**Введение**

Наш проект называется "minisynth-qt" и предназначен для генерации звука на основе нажатий клавиш имитирующих фортепианную клавиатуру

**Шаг 1: Настройка проекта (CMakeLists.txt)**

CMakeLists.txt отвечает за конфигурацию сборки проекта. Его основные части:

1. **cmake\_minimum\_required(VERSION 3.5)**: Указывает минимальную требуемую версию CMake для сборки
2. **project(minisynth-qt VERSION 1.0 LANGUAGES CXX)**: Определяет имя проекта (minisynth-qt), его версию (1.0) и используемый язык программирования (C++)
3. **set(CMAKE\_INCLUDE\_CURRENT\_DIR ON)** и тп команды: Включают автоматическое добавление текущей директории в пути поиска заголовочных файлов, а также автоматическую обработку файлов пользовательского интерфейса (.ui), файлов ресурсов (.qrc) и файлов, содержащих мета-объектную информацию (.moc)
4. **set(CMAKE\_CXX\_STANDARD 11)** и **set(CMAKE\_CXX\_STANDARD\_REQUIRED ON)**: Устанавливают стандарт C++11 как обязательный для компиляции
5. **find\_package(QT NAMES Qt5 Qt6)**: Поиск установленного Qt5 или Qt6
6. **find\_package(Qt${QT\_VERSION\_MAJOR} COMPONENTS Widgets Multimedia REQUIRED)**: Поиск необходимых компонентов Qt: Widgets (для графического интерфейса) и Multimedia (для работы со звуком)
7. **set(PROJECT\_SOURCES ...)**: Перечисление всех файлов исходного кода, входящих в проект
8. **qt\_add\_executable** или **add\_executable**: Создание исполняемого файла minisynth-qt из указанных исходных файлов
9. **target\_link\_libraries**: Прилинковывает к исполняемому файлу необходимые библиотеки Qt (Widgets и Multimedia)
10. **if (CMAKE\_SYSTEM\_NAME MATCHES "Emscripten")**: Условная компиляция для сборки под Emscripten (WebAssembly)
11. **qt\_finalize\_target(minisynth-qt)**: Завершает настройку цели сборки (только для Qt6)

**Шаг 2: Заголовочный файл главного окна (mainwindow.h)**

Файл mainwindow.h объявляет класс MainWindow, отвечающий за главное окно приложения

1. **#ifndef MAINWINDOW\_H**, **#define MAINWINDOW\_H**, **#endif // MAINWINDOW\_H**: Стандартные директивы препроцессора для предотвращения повторного включения заголовочного файла
2. **#include ...**: Включение необходимых заголовочных файлов Qt и пользовательского заголовочного файла tonesynth.h
3. **QT\_BEGIN\_NAMESPACE**, **namespace Ui { class MainWindow; }**, **QT\_END\_NAMESPACE**: Объявление класса MainWindow в пространстве имен Ui, сгенерированного из файла пользовательского интерфейса mainwindow.ui
4. **class MainWindow : public QMainWindow**: Объявление класса MainWindow, наследующегося от QMainWindow
5. **Q\_OBJECT**: Макрос, необходимый для использования механизма сигналов и слотов Qt
6. **public:**:
   * **MainWindow(QWidget \*parent = nullptr);**: Конструктор класса
   * **~MainWindow();**: Деструктор класса
7. **signals:**: Объявление сигналов, которые может испускать MainWindow:
   * **void underrunDetected();**: Сигнал о нехватке данных в аудио-буфере
   * **void stallDetected();**: Сигнал о простое аудиовыхода
8. **private:**:
   * **void initializeWindow();**: Метод для инициализации окна
   * **void initializeAudio();**: Метод для инициализации аудио
9. **private slots(**Объявление приватных слотов):
   * **void deviceChanged(int index);**: Слот, вызываемый при изменении аудиоустройства
   * **void volumeChanged(int value);**: Слот, вызываемый при изменении громкости
   * **void bufferChanged(int value);**: Слот, вызываемый при изменении размера буфера
   * **void octaveChanged(int value);**: Слот, вызываемый при изменении октавы
   * **void underrunMessage();**: Слот для отображения сообщения о нехватке данных
   * **void stallMessage();**: Слот для отображения сообщения о простое
   * **void keyPressEvent(QKeyEvent \*event) override;**: Переопределенный метод обработки нажатия клавиш
   * **void keyReleaseEvent(QKeyEvent \*event) override;**: Переопределенный метод обработки отпускания клавиш
10. **private:**: Приватные члены класса:
    * **Ui::MainWindow \*m\_ui;**: Указатель на объект пользовательского интерфейса
    * **QAudioFormat m\_format;**: Формат аудиоданных
    * **int m\_bufferTime;**: Время буферизации в миллисекундах
    * **bool m\_running;**: Флаг, указывающий, работает ли аудиовыход
    * **QScopedPointer<ToneSynthesizer> m\_synth;**: Умный указатель на объект генератора тонов
    * **QScopedPointer<QAudioOutput> m\_audioOutput;** (или **QAudioSink** для Qt6): Умный указатель на объект аудиовыхода
    * **QTimer m\_stallDetector;**: Таймер для обнаружения простоя

