Министерство образования Республики Беларусь
Учреждение образования «Белорусский государственный университет информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра электронных вычислительных машин

Дисциплина: Программирование на языках высокого уровня

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
к курсовому проекту
на тему

ВИДЕОПЛЕЕР

БГУИР КП 1-40 02 01 218 ПЗ

Студент: гр. 250502 Гончаренко Д.Г.

Руководитель: Богдан Е. В.

МИНСК 2023

Учреждение образования
«Белорусский государственный университет информатики
и радиоэлектроники»

Факультет компьютерных систем и сетей

УТВЕРЖДАЮ
Заведующий кафедрой
_____
(подпись)
_____2023 г.

ЗАДАНИЕ
по курсовому проектированию

Студенту *Гончаренко Даниилу Геннадьевичу*
Тема проекта *Видеоплеер*
2. Срок сдачи студентом законченного проекта *15 декабря 2023 г.*
3. Исходные данные к проекту *Язык программирования – C++, среда разработки – Qt-Creator*
4. Содержание расчетно-пояснительной записки (перечень вопросов, которые подлежат разработке)
1. Лист задания.
2. Введение.
3. Обзор литературы.
4. Функциональное проектирование.
4.1. Структура входных и выходных данных.
4.2. Разработка диаграммы классов.
4.3. Описание классов.
5. Разработка программных модулей.
5.1. Разработка схем алгоритмов (два наиболее важных метода).
5.2. Разработка алгоритмов (описание алгоритмов по шагам, для двух методов).
6. Результаты работы.
7. Заключение
8. Литература
9. Приложения

5. Перечень графического материала (с точным обозначением обязательных чертежей и графиков)
1. Диаграмма классов. _____
2. Схема алгоритма метода next()
*3. Схема алгоритма метода previous()*_____
6. Консультант по проекту (с обозначением разделов проекта) Богдан Е. В.

7. Дата выдачи задания *15.09.2023г.*

8. Календарный график работы над проектом на весь период проектирования (с обозначением сроков выполнения и трудоемкости отдельных этапов):

1. Выбор задания. Разработка содержания пояснительной записки. Перечень графического материала – 15 %;

разделы 2, 3 – 10 %;

разделы 4 к –20 %;

разделы 5 к – 35 %;

раздел 6,7,8 – 5 %;

раздел 9 к – 5%;

оформление пояснительной записки и графического материала к 15.12.22 – 10 %

Защита курсового проекта с 21.12 по 28.12.23г.

РУКОВОДИТЕЛЬ Богдан Е. В.

(подпись)

Задание принял к исполнению _____ Гончаренко Д.Г.

(дата и подпись студента)

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Язык C++ является довольно мощным инструментом для умелого разработчика. Сам язык имеет 4 парадигмы, за счёт этого можно по-разному выстраивать структуру программы. Он сочетает свойства как высокоуровневых, так и низкоуровневых языков программирования.

Если сравнивать данный язык с C, то можно выделить разные парадигмы: C++ в первую очередь был создан для обретения других парадигм, речь идёт про Объектно-ориентированное и обобщённое программирование.

Данный язык широко используется для разработки программного обеспечения, написания игр, создание операционных систем, написания драйверов и решения множества других задач.

Язык имеет богатую стандартную библиотеку, включающую в себя множество алгоритмов, контейнеров, ввод-вывод, строковые, языковую поддержку, регулярные выражения, поддержку многопоточности и многое другое.

Для реализации удобного графического интерфейса в данной курсовой работе был использован фреймворк под названием Qt. Он предлагает широкий набор готовых виджетов и элементов управления, что упрощает создание интуитивно понятного и функционального пользовательского интерфейса. Также Qt имеет встроенные модули для работы с мультимедийными данными.

Таким образом для разработки видеоплеера язык C++ и фреймворк Qt подходят идеально.

1 ОБЗОР ЛИТЕРАТУРЫ

1.1 Анализ существующих аналогов

Перед началом работы необходимо изучить представленные в общем доступе аналоги видеоплееров для определения необходимых функций и особенностей выбранной темы.

Поэтому в данном смысле опущение некоторых особенностей является преимуществом над аналогами.

1.1.1 VLC Media Player

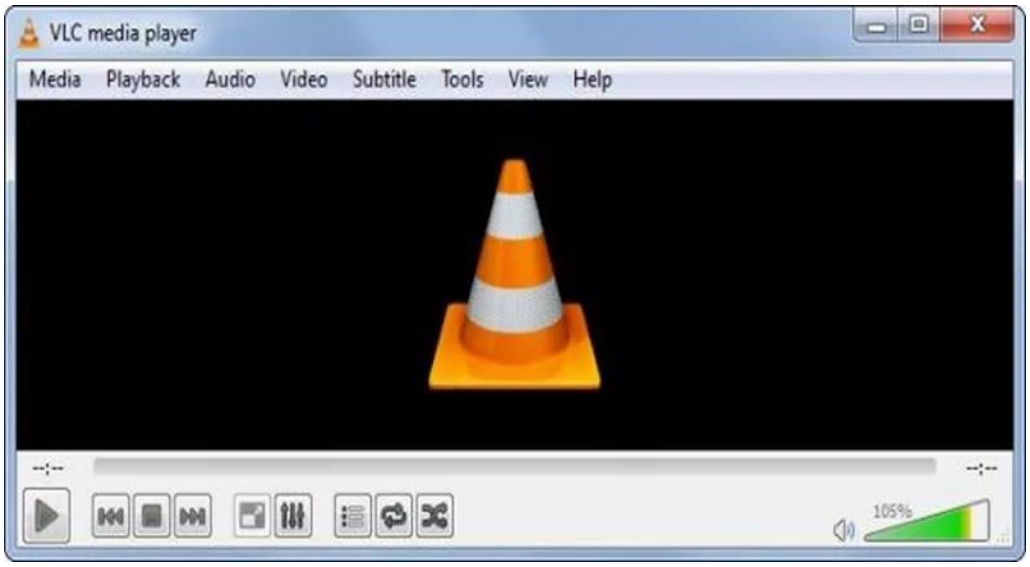

Рис. 1.1 Приложение “VLC Media Player ”

В данном приложении (рис. 1.1) реализованы широкая поддержка форматов, кроссплатформенность, конвертация форматов, потоковая передача.

