Министерство образования Республики Беларусь
Учреждение Образования
«Брестский Государственный Технический Университет»
Кафедра ИИТ

Лабораторная работа №5
По дисциплине ОСиСП за 5 семестр
Тема: Разработка приложения: «Игра Lines»

Выполнил:
Студент 3-го курса
Группы ПО-5
Крощук В.В.
Проверила:
Дряпко А.В.

Брест 2021

Лабораторная работа №5

Вариант 11

Цель работы:
Познакомиться с возможностями, предлагаемыми фреймворком Qt, для разработки многопоточных приложений.

Задания и выполненные решения:

Доработать программу, разработанную в лабораторной работе No1-4, внеся следующие изменения:

1) Основное задание заключается в доработке функционала обновления, разработка которого производилась в ЛР №4. Нужно интегрировать указанную функцию в само приложение, без использования стороннего клиента. При этом серверная часть приложения остается без изменений (возможны некоторые доработки сервера, без изменения общей клиент-серверной архитектуры);

2) Проверка обновления должна осуществляться автоматически по таймеру (QTimer) либо по непосредственному запросу пользователя. Предусмотреть выбор из меню политики обновления (с пользовательским подтверждением, без подтверждения/автоматически);

3) Сам процесс обновления должен осуществляться с использованием отдельного потока (QThread) с минимальной вовлечённостью пользователя;

4) Необходимо отображать прогресс обновления (для этого можно использовать строку состояния – QStatusBar);

5) Для демонстрации процесса обновления и независимой работы основного и вспомогательного потоков приложения осуществлять передачу с сервера обновления помимо основных обновляемых компонентов (в соответствии с вариантом задания) одного-двух крупных файлов с произвольным содержимым (например, видео).

6) Обновляемые компоненты по вариантам (ЛР №4):
11 вар. DLL, конфигурационный файл (внешний вид)

7) Процесс обновления логируется. При завершении обновления пользователю выдается соответствующее сообщение.

Код программы:

main.cpp:
```
#include "mainwindow.h"

#include <QApplication>

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    MainWindow w;
    w.startScreen();
    return a.exec();
}
```

