Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра программного обеспечения информационных технологий

Дисциплина: Системное программирование (СП)

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту

на тему

**ПРОГРАММНОЕ СРЕДСТВО «MUSIC SEQUENCER» ДЛЯ НАПИСАНИЯ МУЗЫКИ**

БГУИР КП 1-40 01 01 115 ПЗ

Студент Кривицкий Ф. Ю.

Руководитель Деменковец Д. В.

Минск 2024

СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 5](#_Toc184646515)

[1 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ 6](#_Toc184646516)

[1.1 Анализ литературных источников 6](#_Toc184646517)

[1.2 Обзор аналогов 7](#_Toc184646518)

[1.3 Постановка задачи 9](#_Toc184646519)

[2 АНАЛИЗ ТРЕБОВАНИЙ К ПРОГРАМНОМУ СРЕДСТВУ И РАЗРАБОТКА ФУНКЦИОНАЛЬНЫХ ТРЕБОВАНИЙ 11](#_Toc184646520)

[2.1 Структура программы 11](#_Toc184646521)

[2.2 Проектирование интерфейса программного средства 11](#_Toc184646522)

[2.3 Проектирование функционала программного средства 12](#_Toc184646523)

[3 ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА 14](#_Toc184646524)

[3.1 Обобщенный алгоритм работы ПС 14](#_Toc184646525)

[3.2 Разработка алгоритмов 14](#_Toc184646526)

[3.3 Работа с WinApi 18](#_Toc184646527)

[4 СОЗДАНИЕ ПРОГРАММНОГО СРЕДСТВА 19](#_Toc184646528)

[4.1 Проектирование основных методов программного средства 19](#_Toc184646529)

[5 ТЕСТИРОВАНИЕ, ПРОВЕРКА РАБОТОСПОСОБНОСТИ И АНАЛИЗ ПОЛУЧЕННЫХ РЕЗУЛЬТАТОВ 24](#_Toc184646530)

[5.1 Тестирование и проверка работоспособности программного средства 24](#_Toc184646531)

[5.2 Анализ полученных результатов 25](#_Toc184646532)

[6 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ 28](#_Toc184646533)

[ЗАКЛЮЧЕНИЕ 29](#_Toc184646534)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 30](#_Toc184646535)

[ПРИЛОЖЕНИЕ А (обязательное) 31](#_Toc184646536)

[ПРИЛОЖЕНИЕ Б (обязательное) 32](#_Toc184646537)

# ВВЕДЕНИЕ

Разработка программных средств для создания и обработки музыки является востребованной задачей, поскольку такие инструменты находят широкое применение в творческой деятельности, обучении и профессиональной аудиопроизводстве. В этом контексте важно создать эффективное приложение, которое позволит пользователям легко и качественно работать с музыкальными композициями.

Данная курсовая работа посвящена разработке программного средства "Music Sequencer", предназначенного для создания и редактирования музыкальных треков. Этот инструмент решает актуальные задачи, такие как написание музыки для образовательных целей, разработки звукового оформления, создания аудиотреков для медиа и других сфер, где требуется музыкальное сопровождение.

Цель настоящей курсовой работы — разработка программного средства "Music Sequencer", которое предоставляет пользователям удобный и интуитивно понятный интерфейс для создания, редактирования и сохранения музыкальных композиций. Программа будет основываться на современных технологиях обработки звука и обеспечивать высокую точность и гибкость в работе с музыкальными данными.

Основные функциональные возможности включают: создание последовательностей нот и звуковых дорожек, использование различных инструментов и эффектов для изменения звучания, экспорт готовых композиций в аудиоформат WAV, написание партий ударных инструментов, партии фортепиано для последующей секвенции.

"Music Sequencer" ориентирован на широкий круг пользователей, в частности начинающих музыкантов, предоставляя эффективное решение для написания музыки в самых различных областях, включая творчество, образование и медиаиндустрию.

# АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

## Анализ литературных источников

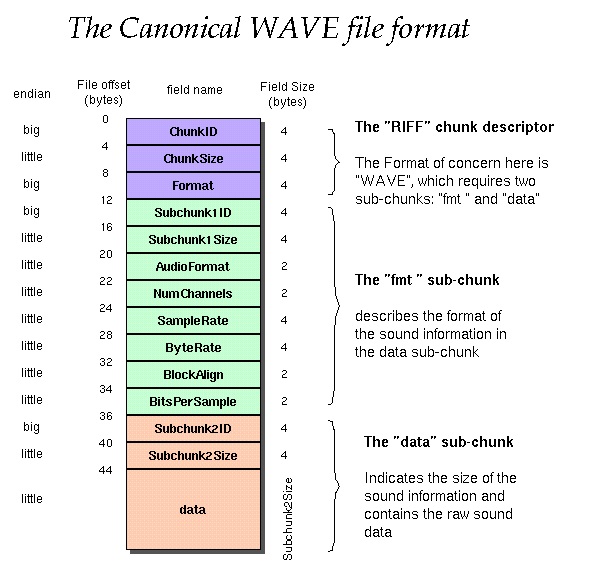
Waveform Audio File Format (WAVE, WAV, от англ. waveform — «в форме волны») — формат файла-контейнера для хранения записи оцифрованного аудиопотока, подвид RIFF. Этот контейнер, как правило, используется для хранения несжатого звука в импульсно-кодовой модуляции.

RIFF (англ. Resource Interchange File Format) — формат файлов-контейнеров для хранения потоковых мультимедиа-данных (видео, аудио, текста). Наиболее известными контейнерными форматами, использующими RIFF в качестве основы, являются AVI (видео), WAV (аудио), RMI (MIDI-треки).

Формат RIFF использует порядок байтов little-endian (младший байт идёт первым). Для машин с форматом данных big-endian предлагался формат RIFX, однако из-за существенно меньшей в бытовом секторе популярности компьютеров с таким форматом данных RIFX не прижился, и в настоящее время формат RIFF воспроизводится и на машинах с big-endian порядком байтов.

В силу особенностей ряда популярных программ по созданию и воспроизведению файлов в Microsoft Windows ограничение усилилось до 2 ГБ (из-за использования в API RIFF signed int32). Для поддержки файлов больше 4 ГБ формат AVI был расширен до AVI-DV, выходящего за рамки контейнера RIFF. На базе RIFF предложено расширение RF64, позволяющее работать с 64-битными числами для размеров chunk’ов, разработанное European Broadcasting Union.

Формат RIFF разработан в 1991 году компаниями Microsoft и IBM на основе формата IFF, разработанного компанией Electronic Arts в 1985 году, для little-endian компьютеров.



## Обзор аналогов

Существует ряд программных средств, предназначенных для написания музыки, каждое из которых имеет свои особенности и преимущества. Проведем обзор нескольких популярных аналогов "Music Sequencer".

1.2.1 «FL Studio»

FL Studio — цифровая звуковая рабочая станция (DAW) и секвенсор для написания музыки. Музыка создаётся путём записи и сведения аудио- или MIDI-материала. FL Studio поставляется вместе с множеством разнообразных генераторов, основанных на собственной архитектуре программы. FL Studio также поддерживает VST- и DirectX-плагины от сторонних разработчиков. Многие плагины FL Studio также существуют в виде отдельных приложений. Интерфейс приложения представлен на рисунке 1.1.



Рисунок 1.1 – Интерфейс FL Studio

1.2.2 «Abelton Live»

Ableton Live — компьютерная программа от немецкой компании Ableton для музыкантов (секвенсор), используемая как для студийной работы (аранжировка, сведение), так и для живой игры (импровизация, диджеинг). В дополнение к инструментам может работать с семплами. Live пытается проанализировать ритм семпла, чтобы вычислить метр, количество тактов и темп. Таким образом Live позволяет «подтягивать» сэмплы, чтобы они соответствовали общему темпу. Интерфейс приложения представлен на рисунке 1.2.



Рисунок 1.2 – Интерфейс Abelton Live

1.2.3 «NanoStudio»

NanoStudio — программа для музыкантов, представляющая собой компактную студию звукозаписи, позволяющую создавать и записывать музыку на ПК. В состав программы включен виртуальные синтезаторы, множественные эффекты, позволяющие реализовывать пользовательские идеи на практике.

В программе NanoStudio реализовано большое количество эффектов и фильтров, есть поддержка звуковых карт и MIDI-клавиатур. Интерфейс NanoStudio представлен на рисунке 1.3.



Рисунок 1.3 – Интерфейс NanoStudio

Cледует отметить, что программное средство «Music Sequencer» занимает свою нишу среди аналогичных решений, предлагая пользователям удобный и интуитивно понятный инструмент для создания базовых музыкальных партий. Основное внимание уделено предоставлению функционала для написания партий пианино и ударных инструментов, а также настройки темпа и выбора звуков. При этом включение сложных функций, характерных для профессиональных музыкальных редакторов, не рассматривалось как целесообразное на данном этапе разработки.

## 1.3 Постановка задачи

В рамках данной курсовой работы планируется разработать программное средство “Music Sequencer” для написания музыки. Средство будет реализовано на языке C++ с использованием WinAPI и библиотеки для воспроизведения звуков “FMOD”.

Основной целью данной работы является создание удобного и функционального инструмента для базового создания музыкальных партий, позволяющего пользователям быстро и просто записывать мелодии и ритмические рисунки.

Для реализации данной задачи необходимо выполнить следующие этапы:

– разработать интерфейс для взаимодействия с пользователем;

– реализовать функционал написания партий пианино и ударных инструментов с использованием пиано-ролла;

– обеспечить возможность задания темпа (BPM) композиции;

– реализовать возможность выбора звуков для ударных инструментов;

– обеспечить сохранение созданных партий в формате WAV.

Результатом работы должно стать готовое программное средство для создания музыкальных сэмплов, которое предоставит пользователю возможность писать партии пианино и ударных длиной в два такта, настраивать темп и звуковые параметры, а также экспортировать композиции в удобном формате для дальнейшего использования.

.

# АНАЛИЗ ТРЕБОВАНИЙ К ПРОГРАМНОМУ СРЕДСТВУ И РАЗРАБОТКА ФУНКЦИОНАЛЬНЫХ ТРЕБОВАНИЙ

## 2.1 Структура программы

При разработке программного средства будут использованы все стандарты разработки под платформу Windows.

## 2.2 Проектирование интерфейса программного средства

При разработке интерфейса музыкального секвенсора было принято решение использовать подходы, реализованные в программе FL Studio, чтобы обеспечить удобство и интуитивность взаимодействия с пользователем. Главное окно программы будет служить центральным элементом управления всеми функциями, включая создание музыкальных паттернов, настройку темпа, изменение звуков и сохранение результатов работы.

2.2.1 Главное окно программы

Главное окно должно содержать области для работы с различными инструментами. Для создания паттернов ударных инструментов предназначен специальный контроллер, который позволяет активировать или деактивировать звуки, нажимая на соответствующие кнопки. Также здесь предусмотрены элементы управления частотой заполнения паттерна. Для работы с паттернами пианино предусмотрен отдельный контроллер, где пользователь выбирает ноты, а при необходимости изменяет их длительность с помощью кнопок в нижней части интерфейса. Пример представлен на рисунке 2.1.

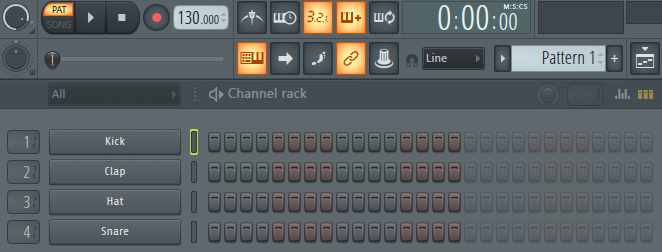




Рисунок 2.1 – Главное окно

2.2.2 Настройка паттернов ударных инструментов

Процесс создания паттернов ударных инструментов реализован через интерактивный контроллер. Пользователь выбирает звуки, активируя кнопки в соответствующей области интерфейса. Частота заполнения регулируется специальными кнопками, расположенными рядом с контроллером. Эти настройки позволяют пользователю гибко настраивать ритмические элементы композиции.

