АННОТАЦИЯ

Отчет о курсовой работе: 44 с., 12 рис., 13 табл., 2 приложения, 6 источников.

Объект исследования – звуковые сигналы в IP-телефонии.

Предмет исследования – модели и методы обработки и кодирования речевых сигналов.

Цель работы – разработка программы для визуализации, записи, воспроизведения, последующего анализа и сжатия звуковых данных.

Метод исследования – изучение особенностей цифровой обработки звука и алгоритмов сжатия речевых сигналов.

В работе были использованы технологии Windows Forms, .NET Framework, NAudio.

В результате решения задачи было разработано приложениепо анализу речевых сигналов для IP-телефонии.Приложение позволяетзаписывать, воспроизводить, визуализировать звуковой сигнал в различных представлениях, кодировать и сжимать аудиоданные.

Приложение может использоватьсядля записи звуковых данных, их анализаи последующего сжатия для IP-телефонии.

Дальнейшее развитие программы связано с расширением ее функционала, увеличением опций редактирования аудио, а так же работой с другими форматами файлов.

ЦИФРОВАЯ ОБРАБОТКА РЕЧЕВЫХ СИГНАЛОВ, АУДИО, КОДЕК, IP-ТЕЛЕФОНИЯ.

1 Анализ предметной области

1.1 Состояние вопроса

IP-телефония — это технология, позволяющая использовать Интернет или любую другую IP-сеть для ведения телефонных разговоров и передачи факсов в режиме реального времени. Особенно актуально, с экономической точки зрения, использование данной технологии для осуществления международных и междугородных телефонных разговоров или для создания распределенных корпоративных телефонных сетей.

Для кодирования звуковой информации обычно используются следующие кодеки: G.711, G.722, GSM0610, G.723, G.723.1, G.728, и G.729.

Так как реализация всех кодеков в приложении слишком трудоемкая работа и в этом нет особой необходимости, было принято решение остановиться на каком-то одном.

Для этого был проведен анализ всех кодеков. В таблице 1.1 приведена сравнительная характеристика.

Таблица 1.1 – Сравнительная характеристика VoIP-кодеков

Кодек	Полезная нагрузка пакета байт	Скорость передачи, кбит/с.	Алгоритмическая задержка, миллисекунд
G.711	160	64	20
G.723.1 (6.3)	24	6,3	37,5
G.723.1 (5.3)	20	5,3	37,5
G.726-32	160	32	20
G.726-24	160	24	20
G.726-16	160	16	20
G.729 (8)	20	8	25
G.729 (6.4)	16	6,4	25

Исходя из результатов сравнения, был выбран кодек G.711.

G.711 — голосовой кодек, который не предполагает никакого сжатия, помимо компандирования — метода уменьшения эффектов каналов с ограниченным динамическим диапазоном. В основе данного метода лежит принцип уменьшения количества уровней квантования сигнала в области высокой громкости, сохраняя при этом качество звука. Две широко использующиеся в телефонии схемы компандирования — a-law и μ-law.

Сигнал в данном кодеке предоставлен потоком величиной 64 кбит/с. Частота дискретизации — 8000 кадров по 8 бит в секунду. Качество голоса субъективно лучше, нежели при применении кодека G.729.

Мю-закон (μ-law):

$$F(x) = \operatorname{sgn}(x)\frac{\ln(1+\mu|x|)}{\ln(1+\mu)} \qquad -1 \leqslant x \leqslant 1 \tag{1.1}$$

где μ принимается равным 255 (8 бит) в стандартах Северной Америки и Японии.

А-закон (A-law):

$$F(x) = \operatorname{sgn}(x)\begin{cases} \frac{A|x|}{1+\ln(A)}, & |x| < \frac{1}{A} \\ \frac{1+\ln(A|x|)}{1+\ln(A)}, & \frac{1}{A} \leq |x| \leq 1, \end{cases} \tag{1.2}$$

Где A — параметр сжатия (обычно принимается равным 87,7).

G.711 был выпущен в 1972 году. Его патент уже истек, поэтому он находится в свободном доступе.

1.2 Актуальность и цель работы

На сегодняшний день IP-телефония все больше вытесняет традиционные телефонные сети за счет легкости развертывания, низкой стоимости звонка,

простоты конфигурирования, высокого качества связи и сравнительной безопасности соединения.

Целью же данного проекта является создание удобного средства для кодирования звука в G.711, при этом которое бы располагало всеми необходимыми функциями звукового редактора и являлось бы бесплатным.

На основании всего написанного выше можно сделать вывод, что актуальность создаваемого продукта полностью оправдана. Применение систем IP-телефонии позволяет компаниям-операторам связи значительно снизить стоимость звонков (особенно международных) и интегрировать телефонию с сервисами Интернета, предоставлять интеллектуальные услуги. Однако, несмотря на популярность кодирования звуковых данных, в приложениях для работы со звуком поддержки этой операции практически нет.

Целью данного проекта является создание звукового редактора со всеми основными возможностями работы с аудиоданными (запись, воспроизведение, визуализация), а также возможностью сжимать и кодировать звуковой сигнал.

Исходя из указанной цели, можно выделить частные задачи, поставленные в данной работе:

1. Провести детальный анализ предметной области. Изучить особенности цифровой обработки звуковых сигналов. Изучить применяемые алгоритмы сжатия голоса при передаче по IP-сети.

2. Изучить технологии: Window Forms (создание графического интерфейса пользователя), C#, NAudio (.NET библиотека для работы с аудио), Microsoft Visual Studio.

3. Разработать структуру приложения и функционал программы соответствующий техническому заданию.

4. Разработать полностью рабочее приложение со всем запланированным функционалом.

2 Техническое задание

2.1 Описание области применения и исходных данных приложения

В системе должна обрабатываться и отображаться следующая информация:

1. Аудиофайлы (имя, длительность, частота дискретизации, разрядность, формат, путь)
2. Амплитудно временное представление звука
3. Амплитудный спектр в трех представлениях
4. Спектрограмма
5. Время выделенного фрагмента звуковой дорожки

Исходными данными для приложения являются:

1. Звук в формате MP3 (.mp3)
2. Звук в формате WAV (.wav)
3. Аналоговый звуковой сигнал с микрофона

2.2 Требования к пользовательским интерфейсам

Интерфейс должен предполагать стандартную системную цветовую палитру и разрабатываться под разрешение экрана 800x600 и более. Окна должны обладать системным меню с кнопкой закрытия.

Требования к окнам:

Главное окно, в котором должны быть разделы:

- Обзор выбранных файлов.

  На этой вкладке должны быть кнопки открытия, создания, удаления файла, а также список и информация по каждому аудиофайлу.
- Сжатие звука.

  Вкладка должна содержать в себе 3 раздела: информация о текущей аудиозаписи (имя, формат, размер), ресэмплинг (возможность

изменять такие параметры: частота, разрядность, количество каналов), компандирование G.711 (закодировать, декодировать).

- Запись звука.
  Тут должен быть расположен список всех доступных записывающий устройств, кнопка “Обновить” список, кнопки запуска и остановки записи и таймер.
- Выделено / Просмотр.
  Должно быть отражено начало, конец и длительность выделенного фрагмента.
- Уровнеграмма звука.
  Под ней должны быть расположены кнопки проигрывания, остановки, приостановки, переключения и перематывания аудиозаписи, полоса регулирования громкости и таймер. Также возможность зуммирования и выделения фрагмента звуковой дорожки.
- Амплитудный спектр.
  Спектр должен быть представлен в трех видах: графический (амплитуда, децибелы), диаграммный (амплитуда)
- Спектрограмма.
- Строка состояния системы.
  В строке состояния отображается текущий статус аудио (воспроизведение, приостановлено, остановлено), а также частота, размер и длительность аудиофайла.

2.3 Требования к аппаратным, программным и коммуникационным интерфейсам

Необходима поддержка микрофона, подключаемого к компьютеру через TRS-порт.

