МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ

 «БРЕСТСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Кафедра ИИТ

ЛАБОРАТОРНАЯ РАБОТА №4

По дисциплине: «ОСИСП»

Выполнил:

Студент ФЭИС

3-го курса, группы ПО-5

Лозейко П.А.

Проверила:

Дряпко А.В.

Брест 2021

**Цель работы:** ознакомиться с возможностями, предлагаемыми Qt для поддержки сетевого взаимодействия программ.

*Задание:*

1) Разработать сетевую утилиту для автоматического обновления приложения, разработанного в

лабораторных работах 1-3. Утилита может иметь произвольный интерфейс, определяемый ее функциональными особенностями.

2) Программа должна состоять из двух взаимодействующих частей – клиентской, устанавливае- мой на компьютере с обновляемым приложением и серверной, выполняющейся на любом ком- пьютере в локальной либо глобальной сети.

3) Клиентская часть осуществляет соединение с сервером и проверку обновлений для приложения.

При наличии обновлений, все необходимые файлы загружаются и копируются в директорию с целевым приложением. В противном случае выдается соответствующее сообщение. Обработать возможные исключительные ситуации (отсутствие соединения с сервером).

4) Внести изменения в исходный проект приложения с учетом специфики загружаемых обновле- ний (например, хранение структуры уровня для игрового приложения в отдельном файле). То

есть обновляемые ресурсы должны быть отделены от основного приложения.

5) Обновляемые компоненты по вариантам (DLL из лабораторной работы No3):

15 DLL, конфигурационный файл (внешний вид)

6) Процесс обновления логируется. При завершении обновления пользователю выдается соответ- ствующее сообщение.

7) Параллельные потоки при выполнении работы не использовать.

**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") {

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);

}

else {

QStringList rec\_data\_update;

QString new\_version;

rec\_data\_update.append(QString(Data).split(" "));

new\_version.append(rec\_data\_update.last());

QDir new\_client\_version(qApp->applicationDirPath());

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]);

}

mTheme.clear();

ui->modeMenu\_2->clear();

Set\_interfase\_plugins();

QMessageBox::information(this, "Информация", "Обновление завершено!");

//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());

}

}

}

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

{

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

flag\_game = true;

InitGame();

}

**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 "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***();

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 **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;

int countLevels = 0;

QByteArray Data;

};

#endif // MAZE\_H

**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** "Maze.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***()

{

QFont palette("Courier", 12, QFont::Bold, true);

return palette;

}

**Server**

**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

**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();

}

Вывод: ознакомился с возможностями, предлагаемыми Qt для поддержки сетевого взаимодействия программ.