2.2.3 Настройка паттернов пианино

Для работы с пианино в программе должен быть предусмотрен контроллер, который предоставляет возможность выбора нот путем нажатия на соответствующие области. При необходимости длительность нот можно изменять, используя элементы управления, размещенные в нижней части интерфейса. Такая организация позволяет создавать мелодические линии.

## 2.3 Проектирование функционала программного средства

Программное средство «Music Sequencer» для написания музыки должно предоставлять пользователю такой минимальный функционал как:

* создание паттернов для различных инструментов (ударные, пианино);
* настройка темпа воспроизведения композиции;
* воспроизведение созданной композиции;
* изменение звуков инструментов;
* сохранение композиции в аудиофайле формата WAV;
* завершение работы программы.

2.3.1 Создание паттернов инструментов

В программе должны быть предусмотрены инструменты для создания музыкальных паттернов для различных инструментов, таких как ударные и пианино. Для этого должны быть предусмотрены кнопки и контроллеры, позволяющие выбирать и настраивать ноты и ритмы. Паттерны должны быть визуализированы на экране, что обеспечит удобство редактирования и контроля за результатом.

2.3.2 Настройка темпа воспроизведения композиции

Программа должна предоставлять пользователю возможность регулировать темп композиции через поле ввода BPM. Темп должен оказывать влияние на скорость воспроизведения паттернов и всех инструментов. В случае ввода недопустимого значения должна быть предусмотрена система уведомлений для информирования пользователя.

2.3.3 Воспроизведение созданной композиции

Программа должна предусматривать возможность воспроизведения созданной композиции. Для этого должна быть предусмотрена кнопка для начала воспроизведения, при этом синхронизация всех инструментов и паттернов должна соответствовать заданному темпу.

2.3.4 Изменение звуков инструментов

В программе должны быть предусмотрены функции для изменения звуков инструментов, включая возможность выбора нового WAV-файла для инструментов, таких как ударные. Это обеспечит гибкость настроек звука и позволит пользователю адаптировать композицию под свои предпочтения.

2.3.5 Сохранение композиции в аудиофайле формата WAV

Программа должна предоставлять возможность сохранить завершенную композицию в аудиофайле формата WAV. Для этого должны быть предусмотрены функции выбора места сохранения и задания имени файла.

2.3.6 Завершение работы программы

Программа должна поддерживать возможность корректного завершения работы с помощью кнопки закрытия окна.

# ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА

## Обобщенный алгоритм работы ПС

Блок-схема обобщённого алгоритма работы программного средства представлена в Приложении А, Рисунок 1.

## Разработка алгоритмов

3.2.1 Алгоритм воспроизведения созданной композиции

Для воспроизведения созданной композиции необходимо сначала создать буфер, который будет зранить данные wav-файла, затем сохранить его и открыть для воспроизведения. Блок-схема алгоритма представлена на рисунке 3.1.



Рисунок 3.1 – Блок-схема алгоритма воспроизведения созданной композиции

3.2.2 Алгоритм создания паттернов для пианино

Для создания паттерна пианино необходимо нажимать на контроллер нот пианино, добавляя или удаляя тем самым ноты. Блок-схема алгоритма представлена на рисунке 3.2.



Рисунок 3.2 – Блок-схема алгоритма создания паттернов для пианино

3.2.3 Алгоритм создания паттерна для ударных инструменов

Для создания паттерна ударных необходимо нажимать на контроллер ударных инструментов, добавляя или удаляя тем самым ноты. Блок-схема алгоритма представлена в рисунке 3.3.



Рисунок 3.3 – Блок-схема алгоритма создания паттерна для ударных инструменов

3.2.4 Настройка темпа композиции

Для изменения темпа композиции необходимо ввести его в поле ввода темпа. При значении менее 60 или более 200 выдается ошибка о неверном темпе. Блок-схема алгоритма представлена на рисунке 3.4.



Рисунок 3.4 – Блок-схема алгоритма настройки темпа композиции

3.2.5 Завершение работы программы

Для корректного завершения работы программы необходимо освобождать используемые ресурсы. Блок-схема алгоритма представлена на рисунке 3.5.



Рисунок 3.5 – Блок-схема алгоритма завершения работы программы

## Работа с WinApi

Ключевой технологией при разработке данного программного средства является WinApi. И функции, позволяющие получить доступ к контексту устройства отображения, сохранению данных в буфер обмена и файл. Также средствами WinApi реализуется основа приложения – создание окна и получение сообщений о действиях пользователя.

# СОЗДАНИЕ ПРОГРАММНОГО СРЕДСТВА

## Проектирование основных методов программного средства

Основные методы, использованные в разработке, перечислены в таблицах 4.1 – 4.7.

Таблица 4.1 – Основные методы модуля Controls.h

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Имя метода | Описание | Заголовок метода | Имя параметра | Назначение параметра |
| createSimple | Создает кнопку среднего размера с надписью | static Button\* createSimple(LPCWSTR caption, int x, int y, int id, HWND parent) | caption  x, y  id  parent | Надпись на кнопке  Координаты расположения на экране  Идентификатор кнопки для обработки сообщений  Дескриптор окна |
| createSmall | Создает кнопку малого размера без надписи | static Button\* createSmall(int x, int y, int id, HWND parent) | x, y  id  parent | Координаты расположения на экране  Идентификатор кнопки для обработки сообщений  Дескриптор окна |
| createFreq | Создает кнопку малого размера с надписью | static Button\* createFreq(LPCWSTR caption, int x, int y, int id, HWND parent) | caption  x, y  id  parent | Надпись на кнопке  Координаты расположения на экране  Идентификатор кнопки для обработки сообщений  Дескриптор окна |

Таблица 4.2 – Основные методы модуля ChannelRack.h

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Имя метода | Описание | Заголовок метода | Имя параметра | Назначение параметра |
| ChannelRack | Создает контроллер состояния ударных инструментов | ChannelRack(HWND hwnd, int x, int y) | hwnd  x, y | Дескриптор окна  Координаты расположения на экране |
| fillStatus | Заполняет партию ударных инструментов с определенной частотой | void fillStatus(int row, int freq) | row  freq | Номер ряда ударных инструментоа  Выбранная частота заполнения |
| getData | Возвращает паттерн состояния партии ударных инструментов | vector<vector<bool>> getData() | - | - |

Таблица 4.3 – Основные методы модуля KeyboardPiano.h

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Имя метода | Описание | Заголовок метода | Имя параметра | Назначение параметра |
| KeyboardPiano | Создает контроллер клавиатурного пианино | KeyboardPiano() | - | - |
| start | Запускает поток обработки нажатия и отпускания клавиш на клавиатуре | void start() | - | - |
| stop | Останавливает поток обработки нажатия и отпускания клавиш на клавиатуре | void stop() | - | - |
| onKeyDown | Обработчик нажатия клавиши на клавиатуре | void OnKeyDown(WPARAM wParam) | wParam | Код нажатой клавиши |
| onKeyUp | Обработчик отпускания клавиши на клавиатуре | void OnKeyUp(WPARAM wParam) | wParam | Код отпущенной клавиши |

Таблица 4.4 – Основные методы модуля MainWindow.h

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Имя метода | Описание | Заголовок метода | Имя параметра | Назначение параметра |
| MainWindow | Создает объект основного окна программы | MainWindow(HINSTANCE hInstance) | hInstance | Дескриптор экземпляра приложения |
| HandleMessage | Обработка сообщений от действий пользователя | LRESULT HandleMessage(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam) | hWnd  message  wParam  lParam | Дескриптор окна  Тип сообщения  Параметр сообщения  Параметр сообщения |
| OnCreate | Создание контроллов и кнопок на главном окне | void OnCreate(HWND hwnd) | hwnd | Дескриптор окна |
| OnCommand | Обработка сообщений от контроллов и кнопок | void OnCommand(HWND hwnd, WPARAM wParam, LPARAM lParam) | hWnd  wParam  lParam | Дескриптор окна  Параметр сообщения  Параметр сообщения |
| OnKeyDown | Обработка нажатия клавиши | void OnKeyDown(HWND hwnd, WPARAM wParam, LPARAM lParam); | hWnd  wParam  lParam | Дескриптор окна  Параметр сообщения  Параметр сообщения |
| OnLButtonDown | Обработка нажатия левой кнопки мыши | void OnLButtonDown(HWND hwnd, WPARAM wParam, LPARAM lParam) | hWnd  wParam  lParam | Дескриптор окна  Параметр сообщения  Параметр сообщения |

Таблица 4.5 – Основные методы модуля PianoRoll.h

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Имя метода | Описание | Заголовок метода | Имя параметра | Назначение параметра |
| PianoRoll | Создает объект объект контролллера пианино | PianoRoll(HWND hwnd, int x, int y) | hwnd  x, y | Дескриптор окна  Координаты расположения на экране |
| OnCommand | Обработка сообщений от контроллов и кнопок | void OnCommand(HWND hwnd, WPARAM wParam, LPARAM lParam) | hWnd  wParam  lParam | Дескриптор окна  Параметр сообщения  Параметр сообщения |
| OnLButtonDown | Обработка нажатия левой кнопки мыши | void OnLButtonDown(HWND hwnd, WPARAM wParam, LPARAM lParam) | hWnd  wParam  lParam | Дескриптор окна  Параметр сообщения  Параметр сообщения |
| OnPaint | Отрисовка контролла на экране | void OnPaint(HDC hdc) | hdc | Контекст отображения |

Таблица 4.6 – Основные методы модуля SoundManager.h

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Имя метода | Описание | Заголовок метода | Имя параметра | Назначение параметра |
| SoundManager | Создает объект объект контролллера звука | SoundManager(HWND hwnd, int x, int y) | hwnd  x, y | Дескриптор окна  Координаты расположения на экране |
| play | Проигрывание звука из буфера | void play() | - | - |
| stop | Прекращение проигрывания | void stop() | - | - |
| master | Формирование буфера звука из нот пианино и ударных инструментов | void master(const vector<vector<bool>>& drumData, const vector<Note>& pianoData) | drumData  pianoData | Матрица состояния ударных инструментов  Массив нот пианино |

Таблица 4.6 – Продолжение

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| saveToFile | Сохранение буфера звука в wav-файл | void saveToFile(const wstring& path) | path | Путь сохранения файла |

Таблица 4.7 – Основные методы модуля SoundManager.h

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Имя метода | Описание | Заголовок метода | Имя параметра | Назначение параметра |
| loadFromWav | Загрузка звука из wav-файла | void loadFromWav(const wstring& path) | path | Путь к файлу для загрузки |
| addToBuffer | Наложение двух звуков в один буфер | void addToBuffer(const WavSound& sound, int start, int duration) | sound  start  duration | Звук для наложения  Расстояние в байтах до места наложения  Длина наложения |
| openFileDialog | Выбор wav-файла для получения пути | static wstring openFileDialog(HWND hWnd) | hWnd | Дескриптор окна |
| getFileName | Получение имени файла из его пути | static wstring getFileName(const wstring& path) | path | Путь к файлу |

# ТЕСТИРОВАНИЕ, ПРОВЕРКА РАБОТОСПОСОБНОСТИ И АНАЛИЗ ПОЛУЧЕННЫХ РЕЗУЛЬТАТОВ

## Тестирование и проверка работоспособности программного средства

Этап тестирования играет ключевую роль в цикле разработки, учитывая неизбежность ошибок при написании программного кода. Основной задачей является выявление отличий в функционировании программы от установленных требований и последующее устранение этих разногласий с целью повышения качества программного продукта. В данном случае акцент был сделан на функциональном тестировании программы, направленном на проверку ее соответствия установленным спецификациям.