**Шаг 3: Заголовочный файл генератора тонов (tonesynth.h)**

tonesynth.h объявляет класс ToneSynthesizer, отвечающий за генерацию звука

1. **#ifndef TONESYNTH\_H**, **#define TONESYNTH\_H**, **#endif // TONESYNTH\_H**: Стандартные директивы препроцессора для предотвращения повторного включения
2. **#include ...**: Включение необходимых заголовочных файлов
3. **class ToneSynthesizer : public QIODevice**: Объявление класса ToneSynthesizer, наследующегося от QIODevice для обеспечения возможности вывода аудиоданных через QAudioOutput (или QAudioSink)
4. **Q\_OBJECT**: Макрос для использования сигналов и слотов
5. **public:**:
   * **enum class EnvelopeState : int { silentState, attackState, sustainState, releaseState };**: Перечисление, описывающее состояния огибающей громкости (тишина, атака, поддержка, затухание)
   * **Q\_ENUM(EnvelopeState)**: Регистрация перечисления в системе мета-объектов Qt
   * **ToneSynthesizer(const QAudioFormat &format);**: Конструктор класса, принимающий формат аудиоданных
   * **qint64 readData(char \*data, qint64 maxlen) override;**: Переопределенный метод чтения данных для генерации звука
   * **qint64 writeData(const char \*data, qint64 len) override;**: Переопределенный метод записи данных (не используется, возвращает 0)
   * **qint64 size() const override;**: Переопределенный метод, возвращающий размер данных (возвращает максимальное значение qint64)
   * **qint64 bytesAvailable() const override;**: Переопределенный метод, возвращающий количество доступных байтов (возвращает максимальное значение qint64)
   * **void setOctave(int newOctave);**: Метод установки октавы
   * **qint64 lastBufferSize() const;**: Метод, возвращающий размер последнего сгенерированного буфера
   * **void resetLastBufferSize();**: Метод сброса размера последнего буфера
6. **public slots:**: Публичные слоты:
   * **void start();**: Слот для запуска генерации
   * **void stop();**: Слот для остановки генерации
   * **void noteOn(const QString &note);**: Слот, вызываемый при нажатии клавиши
   * **void noteOff();**: Слот, вызываемый при отпускании клавиши
7. **private:**: Приватные члены класса:
   * **QAudioFormat m\_format;**: Формат аудиоданных
   * **int m\_octave;**: Текущая октава
   * **const QMap<QString, qreal> m\_freq...**: Словарь, сопоставляющий названия нот с их частотами в герцах
   * **qreal m\_angleDelta;**: Приращение фазы для генерации синусоидальной волны
   * **qreal m\_currentAngle;**: Текущая фаза
   * **qint64 m\_lastBufferSize;**: Размер последнего сгенерированного буфера
   * **EnvelopeState m\_envelState;**: Текущее состояние огибающей
   * **qreal m\_envelVolume;**: Текущая громкость огибающей
   * **qreal m\_envelDelta;**: Приращение громкости огибающей
   * **quint64 m\_envelCount;**: Счетчик сэмплов для состояний атаки и затухания
   * **quint64 m\_attackTime;**: Время атаки в сэмплах
   * **quint64 m\_releaseTime;**: Время затухания в сэмплах

