МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ

 «БРЕСТСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Кафедра ИИТ

ЛАБОРАТОРНАЯ РАБОТА №6

По дисциплине: «ОСИСП»

Выполнил:

Студент ФЭИС

3-го курса, группы ПО-5

Брич М.Н.

Проверила:

Дряпко А.В.

Брест 2021

**Цель работы:** познакомиться с возможностями, предлагаемыми фреймворком Qt, для разработки многопо-

точных приложений

**Вариант 3**

*Задание:*

1) Основное задание заключается в доработке функционала обновления, разработка которого про-

изводилась в ЛР No4. Нужно интегрировать указанную функцию в само приложение, без ис-

пользования стороннего клиента. При этом серверная часть приложения остается без изменений

(возможны некоторые доработки сервера, без изменения общей клиент-серверной архитекту-

ры);

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 обнов-

ляемых компонентов (в соответствии с вариантом задания) одного-двух крупных файлов с

произвольным содержимым (например, видео).

6) Обновляемые компоненты по вариантам (ЛР No4):

3 DLL, новый уровень

***snake.h:***

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#include <QMainWindow>

#include <QPaintEvent>

#include <QKeyEvent>

#include <QDialog>

#include <QComboBox>

#include "interface.h"

const int BLOCK\_SIZE=15;

const int MARGIN=22;

const int AREA\_ROW=30;

const int AREA\_COL=30;

const int TIME\_INTERVAL=100;

enum Direction

{

UP,

DOWN,

LEFT,

RIGHT

};

QT\_BEGIN\_NAMESPACE

namespace Ui { class MainWindow; }

QT\_END\_NAMESPACE

class MainWindow : public QMainWindow

{

Q\_OBJECT

public:

MainWindow(QWidget \*parent = nullptr);

~MainWindow();

virtual void paintEvent(QPaintEvent \*event);

virtual void keyPressEvent(QKeyEvent \*event);

QDialog dlg;

QComboBox\* modeMenu = new QComboBox;

public:

void InitMenu();

void SetPlugins();

void InitGame();

void InitPause();

void PauseResumeGame();

void GameOver();

void GenerateFood();

void GenerateSnake();

bool IsGameOver();

private slots:

void SnakeUpdate();

void aboutQt();

void applyPlugin(int index);

void BtnDlgClick1();

void BtnDlgClick2();

void BtnDlgClick3();

void LoadingLevel();

void CheckUpdate();

private:

Ui::MainWindow\*ui;

QTimer \*gameTimer;

bool isPause;

QPoint foodPoint;

QList<QPoint> wall;

QList<QPoint> snake;

Direction dir;

int score;

int level;

QVector< Interface\* > mPlugins;

};

#endif // MAINWINDOW\_H

***menu.cpp:***

#include <QPainter>

#include <time.h>

#include <QTimer>

#include <QString>

#include <QtWidgets>

#include <QMessageBox>

#include "snake.h"

#include "ui\_snake.h"

typedef void (\*about)();

typedef bool (\*GenerateWall)(int level, int i, int j);

typedef int (\*countLevel)();

void MainWindow::SetPlugins()

{

QDir \*dir;

#ifdef QT\_DEBUG

dir = new QDir("/home/eve9te/bstu/osisp2.0/Snake/plugins/debug");

#else

#ifdef QT\_RELEASE

dir = new QDir("/home/eve9te/bstu/osisp2.0/Snake/plugins/release");

#endif

#endif

if(dir->entryList(QDir::Files).isEmpty())

qDebug() << "dir isEmpty";

foreach(QString str, dir->entryList(QDir::Files))

{

QPluginLoader loader(dir->absoluteFilePath(str));

QObject \*object = qobject\_cast<QObject\*>(loader.instance());//извлекаем плагин

Interface \*plugin = qobject\_cast<Interface\*>(object);//приводим к интерфейсу игры

if(plugin)

{

this->mPlugins.push\_back(plugin);

// const char \* namePlugin = plugin->pluginName().toLocal8Bit().data();

// QAction \*applyPlugin = new QAction(tr(namePlugin), this);

modeMenu->addItem(plugin->pluginName());

connect(modeMenu, SIGNAL(activated(int)), this, SLOT(applyPlugin(int)));

}

else

{

qDebug() << "error";

}

}

delete dir;

