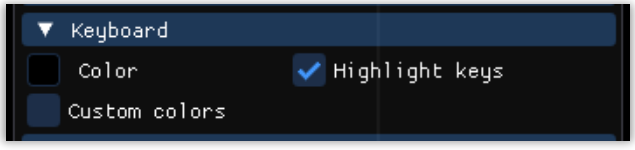

Рис. 2.18 — Функціонал вкладки налаштувань «Keyboard»

На рис. 2.19 відображено функціонал, прихований у вкладках «Flashes», «Particles» та «Blur». Вкладка «Blur» слугує для ввімкнення / вимкнення та налаштування сили підсвітки падаючих нот, вкладка «Flashes» — для встановлення кольору та розміру спалахів, що з'являються над клавіатурою в момент падіння нот на відповідні клавіші.

Вкладка «Particles» призначена для налаштування ефектів частинок: параметр кольору «Color» встановлює колір частинок, параметр «Size» — їх розмір, параметр «Count» — кількість, параметр «Speed» дозволяє встановлювати швидкість поширення частинок, а параметр «Expansion» — масштаб їх поширення. Кнопка «Load images…» дозволяє завантажити в ролі частинок користувацькі зображення.

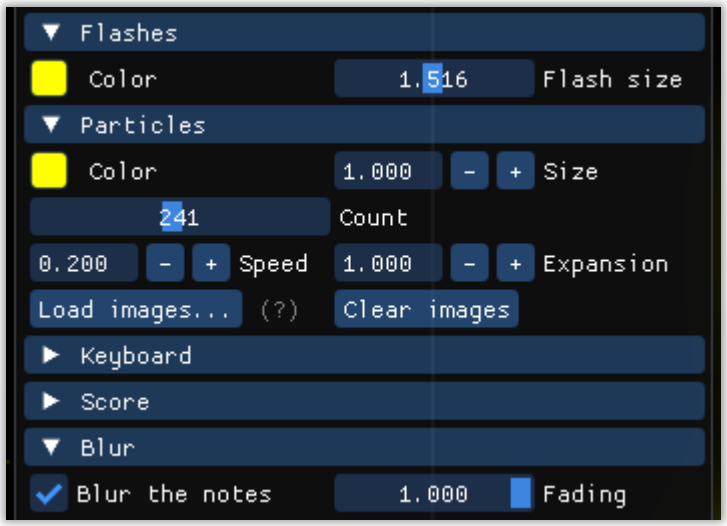

Рис. 2.19 — Функціонал вкладок налаштувань «Flashes», «Particles» та «Blur»

Результати змін описаних вище налаштувань MIDI-візуалізатора продемонстровано на рис. 2.20 – 2.21, аркуш 51. На рис. 2.20 кольором усіх графічних елементів обрано синій, зменшено розмір чорних клавіш клавіатури, збільшено об'єм спалахів, швидкість та масштаб поширення частинок. На рис. 2.21 кольором усіх графічних елементів обрано жовтий,

розмір чорних клавіш клавіатури зменшено, об’єм спалахів, швидкість та масштаб поширення частинок збільшено.

У обох прикладах вимкнено горизонтальні лінії, ввімкнено на максимум підсвітку падаючих нот, встановлено користувацькі фонові зображення та обрано максимальний рівень якості світлових ефектів частинок.

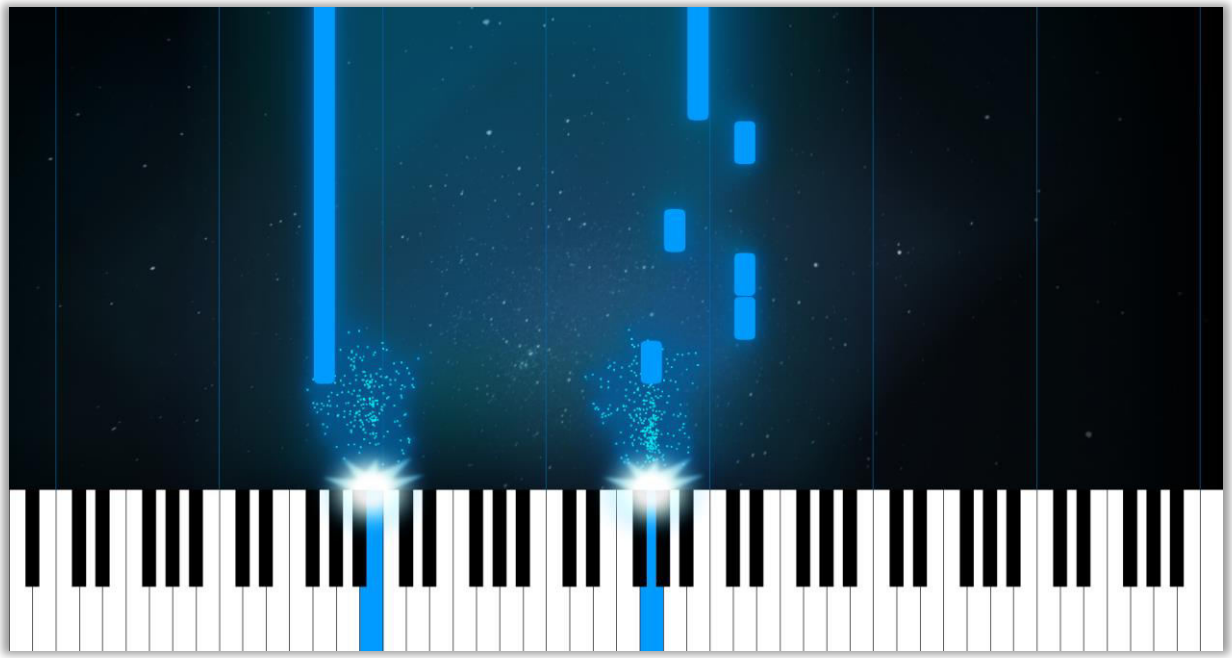

Рис. 2.20 — Встановлення нових користувацьких налаштувань MIDI-візуалізатора

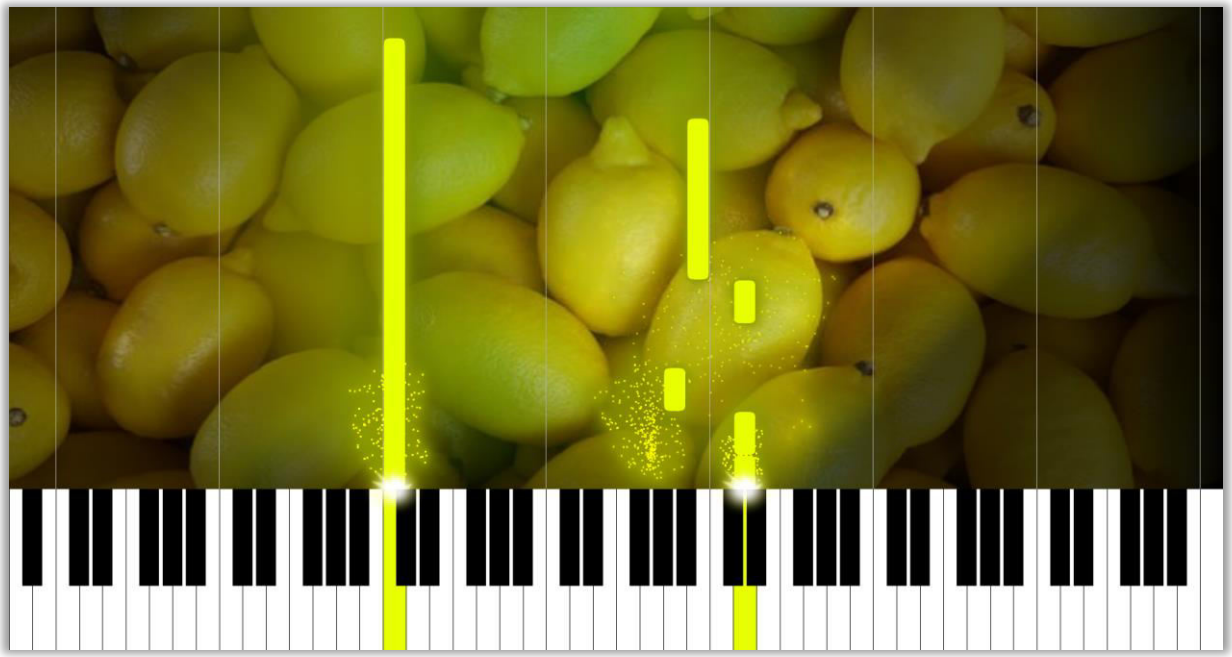

Рис. 2.21 — Встановлення нових користувацьких налаштувань MIDI-візуалізатора

2.3.2 Тестування та перевірка програмної системи

В зв’язку з специфічними сферою та призначенням створеного програмного забезпечення для перевірки його працездатності було обрано ручне тестування. Даний вид тестування може проводитися як тестером, так і звичайним користувачем шляхом моделювання можливих сценаріїв роботи програми. Головна мета ручного тестування — попередньо переконатися у працездатності створеного програмного забезпечення, впевнитись у відсутності помилок та отримати успішні результати роботи програми.

Враховуючи використання програмною системою графічних елементів також є доцільним перевірити навантаження, яке чинить графічний візуалізатор на ресурси персонального комп’ютера за допомогою диспетчера завдань Windows (див. рис. 2.22).