Из минусов можно отметить сложность интерфейса по сравнению с более простыми плеерами, особенно для новых пользователей.

1.1.2 QuickTime Player

Рис. 1.2 Приложение “QuickTime Player ”

Приложение (рис. 1.2) имеет интуитивно понятный интерфейс, может воспроизводить высококачественные видеофайлы, обеспечивает высокую четкость изображения.

Из минусов можно отметить ограниченность на платформах, отличных от macOS, для которой и разработан этот видеоплеер в первую очередь, и ограниченность в поддерживаемых форматах видеофайлов.

1.1.3 Выводы по аналогам

Делая вывод на основе анализа приведенных приложений, можно сказать, что написанное мной приложение не может стать конкурентом в плане технических возможностей, однако оно может предложить простоту пользовательского интерфейса и возможность работы на Windows.

1.2 Обзор методов и алгоритмов решения поставленной задачи

Для создания видеоплеера был использован фреймворк Qt. Он содержит стандартные библиотеки C++, а также набор инструментов для создания удобного графического интерфейса.

Пользователь может совершать действия, нажимая на различные кнопки. Такой метод основан на механизме сигналов и слотов.

Для отображения видеоконтента используются виджеты QVideoWidget, предоставляемые модулем QtMultimedia. Для управления процессом воспроизведения используется объект QMediaPlayer.

Пользовательский интерфейс реализован используя классы QPushButton (для клавиш), QLabel (для надписей) и QSlider (для ползунков).

2 ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ

В данном разделе описываются входные и выходные данные программы, диаграмма классов, а также приводится описание используемых классов и их методов.

2.1 Разработка диаграммы классов

Диаграмма классов данной курсовой работы приведена в приложении А.

2.2 Описание классов

2.2.1 Класс видеоплеера

Класс VideoPlayer - класс
Описание полей класса VideoPlayer:
`QMediaPlayer* player` – объект, отвечающий за работу видеоплеера
`QVideoWidget* video` – объект, отвечающий за отображение видео
`QAudioOutput* audio` – объект, отвечающий за воспроизведение звука
Описание методов класса VideoPlayer:
`Playlist()` – конструктор по умолчанию
`~Playlist()` – деструктор
`void play(Video* video)` – запускает воспроизведение указанного видео
`void seek(qint32 seconds)` – производит перемотку видео на указанное количество секунд вперёд или назад
`void pause()` – приостанавливает воспроизведение видео
`int duration()` – возвращает длительность текущего видео в миллисекундах
`int position()` – возвращает текущую позицию воспроизведения видео в миллисекундах
`void setPosition(qint32 position)` – устанавливает текущую позицию воспроизведения видео
`void setVolume(double volume)` – устанавливает громкость аудио воспроизведения
`void play()` – запускает воспроизведение видео
`QMediaPlayer* getPlayer()` – возвращает указатель на player
`QVideoWidget* getVideoWidget()` – возвращает указатель на video

2.2.2 Класс плейлиста

Класс `Playlist` представляет собой модель плейлиста видео с функционалом управления воспроизведением.

Описание полей класса Playlist:

`bool shuffle` – флаг, указывающий, включено ли перемешивание плейлиста

`RepeatStates repeat` – состояние повторения

`std::vector<Video*> playlist` – контейнер для всех видео

`int currentlyPlaying` – индекс видео, которое воспроизводится в данный момент

Описание методов класса Playlist:

`Playlist()` – конструктор

`~Playlist()` - деструктор

`void addToPlaylist(Video* video)` – добавить видео в плейлист

`void repeatClicked()` – изменяет состояние повторения в зависимости от текущего состояния

`bool getRepeat()` – получение текущего состояния повторения

`qint32 getSize()` – получение размера плейлиста

`bool getShuffle()` – получение состояния перемешивания

`void setShuffle(bool newValue)` – устанавливает состояние перемешивания

`void setRepeat()` – устанавливает состояние повторения

`void raise(int index)` – перемещает видео вверх по плейлисту по указанному индексу

`void lower(int index)` перемещает видео вниз по плейлисту по указанному индексу

`Video* next()` – получение следующего видео из плейлиста

`Video* previous()` – получение предыдущего видео из плейлиста

`void erase(int index)` – удаляет видео из плейлиста по указанному индексу

2.2.3 Класс видео

Класс `Video` представляет собой объект видео с базовой информацией о видеофайле.

Игровое поле составляет из себя кольцо. На экране это кольцо будет отображаться в виде контура прямоугольника. Каждая ячейка имеет позицию в этом контуре.

Описание полей класса Video:

`QString path` – путь к видеофайлу

`QString name` – имя видеофайла

Описание методов класса Video:

`Video(const QString& path, const QString& name)` – конструктор

`~Video()` – деструктор

`const QString& getName(int posX)` – получение имени видеофайла

`void setName(const QString& newName)` – устанавливает новое имя для видеофайла

`const QString& getPath(int posX)` – получение пути к видеофайлу

`void setPath(const QString& newPath)` – устанавливает новый путь к видеофайлу

2.2.4 Класс интерфейса

Класс MainWindow представляет собой интерфейс приложения.