}

void MainWindow::applyPlugin(int index)

{

QFont palette = this->mPlugins[index]->changeView();

QApplication::setFont(palette);

}

void MainWindow::InitMenu()

{

dlg.setWindowTitle(tr("Выбирите уровень"));

dlg.setFixedSize(QSize(MARGIN\*7+(AREA\_COL+1)\*BLOCK\_SIZE,MARGIN\*2+AREA\_ROW\*BLOCK\_SIZE));

dlg.setGeometry(

QStyle::alignedRect(

Qt::LeftToRight,

Qt::AlignCenter,

dlg.size(),

QApplication::desktop()->availableGeometry(this)

)

);

QBoxLayout\* layout = new QVBoxLayout;

layout->setAlignment(Qt::AlignVCenter);

QMenuBar\* menuBar = new QMenuBar();

QMenu\* menu = new QMenu("О программе");

menuBar->addMenu(menu);

menu->addAction("about.dll", this,SLOT(aboutQt()),

Qt::CTRL + Qt::Key\_Q);

layout->setMenuBar(menuBar);

QPushButton\* btn\_easy = new QPushButton("Easy");

QPushButton\* btn\_medium = new QPushButton("Medium");

QPushButton\* btn\_hard = new QPushButton("Hard");

QPushButton\* btn\_update = new QPushButton("Update");

btn\_update->setContentsMargins(QMargins(0,30,0,0));

QPushButton\* btn\_exit = new QPushButton("Exit");

// btn\_exit->setStyleSheet("margin-top: 5px;"

// "padding-top: 3px; padding-bottom: 3px;");

connect(btn\_easy, SIGNAL(clicked()), this, SLOT(BtnDlgClick1()));

connect(btn\_medium, SIGNAL(clicked()), this, SLOT(BtnDlgClick2()));

connect(btn\_hard, SIGNAL(clicked()), this, SLOT(BtnDlgClick3()));

connect(btn\_update, SIGNAL(clicked()), this, SLOT(CheckUpdate()));

connect(btn\_exit, SIGNAL(clicked()), &dlg, SLOT(close()));

layout->addWidget(btn\_easy);

layout->addWidget(btn\_medium);

layout->addWidget(btn\_hard);

layout->addWidget(btn\_update);

layout->addWidget(btn\_exit);

layout->addWidget(modeMenu);

dlg.setLayout(layout);

if(dlg.exec() == QDialog::Accepted) {

InitGame();

}

else

{

exit(0);

}

}

void MainWindow::BtnDlgClick1()

{

level = 1;

LoadingLevel();

}

void MainWindow::BtnDlgClick2()

{

level = 2;

LoadingLevel();

}

void MainWindow::BtnDlgClick3()

{

level = 3;

LoadingLevel();

}

void MainWindow::LoadingLevel()

{

QLibrary \*helperLib = new QLibrary("helper");

if (!helperLib->load())

qDebug() << helperLib->errorString();

if (helperLib->load())

qDebug() << "library loaded";

countLevel countLevel\_ = (countLevel) helperLib->resolve("countLevel");

if(level > countLevel\_())

{

QMessageBox msgBox;

msgBox.setWindowTitle("Error");

msgBox.setText("Уровень еще не добавлен");

msgBox.exec();

return;

}

GenerateWall generateWall\_ = (GenerateWall)helperLib->resolve("GenerateWall");

for (int i = 0; i < AREA\_COL; ++i) {

for (int j = 0; j < AREA\_ROW; ++j) {

if(generateWall\_(level, i, j)){

wall.push\_back(QPoint(i, j));

}

}

}

delete helperLib;

dlg.accept();

}

void MainWindow::CheckUpdate()

{

QMessageBox msgBox;

msgBox.setWindowTitle("Update");

msgBox.setText("This is latest version");

msgBox.exec();

}

void MainWindow::aboutQt()

{

QLibrary \*aboutLib = new QLibrary("about");

if (!aboutLib->load())

qDebug() << aboutLib->errorString();

if (aboutLib->load())

qDebug() << "library loaded";

about showWindowAboutProgramm = (about)aboutLib->resolve("about");

showWindowAboutProgramm();

delete aboutLib;

}

MainWindow::~MainWindow()

{

delete ui;

}

***snake.cpp***

#include "snake.h"

#include "ui\_snake.h"