Необходимо обеспечить программное взаимодействие системы:

- с операционными системами Windows 7/XP/8/8.1

Программа должна занимать не более 64 Мб оперативной памяти.

Модули программы должны занимать не более 25 Мб памяти на жестком диске.

2.4 Требования к пользователям продукта

- Владение компьютером на уровне пользователя ОС Windows
- Навыки работы со звуковыми редакторами
- Понимание принципов цифровой обработки звука

2.5 Требования к адаптации на месте

Необходимы программы-установщики для осуществления развертывания модулей на соответствующих целевых компьютерах. Необходимо предоставить всем категориям пользователей справочную информацию.

Необходимо наличие на компьютере установленного Microsoft .NET Framework 4.6

2.6 Функции продукта

Основной функционал продукта:

1. Создание / Открытие / Редактирование / Удаление аудиофайлов
2. Запись речи с микрофона.
3. Сжатие звука.
4. Визуализация звука
5. Кодирование звука в G.711.

2.6.1 Сценарий «Открыть аудиофайл»:

1. Пользователь в главном окне нажимает на кнопку «Открыть аудиофайл».
2. Система открывает окно выбора файла с компьютера.
3. Пользователь выбирает нужный ему файл.
4. Если выбранный файл имеет неверный формат, программа выдает ошибку и останавливает сценарий.
5. Система добавляет в список выбранный файл, где выводится вся информация о данном аудиофайле.
6. Система отображает сигнал в амплитудно временном представлении, отрисовывает частотную шкалу во вкладке отображения спектра и ожидает дальнейших действий пользователя.

2.6.2 Сценарий «Удалить файл»:

1. Пользователь выделяет файл для удаления и нажимает на кнопку «Удалить аудиофайл».
2. Система удаляет из списка выбранный файл и очищает область с уровнеграммой, спектром и спектрограммой, если файл был выбран как текущий.

2.6.3 Сценарий «Создать аудиофайл»:

1. Пользователь в главном окне нажимает на кнопку «Создать аудиофайл».
2. Программа предлагает пользователю ввести имя аудиофайла и выбрать путь, по которому нужно произвести сохранение файла.
3. Система добавляет в список созданный файл и подсвечивает его синим цветом (пустой файл) и ожидает дальнейших действий пользователя.

2.6.4 Сценарий «Записать звук»:

1. Пользователь в главном окне открывает вкладку «Запись звука» и нажимает кнопку «Обновить» список записывающих устройств.
2. Система выводит список всех устройств.
3. Пользователь выделяет устройство и пустой файл, затем нажимает кнопку «Старт».
4. Система запускает таймер и получает входной сигнал с микрофона.
5. Пользователь нажимает кнопку «Стоп».
6. Система останавливает таймер, сохраняет записанный сигнал в выбранный файл и предлагает открыть аудио.
7. Если пользователь соглашается на открытие файла, то система отображает сигнал в амплитудно временном представлении и обновляет информацию о выбранном файле и ожидает дальнейших действий пользователя.

2.6.5 Сценарий «Кодировать аудиофайл»:

1. Пользователь в главном окне, в разделе компандирования выбирает кодек и нажимает на кнопку «Сохранить».
2. Система производит кодирование аудиофайла в кодек G.711
3. Если выбранный файл невозможно перекодировать, программа выдает ошибку и останавливает сценарий.
4. Система предлагает открыть закодированное аудио.
5. Если пользователь соглашается, система добавляет аудиофайл в список аудиофайлов и выводит его уровнеграмму.

2.6.6 Сценарий «Сжать аудиофайл»:

1. Пользователь вводит параметры для сжатия и нажимает кнопку «Сохранить wav».
2. Система предлагает ввести имя файла и выбрать путь для сохранения.
3. Пользователь вводит имя и нажимает кнопку «Ок».

4. Система предлагает открыть сжатое аудио.
5. Если пользователь соглашается, система добавляет аудиофайл в список аудиофайлов и выводит его уровнеграмму.

2.6.7Сценарий «Воспроизвести аудиофайл»:

1. Пользователь в главном окне нажимает на кнопку «Воспроизвести аудиофайл».
2. Система воспроизводит выбранный аудиофайл.
3. Если выбранный файл невозможно воспроизвести, программа выдает ошибку и останавливает сценарий.
4. Система выводит текущую частоту звука в реальном времени во вкладке спектра.

2.6.8 Сценарий «Выделить фрагмент звуковой дорожки»:

1. Пользователь во вкладке редактора выделяет нужный ему фрагмент.
2. Система выводит начало, конец и длительность выделенного фрагмента во вкладке «Выделено / Просмотр», в разделе «Выделено».
3. Пользователь прекращает выделение и отпускает левую кнопку мыши.
4. Система перерисовывает уровнеграмму для данного фрагмента и выводит начало, конец и длительность фрагмента во вкладке «Выделено / Просмотр», в разделе «Просмотр».

На рисунке 2.1 показана схема функциональной структуры приложения.

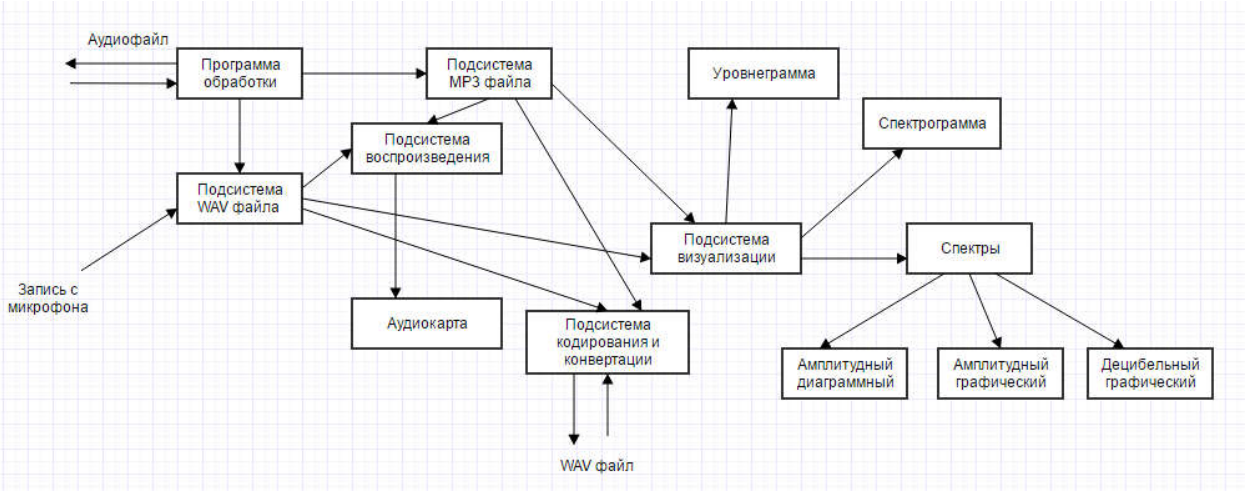

Рисунок 2.1 - Схема функциональной структуры приложения по работе с аудио

2.7Ограничения

- Должна использоваться кодировка UTF-8.
- Продукт будет поддерживать только русский язык пользовательского интерфейса.
- Продукт не предусматривает автоматического перехода на платформы, не перечисленные в данном документе.
- Качество записи аудиофайла будет определяться только качеством микрофона, на который будет подаваться входной сигнал.
- Скорость обработки сигнала будет зависеть только от производительности машины и степени нагрузки ЦП.

3 Обоснование выбора инструментальных средств

Для разработки приложения была выбрана технологи Microsoft .NET Framework 4.6, обладающий огромным функционалом для графического представления данных и построения окон.

Для создания GUI (графического интерфейса пользователя) был выбран интерфейс программирования приложений Windows Forms, являющийся частью Microsoft .NET Framework. Данный интерфейс упрощает доступ к элементам интерфейса Microsoft Windows за счет создания обёртки для существующего Win32 API в управляемом коде.

В качестве среды разработки была выбрана Microsoft Visual Studio 2015Express и язык C#, который является языком разработки приложений для платформы Microsoft .NET Framework.