Функциональные тесты, проведенные над программой, представлены в таблице 5.1.

Таблица 5.1 – Результаты функционального тестирования

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Ном-ер  теста | Тестируе-мая функцио-нальность | Последовательность действий | Ожидаемый результат | Получен-ный результат |
| 1 | Запуск программы | 1. Запуск программы | Отображение окна программы. | Тест пройден, рисунок 5.1 |
| 2 | Заполнение паттерна ударных инструментов | 1. Нажать на кнопки в области контроллера ударных инструментов. 2. Нажать на кнопки, задающие частоту заполнения. | Заполнение паттерна ударных инструментов. | Тест пройден, рисунок 5.2 |
| 3 | Заполнение паттерна пианино | 1. Нажать на область контроллера пианино 2. При необходимости изменить длительность ноты, нажав на кнопки снизу | Заполнение паттерна пианино. | Тест пройден, рисунок 5.3 |

Продолжение таблицы 5.1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Ном-ер  теста | Тестируе-мая функцио-нальность | Последовательность действий | Ожидаемый результат | Получен-ный результат |
| 4 | Установка BPM больше разрешенного максимума | 1)Ввод числа “201” в область ввода BPM  2)Нажатие кнопки “Start” | Сообщение об ошибке с соответствующим текстом. | Тест пройден, рисунок 5.4 |
| 5 | Воспроизведение написанной мелодии | 1)Нажать на кнопку “Start” | Звук начал вопроизводиться. | Тест пройден. |
| 6 | Сохранение в wav-файл | 1)Нажать на кнопку “Render” | Открытие выбора пути нового файла и последующее сохранение. | Тест пройден. |
| 7 | Изменение звука ударного инструмента | 1)Нажатие на кнопку выбора звука инструмента  2)Выбор wav-файла для загруки | Изменение звука ударного инструмента. Изменение названия инструмента на панели кнопок. | Тест пройден, рисунок 5.5 |
| 8 | Завершение работы программы | 1)Нажатие на крестик в правом верхнем углу экрана | Конец работы программы. | Тест пройден. |

## Анализ полученных результатов

В результате тестирования предложенных алгоритмов были получены результаты, представленнные на рисунках 5.1 – 5.5.

