Министерство образования и науки Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего образования
ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
СИСТЕМ УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ (ТУСУР)
Кафедра автоматизированных систем управления (АСУ)

СОЗДАНИЕ СИНТЕЗАТОРА

Пояснительная записка к курсовому проекту

по дисциплине «Объектно-ориентированное программирование»

Студент гр. 442–1

_____ Е. М. Юскеев

«___» _____2024 г.

Руководитель

доцент каф. АСУ, к.т.н:

_____ С. М. Алфёров

«___» _____2024 г.

2024

Министерство образования и науки Российской Федерации
Федеральное государственное автономное образовательное учреждение

высшего образования

ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ СИСТЕМ
УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ (ТУСУР)

Кафедра автоматизированных систем управления (АСУ)

ЗАДАНИЕ К КУРСОВОМУ ПРОЕКТУ

по дисциплине «Объектно-ориентированное программирование»

Студенту группы 442–1 Юскееву Егору Мусаевичу

1. Тема работы: создание синтезатора на языке C++;
2. Срок сдачи студентом законченной работы “__” _____ 20__ г.
3. Исходные данные проекта: постановка задачи;
4. Перечень подлежащих разработке вопросов:

4.1. Постановка задач;

4.2. Обзор средств;

4.3. Описание проекта;

5. Дата выдачи задания: «__» _____ 20__ г.

Руководитель доцент каф. АСУ, к.т.н. _____ Алфёров С. М.

Задание принял к использованию _____ Юскеев Е.М. «__» _____ 20__ г.

Реферат

Данный отчет содержит: 31 страницу, 1 таблицу, 10 рисунков, 1 приложение, 2 источника.

Ключевые слова: Синтезатор, Звук, C++.

Цель работы – получить практические навыки работы в среде разработки Visual Studio на объектно-ориентированном языке C++, а также закрепить навыки, приобретенные за прохождение курса ООП.

Цель синтезатора состоит в том, чтобы дать пользователю возможность получить или улучшить навыки игры на классическом инструменте.

Как только пользователь запускает программу, он имеет возможность выбора между двумя инструментами (Пианино, Электронный синтезатор), а также возможность сыграть и записать мелодию.

После того, как пользователь начинает прослушивать запись мелодии, кнопки управления игрой синтезатора блокируются, после прослушивания, пользователь имеет возможность перезаписать мелодию, прослушать записанную мелодию снова, продолжить игру на инструменте по своему усмотрению.

Оглавление

Введение

Целью данной курсовой работы является получить практические навыки работы в среде разработки Visual Studio на объектно-ориентированном языке C++, а также закрепить навыки, приобретенные за прохождение курса ООП проектирования путём создания синтезатора.

Задачи:

1. Спроектировать логику и интерфейс приложения;
2. Изучить методологию DFD;
3. Разработать программный продукт, совмещающий логику и интерфейс приложения.

В данной работе предполагается использование методологии (DFD) Data Flow Diagram и (UML) Unified Modeling Language.

1 Постановка задачи

1.1 Список функций

Требуется создать на языке C++ программное приложение «Синтезатор», которое должно включать в себя следующие функции:

— воспроизведение ноты по нажатию соответствующей кнопки на клавиатуре;

— выбор музыкального инструмента между пианино и электронным синтезатором;

— запись мелодии по нажатию соответствующей кнопки;

— воспроизведение записанной мелодии по нажатию соответствующей кнопки.

1.2 Список входных данных

Список входных данных:

— нажатые клавиш пианино/кнопок синтезатора;

— музыкальный инструмент.

— данные мелодии (последовательность: нот, длительности и т.д.);

1.3 Список выходных данных

Программа выводит следующие параметры:

— звуковой сигнал (звук ноты);

— визуальное отображение нажатой клавиши;

— записанная мелодия

— длительность записанной мелодии

Приведем DFD диаграммы потока данных программы (рисунок 1.1-1.2).

Рисунок 1.1 — Поток входных-выходных данных программы нулевого уровня

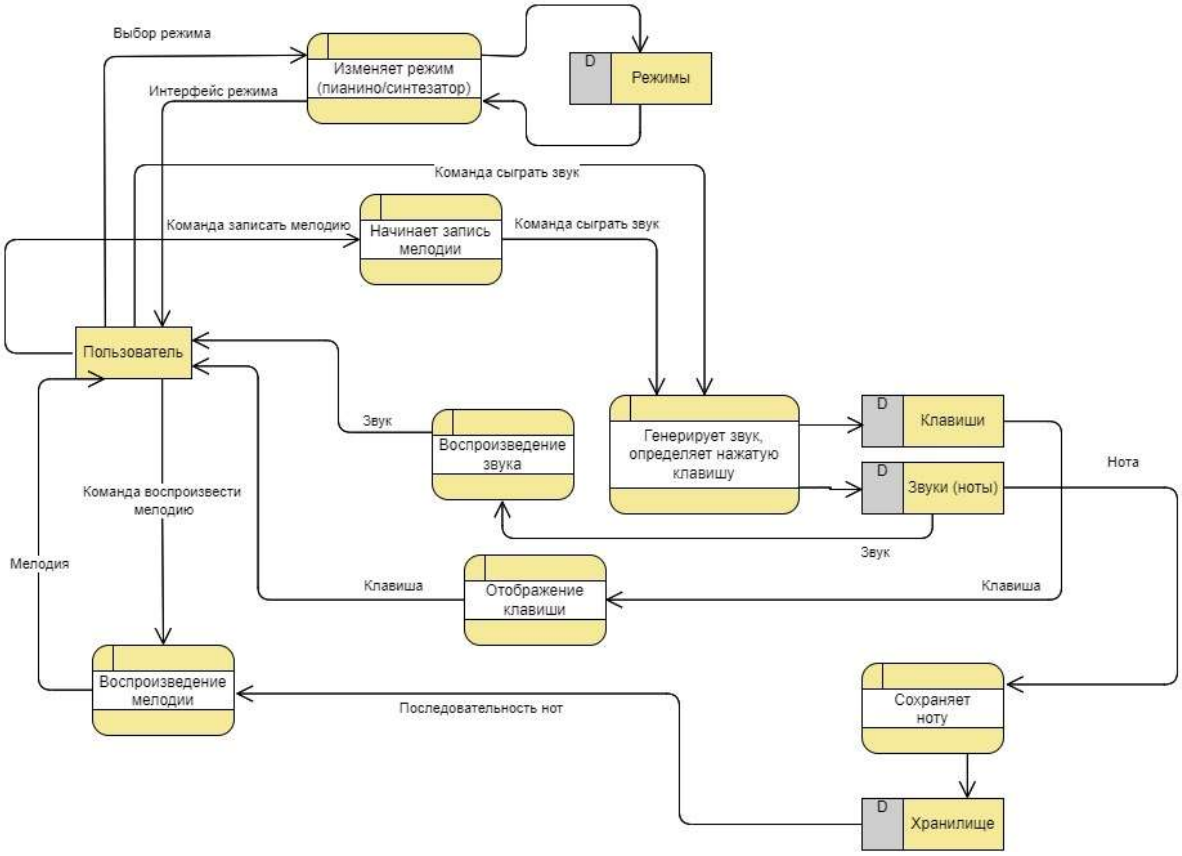

Рисунок 1.2 — Поток входных-выходных данных программы первого уровня (настройки и команда дать пример вместе)