Для написания кода была выбрана парадигма «ООП» (объектно-ориентированное программирование), так как C# - это объектно-ориентированный язык и каждое окно формы является экземпляром класса.

Что бы облегчить работу со звуковыми файлами, в частности: чтение, запись, воспроизведение, разложение сигнала по частотам, в проект была подключена библиотека NAudio.

NAudio – .NET библиотека с открытым исходным кодом, содержащая множество полезных классов, предназначенных для ускорения разработки .NET приложений, связанных с цифровой обработкой звуковых сигналов.

Для ведения истории разработки и контроля версий была выбрана распределённая система управления версиямиGit. Программа является свободной и выпущена под лицензией GNU GPL версии 2.

4 Разработка приложения по анализу речевых сигналов для IP-телефонии

4.1 Входные и выходные данные приложения

Входные данные хранятся в звуковых файлах с форматом .mp3 или .wav. Также есть возможность получить входной сигнал с микрофона.

Выходными данными являются:

1. Звуковой сигнал, переданный на звуковую карту.
2. Звуковая волна в виде амплитудно временного представления
3. График частот в момент времени
4. Изображение, показывающее зависимость спектральной плотности мощности сигнала от времени.
5. Сжатый или закодированный wav файл.
6. Wav файл, с сигналом полученным с микрофона

4.2 Проектирование структуры приложения

Логика приложения (классы и соответствующие файлы) приведена в таблице 4.1.

Таблица 4.1 – Классы и файлы

AudioFile.cs	Абстрактный класс, отвечающий за хранение общих свойств, для mp3 и wav файлов.
MP3File.cs	Класс, в котором хранится информация для аудиофайла типа mp3
WaveFile.cs	Класс для хранения wav файлов
Position.cs	Связывает текущее время воспроизведения с текстовым полем в интерфейсе.

TimePeriod.cs	Класс, отвечающий за отображение времени выделенного фрагмента и фрагмента отображения на уровнеграмме в интерфейсе.

Для упрощения работы с перечисляемыми типами были созданы файлы перечислений, приведенные в таблице 4.2.

Таблица 4.2 – Перечисляемые типы

AudioFormats.cs	Перечисление доступных форматов файлов, с которыми работает приложение.
Codecs.cs	Алгоритмы сжатия для кодирования в G.711.
Directions.cs	Направления перемотки аудиозаписи.

Так как в интерфейсе Windows Forms все окна и графические элементы формы являются отдельными классами, в таблице 4.3 представлены файлы окон и пользовательских элементов управления, созданных для визуализации звуковых сигналов.

Таблица 4.3 – Классы, отвечающие за графический интерфейс

MainForm.cs	Основное окно программы, являющееся контейнером для всех элементов управления.
SEWaveViewer.cs	Пользовательский элемент управления, отвечающий за визуализацию звука в амплитудно временном представлении.
SpectrumViewer.cs	Элемент управления, отвечает за отображение спектра звука в реальном времени.
SpectrogramViewer.cs	Этот элемент отображает спектрограмму выделенного фрагмента звуковой дорожки.

На рисунке 4.1 представлена общая схема связности классов и графических элементов управления.

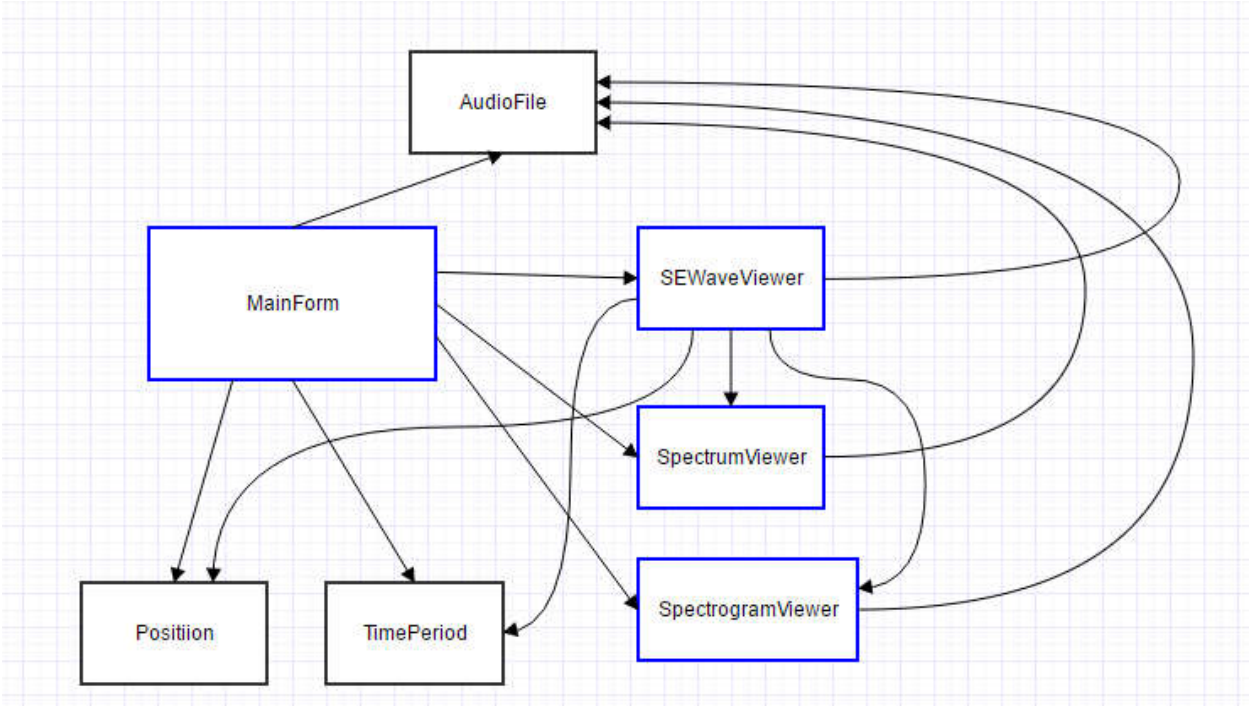

Рисунок4.1 - Схема связности классов в программе

4.3 Описание объектов и их взаимодействия

Подсистема визуализации данных содержит всего три класса: «SEWaveViewer», «SpectrumViewer»и «SpectrogramViewer». Данные классы предназначены для отображения звука в различных представлениях. С этими классами работает класс «MainForm»,который является главным окном приложения.

Классы «Position» и «TimePeriod» являются лишь обертками вокруг элементов интерфейса и отвечают за своевременное изменение информации на форме.

Классы «AudioFile», «MP3File»и «WaveFile»отвечают за хранение информации о звуковом файле.

На рисунке 4.2 приведена UML-диаграмма классов всего приложения.