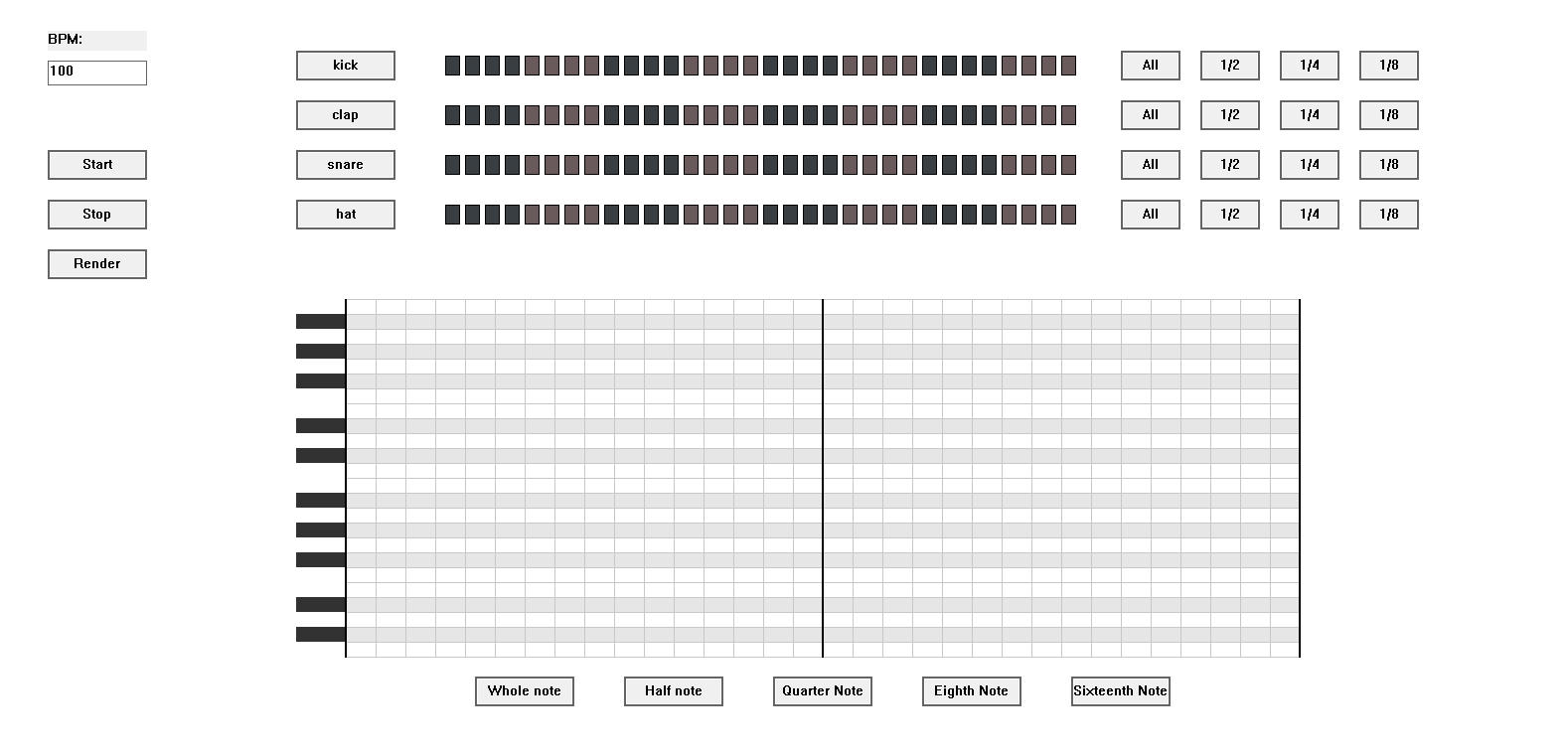


Рисунок 5.1 – Отображение окна программы

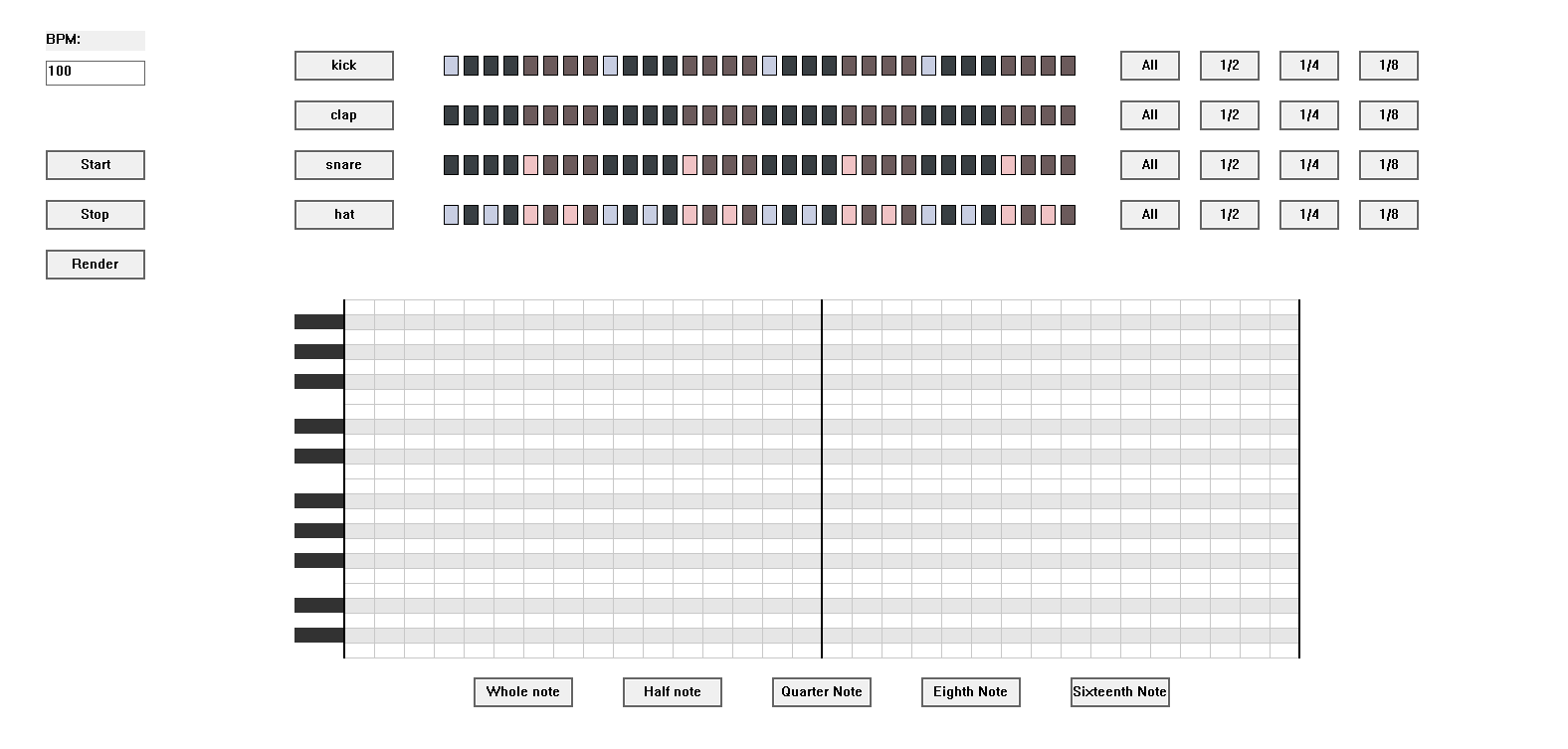


Рисунок 5.2 – Заполнение паттерна ударных инструментов

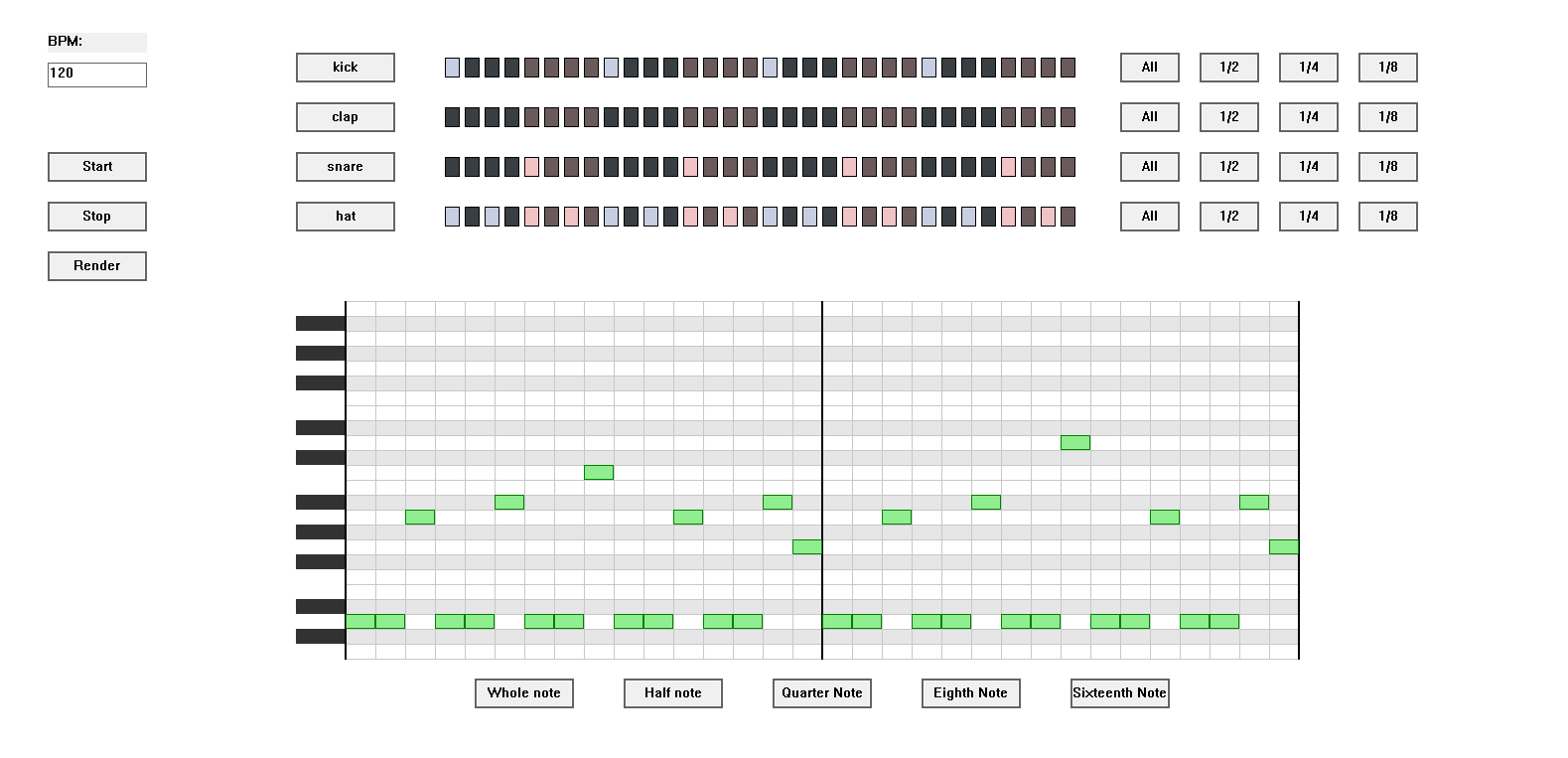


Рисунок 5.3 – Заполнение паттерна пианино

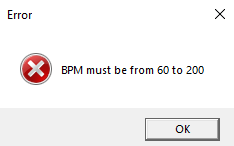


Рисунок 5.4 – Установка BPM больше разрешенного максимума

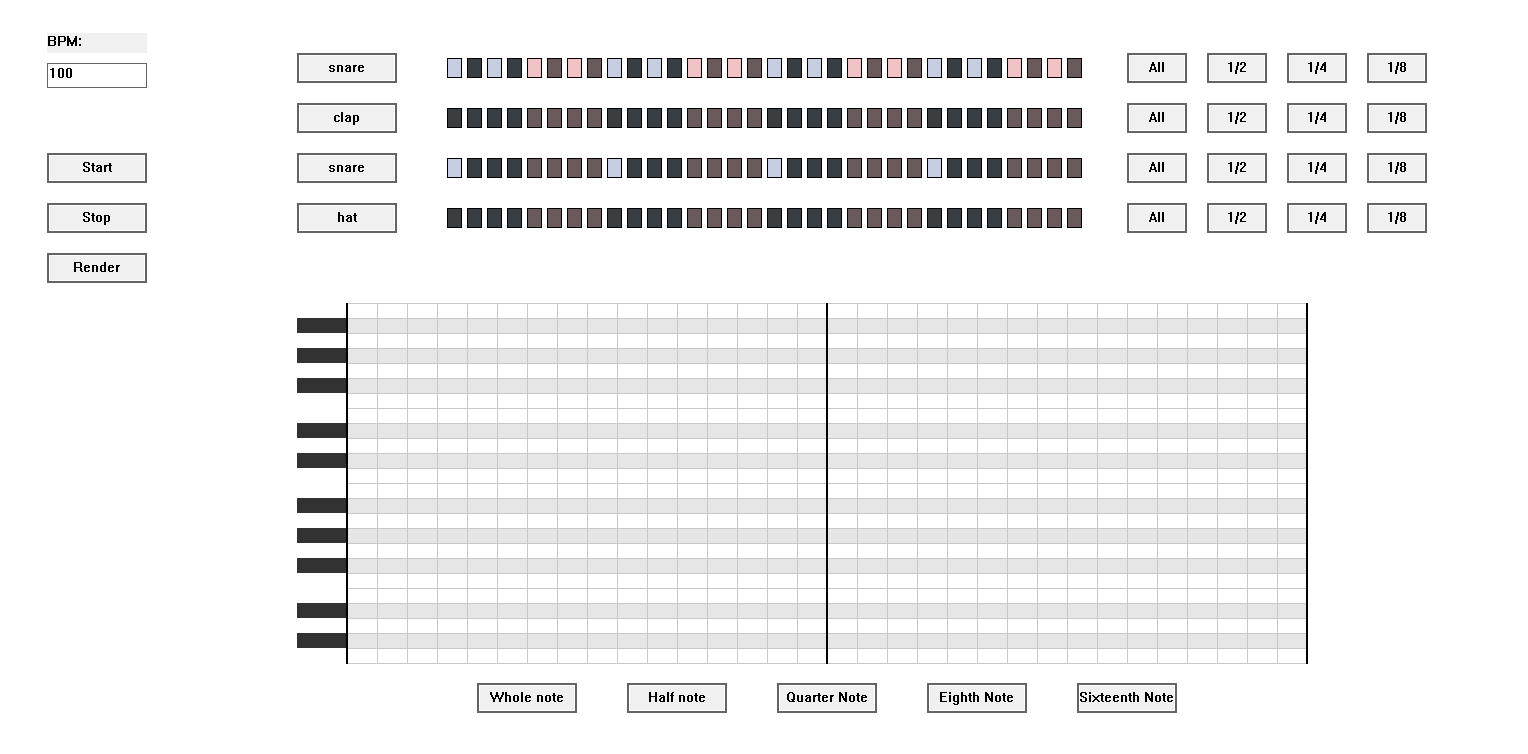


Рисунок 5.5 – Изменение звука ударного инструмента

В результате тестирования программного средства, все тесты прошли успешно, подтверждая стабильность и соответствие функциональности приложения поставленным требованиям.

# РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

Программа Music Sequencer предназначена для создания музыкальных композиций с использованием паттернов инструментов, управления темпом, настройки звуков и сохранения результата в аудиофайле. Для начала работы необходимо запустить приложение, дважды щелкнув по его ярлыку на рабочем столе или в меню «Пуск». После успешного запуска на экране отображается главное окно программы.

Создание паттерна ударных инструментов осуществляется посредством выбора кнопок в области контроллера ударных инструментов. Для настройки частоты заполнения используются кнопки, регулирующие данный параметр. По завершении действий паттерн отображается в интерфейсе программы. Для формирования паттерна пианино пользователь нажимает на соответствующую область контроллера пианино. При необходимости длительность нот можно скорректировать с использованием кнопок, расположенных в нижней части интерфейса. Результаты выполненных действий визуализируются на экране.

Темп воспроизведения композиции задается через поле ввода BPM. Пользователь вводит числовое значение, которое программа обрабатывает. В случае превышения допустимого значения при нажатии кнопки «Start» отображается сообщение об ошибке. Темп в пределах допустимого диапазона применяется при запуске воспроизведения.

Для начала воспроизведения композиции необходимо нажать кнопку «Start», после чего мелодия начнет звучать. Сохранение созданной композиции в формате WAV осуществляется путем нажатия кнопки «Render». В диалоговом окне пользователь выбирает место сохранения файла и указывает его имя. После завершения операции программа уведомляет о том, что файл успешно сохранен.

Изменение звука ударного инструмента производится через кнопку выбора звука инструмента. Пользователь выбирает WAV-файл из предлагаемого списка. После этого программа заменяет текущий звук, а новое название инструмента отображается на панели интерфейса.

Для завершения работы программы пользователь нажимает крестик в правом верхнем углу окна. Это действие приводит к завершению работы приложения, при этом текущие настройки сохраняются.

# ЗАКЛЮЧЕНИЕ

В ходе выполнения курсовой работы было разработано программное средство «Music Sequencer», предназначенное для создания музыкальных сэмплов. Данное программное средство предоставляет базовый функционал для написания партий пианино и ударных инструментов с использованием интерфейса пиано-ролла. Созданные композиции ограничены длиной в два такта и могут быть экспортированы в формате WAV, что обеспечивает их совместимость с другими музыкальными редакторами.

Основной функционал «Music Sequencer» включает возможность задания темпа композиции, что позволяет адаптировать музыкальные сэмплы к различным стилям и жанрам. Также пользователям предоставляется выбор звуков для ударных инструментов, что позволяет создавать партии с различными ритмическими текстурами. Простота и интуитивность интерфейса обеспечивают доступность программы для начинающих пользователей.

Программное средство использует пиано-ролл как основную область для создания партий. Этот инструмент позволяет наглядно и удобно размещать ноты, что снижает требования к музыкальной теории и делает процесс доступным даже для пользователей без специализированной подготовки. Возможность экспорта созданных музыкальных партий в формате WAV упрощает их дальнейшую обработку и интеграцию в более сложные музыкальные проекты.

Перспективы развития программного средства включают увеличение количества тактов для редактирования, расширение набора доступных инструментов и звуков, а также добавление новых функций, таких как наложение эффектов и улучшение интерфейса. Данные усовершенствования позволят сделать «Music Sequencer» более универсальным инструментом, подходящим как для начинающих, так и для более опытных пользователей. На текущем этапе разработка успешно решает поставленные задачи, предоставляя удобное средство для базового создания музыкальных композиций.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] WAV – Waveform Audio File Format (WAVE) — формат аудиофайлов для хранения звука в импульсно-кодовой модуляции. [Электронный ресурс]. – Режим доступа: [https:](%20https://apps.microsoft.com/%20detail/9MZ95KL8MR0L/) //ru.wikipedia.org/wiki/WAV Дата доступа 20.10.2024

[2] RIFF – Resource Interchange File Format — формат обмена ресурсами, используемый в аудиофайлах [Электронный ресурс]. – Режим доступа: <https://ru.wikipedia.org/wiki/RIFF> Дата доступа: 27.10.2024

[3] Разбор WAV-файла – пошаговая инструкция по обработке формата [Электронный ресурс]. – Режим доступа: <https://hasan-hasanov.com/post/2023/10/how_to_parse_wav_file/> Дата доступа: 10.11.2024

[4] FMOD – документация по работе с аудио API [Электронный ресурс]. – Режим доступа: <https://www.fmod.com/docs/2.03/api/welcome.html> Дата доступа: 10.11.2024

[5] WinAPI - набор базовых функций интерфейсов программирования приложений операционных систем семейств Microsoft Windows [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://learn.microsoft.com/ru-ru/windows/win32/learnwin32/learn-to-program-for-windows> Дата доступа: 21.11.2024

[6] СТП 01**–**2017, Стандарт предприятия, дипломные проекты, общие требования [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://library.bsuir.by/m/12_101945_1_141950.pdf> Дата доступа: 29.11.2024

# ПРИЛОЖЕНИЕ А (обязательное)

**Блок-схема алгоритма работы программы (к пункту 3.1)**



Рисунок A.1 – Блок-схема программы

# ПРИЛОЖЕНИЕ Б (обязательное)

**Листинг кода (к пункту 4)**

#include "framework.h"

#include <windows.h>

#include "Resource.h"

#include "MainWindow.h"

#pragma comment(lib, "core\\lib\\x64\\fmod\_vc.lib")

#define MAX\_LOADSTRING 100

HINSTANCE hInst;

int APIENTRY wWinMain(\_In\_ HINSTANCE hInstance, \_In\_opt\_ HINSTANCE hPrevInstance, \_In\_ LPWSTR lpCmdLine, \_In\_ int nCmdShow) {

UNREFERENCED\_PARAMETER(hPrevInstance);

