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, проектирование архитектуры программной системы, проектирование пользовательского интерфейса программной системы.

4.4. Рабочий проект: спецификация классов программной системы, тестирование программной системы.

4.5. Заключение

4.6. Список использованных источников

5. Приложения

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 63 страницам. Работа содержит 28 иллюстраций, 5 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 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 63 pages. The work contains 28 illustrations, 5 tables and 11 bibliographic sources. The number of applications is 1. The program code is presented in annex A.

List of keywords: system, program, editor, interface, user, sound, audio file, 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, 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

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.

ЛКМ - левая кнопка мыши

ПКМ - правая кнопка мыши

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 извлечения 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 аудиоданных;
- набор инструментов для манипуляции аудиоданными;
- возможность сохранения аудиофайла;

Композиция программы представлена на рисунке 2.1.

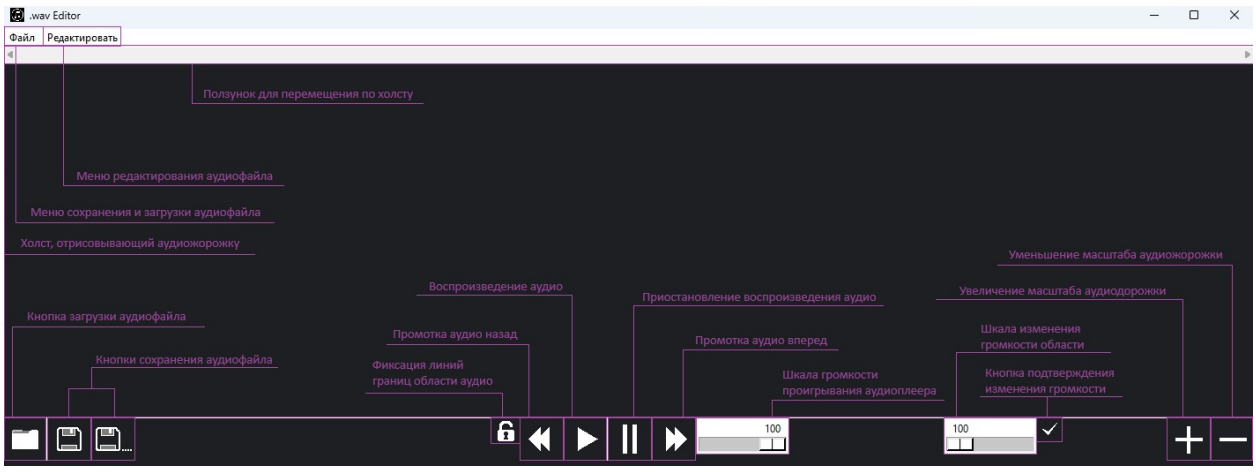

Рисунок 2.1 – Композиция интерфейса программы

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 актеру, который взаимодействует 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, понижение громкости и добавление эффектов нарастания и затухания.

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

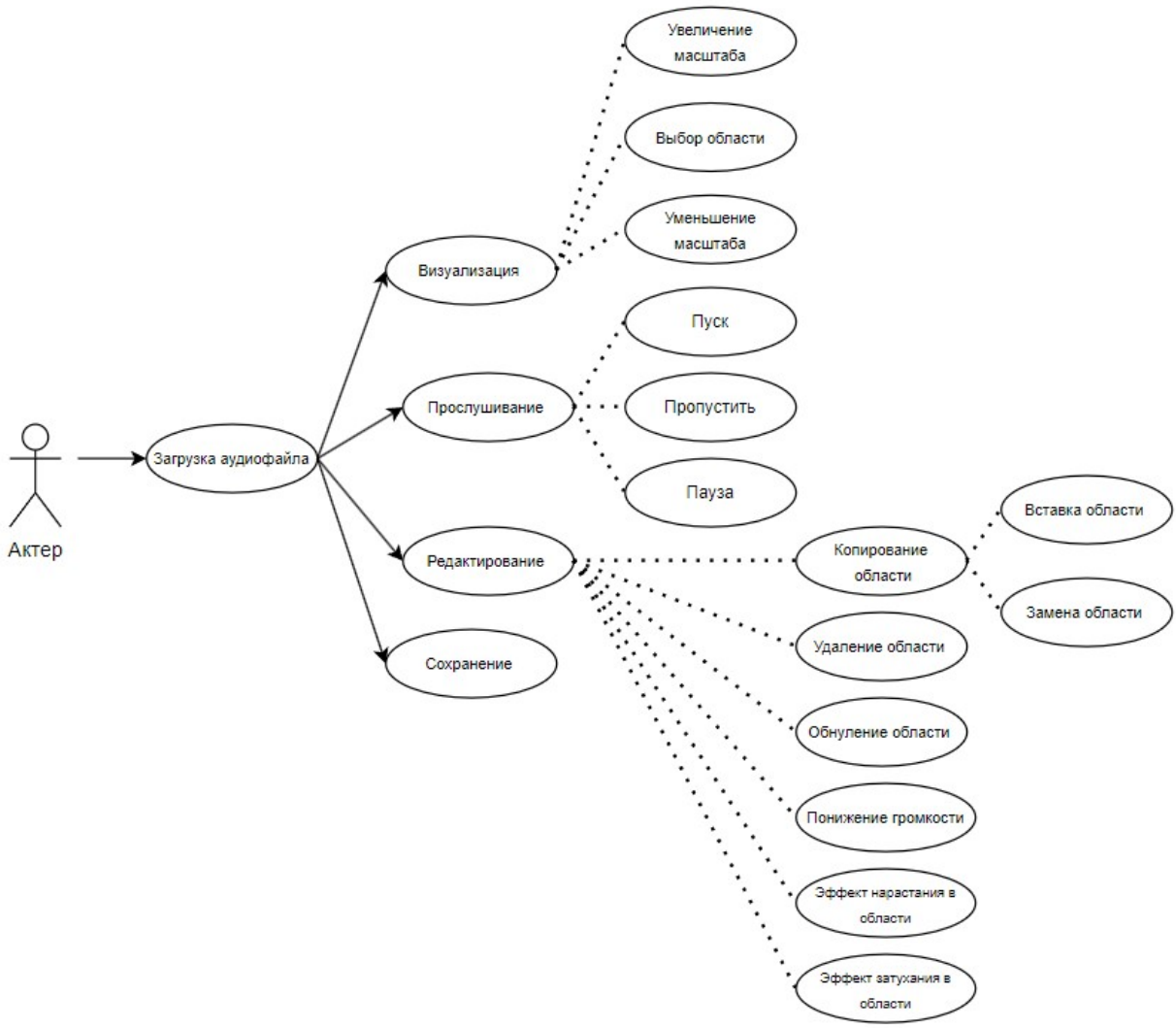

Рисунок 2.2 – Диаграмма прецедентов

2.4.1 Вариант использования «Загрузка»

Заинтересованные лица и их требования: пользователь желает загрузить аудиофайл для редактирования.

Предусловие: программа запущена, форматом аудиофайла является WAV.

Постусловие: аудио успешно загружено и подготовлено для последующего редактирования.

Основной успешный сценарий:

1. Пользователь нажимает на кнопку загрузки.
2. Программа открывает окно просмотра диска.
3. Пользователь выбирает аудиофайл и подтверждает загрузку.
4. Программа загружает аудиофайл и его данные в буфер.

2.4.2 Вариант использования «Увеличение/уменьшение масштаба»

Заинтересованные лица и их требования: пользователю нужно более подробное визуальное представление аудиоданных.

Предусловие: программа запущена, загружен аудиофайл формата WAV, аудиодорожка успешно отрисована.

Постусловие: масштаб аудиодорожки изменен.

Основной успешный сценарий:

1. Пользователь нажимает на кнопку увеличения/уменьшения масштаба.

2. Программа перерисовывает аудиодорожку с большим/меньшим масштабом.

2.4.3 Вариант использования «Выбор области»

Заинтересованные лица и их требования: пользователь хочет выбрать область для редактирования аудиоданных.

Предусловие: программа запущена, загружен аудиофайл формата WAV, аудиодорожка успешно отрисована.

Постусловие: установлена область редактирования.

Основной успешный сценарий:

1. Пользователь нажимает ЛКМ и ПКМ для выбора границы начала и конца области соответственно.

2. Программа устанавливает область редактирования в соответствие с запросами пользователя.

2.4.4 Вариант использования «Пуск»

Заинтересованные лица и их требования: пользователь хочет воспроизвести аудио.

Предусловие: программа запущена, загружен аудиофайл формата WAV.

Постусловие: воспроизведение аудио.