1.4 Ограничения

— каждая клавиша на клавиатуре инструмента соответствует определенной ноте;

— каждая клавиша на клавиатуре ПК соответствует определенной клавише инструмента

— музыкальные инструменты можно выбирать;

— длительность нажатия клавиши влияет на длительность воспроизводимой ноты;

— при воспроизведении записанной мелодии проигрывание нажатых пользователей клавиш инструмента должно быть заблокировано.

2 Обзор средств

2.1 Обзор среды разработки

Visual Studio – интегрированная среда разработки (IDE) для создания программных продуктов, в которой можно писать на множестве языков программирования, например C#, C++, JavaScript, Python. Система полностью конфигурируема, масштабируется подключением автономных модулей (плагинов). Имеется отдельная редакция для macOS. Продукт распространяется в разных редакциях.

1. Visual Studio Community бесплатна и предназначена для индивидуальных разработчиков, студентов, академических учреждений и небольших команд, работающих над открытыми или некоммерческими проектами.

2. Visual Studio Professional и Enterprise являются платными версиями. Первая предназначена для индивидуальных разработчиков и небольших команд, разрабатывающие коммерческие проекты, а вторая для более крупных команд и предприятий, требующих самых мощных и продвинутых функций для разработки, тестирования и управления проектами.

Интерфейс Visual Studio имеет официальную версию на русском. Интегрированная среда содержит инструменты отладки и перевода программных строк в машинный код.[1]

2.2 Обзор языка программирования

C++ (читается «Си плюс плюс») — это компилируемый, статически типизированный язык программирования общего назначения. Наибольшее внимание в нем уделено поддержке объектно-ориентированного и обобщённого программирования.

C++ сочетает свойства как высокоуровневых, так и низкоуровневых языков программирования.

Первые версии языка C++ появились в начале 1980-х годов , когда их создатель — датский программист из компании Bell Laboratories Бьерн Страуструп – моделировал распределение вызовов между телефонными станциями и решил усовершенствовать язык программирования C («Си»), на котором тогда работал.

Этот язык отличается от Си тем, что имеет больший набор возможностей, включая объектно-ориентированное программирование и шаблоны. C++ используется для создания программного обеспечения разного рода: от игр до операционных систем. Этот язык также широко применяется в интенсивной обработке данных и научных расчетах. Данный язык программирования предоставляет разработчикам мощный и гибкий инструмент для создания программного обеспечения. Он позволяет писать эффективный и быстрый код, что делает его одним из наиболее популярных языков программирования в мире [2].

3 Описание проекта

3.1 Стандартные библиотеки и заголовки

В данном проекте были использованы следующие стандартные библиотеки, каждая из которых выполняет определенные функции:

SFML/Audio.hpp – библиотека для работы с аудио, включая воспроизведение звуков и музыку.

SFML/Graphics.hpp – библиотека для работы с графикой, включая отрисовку графических объектов.

iostream – стандартная библиотека C++, содержащая потоки ввода-вывода.

vector – стандартная библиотека для работы с динамическими массивами.

cmath – стандартная библиотека, предоставляющая математические функции.

chrono – библиотека для работы с временем и датами.

thread – библиотека для работы с потоками.

Перечисления:

SoundType – перечисление, определяющее тип звука (Piano или Electronic).

Структуры:

Note – структура для представления ноты с атрибутами:

std::string noteName – название ноты.

std::chrono::milliseconds startTime – время начала воспроизведения ноты.

std::chrono::milliseconds duration – длительность ноты.

Классы:

PianoKey – класс для представления клавиши пианино.

Synthesizer – класс для загрузки клавиш пианино и настройки текстовых объектов.

