Министерство образования Республики Беларусь

Учреждение образования

«Брестский государственный технический университет»

Факультет электронно-информационных систем

Кафедра ИИТ

Лабораторная работа №4-5

за 5 семестр

По дисциплине: «ОСиСП»

Выполнил:

студент 3 курса

группы ПО-4(1)

Калиновский В. Е.

Проверила:

Дряпко А.В.

Брест 2021

Лабораторная работа №4-5

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

Вариант 11

Задание:

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, устанавливаемой на компьютере с обновляемым приложением и серверной, выполняющейся на любом компьютере 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 обновлений (например, хранение структуры уровня для игрового приложения в отдельном файле). То есть обновляемые ресурсы должны быть отделены от основного приложения

11. конфигурационный файл

Проверка обновления должна осуществляться автоматически по таймеру (QTimer) либо по

непосредственному запросу пользователя.

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

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

**Текст программы:**

**server**

1. **mainwindow.h**

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#include <QMainWindow>

#include <QDebug>

#include <QString>

#include <QThread>

#include "serverstuff.h"

namespace **Ui** {

class **MainWindow**;

}

class **MainWindow** : public QMainWindow

{

Q\_OBJECT

public:

explicit **MainWindow**(QWidget \*parent = 0);

~***MainWindow***();

void **setVersion**(QString version);

QString **getVersion**();

QString **getDefaultVersion**();

void **sendFile**();

private slots:

void **on\_pushButton\_stopServer\_clicked**();

void **on\_pushButton\_startServer\_clicked**();

void **on\_pushButton\_testConn\_clicked**();

void **smbConnectedToServer**();

void **smbDisconnectedFromServer**();

void **gotNewMesssage**(QString msg);

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

private:

Ui::MainWindow \*ui;

ServerStuff \*server;

ServerStuff \*fileServer;

QString mVersion;

};

#endif // MAINWINDOW\_H

1. **serverstuff.h**

#ifndef SERVERSTUFF\_H

#define SERVERSTUFF\_H

#include <QTcpServer>

#include <QTcpSocket>

#include <QDataStream>

#include <QList>

class **ServerStuff** : public QObject

{

Q\_OBJECT

public:

**ServerStuff**(QObject \*pwgt);

QTcpServer \*tcpServer;

QList<QTcpSocket \*> **getClients**();

bool **isVacant**();

public slots:

virtual void ***newConnection***();

void **readClient**();

void **gotDisconnection**();

qint64 **sendToClient**(QTcpSocket \*socket, const QString &str);

qint64 **sendFileToClient**(QTcpSocket \*socket, const QString &fileName);

signals:

void **gotNewMesssage**(QString msg);

void **smbDisconnected**();

private:

quint16 m\_nNextBlockSize;

QList<QTcpSocket\*> clients;

bool serverIsVacant;

};

#endif // SERVERSTUFF\_H

1. **main.cpp**

#include <QApplication>

#include "mainwindow.h"

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

{

QApplication a(*argc*, *argv*);

//QTextCodec::setCodecForTr(QTextCodec::codecForName("UTF-8"));

MainWindow w;

w.show();

return a.exec();

}

1. **mainwindow.cpp**

#include "mainwindow.h"

#include "ui\_mainwindow.h"

#include <QJsonObject>

#include <QJsonDocument>

#include <QJsonArray>

#include <QFile>

MainWindow::**MainWindow**(QWidget \*parent) :

QMainWindow(*parent*),

ui(new Ui::MainWindow)

{

ui->setupUi(this);

ui->pushButton\_stopServer->setVisible(false);

ui->pushButton\_update->setVisible(false);

this->server = new ServerStuff(this);

connect(this->server, &ServerStuff::gotNewMesssage,

this, &MainWindow::gotNewMesssage);

connect(this->server->tcpServer, &QTcpServer::newConnection,

this, &MainWindow::smbConnectedToServer);

connect(this->server, &ServerStuff::smbDisconnected,

this, &MainWindow::smbDisconnectedFromServer);

this->fileServer = new ServerStuff(this);

this->setVersion(this->getDefaultVersion());

}

MainWindow::~***MainWindow***()

{

delete server;

delete ui;

}

void MainWindow::**on\_pushButton\_startServer\_clicked**()

{

ui->pushButton\_stopServer->setVisible(true);

ui->pushButton\_update->setVisible(true);

ui->pushButton\_startServer->setVisible(false);

if (!server->tcpServer->listen(QHostAddress::Any, 6547))

{

ui->textEdit\_log->append(tr("<font color=\"red\"><b>Error!</b> The port is taken by some other service.</font>"));

return;

}

connect(server->tcpServer, &QTcpServer::newConnection, server, &ServerStuff::newConnection);

if (!fileServer->tcpServer->listen(QHostAddress::Any, 6788))

{

ui->textEdit\_log->append(tr("<font color=\"red\"><b>Error!</b> The port is taken by some other service.</font>"));

return;

}

connect(fileServer->tcpServer, &QTcpServer::newConnection, fileServer, &ServerStuff::newConnection);

ui->textEdit\_log->append(tr("<font color=\"green\"><b>Server started</b>, port is openned.</font>"));

