Минобрнауки России

Юго-Западный государственный университет

Кафедра программной инженерии

КУРСОВАЯ РАБОТА (ПРОЕКТ)

по дисциплине «Проектирование и архитектура программных систем»
наименование дисциплины

на тему Программа редактирования звуковых файлов

Направление подготовки (специальность) 09.03.04
(код, наименование)
Программная инженерия

Автор работы (проекта) А. Ю. Геворкян _____
(инициалы, фамилия) (подпись, дата)

Группа ПО-12б

Руководитель работы (проекта) А. А. Чаплыгин _____
(инициалы, фамилия) (подпись, дата)

Работа (проект) защищена _____
(дата)

Оценка _____

Члены комиссии _____ _____
подпись, дата фамилия и. о.

_____ _____
подпись, дата фамилия и. о.

_____ _____
подпись, дата фамилия и. о.

Курск 2024 г.

Минобрнауки России

Юго-Западный государственный университет

Кафедра программной инженерии

ЗАДАНИЕ НА КУРСОВУЮ РАБОТУ (ПРОЕКТ)

Студента Геворкяна А. Ю., шифр 21-06-0040, группа ПО-12б

1. Тема «Программа редактирования звуковых файлов» .

2. Срок предоставления работы к защите 2024 г.

3. Исходные данные для создания программной системы:

3.1. Перечень решаемых задач:

1) проанализировать способ представления аудио в цифровом виде;
2) разработать концептуальную модель системы;
3) спроектировать программную систему;
4) сконструировать и протестировать программную систему.

3.2. Входные данные и требуемые результаты для программы:

1) входными данными для программной системы являются: аудиофайлы формата WAV;

2) выходными данными для программной системы являются: аудиофайлы формата WAV.

4. Содержание работы (по разделам):

4.1. Введение

4.1. Анализ предметной области

4.2. Техническое задание: основание для разработки, назначение разработки, требования к программной системе, требования к оформлению документации.

4.3. Технический проект: общие сведения о программной системе, проект данных программной системы, проектирование архитектуры программной системы, проектирование пользовательского интерфейса программной системы.

4.4. Рабочий проект: спецификация компонентов и классов программной системы, тестирование программной системы, сборка компонентов программной системы.

4.5. Заключение

4.6. Список использованных источников

5. Перечень графического материала:

Руководитель работы (проекта)		А. А. Чаплыгин
	(подпись, дата)	(инициалы, фамилия)
Задание принял к исполнению		А. Ю. Геворкян
	(подпись, дата)	(инициалы, фамилия)

РЕФЕРАТ

Объем работы равен 73 страницам. Работа содержит 27 иллюстраций, 10 таблиц и 11 библиографических источников. Количество приложений – 1. Фрагменты исходного кода представлены в приложении А.

Перечень ключевых слов: система, программа, редактор, интерфейс, пользователь, звук, аудиофайл, Python.

Объектом разработки является программа, представляющая собой набор иструментов для совершения операций над WAV аудиофайлами.

В процессе создания программы были выделены основные сущности путем создания информационных блоков, использованы классы и методы модулей, обеспечивающие работу с сущностями предметной области, а также корректную работу программы.

При разработке программы использовался язык программирования Python.

ABSTRACT

The volume of work is 73 pages. The work contains 27 illustrations, 10 tables and 11 bibliographic sources. The number of applications is 1. The program code is presented in annex A.

List of keywords: system, programm, editor, interface, use, sound, audiofile, Python.

The object of development is a program that is a set of tools for performing operations on WAV audio files.

In the process of creating the program, the main entities were identified by creating information blocks, classes and methods of modules were used to ensure work with the entities of the subject area, as well as the correct operation of the program.

When developing the program, the Python programming language was used.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ 9
1 Анализ предметной области 11
1.1 Цифровое кодирование звуковой информации 11
1.2 Методы цифрового кодирования звуковой информации 12
2 Техническое задание 14
2.1 Основание для разработки 14
2.2 Цель и назначение разработки 14
2.3 Требования пользователя к интерфейсу программы 14
2.4 Моделирование вариантов использования 15
2.5 Нефункциональные требования к программной системе 16
2.5.1 Требования к надежности 16
2.5.2 Требования к программному обеспечению 17
2.5.3 Требования к аппаратному обеспечению 17
2.6 Требования к оформлению документации 17
3 Технический проект 18
3.1 Общая характеристика организации решения задачи 18
3.2 Обоснование выбора технологии проектирования 18
3.2.1 Python 18
3.2.2 TKinter 18
3.2.3 NumPy 19
3.2.4 PySDL 19
3.2.5 Ctypes 19
3.3 Диаграмма компонентов 19
3.4 Содержание информационных блоков. Основные сущности 20
4 Рабочий проект 25
4.1 Классы, используемые при разработке программы 25
4.2 Системное тестирование 28
4.2.1 Загрузка аудиофайла 28
4.2.2 Проигрываание аудио 29

4.2.3 Выделение области 30
4.2.4 Редактирование аудиоданных 31
4.2.5 Управление действиями 37
4.2.6 Сохранение аудиофайла 39
ЗАКЛЮЧЕНИЕ 41
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 41
ПРИЛОЖЕНИЕ А Фрагменты исходного кода программы 44

ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

WAV (Waveform Audio File Format) – формат файла-контейнера для хранения записи оцифрованного аудиопотока, подвид RIFF. Как правило, используется для хранения несжатого звука в импульсно-кодовой модуляции.

UML (Unified Modelling Language) – язык графического описания для объектного моделирования в области разработки программного обеспечения.

ВВЕДЕНИЕ

В современном мире аудиозаписи играют значительную роль в нашей повседневной жизни. От музыки, которую мы слушаем на наших устройствах, до голоса, который мы слышим на радио и телевидении, аудио является неотъемлемой частью нашего существования. В связи с этим возникает необходимость в инструментах для обработки и редактирования аудиофайлов. Аудиоредакторы – это программы, которые позволяют выполнять различные операции со звуковыми данными, такими как запись, редактирование, микширование, мастеринг и другие. Они предоставляют пользователям возможность работать со звуком на профессиональном уровне и создавать высококачественные аудиофайлы.

Актуальность аудиоредакторов обусловлена рядом причин. Во-первых, эти инструменты позволяют улучшить качество аудиозаписей, что важно для профессионального использования, например, в студиях звукозаписи или на радиостанциях. Во-вторых, они предоставляют возможность редактировать и микшировать аудиофайлы, что позволяет создавать уникальные звуковые эффекты и композиции. В-третьих, технологии кодирования и обработки звука обеспечивают более эффективное хранение данных, что делает их предпочтительным выбором для многих пользователей.

Таким образом, аудиоредакторы являются актуальными инструментами для работы со звуком, которые предоставляют широкие возможности для его обработки и улучшения. Они используются в различных областях, таких как звукозапись, радиовещание, создание видеоигр и других. С развитием технологий и появлением новых алгоритмов кодирования звука эти инструменты становятся все более мощными и функциональными, что делает их еще более актуальными для современных пользователей.

Цель настоящей работы – разработка аудиоредактора для преобразований WAV аудиофайлов. Для достижения поставленной цели необходимо решить *следующие задачи:*

- провести анализ предметной области;

- разработать и спроектировать концептуальную модель программы;
- реализовать программу средствами языка Python.

Структура и объем работы. Отчет состоит из введения, 4 разделов основной части, заключения, списка использованных источников, 1 приложения.

Во введении сформулирована цель работы, поставлены задачи разработки, описана структура работы, приведено краткое содержание каждого из разделов.

В первом разделе на стадии описания технической характеристики предметной области приводится сбор информации о предметной области.

Во втором разделе на стадии технического задания приводятся требования к разрабатываемой программе.

В третьем разделе на стадии технического проектирования представлены проектные решения для программы.

В четвертом разделе приводится список классов и их методов, использованных при разработке прогораммы, производится тестирование разработанной программы.

В заключении излагаются основные результаты работы, полученные в ходе разработки.

В приложении А представлен графический материал. В приложении Б представлены фрагменты исходного кода.

1 Анализ предметной области

1.1 Цифровое кодирование звуковой информации

Цифровое кодирование звуковой информации – это процесс преобразования аналогового звукового сигнала в цифровую форму, состоящую из последовательности дискретных значений. Цифровое кодирование позволяет представить звуковую информацию в виде чисел, которые могут быть обработаны и переданы с помощью компьютерных систем.

Процесс цифрового кодирования звуковой информации включает в себя несколько этапов:

Дискретизация – это процесс разбиения аналогового звукового сигнала на равные временные интервалы, называемые отсчетами. В каждом отсчете записывается значение амплитуды звукового сигнала в определенный момент времени. Частота дискретизации определяет количество отсчетов, записываемых в секунду, и измеряется в герцах (Гц). Чем выше частота дискретизации, тем более точно представлен звуковой сигнал, но и требуется больше памяти для хранения данных.

Квантование – это процесс преобразования амплитуды звукового сигнала в дискретные уровни. Каждый отсчет амплитуды округляется до ближайшего значения из заданного набора уровней. Число уровней квантования определяет разрешающую способность кодирования и измеряется в битах. Чем больше число уровней квантования, тем более точно представлены амплитуды звукового сигнала, но и требуется больше памяти для хранения данных.

Кодирование – это процесс преобразования дискретных значений амплитуды звукового сигнала в цифровой код. Каждому значению амплитуды сопоставляется определенный код, который может быть представлен в виде битовой последовательности. Различные методы кодирования могут использоваться для оптимизации использования памяти и улучшения качества звука.

Цифровое кодирование звуковой информации имеет ряд преимуществ по сравнению с аналоговым кодированием. Оно позволяет более эффективно использовать память и пропускную способность при хранении и передаче звуковой информации. Кроме того, цифровое кодирование обеспечивает более стабильное и надежное воспроизведение звука, так как цифровые данные менее подвержены искажениям и шумам.

1.2 Методы цифрового кодирования звуковой информации

Пульс-кодовая модуляция (PCM) является одним из наиболее распространенных методов цифрового кодирования звука. Он основан на дискретизации аналогового сигнала и его последующем квантовании. В процессе дискретизации звуковой сигнал разбивается на небольшие отрезки времени, называемые сэмплами. Затем каждый сэмпл аналогового сигнала преобразуется в цифровое значение, которое представляет амплитуду сигнала в данном моменте времени. Эти цифровые значения называются кодами.

Адаптивное дельта-модуляция (ADM) является методом цифрового кодирования звука, который основан на изменении амплитуды сигнала относительно предыдущего значения. Вместо кодирования каждого сэмпла отдельно, ADM кодирует только разницу между текущим и предыдущим значением сигнала. Это позволяет сократить объем передаваемых данных и уменьшить требования к пропускной способности.

Адаптивное предиктивное кодирование (APC) является методом цифрового кодирования звука, который основан на предсказании следующего значения сигнала на основе предыдущих значений. Вместо кодирования каждого сэмпла отдельно, APC кодирует только разницу между предсказанным значением и фактическим значением сигнала. Это позволяет сократить объем передаваемых данных и уменьшить требования к пропускной способности.

Кодирование по Гауссу (ADPCM) является методом цифрового кодирования звука, который основан на адаптивном предиктивном кодировании и квантовании ошибки предсказания. Вместо кодирования разницы между предсказанным и фактическим значением сигнала, ADPCM кодирует раз-

ницу между предсказанным значением и квантованным значением ошибки предсказания. Это позволяет более эффективно использовать пропускную способность и улучшить качество звука.

Кодирование по Фурье (MP3) является методом цифрового кодирования звука, который основан на преобразовании Фурье. Вместо кодирования амплитуды сигнала в каждом сэмпле, MP3 кодирует спектральные коэффициенты, которые представляют различные частотные компоненты сигнала. Это позволяет сократить объем передаваемых данных и сохранить высокое качество звука при сжатии.

Каждый из этих методов имеет свои преимущества и недостатки, и выбор метода зависит от конкретных требований и ограничений приложения. Например, PCM обеспечивает наивысшее качество звука, но требует большей пропускной способности, в то время как MP3 обеспечивает хорошее качество звука при сжатии, но может иметь некоторые потери в качестве.

2 Техническое задание

2.1 Основание для разработки

Основанием для разработки является задание на курсовую работу «Программа редактирования звуковых файлов».

2.2 Цель и назначение разработки

Задачами данной разработки являются:

- осуществление загрузки аудиофайла и извлеения аудиоданных;
- реализация отображения аудиоданных посредством аудиодорожки;
- создание способов обработки аудиоданных;
- осуществление сохранения изменений и запись аудиофайла.

2.3 Требования пользователя к интерфейсу программы

Программа должна включать в себя:

- возможность выбора аудиофайла;
- визуализацию аудиофайла;
- набор инструментов для манипуляции аудиофайлом;
- возможность сохранения аудиофайла;

Композиция программы представлена на рисунке 2.1.

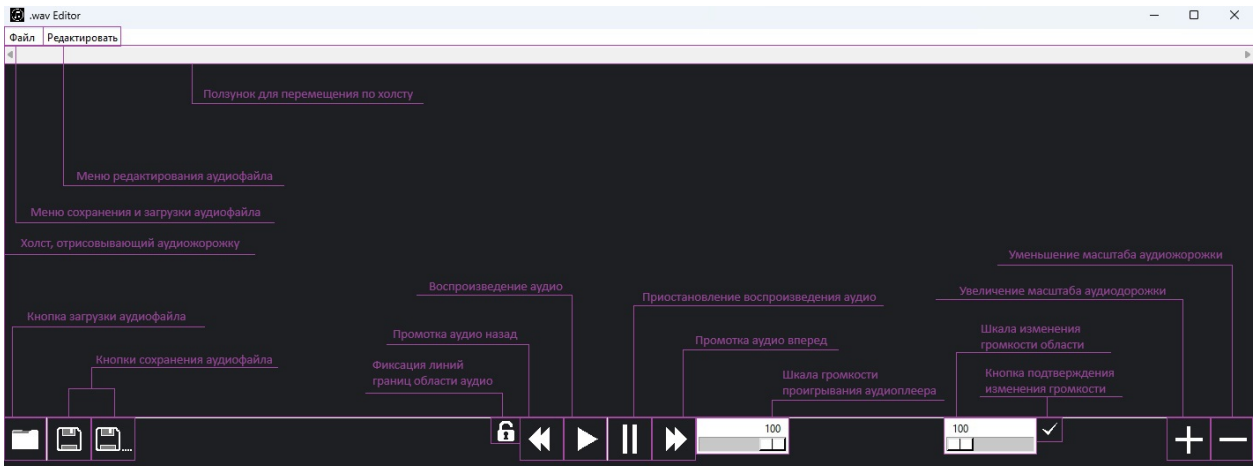

Рисунок 2.1 – Композиция интерфейса программы

2.4 Моделирование вариантов использования

Для разрабатываемой программы была реализована модель, которая обеспечивает наглядное представление вариантов использования аудиоредактора.

Она помогает в физической разработке и детальном анализе взаимосвязей объектов. При построении диаграммы вариантов использования применяется унифицированный язык визуального моделирования UML.