3.2 Объектная декомпозиция

3.2.1 Описание классов

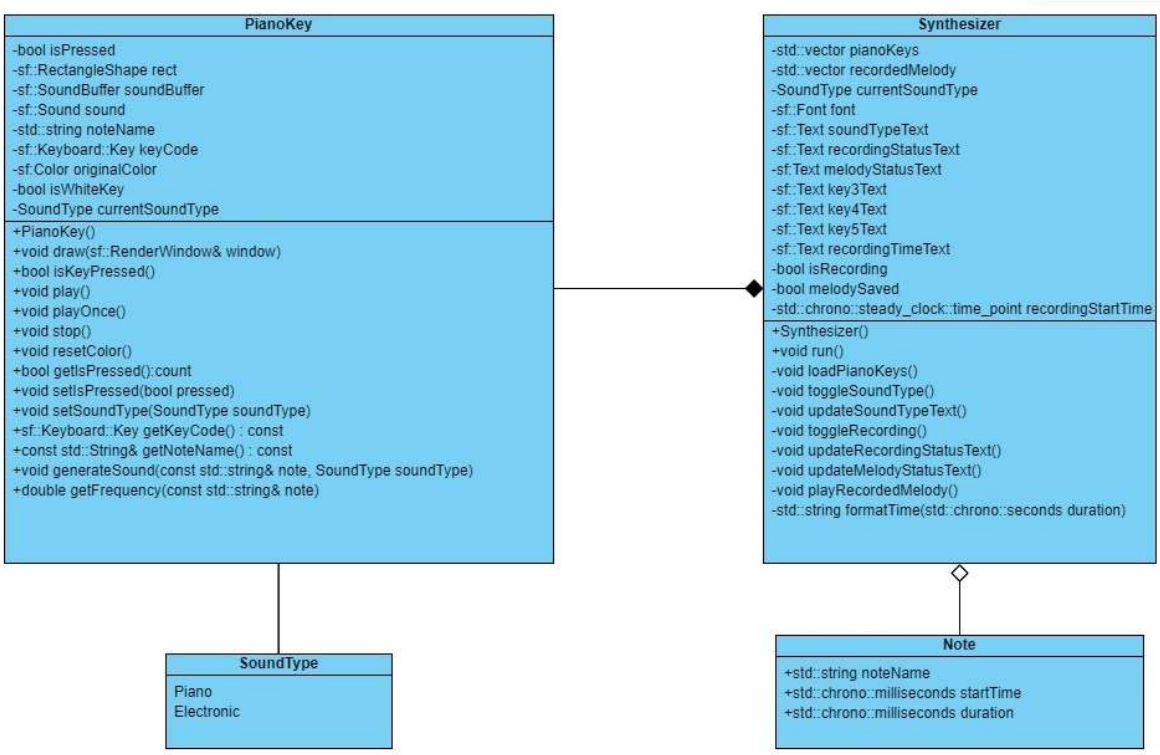

Рисунок 3.1 – Диаграмма классов приложения

Для реализации поставленной задачи были созданы следующие классы и структура:

— Note – структура, которая представляет ноту и содержит её название, время начала и продолжительность;

— PianoKey – класс, который представляет клавишу пианино и содержит методы для её отрисовки, проверки нажатия, воспроизведения и остановки звука, а также методы для управления её состоянием и звуком;

— Synthesizer – класс, который представляет собой само приложение «Синтезатор» и содержит переменные и методы для управления приложением, такие как загрузка клавиш пианино, запись и воспроизведение мелодий, а также обновление текстовых меток и интерфейса;

— SoundType – перечисление, которое определяет тип звука (например, пианино или электронный звук).

3.2.2 Спецификации

Таблица 3.1 – Спецификация функций

Прототип	Описание
Класс PianoKey	
`PianoKey(float x, float y, float width, float height, const sf::Color& color, const std::string& note, sf::Keyboard::Key key, bool isWhite, SoundType soundType)`	Конструктор клавиши пианино, задает позицию, размеры, цвет, ноту, клавишу, тип клавиши (белая/черная) и тип звука.
`void draw(sf::RenderWindow& window)`	Отрисовывает клавишу пианино на окне.
`bool isKeyPressed()`	Проверяет, нажата ли клавиша на клавиатуре, соответствующая клавише пианино.
`void play()`	Воспроизводит звук клавиши, изменяет цвет клавиши при нажатии.
`void playOnce()`	Однократно воспроизводит звук клавиши.
`void stop()`	Останавливает воспроизведение звука и сбрасывает цвет клавиши.
`void resetColor()`	Сбрасывает цвет клавиши на оригинальный.
`bool getIsPressed() const`	Возвращает состояние нажатия клавиши.
`void setIsPressed(bool pressed)`	Устанавливает состояние нажатия клавиши.
`void setSoundType(SoundType soundType)`	Устанавливает тип звука клавиши и генерирует звук.
`sf::Keyboard::Key getKeyCode() const`	Возвращает код клавиши.
`const std::string& getNoteName() const`	Возвращает название ноты.
`void generateSound(const std::string& note, SoundType soundType)`	Генерирует звук для заданной ноты и типа звука.
`double getFrequency(const std::string& note)`	Возвращает частоту для заданной ноты.
Класс Synthesizer	
`Synthesizer()`	Конструктор синтезатора, инициализирует клавиши пианино, загружает шрифты и текст для интерфейса.
`void run()`	Запускает основной цикл синтезатора, обрабатывает события и отрисовывает интерфейс.
`void loadPianoKeys()`	Загружает клавиши пианино и устанавливает их параметры.
`void toggleSoundType()`	Переключает тип звука между пианино и электронным.
`void updateSoundTypeText()`	Обновляет текст с текущим типом звука.
`void toggleRecording()`	Включает или выключает запись мелодии.
`void updateRecordingStatusText()`	Обновляет текст с текущим статусом записи.
`void updateMelodyStatusText()`	Обновляет текст с текущим статусом мелодии.
`void playRecordedMelody()`	Воспроизводит записанную мелодию.
`std::string formatTime(std::chrono::seconds duration)`	Форматирует время записи в строку.

3.3 Алгоритмы обработки

Для использования в программе были разработаны несколько алгоритмов. Приведем ниже описание и блок-схемы.

Ниже представлены методы класса Synthesizer. Он является основообразующим классом для этой программы. Алгоритм метода run, который отвечает за запуск синтезатора и обработку событий, представлен на рисунке (3.2.).