}

void MainWindow::**on\_pushButton\_stopServer\_clicked**()

{

if(server->tcpServer->isListening())

{

ui->pushButton\_update->setVisible(false);

ui->pushButton\_stopServer->setVisible(false);

ui->pushButton\_startServer->setVisible(true);

disconnect(server->tcpServer, &QTcpServer::newConnection, server, &ServerStuff::newConnection);

QList<QTcpSocket \*> clients = server->getClients();

for(int i = 0; i < clients.count(); i++)

{

server->sendToClient(*clients.at(i)*, "0");

}

server->tcpServer->close();

ui->textEdit\_log->append(tr("<b>Server stopped</b>, post is closed"));

}

else

{

ui->textEdit\_log->append(tr("<b>Error!</b> Server was not running"));

}

}

void MainWindow::**on\_pushButton\_testConn\_clicked**()

{

if(server->tcpServer->isListening())

{

ui->textEdit\_log->append(

QString("%1 %2")

.arg("Server is listening, number of connected clients:")

.arg(QString::number(server->getClients().count()))

);

}

else

{

ui->textEdit\_log->append(

QString("%1 %2")

.arg("Server is not listening, number of connected clients:")

.arg(QString::number(server->getClients().count()))

);

}

}

void MainWindow::**smbConnectedToServer**()

{

ui->textEdit\_log->append(tr("Somebody has connected"));

}

void MainWindow::**smbDisconnectedFromServer**()

{

ui->textEdit\_log->append(tr("Somebody has disconnected"));

}

void MainWindow::**gotNewMesssage**(QString msg)

{

ui->textEdit\_log->append(QString("New message: %1").arg(msg));

if (msg.contains("version:"))

{

QList<QTcpSocket \*> clients = server->getClients();

this->server->sendToClient(*clients.at(*0*)*, "version:" + this->mVersion);

}

else if (msg == "OK ON UPDATE")

{

QList<QTcpSocket \*> clients = server->getClients();

// this->server->sendToClient(clients.at(0), "file:" + QApplication::applicationDirPath() + "/helper\_class.dll");

}

}

QString MainWindow::**getDefaultVersion**()

{

QString val;

QFile file;

file.setFileName("server.json");

file.*open*(QIODevice::ReadOnly | QIODevice::Text);

val = file.readAll();

file.*close*();

QString version = "1";

if (val != "") {

QJsonDocument doc = QJsonDocument::fromJson(val.toUtf8());

QJsonObject json = doc.object();

version = json["version"].toString();

}

return QString::number(2);

}

void MainWindow::**setVersion**(QString version)

{

QJsonObject recordObject;

recordObject.insert("version", QJsonValue::fromVariant(version));

QJsonDocument doc(recordObject);

QString jsonString = doc.toJson(QJsonDocument::Indented);

QFile file;

file.setFileName("server.json");

file.*open*(QIODevice::WriteOnly | QIODevice::Text);

QTextStream stream( *&file* );

stream << jsonString;

file.*close*();

this->mVersion = version;

}

QString MainWindow::**getVersion**()

{

return this->mVersion;

}

void MainWindow::**on\_pushButton\_update\_clicked**()

{

this->setVersion(QString::number((this->getVersion()).toInt() + 1));

if(server->tcpServer->isListening())

{

QList<QTcpSocket \*> clients = server->getClients();

for(int i = 0; i < clients.count(); i++)

{

server->sendToClient(*clients.at(i)*, "version:" + this->mVersion);

}

}

}

1. **serverstuff.cpp**

#include "serverstuff.h"

#include <QFile>

#include <QDebug>

#include <iostream>

ServerStuff::**ServerStuff**(QObject \*pwgt) : QObject(*pwgt*), m\_nNextBlockSize(0)

{

tcpServer = new QTcpServer(this);

this->serverIsVacant = true;

}

QList<QTcpSocket \*> ServerStuff::**getClients**()

{

return clients;

}

void ServerStuff::***newConnection***()

{

QTcpSocket \*clientSocket = tcpServer->*nextPendingConnection*();

connect(clientSocket, &QTcpSocket::disconnected, clientSocket, &QTcpSocket::deleteLater);

connect(clientSocket, &QTcpSocket::readyRead, this, &ServerStuff::readClient);

connect(clientSocket, &QTcpSocket::disconnected, this, &ServerStuff::gotDisconnection);

clients << clientSocket;

sendToClient(*clientSocket*, "Reply: connection established");

}

void ServerStuff::**readClient**()

{

QTcpSocket \*clientSocket = (QTcpSocket\*)sender();

QDataStream in(*clientSocket*);

while(true)

{

if (!m\_nNextBlockSize) {

if (clientSocket->*bytesAvailable*() < sizeof(quint16)) { break; }

in >> m\_nNextBlockSize;

}

if (clientSocket->*bytesAvailable*() < m\_nNextBlockSize)

{

break;

}

QString str;

in >> str;

emit gotNewMesssage(str);

m\_nNextBlockSize = 0;

}

}

void ServerStuff::**gotDisconnection**()

