Форма № 24

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Національний університет «Запорізька політехніка»

Факультет комп’ютерних наук і технологій
(повне найменування факультету)

Кафедра програмних засобів
(повне найменування кафедри)

Пояснювальна записка

до дипломного проєкту (роботи)

магістр
(ступінь вищої освіти

на тему РОЗРОБКА ТА ДОСЛІДЖЕННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ ГЕНЕРАЦІЇ МУЗИКИ З ВИКОРИСТАННЯМ ІНТЕРАКТИВНОГО ГЕНЕТИЧНОГО АЛГОРИТМУ

DEVELOPMENT AND RESEARCH OF SOFTWARE FOR GENERATING MUSIC USING AN INTERACTIVE GENETIC ALGORITHM

Виконав(ла): студент(ка) 2 курсу, групи КНТ-214м
Спеціальності 122 Комп'ютерні науки
(код і найменування спеціальності)

Освітня програма (спеціалізація)
Системи штучного інтелекту

ЧОРНОБУК М.О.
(ПРІЗВИЩЕ та ініціали)
Керівник ОЛІЙНИК А.О.
(ПРІЗВИЩЕ та ініціали)
Рецензент СКРУПСЬКИЙ С.Ю.
(ПРІЗВИЩЕ та ініціали)

2025

Форма № 25

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»
(повне найменування закладу вищої освіти)

Факультет КНТ
Кафедра програмних засобів
Ступінь вищої освіти магістр
Спеціальність 122 Комп'ютерні науки
(код і найменування)
Освітня програма (спеціалізація) Системи штучного інтелекту
(назва освітньої програми (спеціалізації)

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.
________________Сергій СУББОТІН
“____” ________________2025 року

З А В Д А Н Н Я

НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

ЧОРНОБУКА Максима Олеговича
(ПРІЗВИЩЕ, ім’я, по батькові)

1. Тема проєкту (роботи) Розробка та дослідження програмного забезпечення для генерації музики з використанням інтерактивного генетичного алгоритму. Development and Research of Software for Generating Music Using an Interactive Genetic Algorithm.

керівник проєкту (роботи) д.т.н., професор, ОЛІЙНИК Андрій Олександрович ,
(науковий ступінь, вчене звання, ПРІЗВИЩЕ, ім’я, по батькові)

затверджені наказом закладу вищої освіти від “ 30 ” вересня 2025 року № 447

2. Строк подання студентом проєкту (роботи) 08 грудня 2025 року

3. Вихідні дані до проєкту (роботи) рекомендована література, технічне завдання

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз проблеми та постановка завдань дослідження. 2. Матеріали і методи. 3. Основні рішення щодо реалізації компонентів системи. 4. Експлуатація, тестування та експериментальне дослідження програми.

5. Перелік графічного матеріалу (з з точним зазначенням обов’язкових креслень, кількість слайдів, плакатів)
Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	ПРІЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-4 Основна частина	СКРУПСЬКИЙ С.Ю., доцент		
Нормоконтроль	КАЛІНІНА М.В., асистент		

7. Дата видачі завдання « 30 » вересня 2025 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи.	1 тиждень	Завдання, ТЗ
2	Аналіз предметної області.	2-3 тижні	Розділ 1
3	Вибір мови програмування та інших технологій	4-5 тижні	Розділ 2
	розробки.		
4	Розробка архітектури програми.	6 тиждень	Розділ 3
5	Розробка програми.	7-8 тижні	Розділ 3, 4
6	Тестування та експериментальне дослідження	9 тиждень	Розділ 4
	програмного забезпечення.		
7	Оформлення пояснювальної записки та документів	10-11 тиждень	Додатки
	до неї.		
8	Нормоконтроль та рецензування.	12 тиждень	
9	Захист роботи.	12 тиждень	

Студент(ка) __________ Максим ЧОРНОБУК
(підпис) (Ім’я ПРІЗВИЩЕ)

Керівник проєкту (роботи) __________ Андрій ОЛІЙНИК
(підпис) (Ім’я ПРІЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи магістра: 100 с., 8 табл., 24 рис., 4 дод., 24 джерела.

ГЕНЕРАТИВНА МУЗИКА, ГЕНЕТИЧНИЙ АЛГОРИТМ, MAUI, MIDI, C#.

Об'єкт дослідження – процес генерації музичних композицій на основі відгуків користувача за допомогою інтерактивного генетичного алгоритму.

Предмет дослідження – методи генерації музичних композицій на основі відгуків користувача.

Мета роботи – дослідження та програмна реалізація методу інтерактивної генерації музики за допомогою інтерактивного генетичного алгоритму.

Актуальність та проблематика роботи полягає в тому, що інтерактивна генерація музики широко використовується у різних галузях індустрії розваг. Методи генерації на основі генеративних нейронних мереж, що набрали популярність в останні роки, вимагають значних обсягів обчислювальних ресурсів, зокрема оперативної пам'яті та процесорного часу, зазвичай надаються на платній основі, а також не можуть бути використані в реальному часі. Методи на основі генетичного алгоритму потребують значно менших обчислювальних ресурсів та знаходять використання у випадках, коли генерація повинна відбуватися в реальному часі.

Матеріали, методи та технічні засоби: мова програмування C# середовище програмування Visual Studio Community, персональний комп'ютер з процесором Intel Core Intel I7-12650H під управлінням операційної системи Microsoft Windows 11.

Результати. Створено застосунок, який здійснює генерацію музичних композицій на основі відгуків користувача у популярному форматі MIDI.

Висновки. Проведений аналіз існуючих методів генерації музики на основі інтерактивного генетичного алгоритму. Розроблено модель на основі модифікованого інтерактивного генетичного алгоритму. На основі розробленої моделі створено застосунок, який здійснює генерацію музичних композицій на основі відгуків користувача у популярному форматі MIDI.

Галузь використання – застосування у цифрових продуктах, що потребують генерації музики в реальному часі.

ABSTRACT

Explanatory note to the diploma qualifying work of the master: 100 pages, 8 tables, 24 figures, 4 appendixes, 24 sources.

GENERATIVE MUSIC, GENETIC ALGORITHM, MAUI, MIDI, C#.

The object of research is the process of generating musical compositions based on user feedback using an interactive genetic algorithm.

The subject of the study is methods of generating musical compositions based on user feedback.

The goal of this work is to study and programmatically implement a method for interactive music generation using an interactive genetic algorithm.

The relevance and subject matter of this work lies in the fact that interactive music generation is widely used in various areas of the entertainment industry. Generative neural network-based generation methods, which have gained popularity in recent years, require significant computing resources, including RAM and processor time, are usually provided on a paid basis, and cannot be used in real time. Methods based on genetic algorithms require significantly fewer computing resources and are used in cases where generation must occur in real time.

Materials, methods, and technical means: C# programming language, Visual Studio Community programming environment, personal computer with Intel Core Intel I7-12650H processor running Microsoft Windows 11 operating system.

Results. An application has been created that generates musical compositions based on user feedback in the popular MIDI format.

Conclusions. An analysis of existing methods of music generation based on an interactive genetic algorithm was conducted. A model based on a modified interactive genetic algorithm was developed. Based on the developed model, an application was created that generates musical compositions based on user feedback in the popular MIDI format.

Field of application: use in digital products that require real-time music generation.

ЗМІСТ

С.

Перелік скорочень та умовних познак ... 10
Вступ ... 11
1 Аналіз проблеми та постановка завдань дослідження ... 13
1.1 Загальний огляд ... 13
1.2 GenJam ... 13
1.3 GP-Music ... 14
1.4 DarwinTunes ... 15
1.5 Порівняльний аналіз існуючих систем ... 15
1.6 Висновок ... 16
2 Матеріали і методи ... 17
2.1 Вибір мови програмування ... 17
2.2 Вибір бібліотеки графічного інтерфейсу ... 19
2.3 Вибір integrated development environment ... 21
2.4 Висновок ... 23
3 Основні рішення щодо реалізації компонентів системи ... 24
3.1 Постановка проблеми ... 24
3.2 Розробка інтерактивного генетичного алгоритму ... 26
3.3 Розроблене програмне забезпечення ... 32
3.4 Висновок ... 34
4 Експлуатація, тестування та експериментальне дослідження програми ... 35
4.1 Призначення й умови використання ... 35
4.2 Використання системи ... 35
4.3 Повідомлення ... 37
4.4 Тестування системи ... 37
4.5 Порівняння системи з існуючими аналогами ... 39
4.6 Висновок ... 41
Висновки ... 42
Перелік джерел посилань ... 44

Додаток А Технічне завдання 47
Додаток Б Опис програмного продукту 53
Додаток В Текст програми 56
Додаток Д Слайди презентації 92

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

IDE – Integrated Development Environment;

IGA – Interactive Generic Algorithm;

ПП – програмний продукт.

ВСТУП

Динамічна генерація музики є відомою технологією та використовується у різних цифрових продуктах ще з кінця XX сторіччя. Типовими прикладами такого використання є комп’ютерні ігри Spore, No Man’s Sky, Mini Metro. Використання згенерованої музики дозволяє зекономити кошти, що могли б піти на оплату праці людини-композитора. Така музика також покращує досвід користувача за рахунок ефекту новизни досвіду використання продукту [1].

В останні роки особливо велику популярність здобули технології генерації музики на основі штучних нейронних мереж. Десятки моделей дозволяють генерувати високоякісну інструментальну музику та вокальну музику на основі опису користувача. Наприклад, широко використовуються MuseNet від OpenAI [2], MusicLM від Google [3]. Такі треки майже неможливо відрізнити від професійних записів, зроблених у студії. Однак, такі технології все ще потребують значних обсягів обчислювальних ресурсів, зокрема оперативної пам’яті та процесорного часу. Зазвичай вони надаються на платній основі або потребують великих обчислювальних ресурсів та не можуть працювати в реальному часі на комп’ютерах кінцевих користувачів. Використання таких технологій наразі обмежується завчасною генерацією музики, а генерація в реальному часі потребує інших рішень [4].

Одним із рішень, що не потребують значних обсягів обчислень, є генетичні алгоритми, зокрема інтерактивні генетичні алгоритми (IGA). Такі методи є модифікацією стандартного генетичного алгоритма у випадках, коли fitness-function неможливо розрахувати об’єктивно. Тоді ввод від користувачів використовується замість fitness-function, керуючи еволюційним процесом. Ці методи широко використовуються у сфері генерації музики, де суб’єктивна привабливість одного і того ж треку залежить від естетичних уподобань конкретного користувача або групи користувачів [5].

Надалі розглядається створення системи на основі вдосконалення адаптивного інтерактивного генетичного алгоритму, призначеного для створення музики у форматі MIDI. Розроблена модель відрізняється від ряду інших моделей простотою та високою швидкодією. Зокрема, розроблена система відстежує відгуки користувачів у різних поколіннях і оцінює залученість користувачів на основі статистичних показників, що розраховуються на основі потоку оцінок користувачів. Розрахований показник динамічно регулює ймовірність мутацій та ін’єкцій у популяції з метою зниження втоми користувачів, прискорення збіжності та підтримання зацікавленості протягом усього процесу використання системи.

1 АНАЛІЗ ПРОБЛЕМИ ТА ПОСТАНОВКА ЗАВДАНЬ ДОСЛІДЖЕННЯ

1.1 Загальний огляд

Кілька існуючих систем демонструють потенціал IGA для генерації музики. Наприклад, GenJam [6] використовує оцінку користувачів для еволюції джазових соло; DarwinTunes [7] використовує подібні еволюційні підходи для генерації музики. Ці системи зазвичай кодують музичні властивості (наприклад, висоту звуку, ритм, гармонію) як геноми, застосовують селекцію та мутацію і покладаються на відгуки користувачів для еволюції все більш привабливих композицій. Однак такі моделі часто страждають від обмежень, таких як висока втома користувачів, повільна конвергенція або фіксовані стратегії мутації, які не адаптуються до залучення слухачів.

1.2 GenJam

У статті [6] описана GenJam, одна за ранніх систем для генерації музики за допомогою інтерактивного генетичного алгоритма. GenJam створена для генерації джазових соло. Система може працювати в одному з трьох режимів:

– режим навчання. Генеруються випадкові композиції, користувач оцінює їх. Еволюційний процес не запускається;

– режим демо. Програються найкращі зі згенерований раніше композицій;

– еволюційний режим. Використовуються генетичні операції для динамічної генерації нової популяції музичних композицій.

GenJam використовує дворівневу схему генетичного кодування для представлення композицій. Музичний контент ієрархічно структурований у фрази та такти, кожна з яких кодується як бінарні хромосоми фіксованої довжини. Таке представлення об'єднує ритм і висоту звуку. Кожен такт

відповідає 32-бітній хромосомі, що представляє вісім послідовних позицій восьмих нот у такті 4/4. Кожна позиція кодується 4-бітним подією: Rest, Hold, New Note.

Система є досить обмеженою: композиції не відрізняються за швидкістю (значенням BPM), використовують завчасно задані послідовності акордів.

1.3 GP-Music

Більш сучасною є система GP-Music, що описана в статті [8]. Ця система має кілька відмінних рис. Листя дерев представляють окремі музичні ноти, псевдоакорди або паузи. Внутрішні вузли представляють музичні перетворення або операції, що застосовуються до послідовностей. Таким чином кожний вузол дерева повертає послідовність нот, а фінальну композицію повертає корінь дерева. Така структура має свої переваги: можна зручно проводити операції мутації та кросоверу. Іншою важливою особливістю є використання простої нейронної мережі, що навчається під час фази відгуків від користувача, а після здатна давати власні оцінки композиціям, зменшуючі навантаження на користувача. Такий метод називається сурогатною фітнес-функцією (Surrogate Fitness Function) та має великий потенціал, хоча в реальних умовах і стикається з проблемою відсутності достатньої кількості ресурсів для складної моделі, що здатна якісно імітувати оцінювання з боку користувача. Тим не менш, система є досить обмеженою і дозволяє створювати лише компактні монофонічні музичні композиції. Великим недоліком є той факт, що всі ноти у згенерованих композиціях мають однакову довжину: це значно обмежує систему, виключаючи створення складних композицій, подібних до реальної музики.

1.4 DarwinTunes

Інший підхід описано в статті [7]. Автори розробили систему DarwinTunes на основі інтерактивного генетичного алгоритма, що не використовувала сурогатну фітнес-функцією, проте об’єднувала відгуки від більш ніж 6000 користувачів, використовуючи їх агреговані оцінки у якості фітнес-функції для невеликих музичних композицій. Запропонована система використовувала популяцію деревоподібних цифрових геномів, кожен з яких описував програму, що генерувала невелику зациклену музичну композицію довжиною у 8 секунд.

1.5 Порівняльний аналіз існуючих систем

Загальне порівняння розглянутих систем наведено в табл. 1.1. Усі розглянуті системи відрізняються спільним недоліком: кожна із систем здатна генерувати лише композицію одного жанру або типу. Системи також генераують лише монофонічну музику, а також не здатні динамічно відслідковувати рівень зацікавленості користувача.