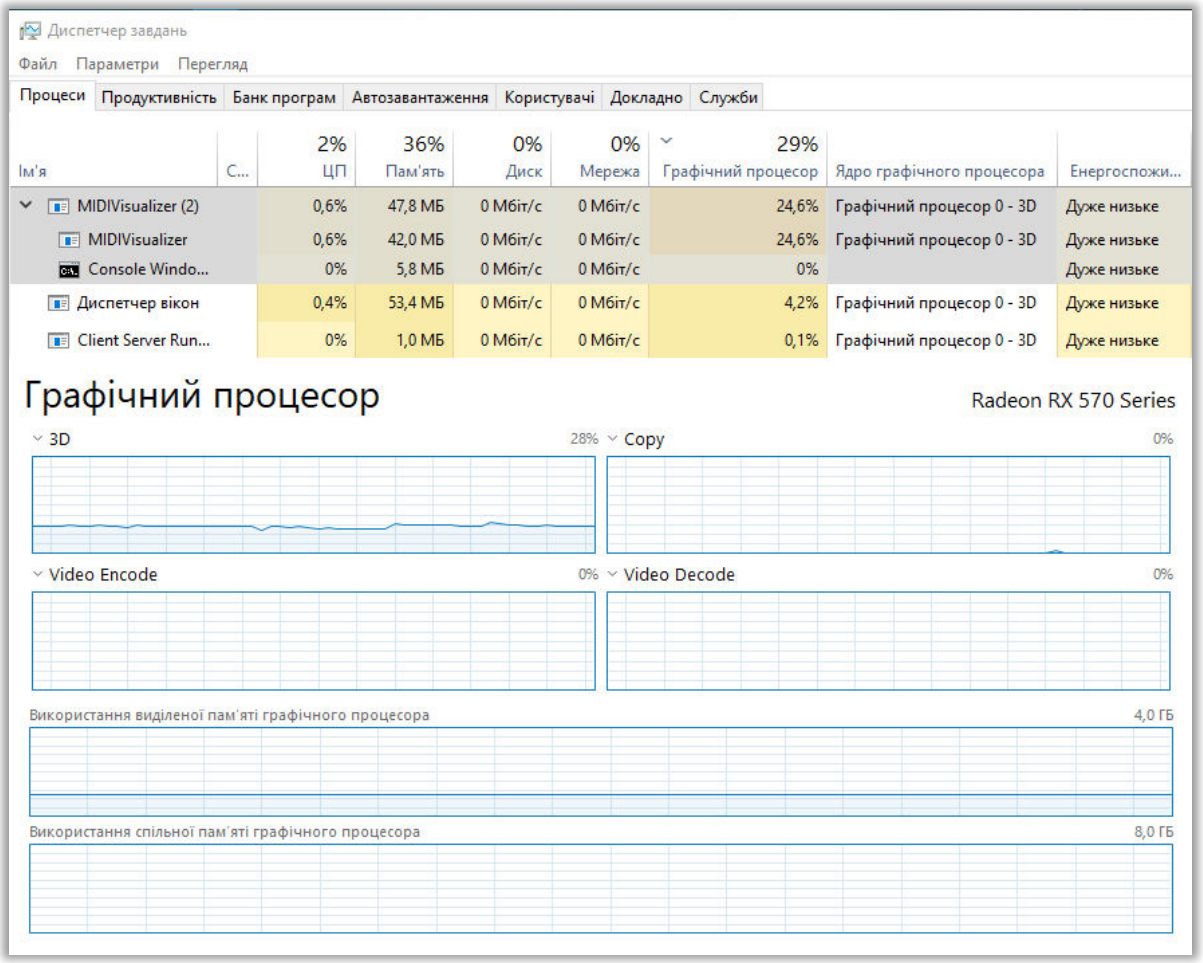

Рис. 2.22 — Дані диспетчера завдань Windows

Результати перевірки показали, що графічний візуалізатор займає лише 48 МБ оперативної пам’яті персонального комп’ютера та завантажує його графічний процесор лише на 25%, що повністю задовольняє вимоги, поставлені до створеного програмного забезпечення. Результати ручного тестування показали, що графічний візуалізатор MIDI-файлів відмінно виконує покладені на нього завдання.

3 ОРГАНІЗАЦІЙНО-ЕКОНОМІЧНА ЧАСТИНА

3.1 Планування етапів проектування програмного забезпечення

Програмна інженерія — процес створення програмного забезпечення, що включає велику кількість різноманітних етапів: виявлення та аналіз вимог до програмної системи, проектування та моделювання архітектури (вибір процесу розробки, мови та інших інструментів для досягнення цілі, побудова діаграм та схем), конструювання та відлагодження програмних модулів, тестування та супровід створеного програмного забезпечення тощо.

Етапи життєвого циклу розробки проекту даної дипломної роботи магістра представлено у таблиці 3.1.

Таблиця 3.1 — Перелік етапів життєвого циклу розробки проекту

№	Етапи	Стадії	Виконавці	Артефакти
1	Аналіз вимог до програмної системи	Огляд існуючих програм та аналіз предметної області, Постановка задачі. Пошук акторів та варіантів використання	Аналітик предметної області, специфікатор варіантів використання	Актори та варіанти використання, модель програмної системи
2	Проектування програмної системи	Вибір процесу розробки, вибір мови та середовища програмування, вибір графічного інтерфейсу, побудова UML-діаграм послідовності та UML-діаграм класів	Архітектор програмної системи, графічний архітектор, інженер варіантів використання	Модель проектування, модель арітектури UML-діаграма послідовно-стей, UML-діаграма класів
3	Конструюван-ня програмної системи	Реалізація класів та методів програмної системи, відлагодження та розгортання програмної системи	Інженер-програміст, системний інтегратор	Компоненти та підсистеми, розгорнута програмна система

4	Тестування програмної системи	Тестування основних компонентів системи, тестування другорядних компонентів системи, оцінка результатів тестування	Інженер з розробки системи тестування, Тестер програмного забезпечення	Модель тестування, результати тестування

3.2 Оцінка економічної ефективності та розрахунок витрат

Ринок інформаційних продуктів та послуг є системою економічних, правових та організаційних відносин галузі торгівлі продуктами інтелектуальної праці на комерційних засадах. Він характеризується визначеною номенклатурою продуктів і послуг, умовами та механізмами їх надання, загальноприйнятою системою встановлення цін. Предметами продажу або обміну на інформаційному ринку виступають прикладне програмне забезпечення, інформаційні системи та технології, ліцензії, патенти, товарні знаки, інженерно-технічні послуги та інші види інформаційних ресурсів.

Метою даної дипломної роботи є розробка графічного візуалізатора MIDI-файлів, яка є актуальною водночас і для її автора, в зв’язку з його безпосередньою діяльністю у музичній сфері, і через стрімку появу попиту у музикантів на програмне забезпечення такого типу. Поняття MIDI-візуалізатора було введено в обіг у 2006 році з розробкою музичної тренувальної програми «Synthesia». З появою відеохостингу YouTube дана програма стала надзвичайно популярною завдяки тому, що музиканти почали записувати процес відтворення композиції у «Synthesia», публікуючи отриманий файл як відео-урок на своєму каналі. До таких музикантів належить і автор даної дипломної роботи, чий YouTube-канал «Gamber Piano Tutorials» за 6 років свого існування отримав більше 32 тисяч підписників та 14 мільйонів переглядів.

В останньому десятилітті YouTube-виконавці створили свої аналоги програми «Synthesia», відтворення MIDI-файлів у яких стало візуально привабливішим, із чим прямо пропорційні ріст популярності та комерційної вигоди YouTube-каналу. У свої версії MIDI-візуалізаторів вони впроваджують світлові ефекти та ефекти частинок, фонові зображення та графічні маніпуляції із ними, нові текстури клавіатури та кольорові схеми, що відчутно впливає на збільшення їх доходу. В якості прикладу приблизної моделі комерційного успіху було розглянуто YouTube-канал «Patrik Pietschmann», який після впровадження MIDI-візуалізатора нового типу отримав більше 900 тисяч нових підписників та більше 170 мільйонів переглядів, що свідчить про отримання каналом 50 тис. доларів приблизного річного доходу. Аналогічний ріст популярності та комерційної вигоди очікується автором роботи щодо його YouTube-каналу.

Усі працівники IT-сфери, що залучені до проекту мають встановлений певний посадовий оклад. Місячний оклад, денну заробітну плату, трудомісткість (днів) і основну заробітну плату кожного учасника технічного процесу представлено у таблиці 3.2.

Таблиця 3.2 — Розрахункова вартість технологічного процесу розробки

Посада	Місячний оклад, грн.	Денна зар. плата, грн.	Об'єктно-орієнтований підхід		Процедурний підхід	
			Днів	Сума, грн.	Днів	Сума, грн.
Керівник	9944,00	452	15	6780,00	14	6328,00
Розробник	11264,00	512	14	7168,00	13	6656,00
Графічний дизайнер	8580,00	390	8	3120,00	6	2340,00
Тестер	6600,00	300	5	1500,00	3	900,00
Додаткова зар. плата 20%			27	3713,60	22	3244,80

Фонд оплати праці 36,77%	6827,45	5965,56
Всього витрат на зар. плату	29109,05	25434,36
Військовий збір 1,5%	436,64	381,52
Єдиний соціальний внесок 3,6%	1047,93	915,64
ПДВ, 15%	4366,36	3815,15
Всього	**34959,97**	**30546,67**

Розрахунок витрат на науково-дослідницьку роботу та здійснення розробки програмних продуктів об'єктно-орієнтованим та процедурним способами потребують наступних обчислень.

Основна заробітна плата:

$ЗП_{осн\ 1} = 18568$ грн;

$ЗП_{осн\ 2} = 16224$ грн.

Додаткова заробітна плата (обчислюється як $ЗП_{дод} = 0,2 \cdot ЗП_{осн}$):

$ЗП_{дод\ 1} = 0,2 \times 18568$ грн $= 3713,6$ грн;

$ЗП_{дод\ 2} = 0,2 \times 16224$ грн $= 3244,8$ грн.

Нарахування на фонд оплати праці (ФОП):

$ФОП_{ЄСВ} = 0,3677 \times ФЗП$

$ФОП_{ЄСВ1} = 0,3677 \times 22281,60$ грн $= 8192,95$ грн;

$ФОП_{ЄСВ1} = 0,3677 \times 19468,80$ грн $= 7158,68$ грн.

Всього витрат:

$В_{ЗП1} = ЗП_{осн\ 1} + ФОП_{ЄСВ1} + ЗП_{дод1} = 30474,55$ грн;

$В_{ЗП2} = ЗП_{осн\ 2} + ФОП_{ЄСВ2} + ЗП_{дод2} = 38937,60$ грн.

З цієї суми утримуються обов'язкові відрахування на заробітну плату: єдиний соціальний внесок, який складає 3,6% від суми нарахованої заробітної плати та податок на доходи фізичних осіб, який складає 15% від суми нарахованої заробітної плати, зменшеної на суму єдиного внеску на загальнообов'язкове соціальне страхування та податкової соціальної пільги, військовий збір у розмірі 1,5%, від суми нарахувань.

До окремих витрат також відносяться витрати на куплені вироби (матеріальне забезпечення) та спеціальне обладнання для експериментів, накладні витрати. Витрати, що будуть супроводжувати проект, порівнюватиметься в обох можливих підходах розробки.

Матеріальні витрати визначаються як добуток кількості витрачених матеріалів та їх ціни (формула 3.1).

$$M_{Bi} = q_i \cdot p_i \,, \tag{3.1}$$

де q_i – кількість витраченого матеріалу i-го виду;

p_i – ціна матеріалу i-го виду.