Snake::Snake(QWidget \*parent): QMainWindow(parent), ui(new Ui::Snake)

{

typedef QString (\*Helper\_set\_name\_window)();

typedef QString (\*Helper\_set\_name\_about)();

ui->setupUi(this);

setFixedSize(QSize(MARGIN\*7+(AREA\_COL+1)\*BLOCK\_SIZE,MARGIN\*2+AREA\_ROW\*BLOCK\_SIZE));

SetPlugins();

Setlevels();

// 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 About

QLibrary \*helper\_set\_n\_about = new QLibrary("Helper");

Helper\_set\_name\_about set\_window\_name\_first = (Helper\_set\_name\_about)helper\_set\_n\_about->resolve("set\_name\_About");

QMenu\* menu = new QMenu(set\_window\_name\_first());

delete helper\_set\_n\_about;

ui->menubar->addMenu(menu);

menu->addAction("about.dll", this,SLOT(aboutQt()), Qt::CTRL + Qt::Key\_Q);

}

void Snake::InitMenu() {

typedef QString (\*Helper\_set\_name\_window)();

typedef QString (\*Helper\_set\_name\_about)();

ui->setupUi(this);

levels.clear();

setFixedSize(QSize(MARGIN\*7+(AREA\_COL+1)\*BLOCK\_SIZE,MARGIN\*2+AREA\_ROW\*BLOCK\_SIZE));

connect(ui->levelslist, SIGNAL(currentIndexChanged(const QString)), this, SLOT(on\_levelslist\_currentIndexChanged(QString)));

connect(ui->btn\_start, SIGNAL(clicked()), this, SLOT(on\_btn\_start\_clicked()));

connect(ui->btn\_update, SIGNAL(clicked()), this, SLOT(on\_btn\_update\_clicked()));

connect(ui->btn\_exit, SIGNAL(clicked()), this, SLOT(on\_btn\_exit\_clicked()));

SetPlugins();

Setlevels();

// 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 About

QLibrary \*helper\_set\_n\_about = new QLibrary("Helper");

Helper\_set\_name\_about set\_window\_name\_first = (Helper\_set\_name\_about)helper\_set\_n\_about->resolve("set\_name\_About");

QMenu\* menu = new QMenu(set\_window\_name\_first());

delete helper\_set\_n\_about;

ui->menubar->addMenu(menu);

menu->addAction("about.dll", this, SLOT(aboutQt()), Qt::CTRL + Qt::Key\_Q);

}

void Snake::PauseResumeGame()

{

if(!isPause)

{

isPause=!isPause;

gameTimer->stop();

InitPause();

}

else

{

isPause=!isPause;

gameTimer->start(TIME\_INTERVAL);

}

}

void Snake::InitPause()

{

QMessageBox::StandardButton pause;

pause = QMessageBox::question(this, "Pause", "Продолжить?",

QMessageBox::Yes|QMessageBox::No);

if (pause == QMessageBox::Yes) {

PauseResumeGame();

}

else {

flag\_game = false;

InitMenu();

}

}

void Snake::GameOver()

{

gameTimer->stop();

snake.pop\_front();

QMessageBox::StandardButton toMain;

toMain = QMessageBox::question(this,"Failed","Game over!", QMessageBox::Ok|QMessageBox::Ok);

if(toMain == QMessageBox::Ok) {

flag\_game = false;

InitMenu();

}

}

void Snake::sockDisc() {

socket->disconnected();

}

void Snake::sockReady() {

if(socket->waitForConnected(100)) {

socket->waitForReadyRead(100);

Data = socket->readAll();

QDir dir\_client(QDir::currentPath() + "/level");

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();

}

}

}

}

***game.cpp***

#include <QPainter>

#include <time.h>

#include <QTimer>

#include <QString>

#include <QtWidgets>

#include <QMessageBox>

#include "snake.h"

#include "ui\_snake.h"

typedef bool (\*Generate\_Snake)(int i);

void MainWindow::InitGame()

{

snake.clear();

GenerateSnake();

dir=RIGHT;

srand(time(0));

GenerateFood();

score=0;

isPause=false;

gameTimer=new QTimer(this);

connect(gameTimer, SIGNAL(timeout()), this, SLOT(SnakeUpdate()));

gameTimer->start(TIME\_INTERVAL);