mainwindow.cpp

```
#include "mainwindow.h"
#include "ui_mainwindow.h"
#include "about.h"
#include "helper.h"

typedef QString (*Helper_set_name_window)();
typedef QString (*Helper_set_first_name)();
typedef QString (*Helper_set_second_name)();

QImage createImageWithOverlay(const QImage& baseImage, const QImage& overlayImage);

MainWindow::MainWindow(QWidget *parent): QMainWindow(parent), ui(new Ui::MainWindow)
{
    ui->setupUi(this);

    // установка имени окна приложения
    QLibrary *helper_set = new QLibrary("Helper");
    Helper_set_name_window set_window_name_ = (Helper_set_name_window)helper_set-
>resolve("set_window_name");
    QString windowTitle = set_window_name_();
    delete helper_set;
    this->setWindowTitle(windowTitle);

    // установка имени текста 'играть'
    QLibrary *helper_set_f_n = new QLibrary("Helper");
    Helper_set_first_name set_window_name_first =
(Helper_set_first_name)helper_set_f_n->resolve("set_first_name_HeaderLabels");
    set_name_first = set_window_name_first();
    delete helper_set_f_n;


    // установка имени текста 'выход'
    QLibrary *helper_set_s_n = new QLibrary("Helper");
    Helper_set_second_name set_window_name_second =
(Helper_set_second_name)helper_set_s_n->resolve("set_second_name_HeaderLabels");
    set_name_second = set_window_name_second();
    delete helper_set_s_n;

    QIcon icon(":/Assets/ghost_green.png");
    this->setWindowIcon(icon);
    this->setFixedHeight(sWidth + 100);
    this->setFixedWidth(sWidth);
}

void MainWindow::closeEvent(QCloseEvent *event) {
    QMessageBox::StandardButton resBtn = QMessageBox::question( this, "Выход?",
                                                                tr("Вы уверены, что
хотите выйти?"),
                                                                QMessageBox::No |
QMessageBox::Yes,
                                                                QMessageBox::No);
    if (resBtn != QMessageBox::Yes) {
        event->ignore();
    } else {
        event->accept();
    }
}

void MainWindow::aboutButtonWasClicked(bool) {
    info();
}

void MainWindow::playButtonWasClicked(bool) {
    startGame();
}
```

```
void MainWindow::updateButtonWasClicked(bool) {
    checkUpdate();
}

void MainWindow::settingsButtonWasClicked(bool) {
    QMessageBox::StandardButton reply;
    reply = QMessageBox::question(this, "Выход?", "Вы уверены, что хотите выйти?",
                                  QMessageBox::Yes|QMessageBox::No, QMessageBox::No);
    if (reply == QMessageBox::Yes) {
        qDebug() << "Yes was clicked";
        QApplication::exit();
    } else {
        qDebug() << "Yes was *not* clicked";
    }

}

void MainWindow::homeButtonWasClicked(bool) {
    QMessageBox::StandardButton reply;
    reply = QMessageBox::question(this, "Назад", "Вы покинете игру. Уверены?",
                                  QMessageBox::Yes|QMessageBox::No);
    if (reply == QMessageBox::Yes) {
        qDebug() << "Yes was clicked";
        startScreen();
    } else {
        qDebug() << "Yes was *not* clicked";
    }
}

void MainWindow::buttonWasPressed(QWidget* buttonW) {
    QPushButton* button = (QPushButton*)(buttonW);
    button->setIconSize(QSize(button->iconSize().width() / 2, button-
>iconSize().height() / 2));
}

void MainWindow::buttonWasReleased(QWidget* buttonW) {
    QPushButton* button = (QPushButton*)(buttonW);
    button->setIconSize(QSize(button->iconSize().width() * 2, button-
>iconSize().height() * 2));
}

void MainWindow::startScreen() {
    GameLogic::m_pInstance = NULL;
    GameLogic::window = this;

    QFont newFont("Ink Free", 14, QFont::Bold, true);

    QWidget *parent = new QWidget();
    parent->resize(sWidth, sWidth);

    list = new QComboBox(parent);
    list->setFont(newFont);
    list->setStyleSheet("border: 0px; QColor(169, 169, 169)");
    setPlugins();

    list_theme = new QComboBox(parent);
    list_theme->move(200, 0);
    list_theme->setFont(newFont);
    list_theme->setStyleSheet("border: 0px; QColor(169, 169, 169)");
    set_interfase_plugins();

    QPushButton* playGame = new QPushButton(QIcon("://Assets/button_play.png"),
set_name_first, parent);
    playGame->move(100, 110);
    playGame->setStyleSheet("border: 0px; color: rgb(0, 0, 0)");
    playGame->setIconSize(QSize(200, 200));
    connect(playGame, SIGNAL(clicked(bool)), this, SLOT(playButtonWasClicked(bool)));
```

```
    QPushButton* update = new QPushButton("Обновить", parent);
    update->move(380, 110);
    update->setStyleSheet("border: 0px; color: rgb(0, 0, 0)");
    connect(update, SIGNAL(clicked(bool)), this, SLOT(updateButtonWasClicked(bool)));


    QPushButton* About = new QPushButton("О программе", parent);
    About->move(420, 310);
    About->setStyleSheet("border: 0px; color: rgb(0, 0, 0)");
    connect(About, SIGNAL(clicked(bool)), this, SLOT(aboutButtonWasClicked(bool)));


    QPushButton* settings = new QPushButton(QIcon("://Assets/button_home.png"),
set_name_second, parent);
    settings->move(200, 400);
    settings->setStyleSheet("border: 0px; color: rgb(0, 0, 0)");
    settings->setIconSize(QSize(200, 200));
    connect(settings, SIGNAL(clicked(bool)), this,
SLOT(settingsButtonWasClicked(bool)));

    this->setCentralWidget(parent);
    this->show();
}

void MainWindow::startGame() {
    board = new QWidget();
    cells = new Cell* *[boardRow];
    for(int i = 0; i < boardRow; i++) {
        cells[i] = new Cell*[boardColumn];
    }
    QGridLayout* layout = new QGridLayout();