Основной успешный сценарий:

1. Пользователь нажимает на кнопку Пуск.
2. Программа начинает воспроизведение аудио.

2.4.5 Вариант использования «Пауза»

Заинтересованные лица и их требования: пользователь хочет остановить воспроизведение аудио.

Предусловие: программа запущена, загружен аудиофайл формата WAV, начато воспроизведение аудио.

Постусловие: остановка воспроизведения аудио.

Основной успешный сценарий:

1. Пользователь нажимает на кнопку Пауза.
2. Программа останавливает воспроизведение аудио.

2.4.6 Вариант использования «Пропустить»

Заинтересованные лица и их требования: пользователь хочет пропустить несколько секунд аудио.

Предусловие: программа запущена, загружен аудиофайл формата WAV.

Постусловие: перемещение по аудиопотоку.

Основной успешный сценарий:

1. Пользователь нажимает на кнопку перемотки аудио вперед/назад.
2. Программа смещает момент начала воспроизведения аудио.

2.4.7 Вариант использования «Копирование»

Заинтересованные лица и их требования: пользователь хочет скопировать выделенную область аудио.

Предусловие: программа запущена, загружен аудиофайл формата WAV, выбрана необходимая область.

Постусловие: копирование выбранной области.

Основной успешный сценарий:

1. Пользователь открывает меню редактирования и выбирает вариант «Копировать».

2. Программа успешно загружает выбранную область в буфер копирования.

2.4.8 Вариант использования «Вставить»

Заинтересованные лица и их требования: пользователь хочет вставить скопированную оласть.

Предусловие: программа запущена, загружен аудиофайл формата WAV, скопирована желаемая и выбрана необходимая области.

Постусловие: вставка скопированной области.

Основной успешный сценарий:

1. Пользователь открывает меню редактирования и выбирает вариант «Вставить».

2. Программа успешно вставляет скопированную пользователем область.

2.4.9 Вариант использования «Заменить»

Заинтересованные лица и их требования: пользователь хочет заменить выбранную область скопированной.

Предусловие: программа запущена, загружен аудиофайл формата WAV, скопирована желаемая и выбрана необходимая для замены области.

Постусловие: замена выбранной области скопированной.

Основной успешный сценарий:

1. Пользователь открывает меню редактирования и выбирает вариант «Заменить».

2. Программа успешно заменяет выбранную область на скопированную пользователем.

2.4.10 Вариант использования «Удаление»

Заинтересованные лица и их требования: пользователь хочет пропустить несколько секунд аудио.

Предусловие: программа запущена, загружен аудиофайл формата WAV, выбрана необходимая область.

Постусловие: удаление выбранной области.

Основной успешный сценарий:

1. Пользователь открывает меню редактирования и выбирает вариант «Копировать».

2. Программа успешно загружает выбранную область в буфер копирования.

2.4.11 Вариант использования «Обнуление»

Заинтересованные лица и их требования: пользователь хочет обнулить сигналы выбранной области.

Предусловие: программа запущена, загружен аудиофайл формата WAV, выбрана необходимая область.

Постусловие: обнуление сигналов выбранной области.

Основной успешный сценарий:

1. Пользователь открывает меню редактирования и выбирает вариант «Обнулить».

2. Программа успешно обнуляет сигналы выбранной области.

2.4.12 Вариант использования «Понижение громкости»

Заинтересованные лица и их требования: пользователь хочет понизить громкость выбранной области.

Предусловие: программа запущена, загружен аудиофайл формата WAV, выбрана необходимая область и, на соответствующей шкале, желаемая громкость.

Постусловие: понижение громкости выбранной области.

Основной успешный сценарий:

1. Пользователь нажимает на кнопку подтверждения или открывает меню редактирования и выбирает вариант «Изменить громкость».
2. Программа успешно изменяет громкость выбранной области.

2.4.13 Вариант использования «Эффект нарастания/затухания»

Заинтересованные лица и их требования: пользователь хочет добавить эффект нарастания/затухания для выбранной области.

Предусловие: программа запущена, загружен аудиофайл формата WAV, выбрана необходимая область.

Постусловие: добавление эффекта для области.

Основной успешный сценарий:

1. Пользователь открывает меню редактирования и выбирает вариант «Нарастание» или «Затухание».
2. Программа успешно применяет соответствующий эффект для выбранной области.

2.4.14 Вариант использования «Сохранение»

Заинтересованные лица и их требования: пользователь хочет сохранить аудиоданные в аудиофайл.

Предусловие: программа запущена, загружен аудиофайл формата WAV.

Постусловие: сохранение аудиофайла.

Основной успешный сценарий:

1. Пользователь загружает аудиофайл.
2. Пользователь проводит все необходимые операции.
3. Пользователь выбирает один из вариантов сохранения, нажав на соответствующую кнопку.
4. Программа успешно записывает аудиоданные в файл формата WAV.

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 Основные компоненты

3.3.1 Main.py

Главный модуль программы. Объединяет в себе все остальные модули и реализует пользовательский интерфейс для взаимодействия с программой и всеми ее функциями.

3.3.2 Audio.py

Модуль, содержащий класс Audio, который является основным способом хранения и взаимодействия с аудиоданными. Также модуль включает в себя классы обработки объекта Audio, наследуемые от класса Command.

3.3.3 Audioplayer.py

Модуль, содержащий класс AudioPlayer, который является посредником при взаимодействии главного модуля программы и модуля Audio.py. Класс реализует базовые функции загрузки, воспроизведения и сохранения аудиоданных.

3.3.4 Canvas.py

Модуль, содержащий класс ScrollableCanvas, представляющий собой основной и единственный способ визуализации аудиоданных посредством создания аудиодорожки.

3.3.5 Command.py

Модуль, содержащий классы Command и CommandBuffer, реализующие обработку и запоминание выполненных над аудиоданными операций.

3.3.6 Eline.py

Модуль, содержащий класс EdgeLine, позволяющий взаимодействовать с аудиоданными путем выбора области для желаемого прослушивания или редактирования.

3.3.7 Tline.py

Модуль, содержащий классы TimeLine, позволяющий отслеживать текущую временную позицию проигрывания аудио.

3.3.8 Window.py

Модуль, содержащий класс Window, представляющий собой основу всего графического интерфейса программы.

3.4 Диаграммы компонентов

Диаграммы компонентов описывают особенности физического представления разрабатываемой системы. Они 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 модулей проектируемой системы. На рисунке 3.2 изображена диаграмма классов проектируемой системы.

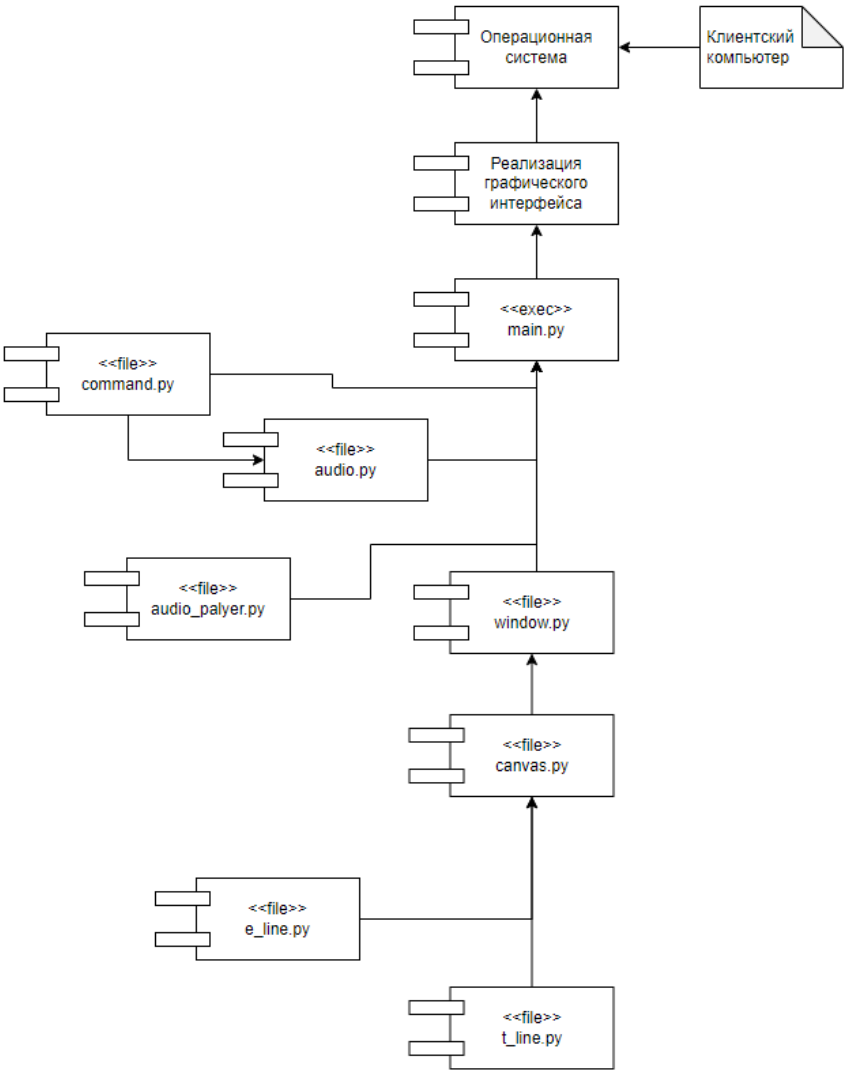

Рисунок 3.1 – Диаграмма компонентов

Основным исполняемым файлов является файл main.py, объединяющий в себе все другие компоненты. При запуске происходит создание гра-