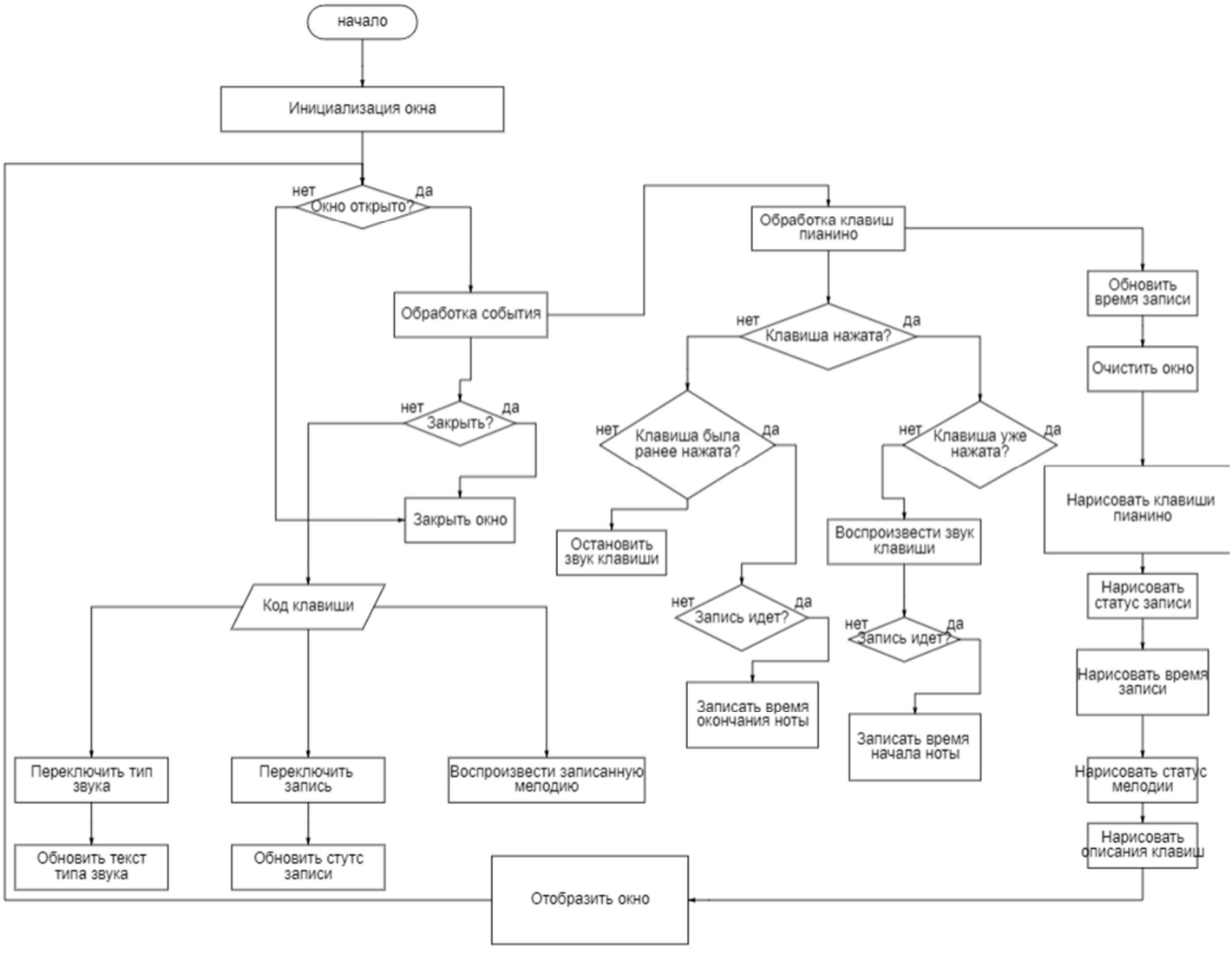

Рисунок 3.2 - Блок-схема метода run

Алгоритм метода toggleSoundType, который переключает тип звука клавиш между "Piano" и "Electronic", представлен на рисунке 3.3.

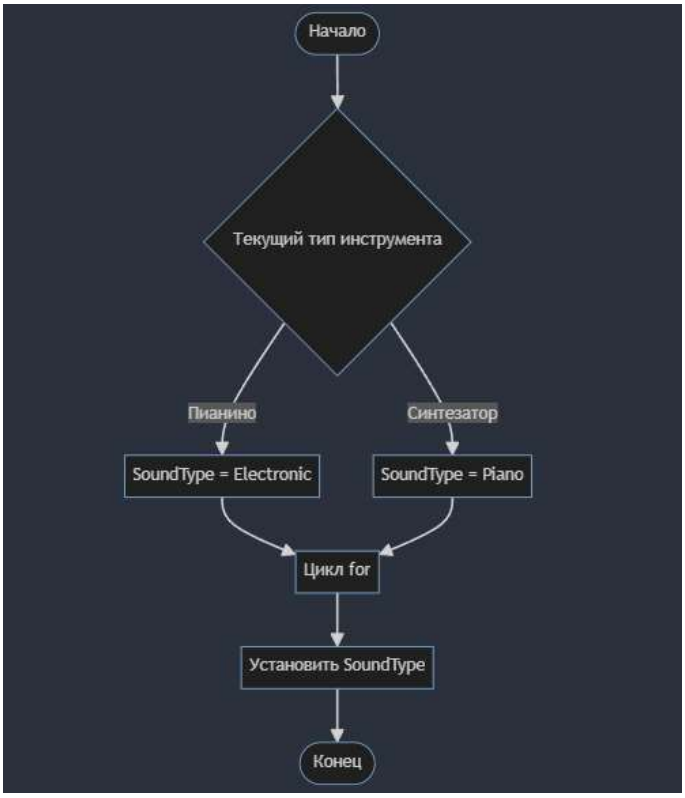

Рисунок 3.3- Блок-схема метода toggleSoundType

Алгоритм метода toggleRecording, представленный на рисунке 3.4, отвечает за управление записью мелодий.

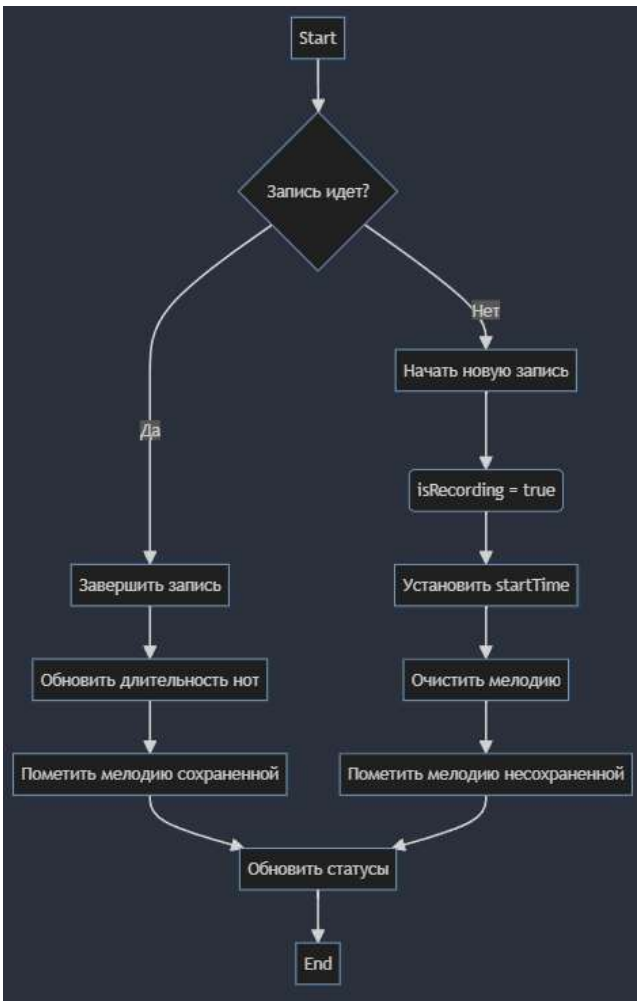

Рисунок 3.4 - Блок-схема метода toggleRecording

Оставшиеся методы содержат простые алгоритмы.

Метод loadPianoKeys загружает клавиши пианино с их параметрами.

Метод updateSoundTypeText обновляет текстовое поле с текущим типом звука.

Метод updateRecordingStatusText обновляет текстовое поле с текущим статусом записи.

Метод updateMelodyStatusText обновляет текстовое поле с текущим статусом сохранения мелодии.