    this->setCentralWidget(board);
    board->setLayout(layout);
    for(int i = 0; i < boardRow; i++)
        for(int j = 0; j < boardColumn; j++){
            Cell* temp = new Cell();
            temp->place = MatrixPoint(i,j);
            QImage cell;
            (i + j) % 2 == 0 ? cell = QImage("://Assets/cell_light.png") : cell =
QImage("://Assets/cell_dark.png");
            temp->setIcon(QIcon(QPixmap::fromImage(cell)));
            temp->setStyleSheet("border: 0px");
            temp->setIconSize(QSize(cellSize, cellSize));


            connect(temp, SIGNAL(wasPressed(MatrixPoint)), GameLogic::Instance(),
SLOT(cellWasPressed(MatrixPoint)));
            connect(temp, SIGNAL(clicked(bool)), temp, SLOT(idiotClick(bool)));

            temp->setMinimumSize(cellSize, cellSize);
            layout->addWidget(temp, i, j);
            cells[i][j] = temp;
        }

    int id = QFontDatabase::addApplicationFont("://Assets/ConcertOne-Regular.ttf");
    QString family = QFontDatabase::applicationFontFamilies(id).at(0);
    QFont monospace(family);
    monospace.setPointSize(30);

    QPushButton* score = new QPushButton(QIcon("://Assets/icon_path.png"), "0",
this);
    score->setFont(monospace);
    score->setStyleSheet("border: 0px; color: rgb(255, 255, 255)");
    score->setIconSize(QSize(100, 100));
```

```
    layout->addWidget(score, boardRow + 1, 0, 2, 5);
    this->score = score;

    QPushButton* home = new QPushButton(QIcon("://Assets/button_home.png"), "",
this);
    home ->setFont(monospace);
    home->setStyleSheet("border: 0px; color: rgb(255, 255, 255)");
    home->setIconSize(QSize(100, 100));

    connect(home, SIGNAL(clicked(bool)), this, SLOT(homeButtonWasClicked(bool)));

    layout->addWidget(home, boardRow + 1, 5, 2, 2);


    GameLogic::Instance()->generateGhosts();

    this->show();
}

void MainWindow::checkUpdate()
{
    socket = new QTcpSocket();
    connect(socket,SIGNAL(readyRead()),this,SLOT(sockReady()));

    Data.clear();

    QDir client_version(qApp->applicationDirPath());
    QStringList filter;
    filter << "*.json";
    foreach(QFileInfo info, client_version.entryInfoList(filter)) {
        filter.clear();
        filter << info.absoluteFilePath();
    }
    QFile file(filter.back());
    if (!file.open(QIODevice::ReadOnly))
        return;
    Data = file.readAll();
    socket->connectToHost("127.0.0.1", 5555);
    socket->waitForConnected(1);
    if(socket->state() == QTcpSocket::ConnectedState) {
        if(socket->isOpen()) {
            socket->write(Data);

            socket->waitForBytesWritten(100);
        }
    }
    else {
        QMessageBox::information(this, "Информация", "Соединение не установлено");
    }
}


void MainWindow::info()
{
    typedef void (*About)();
    QLibrary *aboutLib = new QLibrary("About");
    About showWindowAboutProgramm = (About)aboutLib->resolve("about");
    showWindowAboutProgramm();
    delete aboutLib;
}


void MainWindow::setPlugins()
{
    QDir dir(qApp->applicationDirPath());
    dir.cd("plugins");
    QStringList filter;
    filter << "*.dll";
```

```
    QPluginLoader loader;
    foreach(QFileInfo info, dir.entryInfoList(filter)) {
        loader.setFileName(info.absoluteFilePath());
        Interface* mode = qobject_cast<Interface*>(loader.instance());
        if(mode) {
            QString name =
loader.metaData().value("MetaData").toObject().value("Mode_name").toString();
            list->addItem(name);
            mPlugin.append(mode);
        }
    }
    connect(list, SIGNAL(activated(int)), this, SLOT(applyPlugin(int)));
}

void MainWindow::set_interfase_plugins()
{
    QDir dir(qApp->applicationDirPath());
    dir.cd("themes_interfases");
    QStringList filter;
    filter << "*.dll";
    QPluginLoader loader_l;
    foreach(QFileInfo info, dir.entryInfoList(filter)) {
        loader_l.setFileName(info.absoluteFilePath());
        Interface_theme* mode1 = qobject_cast<Interface_theme*>(loader_l.instance());
        if(mode1) {
            QString name =
loader_l.metaData().value("MetaData").toObject().value("interfas_name").toString();
            list_theme->addItem(name);
            mThemes.append(mode1);
        }
    }
    connect(list_theme, SIGNAL(activated(int)), this, SLOT(applyTheme(int)));
}

void MainWindow::ghostWasMoved(std::vector<Node> road, Ghosts type) {
    QSequentialAnimationGroup* animationManager = new QSequentialAnimationGroup();
    connect(animationManager, SIGNAL(finished()), this, SLOT(finishedAnimating()));
    lastCellPlace = new MatrixPoint(road.back().y, road.back().x);
    lastCellType = type;
    QImage cell;
    (road[0].y + road[0].x) % 2 == 0 ? cell = QImage("://Assets/cell_light.png") :
cell = QImage("://Assets/cell_dark.png");
    cells[road[0].y][road[0].x]->setIcon(QIcon(QPixmap::fromImage(cell)));
    for(unsigned int i = 1; i < road.size(); i++) {
        Node nextCell = road[i];
        Cell* current = cells[nextCell.y][nextCell.x];
        QPropertyAnimation *animation = new QPropertyAnimation(current, "iconSize");
        animation->setDuration(50);
        animation->setStartValue(QSize(0, 0));
        animation->setEndValue(QSize(cellSize, cellSize));

        animationManager->addAnimation(animation);
    }
    animationManager->start();
}

void MainWindow::finishedAnimating() {
    if(lastCellPlace != NULL) {
        cells[lastCellPlace->row][lastCellPlace->column]-
>setIcon(mergedIcon(lastCellType, *lastCellPlace));