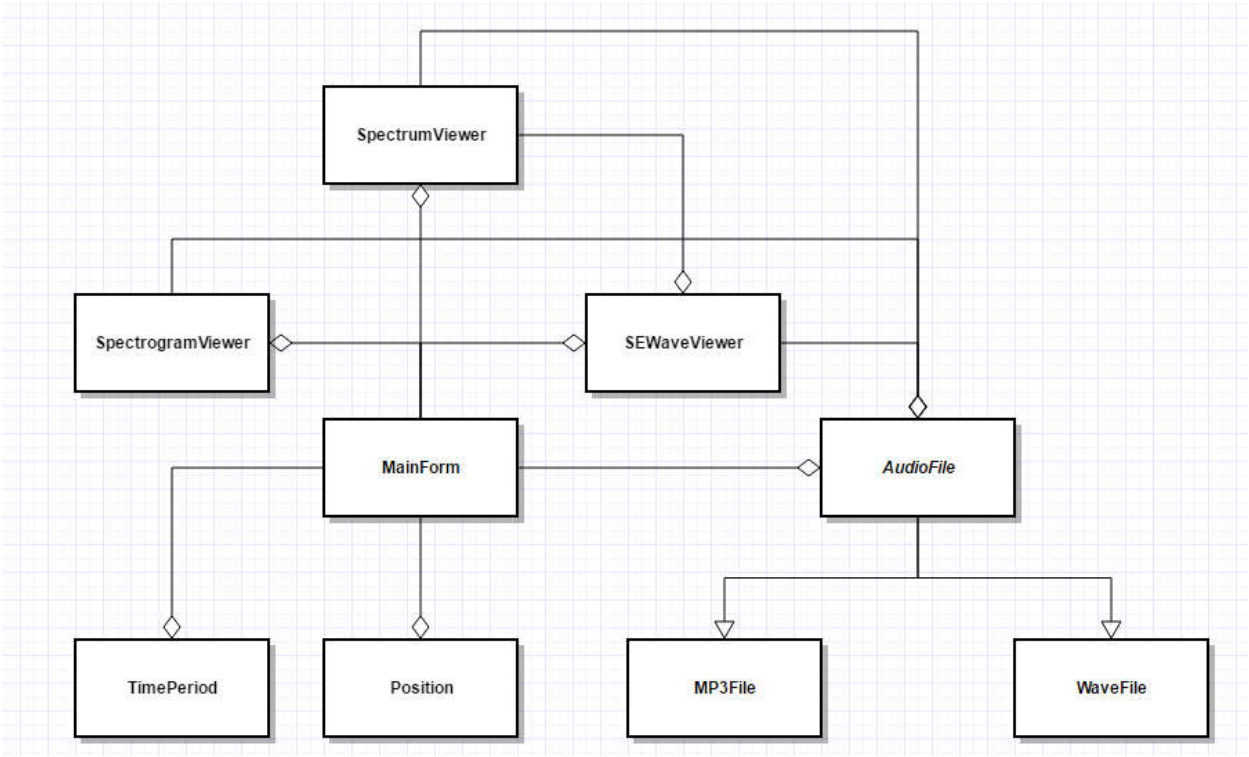

Рисунок 4.2–UML-диаграмма классов приложения

На рисунке 4.3 представлена общая структура проекта.

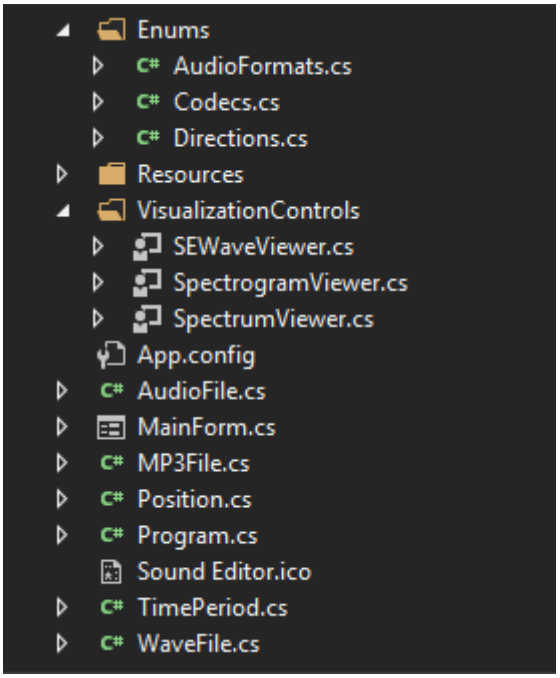

Рисунок 4.3–Структура проекта

Ниже в таблицах будет рассмотрен каждый класс более детально.

Таблица 4.4 – Методы, используемые в классе AudioFile

Название метода	Реализация
protectedabstractvoidreadBytes()	Чтение байт из звукового файла. Метод был определен как абстрактный так как, реализация для mp3 и wav файлов будет отличаться.
protected abstract void readShorts()	Чтение массива типа short из звукового файла. Реализация определена в классах «MP3File»и «WaveFile».
publicstaticvoidgetNameAndFormatFromPath(stringpath, outstringname, outstringformat)	Получение имени и формата файла из строки расположения объекта.
protectedvoidcallRead()	Вызов полиморфных методов«readShorts()»и «readBytes()».
privatevoidgetFloats()	Получения массива типа float из звукового файла.
privatevoidgetAvg()	Получение нормирующего коэффициента в массиве float.

	Нормировать значение необходимо для корректного отображения графиков спектра.

Таблица 4.5 – Методы, используемые в классе MP3File

protectedoverridevoidreadBytes()	Определена конкретная реализация получения массива байт из mp3 файла.
protectedoverridevoidreadShorts()	Определена реализация получения массива типа short из mp3 файла.

Таблица 4.6 – Методы, используемые в классе WaveFile

protectedoverridevoidreadBytes()	Определена конкретная реализация получения массива байт из wav файла.
protectedoverridevoidreadShorts()	Определена конкретная реализация получения массива типа short из wav файла.

Таблица 4.7 – Методы, используемые в классе Position

publicTimeSpanCurrentTime	Отображение текущего времени воспроизведения аудио в текстовом поле.
publicstaticstringgetTimeString(TimeSpantime)	Приведение объекта типа «TimeSpan» к строке определенного формата.

Таблица 4.8 – Методы, используемые в классе TimePeriod

publicTimeSpanStartTime	Отображение в интерфейсе времени начала фрагмента звуковой дорожки.
publicTimeSpanEndTime	Отображение в интерфейсе времени окончания фрагмента звуковой дорожки.
publicTimeSpanDuration	Отображение в интерфейсе длительности фрагмента звуковой дорожки.

Таблица 4.9 – Методы, используемые в классе SEWaveViewer

publicvoidFitToScreen()	Нормировка отображение уровнеграммы относительно размера окна.
publicvoidZoom(intleftSample, intrightSample)	Более детальная прорисовка фрагмента уровнеграммы.
privatevoidDrawVerticalLine(int x)	Отрисовка вертикальной линии на элементе управления.
protectedoverridevoidOnMouseDown(MouseEventArgs e)	Определение начального времени фрагмента и позиции для перемещения указателя.
protectedoverridevoidOnMouseMove(MouseEventArgs e)	Определение конечного времени выделенного фрагмента.
protectedoverridevoidOnMouseUp(MouseEventArgs e)	Определение границ и длительности выделенного фрагмента,

	перерисовка уровнеграммы.
protectedoverridevoidOnPaint(PaintEventArgs e)	Отображение амплитудно временного представления звуковой волны.
protectedoverridevoidDispose(booldisposing)	Освобождение памяти, удаление графических компонентов.

Таблица 4.10 – Методы, используемые в классе SpectrumViewer

publicstaticdouble[] getSpectrum(AudioFile audio, longstartPos)	Получение массива уровней частот (спектра) в момент времени.
privatevoidlogarithmSpectrum(double[] spectrum)	Перевод спектра из амплитудного представления в децибельный.
protectedoverridevoidOnClick(EventArgs e)	Переключение типа представления спектра.
protectedoverridevoidOnPaint(PaintEventArgs e)	Прорисовка шкалы для графика и отображение спектра.
protectedoverridevoidDispose(booldisposing)	Освобождение памяти, удаление графических компонентов.

Таблица 4.11 – Методы, используемые в классе SpectrogramViewer

privatevoidfindMax()	Поиск максимального значения спектра.
privatevoiddrawBitMap()	Прорисовка спектрограммы на объекте «Bitmap»

protectedoverridevoidOnPaint(PaintEventArgs e)	Отображение на графическом элементе сохраненной спектрограммы в памяти.
protectedoverridevoidDispose(booldisposing)	Освобождение памяти, удаление графических компонентов.

