**Решение задания 3го тура**

Project.pro

QT += core gui

greaterThan(QT\_MAJOR\_VERSION, 4): QT += widgets

CONFIG += c++14

UI\_DIR = $$PWD

# The following define makes your compiler emit warnings if you use

# any Qt feature that has been marked deprecated (the exact warnings

# depend on your compiler). Please consult the documentation of the

# deprecated API in order to know how to port your code away from it.

DEFINES += QT\_DEPRECATED\_WARNINGS

# You can also make your code fail to compile if it uses deprecated APIs.

# In order to do so, uncomment the following line.

# You can also select to disable deprecated APIs only up to a certain version of Qt.

#DEFINES += QT\_DISABLE\_DEPRECATED\_BEFORE=0x060000 # disables all the APIs deprecated before Qt 6.0.0

SOURCES += \

field.cpp \

fill\_utils.cpp \

main.cpp \

main\_form.cpp \

PathFinding.cpp

HEADERS += \

field.h \

fill\_utils.h \

main\_form.h \

PathFinding.h

FORMS += \

field.ui \

main\_form.ui \

FORMS += \

main\_form.ui

# Default rules for deployment.

qnx: target.path = /tmp/$${TARGET}/bin

else: unix:!android: target.path = /opt/$${TARGET}/bin

!isEmpty(target.path): INSTALLS += target

# Field.cpp

#include "ui\_field.h"

#include <QPainter>

#include "field.h"

Field::Field(

QVector<QLine> roads,

QVector<QPoint> positions,

QVector<QLine> routes,

QPoint mainPoint,

QWidget \*parent

) :

QWidget(parent),

ui(new Ui::Field)

{

ui->setupUi(this);

this->roads = roads;

this->positions = positions;

this->routes = routes;

this->mainPoint = mainPoint;

}

Field::~Field()

{

delete ui;

}

void Field::paintEvent(QPaintEvent \*event) {

Q\_UNUSED(event);

QPainter painter(this);

painter.scale(10.0, 10.0);

QPen pen;

// Отрисовываем дороги

pen.setColor(Qt::black);

pen.setWidth(2);

painter.setPen(pen);

for(auto road: roads)

painter.drawLine(road);

// Отрисовываем пути

pen.setColor(Qt::green);

pen.setWidth(1);

painter.setPen(pen);

for(auto route: routes)

painter.drawLine(route);

// Отрисовываем позиции

pen.setColor(Qt::blue);

pen.setWidth(3);

painter.setPen(pen);

for(auto pos: positions)

painter.drawPoint(pos);

// Отрисовываем базу

pen.setColor(Qt::red);

pen.setWidth(5);

painter.setPen(pen);

painter.drawPoint(mainPoint);

}

# Field.h

#ifndef FIELD\_OLD\_H

#define FIELD\_OLD\_H

#include <QWidget>

namespace Ui {

class Field;

}

class Field : public QWidget

{

Q\_OBJECT

public:

Field(

QVector<QLine> roads,

QVector<QPoint> positions,

QVector<QLine> routes,

QPoint mainPoint,

QWidget \*parent = nullptr

);

~Field();

protected:

void paintEvent(QPaintEvent \*event) override;

private:

Ui::Field \*ui;

QVector<QLine> roads;

QVector<QPoint> positions;

QVector<QLine> routes;

QPoint mainPoint;

};

#endif // FIELD\_OLD\_H

# Field.ui

<ui version="4.0" >

<author></author>

<comment></comment>

<exportmacro></exportmacro>

<class>Form</class>

<widget class="QWidget" name="Form" >

<property name="geometry" >

<rect>

<x>0</x>

<y>0</y>

<width>400</width>

<height>300</height>

</rect>

</property>

<property name="windowTitle" >

<string>Form</string>

</property>

</widget>

<pixmapfunction></pixmapfunction>

<connections/>

</ui>

# fill\_utils.cpp

#include "fill\_utils.h"

void fillArray(QStringList &list, QVector<int> &array)

{

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

{

bool is\_ok = true;

array.push\_back(list[i].toInt(&is\_ok));

if (!is\_ok) { qDebug() << "Convert error"; }

}

}

void fillArray(QStringList &list, QVector<QPoint> &array)

{

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

{

bool is\_ok = true;

auto cord\_paths = list[i].split(";");

array.push\_back(QPoint(cord\_paths[0].toInt(), cord\_paths[1].toInt()));

if (!is\_ok) { qDebug() << "Convert error"; }

}

}

void fillArray(QStringList &list, QVector<QLine> &array)