        lastCellPlace = NULL;
        GameLogic::Instance()->nextMove();
    }
}
```

```
QIcon MainWindow::mergedIcon(Ghosts type, MatrixPoint place, bool select) {
    QImage icon;
    switch (type) {
    case yellow:
        icon = QImage(":/Assets/ghost_yellow.png");
        break;
    case white:
        icon = QImage(":/Assets/ghost_white.png");
        break;
    case green:
        icon = QImage(":/Assets/ghost_green.png");
        break;
    case red:
        icon = QImage(":/Assets/ghost_red.png");
        break;
    }

    QImage cellIcon;
    if(select)
        cellIcon = (place.row + place.column) % 2 == 0 ?
QImage(":/Assets/cell_light.png") : QImage(":/Assets/cell_dark.png");
    else
        cellIcon = QImage(":/Assets/cell_selected.png");
    QImage merged = createImageWithOverlay(cellIcon, icon);
    return QIcon(QPixmap::fromImage(merged));

}

void MainWindow::ghostWasGenerated(Ghosts type, MatrixPoint place){
    Cell* current = cells[place.row][place.column];

    current->setIcon(mergedIcon(type, place));

    QPropertyAnimation *animation = new
QPropertyAnimation(cells[place.row][place.column], "iconSize");
    animation->setDuration(100);
    animation->setStartValue(QSize(0, 0));
    animation->setEndValue(QSize(cellSize, cellSize));
    animation->start();
}

void MainWindow::ghostWasDeleted(MatrixPoint place) {
    Cell* temp = cells[place.row][place.column];
    if((place.row + place.column) % 2 == 0)
        temp->setIcon(QIcon(":/Assets/cell_light.png"));
    else
        temp->setIcon(QIcon(":/Assets/cell_dark.png"));
    temp->setIconSize(QSize(cellSize, cellSize));
}

void MainWindow::ghostWasSelected(Ghosts type, MatrixPoint place) {
    if(GameLogic::Instance()->gameBoard[place.row][place.column])
        cells[place.row][place.column]->setIcon(mergedIcon(type, place, false));
}

void MainWindow::ghostWasDeselected(Ghosts type, MatrixPoint place) {
    if(GameLogic::Instance()->gameBoard[place.row][place.column])
        cells[place.row][place.column]->setIcon(mergedIcon(type, place, true));
}

void MainWindow::gameOver() {
    GameLogic::m_pInstance  = NULL;
    QMessageBox::StandardButton reply;
    reply = QMessageBox::question(this, "Game Over", "You have lost. Do you want to
try again?",
                                  QMessageBox::Yes|QMessageBox::No);
    if (reply == QMessageBox::Yes) {
        startGame();
```

```
    } else {
        startScreen();
    }
}

QImage createImageWithOverlay(const QImage& baseImage, const QImage& overlayImage)
{
    QImage imageWithOverlay = QImage(overlayImage.size(),
QImage::Format_ARGB32_Premultiplied);
    QPainter painter(&imageWithOverlay);

    painter.setCompositionMode(QPainter::CompositionMode_Source);
    painter.fillRect(imageWithOverlay.rect(), Qt::transparent);

    painter.setCompositionMode(QPainter::CompositionMode_SourceOver);
    painter.drawImage(0, 0, baseImage);

    painter.setCompositionMode(QPainter::CompositionMode_SourceOver);
    painter.drawImage(0, 0, overlayImage);

    painter.end();

    return imageWithOverlay;
}


MainWindow::~MainWindow()
{
    delete ui;
}

void MainWindow::sockDisc()
{
    socket->disconnected();
}

void MainWindow::sockReady()
{
    if(socket->waitForConnected(100)) {
        socket->waitForReadyRead(100);
        Data = socket->readAll();

        QDir dir_client(QDir::currentPath() + "/themes_interfases");

        if(Data == "Actuale") {
            QMessageBox::information(this, "Информация", "Соединение установлено\nУ
вас актуальная версия программы!");
            socket->disconnected();
        }

    else if(Data == "Need update") {
            QMessageBox msg;
            msg.setText("New version is available");
            msg.setInformativeText("Do you want to update app?");
            msg.setStandardButtons(QMessageBox::Yes | QMessageBox::No);
            msg.setDefaultButton(QMessageBox::Yes);
            int res = msg.exec();

            //loading

            if (res == QMessageBox::Yes) {
                sockDisc();
                socket->waitForDisconnected(1);

                QThread *thread= new QThread;
                Thread *my = new Thread("B");
```

```
                my->moveToThread(thread);

                connect(my, SIGNAL(time_load(int)), this, SLOT(showProgress(int)));
                connect(my, SIGNAL(load_update(int)), this, SLOT(loading(int)));
                connect(my, SIGNAL(load_video(int)), this,
SLOT(download_video(int)));

                connect(thread, SIGNAL(started()), my, SLOT(run()));

                thread->start();
            }
        }
    }
}

void MainWindow::applyPlugin(int index)
{
    QFont palette = this->mPlugin[index]->changeView();
    QApplication::setFont(palette);
    startScreen();
}

void MainWindow::applyTheme(int index)
{
    QPalette palette = this->mThemes[index]->changeInterfase();
    QApplication::setPalette(palette);
    startScreen();
}

void MainWindow::loading(int res) {
    if(res == 1) {
        mThemes.clear();
        list_theme->clear();
        set_interfase_plugins();
    }
}

void MainWindow::showProgress(int i) {
    progress = i;
    ui->statusbar->showMessage(QString::number(progress)+"%");
    ui->statusbar->showMessage("100%");
    QMessageBox::information(this, "Информация", "Обновление завершено!");
}

void MainWindow::download_video(int msg) {
    if(msg == 1) QMessageBox::information(this, "Информация", "Видео получено!");
}
```