Матеріальні витрати в рамках проекту наведені в таблиці 3.3. Загальна сума матеріальних витрат становить 2188 гривень.

Таблиця 3.3 – Матеріальні витрати

Найменування ресурсу	Кількість, шт.	Ціна одиниці, грн	Загальна сума, грн
Флешки	5	250,00	1250,00
Папір для друку А4, арк.	1000	0,178	178,00
Тонер для принтера	2	80,00	160,00
Дошка для записів	1	500,00	500,00
Перманентний маркер	10	10,00	100,00
Всього			2188.00

Розрахунок витрат на електроенергію одиниці обладнання визначаються за формулою 3.2:

$$З_в = W * T * S, \tag{3.2}$$

де W – необхідна потужність, кВт; T – кількість годин роботи обладнання; S – вартість кіловат-години електроенергії, S = 2,50 грн/кВт·год.

$З_{в1}$ =1,2 * 336 * 2,50 = 1008,00 грн;

$З_{в2}$ =1,2 * 288 * 2,50 = 864,00 грн;

Розрахунок суми амортизаційних відрахувань. Комп'ютери та оргтехніка належать до четвертої групи основних фондів. Для цієї групи річна норма амортизації дорівнює 60 %, вартість яких перевищує 1000 грн. і визначається за формулою:

$$A = \frac{C_Б \cdot N_A \cdot T_{ФАК}}{T_{ГОД}} \quad (3.3)$$

де $C_Б$ – балансова вартість обладнання, грн; N_A – норма амортизаційних відрахувань в рік, %; $T_{ФАК}$ – фактичний час роботи обладнання по написанню програми, год; $T_{ГОД}$ – річний робочий фонд часу, год.

У даній формулі норма відрахувань на амортизацію рівна N_A = 0,6. Балансова вартість обладнання вказана в таблиці 3.3 і рівна $C_Б$ = 37800 гривень. Річний робочий фонд часу приймемо за $T_{ГОД}$ = 2112 годин. З них реальний фактичний робочий час становить $T_{ФАК}$ = 336 години згідно об'єктно-орієнтованого підходу та $T_{ФАК}$ = 288 годин згідно процедурного підходу.

Згідно вищезгаданої формули витрати на амортизацію становлять 2310,79 гривень та 1882,87 гривень для кожного підходу відповідно.

Таблиця 3.4 – Перелік необхідного обладнання

Найменування	Кількість, шт	Ціна, грн	Сума, грн
Комп'ютер	3	11200,00	33600,00
Принтер	1	4200,00	4200,00

Середовища розробки	3	безкоштовно	безкоштовно	
Операційна система	3	безкоштовно	безкоштовно	
Всього більше 1000 грн.			37800,00	
Всього витрат на амортизацію			2310,79	1882,87
Всього			40110,79	39682,87

Накладні витрати, пов’язані з обслуговуванням виробництва, утриманням апарату управління та створення необхідних умов праці та закупівлю ресурсів та обладнання для розробки, наведено в таблиці 3.4.

Залежно від організаційно-правової форми діяльності господарюючого суб’єкта, накладні витрати можуть становити 20–60 % від суми основної та додаткової заробітної плати працівників. Нехай вона буде дорівнювати 40%, що становить 13983,99 грн для об’єктно-орієнтованого і 12218,67 грн для процедурного підходу розробки.

Проведемо розрахунок вартості створюваного програмного продукту. Вартість продукції включає у себе собівартість і планований прибуток. Найважливішим моментом для розробника, з економічної точки зору, є процес встановлення ціни.

Можна отримати загальні значення витрат на розробку та реалізацію проекту, враховуючи всі вище описані затрати та нарахування. Цей вид витрат складається з сум витрат на оплату праці (всього витрати на оплату праці), матеріальні затрати, затрати на електроенергію, накладні витрати, витрати на обладнання, враховуючи амортизації обладнання на час виконання проекту.

Собівартість продукції — це сума грошових витрат підприємства (фірми) на виробництво і збут одиниці продукції, виконання робіт та надання послуг [17].

Повна собівартість програмного продукту дорівнює сумі усіх витрат на його виробництво: 91890,76 грн при об’єктно-орієнтованому підході та 85164,21 грн при процедурному підході розробки.

Ефективність виробництва — це узагальнене і повне відображення кінцевих результатів використання робочої сили, засобів та предметів праці на підприємстві за певний проміжок часу.

Економічна ефективність (Е) полягає у відношенні результату виробництва до затрачених ресурсів:

$$E = \frac{П}{Св} \tag{3.4}$$

де *П* – прибуток, *П= В - Св; Св* – собівартість.

У випадку даної розробки, маючи некомерційний проект без економічно корисного результату, можна прогнозувати, що економічна ефективність прямує до 0 у обох випадках. Однак це не є причиною для негативного економічного висновку щодо даного проекту, адже такого плану розробки приносять користь у вигляді інтелектуальних ресурсів, і, переважно, фінансуються або виконуються на замовлення організацій, зацікавлених в отриманні результатів досліджень та розробок. Фінансування не можна вважати отриманим доходом від реалізації. Однак за надходження коштів на реалізацію ззовні можна вважати ефективність проекту рівною 1 (Е = 1), що означає перекриття витрат на розробку у повній мірі, тобто фінансування.

Якщо ринкова вартість програмного продукту рівна прийнятій, то економічна ефективність визначається встановленим рівнем прибутку. Поряд із економічною ефективністю розраховують термін окупності капітальних вкладень ($T_{ок}$):

$$T_{ок} = \frac{1}{E} \tag{3.5}$$

У нашому випадку прямого прибутку не існує. Прибуток можна прогнозувати на підприємствах, організаціях чи відомствах, що зацікавлені в дослідженні. Окупність же для розробки даного ПЗ за ефективності рівній одиниці можна вважати теж рівній 1 згідно формули 3.4.

Виходячи із експертних оцінок і складності програми, приймемо величину витрат на супровід і модернізацію програмного забезпечення, створеного за об'єктно-орієнтованим методом 25% (22972,69 грн) від початкових витрат, а за процедурним – 30% (25549,26 грн).

Однак варто зауважити, що розробка спрямована на короткотривалу підтримку та не прогнозує модернізації. У разі ж необхідності розробки такого ж або суміжного ПЗ можна вважати за доцільно розпочинати розробку з початкових етапів, що потягне за собою нові витрати у повній мірі. Тобто у разі короткотермінової підтримки все ж за доцільніше обирати об'єктно-орієнтовану модель розробки, адже вартість короткотермінової підтримки згідно цієї моделі не вплине значно на сукупну вартість розробки.

Здана в експлуатацію система не завжди цілком завершена, її треба змінювати протягом терміну експлуатації. Внаслідок змін система стає більш складною і погано керованою. Об'єктно-орієнтоване представлення програми дозволяє навіть середньому програмісту швидко і ефективно супроводжувати і модернізувати програми, що значно скорочує подальші витрати на супровід і модернізацію [17].

Сумарні дані економічного розрахунку розробки даного проекту наведені в таблиці 3.5.

Таблиця 3.5 – Загальні витрати

Вид витрат	Об'єктно-орієнтований підхід, грн	Процедурний підхід, грн

Зарплата основна	18568,00	16224,00
Зарплата додаткова	3713,6	3244,8
Фонд заробітної плати	22281,60	19468,80
Відрахування на ФОП	8192,95	7158,68
Разом на оплату праці	30474,55	38937,60
Матеріальні витрати	2188,00	2188,00
Електроенергія	648,00	528,00
Амортизація	2310,79	1882,87
Накладні витрати	13983,99	12218,67
Разом на інші витрати	19130,78	16817,54
Обладнання	41052,23	41373,17
Собівартість	91890,76	85164,21
Прибуток	відсутній	відсутній
Вартість розробленого ПЗ	91890,76	85164,21
Модернізація і супровід	22972,69	25549,26
Загальні витрати на розробку	114863,44	110713,47
Економія ($ЗВ_1$-$ЗВ_2$)	**4149,97**	

Загальна вартість пропонованих робіт становить 114863,44 гривень для процедурного і 110713,47 гривень для об’єктно-орієнтованого підходів розробки. В даному випадку реалізації проекту варто вибрати об’єктно орієнтований підхід для розробки даного ПЗ, адже фінансово це більш вигідно. Також у даній методиці розробки кращі часові рамки та перспективи підтримки і модернізації. У оцінці вартості продукту варто враховувати можливість фінансування та спонсорування проекту, що дозволить гнучко та ефективно підійти до розробки та організації праці.

4 ОХОРОНА ПРАЦІ ТА БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ

4.1 Охорона праці

Створення програмного забезпечення повинно завжди відбуватись в умовах нанесення мінімальної шкоди здоров'ю розробника. Метою дипломної роботи магістра є розробка середовища графічного візуалізатора MIDI-файлів, що означає виконання великого обсягу роботи за персональним комп'ютером протягом тривалого часу, тому є доцільним відзначити важливість дотримання норм охорони праці та техніки безпеки під час роботи з електронно-обчислювальними машинами, що регулюються НПАОП 0.00-7.15-18 «Вимоги щодо безпеки та захисту здоров'я працівників під час роботи з екранними пристроями», Державними санітарними правилами і нормами роботи з візуальними дисплейними терміналами електронно-обчислювальних машин ДСанПІН 3.3.2.007-98, Санітарними нормами виробничого шуму, ультразвуку та інфразвуку ДСН 3.3.6.037-99 та Санітарними нормами мікроклімату виробничих приміщень ДСН 3.3.6.042-99 [18].

Згідно розглянутого нормативного документу під час облаштування робочого місця користувача персонального комп'ютера необхідно обирати таке устаткування, що не створює зайвого шуму та не виділяє надлишкового тепла. Рівень шуму на робочому місці особи, що працює з персональним комп'ютером, має відповідати вимогам ДСН 3.3.6.037-99. У цьому документі описано класифікацію виробничих акустичних коливань, а саме шумів, ультразвуку та інфразвуку, методи їх вимірювання та прилади, призначені для цього, та нормативи виробничого шуму, ультразвуку та інфразвуку [19].

Організація робочого місця розробника середовища графічного візуалізатора MIDI-файлів має відповідати ергономічним вимогам ГОСТ 12.2.032-78 «Система стандартов безопасности труда. Рабочее место при

выполнении работ сидя. Общие эргономические требования» та уже згаданим Державним санітарним правилам і нормам роботи з візуальними дисплейними терміналами електронно-обчислювальних машин ДСанПІН 3.3.2.007-98 [20].

