НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет інформатики та обчислювальної техніки

Кафедра інформаційних систем та технологій

До захисту допущено:

Завідувач кафедри

_____ Олександр РОЛІК

«___»_____20__ р.

Дипломний проєкт

на здобуття ступеня бакалавра
за освітньо-професійною програмою «Інтегровані інформаційні системи»
спеціальності 126 «Інформаційні системи та технології»
на тему: «Автоматизована система налаштування аудіо-апаратури під приміщення»

Виконав:
студент IV курсу, групи ІА-91
Овчинніков Сергій Дмитрович _____

Керівник:
асистент кафедри ІСТ
Шинкевич Микола Костянтинович _____

Рецензент:
доцент каф ОТ, к.т.н.
Верба Олександр Андрійович _____

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ – 2023 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки
Кафедра інформаційних систем та технологій

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 126 «Інформаційні системи та технології»

Освітньо-професійна програма «Інформаційні системи та технології»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Олександр РОЛІК
«___»_____20__ р.

ЗАВДАННЯ
на дипломний проєкт студенту
Овчиннікову Сергію Дмитровичу

1. Тема проєкту «Автоматизована система налаштування аудіо-апаратури під приміщення», керівник проєкту Шинкевич Микола Костянтинович, асистент кафедри ІСТ, затверджені наказом по університету від 31 травня 2023 р. №2101-с

2. Термін подання студентом проєкту: 12 червня 2023 року

3. Вихідні дані до проєкту: технічна документація, програмний код мовою Rust та C#

4. Зміст пояснювальної записки: аналіз предметної області, вибір технологій розробки, реалізація системи, тестування розробленого застосунку.

5. Перелік графічного матеріалу (із зазначенням обов'язкових креслеників, плакатів, презентацій тощо): структура інтеграції системи, приклад роботи ШПФ з 8 змінними, структура модулів системи, структура аналізатора як тренувальника нейромережі.

6. Дата видачі завдання: 1 березня 2023 року

Календарний план

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
1.	Аналіз існуючих рішень	17.04.23 – 21.04.23	
2.	Вибір технологій розробки	24.04.23 – 28.04.23	
3.	Розробка структури системи	01.05.23 – 05.05.23	
4.	Розробка архітектури системи	08.05.23 – 12.05.23	
5.	Розробка користувацького інтерфейсу	15.05.23 – 19.05.23	
6.	Тестування застосунку	21.05.23 – 27.05.23	
7.	Оформлення текстової документації	29.05.23 – 09.06.23	

Студент Сергій ОВЧИННІКОВ

Керівник Микола ШИНКЕВИЧ

АНОТАЦІЯ

Овчинніков С. Д. Автоматизована система налаштування аудіо-апаратури під приміщення. КПІ ім. Ігоря Сікорського, Київ, 2023

Проєкт містить 60 сторінок, 38 рисунків, 1 таблицю, 9 формул, посилання на 21 джерело.

Ключові слова: обробка звуку, аналіз звуку, аудіо-апаратура, АЧХ, аудіо-фільтр.

Об’єктом розробки є автоматизована система налаштування аудіо-апаратури під приміщення для покращення звучання.

Мета розробки – створення системи, яка надає користувачам можливості для легкого налаштування аудіо-апаратури.

В результаті виконання дипломного проєкта поставлена мета була досягнута, система була спроєктована та реалізована. Автоматизована система аналізує звучання в різних точках приміщення, створює оптимальні налаштування та виконує обробку звуку. Ця система може бути використана для автоматизації роботи фахівця сфери аудіо-інженерії.

ABSTRACT

Ovchynnikov S. Automated system for setting up audio equipment for the room. Igor Sikorsky Kyiv Polytechnic Institute, Kyiv, 2023

The project contains 60 pages, 38 images, 1 table, 8 formulas, references to 21 source.

Keywords: sound processing, sound analysis, audio equipment, frequency response, audio filter.

The object of development is an automated system for setting up audio equipment for the room to improve the sound.

The goal of the development is to create a system that provides users with the ability to easily configure audio equipment.

As a result of the graduation project, the goal was achieved, the system was designed and implemented. An automated system analyzes the sound in different points of the room, creates optimal settings and performs sound processing. This system can be used to automate the work of a specialist in the field of audio engineering.

Номер рядка	Формат	Позначення	Найменування	Кільк. аркушів	Номер екзем.	Примітка
1			Документація загальна			
2						
3			Знову розроблена			
4						
5	А4	ІА91.220БАК.004 ПЗ	Пояснювальна записка	60		
6						
7	А3	ІА91.220БАК.004 Д1	Автоматизована система	1		
8			налаштування аудіо-			
9			апаратури під приміщення.			
10			Структура інтеграції системи			
11						
12	А3	ІА91.220БАК.004 Д2	Автоматизована система	1		
13			налаштування аудіо-			
14			апаратури під приміщення.			
15			Приклад роботи ШПФ			
16			з 8 змінними			
17						
18	А3	ІА91.220БАК.004 Д3	Автоматизована система	1		
19			налаштування аудіо-			
20			апаратури під приміщення.			
21			Структура модулів			
22			системи			
23						
24	А3	ІА91.220БАК.004 Д4	Автоматизована система	1		
25			налаштування аудіо-			
26			апаратури під приміщення.			
27			Структура аналізатора як			
28			тренувальника нейромережі			

Зм.	Аркуш	№ докум.	Підпис	Дата	ІА91.220БАК.004 ТП		
Розроб.		Овчинніков С.Д.			Автоматизована система налаштування аудіо-апаратури під приміщення. Відомість проєкту	Літ.	Аркуш 1 / Аркушів 1
Керівн.		Шинкевич М.К.				Т	
Затв.						КПІ ім. Ігоря Сікорського Група ІА-91	

Пояснювальна записка
до дипломного проєкту
на тему: «Автоматизована система налаштування аудіо-апаратури під приміщення»

Київ – 2023 року

ЗМІСТ

ВСТУП

Світ музики завжди приваблював багато людей незалежно від їхнього місця проживання. Музиканти з різних куточків світу присвячують своє життя музичній професії з мрією стати визнаними в своєму галузі.

З історії відомо, що професія музиканта не завжди вважалася успішною або прибутковою. Україна також має свій внесок у цю тему. Відомо, що у минулому музикою займалися переважно люди з вадами зору, які мали обмежені можливості заробляти на життя іншими способами. Це свідчить про те, що відданість музиці і талант можуть стати важливими факторами для багатьох музикантів незалежно від їхніх обставин.

Навіть у сучасному світі багато сімей відправляють своїх дітей до музичних шкіл і академій, але не завжди сприймають музику як потенційну професію. Це пояснюється тим, що професія музиканта є вкрай складною і вимагає багато часу, зусиль і посвяченості. Шлях до визнання та успіху у музиці може бути довгим і важким.

Сучасні музичні колективи складаються не лише з виконавців, але й з операторів, аудіо-інженерів та операторів світлових ефектів. Ці професіонали відіграють важливу роль у створенні вражаючих виступів та концертів. Вони відповідають за звукове оформлення, якість запису та передачу емоцій через музику. Виконавці на сцені є лише видимою частиною великої команди, яка працює разом, щоб створити незабутнє музичне враження для аудиторії.

Тому було створено проект, який має на меті полегшити життя починаючим виконавцям на сцені. Цей проект надає можливість кожному талановитому виконавцю проявити себе без необхідності глибокого вивчення аудіо-обробки та технічних аспектів. Шляхом використання новітніх технологій і розробок, цей проект надає інструменти і можливості для створення професійного звучання без значних витрат часу і коштів.

Результати цього проекту також можуть знайти застосування в різних місцях, наприклад, у локальних закладах харчування або невеликих сценах, де

немає можливості або фінансових ресурсів для наймання аудіо-інженера. Це дозволить виконавцям і закладам створювати приємну та професійну атмосферу для своїх гостей і аудиторії, що впливає на загальний досвід і задоволення від музичних виступів.

У сучасному світі, на фоні широкої глобалізації та загального доступу до інформації, культура переживає справжню революцію. Завдяки передовим технологіям та програмним розробкам, виконавцям, незалежно від їхнього професійного рівня, відкриваються безліч можливостей для творчого самовираження та спілкування з аудиторією по всьому світу.

Те, що раніше вимагало великого зусилля та навичок в галузі налаштування техніки, тепер може бути виконане з легкістю завдяки програмним рішенням. Важливим є сам творчий процес, а не витрачені години на налаштування обладнання. Завдяки автоматизованим інструментам, музиканти, актори, письменники та інші творчі особистості можуть зосередитися на своїх ідеях, натхненні та здібностях.

Загалом, людство вже вступило до ери, коли технології стають незамінними помічниками для творців у всіх галузях. Це дає можливість кожній талановитій особистості розкрити свій потенціал та поділитися своїми творчими витворами зі світом, незалежно від географічних обмежень. Таким чином, використання програм та автоматизованих інструментів у творчому процесі стає ключем до успіху і дозволяє кожному зосередитися на найважливішому - на самій творчості.

1 АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ

Перед початком розробки системи важливо провести докладне ознайомлення та аналіз існуючих рішень, які вже існують у даній області. Цей етап дозволяє оцінити поточний стан технологій та інновацій, що застосовуються в схожих проектах.

Після збору та аналізу інформації про існуючі рішення, можна визначити переваги та недоліки кожного з них, а також виділити прогалини, які можна заповнити у своєму проекті. Такий підхід дозволить нам створити більш ефективну, конкурентоспроможну та інноваційну систему.

1.1 Огляд існуючих систем

Першим, що приходить на згадку при спробах вирішити проблему поганого звучання, це редактори-секвенсори для написання музики. Усі вони мають не лише режим написання музики, а й живого відтворення.

1.1.1 FL Studio

FL Studio, раніше відомий як FruityLoops, є цифровим аудіо редактором та секвенсором, розробленим компанією Image-Line Software. Вперше випущений у 1997 році, FL Studio став популярним серед музикантів, продюсерів та діджеїв, як початківців, так і професіоналів. На рисунку 1.1 можна побачити сторінку вебсайту FruityLoops.

FL Studio надає широкі можливості для створення, запису та редагування музики. Він включає в себе різноманітні інструменти, ефекти та засоби для створення різних жанрів музики, включаючи електронну, хіп-хоп, поп, рок та інші. FL Studio пропонує унікальний робочий процес, відомий як "петельний запис" (loop-based recording), який дозволяє користувачам швидко створювати й редагувати музику шляхом зсуву та змішування петель звуків.

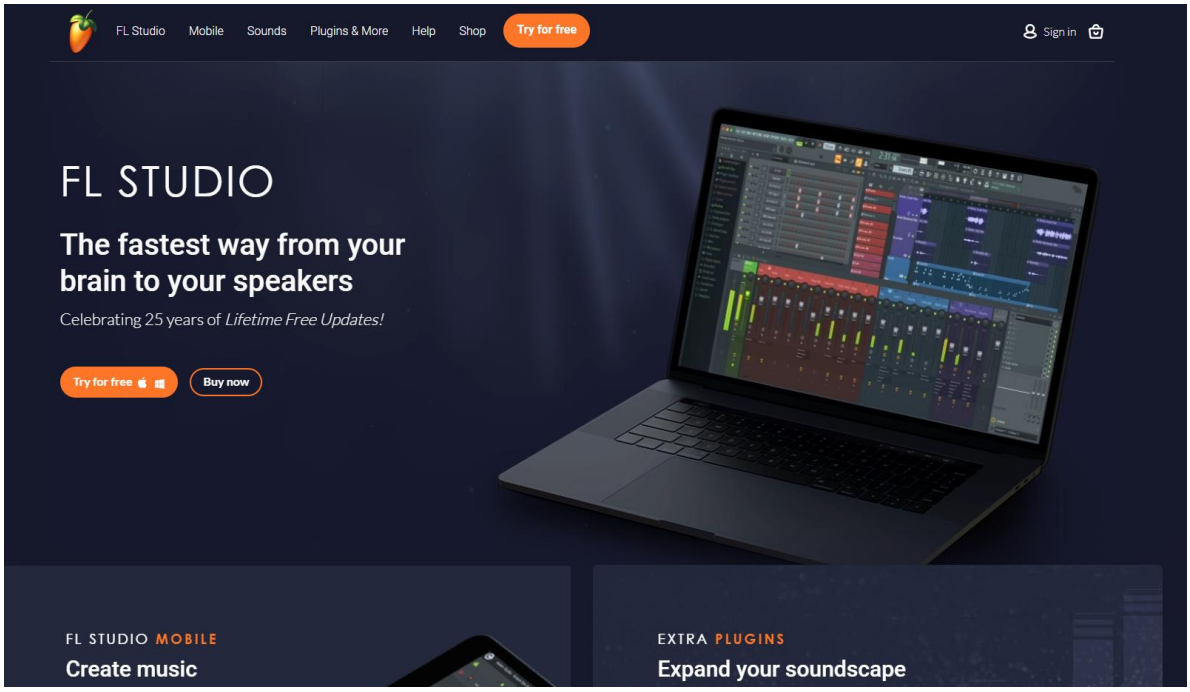

Рисунок 1.1 – Сторінка вебсайту FL Studio

Одним з ключових елементів FL Studio є його інтуїтивно зрозумілий інтерфейс, який дозволяє користувачам працювати зі звуками, міді-данними, ефектами та іншими аспектами музичного проекту. Інтерфейс FL Studio зображено на рисунку 1.2

Рисунок 1.2 – Інтерфейс FL Studio

Інтерфейс FL Studio зображено на рисунку 1.2. FL Studio підтримує плагіни VST і підтримує підключення зовнішніх пристроїв MIDI, таких як клавіатури та контролери.

Програма також має вбудовані засоби для мастерингу (завершення та оптимізації звуку) та міксування (збалансування та обробка окремих звукових доріжок). Вона надає широкі можливості для обробки звуків, включаючи різні ефекти, звукові фільтри, еквалайзери, компресори та інші інструменти.

1.1.2 Ableton Live

Ableton Live - це професійне програмне забезпечення для створення та виконання музики в режимі реального часу. Воно спеціалізується на електронній музиці та є популярним інструментом серед музикантів, діджеїв та продюсерів. На рисунку 1.3 зображено сторінку вебсайту Ableton Live.

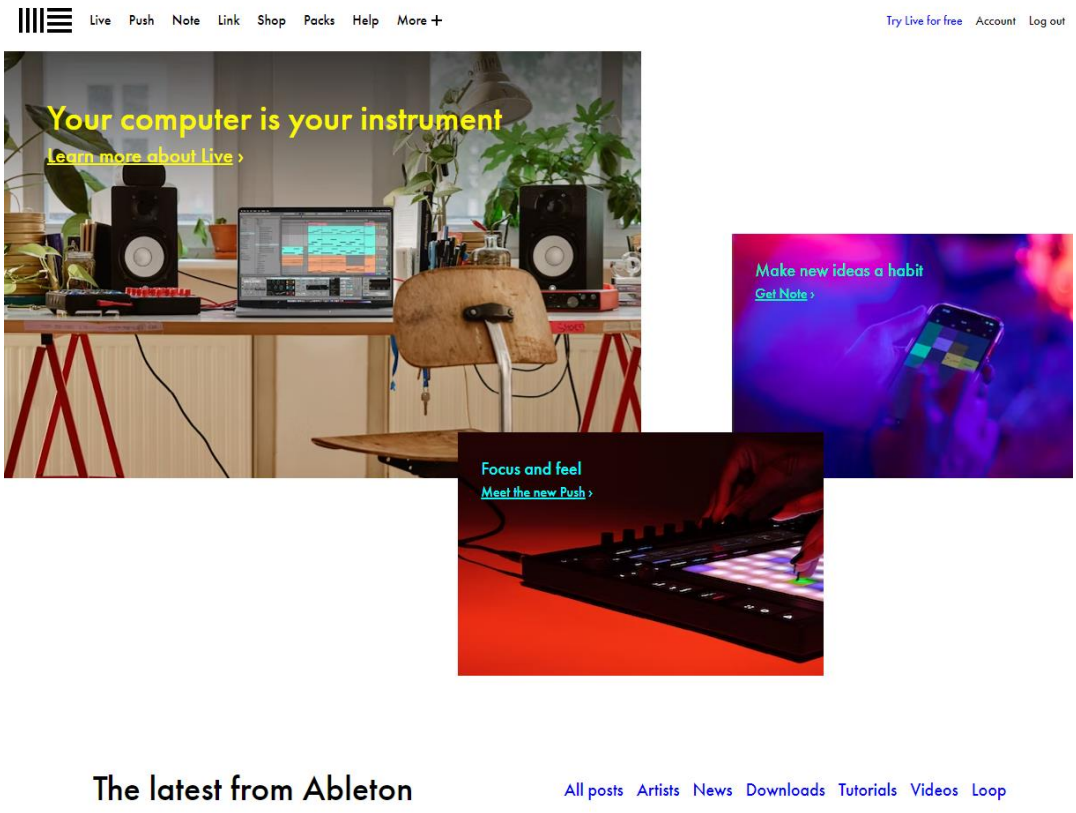

Рисунок 1.3 – Сторінка вебсайту Ableton Live

Ableton Live надає гнучкість та потужність для композиції, аранжування, запису, змішування та мастерингу музики. Воно має інтуїтивний інтерфейс, що дозволяє легко маніпулювати звуками та зразками, використовувати вбудовані інструменти, ефекти та засоби для створення музики у реальному часі.