Метод playRecordedMelody воспроизводит записанную мелодию. Он проходит по списку записанных нот и воспроизводит каждую ноту с соответствующей задержкой между ними. Если записанная мелодия пуста, метод просто возвращает без выполнения действий.

4 Результат работы

Во время написания кода проводилось постепенное тестирование по этапам разработки:

— смоук-тестирование – первичная проверка основных функций для выявления критических ошибок.

— регрессионное тестирование – проверка корректности работы системы после внесения изменений или исправлений.

— альфа-тестирование – тестирование со стороны разработчика перед окончанием работы.

Таким образом, в ходе выполнения курсовой работы были повторены основы объектно-ориентированного программирования на языке C++.

Написание программы способствовало закреплению теоретического материала на практике.

В результате разработки была создан синтезатор (рисунок 4.1 – 4.5).

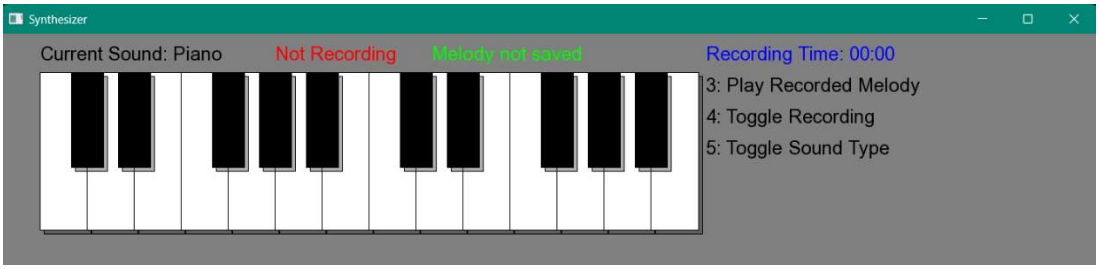

Рисунок 4.1 – Синтезатор, его внешний вид при запуске

На рисунке 4.2 были введена с клавиатуры команда 5:Toggle Sound Type, тип инструмента был изменен с «Piano» на «Electronic»

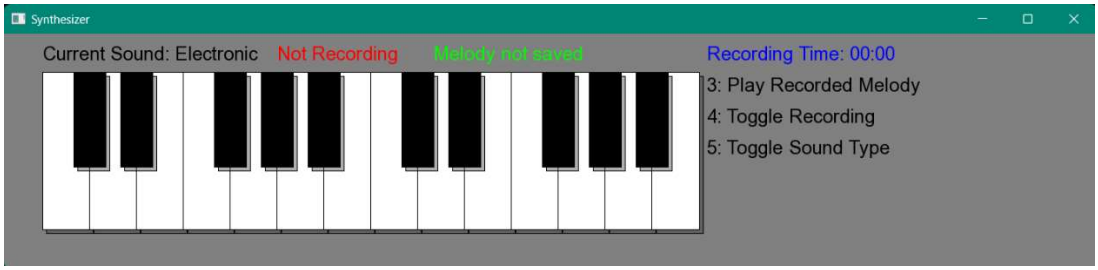

Рисунок 4.2 – Режим выбора инструмента

На рисунках 4.3– 4.4 продемонстрирован процесс записи мелодии.

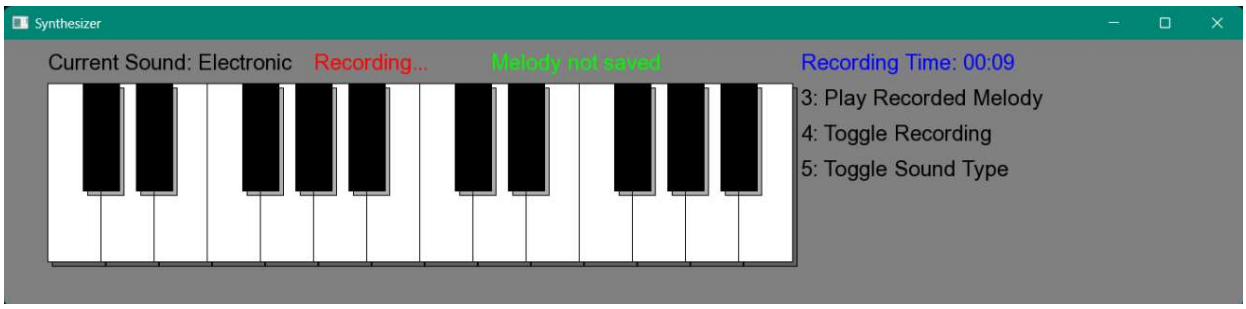

Рисунок 4.3 – Режим записи мелодии

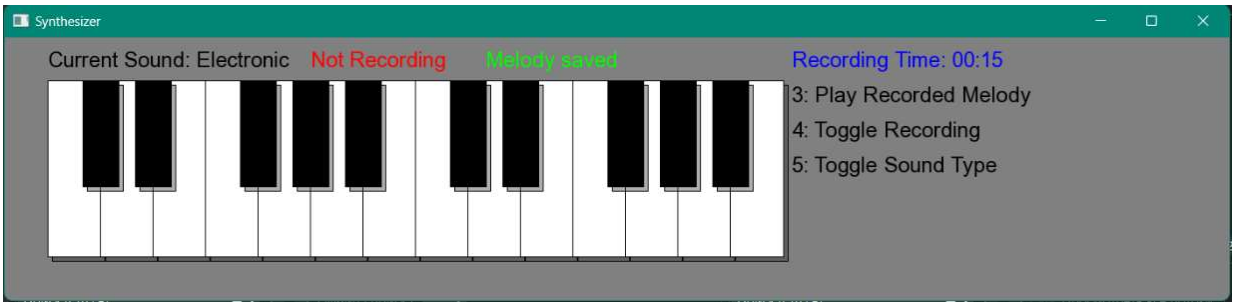

Рисунок 4.4 – Режим прослушивания мелодии

Заключение

В результате курсового проектирования были решены следующие задачи:

— созданы DFD и UML диаграммы проекта;

— реализованы функции: воспроизведение ноты по нажатию соответствующей кнопки на клавиатуре, выбор музыкального инструмента между пианино и электронным синтезатором, запись мелодии по нажатию соответствующей кнопки, воспроизведение записанной мелодии по нажатию соответствующей кнопки;

— создано приложение «Синтезатор».

Цель работы достигнута.

Список использованных источников

1. Microsoft [Электронный ресурс]: сайт компании Microsoft. URL: https://learn.microsoft.com/ru-ru/visualstudio/get-started/visual-studio-ide?view=vs-2022 (дата обращения: 29.05.2024).