Конструкція робочого місця розробника повинна відповідати сучасним вимогам ергономіки, характеру виконуваної роботи і забезпечити оптимальне розміщення на робочій поверхні обладнання персонального комп'ютера (монітора, системного блоку, клавіатури, пристрою «миша», принтера та інших периферійних пристроїв з урахуванням їх кількості та конструктивних особливостей).

Монітор на робочому місці встановлюється так, щоб верхній край екрана знаходився на рівні очей. Розташування монітора має забезпечувати безпечність роботи в цілому, зручність та ефективність зорової роботи з екраном в вертикальній площині під кутом ±30° від лінії зору, а площина екрану при цьому має бути перпендикулярною нормальній лінії зору користувача.

Робочі місця мають бути спроектовані так і мати такі розміри, щоб працівники мали простір для зміни робочого положення та рухів. Площа, виділена на одне робоче місце, має становити не менше 6,0 $м^2$, а об'єм — не менше 20,0 $м^3$.

Мікроклімат приміщення з робочим місцем розробника з екранним пристроєм має підтримуватись на постійному рівні та відповідати вимогам ДСН 3.3.6.042-99 [21]. Висока чи низька температура повітря в приміщенні з персональним комп'ютером негативно впливає на функціональний стан його користувача. Недостатня вологість в приміщенні призводить до надмірного висихання слизових оболонок очей, носа, горла та до нагромадження зарядів статичної електрики, що утворюються в процесі роботи комп'ютера. Водночас вологість повітря більше 75% також є недопустимою. При роботі за персональним комп'ютером оптимальними умовами мікроклімату є

температура повітря від +22°С до +24°С, відносна вологість 60-40%, швидкість руху повітря не більш 0,1 м/с.

4.2 Освітлення виробничих приміщень для роботи з ВДТ

Під час розробки програмного забезпечення є надзвичайно важливим забезпечення контролю за правильним освітленням під час використання комп’ютерної техніки, адже його відсутність може призвести до нанесення невідворотної шкоди здоров’ю розробнику. Всесвітньою організацією охорони здоров’я ще в 1989 р. у публікації «Відеодисплейні термінали та здоров’я користувачів» було підкреслено, що робота за персональними комп’ютерами супроводжується зоровим та нервово-емоційним напруженням, в зв’язку з чим необхідно строго дотримуватись встановлених правил освітлення виробничих приміщень для роботи з ВДТ.

Відеодисплейний термінал (ВДТ) — це частина електронно-обчислювальної машини, що містить пристрій для візуального відображення інформації, тобто монітор. Перелік нормативно-правових актів, що регулюють техніку безпеки та конкретно питання роботи з ВДТ, є досить широким. Основними з таких є наказ №207 від 14 лютого 2018 р. «Про затвердження Вимог щодо безпеки та захисту здоров'я працівників під час роботи з екранними пристроями», наказ №443 від 5 вересня 2013 р. «Про затвердження Примірної інструкції з охорони праці під час експлуатації електронно-обчислювальних машин», та Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин ДСанПіН 3.3.2.007-98, затверджені постановою Головного державного санітарного лікаря України від 10 грудня 1998 р. №7.

Згідно останнього документу з переліку приміщення для роботи з ВДТ повинні мати природне та штучне освітлення відповідно до ДБН В.2.5-28-2018 «Природне і штучне освітлення», що замінили свої попередні версії, а перед тим СНиП II-4-79. Документ набув чинності 28 лютого 2019 р. та

поділив освітлення на такі три основних види: природне, штучне та суміщене [22].

Природне освітлення — це освітлення приміщень світлом неба (природним або денним світлом) та сонячними променями, які проникають крізь світлові отвори в зовнішніх огороджувальних конструкціях. Денне (природне) світло — теплове випромінювання Сонця, що пройшло крізь атмосферу. Це випромінювання з суцільним спектром на довжинах хвиль від 300 до 4500 нм з колірною температурою близько 5000 К.

Приміщення з постійним перебуванням людей обов'язково повинні мати природне освітлення. Без природного освітлення допускається проектування лише тих приміщень, що визначені відповідними державними будівельними нормами та стандартами, а також приміщень, розміщення яких дозволено в підвальних поверхах будівель.

Штучне освітлення — це освітлення будинків, приміщень і споруд, зовнішнє освітлення міст, селищ і сільських населених пунктів, територій підприємств і закладів за допомогою спеціальних електроосвітлювальних установок — світильників, а також установки оздоровчого ультрафіолетового випромінювання тривалої дії, установки світлової реклами, світлові знаки та ілюмінаційні установки. Штучне освітлення поділяється на робоче, аварійне, охоронне, чергове.

У приміщеннях житлових будинків, громадських будівель та споруд, адміністративних і побутових будівель підприємств, як правило, застосовують систему загального освітлення. У приміщеннях виробничого характеру, в яких виконується зорова робота I-IV розрядів (ювелірних і гравірувальних робіт, ремонту годинників, телевізорів, радіоапаратури, комп'ютерів, мобільних телефонів, пральних машин, взуття, металовиробів тощо), необхідно застосовувати систему комбінованого освітлення.

Суміщене освітлення — це освітлення, при якому недостатнє за нормами природне освітлення доповнюється штучним. Цей вид освітлення

передбачено для виробничих приміщень, в яких виконуються роботи I-III розрядів; для виробничих та інших приміщень у випадках, коли за умов технології, організації виробництва або клімату в місці будівництва необхідні об'ємно-планувальні рішення, які не дозволяють забезпечити нормоване значення КПО (багатоповерхові будівлі великої ширини тощо), а також у випадках, коли техніко-економічна доцільність суміщеного освітлення порівняно з природним підтверджена відповідними розрахунками; відповідно до нормативних документів з будівельного проектування будівель і споруд окремих галузей промисловості, затверджених в установленому порядку [23].

Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин ДСанПіН 3.3.2.007-98 встановлюють, що штучне освітлення в приміщеннях з робочими місцями, обладнаними ВДТ, повинне здійснюватись системою загального рівномірного освітлення. У адміністративно-громадських та виробничих приміщеннях, у разі частої роботи з документами, допускається застосування системи комбінованого освітлення, тобто крім системи загального освітлення додатково встановлюються світильники місцевого освітлення.

Значення освітленості на поверхні робочого столу в зоні розміщення документів має становити 300-500 лк. Якщо такі значення освітленості неможливо створити з допомогою системи загального освітлення, допускається використання місцевого освітлення. При цьому світильники місцевого освітлення слід встановлювати таким чином, щоб не створювати бліків на поверхні екрану, а його освітленість не повинна перевищувати 300 лк.

В якості джерела світла у випадку штучного освітлення повинні застосовуватись люмінесцентні лампи типу ЛБ. При наявності відбитого освітлення у виробничих та адміністративно-громадських приміщеннях допускається застосування металогалогенних ламп потужністю 250 Вт.

Також допускається застосування ламп розжарювання у світильниках місцевого освітлення.

Система загального освітлення має становити суцільні або переривчасті лінії світильників, розташовані збоку від робочих місць (переважно ліворуч), паралельно лінії зору працюючих. Допускається використання світильників таких класів світлорозподілу: прямого світла (П), переважно прямого світла (Н) та переважно відбитого світла (В).

Для загального освітлення слід застосовувати світильники серії ЛПО 3б із дзеркальними гратами, укомплектовані високочастотними пускорегулювальними апаратами (ВЧ ПРА). Допускається застосовувати світильники цієї серії без ВЧ ПРА тільки в модифікації «Кососвітло». Застосування світильників без розсіювачів та екрануючих грат заборонено.

Яскравість світильників загального освітлення в зоні кутів випромінювання від 50 до 90 град. з вертикаллю в повздовжній та поперечній площинах має становити не більше ніж 200 кд/м^2, захисний кут світильників — не менше ніж 40 град. Світильники місцевого освітлення повинні мати просвічуючий відбивач із захисним кутом, не меншим ніж 40 град.

Слід передбачити обмеження прямої блискості від джерел природного та штучного освітлення. При цьому яскравість світлих поверхонь (вікна, джерела штучного освітлення), що розташовані в полі зору, повинна бути не більшою, ніж 200 кд/м^2.

Необхідно обмежувати відбиту блискість на робочих поверхнях відносно джерел природного і штучного освітлення. При цьому яскравість бліків на екрані ВДТ має не перевищувати 40 кд/м^2, а яскравість стелі в разі застосування системи відбитого освітлення 200 кд/м^2.

Показник осліпленості у разі використання джерел загального штучного освітлення у виробничих приміщеннях має не перевищувати 20, а

показник дискомфорту в адміністративно-громадських приміщеннях має бути не більше за 40.

Необхідно обмежувати нерівномірність розподілу яскравості в полі зору працюючих з ВДТ. При цьому співвідношення яскравостей робочих поверхонь має бути не більшим ніж 3:1, а співвідношення яскравостей робочих поверхонь та поверхонь стін, обладнання тощо — 5:1.

Коефіцієнт запасу ($К_{куб.}$) для освітлювальних установок загального освітлення має дорівнювати 1,4. Коефіцієнт пульсації має не перевищувати 5%, що забезпечується застосуванням газорозрядних ламп у світильниках загального та місцевого освітлення з ВЧ ПРА для світильників будь-яких типів. Якщо не має світильників з ВЧ ПРА, то лампи багатолампових світильників або світильники загального освітлення, розташовані поруч, слід вмикати на різні фази трьохфазної мережі.

Для забезпечення нормованих значень освітленості у приміщеннях з ВДТ слід чистити шибки і світильники принаймні двічі на рік і вчасно замінювати лампи, що перегоріли [24].

ВИСНОВКИ

У результаті виконання дипломної роботи було здійснено розробку та тестування роботи середовища графічного візуалізатора MIDI-файлів з використанням світлових ефектів частинок. З метою реалізації поставленої задачі було проаналізовано розвиток інженерії музичного програмного забезпечення, методів розробки згаданих програмних систем оцінка перспектив їх застосування у сфері написання та обробки музики. У першому розділі дипломної роботи досліджено обставини появи концепції візуалізатора MIDI-файлів та причини її великої популярності.

Під час виконання аналізу предметної області було здійснено огляд існуючих програмних систем такого типу, а саме «Synthesia», «Piano From Above», «Perfect Piano» та «Flowkey». Перелічені програми успішно реалізовують графічну візуалізацію MIDI-файлів, проте мають і недоліки, в зв’язку з чим дана концепція досі має сталий потенціал до вдосконалення.