Диаграмма прецедентов (рис. 2.2) описывает функциональное назначение разрабатываемой программы. То есть это то, что программа будет непосредственно делать в процессе своего функционирования. Проектируемая программа представляется в виде ряда прецедентов, предоставляемых системой актерам или сущностям, которые взаимодействуют с ней. Актером или действующим лицом является сущность, взаимодействующая с программой извне. Прецедент служит для описания набора действий, которые программа предоставляет актеру.

На основании анализа предметной области в программе должны быть реализованы следующие прецеденты работы с аудиофайлом, все прецеденты осуществляются посредством взаимодействия с компонентами интерфейса:

1. Загрузка - возможность выбора аудиофайла для последующей визализации и редактирования.

2. Визуализация - отображение аудиоданных в виде аудиодорожки с возможностью увеличения и уменьшения масштаба, а также выбора необходимой области.

3. Прослушивание - возможность воспроизведения и остановки прослушивания аудио, также как и перемотки на определенное количество секунд вперед/назад.

4. Редактирование - наличие набора инструментов для редактирования аудиоданных, включающего в себя такие операции, как копирование, удаление, вставку, замену, обнуление, добавление эффекта нарастания и затухания и изменение громкости.

5. Сохранение - возможность записи измененных аудиоданных в WAV файл, как новый, так и оригинальный.

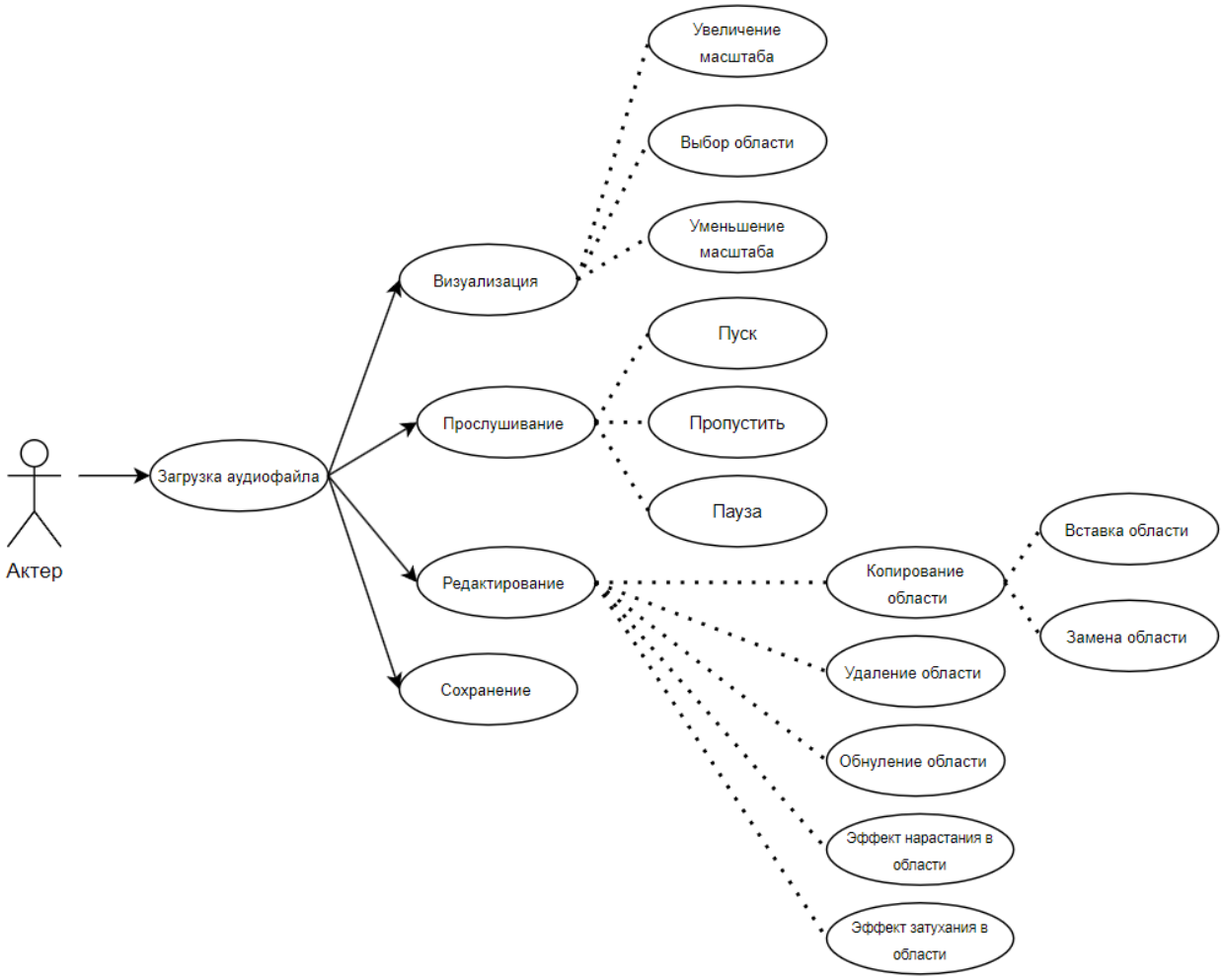

Рисунок 2.2 – Диаграмма прецедентов

2.5 Нефункциональные требования к программной системе

2.5.1 Требования к надежности

В связи с тем, что работа в программе ведется не с самим файлом непосредственно, а с буфером данных, получаемых из него, то даже при удалении исходного аудиофайла все данные о нем будут сохранены в случае, если они уже были загружены в программу.

2.5.2 Требования к программному обеспечению

Для реализации программы должен быть использован язык Python, а также связанные с ним библиотеки: Tkinter, PySDL, NumPy и Ctypes. Для корректной работы программы должен быть установлен Python версии не ниже 3.11.

2.5.3 Требования к аппаратному обеспечению

Для корректной работоспособности программы необходим процессор с 2 и более ядрами. Объем оперативной памяти должен быть не менее 512 МБ.

2.6 Требования к оформлению документации

Разработка программной документации и программного изделия должна производиться согласно ГОСТ 19.102-77 и ГОСТ 34.601-90. Единая система программной документации.

3 Технический проект

3.1 Общая характеристика организации решения задачи

Необходимо спроектировать и разработать компьютерную программу, позволяющую редактировать WAV аудиофайлы.

Компьютерная программа представляет собой комбинацию компьютерных инструкций и данных, позволяющую аппаратному обеспечению вычислительной системы выполнять вычисления или функции управления.

3.2 Обоснование выбора технологии проектирования

На сегодняшний день информационный рынок, поставляющий программные решения в выбранной сфере, предлагает множество продуктов, позволяющих достигнуть поставленной цели – разработки компьютерной программы для работы с аудио.

3.2.1 Python

Python — высокоуровневый язык программирования общего назначения с динамической строгой типизацией и автоматическим управлением памятью, ориентированный на повышение производительности разработчика, читаемости кода и его качества, а также на обеспечение переносимости написанных на нём программ.

3.2.2 TKinter

Tkinter – кросс—платформенная событийно—ориентированная графическая Python—библиотека на основе средств Tk, написанная Стином Лумхольтом и Гвидо ван Россумом. Входит в стандартную библиотеку Python и предназначена для разработки графического интерфейса.

3.2.3 NumPy

NumPy – открытая бесплатная Python—библиотека для работы с многомерными массивами, чаще всего используемая в анализе данных и обучении нейронных сетей.

3.2.4 PySDL

Python Simple DirectMedia Layer (PySDL) – открытая бесплатная кроссплатформенная мультимедийная Python—библиотека, реализующая единый программный интерфейс к графической подсистеме, звуковым устройствам и средствам ввода для широкого спектра платформ. Данная библиотека активно используется при написании кроссплатформенных мультимедийных программ.

3.2.5 Ctypes

Ctypes – Python—библиотека внешних функций, представляющая собой C–совместимые типы данных и позволяющая вызывать функции из DLL или разделяемых библиотек. Её можно использовать для оборачивания этих библиотек в чистый Python.

3.3 Диаграмма компонентов

Диаграмма компонентов описывает особенности физического представления разрабатываемой системы. Она позволяет определить архитектуру системы, установив зависимости между программными компонентами, в роли которых может выступать как исходный, так и исполняемый код. На рисунке 3.1 изображена диаграмма компонентов для проектируемой системы.

Основным исполняемым файлов является файл main.py, объединяющий в себе все другие компоненты. При запуске происходит создание графичексого интерфейса, посредством которого пользователь может взаимодействовать с программой.

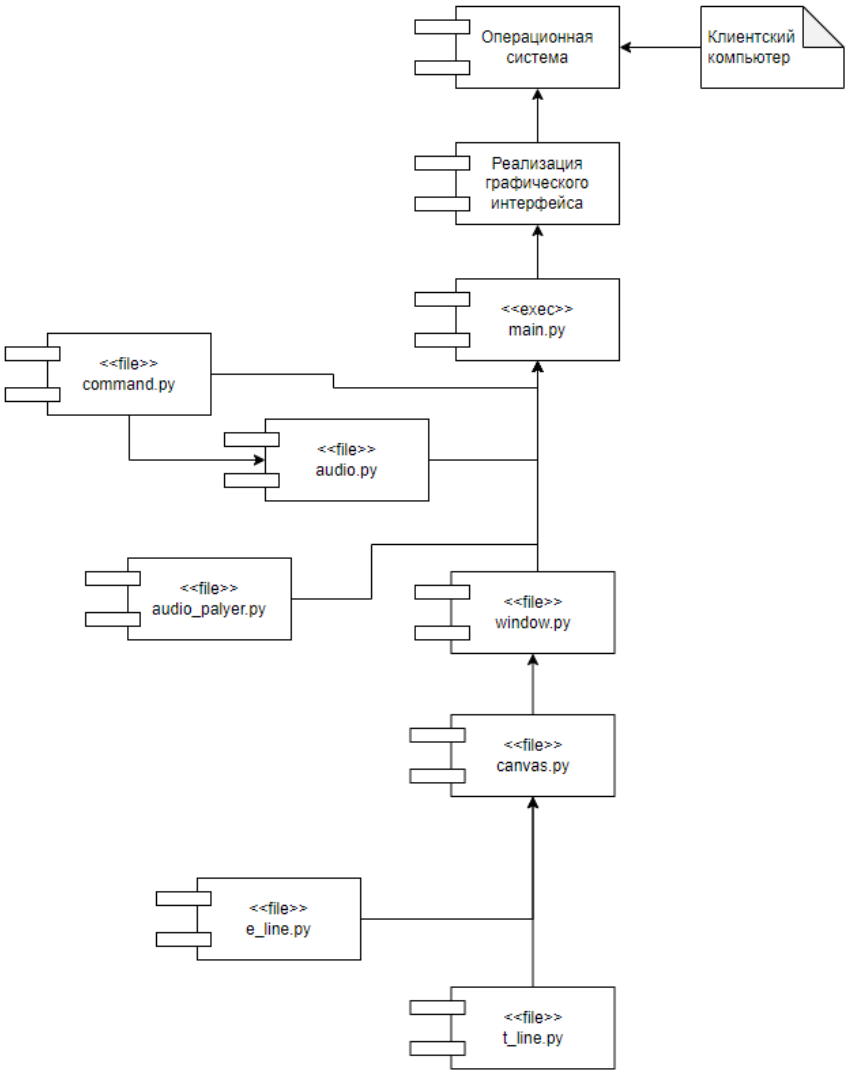

Рисунок 3.1 – Диаграмма компонентов

3.4 Содержание информационных блоков. Основные сущности

Проанализировав требования, можно выделить 5 основных сущностей:

- «Аудио»;
- «Аудиоплеер»;
- «Окно»;
- «Холст»;
- «Команда».

Ниже представлены таблицы составов перечисленных выше сущностей.

Таблица 3.1 – Атрибуты сущности «Аудио»

Поле	Тип	Начальное значение	Описание
1	2	3	4
nchannels	Integer	0	Количество каналов
sampwidth	Integer	0	Размер сэмплов
framerate	Integer	0	Частота сэмплов
nframes	Integer	0	Количество сэмплов
comptype	String	NONE	Тип сжатия
compname	String	not compressed	Название сжатия
chunksize	Integer	0	Размер буфера за ед. времени
copy buffer	Numpy. array	Numpy.array([], dtype=int16)	Буфер для хранения скопированной области
signals data	Numpy. array	Numpy.array([], dtype=int16)	Буфер для хранения прочитанных сэмплов
duration	Float	0.0	Длительность

Таблица 3.3 – Атрибуты сущности «Аудиоплеер»

Поле	Тип	Начальное значение	Описание
1	2	3	4
audio	Audio	Audio(kwargs)	Объект класса Audio
bactive	Bool	False	Флаг для проверки активности
bplaying	Bool	False	Флаг для проверки проигрывания

Продолжение таблицы 3.3

1	2	3	4
chunk	Bytes	Пустрокая строка байтов	Проигрываемый аудио-буфер
volume	Integer	100	Громкость
start	Float	0.0	Время начала выбранной области
start index	Integer	0	Индекс начала выбранной области в буфере
end	Float	0.0	Время конца выбранной области
end index	Integer	0	Индекс конца выбранной области в буфере
current	Float	0.0	Время проигрывания

Таблица 3.5 – Атрибуты сущности «Окно»

Поле	Тип	Начальное значение	Описание
1	2	3	4
scrollbar	Scrollbar	Scrollbar (kwargs)	Ползунок для просмотра дорожки
canvas	Canvas	Canvas(kwargs)	Вспомагательный холст
scrollable frame	Frame	Frame(kwargs)	Вспомагательная прокручиваемая рамка
scrollable canvas	Scrollable Canvas()	Scrollable Canvas(kwargs)	Прокручиваемый холст
frame	Frame	Frame(kwargs)	Главная рамка

Таблица 3.7 – Атрибуты сущности «Холст»

Поле	Тип	Начальное значение	Описание
1	2	3	4
loaded	Bool	False	Флаг для проверки загрузки холста
player	AudioPla-yer	AudioPlayer(kwargs)	Объект класса AudioPlayer
signals	List	[]	Группированные значения амплитуд
signals count	Integer	1	Количество линий холста
pixels per second	Integer	12	Количество пикселей, необ-ходимое для отображения 1 секунды звука
bdrawing	Bool	False	Флаг для проверки отрисовки
block	Bool	False	Флаг для блокировки границ
s line	EdgeLine	EdgeLine(kwargs)	Граница начала области
e line	EdgeLine	EdgeLine(kwargs)	Граница конца области
t line	TimeLine	TimeLine(kwargs)	Линия времени
labels	List	[0, [Label()]]	Временные отметки

Таблица 3.9 – Атрибуты сущности «Команда»

Поле	Тип	Начальное значение	Описание
1	2	3	4
audio	Audio()	Audio(kwargs)	Объект класса Audio

Продолжение таблицы 3.9

1	2	3	4
start index	Integer	Задается при вызове конструктора	Индекс начала выбранной области в буфере
end index	Integer	Задается при вызове конструктора	Индекс конца выбранной области в буфере
buffer	Numpy. array()	[]	Неизмененный фрагмент аудио

В системе предусмотрен внутренний механизм связи между разделами и элементами информационных блоков, поэтому введения дополнительных идентификаторов при реализации связей между сущностями не предполагается.