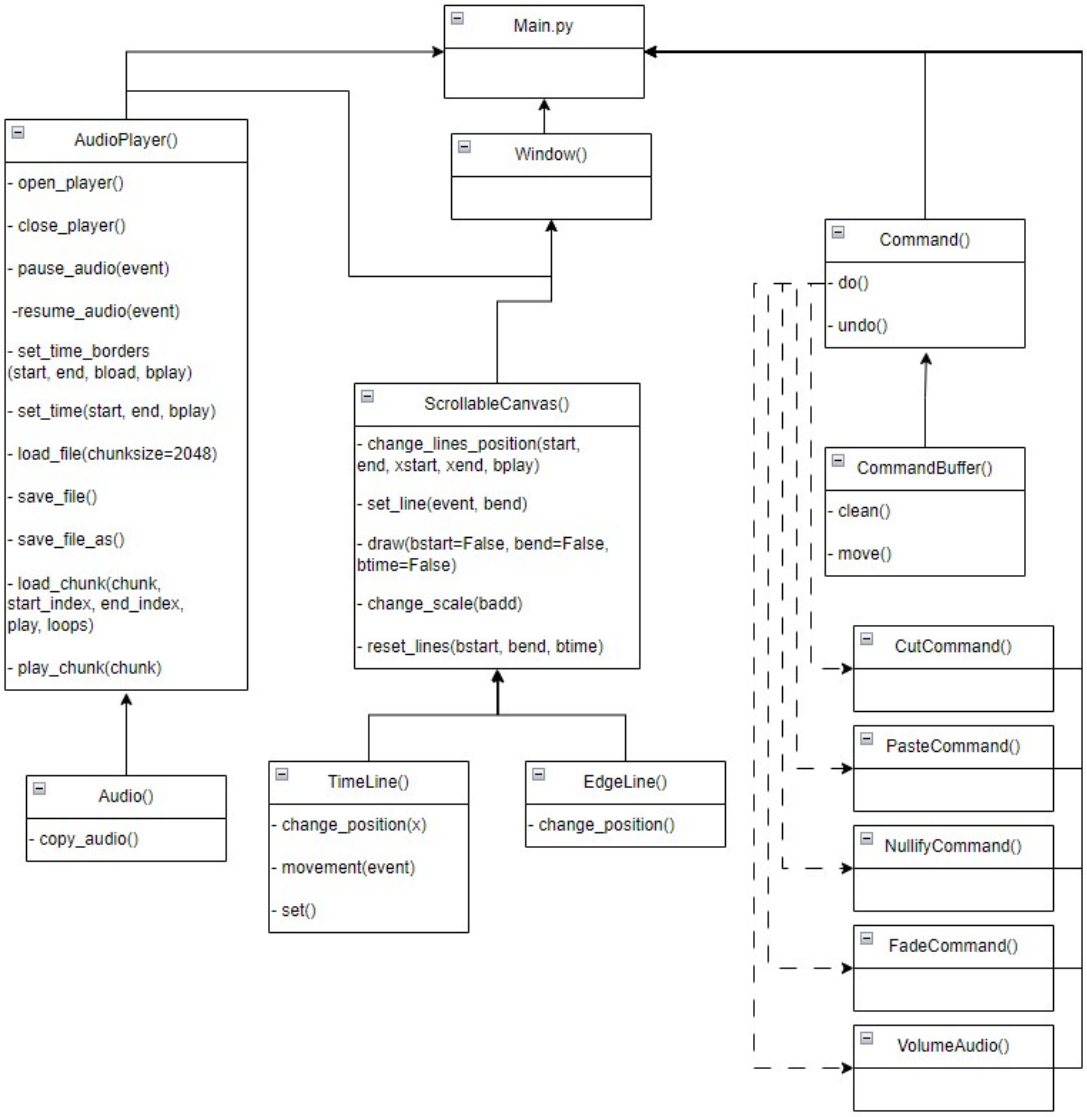

Рисунок 3.2 – Диаграмма классов

фичексого интерфейса, посредством которого пользователь может взаимодействовать с программой.

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, set() - смена текста надписи и ее положения
AudioPlayer	audioplayer.py	Класс для загрузки, воспроизведения и сохранения аудио	open player() - инициализация плеера SDL, close 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	add() – добавление операции в буфер, 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

Продолжение таблицы 4.1

1	2	3	4
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.

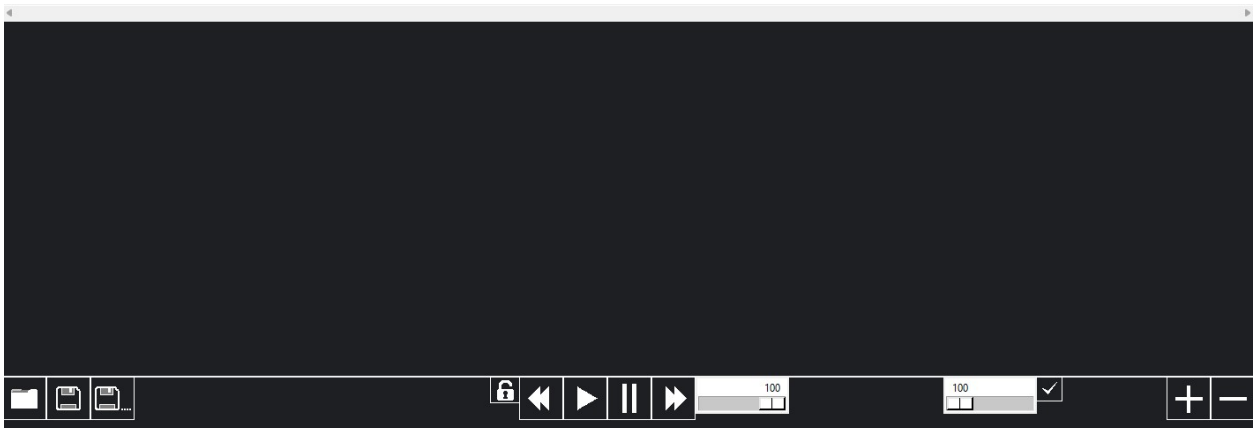

Рисунок 4.1 – Аудиодорожка до загрузки аудиофайла

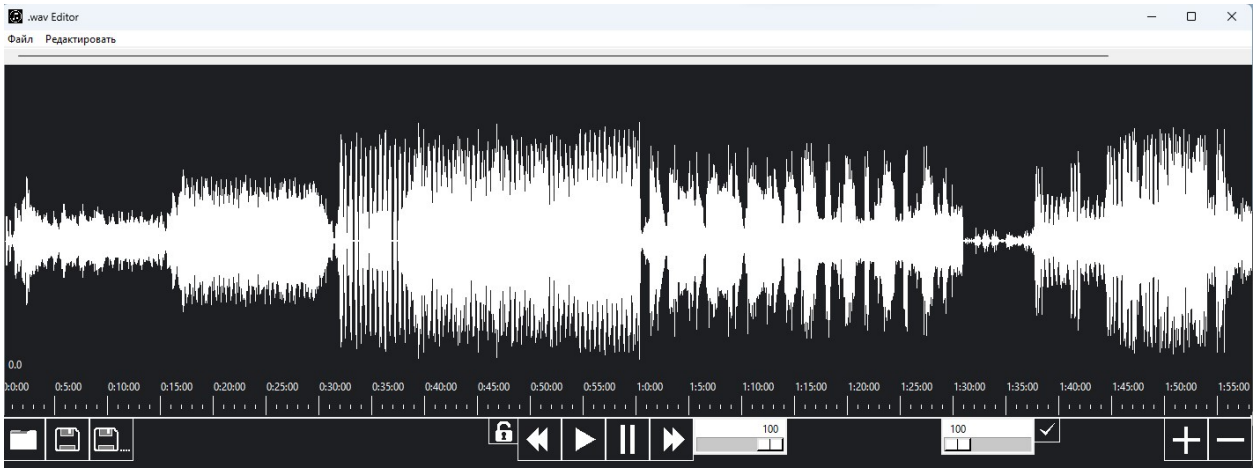

Рисунок 4.2 – Аудиодорожка после загрузки аудиофайла

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.

