Министерство образования Республики Беларусь

Учреждение образования

«Брестский государственный технический университет»

**Отчёт**

Лабораторная работа №5

По дисциплине ОСиСП

«Многопоточное приложение»

Выполнил

Студент ФЭИС

3-го курса, группы ПО-5

Бриштен Д.С.

Проверила Дряпко А.В.

Брест 2021

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

Вариант 4

Задание. 1) Основное задание заключается в доработке функционала обновления, разработка которого производилась в ЛР No4. Нужно интегрировать указанную функцию в само приложение, без использования стороннего клиента. При этом серверная часть приложения остается без изменений (возможны некоторые доработки сервера, без изменения общей клиент-серверной архитектуры);  
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 обновляемых компонентов (в соответствии с вариантом задания) одного-двух крупных файлов с произвольным содержимым (например, видео).

6) Обновляемый компонент по варианту: DLL - новый уровень.

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

Файлы интерфейса

**thread.h**

#ifndef THREAD\_H

#define THREAD\_H

#include <QObject>

#include <QTcpSocket>

#include <QThread>

#include <QFile>

#include <QTimer>

#include <QDir>

class **Thread** : public QObject

{

Q\_OBJECT

public:

**Thread**();

QTcpSocket\* socket;

QByteArray Data;

int progress = 0;

int fileNum = 2;

QTimer \*timer;

public slots:

void **sockReady**();

void **sockDisc**();

void **run**();

signals:

void **time\_load**();

void **load\_end**(QString);

private:

};

## #endif // THREAD\_H

**window.h**

#ifndef WINDOW\_H

#define WINDOW\_H

#include <QDialog>

#include <QtCore>

#include <QtGui>

#include <QMenuBar>

#include <QPainter>

#include <QString>

#include <QTimer>

#include <QTime>

#include <QThread>

#include <QStatusBar>

#include <QProgressBar>

#include <QTcpSocket>

#include <QMessageBox>

#include <QDebug>

#include <QListIterator>

#include "pacman.h"

#include "map.h"

#include "ball.h"

#include "ghost.h"

#include "powerball.h"

#include "textdrawing.h"

#include "fontPluginInterface.h"

#include "fontPluginInterface2.h"

#include "thread.h"

#include "dialogwindow.h"

typedef void (\*about)(QWidget \*painter);

namespace **Ui** {

class **window**;

}

class **window** : public QDialog

{

Q\_OBJECT

public:

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

void **pacman\_move**();

void **ghostsmove**();

void **ghostsmove1**();

void **ghostsmove2**();

void **moveghostsinrect2**();

void **moveghostsinrect1**();

void **moveghostsinrect**();

void **checklost**();

void **delay**();

void **start\_Game**();

void **end\_Game**();

void **loadPlugins**();

void **loadLevel**();

bool **getFlag**();

void **stopConn**();

QTcpSocket\* socket;

QByteArray Data;

Pacman \*pacman;

Ghost \*ghost;

Ghost \*ghost1;

Ghost \*ghost2;

Map \*pac\_map;

Ball \*ball;

PowerBall \*powerball;

Textdrawing \*text;

about f2;

int level = 1, fileNum = 2;

int progress = 0;

int pacx,pacy,direction,nextdirection;

int gosx,gosy,ghostdir,nextghostdir;

int gosx1,gosy1,ghostdir1,nextghostdir1;

int gosx2,gosy2,ghostdir2,nextghostdir2;

bool moving,ghostmoving,ghostmoving1,ghostmoving2;

bool scared,scared1,scared2;

int score,state3,state2,state1,state;

bool start,delayb;

bool ghoststart,ghoststart1,ghoststart2;

bool playing;

bool updateAuto = false;

QVector<QPoint> ballpoints;

QVector<QPoint> Powerballpoints;

QThread \*sleeper;

~***window***();

public slots:

void **updater**();

void **ghostupdater**();

void **About**();

void **changeAppFont**(QFont font);

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

void **sockReady**();

void **sockDisc**();

void **onRecentOpenFiles**(QAction \*action);

void **changeLevel**(int x);

void **defaultLevel**();

void **setFlag**();

void **setFlag\_2**();

void **loading**(QString mes);

void **showProgress**();

void **Progress**();

protected:

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

private:

Ui::window \*ui;

QGraphicsScene \*scene;

QTimer \*timer, \*timer\_2;

QTimer \*ghoststimer;

QMenuBar \*menuBar;

QMenu \*style, \*updated, \*levelMenu;

QThread \*Update;

DialogWindow dialog;

QProgressBar\* bar;

};

## #endif // WINDOW\_H

**myserver.h**

#include <QTcpSocket>

#include <QFile>

#include <QDir>

#include <QtCore>

class **myserver**: public QTcpServer

{

Q\_OBJECT

public:

**myserver**();

~***myserver***();

QTcpServer \*tcpSever;

QTcpSocket\* socket;

QByteArray Data;

public slots:

void **startServer**();

void ***incomingConnection***(int socketDescriptor);

void **sockReady**();

void **sockDisc**();

};

## #endif // MYSERVER\_H

Файлы реализации

**thread.cpp**

#include "thread.h"

#include "window.h"

#include <QTextCodec>

#include <QMessageBox>

Thread::**Thread**() : QObject() {

}

void Thread::**sockReady**() {

if(socket->*waitForConnected*(100)) {

QByteArray testing;

while (socket->*waitForReadyRead*(3000))

{

while(socket->*bytesAvailable*() > 0)

{

testing.append(socket->readAll());

socket->flush();

}

}

if(testing != "")

{

emit time\_load();

QTextCodec \*codec = QTextCodec::codecForName("CP866");

QString s = codec->toUnicode(testing);

QStringList list = s.split("\n\n\n\n\n");

for (int i = 0; i < list.size() - 1; i++)

{

QFile target;

target.setFileName("D:/5 semester/OSISP/Lab5/Pacman-build/debug/updates/level" + QString::number(fileNum) + ".dll");

if (!target.*open*(QIODevice::WriteOnly | QIODevice::Append)) {

qDebug() << "Can't open file for written";

return;

}

QByteArray output;

output.append(codec->fromUnicode(list[i]));

target.write(output);

target.*close*();

output.clear();

fileNum += 1;

}

}

else

{

emit load\_end("Установлены последние обновления");

}

}

}

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);

## }

**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 socketDescriptor)

{

socket = new QTcpSocket(this);

socket->*setSocketDescriptor*(socketDescriptor);

connect(socket,SIGNAL(readyRead()),this,SLOT(sockReady()));

connect(socket,SIGNAL(disconnected()),this,SLOT(sockDisc()));

qDebug()<<socketDescriptor<<" Client connected";

QByteArray data;

QFileInfoList checkUpdate = QDir("D:\\5 semester\\OSISP\\Lab5\\Pacman-build\\debug\\updates").entryInfoList(QStringList()<< "\*.dll",QDir::Files);

if (checkUpdate.size() == 0) {

QFileInfoList dirContent = QDir(qApp->applicationDirPath() + "/plugins").entryInfoList(QStringList()<< "\*",QDir::Files);

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

QFileInfo fileInfo = dirContent.at(i);

QString fileName = dirContent.at(i).fileName();

qDebug() << fileName;

QFile file(dirContent.at(i).filePath());

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

data += file.readAll();

data += "\n\n\n\n\n";

}

}

if(data.isEmpty())

{

data = "empty";

}

socket->write(data);

qDebug()<<"Send client connect status - YES";

}

void myserver::**sockReady**()

{

// Принимаем что-то от клиента

}

void myserver::**sockDisc**()

{

qDebug()<<"Disconnect";

socket->deleteLater();

## }

**window.cpp**

#include "window.h"

#include "ui\_window.h"

#include "textdrawing.h"

#include "thread.h"

#include <QThread>

#include <QMenuBar>

#include <QMenu>

#include <QVBoxLayout>

#include <QFontDialog>

#include <QDebug>

#include <QProgressBar>

window::**window**(QWidget \*parent) :

QDialog(*parent*),

ui(new Ui::window)