UNREFERENCED\_PARAMETER(lpCmdLine);

MainWindow mainWindow(hInstance);

if (!mainWindow.InitInstance(nCmdShow)) {

return FALSE;

}

MSG msg;

while (GetMessage(&msg, nullptr, 0, 0)) {

TranslateMessage(&msg);

DispatchMessage(&msg);

}

return (int)msg.wParam;

}

#ifndef CHANNEL\_RACK\_H

#define CHANNEL\_RACK\_H

#define NOMINMAX

#include <windows.h>

#include "Controls.h"

#include <vector>

#include <memory>

#define CHANNEL\_RACK\_START\_ID 1000

#define CHANNEL\_RACK\_END\_ID 1999

using std::vector, std::tuple, std::unique\_ptr;

class ChannelRack final {

public:

ChannelRack(HWND hwnd, int x, int y);

~ChannelRack();

void toggleButton(int track, int step);

bool isClicked(WPARAM wParam, LPARAM lParam);

void onClick(HWND hwnd, WPARAM wParam, LPARAM lParam);

void fillStatus(int row, int freq);

vector<vector<bool>> getData();

void FillButtonColor(WPARAM wParam, LPARAM lParam);

private:

tuple<int, int, int, int> getCoord(int btnId);

pair<int, int> getRowCol(int btnId);

void createButtons(HWND hwnd);

const int numTracks{ 4 };

const int numSteps{ 32 };

const int posx, posy;

vector<vector<bool>> buttonStates;

HBRUSH lightBrush, pinkBrush, greyBrush, redBrush;

vector<vector<unique\_ptr<Button>>> buttons;

const int startId{ CHANNEL\_RACK\_START\_ID };

int lastId{ CHANNEL\_RACK\_START\_ID };

const int swidth{ 15 };

const int sheight{ 20 };

};

#endif

#ifndef WAV\_H

#define WAV\_H

#define NOMINMAX

#include <windows.h>

#include <string>

#include <vector>

#include <commdlg.h>

using namespace std;

typedef struct \_WAVHEADER

{

uint32\_t chunkId = 1179011410;

uint32\_t chunkSize = 0;

uint32\_t format = 1163280727;

uint32\_t subchunk1Id = 544501094;

uint32\_t subchunk1Size = 16;

uint16\_t audioFormat = 1;

uint16\_t numChannels = 2;

uint32\_t sampleRate = 44100;

uint32\_t byteRate = numChannels \* sampleRate \* 2;

uint16\_t blockAlign = 4;

uint16\_t bitsPerSample = 16;

uint32\_t subchunk2ID = 1635017060;

uint32\_t subchunk2Size = 0;

} WAVHEADER;

class WavSound final {

public:

WavSound();

void setSize(int size);

void addToBuffer(const WavSound& sound, int start);

void addToBuffer(const WavSound& sound, int start, int duration);

void loadFromWav(const wstring& path);

int getSize() const;

constexpr const int\* data() const;

void saveToFile(const std::wstring& path);

static wstring openFileDialog(HWND hWnd);

static wstring saveFileDialog(HWND hWnd);

static wstring getFileName(const wstring& path);

static bool fileExists(const wstring& path);

private:

vector<int> buffer{};

vector<short> wavData{};

bool checkWav(const WAVHEADER& header);

void master();

};

#endif

#ifndef SOUND\_MANAGER\_H

#define SOUND\_MANAGER\_H

#define NOMINMAX

#include <windows.h>

#include <vector>

#include <string>

#include <array>

#include "Controls.h"

#include "wav.h"

#include "core/inc/fmod.hpp"

#include "pianoRoll.h"

using FMOD::Sound;

using std::vector, std::array;

class SoundManager final {

public:

SoundManager(HWND hwnd, int x, int y);

~SoundManager();

void play();

void stop();

void saveToFile(const wstring& path);

void master(const vector<vector<bool>>& drumData, const vector<Note>& pianoData);

void setBpm(const int \_bpm);

bool isClicked(WPARAM wParam, LPARAM lParam);

void onClick(HWND hwnd, WPARAM wParam, LPARAM lParam);

private:

void loadDefaultSounds();

void createButtons(HWND hwnd, int x, int y);

int bpm{ 100 };

const int tacts{ 2 };

const int rate{ 44100 };

vector<unique\_ptr<Button>> sampleButtons;

vector<WavSound> loadedSamples;

WavSound masterSound;

unique\_ptr<char> wavData;

array<array<WavSound, 12>, 4> piano;

};

#endif

#ifndef MAIN\_WINDOW\_H

#define MAIN\_WINDOW\_H

#define NOMINMAX

#include <windows.h>

#include <memory>

#include "PianoRoll.h"

#include "ChannelRack.h"

#include "SoundManager.h"

#include "KeyboardPiano.h"

#include "PianoRoll.h"

class MainWindow final {

public:

MainWindow(HINSTANCE hInstance);

~MainWindow();

bool InitInstance(int nCmdShow);

static LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam);

private:

LRESULT HandleMessage(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam);

void OnCreate(HWND hwnd);

void OnPaint(HWND hwnd);

void OnCommand(HWND hwnd, WPARAM wParam, LPARAM lParam);

void OnKeyDown(HWND hwnd, WPARAM wParam, LPARAM lParam);

void OnKeyUp(HWND hwnd, WPARAM wParam, LPARAM lParam);

void OnLButtonDown(HWND hwnd, WPARAM wParam, LPARAM lParam);

void OnRButtonDown(HWND hwnd, WPARAM wParam, LPARAM lParam);

void master();

HWND hwndMain;

std::unique\_ptr<SoundManager> manager;

std::unique\_ptr<ChannelRack> channelRack;

std::unique\_ptr<BPMController> bpmController;

std::unique\_ptr<Button> buttonStart;

std::unique\_ptr<Button> buttonStop;

std::unique\_ptr<KeyboardPiano> keyboardPiano;

std::unique\_ptr<PianoRoll> pianoRoll;

std::unique\_ptr<Button> buttonSave;

};

#endif

#ifndef KEYBOARD\_PIANO\_H

#define KEYBOARD\_PIANO\_H

#include <windows.h>

#include <vector>

#include <array>

#include <mutex>

#include <unordered\_set>

#include "core/inc/fmod.hpp"

using std::vector, std::mutex, std::condition\_variable, std::thread;

using std::array, std::string, std::unordered\_set;

using FMOD::Channel, FMOD::Sound, FMOD::System;

class KeyboardPiano {

public:

KeyboardPiano();

~KeyboardPiano();

void start();

void stop();

void OnKeyDown(WPARAM wParam);

void OnKeyUp(WPARAM wParam);

void playNote(int note);

private:

int currentNote;

bool stopThread;

bool hasNewNote;

mutex mtx;

condition\_variable cv;

thread soundThread;

System\* system;

array<array<Sound\*, 12>, 3> piano;

unordered\_set<int> activeKeys;

void SoundWorker();

void loadDefaultSounds();

};

#endif  
#ifndef CONTROLS\_H

#define CONTROLS\_H

#define NOMINMAX

#include <windows.h>

#include <string>

#define BTNSTART 2

#define BTNSTOP 3

#define ID\_BPM 101

using std::wstring, std::pair;

class Button final {

public:

Button(LPCWSTR caption, int \_x, int \_y, int w, int h, int \_id, HWND parent, DWORD flag);

~Button();

static Button\* createSimple(LPCWSTR caption, int x, int y, int id, HWND parent);

static Button\* createSmall(int x, int y, int id, HWND parent);

static Button\* createFreq(LPCWSTR caption, int x, int y, int id, HWND parent);

int getId() const;

void setText(const wstring& text);

pair<int, int> getPos() const;

pair<int, int> getSize() const;

private:

HWND btnwnd;

int x, y, width, height, id;

};

class BPMController {

public:

BPMController(HWND parent, int x, int y);

static BPMController\* create(HWND parent, int x, int y);

int getValue();

void setValue(int value);

private:

HWND hLabel, hEdit;

};

#endif  
  
#ifndef PIANO\_ROLL\_H

#define PIANO\_ROLL\_H

#define NOMINMAX

#include <windows.h>

#include <vector>

#include <array>

#include <memory>

#include "Controls.h"

using std::vector, std::array, std::unique\_ptr;

struct Note {

int x, y;

int length;

};

class PianoRoll final {

public:

PianoRoll(HWND hwnd, int x, int y);

vector<Note> getData();

bool isClicked(HWND hwnd, WPARAM wParam, LPARAM lParam);

void OnCommand(HWND hwnd, WPARAM wParam, LPARAM lParam);

void OnLButtonDown(HWND hwnd, WPARAM wParam, LPARAM lParam);

void OnRButtonDown(HWND hwnd, WPARAM wParam, LPARAM lParam);

void OnPaint(HDC hdc);

private:

void DrawGrid(HDC hdc);

void DrawKeyboard(HDC hdc);

void DrawNotes(HDC hdc);

void AddNoteAt(int x, int y);

void RemoveNoteAt(int x, int y);

bool IsBlackKey(int midiNote);

vector<Note> notes;

array<unique\_ptr<Button>, 5> durationButtons;

int currentDuration = 4;

const int CELL\_WIDTH = 30;

const int CELL\_HEIGHT = 15;

const int ROWS = 24;

const int COLS = 32;

const int MEASURE\_DIVISION = 16;

const int KEYBOARD\_WIDTH = 100;

const int GRID\_LEFT = 50;

const COLORREF WHITE\_KEY\_COLOR = RGB(255, 255, 255);

const COLORREF BLACK\_KEY\_COLOR = RGB(50, 50, 50);

const COLORREF NOTE\_COLOR = RGB(144, 238, 144);

const COLORREF NOTE\_BORDER\_COLOR = RGB(0, 128, 0);

const COLORREF NOTE\_LINE\_COLOR = RGB(200, 200, 200);

const COLORREF MEASURE\_COLOR = RGB(0, 0, 0);

const COLORREF BLACK\_NOTE\_STRIP\_COLOR = RGB(230, 230, 230);

const int BUTTON\_HEIGHT = 40;

const int BUTTON\_WIDTH = 150;

const int BUTTON\_PADDING = 20;

const int x;

const int y;

};

#endif  
  
#include "wav.h"

#include <fstream>

#include <algorithm>

#include <iterator>

WavSound::WavSound() { }

bool WavSound::checkWav(const WAVHEADER& header) {

return (header.chunkId == 1179011410 &&

header.format == 1163280727);

}

void WavSound::loadFromWav(const wstring& path) {

if (!fileExists(path)) {

throw std::exception("Incorrect file path");

}

const wstring name = getFileName(path);

WAVHEADER wavHeader;

FILE\* wavFile;

const string pathStr(path.begin(), path.end());

fopen\_s(&wavFile, pathStr.c\_str(), "rb");

if (wavFile == nullptr) return;

if (size\_t bytesRead = fread(&wavHeader, 1, sizeof(WAVHEADER), wavFile))

{

if (!checkWav(wavHeader)) {

fclose(wavFile);

return;

}

int size = wavHeader.chunkSize / 2 - 200;

vector<short> shortBuffer(size);

size = static\_cast<int>(fread(shortBuffer.data(), sizeof(short), size, wavFile));

shortBuffer.resize(size);

buffer.assign(shortBuffer.begin(), shortBuffer.end());

}

fclose(wavFile);

}

constexpr const int\* WavSound::data() const {

return buffer.data();

}

void WavSound::setSize(int size) {

buffer.clear();

buffer.resize(size, 0);

}

int WavSound::getSize() const {

return static\_cast<int>(buffer.size());

}

void WavSound::addToBuffer(const WavSound& sound, int start) {

const int size = static\_cast<int>(buffer.size());

const int count = min(sound.getSize(), size - start);

auto src = sound.data();

auto srcend = std::next(src, count);

auto dest = std::next(buffer.begin(), start);

std::transform(

src,

srcend,

dest,

dest,

std::plus<int>()