Таблица 4.12 – Методы, используемые в классе MainForm

privatevoidMainForm_Load(objectsender, EventArgs e)	Загрузка формы. Инициализация графических элементов и объекта воспроизведения аудио.
privatevoidinitAudioInfo()	Отображение информации о выбранном файле.
privatevoidinitAudio(AudioFile f)	Инициализация аудиофайла.
privatevoidaddFileToListView(AudioFile f)	Добавление аудиофайла в таблицу.
privatevoidaddFileToListView(stringpath)	Добавление пустого аудиофайла в таблицу.
privatevoidopenToolStripButton_Click(objectsender, EventArgs e)	Открытие и чтение выбранного аудиофайла.
privatevoidlistAudio_MouseDoubleClick(objectsender, MouseEventArgs e)	Выбор файла для визуализации из списка добавленных.
privatevoiddeleteToolStripButton_Click(objectsender, EventArgs e)	Удаление файла из списка и освобождение памяти.

privatevoid toolStripButton3_Click(objectsender, EventArgs e)	Приостановка воспроизведения аудио.
privatevoid toolStripButton1_Click(objectsender, EventArgs e)	Воспроизведение аудио.
privatevoid toolStripButton2_Click(objectsender, EventArgs e)	Остановка воспроизведения.
privatevoidOutput_PlaybackStopped(objectsender, StoppedEventArgs e)	Остановка всех таймеров.
privatevoid toolStripButton4_Click(objectsender, EventArgs e)	Переключение на предыдущий файл.
privatevoid toolStripButton7_Click(objectsender, EventArgs e)	Переключение на следующий файл.
privatevoidback()	Перемотка аудиозаписи назад.
privatevoid toolStripButton5_MouseDown(objectsender, MouseEventArgs e)	Запуск таймера перемотки назад.
privatevoid toolStripButton5_MouseUp(objectsender, MouseEventArgs e)	Остановка таймера перемотки назад.
privatevoidforward()	Перемотка аудиозаписи вперед.
privatevoid toolStripButton6_MouseDown(objectsender, MouseEventArgs e)	Запуск таймера перемотки вперед.
privatevoid toolStripButton6_MouseUp(objectsender, MouseEventArgs e)	Остановка таймера перемотки вперед.
privatevoidchangePositionTimer_Tick(objectsender, EventArgs e)	Вызов методов перемотки при событии тика таймера.

privatevoidDisposeWave()	Удаление объекта воспроизведения аудио и освобождение памяти.
privatevoid Form1_FormClosing(objectsender, FormClosingEventArgs e)	Очистка памяти при закрытии формы.
privatevoidoriginalPlayTimer_Tick(objectsender, EventArgs e)	Изменение информации о текущем времени воспроизведения.
privatevoidtrackBarOriginal_Scroll(objectsender, EventArgs e)	Изменение уровня громкости воспроизведения аудио.
privatevoidspectrumTimer_Tick(objectsender, EventArgs e)	Обновление графического элемента отображения спектра во время воспроизведения.
privatevoidnewToolStripButton_Click(objectsender, EventArgs e)	Создание объекта аудиофайла для последующей записи сигнала.
privatevoidrefreshDeviceListButton_Click(objectsender, EventArgs e)	Обновление списка доступных записывающих устройств.
privatevoidstartRecordButton_Click(objectsender, EventArgs e)	Создание буфера для записи входного сигнала и запуск таймера.
privatevoidSourceStream_DataAvailable(objectsender, WaveInEventArgs e)	Запись звука с микрофона.

privatevoidstopRecordButton_Click(objectsender, EventArgs e)	Остановка записи.
privatevoidopenSavedFile()	Открытие сохраненного файла.
privatevoidrecordingTimer_Tick(objectsender, EventArgs e)	Обновление текущего времени записи.
privatevoidsaveWavButton_Click(objectsender, EventArgs e)	Сохранение сжатого аудио в новый файл.
privatevoid button2_Click(objectsender, EventArgs e)	Кодирование аудио в кодек G.711.
privatevoid button1_Click(objectsender, EventArgs e)	Выбор файла для декодирования.
privatevoid decodeG711(stringfilename, Codecscodec)	Декодирование аудио из G.711.

Реализации основных методов описаны в приложении Б.

5 Тестирование программного продукта

5.1 Аппаратные и программные средства создания и эксплуатации приложения

Аппаратные требования для работы приложения:

- Процессор с тактовой частотой 1.0 ГГц.
- Оперативная память 512 Мб и более.
- Видеокарта с объёмом памяти 64 Мб и выше.
- Монитор 800x600 или с более высоким разрешением.

Программные требования к приложению.

Установленные на компьютере:

- Операционная система —Windows XP или Windows 7/8/8.1;
- Microsoft .NET Framework 4.6

5.2 Руководство пользователя

Для работы приложения необходимо поместить в одну директорию исполняемый файл приложения и библиотеку NAudio.dll.

На рисунке 5.1 показана файловая система приложения.

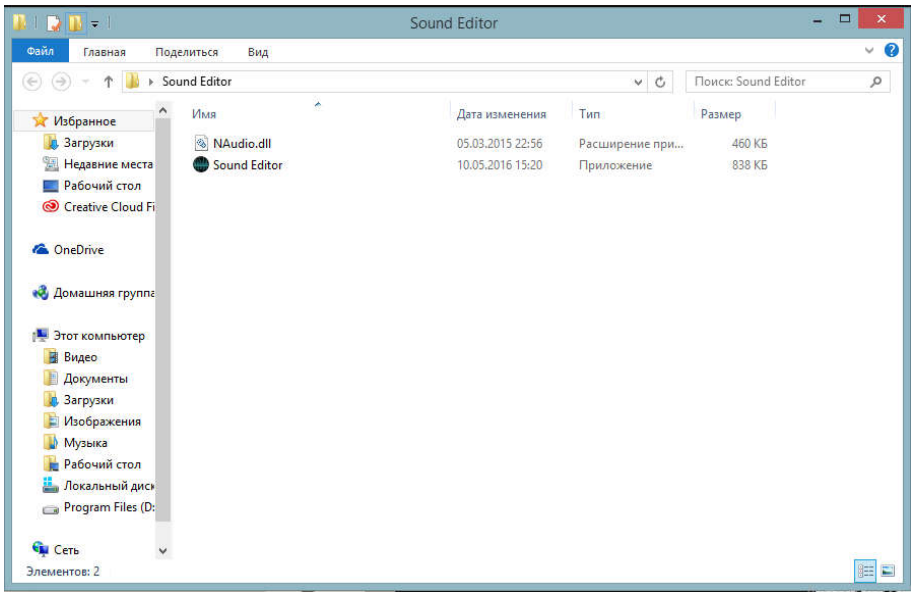

Рисунок 5.1–Файловая система приложения

5.3 Описание контрольных примеров

Пользователь запускает приложение, и система открывает на весь размер экрана главное окно приложения (рисунок А.1).

Пользователь нажимает на кнопку открытия аудиофайла (рисунок А.2) и выбирает в проводнике нужный ему файл. После выбора файла система добавляет в список добавленный файл и отрисовывает уровнеграмму звука (рисунок А.3).

Для более детального просмотра уровнеграммы пользователь на графическом элементе интерфейса выделяет нужный ему фрагмент и система перерисовывает график, так как указатель был перемещен, то система отображает спектр в начальной точке выделенного фрагмента (рисунок А.4).

Для отображения спектра в другом представлении пользователь кликает по графическому элементу отображения спектра и система переключает режим (рисунки А.5, А.6).

Если пользователь хочет увидеть спектрограмму выделенного фрагмента, то он переключается на вкладку спектрограммы и система отображает график (рисунок А.7).