Таблиця 1.1 – Порівняння розглянутих алгоритмів

Система	Тип музики	Сурогатна фітнес-функція	Розрахунок зацікавленості користувача	Монофонічна музика
GenJam [6]	Джазові соло	Ні	Ні	Так
GP Music [8]	Компактні композиції та однакова довжина нот	Так	Ні	Так
DarwinTunes [7]	Компактні зациклені композиції	Ні	Ні	Так

1.6 Висновок

У ході виконання роботи було проведено існуючих систем для генерації музичних композицій з використанням інтерактивного генетичного алгоритму. Було зроблено висновок, про спільні недоліки розглянутих систем, які необхідно вирішити у запропонованій системі.

2 МАТЕРІАЛИ І МЕТОДИ

2.1 Вибір мови програмування

Для подальшої розробки системи необхідно вибрати мову програмування, яка б відповідала всім необхідним вимогам для вирішення поставленої задачі. Розповсюдженими мовами для розробки desktop застосунків використовують C#, Java, Python.

2.1.1 Мова C#

Мова C# – це розроблена Microsoft високорівнева об’єктно-орієнтована мова, що підтримує різні парадигми. C# знаходить широке застосування у веброзробці, розробці комп’ютерних ігор, а також у розробці настільних застосунків для Windows.

C# відрізняється великою вбудованою бібліотекою, розвиненою екосистемою, динамічним ком’юніті, а також відносно високою швидкодією [9]. Microsoft підтримує ряд фреймворків, що можуть бути використані для створення графічного користувальницького інтерфейсу: Windows Forms, WPF, MAUI.

Зазвичай для проєктів на C# використовують пакетний менеджер NuGET, що дозволяє швидко та зручно керувати залежностями проєкту [10].

2.1.2 Мова Java

Java – відома об’єктно орієнтована мова програмування, розроблена Oracle. Однією із найголовніших відмінностей Java є широка підтримка кросс-платформності. Програми на Java можуть бути виконані на більшості настільних та мобільних пристроїв. Для мови створено ряд фреймворків для розробки користувальницького інтерфейсу: Swing, JavaFX, SWT. Java демонструє швидкодію на рівні C# [9].

Для Java відсутній єдиний стандартний пакетний менеджер. Двома найбільш популярними є Maven та Gradle [11].

2.1.3 Мова Python

Python – інтерпретована мова програмування загального призначення. Вона є найвикористовуванішою мовою програмування у світі та наразі широко використовується у таких сферах: веброзробка, розробка прикладних програм, машинне навчання. Python відноситься до вільного програмного забезпечення та розповсюджується під ліцензією Python Software Foundation License. Для Python також існує ряд бібліотек для створення користувальницького інтерфейсу: PyQt5, Tkinter, Kivy. Хоча ці бібліотеки поступаються аналогами на Java та C# за рівнем зручності використання.

Важливою перевагою Python також є наявність зручної централізованої системи розповсюдження користувальницьких бібліотек pip, яка значно спрощує процес розробки [12].

Недоліком Python є швидкодія програм, створених за допомогою цієї мови. Вони поступаються аналогам на Java та C# [9]. Тим не менш, деякі дослідження вказують на те, що Python відрізняється лаконічністю коду та високою швидкістю розробки програм [13].

2.1.4 Порівняння мов програмування

Порівняння розглянутих вище мов програмування наведено в табл. 2.1.

Таблиця 2.1 – Порівняння мов програмування

Мова програмування	C#	Java	Python
Єдина система керування пакетами	Так	Ні	Так
Широкий набір бібліотек	Так	Так	Так
Зручні фреймворки для створення UI	Так	Так	Ні
Велика екосистема	Так	Так	Так
Кросс-платформність	Так	Так	Так
Відносна швидкодія [9]	Висока	Висока	Низька
Відносна лаконічність коду [13]	Низька	Низька	Висока

Зважаючи на дані, наведені в табл. 2.1, можна зробити висновок про те, що саме C# є найкращою мовою для вирішення поставленої задачі. Наявність великої екосистеми з усіма необхідними інструментами, а також фреймворків для зручної розробки графічного інтерфейсу грають значну роль під час реалізації системи.

2.2 Вибір бібліотеки графічного інтерфейсу

Жодний програмний продукт, що призначений для використання звичайними користувачами, не може бути конкурентноздатним без зручного графічного інтерфейсу. Для мови C# існує ряд бібліотек, що дозволяють реалізувати графічний інтерфейс, хоча кожна з них має свої відмінності. Нижче розглянуті основні доступні варіанти.

2.2.1 Windows Forms

Windows Forms – найстаріша доступна бібліотека, що була розроблена в 2002 році. Вона дозволяє створювати прості користувальницькі інтерфейси

для платформи Windows. Можливості для створення інтерфейсів із складною системою стилів обмежені [14].

2.2.2 WPF

WPF є більш новою бібліотекою для C#, представленою в 2006 році. WPF також обмежена платформою Windows, проте використовує мову розмітки XML для створення значно більш складних інтерфейсів. Ця бібліотека підтримує розширену систему стилів, зв’язування даних, створення шаблонів [15].

2.2.3 MAUI

MAUI – сучасна кросс-платформна бібліотека для створення графічних інтерфейсів для мови C#. Бібліотека використовує діалект мови розмітки XAML для створення інтерфейсів. Підтримуються мобільні платформи, Windows, MacOS. Наявна просунута система стилів, що робить можливим створення сучасних кросс-платформних інтерфейсів [16].

2.2.4 MAUI

Порівняння розглянутих бібліотек графічного інтерфейсу наведено в табл. 2.2.

Таблиця 2.2 – Порівняння бібліотек графічного інтерфейсу

Бібліотека	Windows Forms	WPF	MAUI
Складна система стилів	Ні	Так	Так
Кросплатформеність	Ні	Ні	Так
Рекомендована для сучасних проєктів	Ні	Так	Ні

Очевидним вибором для реалізації графічного інтерфейсу є MAUI. Ця сучасна бібліотека дозволить швидко створити зручний та лаконічний інтерфейс. Створений застосунок потенційно можна буде використовувати не лише на Windows, а й на MacOS.

2.3 Вибір integrated development environment

Розробка сучасного програмного забезпечення не є можливою без використання integrated development environment (IDE). Згідно статистики, близько 75% розробників використовують IDE [17]. IDE спрощує та пришвидшує розробку, дозволяючи швидко виконувати основні операції: дебаг, компіляцію, тощо. Зазвичай IDE також надають додаткові функції: підсвічування синтаксису та синтаксичних помилок, інтеграція з системами контролю версій, вбудовані редактори графічного інтерфейсу користувача, а також розширення функціоналу за допомогою плагінів.

2.3.1 Visual Studio Community

Visual Studio Community є безкоштовною IDE від Microsoft, що у більшості випадків використовується для розробки проєктів на C#. Більшість необхідних інструментів, як то підсвічування синтаксису, відображення синтаксичних помилок, інтеграція з системами контролю версій, інтеграція з MAUI наявні в Visual Studio Community [18].

2.3.2 JetBrains Rider

Rider – крос-платформна IDE від JetBrains для сучасної розробки на C#. Відрізняється високою швидкодією, розширеним інструментарієм для рефакторингу. На відміну від Visual Studio Community, Rider не є

безкоштовною для комерційної розробки. Інтеграція з MAUI також є обмеженою [19].

2.3.3 Visual Studio Code

Visual Studio Code є безкоштовною легковажною IDE від Microsoft. Більшість необхідного функціоналу, як то автодоповнення коду та дебаг на C#, забезпечується за допомогою системи плагінів, що активно підтримується суспільством. Повноцінна інтеграція MAUI відсутня [20].

2.3.4 Порівняння integrated development environment

Порівняння розглянутих integrated development environment наведено в табл. 2.3.

Таблиця 2.3 – Порівняння integrated development environment

IDE	Visual Studio Community	JetBrains Rider	Visual Studio Code
Безкоштовна	Так	Ні	Так
Розширена підтримка MAUI	Так	Ні	Ні
Автодоповнення коду	Так	Так	Так
Інтеграція із Git	Так	Так	Так
Система плагінів	Так	Так	Так

Найбільш ефективним вибором для розробки на MAUI є безкоштовна IDE Visual Studio Community від Microsoft. Глибока інтеграція з MAUI, а також усі необхідні у сучасній розробці інструменти зумовлюють цей вибір.

2.4 Висновок

У ході виконання роботи було обрано мову програмування C#. У якості бібліотеки для розробки користувальницького інтерфейсу обрано MAUI. Середовищем розробки було обрано Visual Studio Community.

3 ОСНОВНІ РІШЕННЯ ЩОДО РЕАЛІЗАЦІЇ КОМПОНЕНТІВ СИСТЕМИ

Завданням роботи є програмна реалізація системи генерації музичних композицій на основі відгуків користувача за допомогою інтерактивного генетичного алгоритму.

У роботі розглядаються проблеми кодування геному композицій, відслідковування рівня зацікавленості користувача та адекватної реакції на його падіння.

Для досягнення поставленої мети необхідно розробити модифікацію інтерактивного генетичного алгоритму, пристосовану для швидкої генерації даних про музичні композиції.

3.1 Постановка проблеми

Музичний цифровий інтерфейс (MIDI) – це стандарт, розроблений у 1983 році в результаті співпраці між провідними виробниками електронних інструментів. Стандарт забезпечує взаємодію між обладнанням різних виробників та надає цифровий абстрактний рівень над аналоговими та цифровими процесами генерації звуку [21]. Наразі стандарт має широке використання, а об’єм ринку апаратного забезпечення, сумісного з MIDI становить близько 850 мільйонів доларів та продовжує зростати [22].

Формат Standard MIDI File (SMF) дозволяє компактно зберігати дані про музичні композиції. На відміну від класичних аудіо файлів, що зберігають дані про безпосередньо звук, MIDI-файли зберігають послідовності дискретних музичних подій, які потім перетворюють на музику за допомогою програмних чи апаратних рішень. Кожен MIDI-файл складається з заголовка та одного або декількох треків. Заголовок визначає глобальні параметри, такі як тип файлу, кількість треків та темп. Кожен трек складається з послідовності упорядкованих за часом подій, кожна з яких має

відмітку часу відносно попередньої події. Це дозволяє точно розміщувати ноти та інші музичні події упродовж треку.

Найбільш розповсюдженими є типи подій Note On та Note Off, які разом визначають, коли нота починається та закінчується. Кожна подія програвання ноти характеризується MIDI-каналом (0–15), висотою ноти (0–127), інтенсивністю ноти (0–127). Ці параметри описують як горизонтальну (засновану на часі), так і вертикальну (засновану на висоті звуку) структуру музичної композиції [23].

Простота та розповсюдженість формату MIDI обґрунтовують його вибір у якості основи для розроблюваної системи. Зважаючи на необхідність створення достатньо простих монофонічних композицій, а також відсутності необхідності зміни інтенсивності нот, можна формально описати музичну композицію s за формулами (3.1, 3.2, 3.3):

$$s = (BPM, \{e_1, e_2, \dots, e_N\}), \tag{3.1}$$

$$e_i = (p_i, d_i) \text{ або } e_i = (\emptyset, d_i), \tag{3.2}$$

$$p_i \in P, d_i \in D, \tag{3.3}$$

де P – множина дозволених значень висоти ноти;

D – множина дозволених значень довжини ноти.

Тоді формально задача розробки системи генерації музики зводиться до знаходження такої функції M за формулами (3.4, 3.5):

$$S_{n+1} = M(S_n, Y_n)\ , \tag{3.4}$$

$$Y_n = \{y_1, y_2, \dots, y_x\}, \tag{3.5}$$

де y_i – оцінка користувача для i-ї композиції n-го покоління, $y \in [0,1]$;

x – розмір покоління.

Покоління містить в собі множину композицій та описується формулою (3.6):

$$S_n = \{s_1, s_2, \dots, s_x\}, \tag{3.6}$$

де S_n – n-не покоління, s ∈ T;

T – множина всіх можливих музичних композицій з урахуванням обмежень системи.

Найскладнішим викликом під час розробки системи є знаходження такої функції М, що здатна швидко генерувати композиції, які отримають високу оцінку від користувача.

3.2 Розробка інтерактивного генетичного алгоритму

Робота генетичного алгоритму базується на реалізації декількох базових операторів, що імітують реальні еволюційні процеси для покращення значення фітнес-функції осіб в популяції: кросинговер, мутація та ін'єкція.

Кросинговер – це процес генерація нової особи на основі генетичного матеріалу двох батьківських осіб. Його суть полягає у поєднанні успішних характеристик обох батьків для утворення потенційно кращих рішень. Під час кросинговеру обираються дві особини, визначається точка або кілька точок обміну, після чого частини їхніх геномів змінюються місцями. У результаті виникають нові комбінації генів, що дозволяють досліджувати нові області простору рішень.

Мутація – це випадкова зміна окремих генів у хромосомі, яка виконується з певною ймовірністю. Мета мутації полягає у запобіганні передчасній збіжності популяції, тобто уникнення так званого локального максимуму, а також у підтриманні її різноманіття. Кожний ген нової особи випадково змінюється із деякою вірогідністю. У випадку кількісних генів це може бути відхилення від початкового значення, для якісних – заміна на інше

допустиме значення, для секвенційних – перестановка або зміна елементів у межах хромосоми.

Ін'єкція – це введення у популяцію нових, випадково згенерованих особин, які не є нащадками попередніх поколінь. Її основна мета – вихід із ситуації локального максимуму та генетичної схожості всіх осіб у деякому поколінні, якщо її не вдалося запобігти. Ін'єкція виконується з певною ймовірністю, і частина популяції замінюється новими випадковими особами.

Таким чином, кросинговер відповідає за комбінування існуючих осіб, мутація забезпечує випадкові зміни, а ін'єкція вводить нові елементи у популяцію. Разом ці оператори формують основу еволюційного пошуку, який забезпечує збалансоване поєднання стабільності та різноманіття в процесі генерації музики. Зміна вірогідності мутації чи ін'єкції може фундаментально впливати на швидкість та якість роботи алгоритму [24].

Було розроблено систему на основі модифікованого інтерактивного генетичного алгоритму, здатну генератувати та програвати невеликі музичні композиції за допомогою описаної вище технології MIDI. Після генерації чергового покоління S_n система пропонує користувачеві оцінити кожну з композицій за 10-бальною шкалою. Отримані значення y_i використовуються в якості фітнес-функції композицій, що відрізняє інтерактивні генетичні алгоритми від інших, де фітнес-функція розраховується деяким чином на основі геному.

3.2.1 Геном композицій

Система кодує кожну композицію за допомогою генома, що складається з двох якісних, одного кількісного та трьох секвенційних генів.

Якісні гени кодують значення гами та тоніки. Значення гамми зберігається у вигляді цілого числа, що кодує одне з можливих значень. За замовчуванням використовуються наступні можливі значення: мажорна,

натуральна мінорна, гармонічна мінорна. Значення тоніки зберігається у вигляді цілого числа, що кодує номер ноти.