);

}

void WavSound::addToBuffer(const WavSound& sound, int start, int duration) {

const int size = static\_cast<int>(buffer.size());

const int count = min({ sound.getSize(), size - start, duration });

auto src = sound.data();

auto srcend = std::next(src, count);

auto dest = std::next(buffer.begin(), start);

const int DECAY = min(2000, count);

auto decay\_start\_src = std::next(src, count - DECAY);

auto decay\_start\_dest = std::next(dest, count - DECAY);

std::transform(

src,

decay\_start\_src,

dest,

dest,

std::plus<int>()

);

std::transform(

decay\_start\_src,

srcend,

decay\_start\_dest,

decay\_start\_dest,

[=, velocity=DECAY] (const int src\_val, const int dest\_val) mutable {

const float k = static\_cast<float>(velocity) / DECAY;

const int val = static\_cast<int>(src\_val \* k);

velocity--;

return dest\_val + val;

}

);

}

void WavSound::master() {

const float volume = 0.8f;

auto [f, s] = std::minmax\_element(buffer.begin(), buffer.end());

const int mn = \*f, mx = \*s;

if (mn == 0 && mx == 0) return;

const float k = SHRT\_MAX \* volume / std::max(std::abs(mn), std::abs(mx));

wavData = std::vector<short>(buffer.size());

std::transform(

buffer.begin(),

buffer.end(),

wavData.begin(),

[=](const int elem) { return static\_cast<short>(elem \* k); }

);

}

void WavSound::saveToFile(const wstring& filename) {

master();

WAVHEADER header;

const int size = static\_cast<int>(buffer.size());

header.subchunk2Size = size \* sizeof(short);

header.chunkSize = 36 + header.subchunk2Size;

std::ofstream outFile(filename, std::ios::binary);

outFile.write(reinterpret\_cast<const char\*>(&header), sizeof(WAVHEADER));

outFile.write(reinterpret\_cast<const char\*>(wavData.data()), header.subchunk2Size);

outFile.close();

}

wstring WavSound::openFileDialog(HWND hWnd) {

OPENFILENAME ofn;

wchar\_t szFile[260] = { 0 };

ZeroMemory(&ofn, sizeof(ofn));

ofn.lStructSize = sizeof(ofn);

ofn.hwndOwner = hWnd;

ofn.lpstrFile = szFile;

ofn.nMaxFile = sizeof(szFile);

ofn.lpstrFilter = L"Wave Files\0\*.wav\0All Files\0\*.\*\0";

ofn.nFilterIndex = 1;

ofn.lpstrFileTitle = NULL;

ofn.nMaxFileTitle = 0;

ofn.lpstrInitialDir = NULL;

ofn.Flags = OFN\_PATHMUSTEXIST | OFN\_FILEMUSTEXIST;

if (GetOpenFileName(&ofn) == TRUE) {

return ofn.lpstrFile;

}

return L"";

}

std::wstring WavSound::saveFileDialog(HWND hWnd) {

OPENFILENAME ofn;

wchar\_t szFile[260] = { 0 };

ZeroMemory(&ofn, sizeof(ofn));

ofn.lStructSize = sizeof(ofn);

ofn.hwndOwner = hWnd;

ofn.lpstrFile = szFile;

ofn.nMaxFile = sizeof(szFile) / sizeof(szFile[0]);

ofn.lpstrFilter = L"Wave Files\0\*.wav\0All Files\0\*.\*\0";

ofn.nFilterIndex = 1;

ofn.lpstrFileTitle = NULL;

ofn.nMaxFileTitle = 0;

ofn.lpstrInitialDir = NULL;

ofn.Flags = OFN\_PATHMUSTEXIST | OFN\_OVERWRITEPROMPT;

if (GetSaveFileName(&ofn) == TRUE) {

std::wstring filePath = ofn.lpstrFile;

if (const size\_t length = filePath.length();

length < 4 || filePath.substr(length - 4) != L".wav") {

filePath += L".wav";

}

return filePath;

}

return L"";

}

bool WavSound::fileExists(const wstring& path) {

DWORD fileAttr = GetFileAttributesW(path.c\_str());

return (fileAttr != INVALID\_FILE\_ATTRIBUTES && !(fileAttr & FILE\_ATTRIBUTE\_DIRECTORY));

}

wstring WavSound::getFileName(const wstring& path)

{

size\_t lastSlashPos = path.find\_last\_of(L"\\/");

wstring fileName = (lastSlashPos == wstring::npos)

? path

: path.substr(lastSlashPos + 1);

size\_t lastDotPos = fileName.find\_last\_of(L'.');

if (lastDotPos != wstring::npos) {

fileName = fileName.substr(0, lastDotPos);

}

return fileName;

}

#include "MainWindow.h"

#include "Resource.h"

#include <windowsx.h>

#define BTN\_START 201

#define BTN\_STOP 202

#define BTN\_RENDER 203

MainWindow::MainWindow(HINSTANCE hInstance) {

WNDCLASSEX wcex = { sizeof(WNDCLASSEX) };

wcex.style = CS\_HREDRAW | CS\_VREDRAW;

wcex.lpfnWndProc = MainWindow::WndProc;

wcex.hInstance = hInstance;

wcex.hCursor = LoadCursor(nullptr, IDC\_ARROW);

wcex.hbrBackground = (HBRUSH)(COLOR\_WINDOW + 1);

wcex.lpszClassName = L"MainWindow";

RegisterClassEx(&wcex);

hwndMain = CreateWindow(L"MainWindow", L"Music Sequencer",

WS\_OVERLAPPEDWINDOW, CW\_USEDEFAULT, 0, 1600, 800,

nullptr, nullptr, hInstance, this);

}

MainWindow::~MainWindow() {

UnregisterClass(L"MainWindow", GetModuleHandle(nullptr));

}

bool MainWindow::InitInstance(int nCmdShow) {

if (!hwndMain) return false;

ShowWindow(hwndMain, nCmdShow);

UpdateWindow(hwndMain);

return true;

}

LRESULT CALLBACK MainWindow::WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam) {

MainWindow\* pThis = nullptr;

if (message == WM\_CREATE) {

LPCREATESTRUCT pCreate = reinterpret\_cast<LPCREATESTRUCT>(lParam);

pThis = reinterpret\_cast<MainWindow\*>(pCreate->lpCreateParams);

SetWindowLongPtr(hWnd, GWLP\_USERDATA, reinterpret\_cast<LONG\_PTR>(pThis));

}

else {

pThis = reinterpret\_cast<MainWindow\*>(GetWindowLongPtr(hWnd, GWLP\_USERDATA));

}

if (pThis) {

return pThis->HandleMessage(hWnd, message, wParam, lParam);

}

return DefWindowProc(hWnd, message, wParam, lParam);

}

LRESULT MainWindow::HandleMessage(HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam) {

switch (message) {

case WM\_CREATE:

OnCreate(hwnd);

break;

case WM\_PAINT: {

OnPaint(hwnd);

break;

}

case WM\_KEYDOWN:

OnKeyDown(hwnd, wParam, lParam);

break;

case WM\_KEYUP:

OnKeyUp(hwnd, wParam, lParam);

break;

case WM\_DRAWITEM:

channelRack->FillButtonColor(wParam, lParam);

break;

case WM\_COMMAND:

OnCommand(hwnd, wParam, lParam);

break;

case WM\_LBUTTONDOWN:

OnLButtonDown(hwnd, wParam, lParam);

break;

case WM\_RBUTTONDOWN:

OnRButtonDown(hwnd, wParam, lParam);

break;

case WM\_DESTROY:

PostQuitMessage(0);

break;

default:

return DefWindowProc(hwnd, message, wParam, lParam);

}

return 0;

}

void MainWindow::OnCreate(HWND hwnd) {

manager.reset(new SoundManager(hwnd, 300, 50));

bpmController.reset(new BPMController(hwnd, 50, 30));

channelRack.reset(new ChannelRack(hwnd, 450, 50));

pianoRoll.reset(new PianoRoll(hwnd, 300, 250));

keyboardPiano.reset(new KeyboardPiano());

keyboardPiano->start();

buttonStart.reset(Button::createSimple(L"Start", 50, 150, BTN\_START, hwnd));

buttonStop.reset(Button::createSimple(L"Stop", 50, 200, BTN\_STOP, hwnd));

buttonSave.reset(Button::createSimple(L"Render", 50, 250, BTN\_RENDER, hwnd));

}

void MainWindow::OnKeyDown(HWND hwnd, WPARAM wParam, LPARAM lParam) {

keyboardPiano->OnKeyDown(wParam);

}

void MainWindow::OnKeyUp(HWND hwnd, WPARAM wParam, LPARAM lParam) {

keyboardPiano->OnKeyUp(wParam);

}

void MainWindow::OnPaint(HWND hwnd) {

PAINTSTRUCT ps;

HDC hdc = BeginPaint(hwnd, &ps);

RECT clientRect;

GetClientRect(hwnd, &clientRect);

int width = clientRect.right - clientRect.left;

int height = clientRect.bottom - clientRect.top;

HDC memDC = CreateCompatibleDC(hdc);

HBITMAP memBitmap = CreateCompatibleBitmap(hdc, width, height);

HBITMAP oldBitmap = (HBITMAP)SelectObject(memDC, memBitmap);

HBRUSH backgroundBrush = CreateSolidBrush(RGB(255, 255, 255));

FillRect(memDC, &clientRect, backgroundBrush);

DeleteObject(backgroundBrush);

pianoRoll->OnPaint(memDC);

BitBlt(hdc, 0, 0, width, height, memDC, 0, 0, SRCCOPY);

SelectObject(memDC, oldBitmap);

DeleteObject(memBitmap);

DeleteDC(memDC);

EndPaint(hwnd, &ps);

}

void MainWindow::OnCommand(HWND hwnd, WPARAM wParam, LPARAM lParam) {

const int wmId{ static\_cast<int>(wParam) };

if (channelRack && channelRack->isClicked(wParam, lParam)) {

channelRack->onClick(hwnd, wParam, lParam);

SetFocus(hwnd);

}

else if (manager && manager->isClicked(wParam, lParam)) {

try {

manager->onClick(hwnd, wParam, lParam);

}

catch (std::exception e) {

string text = e.what();

wstring wtext(text.begin(), text.end());

MessageBoxW(hwnd, wtext.c\_str(), L"Ошибка", MB\_OK | MB\_ICONERROR);

}

SetFocus(hwnd);

}

else if (wmId == IDM\_EXIT) {

DestroyWindow(hwnd);

}

else if (wmId == BTN\_START) {

master();

manager->play();

SetFocus(hwnd);

}

else if (wmId == BTN\_STOP) {

manager->stop();

SetFocus(hwnd);

}

else if (wmId == BTN\_RENDER) {

master();

const wstring path = WavSound::saveFileDialog(hwnd);

manager->saveToFile(path);

}

else if (pianoRoll && pianoRoll->isClicked(hwnd, wParam, lParam)){

pianoRoll->OnCommand(hwnd, wParam, lParam);

}

}

void MainWindow::master() {

const auto drumData = channelRack->getData();

const auto pianoData = pianoRoll->getData();

int bpm{ 100 };

try {

bpm = bpmController->getValue();

}

catch (std::invalid\_argument e) {

bpmController->setValue(100);

string text(e.what());

wstring wtext(text.begin(), text.end());

MessageBox(

NULL,

wtext.c\_str(),

(LPCWSTR)L"Error",

MB\_ICONERROR | MB\_OK

);

}

manager->setBpm(bpm);

manager->master(drumData, pianoData);

}

void MainWindow::OnLButtonDown(HWND hwnd, WPARAM wParam, LPARAM lParam) {

if (pianoRoll->isClicked(hwnd, wParam, lParam)) {

pianoRoll->OnLButtonDown(hwnd, wParam, lParam);