ПРИЛОЖЕНИЕ А

Экранные формы

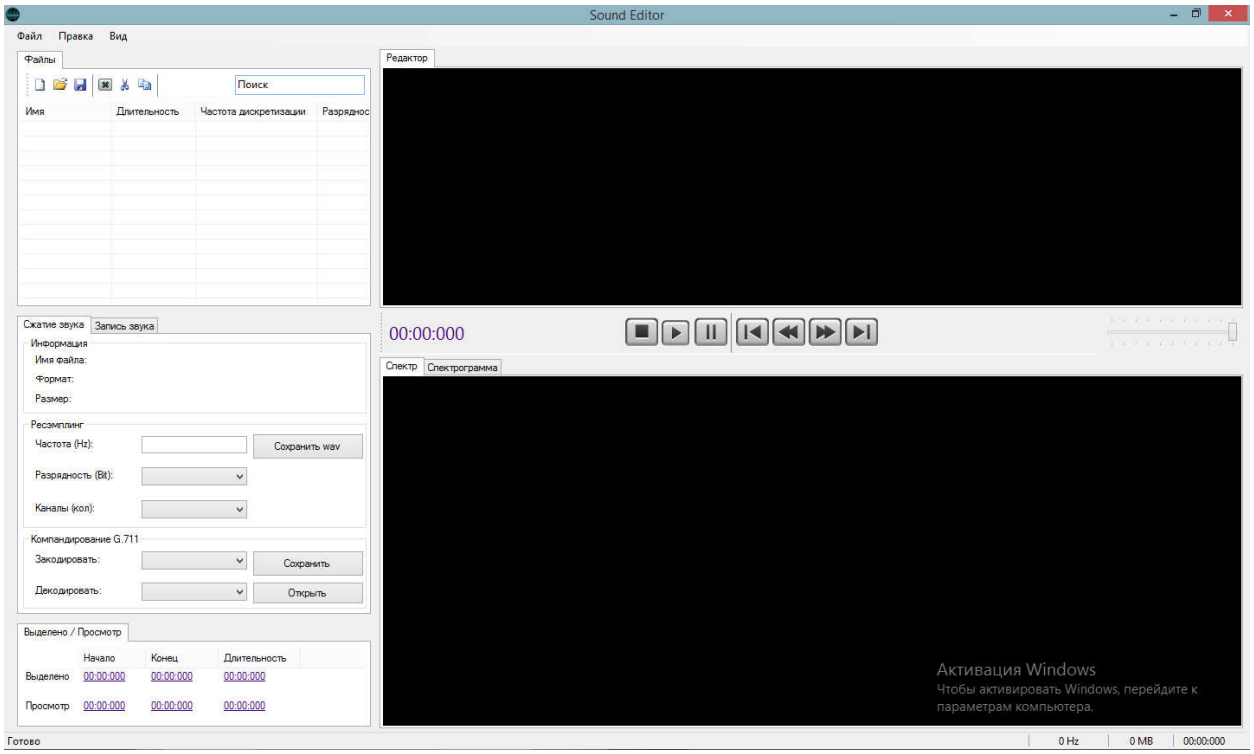

Рисунок А.1–Главное окно приложения

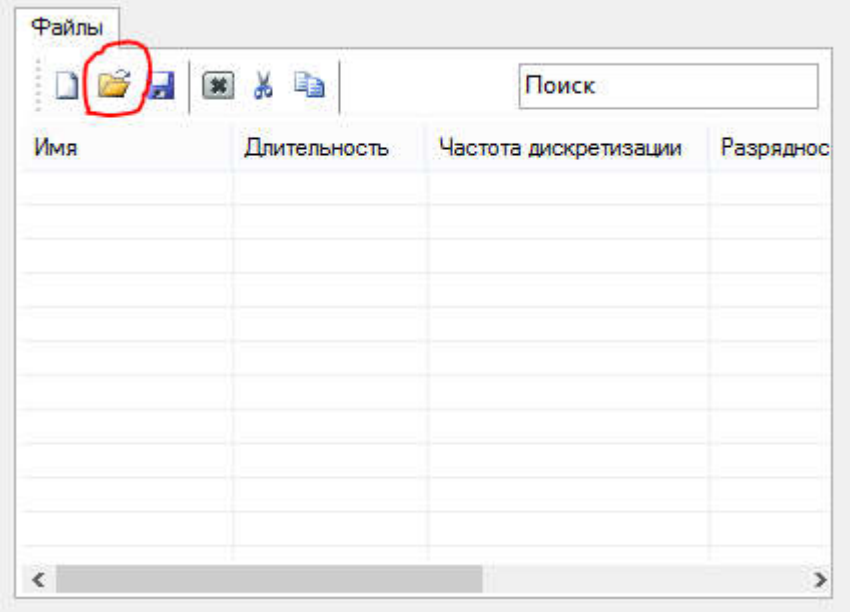

Рисунок А.2–Кнопка открытие аудиофайла

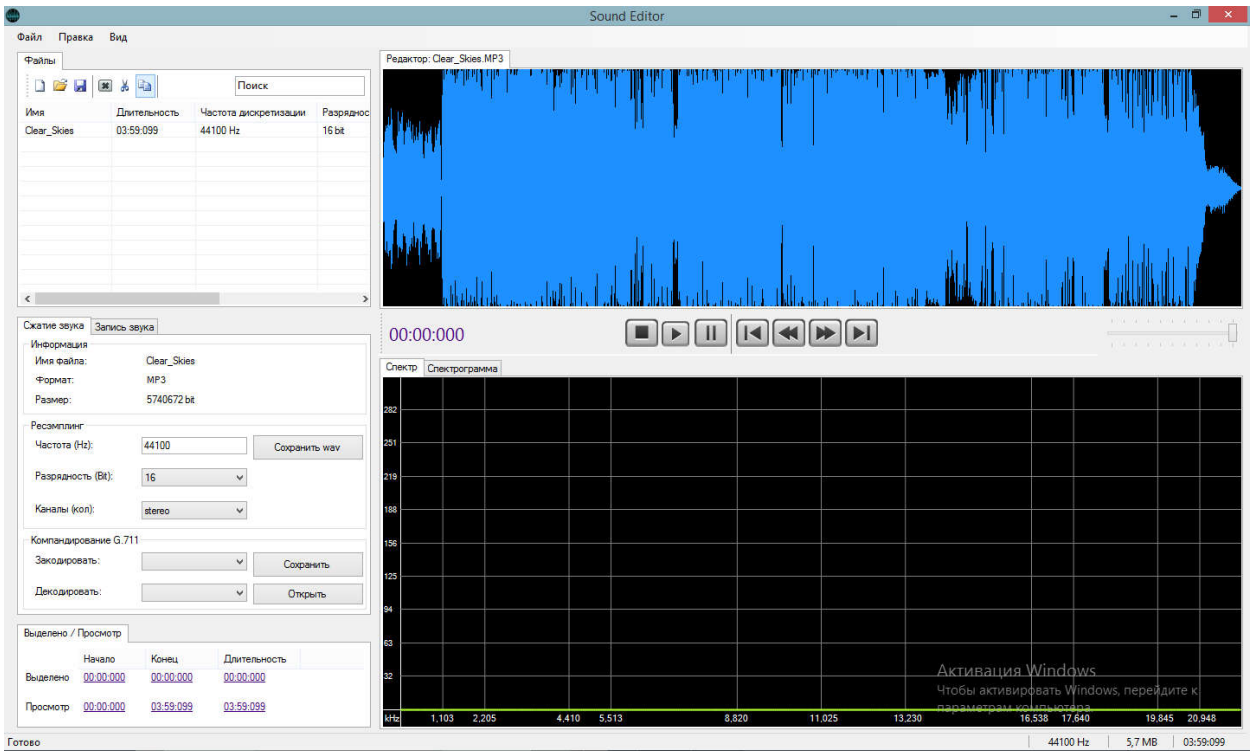

Рисунок А.3–Визуализация выбранного файла

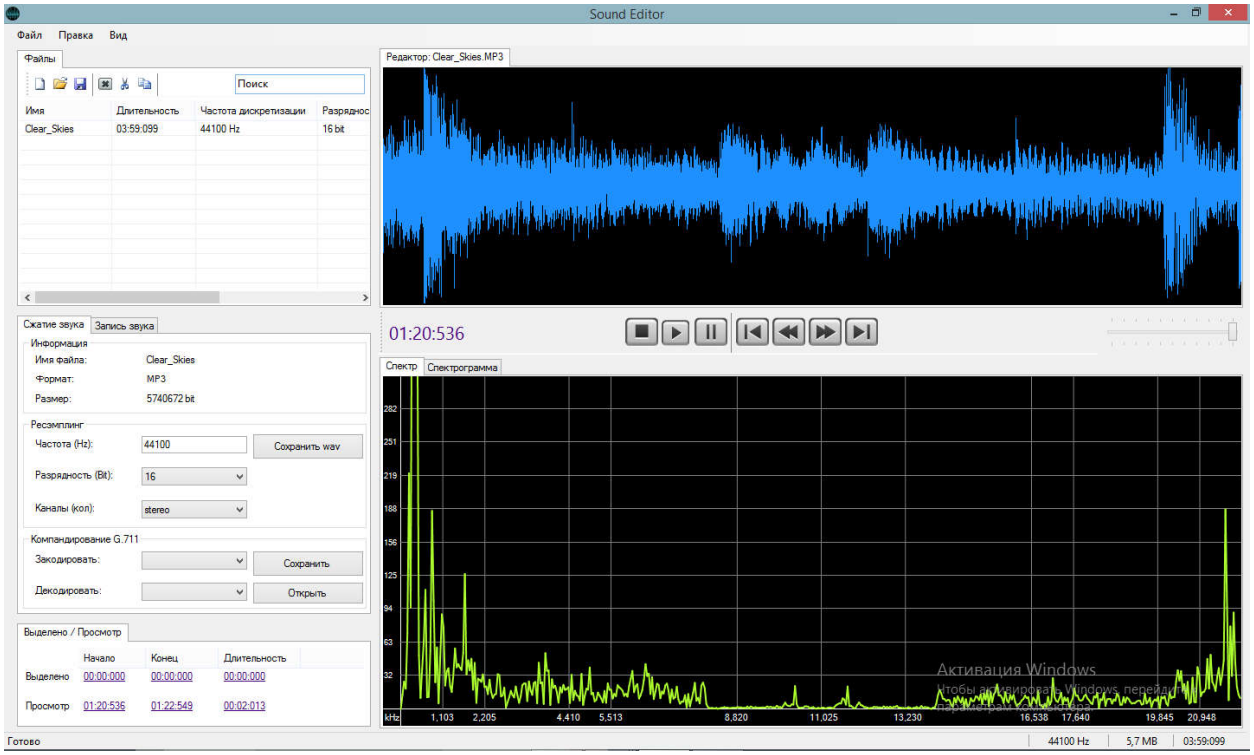

Рисунок А.4–Выделение фрагмента звуковой дорожки

Рисунок А.5–Спектр в децибельном представлении

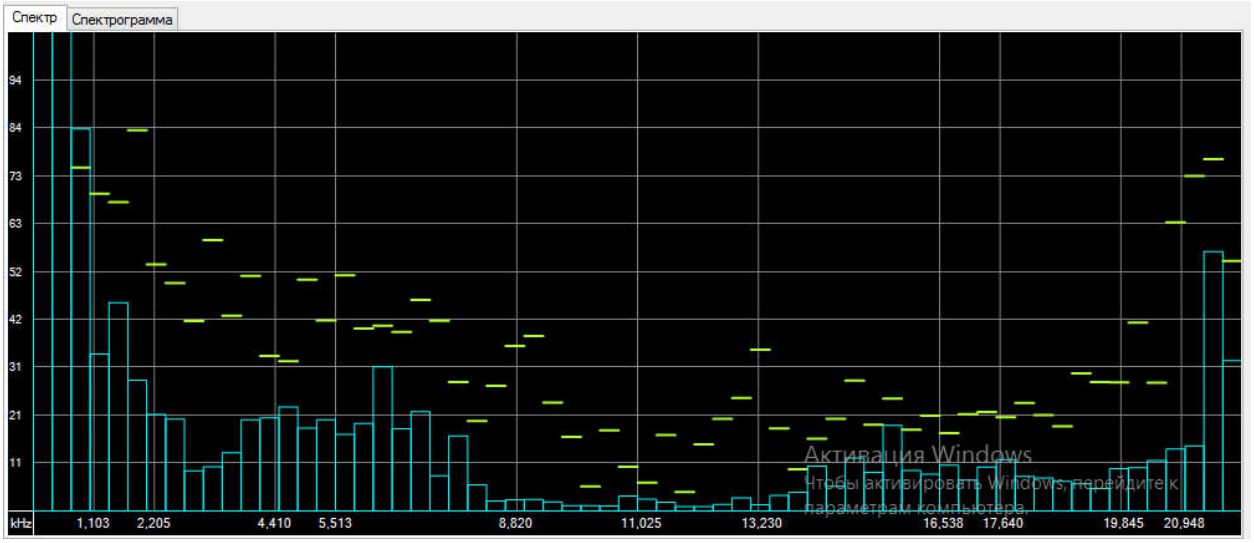

Рисунок А.6–Спектр в столбчатом представлении

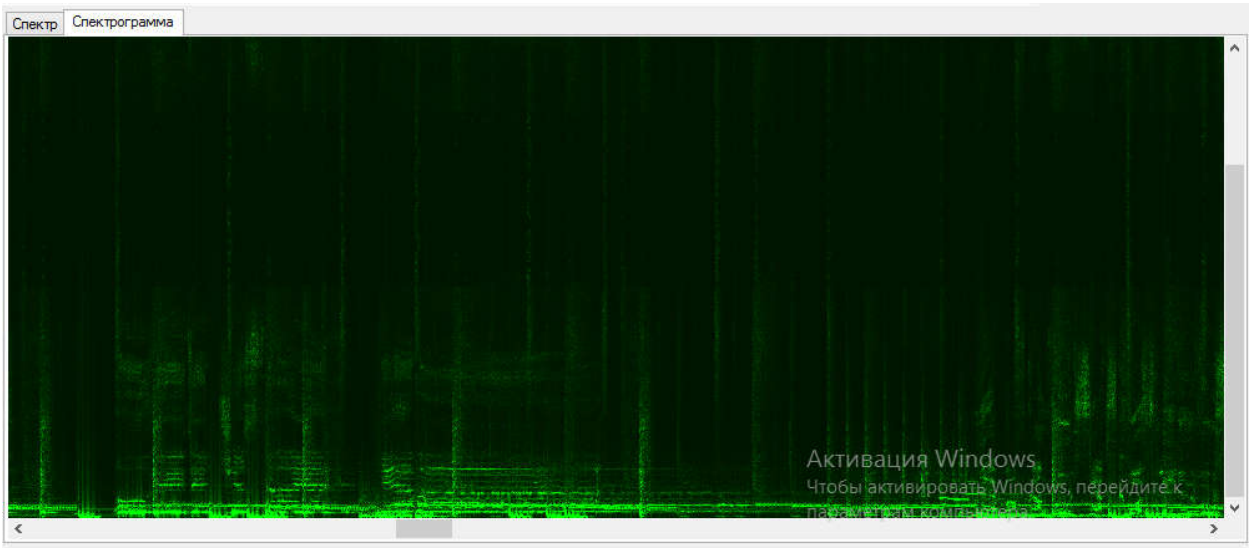

Рисунок А.7–Спектрограмма выделенного фрагмента

ПРИЛОЖЕНИЕ Б

Фрагменты листинга

Листинг алгоритма прорисовки спектра:

```
protectedoverridevoidOnPaint(PaintEventArgs e) {
if (this.Audio != null) {
longposition = 0;
if (audio.Format == Enums.AudioFormats.MP3) {
                    MP3File file = audioas MP3File;
position = file.Reader.Position / 2;
                } elseif (audio.Format == Enums.AudioFormats.WAV)
{
                    WaveFile file = audioas WaveFile;
position = file.Reader.Position / 2;
                }

double[] spectrum = SpectrumViewer.getSpectrum(this.Audio,
position);
if (this.state == ViewState.LOGARITHM) {
this.logarithmSpectrum(spectrum);
                }

e.Graphics.SmoothingMode =
System.Drawing.Drawing2D.SmoothingMode.AntiAlias;
PenlinePen = newPen(this.PenColor, this.PenWidth);
linePen.StartCap = System.Drawing.Drawing2D.LineCap.Round;
linePen.EndCap = System.Drawing.Drawing2D.LineCap.Round;

floatstep = (float)(this.Width - 20) / spectrum.Length;

                // Отрисовка шкалы по оси X
intyLinePos = (this.state != ViewState.LOGARITHM) ?this.Height -
20 : 20;
```

```
intyStringPos = (this.state != ViewState.LOGARITHM) ?yLinePos + 3
: 3;
e.Graphics.DrawLine(Pens.White, 0, yLinePos, this.Width,
yLinePos);
e.Graphics.DrawString("kHz", newFont(FontFamily.GenericSansSerif,
7.5f), Brushes.White, 0, yStringPos);
int[] freqPointsPercents = { 5, 10, 20, 25, 40, 50, 60, 75, 80,
90, 95 };   // Позиции значений частот (в %)
floatfreqPoint;
for (int i = 0; i <freqPointsPercents.Length; i++) {
freqPoint = 20 + (freqPointsPercents[i] * (this.Width - 20) /
100f);
if (this.state == ViewState.COLUMNAR || this.state ==
ViewState.DEFAULT) {
e.Graphics.DrawLine(newPen(Color.Gray, 1f), freqPoint, 0,
freqPoint, this.Height - 20);
                    } else {
e.Graphics.DrawLine(newPen(Color.Gray, 1f), freqPoint, 20,
freqPoint, this.Height);
                    }
                    // Получение частоты в текущей точке и ее
отображение
doublesample = (spectrum.Length * freqPointsPercents[i]) / 100.0;
stringcurrentFreq = ((sample * freq) * Math.Pow(10, -
3)).ToString("0.000");
e.Graphics.DrawString(currentFreq,
newFont(FontFamily.GenericSansSerif, 7.5f), Brushes.White,