2. VerITy [Электронный ресурс]: платформа с курсами программирования. URL: https://verity.by/news/yazyk-programmirovaniya-c-chast-1-istoriya-sozdaniya-plyusy-i-minusy (дата обращения: 29.05.2024).

Приложение А

Исходный код программы

```
#include <sfml audio.hpp="">
#include <sfml graphics.hpp="">
#include <iostream>
#include <vectorp>
#include <cmath>
#include <chrono>
#include <thread>

// Перечисление для типа звука
    enum class SoundType {
        Piano,
        Electronic
    };

    // Структура для представления ноты
    struct Note {
        std::string noteName;
        std::chrono::milliseconds startTime;
        std::chrono::milliseconds duration;
    };

    // Класс для представления клавиши пианино
    class PianoKey {
    public:
        PianoKey(float x, float y, float width, float height, const sf::Color& color, const std::string& note, sf::Keyboard::Key key, bool isWhite, SoundType soundType)
            : rect(sf::Vector2f(width, height)), soundBuffer(), sound(), noteName(note), keyCode(key), originalColor(color), isWhiteKey(isWhite), currentSoundType(soundType) {
            rect.setPosition(x, y);
            rect.setFillColor(color);
            rect.setOutlineThickness(1);
            rect.setOutlineColor(sf::Color::Black);
            generateSound(noteName, currentSoundType);
        }

        // Метод для отрисовки клавиши
        void draw(sf::RenderWindow& window) {
            // Добавляем тень
            sf::RectangleShape shadow = rect;
            shadow.setFillColor(sf::Color(50, 50, 50, 100)); // Полупрозрачная тень
            shadow.move(5, 5); // Смещение тени
            window.draw(shadow);