Описание полей класса MainWindow:

`Ui::MainWindow* ui` – указатель на объект, представляющий собой разметку окна

`VideoPlayer* videoplayer` – указатель на видеоплеер

`Playlist* playlist` – указатель на плейлист

`bool toolsAreHidden` – флаг, указывающий, скрыта ли панель инструментов

`bool playlistIsHidden` – флаг, указывающий, скрыт ли плейлист

Описание методов класса MainWindow:

`MainWindow(QWidget* parent)` – конструктор по умолчанию

`~MainWindow()` – деструктор

`void keyPressEvent(QKeyEvent* event)` – обрабатывает события нажатия клавиш

`void keyPressedP()` – обрабатывает нажатие клавиши “P”

`void keyPressedI()` – обрабатывает нажатие клавиши “I”

`void resizeEvent(QResizeEvent* event)` – обрабатывает события изменения размеров окна

`void updateTimeLabel()` – обновляет метки времени

3 РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ

3.1 Разработка схем алгоритмов

Метод `next()` класса `Playlist` определяет какое видео в этом плейлисте будет следующим для воспроизведения. Схема алгоритма `next()` приведена в приложении Б.

Метод `previous()` класса `Playlist` определяет какое видео в этом плейлисте было предыдущим. Схема алгоритма `previous()` приведена в приложении В.

3.2 Разработка алгоритмов

3.2.1 Разработка алгоритма метода next()

Метод `next()` класса `Playlist` определяет какое видео в этом плейлисте будет следующим для воспроизведения

Шаг 1. Начало.

Шаг 2. Если перемешивание плейлиста включено, вернуть видео с индексом, равным остатку от деления случайного числа на размер плейлиста.

Шаг 3. Если состояние повтора – повтор видео, вернуть видео, которое воспроизводится в данный момент.

Шаг 4. Если видео, которое воспроизводится в данный момент, - не последнее в плейлисте, перейти к шагу 7.

Шаг 5. Если состояние повтора – повтор плейлиста, установить значение currentlyPlaying равным 0.

Шаг 6. Если состояние повтора – нет повтора, то удалить плейлист и вернуть nullptr.

Шаг 7. Увеличить currentlyPlaying на 1.

Шаг 8. Вернуть видео, индекс которого равен currentlyPlaing.

Шаг 9. Конец.

3.2.2 Разработка алгоритма метода previous()

Метод `previous()` класса `Playlist` определяет какое видео в этом плейлисте было предыдущим.

Шаг 1. Начало.

Шаг 2. Если перемешивание плейлиста включено, вернуть видео с индексом, равным остатку от деления случайного числа на размер плейлиста.

Шаг 3. Если состояние повтора – повтор видео, вернуть видео, которое воспроизводится в данный момент.

Шаг 4. Если индекс видео равен 0, установить значение currentlyPlaying равным размеру плейлиста, уменьшенным на 1.

Шаг 5. Если индекс видео больше 0, уменьшить значение currentlyPlaying на 1.

Шаг 6. Вернуть видео, индекс которого равен currentlyPlaying.

Шаг 7. Конец

4 РЕЗУЛЬТАТЫ РАБОТЫ

При запуске приложения открывается окно с черным фоном и панелью инструментов.

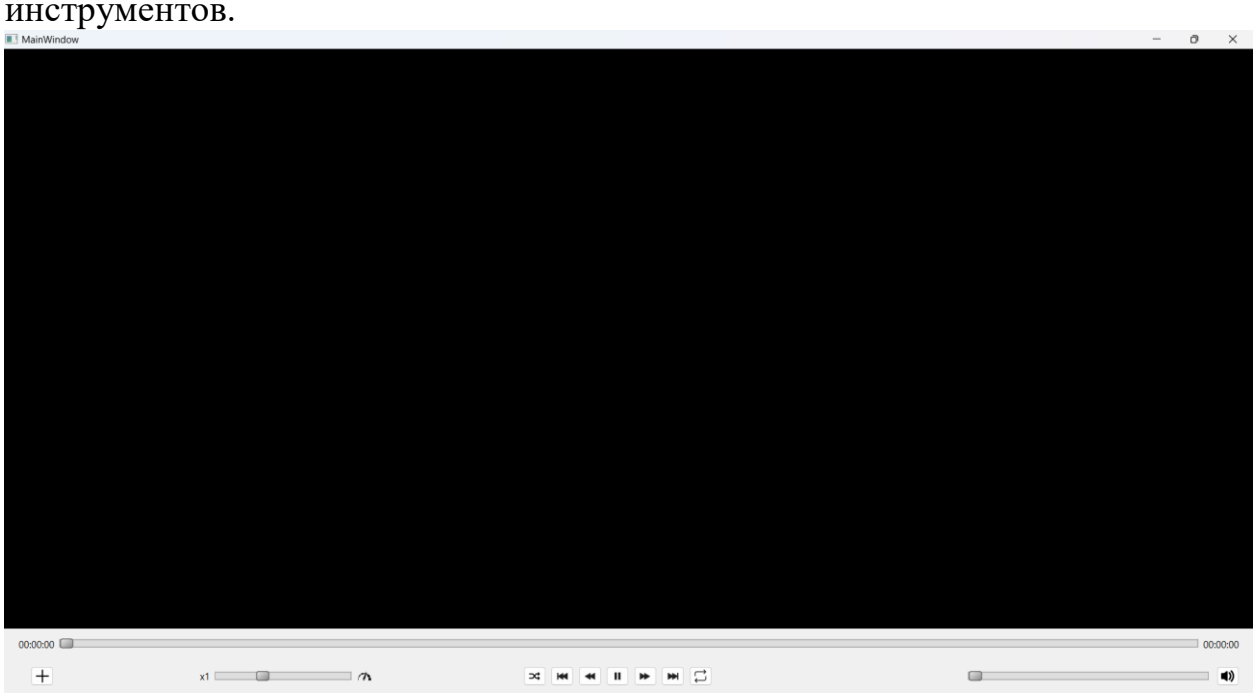

Рисунок 4.1 – Видеоплеер при первом запуске

Пользователь может открыть видеофайл используя соответствующую кнопку. Как только он это сделает, начнется воспроизведение соответствующего видеофайла.