{

score=0;

gosx=450/2;

gosy=480/2;

gosx1=450/2;

gosy1=480/2;

gosx2=450/2;

gosy2=480/2;

pacx=410/2;

pacy=360;

ghostmoving=false;

ghostmoving1=false;

ghostmoving2=false;

scared=false;

scared1=false;

scared2=false;

start=false;

direction=0;

moving=false;

delayb=false;

playing=false;

ui->setupUi(this);

scene = new QGraphicsScene(this);

ui->graphicsView->setScene(*scene*);

ui->graphicsView->setRenderHint(QPainter::Antialiasing);

scene->setSceneRect(0,0,640,480);

ui->graphicsView->setSceneRect(scene->sceneRect());

pacman=new Pacman;

pacx=410/2;

pacy=410;

text=new Textdrawing;

text->over=false;

scene->addItem(*text*);

text->setZValue(7);

menuBar = new QMenuBar();

QAction \*aboutAction = menuBar->addAction(tr("О программе"));

connect(aboutAction, SIGNAL(triggered()), this, SLOT(About()));

style = new QMenu(tr("Стили"), this);

menuBar->addMenu(*style*);

QAction \*updateAction = menuBar->addAction(tr("Обновления"));

connect(updateAction, SIGNAL(triggered()), this, SLOT(on\_pushButton\_clicked()));

levelMenu = new QMenu(tr("Уровни"), this);

menuBar->addMenu(*levelMenu*);

QAction\* actionLevelEasy = levelMenu->addAction(tr("Лёгкий"));

connect(actionLevelEasy, SIGNAL(triggered()), this, SLOT(defaultLevel()));

QMenu \*politic = new QMenu(tr("Политика"), this);

menuBar->addMenu(*politic*);

QAction\* actionManually = politic->addAction(tr("Вручную"));

QAction\* actionAutomatic = politic->addAction(tr("Автоматически"));

connect(actionManually, SIGNAL(triggered()), this, SLOT(setFlag()));

connect(actionAutomatic, SIGNAL(triggered()), this, SLOT(setFlag\_2()));

QVBoxLayout \*mainLayout = new QVBoxLayout;

mainLayout->setMenuBar(*menuBar*);

setLayout(*mainLayout*);

loadPlugins();

socket = new QTcpSocket(this);

connect(socket,SIGNAL(readyRead()),this,SLOT(sockReady()));

connect(socket,SIGNAL(disconnected()),this,SLOT(sockDisc()));

loadLevel();

timer = new QTimer(this);

connect(timer, SIGNAL(timeout()), this, SLOT(on\_pushButton\_clicked()));

}

void window::**showProgress**() {

timer\_2 = new QTimer(this);

connect(timer\_2, SIGNAL(timeout()), this, SLOT(Progress()));

timer\_2->start(1000);

bar = new QProgressBar;

bar->setRange(0, 100);

bar->setMinimumWidth(400);

bar->setAlignment(Qt::AlignCenter);

bar->show();

}

void window::**sockReady**()

{

if (socket->*waitForConnected*(500))

{

if(getFlag())

{

QMessageBox::information(this, "Информация", "Автоматическая проверка обновлений");

sockDisc();

socket->*waitForDisconnected*(1);

QThread \*thread= new QThread;

Thread \*my = new Thread();

my->moveToThread(*thread*);

connect(my, SIGNAL(time\_load()), this, SLOT(showProgress()));

connect(my, SIGNAL(load\_end(QString)), this, SLOT(loading(QString)));

connect(thread, SIGNAL(started()), my, SLOT(run()));

thread->start();

}

else

{

QMessageBox \*msgBox = new QMessageBox(QMessageBox::Information,"Обновление","Установить обновления?",

QMessageBox::Ok| QMessageBox::No);

if(msgBox->*exec*() == QMessageBox::Ok)

{

sockDisc();

socket->*waitForDisconnected*(1);

QThread \*thread= new QThread;

Thread \*my = new Thread();

my->moveToThread(*thread*);

connect(my, SIGNAL(time\_load()), this, SLOT(showProgress()));

connect(my, SIGNAL(load\_end(QString)), this, SLOT(loading(QString)));

connect(thread, SIGNAL(started()), my, SLOT(run()));

thread->start();