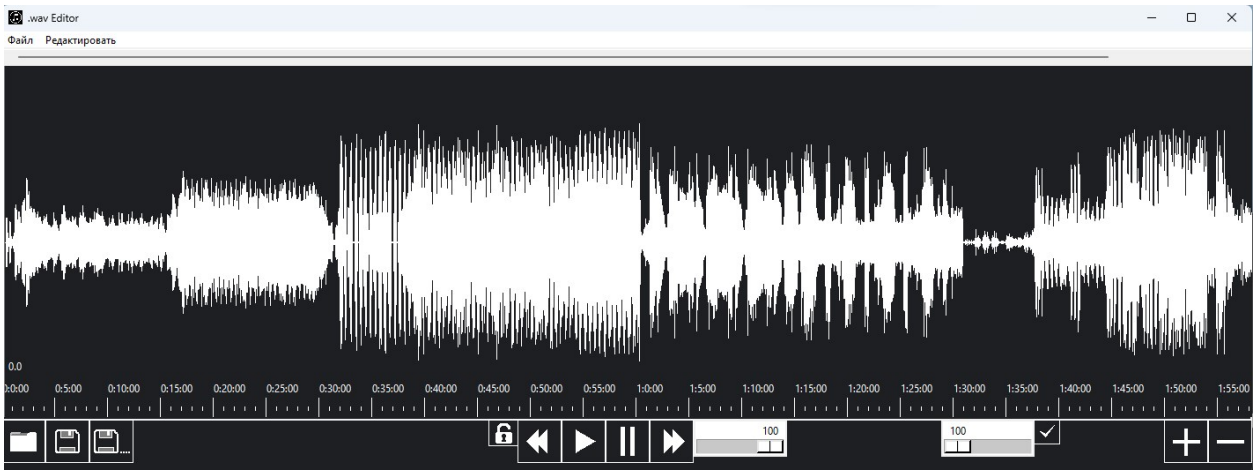

Рисунок 4.3 – Аудиодорожка до воспроизведения аудиофайла

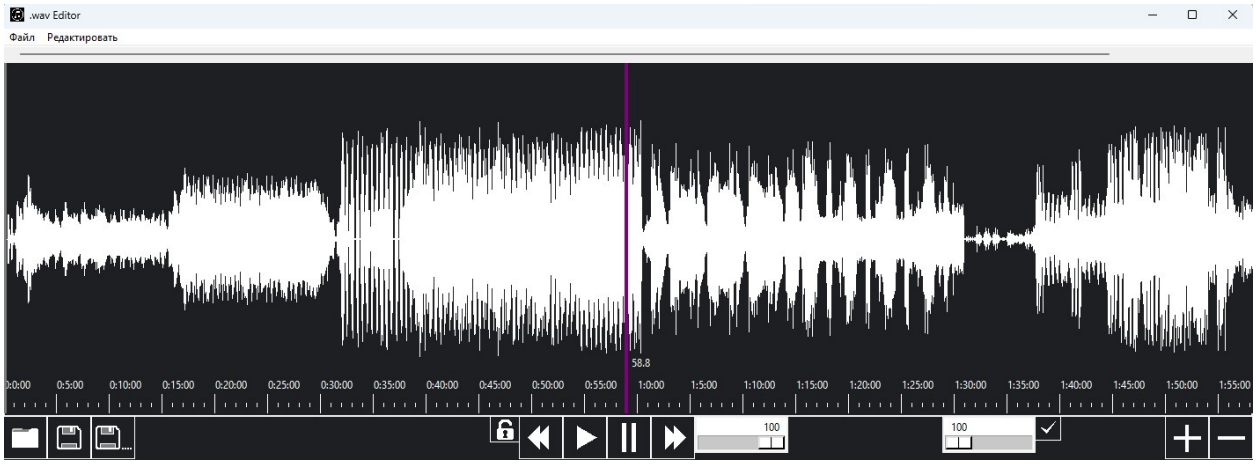

Рисунок 4.4 – Аудиодорожка после воспроизведения аудиофайла

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.

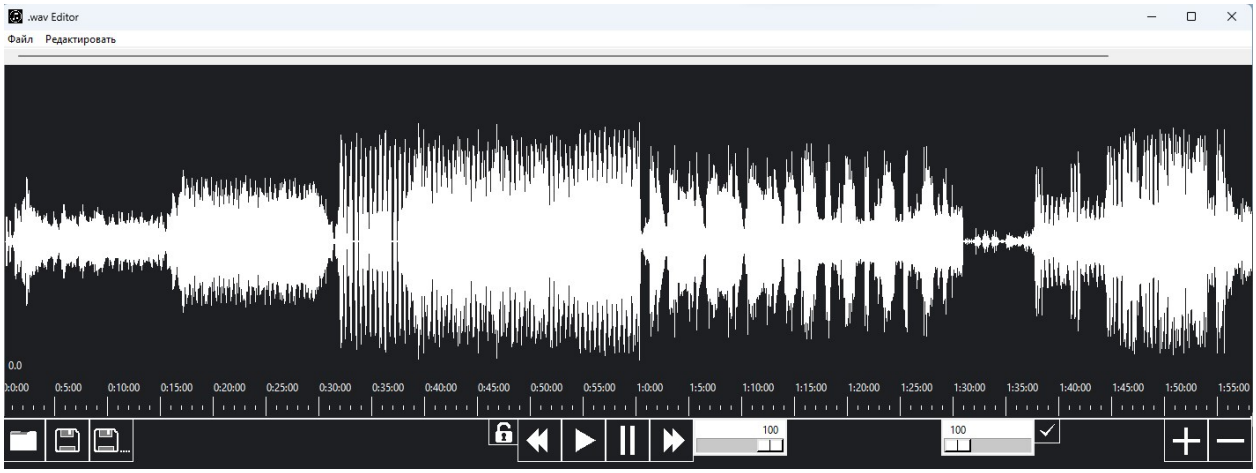

Рисунок 4.5 – Аудиодорожка до выбора области

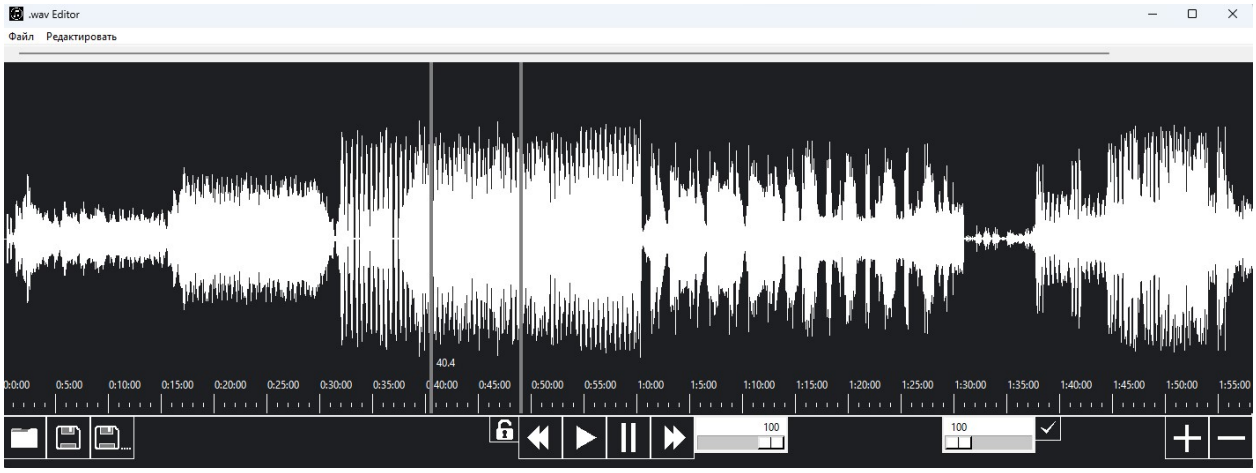

Рисунок 4.6 – Аудиодорожка после выбора области

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.