Thread.h

```
#ifndef THREAD_H
#define THREAD_H

#include <QTcpSocket>
#include <QThread>
#include <QTimer>
#include <QDir>
#include <QFile>


class Thread : public QObject
{
    Q_OBJECT
public:
    explicit Thread(QString name);
```

```
    QTcpSocket* socket;
    QByteArray Data;
    int progress = 0;
    QTimer *timer;


public slots:
    void sockReady();
    void sockDisc();
    void run();
    void share_video();

signals:
    void time_load(int);
    void load_update(int);
    void load_video(int);

private:
    QString name;
    QString path_server;
};


#endif // THREAD_H
```

Thread.cpp

```
#include "thread.h"

QDir dir_client(QDir::currentPath() + "/themes_interfases");

Thread::Thread(QString s) : name(s) {

}

void Thread::sockReady() {
    if(socket->waitForConnected(100)) {
        socket->waitForReadyRead(100);
        Data = socket->readAll();

        QStringList rec_data_update;
        QString new_version;

        rec_data_update.append(QString(Data).split(" "));
        new_version.append(rec_data_update.last());

        path_server = rec_data_update.front();

        QDir new_client_version(QDir::current());
        QStringList filter;
        filter << "*.json";
        foreach(QFileInfo info, new_client_version.entryInfoList(filter)) {
            filter.clear();
            filter << info.absoluteFilePath();
        }

        QFile file(filter.back());
        if (!file.open(QIODevice::WriteOnly))
            return;
        file.write(new_version.toStdString().data());

        for(int i = 1; i < rec_data_update.size()-1; i++) {
            QFile::copy(rec_data_update.front()+'/'+rec_data_update[i],
dir_client.path()+'/'+rec_data_update[i]);
        }
```

```
        //create log

        QDir write_log(QDir::currentPath() + "/logs");
        QStringList formatFile;
        formatFile << "*.txt";
        foreach(QFileInfo info, write_log.entryInfoList(formatFile)) {
            formatFile.clear();
            formatFile << info.absoluteFilePath();
        }

        QFile log(formatFile.front());
        if (!log.open(QIODevice::WriteOnly))
            return;
        QString text = "Update Modules and version "+ rec_data_update.back();
        log.write(text.toStdString().data());
    }
//    sockDisc();
    emit load_update(1);
    share_video();
}

void Thread::sockDisc() {
    socket->disconnected();
}

void Thread::run() {
    socket = new QTcpSocket();
    connect(socket,SIGNAL(readyRead()),this,SLOT(sockReady()));
    socket->connectToHost("127.0.0.1", 5555);

    Data.clear();
    QStringList find_filter;

    bool ok = dir_client.exists();
    if (ok) {
        dir_client.setFilter(QDir::Files | QDir::Hidden | QDir::NoSymLinks);
        dir_client.setSorting(QDir::Name);
        QFileInfoList list = dir_client.entryInfoList();

        for (int i = 0; i < list.size(); ++i) {
            QFileInfo fileInfo = list.at(i);
            find_filter.append(fileInfo.fileName());
        }
        qDebug() << find_filter << endl;
    }
    foreach (const QString &str, find_filter) {
        Data.append(str);
        if(&str != find_filter.last()) {
            Data.append(" ");
        }
    }
    qDebug() << Data << endl; //get list themes_interfases client


    socket->write(Data);
    socket->waitForBytesWritten(100);

    emit time_load(58);
}

void Thread::share_video() {


if(QFile::copy(path_server+"/video/DCs.Legends.of.Tomorrow.S06E11.1080p.rus.LostFilm.
TV.mkv", dir_client.path() +
"/video/DCs.Legends.of.Tomorrow.S06E11.1080p.rus.LostFilm.TV.mkv")) emit
load_video(1);
}
```