Экземпляры сущностей реализуются в информационных блоках посредством элементов, атрибуты сущности – посредством полей и свойств элемента.

4 Рабочий проект

4.1 Классы, используемые при разработке программы

Можно выделить следующий список классов и их методов, использованных при разработке программы (таблица 4.1).

Таблица 4.1 – Описание классов, используемых в приложении

Название класса	Модуль, к которому относится класс	Описание класса	Методы
1	2	3	4
Window	window.py	Класс графического интерфеса программы	Дополнительные методы отсутствуют
Audio	audio.py	Класс для работы с аудиоданными	copy audio(start index, end index) - копирование области в буфер
Canvas	canvas.py	Класс для визуализации аудиоданных	change lines position(start, end, xstart, xend, bplay) - смена позиций границ области, set line(event, bend) - перемещение границ области аудио по нажатию, draw(bstart=False, bend=False, btime=False) - отрисовка аудиодорожки, change scale(badd) - смена масштаба, reset lines(bstart, bend, btime) - сброс позиций границ области
EdgeLine	eline.py	Класс границы области аудио	change position(x) - смена позиции

Продолжение таблицы 4.1

1	2	3	4
TimeLine	tline.py	Класс линии времени	change position(x) - смена позиции, movement(event) - событие перемещения
TimeLineLabel	tline.py	Класс надписи линии времени	set(x) - смена текста и позиции
AudioPlayer	audioplayer.py	Класс для загрузки, воспроизведения и сохранения аудио	open player() - инициализация плеера SDL, closr player() - закрытие плеера SDL, pause audio(event) - прекращение воспроизведения, resume audio(event) - восстановление воспроизведения, set time borders(start, end, bload, bplay) - установка границ области, set time(start, end, bplay) - установка линии времени, load file(chunksize=2048) - загрузка аудиофайла, save file() - сохранение аудиофайла, save file where() - сохранение аудиофайла с выбором имени и расположения, load chunk(chunk, start idnex, end index, play, loops) - загрузка аудиоданных в буфер для воспроизведения, play chunk(chunk) - воспроизведения из буфера аудиоданных

Продолжение таблицы 4.1

1	2	3	4
CommandBuffer	command.py	Класс буфера совершенных над аудиоданными операций	clean() - очищение буфера, move() - перемещение команд в буфере во избежание переполнения
Command	command.py	Класс команды редактирования аудио	do() - совершение операции, undo() - отмена операции
CutCommand	audio.py	Наследник класса Command; представляет собой обработку удаления области аудио	do() - совершение операции, undo() - отмена операции
PasteCommand	audio.py	Наследник класса Command; представляет собой обработку замены и вставки области в аудио	do() - совершение операции, undo() - отмена операции
NullifyCommand	audio.py	Наследник класса Command; представляет собой обработку обнуления области аудио	do() - совершение операции, undo() - отмена операции
FadeCommand	audio.py	Наследник класса Command; представляет собой обработку эффекта нарастания и затухания области аудио	do() - совершение операции, undo() - отмена операции

VolumeCommand	audio.py	Наследник класса Command; представляет собой обработку громкости аудио	do() - совершение операции, undo() - отмена операции

4.2 Системное тестирование

Для отладки программы были разработаны следующие тестовые наборы:

4.2.1 Загрузка аудиофайла

Предусловие: програма запущена.

Тестовый случай: загрузка аудиофайла.

Ожидаемый результат: корректная визуализация аудиоданных.

Результат представлен на рисунках 4.1 и 4.2.

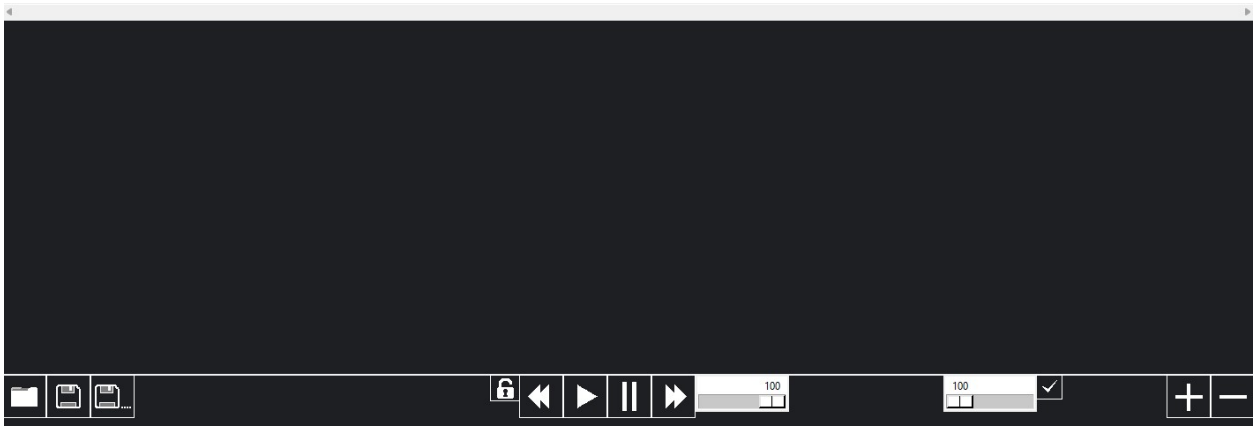

Рисунок 4.1 – Аудиодорожка до загрузки аудиофайла

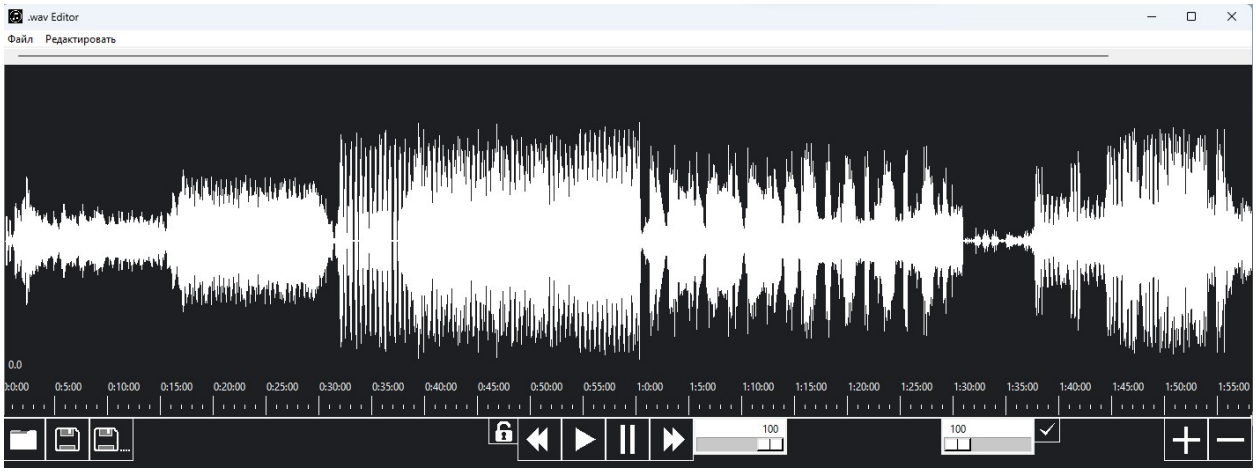

Рисунок 4.2 – Аудиодорожка после загрузки аудиофайла

4.2.2 Проигрываание аудио

Предусловие: програма запущена, аудиофайл загружен.

Тестовый случай: проигрывание аудио.

Ожидаемый результат: корректное воспроизведение аудиоданных.

Результат представлен на рисунках 4.3 и 4.4.

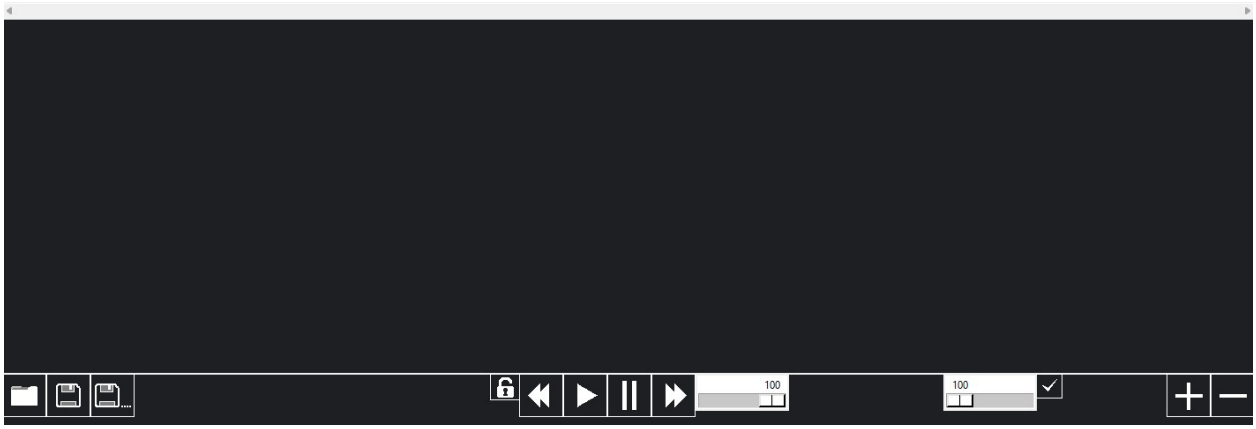

Рисунок 4.3 – Аудиодорожка до воспроизведения аудиофайла

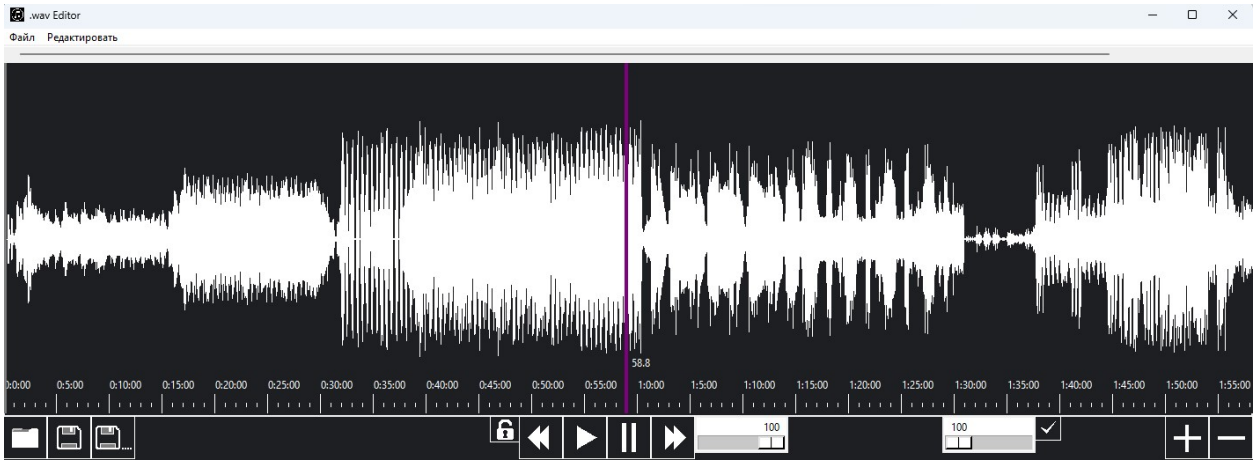

Рисунок 4.4 – Аудиодорожка после воспроизведения аудиофайла

4.2.3 Выделение области

Предусловие: програма запущена, аудиофайл загружен.

Тестовый случай: выделение области аудио.

Ожидаемый результат: корректное выделение области.

Результат представлен на рисунках 4.5 и 4.6.

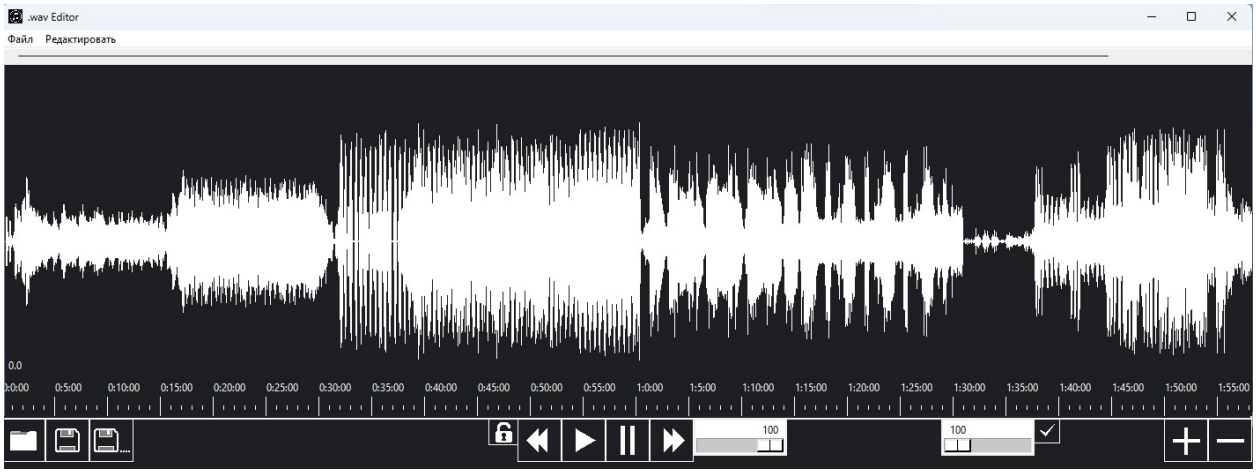

Рисунок 4.5 – Аудиодорожка до выбора области

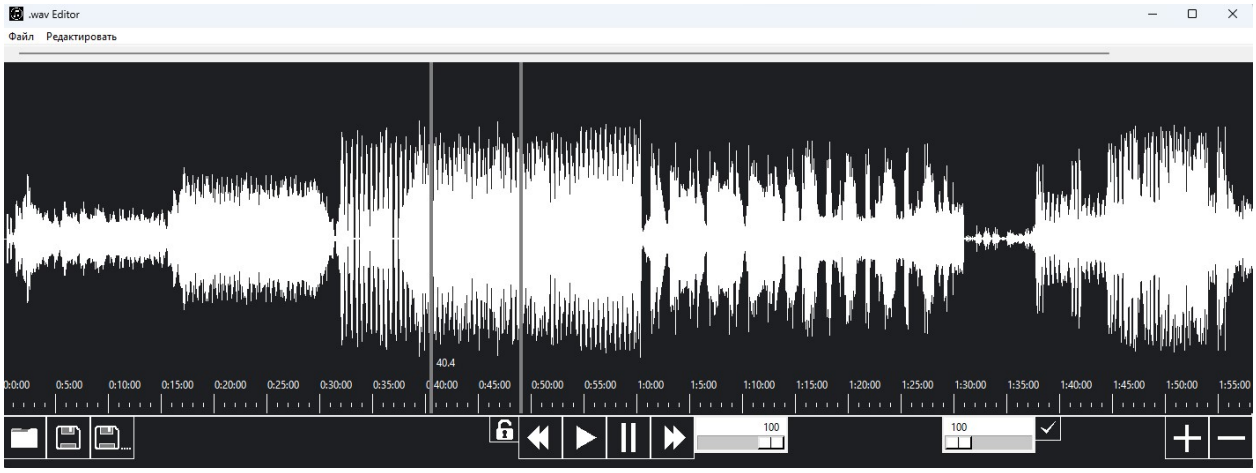

Рисунок 4.6 – Аудиодорожка после выбора области

4.2.4 Редактирование аудиоданных

Предусловие: програма запущена, аудиофайл загружен, выбрана необходимая область.