**Шаг 4: Исходный файл главного окна (mainwindow.cpp)**

mainwindow.cpp содержит реализацию методов класса MainWindow

1. **#include ...**: Включение необходимых заголовочных файлов
2. **MainWindow::MainWindow(QWidget \*parent) : ...**: Конструктор класса
   * **m\_ui(new Ui::MainWindow)**: Создание объекта пользовательского интерфейса
   * **m\_bufferTime(...)**: Инициализация времени буферизации
   * **m\_running(false)**: Инициализация флага работы аудио
   * **m\_ui->setupUi(this)**: Настройка пользовательского интерфейса
   * **initializeWindow();**: Вызов метода инициализации окна
   * **initializeAudio();**: Вызов метода инициализации аудио
3. **MainWindow::~MainWindow()**: Деструктор класса
   * Остановка m\_stallDetector
   * Остановка m\_audioOutput
   * Остановка m\_synth, если он существует
   * Удаление m\_ui
4. **void MainWindow::initializeWindow()**: Инициализация окна
   * Настройка m\_format: частота дискретизации (44100 Гц), количество каналов (1), размер сэмпла (32 бита), кодек ("audio/pcm"), порядок байтов (LittleEndian), тип сэмпла (Float)
   * Добавление доступных аудиоустройств в m\_ui->deviceBox (с проверкой поддержки формата)
   * Создание объекта ToneSynthesizer и привязка его к m\_synth
   * Установка значения m\_ui->bufferSpin равным m\_bufferTime
   * Подключение сигналов и слотов для m\_ui->deviceBox, m\_ui->volumeSlider, m\_ui->bufferSpin, m\_ui->octaveSpin, а также для кнопок, сгенерированных из UI
   * Подключение m\_stallDetector к лямбда-функции, которая проверяет, работает ли аудио и не равен ли размер буфера нулю, если нет, испускает сигнал stallDetected и сбрасывает размер буфера
5. **void MainWindow::initializeAudio()**: Инициализация аудио
   * Получение информации об аудиоустройстве из m\_ui->deviceBox
   * Проверка поддержки формата аудиоустройством, если не поддерживается, выдача предупреждения
   * Расчет размера буфера в байтах на основе m\_bufferTime
   * Запуск m\_synth
   * Создание объекта QAudioOutput (или QAudioSink) и подключение к нему m\_synth
   * Подключение сигнала stateChanged m\_audioOutput к лямбда-функции, которая проверяет состояние аудиовыхода и в случае QAudio::UnderrunError испускает сигнал underrunDetected
   * Установка размера буфера m\_audioOutput
   * Запуск m\_audioOutput, передав ему m\_synth
   * Установка громкости и октавы
   * Запуск m\_stallDetector (только если не WASM)
6. **void MainWindow::deviceChanged(int index)**: Слот изменения аудиоустройства
   * Остановка m\_stallDetector и m\_audioOutput
   * Остановка m\_synth
   * Повторная инициализация аудио
7. **void MainWindow::volumeChanged(int value)**: Слот изменения громкости
   * Преобразование логарифмической шкалы громкости в линейную
   * Установка громкости m\_audioOutput
8. **void MainWindow::bufferChanged(int value)**: Слот изменения размера буфера
   * Если новое значение отличается от m\_bufferTime, обновляет m\_bufferTime и вызывает deviceChanged для переинициализации аудио
9. **void MainWindow::octaveChanged(int value)**: Слот изменения октавы
   * Вызов m\_synth->setOctave(value) для установки новой октавы
10. **void MainWindow::underrunMessage()**: Слот отображения сообщения о нехватке данных
    * Отображение QMessageBox с предупреждением
11. **void MainWindow::stallMessage()**: Слот отображения сообщения о простое
    * Отображение QMessageBox с критической ошибкой
12. **void MainWindow::keyPressEvent(QKeyEvent \*event)**: Обработчик нажатия клавиш
    * Определяет, какая клавиша была нажата, и сопоставляет её с нотой.
    * Если нота найдена, вызывает m\_synth->noteOn(note)
13. **void MainWindow::keyReleaseEvent(QKeyEvent \*event)**: Обработчик отпускания клавиш
    * Вызывает m\_synth->noteOff() при отпускании любой клавиши, связанной с нотой