Кількісний ген кодує значення кількості ударів на хвилину у вигляді цілого числа. За замовчуванням кількість ударів на хвилину приймає значення у діапазоні від 150 до 240.

Секвенційні гени кодують гармонічні, ритмічні послідовності та напрям руху нот в тактах. Гармонічна послідовність зберігається у вигляді списку, де кожен елемент описує тип аккорда. Ритмічна послідовність композиції зберігається у вигляді списку, де кожен елемент описує довжину ноти та наявність чи відсутність паузи. Напрям руху для кожного такту зберігається у вигляді списку, де кожен з елементів приймає бінарне значення: висхідний чи низхідний.

Таких даних вистачає для кодування композицій, що не обмежені одним жанром музики, як у системі GenJam. Система є гнучкою та піддається налаштуванню, що дозволяє задавати граничні значення для кількісних параметрів та можливі значення для дискретних параметрів.

3.2.2 Схема роботи алгоритму

Система має наступний робочий цикл:

– згенерувати початкову популяцію S_1 випадкових музичних композицій (осіб). Значення x (розмір покоління) встановлено у 5 емпіричним шляхом;

– продемонструвати користувачу поточну популяцію. Отримати значення оцінки y_i для кожної особи s_i від користувача;

– оцінити зацікавленість користувача δ;

– згенерувати нову популяцію з x осіб. Кожна з осіб популяції S_n буде нащадком двох осіб з попередньої популяції S_{n-1} з вірогідністю $1-\alpha$ (кросинговер) або результатом ін’єкції з вірогідністю α;

– провести мутації в новій популяції S_n. Вірогідність мутації кожного гену – β;

– повернутися до другого шагу.

Під час розмноження нащадок отримує випадкову комбінацію генів своїх батьків. Кожен з генів незалежно генеруються на основі батьківських генів. Гени, що кодують кількісні параметри обираються випадково між батьківськими значеннями. Гени, що кодують дискретних параметрів випадково приймають одне з батьківським значень. Гени, що кодують секвенційні параметри, генеруються на основі випадкового об’єднання послідовностей параметрів батьківських осіб. Батьківські особи обираються випадково за формулою (3.7):

$$P_i = \frac{y_i}{\sum_{j=0}^{n-1} y_j}, \tag{3.7}$$

де P_i – вірогідність вибору і-ї особи у якості батьківської;

y_i – і-та оцінка, отримана від користувача, $y_i \in [0,1]$.

Особливістю розробленої модифікованої системи є зміна вірогідності ін’єкцій та мутації на основі параметра зацікавленості користувача. Ця техніка є важливою для системи, що виконує таке завдання, як генерація музики, коли формальна оцінка якості згенерованої популяції неможлива. Використання цього параметра дозволяє динамічно змінювати швидкість змін у результаті мутацій та ін’єкцій у популяції музичних композицій, збільшуючи різноманітність пропорційно зацікавленості.

Параметр δ приймає значення від 0 до 1 та розраховується за формулами (3.8, 3.9):

$$\delta = \frac{\sigma}{0.5} * r_{max}, \tag{3.8}$$

$$r_{max} = \max(r_1, r_2, \dots, r_N), \tag{3.9}$$

де r_i – і-та оцінка, отримана від користувача, в діапазоні 0–1;

σ – середнє квадратичне відхилення значень r.

Вірогідність ін’єкції α розраховується за формулою (3.10):

$$\alpha = \alpha_{min} + (\alpha_{max} - \alpha_{min}) * (1 - \delta), \quad (3.10)$$

де α_{max} – максимальна вірогідність ін’єкції;

α_{min} – мінімальна вірогідність ін’єкції.

Вірогідність мутації β розраховується за формулою (3.11):

$$\beta = \beta_{min} + (\beta_{max} - \beta_{min}) * (1 - \delta), \quad (3.11)$$

де β_{max} – максимальна вірогідність мутації;

β_{min} – мінімальна вірогідність мутації.

Кросинговер реалізовано наступним чином. Для якісних генів використовується формула (3.12):

$$g_k^{(child)} = \begin{cases} g_k^{(parent1)}, \text{якщо } U(0,1) < 0.5 \\ g_k^{(parent2)}, \text{ якщо } U(0,1) \geq 0.5 \end{cases}, \quad (3.12)$$

де $g_k^{(child)}$ – значення гену k для згенерованої особи;

$g_k^{(parent1)}$ – значення гену k для першої батьківської особи;

$g_k^{(paren2)}$ – значення гену k для другої батьківської особи;

$U(x, y)$ – випадкова величина з рівномірним розподілом на інтервалі [x,y].

Для кількісних генів використовується формула (3.13):

$$g_k^{(child)} = g_k^{(parent1)} + \left(g_k^{(parent2)} - g_k^{(parent1)}\right) * U(0,1) \quad (3.13)$$

Для секвенційних генів використовуються формули (3.14, 3.15):

$$L_{child} = U(\min(L_1, L_2), \max(L_1, L_2)), \qquad (3.14)$$

де L_{child} – довжина згенерованого секвенційного гену;

L1 – довжина цього гену для першої батьківської особи;

L2 – довжина цього гену для другої батьківської особи.

$$g_{k,i}^{(child)} = \begin{cases} g_{k,i}^{(parent1)}, \text{якщо } i \leq L_1 \text{ і } (U(0,1) < 0.5 \text{ або } i > L_2) \\ g_{k,i}^{(parent2)}, \text{якщо } i \leq L_2 \text{ і } (U(0,1) \geq 0.5 \text{ або } i > L_1) \end{cases}, \qquad (3.15)$$

де $g_{k,i}$ – i-тий елемент k-го секвенційного гену.

Така реалізація кросинговеру коректно обробляє різні гени згідно їхньої природи. Наприклад, значення BPM (швидкості композиції) для згенерованої особи буде лежати між значеннями цього гену в батьківських особах. А послідовність акордів буде комбінацією акордів, що використовувалися в батьківських особах.

3.2.3 Процес генерації композицій

Генерація музичних композицій у розробленій системі здійснюється шляхом поетапного перетворення отриманих у результаті роботи генетичного алгоритму геномів у послідовності ритміко-мелодичних пар (p_i,d_i), що потім конвертується у відповідні послідовності подій MIDI. Процес генерації ритміко-мелодичних пар складається з наступних кроків:

– генерація послідовності акордів на основі даних з геному;

– генерація ритмічної послідовності композиції на основі даних з геному, що є незалежною від послідовності акордів;

– інтеграція нот акорду у межах кожного такту. Під час цього процесу напрямок руху нот (висхідного або низхідного) у кожному такті визначається на основі даних, що зберігаються у геномі;

– доповнення кожного такту похідними нотами, які обираються між сусідніми інтервалами з урахуванням уникнення дисонансів. Цей процес відбувається за допомогою генератора псевдовипадкових чисел на основі хеш-коду геному, що гарантує відтворюваність результатів.

3.3 Розроблене програмне забезпечення

Для системи було розроблено простий і лаконічний користувальницький інтерфейс, заснований на популярній кросс-платформенній бібліотеці .NET MAUI. Це дозволяє використовувати розроблену систему на платформах Windows та MacOS [16].

Система дозволяє прослуховувати та оцінювати кожну зі зегенрований у поточному поколінні композицій безліч разів у довільному порядку. Кожну з композицій можна експортувати у вигляді файлу у форматі “mid”. Система відображає поточне розраховане значення зацікавленості користувача у вигляді графічного індикатору знизу вікна. Також надається додатковий функціонал у вигляді експорту статистики, що містить оцінки користувача для кожної з композицій у кожному поколінні.

Розроблений програмний продукт складається з просторів імен C#„ що дозволяють генерувати музичні композиції на основі відгуків користувача за допомогою інтерактивного генетичного алгоритму, а також зберігати та відтворювати їх. Бібліотека .NET MAUI забезпечує простий та лаконічний користувальницький інтерфейс, що дозволяє використовувати систему навіть непідготовленим користувачам. Структуру просторів імен проєкту наведено в табл. 3.1.

Таблиця 3.1 – Структура модулів проєкту

Простір імен	Функціонал	На які простори імен посилається
Algo	Реалізація генетичного алгоритму та генерація композицій	Зовнішні
MIDI	Відтворення згенрований композицій	Зовнішні
Models	Шаблони користувальницького інтерфейсу	MIDI, Зовнішні
Pages	Керування користувальницьким інтерфейсом	Algo, Models, MIDI, Зовнішні

На рис. 3.1 міститься загальна схема структури програмного комплексу. Відображені основні розроблені компоненти та залежності ними.

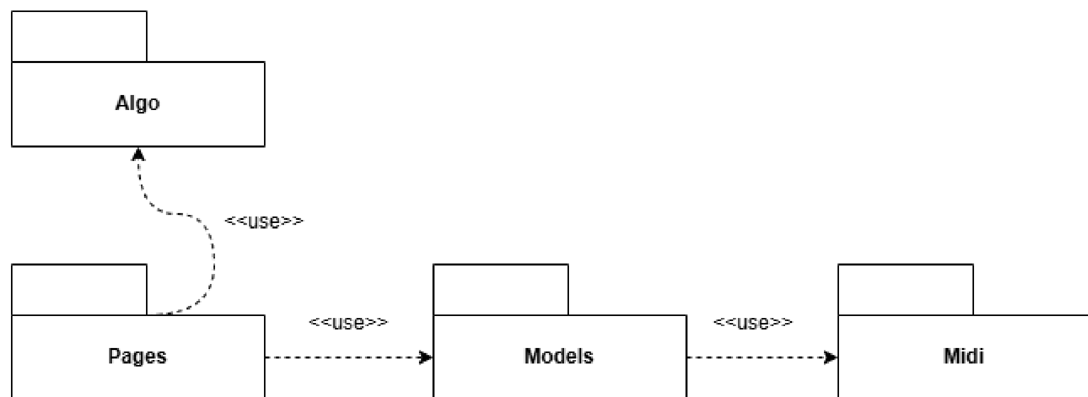


Рисунок 3.1 – Структура програмного продукту

На рис. 3.2 зображена UML-діаграма основних класів програмного комплексу. Класс MainPage обробляє високорівневі команди користувача та делегую реалізацію більш специфічним компонентам. Клас MidiPlayer відповідає за відтворення створених композицій з використанням бібліотеки DryWetMIDI. Клас GeneticAlgo приймає від користувача оцінки окремих треків та керує процесом генерації нових поколінь.

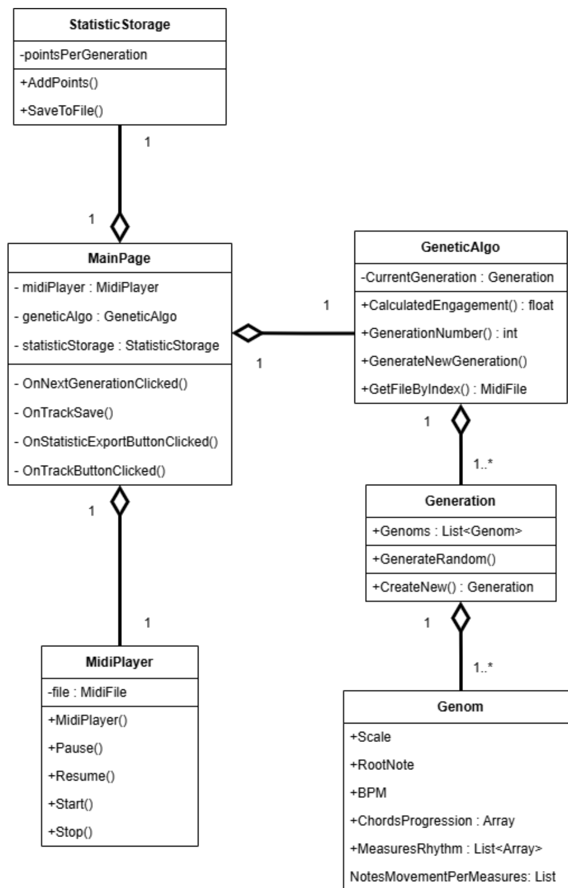


Рисунок 3.2 – UML-діаграма основних класів продукту

3.4 Висновок

У цьому розділі описано процес створення системи MIDI Music Generator для генерації музичних композицій на основі відгуків користувача за допомогою інтерактивного генетичного алгоритму. Розглянуто структуру використаного геному; математичний апарат, використаний для модифікації алгоритму; проблему зниження зацікавленості користувача та шляхи її вирішення.

4 ЕКСПЛУАТАЦІЯ, ТЕСТУВАННЯ ТА ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ПРОГРАМИ

4.1 Призначення й умови використання

Розроблений програмний продукт призначений для генерації музичних композицій на основі відгуків користувача за допомогою інтерактивного генетичного алгоритму. Він складається з просторів імен C#, що відповідають за генерацію композицій, роботу генетичного алгоритму, взаємодію користувача з використанням користувальницького графічного інтерфейсу.

Для коректної роботи ПП необхідні такі мінімальні системні вимоги:

– платформа з підтримкою .NET Desktop Runtime 9;

– процесор з мінімальною тактовою частотою 1,8 ГГц. Рекомендується використовувати як мінімум двоядерний процесор;

– оперативна пам'ять від 2 ГБ (рекомендується 4 Гб);

– роздільна здатність екрану 1024x768 пікселів з 16-бітовим кольором;

– мінімум 512 МБ вільного місця на жорсткому диску.

У своїй роботі ПП використовує такі необхідні компоненти:

– середовище .NET 9;

– бібліотеку MAUI.

4.2 Використання системи

Після запуску ПП відображається головне вікно програми, що відображено на рис. 4.1.

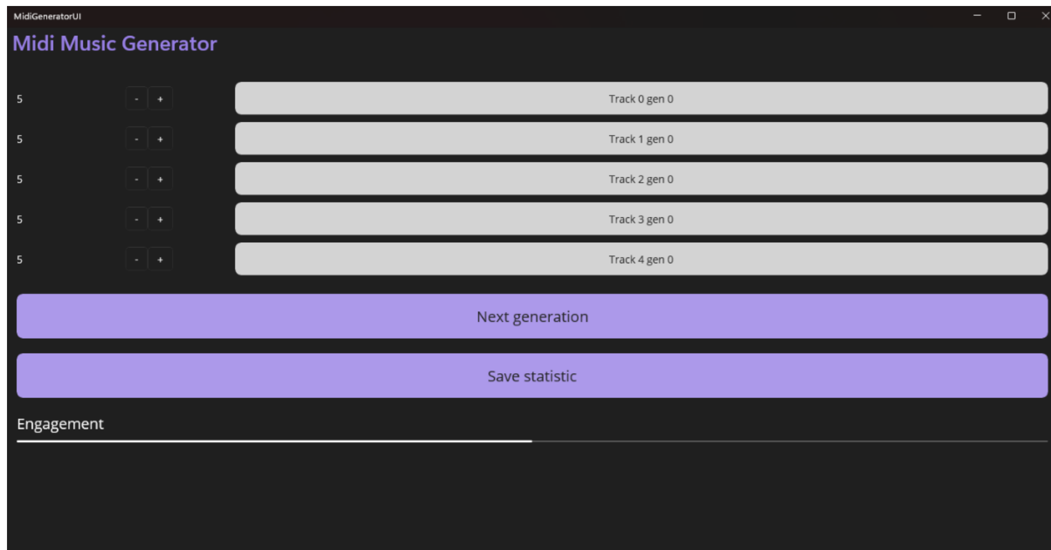


Рисунок 4.1 – Головне вікно програми

У центральній частині вікна наведений список музичних композицій з поточного покоління. Зліва від композиції наявні кнопки для зміни оцінки користувача, що надається кожній композиції у діапазоні від 1 до 10 балів. При натисканні на композиції відображається інтерфейс, наведений на рис. 4.2. Він надає наступні операції:

– Play. Ця кнопка запускає відтворення композиції;
– Pause. Ця кнопка ставить відтворення композиції на паузу;
– Stop. Ця кнопка зупиняє відтворення композиції. Під час наступного натискання на Play композиція буде відтворена з початку;
– Save. Ця кнопка відкриває стандартний інтерфейс операційної системи для збереження композиції.