Тестовый случай: удаление области.

Ожидаемый результат: корректное удаление области.

Результат представлен на рисунках 4.7 и 4.8.

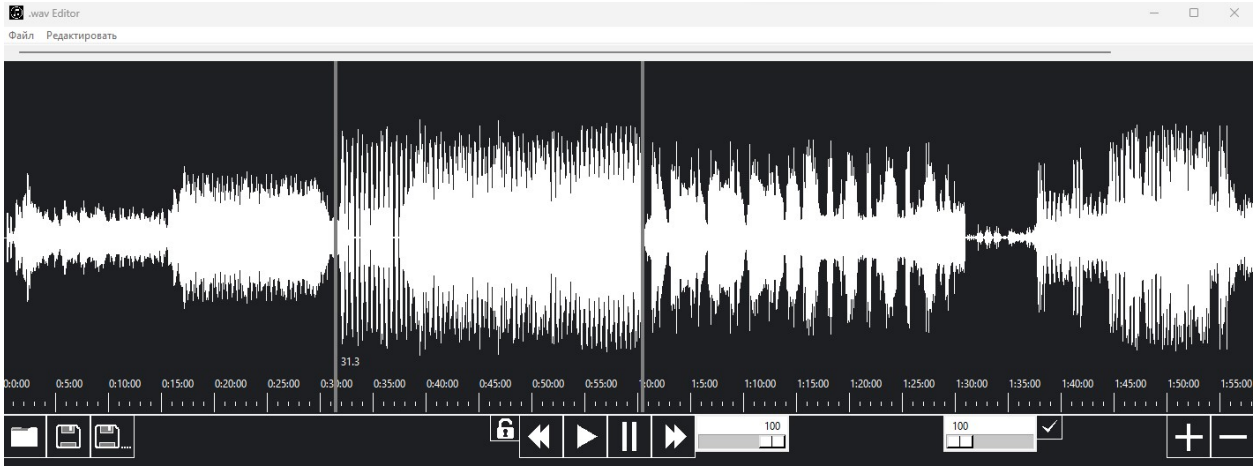

Рисунок 4.7 – Аудиодорожка до удаления области

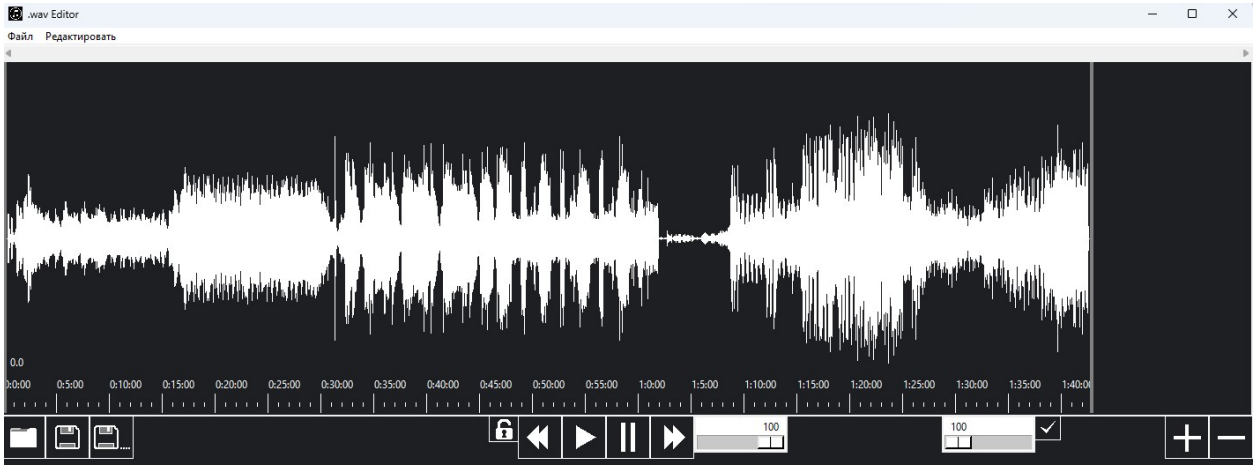

Рисунок 4.8 – Аудиодорожка после удаления области

Предусловие: програма запущена, аудиофайл загружен, скопирована вставляемая область.

Тестовый случай: вставка области.

Ожидаемый результат: корректная вставка области.

Результат представлен на рисунках 4.9 и 4.10.

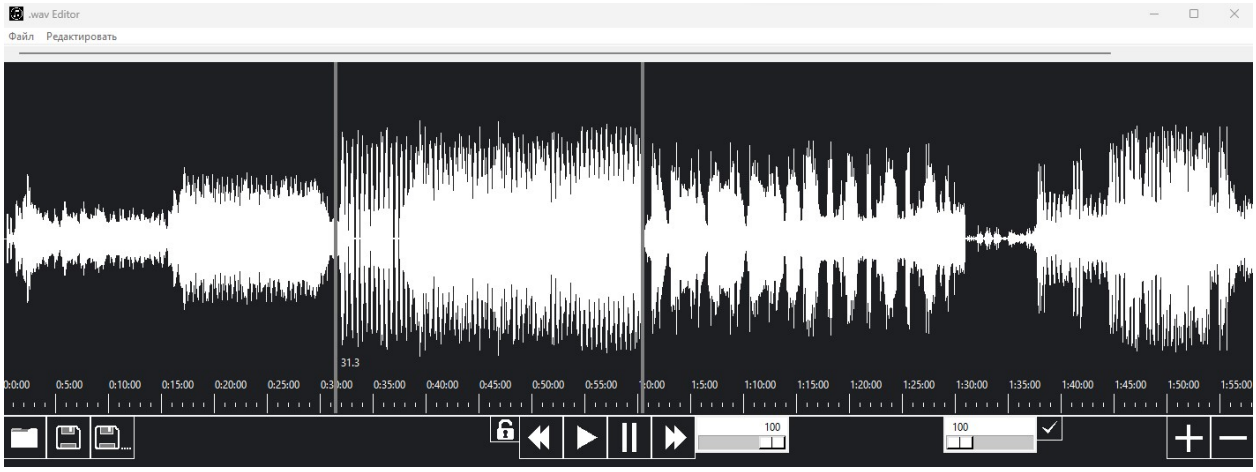

Рисунок 4.9 – Аудиодорожка до вставки области

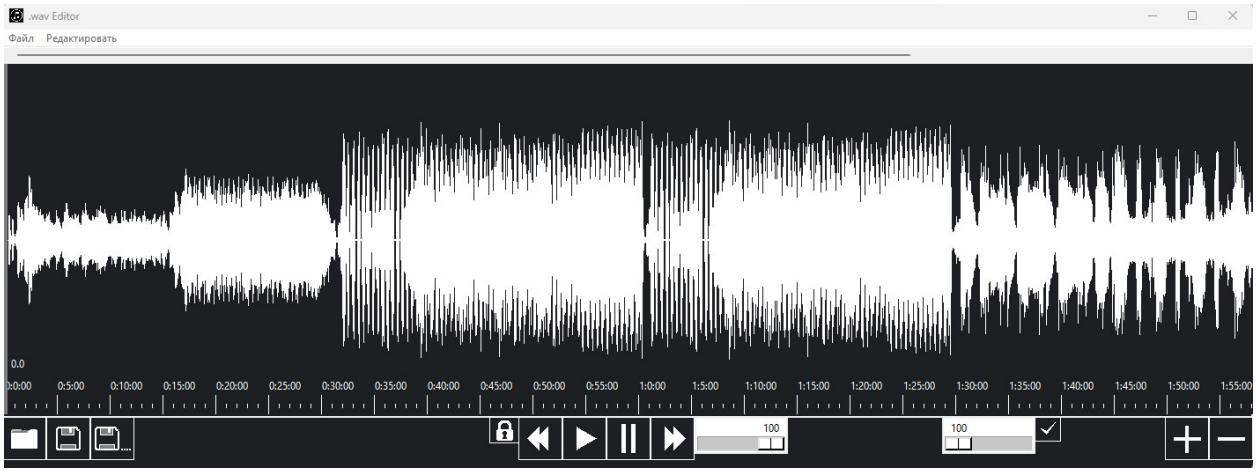

Рисунок 4.10 – Аудиодорожка после вставки области

Предусловие: програма запущена, аудиофайл загружен, скопирована замещающая и выбрана заменяемая области.

Тестовый случай: вставка области.

Ожидаемый результат: корректная замена области.

Результат представлен на рисунках 4.11 и 4.12.

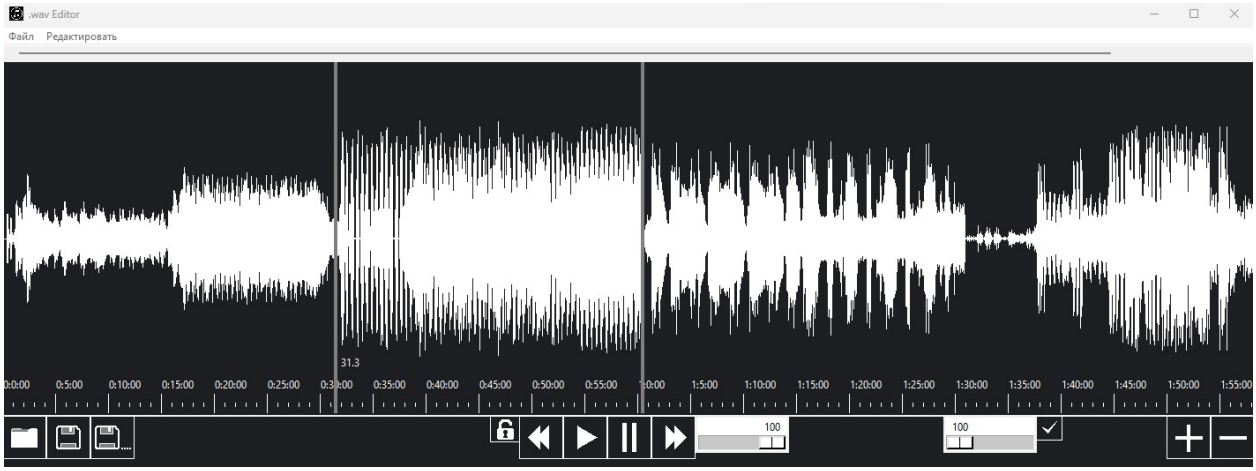

Рисунок 4.11 – Аудиодорожка до замены области

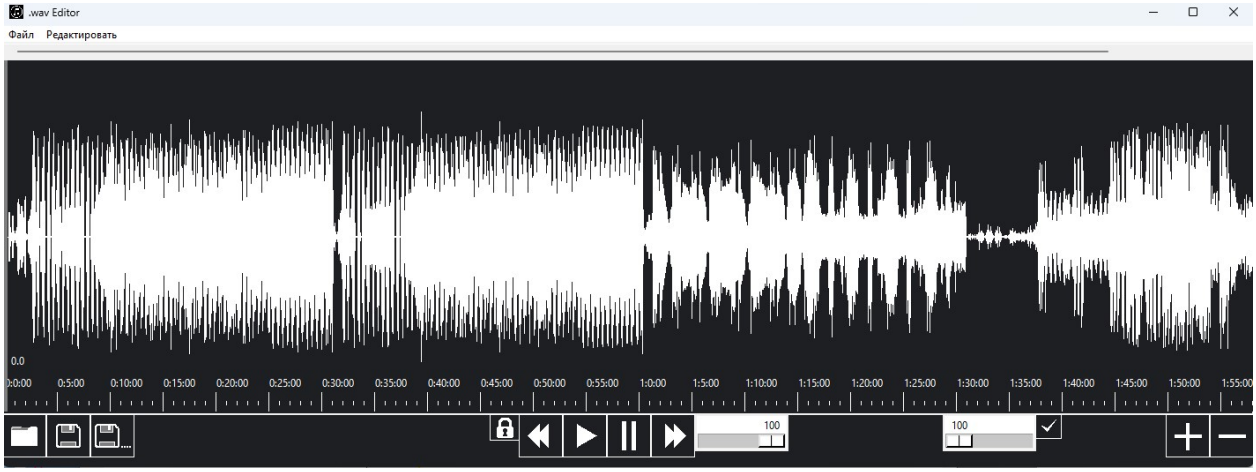

Рисунок 4.12 – Аудиодорожка после замены области

Предусловие: програма запущена, аудиофайл загружен, выбрана необходимая область.

Тестовый случай: обнуление области.

Ожидаемый результат: корректное обнуление области.

Результат представлен на рисунках 4.13 и 4.14.

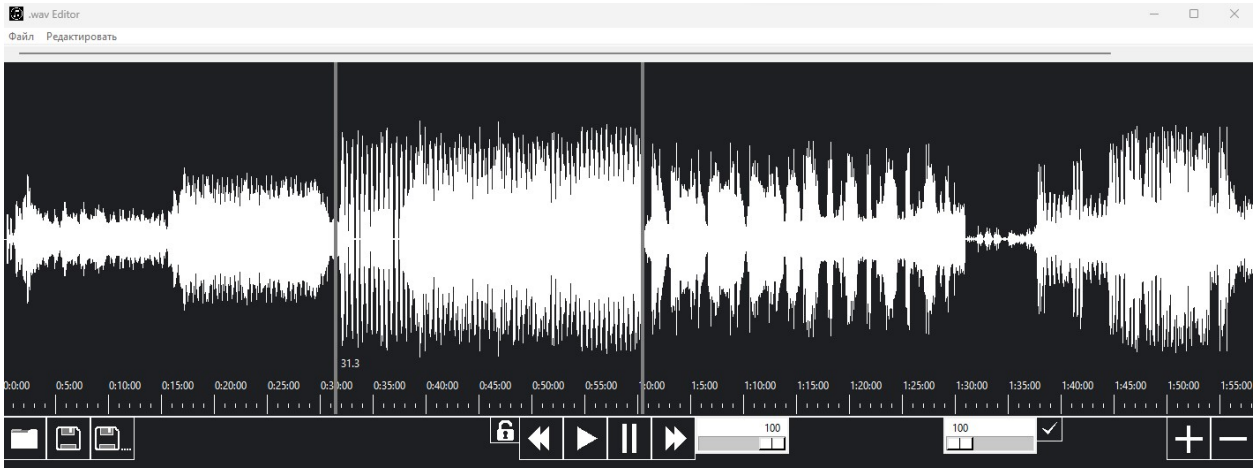

Рисунок 4.13 – Аудиодорожка до обнуления области

Рисунок 4.14 – Аудиодорожка после обнуления области

Предусловие: програма запущена, аудиофайл загружен, выбрана необходимая область.

Тестовый случай: нарастание области.

Ожидаемый результат: корректное применение эффекта нарастания области.