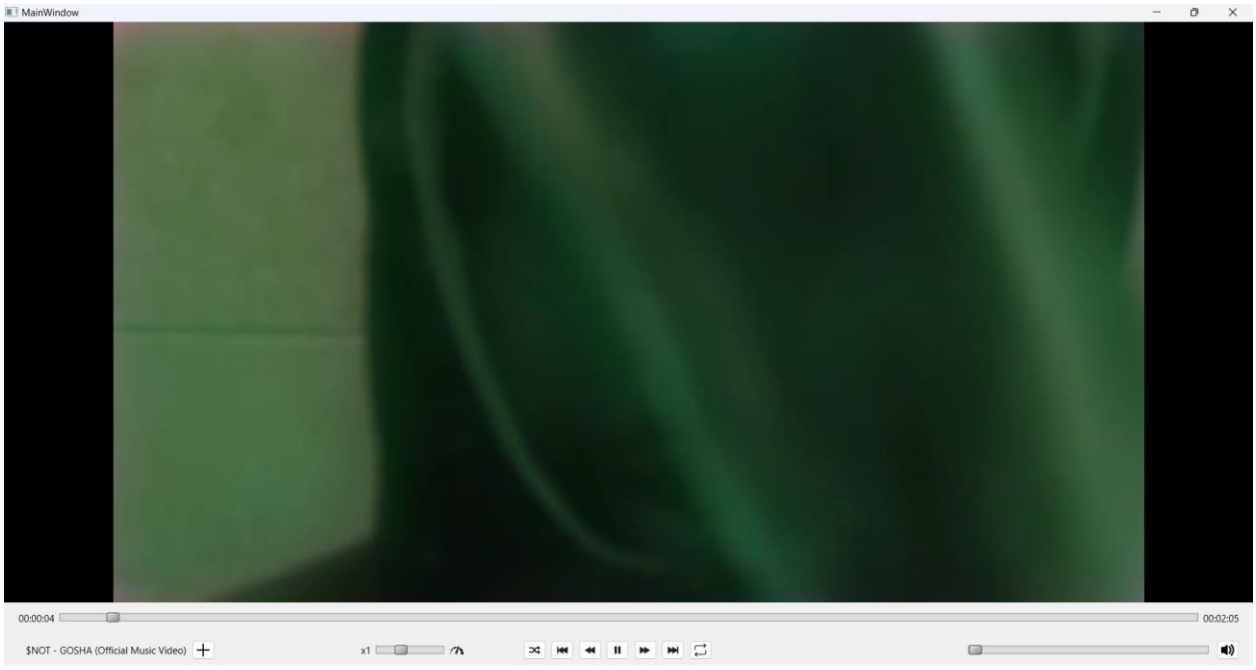

Рисунок 4.2 – Видеоплеер при добавлении видеофайла

При нажатии на клавишу “P”, скрывается или раскрывается панель, где находится плейлист.

Рисунок 4.3 – Режим видеоплеера (плейлист – раскрыт, панель управления – раскрыта)

При нажатии на клавишу “I”, скрывается или раскрывается панель управления.

Рисунок 4.4 – Режим видеоплеера (плейлист – раскрыт, панель управления – скрыта)

Рисунок 4.4 – Режим видеоплеера (плейлист – скрыт, панель управления – скрыта)

Также при помощи интерфейса можно управлять скоростью воспроизведения, перематывать видео, включать режимы повтора и перемешивания, удалять видео из плейлиста, добавлять новые видео в плейлист.

ЗАКЛЮЧЕНИЕ

В процессе выполнения курсовой работы были закреплены знания основ программирования и алгоритмизации. Были приобретены знания в объектно-ориентированном программировании. Также я познакомился с созданием полноценных проектов на фреймворке Qt и убедился, что он идеально подходит для создания несложных проектов с удобным интерфейсом.

Разработанный видеоплеер демонстрирует основные принципы работы с мультимедийным контентом, включая управление воспроизведением, выбор видео из плейлиста, регулировка громкости и др. Как упоминалось раннее, моя версия является неполной и открыта для модернизации. Например, можно добавить поддержку широкого спектра форматов видеофайлов.

Видеоплеер имеет удобный и интуитивно понятный интерфейс.

На основе полученных знаний при создании игры можно несложно будет создать приложения, где от пользователя будет требоваться ввод каких-либо данных и нажимание кнопок на экране. Такими примерами могут быть приложение для ведения заметок, ежедневник и т.д.

Создание игры было выполнено на ОС Windows 11, используя среду разработки QtCreator.

Код программы приведен в приложении Г.