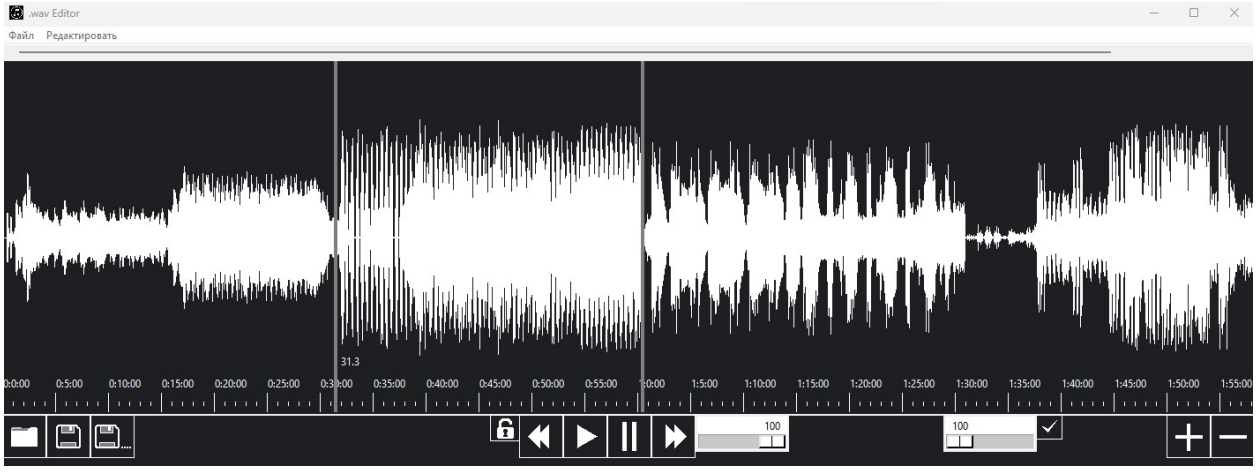

Рисунок 4.7 – Аудиодорожка до удаления области

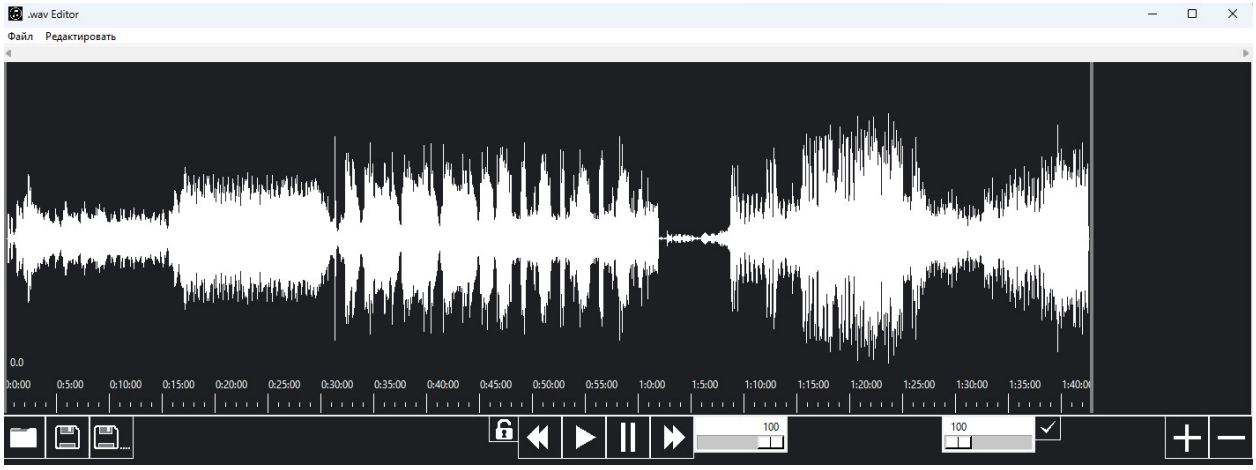

Рисунок 4.8 – Аудиодорожка после удаления области

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.

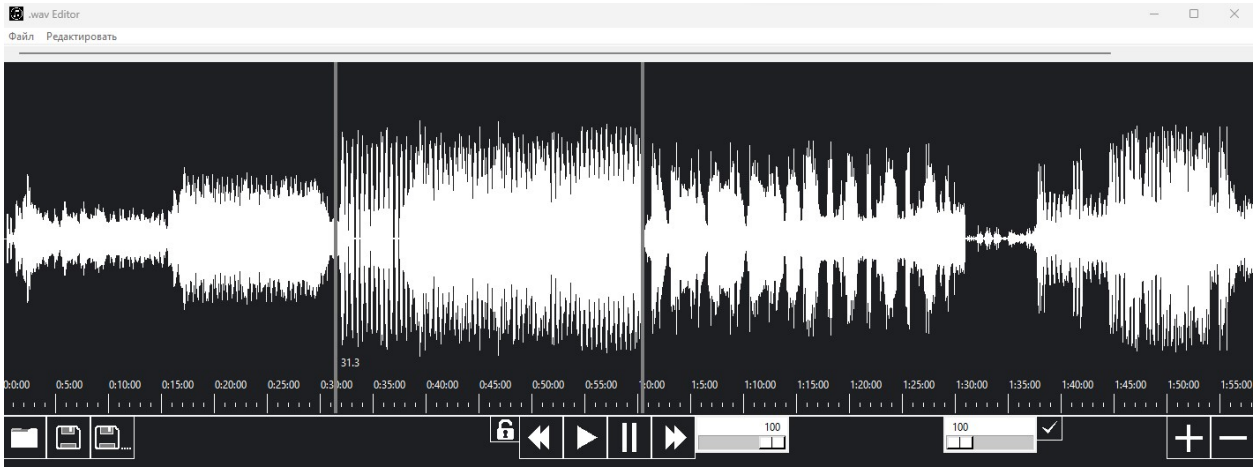

Рисунок 4.9 – Аудиодорожка до вставки области

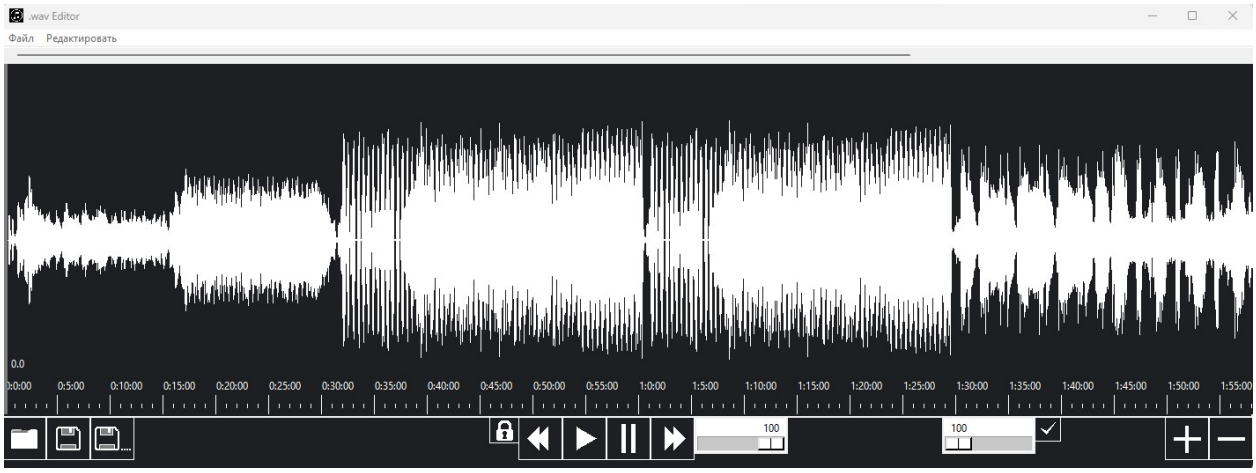

Рисунок 4.10 – Аудиодорожка после вставки области

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.