}

}

}

}

void window::**loading**(QString mes)

{

QMessageBox::information(this, "Информация", mes);

}

void window::**setFlag**()

{

updateAuto = false;

timer->stop();

}

void window::**setFlag\_2**()

{

updateAuto = true;

timer->start(10000);

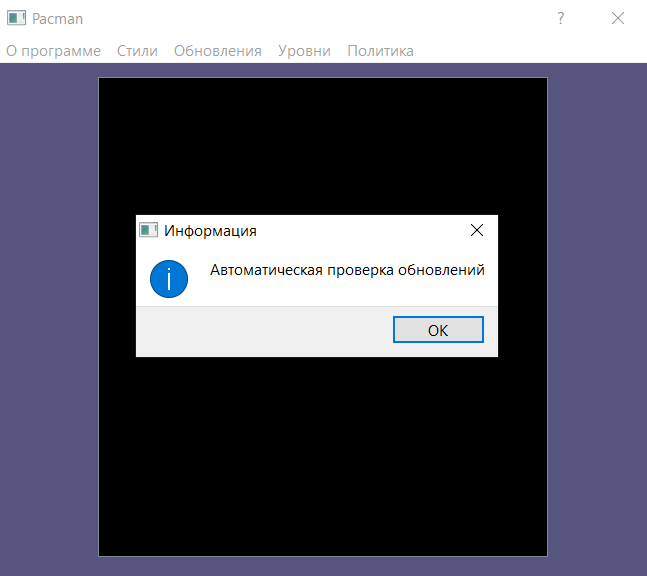
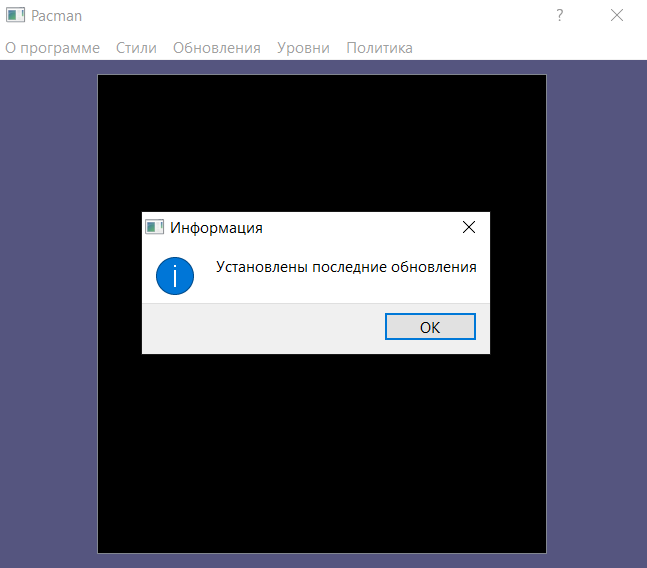
}