```

```
            // Добавляем градиент
            sf::RectangleShape gradient = rect;
            sf::VertexArray gradientArray(sf::Quads, 4);
            sf::Color lightColor = originalColor;
            sf::Color darkColor = originalColor - sf::Color(50, 50, 50); // затемнение цвета
            gradientArray[0].position = gradient.getPosition();
            gradientArray[0].color = lightColor;
            gradientArray[1].position = sf::Vector2f(gradient.getPosition().x + gradient.getSize().x, gradient.getPosition().y);
            gradientArray[1].color = lightColor;
            gradientArray[2].position = sf::Vector2f(gradient.getPosition().x + gradient.getSize().x, gradient.getPosition().y + gradient.getSize().y);
            gradientArray[2].color = darkColor;
            gradientArray[3].position = sf::Vector2f(gradient.getPosition().x, gradient.getPosition().y + gradient.getSize().y);
            gradientArray[3].color = darkColor;

            window.draw(gradientArray);

            // Рисуем клавишу поверх градиента
            window.draw(rect);
        }

        // Метод для проверки нажатия клавиши
        bool isKeyPressed() {
            return sf::Keyboard::isKeyPressed(keyCode);
        }

        // Метод для воспроизведения звука
        void play() {
            if (!isPressed) {
                sound.play();
                rect.setFillColor(sf::Color(192, 192, 192));
                isPressed = true;
            }
        }

        // Метод для однократного воспроизведения звука
        void playOnce() {
            sound.play();
        }

        // Метод для остановки воспроизведения звука
        void stop() {
            sound.stop();
            resetColor();
            isPressed = false;
```

```
86.         }
87.
88.         // Метод для сброса цвета клавиши
89.         void resetColor() {
90.             rect.setFillColor(originalColor);
91.         }
92.
93.         // Геттер для проверки нажатия клавиши
94.         bool getIsPressed() const {
95.             return isPressed;
96.         }
97.
98.         // Сеттер для установки состояния нажатия клавиши
99.         void setIsPressed(bool pressed) {
100.            isPressed = pressed;
101.        }
102.
103.        // Сеттер для установки типа звука клавиши
104.        void setSoundType(SoundType soundType) {
105.            currentSoundType = soundType;
106.            generateSound(noteName, currentSoundType);
107.        }
108.
109.        // Геттер для получения кода клавиши
110.        sf::Keyboard::Key getKeyCode() const {
111.            return keyCode;
112.        }
113.
114.        // Геттер для получения названия ноты
115.        const std::string& getNoteName() const {
116.            return noteName;
117.        }
118.
119.    private:
120.        bool isPressed = false;
121.        sf::RectangleShape rect;
122.        sf::SoundBuffer soundBuffer;
123.        sf::Sound sound;
124.        std::string noteName;
125.        sf::Keyboard::Key keyCode;
126.        sf::Color originalColor;
127.        bool isWhiteKey;
128.        SoundType currentSoundType;
129.
130.        // Метод для генерации звука
131.        void generateSound(const std::string& note, SoundType
    soundType) {
132.            const unsigned sampleRate = 44100;
133.            const unsigned amplitude = 30000;
134.            const double twoPi = 6.28318;
135.            double frequency = getFrequency(note);
```

```
136.             double increment = twoPi * frequency / sampleRate;
137.
138.             std::vector<sf::int16> samples;
139.             samples.reserve(sampleRate);
140.
141.             if (soundType == SoundType::Piano) {
142.                 for (unsigned i = 0; i < sampleRate; ++i) {
143.                     samples.push_back(static_cast<sf::int16>(amp
    litude * sin(i * increment)));
144.                 }
145.             }
146.             else if (soundType == SoundType::Electronic) {
147.                 for (unsigned i = 0; i < sampleRate; ++i) {
148.                     samples.push_back(static_cast<sf::int16>(amp
    litude * (sin(i * increment) + 0.5 * sin(i * increment * 2.0))));
149.                 }
150.             }
151.
152.             soundBuffer.loadFromSamples(&samples[0],
    samples.size(), 1, sampleRate);
153.             sound.setBuffer(soundBuffer);
154.             sound.setLoop(false);
155.         }
156.
157.         // Метод для получения частоты ноты
158.         double getFrequency(const std::string& note) {
159.             if (note == "C3") return 130.81;
160.             if (note == "C#3") return 138.59;
161.             if (note == "D3") return 146.83;
162.             if (note == "D#3") return 155.56;
163.             if (note == "E3") return 164.81;
164.             if (note == "F3") return 174.61;
165.             if (note == "F#3") return 185.00;
166.             if (note == "G3") return 196.00;
167.             if (note == "G#3") return 207.65;
168.             if (note == "A3") return 220.00;
169.             if (note == "A#3") return 233.08;
170.             if (note == "B3") return 246.94;
171.             if (note == "C4") return 261.63;
172.             if (note == "C#4") return 277.18;
173.             if (note == "D4") return 293.66;
174.             if (note == "D#4") return 311.13;
175.             if (note == "E4") return 329.63;
176.             if (note == "F4") return 349.23;
177.             if (note == "F#4") return 369.99;
178.             if (note == "G4") return 392.00;
179.             if (note == "G#4") return 415.30;
180.             if (note == "A4") return 440.00;
181.             if (note == "A#4") return 466.16;
182.             if (note == "B4") return 493.88;
183.             if (note == "C5") return 523.25;
```

```
184.             return 440.00;
185.         }
186.     };
187.
188.     // Класс для синтезатора
189.     class Synthesizer {
190.     public:
191.         Synthesizer() : currentSoundType(SoundType::Piano),
   isRecording(false), melodySaved(false) {
192.             loadPianoKeys();
193.             if (!font.loadFromFile("ArialRegular.ttf")) {
194.                 std::cerr << "Error loading font\n";
195.             }
196.             soundTypeText.setFont(font);
197.             soundTypeText.setCharacterSize(24);
198.             soundTypeText.setFillColor(sf::Color::Black);
199.             soundTypeText.setPosition(50, 10);
200.             updateSoundTypeText();
201.
202.             recordingStatusText.setFont(font);
203.             recordingStatusText.setCharacterSize(24);
204.             recordingStatusText.setFillColor(sf::Color::Red);
205.             recordingStatusText.setPosition(350, 10);
206.             updateRecordingStatusText();
207.
208.             melodyStatusText.setFont(font);
209.             melodyStatusText.setCharacterSize(24);
210.             melodyStatusText.setFillColor(sf::Color::Green);
211.             melodyStatusText.setPosition(550, 10);
212.             updateMelodyStatusText();
213.
214.             // Add descriptions for keys 3, 4, and 5
215.             key3Text.setFont(font);
216.             key3Text.setCharacterSize(24);
217.             key3Text.setFillColor(sf::Color::Black);
218.             key3Text.setPosition(900, 50);
219.             key3Text.setString("3: Play Recorded Melody");
220.
221.             key4Text.setFont(font);
222.             key4Text.setCharacterSize(24);
223.             key4Text.setFillColor(sf::Color::Black);
224.             key4Text.setPosition(900, 90);
225.             key4Text.setString("4: Toggle Recording");
226.
227.             key5Text.setFont(font);
228.             key5Text.setCharacterSize(24);
229.             key5Text.setFillColor(sf::Color::Black);
230.             key5Text.setPosition(900, 130);
231.             key5Text.setString("5: Toggle Sound Type");
232.
233.             // Time recording text
```

```
234.             recordingTimeText.setFont(font);
235.             recordingTimeText.setCharacterSize(24);
236.             recordingTimeText.setFillColor(sf::Color::Blue);
237.             recordingTimeText.setPosition(900, 10);
238.             recordingTimeText.setString("Recording Time:
   00:00");
239.         }
240.
241.         // Метод для запуска синтезатора
242.         void run() {
243.             sf::RenderWindow window(sf::VideoMode(1400, 300),
   "Synthesizer");
244.
245.             while (window.isOpen()) {
246.                 sf::Event event;
247.                 while (window.pollEvent(event)) {
248.                     if (event.type == sf::Event::Closed)
249.                         window.close();
250.
251.                     if (event.type == sf::Event::KeyPressed) {
252.                         if (event.key.code ==
   sf::Keyboard::Num5) {
253.                             toggleSoundType();
254.                             updateSoundTypeText();
255.                         }
256.                         else if (event.key.code ==
   sf::Keyboard::Num4) {
257.                             toggleRecording();
258.                             updateRecordingStatusText();
259.                         }
260.                         else if (event.key.code ==
   sf::Keyboard::Num3) {
261.                             playRecordedMelody();
262.                         }
263.                     }
264.                 }
265.
266.                 for (auto& key : pianoKeys) {
267.                     if (key.isKeyPressed()) {