}

void MainWindow::GenerateSnake()

{

QLibrary \*helperLib = new QLibrary("helper");

if (!helperLib->load())

qDebug() << helperLib->errorString();

if (helperLib->load())

qDebug() << "library loaded";

Generate\_Snake generateSnake\_ = (Generate\_Snake)helperLib->resolve("GenerateSnake");

for (int i=AREA\_COL;i>=0;i--) {

if(generateSnake\_(i)){

snake.push\_back(QPoint(i, 0));

}

}

delete helperLib;

}

void MainWindow::paintEvent(QPaintEvent \*event)

{

Q\_UNUSED(event);

QPainter painter(this);

painter.setBrush(Qt::yellow);

painter.setPen(Qt::blue);

painter.drawRect(MARGIN,MARGIN,AREA\_COL\*BLOCK\_SIZE,AREA\_ROW\*BLOCK\_SIZE);

painter.setBrush(Qt::blue);

painter.setPen(Qt::black);

for(int i=0;i<wall.size();i++)

painter.drawRect(MARGIN+wall[i].x()\*BLOCK\_SIZE,MARGIN+wall[i].y()\*BLOCK\_SIZE,BLOCK\_SIZE,BLOCK\_SIZE);

painter.setBrush(Qt::red);

painter.setPen(Qt::green);

for(int i=0;i<snake.size();i++)

painter.drawRect(MARGIN+snake[i].x()\*BLOCK\_SIZE,MARGIN+snake[i].y()\*BLOCK\_SIZE,BLOCK\_SIZE,BLOCK\_SIZE);

painter.setBrush(Qt::green);

painter.drawEllipse(MARGIN+foodPoint.x()\*BLOCK\_SIZE,MARGIN+foodPoint.y()\*BLOCK\_SIZE,BLOCK\_SIZE,BLOCK\_SIZE);

painter.setPen(Qt::black);

painter.setFont(QFont("Arial",14));

painter.drawText(MARGIN\*3+AREA\_COL\*BLOCK\_SIZE,MARGIN+2\*BLOCK\_SIZE,"score: "+QString::number(score));

}

void MainWindow::keyPressEvent(QKeyEvent \*event)

{

switch(event->key())

{

case Qt::Key\_Up:

if(dir!=DOWN)

dir=UP;

break;

case Qt::Key\_Down:

if(dir!=UP)

dir=DOWN;

break;

case Qt::Key\_Left:

if(dir!=RIGHT)

dir=LEFT;

break;

case Qt::Key\_Right:

if(dir!=LEFT)

dir=RIGHT;

break;

case Qt::Key\_P:

PauseResumeGame();

break;

case Qt::Key\_Escape:

PauseResumeGame();

break;

default:

break;

}

}

bool MainWindow::IsGameOver()

{

int x=snake.front().x();

int y=snake.front().y();

if(x<0||x>AREA\_COL-1||y<0||y>AREA\_ROW-1)

return true;

for(int i=1;i<snake.size();i++)

if(snake[i]==snake.front())

return true;

for(int i=0;i<wall.size();i++)

if(wall[i]==snake.front())

return true;

return false;

}

void MainWindow::SnakeUpdate()

{

switch(dir)

{

case UP:

snake.push\_front(QPoint(snake.front().x(),snake.front().y()-1));

break;

case DOWN:

snake.push\_front(QPoint(snake.front().x(),snake.front().y()+1));

break;

case LEFT:

snake.push\_front(QPoint(snake.front().x()-1,snake.front().y()));

break;

case RIGHT:

snake.push\_front(QPoint(snake.front().x()+1,snake.front().y()));

break;

default:

break;

}

if(snake.contains(foodPoint))

{

score+=1;

GenerateFood();

gameTimer->start(TIME\_INTERVAL - score);

}

else

snake.pop\_back();

if(IsGameOver())

{

GameOver();

return;

}

update();