Результат представлен на рисунках 4.15 и 4.16.

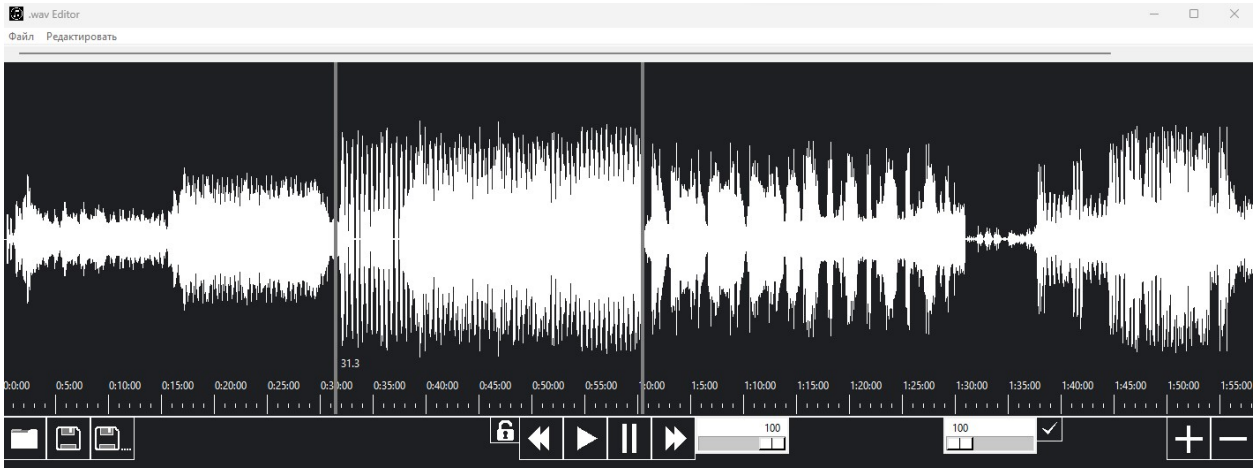

Рисунок 4.15 – Аудиодорожка до применения эффекта

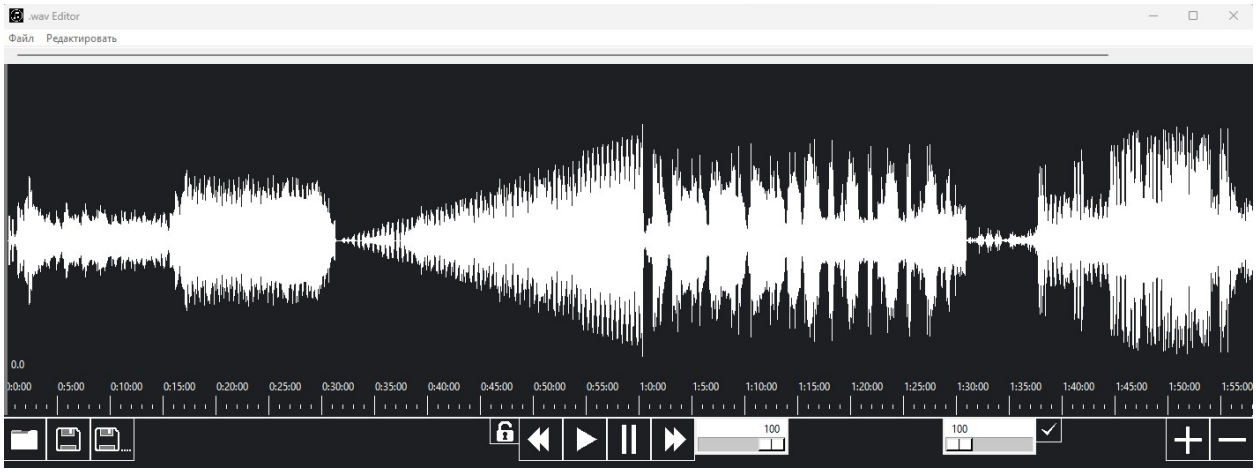

Рисунок 4.16 – Аудиодорожка после применения эффекта

Предусловие: програма запущена, аудиофайл загружен, выбрана необходимая область.

Тестовый случай: затухание области.

Ожидаемый результат: корректное применение эффекта затухания области.

Результат представлен на рисунках 4.17 и 4.18.

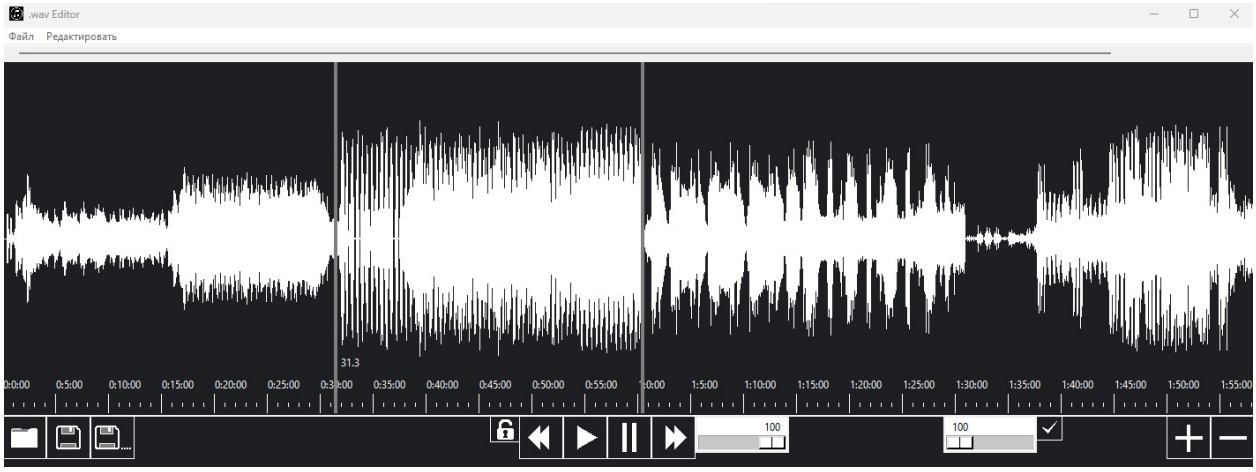

Рисунок 4.17 – Аудиодорожка до применения эффекта

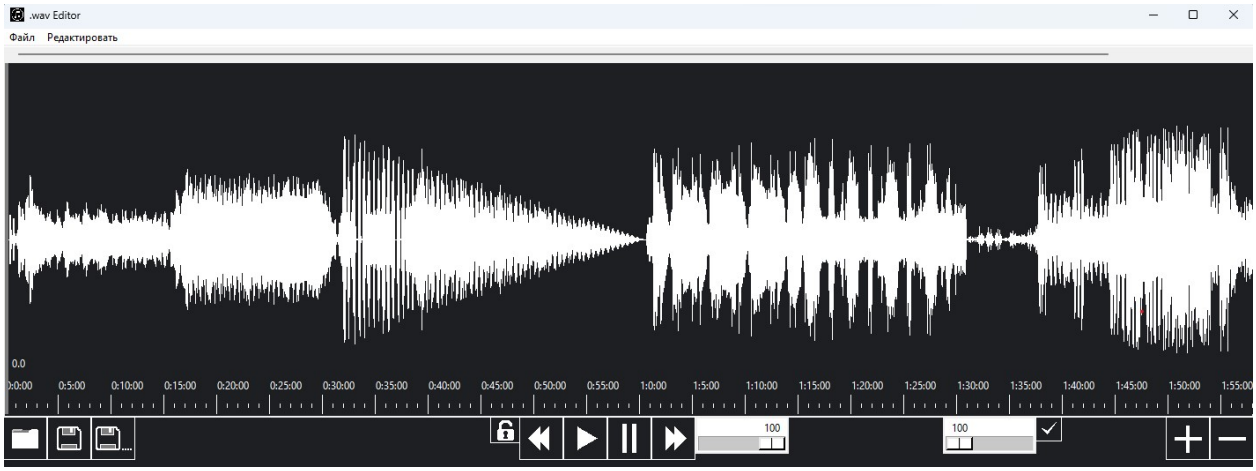

Рисунок 4.18 – Аудиодорожка после применения эффекта

4.2.5 Управление действиями

Предусловие: програма запущена, аудиофайл загружен, совершенно какое-либо изменение аудиоданных.

Тестовый случай: отмена операции.

Ожидаемый результат: корректная отмена совершенной операции.

Результат представлен на рисунках 4.19 и 4.20.

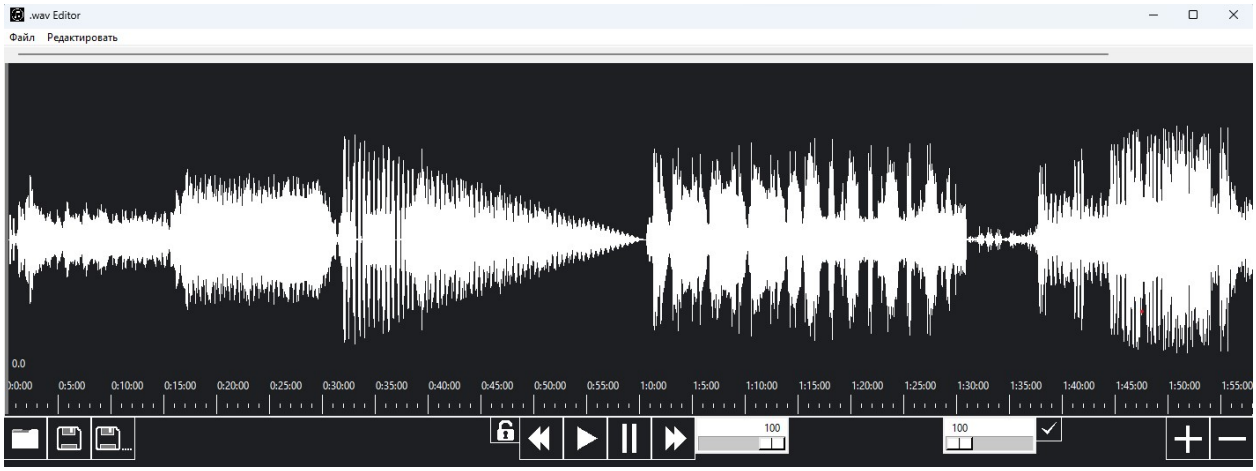

Рисунок 4.19 – Аудиодорожка до отмены операции

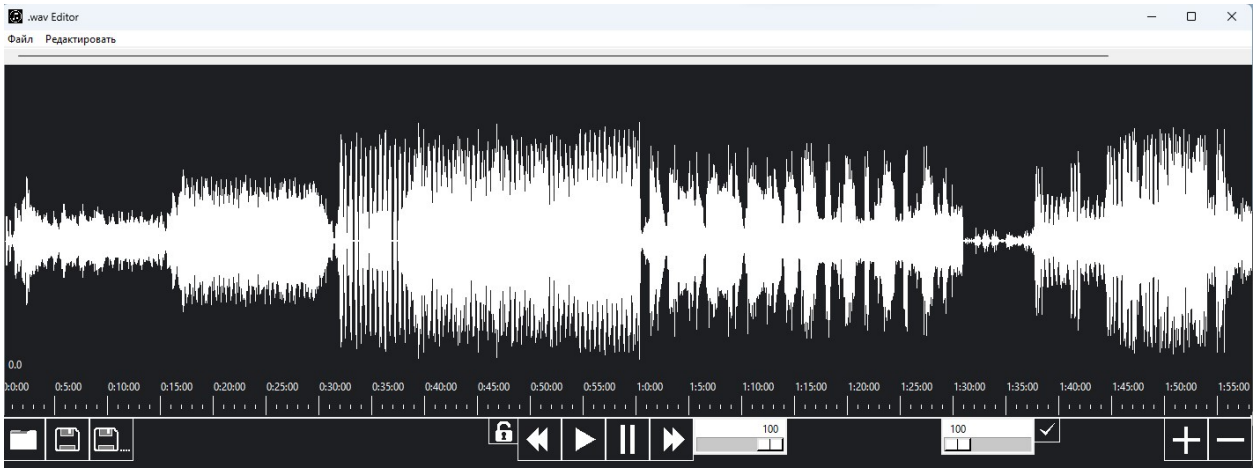

Рисунок 4.20 – Аудиодорожка после отмены операции

Предусловие: програма запущена, аудиофайл загружен, совершенно какое-либо изменение аудиоданных.

Тестовый случай: возврат операции.

Ожидаемый результат: корректный возврат совершенной операции.

Результат представлен на рисунках 4.21 и 4.22.

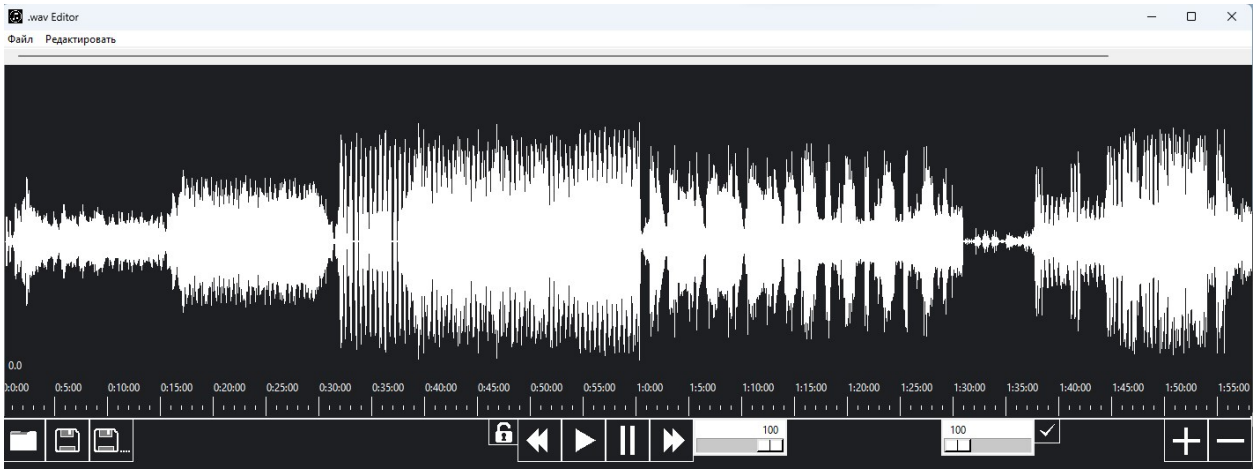

Рисунок 4.21 – Аудиодорожка до возврата операции

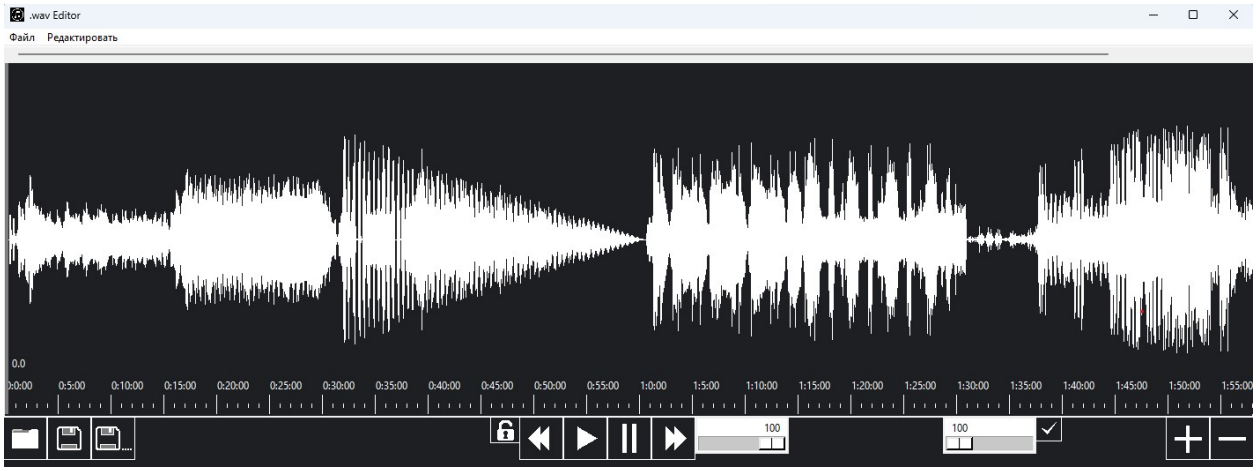

Рисунок 4.22 – Аудиодорожка после возврата операции

4.2.6 Сохранение аудиофайла

Предусловие: програма запущена, аудиофайл загружен, совершенны какие-либо операции над аудиоданными (необязательно).