Mode.h

```
#ifndef MODE_H
#define MODE_H

#include <QObject>
#include <interface.h>

class Mode : public QObject, public Interface
{
    Q_OBJECT
    Q_PLUGIN_METADATA(IID "lines.Interface" FILE "Interface.json")
    Q_INTERFACES(Interface)
public:
    Mode(QObject *parent =0);
    ~Mode();
    QString name();
    virtual QFont changeView();
};

#endif // MODE_H
```

Mode.cpp

```
#include "mode.h"
#include <QDebug>

Mode::Mode(QObject *parent) : QObject(parent)
{
    qDebug() << name() << "created";
}

Mode::~Mode()
{
    qDebug() << name() << "destroy";
}

QString Mode::name()
{
    return "mode 1";
}

QFont Mode::changeView()
{
    qDebug() << name() << "print";
    QFont newFont("Ink Free", 9, QFont::Bold, true);
    return newFont;
}
```

Interface.h

```
#ifndef INTERFACE_H
#define INTERFACE_H

#include <QObject>
#include <QString>
#include <QFont>

class Interface
{
public:
    virtual QFont changeView() = 0;//плагин выполняет действия над объектами в
структуре Styles
};

Q_DECLARE_INTERFACE(Interface, "lines.Interface");

#endif // INTERFACE_H
```

Helper.h

```
#ifndef HELPER_H
#define HELPER_H

#include "helper_global.h"
#include <QString>

extern "C" HELPER_EXPORT QString set_window_name();
extern "C" HELPER_EXPORT QString set_first_name_HeaderLabels();
extern "C" HELPER_EXPORT QString set_second_name_HeaderLabels();

#endif // HELPER_H
```

Helper.cpp

```
#include "helper.h"

QString set_window_name() {
    return QString("Line Balls");
}

QString set_first_name_HeaderLabels() {
    return QString("Game");
}

QString set_second_name_HeaderLabels() {
    return QString("Quit the game");
}
```

About.h

```
#ifndef ABOUT_H
#define ABOUT_H

#include "about_global.h"
extern "C" ABOUT_EXPORT void about();
#endif // ABOUT_H
```

About.cpp

```
#include "about.h"
#include <QMessageBox>

void about()
{
    QMessageBox msgBox;
    msgBox.setText("Программу выполнил студент группы ПО-5 Крощук Виктор \nДанная
программа - игра");
    msgBox.exec();
}
```

Server

Main.cpp

```
#include <QCoreApplication>
#include <myserver.h>

int main(int argc, char *argv[])
{
    QCoreApplication a(argc, argv);

    MyServer Server;
    Server.StartServer();

    return a.exec();
}
```

Myserver,cpp

```
#include "myserver.h"

MyServer::MyServer(){}
MyServer::~MyServer(){}

void MyServer::StartServer(){
    if(this->listen(QHostAddress::Any, 5555)) {
       qDebug() << "Listening";
    }
    else {
        qDebug() << "Not Listening";
    }
}

void MyServer::incomingConnection(int socketDeskriptor) {
    socket = new QTcpSocket(this);
    socket->setSocketDescriptor(socketDeskriptor);
    connect(socket,SIGNAL(readyRead()), this, SLOT(sockReady()));
    connect(socket,SIGNAL(disconnected()),this,SLOT(sockDisc()));
    qDebug()<<socketDeskriptor<<"Client connected";
    qDebug()<<"Send client connect status - YES";
}

void MyServer::sockReady() {
    Data = socket->readAll();
    qDebug() << "Select from Client" << Data;


    if(!Data.isEmpty()) {

        QDir server_version(QDir::currentPath());
        QStringList filter;
        filter << "*.json";
        foreach(QFileInfo info, server_version.entryInfoList(filter)) {
            filter.clear();
            filter << info.absoluteFilePath();
        }
        qDebug() << filter;

        QFile file(filter.back());
        if (!file.open(QIODevice::ReadOnly))
            return;
        path_to_Download = file.readAll();
        if(QString(Data) == path_to_Download) {
            qDebug() << "Send to Client" << "Actuale";
            socket->write("Actuale");
        }

        else if(Data[0] == 'T') {
            QStringList client_data;
            client_data.append(QString(Data).split(" "));
            Data.clear();
            QDir dir_server(QDir::currentPath() + "/themes_interfases");
            QStringList find_filter;

            Data.append(dir_server.path()+' ');

            bool ok = dir_server.exists();
            if (ok)
            {
                dir_server.setFilter(QDir::Files | QDir::Hidden | QDir::NoSymLinks);
                dir_server.setSorting(QDir::Name);
                QFileInfoList list = dir_server.entryInfoList();

                for (int i = 0; i < list.size(); ++i)
                {
```

```
                    QFileInfo fileInfo = list.at(i);
                    find_filter.append(fileInfo.fileName());
                }
            }
             QStringList sen_to_clien;
             for(int i = client_data.size(); i < find_filter.size(); i++) {
                 sen_to_clien.append(find_filter[i]);
             }
            foreach (const QString &str, sen_to_clien)
            {
                Data.append(str);
                if(&str != find_filter.last()) {
                    Data.append(" ");
                }
                else {
                    Data.append(" ");
                    Data.append(path_to_Download);

                }
            }
            qDebug() << "Send to Client" << Data;
            socket->write(Data);
        }

        else {
            qDebug() << "Send to Client" << "Need update";
            socket->write("Need update");
        }
        socket->waitForBytesWritten(100);
    }
}

void MyServer::sockDisc(){
    qDebug()<<"Disconnect";
    socket->deleteLater();
}
```

