Специальность: 09.02.07 Информационные системы и программирование

Проверил:

\_\_\_\_\_\_\_\_\_\_\_ Антонов М.Б.

Дата: «23» мая 2020 г.

Оценка: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Министерство образования и науки Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего образования

«Санкт-Петербургский национальный исследовательский университет информационных технологий, механики и оптики»

(Университет ИТМО)

факультет среднего профессионального образования

Отчет

о Лабораторной работе №17

по теме: Работа с аудио и видео Qt

по дисциплине: Разработка программных модулей

Выполнил:

студент группы 2333

\_\_\_\_\_\_\_\_\_\_\_ Агапова Е.А.

Санкт-Петербург 2020

Содержание

[1 Постановка задачи 3](#_Toc41139419)

[1.1 Цель 3](#_Toc41139420)

[1.2 Задачи 3](#_Toc41139421)

[1.3 Задание 3](#_Toc41139422)

[1.4 Общее ограничение на лабораторную работу 3](#_Toc41139423)

[1.5 Требования к отчёту 3](#_Toc41139424)

[2 Диаграммы 5](#_Toc41139425)

[3 Исходный код 6](#_Toc41139426)

[4 ЛИСТИНГ ВЫполнения программы 13](#_Toc41139427)

[5 Вывод 14](#_Toc41139428)

1. Постановка задачи
   1. Цель

Познакомиться с основами работы мультимедиа в Qt.

* 1. Задачи

1. изучить основы работы с модулем multimedia;
2. Создать простейшее приложение для воспроизведения видео.
   1. Задание

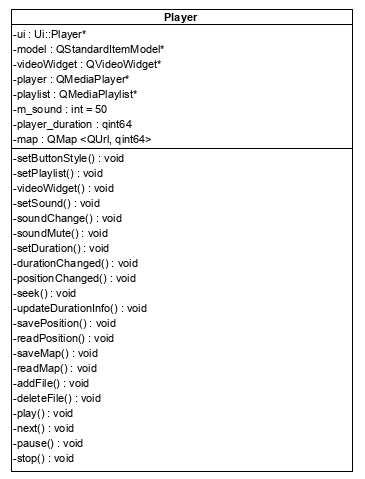
Требуется создать приложение для воспроизведения видеороликов с плейлистом.

Приложение при повторном запуске должно запоминать файлы, которые уже просматривало и воспроизводить их с места остановки.

Программа должна уметь переходить на заданную точку воспроизведения видео и изменять громкость.

* 1. Общее ограничение на лабораторную работу
* использовать разбитие на файлы основной программы (в main.cpp только функция main.cpp);
* документировать весь код используя аннотации Doxygen.
  1. Требования к отчёту
* цели лабораторной работы;
* задачи лабораторной работы;
* выполняемое задание;
* диаграмма(ы) классов;
* диаграммы деятельности (для методов и/или функций больше 10 строк кода (исключая форматирования и пустые строки));
* исходный код программы с комментариями (2 пробела вместо 4, 10pt, моноширинный шрифт);
* листинг выполнения программы;
* исходный код в виде файлов для запуска на произвольном ПК.

1. Диаграммы



*Рисунок 1 – Диаграмма классов*

1. Исходный код

В листингах 2.1-2.3 представлен исходный код программы, в листинге 2.4 – файл ресурсов.

Листинг 2.1 – Код программы

/\*\*

\* @file main.cpp

\* @author Agapova Ekaterina

\*/

#include "player.h"

#include <QApplication>

int main(int argc, char \*argv[])

{

QApplication a(argc, argv);

Player w;

w.show();

return a.exec();

}

Листинг 2.2 – Заголовочный файл player.h

/\*\*

\* @file player.h

\* @brief audio and video player

\* @author Agapova Ekaterina

\*/

#ifndef PLAYER\_H

#define PLAYER\_H

#include <QWidget>

#include <QMediaPlayer>

#include <QMediaPlaylist>

#include <QStandardItemModel>

#include <QVideoWidget>

QT\_BEGIN\_NAMESPACE

namespace Ui { class Player; }

QT\_END\_NAMESPACE

class Player : public QWidget

{

Q\_OBJECT

public:

Player(QWidget \*parent = nullptr);

~Player();

private:

Ui::Player \*ui;

QStandardItemModel \*model;

QVideoWidget \*widget;

QMediaPlayer \*player;

QMediaPlaylist \*playlist;

int m\_sound = 50;

qint64 player\_duration;

QMap <QUrl, qint64> map;

private:

/\*\*

\* @brief set ToolButton style

\*/

void setButtonStyle();

/\*\*

\* @brief setPlaylist

\*/

void setPlaylist();

/\*\*

\* @brief set videoWidget

\*/

void videoWidget();

/\*\*

\* @brief set sound

\*/

void setSound();

/\*\*

\* @brief set duration

\*/

void setDuration();

/\*\*

\* @brief save position to map

\*/

void savePosition();

/\*\*

\* @brief read position from map

\*/

void readPosition();

/\*\*

\* @brief save map to file

\*/

void saveMap();