}

***food.cpp***

*#include <QPainter>*

*#include <time.h>*

*#include <QTimer>*

*#include <QString>*

*#include <QtWidgets>*

*#include <QMessageBox>*

*#include "snake.h"*

*#include "ui\_snake.h"*

*void MainWindow::GenerateFood()*

*{*

*foodPoint.setX(rand()%AREA\_COL);*

*foodPoint.setY(rand()%AREA\_ROW);*

*if(snake.contains(foodPoint) || wall.contains(foodPoint))*

*GenerateFood();*

*}*

***myserver.cpp***

*#include "thread.h"*

*QDir dir\_client(QDir::currentPath() + "/level");*

*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;*

*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);*

*}"*

***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] == 'l') {*

*QStringList client\_data;*

*client\_data.append(QString(Data).split(" "));*

*Data.clear();*

*QDir dir\_server(QDir::currentPath() + "/level");*

*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);*

*}*

*}*

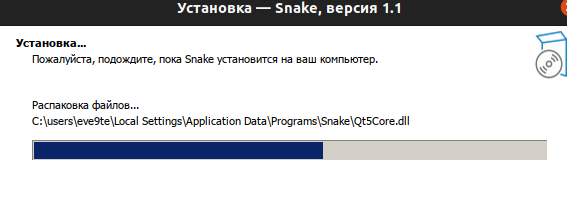
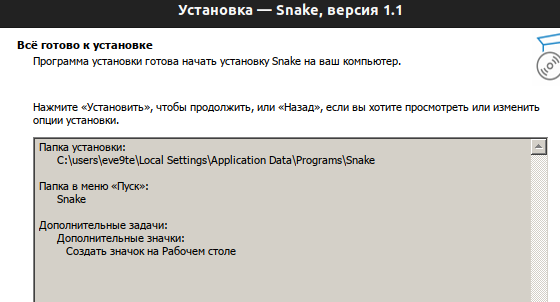
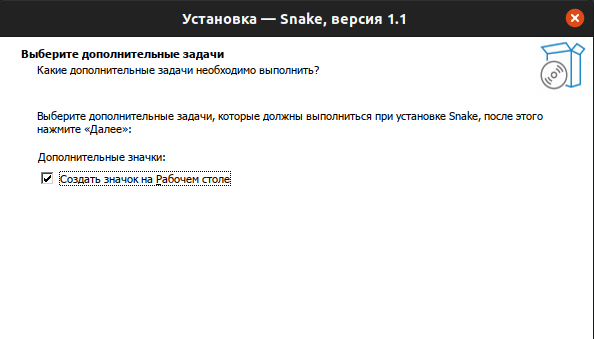
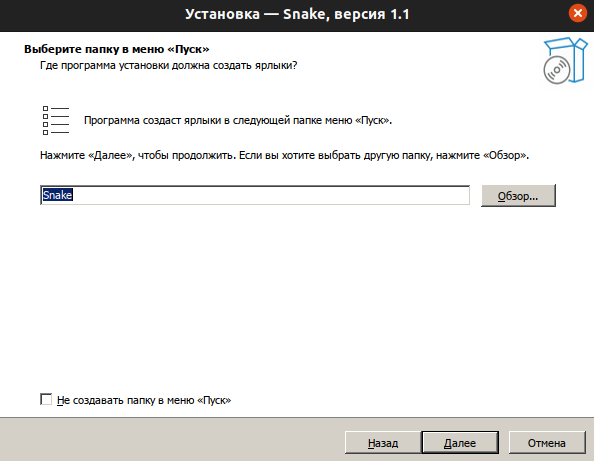
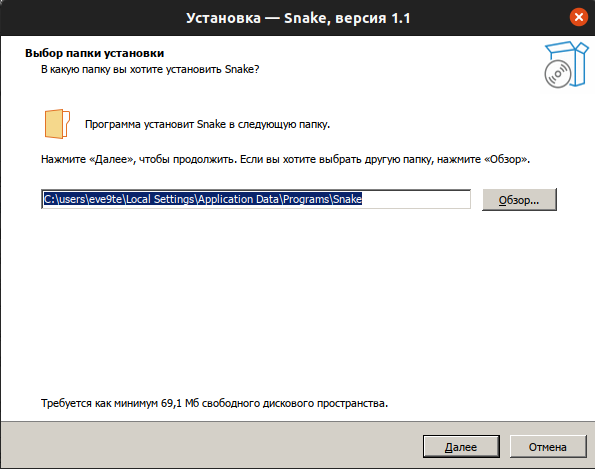
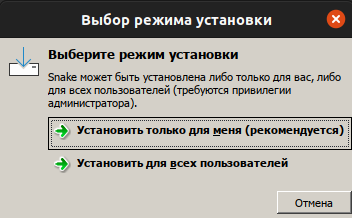
*void MyServer::sockDisc(){*