В якості процесу розробки було обрано водоспадну модель, що виявилось найбільш ефективним рішенням. Дана модель, що є найпростішою для розуміння та реалізації, ідеально підійшла для виконання поставлених завдань. Водоспадна модель дозволила здійснювати строгий контроль за процесом створення програмного забезпечення, оскільки її графік розробки містить терміни для завершення кожного етапу. Результати виконання дипломної роботи показали, що дана модель найкраще підходить для проектів такого типу.

Написання програмної системи здійснювалось з використанням мови програмування C# (для середовища Visual Studio 2019). Мова C# дозволила швидко написати якісну програму, що ефективно справляється зі своїми функціями. Дана мова, що відома як така, що надає можливість швидкого написання якісного коду та створення зручного інтерфейсу програми, повністю виправдала свою актуальність.

В якості графічного інтерфейсу було обрано специфікацію OpenGL, що дозволила швидко та ефективно реалізувати світлові ефекти частинок.

UML-діаграми було використано для візуалізації та глибшого розуміння розроблюваної системи. Діаграми послідовності допомогли розкласти проект на прості складові та побудувати найправильніші алгоритми виконання завдань, а діаграми класів допомогли коректно побудувати структуру програмної системи, щоб створити програмне забезпечення оптимальної складності та максимальної якості.

Програмний продукт успішно пройшов етапи розгортання та тестування, крім того залишає за собою можливості до його розширення: проект можливо вдосконалити, додавши до системи використання VST-інструментів та функції секвенсера (створення нових та редагування існуючих MIDI-файлів у нотному редакторі), що може зробити створене програмне забезпечення першим представником нового покоління цифрових аудіо робочих станцій.

Розроблене програмне забезпечення є вдалим проектом, оскільки виконує усі необхідні завдання, працює без помилок та з достатньо малим навантаженням на персональний комп’ютер. Програма має значний потенціал до вдосконалення та перспективи великої економічної вигоди.

СПИСОК ПОСИЛАНЬ

1. Möllenkamp, A. Paradigms Of Music Software Development. In Proceedings of the 9th Conference on Interdisciplinary Musicology (CIM14). Berlin, Germany, 4-6 December 2014.

2. Nevels, D. L. Using Music Software In The Compositional Process: A Case Study Of Electronic Music Composition. Journal Of Music, Technology & Education 5.3 (2012): 257-271.

3. Ekeroot, J., Berg, J. Audio software development: an audio quality perspective. Part of: Audio Engineering Society Convention 124th, Audio Engineering Society Convention Papers, Curran Associates, Inc., 2008, 7438.

4. Федоров, Александр. MIDI в деталях. Часть 1 — Основы. Музыкальное Оборудование, 2003, август. [Електронний ресурс] – Режим доступу: http://www.muzoborudovanie.ru/articles/midi/midi1.php

5. Музыченко, Евгений. FAQ: Описание интерфейса MIDI [Електронний ресурс] – Режим доступу: http://www.midi.ru/doc/35.htm

6. Секвенсор и его основные возможности. Музыкальное Оборудование, 1996, № 3. [Електронний ресурс] – Режим доступу: http://www.muzoborudovanie.ru/articles/midi/midi1.php

7. Нечитайло, Сергей. Image-Line Software FL Studio 5, часть 1. Музыкальное Оборудование, 2005, июнь. [Електронний ресурс] – Режим доступу: http://www.muzoborudovanie.ru/equip/studio/softsynth/fl/flstudio51.php

8. Synthesia – Wikipedia [Електронний ресурс] – Режим доступу: URL: https://en.wikipedia.org/wiki/Synthesia

9. Synthesia, Piano For Everyone – Synthesia Official Website. [Електронний ресурс] – Режим доступу: https://www.synthesiagame.com/

10. Piano From Above 1.1. [Електронний ресурс] – Режим доступу: https://piano-from-above.software.informer.com/

11. Revontulet Soft, Perfect Piano, Walk Band, X Drum 3D. [Електронний ресурс] – Режим доступу: www.revontuletsoft.com

12. How to play piano - Learn to play piano online with Flowkey. [Електронний ресурс] – Режим доступу: https://www.flowkey.com/en

13. WaterFall Model in Software Developement Life Cycle | SDLC. [Електронний ресурс] – Режим доступу: https://www.toolsqa.com/software-testing/waterfall-model/

14. Офіційна документація C#. [Електронний ресурс] – Режим доступу: https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/

15. Офіційна документація OpenGL. [Електронний ресурс] – Режим доступу: https://docs.microsoft.com/ru-ru/windows/win32/opengl/opengl

16. Unified Modeling Language (UML), An Introduction – GeeksforGeeks [Електронний ресурс] – Режим доступу: https://www.geeksforgeeks.org/unified-modeling-language-uml-introduction/

17. Методичні вказівки для виконання розділу дипломної роботи щодо техніко-економічного обґрунтування вибору проектного рішення розробки та оцінки якості програмного забезпечення / Укладачі: Петрик М.Р., Кінах Я.І., Головатий А.І., Рогатинська Л.Р. – Тернопіль: Вид-во ТНТУ ім. І. Пулюя. – 2013. – 34 с.

18. Наказ 14.02.2018 № 207 «Про затвердження Вимог щодо безпеки та захисту здоров'я працівників під час роботи з екранними пристроями» [Електронний ресурс] – Режим доступу: https://zakon.rada.gov.ua/laws/show/z0508-18#n14

19. Санітарні норми виробничого шуму, ультразвуку та інфразвуку [Електронний ресурс] : ДСН 3.3.6.037–99. − [Чинний від 1999–01–12]. − Режим доступу : http: //mozdocs.kiev.ua/view.php?id=1789

20. Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин [Електронний

ресурс] - Режим доступу: URL: https://zakon.rada.gov.ua/rada/show/v0007282-98

21. Санітарні норми мікроклімату виробничих приміщень [Електронний ресурс] – Режим доступу: URL: https://zakon.rada.gov.ua/rada/show/va042282-99

22. Природне і штучне освітлення [Текст]: ДБН В.2.5-28-2006. - Київ : Мінбуд України, 2006. 96 с.

23. Освітлення виробничих приміщень [Електронний ресурс] – Режим доступу: https://buklib.net/books/35234/

24. Аварійне освітлення [Електронний ресурс] – Режим доступу: http://eet.ua/ua/publication/ua-avariyne-osvitlennya-vse-shho-varto-z/

25. Методичні вказівки до виконання магістерської роботи освітнього рівня «магістр» студентами усіх форм навчання для напряму підготовки 121 – «Інженерія програмного забезпечення» / Укладачі: Петрик М.Р., Михалик Д.М., Кінах Я.І., Гладьо С.В., Цуприк Г.Б. – Тернопіль: Вид-во ТНТУ імені Івана Пулюя. – 2016. – 26 с.

ДОДАТКИ

Додаток А

Міністерство освіти і науки України

Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп’ютерно-інформаційних систем і програмної інженерії

Кафедра програмної інженерії

ЗАТВЕРДЖУЮ

Завідувач кафедрою

програмної інженерії

Петрик Михайло Романович

“___ “_____ 2019 р.

ТЕХНІЧНЕ ЗАВДАННЯ

на виконання дипломної роботи магістра

на тему: «Розробка середовища графічного візуалізатора MIDI-файлів з використанням мови програмування C# та інструментів системи Image-Line FL Studio»

Керівник роботи:

к. ф.-м. н., доцент

Бойко Ігор Володимирович

“___ “_____ 2019 р.

_____

Виконавець:

студент групи СПм-61

Гладій Максим Богданович

“___ “_____ 2019 р.

_____

м. Тернопіль – 2019

ЗМІСТ

1 ПІДСТАВИ ДО РОЗРОБКИ

Розробка проводиться у відповідності до графіку навчального плану на 2019 рік та згідно наказу на виконання дипломної роботи студента-магістра.

Тема проекту: «Розробка середовища графічного візуалізатора MIDI-файлів з використанням мови програмування C# та інструментів системи Image-Line FL Studio».

2 ПРИЗНАЧЕННЯ РОЗРОБКИ

Дипломна робота присвячена створенню програмного забезпечення графічної візуалізації MIDI-файлів з використанням з використанням мови програмування C# та інструментів системи Image-Line FL Studio.

Предметом дослідження є інженерія музичного програмного забезпечення, розвиток методів створення програмних систем даної галузі та розробка графічного візуалізатора MIDI-файлів.

Метою дипломної роботи є розробка середовища графічного візуалізатора MIDI-файлів з використанням світлових ефектів частинок.

За результатами виконаної роботи необхідно отримати програмне забезпечення, що відмінно виконує поставлене на нього завдання, а саме графічну візуалізацію MIDI-файлів з використанням світлових ефектів частинок.

3 ВИМОГИ ДО ПРОГРАМНОГО ПРОДУКТУ

3.1 Функціональні характеристики

Програмне забезпечення має реалізувати наступні можливості:

– відтворення та графічна візуалізація MIDI-файлів на екранному віртуальному фортепіано;

– можливість зміни масштабу та якості графічної візуалізації за шкалою «half – low – medium – high – double»;

– можливість зміни кольору об’єктів, що відображають звуки MIDI-файлу, та світлових ефектів частинок; зміни розміру, швидкості та кількості частинок, що генеруються;

– можливість задання фонового зображення під час візуалізації MIDI-файлу та регулювання його прозорості.

3.2 Склад та параметри технічних засобів

Функціонування програми забезпечується: ПК x64, з 512 Мб оперативної пам‘яті, встановленою системою Windows XP/7/8/8.1/10, та з не менш, ніж 100 Мб вільного місця на жорсткому диску.

3.3 Інформаційна та програмна сполучність

Програмний продукт повинен коректно функціонувати під керуванням ОС Microsoft Windows XP/7/8/8.1/10. Програмний продукт має назву „MIDI-Visualizer” і повинен бути написаний мовою C#.

4. СТАДІЇ РОЗРОБКИ

В ходів реалізації роботи проект повинен пройти крізь наступні стадії розробки:

- аналіз предметної області;
- проектування архітектури;
- реалізація класів і модулів;
- тестування результатів розробки;
- оформлення супровідної документації;
- здача роботи.

5. ПРОГРАМНА ДОКУМЕНТАЦІЯ

Для програмного продукту повинні бути розроблені наступні документи:

- Пояснювальна записка;
- Технічне завдання;
- Презентаційний матеріал;
- Додатки.

6. ПОРЯДОК КОНТРОЛЮ ТА ПРИЙМАННЯ

Розроблений програмний продукт має виконувати всі вимоги, що складаються з перерахованих у п. 3.1 характеристик.