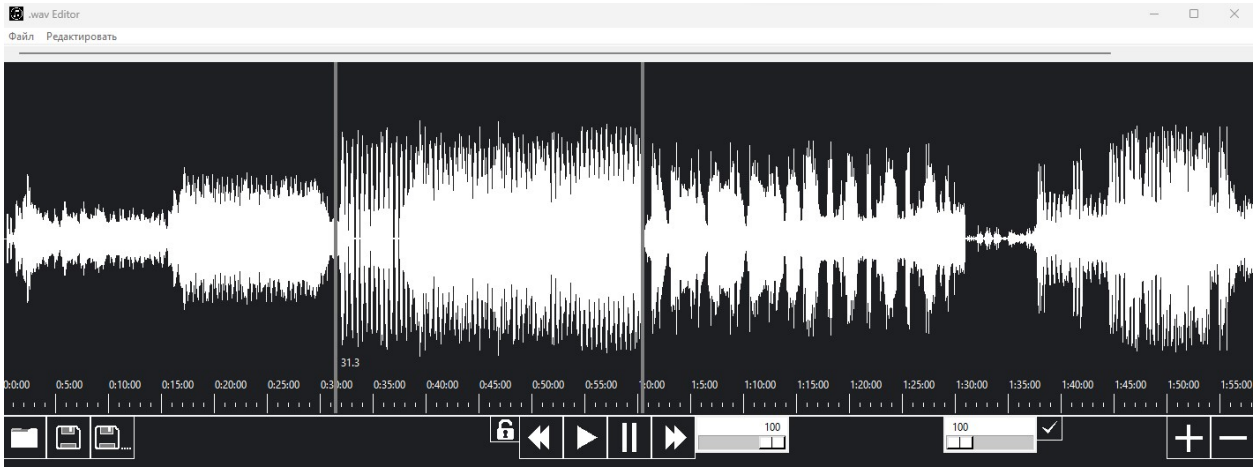

Рисунок 4.11 – Аудиодорожка до замены области

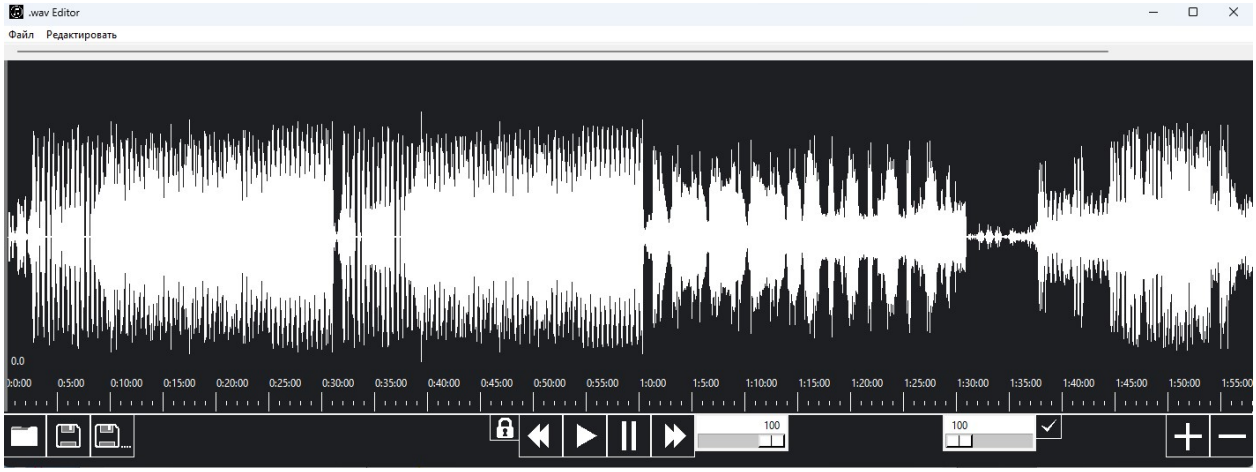

Рисунок 4.12 – Аудиодорожка после замены области

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.

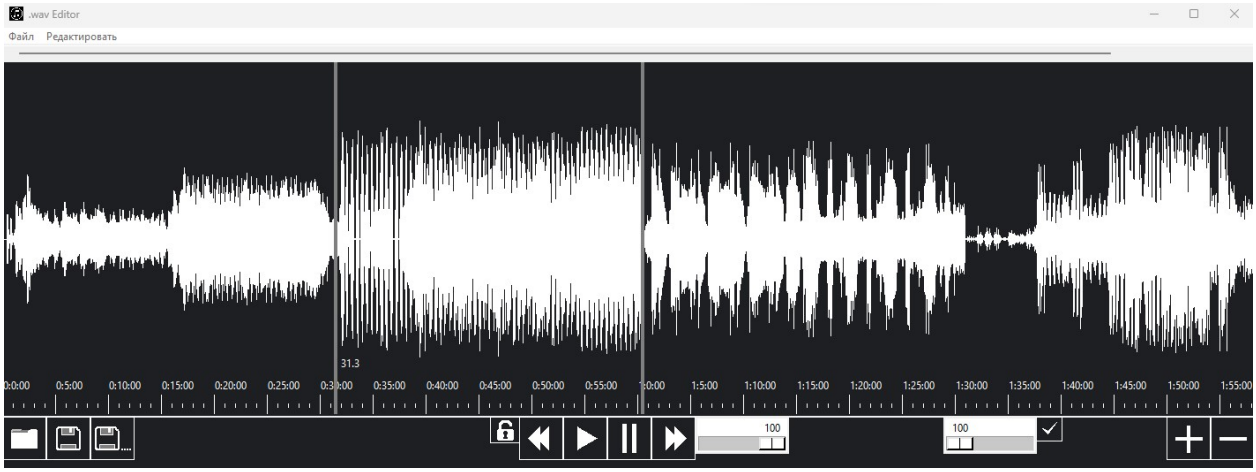

Рисунок 4.13 – Аудиодорожка до обнуления области

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.

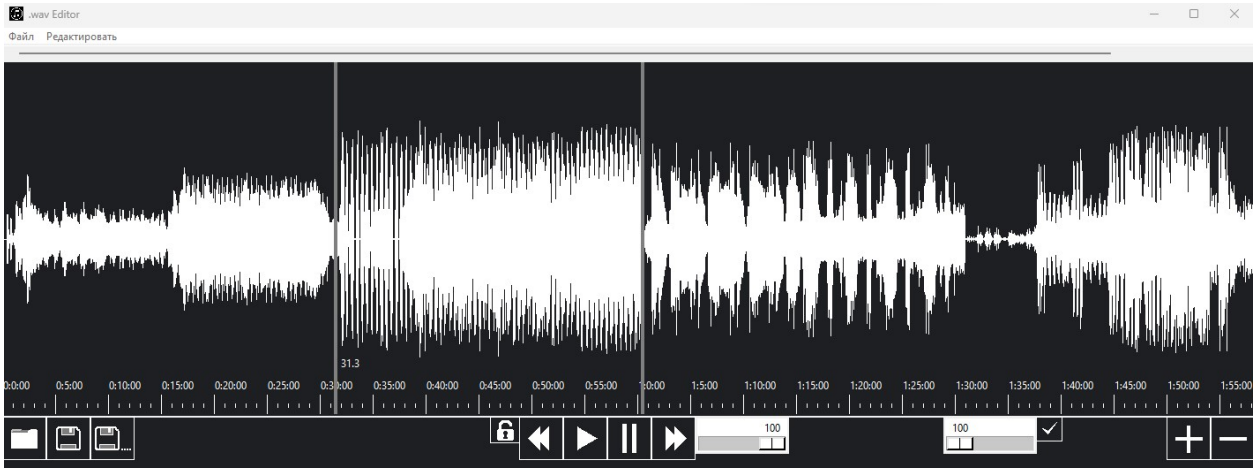

Рисунок 4.15 – Аудиодорожка до применения эффекта

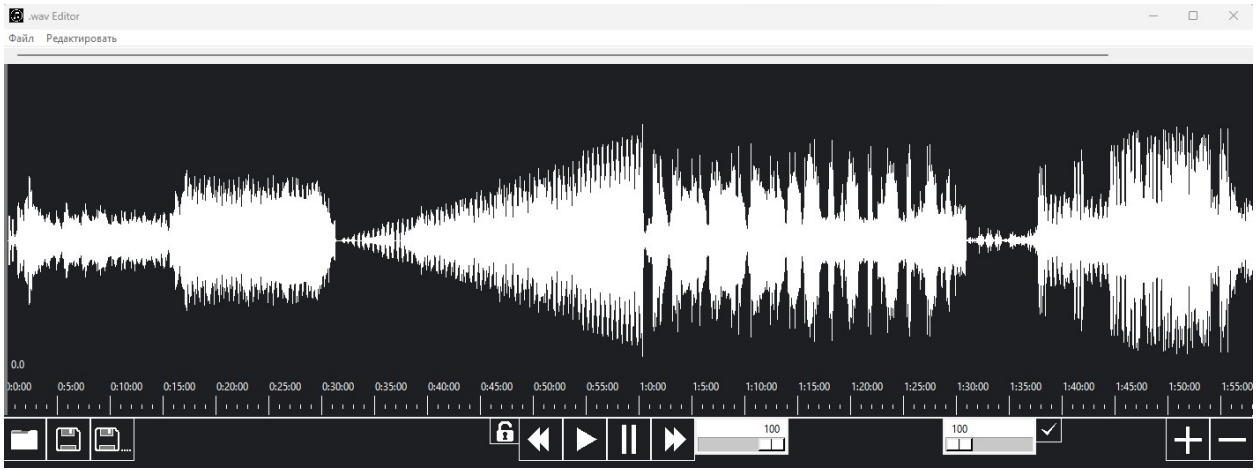

Рисунок 4.16 – Аудиодорожка после применения эффекта

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.