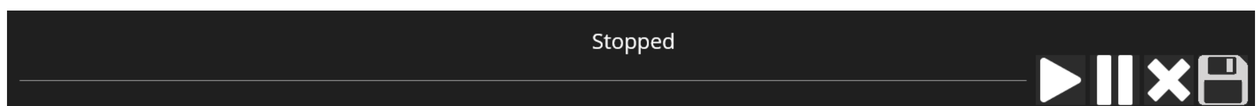


Рисунок 4.2 – Інтерфейс взаємодії з композицією

Після прослуховування всіх композицій користувач може виставити кожній композиції оцінку, а після натиснути на кнопку Next generation. Наступне покоління буде згенероване, а всі оцінки будут скинуті на значення за замовчуванням – 5 балів.

Кнопка Save statistic дозволяє зберегти статистику оцінок користувача для всіх попередніх поколінь у форматі csv. Ця кнопка відкриває стандартний інтерфейс операційної системи для збереження файлу.

У нижній частині головного вікна програми відображається рівень зацікавленості користувача, розрахований за формулою, наведеною вище. Рівень зацікавленості користувача надається в інформаційних цілях.

4.3 Повідомлення

Під час роботи ПП на екран може бути виведено ряд повідомлень, що надають користувачу інформацію про результати виконання операцій у зручного вигляді. Список повідомлень наведено в табл. 4.1.

Таблиця 4.1 – Структура модулів проєкту

Повідомлення	Опис
Next generation generated	Успішно згенеровано нове покоління композицій
Statistics saved successfully	Статистика за поколіннями експортована успішна
Failed to save statistics	Виникла помилка під час експорту статистики
Track saved successfully as <шлях>	Композиція експортована у файл <шлях>
Failed to save track	Виникла помилка під час збереження композиції

4.4 Тестування системи

На рис. 4.3 наведено приклад зегенрованої системою у першому поколінні композиції, яку користувач оцінив на мінімальну кількість балів. Композиція звучить як випадковий набір звуків і не є привабливою для користувача. На рис. 4.4 наведено композицію, що була згенерована

системою на 5 поколінні та оцінена на максимальну кількість балів. Якщо перша композиція характеризується різким переходом до використання коротких шістнадцятих нот та пауз, багаторазовим повторенням окремих нот в середині композиції, відсутністю гармонійного руху, то у другій композиції такі проблеми відсутні. У другій композиції спостерігається гармонійний рух, який кілька разів чергує підйоми і спуски, відсутнє різке і ненатуральне використання коротких нот і пауз. Відсутнє повторення однакових нот кілька разів поспіль, замість цього можна побачити прості, але ефективні техніки створення напруги для слухача і її розрядки. У цілому друга композиція сприймається як більш естетично приємна завдяки більшій музикальності, структурній узгодженості.

Рисунок 4.3 – Приклад суб’єктивно непривабливої композиції

Рисунок 4.4 – Приклад суб’єктивно більш привабливої композиції

Тестування розробленої програмної системи проводилося на комп’ютері з операційною системою Windows 11, обладнаною центральний процесором Intel I7-12650H. В табл. 4.2 та на рис. 4.5 наведено результати тестування упродовж 10 поколінь. За результатами тестів встановлено, що середній час для генерації наступного покоління на даному комп’ютері складає менше 1 мілісекунди, а в найгіршому випадку – близько двох мілісекунд, що вказує на високу швидкодію системи та невелику витрати ресурсів. Час, що витрачається на генерацію наступного покоління, є непомітним для реальних користувачів системи.

Таблиця 4.2 – Результати тестування швидкодії системи

Номер покоління	Витрачений час, мілісекунди
1	2,3868
2	0,3068
3	0,1194
4	0,119
5	0,0923
6	0,0747
7	0,0646

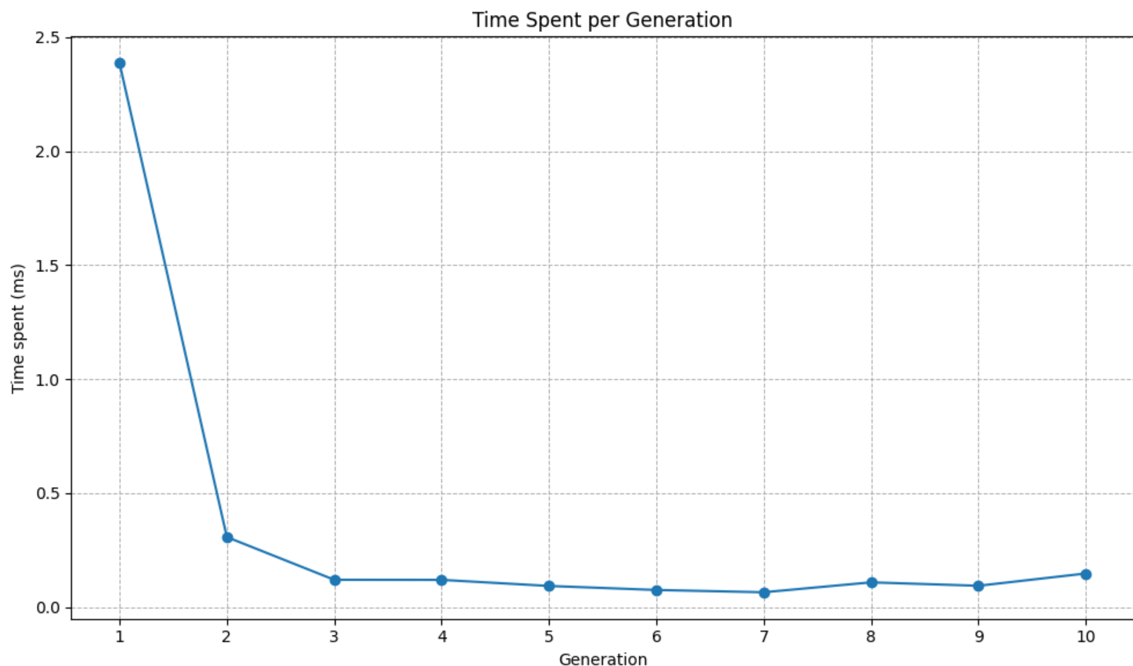


Рисунок 4.5 – Результати тестування швидкодії системи

4.5 Порівняння системи з існуючими аналогами

Хоча MIDI Music Generator, як і більшість моделей, заснованих на генетичних алгоритмах, все ще обмежена створенням лише монофонічних композицій, це обмеження слід розглядати в більш широкому контексті її характеристик продуктивності. Для багатьох практичних застосувань: фонової музики для ігор, освітніх інструментів або інтерактивних арт-інсталяцій, монофонічного звуку може бути достатньо. Що ще важливіше,

система MIDI Music Generator демонструє кілька помітних переваг, які відрізняють її від аналогів.

По-перше, динамічне відстеження зацікавленості користувачів безпосередньо вирішує одну з ключових недоліків інтерактивних генетичних алгоритмів: втому користувачів. Постійно адаптуючи ймовірності мутацій та ін'єкцій, система зменшує повторювані або непродуктивні ітерації, дозволяючи користувачам досягти задовільних результатів за меншу кількість циклів. Цей підхід можна розглядати як легку альтернативу сурогатним фітнес-функціям, які вимагають значних обчислювальних ресурсів і які часто важко узагальнити для різних користувачів.

По-друге, обчислювальна ефективність моделі робить її практичною для застосування в режимі реального часу. Тести підтверджують, що нові покоління можуть бути створені менш ніж за одну мілісекунду, навіть на дешевому обладнанні. Таким чином, система може використовуватися з програмним забезпеченням, де продуктивність є критично важливою, без необхідності хмарної обробки або дорогого обладнання. Наприклад, у відеоіграх або інтерактивних виставах у реальному часі.

По-третє, система дозволяє користувачам налаштовувати параметри, такі як дозволені темп, гама та набори акордів, тим самим забезпечуючи більшу відповідність створеної музики бажаному жанру або стилю. Ця адаптивність робить систему більш гнучкою, ніж такі рішення, як GenJam або DarwinTunes, які мають жанрові або структурні обмеження.

Підсумовуючи недоліки та переваги системи, хоча відсутність поліфонічних можливостей є структурним обмеженням обраної структури геному. Висока продуктивність системи, її адаптивність та інноваційний підхід до зменшення втоми користувачів роблять її перспективним і практичним інструментом. Ці сильні сторони в сукупності свідчать про те, що запропоноване рішення може слугувати не тільки дослідним прототипом, але й основою для реальної інтерактивної системи генерації музики.

Подальшими перспективами розвитку системи можуть бути розширення системи до поліфонічної генерації, використання машинного навчання для побудови сурогатних функцій оцінки, а також розробку вебсторінки або мобільної версії з підтримкою спільної взаємодії кількох користувачів.

В табл. 4.3 наведено порівняння розробленої системи MIDI Music Generator з розглянутими вище аналогами.

Таблиця 4.3 – Порівняння розробленої системи з аналогами

Система	Тип музики	Сурогатна фітнес-функція	Розрахунок зацікавленості користувача	Монофонічна музика
MIDI Music Generator	Довільні компактні композиції	Ні	Так	Так
GenJam [6]	Джазові соло	Ні	Ні	Так
GP Music [8]	Компактні композиції та однакова довжина нот	Так	Ні	Так
DarwinTunes [7]	Компактні зациклені композиції	Ні	Ні	Так

4.6 Висновок

У цьому розділі описані умови використання та процес тестування системи MIDI Music Generator для генерації музичних композицій на основі відгуків користувача за допомогою інтерактивного генетичного алгоритму. Було проведено порівняння розробленої системи з існуючими аналогами. Встановлено, що відмінностями розробленої системи є її універсальність, висока швидкодія, система відслідковування зацікавленості користувачів.

ВИСНОВКИ

У ході виконання дипломної роботи було розроблено, досліджено та реалізовано програмне забезпечення для генерації музичних композицій із використанням інтерактивного генетичного алгоритму. Розроблена система MIDI Music Generator дозволяє генерувати музичні композиції у форматі MIDI в реальному часі, ґрунтуючись на суб’єктивних оцінках користувачів. Це забезпечує можливість генерувати суб’єктивно привабливі композиції для будь-яких користувачів.

Було проведено аналіз подібних систем з літератури, таких як GenJam, GP-Music та DarwinTunes. Виявлено ряд їхніх спільних недоліків: обмеження за жанром, відсутність адаптації до зміни рівня зацікавленності користувача. Запропонована система MIDI Music Generator усуває ці недоліки шляхом застосування модифікованого інтерактивного генетичного алгоритму, у якому ймовірність мутацій та ін’єкцій динамічно змінюється залежно від показника зацікавлености користувача, що розраховується після кожного обробленого покоління композицій. Це дозволяє зменшити втому користувача та поліпшити еволюційний процес.

Експериментальне дослідження системи підтвердило її високу швидкодію: середній час генерації нового покоління становить менше 1 мілісекунди. Це свідчить про придатність розробленої моделі до використання в інтерактивних системах реального часу, таких як комп’ютерні ігри, освітні інструменти, генератори фонової музики.

Розроблена система MIDI Music Generator реалізована на мові C# з використанням фреймворку .NET MAUI, що забезпечує кросплатформність і зручність користувацького інтерфейсу. Система є модульною, розширюваною і може бути інтегрована до більших проєктів без значних змін у коді.

Порівняльний аналіз показав, що розроблена система перевершує аналоги, дозволяючи точно налаштовувати процес генерації, а також не

обмежуючи користувача один жанром композицій. Водночас обмеженням є підтримка лише монофонічної музики. Тому у подальшому можливо розширення системи для поліфонічної генерації, використання машинного навчання для побудови сурогатних функцій оцінки. Можливою є розробка вебсторінки або мобільної версії системи з підтримкою спільної взаємодії кількох користувачів.

Підсумовуючи, результати роботи доводять доцільність використання інтерактивних генетичних алгоритмів у завданнях генерації музики. Запропонований підхід забезпечує баланс між обчислювальною ефективністю, творчою варіативністю та інтерактивністю, що робить його перспективним для подальших досліджень у галузі штучного інтелекту.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. Collins K. An introduction to procedural music in video games [Electronic resource] / Karen Collins // Contemporary music review. – 2009. – Vol. 28, no. 1. – P. 5–15. – Access mode: https://doi.org/10.1080/07494460802663983.
2. Musenet [Electronic resource] // openai.com. – Access mode: https://openai.com/index/musenet/.
3. MusicLM: generating music from text [Electronic resource] / Andrea Agostinelli [et al.] // arXiv.org. – Access mode: https://doi.org/10.48550/arXiv.2301.11325.
4. Ji S. A survey on deep learning for symbolic music generation: representations, algorithms, evaluations, and challenges [Electronic resource] / Shulei Ji, Xinyu Yang, Jing Luo // ACM computing surveys. – 2023. – Access mode: https://doi.org/10.1145/3597493.
5. Takagi H. Interactive evolutionary computation: fusion of the capabilities of EC optimization and human evaluation [Electronic resource] / H. Takagi // Proceedings of the IEEE. – 2001. – Vol. 89, no. 9. – P. 1275–1296. – Access mode: https://doi.org/10.1109/5.949485.
6. Biles J. A. GenJam: an interactive genetic algorithm jazz improviser [Electronic resource] / John A. Biles // The journal of the acoustical society of america. – 1997. – Vol. 102, no. 5. – P. 3181. – Access mode: https://doi.org/10.1121/1.420841.
7. Evolution of music by public choice [Electronic resource] / Robert M. MacCallum [et al.] // Proceedings of the national academy of sciences. – 2012. – Vol. 109, no. 30. – P. 12081–12086. – Access mode: https://doi.org/10.1073/pnas.1203182109.
8. Johanson B. GP-Music: an interactive genetic programming system for music generation with automated fitness raters [Electronic resource] / Brad Johanson, Riccardo Poli // Genetic programming 1998: proceedings of the

third annual conference. – San Francisco, 1998. – P. 181–186. – Access mode: http://graphics.stanford.edu/~bjohanso/papers/gp98/johanson98gpmusic.pdf.