Приймання проводиться спеціально створеною екзаменаційною комісією в термін до 23 грудня 2019 р.

Додаток Б

УДК 004.415.2

М. Б. Гладій – магістрант
І. В. Бойко – кандидат фіз.-мат. наук, доцент
Тернопільський національний технічний університет ім. І. Пулюя, Україна

РОЗРОБКА СЕРЕДОВИЩА ГРАФІЧНОГО ВІЗУАЛІЗАТОРА MIDI-ФАЙЛІВ

M. B. Hladiy – graduate student
I.V. Boyko – Ph.D, Assoc. Prof.
Ternopil Ivan Pul'uj National Technical University, Ukraine

MIDI-FILES GRAPHIC VISUALIZER ENVIRONMENT DEVELOPMENT

У 70-х роках XX століття широкого поширення набули музичні синтезатори. Вони представляли собою набори генераторів звукових частот, керованих напругою. Натискання клавіші на клавіатурі синтезатора вмикало генератор, частота якого задавалася напругою від регулятора тональності даної клавіші. Реалізувати управління всією гамою потенційних можливостей синтезу звуку в рамках одного аналогового пристрою було неможливо, в зв'язку з чим компанії-виробники синтезаторів змогли успішно домовитися про розробку і підтримку єдиного стандарту на інтерфейс управління синтезаторами, що з'явився у 1982 році[1].

Ще з моменту свого народження протокол MIDI (Musical Instrument Digital Interface - цифровий інтерфейс музичних інструментів) став стандартом для всієї електромузичної промисловості з небаченим до того ступенем сумісності. На відміну від інших форматів MIDI – це не оцифрований звук, а набори команд, що можуть відтворюватися по-різному залежно від пристрою відтворення. Замість звукової інформації MIDI працює з «повідомленнями», такими як висота та динаміка взятої на інструменті ноти, контрольних сигналів таких параметрів як гучність, панорама, сигнали відліку часу для синхронізації темпу тощо[2].

Термін «секвенсер», що раніше означав пристрій запису та відтворення послідовності контрольної інформації для електронного музичного інструменту, поступово змінив значення на функцію записуючого програмного забезпечення, що дозволяє користувачеві зберігати, програвати та редагувати MIDI-інформацію. Програма звукозапису, що оснащена такими функціями, стала називатись цифровою звуковою робочою станцією. Їх використання істотно полегшило процес запису музичних творів і автоматизувало їх концертне відтворення[3].

В 2006 році американський розробник Ніколас П'єґдон створив музичну гру «Synthesia» — тренувальну програму, спрямовану на навчання гри на фортепіано, що дозволяє користувачам використовувати MIDI- або комп'ютерну клавіатуру, щоб зіграти графічно візуалізований на екранному віртуальному фортепіано MIDI-файл (Рис. 1). Чимало користувачів вважають, що створена програма зробила навчання гри на фортепіано набагато простішим для початківців[4].

Незважаючи на велику цінність «Synthesia», програма повністю не розкриває потенціалу графічної візуалізації MIDI-файлів, що дає підстави до перегляду її концепції та створення нового програмного забезпечення, що поєднуватиме в собі графічну візуалізацію MIDI-файлів з використанням світлових ефектів частинок та функції секвенсера: створення нових та редагування існуючих MIDI-файлів у нотному редакторі. В якості успіху нова концепція може бути розширена до використання VST-інструментів, що може зробити розроблене програмне забезпечення першим представником нового покоління цифрових звукових робочих станцій.

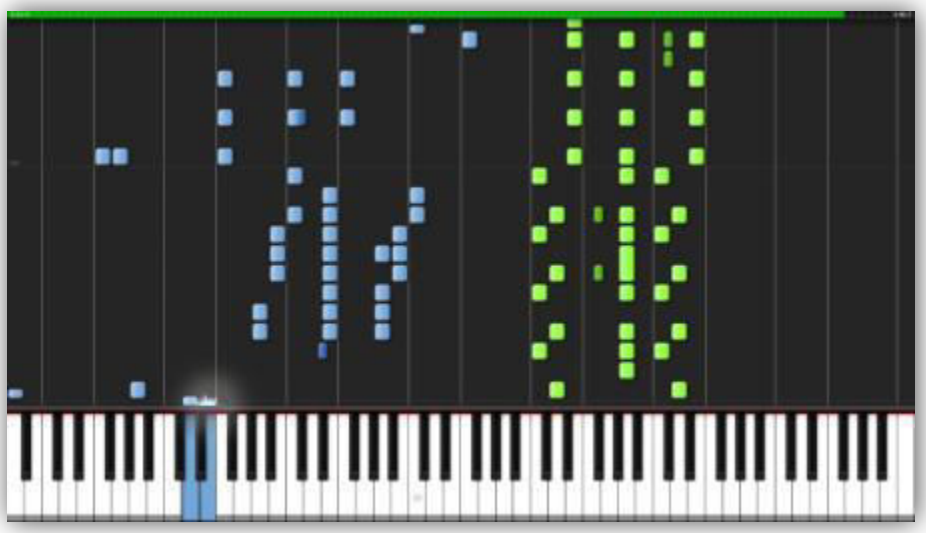

Рис. 1 — Навчальна музична програма «Synthesia»

Література

[1] — Федоров Александр MIDI в деталях. Часть 1 — Основы. *Музыкальное Оборудование,* 2003, август. [Електронний ресурс] – Режим доступу: http://www.muzoborudovanie.ru/articles/midi/midi1.php

[2] — Николенко Д. В. MIDI — язык богов. — СПб.: Регата, 2000. — 144 с.

[3] — Секвенсор и его основные возможности. *Музыкальное Оборудование*, 1996, № 3. [Електронний ресурс] – Режим доступу: http://www.muzoborudovanie.ru/articles/midi/midi1.php

[4] — "Synthesia Features". Synthesia Official Website. [Електронний ресурс] – Режим доступу: https://www.synthesiagame.com/

Програмний код

Лістинг коду класу PianoMaster:

```
using System;
using System.Diagnostics.CodeAnalysis;
using System.Drawing;
using System.Threading;
using System.Windows.Forms;
using MIDIVisualizer.Construct;
using MIDIVisualizer.Graphics;
using SlimDX;
using SlimDX.Direct2D;
using SlimDX.Direct3D11;
using SlimDX.DXGI;
using SlimDX.Windows;
using Device = SlimDX.Direct3D11.Device;
using FactoryD2D = SlimDX.Direct2D.Factory;
using FactoryDXGI = SlimDX.DXGI.Factory;
using Format = SlimDX.DXGI.Format;
using PresentFlags = SlimDX.DXGI.PresentFlags;
using Surface = SlimDX.DXGI.Surface;
using SwapChain = SlimDX.DXGI.SwapChain;
using SwapEffect = SlimDX.DXGI.SwapEffect;
using Usage = SlimDX.DXGI.Usage;

namespace MIDIVisualizer
{
    class PianoMaster
    {
        private RenderTarget renderTarget;
        private RenderForm Form;
        private PianoControl ControlForm;

        public static MIDISequencer Sequencer;

        private long LastTick = Environment.TickCount;
        public static long Elapsed;
        private long LastSample = Environment.TickCount;
        private const long SampleRate = 1000;

        public PianoMaster()
        {
            Sequencer = new MIDISequencer();
        }

        [SuppressMessage("ReSharper", "AccessToDisposedClosure")]
        public void Init()
        {
```

```
            #region init_gfx
            Form = new RenderForm("Kazedan");
            var factory = new FactoryD2D();
            SizeF dpi = factory.DesktopDpi;

            var swapChainDesc = new SwapChainDescription()
            {
                BufferCount = 2,
                Usage = Usage.RenderTargetOutput,
                OutputHandle = Form.Handle,
                IsWindowed = true,
                ModeDescription = new
ModeDescription((int)(GFXResources.Bounds.Width * (dpi.Width / 96f)),
(int)(GFXResources.Bounds.Height * (dpi.Height / 96f)), new
Rational(60, 1), Format.R8G8B8A8_UNorm),
                SampleDescription = new SampleDescription(1, 0),
                Flags = SwapChainFlags.AllowModeSwitch,
                SwapEffect = SwapEffect.Discard,
            };

            Device device;
            SwapChain swapChain;
            Device.CreateWithSwapChain(DriverType.Hardware,
DeviceCreationFlags.BgraSupport, swapChainDesc, out device, out
swapChain);

            Surface backBuffer = Surface.FromSwapChain(swapChain, 0);

            renderTarget = RenderTarget.FromDXGI(factory, backBuffer,
new RenderTargetProperties()
            {
                HorizontalDpi = dpi.Width,
                VerticalDpi = dpi.Height,
                MinimumFeatureLevel =
SlimDX.Direct2D.FeatureLevel.Default,
                PixelFormat = new PixelFormat(Format.R8G8B8A8_UNorm,
AlphaMode.Ignore),
                Type = RenderTargetType.Default,
                Usage = RenderTargetUsage.None
            });
            factory.Dispose();

            // Freaking antialiasing lagging up my programs
            renderTarget.AntialiasMode = AntialiasMode.Aliased;
            renderTarget.TextAntialiasMode =
TextAntialiasMode.Grayscale;

            using (var DXGIFactory =
swapChain.GetParent<FactoryDXGI>())
```

```
                DXGIFactory.SetWindowAssociation(Form.Handle,
WindowAssociationFlags.IgnoreAltEnter);

            Form.ClientSize = GFXResources.Bounds;
            Form.AutoSizeMode = AutoSizeMode.GrowAndShrink;
            Form.Icon = Properties.Resources.KazedanIcon;
            #endregion

            GFXResources.Init(renderTarget);

            Form.KeyDown += (o, e) =>
            {
                Keys key = e.KeyCode;
                switch (key)
                {
                    case Keys.F11:
                        swapChain.IsFullScreen =
!swapChain.IsFullScreen;
                        break;
                    case Keys.F:
                        Sequencer.NoteManager.UserEnabledFancy =
!Sequencer.NoteManager.UserEnabledFancy;
                        Sequencer.NoteManager.RenderFancy =
Sequencer.NoteManager.UserEnabledFancy;
                        break;
                    case Keys.Up:
                        Sequencer.Delay += 100;
                        Sequencer.Reset();
                        break;
                    case Keys.Down:
                        if (Sequencer.Delay >= 100)
                        {
                            Sequencer.Delay -= 100;
                            Sequencer.Reset();
                        }
                        break;
                    case Keys.Left:
                        if (GFXResources.NoteOffset > 0)
                            GFXResources.NoteOffset--;
                        break;
                    case Keys.Right:
                        if (GFXResources.NoteOffset < 128 -
GFXResources.NoteCount)
                            GFXResources.NoteOffset++;
                        break;
                    case Keys.D:
                        Sequencer.ShowDebug = !Sequencer.ShowDebug;
                        break;
                    case Keys.Space:
                        if (Sequencer.Stopped)
```

```
                Sequencer.Start();
            else
                Sequencer.Stop();
            break;
    }
};

Thread loadThread = new Thread(Load);
loadThread.Start();

Thread controlThread = new Thread(() =>
{
    loadThread.Join();
    Application.EnableVisualStyles();
    ControlForm = new PianoControl(Sequencer);
    Application.Run(ControlForm);
});
controlThread.SetApartmentState(ApartmentState.STA);
controlThread.Start();

Form.Disposed += (o, e) =>
{
    ControlForm?.BeginInvoke((MethodInvoker)delegate
    {
        ControlForm.Close();
    });
};

MessagePump.Run(Form, () =>
{
    if (!Sequencer.Stopped)
    {
        Sequencer.UpdateNotePositions();
        Sequencer.UpdateRenderer();
    }
    Paint(renderTarget);

    long tick = Environment.TickCount;
    if (tick - LastSample >= SampleRate)
    {
        Elapsed = tick - LastTick;
        LastSample = tick;
    }
    LastTick = tick;
    swapChain.Present(1, PresentFlags.None);
});

renderTarget.Dispose();
swapChain.Dispose();
device.Dispose();
```

```
            Sequencer.Dispose();
        }

        private void Load()
        {
            Sequencer.Init();
        }

        public void Paint(RenderTarget target)
        {
            target.BeginDraw();
            target.Transform = Matrix3x2.Identity;
            Sequencer.Render(target);
            target.EndDraw();
        }

        public string ToSI(double d, string format = null)
        {
            char[] incPrefixes = new[] { 'k', 'M', 'G', 'T', 'P', 'E',
'Z', 'Y' };
            char[] decPrefixes = new[] { 'm', '\u03bc', 'n', 'p', 'f',
'a', 'z', 'y' };

            int degree = (int)Math.Floor(Math.Log10(Math.Abs(d)) / 3);
            double scaled = d * Math.Pow(1000, -degree);

            char? prefix = null;
            switch (Math.Sign(degree))
            {
                case 1: prefix = incPrefixes[degree - 1]; break;
                case -1: prefix = decPrefixes[-degree - 1]; break;
            }

            return scaled.ToString(format) + prefix;
        }

        public static void Main(string[] args)
        {
            PianoMaster program = new PianoMaster();
            program.Init();
        }
    }
}
```