freqPoint - 15, yStringPos);
                }

                // Отрисовка шкалы по оси Y
e.Graphics.DrawLine(Pens.White, 20, 0, 20, this.Height);
if (this.state == ViewState.LOGARITHM) {
e.Graphics.DrawString("dB", newFont(FontFamily.GenericSansSerif,
7.5f), Brushes.White, 2, this.Height - 15);
```

```
                }
int[] gradePointsPercents = { 10, 20, 30, 40, 50, 60, 70, 80, 90
}; // Позиции значений уровня спектра (в %)
intgradePoint;
doublecurrentGrade;
for (int i = 0; i <gradePointsPercents.Length; i++) {
if (i == gradePointsPercents.Length - 1 &&this.state ==
ViewState.LOGARITHM) break;    // Для dB не выводить последнее
значение
gradePoint = gradePointsPercents[i] * (this.Height - 20) / 100;
gradePoint = (this.state == ViewState.LOGARITHM) ?gradePoint + 20
: gradePoint;
e.Graphics.DrawLine(newPen(Color.Gray, 1f), 20, gradePoint,
this.Width, gradePoint);
if (this.state != ViewState.LOGARITHM) {
                        // Нормировка значения уровня спектра
currentGrade = (this.Height - 20 - gradePoint) * this.Audio.Avg /
(this.Height - 20);
if (this.state == ViewState.COLUMNAR) {
currentGrade /= 3;
                        }
currentGrade *= 10000;
currentGrade = Math.Round(currentGrade);
                    } else {
currentGrade = Math.Round(gradePoint / -1.5);
                    }
e.Graphics.DrawString(currentGrade.ToString(),
newFont(FontFamily.GenericSansSerif, 7f), Brushes.White, -1,
gradePoint - 6);
                }

                // Отрисовка спектра
floatkoef = (this.state != ViewState.LOGARITHM) ? (this.Height -
20) / this.Audio.Avg : 1.5f;
```

```
float x = e.ClipRectangle.X + 20;
float y = (float)(this.Height - 20);
                y = (this.state == ViewState.LOGARITHM) ? 20 : y;
if (this.state != ViewState.COLUMNAR) { // Для грфаического
представления
float x1, y1;
for (int i = 1; i <spectrum.Length; i++) {
                        x1 = x + step;
                        y1 = (this.state != ViewState.LOGARITHM) ?
(this.Height - 20) - (float)(spectrum[i] * koef) : 20 -
(float)(spectrum[i] * koef);
if (float.IsInfinity(y1)) continue;
e.Graphics.DrawLine(linePen, x, y, x1, y1);
                        x = x1; y = y1;
                    }
                } else{    // Для столбчатого
floatcolumnWidth = (float)(this.Width - 20) / this.columnCount;
floatcolumnHeight = 0;
floatcolumnHeightCoord;
floatoldLineYCoord;
for (int i = 0; i <this.columnCount; i++, columnHeight = 0) {
                        // Находим среднее значение уровня для
текущего столбца
for (int j = i * (spectrum.Length / this.columnCount), count = 0;
count< (spectrum.Length / this.columnCount); j++, count++) {
if (i == 0 && j == 0) continue;
columnHeight += (float)spectrum[j];