Тестовый случай: сохранение аудиофайла.

Ожидаемый результат: корректное сохранение аудиофайла.

Результат представлен на рисунках 4.23 и 4.24.

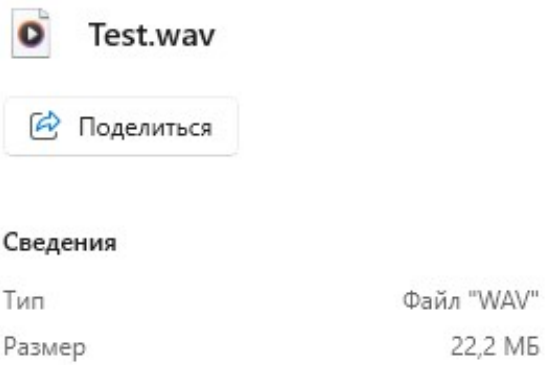

Рисунок 4.23 – Размер оригинального аудиофайла

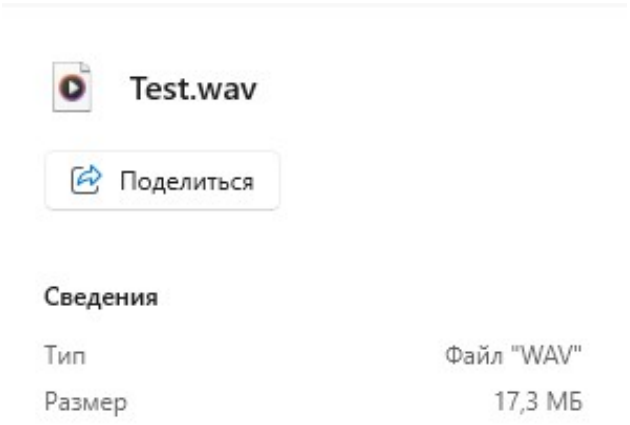

Рисунок 4.24 – Размер аудиофайла после сохранения с изменениями

ЗАКЛЮЧЕНИЕ

Современный мир музыки и звукозаписи не может обойтись без аудиоредакторов. Они стали незаменимыми инструментами в музыкальной индустрии, радиовещании, телевидении, видеопроизводстве, кино и других областях, где необходимо обрабатывать и улучшать звуковые файлы. Соответственно, профессиональные навыки работы с аудиоредакторами являются очень актуальными на современном рынке труда. Владение компьютерными программами для звукозаписи и редактирования аудиофайлов открывает широкие возможности для работы в медиа-индустрии и музыкальном бизнесе. Кроме этого, возможности аудиоредакторов легко доступны и для любителей – они могут использоваться для создания и обработки интересных и креативных музыкальных проектов.

Основные результаты работы:

1. Проведен анализ предметной области.
2. Разработана концептуальная модель программы.
3. Разработана архитектура программы.
4. Реализован пользовательский интерфейс.
5. Проведено системное тестирование.

Все требования, объявленные в техническом задании, были полностью реализованы, все задачи, поставленные в начале разработки проекта, были также решены.

Готовый рабочий проект представлен программой расширения .exe.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Лерч А. ”Python для анализа звука”/ Александр Лерч. – Packt Publishing, 2013. – 504 с. – ISBN: 978-1-78216-889-6. – Текст: непосредственный.

2. Бек Э. ”Python для музыкального программирования”/ Эндрю Бек. – Course Technology PTR, 2008. – 351 с. – ISBN: 978-1-59863-601-9. – Текст: непосредственный.

3. Рид М. ”Обработка звука в Python”/ Майк Рид. – CreateSpace Independent Publishing Platform, 2015. – 286 с. – ISBN: 978-1-5177-1464-8. – Текст: непосредственный.

4. Бильбао С. ”Python Sound: обработка сигналов и акустика”/ Стефан Бильбао. – CRC Press, 2018. – 454 с. – ISBN: 978-0-367-19283-3. – Текст: непосредственный.

5. Мэнк Дж. ”Python для обработки аудиосигналов”/ Джозеф Мэнк. – Springer, 2015. – 426 с. – ISBN: 978-3-319-21947-8. – Текст: непосредственный.

6. Гвидо ван Россум. ”Python: основы программирования”/ Гвидо ван Россум. – Москва: ДМК Пресс, 2019. – 232 с. – ISBN: 978-5-97060-762-7. – Текст: непосредственный.

7. Маннинг С. ”Python машинного обучения: Сборник рецептов”/ Стивен Маннинг. - SPb.: Питер, 2018. - 384 с. - ISBN: 978-5-4461-0598-9. - Текст: непосредственный.

8. Миллер М. ”Учимся писать игры на Python”/ Майк Миллер. - М.: ДМК Пресс, 2018. - 320 с. - ISBN: 978-5-97060-560-9. - Текст: непосредственный.

9. Златопольский Д.М. ”Python. Курс основного уровня”/ Златопольский Д.М. - СПб.: Питер, 2018. - 400 с. - ISBN: 978-5-496-02465-5. - Текст: непосредственный.

10. Мартелли А. ”Python в примерах: учебное пособие”/ Алекс Мартелли, Дэвид Гудгертьс, Идан Газит. - М.: ДМК Пресс, 2018. - 616 с. - ISBN: 978-5-97060-533-3. - Текст: непосредственный.

11. ВандерПлас Дж. ”Python для анализа данных”/ Джейк ВандерПлас. - М.: ДМК Пресс, 2018. - 592 с. - ISBN: 978-5-97060-279-0. - Текст: непосредственный.

ПРИЛОЖЕНИЕ А

Фрагменты исходного кода программы

main.py

```
from audio import Audio, CutCommand, PasteCommand, NullifyCommand,
    VolumeCommand, FadeCommand
from audioplayer import AudioPlayer
from command import command_buffer
from sdl2.sdlmixer import Mix_Paused, Mix_Volume, Mix_Pause
from tkinter import PhotoImage, Button, Scale, Menu
from window import Window

if __name__ == "__main__":
    audio = Audio()
    player = AudioPlayer(audio)
    window = Window(player)
    canvas = window.scrollable_canvas
    t_line = canvas.t_line


    def load_file():
        """
        Событие нажатия на кнопку загрузки аудио.
        """
        player.load_file()
        canvas.draw(True, True, True)


    def play_audio():
        """
        Событие нажатия на кнопку воспроизведения аудио.
        """
        if len(audio.signals_data) == 0:
            return

        if not player.bactive:
            player.open_player()

        if not player.bplaying:
            if t_line.bend:
                t_line.change_position(canvas.s_line.x, True)
                player.set_time_borders(player.start, player.end, True, True)
            else:
                player.set_time(player.current, True)
            t_line.start()


    def stop_audio():
        """
        Событие нажатия на кнопку остановки воспроизведения аудио.
        """
        player.pause_audio()
        t_line.pause()
```

```
49
50
51      def skip_audio(bback):
52          """
53          Событие нажатия на кнопку загрузки.
54          :param bback: флаг для направления пропуска.
55          """
56          value = -(300 / canvas.pixel_values[canvas.pixel_values[0]]) if bback
                else (
57                  300 / canvas.pixel_values[canvas.pixel_values[0]])
58          t_line.change_position(t_line.x + value, True)
59          if player.current <= 0.0:
60              return
61
62          if not t_line.bend and not Mix_Paused(0):
63              player.set_time(player.current, True)
64              t_line.start()
65          else:
66              Mix_Pause(0)
67
68
69      def lock_lines():
70          """
71          Событие нажатия на кнопку закрепления линий.
72          """
73          canvas.block = not canvas.block
74          lock_button.config(image=bimages[8] if canvas.block else bimages[9])
75
76
77      def copy_audio():
78          """
79          Событие нажатия на кнопку копирования области аудио.
80          """
81          audio.copy_audio(player.start_index, player.end_index)
82          if not canvas.block:
83              lock_lines()
84
85
86      def cut_audio(bcopy):
87          """
88          Событие нажатия на кнопки удаления и выреза области аудио.
89          :param bcopy: флаг для копирования области аудио.
90          """
91          command_buffer.add(
92              CutCommand(
93                  audio=audio, bcopy=True, start_index=player.start_index,
                        end_index=player.end_index
94              )
95          )
96          player.set_time_borders(0.0, audio.duration, True, False)
97          canvas.draw(True, True, True)
98          if canvas.block:
99              lock_lines()
100
```

```
101
102     def paste_audio(breplace):
103         """
104         Событие нажатия на кнопку вставки области аудио.
105         :param breplace: флаг для замены области аудио.
106         """
107         command_buffer.add(
108             PasteCommand(
109                 audio=audio, breplace=breplace, start_index=player.
                        start_index, end_index=player.end_index,
110             )
111         )
112         player.set_time_borders(player.start, player.end, True, False)
113         canvas.draw(True, True, True)
114
115
116     def nullify_audio():
117         """
118         Событие нажатия на кнопку обнуления области аудио.
119         """
120         command_buffer.add(
121             NullifyCommand(
122                 audio=audio, start_index=player.start_index, end_index=player
                        .end_index
123             )
124         )
125         player.set_time_borders(player.start, player.end, True, False)
126         canvas.draw(True, True, True)
127
128
129     def redo():
130         if command_buffer.buffer[command_buffer.index] != 0:
131             command_buffer.buffer[command_buffer.index].do()
132
133             player.set_time_borders(player.start, player.end, True, False)
134             canvas.draw(True, True, True)
135
136     def undo():
137         if command_buffer.index - 1 >= 0:
138             command_buffer.buffer[command_buffer.index - 1].undo()
139
140             player.set_time_borders(player.start, player.end, True, False)
141             canvas.draw(True, True, True)
142
143
144     def fade_audio(bout):
145         """
146         Событие нажатия на кнопку нарастания/затухания области аудио.
147         :param bout: флаг для затухания.
148         """
149         command_buffer.add(
150                 FadeCommand(
```

```
                    seconds=player.end - player.start, bout=bout, audio=audio
                        , start_index=player.start_index, end_index=player.
                        end_index
                )
            )
        player.set_time_borders(player.start, player.end, True, False)
        canvas.draw(btime=True)

    def volume_audio():
        """
        Событие нажатия на кнопку изменения громкости области аудио.
        """
        volume = volume_value.get()
        if volume > 0:
            command_buffer.add(
                VolumeCommand(
                    volume=volume, audio=audio, start_index=player.
                        start_index, end_index=player.end_index
                )
            )
        else:
            command_buffer.add(
                NullifyCommand(
                    audio=audio, start_index=player.start_index, end_index=
                        player.end_index
                )
            )
        player.set_time_borders(player.start, player.end, True, False)
        canvas.draw(btime=True)


    def volume_player(value):
        player.volume = int(value)
        Mix_Volume(0, int(value))


    edit_menu = Menu(tearoff=0)
    edit_menu.add_command(label="Копировать", command=lambda: copy_audio())
    edit_menu.add_command(label="Удалить", command=lambda x=False: cut_audio(
        x))
    edit_menu.add_command(label="Вырезать", command=lambda x=True: cut_audio(
        x))
    edit_menu.add_command(label="Вставить", command=lambda x=False:
        paste_audio(x))
    edit_menu.add_command(label="Заменить", command=lambda x=True:
        paste_audio(x))
    edit_menu.add_command(label="Обнулить", command=lambda: nullify_audio())
    edit_menu.add_command(label="Нарастание", command=lambda x=False:
        fade_audio(x))
    edit_menu.add_command(label="Затухание",  command=lambda x=True:
        fade_audio(x))
    edit_menu.add_separator()
    edit_menu.add_command(label="Отменить", command=lambda: undo())
    edit_menu.add_command(label="Вернуть", command=lambda: redo())
```

```
195
196
197     def show_edit_menu(event):
198         """
199         Событие вызова меню.
200         :param event: событие.
201         """
202         if canvas.bdrawing:
203             return
204
205         edit_menu.post(x=event.x_root, y=event.y_root)
206
207
208     canvas.bind("<ButtonPress-2>", show_edit_menu)
209
210     bimages = [
211         PhotoImage(file="load_button.png"),
212         PhotoImage(file="save_button.png"),
213         PhotoImage(file="save_loc_button.png"),
214         PhotoImage(file="play_button.png"),
215         PhotoImage(file="pause_button.png"),
216
217         PhotoImage(file="skipb_button.png"),
218         PhotoImage(file="skipf_button.png"),
219
220         PhotoImage(file="volume_button.png"),
221
222         PhotoImage(file="lock_button.png"),
223         PhotoImage(file="lock2_button.png"),
224
225         PhotoImage(file="less_button.png"),
226         PhotoImage(file="more_button.png")
227     ]
228
229     load_button = Button(
230         master=window,
231         image=bimages[0],
232         command=lambda: load_file(),
233         bd=0, highlightthickness=0
234     )
235     load_button.place(
236         relx=0, x=25, y=420, anchor="n"
237     )
238
239     save_button = Button(
240         master=window,
241         image=bimages[1],
242         command=lambda: player.save_file(),
243         bd=0, highlightthickness=0
244     )
245     save_button.place(
246         relx=0, x=75, y=420, anchor="n"
247     )
248
```

```
    save_as_button = Button(
        master=window,
        image=bimages[2],
        command=lambda: player.save_file_where(),
        bd=0, highlightthickness=0
    )
    save_as_button.place(
        relx=0, x=125, y=420, anchor="n"
    )

    lock_button = Button(
        master=window,
        image=bimages[9],
        command=lambda: lock_lines(),
        bd=0, highlightthickness=0
    )
    lock_button.place(
        relx=0.43, x=-41, y=420, anchor="n"
    )
    skipb_button = Button(
        master=window,
        image=bimages[5],
        command=lambda x=True: skip_audio(x),
        bd=0, highlightthickness=0
    )
    skipb_button.place(
        relx=0.43, x=0, y=420, anchor="n"
    )
    play_button = Button(
        master=window,
        image=bimages[3],
        command=lambda: play_audio(),
        bd=0, highlightthickness=0
    )
    play_button.place(
        relx=0.43, x=50, y=420, anchor="n"
    )
    pause_button = Button(
        master=window,
        image=bimages[4],
        command=lambda: stop_audio(),
        bd=0, highlightthickness=0
    )
    pause_button.place(
        relx=0.43, x=100, y=420, anchor="n"
    )
    skipf_button = Button(
        master=window,
        image=bimages[6],
        command=lambda x=False: skip_audio(x),
        bd=0, highlightthickness=0
    )
    skipf_button.place(
        relx=0.43, x=150, y=420, anchor="n"
```

```
303     )
304
305     volume_scale = Scale(
306         master=window, orient="horizontal", from_=0, to=100, bg="white",
307         command=lambda v: volume_player(v)
308     )
309     volume_scale.set(100)
310     volume_scale.place(
311         relx=0.43, x=228, y=420, anchor="n"
312     )
313
314     volume_value = Scale(
315         master=window,
316         from_=100, to=0,
317         orient="horizontal", bg="white"
318     )
319     volume_value.set(100)
320     volume_value.place(
321         relx=1, x=-300, y=420, anchor="n"
322     )
323
324     volume_button = Button(
325         master=window,
326         image=bimages[7],
327         command=lambda: volume_audio(),
328         bd=0, highlightthickness=0
329     )
330     volume_button.place(
331         relx=1, x=-233, y=420, anchor="n"
332     )
333
334     less_button = Button(
335         master=window,
336         image=bimages[10],
337         command=lambda x=False: canvas.change_scale(x),
338         bd=0, highlightthickness=0
339     )
340     less_button.place(
341         relx=1, x=-25, y=420, anchor="n"
342     )
343     more_button = Button(
344         master=window,
345         image=bimages[11],
346         command=lambda x=True: canvas.change_scale(x),
347         bd=0, highlightthickness=0
348     )
349     more_button.place(
350         relx=1, x=-75, y=420, anchor="n"
351     )
352
353     load_menu = Menu(tearoff=0)
354     load_menu.add_command(label="Загрузить", command=lambda: load_file())
355     load_menu.add_command(label="Сохранить", command=lambda: player.save_file
        ())
```

```
356     load_menu.add_command(label="Сохранить...", command=lambda: player.
            save_file_where())
357
358     window_menu = Menu(tearoff=0)
359     window_menu.add_cascade(label="Файл", menu=load_menu)
360     window_menu.add_cascade(label="Редактировать", menu=edit_menu)
361
362     window.config(menu=window_menu)
363     window.mainloop()
```