Лістинг коду класу PianoControl:

```
namespace MIDIVisualizer.Graphics
{
    partial class PianoControl
    {
```

```
        private System.ComponentModel.IContainer components = null
        protected override void Dispose(bool disposing)
        {
            if (disposing && (components != null))
            {
                components.Dispose();
            }
            base.Dispose(disposing);
        }

        #region Windows Form Designer generated code
        private void InitializeComponent()
        {
            this.btnLoad = new System.Windows.Forms.Button();
            this.btnSelect = new System.Windows.Forms.Button();
            this.lblFile = new System.Windows.Forms.Label();
            this.btnHelp = new System.Windows.Forms.Button();
            this.btnJump = new System.Windows.Forms.Button();
            this.jumpTick = new System.Windows.Forms.NumericUpDown();

((System.ComponentModel.ISupportInitialize)(this.jumpTick)).BeginInit(
);
            this.SuspendLayout();
            //
            // btnLoad
            //
            this.btnLoad.Location = new System.Drawing.Point(12, 41);
            this.btnLoad.Name = "btnLoad";
            this.btnLoad.Size = new System.Drawing.Size(186, 42);
            this.btnLoad.TabIndex = 2;
            this.btnLoad.Text = "Load";
            this.btnLoad.UseVisualStyleBackColor = true;
            this.btnLoad.Click += new
System.EventHandler(this.btnLoad_Click);
            //
            // btnSelect
            //
            this.btnSelect.Location = new System.Drawing.Point(12,
12);
            this.btnSelect.Name = "btnSelect";
            this.btnSelect.Size = new System.Drawing.Size(75, 23);
            this.btnSelect.TabIndex = 3;
            this.btnSelect.Text = "...";
            this.btnSelect.UseVisualStyleBackColor = true;
            this.btnSelect.Click += new
System.EventHandler(this.btnSelect_Click);
            //
            // lblFile
            //
            this.lblFile.AutoSize = true;
```

```
            this.lblFile.Location = new System.Drawing.Point(93, 15);
            this.lblFile.Name = "lblFile";
            this.lblFile.Size = new System.Drawing.Size(105, 17);
            this.lblFile.TabIndex = 4;
            this.lblFile.Text = "No file selected";
            //
            // btnHelp
            //
            this.btnHelp.Location = new System.Drawing.Point(204, 60);
            this.btnHelp.Name = "btnHelp";
            this.btnHelp.Size = new System.Drawing.Size(75, 23);
            this.btnHelp.TabIndex = 5;
            this.btnHelp.Text = "Help";
            this.btnHelp.UseVisualStyleBackColor = true;
            this.btnHelp.Click += new
System.EventHandler(this.btnHelp_Click);
            //
            // btnJump
            //
            this.btnJump.Location = new System.Drawing.Point(330, 31);
            this.btnJump.Name = "btnJump";
            this.btnJump.Size = new System.Drawing.Size(75, 23);
            this.btnJump.TabIndex = 6;
            this.btnJump.Text = "Jump";
            this.btnJump.UseVisualStyleBackColor = true;
            this.btnJump.Click += new
System.EventHandler(this.btnJump_Click);
            //
            // jumpTick
            //
            this.jumpTick.Location = new System.Drawing.Point(204,
32);
            this.jumpTick.Maximum = new decimal(new int[] {
            1000000000,
            0,
            0,
            0});
            this.jumpTick.Name = "jumpTick";
            this.jumpTick.Size = new System.Drawing.Size(120, 22);
            this.jumpTick.TabIndex = 7;
            //
            // PianoControl
            //
            this.AutoScaleDimensions = new System.Drawing.SizeF(8F,
16F);
            this.AutoScaleMode =
System.Windows.Forms.AutoScaleMode.Font;
            this.ClientSize = new System.Drawing.Size(414, 93);
            this.Controls.Add(this.jumpTick);
            this.Controls.Add(this.btnJump);
```

```
            this.Controls.Add(this.btnHelp);
            this.Controls.Add(this.lblFile);
            this.Controls.Add(this.btnSelect);
            this.Controls.Add(this.btnLoad);
            this.FormBorderStyle =
System.Windows.Forms.FormBorderStyle.FixedDialog;
            this.MaximizeBox = false;
            this.Name = "KZControl";
            this.Text = "KZControl";
            this.TopMost = true;
            this.Load += new System.EventHandler(this.KZControl_Load);

((System.ComponentModel.ISupportInitialize)(this.jumpTick)).EndInit();
            this.ResumeLayout(false);
            this.PerformLayout();

        }

        #endregion
        private System.Windows.Forms.Button btnLoad;
        private System.Windows.Forms.Button btnSelect;
        private System.Windows.Forms.Label lblFile;
        private System.Windows.Forms.Button btnHelp;
        private System.Windows.Forms.Button btnJump;
        private System.Windows.Forms.NumericUpDown jumpTick;
    }
}
```

Лістинг коду класу Note:

```
namespace MIDIVisualizer.Construct
{
    class Note
    {
        public int Key { get; set; }
        public int Velocity { get; set; }
        public float Position { get; set; }
        public float Length { get; set; }
        public bool Playing { get; set; }
        public long Time { get; set; }
        public int Channel { get; set; }
    }
}
```

Лістинг коду класу NoteManager:

```
using System.Collections.Generic;

namespace MIDIVisualizer.Construct
{
```

```
    class NoteManager
    {
        public readonly Queue<Event> Backlog = new Queue<Event>();
        public readonly List<Note> Notes = new List<Note>();
        public readonly Note[,] LastPlayed = new Note[16, 128];
        public const int ReturnToFancyDelay = 3000;
        public const int ForcedFastThreshold = 3000;
        public bool UserEnabledFancy = true;
        public bool RenderFancy = true;

        public void Reset()
        {
            lock (Notes)
            {
                Notes.Clear();
            }
            Backlog.Clear();
        }
    }
}
```