Особлива риса Ableton Live - це його "Session View" (Вид сесії), який дозволяє організовувати та виконувати музичні ідеї у вигляді живих лупів та запускати їх на льоту. Він також має "Arrangement View" (Вид аранжування) для детального розміщення та редагування композицій. На рисунку 1.4 зображено вид аранжування Ableton Live.

Рисунок 1.4 – Інтерфейс Ableton Live

Ableton Live використовується як професіоналами, так і початківцями в музичній індустрії для створення різних жанрів музики, від електроніки та хіп-хопу до популярної музики та саундтреків. Він є потужним інструментом для творчості та вираження музичних ідей.

1.1.3 Steinberg Cubase

Steinberg Cubase є одним з провідних музичних програмних пакетів, створених компанією Steinberg Media Technologies. Це професійний аудіо- та MIDI-секвенсер, призначений для запису, редагування та змішування музики. Cubase використовується в студіях звукозапису, а також домашніми продюсерами та музикантами для створення музики різних жанрів. На рисунку 1.5 зображено сторінку вебсайту Steinberg Cubase.

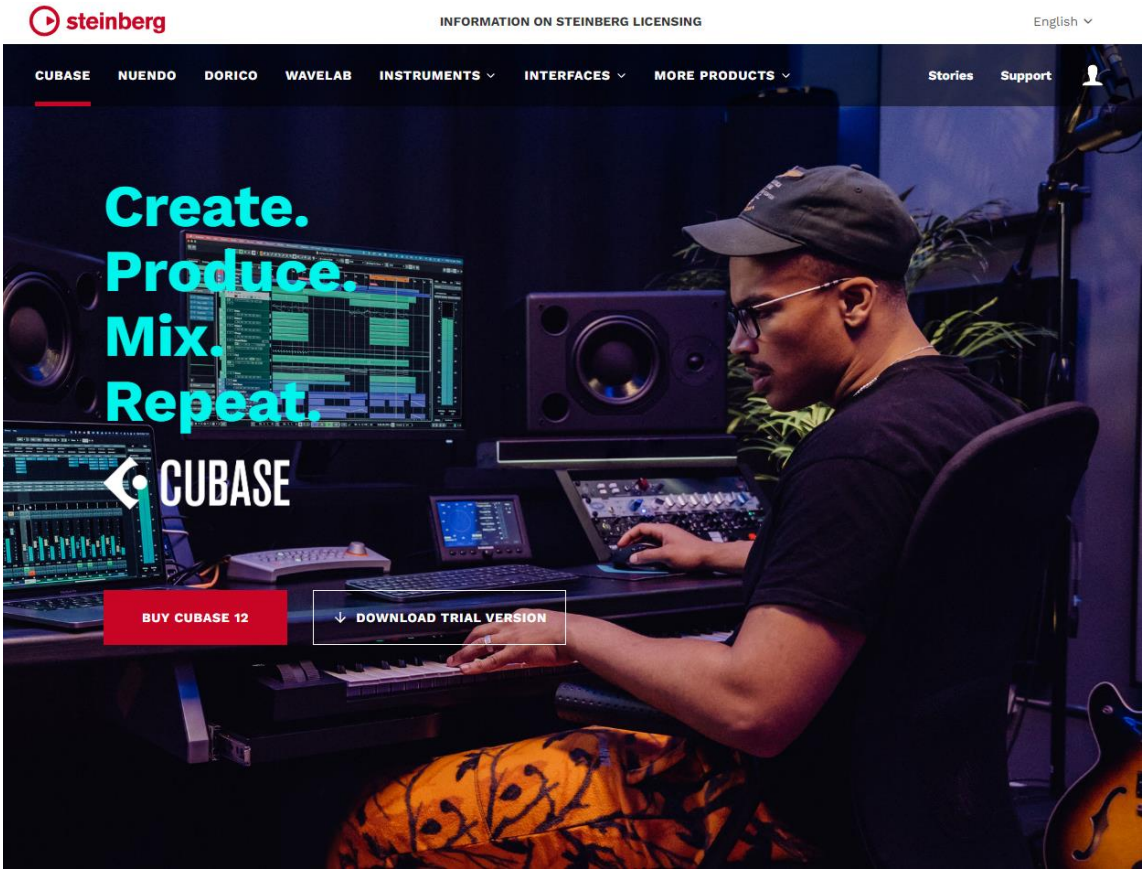

Рисунок 1.5 – Сторінка вебсайту Steinberg Cubase

Програма надає розширені функції для запису аудіо- та MIDI-доріжок, обробки звуку, включаючи ефекти та плагіни, секвенціювання, змішування і мастеринг. Вона має інтуїтивний інтерфейс користувача, багатофункціональний мікшер, а також підтримку широкого спектра аудіоформатів.

Cubase також має додаткові інструменти, такі як вбудований синтезатор, семплер, віртуальні інструменти та MIDI-редактори, які дозволяють музикантам

створювати комплексну музику без додаткового обладнання або програм. На рисунку 1.6 зображено зазначені інструменти.

Рисунок 1.6 – Інтерфейс Steinberg Cubase

Загалом, Steinberg Cubase визнаний як потужний інструмент для професійної музичної продукції і використовується в галузі звукозапису, створення музики та звукового дизайну.

1.2 Порівняльний аналіз

FL Studio (раніше відомий як FruityLoops) є популярною програмою, особливо серед продюсерів електронної музики. Вона має інтуїтивний інтерфейс та широкий набір інструментів, ефектів та плагінів. FL Studio має зручну аранжувальну секцію та потужний секвенсор, що дозволяє створювати складні композиції. Вона також підтримує плагіни третіх сторін, що робить її гнучким

інструментом для створення музики. Для новачків може знадобитися трохи часу, щоб оволодіти всіма функціями та особливостями FL Studio.

Ableton Live є популярним серед музикантів, діджеїв і виконавців живих виступів. Вона має унікальну інтерфейсну концепцію з двома основними режимами роботи: Arrangement і Session. Режим Session дозволяє створювати та відтворювати музику в реальному часі, що робить її ідеальним вибором для живих виступів. Ableton Live також має широкий вибір звукових ефектів і інструментів, а також можливості для семплювання та зациклення звуків. Інтерфейс Ableton Live може вимагати деякого знайомства, але вона надає потужні можливості для обробки звуку у живих виступах.

Steinberg Cubase є іншою популярною програмою для студійного запису та обробки звуку. Вона має багатий набір функцій, включаючи широкий вибір інструментів, ефектів та плагінів. Cubase відомий своєю стабільністю та простотою використання, а його інтерфейс орієнтований на роботу з MIDI. Він також підтримує широкий вибір аудіоформатів та має потужні можливості з мікшування та мастерингу. Хоча Cubase може бути дружелюбнішим для початківців, він все ще має деяку крутість, що вимагає вивчення для повного освоєння всіх його можливостей.

Загалом, FL Studio, Ableton Live і Steinberg Cubase є потужними засобами для обробки звуку на живих виступах. Вибір між ними залежить від особистих вподобань, стилю музики та індивідуальних потреб. Для новачків може знадобитися час і практика, щоб оволодіти будь-яким із цих засобів повністю, але з відповідним навчанням та експериментуванням вони можуть стати потужними інструментами для творчості у живих виступах.

Висновки до розділу 1

Порівнюючи FL Studio, Ableton Live та Steinberg Cubase як засоби для обробки звуку на живих виступах, можна зазначити, що всі три програми є потужними і популярними серед професіоналів у музичній індустрії. Проте, вони

досить складні для новачків, оскільки мають комплексні інтерфейси та різноманітні інструменти.

При порівнянні FL Studio, Ableton Live та Steinberg Cubase як засобів для обробки звуку на живих виступах, варто відзначити, що усі ці програми належать до потужних і широко використовуваних в професійному музичному середовищі. Їх висока популярність серед фахівців музичної індустрії пояснюється їхнім багатофункціональним набором інструментів та можливостями, які вони пропонують.

Проте, варто зазначити, що ці програми не є найпростішими у використанні, особливо для початківців. Вони мають досить складний інтерфейс, який може викликати певні труднощі в освоєнні. Крім того, їхній набір інструментів і функцій є досить розгалуженим і великим, що може викликати певну заплутаність у користувачів.

Обрана тема про розробку простого у використанні обробника звуку є надзвичайно актуальною в сучасному музичному середовищі. Хоч FL Studio, Ableton Live та Steinberg Cubase є потужними і популярними програмами для обробки звуку, вони можуть бути складними для новачків через свої комплексні інтерфейси та різноманітні інструменти.

Основна мета розробки простого у використанні обробника звуку полягає в створенні програмного забезпечення, яке буде доступним і зручним для початківців у музичній продукції. Воно має надати користувачам простий і зрозумілий інтерфейс, що дозволить легко навчитися його використовувати та освоїти основні функції обробки звуку.

Таким чином, обрана тема розробки простого у використанні обробника звуку є актуальною та краще задовольняє мету, порівняно зі складними програмами, які застосовуються професіоналами в музичній індустрії. Вона спрощує процес обробки звуку для новачків та надає їм зручні інструменти для досягнення високої якості звучання своєї музики.

2 ВИБІР ТЕХНОЛОГІЙ РОЗРОБКИ

В даній роботі розроблено систему автоматизованого налаштування апаратури під приміщення.

Система являє собою комбінацію аналізатора аудіо-сигналів та застосовника аудіо-сигналів.

2.1 Вибір технологій розробки

Розробка системи розподіляється на підсистему аналізу-обробки звуку та підсистему віртуального середовища з перешкодами.

2.1.1 Вибір технологій аналізатора-обробника звуку

Обробка звуку потребує великої обчислювальної потужності. Через це вибір технологій слід робити з огляду на швидкодію.

C++ є потужною мовою програмування, яка використовується для створення систем обробки звукових сигналів завдяки своїм особливостям і можливостям. Ось деякі аспекти використання C++ для розробки таких систем, а також проблеми, з якими можна стикнутися при роботі з цією мовою.

Переваги використання C++ для систем обробки звукових сигналів:

– продуктивність: C++ надає високий рівень продуктивності, що важливо для обробки великого обсягу звукових даних в реальному часі. Мова має низькорівневі можливості, які дозволяють ефективно працювати з пам'яттю і оптимізувати обчислення;

– багатопоточність: Звукова обробка часто вимагає паралельної обробки даних для досягнення високої продуктивності. C++ надає потужні засоби для роботи з потоками і багатопоточністю, такі як стандартна бібліотека потоків (C++11) і бібліотеки, такі як OpenMP і Intel Threading Building Blocks (TBB);

– можливість використання низькорівневих операцій: C++ дозволяє прямий доступ до пам'яті, що дозволяє виконувати операції звукової обробки на низькорівневому рівні. Це корисно для реалізації оптимізованих алгоритмів та роботи зі звуковими даними без зайвого накладу.

Проблеми, пов'язані з використанням C++ для систем обробки звукових сигналів:

– контроль за пам'яттю: C++ надає велику свободу управління пам'яттю, що може бути складним для новачків або недосвідчених розробників. Неправильне управління пам'яттю може призвести до витоку пам'яті або переповнення буфера, що призводить до некоректної роботи програми або вразливостей безпеки;

– потенційні помилки: Використання низькорівневих можливостей C++ може призвести до потенційних помилок, таких як ділення на нуль, некоректні покажчики або неправильна адресація пам'яті. На рисунку 2.1 зображено приклад програми з потенційними помилками. Ці помилки можуть бути складними відслідковувати та виправляти, особливо великих системах з багатою кодовою базою.

```
struct Buffer final {
    std::array<int, 5> cache;
    int* memory;

public:
    Buffer() : cache { } {
        memory = new int(1024);
    }

    const int& get_cache() const {
        // dangling reference
        return cache[0];
    }

    // no destructor, memory leak
};
```

Рисунок 2.1 – Типовий блок коду C++ з критичними помилками, що компілюються

Незважаючи на деякі виклики, C++ залишається потужною мовою програмування для створення систем обробки звукових сигналів, завдяки своїм можливостям щодо продуктивності, багатопоточності та низькорівневого програмування. Професійні розробники, які розуміють мову та дотримуються кращих практик, можуть створити ефективні та стабільні системи обробки звуку на базі C++.

Rust - це мова програмування загального призначення, яка пропонує безпеку пам'яті, конкурентність і високу продуктивність. Вона є ідеальним вибором для створення систем обробки звукових сигналів завдяки своїм особливостям, які забезпечують ефективну роботу з пам'яттю, низьку латентність та можливості паралельного виконання.

– однією з головних переваг Rust для обробки звукових сигналів є безпека пам'яті. Мова використовує систему власних покажчиків та суміщення забезпечують надійне керування пам'яттю і уникнення багатьох типових помилок, пов'язаних з пам'яттю, таких як витоки пам'яті або помилки доступу до пам'яті. На рисунку 2.2 зображено приклад такого коду. Це особливо важливо при обробці великих обсягів звукових даних, оскільки може бути суттєво скоротити час розробки та виявлення помилок;

– другою важливою характеристикою Rust є його здатність до конкурентного програмування. Використання багатопотоковості та асинхронного програмування дозволяє ефективно обробляти багато звукових потоків одночасно, забезпечуючи низьку латентність і підвищену продуктивність. Це особливо важливо для реалізації вимогливих застосунків звукової обробки, таких як аудіоінтерфейси, синтезатори, ефекти та інші алгоритми обробки сигналів реального часу;

– Rust надає розробникам широкий набір інструментів для роботи зі звуком. Існують високоякісні бібліотеки, наприклад, cpal, rodio, hound, які дозволяють зчитувати, записувати, обробляти та відтворювати звукові файли у різних форматах. Використовуючи ці бібліотеки, можна легко реалізувати

функції, такі як зчитування аудіо з мікрофону, відтворення аудіо на динаміки, застосування ефектів до звукових сигналів, аналіз спектру та багато іншого;

– крім того, Rust має активну спільноту, яка розвиває багато корисних інструментів і бібліотек для обробки звуку. Наприклад, бібліотека "aubio" надає реалізацію різних алгоритмів обробки звуку, таких як детекція темпу, виділення звуку із шумом, екстракція акустичних ознак та інші. Ці інструменти роблять Rust привабливим вибором для розробників, які цінують якість, продуктивність та безпеку.