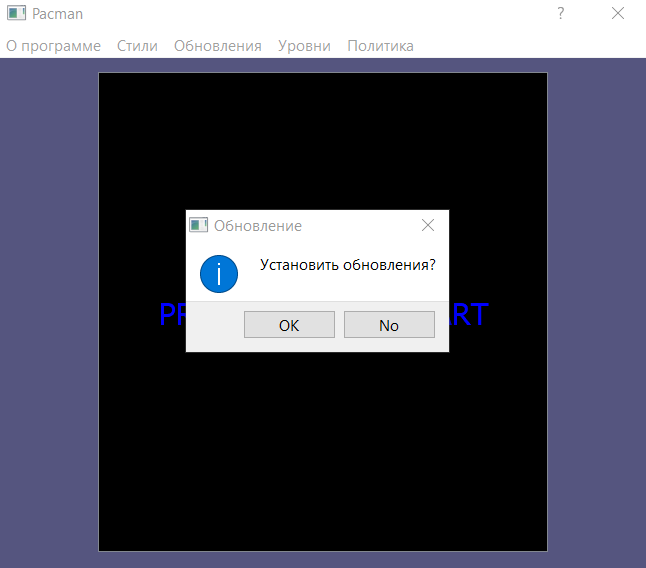
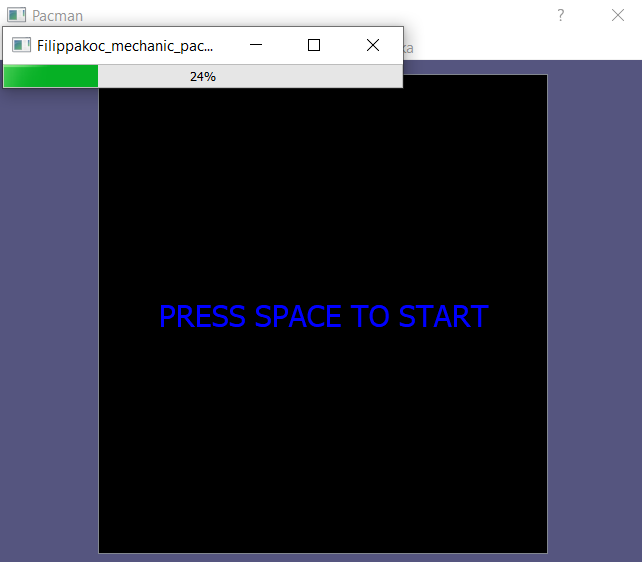
bool window::**getFlag**()

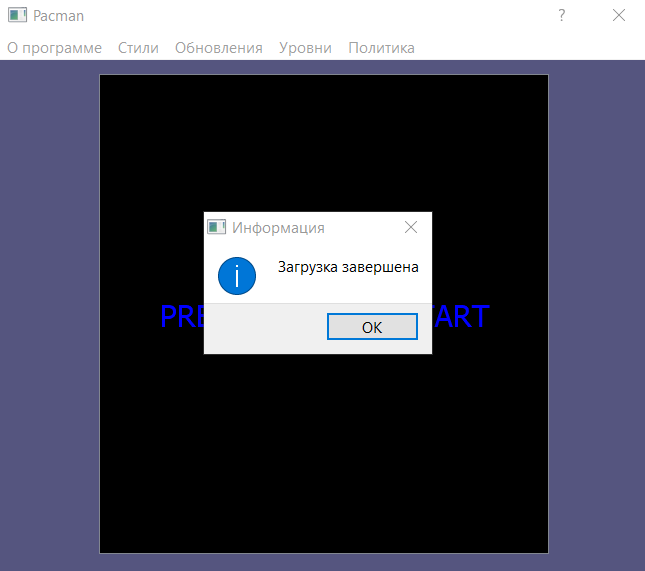
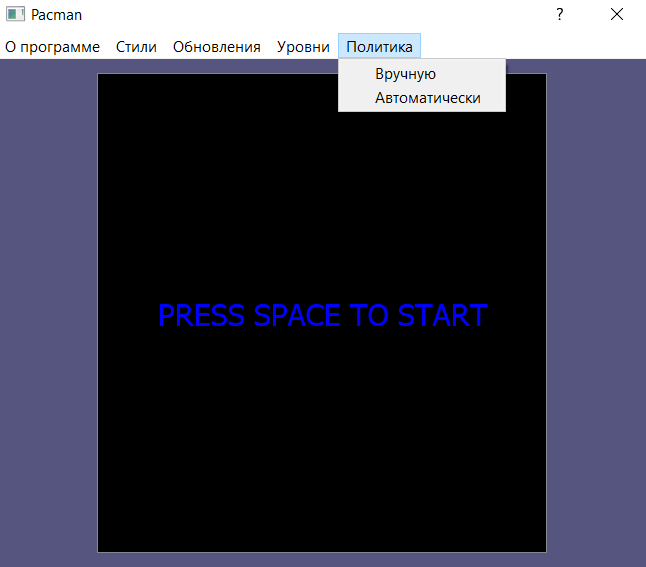
{

return updateAuto;

## }

** **

** **

** **

Вывод: Произведена доработка функционала обновления: процесс обновления осуществляется автоматически по таймеру с использованием отдельного потока и демонстрацией прогресса.