9. Stein M. A comparison of five programming languages in a graph clustering scenario [Electronic resource] / Martin Stein // JUCS - journal of universal computer science. – 2013. – Vol. 19, no. 3. – P. 428–456. – Access mode: https://doi.org/10.3217/jucs-019-03-0428.

10. Overview - A tour of C# [Electronic resource] // Microsoft Learn: Build skills that open doors in your career. – Access mode: https://learn.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/overview.

11. Java documentation - get started [Electronic resource] // Oracle Help Center. – Access mode: https://docs.oracle.com/en/java/.

12. Python 3.14 documentation [Electronic resource] // Python documentation. – Access mode: https://docs.python.org/3/.

13. Nanz S. A comparative study of programming languages in rosetta code [Electronic resource] / Sebastian Nanz, Carlo A. Furia // 2015 IEEE/ACM 37th IEEE international conference on software engineering (ICSE), Florence, Italy, 16–24 May 2015. – [S. l.], 2015. – Access mode: https://doi.org/10.1109/icse.2015.90.

14. What is windows forms - windows forms [Electronic resource] // Microsoft Learn: Build skills that open doors in your career. – Access mode: https://learn.microsoft.com/en-us/dotnet/desktop/winforms/overview.

15. What is windows presentation foundation - WPF [Electronic resource] // Microsoft Learn: Build skills that open doors in your career. – Access mode: https://learn.microsoft.com/en-us/dotnet/desktop/wpf/overview.

16. .Net maui [Electronic resource] // Microsoft Learn: Build skills that open doors in your career. – Access mode: https://learn.microsoft.com/en-us/dotnet/maui/what-is-maui?view=net-maui-9.0.

17. Global development survey 2017, volume 1 [Electronic resource] // Evans Data Corporation. – Access mode: https://evansdata.com/_download/overviews/EDC_Global_%202017_V1_Overvie

w.pdf.

18. Visual studio community | Overview [Electronic resource] // Visual Studio. – Access mode: https://visualstudio.microsoft.com/vs/community.

19. JetBrains. Rider: The Cross-Platform .NET IDE from JetBrains [Electronic resource] / JetBrains // JetBrains. – Access mode: https://www.jetbrains.com/rider.

20. Microsoft. Visual Studio Code - The open source AI code editor [Electronic resource] / Microsoft // Visual Studio Code - The open source AI code editor. – Access mode: https://code.visualstudio.com.

21. MIDI history chapter 7- MIDI associations (1983-1985) [Electronic resource] // midi.org. – Access mode: https://midi.org/midi-history-chapter-7-midi-associations-1983-1985.

22. Patel D. MIDI 2.0 devices market research report 2033 [Electronic resource] / Debadatta Patel // growthmarketreports.com. – Access mode: https://growthmarketreports.com/report/midi-20-devices-market.

23. Standard MIDI Files [Electronic resource] // MIDI.org. – Access mode: https://midi.org/standard-midi-files.

24. An introduction to genetic algorithms. – Cambridge, Mass : MIT Press, 1998. – 221 p.

ДОДАТОК А

Технічне завдання

Вступ

Розроблений програмний продукт призначений для генерації музичних композицій на основі відгуків користувача за допомогою інтерактивного генетичного алгоритму. Він складається з просторів імен C#, що відповідають за генерацію композицій, роботу генетичного алгоритму, взаємодію користувача з використанням користувальницького графічного інтерфейсу. ПП «MIDI Music Generator» представлений у вигляді окремого віконного додатку, що дозволяє здійснювати генерацію музичних композицій на основі відгуків користувача, а також прослуховувати за зберігати згенеровані композиції.

А.1 Мета розробки і її призначення

Об’єкт дослідження – процес генерації музичних композицій на основі відгуків користувача за допомогою інтерактивного генетичного алгоритму.

Предмет дослідження – методи генерації музичних композицій на основі відгуків користувача.

Мета роботи – дослідження та програмна реалізація методу автоматичного виявлення переломів шийного відділу хребта на основі знімків комп’ютерної томографії.

Актуальність та проблематика роботи полягає в тому, що інтерактивна генерація музики широко використовується у різних галузях індустрії розваг. Методи генерації на основі генеративних нейронних мереж, що набрали популярність в останні роки, вимагають значних обсягів обчислювальних ресурсів, зокрема оперативної пам’яті та процесорного часу, зазвичай надаються на платній основі, а також не можуть бути використані в реальному часі. Методи на основі генетичного алгоритму потребують значно

менших обчислювальних ресурсів та знаходять використання у випадках, коли генерація повинна відбуватися в реальному часі.

А.2 Стислий зміст основних теоретичних посилок до розробки

Динамічна генерація музики є відомою технологією та використовується у різних цифрових продуктах ще з кінця XX сторіччя. Типовими прикладами такого використання є комп'ютерні ігри Spore, No Man's Sky, Mini Metro. Використання згенерованої музики дозволяє зекономити кошти, що могли б піти на оплату праці людини-композитора. Така музика також покращує досвід користувача за рахунок ефекту новизни досвіду використання продукту [1].

Одним із рішень, що не потребують значних обсягів обчислень, є генетичні алгоритми, зокрема інтерактивні генетичні алгоритми (IGA). Такі методи є модифікацією стандартного генетичного алгоритма у випадках, коли fitness-function неможливо розрахувати об'єктивно. Тоді ввод від користувачів використовується замість fitness-function, керуючи еволюційним процесом. Ці методи широко використовуються у сфері генерації музики, де суб'єктивна привабливість одного і того ж треку залежить від естетичних уподобань конкретного користувача або групи користувачів [5].

А.3 Підстави для розробки

Підставою для виконання розробки є завдання на дипломний проєкт на тему “Розробка та дослідження програмного забезпечення для генерації музики з використанням інтерактивного генетичного алгоритму”, затверджене наказом по НУ «Запорізька політехніка» № 447 від 30 вересня 2025 року.

А.4 Основні вимоги до програми, початковим даним і результату

А.4.1 Вимоги до функціональних характеристик

Вимоги до функціональних характеристик:

– розроблений ПП повинен мати простий та зрозумілий графічний інтерфейс користувача;

– розроблений ПП повинен приймати від користувачів числову оцінку суб'єктивної привабливості музичних композицій;

– розроблений ПП повинен надавати можливість згенерувати нове покоління композицій;

– розроблений ПП повинен надавати можливість програвати зегенровані на поточному поколінні композиції у довільному порядку довільну кількість разів;

– розроблений ПП повинен надавати можливість експортувати згенеровані композиції у форматі “MIDI”;

– розроблений ПП повинен надавати можливість експортувати статистику оцінок користувачів у рамках поточної сессії роботи у форматі “CSV”.

А.4.2 Вимоги до надійності

ПП повинен нормально функціонувати при безперервній роботі комп'ютера. У разі виникнення збоїв в роботі апаратури, відновлення нормальної роботи ПП повинно проводитися після перезавантаження системи.

А.4.3 Умови експлуатації

Умови експлуатації:

– програмне забезпечення не потребує обслуговуючого персоналу;

– для користування програмним забезпеченням не потрібна додаткова кваліфікація.

А.4.4 Вимоги до складу та параметрів технічних засобі

Вимоги до складу та параметрів технічних засобів:

– платформа з підтримкою .NET Desktop Runtime 9;

– процесор з мінімальною тактовою частотою 1,8 ГГц. Рекомендується використовувати як мінімум двоядерний процесор;

– оперативна пам'ять від 2 ГБ (рекомендується 4 Гб);

– роздільна здатність екрану 1024x768 пікселів з 16-бітовим кольором;

– мінімум 512 МБ вільного місця на жорсткому диску.

У своїй роботі ПП використовує такі необхідні компоненти:

– середовище .NET 9;

– бібліотеку MAUI.

А.4.5 Вимоги до маркування та упакування

Програмне забезпечення не потребує додаткових дій при маркуванні та упакуванні.

А.4.6 Вимоги до транспортування та збереження

Вимоги до транспортування й зберігання аналогічні вимогам, які пропонуються до носія, на якому записана програма.

А.5 Вимоги до програмної документації

Склад програмної документації повинен включати в себе:

– технічне завдання;
– керівництво програміста;
– керівництво оператора.

А.6 Обмеження на установку і використання програмного продукту

Розроблений програмний продукт потребує інсталяції. Програмний продукт повинен використовуватись на комп’ютерах, які задовольняють вимогам та параметрам технічних засобів. Програмний продукт може використовуватись користувачем у будь-який момент часу, не конфліктуючи з іншими програмами.

ДОДАТОК Б

Опис програмного продукту

Б.1 Загальні відомості

Назва програмного продукту – «MIDI Music Generator».

Розроблений програмний продукт являє собою програму на мові C#, призначену для генерації музичних композицій на основі відгуків користувача за допомогою інтерактивного генетичного алгоритму.

Для коректної роботи ПП необхідні такі мінімальні системні вимоги:

– платформа з підтримкою .NET Desktop Runtime 9;

– процесор з мінімальною тактовою частотою 1,8 ГГц. Рекомендується використовувати як мінімум двоядерний процесор;

– оперативна пам'ять від 2 ГБ (рекомендується 4 Гб);

– роздільна здатність екрану 1024x768 пікселів з 16-бітовим кольором;

– мінімум 512 МБ вільного місця на жорсткому диску.

У своїй роботі ПП використовує такі необхідні компоненти:

– середовище .NET 9;

– бібліотеку MAUI.

Б.2 Функціональне призначення

Вимоги до функціональних характеристик:

– розроблений ПП повинен мати простий та зрозумілий графічний інтерфейс користувача;

– розроблений ПП повинен приймати від користувачів числову оцінку суб’єктивної привабливості музичних композицій;

– розроблений ПП повинен надавати можливість згенерувати нове покоління композицій;

– розроблений ПП повинен надавати можливість програвати зегенровані на поточному поколінні композиції у довільному порядку довільну кількість разів;

– розроблений ПП повинен надавати можливість експортувати згенеровані композиції у форматі “MIDI”;

– розроблений ПП повинен надавати можливість експортувати статистику оцінок користувачів у рамках поточної сессії роботи у форматі “CSV”.

Б.3 Вибір інструментарію розробки програмного продукту

Для розробки ПП використано мову C#. Для розробки користувальницького інтерфейсу обрано фреймворк MAUI. Для розробки ПП було обрано безкоштовне IDE Visual Studio Community.

Б.4 Виклик і завантаження

Для запуску ПП необхідно запустити файл MidiGeneratorUI.exe. Після завантаження програми буде відображено її головне вікно, яке наведено на рис. Б.1.

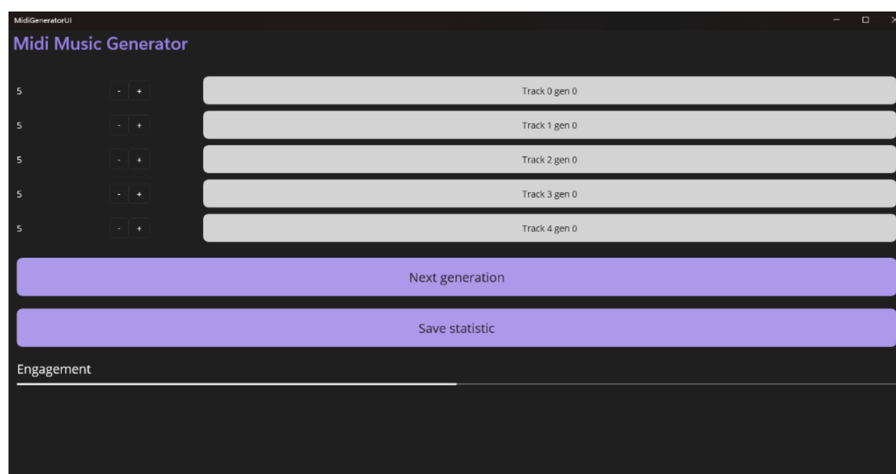


Рисунок Б.1 – Головне вікно ПП

ДОДАТОК В

Текст програми

В.1 Текст файлу MauiProgram.cs

```
using Microsoft.Extensions.Logging;
using CommunityToolkit.Maui;

namespace MidiGeneratorUI
{
    public static class MauiProgram
    {
        public static MauiApp CreateMauiApp()
        {
            var builder = MauiApp.CreateBuilder();
            builder
                .UseMauiApp<App>()
                .UseMauiCommunityToolkit()
                .ConfigureFonts(fonts =>
                {
                    fonts.AddFont("OpenSans-Regular.ttf", "OpenSansRegular");
                    fonts.AddFont("OpenSans-Semibold.ttf", "OpenSansSemibold");
                });

#if DEBUG
            builder.Logging.AddDebug();
#endif

            return builder.Build();
        }
    }
}
```

В.2 Текст файлу MainPage.xaml

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="MidiGeneratorUI.Pages.MainPage">

    <VerticalStackLayout Padding="20" Spacing="20">

        <!-- Button list -->
        <CollectionView x:Name="SelectableButtonList"
                ItemsSource="{Binding Buttons}"
                SelectionMode="None">
            <CollectionView.ItemTemplate>
```

```
            <DataTemplate>
               <Grid ColumnSpacing="10" Padding="0,5">
                  <Grid.ColumnDefinitions>
                     <ColumnDefinition Width="1*" />
                     <ColumnDefinition Width="1*" />
                     <ColumnDefinition Width="8*" />
                  </Grid.ColumnDefinitions>

                  <Label Text="{Binding NumericValue}"
                  Grid.Column="0"
                  VerticalOptions="Center"
                  HorizontalOptions="Start"
                  FontSize="14" />

                  <Stepper Value="{Binding NumericValue, Mode=TwoWay}"
                  Grid.Column="1"
                  Minimum="0"
                  Maximum="10"
                  Increment="1"
                  WidthRequest="120"
                  HorizontalOptions="Start" />


                  <!-- Main button -->
                  <Button Text="{Binding Text}"
                  Grid.Column="2"
                  BackgroundColor="{Binding                        IsClickedSelected,
Converter={StaticResource BoolToColorConverter}}"
                  Clicked="OnButtonClicked"/>

               </Grid>
            </DataTemplate>
         </CollectionView.ItemTemplate>
      </CollectionView>


      <!-- Progress bar section -->
      <Grid x:Name="ProgressSection"
     IsVisible="{Binding PlayerModel.IsVisible}">
         <Grid.RowDefinitions>
            <RowDefinition Height="Auto"/>
            <RowDefinition Height="Auto"/>
         </Grid.RowDefinitions>
         <Grid.ColumnDefinitions>
            <ColumnDefinition Width="*" />
```

```
        <ColumnDefinition Width="Auto" />
    </Grid.ColumnDefinitions>

    <!-- Label on top -->
    <Label Grid.Row="0" Grid.ColumnSpan="2"
   Text="{Binding PlayerModel.LabelText}"
   FontSize="24"
   HorizontalOptions="Center" />