{

clients.removeAt(clients.indexOf((QTcpSocket\*)sender()));

emit smbDisconnected();

}

qint64 ServerStuff::**sendToClient**(QTcpSocket\* socket, const QString& str)

{

QByteArray arrBlock;

QDataStream out(*&arrBlock*, QIODevice::WriteOnly);

out << quint16(0) << str;

out.device()->*seek*(0);

out << quint16(arrBlock.size() - sizeof(quint16));

return socket->write(arrBlock);

}

qint64 ServerStuff::**sendFileToClient**(QTcpSocket\* socket, const QString& fileName)

{

this->serverIsVacant = false;

QByteArray arrBlock;

QFile file(fileName);

file.*open*(QIODevice::ReadOnly);

arrBlock = file.readAll();

file.*close*();

std::cout <<fileName.toStdString() << std::endl;

qDebug() << fileName;

this->serverIsVacant = true;

return socket->write(arrBlock);

}

bool ServerStuff::**isVacant**()

{

return this->serverIsVacant;

}

**minesweeper**

1. **clientstuff.h**

#ifndef CLIENTSTUFF\_H

#define CLIENTSTUFF\_H

#include <QString>

#include <QTcpSocket>

#include <QDataStream>

#include <QTimer>

class **ClientStuff** : public QObject

{

Q\_OBJECT

public:

**ClientStuff**(const QString hostAddress, int portMessVal, int portFileVal, QObject \*parent = 0);

QTcpSocket \*tcpSocket;

QTcpSocket \*tcpFileSocket;

bool **getStatus**();

public slots:

void **closeConnection**();

void **connect2host**();

signals:

void **statusChanged**(bool);

void **hasReadSome**(QString msg);

// void tconnectFileSocket();

private slots:

void **readyRead**();

void **readyReadFile**();

void **connected**();

void **connectionTimeout**();

// void connectFileSocket();

private:

void **connectFileSocket**();

void **connectMessSocket**();

QString host;

int portMess;

int portFile;

bool status;

quint16 m\_nNextBlockSize;

quint16 m\_nNextFileBlockSize;

QTimer \*timeoutTimer;

QString fileName;

};

#endif // CLIENTSTUFF\_H

1. **clientstuff.cpp**

#include "clientstuff.h"

#include <QFile>

#include <QDir>

#include <iostream>

#include <QAbstractSocket>

#include <QDirIterator>

#include <QThread>

ClientStuff::**ClientStuff**(

const QString hostAddress,

int portMessNumber,

int portFileNumber,

QObject \*parent

) : QObject(*parent*), m\_nNextBlockSize(0)

{

this->status = false;

this->host = hostAddress;

this->portMess = portMessNumber;

this->portFile = portFileNumber;

this->fileName = "file";

this->tcpSocket = new QTcpSocket(this);

connect(this->tcpSocket, &QTcpSocket::disconnected, this, &ClientStuff::closeConnection);

this->tcpFileSocket = new QTcpSocket(this);

connect(this->tcpFileSocket, &QTcpSocket::disconnected, this, &ClientStuff::closeConnection);

this->timeoutTimer = new QTimer();

this->timeoutTimer->setSingleShot(true);

connect(this->timeoutTimer, &QTimer::timeout, this, &ClientStuff::connectionTimeout);

}

void ClientStuff::**connect2host**()

{

this->timeoutTimer->start(3000);

this->connectMessSocket();

QThread \*fileThread = new QThread();

connect( fileThread, SIGNAL(connectFileSocket()), this, SLOT(connectFileSocket()), Qt::BlockingQueuedConnection ) ;

fileThread->start();

this->connectFileSocket();

}

void ClientStuff::**connectMessSocket**()

{

this->tcpSocket->*connectToHost*(host, this->portMess);

connect(this->tcpSocket, &QTcpSocket::connected, this, &ClientStuff::connected);

connect(this->tcpSocket, &QTcpSocket::readyRead, this, &ClientStuff::readyRead);

}

void ClientStuff::**connectFileSocket**()

{

this->tcpFileSocket->*connectToHost*(host, this->portFile);

connect(this->tcpFileSocket, &QTcpSocket::connected, this, &ClientStuff::connected);

connect(this->tcpFileSocket, &QTcpSocket::readyRead, this, &ClientStuff::readyReadFile);

}

void ClientStuff::**connectionTimeout**()

{

if(this->tcpSocket->state() == QAbstractSocket::ConnectingState)

{

this->tcpSocket->abort();

// emit this->tcpSocket->error(QAbstractSocket::SocketTimeoutError);

}

if(this->tcpFileSocket->state() == QAbstractSocket::ConnectingState)

{

this->tcpFileSocket->abort();

// emit this->tcpFileSocket->error(QAbstractSocket::SocketTimeoutError);

}

}

void ClientStuff::**connected**()

{

this->status = true;

emit statusChanged(this->status);

}

bool ClientStuff::**getStatus**()

{

return this->status;

}

void ClientStuff::**readyRead**()

{

QDataStream in(this*->tcpSocket*);

while(true)

{

if (!this->m\_nNextBlockSize)