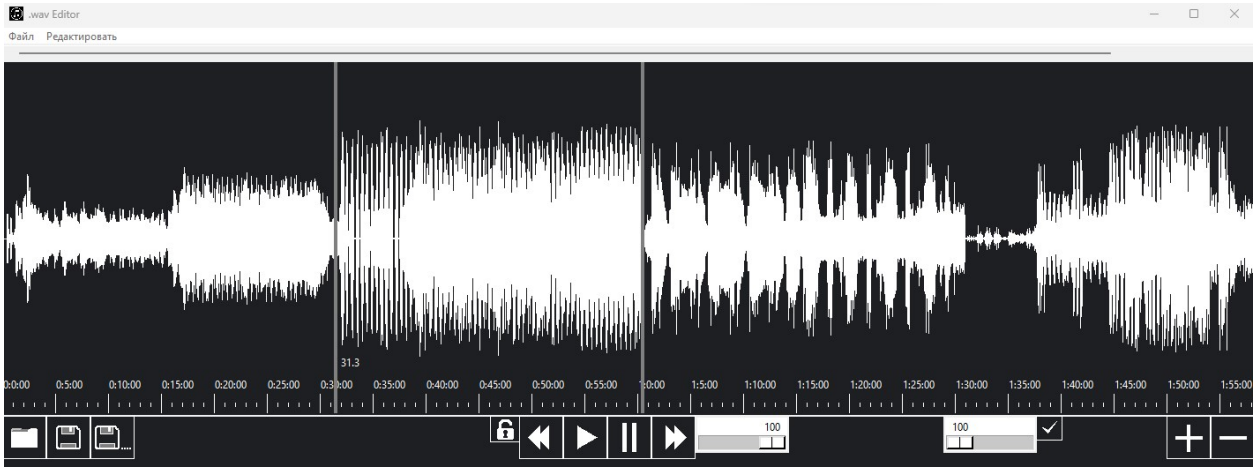

Рисунок 4.17 – Аудиодорожка до применения эффекта

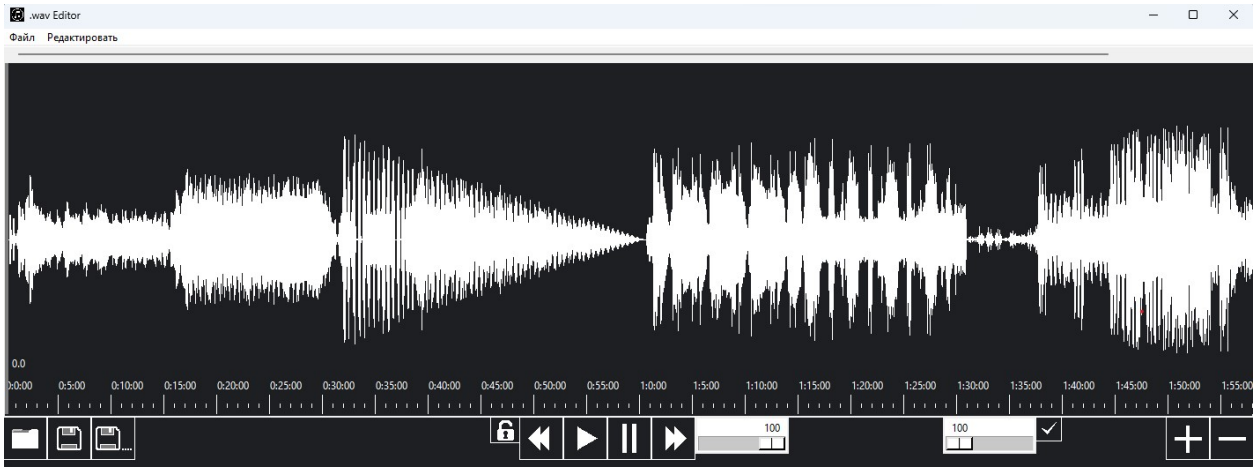

Рисунок 4.18 – Аудиодорожка после применения эффекта

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.

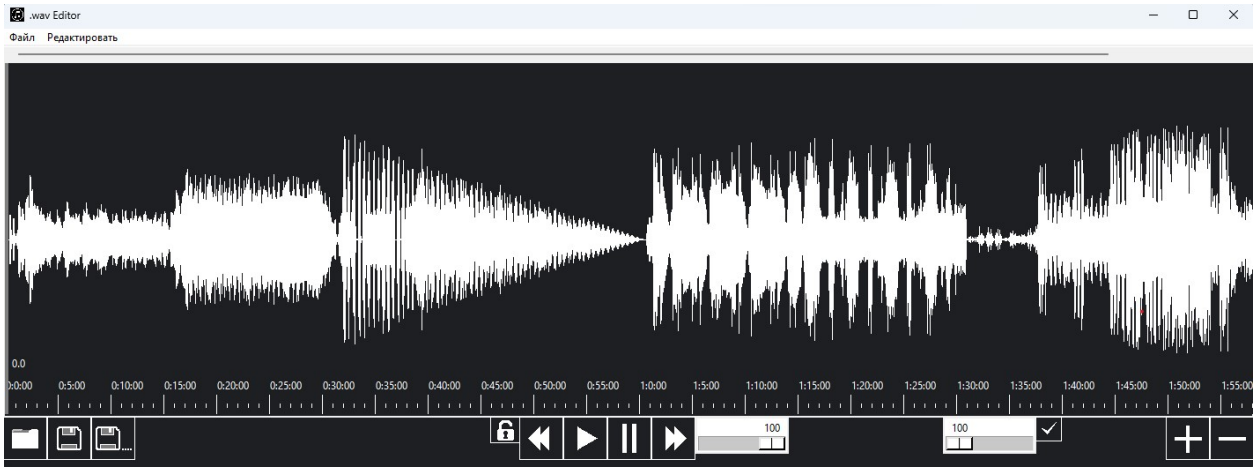

Рисунок 4.19 – Аудиодорожка до отмены операции

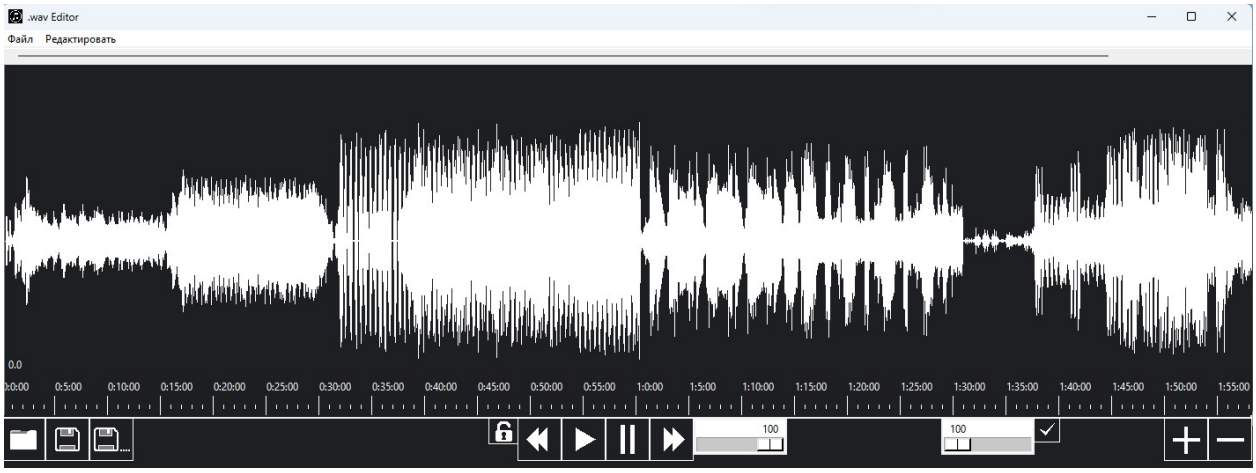

Рисунок 4.20 – Аудиодорожка после отмены операции

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.