    <!-- ProgressBar -->
    <ProgressBar Grid.Row="1" Grid.Column="0"
         HeightRequest="10"
         Progress="{Binding PlayerModel.Progress}" />

    <!-- Buttons -->
    <HorizontalStackLayout Grid.Row="1" Grid.Column="1"
                Spacing="5"
                Margin="10,0,0,0">
        <ImageButton Source="play.png"
            Command="{Binding PlayerModel.PlayCommand}" />
        <ImageButton Source="pause.png"
            Command="{Binding PlayerModel.PauseCommand}" />
        <ImageButton Source="stop.png"
            Command="{Binding PlayerModel.StopCommand}" />
        <ImageButton Source="save.png"
            Command="{Binding PlayerModel.SaveCommand}" />
    </HorizontalStackLayout>
</Grid>

<!-- Next gen button -->
<Button x:Name="NextGenerationButton"
        Text="Next generation"
        FontSize="20"
        HeightRequest="60"
        VerticalOptions="EndAndExpand"
        Clicked="OnNextGenerationClicked"/>

<!-- Write statistic button -->
<Button x:Name="StatisticExportButton"
        Text="Save statistic"
        FontSize="20"
        HeightRequest="60"
        VerticalOptions="EndAndExpand"
        Clicked="OnStatisticExportButtonClicked"/>
```

```
<StackLayout x:Name="EngagementSection" Spacing="5"
IsVisible="True">

     <Label Text="Engagement"
    FontSize="20"
    HorizontalOptions="Start"
    TextColor="White"/>

     <ProgressBar x:Name="EngagementProgressBar"
         Progress="0"
         HeightRequest="10"
         HorizontalOptions="FillAndExpand" />
</StackLayout>

   </VerticalStackLayout>

</ContentPage>
```

B.3 Текст файлу MainPage.xaml.cs

```
using CommunityToolkit.Maui.Storage;
using Melanchall.DryWetMidi.Core;
using Microsoft.Maui.Controls;
using MidiGeneratorUI.Algo;
using MidiGeneratorUI.Models;
using System.Collections.ObjectModel;
using Windows.Media.Playback;
using MidiGeneratorUI;
using MidiGeneratorUI.MIDI;

namespace MidiGeneratorUI.Pages
{
   public partial class MainPage : ContentPage
   {
      public ObservableCollection<SelectableButtonModel> Buttons { get; set; } =
[];
      public MidiPlayerModel PlayerModel { get; } = new();

      MidiPlayer? midiPlayer;

      GeneticAlgo geneticAlgo = new(5, 0.1f, 0.2f);

      StatisticStorage statisticStorage = new();
```

```
public MainPage()
{
    InitializeComponent();

    BindingContext = this;


    RepaintTracks();
    SetEngagementProgress(geneticAlgo.CalculatedEngagement);

    PlayerModel.IsVisible = false;
}

public void SetEngagementProgress(double normalizedValue)
{
    // Clamp between 0 and 1
    var clamped = Math.Max(0, Math.Min(1, normalizedValue));
    EngagementProgressBar.Progress = clamped;
}

void RepaintTracks()
{
    Buttons.Clear();

    foreach (var track in Enumerable.Range(0, geneticAlgo.TargetCount))
    {
        var model = new SelectableButtonModel()
        {
            Text = $"Track {track} gen {geneticAlgo.GenerationNumber}",
            NumericValue = 5,
            Index = track
        };
        Buttons.Add(model);
    }
}


private async void OnStatisticExportButtonClicked(object sender, EventArgs
e)
{
    var result = await FolderPicker.Default.PickAsync();

    if (result == null || !result.IsSuccessful)
    {
        return; // User cancelled
```

```
        }

        try
        {
            statisticStorage.SaveToFile(Path.Combine(result.Folder.Path,
"statistic.csv"));
            await DisplayAlert("Success", "Statistics saved successfully", "OK");
        }
        catch
        {
            await DisplayAlert("Error", "Failed to save statistics", "OK");
        }

    }

    private async Task OnTrackSave(MidiFile midiFile, string title)
    {
        var result = await FolderPicker.Default.PickAsync();

        if (result == null || !result.IsSuccessful)
        {
            return; // User cancelled
        }

        try
        {
            var targetPath = Path.Combine(result.Folder.Path, $"{title}.mid");
            midiFile.Write(targetPath);
            await DisplayAlert("Result", $"Track saved successfully as
{targetPath}", "OK");
        }
        catch
        {
            await DisplayAlert("Error", "Failed to save track", "OK");
        }
    }

    private void OnButtonClicked(object sender, EventArgs e)
    {
        if (sender is Button btn && btn.BindingContext is SelectableButtonModel
clickedModel)
        {
            foreach (var model in Buttons)
            {
                model.IsClickedSelected = model == clickedModel;
```

```
            }

            midiPlayer?.Dispose();
            midiPlayer = null;

            var midiFile = geneticAlgo.GetFileByIndex(clickedModel.Index);
            midiPlayer = new MidiPlayer(midiFile);


            PlayerModel.Attach(midiPlayer,                                    midiFile,
$"Track{clickedModel.Index}gen{geneticAlgo.GenerationNumber}",
OnTrackSave);
            PlayerModel.IsVisible = true;
         }
      }

      private void OnNextGenerationClicked(object sender, EventArgs e)
      {
         midiPlayer?.Dispose();
         midiPlayer = null;


         var points = Buttons.Select(b => (b.NumericValue + 1) / 11f).ToList();

         statisticStorage.AddPoints(Buttons.Select(b                               =>
(b.NumericValue)).ToArray());
         geneticAlgo.GenerateNewGeneration(points);

         PlayerModel.IsVisible = false;
         RepaintTracks();
         SetEngagementProgress(geneticAlgo.CalculatedEngagement);

         DisplayAlert("Result", "Next generation generated", "OK");
      }
   }
}
```

В.4 Текст файлу StatisticStorage.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
```

```
namespace MidiGeneratorUI.Models
{
   /// <summary>
   /// Stores user engagement statistics
   /// </summary>
   public class StatisticStorage
   {
      readonly Dictionary<int, int[]> pointsPerGeneration = new();
      int currentGeneration = 0;

      public void AddPoints(int[] points)
      {
         if (points == null || points.Length == 0)
         {
            throw new ArgumentException("Points cannot be null or empty.",
nameof(points));
         }

         if (pointsPerGeneration.Count > 0 &&
pointsPerGeneration.Values.First().Length != points.Length)
         {
            throw new ArgumentException("All points arrays must have the same
length.", nameof(points));
         }

         pointsPerGeneration[currentGeneration] = points;
         currentGeneration++;
      }

      public void SaveToFile(string filePath)
      {
         // Save as csv

         using (var writer = new System.IO.StreamWriter(filePath))
         {
            // Write header
            writer.WriteLine("Generation," + string.Join(",", Enumerable.Range(0,
pointsPerGeneration.Values.First().Length)));
            // Write data
            foreach (var kvp in pointsPerGeneration)
            {
               writer.WriteLine($"{kvp.Key}," + string.Join(",", kvp.Value.Select(v
=> v.ToString())));
            }
```

```
      }
    }
  }
}
```

В.5 Текст файлу SelectableButtonModel.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace MidiGeneratorUI.Models
{
  public class SelectableButtonModel : BindableObject
  {
    private bool isClickedSelected;

    public string Text { get; set; } = string.Empty;

    public int Index { get; set; } = 0;

    public bool IsClickedSelected
    {
      get => isClickedSelected;
      set
      {
        isClickedSelected = value;
        OnPropertyChanged();
      }
    }

    private int numericValue;

    public int NumericValue
    {
      get => numericValue;
      set
      {
        if (value != numericValue)
        {
          numericValue = value;
          OnPropertyChanged();
```

```
                }
            }
        }
    }

}
```

В.6 Текст файлу MidiPlayerModel.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Linq;
using System.Runtime.CompilerServices;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Input;
using Windows.Media.Playback;
using Melanchall.DryWetMidi.Core;
using CommunityToolkit.Maui.Storage;
using MidiGeneratorUI.MIDI;

namespace MidiGeneratorUI.Models
{
    public class MidiPlayerModel : INotifyPropertyChanged
    {
        public event PropertyChangedEventHandler? PropertyChanged;

        private MidiPlayer? player;
        private MidiFile? file;
        private bool isVisible;
        private string labelText = "";
        private float progress;
        private string title = "";

        public bool IsVisible
        {
            get => isVisible;
            set
            {
                if (isVisible != value)
                {
                    isVisible = value;
                    OnPropertyChanged();
```

```
        }
    }
}

public string LabelText
{
    get => labelText;
    set
    {
        if (labelText != value)
        {
            labelText = value;
            OnPropertyChanged();
        }
    }
}

public float Progress
{
    get => progress;
    private set
    {
        if (progress != value)
        {
            progress = value;
            OnPropertyChanged();
        }
    }
}

public ICommand PlayCommand { get; }
public ICommand PauseCommand { get; }
public ICommand StopCommand { get; }
public ICommand SaveCommand { get; }

Func<MidiFile, string, Task>? onSaveRequestedCallback;

public MidiPlayerModel()
{
    PlayCommand = new Command(OnPlay);
    PauseCommand = new Command(OnPause);
    StopCommand = new Command(OnStop);
    SaveCommand = new Command(OnSave);

    LabelText = "Ready";
```

```
}

void UpdateProgress()
{
    if (player == null)
        return;
    Progress = player.ProgressNormalized;
}

public void Attach(MidiPlayer midiPlayer, MidiFile midiFile, string title, Func<MidiFile, string, Task> onSaveRequestedCallback)
{
    this.title = title;
    file = midiFile;
    player = midiPlayer;
    Progress = player.ProgressNormalized;
    player.OnUpdate += UpdateProgress;
    player.OnFinished += OnStop;

    this.onSaveRequestedCallback = onSaveRequestedCallback;

    OnStop();
}

private async void OnSave()
{
    if (file == null)
    {
        return;
    }

    if (onSaveRequestedCallback != null)
    {
        await onSaveRequestedCallback.Invoke(file, title);
    }
}

private void OnPlay()
{
    if (player == null) return;

    LabelText = "Playing...";
    player.Start();
}
```

```
        private void OnPause()
        {
            LabelText = "Paused";
            player?.Pause();
        }

        private void OnStop()
        {
            if (player == null) return;

            LabelText = "Stopped";
            player.Stop();
            Progress = 0;
        }

        protected void OnPropertyChanged([CallerMemberName] string? name = null)
            => PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(name));
    }

}
```

В.7 Текст файлу MidiPlayer.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Melanchall.DryWetMidi.Core;
using Melanchall.DryWetMidi.Interaction;
using Melanchall.DryWetMidi.Multimedia;

namespace MidiGeneratorUI.MIDI
{
    /// <summary>
    /// Plays MIDI music
    /// </summary>
    public class MidiPlayer : IDisposable
    {
        readonly MidiFile file;
        readonly OutputDevice device;
        readonly Playback playback;
```

```
        bool wasDisposed = false;

        public bool IsPlaying => !wasDisposed && playback.IsRunning;


        public event Action? OnUpdate;
        public event Action? OnFinished;

        public MidiPlayer(MidiFile file)
        {
            this.file = file;
            device = OutputDevice.GetAll().First();
            playback = file.GetPlayback(device);

            playback.EventPlayed += (sen, args) => Update();
            playback.Stopped += (sen, args) => Update();
            playback.Started += (sen, args) => Update();
            playback.Finished += (sen, args) => Finish();
        }

        public float ProgressNormalized
        {
            get
            {
                if (!IsPlaying)
                    return 0;

                var ellapsed =
    (MetricTimeSpan)playback.GetCurrentTime(TimeSpanType.Metric);
                var total =
    (MetricTimeSpan)playback.GetDuration(TimeSpanType.Metric);

                float normalized = ellapsed.TotalMicroseconds /
    (float)total.TotalMicroseconds;

                return normalized;
            }
        }

        void Finish()
        {
            Stop();
            OnFinished?.Invoke();
        }
```

```
        protected virtual void Update()
        {
            OnUpdate?.Invoke();
        }

        public void Pause()
        {
            ObjectDisposedException.ThrowIf(wasDisposed, nameof(MidiPlayer));
            playback.Stop();
        }

        public void Resume()
        {
            ObjectDisposedException.ThrowIf(wasDisposed, nameof(MidiPlayer));
            Start();
        }

        public void Start()
        {
            ObjectDisposedException.ThrowIf(wasDisposed, nameof(MidiPlayer));

            playback.Start();
        }

        public void Stop()
        {
            ObjectDisposedException.ThrowIf(wasDisposed, nameof(MidiPlayer));

            playback.Stop();
            playback.MoveToStart();
        }

        public void Dispose()
        {
            playback.Dispose();
            device.Dispose();
            wasDisposed = true;
        }
    }
}
```

В.8 Текст файлу MusicGenerator.cs

```
using MathNet.Numerics;
using Melanchall.DryWetMidi.Common;
using Melanchall.DryWetMidi.Composing;
using Melanchall.DryWetMidi.Core;
using Melanchall.DryWetMidi.Interaction;
using Melanchall.DryWetMidi.Multimedia;
using Melanchall.DryWetMidi.MusicTheory;
using Note = Melanchall.DryWetMidi.MusicTheory.Note;
using Chord = Melanchall.DryWetMidi.MusicTheory.Chord;

namespace MidiGeneratorUI.Algo
{
    internal class MusicGenerator
    {

        /// <summary>
        /// Serves as an intermediate representation of a measure's part
        /// </summary>
        class GeneratedMeasurePart
        {
            public required Genom.RhythmPart RhythmPart { get; set; }
            public required Note? Note { get; set; }
        }

        static ChordProgression GenerateProgression(Genom genom)
        {
            var scale = new Scale(genom.Scale, genom.RootNote);

            return                                        ChordProgression.Parse(string.Join('-',
genom.ChordsProgression.Select(c => c.Name)), scale);
        }

        static List<Note> GeneratePassingNotesBetweenTwo(Note a, Note b)
        {

            if (a == b || Math.Abs(a.NoteNumber - b.NoteNumber) == 1)
            {
                return [a, b];
            }

            int midiNumberA = a.NoteNumber;
            int midiNumberB = b.NoteNumber;
```

```
            int start = Math.Min(midiNumberA, midiNumberB);
            int end = Math.Max(midiNumberA, midiNumberB);

            var possibleNotes = Enumerable.Range(start + 1, end - start - 1);

            return possibleNotes.Select(number =>
Note.Get((SevenBitNumber)number)).ToList();
        }

        static void AddPassingNotesToList(List<Note> notes, int targetCount,
Random random)
        {

            while (notes.Count < targetCount)
            {
                int randomIndex = random.Next(0, notes.Count() - 1);
                int nextIndex = randomIndex + 1;
                var passingNotesPossible =
GeneratePassingNotesBetweenTwo(notes[randomIndex], notes[nextIndex]);
                var passingNote = passingNotesPossible.SelectCombination(1,
random).Single();

                notes.Insert(nextIndex, passingNote);
            }
        }

        static List<Note> OrderNotes(IEnumerable<Note> notes,
Genom.NotesMovement notesMovement)
        {
            switch (notesMovement)
            {
                case Genom.NotesMovement.Up:
                    return notes.OrderBy(n => n.NoteNumber).ToList();
                case Genom.NotesMovement.Down:
                    return notes.OrderByDescending(n => n.NoteNumber).ToList();
                default:
                    throw new ArgumentException("Invalid notes movement type.");
            }
        }

        static void FillPatternBuilderWithMeasure(PatternBuilder patternBuilder,
IEnumerable<GeneratedMeasurePart> parts)
        {
            foreach (var part in parts)
```