}

SetFocus(hwnd);

}

void MainWindow::OnRButtonDown(HWND hwnd, WPARAM wParam, LPARAM lParam) {

if (pianoRoll->isClicked(hwnd, wParam, lParam)) {

pianoRoll->OnRButtonDown(hwnd, wParam, lParam);

}

SetFocus(hwnd);

}

#include "Controls.h"

#include <stdexcept>

Button::Button(LPCWSTR caption, int \_x, int \_y, int w, int h, int \_id, HWND parent, DWORD flag)

: x(\_x), y(\_y), width(w), height(h), id(\_id) {

btnwnd = CreateWindowEx(

0, L"BUTTON", caption,

WS\_TABSTOP | WS\_VISIBLE | WS\_CHILD | BS\_DEFPUSHBUTTON | BS\_FLAT | flag,

x, y, width, height, parent, reinterpret\_cast<HMENU>(id),

(HINSTANCE)GetWindowLongPtr(parent, GWLP\_HINSTANCE), NULL);

}

Button::~Button() { }

Button\* Button::createSimple(LPCWSTR caption, int x, int y, int id, HWND parent) {

return new Button(caption, x, y, 100, 30, id, parent, 0);

}

Button\* Button::createFreq(LPCWSTR caption, int x, int y, int id, HWND parent) {

return new Button(caption, x, y, 60, 30, id, parent, 0);

}

Button\* Button::createSmall(int x, int y, int id, HWND parent) {

return new Button(L"", x, y, 15, 20, id, parent, BS\_OWNERDRAW);

}

int Button::getId() const {

return id;

}

void Button::setText(const wstring& text) {

SetWindowText(btnwnd, text.c\_str());

}

pair<int, int> Button::getPos() const {

return { x, y };

}

pair<int, int> Button::getSize() const {

return { width, height };

}

BPMController::BPMController(HWND parent, int x, int y) {

hLabel = CreateWindow(

L"STATIC", L"BPM:",

WS\_VISIBLE | WS\_CHILD,

x, y, 100, 20,

parent, NULL, NULL, NULL

);

hEdit = CreateWindow(

L"EDIT", L"100",

WS\_VISIBLE | WS\_CHILD | WS\_BORDER | ES\_NUMBER,

x, y + 30, 100, 25,

parent, (HMENU)ID\_BPM, NULL, NULL

);

SendMessage(hEdit, EM\_SETLIMITTEXT, 3, 0);

}

BPMController\* BPMController::create(HWND parent, int x, int y) {

return new BPMController(parent, x, y);

}

int BPMController::getValue() {

wchar\_t buffer[16] = {};

const size\_t bufferSize = sizeof(buffer) / sizeof(buffer[0]);

GetWindowText(hEdit, buffer, bufferSize - 1);

if (buffer[0] == '\0') {

throw std::invalid\_argument("You must choose the tempo");

}

const int number = std::stoi(buffer);

if (number < 60 || number > 200) {

throw std::invalid\_argument("BPM must be from 60 to 200");

}

return number;

}

void BPMController::setValue(int value) {

wstring data = std::to\_wstring(value);

data.resize(16);

SetWindowText(hEdit, data.c\_str());

}

#include "ChannelRack.h"

#include <windowsx.h>

#include <algorithm>

#include <array>

using std::fill, std::array;

ChannelRack::ChannelRack(HWND hwnd, int x, int y)

: posx(x),

posy(y),

buttonStates(numTracks, vector<bool>(numSteps, false)) {

buttons.resize(numTracks);

std::ranges::for\_each(buttons, [&](auto& buttonRow) {

buttonRow.resize(numSteps + 4);

});

createButtons(hwnd);

lightBrush = CreateSolidBrush(RGB(200, 206, 226));

pinkBrush = CreateSolidBrush(RGB(240, 195, 197));

greyBrush = CreateSolidBrush(RGB(56, 62, 66));

redBrush = CreateSolidBrush(RGB(107, 90, 91));

}

void ChannelRack::createButtons(HWND hwnd) {

constexpr std::array<const LPCWSTR, 4> extraNames = { L"All", L"1/2", L"1/4", L"1/8" };

std::ranges::for\_each(buttons, [&, track = 0](auto& buttonRow) mutable {

std::ranges::for\_each(buttonRow, [&, step = 0](auto& button) mutable {

if (step < numSteps) {

const int x = posx + step \* 20;

const int y = posy + 5 + track \* 50;

button.reset(Button::createSmall(x, y, lastId++, hwnd));

}

else {

const int x = posx + (step + 3 \* (step - numSteps)) \* 20 + 40;

const int y = posy + track \* 50;

button.reset(Button::createFreq(extraNames[step - numSteps], x, y, lastId++, hwnd));

}

++step;

});

++track;

});

}

ChannelRack::~ChannelRack() { }

bool ChannelRack::isClicked(WPARAM wParam, LPARAM lParam) {

const int id = static\_cast<int>(wParam);

return id >= CHANNEL\_RACK\_START\_ID && id <= CHANNEL\_RACK\_END\_ID;

}

void ChannelRack::toggleButton(int track, int step) {

vector<bool>::reference ref = buttonStates[track][step];

ref = !ref;

}

void ChannelRack::onClick(HWND hwnd, WPARAM wParam, LPARAM lParam) {

const int btnId = static\_cast<int>(wParam);

const auto [row, col] = getRowCol(btnId);

if (col < 32) {

toggleButton(row, col);

}

else {

fillStatus(row, 1 << (col - 32));

}

const auto [x, y, w, h] = getCoord(wParam);

const RECT rect = { x - 10, y - 10, x + w + 10, y + h + 10 };

InvalidateRect(hwnd, &rect, TRUE);

}

void ChannelRack::fillStatus(int row, int freq) {

auto& vec = buttonStates[row];

vector<bool> status(32, false);

bool flag = false;

for (int i = 0; i < 32; i += freq) {

if (!vec[i]) flag = true;

status[i] = true;

}

if (flag) vec = status;

else fill(vec.begin(), vec.end(), false);

}

tuple<int, int, int, int> ChannelRack::getCoord(int btnId) {

const auto [row, col] = getRowCol(btnId);

if (col < 32 && row < 4) {

const auto elem = buttons[row][col].get();

const auto [x, y] = elem->getPos();

return { x, y, 15, 20};

}

else if (row < 4) {

const auto elem = buttons[row][0].get();

const auto [x, y] = elem->getPos();

return { x, y, 635, 20 };

}

return {posx, posy, 635, 200};

}

void ChannelRack::FillButtonColor(WPARAM wParam, LPARAM lParam) {

const int id = static\_cast<int>(wParam);

if (id < startId || id > lastId) return;

const size\_t index = (id - CHANNEL\_RACK\_START\_ID);

const auto [row, col] = getRowCol(id);

if (col >= 32) return;

const bool isSelected = buttonStates[row][col];

const bool mode = (col % 8 >= 4);

HBRUSH brush = nullptr;

if (isSelected && mode)

brush = pinkBrush;

else if (isSelected)

brush = lightBrush;

else if (mode)

brush = redBrush;

else

brush = greyBrush;

LPDRAWITEMSTRUCT pDIS = (LPDRAWITEMSTRUCT)lParam;

HDC hdc = pDIS->hDC;

RECT rect = pDIS->rcItem;

FillRect(hdc, &rect, brush);

FrameRect(hdc, &rect, (HBRUSH)GetStockObject(BLACK\_BRUSH));

SetTextColor(hdc, RGB(0, 0, 0));

SetBkMode(hdc, TRANSPARENT);

}

vector<vector<bool>> ChannelRack::getData() {

return buttonStates;

}

pair<int, int> ChannelRack::getRowCol(int btnId) {

const int id = btnId - startId;

const int track = id / 36;

const int step = id % 36;

return { track, step };

}  
  
#include "KeyboardPiano.h"

#include <algorithm>

#include <string>

#include <unordered\_map>

using std::lock\_guard, std::unique\_lock, std::unordered\_map, std::string, std::mutex;

KeyboardPiano::KeyboardPiano() : currentNote{ -1 }, stopThread(false), hasNewNote(false) {

FMOD::System\_Create(&system);

system->init(512, FMOD\_INIT\_NORMAL, nullptr);

loadDefaultSounds();

system->setDSPBufferSize(64, 4);

}

KeyboardPiano::~KeyboardPiano() {

stop();

std::ranges::for\_each(piano, [](auto& octave) {

std::ranges::for\_each(octave, [](auto& note) {

note->release();

});

});

system->close();

system->release();

}

void KeyboardPiano::loadDefaultSounds() {

static const array<string, 12> notes = {

"C", "C#", "D", "D#",

"E", "F", "F#", "G",

"G#", "A", "A#", "B"

};

std::ranges::for\_each(piano, [&, octave = 3](auto& row) mutable {

std::ranges::for\_each(row, [&, note = 0](auto& elem) mutable {

string path = "sounds\\piano\\" + notes[note] + std::to\_string(octave) + ".wav";

system->createSound(path.c\_str(), FMOD\_DEFAULT, nullptr, &elem);

note++;

});

octave++;

});

}

void KeyboardPiano::start() {

soundThread = std::thread(&KeyboardPiano::SoundWorker, this);

}

void KeyboardPiano::stop() {

{

lock\_guard<mutex> lock(mtx);

stopThread = true;

cv.notify\_one();

}

if (soundThread.joinable()) {

soundThread.join();

}

}

void KeyboardPiano::playNote(int note) {

lock\_guard<mutex> lock(mtx);

currentNote = note;

hasNewNote = true;

cv.notify\_one();

}

void KeyboardPiano::OnKeyDown(WPARAM wParam) {

static const unordered\_map<WPARAM, int> keyToNote = {

{'Z', 0}, {'S', 1}, {'X', 2}, {'D', 3}, {'C', 4},

{'V', 5}, {'G', 6}, {'B', 7}, {'H', 8}, {'N', 9},

{'J', 10}, {'M', 11}, {VK\_OEM\_COMMA, 12}, {'L', 13}, {VK\_OEM\_PERIOD, 14},

{VK\_OEM\_1, 15}, {VK\_OEM\_2, 16},

{'Q', 12}, {'2', 13}, {'W', 14},

{'3', 15}, {'E', 16}, {'R', 17}, {'5', 18}, {'T', 19},

{'6', 20}, {'Y', 21}, {'7', 22}, {'U', 23}, {'I', 24},

{'9', 25}, {'O', 26}, {'0', 27}, {'P', 28}, {VK\_OEM\_4, 29},

{VK\_OEM\_PLUS, 30}, {VK\_OEM\_6, 31}

};

const int note = static\_cast<int>(wParam);

if (!activeKeys.contains(note)) {

activeKeys.insert(note);

if (auto it = keyToNote.find(wParam); it != keyToNote.end()) {

playNote(it->second);

}

}

}

void KeyboardPiano::OnKeyUp(WPARAM wParam) {

const int note = static\_cast<int>(wParam);

activeKeys.erase(note);

}

void KeyboardPiano::SoundWorker() {

while (true) {

unique\_lock<mutex> lock(mtx);

cv.wait(lock, [this] { return hasNewNote || stopThread; });

if (stopThread) {

break;

}

if (currentNote != -1) {

const int t = currentNote / 12;

const int i = currentNote % 12;

system->playSound(piano[t][i], nullptr, false, nullptr);

system->update();

}

hasNewNote = false;

}

}  
#include "SoundManager.h"

#include <algorithm>

#define DEFAULT\_KICK\_PATH L"sounds/kick.wav"

#define DEFAULT\_CLAP\_PATH L"sounds/clap.wav"

#define DEFAULT\_SNARE\_PATH L"sounds/snare.wav"

#define DEFAULT\_HAT\_PATH L"sounds/hat.wav"