/\*\*

\* @brief read map from file

\*/

void readMap();

private slots:

/\*\*

\* @brief play

\*/

void play();

/\*\*

\* @brief next

\*/

void next();

/\*\*

\* @brief pause

\*/

void pause();

/\*\*

\* @brief stop

\*/

void stop();

/\*\*

\* @brief addFile

\*/

void addFile();

/\*\*

\* @brief deleteFile

\*/

void deleteFile();

/\*\*

\* @brief soundChange

\* @param sound

\*/

void soundChange(QString sound);

/\*\*

\* @brief soundMute

\*/

void soundMute();

/\*\*

\* @brief durationChanged

\* @param duration

\*/

void durationChanged(qint64 duration);

/\*\*

\* @brief positionChanged

\* @param progress

\*/

void positionChanged(qint64 progress);

/\*\*

\* @brief seek

\* @param seconds

\*/

void seek(int seconds);

/\*\*

\* @brief updateDurationInfo

\* @param currentInfo

\*/

void updateDurationInfo(qint64 currentInfo);

};

#endif // PLAYER\_H

Листинг 2.3 – Заголовочный файл player.cpp

/\*\*

\* @file player.cpp

\* @brief audio and video player

\* @author Agapova Ekaterina

\*/

#include "player.h"

#include "ui\_player.h"

#include <QFileDialog>

#include <QFileInfo>

#include <QDir>

#include <QTime>

#include <QSpinBox>

#include <QDataStream>

#include <QDebug>

Player::Player(QWidget \*parent)

: QWidget(parent), ui(new Ui::Player)

{

ui->setupUi(this);

player = new QMediaPlayer(this);

playlist = new QMediaPlaylist(player);

this->readMap();

this->setButtonStyle();

this->setPlaylist();

this->videoWidget();

this->setDuration();

connect(ui->previous\_but, &QToolButton::clicked,

playlist, &QMediaPlaylist::previous);

connect(ui->next\_but, &QToolButton::clicked,

this, &Player::next);

connect(ui->play\_but, &QToolButton::clicked,

this, &Player::play);

connect(ui->pause\_but, &QToolButton::clicked,

this, &Player::pause);

connect(ui->stop\_but, &QToolButton::clicked,

this, &Player::stop);

connect(ui->add\_but, &QToolButton::clicked,

this, &Player::addFile);

connect(ui->del\_but, &QToolButton::clicked,

this, &Player::deleteFile);

}

Player::~Player()

{

this->savePosition();

this->saveMap();

delete ui;

delete model;

delete playlist;

delete player;

}

void Player::setButtonStyle()

{

ui->stop\_but->setIcon(style()->standardIcon(QStyle::SP\_MediaStop));

ui->play\_but->setIcon(style()->standardIcon(QStyle::SP\_MediaPlay));

ui->pause\_but->setIcon(style()->standardIcon(QStyle::SP\_MediaPause));

ui->sound->setIcon(style()->standardIcon(QStyle::SP\_MediaVolume));

ui->next\_but->setIcon(style()->standardIcon(QStyle::SP\_MediaSkipForward));

ui->previous\_but->setIcon(style()->standardIcon(QStyle::SP\_MediaSkipBackward));

}

void Player::setPlaylist()

{

model = new QStandardItemModel(this);

ui->tableView->setModel(model);

model->setHorizontalHeaderLabels(QStringList()<< tr("File")

<< tr("File Path"));

ui->tableView->hideColumn(1);

ui->tableView->setSelectionBehavior(QAbstractItemView::SelectRows);

ui->tableView->setSelectionMode(QAbstractItemView::SingleSelection);

ui->tableView->setEditTriggers(QAbstractItemView::NoEditTriggers);

ui->tableView->horizontalHeader()->setStretchLastSection(true);

player->setPlaylist(playlist);

this->setSound();

playlist->setPlaybackMode(QMediaPlaylist::Loop);

connect(ui->tableView, &QTableView::doubleClicked,

[this](const QModelIndex &index){

playlist->setCurrentIndex(index.row());

player->play();

});

connect(playlist, &QMediaPlaylist::currentIndexChanged,

[this](int index){

ui->currentTrack->setText(model->data(model->index(index, 0)).toString());

});

}

void Player::videoWidget()

{

widget = new QVideoWidget(ui->widget);

player->setVideoOutput(widget);

widget->resize(ui->widget->size());

widget->show();

}

void Player::addFile()

{

QStringList files = QFileDialog::getOpenFileNames(this, tr("Open files"),

QString(),

tr("Audio Files (\*.mp3 \*.wav);; "

"Video files (\*.avi)"));

foreach (QString filePath, files) {

QList<QStandardItem \*> items;

items.append(new QStandardItem(QDir(filePath).dirName()));

items.append(new QStandardItem(filePath));

model->appendRow(items);

playlist->addMedia(QUrl::fromLocalFile(filePath));

widget = new QVideoWidget;

}

}

void Player::deleteFile()