                        }
columnHeight /= (i == 0) ? (spectrum.Length / this.columnCount) -
1 : spectrum.Length / this.columnCount;
this.oldValues[i] -= 0.00005f; // Плавное падение полосы-указателя
предыдущего уровня
oldLineYCoord = (this.Height - 20) - this.oldValues[i] * koef * 3;
if (columnHeight>this.oldValues[i] ) {
this.oldValues[i] = columnHeight;   // Обновление указателя уровня
```

```
                        }
columnHeightCoord = (this.Height - 20) - columnHeight * koef * 3;
e.Graphics.DrawLine(linePen, x, oldLineYCoord, x + columnWidth,
oldLineYCoord);
e.Graphics.DrawRectangle(Pens.Aqua, x, columnHeightCoord,
columnWidth, y - columnHeightCoord);
                        x += columnWidth;
                    }
                }
            }
base.OnPaint(e);
        }
```

Листинг алгоритма отображение спектрограммы:

```
private void findMax() {
long position = 0;
for (int i = 0; i <this.Audio.FloatSamples.Length / 1024; i++) {
double[] spectrum = SpectrumViewer.getSpectrum(this.Audio,
position);
position += 1024;
for (int j = 0; j <spectrum.Length; j++) {
if (spectrum[j] >this.max) {
this.max = spectrum[j];
                    }
                }
            }
        }

private void drawBitMap() {
if (this.max == 0) this.findMax();
if (this.count == 0) throw new Exception();
this.bitMap = new Bitmap(this.count, 512);
long position = 0;
position = this.StartPosition;
doublekoef; int x = 0;
```

```
for (int i = 0; i <this.count; i++, x++) {
double[] spectrum = SpectrumViewer.getSpectrum(this.Audio,
position);
position += 1024;
koef = 135 / max * 16;
int color;
for (int j = 0; j <spectrum.Length; j++) {
color = 20 + (int)(spectrum[j] * koef);
if (color > 255) color = 255;
this.bitMap.SetPixel(x, 512 - j - 1, Color.FromArgb(0, color, 0));
                }
            }
        }

protected override void OnPaint(PaintEventArgs e) {
if (this.Audio != null) {
if (this.bitMap == null) {
try {
this.drawBitMap();
                    } catch (Exception) {
base.OnPaint(e);
return;
                    }
                }
e.Graphics.DrawImage(this.bitMap, new PointF(0, 0));
if (this.count<this.Width) {
e.Graphics.DrawLine(new Pen(Color.White, 1), this.count + 1, 0,
this.count + 1, this.Height);
                }
            }
base.OnPaint(e);
        }
```

Методы чтения массива байт из аудиофайла:

```
protected override void readBytes() {
```

```
this.Reader.Read(this.Samples, 0, (int)this.Reader.Length);
this.Reader.Position = 0;
        }

protected override void readShorts() {
this.ShortSamples = new short[this.Reader.Length / 2];
for (int i = 0; i <this.Reader.Length - 1; i++) {
this.ShortSamples[i / 2] = (short)((this.Samples[i] << 8) |
this.Samples[i + 1]);
            }
        }
```