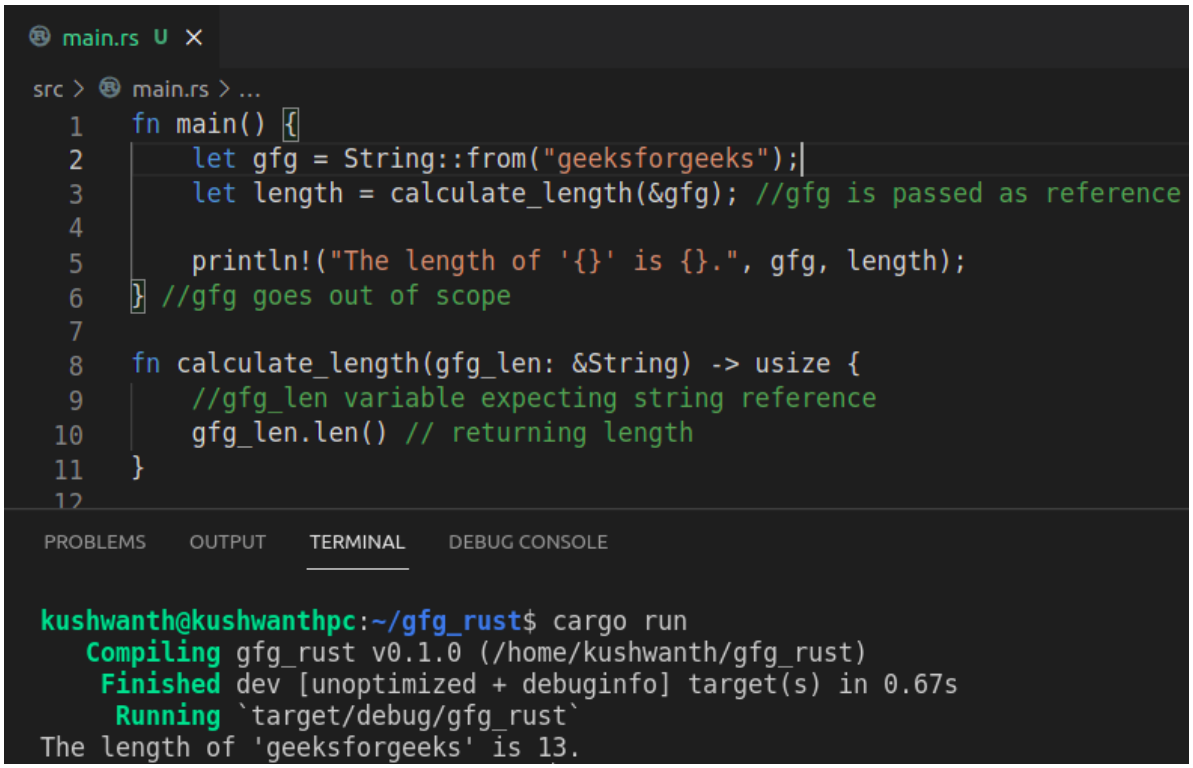

Рисунок 2.2 – Типовий блок коду Rust з безпечною роботою з пам’яттю

Загалом, Rust є потужною мовою програмування для створення систем обробки звукових сигналів. Вона забезпечує безпеку пам'яті, конкурентність, високу продуктивність та широкі можливості для роботи зі звуком, що робить її прекрасним вибором для розробки звукових додатків, які вимагають високої якості та низької латентності.

C# (C-Sharp) є потужною і популярною мовою програмування, яка може бути використана для створення систем обробки звукових сигналів. Вона має ряд функцій і інструментів, які полегшують розробку таких систем.

Одною з переваг використання C# для обробки звукових сигналів є його багатофункціональність і можливості роботи з аудіо. Він має вбудовану підтримку роботи зі звуковими файлами, запису звуку, відтворення аудіо та вироблення звукових ефектів. Крім того, в C# доступні різні бібліотеки, такі як NAudio і WaveStream, які дозволяють робити більш складну обробку звуку, наприклад, зчитування та запис аудіо з мікрофону, обробку сигналу, зміна його параметрів тощо.

Ще одним важливим аспектом C# є його інтеграція з платформою .NET. Для створення систем обробки звуку можна використовувати .NET Framework або .NET Core для створення систем обробки звуку. Це дає доступ до широкого спектру класів і функцій, що спрощують роботу зі звуком.

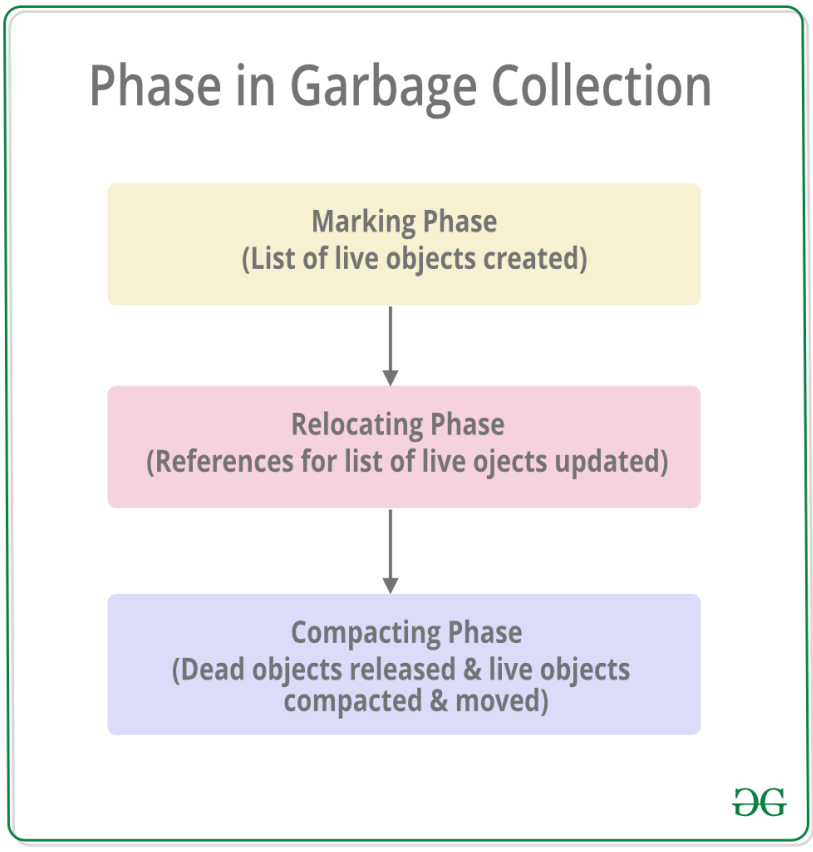

Рисунок 2.3 – Алгоритм роботи збиральника сміття C#

Проте, є деякі обмеження і незручності, які варто враховувати при роботі з C# для обробки звукових сигналів.

Одна з них – це система управління пам'яттю (Garbage Collector, GC). GC автоматично вивільняє пам'ять, використану об'єктами, які більше не використовуються. Однак, це може впливати на продуктивність системи обробки звуку, особливо якщо обробка вимагає низької латентності. На рисунку 2.3 зображено алгоритм роботи збиральника сміття C#. Для досягнення найкращої продуктивності може знадобитися вручне керувати пам'яттю та уникати зайвих алокацій.

Ще одним незручним аспектом C# для роботи з обробкою звуку є потреба в CLR (Common Language Runtime) та CLI (Common Language Infrastructure) для виконання скомпільованих програм. Це означає, що ваша програма буде залежати від .NET Framework або .NET Core для своєї роботи. Це може вплинути на мобільність або вимоги до середовища, в якому виконується програма.

Незважаючи на ці незручності, C# все ще є потужним інструментом для створення систем обробки звукових сигналів завдяки своїй широкій функціональності, багатому набору бібліотек і інтеграції з платформою .NET. При правильному використанні він може допомогти вам створити потужну та ефективну систему обробки звуку.

Java є потужною мовою програмування, яка широко використовується для створення різноманітних програм і систем. Вона також може бути успішно використана для розробки систем обробки звукових сигналів, проте варто враховувати кілька особливостей, пов'язаних з її використанням у цій сфері.

Java має багатий екосистему засобів та бібліотек, які допомагають у роботі зі звуком.

– наприклад, для обробки звукових сигналів в Java можна використовувати бібліотеку Java Sound API, яка надає можливості для запису та відтворення звуку, а також для обробки аудіоданих, таких як фільтрація, зміна гучності, ефекти та інше;

– для більш специфічних задач існують також спеціалізовані бібліотеки, які пропонують розширені можливості для обробки звукових сигналів, наприклад, JFugue для компонування музики або TarsosDSP для аналізу та обробки аудіоданих.

Однак, при використанні Java для обробки звуку важливо враховувати певні обмеження.

– одним з них є збирач сміття (Garbage Collector, GC) у Java, який автоматично відслідковує та звільняє пам'ять, використану об'єктами, що вже не потрібні. Використання GC може вплинути на виконання системи обробки звукових сигналів, особливо якщо вона працює в режимі реального часу, де низька затримка є критичною;

– для досягнення найкращої продуктивності може знадобитися оптимізація використання пам'яті та управління об'єктами;

– ще одним важливим аспектом використання Java для обробки звуку є потреба у встановленні та використанні Java Virtual Machine (JVM), щоб виконувати скомпільовану програму. Це означає, що користувачам, які хочуть використовувати систему обробки звукових сигналів, необхідно мати встановлену JVM на своєму комп'ютері. Це може становити певну необхідність та складність для розповсюдження програми. На рисунку 2.4 зображено алгоритм роботи віртуальної машини Java.

Незважаючи на ці обмеження, Java є популярним вибором для розробки систем обробки звукових сигналів завдяки своїм перевагам, таким як простота використання, масштабованість, багатий вибір бібліотек та платформонезалежність. Крім того, розробники Java продовжують працювати над удосконаленням мови та її екосистеми, що робить її все більш привабливою для розробки систем обробки звуку.

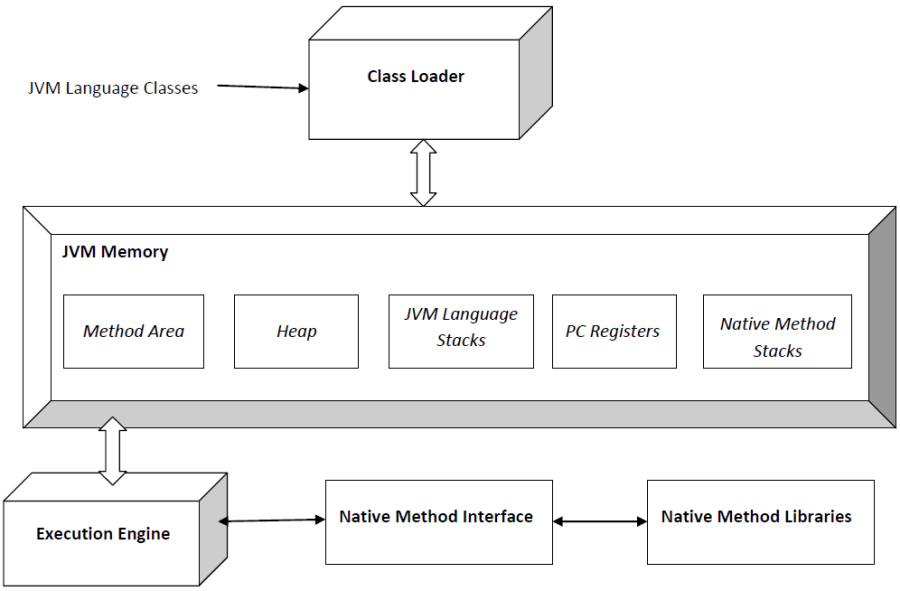

Рисунок 2.4 – Алгоритм роботи віртуальної машини Java

2.1.2 Вибір технологій симуляції навколишнього середовища

Симуляція навколишнього середовища є невід'ємною частиною тестування системи, особливо при створенні систем обробки звуку. Вона дозволяє моделювати різні аспекти реального світу, такі як акустичні властивості простору, розташування об'єктів і перешкод, що впливають на поширення звуку.

Для симуляції світу та перешкод для звуку ігрові двигуни є чудовим вибором. Ігрові двигуни, такі як Unity, Unreal Engine або CryEngine, вже мають потужні функції для створення віртуальних світів з великою кількістю об'єктів, реалістичною графікою та фізикою.

Unity, зокрема, є популярним вибором для симуляції навколишнього середовища у задачах обробки звуку, особливо зважаючи на його простоту використання та зручність. Unity надає інтуїтивний інтерфейс, широкий набір готових компонентів та можливість налаштування акустичних властивостей об'єктів і простору. Інтерфейс Unity можна побачити на рисунку 2.5.

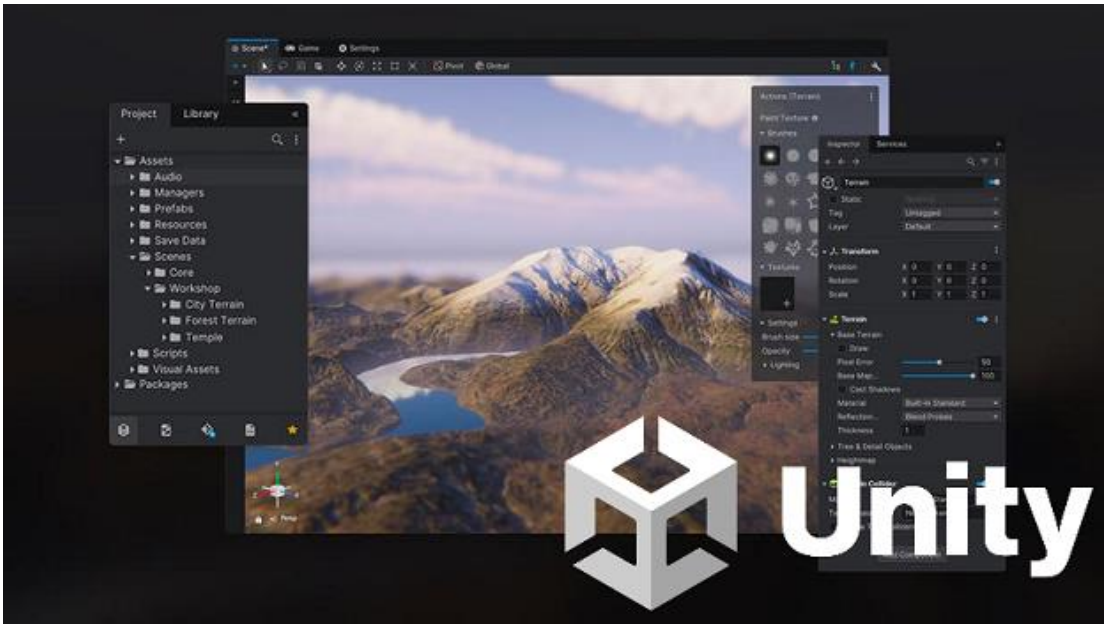

Рисунок 2.5 – Двигун Unity

Один з головних факторів, які роблять Unity зручним для симуляції навколишнього середовища, полягає у його здатності дошвидкого прототипування та візуалізації. Завдяки візуальному інтерфейсу і різноманітним інструментам, Unity дозволяє швидко створювати і відлажувати сцени, додавати звукові джерела, визначати властивості матеріалів і перешкод.

Крім того, Unity має широку підтримку різноманітних платформ і пристроїв, що дозволяє легко адаптувати симуляційну систему до різних середовищ і використовувати її для презентацій або в реальному часі.

У порівнянні з іншими ігровими двигунами, Unity може бути простішим у використанні, що особливо важливо для тих, хто не має глибоких знань у програмуванні або симуляціях. Він надає розширений набір інструментів, які допомагають управляти звуковими ефектами, акустикою та іншими параметрами для досягнення потрібного результату.

Загалом, використання ігрових двигунів, зокрема Unity, для симуляції навколишнього середовища у задачах обробки звуку має багато переваг. Вони дозволяють моделювати реалістичні акустичні умови, створювати і відлажувати сцени швидко і ефективно, а також використовувати готові компоненти для досягнення потрібного звукового ефекту.

2.2 Вибір архітектури

Для створення компресуючого еквалайзера не потрібна складна архітектура, але розбиття системи на декілька модулів може бути корисним і має свої переваги. Деталі структури інтеграції системи з навколишніми пристроями зображено на кресленику ІА91.220БАК.004 Д1. Виділено такі модулі:

– модуль зчитування семплів: Цей модуль відповідає за отримання аудіосигналу та перетворення його на числові значення (семпли), які можна обробляти. Він може включати у себе читання аудіофайлу, потокове отримання даних або навіть отримання семплів з мікрофону в реальному часі. Модуль зчитування семплів це лише інтерфейс. Його задача – абстрагувати доступ до низькорівневих деталей запису звуку, тому детально цей модуль розглянуто не буде;

– модуль аналізу семплів: Після отримання семплів необхідно проаналізувати їх для отримання інформації про частотні характеристики аудіосигналу. Цей модуль може включати в себе розрахунок спектрального аналізу сигналу, виявлення основних частот або інші методи для отримання інформації про спектр сигналу;

– модуль обробки семплів: Після аналізу семплів можна здійснити обробку сигналу, наприклад, застосувати компресію або еквалайзацію до семплів. Цей модуль включатиме логіку для обчислення ефектів обробки сигналу, таких як зменшення динамічного діапазону сигналу або регулювання рівнів окремих частот;

– модуль застосування семплів: Завершальний модуль відповідає за застосування оброблених семплів звукового сигналу до вихідного каналу. Це може бути збереження оброблених семплів в аудіофайл, передача їх через аудіовихід або навіть стрімінг в реальному часі. Модуль застосування семплів це лише інтерфейс. Його задача – абстрагувати доступ до низькорівневих деталей програвання звуку, тому детально цей модуль розглянуто не буде.