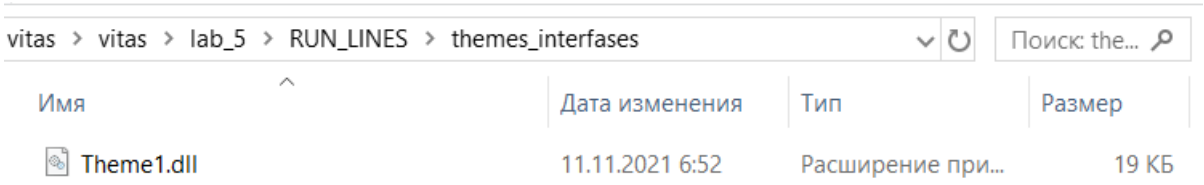
Myserver.h

```
#ifndef MYSERVER_H
#define MYSERVER_H

#include <QTcpServer>
#include <QTcpSocket>
#include <QDir>
#include <QFile>

class MyServer: public QTcpServer{
    Q_OBJECT
public:
    MyServer();
    ~MyServer();
    QTcpSocket * socket;
    QByteArray Data;
    QString path_to_Download;

public slots:
    void StartServer();
    void incomingConnection(int socketDeskriptor);
    void sockReady();
    void sockDisc();
};


#endif // MYSERVER_H
```

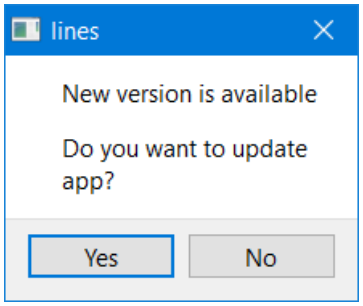
Работа программы:

Изначально до обновления программы, мы подключаем сервер. И после запуска программы посмотрим какие темы внешнего вида существуют:
Как мы видим, в данном окне одна тема.

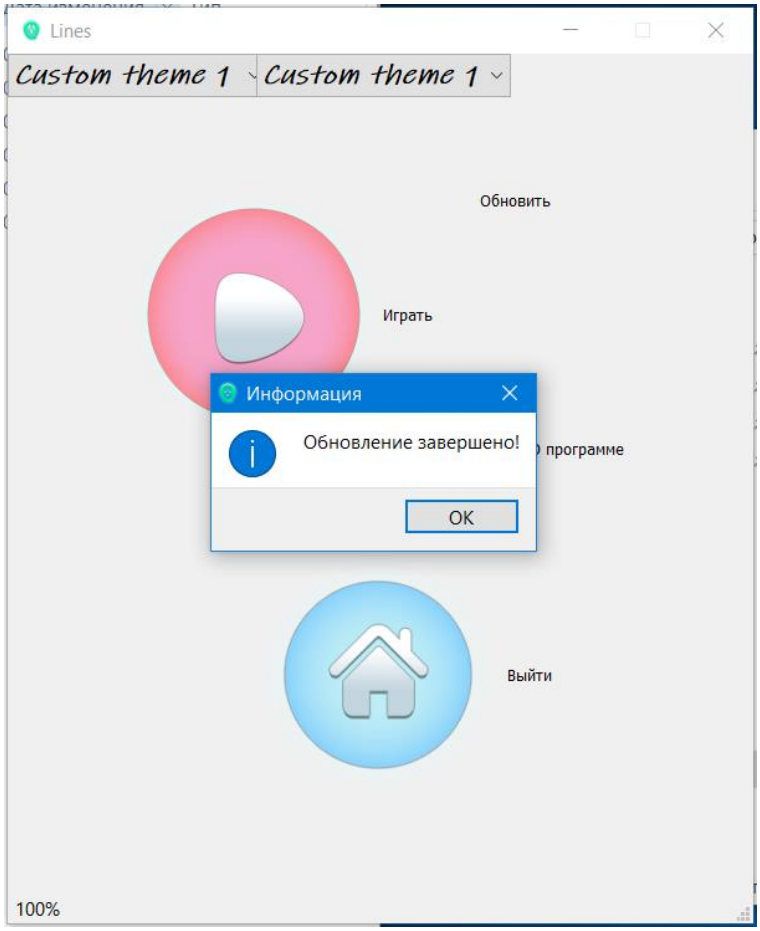
В папке тем мы также увидим, что находится одна тема.



