МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ

 «БРЕСТСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Кафедра ИИТ

ЛАБОРАТОРНАЯ РАБОТА №5

По дисциплине: «ОСИСП»

Выполнил:

Студент ФЭИС

3-го курса, группы ПО-5

Лозейко П.А.

Проверила:

Дряпко А.В.

Брест 2021

**Цель работы:** познакомиться с возможностями, предлагаемыми фреймворком Qt, для разработки многопоточных приложений

*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):

15 DLL, конфигурационный файл (внешний вид)

7) Процесс обновления логируется. При завершении обновления пользователю выдается соответ-

ствующее сообщение.

**maze.h**

#ifndef MAZE\_H

#define MAZE\_H

#include <QMainWindow>

#include <QPaintEvent>

#include <QKeyEvent>

#include <QWidget>

#include <QComboBox>

#include <QPainter>

#include <QTcpSocket>

#include <time.h>

#include <QLibrary>

#include <QString>

#include <QtWidgets>

#include <QMessageBox>

#include <QThread>

#include "thread.h"

#include "generator.h"

#include "interface.h"

#include "interface\_theme.h"

const int BLOCK\_SIZE=15;

const int MARGIN=22;

const int AREA\_ROW=31;

const int AREA\_COL=31;

enum **moved**

{

UP,

DOWN,

LEFT,

RIGHT,

};

QT\_BEGIN\_NAMESPACE

namespace **Ui** { class **Maze**; }

QT\_END\_NAMESPACE

class **Maze** : public QMainWindow

{

Q\_OBJECT

public:

**Maze**(QWidget \*parent = nullptr);

virtual void ***paintEvent***(QPaintEvent \*event);

virtual void ***keyPressEvent***(QKeyEvent \*event);

~***Maze***();

QThread \*thread\_update;

int progress = 0;

int place\_x;

int place\_y;

int place\_finish\_x, place\_finish\_y;

QVector<QVector<int> > field;

public:

void **Set\_interfase\_plugins**();

void **SetPlugins**();

void **InitGame**();

void **InitPause**();

void **PauseResumeGame**();

bool **IsGameOver**();

void **InitMenu**();

void **gener**();

public slots:

void **on\_btn\_exit\_clicked**();

void **on\_btn\_update\_clicked**();

void **sockDisc**();

void **sockReady**();

private slots:

void **showProgress**(int i);

void **loading**(int res);

void **download\_video**(int msg);

void **SnakeUpdate**(int key);

void **aboutQt**();

void **applyPlugin**(int index);

void **applyPlugin\_theme**(int index);

void **on\_btn\_start\_clicked**();

private:

Ui::Maze \*ui;

Ui::Maze \*game\_win; // окно для игры

bool flag\_game = false;

bool isPause;

QList<QPoint> wall;

QList<QPoint> man;

moved dir;

QVector< Interface\* > mPlugins;

QVector< Interface\_theme\* > mTheme;

QTcpSocket\* socket;

QByteArray Data;

};

#endif // MAZE\_H

**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

**game.cpp**

#include "maze.h"

#include "ui\_maze.h"

void Maze::**InitGame**()

{

field = generate(30, 30);

game\_win->setupUi(this);

game\_win->centralwidget->*setVisible*(false);

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

// 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);

man.clear();

wall.clear();

gener();

srand(time(0));

isPause=false;

}

void Maze::**gener**() {

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

for (int j = 0; j < field.at(i).size(); j++) {

int cur = field.at(i).at(j);

if (cur == 0) {qMakePair(i, j); wall.push\_back(QPoint(i, j));}

if (cur == 4) {qMakePair(i, j); place\_finish\_x = i, place\_finish\_y = j; qDebug() << place\_finish\_x << place\_finish\_y;}

else if (cur == 3) {qMakePair(i, j); man.push\_back(QPoint(i, j));}

}

}

}

void Maze::***paintEvent***(QPaintEvent \*event)

{

if(flag\_game) {

Q\_UNUSED(event);

QPainter painter(this);

painter.setBrush(Qt::black);

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

painter.setBrush(Qt::blue);

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.drawRect(MARGIN+man[0].x()\*BLOCK\_SIZE,MARGIN+man[0].y()\*BLOCK\_SIZE,BLOCK\_SIZE,BLOCK\_SIZE);

}

}

void Maze::***keyPressEvent***(QKeyEvent \*event)

{

if(flag\_game) {

switch(event->key())

{

case Qt::Key\_Up:

SnakeUpdate(Qt::Key\_Up);

break;

case Qt::Key\_Down:

SnakeUpdate(Qt::Key\_Down);

break;

case Qt::Key\_Left:

SnakeUpdate(Qt::Key\_Left);

break;

case Qt::Key\_Right:

SnakeUpdate(Qt::Key\_Right);

break;

case Qt::Key\_Escape:

PauseResumeGame();

break;

default:

break;

}

}

}

bool Maze::**IsGameOver**()

{

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

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

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

return true;

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

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

return true;

return false;

}

void Maze::**SnakeUpdate**(int key)

{

if(key == Qt::Key\_Up) man.push\_front(QPoint(man.front().x(),man.front().y()-1));

if(key == Qt::Key\_Down) man.push\_front(QPoint(man.front().x(),man.front().y()+1));

if(key == Qt::Key\_Left) man.push\_front(QPoint(man.front().x()-1,man.front().y()));

if(key == Qt::Key\_Right) man.push\_front(QPoint(man.front().x()+1,man.front().y()));

if(IsGameOver())

{

man.clear();

man.push\_back(QPoint(place\_x, place\_y));

}

else {

man.pop\_back();

place\_x = man.front().x();

place\_y = man.front().y();

qDebug() << place\_x << place\_y;

update();

}

if(place\_x == place\_finish\_x && place\_y == place\_finish\_y) {

QMessageBox::information(this, "Информация", "Вы прошли уровень");

flag\_game = false;

InitMenu();

}

}

**maze.cpp**

#include "maze.h"

#include "ui\_maze.h"

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

{

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

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

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

ui->setupUi(this);

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

SetPlugins();

Set\_interfase\_plugins();

// 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 кнопки

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

Helper\_set\_name\_btn set\_btn\_name\_ = (Helper\_set\_name\_btn)helper\_set\_n\_btn->resolve("set\_window\_btn\_name");

ui->btn\_start->setText(set\_btn\_name\_());

delete helper\_set\_n\_btn;

// 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 Maze::**InitMenu**() {

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

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

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

ui->setupUi(this);

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

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();

Set\_interfase\_plugins();

// 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 кнопки

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

Helper\_set\_name\_btn set\_btn\_name\_ = (Helper\_set\_name\_btn)helper\_set\_n\_btn->resolve("set\_window\_btn\_name");

ui->btn\_start->setText(set\_btn\_name\_());

delete helper\_set\_n\_btn;

// 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 Maze::**PauseResumeGame**()

{

if(!isPause)

{

isPause=!isPause;

InitPause();

}

else isPause=!isPause;

}

void Maze::**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 Maze::**sockDisc**() {

socket->disconnected();

}

void Maze::**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 Maze::**on\_btn\_start\_clicked**()

{

ui->centralwidget->*setVisible*(false);

flag\_game = true;

InitGame();

}

**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);

}

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