СПИСОК ЛИТЕРАТУРЫ

[1] Стивен Прата. Язык программирования C++. Лекции и упражнения: учеб. пособие / С. Прата – СПб.: ООО «ДиаСофтЮП», 2003. – 1104с.
[2] Конструирование программ и языки программирования: метод. указания по курсовому проектированию – А. В. Бушкевич, А. М. Ковальчук, И. В. Лукьянова. – Минск : БГУИР, 2009.
[3] Qt Documentation [Электронный ресурс]. -Электронные данные. -Режим доступа: https://doc.qt.io/ Дата доступа: 24.10.2023
[4] UML-диаграммы классов [Электронный ресурс]. -Электронные данные. -Режим доступа: https://habr.com/ru/articles/150041/ Дата доступа 27.10.2023
[5] СТП 01–2017. Дипломные проекты (работы): общие требования. – Введ. 2017–01–01. – [Электронный ресурс].– 2017– Режим доступа: http://library.bsuir.by/online/showpage.jsp?PageID=86151 – Дата доступа: 23.11.2023

ПРИЛОЖЕНИЕ А

(обязательное)

Диаграмма классов

ПРИЛОЖЕНИЕ Б

(обязательное)

Схема метода rentCheck()

ПРИЛОЖЕНИЕ В

(обязательное)

Схема метода forfeitPressed()

ПРИЛОЖЕНИЕ Г

(обязательное)

Код программы

Файл main.cpp:

```
#include "mainwindow.h"
#include <QApplication>
int main(int argc, char *argv[]) {
  QApplication a(argc, argv);
  MainWindow w;
  w.show();
  return a.exec(); {
```