Розділення системи на модулі дозволяє краще організувати функціональність та спростити розробку та підтримку. Кожен модуль може бути реалізований окремо і працювати незалежно, що полегшує тестування і модифікацію кожного модуля окремо.

2.3 Аналіз інструментів обробки звуку

Звукова обробка є важливою складовою виробництва музики, аудіоінженерінгу, радіо та багатьох інших галузях. Існує багато різних інструментів обробки звуку, кожен з яких має свою функцію і використовується для певних задач. Деякі з найпоширеніших інструментів обробки звуку включають такі:

– еквалайзер (Equalizer): Еквалайзер використовується для контролю та налаштування рівнів звукових частот у звуковому сигналі. Він дозволяє підсилити або приглушити конкретні частоти, щоб впливати на звучання звуку. Еквалайзери зазвичай мають різні смуги частот і регулюються за допомогою полосових регуляторів або параметричних настройок. Приклад еквалайзера зображено на рисунку 2.6;

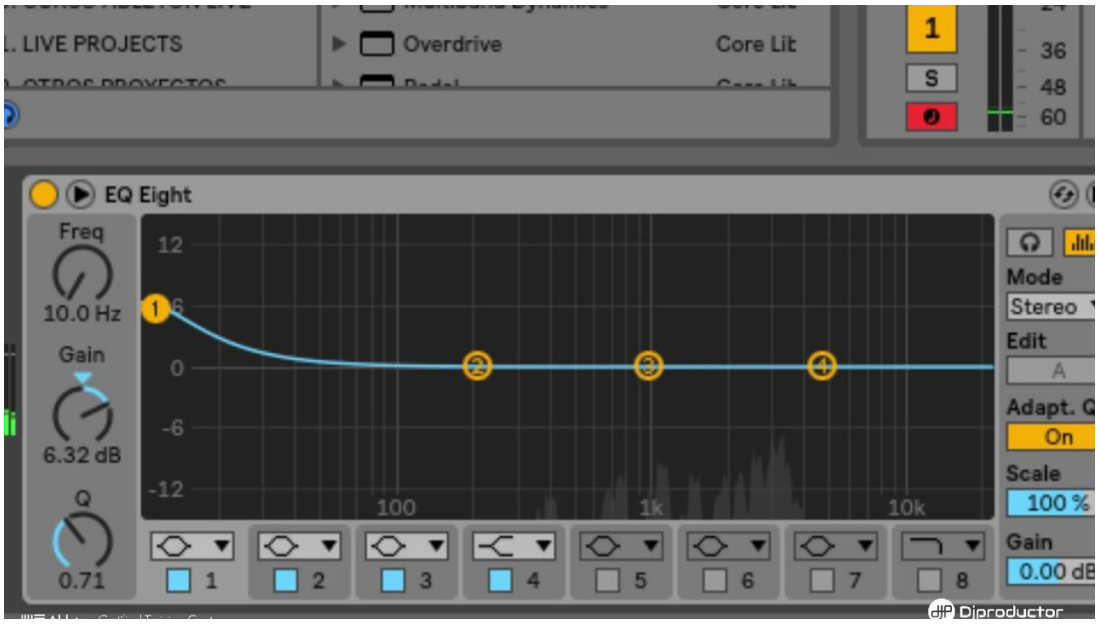

Рисунок 2.6 – Еквалайзер

– компресор (Compressor): Компресор використовується для регулювання динамічного діапазону звукового сигналу. Він знижує гучність сильних звуків і підсилює слабкі звуки, зменшуючи різницю між ними. Компресори корисні для управління рівнями гучності та забезпечення рівномірного звучання. Приклад компресора зображено на рисунку 2.7;

Рисунок 2.7 – Компресор

– лімітер (Limiter): Лімітер є розширенням компресора і використовується для обмеження пікув звукового сигналу до заданого рівня. Він дозволяє забезпечити, щоб сигнал не перевищував певного максимального рівня гучності, що допомагає уникнути спотворень або перевантаження аудіо обладнання. Приклад лімітера зображено на рисунку 2.8;

Рисунок 2.8 – Лімітер

– дилей (Delay): Дилей використовується для створення ефекту відлуння або повторення звукового сигналу. Він додає затримку до звуку, що створює враження просторовості або розширення звукового поля. Приклад дилея зображено на рисунку 2.9;

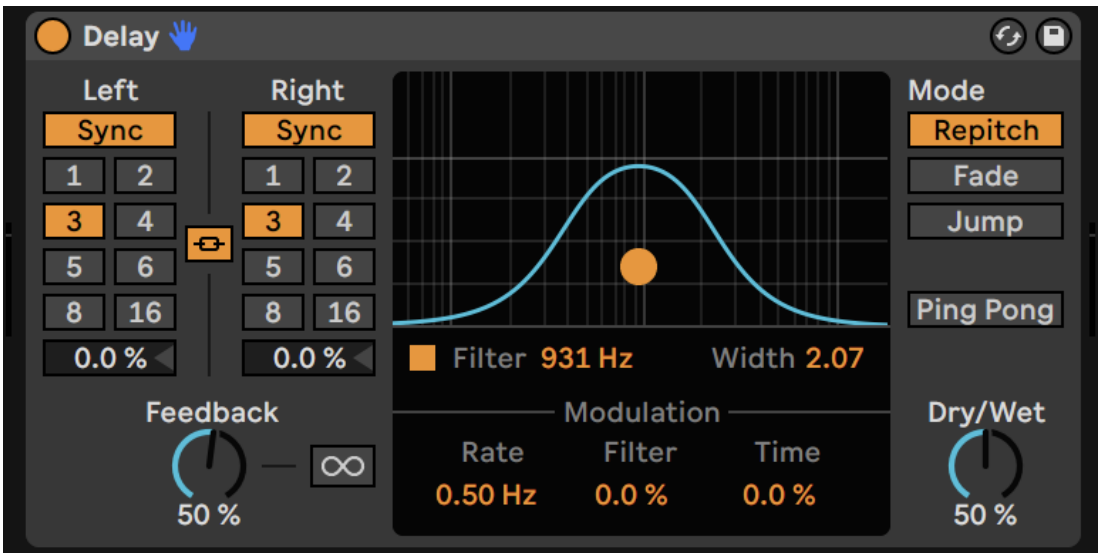

Рисунок 2.9 – Дилей

– реверберація (Reverb): Реверберація також дозволяє створювати просторовий ефект у звуковому сигналі, але вона симулює звукові відбиття у приміщенні. Реверберація додає глибину та просторовість до аудіо, надаючи враження того, що звук знаходиться у конкретному середовищі. Приклад реверберації зображено на рисунку 2.10;

Рисунок 2.10 – Реверберація

– фленжер (Flanger): Фленжер створює ефект зміщення фази для звукового сигналу. Він створює характерний "змив" або "шелест" у звучанні, який може бути використаний для створення цікавих аудіо ефектів. Приклад фланджера зображено на рисунку 2.11;

Рисунок 2.11 – Фланджер

– хорус (Chorus): Хорус створює ефект широкого звукового поля, схожий на звучання хору або групи інструментів, які грають одночасно. Він додає глибину та розширює звуковий образ. Приклад хоруса зображено на рисунку 2.12.

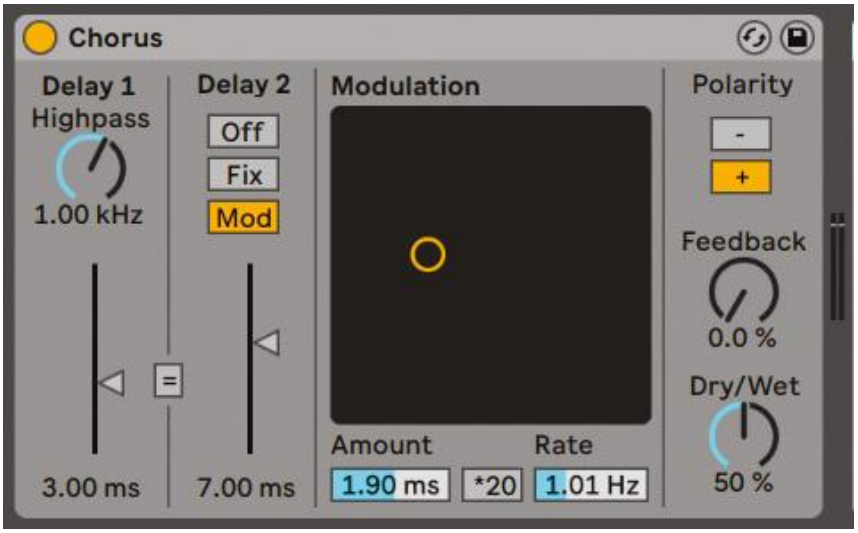

Рисунок 2.12 – Хорус

Це лише кілька прикладів інструментів обробки звуку. Існує безліч інших, включаючи фазові зсуви, дисторшн, гейт, ехо та багато інших, які мають свої

особливості та застосування, та представлені вище є найпопулярнішими та найефективнішими.

Висновки до розділу 2

В даному дипломному дослідженні було проведено аналіз різних мов програмування, технологій симуляції навколишнього середовища та інструментів обробки звуку. Головною метою аналізу було визначення найбільш підходящих варіантів для реалізації поставлених завдань.

У результаті дослідження було виявлено, що мова програмування Rust володіє широким набором функцій і можливостей, які забезпечують високу продуктивність та ефективність розробки. Використання Rust дозволить нам реалізувати складні алгоритми та оптимізувати роботу програмного забезпечення.

Для реалізації симуляції навколишнього середовища було обрано технологію Unity. Unity володіє потужними інструментами для створення ігрових середовищ та симуляцій, що дозволяє нам легко втілити задумані ідеї. Крім того, Unity має широку спільноту розробників, що сприяє обміну знаннями та підтримці.

У контексті обробки звуку були обрані еквалайзер та компресор. Еквалайзер дозволяє налаштовувати частотну характеристику звуку, що дає змогу збалансувати його тон і створити бажаний звуковий ефект. Компресор, з свого боку, дозволяє контролювати динаміку звуку шляхом зменшення різниці між найтихішим і найгучнішим моментами. Використання цих інструментів допоможе покращити якість обробки звукових даних та створити більш збалансоване звучання.

Обрання мови програмування Rust, технології Unity та використання еквалайзера та компресора для обробки звуку вважається оптимальним рішенням для досягнення поставлених цілей у даному дослідженні. Ці вибори надають нам необхідні інструменти та можливості для успішної реалізації проекту і досягнення високої якості результатів.

3 РЕАЛІЗАЦІЯ СИСТЕМИ

3.1 Реалізація архітектури

Архітектура застосунку складається з двох основних модулів, а саме аналізатора звуку та обробника звуку. Кожен з цих модулів, у свою чергу, має в собі дві складові частини - обчислювач та користувацький інтерфейс. Ця дворівнева структура дозволяє ефективно виконувати різноманітні завдання пов'язані з аналізом та обробкою звукових даних. Загальна структура модулів систем зображена на кресленику ІА91.220БАК.004 Д3.

3.2 Реалізація обробника звуку

Задача обробника звуку полягає в тому, щоб змінювати вхідні зразки звуку згідно з налаштуваннями обробника. Це дозволяє задовольняти специфічні вимоги, які встановлені для обробника звуку. Крім того, обробник звуку надає користувачам інтерфейс, який дозволяє їм налаштувати параметри обробки звуку за своїми вподобаннями.

Обробник звуку складається з двох основних компонентів: обчислювальної частини та користувацького інтерфейсу. Обчислювальна частина відповідає за виконання алгоритмів і операцій над вхідними зразками звуку згідно з налаштуваннями, які встановлені користувачем або задані за замовчуванням.

Користувацький інтерфейс обробника звуку забезпечує взаємодію користувача з обробником і дозволяє йому встановлювати налаштування обробки звуку.

Ця двокомпонентна структура обробника звуку дозволяє досягти гнучкості та легкої розширюваності обробки звуку. Користувач може налаштовувати ефекти еквалайзера залежно від своїх потреб та вподобань, що дозволяє досягти бажаного звукового результату.

3.2.1 Реалізація обчислювача

Обчислювач обробника звуку, реалізований на мові програмування Rust, є ключовим модулем, який відповідає за зміну звукових зразків. Цей модуль має широкий набір функцій та можливостей, що дозволяють ефективно виконувати обробку звуку згідно з налаштуваннями, встановленими користувачем.

В модулі наявні функції для перетворення Фур’є, функції застосування налаштувань: підсилення звуку, затримка для атаки, тощо, а також утилітні функції.

Для отримання доступу до вихідних звукових зразків використовується Unity API, яке надає широкі можливості для маніпулювання звуковими даними. Завдяки цьому API зчитується масив зразків звуку, які надалі будуть змінені та оброблені.

Амплітудно-частотна характеристика та амплітудно-часова характеристика є двома поняттями, пов'язаними з аналізом сигналів. Ці характеристики використовуються в сферах, пов'язаних з акустикою, електронікою, сигнальною обробкою та іншими дисциплінами.

Амплітудно-частотна характеристика описує залежність амплітуди сигналу від його частоти. Вона відображає, як змінюється амплітуда сигналу при різних частотах. Вона може бути представлена графіком, де по горизонтальній вісі відображена частота, а по вертикальній вісі - амплітуда сигналу. Також вона використовується для визначення частотних характеристик пристроїв, таких як фільтри, підсилювачі, акустичні системи тощо.

Амплітудно-часова характеристика, натомість, описує залежність амплітуди сигналу від часу. Вона показує, як змінюється амплітуда сигналу протягом певного періоду часу. Вона може бути представлена графіком, де по горизонтальній вісі відображений час, а по вертикальній вісі - амплітуда сигналу. Вона застосовується для вивчення властивостей сигналу в домені часу, таких як тривалість, амплітудні зміни, форма хвилі тощо.

Перехід між амплітудно-частотною характеристикою та амплітудно-часовою характеристикою відбувається за допомогою перетворень сигналу. Наприклад, для перетворення сигналу з домену часу в домен частоти використовуються методи, такі як перетворення Фур'є. Це дозволяє отримати амплітудно-частотну характеристику, яка показує розподіл амплітуди сигналу в залежності від частоти.

Зворотне перетворення Фур'є або інші методи дозволяють перетворити сигнал з домену частоти назад у домен часу, що дозволяє отримати амплітудно-часову характеристику. Це використовується для аналізу та модифікації сигналів в залежності від їхньої амплітуди в різні моменти часу.

Різниця між амплітудно-частотною характеристикою та амплітудно-часовою характеристикою полягає в тому, що перша описує залежність амплітуди від частоти сигналу, а друга - залежність амплітуди від часу. Перехід між цими характеристиками відбувається через відповідні математичні преобразування сигналу, такі як перетворення Фур'є та зворотне перетворення Фур'є.

Для отримання Амплітудно-Частотної Характеристики вхідних зразків використовуватиметься перетворення Фур’є

Перетворення Фур'є є одним з найважливіших інструментів у сигнальній обробці. Це математичний метод, який дозволяє розкласти сигнал на його складові частоти. Перетворення Фур'є перетворює сигнал з часового домену в частотний домен.

Одна з ключових ідей, на якій ґрунтується перетворення Фур'є, полягає в тому, що будь-який періодичний сигнал можна представити як суму гармонічних сигналів різних частот. Перетворення Фур'є дозволяє розкласти складний сигнал на його прості компоненти (частоти). Це можна побачити з формули (3.1) [3], яка показує функцію, що приймає частоту і повертає єдине значення амплітуди і фази у вигляді комплексного числа.

З точки зору обробки сигналів, перетворення Фур'є має багато практичних застосувань. Одним з них є аналіз спектра сигналу. Перетворення Фур'є дозволяє визначити, які частоти присутні в сигналі і з якою інтенсивністю. Це корисно,

наприклад, для аналізу звукових сигналів, виявлення резонансних явищ або визначення частоти коливань в електричних сигналах.