{

if (this->tcpSocket->*bytesAvailable*() < sizeof(quint16))

{

break;

}

in >> this->m\_nNextBlockSize;

}

if (this->tcpSocket->*bytesAvailable*() < this->m\_nNextBlockSize)

{

break;

}

QString str;

in >> str;

if (str == "0")

{

str = "Connection closed";

this->closeConnection();

}

/\*

if(str.contains("file:"))

{

this->fileName = str.remove(0, 5);

str = "file:" + this->fileName;

QString newDir = "../figures";

#ifdef QT\_DEBUG

this->fileName += "/debug";

newDir += "/debug";

#else

#ifdef QT\_RELEASE

this->fileName = "/release";

newDir += "/release";

#endif

#endif

}

\*/

emit hasReadSome(str);

m\_nNextBlockSize = 0;

}

}

void ClientStuff::**readyReadFile**()

{

if(this->fileName == "")

{

QDataStream in(this*->tcpFileSocket*);

while(true)

{

if (!this->m\_nNextBlockSize)

{

if (this->tcpFileSocket->*bytesAvailable*() < sizeof(quint16))

{

break;

}

in >> this->m\_nNextBlockSize;

}

if (this->tcpFileSocket->*bytesAvailable*() < this->m\_nNextBlockSize)

{

break;

}

QByteArray byteArr;

in >> byteArr;

QFile file(this->fileName);

file.*open*(QIODevice::WriteOnly);

file.write(byteArr);

file.*close*();

m\_nNextFileBlockSize = 0;

}

}

}

void ClientStuff::**closeConnection**()

{

this->timeoutTimer->stop();

disconnect(this->tcpSocket, &QTcpSocket::connected, 0, 0);

disconnect(this->tcpSocket, &QTcpSocket::readyRead, 0, 0);

disconnect(this->tcpFileSocket, &QTcpSocket::readyRead, 0, 0);

bool shouldEmit = false;

switch (this->tcpSocket->state())

{

case 0:

this->tcpSocket->*disconnectFromHost*();

shouldEmit = true;

break;

case 2:

this->tcpSocket->abort();

shouldEmit = true;

break;

default:

this->tcpSocket->abort();

}

switch (this->tcpFileSocket->state())

{

case 0:

this->tcpFileSocket->*disconnectFromHost*();

shouldEmit = true;

break;

case 2:

this->tcpFileSocket->abort();

shouldEmit = true;

break;

default:

this->tcpFileSocket->abort();

}

if (shouldEmit)

{

this->status = false;

emit statusChanged(this->status);

}

}

1. **mainwindow.h**

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#include <QMainWindow>

#include <QVBoxLayout>

#include <QHBoxLayout>

#include <QGridLayout>

#include <QPushButton>

#include <QLabel>

#include <QPushButton>

#include <QComboBox>

#include <QTimer>

#include <QMap>

#include <QMessageBox>

#include "MyHeader.h"

#include "clientstuff.h"

QT\_BEGIN\_NAMESPACE

namespace **Ui** { class **MainWindow**; }

QT\_END\_NAMESPACE

class **MainWindow** : public QMainWindow

{

Q\_OBJECT

public:

QTimer \*mTimerBar;

QTimer \*mTimerUpdates;

ClientStuff \*mClient;

QString mVersion;

void **setVersion**(QString version);

void **download**();

QString **getVersion**();

QString **getDefaultVersion**();

Ui::MainWindow \*ui;

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

~***MainWindow***();

private:

QHBoxLayout\* firstLayout;

QGridLayout\* gameLayout;

QVBoxLayout\* mainLayout;

QComboBox\* difficulties;

QFont font;

MyButton\*\*\* buttons;

QPushButton\* resetButton;

QPushButton\* showHighscores;

QString\*\* buttonText;

QLabel\* flags;

QLabel\* timeLabel;

QTimer\* timer;

double time;

int difficulty;

int buttonRows, buttonColumns;

int numberOfBombs;

int disabledButtons;

int numberOfFlags;

bool endOfGame;

QMap <double, QString> highscoresEasy;

QMap <double, QString> highscoresMedium;

QMap <double, QString> highscoresHard;

QMessageBox\* top10;

void **clearButtons**();

void **setButtons**(int n, int m, int size);

void **setBombs**();

void **setNumbers**();

void **onFreeButtonClicked**(int i, int j);

void **bombClicked**();

void **win**();

void **reset**();

void **showTop10**(int difficulty);

QComboBox\* **createComboBox**();

private slots:

void **difficultyChanged**();

void **buttonClicked**();

void **rightButtonClicked**();

void **tick**();

public slots:

void **on\_CheckUpdateButton\_triggered**();

void **on\_ConnectButton\_triggered**();

void **on\_DisconnectButton\_triggered**();

void **on\_Version\_triggered**();

void **receivedSomething**(QString msg);

};

#endif // MAINWINDOW\_H

1. **mainwindow.cpp**

#include "mainwindow.h"

#include "ui\_mainwindow.h"

#include <QDebug>

#include <QDateTime>

#include <QInputDialog>

#include <QMapIterator>

#include <QMessageBox>

#include <QJsonDocument>

#include <QJsonObject>

#include <QFile>

MainWindow::**MainWindow**(QWidget \*parent) : QMainWindow(*parent*)