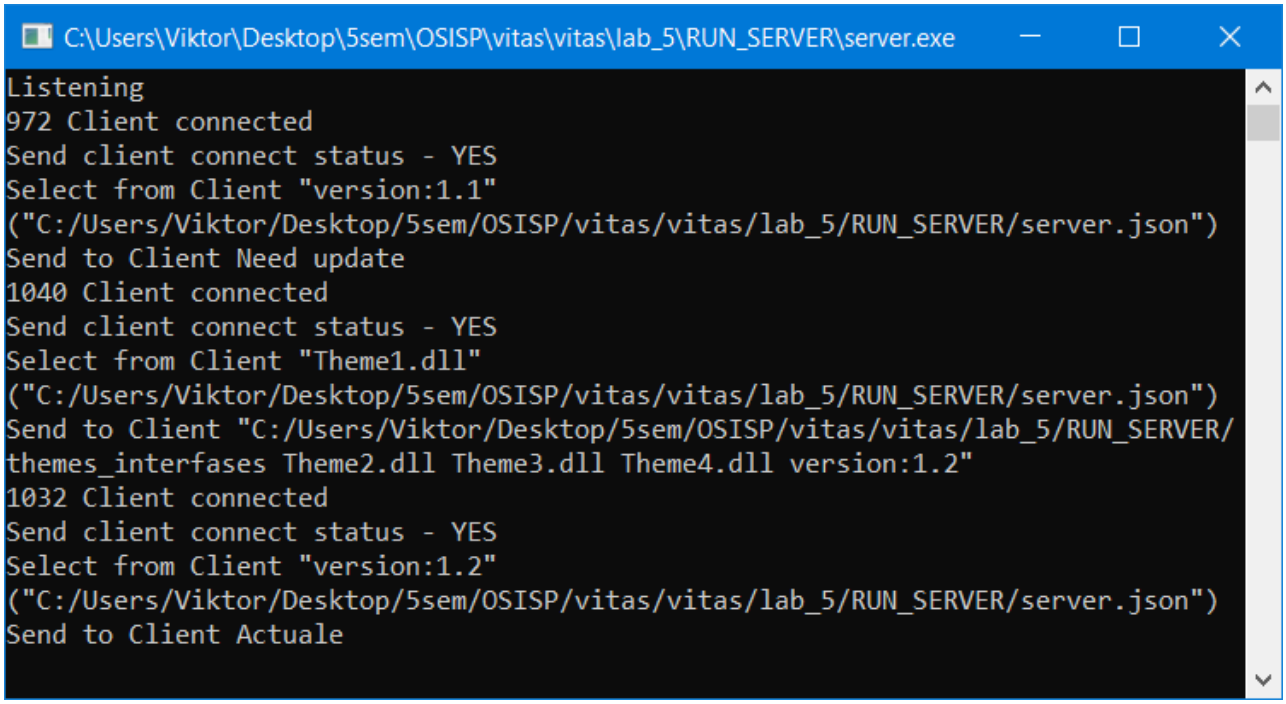
После нажатия кнопки «Обновить» запрашивается запрос выполнять ли обновление:



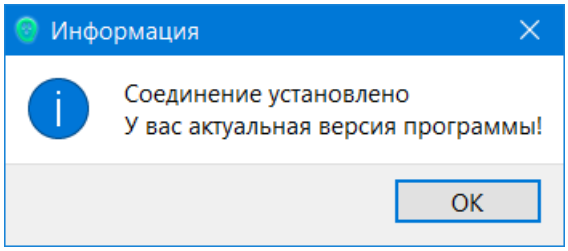
После нажатия кнопки «Обновить» производится обновление и выводится соответствующее сообщение, а также в процессе обновления в правом нижнем углу можем увидеть прогресс:



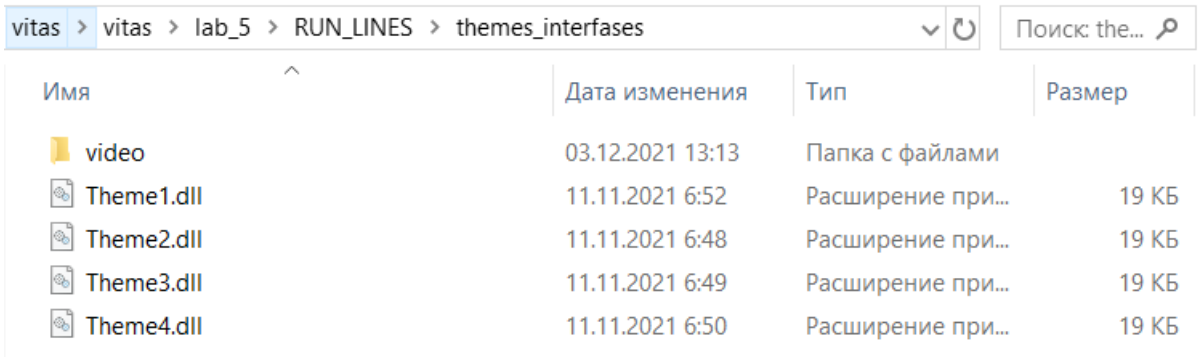
В данный момент обновления происходит общение программы с сервером и выводятся соответствующие сообщения, а также в здесь мы также увидим, что версия обновилась до «version: 1.2»:



При повторном нажатии на кнопку «Обновить», мы получаем сообщение о том, что приложение имеет актуальную версию:



А после получения завершения мы видим, что добавились новые темы в файл, а также папка с видео, с вложенным видео, требуемым по заданию:



Вывод: познакомился с возможностями, предлагаемыми фреймворком Qt, для разработки многопоточных приложений