audio.py

```
from numpy import array, int16, concatenate, zeros, copy
from command import Command


class Audio:
    def __init__(self):
        self.name = ""

        self.nchannels = 0
        self.sampwidth = 0
        self.framerate = 0
        self.nframes = 0
        self.comptype = "NONE"
        self.compname = "not compressed"

        self.chunksize = 0
        self.copy_buffer = array([], dtype=int16)
        self.signals_data = array([], dtype=int16)

        self.duration = 0.0

    def copy_audio(self, start_index, end_index):
        """
        Копирование области аудио.
        :param start_index: индекс начала области аудио.
        :param end_index: индекс конца области аудио.
        """
        if len(self.signals_data) == 0:
            return

        self.copy_buffer = self.signals_data[start_index: end_index]


class CutCommand(Command):
    def __init__(self, bcopy, **kwargs):
        super().__init__(**kwargs)
        self.bcopy = bcopy

    def do(self):
        """
        Удаление области аудио.
        """
        if super().do():
            if self.bcopy:
                self.audio.copy_buffer = array(self.audio.signals_data[self.
                    start_index: self.end_index])

            self.audio.signals_data = array(
                concatenate((
                    self.audio.signals_data[:self.start_index],
                    self.audio.signals_data[self.end_index:]
                ))
            )
```

```
53
54              self.audio.nframes = len(self.audio.signals_data) // self.audio.
                    nchannels
55              self.audio.duration = self.audio.nframes / self.audio.framerate
56
57      def undo(self):
58          """
59          Отмена удаления области аудио.
60          """
61          if super().undo():
62              self.audio.signals_data = array(
63                  concatenate((
64                      self.audio.signals_data[: self.start_index],
65                      self.buffer,
66                      self.audio.signals_data[self.start_index:]
67                  ))
68              )
69
70              self.audio.nframes = len(self.audio.signals_data) // self.audio.
                    nchannels
71              self.audio.duration = self.audio.nframes / self.audio.framerate
72
73
74  class PasteCommand(Command):
75      def __init__(self, breplace, **kwargs):
76          super().__init__(**kwargs)
77          self.breplace = breplace
78
79      def do(self):
80          """
81          Вставка области аудио.
82          """
83          if super().do():
84              if self.breplace:
85                  self.audio.signals_data = array(
86                      concatenate((
87                          self.audio.signals_data[:self.start_index],
88                          self.audio.copy_buffer,
89                          self.audio.signals_data[self.end_index:]
90                      ))
91                  )
92              else:
93                  self.audio.signals_data = array(
94                      concatenate((
95                          self.audio.signals_data[:self.start_index],
96                          self.audio.copy_buffer,
97                          self.audio.signals_data[self.start_index:]
98                      ))
99                  )
100                 self.audio.nframes = len(self.audio.signals_data) // self.
                        audio.nchannels
101                 self.audio.duration = self.audio.nframes / self.audio.
                        framerate
102
```

```
103     def undo(self):
104         """
105         Отмена вставки области аудио.
106         """
107         if super().undo():
108             if self.breplace:
109                 self.audio.signals_data = array(
110                     concatenate((
111                         self.audio.signals_data[:self.start_index],
112                         self.buffer,
113                         self.audio.signals_data[self.end_index:]
114                     ))
115                 )
116             else:
117                 self.audio.signals_data = array(
118                     concatenate((
119                         self.audio.signals_data[:self.start_index],
120                         self.audio.signals_data[self.end_index:]
121                     ))
122                 )
123                 self.audio.nframes = len(self.audio.signals_data) // self.
                        audio.nchannels
124                 self.audio.duration = self.audio.nframes / self.audio.
                        framerate
125
126
127 class NullifyCommand(Command):
128     def __init__(self, **kwargs):
129         super().__init__(**kwargs)
130
131     def do(self):
132         """
133         Обнуление области аудио.
134         """
135         if super().do():
136             self.audio.signals_data = array(
137                 concatenate((
138                     self.audio.signals_data[:self.start_index],
139                     array(
140                         zeros(len(self.buffer), dtype=int16)
141                     ),
142                     self.audio.signals_data[self.end_index:]
143                 ))
144             )
145
146     def undo(self):
147         """
148         Отмена обнуления области аудио.
149         """
150         if super().undo():
151             self.audio.signals_data = array(
152                 concatenate((
153                     self.audio.signals_data[:self.start_index],
154                     self.buffer,
```

```
                    self.audio.signals_data[self.end_index:]
                ))
            )


class FadeCommand(Command):
    def __init__(self, seconds, bout, **kwargs):
        super().__init__(**kwargs)
        self.seconds = seconds
        self.bout = bout

    def do(self):
        """
        Эффект нарастания/затухания громкости области аудио.
        """
        if super().do():
            interm_data = copy(self.buffer)
            indexes = self.seconds * self.audio.framerate * self.audio.
                nchannels
            if not self.bout:
                k = 0
                for index in range(0, len(interm_data)):
                    interm_data[index] *= k
                    k += 1 / indexes
                    if k >= 1:
                        break
            else:
                k = 1
                for index in range(0, len(interm_data)):
                    interm_data[index] *= k
                    k -= 1 / indexes
                    if k <= 0:
                        break
            self.audio.signals_data = array(
                concatenate((
                    self.audio.signals_data[:self.start_index],
                    interm_data,
                    self.audio.signals_data[self.end_index:]
                ))
            )
            del interm_data

    def undo(self):
        """
        Отмена эффекта нарастания/затухания громкости области аудио.
        """
        if super().undo():
            self.audio.signals_data = array(
                concatenate((
                    self.audio.signals_data[:self.start_index],
                    self.buffer,
                    self.audio.signals_data[self.end_index:]
                ))
            )
```

```
208
209
210 class VolumeCommand(Command):
211     def __init__(self, volume, **kwargs):
212         super().__init__(**kwargs)
213         self.volume = volume
214
215     def do(self):
216         """
217         Изменение громкости области аудио.
218         """
219         if super().do():
220             self.audio.signals_data = array(
221                 concatenate((
222                     self.audio.signals_data[: self.start_index],
223                     (self.buffer * (self.volume / 100)).astype(dtype=int16),
224                     self.audio.signals_data[self.end_index:]
225                 ))
226             )
227
228     def undo(self):
229         """
230         Отмена изменения громкости области аудио.
231         """
232         if super().undo():
233             self.audio.signals_data = array(
234                 concatenate((
235                     self.audio.signals_data[: self.start_index],
236                     self.buffer,
237                     self.audio.signals_data[self.end_index:]
238                 ))
239             )
```

audioplayer.py

```
from ctypes import c_ubyte, cast, POINTER
from numpy import array, frombuffer, int16
from sdl2.sdlmixer import (Mix_Pause, Mix_Resume, Mix_Paused,
    Mix_QuickLoad_RAW,
                           Mix_OpenAudio, Mix_CloseAudio, Mix_HaltChannel,
                               Mix_PlayChannel, Mix_Volume,
                           AUDIO_S16LSB)
from tkinter.filedialog import askopenfilename, asksaveasfile
from wave import *


class AudioPlayer:
    def __init__(self, audio):
        self.audio = audio
        self.bactive = False
        self.bplaying = False

        self.chunk = b""
        self.volume = 100

        self.start = 0.0
        self.start_index = 0
        self.end = 0.0
        self.end_index = 0
        self.current = 0.0

    def open_player(self):
        """
        Инициализация SDL аудио устройства.
        """
        if self.bactive:
            self.close_player()

        result = Mix_OpenAudio(
            self.audio.framerate, AUDIO_S16LSB, self.audio.nchannels, self.
                audio.chunksize
        )
        if result == 0:
            self.bactive = True

    def close_player(self):
        """
        Закрытие SDL аудио устройства.
        """
        if not self.bactive:
            return

        Mix_CloseAudio()
        self.bactive = False
        self.bplaying = False

    def pause_audio(self, event=None):
        """
```

```
        Остановка проигрывания аудио.
        :param event: событие нажатия.
        """
        self.bplaying = False
        Mix_Pause(0)

    def resume_audio(self, event=None):
        """
        Возобновление проигрывания аудио.
        :param event: событие нажатия.
        """
        if not self.bplaying:
            self.bplaying = True
            Mix_Resume(0)

    def set_time_borders(self, start, end, bload, bplay):
        """
        Установка границ области аудио.
        :param start: время начада области аудио в секудах.
        :param end: время конца области аудио в секундах.
        :param bload: флаг для загрузки области аудио в буфер.
        :param bplay: флаг для воспроизведения аудио.
        """
        self.start = start
        self.start_index = int(self.start * self.audio.framerate * self.audio
            .nchannels)
        self.end = end
        self.end_index = int(self.end * self.audio.framerate * self.audio.
            nchannels)

        if bload:
            Mix_HaltChannel(0)
            self.load_chunk(self.audio.signals_data, self.start_index, self.
                end_index, bplay)

    def set_time(self, start, bplay):
        """
        Установка временной отметки в области аудио.
        :param start: время линии отметки времени в секудах.
        :param bplay: флаг для воспроизведения аудио.
        """
        self.current = start
        time = int(self.current * self.audio.framerate * self.audio.nchannels
            )

        if time < self.start_index:
            time = self.start_index

        Mix_HaltChannel(0)
        self.load_chunk(self.audio.signals_data, time, self.end_index, bplay)

    def load_file(self, chunksize=2048):
        """
        Загрузка аудиофайла в буфер.
```

```
101         :param chunksize: размер буфера, воспроизводимого в единицу времени,
                в байтах.
102         """
103         try:
104             filename = askopenfilename(filetypes=(("WAVE files", "*.wav"), ("
                    All files", "*.*")))
105 
106             with open(filename, "rb") as wave_sample:
107                 params = wave_sample.getparams()
108                 self.audio.name = filename.split(".")[-1]
109 
110                 self.audio.nchannels = params[0]
111                 self.audio.sampwidth = params[1]
112                 self.audio.framerate = params[2]
113                 self.audio.nframes = params[3]
114                 self.audio.comptype = params[4]
115                 self.audio.compname = params[5]
116                 self.audio.chunksize = chunksize
117                 self.audio.signals_data = array(frombuffer(wave_sample.
                        readframes(self.audio.nframes), int16))
118 
119             self.audio.duration = self.audio.nframes / self.audio.framerate
120             self.set_time_borders(0.0, self.audio.duration, True, False)
121 
122             self.open_player()
123         except FileNotFoundError:
124             return
125 
126     def save_file(self):
127         """
128         Запись аудиофайла на компьютер.
129         """
130         with open(self.audio.name + "_changed.wav", "wb") as wave_sample:
131             params = (
132                 self.audio.nchannels, self.audio.sampwidth, self.audio.
                        framerate, self.audio.nframes,
133                 self.audio.comptype, self.audio.compname
134             )
135             wave_sample.setparams(params)
136             wave_sample.writeframes(self.audio.signals_data.tobytes())
137 
138     def save_file_where(self):
139         """
140         Запись аудиофайла на компьютер с выбором названия и папки.
141         """
142         file = asksaveasfile(mode="w", defaultextension=".wav", filetypes=[("
                WAVE files", "*.wav")])
143 
144         if file is None:
145             return
146         file.write("")
147 
148         with open(file.name.split("/")[-1], "w") as wave_sample:
149             params = (
```

```
150                 self.audio.nchannels, self.audio.sampwidth, self.audio.
                        framerate, self.audio.nframes,
151                 self.audio.comptype, self.audio.compname
152             )
153             wave_sample.setparams(params)
154             wave_sample.writeframes(self.audio.signals_data.tobytes())
155 
156     def load_chunk(self, chunk, start_index, end_index, play, loops=0):
157         """
158         Установка аудиобуфера.
159         :param chunk: аудиобуфер.
160         :param start_index: индекс начала области аудио.
161         :param end_index: индекс конца области аудио.
162         :param play: флаг для проигрывания аудио.
163         :param loops: количество циклов воспроизведения.
164         """
165         if len(self.audio.signals_data) == 0:
166             return
167 
168         if start_index < 0:
169             start_index = 0
170         if end_index >= len(chunk) - 1:
171             end_index = len(chunk) - 1
172 
173         buflen = len(chunk[start_index: end_index]) * self.audio.nchannels
174         buffer = (c_ubyte * buflen).from_buffer_copy(chunk[start_index:
                end_index])
175 
176         self.chunk = Mix_QuickLoad_RAW(cast(buffer, POINTER(c_ubyte)), buflen
                )
177         self.play_chunk(self.chunk, loops) if play else self.pause_audio()
178 
179     def play_chunk(self, chunk, loops):
180         """
181         Воспроизведение аудио.
182         :param chunk: аудиобуфер.
183         :param loops: количество циклов воспроизведения.
184         """
185         if len(self.audio.signals_data) == 0:
186             return
187 
188         Mix_Volume(0, self.volume)
189         if Mix_Paused(0):
190             Mix_Resume(0)
191         else:
192             Mix_HaltChannel(0)
193             Mix_PlayChannel(0, chunk, loops)
194         self.bplaying = True
```