, ui(new Ui::MainWindow)

{

mClient = new ClientStuff("localhost", 6547, 6788);

connect(this->mClient, &ClientStuff::hasReadSome, this, &MainWindow::receivedSomething);

this->mTimerUpdates = new QTimer(this);

connect(this->mTimerUpdates, SIGNAL(timeout()), this, SLOT(on\_CheckUpdateButton\_triggered()));

this->mTimerUpdates->start(10000);

ui->setupUi(this);

QWidget \*w = new QWidget;

setCentralWidget(*w*);

setWindowTitle("Minesweeper");

w->setStyleSheet("background-color:green");

setVersion(getDefaultVersion());

// layout-ok beállítása

font.setFamily("Comic Sans MS");

mainLayout = new QVBoxLayout;

firstLayout = new QHBoxLayout;

firstLayout->*setSpacing*(50);

gameLayout = new QGridLayout;

gameLayout->*setSpacing*(0);

gameLayout->setContentsMargins(0, 0, 0, 0);

mainLayout->addLayout(*firstLayout*);

mainLayout->addLayout(*gameLayout*);

w->setLayout(*mainLayout*);

time = 0;

disabledButtons = 0;

difficulty = 0;

endOfGame = false;

difficulties = createComboBox();

firstLayout->addWidget(*difficulties*);

flags = new QLabel;

font = flags->font();

font.setPointSize(font.pointSize() + 8);

flags->setFont(font);

flags->setText("Flags: 10");

firstLayout->addWidget(*flags*);

timeLabel = new QLabel("0:0");

timeLabel->setFont(font);

timer = new QTimer;

connect(timer, SIGNAL(timeout()), this, SLOT(tick()));

firstLayout->addWidget(*timeLabel*);

showHighscores = new QPushButton("Highscores");

showHighscores->setFont(font);

showHighscores->setEnabled(false);

connect(showHighscores, &QPushButton::clicked, this, [&](){showTop10(difficulty);});

firstLayout->addWidget(*showHighscores*);

resetButton = new QPushButton("Reset");

resetButton->setFont(font);

connect(resetButton, &QPushButton::clicked, this, [&](){reset();} );

firstLayout->addWidget(*resetButton*);

top10 = new QMessageBox;

top10->setWindowTitle("Top 10");

QPushButton\* ok = new QPushButton("Ok");

QPushButton\* resetScores = new QPushButton("Reset highscores");

connect(resetScores, &QPushButton::pressed, this, [&](){

if(difficulty == 0){

highscoresEasy.clear();

}

else if(difficulty == 1){

highscoresMedium.clear();

}

else{

highscoresHard.clear();

}

top10->setText("");

});

top10->addButton(*resetScores*, QMessageBox::ResetRole);

top10->addButton(*ok*, QMessageBox::AcceptRole);

setButtons(8, 10, 50);

setBombs();

setNumbers();

}

void MainWindow::**setButtons**(int n, int m, int size){

buttonRows = n;

buttonColumns = m;

buttons = new MyButton\*\*[n];

buttonText = new QString\*[n];

for(int i=0; i<n; ++i){

buttons[i] = new MyButton\*[m];

buttonText[i] = new QString[m];

}

for(int i=0; i<n; ++i){

for(int j=0; j<m; ++j){

buttonText[i][j] = '-';

}

}

int fontsize;

switch (difficulty) {

case 0:

fontsize = 12;

numberOfBombs = 10;

numberOfFlags = 10;

break;

case 1:

fontsize = 10;

numberOfBombs = 40;

numberOfFlags = 40;

break;

case 2:

fontsize = 8;

numberOfBombs = 99;

numberOfFlags = 99;

break;

}

for(int i=0; i<n; ++i){

for(int j=0; j<m; ++j){

buttons[i][j] = new MyButton(this);

buttons[i][j]->setMinimumSize(size, size);

buttons[i][j]->setSizePolicy(QSizePolicy::Expanding, QSizePolicy::Expanding);

font = buttons[i][j]->font();

font.setPointSize(font.pointSize() + fontsize);

buttons[i][j]->setFont(font);

buttons[i][j]->installEventFilter(this);

connect(buttons[i][j], SIGNAL(clicked()), this, SLOT(buttonClicked()));

connect(buttons[i][j], SIGNAL(rightClick()), this, SLOT(rightButtonClicked()));

if( (i+j) % 2 == 0 ){

buttons[i][j]->setStyleSheet("background-color:palegreen");

}

else{

buttons[i][j]->setStyleSheet("background-color:limegreen");

}

gameLayout->addWidget(*buttons[i][j]*, i, j);

}

}

}

void MainWindow::**setBombs**(){

int nB = numberOfBombs;

while(nB > 0){

QTime time = QTime::currentTime();

srand((uint)time.msec());

int x = rand() % buttonRows;

int y = rand() % buttonColumns;

if (buttonText[x][y] == 'X'){

continue;

}

buttonText[x][y] = 'X';

--nB;