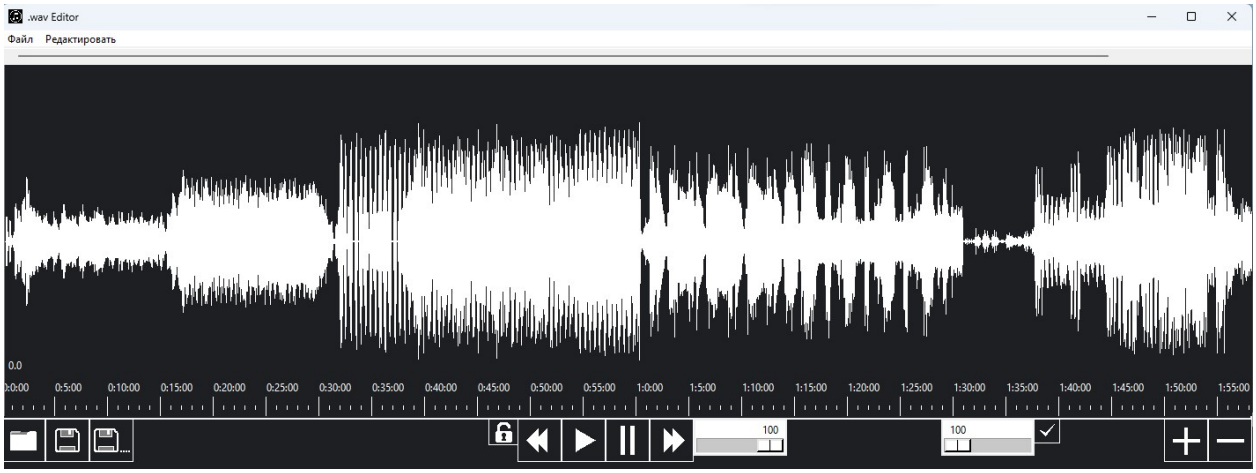

Рисунок 4.21 – Аудиодорожка до возврата операции

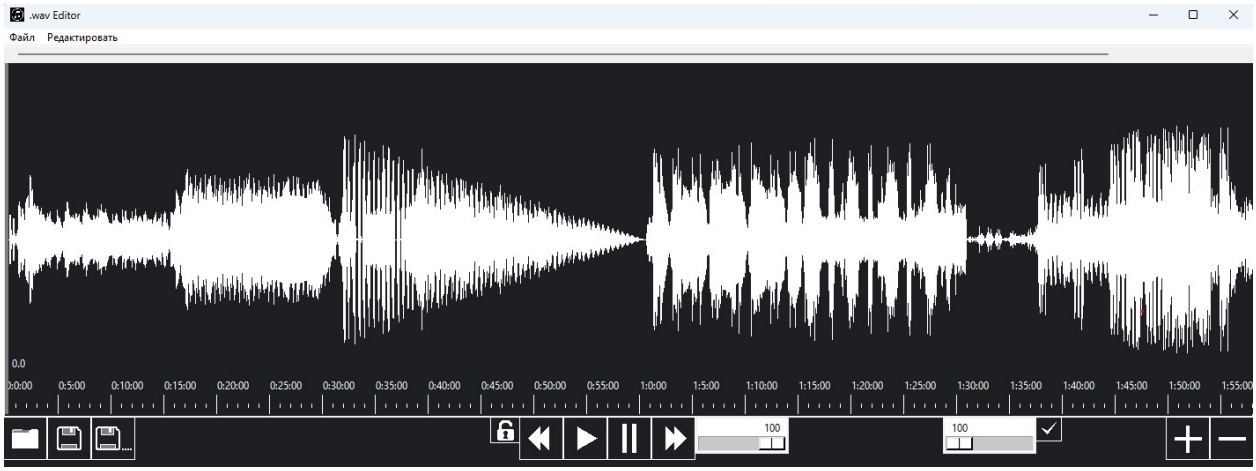

Рисунок 4.22 – Аудиодорожка после возврата операции

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.

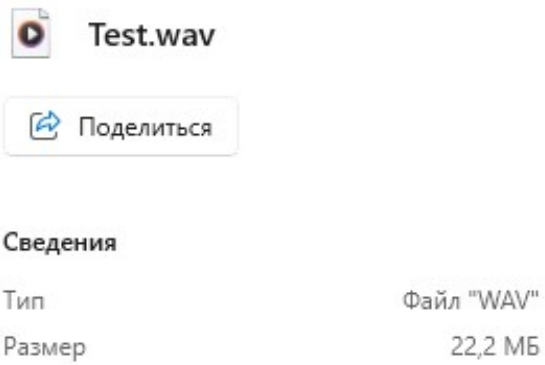

Рисунок 4.23 – Размер оригинального аудиофайла

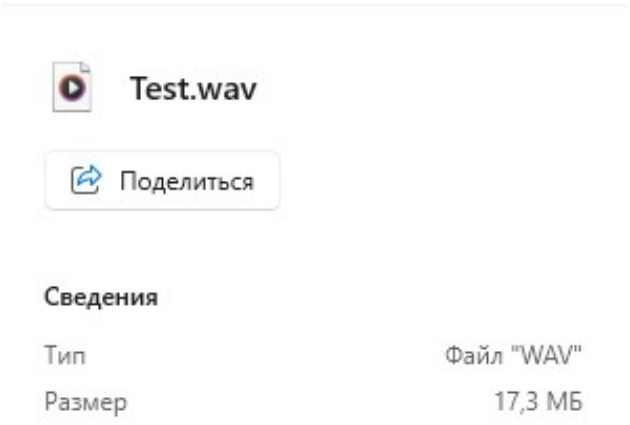

Рисунок 4.24 – Размер аудиофайла после сохранения с изменениями

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=bcopy, 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(
151                     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())
```

```
197
198     def show_edit_menu(event):
199         """
200         Событие вызова меню.
201         :param event: событие.
202         """
203         if canvas.bdrawing:
204             return
205
206         edit_menu.post(x=event.x_root, y=event.y_root)
207
208
209     canvas.bind("<ButtonPress-2>", show_edit_menu)
210
211     bimages = [
212         PhotoImage(file="load_button.png"),
213         PhotoImage(file="save_button.png"),
214         PhotoImage(file="save_loc_button.png"),
215         PhotoImage(file="play_button.png"),
216         PhotoImage(file="pause_button.png"),
217
218         PhotoImage(file="skipb_button.png"),
219         PhotoImage(file="skipf_button.png"),
220
221         PhotoImage(file="volume_button.png"),
222
223         PhotoImage(file="lock_button.png"),
224         PhotoImage(file="lock2_button.png"),
225
226         PhotoImage(file="less_button.png"),
227         PhotoImage(file="more_button.png")
228     ]
229
230     load_button = Button(
231         master=window,
232         image=bimages[0],
233         command=lambda: load_file(),
234         bd=0, highlightthickness=0
235     )
236     load_button.place(
237         relx=0, x=25, y=420, anchor="n"
238     )
239
240     save_button = Button(
241         master=window,
242         image=bimages[1],
243         command=lambda: player.save_file(),
244         bd=0, highlightthickness=0
245     )
246     save_button.place(
247         relx=0, x=75, y=420, anchor="n"
248     )
249
250     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"
    )
```

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

```
357     load_menu.add_command(label="Сохранить...", command=lambda: player.
            save_file_where())
358 
359     window_menu = Menu(tearoff=0)
360     window_menu.add_cascade(label="Файл", menu=load_menu)
361     window_menu.add_cascade(label="Редактировать", menu=edit_menu)
362 
363     window.config(menu=window_menu)
364     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(".")[0]
109                 self.audio.nchannels = params[0]
110                 self.audio.sampwidth = params[1]
111                 self.audio.framerate = params[2]
112                 self.audio.nframes = params[3]
113                 self.audio.comptype = params[4]
114                 self.audio.compname = params[5]
115                 self.audio.chunksize = chunksize
116                 self.audio.signals_data = array(frombuffer(wave_sample.
                        readframes(self.audio.nframes), int16))
117 
118             self.audio.duration = self.audio.nframes / self.audio.framerate
119             self.set_time_borders(0.0, self.audio.duration, True, False)
120 
121             self.open_player()
122         except:
123             return
124 
125     def save_file(self):
126         """
127         Запись аудиофайла на компьютер.
128         """
129         if not self.bactive:
130             return
131 
132         with open(self.audio.name + "_changed.wav", "wb") as wave_sample:
133             params = (
134                 self.audio.nchannels, self.audio.sampwidth, self.audio.
                        framerate, self.audio.nframes,
135                 self.audio.comptype, self.audio.compname
136             )
137             wave_sample.setparams(params)
138             wave_sample.writeframes(self.audio.signals_data.tobytes())
139 
140     def save_file_where(self):
141         """
142         Запись аудиофайла на компьютер с выбором названия и папки.
143         """
144         if not self.bactive:
145             return
146 
147         file = asksaveasfile(mode="w", defaultextension=".wav", filetypes=[("
                WAVE files", "*.wav")])
148 
149         if file is None:
```

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