{

savePosition();

model->removeRow(ui->tableView->currentIndex().row());

playlist->removeMedia(ui->tableView->currentIndex().row()+1);

if(playlist->isEmpty()){

ui->currentTrack->setText("Current Track");

player->stop();

}

else{

ui->currentTrack->setText(model->data(model->index(playlist->currentIndex(), 0)).toString());

}

}

void Player::setSound()

{

player->setVolume(m\_sound);

ui->spinBox->setValue(50);

connect(ui->sound, &QToolButton::clicked,

this, &Player::soundMute);

connect(ui->spinBox, &QSpinBox::textChanged,

this, &Player::soundChange);

}

void Player::soundChange(QString sound)

{

if(sound.toInt()!=0)

this->m\_sound = sound.toInt();

player->setVolume(sound.toInt());

}

void Player::soundMute()

{

if(ui->spinBox->value() != 0){

ui->spinBox->setValue(0);

player->setMuted(true);

ui->sound->setIcon(style()->standardIcon(QStyle::SP\_MediaVolumeMuted));

} else {

ui->spinBox->setValue(this->m\_sound);

player->setMuted(false);

ui->sound->setIcon(style()->standardIcon(QStyle::SP\_MediaVolume));

}

}

void Player::setDuration()

{

ui->timeSlider->setRange(0, player->duration()/1000);

connect(ui->timeSlider, &QSlider::sliderMoved,

this, &Player::seek);

connect(player, &QMediaPlayer::durationChanged,

this, &Player::durationChanged);

connect(player, &QMediaPlayer::positionChanged,

this, &Player::positionChanged);

}

void Player::durationChanged(qint64 duration)

{

player\_duration = duration/1000;

ui->timeSlider->setMaximum(duration/1000);

}

void Player::positionChanged(qint64 progress)

{

if(!ui->timeSlider->isSliderDown())

ui->timeSlider->setValue(progress/1000);

updateDurationInfo(progress/1000);

}

void Player::updateDurationInfo(qint64 currentInfo)

{

QString tStr;

if (currentInfo || player\_duration) {

QTime currentTime((currentInfo / 3600) % 60, (currentInfo / 60) % 60,

currentInfo % 60, (currentInfo \* 1000) % 1000);

QTime totalTime((player\_duration / 3600) % 60, (player\_duration / 60) % 60,

player\_duration % 60, (player\_duration \* 1000) % 1000);

QString format = "mm:ss";

if (player\_duration > 3600)

format = "hh:mm:ss";

tStr = currentTime.toString(format) + " / " + totalTime.toString(format);

}

ui->time->setText(tStr);

}

void Player::seek(int seconds)

{

player->setPosition(seconds\*1000);

}

void Player::savePosition()

{

map.insert(playlist->currentMedia().request().url(), player->position());

}

void Player::readPosition()

{

foreach(QUrl key, map.keys()){

if(key.toString().compare(playlist->currentMedia().request().url().toString()) == 0){

qint64 value = map.value(key);

player->setPosition(value);

}

}

}

void Player::saveMap()

{

QFile f ("position.txt");

if (!f.open(QIODevice::WriteOnly)) return;

QDataStream str(&f);

str << map;

}

void Player::readMap()

{

QFile f ("position.txt");

if (!f.open(QIODevice::ReadOnly)) return;

QDataStream str(&f);

map.clear();

str >> map;

f.remove();

}

void Player::play()

{

player->play();

ui->play\_but->setEnabled(false);

ui->pause\_but->setEnabled(true);

ui->stop\_but->setEnabled(true);

readPosition();

}

void Player::next()

{

savePosition();

playlist->next();

player->play();

}

void Player::pause()

{

savePosition();

ui->pause\_but->setEnabled(false);

ui->play\_but->setEnabled(true);

player->pause();

}

void Player::stop()

{

ui->stop\_but->setEnabled(false);

ui->play\_but->setEnabled(true);

map.remove(playlist->currentMedia().request().url());

player->stop();

}

1. ЛИСТИНГ ВЫполнения программы

На рисунке 2 представлен интерфейс работы программы.

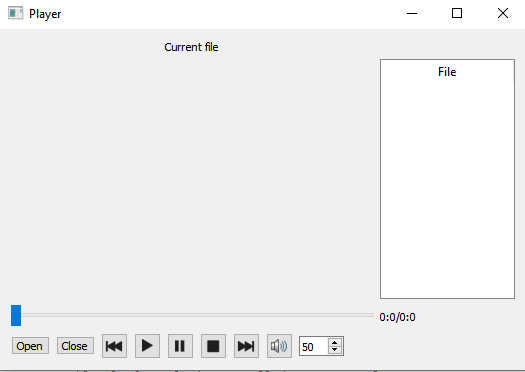


Рисунок 2 – Интерфейс программы

1. Вывод

В ходе выполнения лабораторной работы была написана программа, которая может воспроизводить аудио и видео файлы с плейлистом. В программе возможно останавливать, перематывать файлы. Есть возможность изменения громкости с помощью QSpinBox. При закрытии программы и повторном открытии воспроизведение ранее открытых файлов начинается с места последней остановки.