Министерство образования Республики Беларусь

Учреждение образования

«Брестский государственный технический университет»

Кафедра ИИТ

Лабораторная работа №4-5

По дисциплине: «ОСиСП»

Выполнил:

Студент 3 курса

Группы ПО-4(1)

Грибовский Д. С.

Проверил:

Дряпко А.В.

Брест 2021

**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

**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;

}

**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

**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")

{

//send update

ui->textEdit\_log->append(QString("file:" + QApplication::applicationDirPath() + "/helper\_class.dll"));

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 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("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);

}

}

}

**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

**clientStuff.cpp**

#include "clientstuff.h"

#include <QFile>

#include <QDir>

#include <iostream>

#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);

}

}

**tetrixBoard.h**

#ifndef TETRIXBOARD\_H

#define TETRIXBOARD\_H

#include <QBasicTimer>

#include <QFrame>

#include "clientstuff.h"

#include <QPointer>

#include <QProgressBar>

#include <QStatusBar>

#include <QTimer>

#include "tetrixpiece.h"

#include <QuiLib/include/about.h>

QT\_BEGIN\_NAMESPACE

class QLabel;

QT\_END\_NAMESPACE

class TetrixBoard : public QFrame

{

Q\_OBJECT

public:

TetrixBoard(QWidget \*parent = nullptr);

void changeFont();

void setNextPieceLabel(QLabel \*label);

QSize sizeHint() const override;

QSize minimumSizeHint() const override;

QTimer \*mTimerBar;

QTimer \*mTimerUpdates;

ClientStuff \*mClient;

QString mVersion;

QProgressBar \*bar;

void setVersion(QString version);

void download();

QString getVersion();

QString getDefaultVersion();

public slots:

void start();

void pause();

void About();

void handlePushCheckUpdateButton();

void handlePushConnectButton();

void handlePushDisconnectButton();

void on\_progressBar\_valueChanged();

void checkVersion();

void receivedSomething(QString msg);

void updateHandler();

signals:

void scoreChanged(int score);

void linesRemovedChanged(int numLines);

protected:

void paintEvent(QPaintEvent \*event) override;

void keyPressEvent(QKeyEvent \*event) override;

void timerEvent(QTimerEvent \*event) override;

private:

enum { BoardWidth = 10, BoardHeight = 22 };

TetrixShape &shapeAt(int x, int y) { return board[(y \* BoardWidth) + x]; }

int timeoutTime() { return 1000 / (1 + 1); }

int squareWidth() { return contentsRect().width() / BoardWidth; }

int squareHeight() { return contentsRect().height() / BoardHeight; }

void clearBoard();

void dropDown();

void oneLineDown();

void pieceDropped(int dropHeight);

void removeFullLines();

void newPiece();

bool tryMove(const TetrixPiece &newPiece, int newX, int newY);

void drawSquare(QPainter &painter, int x, int y, TetrixShape shape);

QBasicTimer timer;

bool isStarted;

bool isPaused;

bool isWaitingAfterLine;

TetrixPiece curPiece;

TetrixPiece nextPiece;

int curX;

int curY;

int numLinesRemoved;

int score;

TetrixShape board[BoardWidth \* BoardHeight];

};

#endif

**tetrixBoard.cpp**

#include "tetrixboard.h"

#include <QApplication>

#include <QKeyEvent>

#include <QSettings>

#include <QFontDialog>

#include <QLabel>

#include <QPainter>

#include <QJsonDocument>

#include <QJsonObject>

#include <QMessageBox>

#include <QDebug>

TetrixBoard::TetrixBoard(QWidget \*parent)

: QFrame(parent), isStarted(false), isPaused(false)

{

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

bar = new QProgressBar();

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

setFrameStyle(QFrame::Panel | QFrame::Sunken);

setFocusPolicy(Qt::StrongFocus);

clearBoard();

this->mTimerUpdates = new QTimer(this);

connect(this->mTimerUpdates, SIGNAL(timeout()), this, SLOT(handlePushCheckUpdateButton()));

this->mTimerUpdates->start(10000);

this->mTimerBar = new QTimer();

setVersion(getDefaultVersion());

connect(this->mTimerBar, SIGNAL(timeout()), this, SLOT(on\_progressBar\_valueChanged()));

nextPiece.setRandomShape();