Ще одним застосуванням перетворення Фур'є є фільтрація сигналів. Використовуючи перетворення Фур'є, можна відфільтрувати певні частоти сигналу, видаливши непотрібні складові або виділивши потрібні. Це дозволяє знизити шум, видалити спотворення або виділити корисну інформацію.

$$F(\omega) = \int_{-\infty}^{\infty} f(t) e^{-i\omega t}\, dt, \qquad (3.1)$$

де:

- $f(t)$ – функція, для якої виконується перетворення;
- ω – цільова частота, для якої обчислюється АЧХ;
- t – час.

Дискретне перетворення Фур'є (ДПФ) є потужним інструментом в області обробки сигналів, який використовується для аналізу і модифікації цифрових сигналів. Воно базується на математичній теорії перетворень Фур'є і дає змогу розкласти сигнал на його складові частоти.

Основна ідея ДПФ полягає в тому, щоб представити цифровий сигнал у частотному домені, де можна аналізувати його спектральні властивості. Для цього використовується розклад сигналу на комплексні експоненти з різними частотами.

ДПФ використовується для розв'язання багатьох задач обробки сигналів, таких як фільтрація, згладжування, спектральний аналіз, кореляція і синтез звуку. Наприклад, за допомогою ДПФ можна визначити, які частоти присутні у сигналі, або які компоненти сигналу вносять найбільший внесок у його спектр.

Процес ДПФ можна уявити як розклад сигналу на гармонічні компоненти різних частот. Результатом є спектральне представлення сигналу, яке включає амплітуду та фазу кожної частоти. За допомогою цього спектрального представлення можна виконувати різні операції, такі як фільтрація шуму, виділення певних.

Алгоритм роботи Дискретного перетворення Фур’є нескладно зрозуміти з формули (3.2) [4], на якій зображена чітка залежність частоти і вихідного значення – фази і амплітуди. Формула ДПФ використовується декілька разів при одному перетворенні віхдних зразків до АЧХ, а саме один раз на одну частоту.

$$X_k = \sum_{n=0}^{N-1} x_n * e^{\frac{-i2\pi}{N}kn}, \quad (3.2)$$

де:

- N – кількість вхідних значень;
- x_n – амплітуда кожного окремого аудіо-зразка;
- k – частота, для якої обчислюється АЧХ;
- X_k – АЧХ для обраної частоти.

Для обчислення ДПФ використовуються алгоритми, такі як алгоритм Баттерворта, алгоритм Кулі-Тьюкі і алгоритм Шухарда-Хартлі. Однак найбільш поширеним і ефективним є алгоритм швидкого перетворення Фур'є (ШПФ), який забезпечує швидке обчислення ДПФ для послідовності з N елементів.

Швидке перетворення Фур'є (ШПФ) - це алгоритм, який використовується для швидкого обчислення дискретного перетворення Фур'є (ДПФ) із метою ефективного оброблення сигналів у часовій або просторовій областях.

Обчислення ДПФ за допомогою прямого підходу може бути дорого і обійдеться значними обчислювальними витратами, особливо для великих сигналів. Тут і на допомогу приходить ШПФ, який є оптимізованою версією ДПФ і здатний прискорити обчислення.

ШПФ базується на розкладанні ДПФ на менші часткові перетворення Фур'є (ЧПФ) та комбінуванні їх результатів. Він використовує деякі особливості симетрії та періодичності, що дозволяють зменшити кількість операцій для обчислення.

Найпоширенішим алгоритмом ШПФ є алгоритм Кулі-Тьюкі. Він базується на ідеї розбиття сигналу на парні та непарні елементи, обчисленні ЧПФ для кожної половини сигналу, а потім їх комбінуванні для отримання остаточного результату.

Алгоритм ШПФ має часову складність O(N log N), де N - кількість вхідних даних. Це значно ефективніше, ніж прямий підхід з часовою складністю O(N^2).

Швидке перетворення Фур’є це вже алгоритмічна задача, тому його роботу показано діаграмою. На кресленику ІА91.220БАК.004 Д2 зображено приклад роботи ШПФ з 8 змінними. В такому випадку виконується одна стадія перестановки і три стадії обробки – звідси складність O(log N). А оскільки на кожній стадії виконується прохід по всім змінним, складність кожної стадії складає O(N). Тому результуюча складність алгоритму складає O(N log N).

Обернене перетворення Фур'є (Inverse Fourier Transform) є математичним методом, який дозволяє відновити оригінальну функцію з її спектрального представлення. В основі цього перетворення лежить ідея розкладу будь-якої функції на суму гармонічних складових різних частот.

Що стосується самої формули комбінації змінних, то і для прямого перетворення Фур'є, і для оберненого перетворення Фур'є використовуються дуже схожі алгоритми. Це видно з формули (3.3) [5]. Однак, при оберненому перетворенні Фур'є, результатом буде оригінальна функція, а не її спектральне представлення.

$$x_n = \frac{1}{N}\sum_{k=0}^{N-1} X_k * e^{\frac{i2\pi}{N}kn}, \quad (3.3)$$

де:

- N – кількість вхідних значень;
- x_n – амплітуда обраного аудіо-зразка;
- k – частота, для якої обчислюється АЧХ;
- X_k – АЧХ для кожної окремої частоти.

Обробник, який виконує функції еквалайзера та компресора одночасно, є складним пристроєм з багатьма можливостями налаштувань. Його функціонал включає в себе масив точок контролю, кожна з яких пропонує широкий набір параметрів для налаштування. Детально параметри описані в таблиці 3.1.

Одним з головних параметрів є поріг вхідної амплітуди, який визначає мінімальний рівень сигналу, який сприймається обробником. Це дозволяє контролювати, коли обробка сигналу повинна починатися. Наступним налаштуванням є посилення, яке визначає зміну амплітуди сигналу після обробки. Це дає змогу регулювати гучність вихідного сигналу обробника.

Атака та згасання є ще двома важливими параметрами. Атака визначає, як швидко обробник реагує на зростання амплітуди вхідного сигналу, встановлюючи час, протягом якого обробник збільшує амплітуду. Згасання, навпаки, визначає, як швидко обробник реагує на спад амплітуди, встановлюючи час, протягом якого обробник зменшує амплітуду.

Налаштування атаки та згасання вимагають ретельного підходу при реалізації аналізатора звуку, адже значення цих параметрів не так просто розрахувати. Але атака і згасання мають на меті досягнення більш точного, ефективного оброблення даних, що запобігає травмувань слухачів.

Окрім цього, обробник пропонує налаштування частоти і ширини діапазону частот. Ці параметри дозволяють вибрати конкретні частоти, на які впливає обробник, а також контролювати ширину діапазону, в якому здійснюється обробка. Це надає можливість точно настроїти звуковий спектр обробки.

Таблиця 3.1 – Налаштування точки контролю обробника та їх опис

Назва параметру	Опис параметру
Поріг вхідної амплітуди	Мінімальне та максимальне значення амплітуди, за яких буде застосовано мінімальне або максимальне значення підсилення відповідно.
Посилення	Множник амплітуди сигнала.
Атака	Час, за який амплітуда зростає з мінімального до максимального значення

Згасання	Час, за який амплітуда спадає змаксимального до мінімального значення
Частота	Частота, амплітуда якого зазнає впливу
Ширина діапазону частот	Коефіцієнт ширини частот, в якому значення 0 означає, що змінена лише одна частота, а значення 1 означає, що змінені усі частоти, а значення між 0 та 1 є проміжними і налаштовують ширину діапазону частот змінених амплітуд

Поріг вхідної амплітуди та посилення – це пов’язані налаштування. До рівня вхідної амплітуди застосовується відповідне значення посилення таким чином, що значення підсилення обмежується максимальним та мінімальним значенням порогу вхідної амплітуди, а всередині діапазону значення посилення лінійно залежить від значення амплітуди. На рисунку 3.2 зображено графік залежності посилення від амплітуди.

В майбутньому таблиця налаштувань точок контролю обробника звуку може бути розширена додатковими параметрами для більш тонкого налаштування обробки звуку чи розширення предметної області системи шляхом додавання нових обробників, наприклад, для підвищення уявного рівня гучності без пошкодження апаратури або органів слуху або видалення шуму, створеного через проходження звуку крізь навколишнє середовище і перешкоди.

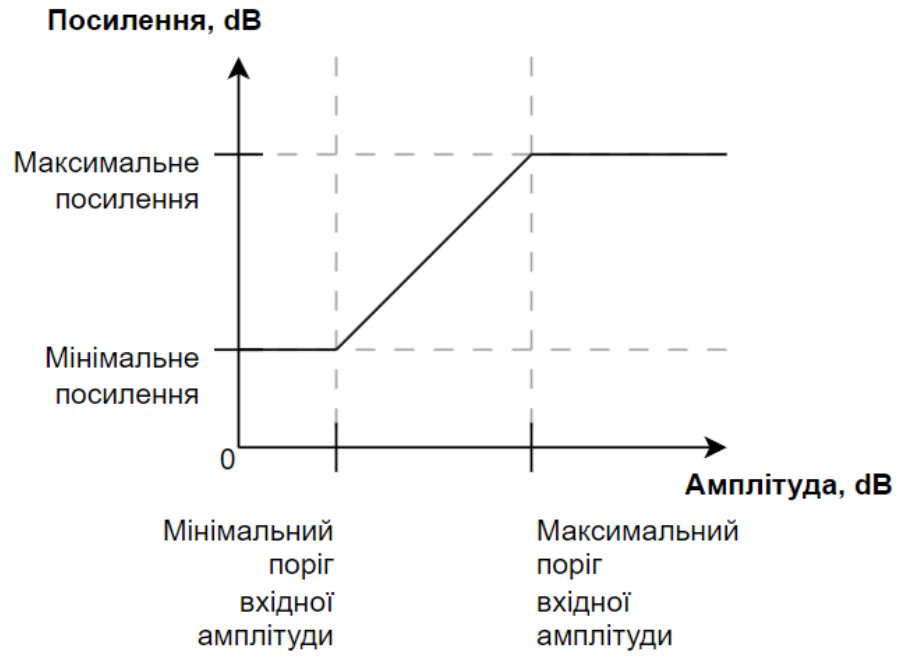

Рисунок 3.2 – Графік залежності посилення від вхідної амплітуди

Атака та згасання – це складові ADSR-обвідної. На рисунку 3.3 зображено графік залежності результуючої амплітуди від характеристик ADSR-обвідної.

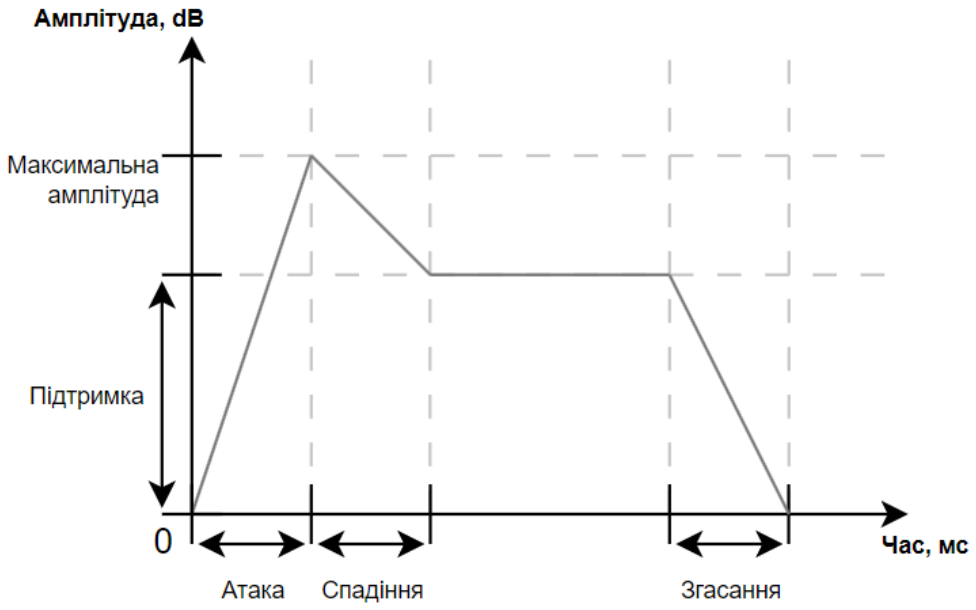

Рисунок 3.3 – Графік впливу ADSR-обвідної на амплітуду сигналу

Спадіння та підтримка не є корисними у випадку компресора, тому їх не використано.

Атака та згасання відповідають за швидкість реакції посилення на зміну вхідної амплітуди.

Атака являє собою час, за який посилення досягає максимального значення при мінімальному початковому значенні.

Згасання являє собою час, за який посилення досягає мінімального значення при максимальному початковому значенні.

На рисунку 3.4 зображено зміну посилення, як реакцію на різкі зміни амплітуди.

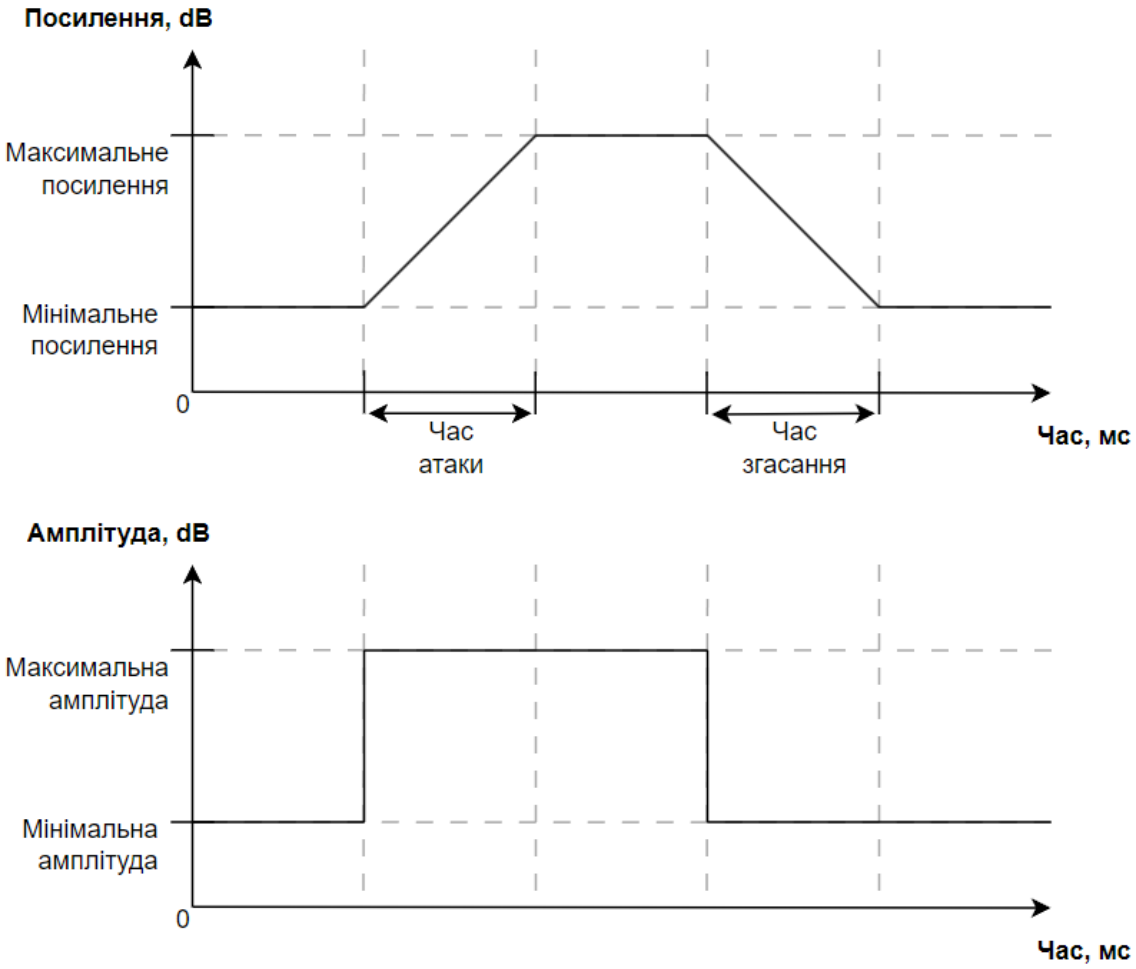

Рисунок 3.4 – Графік зміни посилення, при миттєвій зміні амплітуди вхідного сигналу

Атака та згасання потрібні для більш плавної реакції системи на різкі впливи.