268.                         if (!key.getIsPressed()) {
269.                             key.play();
270.                             if (isRecording) {
271.                                 recordedMelody.push_back({
   key.getNoteName(),
   std::chrono::duration_cast<std::chrono::milliseconds>(std::chrono
   ::steady_clock::now() - recordingStartTime),
   std::chrono::milliseconds(0) });
272.                             }
273.                         }
274.                     }
275.                     else {
```

```
276.                         if (key.getIsPressed()) {
277.                             if (isRecording) {
278.                                 auto it =
    std::find_if(recordedMelody.rbegin(), recordedMelody.rend(),
    [&key](const Note& note) {
279.                                     return note.noteName ==
    key.getNoteName() && note.duration.count() == 0;
280.                                     });
281.                                 if (it != recordedMelody.rend())
    {
282.                                     it->duration =
    std::chrono::duration_cast<std::chrono::milliseconds>(std::chrono
    ::steady_clock::now() - recordingStartTime) - it->startTime;
283.                                 }
284.                             }
285.                             key.stop();
286.                         }
287.                     }
288.                 }
289.
290.                 if (isRecording) {
291.                     auto elapsed =
    std::chrono::duration_cast<std::chrono::seconds>(std::chrono::ste
    ady_clock::now() - recordingStartTime);
292.                     recordingTimeText.setString("Recording Time:
    " + formatTime(elapsed));
293.                 }
294.
295.                 sf::Color grey(128, 128, 128);
296.                 window.clear(grey);
297.
298.                 for (auto& key : pianoKeys) {
299.                     key.draw(window);
300.                 }
301.
302.                 window.draw(soundTypeText);
303.                 window.draw(recordingStatusText);
304.                 window.draw(recordingTimeText);
305.                 window.draw(melodyStatusText);
306.
307.                 // Draw key descriptions
308.                 window.draw(key3Text);
309.                 window.draw(key4Text);
310.                 window.draw(key5Text);
311.
312.                 window.display();
313.             }
314.         }
315.
316.     private:
317.         std::vector<pianokey> pianoKeys;
```

```
318.         std::vector<note> recordedMelody;
319.         SoundType currentSoundType;
320.         sf::Font font;
321.         sf::Text soundTypeText;
322.         sf::Text recordingStatusText;
323.         sf::Text melodyStatusText;
324.         sf::Text key3Text;
325.         sf::Text key4Text;
326.         sf::Text key5Text;
327.         sf::Text recordingTimeText;
328.         bool isRecording;
329.         bool melodySaved;
330.         std::chrono::steady_clock::time_point
  recordingStartTime;
331.
332.         // Метод для загрузки клавиш
333.         void loadPianoKeys() {
334.             pianoKeys.reserve(26);
335.             const int numWhiteKeys = 14;
336.             const int numBlackKeys = 10;
337.             const float whiteKeyWidth = 60;
338.             const float whiteKeyHeight = 200;
339.             const float blackKeyWidth = 40;
340.             const float blackKeyHeight = 120;
341.             const float startX = 50;
342.             const float startY = 50;
343.
344.             std::vector<std::string> whiteNotes = { "C3", "D3",
  "E3", "F3", "G3", "A3", "B3", "C4", "D4", "E4", "F4", "G4", "A4",
  "B4" };
345.             std::vector<sf::keyboard::key> whiteKeys = {
346.                 sf::Keyboard::Z, sf::Keyboard::X,
  sf::Keyboard::C,
347.                 sf::Keyboard::V, sf::Keyboard::B,
  sf::Keyboard::N,
348.                 sf::Keyboard::M, sf::Keyboard::A,
  sf::Keyboard::S,
349.                 sf::Keyboard::D, sf::Keyboard::F,
  sf::Keyboard::G,
350.                 sf::Keyboard::H, sf::Keyboard::J
351.             };
352.             for (int i = 0; i < numWhiteKeys; ++i) {
353.                 pianoKeys.emplace_back(startX + i *
  whiteKeyWidth, startY, whiteKeyWidth, whiteKeyHeight,
  sf::Color::White, whiteNotes[i], whiteKeys[i], true,
  currentSoundType);
354.             }
355.
356.             std::vector<std::string> blackNotes = { "C#3",
  "D#3", "F#3", "G#3", "A#3", "C#4", "D#4", "F#4", "G#4", "A#4" };
357.             std::vector<sf::keyboard::key> blackKeys = {
```

```
358.                 sf::Keyboard::Q, sf::Keyboard::W,
   sf::Keyboard::E,
359.                 sf::Keyboard::R, sf::Keyboard::T,
   sf::Keyboard::Y,
360.                 sf::Keyboard::U, sf::Keyboard::I,
   sf::Keyboard::O,
361.                 sf::Keyboard::P
362.             };
363.             std::vector<int> blackKeyPositions = { 0, 1, 3, 4,
   5, 7, 8, 10, 11, 12 };
364.             for (int i = 0; i < numBlackKeys; ++i) {
365.                 pianoKeys.emplace_back(startX +
   blackKeyPositions[i] * whiteKeyWidth + whiteKeyWidth -
   blackKeyWidth / 2, startY, blackKeyWidth, blackKeyHeight,
   sf::Color::Black, blackNotes[i], blackKeys[i], false,
   currentSoundType);
366.             }
367.         }
368.
369.         // Метод для смены типа звука
370.         void toggleSoundType() {
371.             if (currentSoundType == SoundType::Piano) {
372.                 currentSoundType = SoundType::Electronic;
373.             }
374.             else {
375.                 currentSoundType = SoundType::Piano;
376.             }
377.             for (auto& key : pianoKeys) {
378.                 key.setSoundType(currentSoundType);
379.             }
380.         }
381.
382.         // Метод для обновления текста с типом звука
383.         void updateSoundTypeText() {
384.             soundTypeText.setString("Current Sound: " +
   std::string(currentSoundType == SoundType::Piano ? "Piano" :
   "Electronic"));
385.         }
386.
387.         // Метод для смены состояния записи
388.         void toggleRecording() {
389.             if (isRecording) {
390.                 // Завершаем запись
391.                 isRecording = false;
392.
393.                 // Обновляем длительность всех нот
394.                 auto now = std::chrono::steady_clock::now();
395.                 for (auto& note : recordedMelody) {
396.                     if (note.duration.count() == 0) {
```

```
397.                         note.duration =
    std::chrono::duration_cast<std::chrono::milliseconds>(now -
    recordingStartTime) - note.startTime;
398.                     }
399.                 }
400.
401.                 // Помечаем мелодию как сохраненную
402.                 melodySaved = true;
403.             }
404.             else {
405.                 // Начинаем новую запись
406.                 isRecording = true;
407.                 recordingStartTime =
    std::chrono::steady_clock::now();
408.                 recordedMelody.clear();
409.                 melodySaved = false; // Помечаем мелодию как
    несохранённую при начале новой записи
410.             }
411.             updateRecordingStatusText();
412.             updateMelodyStatusText(); // Добавляем обновление
    статуса мелодии
413.         }
414.
415.         // Метод для обновления текста статуса записи
416.         void updateRecordingStatusText() {
417.             recordingStatusText.setString(isRecording ?
    "Recording..." : "Not Recording");
418.         }
419.
420.         // Метод для обновления текста статуса мелодии
421.         void updateMelodyStatusText() {
422.             melodyStatusText.setString(melodySaved ? "Melody
    saved" : "Melody not saved");
423.         }
424.
425.         // Метод для воспроизведения записанной мелодии
426.         void playRecordedMelody() {
427.             if (recordedMelody.empty()) return;
428.
429.             auto startTime = std::chrono::steady_clock::now();
430.
431.             for (const auto& note : recordedMelody) {
432.                 auto now = std::chrono::steady_clock::now();
433.                 auto waitDuration = note.startTime -
    std::chrono::duration_cast<std::chrono::milliseconds>(now -
    startTime);
434.                 if (waitDuration.count() > 0) {
435.                     std::this_thread::sleep_for(waitDuration);
436.                 }
437.                 for (auto& key : pianoKeys) {
438.                     if (key.getNoteName() == note.noteName) {
```

```
439.                         key.playOnce();
440.                         std::this_thread::sleep_for(note.duratio
    n);
441.                         key.stop();
442.                         break;
443.                     }
444.                 }
445.             }
446.
447.             melodySaved = true;
448.             updateMelodyStatusText();
449.         }
450.
451.         // Метод для форматирования времени записи
452.         std::string formatTime(std::chrono::seconds duration) {
453.             auto minutes =
    std::chrono::duration_cast<std::chrono::minutes>(duration);
454.             auto seconds = duration - minutes;
455.             return (minutes.count() < 10 ? "0" : "") +
    std::to_string(minutes.count()) + ":" + (seconds.count() < 10 ?
    "0" : "") + std::to_string(seconds.count());
456.         }
457.     };
458.
459.     // Главная функция
460.     int main() {
461.         Synthesizer synth;
462.         synth.run();
463.         return 0;
```