canvas.py

```
from eline import EdgeLine
from tkinter import Canvas, Label
from tline import TimeLine
from numpy import absolute


class ScrollableCanvas(Canvas):
    def __init__(self, player, **keyargs):
        Canvas.__init__(self, **keyargs)

        self.loaded = False
        self.player = player

        self.signals = []
        self.signals_count = 1

        self.pixel_values = [
            2,
            1,
            5,
            10,
            30,
            60
        ]
        self.pixels_per_second = 60 // self.pixel_values[self.pixel_values
           [0]]

        self.bdrawing = False
        self.block = False

        self.s_line = EdgeLine(
            master=self.master, x=0,
            height=400, width=0, bg="grey"
        )
        self.e_line = EdgeLine(
            master=self.master, x=0,
            height=400, width=0, bg="grey"
        )
        self.t_line = TimeLine(
            player=self.player, master=self,
            s_line=self.s_line, e_line=self.e_line,
            interval=0, x=0, height=400, width=0, bg="purple"
        )
        self.labels = [0, [Label()]]
        self.pack(side="top", fill="x")

        self.bind("<ButtonPress-1>", lambda e, x=False: self.set_line(e, x))
        self.bind("<ButtonPress-3>", lambda e, x=True: self.set_line(e, x))

    def change_lines_position(self, start, end, xstart, xend, bplay):
        """
        Установка позиции линий.
        :param start: время начада области аудио в секудах.
```

```
53          :param end: время конца области аудио в секудах.
54          :param xstart: индекс начала области аудио.
55          :param xend: индекс конца области аудио.
56          :param bplay: флаг для проигрывания аудио.
57          """
58          self.s_line.change_position(xstart)
59          self.e_line.change_position(xend)
60          self.player.set_time_borders(start, end, True, bplay)
61
62      def set_line(self, event, bend):
63          """
64          Событие установки позиции линий.
65          :param event: событие нажатия.
66          :param bend: флаг для разделения типа линий.
67          """
68          if self.bdrawing or self.signals_count < 1:
69              return
70
71          time = self.player.start - self.player.end if bend else self.player.
               end - self.player.start
72          start = self.canvasx(event.x) * self.player.audio.duration / self.
               signals_count
73          if self.block:
74              end = (
75                      (self.canvasx(event.x) + time * self.pixels_per_second) *
76                      self.player.audio.duration / self.signals_count
77              )
78              if bend:
79                  if self.canvasx(event.x) + time * self.pixels_per_second < 0:
80                      return
81
82                  self.change_lines_position(
83                      end, start, self.canvasx(event.x) + time * self.
                           pixels_per_second,
84                      self.canvasx(event.x), self.player.bplaying
85                  )
86              else:
87                  if self.canvasx(event.x) + time * self.pixels_per_second >
                       self.signals_count - 4:
88                      return
89
90                  self.change_lines_position(
91                      start, end, self.canvasx(event.x),
92                      self.canvasx(event.x) + time * self.pixels_per_second,
                           self.player.bplaying
93                  )
94          else:
95              if bend:
96                  if self.s_line.x < self.canvasx(event.x):
97                      self.e_line.change_position(self.canvasx(event.x))
98
99                      self.player.set_time_borders(self.player.start, start,
                           True, self.player.bplaying)
100             else:
```

```
                if self.e_line.x > self.canvasx(event.x):
                    self.s_line.change_position(self.canvasx(event.x))

                    self.player.set_time_borders(start, self.player.end, True
                        , self.player.bplaying)

            self.t_line.change_position(self.s_line.x, True)

        self.t_line.change_position(self.s_line.x, True)

    def draw(self, bstart=False, bend=False, btime=False):
        """
        Отрисовка аудиодорожки.
        :param bstart: флаг для сброса линии начала границы области аудио.
        :param bend: флаг для сброса линии конца границы области аудио.
        :param btime: флаг для сброса линии отметки времени области аудио.
        """
        if len(self.player.audio.signals_data) == 0:
            self.delete("all")
            if len(self.labels) > 1:
                for i in self.labels[1]:
                    i.destroy()
            self.reset_lines(bstart, bend, btime)
            return

        self.pixels_per_second = 60 // self.pixel_values[self.pixel_values
            [0]]
        self.signals_count = int(self.player.audio.duration * self.
            pixels_per_second)

        if self.signals_count < 1:
            self.signals_count = 1
            width = 0
        else:
            width = self.signals_count

        self.config(
            width=width, height=400 - 2
        )

        if not self.loaded:
            self.s_line.config(width=4)
            self.e_line.config(width=4)
            self.t_line.config(width=4)
            self.reset_lines(True, True, True)
            self.loaded = True

        self.delete("all")
        self.update()

        self.master.master.config(
            scrollregion=(
                0, 0, self.signals_count, 0
            )
```

```
        )
        self.reset_lines(bstart, bend, btime)
        self.create_line(0, 200, self.signals_count, 200, width=1, fill="
            white")
        self.create_line(self.signals_count - 1, 0, self.signals_count - 1,
            400, width=1, fill="white")

        if self.player.audio.nchannels == 2:
            buflen = len(self.player.audio.signals_data[0::2])
        else:
            buflen = len(self.player.audio.signals_data)

        step = int(buflen // self.signals_count)

        if self.player.audio.nchannels == 2:
            self.signals = [
                self.player.audio.signals_data[0::2][i: i + step]
                for i in range(0, buflen - step, step)
            ]
        else:
            self.signals = [
                self.player.audio.signals_data[i: i + step]
                for i in range(0, buflen - step, step)
            ]

        xlineposition = 0
        if len(self.labels) > 1:
            for i in self.labels[1]:
                i.destroy()
            self.labels = [0, [0] * int(self.player.audio.duration / self.
                pixel_values[self.pixel_values[0]] + 1)]

        for index in range(self.signals_count):
            if index > len(self.signals) - 1:
                break

            self.bdrawing = True
            average = sum(absolute(self.signals[index])) // len(self.signals[
                index])
            self.create_line(xlineposition, 200 - average // 110,
                xlineposition, 200 + average // 110, width=1, fill="white")

            if xlineposition % 12 == 0:
                if xlineposition % 60 == 0:
                    self.create_line(xlineposition, 400, xlineposition, 375,
                        width=1, fill="white")

                    seconds = self.labels[0] * self.pixel_values[self.
                        pixel_values[0]]
                    self.labels[1][self.labels[0]] = Label(
                        master=self, text=f"{seconds // 60}:{(seconds - (
                            seconds // 60) * 60)}:00",
                        bg="#1E1F22", fg="white"
                    )
```

```
198                     self.labels[1][self.labels[0]].place(x=xlineposition - 2,
                            y=355)
199                     self.labels[0] += 1
200
201                 self.create_line(xlineposition, 390, xlineposition, 385,
                        width=1, fill="white")
202
203             if xlineposition % 7 == 0:
204                 self.update()
205
206             xlineposition += 1
207
208         self.bdrawing = False
209
210     def change_scale(self, badd):
211         """
212         Масштабирование аудиодорожки.
213         :param badd: флаг для увеличения.
214         """
215         if self.signals_count == 0 or self.bdrawing:
216             return
217
218         self.pixel_values[0] += -1 if badd else 1
219
220         if self.pixel_values[0] >= len(self.pixel_values):
221             self.pixel_values[0] = len(self.pixel_values) - 1
222             if not badd:
223                 return
224
225         if self.pixel_values[0] < 1:
226             self.pixel_values[0] = 1
227             if badd:
228                 return
229
230         if badd:
231             xmul = self.pixel_values[self.pixel_values[0] + 1] / self.
                    pixel_values[self.pixel_values[0]]
232         else:
233             xmul = 1 / (self.pixel_values[self.pixel_values[0]] / self.
                    pixel_values[self.pixel_values[0] - 1])
234
235         self.pixels_per_second = 60 // self.pixel_values[self.pixel_values
                [0]]
236         self.signals_count = int(self.player.audio.duration * self.
                pixels_per_second)
237
238         xend = self.e_line.x * xmul
239         if xend > self.signals_count - 4:
240             xend = self.signals_count - 4
241
242         self.change_lines_position(self.player.start, self.player.end, self.
                s_line.x * xmul, xend, False)
243
244         self.t_line.change_position(self.s_line.x, False)
```

```
245         self.t_line.interval = self.pixel_values[self.pixel_values[0]] / 60
246 
247         self.draw(btime=True)
248 
249     def reset_lines(self, bstart, bend, btime):
250         """
251         Событие сброса позиции линий.
252         :param bstart: флаг для сброса линии начала границы области аудио.
253         :param bend: флаг для сброса линии конца границы области аудио.
254         :param btime: флаг для сброса линии отметки времени области аудио.
255         """
256         if self.signals_count == 0:
257             return
258 
259         if self.loaded:
260             if bstart:
261                 self.s_line.change_position(0)
262 
263             if bend:
264                 self.e_line.change_position(self.signals_count - 4)
265 
266             if btime:
267                 self.t_line.pause()
268                 self.t_line.change_position(0, True)
269                 self.t_line.interval = self.pixel_values[self.pixel_values
                        [0]] / 60
```

command.py

```
class CommandBuffer:
    def __init__(self):
        self.buffer = [0] * 3
        self.index = 0

    def add(self, command):
        """
        Добавление команды в буфер.
        """
        if self.index >= len(self.buffer):
            self.extend()

        self.buffer[self.index] = command
        command.do()

        if self.index < len(self.buffer) and self.buffer[self.index] is not
            None:
            self.clean()

    def clean(self):
        """
        Очищение буфера команд.
        """
        self.buffer[self.index:] = [0] * (len(self.buffer) - self.index)

    def extend(self):
        self.buffer = self.buffer[1:] + [0]
        self.index = len(self.buffer) - 1


command_buffer = CommandBuffer()


class Command:
    def __init__(self, audio, start_index, end_index):
        self.audio = audio

        self.start_index = start_index
        self.end_index = end_index

        self.buffer = self.audio.signals_data[self.start_index: self.
            end_index]

    def undo(self):
        """
        Отмена совершенного изменения аудио.
        """
        if command_buffer.index <= 0:
            return False

        command_buffer.index -= 1
        return True

```

```
52      def do(self):
53          """
54          Совершение изменения аудио.
55          """
56          if command_buffer.index >= len(command_buffer.buffer):
57              return False
58
59          command_buffer.index += 1
60          return True
```

eline.py

```
from tkinter import Frame


class EdgeLine(Frame):
    def __init__(self, x, **keyargs):
        Frame.__init__(self, **keyargs)

        self.x = x
        self.place(x=self.x)

    def change_position(self, x):
        """
        Смена позиции линии.
        :param x: коррдината x.
        """
        self.x = x
        self.place(x=self.x)
```

tline.py

```
from tkinter import Frame, Label, StringVar
from threading import Thread, Event
from time import sleep


class TimeLine(Frame):
    def __init__(self, s_line, e_line, player, x, interval, **keyargs):
        Frame.__init__(self, **keyargs)

        self.s_line = s_line
        self.e_line = e_line

        self.player = player

        self.x = x
        self.place(x=self.x)

        self.time = 0.0
        self.interval = interval

        self.event = Event()
        self.event.clear()
        self.bend = False

        self.thread = Thread(target=self.movement, args=(self.event,))
        self.thread.start()

        self.label = TimeLineLabel(master=self.master)

    def start(self):
        """
        Запуск движения линии.
        """
        self.bend = False
        self.event.set()
        if not self.player.bplaying:
            self.player.bplaying = True

    def pause(self):
        """
        Остановка движения линии.
        """
        self.event.clear()
        if self.player.bplaying:
            self.player.bplaying = False

    def change_position(self, x, bchange):
        """
        Смена позиции линии.
        :param x: координата x.
        :param bchange: флаг для смены текста надписи.
        """
        if x < self.s_line.x:
```

```
54              x = self.s_line.x
55
56          if x >= self.e_line.x:
57              x = self.e_line.x
58              self.bend = True
59          else:
60              self.bend = False
61
62          self.x = x
63          self.place(x=self.x)
64
65          if self.master.signals_count < 1:
66              xpos = 0.0
67          else:
68              xpos = x * self.player.audio.duration / self.master.signals_count
69
70          self.time = xpos
71          self.player.current = self.time
72
73          if bchange:
74              self.label.set(self.x + 5, self.time)
75
76      def movement(self, event):
77          """
78          Движение линии.
79          :param event: событие.
80          """
81          while True:
82              sleep(self.interval)
83              self.event.wait()
84
85              self.label.set(self.x + 5, self.time)
86
87              if self.x >= self.e_line.x:
88                  self.pause()
89                  self.bend = True
90              else:
91                  self.change_position(self.x + 1, True)
92
93
94  class TimeLineLabel(Label):
95      def __init__(self, **keyargs):
96          Label.__init__(self, **keyargs)
97
98          self.string = StringVar()
99          self.place(x=5, y=330)
100
101         self.config(textvariable=self.string, width=0, bg="#1E1F22", fg="
                white")
102         self.string.set("0.0")
103
104     def set(self, x, text):
105         """
106         Изменение текста надписи и ее позиции.
```

```
107         :param x: координата x.
108         :param text: текст надписи.
109         """
110         dint, dloat = f"{text}".split(".")
111         self.string.set(dint + "." + dloat[:1])
112         self.place(x=x)
```

window.py

```
from canvas import ScrollableCanvas
from tkinter import Tk, Frame, Canvas, Scrollbar


# Поудалять ненужные объявления?
class Window(Tk):
    def __init__(self, player):
        Tk.__init__(self)

        self.title(".wav Editor")
        self.config(bg="#1E1F22")
        self.iconbitmap("music_icon.ico")
        self.geometry(f"{self.winfo_screenwidth() - 500}x500+0+0")
        self.resizable(True, False)

        self.scrollbar = Scrollbar(master=self, orient="horizontal")
        self.scrollbar.pack(side="top", anchor="n", fill="x")

        self.canvas = Canvas(
            master=self, height=400,
            xscrollcommand=self.scrollbar.set, bg="#1E1F22"
        )
        self.scrollbar.config(command=self.canvas.xview)

        self.scrollable_frame = Frame(master=self.canvas)
        self.canvas.create_window(0, 0, window=self.scrollable_frame, anchor=
            "nw")

        self.scrollable_canvas = ScrollableCanvas(
            player=player,
            master=self.scrollable_frame,
            width=0, height=0,
            bg="#1E1F22", bd=0,
            highlightthickness=0
        )
        self.frame = Frame(master=self, bg="#1E1F22")
        self.canvas.pack(side="top", anchor="n", fill="x")
        self.frame.pack(side="top", anchor="n", fill="both")
```