Частотний діапазон точки контролю обробника визначається частотою та шириною діапазону.

Кожна точка контролю обробника виконує роль смугового фільтра.

Існує формула смугового фільтра (3.4) [1].

$$f(x) = e^{x^2(1-a^{-2})}, \tag{3.4}$$

де a – коефіцієнт ширини діапазону частот.

Вона приймає значення від 0 до 1, де 0 – надвузький діапазон (впливає на амплітуду одної частоти) а 1 – надширокий діапазон (впливає на амплітуду усіх частот). Графік функції смугового фільтра зображено на рисунку 3.5

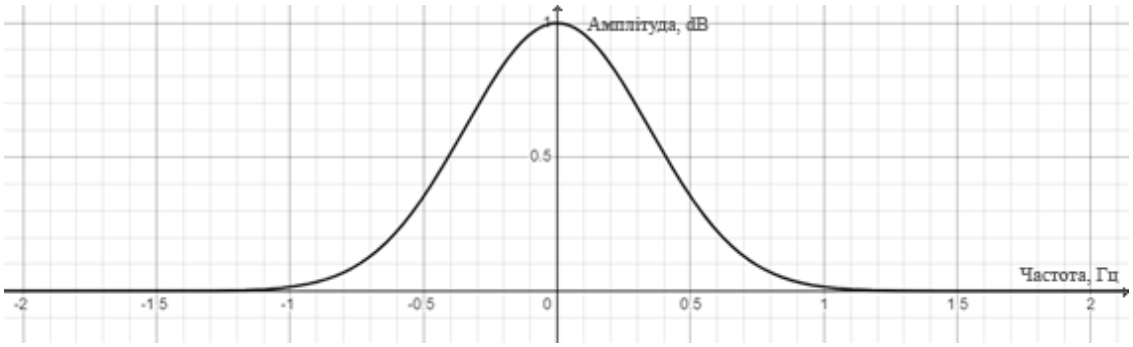

Рисунок 3.5 – Графік функції смугового фільтра

Коефіцієнт а дає нам можливість впливати на діапазон частот. Змінюючи його ми можемо отримати вузький і широкий фільтри, а також їх варіації. Графік функції смугового фільтра з різними налаштуваннями ширини діапазону зображено на рисунку 3.6. Червоним кольором зображено графік функції з коефіцієнтом a = 0.2, синім – a = 0.4, зеленим – a = 0.6, фіолетовим – 0.8.

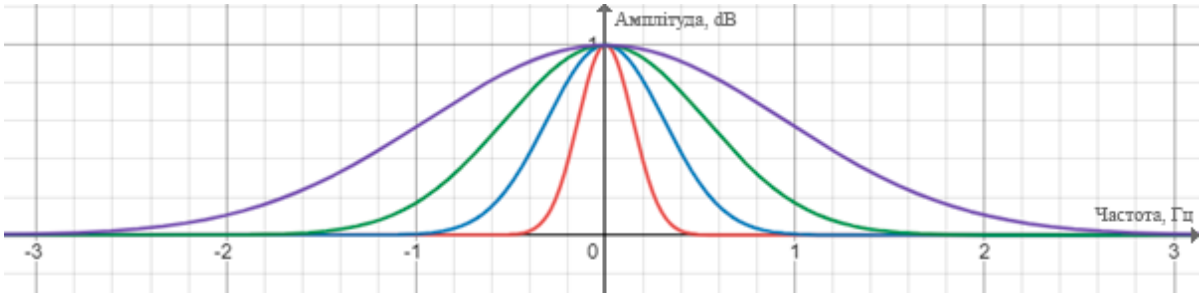

Рисунок 3.6 – Графіки функції смугового фільтра з різними налаштуваннями ширини діапазону

Також існує необхідність впливу на зміщені діапазони частот, тому до формули додано коефіцієнт зміщення. Результатом є функція (3.5).

$$f(x) = e^{(x-b)^2(1-a^{-2})}, \tag{3.5}$$

де:

– a – коефіцієнт ширини діапазону частот;

– b – коефіцієнт зміщення.

Отримана функція дає можливість впливати на будь-який діапазон частот. Графіки отриманої функції з різними налаштуваннями діапазону та цільовою частотою зображені на рисунку 3.7. Червоним кольором зображено графік функції з коефіцієнтом b = -1, фіолетовим – b = 0, зеленим – b = 2.

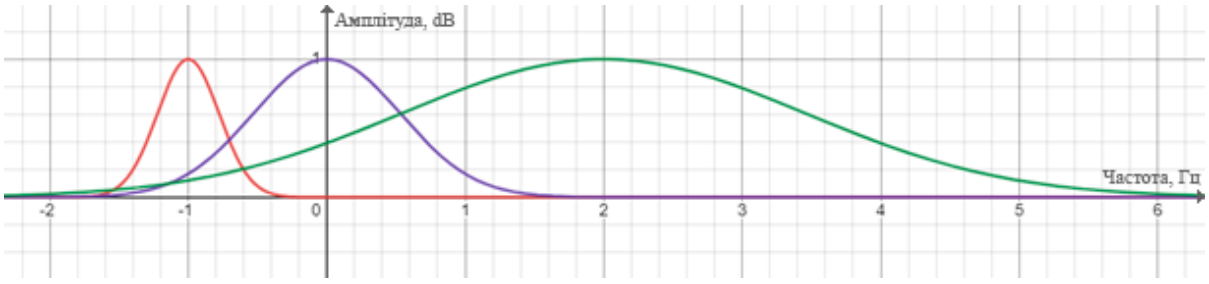

Рисунок 3.7 – Графіки функції смугового фільтра з різними налаштуваннями ширини діапазону та цільовою частотою

З характеристик мінімального та максимального підсилень система отримує коефіцієнт підсилення, що додано у функцію (3.6).

$$f(x) = ce^{(x-b)^2(1-a^{-2})}, \quad (3.6)$$

де:

- a – коефіцієнт ширини діапазону частот;
- b – коефіцієнт зміщення;
- c – коефіцієнт підсилення.

Графіки результуючої функції смугового фільтра з різними налаштуваннями ширини діапазону, цільовою частотою та значенням посилання зображено на рисунку 3.8. Червоним кольором зображено графік функції з коефіцієнтом c = 1.5, фіолетовим – c = 1, зеленим – c = 0.5.

Рисунок 3.8 – Графіки функції смугового фільтра з різними налаштуваннями ширини діапазону, цільовою частотою та значенням посилення

Для комбінації впливів, результати функцій усіх точок котролю додано. Результуючою та такою, що є сумою функцій усіх точок контролю, є функція (3.7). Графік функції зображено на рисунку 3.9 чорним кольором. Іншими кольорами зображено функції-доданки.

$$f(x) = \sum_{n=1}^{N} c_n e^{(x-b_n)^2(1-a_n{}^{-2})}, \quad (3.7)$$

де:

- a_n – коефіцієнт ширини діапазону частот кожної точки контролю;
- b_n – коефіцієнт зміщення кожної точки контролю;
- c_n – коефіцієнт підсилення кожної точки контролю;
- N – кількість точок контролю.

Рисунок 3.9 – Графіки функцій точок контролю та графік суми цих функцій

А оскільки отримана функція вливає на сигнал посиленням, тобто вихідне значення буде добутком вхідного сигналу та результуючого впливу, функцію приведено до зручної форми (3.8). Графік фінальної функції впливу наведено на рисунку 3.10.

$$f(x) = 1 + \sum_{n=1}^{N} c_n e^{(x-b_n)^2(1-a_n{}^{-2})}, \quad (3.8)$$

де:

- a_n – коефіцієнт ширини діапазону частот кожної точки контролю;
- b_n – коефіцієнт зміщення кожної точки контролю;
- c_n – коефіцієнт підсилення кожної точки контролю;
- N – кількість точок контролю.

Функція дуже плавна, що позитивно впливає на вихідний сигнал. Обробка аудіо потребує обережної роботи через її складність і можливості погіршити аудіо-пристрої, а також слух людини.

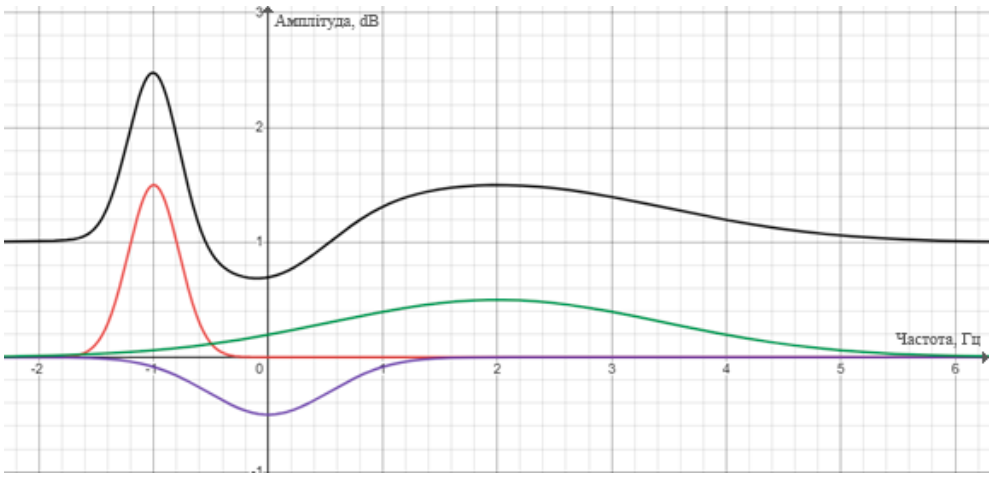

Рисунок 3.10 – Графіки функцій точок контролю та графік фінальної функції впливу

Вихідний сигнал розраховується за формулою (3.9).

$$y = x * f(x), \tag{3.9}$$

де:

– x – вхідний сигнал;

– $f(x)$ – функція впливу;

Таким чином комбінація кількох таких функцій з різними налаштуваннями може дати нам такий результат, що зображений на рисунку 3.11.

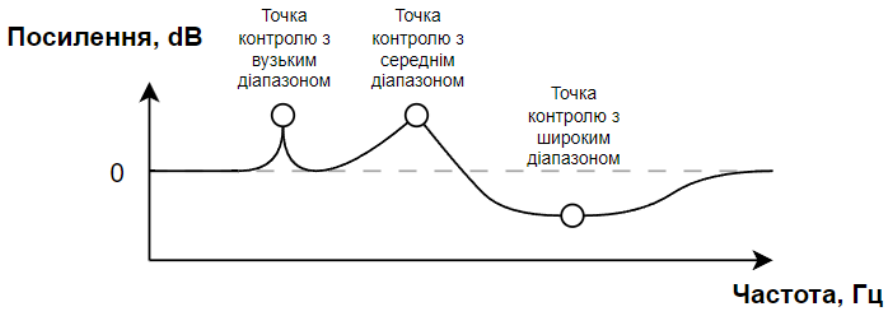

Рисунок 3.11 – Очікуваний приблизний графік функції впливу

3.2.2 Реалізація користувацького інтерфейсу

Користувацький інтерфейс реалізовано на мові програмування C# та інтегровано в інфраструктуру Unity.

Один з ключових аспектів реалізації інтерфейсу в Unity - це використання Custom Editor. Цей інструмент дозволяє створити власні, настроювані редактори для роботи з різними об'єктами і компонентами в Unity. За допомогою Custom Editor можна забезпечити зручну взаємодію з редактором Unity, додати додаткові функції та налаштування, специфічні для даної роботи.

Цей підхід до розробки інтерфейсу дозволяє реалізувати більш гнучкий та придатний для налаштувань інтерфейс. Завдяки використанню мови програмування C# та інфраструктури Unity, створено інтерактивні елементи, включаючи кнопки, текстові поля, випадаючі списки та багато іншого. Крім того, за допомогою Custom Editor досягнуто глибокої інтеграції розробленої системи з інфраструктурою Unity, що сприяє зручності використання та подальшого розвитку.

Також, в майбутньому такий інтерфейс нескладно буде переробити для використання на інший платформах, адже C# використовується не тільки в Unity.

3.2.2.1 Інтерфейс зовнішньої функції

Для доступу до обчислювань з рівня користувацького інтерфейсу буде використано Інтерфейс зовнішньої функції.

Інтерфейс зовнішньої функції (FFI) - це механізм, який дозволяє мовам програмування взаємодіяти з функціями, написаними на іншій мові або бібліотеками, які написані на інших мовах. FFI дозволяє здійснювати виклики з однієї мови до функцій, реалізованих на іншій мові, і передавати дані між ними.

У багатьох мовах програмування існує підтримка FFI, хоча конкретний синтаксис і механізми можуть відрізнятися. Основна ідея полягає в тому, що

реалізація функцій на одній мові може бути викликана з коду на іншій мові, і обидві мови можуть обмінюватися даними.

FFI може бути реалізований за допомогою різних підходів. Одним із поширених підходів є використання механізму зовнішніх бібліотек (наприклад, DLL або shared library), які містять функції, які можна викликати з інших мов. У такому випадку, FFI забезпечує механізми для завантаження бібліотеки, знаходження інтерфейсу функцій, передачі параметрів та отримання результатів.

Іншим підходом є використання протоколів обміну даними, таких як C-ABI (Application Binary Interface) або стандартизованих протоколів, які дозволяють мовам спілкуватися одна з одною. У такому випадку, FFI визначає правила передачі параметрів, описує формати даних та дозволяє викликати функції на різних мовах.

Зазвичай FFI забезпечує можливості для виклику зовнішніх функцій, передачі параметрів за значенням або посиланню, роботи зі змінними типами даних, обробки виключень та отримання результатів. Використання FFI може бути корисним в таких ситуаціях, як використання наявного коду на інших мовах, оптимізація чутливих до продуктивності фрагментів програми, або спілкування з системними ресурсами, які недоступні з поточної мови програмування.

3.2.2.2 Взаємодія Rust та C#

Для забезпечення зручного доступу до обчислювача розроблено два модулі на мові програмування Rust. Перший модуль включає в себе основну логіку обчислювача, а другий модуль надає доступ до першого та компілюється в динамічну бібліотеку. Це рішення дозволяє імпортувати створену бібліотеку у середу C#, що забезпечує зручну інтеграцію обчислювача у проекти, що використовують C#.

Використання динамічних бібліотек має переваги у випадку, коли потрібно використовувати обчислювач у різних середовищах або з різними мовами програмування для візуалізації чи інших проєктів. Завдяки цьому підходу,

обчислювач може залишатися без змін, і для його використання в новому середовищі або з новою мовою програмування потрібно лише розробити нову бібліотеку, яка експортує код на мові програмування Rust, враховуючи специфічні потреби даного середовища чи проекту.

Цей підхід до розробки обчислювача забезпечує гнучкість та розширюваність системи. Можливість використання обчислювача у різних проєктах та з різними мовами програмування дозволяє максимально використовувати його потенціал і забезпечує більш широке застосування системи в різних галузях індустрії. Крім того, використання мови програмування Rust дозволяє отримати високу швидкодію та надійність обчислювача, що є важливими факторами у багатьох областях, таких як наукові дослідження, аналіз даних та багатопотокові обчислення.

3.2.2.3 Компоненти Unity

У роботі було використано Unity - потужний гровий рушій, який базується на компонено-орієнтованій системі. У компоненто-орієнтованій системі усі об’єкти – це сутності, кожна з яких являє собою набір компонентів. Це дозволяє створювати різноманітні механіки, що працюють незалежно одна від одної. Архітектура компоненто-орієнтованої системи зображена на рисунку 3.12.