```
            {
                if (part.Note != null)
                {
                    patternBuilder.Note(part.Note, part.RhythmPart.Length);
                }
                else
                {
                    patternBuilder.StepForward(part.RhythmPart.Length);
                }
            }
        }

        static List<GeneratedMeasurePart> GenerateMeasureFromRhythmAndChord(Octave octave, Chord chord, Genom.RhythmPart[] rhythm, Genom.NotesMovement notesMovement, int seed)
        {
            var result = new List<GeneratedMeasurePart>();

            var random = new Random(seed);

            var nonRestNotesCount = rhythm.Where(r => !r.IsRest).Count();

            var notesList = OrderNotes(chord.ResolveNotes(octave), notesMovement);

            if (nonRestNotesCount < notesList.Count)
            {
                throw new InvalidOperationException("Not enough notes in rhythm to insert chord.");
            }

            AddPassingNotesToList(notesList, nonRestNotesCount, random);

            int noteNumber = 0;
            for (int i = 0; i < rhythm.Length; i++)
            {
                var rhythmPart = rhythm[i];

                if (rhythmPart.IsRest)
                {
                    result.Add(new GeneratedMeasurePart
                    {
                        RhythmPart = rhythmPart,
                        Note = null
                    });
                }
```

```
            else
            {
                var note = notesList[noteNumber];
                noteNumber++;

                result.Add(new GeneratedMeasurePart
                {
                    RhythmPart = rhythmPart,
                    Note = note
                });
            }
        }

        return result;
    }

    static List<GeneratedMeasurePart> GenerateAllMeasures(Genom genom,
Octave octave)
    {
        var progression = GenerateProgression(genom);
        var result = new List<GeneratedMeasurePart>();

        for (int i = 0; i < progression.Chords.Count(); i++)
        {
            var chord = progression.Chords.ElementAt(i);
            var rhythm = genom.MeasuresRhythm[i %
genom.MeasuresRhythm.Count];
            var notesMovement = genom.NotesMovementPerMeasures[i %
genom.NotesMovementPerMeasures.Count];
            var measure = GenerateMeasureFromRhythmAndChord(octave, chord,
rhythm, notesMovement, genom.GetHashCode());
            result.AddRange(measure);
        }

        return result;
    }

    static Pattern GeneratePattern(Genom genom)
    {
        var patternBuilder = new PatternBuilder()
        .SetVelocity((SevenBitNumber)120)
        .SetOctave(Octave.Middle);


        var measures = GenerateAllMeasures(genom, Octave.Middle);
```

```
            FillPatternBuilderWithMeasure(patternBuilder, measures);


            var result = patternBuilder.Build();

            return result;
        }

        static TempoMap GenerateTempoMap(Genom genom)
        {
            return                                        TempoMap.Create(tempo:
Tempo.FromBeatsPerMinute(genom.BPM));
        }

        public static MidiFile GenerateTrack(Genom genom)
        {
            var result = GeneratePattern(genom);

            var tempo = GenerateTempoMap(genom);

            var file = result.ToFile(tempo);

            return file;
        }

        public static List<float> GenerateDigitalRepresentation(Genom genom)
        {
            var measures = GenerateAllMeasures(genom, Octave.Middle);

            var result = new List<float>();

            result.Add((float)(genom.BPM        -        GenomUtils.minBPM)        /
(GenomUtils.maxBPM - GenomUtils.minBPM)); // Add BPM scaled down to a
float representation


            foreach (var part in measures)
            {
                float noteRepresentation;
                if (part.Note != null)
                {
                    noteRepresentation = part.Note.NoteNumber / 127f; // Normalize
MIDI note number to [0, 1]
                }
                else
```

```
                {
                    noteRepresentation = -1f; // Represent rest as -1
                }

                var length = (float)part.RhythmPart.Length.Numerator / part.RhythmPart.Length.Denominator;

                result.Add(noteRepresentation);
                result.Add(length);
            }

            return result;

        }
    }
}
```

В.9 Текст файлу GenomUtils.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using MathNet.Numerics;
using Melanchall.DryWetMidi.Interaction;
using Melanchall.DryWetMidi.MusicTheory;

namespace MidiGeneratorUI.Algo
{
    internal static class GenomUtils
    {
        readonly static string[] chords = ["I", "II", "III", "IV", "V", "IV", "VII"];
        readonly static string[] characteristics = ["dim", "aug", "min6", "maj6", "min7", "maj7"];

        static Genom.ChordDescriptor GenerateRandomChord(Random random)
        {
            const float characteristicChance = 0.2f;

            var chord = chords[random.Next(chords.Length)];
            var characteristic = "";
            if (random.NextDouble() < characteristicChance)
            {
```

```
            characteristic = characteristics[random.Next(characteristics.Length)];
        }

        return new Genom.ChordDescriptor
        {
            Number = chord,
            Characteristic = characteristic
        };
    }

    static List<Genom.NotesMovement> GenerateRandomNotesMovement(int
count, Random random)
    {
        var result = new List<Genom.NotesMovement>();
        for (int i = 0; i < count; i++)
        {
            result.Add(random.NextDouble() < 0.5 ? Genom.NotesMovement.Up :
Genom.NotesMovement.Down);
        }
        return result;
    }

    static Genom.ChordDescriptor[] GenerateRandomChordProgression(Random
random)
    {
        const int minLength = 3;
        const int maxLength = 5;

        var length = random.Next(minLength, maxLength + 1);

        var progression = new Genom.ChordDescriptor[length];

        for (int i = 0; i < length; i++)
        {
            progression[i] = GenerateRandomChord(random);
        }

        return progression;
    }


    static List<MusicalTimeSpan> Divide(MusicalTimeSpan span)
    {
        return
        [
```

```
            new MusicalTimeSpan(span.Numerator, span.Denominator * 2),
            new MusicalTimeSpan(span.Numerator, span.Denominator * 2)
        ];
    }

    static void DivideSpanAtIndex(List<MusicalTimeSpan> spans, int index)
    {
        var span = spans[index];

        var divisionResults = Divide(span);

        spans.RemoveAt(index);

        spans.InsertRange(index, divisionResults);
    }

    public static readonly MusicalTimeSpan minTimeSpanUsed =
MusicalTimeSpan.Sixteenth;

    static int SelectRandomSpanIndex(List<MusicalTimeSpan> spans, Random
random)
    {
        var validSpans = Enumerable.Range(0, spans.Count).Where(i => spans[i] >
minTimeSpanUsed);
        if (validSpans.Count() == 0)
        {
            throw new ArgumentException("No valid spans available for
selection.");
        }
        var randomIndex = validSpans.SelectCombination(1, random).Single();
        return randomIndex;
    }

    static Genom.RhythmPart[] GenerateRandomRhythmPart(Random random)
    {
        const int targetLengthInWholes = 2;
        const int minNonRests = 4;
        const float maxRestsRatio = 0.2f;


        var spans = Enumerable.Repeat(MusicalTimeSpan.Half,
targetLengthInWholes * 2).ToList();


        while (spans.Count < minNonRests)
```

```
            {
                var randomIndex = SelectRandomSpanIndex(spans, random);
                DivideSpanAtIndex(spans, randomIndex);
            }

            const int minAdditionalDivisions = 2;
            const int maxAdditionalDivisions = 8;

            var additionalDivisionsCount = random.Next(minAdditionalDivisions,
maxAdditionalDivisions + 1);

            for (int i = 0; i < additionalDivisionsCount; i++)
            {
                var randomIndex = SelectRandomSpanIndex(spans, random);
                DivideSpanAtIndex(spans, randomIndex);
            }


            var result = spans.Select(span => new Genom.RhythmPart() { IsRest =
false, Length = span }).ToArray();

            int targetRestsNumber = random.Next(0, result.Length);

            targetRestsNumber = Math.Min(targetRestsNumber, result.Length -
minNonRests);
            targetRestsNumber = Math.Min(targetRestsNumber, (int)(result.Length *
maxRestsRatio));

            var restsIndices = Enumerable.Range(0,
result.Length).SelectCombination(targetRestsNumber, random);

            foreach (var restIndex in restsIndices)
            {
                result[restIndex].IsRest = true;
            }

            return result;
        }

        static List<Genom.RhythmPart[]> GenerateRhythmForMeasures(int count,
Random random)
        {
            var result = new List<Genom.RhythmPart[]>();
            for (int i = 0; i < count; i++)
            {
```

```
            var rhythmPart = GenerateRandomRhythmPart(random);
            result.Add(rhythmPart);
        }
        return result;
    }


    readonly static NoteName[] noteNames = [NoteName.CSharp, NoteName.D,
NoteName.DSharp, NoteName.E, NoteName.F, NoteName.FSharp, NoteName.G,
NoteName.GSharp, NoteName.A, NoteName.ASharp, NoteName.B];
    readonly static IEnumerable<Interval>[] scales = [ScaleIntervals.Major,
ScaleIntervals.Minor, ScaleIntervals.HarmonicMinor];
    public const int minBPM = 150;
    public const int maxBPM = 240;

    public static Genom GenerateRandomGenom(Random random)
    {
        var chordsProgression = GenerateRandomChordProgression(random);

        var rhythm = GenerateRhythmForMeasures(chordsProgression.Length,
random);

        var scale = scales[random.Next(scales.Length)];

        var bpm = random.Next(minBPM, maxBPM + 1);

        var rootNote = noteNames[random.Next(noteNames.Length)];

        var notesPositionsSeed = random.Next(0, int.MaxValue);

        var notesMovement =
GenerateRandomNotesMovement(chordsProgression.Length, random);

        return new Genom()
        {
            Scale = scale,
            RootNote = rootNote,
            BPM = bpm,
            ChordsProgression = chordsProgression,
            MeasuresRhythm = rhythm,
            NotesMovementPerMeasures = notesMovement
        };

    }
```

```
static Genom.ChordDescriptor[] CombineTwoProgressionsRandomly(Genom.ChordDescriptor[] progression1, Genom.ChordDescriptor[] progression2, Random random)
{
    int targetLength = random.Next(Math.Min(progression1.Length, progression2.Length), Math.Max(progression1.Length, progression2.Length) + 1);
    var result = new Genom.ChordDescriptor[targetLength];

    for (int i = 0; i < targetLength; i++)
    {
        List<Genom.ChordDescriptor> optionsToChooseFrom = [];

        if (i < progression1.Length)
        {
            optionsToChooseFrom.Add(progression1[i]);
        }
        if (i < progression2.Length)
        {
            optionsToChooseFrom.Add(progression2[i]);
        }

        result[i] = optionsToChooseFrom.SelectCombination(1, random).Single();
    }

    return result;
}

static List<Genom.NotesMovement> CombineTwoNoteMovementsRandomly(int targetCount, List<Genom.NotesMovement> movement1, List<Genom.NotesMovement> movement2, Random random)
{
    var result = new List<Genom.NotesMovement>();
    var chooseFrom1 = new List<Genom.NotesMovement>(movement1);
    var chooseFrom2 = new List<Genom.NotesMovement>(movement2);

    for (int i = 0; i < targetCount; i++)
    {
        List<int> options = [];

        if (chooseFrom1.Count > i)
        {
            options.Add(0);
```

```
            }
            if (chooseFrom2.Count > i)
            {
                options.Add(1);
            }

            if (options.Count == 0)
            {
                options.Add(2);
            }

            var option = options.SelectCombination(1, random).Single();

            if (option == 0)
            {
                result.Add(chooseFrom1[i]);
            }
            else if (option == 1)
            {
                result.Add(chooseFrom2[i]);
            }
            else if (option == 2)
            {
                result.Add(random.NextDouble() < 0.5 ? Genom.NotesMovement.Up
: Genom.NotesMovement.Down);
            }
        }


        return result;
    }

    static List<Genom.RhythmPart[]> CombineTwoRhythmsRandomly(int
targetCount, List<Genom.RhythmPart[]> rhythm1, List<Genom.RhythmPart[]>
rhythm2, Random random)
    {
        var result = new List<Genom.RhythmPart[]>();

        var chooseFrom = new List<Genom.RhythmPart[]>();

        chooseFrom.AddRange(rhythm1);
        chooseFrom.AddRange(rhythm2);

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

```
            result.Add(chooseFrom.SelectCombination(1, random).Single());
        }

        return result;
    }


    public static Genom CombineGenoms(Genom genom1, Genom genom2, Random random)
    {
        var result = new Genom();

        result.BPM = random.Next(Math.Min(genom1.BPM, genom2.BPM), Math.Max(genom1.BPM, genom2.BPM) + 1);
        result.Scale = random.NextDouble() < 0.5 ? genom1.Scale : genom2.Scale;

        result.RootNote = random.NextDouble() < 0.5 ? genom1.RootNote : genom2.RootNote;

        result.ChordsProgression = CombineTwoProgressionsRandomly(genom1.ChordsProgression, genom2.ChordsProgression, random);

        result.MeasuresRhythm = CombineTwoRhythmsRandomly(result.ChordsProgression.Length, genom1.MeasuresRhythm, genom2.MeasuresRhythm, random);

        result.NotesMovementPerMeasures = CombineTwoNoteMovementsRandomly(result.ChordsProgression.Length, genom1.NotesMovementPerMeasures, genom2.NotesMovementPerMeasures, random);

        return result;
    }

    public static Genom AddRandomMutations(Genom initialGenom, float mutationInGeneProbability, Random random)
    {
        var randomGenom = GenerateRandomGenom(random);

        var result = new Genom();

        if (random.NextDouble() < mutationInGeneProbability)
        {
            result.BPM = randomGenom.BPM;
```

```
            }
            else
            {
               result.BPM = initialGenom.BPM;
            }

            if (random.NextDouble() < mutationInGeneProbability)
            {
               result.Scale = randomGenom.Scale;
            }
            else
            {
               result.Scale = initialGenom.Scale;
            }

            if (random.NextDouble() < mutationInGeneProbability)
            {
               result.RootNote = randomGenom.RootNote;
            }
            else
            {
               result.RootNote = initialGenom.RootNote;
            }

            if (random.NextDouble() < mutationInGeneProbability)
            {
               result.ChordsProgression =
CombineTwoProgressionsRandomly(initialGenom.ChordsProgression,
randomGenom.ChordsProgression, random);
               result.MeasuresRhythm =
CombineTwoRhythmsRandomly(result.ChordsProgression.Length,
initialGenom.MeasuresRhythm, randomGenom.MeasuresRhythm, random);
            }
            else
            {
               result.ChordsProgression = initialGenom.ChordsProgression;
               result.MeasuresRhythm = initialGenom.MeasuresRhythm;
            }

            if (random.NextDouble() < mutationInGeneProbability)
            {
               result.NotesMovementPerMeasures =
CombineTwoNoteMovementsRandomly(result.ChordsProgression.Length,
initialGenom.NotesMovementPerMeasures,
randomGenom.NotesMovementPerMeasures, random);
```