}

}

void MainWindow::**setNumbers**(){

for(int i=0; i<buttonRows; ++i){

for(int j=0; j<buttonColumns; ++j){

if(buttonText[i][j] == 'X'){

continue;

}

int db = 0;

for(int x=i-1; x<i+2; ++x){

for(int y=j-1; y<j+2; ++y){

if(x<0 || x>buttonRows-1 || y<0 || y>buttonColumns-1){

continue;

}

if(buttonText[x][y] == "X"){

++db;

}

}

}

if(db != 0){

buttonText[i][j] = QString::number(db);

}

}

}

}

void MainWindow::**onFreeButtonClicked**(int i, int j){

if(i<0 || i>buttonRows-1 || j<0 || j>buttonColumns-1 || buttonText[i][j] == "X" || !buttons[i][j]->isEnabled()){

return;

}

if(buttonText[i][j] == '-'){

buttons[i][j]->setStyleSheet("background-color:azure");

buttons[i][j]->setEnabled(false);

++disabledButtons;

for(int x=i-1; x<i+2; ++x){

for(int y=j-1; y<j+2; ++y){

onFreeButtonClicked(x,y);

}

}

}

if(buttonText[i][j] != '-'){

buttons[i][j]->setStyleSheet("background-color:wheat");

buttons[i][j]->setText(buttonText[i][j]);

buttons[i][j]->setEnabled(false);

++disabledButtons;

return;

}

}

void MainWindow::**bombClicked**(){

endOfGame = true;

timer->stop();

flags->setText("You lost!");

for(int i=0; i<buttonRows; ++i){

for(int j=0; j<buttonColumns; ++j){

if(buttonText[i][j] == "X"){

buttons[i][j]->setStyleSheet("background-color:tomato");

buttons[i][j]->setText(buttonText[i][j]);

}

}

}

showHighscores->setEnabled(true);

}

void MainWindow::**win**(){

endOfGame = true;

timer->stop();

flags->setText("You win!");

for(int i=0; i<buttonRows; ++i){

for(int j=0; j<buttonColumns; ++j){

if(buttonText[i][j] == "X"){

buttons[i][j]->setStyleSheet("background-color:pink");

buttons[i][j]->setText(buttonText[i][j]);

}

else{

if(buttonText[i][j] == "-"){

buttons[i][j]->setStyleSheet("background-color:powderblue");

}

else{

buttons[i][j]->setStyleSheet("background-color:powderblue");

buttons[i][j]->setText(buttonText[i][j]);

}

}

}

}

QString name = QInputDialog::getText(this, "Best Results", "Your Result: " +

QString::number(time) + "s\nName:");

if(difficulty == 0){

highscoresEasy[time] = name;

if(highscoresEasy.size() > 10){

auto it = highscoresEasy.end();

highscoresEasy.erase(--it);

}

}

else if(difficulty == 1){

highscoresMedium[time] = name;

if(highscoresMedium.size() > 10){

auto it = highscoresMedium.end();

highscoresMedium.erase(--it);

}

}

else{

highscoresHard[time] = name;

if(highscoresHard.size() > 10){

auto it = highscoresHard.end();

highscoresHard.erase(--it);

}

}

showTop10(difficulty);

showHighscores->setEnabled(true);

}

void MainWindow::**reset**(){

endOfGame = false;

time = 0;

timer->stop();

timeLabel->setText("0:0");

showHighscores->setEnabled(false);

disabledButtons = 0;

if(difficulty == 0){

clearButtons();

setButtons(8, 10, 50);

setBombs();

setNumbers();

flags->setText("Flags: 10");

}

else if(difficulty == 1){

clearButtons();

setButtons(14, 18, 40);

setBombs();

setNumbers();

flags->setText("Flags: 40");

}

else{

clearButtons();

setButtons(20, 24, 30);

setBombs();

setNumbers();

flags->setText("Flags: 99");

}

}

void MainWindow::**clearButtons**(){

while(gameLayout->*count*()){

for(int i=0; i<gameLayout->*count*(); ++i){

QWidget \*tempWidget = gameLayout->*itemAt*(i++)->*widget*();

mainLayout->removeWidget(*tempWidget*);

delete tempWidget;

}

}

delete buttons;

delete buttonText;

}

void MainWindow::**buttonClicked**(){

if(endOfGame){

return;

}

if(!timer->isActive()){

timer->start(80);

}

int x, y;

QPushButton\* tmp = (QPushButton\*)sender();

if(tmp->styleSheet() == "background-color:brown"){

return;

}

for(int i=0; i<buttonRows; ++i){

for(int j=0; j<buttonColumns; ++j){

if(buttons[i][j] == tmp){

x = i;

y = j;

break;

}

}

}

if(buttonText[x][y] != "X"){

onFreeButtonClicked(x,y);

if( disabledButtons == ((buttonRows\*buttonColumns) - numberOfBombs) ){

win();

}

}

else{

bombClicked();

}

}

void MainWindow::**rightButtonClicked**(){

if(!timer->isActive()){

timer->start(80);