Лістинг коду класу MIDISequencer:

```
using System;
using System.ComponentModel;
using System.Diagnostics;
using System.Drawing;
using System.Linq;
using System.Windows.Forms;
using MIDIVisualizer.Graphics;
using MIDIVisualizer.Graphics.Renderer;
using Sanford.Multimedia.Midi;
using SlimDX;
using SlimDX.Direct2D;
using SlimDX.DirectWrite;
using Timer = System.Timers.Timer;

namespace MIDIVisualizer.Construct
{
    class MIDISequencer : IDisposable
    {
        public int Delay { get; set; } = 500;
        public bool ShowDebug { get; set; } = true;
        private int LoadingStatus { get; set; } = -1;
        private long LastFancyTick { get; set; }
        public bool Stopped { get; private set; } = true;
        public bool Initialized { get; private set; }
        private readonly Stopwatch Stopwatch;

        public string MIDIFile = @"null";
```

```
        private OutputDevice outDevice;
        private Sequence sequence;
        private Sequencer sequencer;

        private Timer eventTimer;

        public MIDIKeyboard Keyboard { get; set; }
        public NoteManager NoteManager { get; set; }
        private readonly NoteRenderer[] NoteRenderers = { new
FastNoteRenderer(), new FancyNoteRenderer() };
        private readonly KeyRenderer[] KeyRenderers = { new
FastKeyRenderer(), new FancyKeyRenderer() };

        public MIDISequencer()
        {
            Keyboard = new MIDIKeyboard();
            NoteManager = new NoteManager();
            Stopwatch = Stopwatch.StartNew();
            Stopwatch.Stop();

        }

        public void Init()
        {
            if (Initialized)
                return;
            Initialized = true;

            eventTimer = new Timer(10);
            eventTimer.Elapsed += delegate
            {
                lock (NoteManager.Backlog)
                {
                    while (NoteManager.Backlog.Any() &&
NoteManager.Backlog.First().StartTime <=
Stopwatch.ElapsedMilliseconds)
                    {
                        Event ev = NoteManager.Backlog.Dequeue();
                        ev.Method();
                    }
                }
            };

            LoadingStatus = 0;
            outDevice = new OutputDevice(1);
            sequencer = new Sequencer();
            sequence = new Sequence();
            LoadingStatus = -1;
```

```
            sequencer.ChannelMessagePlayed += delegate (object o,
ChannelMessageEventArgs args)
            {
                ChannelCommand cmd = args.Message.Command;
                int channel = args.Message.MidiChannel;
                int data1 = args.Message.Data1;
                int data2 = args.Message.Data2;
                if (cmd == ChannelCommand.NoteOff || (cmd ==
ChannelCommand.NoteOn && data2 == 0))
                {
                    if (NoteManager.LastPlayed[channel, data1] !=
null)
                    {
                        Note n = NoteManager.LastPlayed[channel,
data1];
                        n.Playing = false;
                    }
                }
                else if (cmd == ChannelCommand.NoteOn)
                {
                    Note n = new Note
                    {
                        Key = data1,
                        Length = 0,
                        Playing = true,
                        Position = 0,
                        Time = Stopwatch.ElapsedMilliseconds,
                        Channel = channel,
                        Velocity = data2
                    };
                    lock (NoteManager.Notes)
                        NoteManager.Notes.Add(n);
                    if (NoteManager.LastPlayed[channel, data1] !=
null)
                        NoteManager.LastPlayed[channel, data1].Playing
= false;
                    NoteManager.LastPlayed[channel, data1] = n;
                }

                lock (NoteManager.Backlog)
                {
                    NoteManager.Backlog.Enqueue(new Event(delegate
                    {
                        outDevice.Send(args.Message);
                        if (cmd == ChannelCommand.NoteOff || (cmd ==
ChannelCommand.NoteOn && data2 == 0))
                        {
                            if (Keyboard.KeyPressed[data1] > 0)
                                Keyboard.KeyPressed[data1]--;
                        }
```

```
                        else if (cmd == ChannelCommand.NoteOn)
                        {
                            Keyboard.KeyPressed[data1]++;
                        }
                        else if (cmd == ChannelCommand.Controller)
                        {
                            if (data1 == 0x07)
                                Keyboard.ChannelVolume[channel]       =
data2;
                        }
                        else if (cmd == ChannelCommand.PitchWheel)
                        {
                            int   pitchValue   =   Get14BitValue(data1,
data2);
                            Keyboard.Pitchwheel[channel] = pitchValue;
                        }
                    }, Stopwatch.ElapsedMilliseconds, Delay));
                }
            };

            sequencer.Chased  +=  delegate  (object  o,  ChasedEventArgs
args)
            {
                foreach (ChannelMessage message in args.Messages)
                    lock (NoteManager.Backlog)
                        NoteManager.Backlog.Enqueue(new  Event(()  =>  {
try     {     outDevice.Send(message);     }     catch     {     }     },
Stopwatch.ElapsedMilliseconds, Delay));
            };
            sequencer.Stopped += delegate (object o, StoppedEventArgs
args)
            {
                foreach (ChannelMessage message in args.Messages)
                    lock (NoteManager.Backlog)
                        NoteManager.Backlog.Enqueue(new  Event(()  =>  {
try     {     outDevice.Send(message);     }     catch     {     }     },
Stopwatch.ElapsedMilliseconds, Delay));
                Stop();
            };
            sequence.LoadCompleted    +=    delegate    (object    o,
AsyncCompletedEventArgs args)
            {
                LoadingStatus = -1;
                if (args.Cancelled)
                {
                    MessageBox.Show("The  operation  was  cancelled.",
"MIDITrailer - Error", MessageBoxButtons.OK,
                        MessageBoxIcon.Error);
                    return;
                }
```

```
                sequencer.Sequence = sequence;
            };
            sequence.LoadProgressChanged += delegate (object sender,
ProgressChangedEventArgs args)
            {
                LoadingStatus = args.ProgressPercentage;
            };

        }

        public void Load(string file)
        {
            MIDIFile = file;
            sequencer.Stop();
            sequencer.Position = 0;
            sequence.LoadAsync(file);
        }

        public void Reset()
        {
            Keyboard.Reset();
            NoteManager.Reset();
            outDevice?.Reset();
        }

        public void Dispose()
        {
            outDevice.Close();
            outDevice.Dispose();
            sequencer.Stop();
            sequencer.Dispose();
            sequence?.Dispose();
        }

        public void Stop()
        {
            if (Stopped)
                return;
            eventTimer.Stop();
            sequencer.Stop();
            Stopwatch.Stop();
            Stopped = true;
        }

        public void Start()
        {
            if (!Stopped)
                return;

            eventTimer.Start();
```

```
            if (sequencer.Position > 0)
                sequencer.Continue();
            else
                sequencer.Start();

            Stopwatch.Start();
            Stopped = false;
        }

        public void Jump(int tick)
        {
            Reset();
            sequencer.Position = tick;
            if (Stopped)
                sequencer.Stop();
        }

        public void Render(RenderTarget target)
        {
            if (NoteManager.RenderFancy)
                target.FillRectangle(GFXResources.BackgroundGradient,
new RectangleF(PointF.Empty, target.Size));
            else
                target.Clear(Color.Black);

            lock (NoteManager.Notes)
            {
                if (NoteManager.RenderFancy)
                    NoteRenderers[1].Render(target,  NoteManager.Notes,
Keyboard);
                else
                    NoteRenderers[0].Render(target,  NoteManager.Notes,
Keyboard);
            }
            lock (Keyboard.KeyPressed)
            {
                if (NoteManager.RenderFancy)
                    KeyRenderers[1].Render(target,
Keyboard.KeyPressed);
                else
                    KeyRenderers[0].Render(target,
Keyboard.KeyPressed);
            }

            if (sequence?.GetLength() > 0)
            {
                float  percentComplete  =  1f  *  sequencer.Position  /
sequence.GetLength();
                target.FillRectangle(GFXResources.DefaultBrushes[5],
```

```
                    new    RectangleF(GFXResources.ProgressBarBounds.X,
GFXResources.ProgressBarBounds.Y,  GFXResources.ProgressBarBounds.Width
* percentComplete, GFXResources.ProgressBarBounds.Height));
                target.DrawRectangle(GFXResources.DefaultBrushes[2],
GFXResources.ProgressBarBounds, .8f);
            }

            string[] debug;
            string   usage   =   Application.ProductName   +   "   "   +
Application.ProductVersion + " (c) " + Application.CompanyName;
            if (ShowDebug)
            {
                debug = new[]
                {
                    usage,
                    "      file: " + MIDIFile,
                    "note_count: " + NoteManager.Notes.Count,
                    "   frames/s: " + (Kazedan.Elapsed == 0 ? "NaN" :
1000 / Kazedan.Elapsed + "") +" fps",
                    "    renderer:   "   +   (NoteManager.RenderFancy   ?
"fancy" : NoteManager.UserEnabledFancy ? "forced-fast" : "fast"),
                    "   seq_tick: " + (sequence == null ? "? / ?" :
$"{sequencer.Position} / {sequence.GetLength()}"),
                    "     delay: " + Delay + "ms",
                    "       kbd: " + GFXResources.NoteCount + " key(s)
+" + GFXResources.NoteOffset + " offset",
                    "   stopped: " + Stopped
                };

            }
            else
            {
                debug = new[] { usage };
            }
            string  debugText  =  debug.Aggregate("",  (current,  ss)  =>
current + ss + '\n');
            target.DrawText(debugText,        GFXResources.DebugFormat,
GFXResources.DebugRectangle,            GFXResources.DefaultBrushes[0],
DrawTextOptions.None,
                MeasuringMethod.Natural);

            if (LoadingStatus == 0)
                target.DrawText("INITIALIZING      MIDI      DEVICES",
GFXResources.HugeFormat,                    GFXResources.FullRectangle,
GFXResources.DefaultBrushes[0],                  DrawTextOptions.None,
MeasuringMethod.Natural);
            else if (LoadingStatus > 0 && LoadingStatus < 100)
                target.DrawText("LOADING  "  +  LoadingStatus  +  "%",
GFXResources.HugeFormat,                    GFXResources.FullRectangle,
```

```
GFXResources.DefaultBrushes[0],                    DrawTextOptions.None,
MeasuringMethod.Natural);
        }

        public void UpdateNotePositions()
        {
            int keyboardY = (int)GFXResources.KeyboardY;
            long now = Stopwatch.ElapsedMilliseconds;
            float speed = 1.0f * keyboardY / Delay;
            lock (NoteManager.Notes)
            {
                for (int i = 0; i < NoteManager.Notes.Count; i++)
                {
                    Note n = NoteManager.Notes[i];
                    if (n.Position > keyboardY)
                    {
                        NoteManager.Notes.RemoveAt(i);
                        i--;
                    }
                    else
                    {
                        if (n.Playing)
                            n.Length = (now - n.Time) * speed;
                        else
                            n.Position  =  (now  -  n.Time)  *  speed  -
n.Length;
                    }

                }
            }
        }

        public void UpdateRenderer()
        {
            if                (NoteManager.Notes.Count                >
NoteManager.ForcedFastThreshold)
            {
                LastFancyTick = Stopwatch.ElapsedMilliseconds;
                NoteManager.RenderFancy = false;
            }
            else
            {
                if (NoteManager.UserEnabledFancy)
                    if  (Stopwatch.ElapsedMilliseconds  -  LastFancyTick
> NoteManager.ReturnToFancyDelay)
                        NoteManager.RenderFancy = true;
            }
        }
```

```
        public static int Get14BitValue(int nLowerPart, int
nHigherPart)
        {
            return (nLowerPart & 0x7F) | ((nHigherPart & 0x7F) << 7);
        }
    }
}
```

Лістинг коду класу MIDIKeyboard:

```
using System;
using System.Drawing;
using MIDIVisualizer.Graphics;
using SlimDX.Direct2D;

namespace MIDIVisualizer.Construct
{
    class MIDIKeyboard
    {
        public int[] KeyPressed { get; }
        public int[] ChannelVolume { get; }
        public int[] Pitchwheel { get; }

        public MIDIKeyboard()
        {
            KeyPressed = new int[128];
            ChannelVolume = new int[16];
            Pitchwheel = new int[16];

            for (int i = 0; i < 16; i++)
            {
                ChannelVolume[i] = 127;
                Pitchwheel[i] = 0x2000;
            }
        }

        public void Reset()
        {
            Array.Clear(KeyPressed, 0, KeyPressed.Length);
        }
    }
}
```

Додаток Г

Диск