#define DEFAULT\_PIANO\_PATH L"sounds/piano/"

#define SOUND\_MANAGER\_START\_ID 2000

#define SOUND\_MANAGER\_END\_ID 2005

SoundManager::SoundManager(HWND hwnd, int x = 0, int y = 0) {

createButtons(hwnd, x, y);

loadDefaultSounds();

}

SoundManager::~SoundManager() { }

void SoundManager::saveToFile(const wstring& path) {

masterSound.saveToFile(path);

}

void SoundManager::createButtons(HWND hwnd, int x, int y) {

int lastId{ SOUND\_MANAGER\_START\_ID };

int track{ 0 };

for (int track = 0; track < 4; ++track) {

unique\_ptr<Button> uptr{ Button::createSimple(L"", x, y + 50 \* track, lastId++, hwnd) };

sampleButtons.emplace\_back(std::move(uptr));

}

}

void SoundManager::loadDefaultSounds() {

static const array<string, 12> notes = {

{"C", "C#", "D", "D#",

"E", "F", "F#", "G",

"G#", "A", "A#", "B"}

};

std::ranges::for\_each(piano, [&, t=1](auto& row) mutable {

std::ranges::for\_each(row, [&, t, i=0](auto& elem) mutable {

const auto& note = notes[i];

const wstring path = DEFAULT\_PIANO\_PATH + wstring(note.begin(), note.end()) + to\_wstring(t) + L".wav";

elem.loadFromWav(path.c\_str());

i++;

});

t++;

});

static const array<LPCWSTR, 4> defaultPaths = {

DEFAULT\_KICK\_PATH,

DEFAULT\_CLAP\_PATH,

DEFAULT\_SNARE\_PATH,

DEFAULT\_HAT\_PATH

};

std::ranges::for\_each(defaultPaths, [&, i=0](const auto& path) mutable {

WavSound sound;

sound.loadFromWav(path);

loadedSamples.emplace\_back(std::move(sound));

wstring name = WavSound::getFileName(path);

sampleButtons[i]->setText(name);

i++;

});

}

bool SoundManager::isClicked(WPARAM wParam, LPARAM lParam) {

const int id = static\_cast<int>(wParam);

return id >= SOUND\_MANAGER\_START\_ID && id <= SOUND\_MANAGER\_END\_ID;

}

void SoundManager::onClick(HWND hwnd, WPARAM wParam, LPARAM lParam) {

const int index = static\_cast<int>(wParam) - SOUND\_MANAGER\_START\_ID;

const wstring path = WavSound::openFileDialog(hwnd);

loadedSamples[index].loadFromWav(path);

const wstring name = WavSound::getFileName(path);

sampleButtons[index]->setText(name);

}

void SoundManager::master(const vector<vector<bool>>& drumData, const vector<Note>& pianoData) {

int tactSize = rate \* 60 \* 2 \* 4 / bpm;

int SIZE = tactSize \* tacts;

masterSound.setSize(SIZE);

for (size\_t index = 0; index < 4; ++index) {

auto& sample = loadedSamples[index];

const vector<bool>& vec = drumData[index];

WavSound curr;

curr.setSize(SIZE);

for (int i = 0; i < 32; ++i) {

if (vec[i])

curr.addToBuffer(sample, i \* tactSize / 16);

}

masterSound.addToBuffer(curr, 0);

}

for (const auto& elem : pianoData) {

int start = elem.x \* rate \* 30 / bpm;

int duration = elem.length \* rate \* 30 / bpm;

int note = elem.y + 24;

const auto& sample = piano[note / 12][note % 12];

masterSound.addToBuffer(sample, start, duration);

}

}

void SoundManager::setBpm(const int \_bpm) {

bpm = \_bpm;

}

void SoundManager::play() {

masterSound.saveToFile(L"output.wav");

PlaySound(L"output.wav",

NULL, SND\_FILENAME | SND\_ASYNC | SND\_LOOP);

}

void SoundManager::stop() {

PlaySound(nullptr, nullptr, 0);

}

#include "PianoRoll.h"

#include <algorithm>

#define BTN\_DURATION 205

PianoRoll::PianoRoll(HWND hwnd, int \_x, int \_y) : x(\_x), y(\_y){

const int yPos = 50 + ROWS \* CELL\_HEIGHT + 20;

const array<LPCWSTR, 5> btnNames = { {

L"Whole note",

L"Half note",

L"Quarter Note",

L"Eighth Note",

L"Sixteenth Note"

} };

std::ranges::for\_each(durationButtons, [=, nameIndex=0, dx=50](auto& button) mutable {

const LPCWSTR name = btnNames[nameIndex];

const int btnId = BTN\_DURATION + nameIndex;

button.reset(Button::createSimple(name, x + 130 + dx, y + yPos, btnId, hwnd));

nameIndex++;

dx += 150;

});

}

vector<Note> PianoRoll::getData() {

return notes;

}

bool PianoRoll::isClicked(HWND hwnd, WPARAM wParam, LPARAM lParam) {

const int posx = LOWORD(lParam);

const int posy = HIWORD(lParam);

const int localX = posx - x - GRID\_LEFT;

const int localY = posy - y - 50;

const int gridX = localX / CELL\_WIDTH;

const int gridY = localY / CELL\_HEIGHT;

if (gridX >= 0 && gridX < COLS && gridY >= 0 && gridY < ROWS) {

return true;

}

const int btnId = static\_cast<int>(wParam);

return (btnId >= BTN\_DURATION) && (btnId <= BTN\_DURATION + 4);

}

void PianoRoll::OnCommand(HWND hwnd, WPARAM wParam, LPARAM lParam) {

const int btnId = static\_cast<int>(wParam);

const int index = btnId - BTN\_DURATION;

static const array<int, 5> durations = { {16, 8, 4, 2, 1} };

currentDuration = durations[index];

}

void PianoRoll::OnLButtonDown(HWND hwnd, WPARAM wParam, LPARAM lParam) {

const int mouseX = LOWORD(lParam);

const int mouseY = HIWORD(lParam);

const int localX = mouseX - x - GRID\_LEFT;

const int localY = mouseY - y - 50;

const int gridX = localX / CELL\_WIDTH;

const int gridY = ROWS - 1 - localY / CELL\_HEIGHT;

if (gridX >= 0 && gridX < COLS && gridY >= 0 && gridY < ROWS) {

AddNoteAt(gridX, gridY);

}

const RECT rect = { mouseX - CELL\_WIDTH \* 16, mouseY - CELL\_HEIGHT, mouseX + CELL\_WIDTH \* 16, mouseY + CELL\_HEIGHT};

InvalidateRect(hwnd, &rect, TRUE);

}

void PianoRoll::OnRButtonDown(HWND hwnd, WPARAM wParam, LPARAM lParam) {

const int mouseX = LOWORD(lParam);

const int mouseY = HIWORD(lParam);

const int localX = mouseX - x - GRID\_LEFT;

const int localY = mouseY - y - 50;

const int gridX = localX / CELL\_WIDTH;

const int gridY = ROWS - 1 - localY / CELL\_HEIGHT;

if (gridX >= 0 && gridX < COLS && gridY >= 0 && gridY < ROWS) {

RemoveNoteAt(gridX, gridY);

}

const RECT rect = { mouseX - CELL\_WIDTH \* 16, mouseY - CELL\_HEIGHT, mouseX + CELL\_WIDTH \* 16, mouseY + CELL\_HEIGHT };

InvalidateRect(hwnd, &rect, TRUE);

}

void PianoRoll::OnPaint(HDC hdc) {

DrawKeyboard(hdc);

DrawGrid(hdc);

DrawNotes(hdc);

}

void PianoRoll::DrawKeyboard(HDC hdc) {

HBRUSH whiteBrush = CreateSolidBrush(WHITE\_KEY\_COLOR);

HBRUSH blackBrush = CreateSolidBrush(BLACK\_KEY\_COLOR);

for (int i = 0; i < ROWS; ++i) {

RECT keyRect = { x, y + 50 + i \* CELL\_HEIGHT, x + KEYBOARD\_WIDTH, y + 50 + (i + 1) \* CELL\_HEIGHT };

if (IsBlackKey(ROWS - i - 1)) {

FillRect(hdc, &keyRect, blackBrush);

}

else {

FillRect(hdc, &keyRect, whiteBrush);

}

}

DeleteObject(whiteBrush);

DeleteObject(blackBrush);

}

void PianoRoll::DrawGrid(HDC hdc) {

HPEN gridPen = CreatePen(PS\_SOLID, 1, NOTE\_LINE\_COLOR);

HPEN measurePen = CreatePen(PS\_SOLID, 2, MEASURE\_COLOR);

HBRUSH stripBrush = CreateSolidBrush(BLACK\_NOTE\_STRIP\_COLOR);

SelectObject(hdc, gridPen);

for (int i = 0; i <= ROWS; ++i) {

if (IsBlackKey(ROWS - i - 1)) {

RECT stripRect = { x + GRID\_LEFT, y + 50 + i \* CELL\_HEIGHT, x + GRID\_LEFT + COLS \* CELL\_WIDTH, y + 50 + (i + 1) \* CELL\_HEIGHT };

FillRect(hdc, &stripRect, stripBrush);

}

MoveToEx(hdc, x + GRID\_LEFT, y + 50 + i \* CELL\_HEIGHT, NULL);

LineTo(hdc, x + GRID\_LEFT + COLS \* CELL\_WIDTH, y + 50 + i \* CELL\_HEIGHT);

}

for (int j = 0; j <= COLS; ++j) {

SelectObject(hdc, (j % MEASURE\_DIVISION == 0) ? measurePen : gridPen);

MoveToEx(hdc, x + GRID\_LEFT + j \* CELL\_WIDTH, y + 50, NULL);

LineTo(hdc, x + GRID\_LEFT + j \* CELL\_WIDTH, y + 50 + ROWS \* CELL\_HEIGHT);

}

DeleteObject(gridPen);

DeleteObject(measurePen);

DeleteObject(stripBrush);

}

void PianoRoll::DrawNotes(HDC hdc) {

HBRUSH noteBrush = CreateSolidBrush(NOTE\_COLOR);

HPEN notePen = CreatePen(PS\_SOLID, 1, NOTE\_BORDER\_COLOR);

SelectObject(hdc, noteBrush);

SelectObject(hdc, notePen);

for (const Note& note : notes) {

RECT noteRect = {

x + GRID\_LEFT + note.x \* CELL\_WIDTH,

y + 50 + (ROWS - note.y) \* CELL\_HEIGHT,

x + GRID\_LEFT + (note.x + note.length) \* CELL\_WIDTH,

y + 50 + (ROWS - note.y - 1) \* CELL\_HEIGHT

};

Rectangle(hdc, noteRect.left, noteRect.top, noteRect.right, noteRect.bottom);

}

DeleteObject(noteBrush);

DeleteObject(notePen);

}

void PianoRoll::AddNoteAt(int x, int y) {

if (x < 0 || x >= COLS || y < 0 || y >= ROWS) return;

const int maxLength = std::min(currentDuration, COLS - x);

const auto it = std::find\_if(notes.begin(), notes.end(), [x, y](const Note& note) {

return note.y == y && x >= note.x && x < note.x + note.length;

});

if (it == notes.end()) {

notes.push\_back({ x, y, maxLength });

}

}

void PianoRoll::RemoveNoteAt(int x, int y) {

notes.erase(std::remove\_if(notes.begin(), notes.end(), [x, y](const Note& note) {

return note.y == y && x >= note.x && x < note.x + note.length;

}), notes.end());

}

bool PianoRoll::IsBlackKey(int midiNote) {

static const int blackKeys[] = { 1, 3, 6, 8, 10 };

return std::find(std::begin(blackKeys), std::end(blackKeys), midiNote % 12) != std::end(blackKeys);

}