```
            }
            else
            {
                result.NotesMovementPerMeasures                                    =
initialGenom.NotesMovementPerMeasures;
            }

            return result;
        }
    }
}
```

В.10 Текст файлу Genom.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Melanchall.DryWetMidi.Interaction;
using Melanchall.DryWetMidi.MusicTheory;

namespace MidiGeneratorUI.Algo
{
    /// <summary>
    /// Contains all the necessary information to generate a MIDI file
    /// </summary>
    public class Genom
    {
        public enum NotesMovement
        {
            Up, Down
        }
        public class RhythmPart
        {
            public bool IsRest { get; set; } = false;
            public MusicalTimeSpan Length { get; set; } = MusicalTimeSpan.Quarter;
        }

        public class ChordDescriptor
        {
            public string Number { get; set; } = "I";
            public string Characteristic { get; set; } = "";
            public string Name => $"{Number}{Characteristic}";
```

```
        }

        public IEnumerable<Interval> Scale { get; set; } = ScaleIntervals.Major;
        public NoteName RootNote { get; set; } = NoteName.C;
        public int BPM { get; set; } = 120;
        public ChordDescriptor[] ChordsProgression { get; set; } = [];
        public List<RhythmPart[]> MeasuresRhythm { get; set; } = [];
        public List<NotesMovement> NotesMovementPerMeasures { get; set; } = [];
    }
}
```

В.11 Текст файлу GeneticAlgo.cs

```
using Melanchall.DryWetMidi.Core;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace MidiGeneratorUI.Algo
{
    /// <summary>
    /// Represents generial algorithm for generating MIDI sequences
    /// </summary>
    public class GeneticAlgo
    {
        int generationNumber = 0;
        readonly int targetCount;
        float randomInjectionProbability;
        float mutationsProbability;

        const float minRandomInjectionProbability = 0.01f,
maxRandomInjectionProbability = 0.5f;
        const float minMutationsProbability = 0.01f, maxMutationsProbability = 0.5f;

        double lastEngagement = 0.5f;

        public int GenerationNumber => generationNumber;
        public int TargetCount => targetCount;

        public double CalculatedEngagement => lastEngagement;

        Generation currentGeneration;
```

```
    public Generation CurrentGeneration => currentGeneration;

    public GeneticAlgo(int targetCount, float randomInjectionProbability, float mutationsProbability)
    {
        this.targetCount = targetCount;
        this.randomInjectionProbability = randomInjectionProbability;
        this.mutationsProbability = mutationsProbability;
        currentGeneration = Generation.GenerateRandom(targetCount);
    }

    public void GenerateNewGeneration(List<float> pointsForTracksInCurrentGeneration)
    {
        UpdateRatesBasedOnEngagement(pointsForTracksInCurrentGeneration);

        generationNumber++;

        currentGeneration = currentGeneration.CreateNew(targetCount, randomInjectionProbability, mutationsProbability, pointsForTracksInCurrentGeneration);
    }

    public List<float> GetDigitalRepresentationByIndex(int index)
    {
        var genom = currentGeneration.Genoms[index];
        var digitalRepresentation = MusicGenerator.GenerateDigitalRepresentation(genom);

        return digitalRepresentation;
    }

    public MidiFile GetFileByIndex(int index)
    {
        var genom = currentGeneration.Genoms[index];
        var midiFile = MusicGenerator.GenerateTrack(genom);

        return midiFile;
    }
    void UpdateRatesBasedOnEngagement(List<float> ratings)
    {
        double avg = ratings.Average();
        double variance = ratings.Select(r => Math.Pow(r - avg, 2)).Average();
        double stdDev = Math.Sqrt(variance);
```

```
            double normalizedStdDev = stdDev / 0.5;

            double maxValue = ratings.Max();

            double engagement = normalizedStdDev * maxValue;

            lastEngagement = engagement = Math.Max(0.0, Math.Min(1.0, engagement));

            var reverseEngagement = 1.0 - engagement;

            randomInjectionProbability = (float)(minRandomInjectionProbability + (maxRandomInjectionProbability - minRandomInjectionProbability) * reverseEngagement);
            mutationsProbability = (float)(minMutationsProbability + (maxMutationsProbability - minMutationsProbability) * reverseEngagement);
        }
    }
}
```

В.12 Текст файлу Generation.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace MidiGeneratorUI.Algo
{
    /// <summary>
    /// Represents one generation of genoms
    /// </summary>
    public class Generation
    {
        readonly List<Genom> genoms;

        public List<Genom> Genoms => genoms;

        public Generation(List<Genom> genoms)
        {
            this.genoms = genoms;
        }
```

```
public Generation CreateNew(int targetCount, float randomInjectionProbability, float mutationsProbability, List<float> selectionAsParentProbability)
{
  ArgumentOutOfRangeException.ThrowIfLessThan(targetCount, 2, nameof(targetCount));

  var result = new List<Genom>(targetCount);
  var random = new Random();

  // Precompute cumulative probabilities for parent selection
  var cumulativeProbabilities = new List<float>(selectionAsParentProbability.Count);
  float sum = 0f;
  foreach (var prob in selectionAsParentProbability)
  {
    sum += prob;
    cumulativeProbabilities.Add(sum);
  }

  int GetRandomParentIndex()
  {
    float value = (float)(random.NextDouble() * sum);
    for (int i = 0; i < cumulativeProbabilities.Count; i++)
    {
      if (value <= cumulativeProbabilities[i])
        return i;
    }
    return cumulativeProbabilities.Count - 1;
  }

  for (int i = 0; i < targetCount; i++)
  {
    Genom newGenom;

    if (random.NextDouble() < randomInjectionProbability)
    {
      // Create a completely random genom
      newGenom = GenomUtils.GenerateRandomGenom(random);
    }
    else
    {
      // Generate from two parents
      var parentIndex1 = GetRandomParentIndex();
```

```
                var parentIndex2 = GetRandomParentIndex();

                var parent1 = genoms[parentIndex1];
                var parent2 = genoms[parentIndex2];
                newGenom = GenomUtils.CombineGenoms(parent1, parent2,
random);

                // Apply mutations

                newGenom = GenomUtils.AddRandomMutations(newGenom,
mutationsProbability, random);

            }

            result.Add(newGenom);
        }

        return new Generation(result);
    }

    public static Generation GenerateRandom(int targetCount)
    {
        var random = new Random();
        var result = new List<Genom>();
        for (int i = 0; i < targetCount; i++)
        {
            result.Add(GenomUtils.GenerateRandomGenom(random));
        }
        return new Generation(result);
    }
  }
}
```

ДОДАТОК Д

Слайди презентації

Національний університет «Запорізька політехніка»
Кафедра програмних засобів

Розробка та дослідження програмного забезпечення для генерації музики з використанням інтерактивного генетичного алгоритму

Виконав:
студент групи КНТ-214м Максим ЧОРНОБУК

Керівник:
д.т.н., професор Андрій ОЛІЙНИК

Рисунок Д.1 – Слайд № 1

Мета, об’єкт, предмет дослідження

Об’єкт дослідження – процеси обчислень, пов'язані з генерацією музичних композицій на основі відгуків користувача за допомогою інтерактивного генетичного алгоритму.

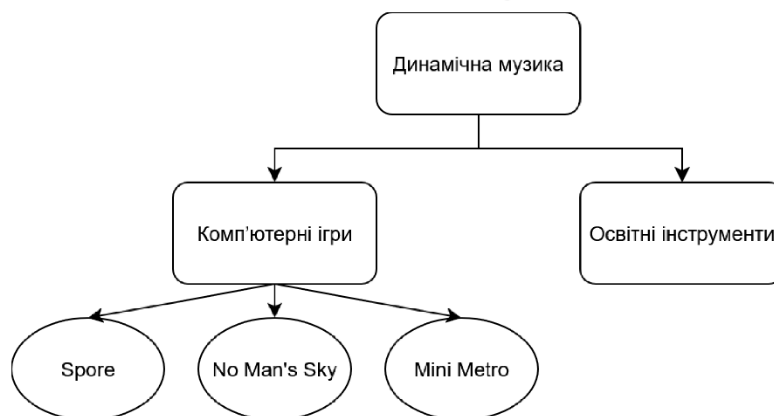
Предмет дослідження – методи генерації музичних композицій на основі відгуків користувача.

Мета роботи – дослідження та програмна реалізація методу інтерактивної генерації музики за допомогою інтерактивного генетичного алгоритму.

2

Рисунок Д.2 – Слайд № 2

Динамічна музика



Приклади практичного використання динамічної музики

3

Рисунок Д.3 – Слайд № 3

Порівняння методів генерації процедурної музики

	Витрати ресурсів	Використання без спеціальних навичок	Точне налаштування	Творче різноманіття композицій
Процедурні системи	Низькі	Так	Так	Низьке
Системи на основі нейронних мереж	Дуже високі	Так	Залежить від реалізації	Дуже високе
Еволюційні системи	Низькі	Ні	Так	Середнє
Інтерактивні еволюційні системи (IGA)	Низькі	Так	Так	Середнє

4

Рисунок Д.4 – Слайд № 4

Формальна постановка задачі

$$s = (BPM, \{e_1, e_2, \dots, e_N\})$$

$$e_i = (p_i, d_i) \text{ або } e_i = (\emptyset, d_i)$$

$$p_i \in P, d_i \in D$$

Формальне представлення монофонічних музичних композицій

5

Рисунок Д.5 – Слайд № 5

Формальна постановка задачі

$$S_{n+1} = M(S_n, Y_n)$$

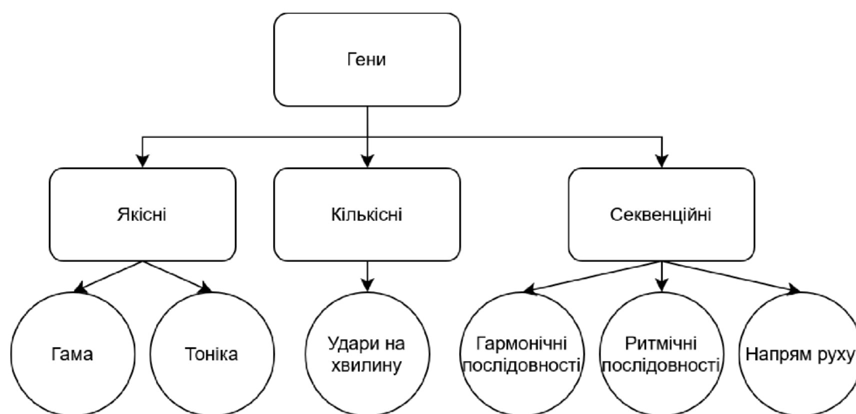
$$Y_n = \{y_1, y_2, \dots, y_x\}$$

Використання функції *M* для генерації нового покоління *S* з урахуванням відгуків користувача *Y*

6

Рисунок Д.6 – Слайд № 6

Типи генів у геномі композиції



Типи використаних генів та інформація, яку вони кодують

7

Рисунок Д.7 – Слайд № 7

Реалізація генетичного алгоритму

$$P_i = \frac{y_i}{\sum_{j=0}^{n-1} y_j}$$

Формула розрахунку вірогідності вибору особи *P* у якості батьківської під час кросинговеру

8

Рисунок Д.8 – Слайд № 8

Проблема втоми користувача

$$\delta = \frac{\sigma}{0.5} * r_{max}$$

Формула розрахунку рівня зацікавленості користувача *δ*

9

Рисунок Д.9 – Слайд № 9

Реалізація генетичного алгоритму

$$\beta = \beta_{min} + (\beta_{max} - \beta_{min}) * (1 - \delta)$$

$$\alpha = \alpha_{min} + (\alpha_{max} - \alpha_{min}) * (1 - \delta)$$

Формули розрахунку вірогідності мутації *β* та ін’єкції *α* у залежності від рівня зацікавленості користувача *δ*

10

Рисунок Д.10 – Слайд № 10

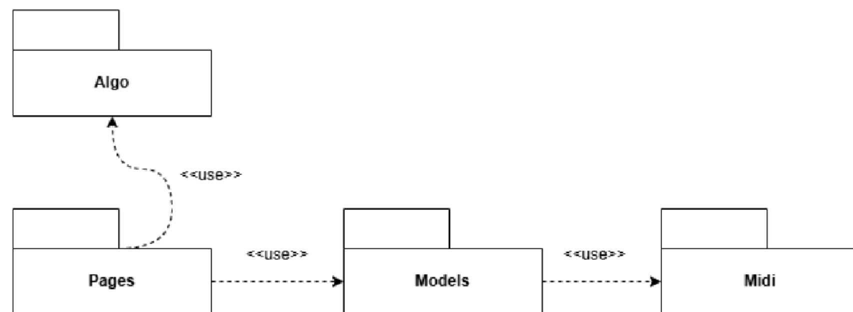
Реалізація генетичного алгоритму

UML-діаграма схеми роботи алгоритму

11

Рисунок Д.11 – Слайд № 11

Схема розробленого програмного продукту



UML-діаграма просторів імен проєкту

12

Рисунок Д.12 – Слайд № 12

Схема розробленого програмного продукту

UML-діаграма основних класів проєкту

13

Рисунок Д.13 – Слайд № 13

Приклади результатів роботи системи

Композиція, згенерована в першому поколінні

Композиція, згенерована в п’ятому поколінні

14

Рисунок Д.14 – Слайд № 14

Порівняння з іншими системами

	Тип музики	Спосіб кодування композицій	Сурогатна фітнес-функція	Динамічне відслідковування зацікавленості користувача	Лише монофонічна музика
MIDI Music Generator	Довільні компактні композиції	Складний геном, гени трьох типів	Ні	Так	Так
GenJam	Джазові соло	Дворівнева ієрархія, обмежена кількість 4-бітних подій на такт	Ні	Ні	Так
GP-Music	Лише однакова довжина нот	Деревоподібна багаторівнева структура	Так	Ні	Так
DarwinTunes	Компактні зациклені композиції	Деревоподібна багаторівнева структура	Ні	Ні	Так

15

Рисунок Д.15 – Слайд № 15

Висновок

- Розглянуто способи генерації музики на основі інтерактивних генетичних алгоритмів
- Розроблено модифікацію інтерактивного генетичного алгоритма з динамічним відслідковуванням зацікавленості користувача для генерації музики
- Проведено порівняння розробленої системи з аналогами з літератури
- Розроблено Desktop застосунок MIDI Music Generator для простої взаємодії користувачів із розробленою системою

16

Рисунок Д.16 – Слайд № 16