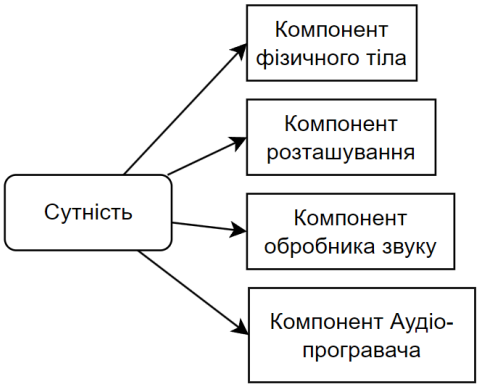

Рисунок 3.12 – Архітектура компоненто-орієнтованої системи

Одним із важливих аспектів дослідження було оброблення звуку, і для цього був використаний ігровий об'єкт, що містить компонент AudioSource. Однак, з метою поліпшення функціональності і забезпечення зручного інтерфейсу для обчислювача, було розроблено новий компонент.

Цей новий компонент був створений з метою перенаправлення аудіо зразків до інтерфейсу обчислювача. Його функціонал полягає в отриманні аудіо даних від компонента AudioSource і передачі їх на обчислювач для подальшої обробки. Це дозволяє використовувати потужні можливості обчислювача для редагування та обробки звукових зразків, що відкриває широкі перспективи для реалізації різноманітних аудіо ефектів та обробки звуку в грі.

Розроблений компонент забезпечує взаємодію між грою, аудіо джерелами та обчислювачем. Він може виконувати такі завдання, як зчитування аудіо даних з різних джерел, передача їх до обчислювача для обробки, а також повернення оброблених даних для подальшого відтворення звуку. Це дозволяє реалізувати складні аудіо ефекти, синтез звуків, мікшировання та інші операції над звуком у грі.

При використанні розробленого компонента здійснюється ефективне управління звуковими ресурсами та їх обробкою в грі. Він забезпечує гнучкий підхід до роботи зі звуком, що дозволяє розширити можливості гри і забезпечити більш реалістичне та захоплююче аудіо оточення для гравців. Таким чином, розроблений компонент є важливою складовою системи обробки звуку у грі, забезпечуючи високу якість звукового досвіду для користувачів.

3.2.2.4 Візуалізація

Користувацький інтерфейс обробника дозволяє користувачу налаштовувати різноманітні параметри, що впливають на процес обробки даних. Це включає в себе параметри, такі як частота, ширина діапазону, атака та посилення.

Unity Editor надає зручні інструменти для валідації та обмеження вводу користувача. За допомогою цих інструментів можна встановити мінімальні та

максимальні значення для кожного параметра, використовувати обмеження типів даних, перевіряти правильність введених даних та надавати зрозумілі повідомлення про помилки користувачеві.

Однак для досягнення візуальної репрезентації змінених параметрів необхідно використати шейдери, зокрема фрагментні шейдери, та мову HLSL. Графічний API дозволяє нам використовувати різні шейдерні стадії, окрім фрагментної. Приклади зображені на рисунку 3.13.

Але оскільки тривимірний рендеринг в даній ситуації не використовується, обрано саме фрагментний шейдер. Такі шейдери використовуються для обробки пікселів на графічному процесорі, що дозволяє змінювати їх кольори, прозорість, насиченість та інші властивості. Мова HLSL (High-Level Shading Language) використовується для написання програм, які виконуються на графічному процесорі.

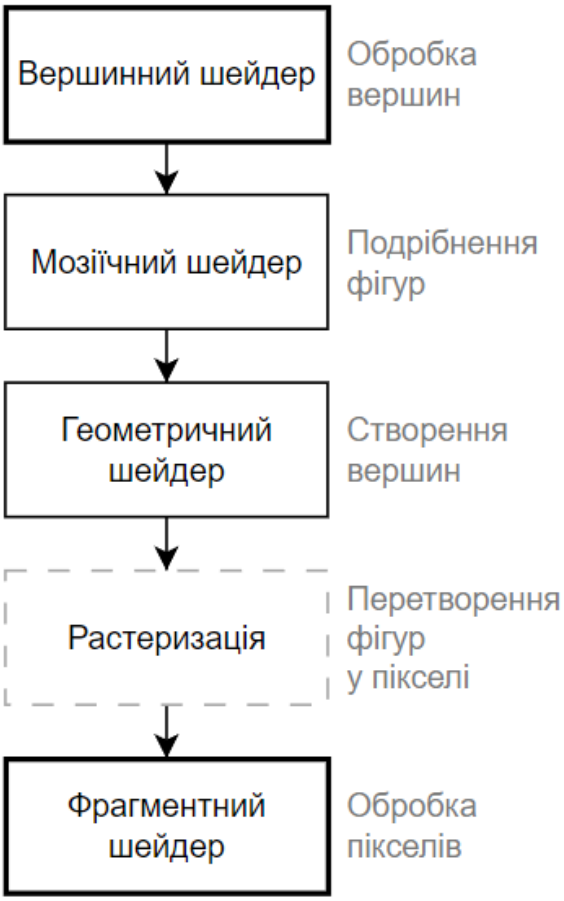

Рисунок 3.13 – Шейдерні стадії

За допомогою шейдерів ми можемо передавати параметри обробки даних з компонента на стороні C# у фрагментний шейдер, де вони будуть використовуватись для зміни вигляду пікселів. Результат обробки даних відображається на текстурі, яка повертається назад до скрипта на стороні C#, де вона може бути відображена на екрані в потрібному місці користувацького інтерфейсу. Таким чином, користувач може в реальному часі спостерігати за впливом змінених параметрів на вихідні дані і вносити необхідні корективи.

Використання шейдерів та мови HLSL надає гнучкість та широкі можливості для візуалізації процесу обробки даних. Можна створити різні ефекти, анімацію, переходи та інші візуальні елементи, що допоможуть користувачу краще розуміти та контролювати процес обробки даних у користувацькому інтерфейсі обробника.

Першим кроком створення інтерфейсу є створення компоненту і додавання відповідних атрибутів для обмеження значень параметрів. Результат зображено на рисунку 3.14

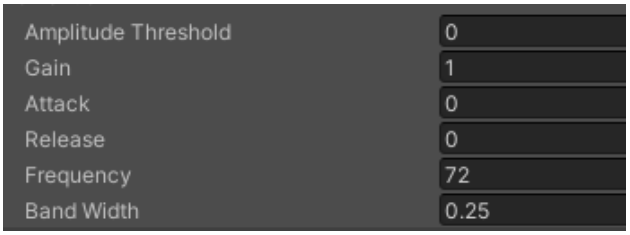

Рисунок 3.14 – Інтерфейс параметрів точки контролю

Наступним кроком є налаштування кількості точок контролю і, відповідно, кількості налаштувань. Для цього додамо масив параметрів та змінімо код відображення для можливості додавання та вилучення нових точок контролю. Результат зображено на рисунку 3.15.

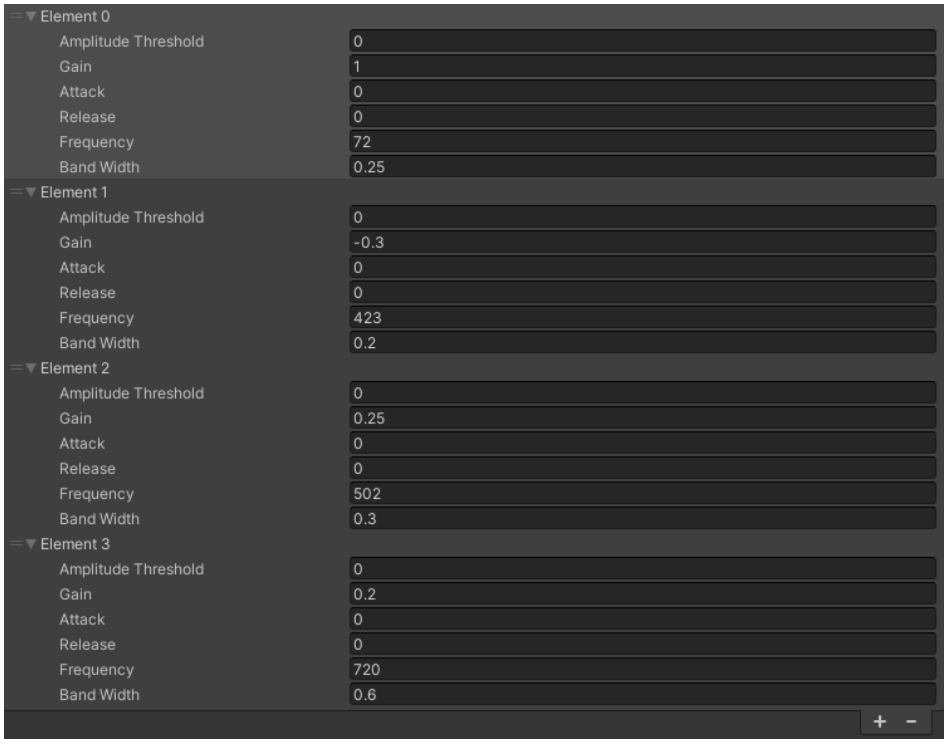

Рисунок 3.15 – Інтерфейс масиву параметрів точок контролю

Наступним важливим кроком у розробці є створення візуалізації роботи обробника. У середовищі Unity використано API для створення, рендерингу та очищення текстур для ефективної візуалізації роботи обробника та його вплив на вхідні дані.

Роботу обробника показано за допомогою візуалізації амплітуд контрольних точок, функції впливу та результуючої АЧХ. Результат зображено на рисунку 3.16

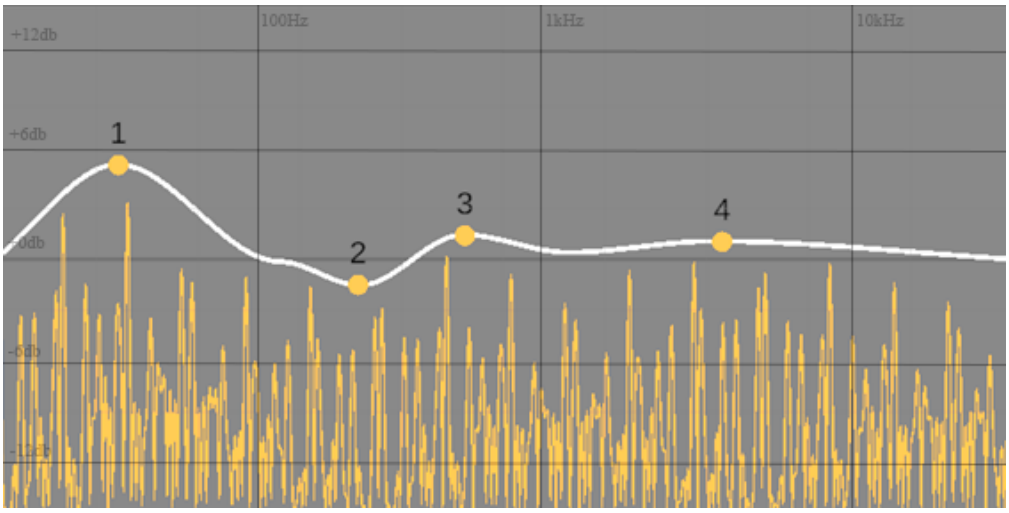

Рисунок 3.16 – Інтерфейс візуалізації роботи обробника звуку

3.3 Реалізація аналізатора звуку

Задача аналізатора звуку полягає в багатоаспектному дослідженні та аналізі звучання в різних точках приміщення. Основною метою є визначення оптимальних налаштувань обробника звуку, які спрямовані на зменшення середнього відхилення амплітуд аудіо зразків до мінімального рівня.

Процес аналізу звуку включає збір звукових даних з різних джерел, таких як мікрофони або аудіо вхідні пристрої. Отримані аудіо зразки потім піддаються детальному дослідженню та аналізу. Важливим етапом цього процесу є визначення середнього відхилення амплітуд звукових зразків.

Для досягнення мінімального середнього відхилення амплітуд аудіо зразків необхідно встановити налаштування обробника звуку, які оптимізують звуковий сигнал у різних точках приміщення. Це може включати налаштування еквалайзера, компресора, шумозаглушення та інших параметрів обробки звуку.

Після визначення оптимальних налаштувань обробника звуку проводиться перевірка та валідація результатів. Це включає аналіз отриманого звуку в різних точках приміщення після застосування визначених налаштувань. У разі необхідності можуть бути внесені корективи та вдосконалення налаштувань з метою досягнення оптимальних результатів.

3.3.1 Реалізація обчислювача

Створення формули для оптимального аналізу аудіо зразків та визначення оптимальних налаштувань для обробника звуку може бути складним завданням, вимагаючим глибоких знань та досліджень. Однак, з використанням новітніх технологій, ми можемо розглянути альтернативний підхід до цього завдання.

Один з можливих шляхів - це представити аналізатор звуку у вигляді тривіального нейрона. У такому випадку, кожен параметр налаштування аналізатора може бути розглянутий як окрема вага вхідного значення до нейрона.

Головне завдання полягає в створенні тренера для цього нейрона, який виступатиме в ролі аналізатора звуку. Цей тренер буде відповідальний за навчання нейрона на основі вхідних аудіо зразків і вибір оптимальних налаштувань обробника звуку.

Застосування нейромережевого підходу до аналізу звуку дозволяє зробити процес обробки звуку більш автоматизованим та ефективним. Він може пристосовуватись до різноманітних акустичних середовищ та надавати індивідуально підлаштовані параметри звукового обробника.

Для навчання обрано парадигму навчання з підкріпленням. Евристичною похибкою обрано середній квадрат відхилень між набором оброблених зразків та набором цільових зразків. Для вирішення задачі також можна було б обрати метод навчання з вчителем, але задача обробника не досягти ідеального збігу з цільовими зразками – це неможливо. Ціллю є наближення до значень цільових зразків, а саме зменшення різниці між вихідними та цільовими значеннями.

Вектор початкової зміні налаштувань обрано стандартний, а кожен наступний крок оцінюється похибка та доналаштовується швидкість зміни значення. Наразі аналіз неідеальний, та задовільний, залишається можливість розширити аналізатор у майбутньому для отримання додаткової інформації та покращення результатів. Кожен крок обчислюється похідна кожного параметру. Послідовність кроків аналізатора зображено на рисунку 3.17. Схему структури аналізатора як тренувальника нейромережі зображено на кресленику ІА91.220БАК.004 Д4.

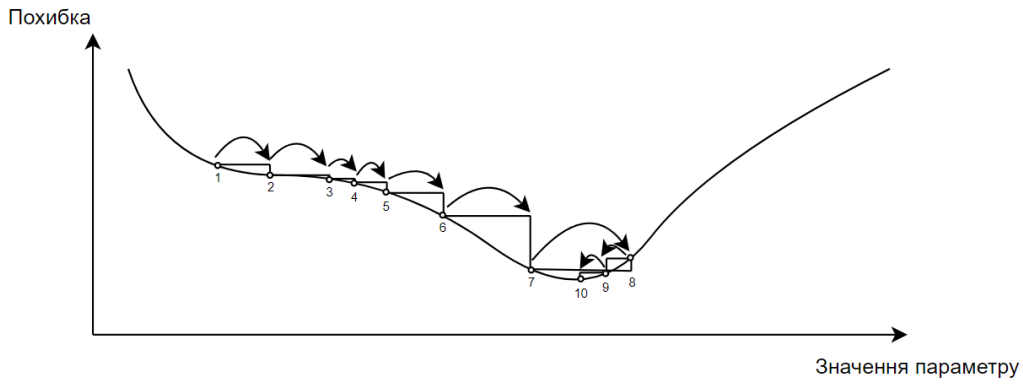

Рисунок 3.16 – Візуалізація роботи обробника звуку

4 ТЕСТУВАННЯ РОЗРОБЛЕНОГО ВЕБ-ЗАСТОСУНКУ

4.1 Тестування інтерфейсу обробника звуку

Розроблений користувацький інтерфейс зображений на рисунку 4.1.