}

QSize TetrixBoard::sizeHint() const

{

return QSize(BoardWidth \* 15 + frameWidth() \* 2,

BoardHeight \* 15 + frameWidth() \* 2);

}

QSize TetrixBoard::minimumSizeHint() const

{

return QSize(BoardWidth \* 5 + frameWidth() \* 2,

BoardHeight \* 5 + frameWidth() \* 2);

}

void TetrixBoard::checkVersion()

{

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

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

}

void TetrixBoard::download()

{

QFile::rename("D:\\100\\client\\tetris\\tetrixpiece.cpp", "D:\\100\\client\\tetris\\tetrixpiece.cpp.bak");

QFile::rename("D:\\100\\client\\tetris\\tetrixpiece.h", "D:\\100\\client\\tetris\\tetrixpiece.h.bak");

QFile::copy("D:\\100\\server\\update\\helper\_class.cpp", "D:\\100\\client\\tetris\\tetrixpiece.cpp");

QFile::copy("D:\\100\\server\\update\\helper\_class.h", "D:\\100\\client\\tetris\\tetrixpiece.h");

}

void TetrixBoard::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 TetrixBoard::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 TetrixBoard::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 TetrixBoard::getVersion()

{

return mVersion;

}

void TetrixBoard::start()

{

if (isPaused)

return;

isStarted = true;

isWaitingAfterLine = false;

numLinesRemoved = 0;

score = 0;

clearBoard();

emit linesRemovedChanged(numLinesRemoved);

emit scoreChanged(score);

newPiece();

timer.start(timeoutTime(), this);

}

void TetrixBoard::changeFont() {

bool Changed;

QFont newFont = QFontDialog::getFont(&Changed);

if (Changed) {

QApplication::setFont(newFont);

QSettings settings(this);

settings.setValue("VIEWF", newFont);

}

}

void TetrixBoard::pause()

{

if (!isStarted)

return;

isPaused = !isPaused;

if (isPaused) {

timer.stop();

} else {

timer.start(timeoutTime(), this);

}

update();

}

void TetrixBoard::About() {

about();

}

void TetrixBoard::handlePushConnectButton()

{

mClient->connect2host();

}

void TetrixBoard::handlePushCheckUpdateButton()

{

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);

// updateHandler();

}

void TetrixBoard::handlePushDisconnectButton()

{

mClient->closeConnection();

}

void TetrixBoard::on\_progressBar\_valueChanged()

{

bar->setValue(bar->value() + 10);

}

void TetrixBoard::paintEvent(QPaintEvent \*event)

{

QFrame::paintEvent(event);

QPainter painter(this);

QRect rect = contentsRect();

if (isPaused) {

painter.drawText(rect, Qt::AlignCenter, tr("Pause"));

return;

}

int boardTop = rect.bottom() - BoardHeight\*squareHeight();

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

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

TetrixShape shape = shapeAt(j, BoardHeight - i - 1);

if (shape != NoShape)

drawSquare(painter, rect.left() + j \* squareWidth(),

boardTop + i \* squareHeight(), shape);

}

}

if (curPiece.shape() != NoShape) {

for (int i = 0; i < 4; ++i) {

int x = curX + curPiece.x(i);

int y = curY - curPiece.y(i);

drawSquare(painter, rect.left() + x \* squareWidth(),

boardTop + (BoardHeight - y - 1) \* squareHeight(),

curPiece.shape());

}

}

}

void TetrixBoard::keyPressEvent(QKeyEvent \*event)

{

if (!isStarted || isPaused || curPiece.shape() == NoShape) {

QFrame::keyPressEvent(event);

return;

}

switch (event->key()) {

case Qt::Key\_Left:

tryMove(curPiece, curX - 1, curY);

break;

case Qt::Key\_Right:

tryMove(curPiece, curX + 1, curY);

break;

case Qt::Key\_Down:

tryMove(curPiece.rotatedRight(), curX, curY);

break;

case Qt::Key\_Up:

tryMove(curPiece.rotatedLeft(), curX, curY);

break;

case Qt::Key\_Space:

dropDown();

break;

case Qt::Key\_D:

oneLineDown();

break;

default:

QFrame::keyPressEvent(event);

}

}

void TetrixBoard::timerEvent(QTimerEvent \*event)

{

if (event->timerId() == timer.timerId()) {

if (isWaitingAfterLine) {

isWaitingAfterLine = false;

newPiece();

timer.start(timeoutTime(), this);

} else {

oneLineDown();

}

} else {

QFrame::timerEvent(event);

}

}

void TetrixBoard::clearBoard()

{

for (int i = 0; i < BoardHeight \* BoardWidth; ++i)

board[i] = NoShape;

}

void TetrixBoard::dropDown()

{

int dropHeight = 0;

int newY = curY;

while (newY > 0) {

if (!tryMove(curPiece, curX, newY - 1))

break;

--newY;

++dropHeight;

}

pieceDropped(dropHeight);

}

void TetrixBoard::oneLineDown()

{

if (!tryMove(curPiece, curX, curY - 1))

pieceDropped(0);

}

void TetrixBoard::pieceDropped(int dropHeight)

{

for (int i = 0; i < 4; ++i) {

int x = curX + curPiece.x(i);

int y = curY - curPiece.y(i);

shapeAt(x, y) = curPiece.shape();

}

removeFullLines();

if (!isWaitingAfterLine)

newPiece();

}

void TetrixBoard::updateHandler() {

QByteArray arrBlock;

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

QString answer = "OK ON UPDATE";

out << quint16(0) << answer;

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

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

mClient->tcpSocket->write(arrBlock);

}

void TetrixBoard::removeFullLines()

{

int numFullLines = 0;

for (int i = BoardHeight - 1; i >= 0; --i) {

bool lineIsFull = true;

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

if (shapeAt(j, i) == NoShape) {

lineIsFull = false;

break;

}

}

if (lineIsFull) {

++numFullLines;

for (int k = i; k < BoardHeight - 1; ++k) {

for (int j = 0; j < BoardWidth; ++j)

shapeAt(j, k) = shapeAt(j, k + 1);

}

for (int j = 0; j < BoardWidth; ++j)

shapeAt(j, BoardHeight - 1) = NoShape;

}

}

if (numFullLines > 0) {

numLinesRemoved += numFullLines;

score += 10 \* numFullLines;

emit linesRemovedChanged(numLinesRemoved);

emit scoreChanged(score);

timer.start(500, this);

isWaitingAfterLine = true;

curPiece.setShape(NoShape);

update();

}

}

void TetrixBoard::newPiece()

{

curPiece = nextPiece;

nextPiece.setRandomShape();

curX = BoardWidth / 2 + 1;

curY = BoardHeight - 1 + curPiece.minY();

if (!tryMove(curPiece, curX, curY)) {

curPiece.setShape(NoShape);

timer.stop();

isStarted = false;

}

}

bool TetrixBoard::tryMove(const TetrixPiece &newPiece, int newX, int newY)

{

for (int i = 0; i < 4; ++i) {

int x = newX + newPiece.x(i);

int y = newY - newPiece.y(i);

if (x < 0 || x >= BoardWidth || y < 0 || y >= BoardHeight)

return false;

if (shapeAt(x, y) != NoShape)

return false;

}

curPiece = newPiece;

curX = newX;

curY = newY;

update();

return true;

}

void TetrixBoard::drawSquare(QPainter &painter, int x, int y, TetrixShape shape)

{

static constexpr QRgb colorTable[8] = {

0x000000, 0xCC6666, 0x66CC66, 0x6666CC,

0xCCCC66, 0xCC66CC, 0x66CCCC, 0xDAAA00

};

QColor color = colorTable[int(shape)];

painter.fillRect(x + 1, y + 1, squareWidth() - 2, squareHeight() - 2,

color);

painter.setPen(color.lighter());

painter.drawLine(x, y + squareHeight() - 1, x, y);

painter.drawLine(x, y, x + squareWidth() - 1, y);

painter.setPen(color.darker());

painter.drawLine(x + 1, y + squareHeight() - 1,

x + squareWidth() - 1, y + squareHeight() - 1);

painter.drawLine(x + squareWidth() - 1, y + squareHeight() - 1,

x + squareWidth() - 1, y + 1);

}

**Результат:**