**Шаг 5: Исходный файл генератора тонов (tonesynth.cpp)**

tonesynth.cpp содержит реализацию методов класса ToneSynthesizer

1. **#include ...**: Включение необходимых заголовочных файлов
2. **ToneSynthesizer::ToneSynthesizer(const QAudioFormat &format) : ...**: Конструктор класса
   * Инициализация членов класса: m\_octave (3), m\_envelState (silentState), m\_envelVolume (0.0)
   * Если формат валиден, сохраняет его в m\_format, вычисляет время атаки и затухания (m\_attackTime, m\_releaseTime) и приращение громкости (m\_envelDelta)
3. **void ToneSynthesizer::start()**: Запуск генерации
   * Открытие QIODevice в режиме ReadOnly и Unbuffered
4. **void ToneSynthesizer::stop()**: Остановка генерации
   * Закрытие QIODevice
5. **void ToneSynthesizer::noteOn(const QString &note)**: Обработчик нажатия ноты
   * Проверяет, есть ли нота в словаре m\_freq
   * Если есть, вычисляет частоту ноты с учетом октавы, приращение фазы (m\_angleDelta), устанавливает текущую фазу (m\_currentAngle) в 0
   * Устанавливает состояние огибающей в attackState, сбрасывает счетчик m\_envelCount и устанавливает громкость огибающей в 0
6. **void ToneSynthesizer::noteOff()**: Обработчик отпускания ноты
   * Устанавливает состояние огибающей в releaseState и сбрасывает счетчик m\_envelCount
7. **qint64 ToneSynthesizer::lastBufferSize() const**: Возвращает размер последнего сгенерированного буфера
8. **void ToneSynthesizer::resetLastBufferSize()**: Сбрасывает размер последнего буфера
9. **void ToneSynthesizer::setOctave(int newOctave)**: Устанавливает октаву
10. **qint64 ToneSynthesizer::readData(char \*data, qint64 maxlen)**: Генерация аудиоданных
    * Определяет размер сэмпла в байтах (channelBytes)
    * Вычисляет длину данных для обработки (length)
    * Получает указатель на буфер данных (ptr)
    * В цикле while (length > 0) генерирует сэмплы:
      + Инициализирует currentSample нулем
      + В зависимости от состояния огибающей (m\_envelState) изменяет громкость m\_envelVolume
      + Если состояние не silentState, вычисляет currentSample как синус от текущей фазы, умноженный на громкость
      + Увеличивает текущую фазу на m\_angleDelta
      + Записывает currentSample в буфер
      + Сдвигает указатель ptr и уменьшает length
    * Сохраняет размер сгенерированного буфера в m\_lastBufferSize
    * Возвращает размер сгенерированного буфера
11. **qint64 ToneSynthesizer::writeData(const char \*data, qint64 len)**: Запись данных (не используется, возвращает 0)
12. **qint64 ToneSynthesizer::size() const**: Возвращает максимальный размер данных
13. **qint64 ToneSynthesizer::bytesAvailable() const**: Возвращает максимальное количество доступных байт

**Шаг 6: Исполняемый файл (main.cpp)**

Файл main.cpp содержит точку входа в программу

1. **#include ...**: Включение необходимых заголовочных файлов
2. **int main(int argc, char \*argv[])**: Главная функция
   * **QApplication a(argc, argv);**: Создание объекта приложения Qt
   * **MainWindow w;**: Создание объекта главного окна
   * **w.show();**: Отображение главного окна
   * **return a.exec();**: Запуск основного цикла обработки событий Qt