Файл mainwindow.cpp:

```
#include "mainwindow.h"
#include "ui_mainwindow.h"
#include <QStyle>
#include <QFileDialog>
#include <algorithm>
QString formatTime(qint64 milliseconds) {
  qint64 seconds = milliseconds / 1000;
  qint64 hours = seconds / 3600;
  qint64 minutes = (seconds % 3600) / 60;
  seconds = seconds % 60;
  std::ostringstream oss;
  oss << std::setw(2) << std::setfill('0') << hours << ":"
    << std::setw(2) << std::setfill('0') << minutes << ":"
    << std::setw(2) << std::setfill('0') << seconds;
  return QString::fromStdString(oss.str()); {
MainWindow::MainWindow(QWidget *parent)
  : QMainWindow(parent)
  , ui(new Ui::MainWindow) {
  ui->setupUi(this);
  setWindowState(Qt::WindowMaximized);
  ui->button_pause->setIcon(style()-
>standardIcon(QStyle::SP_MediaPause));
  ui->button_forward->setIcon(style()-
>standardIcon(QStyle::SP_MediaSeekForward));
  ui->button_next->setIcon(style()-
>standardIcon(QStyle::SP_MediaSkipForward));
  ui->button_backwards->setIcon(style()-
>standardIcon(QStyle::SP_MediaSeekBackward));
  ui->button_previous->setIcon(style()-
>standardIcon(QStyle::SP_MediaSkipBackward));
  ui->button_shuffle->setIcon(QIcon("D:/shuffle.png"));
  ui->button_repeat->setIcon(QIcon("D:/repeatPlaylist.png"));
  ui->playerBox->setStyleSheet("background-color: black");
  ui->button_addToPlaylist->setIcon(QIcon("D:/plus.png"));
  videoplayer = new VideoPlayer();
  videoplayer->getVideoWidget()->setParent(ui->playerBox);
```

```
  videoplayer->getVideoWidget()->setGeometry(0,0, ui->playerBox-
>width(), ui->playerBox->height());
  videoplayer->play();
  playlist = new Playlist();
  ui->label_name->setText("");
  ui->label_duration->setText(formatTime(videoplayer-
>duration()));
  ui->slider_time->setMaximum(videoplayer->duration());
  ui->slider_volume->setRange(0, 100);
  connect(ui->slider_volume, &QSlider::valueChanged,
videoplayer, &VideoPlayer::setVolume);
  QTimer *timer = new QTimer(this);
  connect(timer, &QTimer::timeout, this,
&MainWindow::updateTimeLabel);
  timer->start(1000);
  connect(videoplayer->getPlayer(),
&QMediaPlayer::mediaStatusChanged, this,
&MainWindow::handleMediaStatusChanged);
  connect(videoplayer->getPlayer(),
&QMediaPlayer::durationChanged, ui->slider_time,
&QSlider::setMaximum);
  connect(videoplayer->getPlayer(),
&QMediaPlayer::positionChanged, ui->slider_time,
&QSlider::setValue);
  connect(ui->slider_time, &QSlider::sliderMoved, videoplayer-
>getPlayer(), &QMediaPlayer::setPosition);
  connect(ui->button_shuffle, &QPushButton::clicked, this,
&MainWindow::shuffleButtonClicked);
  connect(ui->button_previous, &QPushButton::clicked, this,
&MainWindow::previousButtonClicked);
  connect(ui->button_backwards, &QPushButton::clicked, this,
&MainWindow::backwardsButtonClicked);
  connect(ui->button_pause, &QPushButton::clicked, this,
&MainWindow::pauseButtonClicked);
  connect(ui->button_forward, &QPushButton::clicked, this,
&MainWindow::forwardButtonClicked);
  connect(ui->button_next, &QPushButton::clicked, this,
&MainWindow::nextButtonClicked);
  connect(ui->button_repeat, &QPushButton::clicked, this,
&MainWindow::repeatButtonClicked);
  connect(ui->button_addToPlaylist, &QPushButton::clicked, this,
&MainWindow::addButtonClicked);
  QVBoxLayout *l = new QVBoxLayout(ui-
>scrollAreaWidgetContents);
  l->setAlignment(Qt::AlignTop);
  ui->scrollAreaWidgetContents->setLayout(l); {
void
MainWindow::handleMediaStatusChanged(QMediaPlayer::MediaStatus
status) {
  if (status == QMediaPlayer::EndOfMedia) {
    if (playlist->getRepeat() == RepeatVideo){
      videoplayer->getPlayer()->setPosition(0);
```

```
      videoplayer->getPlayer()->play(); {
    else{
      nextButtonClicked(); { { {
void MainWindow::resizeEvent(QResizeEvent *event) {
  QMainWindow::resizeEvent(event);
  if (toolsAreHidden && playlistIsHidden){
    ui->playerBox->setGeometry(0,0,this->size().width(), this-
>size().height()); {
  else if (toolsAreHidden && !playlistIsHidden){
    ui->playerBox->setGeometry(0,0,this->size().width() * 0.8,
this->size().height());
    ui->playlistBox->setGeometry(this->size().width() * 0.8, 0,
this->size().width() * 0.2, this->size().height()); {
  else if (!toolsAreHidden && !playlistIsHidden){
    ui->playerBox->setGeometry(0,0,this->size().width() * 0.8,
this->size().height() * 0.9);
    ui->playlistBox->setGeometry(this->size().width() * 0.8, 0,
this->size().width() * 0.2, this->size().height()* 0.9);
    ui->toolBox->setGeometry(0, this->size().height() * 0.9,
this->size().width(), this->size().height() * 0.1); {
  else if (!toolsAreHidden && playlistIsHidden){
    ui->playerBox->setGeometry(0,0,this->size().width(), this-
>size().height() * 0.9);
    ui->toolBox->setGeometry(0, this->size().height() * 0.9,
this->size().width(), this->size().height() * 0.1); {
  videoplayer->getVideoWidget()->setGeometry(0, 0, ui-
>playerBox->width(), ui->playerBox->height()); {
void MainWindow::keyPressedI(){
  if (toolsAreHidden){
    ui->toolBox->show(); {
  else {
    ui->toolBox->hide(); {
  toolsAreHidden = !toolsAreHidden; {
void MainWindow::keyPressedP(){
  if (playlistIsHidden){
    ui->playlistBox->show(); {
  else {
    ui->playlistBox->hide(); {
  playlistIsHidden = !playlistIsHidden; {
void MainWindow::updateTimeLabel() {
  qint64 position = videoplayer->getPlayer()->position();
  QString timeString = formatTime(position);
  ui->label_currentTime->setText(timeString); {
void MainWindow::onUpButtonClicked(){
  QPushButton *clickedButton =
qobject_cast<QPushButton*>(sender());
  if (clickedButton) {
    QGroupBox *videoBox = qobject_cast<QGroupBox
*>(clickedButton->parentWidget()->parentWidget());
    if (videoBox) {
      int elementId = videoBox->property("id").toInt();
      if (elementId == 0){
```

```
        return; {
      playlist->erase(elementId);
      QVBoxLayout* layout = qobject_cast<QVBoxLayout*>(ui-
>scrollAreaWidgetContents->layout());
      int index = layout->indexOf(videoBox);
      qDebug() << index;
      if (index > 0) {
        layout->removeWidget(videoBox);
        layout->insertWidget(index - 1, videoBox); { { { {
void MainWindow::onDownButtonClicked(){
  QPushButton *clickedButton =
qobject_cast<QPushButton*>(sender());
  if (clickedButton) {
    QGroupBox *videoBox = qobject_cast<QGroupBox
*>(clickedButton->parentWidget()->parentWidget());
    if (videoBox) {
      int elementId = videoBox->property("id").toInt();