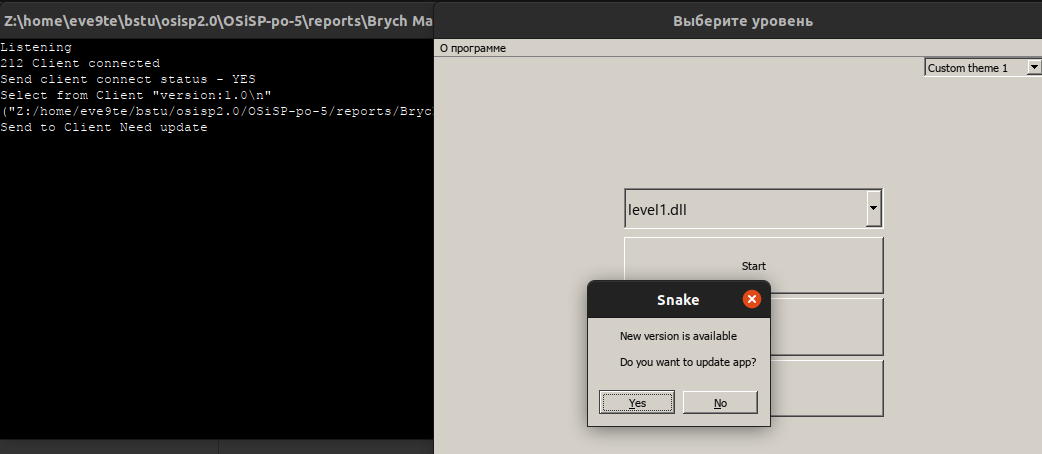
*qDebug()<<"Disconnect";*

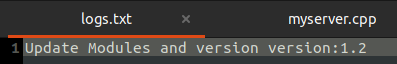
*socket->deleteLater();*

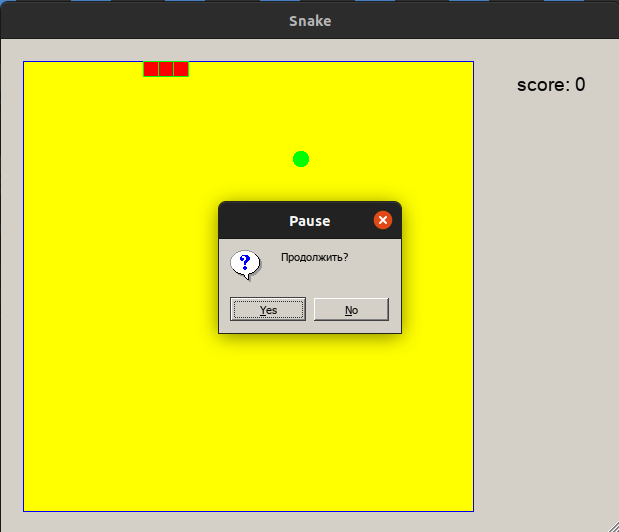
*}*

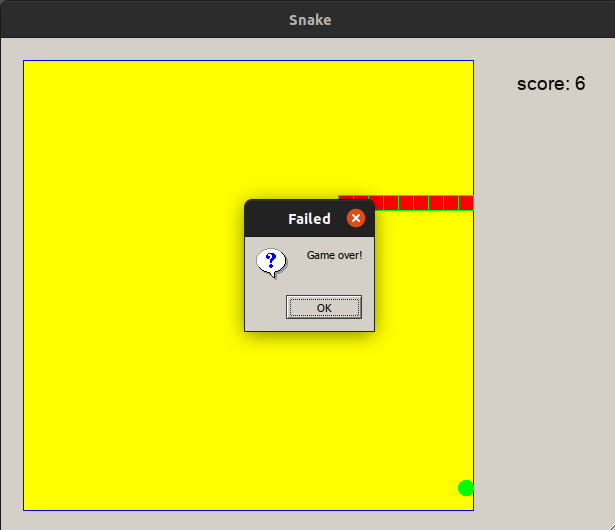
*Тестирование:*

**

**

**





Вывод: приобрел практические навыки проектирования и разработки приложений с графическим пользовательским интерфейсом в ОС Windows средствами Qt.