}

QPushButton\* tmp = (QPushButton\*)sender();

int x, y;

if(tmp->styleSheet() == "background-color:brown"){

for(int i=0; i<buttonRows; ++i){

for(int j=0; j<buttonColumns; ++j){

if(buttons[i][j] == tmp){

x = i;

y = j;

break;

}

}

}

if( (x+y) % 2 == 0 ){

buttons[x][y]->setStyleSheet("background-color:palegreen");

}

else{

buttons[x][y]->setStyleSheet("background-color:limegreen");

}

++numberOfFlags;

flags->setText("Flags: " + QString::number(numberOfFlags));

}

else{

tmp->setStyleSheet("background-color:brown");

--numberOfFlags;

flags->setText("Flags: " + QString::number(numberOfFlags));

}

}

void MainWindow::**showTop10**(int difficulty){

if(difficulty == 0){

QMapIterator<double, QString> it(highscoresEasy);

top10->setText("Level: Easy\n\n");

for(int i=1; i<11; ++i){

if(it.hasNext()){

it.next();

top10->setText(top10->text() + QString::number(i) + ". " + it.value() + ": " + QString::number(it.key()) + "s\n");

}

else{

break;

}

}

}

else if(difficulty == 1){

QMapIterator<double, QString> it(highscoresMedium);

top10->setText("Level: Medium\n\n");

for(int i=1; i<11; ++i){

if(it.hasNext()){

it.next();

top10->setText(top10->text() + QString::number(i) + ". " + it.value() + ": " + QString::number(it.key()) + "s\n");

}

else{

break;

}

}

}

else{

QMapIterator<double, QString> it(highscoresHard);

top10->setText("Level: Hard\n\n");

for(int i=1; i<11; ++i){

if(it.hasNext()){

it.next();

top10->setText(top10->text() + QString::number(i) + ". " + it.value() + ": " + QString::number(it.key()) + "s\n");

}

else{

break;

}

}

}

top10->*exec*();

}

QComboBox\* MainWindow::**createComboBox**(){

QComboBox\* temp = new QComboBox;

temp->setStyleSheet("background-color:aliceblue");

font = temp->font();

font.setPointSize(font.pointSize() + 8);

temp->setFont(font);

temp->addItem("Easy");

temp->addItem("Medium");

temp->addItem("Hard");

connect(temp, SIGNAL(currentIndexChanged(int)), this, SLOT(difficultyChanged()));

return temp;

}

void MainWindow::**tick**(){

time += 0.1;

timeLabel->setText(QString::number(time));

}

void MainWindow::**difficultyChanged**(){

difficulty = difficulties->currentIndex();

reset();

}

MainWindow::~***MainWindow***()

{

}

void MainWindow::**on\_CheckUpdateButton\_triggered**()

{

QByteArray arrBlock;

QDataStream out(*&arrBlock*, QIODevice::WriteOnly);

out << quint16(0) << "version:" + this->mVersion;

out.device()->*seek*(0);

out << quint16(arrBlock.size() - sizeof(quint16));

mClient->tcpSocket->write(arrBlock);

}

void MainWindow::**on\_ConnectButton\_triggered**()

{

mClient->connect2host();

}

void MainWindow::**on\_DisconnectButton\_triggered**()

{

mClient->closeConnection();

}

void MainWindow::**on\_Version\_triggered**()

{

QString version = QString("Version: %1").arg(mVersion);

QMessageBox::information(this, tr("Info"), version);

}

void MainWindow::**receivedSomething**(QString msg)

{

if(msg.contains("version:"))

{

QString version = msg.remove(0, 8);

QMessageBox msgBox;

if(version == this->mVersion)

{

msgBox.setText("No available updates");

msgBox.*exec*();

}

else

{

msgBox.setText("New version " + version + " is available");

msgBox.setInformativeText("Do you want to update app?");

msgBox.setStandardButtons(QMessageBox::Yes | QMessageBox::No);

msgBox.setDefaultButton(QMessageBox::Yes);

int res = msgBox.*exec*();

if (res == QMessageBox::Yes)

{

this->setVersion(version);

QByteArray arrBlock;

QDataStream out(*&arrBlock*, QIODevice::WriteOnly);

QString answer = "OK ON UPDATE";

out << quint16(0) << answer;

download();

out.device()->*seek*(0);

out << quint16(arrBlock.size() - sizeof(quint16));

this->mClient->tcpSocket->write(arrBlock);

}

}

}

if(msg.contains("file:"))

{

download();

for(int i = 0; i < 100; ++i)

{

this->mTimerBar->start(500);

}

}

}

QString MainWindow::**getDefaultVersion**()

{

QString val;

QFile file;

file.setFileName("client.json");

file.*open*(QIODevice::ReadOnly | QIODevice::Text);

val = file.readAll();

file.*close*();

QString version = "1";

if (val != "") {

QJsonDocument doc = QJsonDocument::fromJson(val.toUtf8());

QJsonObject json = doc.object();

version = json["version"].toString();

}

return version;

}

void MainWindow::**setVersion**(QString version)

{

QJsonObject recordObject;

recordObject.insert("version", QJsonValue::fromVariant(version));

QJsonDocument doc(recordObject);