      if (elementId == playlist->getSize()- 1) {
        return; {
      playlist->lower(elementId);
      // Move the videoBox down in the layout
      QVBoxLayout *layout = qobject_cast<QVBoxLayout*>(ui-
>scrollAreaWidgetContents->layout());
      if (layout) {
        qDebug() << elementId << " " << elementId + 1;
        int index1 = elementId; // Replace with your actual
index 1
        int index2 = index1 + 1; // Replace with your actual
index 2
        QGroupBox *groupBox1 = qobject_cast<QGroupBox*>(layout-
>itemAt(index1)->widget());
        QGroupBox *groupBox2 = qobject_cast<QGroupBox*>(layout-
>itemAt(index2)->widget());
        if (groupBox1 && groupBox2) {
          QString text1 = groupBox1->title();
          QString text2 = groupBox2->title();
          groupBox1->setTitle(text2);
          groupBox2->setTitle(text1); { { { { {
void MainWindow::onDeleteButtonClicked(){
  QPushButton *clickedButton = qobject_cast<QPushButton
*>(sender());
  if (clickedButton) {
    QGroupBox *videoBox = qobject_cast<QGroupBox
*>(clickedButton->parentWidget());
    if (videoBox) {
      int elementId = videoBox->property("id").toInt();
      qDebug() << elementId;
      playlist->erase(elementId);
      QLayout *layout = ui->scrollAreaWidgetContents->layout();
      layout->removeWidget(videoBox);
      delete videoBox; { { {
void MainWindow::addButtonClicked(){
```

```
  QString FileName = QFileDialog::getOpenFileName(this,
tr("Select Video File"), "", tr("MP4 Files (*.mp4)"));
  QFileInfo fileInfo(FileName);
  playlist->addToPlaylist(new Video(FileName,
fileInfo.baseName()));
  if (playlist->getSize() == 1){
    nextButtonClicked(); {
  QGroupBox* newVideoBox = new QGroupBox();
  QHBoxLayout *layout = new QHBoxLayout();
  QPushButton* deleteVideoButton = new QPushButton("");
  connect(deleteVideoButton, &QPushButton::clicked, this,
&MainWindow::onDeleteButtonClicked);
  deleteVideoButton->setIcon(QIcon("D:/delete"));
  QLabel* newVideoLabel = new QLabel(fileInfo.baseName());
  QGroupBox* buttonsBox = new QGroupBox();
  QVBoxLayout  *buttonsLayout = new QVBoxLayout ();
  buttonsBox->setLayout(buttonsLayout);
  deleteVideoButton->setStyleSheet("margin-left: 0");
  layout->addWidget(newVideoLabel);
  layout->addWidget(buttonsBox, 0, Qt::AlignRight);
  layout->addWidget(deleteVideoButton, 0, Qt::AlignRight);
  layout->setStretch(0,5);
  layout->setStretch(1,1);
  layout->setStretch(1,1);
  newVideoBox->setLayout(layout);
  newVideoBox->setSizePolicy(QSizePolicy::Expanding,
QSizePolicy::Fixed);
  newVideoBox->resize(newVideoBox->size().width(), 50);
  newVideoBox->setStyleSheet("#newVideoBox { border: 2px solid
gray; border-radius: 5px; margin-top: 1ex; }");
  qDebug() << playlist->getSize();
  newVideoBox->setProperty("id", playlist->getSize() - 1);
  newVideoBox->setObjectName("video " +
QString::number(playlist->getSize()));
  connect(deleteVideoButton, &QPushButton::clicked,this,
&MainWindow::onDeleteButtonClicked);
  ui->scrollAreaWidgetContents->layout()-
>addWidget(newVideoBox); {
void MainWindow::pauseButtonClicked(){
  if (videoplayer->getPlayer()->isPlaying()) {
    videoplayer->getPlayer()->pause();
    ui->button_pause->setIcon(style()-
>standardIcon(QStyle::SP_MediaPlay)); {
  else {
    videoplayer->getPlayer()->play();
    ui->button_pause->setIcon(style()-
>standardIcon(QStyle::SP_MediaPause)); { {
void MainWindow::backwardsButtonClicked(){
  qint64 currentPosition = videoplayer->getPlayer()->position();
  videoplayer->getPlayer()->setPosition(currentPosition -
10000); {
void MainWindow::forwardButtonClicked(){
```

```
  qint64 currentPosition = videoplayer->getPlayer()->position();
  videoplayer->getPlayer()->setPosition(currentPosition +
30000); {
void MainWindow::previousButtonClicked(){
  Video* newVideo = playlist->previous();
  if (newVideo == nullptr){
    videoplayer->pause();
    ui->label_name->setText("");
    ui->label_duration->setText("00:00");
    return; {
  videoplayer->getPlayer()->setSource(newVideo->getPath());
  videoplayer->play();
  ui->label_name->setText(newVideo->getName());
  ui->label_duration->setText(formatTime(videoplayer-
>duration()));
  ui->slider_time->setMaximum(videoplayer->duration()); {
void MainWindow::shuffleButtonClicked(){
  if (!playlist->getShuffle()){
    ui->button_shuffle->setStyleSheet("background-color:
rgb(138, 218, 178);");
  } else {
    ui->button_shuffle->setStyleSheet(""); {
  playlist->setShuffle(!playlist->getShuffle()); {
void MainWindow::nextButtonClicked(){
  Video* newVideo = playlist->next();
  if (newVideo == nullptr){
    videoplayer->getPlayer()->stop();
    ui->label_currentTime->setText("00:00");
    ui->slider_time->setValue(0);
    ui->label_name->setText("");
    ui->label_duration->setText("00:00");
    return; {
  videoplayer->getPlayer()->setSource(newVideo->getPath());
  videoplayer->play();
  ui->label_name->setText(newVideo->getName());
  ui->label_duration->setText(formatTime(videoplayer-
>duration()));
  ui->slider_time->setMaximum(videoplayer->duration()); {
void MainWindow::repeatButtonClicked(){
  qDebug() << playlist->getRepeat();
  if (playlist->getRepeat() == NoRepeat){
    playlist->setRepeat(RepeatPlaylist);
    ui->button_repeat->setIcon(QIcon("D:/repeatPlaylist.png"));
    ui->button_repeat->setStyleSheet("background-color: rgb(138,
218, 178);"); {
  else if (playlist->getRepeat() == RepeatPlaylist){
    playlist->setRepeat(RepeatVideo);
    ui->button_repeat->setIcon(QIcon("D:/repeatSong.png"));
    ui->button_repeat->setStyleSheet("background-color: rgb(138,
218, 178);"); {
  else if (playlist->getRepeat() == RepeatVideo){
    playlist->setRepeat(NoRepeat);
```

```
    ui->button_repeat->setIcon(QIcon("D:/repeatPlaylist.png"));
    ui->button_repeat->setStyleSheet(""); { {
MainWindow::~MainWindow() {
  delete ui; {
void MainWindow::keyPressEvent(QKeyEvent *event) {
  if (event->key() == Qt::Key_I) {
    keyPressedI();
    QTimer::singleShot(0, this, [=]() {
      this->resize(this->width() + 1, this->height());
      this->resize(this->width() - 1, this->height());
    }); {
  else if (event->key() == Qt::Key_P){
    keyPressedP();
    QTimer::singleShot(0, this, [=]() {
      this->resize(this->width() + 1, this->height());
      this->resize(this->width() - 1, this->height());
    }); {
  QWidget::keyPressEvent(event); {
```