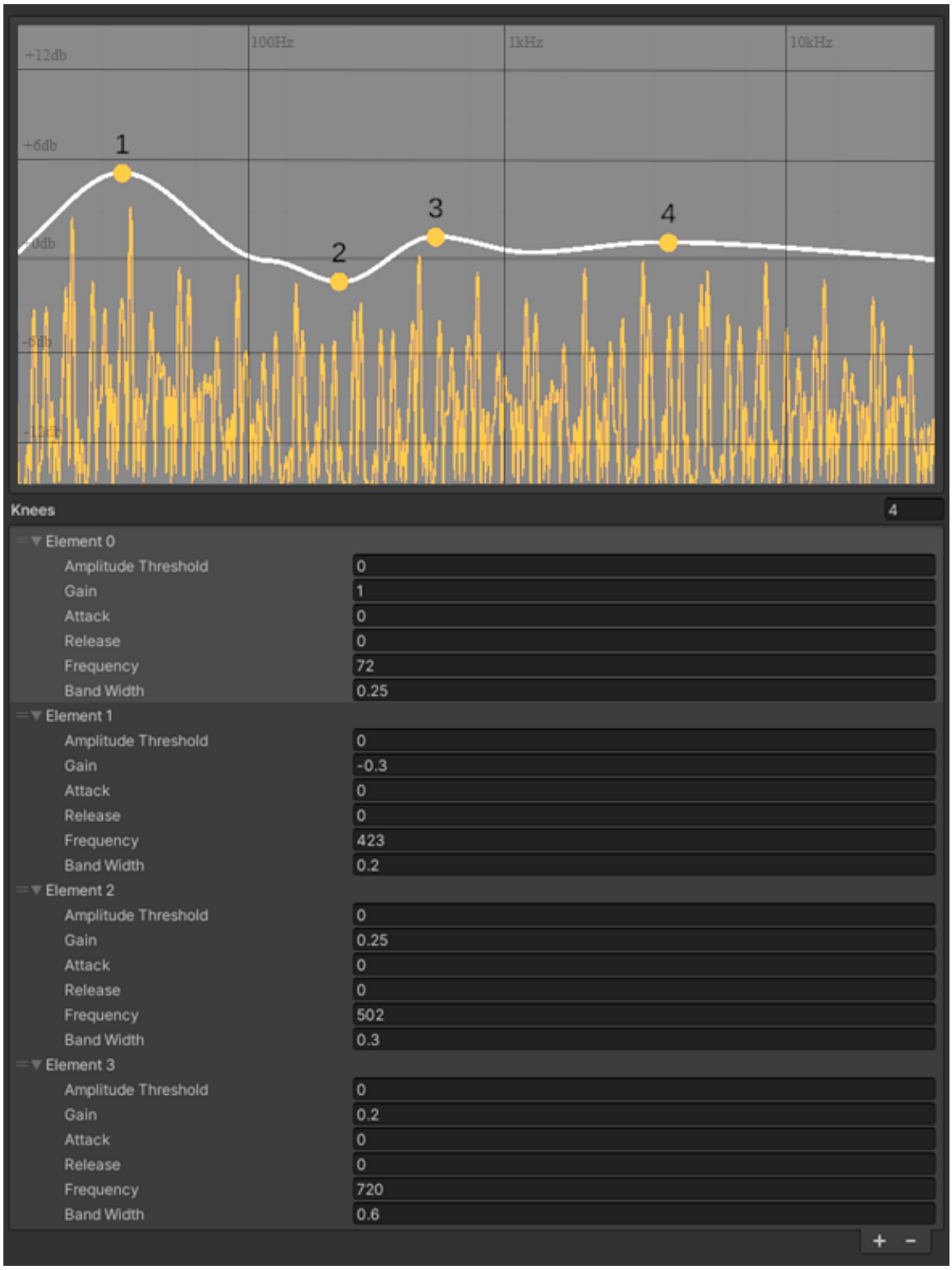

Рисунок 4.1 – Користуваціький інтерфейс обробника звуку

У роботі було розроблено інтерфейс, який повністю відповідає вимогам, що ставляться до користувацького інтерфейсу. Цей інтерфейс надає широкі можливості для редагування налаштувань точок контролю обробника, що значно збільшує його функціональність. Крім того, його особливістю є можливість відображення результатів обробки в реальному часі, що дозволяє користувачу миттєво спостерігати вплив внесених змін.

Крім того, інтерфейс відрізняється високою стабільністю та швидкодією. Він ефективно впоряється з обробкою великого обсягу даних та забезпечує миттєве відображення результатів навіть при високих навантаженнях. Це забезпечує незавершену продуктивність користувача та покращує його робочий досвід.

4.2 Тестування обробника звуку та аналізатора звуку

Тестування у мові Rust використовується для перевірки коректності програмного коду та забезпечення надійності програм. Вона базується на концепції модульних тестів і забезпечує можливість автоматизованого виконання тестів з метою перевірки правильності функцій, методів та інших частин коду.

Rust має вбудовану підтримку модульних тестів, що дозволяє визначати тести безпосередньо в коді програми. Тести можуть бути написані в спеціальних анотованих функціях з використанням макросів #[test] та assert_eq!. Наприклад:

```
#[test]
fn test_addition() {
    assert_eq!(2 + 2, 4);
}
```

Рисунок 4.2 – Приклад тесту мовою програмування Rust

У цьому прикладі визначено тест з назвою test_addition, який перевіряє, чи дорівнює сума 2 + 2 значенню 4. Макрос assert_eq! перевіряє, чи виконується рівність, і в разі порушення виводить повідомлення про помилку.

Для запуску тестів у мові Rust використовуються інструменти збірки проекту, такі як Cargo. Запуск тестів може бути виконаний за допомогою команди cargo test в кореневій папці проекту. Під час виконання тестів, Rust автоматично зібере й виконає всі функції з анотацією #[test], виведе результати їх виконання та повідомить про будь-які помилки, якщо такі є.

Можна також виконувати певні додаткові дії перед або після виконання тестів, використовуючи функції з анотацією #[before] та #[after] відповідно. Це дозволяє налаштовувати середовище перед тестами або виконувати певні дії після їх завершення.

Rust також підтримує поняття інтеграційних тестів, які дозволяють перевірити взаємодію різних компонентів або модулів програми. Інтеграційні тести можуть бути розташовані в окремих файлах та викликатися зовнішніми процесами, що дозволяє перевіряти поведінку програми в реальних умовах.

По результатам етапу тестування для обробника та аналізатора створено автоматизовані юніт-тести, що перевіряють правильність обробки невірних вхідних даних та коректність обробки звукових сигналів, а також валідують генерацію налаштувань на основі занадто гучних звуків, чи звуків знестачею гучності деяких частот. На рисунку 4.3 відображено результати тестування.

```
    Finished test [unoptimized + debuginfo] target(s) in 0.36s
     Running unittests src\lib.rs (target\debug\deps\comp_eq-bd8ac8c6ed586c60.exe)

running 11 tests
test test_add_3_amplitude_and_2_amplitude_5_amplitude ... ok
test test_attack_2s_increases_amplitude_from_bottom_to_top_for_2s ... ok
test test_bandwidth_half_main_frequency_amplitude_equals_1 ... ok
test test_bandwidth_less_than_0_throws ... ok
test test_bandwidth_more_than_1_throws ... ok
test test_fourier_transform_supports_only_powers_of_2 ... ok
test test_release_2s_decreases_amplitude_from_top_to_bottom_for_2s ... ok
test test_samples_sin_40hz_to_fft_returns_40hz_only_spectrum ... ok
test test_samples_to_fft_and_then_to_ifft_are_not_changed ... ok
test test_samples_with_amplitude_2_creates_half_gain_properties ... ok
test test_samples_with_half_mid_frequencies_creates_2_gain_and_between_03_06_bandwidth_properties ... ok

test result: ok. 11 passed; 0 failed; 0 ignored; 0 measured; 0 filtered out; finished in 0.00s
```

Рисунок 4.3 – Результати автоматичного тестування

4.3 Тестування інтерфейсу аналізатора звуку

У рамках даної дипломної роботи було розроблено інтерфейс, який повністю задовольняє вимоги, що ставляться до сучасного користувацького інтерфейсу. Основною метою розробки було створення простого та зручного інструменту, який надаватиме користувачам можливості для аналізу кількох записів одночасно. Користувацький інтерфейс зображено на рисунку 4.4.

Один із ключових аспектів інтерфейсу полягає в його простоті. Розроблене рішення забезпечує легку навігацію та зрозумілість усіх функцій, що доступні користувачу. Всі елементи інтерфейсу розташовані таким чином, щоб було зручно взаємодіяти з ними та здійснювати необхідні дії без зайвих зусиль.

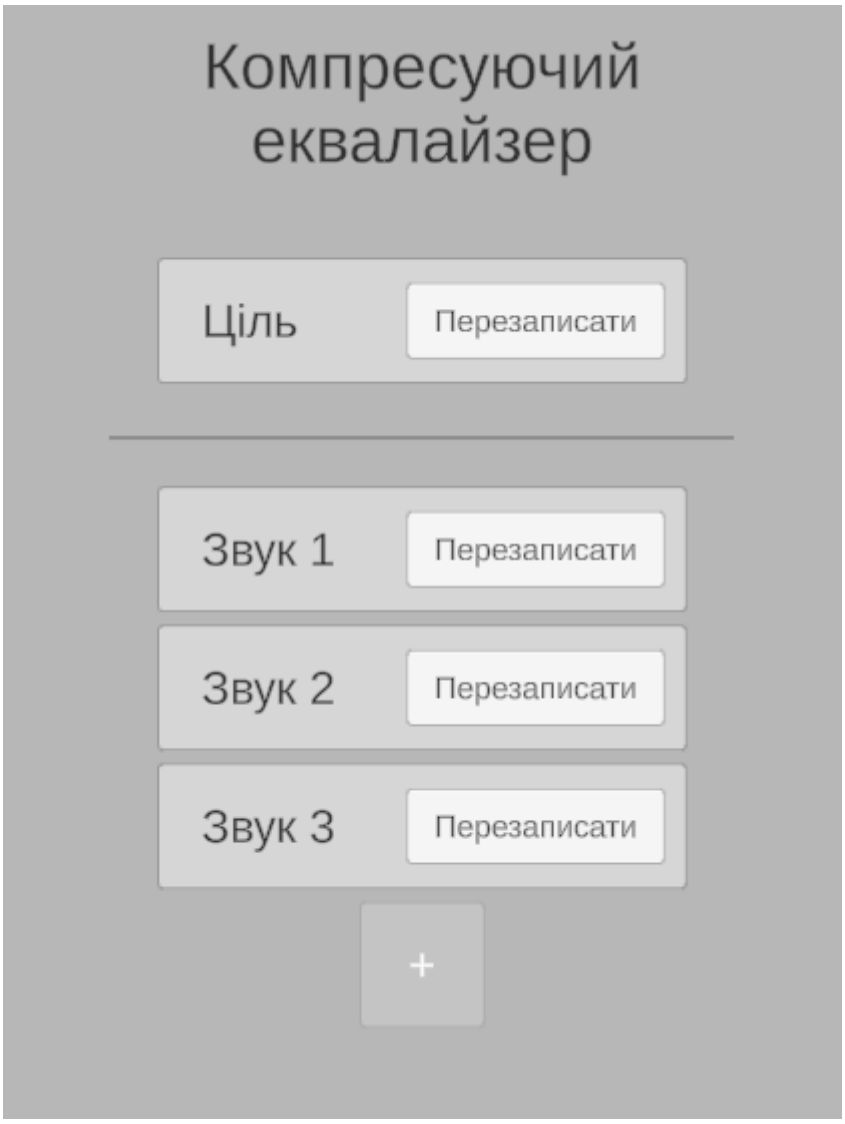

Рисунок 4.4 – Користувацький інтерфейс аналізатора звуку

Додатково, інтерфейс надає широкі можливості для аналізу кількох записів. Користувач може активувати одночасний аналіз кількох записів. Результатом розробки інтерфейсу став потужний інструмент, який дозволяє користувачам здійснювати аналіз кількох записів з великою ефективністю та точністю.

Висновки до розділу 4

У процесі дослідження був протестований програмний застосунок. Випробування врахували різні аспекти, включаючи його зручність та швидкодію. Після аналізу було оцінено, наскільки відповідає програмний застосунок вимогам. Далі були розроблені автоматичні тести для перевірки його роботи. За допомогою цих тестів перевірялася коректність функціонування програмного застосунку в різних сценаріях. Цей підхід дав можливість зробити більш повну оцінку якості роботи програмного застосунку і переконатися, що він задовольняє потреби користувачів і вимоги проекту.

В даному розділі було проведено дослідження розробленого додатку, включаючи огляд інтерфейсу користувача, розробку та перевірку функцій обчислювача. Було протестовано всі можливі сценарії роботи додатку. Тестування є важливим етапом у процесі розробки програмного забезпечення, оскільки допомагає виявити помилки та недоліки, а також гарантує високу якість продукту перед його впровадженням. Тестування було успішним, всі помилки були виправлені. Результати тестування також підтвердили, що інтерфейс додатку є зручним та інтуїтивно зрозумілим для користувачів, сприяючи його ефективному використанню.

На основі отриманих результатів тестування можна зробити висновок, що розроблена автоматизована система налаштування апаратури для приміщень відповідає встановленим вимогам та функціональним характеристикам. Вона надає користувачам зручний та надійний інструмент для легкого керування якістю відтворюваних звуків.

ВИСНОВКИ

Під час розробки системи автоматичного налаштування аудіо-апаратури під приміщення було виконано такі дії:

- було проаналізовано існуючі рішення, порівняно звукові редактори-секвенсори FL Studio, Ableton Live, Steinberg Cubase;

- було сформульовано проблему та визначено мету;

- було розглянуто технології розробки та обрано Rust в якості мови програмування обчислювача, C# в якості мови користувацького інтерфейсу, мову HLSL для створення складної візуалізації та рушій Unity для поєднання модулів у єдину комплексну систему;

- було розроблено модульну архітектуру, при розробці якої було враховано розширюваність системи, можливість портування системи на інші платформи та середовища, необхідність стабільної роботи системи;

- було проаналізовано технології обробки звуку та обрано найбільш придатні для задачі автоматичної обробки звуку;

- було проведено тестування системи. Створено автоматичні тести для обчислювальних маодулів системи.

Підсумком є створена система автоматичного налаштування апаратури під приміщення з можливістю доналаштування обробки. Метою роботи було вирішення проблеми складного і коштовного налаштування аудіо-апаратури. Розроблена система вирішує визначену проблему і тим самим задовольняє поставлену мету.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. ScienceDirect. Bandpass Filter. URL: https://www.sciencedirect.com/topics/engineering/bandpass-filter
2. WolfSound. Envelopes. URL: https://thewolfsound.com/envelopes/
3. AllAboutCircuits. What Is the Fourier Trasform. URL: https://www.allaboutcircuits.com/technical-articles/what-is-the-fourier-transform/
4. AllAboutCircuits. An Introduction to the Discrete Fourier Transform. URL: https://www.allaboutcircuits.com/technical-articles/an-introduction-to-the-discrete-fourier-transform/
5. wikiwand. Fast Fourier transform. URL: https://www.wikiwand.com/en/Fast_Fourier_transform
6. proximacentauri360. Fourier Transform. URL: https://proximacentauri360.wordpress.com/2012/08/15/fourier-transform/
7. StudFiles. Амплітудно-частотні характеристики підсилювача. URL: https://studfile.net/preview/9188217/page:5/
8. Image-Line. FL STUDIO. URL: https://www.image-line.com/
9. Splice. Ableton. URL: https://sounds.splice.com/ableton
10. steinberg. CUBASE. URL: https://www.steinberg.net/cubase/
11. Prossimo. What is memory safety and why does it matter? URL: https://www.memorysafety.org/docs/memory-safety/
12. Rust. URL: https://www.rust-lang.org/
13. Microsoft. Common Language Runtime (CLR) overview. URL: https://learn.microsoft.com/en-us/dotnet/standard/clr
14. Microsoft. Fundamentals of garbage collection. URL: https://learn.microsoft.com/en-us/dotnet/standard/garbage-collection/fundamentals
15. JavaTpoint. JVM (Java Virtual Machine) Architecture. URL: https://www.javatpoint.com/jvm-java-virtual-machine
16. Unity. URL: https://unity.com/
17. Microsoft. Unity. URL: https://dotnet.microsoft.com/en-us/apps/games/unity

18. Ableton.com. 24 Live Audio Effect Reference. URL: https://www.ableton.com/en/manual/live-audio-effect-reference/

19. td. Getting started with FFI: Rust & Unity. URL: https://blog.testdouble.com/posts/2018-01-02-unity-rust-ffi-getting-started/

20. Microsoft. High-level shader language (HLSL) . URL: https://learn.microsoft.com/en-us/windows/win32/direct3dhlsl/dx-graphics-hlsl

21. The Rust Programming Language. How to Write Tests. URL: https://doc.rust-lang.org/book/ch11-01-writing-tests.html