QString jsonString = doc.toJson(QJsonDocument::Indented);

QFile file;

file.setFileName("client.json");

file.*open*(QIODevice::WriteOnly | QIODevice::Text);

QTextStream stream( *&file* );

stream << jsonString;

file.*close*();

mVersion = version;

}

QString MainWindow::**getVersion**()

{

return mVersion;

}

void MainWindow::**download**()

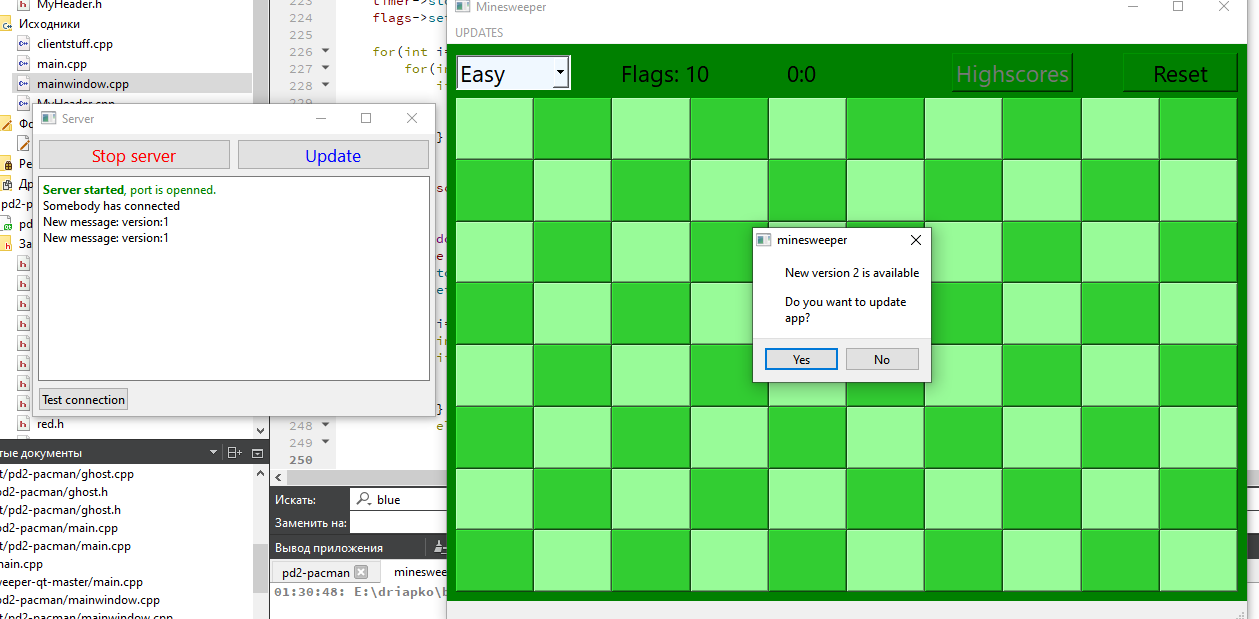
{

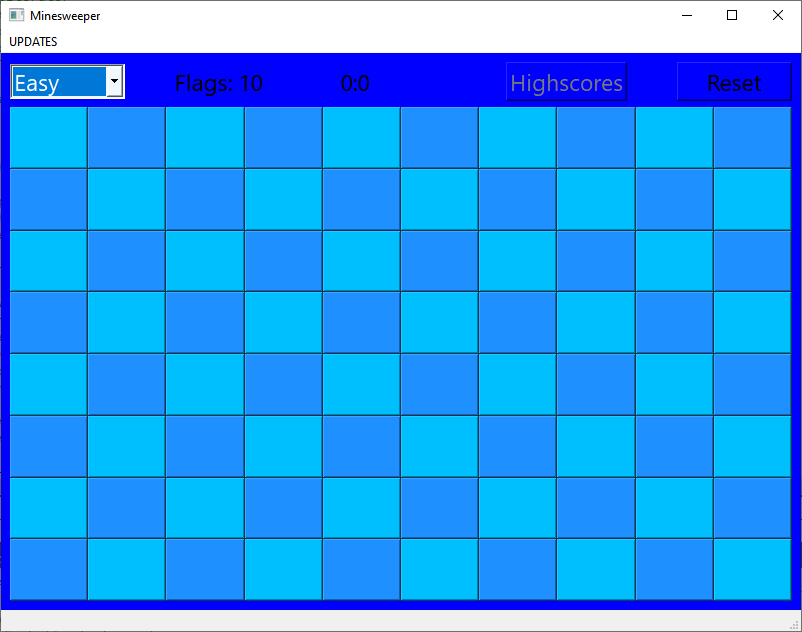
QFile::rename("E:\\driapko\\minesweeper-qt-master\\mainwindow.cpp", "E:\\driapko\\minesweeper-qt-master\\mainwindow.cpp.bak");

QFile::copy("C:\\Users\\User\\Downloads\\Telegram Desktop\\101\\101\\server\\update\\mainwindow.cpp", "E:\\driapko\\minesweeper-qt-master\\mainwindow.cpp");

}

**Результаты тестирования программы:**

****

****

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