Файл mainwindow.h:

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H
#include <QMainWindow>
#include "VideoPlayer.h"
#include "playlist.h"
QT_BEGIN_NAMESPACE
namespace Ui { class MainWindow; }
QT_END_NAMESPACE
class MainWindow : public QMainWindow {
  Q_OBJECT
public:
  MainWindow(QWidget *parent = nullptr);
  ~MainWindow();
  void keyPressEvent(QKeyEvent *event) override;
  void keyPressedI();
  void keyPressedP();
  void handleMediaStatusChanged(QMediaPlayer::MediaStatus
status);
  void resizeEvent(QResizeEvent *event);
  void updateTimeLabel();
private slots:
  void onUpButtonClicked();
  void onDownButtonClicked();
  void onDeleteButtonClicked();
  void pauseButtonClicked();
  void backwardsButtonClicked();
  void forwardButtonClicked();
  void previousButtonClicked();
  void shuffleButtonClicked();
  void nextButtonClicked();
  void repeatButtonClicked();
  void addButtonClicked();
```

```
private:
  Ui::MainWindow *ui;
  VideoPlayer* videoplayer;
  Playlist* playlist;
  bool toolsAreHidden = false, playlistIsHidden = true;
};
#endif // MAINWINDOW_H
```

Файл playlist.cpp:

```
#include "playlist.h"
void Playlist::addToPlaylist(Video *video){
  playlist.push_back(video); {
void Playlist::repeatClicked(){
  if (repeat == NoRepeat){
    repeat = RepeatPlaylist; {
  else if (repeat == RepeatPlaylist){
    repeat = RepeatVideo; {
  else if (repeat == RepeatVideo){
    repeat = NoRepeat; { {
RepeatStates Playlist::getRepeat(){
  return repeat; {
int Playlist::getSize(){
  return playlist.size(); {
bool Playlist::getShuffle(){
  return shuffle; {
void Playlist::setShuffle(bool newShuffle){
  shuffle = newShuffle; {
void Playlist::setRepeat(RepeatStates newRepeat){
  repeat = newRepeat; {
void Playlist::raise(int index){
  if (index == 0){
    return; {
  Video* temp = playlist[index];
  playlist[index] = playlist[index - 1];
  playlist[index - 1] = temp; {
void Playlist::lower(int index){
  if (index == playlist.size() - 1){
    return; {
  Video* temp = playlist[index];
  playlist[index] = playlist[index + 1];
  playlist[index + 1] = temp; {
Video *Playlist::next(){
  if (repeat == RepeatVideo){
    return playlist[currentlyPlaying]; {
  if (currentlyPlaying == playlist.size() - 1){
    if (repeat == RepeatPlaylist){
      currentlyPlaying = 0; {
    else {
      playlist.clear();
      return nullptr; { {
  else{
    currentlyPlaying++; {
```

```
  return playlist[currentlyPlaying]; {
Video *Playlist::previous(){
  if (repeat == RepeatVideo){
    return playlist[currentlyPlaying]; {
  if (currentlyPlaying > 0){
    currentlyPlaying--; {
  else if (currentlyPlaying == 0){
    currentlyPlaying = playlist.size() - 1; {
  return playlist[currentlyPlaying]; {
void Playlist::erase(int index){
  playlist.erase(playlist.begin() + index); {
```

Файл playlist.h:

```
#ifndef PLAYLIST_H
#define PLAYLIST_H
#include <QObject>
#include "video.h"
#include <QDebug>
enum RepeatStates{
  NoRepeat = 0,
  RepeatPlaylist = 1,
  RepeatVideo = 2,
};
class Playlist : public QObject {
  Q_OBJECT
public:
  Playlist() = default;
  ~Playlist() = default;
  void addToPlaylist(Video* video);
  void repeatClicked();
  RepeatStates getRepeat();
  int getSize();
  bool getShuffle();
  void setShuffle(bool newShuffle);
  void setRepeat(RepeatStates newRepeat);
  void raise(int index);
  void lower(int index);
  Video* next();
  Video* previous();
  void erase(int index);
private:
  bool shuffle = false;
  RepeatStates repeat = NoRepeat;
  std::vector<Video*> playlist;
  int currentlyPlaying = -1;
};
#endif // PLAYLIST_H
```

Файл video.cpp:

```
#include "video.h"
Video::Video(const QString &path, const QString &name) :
path(path), name(name) {}
```

```
const QString &Video::getPath() const {
  return path; {
void Video::setPath(const QString &newPath) {
  path = newPath; {
const QString &Video::getName() const {
  return name; {
void Video::setName(const QString &newName) {
  name = newName; {
```

Файл video.h:

```
#ifndef VIDEO_H
#define VIDEO_H
#include <QString>
class Video {
private:
  QString path;
  QString name;
public:
  Video(const QString &path, const QString &name);
  const QString& getPath() const;
  void setPath(const QString &newPath);
  const QString& getName() const;
  void setName(const QString &newName);
};
#endif // VIDEO_H
```

Файл videoplayer.cpp:

```
#include "videoplayer.h"
```

Файл videoplayer.h:

```
#ifndef VIDEOPLAYER_H
#define VIDEOPLAYER_H
#include <QObject>
#include <QtMultimedia>
#include <QtMultimediaWidgets>
#include "video.h"
class VideoPlayer : public QObject {
  Q_OBJECT
public:
  VideoPlayer(){
    player = new QMediaPlayer();
    video = new QVideoWidget();
    audio = new QAudioOutput();
    player->setAudioOutput(audio);
    player->setVideoOutput(video);
    video->setVisible(true);
    audio->setVolume(0);
    video->show();
    player->play();
  };
  ~VideoPlayer() = default;
  void play(const Video* video);
```

```
  void seek(int seconds){
    qint64 currentPosition = player->position();
    player->setPosition(currentPosition + seconds * 1000); {
  void pause(){
    player->pause(); {
  int duration(){
    return player->duration(); {
  int position(){
    return player->position(); {
  QMediaPlayer* getPlayer(){
    return player; {
  void setPosition(qint64 position);
  QVideoWidget* getVideoWidget(){
    return video; {
  void setVolume(double volume){
    volume /= 100;
    audio->setVolume(volume); {
  void play(){
    player->play(); {
private:
  QMediaPlayer *player;
  QVideoWidget *video;
  QAudioOutput *audio;
};
#endif // VIDEOPLAYER_H
```