{

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

{

bool is\_ok = true;

auto cord\_paths = list[i].split(";");

const auto start = QPoint(cord\_paths[0].toInt(), cord\_paths[1].toInt());

const auto end = QPoint(cord\_paths[2].toInt(), cord\_paths[3].toInt());

auto line = QLine(start, end);

array.push\_back(line);

if (!is\_ok) { qDebug() << "Convert error"; }

}

}

# fill\_utils.h

#ifndef FILL\_UTILS\_H

#define FILL\_UTILS\_H

#include <QLine>

#include <QPoint>

#include <QVector>

#include <QStringList>

#include <QDebug>

void fillArray(QStringList &list, QVector<QLine> &array);

void fillArray(QStringList &list, QVector<QPoint> &array);

void fillArray(QStringList &list, QVector<int> &array);

#endif // FILL\_UTILS\_H

# main.cpp

#include "main\_form.h"

#include <QApplication>

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

{

QApplication a(argc, argv);

MainForm w;

w.show();

return a.exec();

}

# main\_form.cpp

#include "main\_form.h"

MainForm::MainForm(QWidget \*parent) :

QDialog(parent),

ui(new Ui::MainForm)

{

ui->setupUi(this);

this->input\_conf = InputFile();

connect(ui->importPushButton, SIGNAL(clicked()), this, SLOT(handleImportButtonClicked()));

connect(ui->startPushButton, SIGNAL(clicked()), this, SLOT(handleStartButtonClicked()));

}

QVector<int> MainForm::GetFuelConsumption(QVector<int> &path\_lengths)

{

QVector<int> fuel\_consumptions;

for (int i(0); i < path\_lengths.size(); i++)

{

fuel\_consumptions.push\_back(path\_lengths[i] \* input\_conf.squad\_fuel\_consumption[i]);

}

return fuel\_consumptions;

}

float MainForm::GetMaxTime(QVector<int> &path\_lengths)

{

int maximal = -10000;

for (int i(0); i < path\_lengths.size(); i++)

maximal = std::max(path\_lengths[i] / input\_conf.squad\_speeds[i], maximal);

return maximal;

}

void MainForm::WriteAnswers(QVector<int> &fuel\_consumptions, int max\_time)

{

QString fuel\_consumptions\_result;

for (int i(0); i < fuel\_consumptions.size(); i++)

fuel\_consumptions\_result += (QString("%1").arg(fuel\_consumptions[i]) + " ");

fuel\_consumptions\_result.chop(1);

qDebug() << fuel\_consumptions\_result;

QString filename = "OUTPUT.TXT";

QFile file( filename );

if ( file.open(QIODevice::WriteOnly) )

{

qDebug() << 1;

QTextStream stream( &file );

stream << fuel\_consumptions\_result << '\n';

stream << QString("%1").arg(max\_time);

}

}

void MainForm::handleStartButtonClicked()

{

auto shortest\_path = WaveAlgo(input\_conf.squads\_cnt, input\_conf.dest\_cords, input\_conf.squad\_cords, input\_conf.roads\_cnt, input\_conf.roads\_cords);

QVector<QLine> shortest\_roads;

QVector<int> path\_lenghts;

std::tie(shortest\_roads, path\_lenghts) = shortest\_path;

auto fuel\_consumtpions = GetFuelConsumption(path\_lenghts);

auto max\_time = GetMaxTime(path\_lenghts);

WriteAnswers(fuel\_consumtpions, max\_time);

this->field\_dlg = new Field(

input\_conf.roads\_cords,

input\_conf.squad\_cords,

shortest\_roads,

input\_conf.dest\_cords

);

this->field\_dlg->show();

}

void MainForm::handleImportButtonClicked()

{

QString file\_path = QFileDialog::getOpenFileName(this,

QString::fromUtf8("Открыть файл"),

QDir::currentPath(),

"Text (\*.txt)");

if (file\_path.length() == 0)

return;

QFile input\_file(file\_path);

if (input\_file.open(QIODevice::ReadOnly))

{

QTextStream in(&input\_file);

QString line;

QStringList parts;

// read squads\_cnt

line = in.readLine();

input\_conf.squads\_cnt = line.toInt();

// read dest\_cords

line = in.readLine();

parts = line.split(";");

input\_conf.dest\_cords = QPoint(parts[0].toInt(), parts[1].toInt());

// read squad\_cords

line = in.readLine();

parts = line.split(" ");

fillArray(parts, input\_conf.squad\_cords);

// read squad\_types

line = in.readLine();

parts = line.split(" ");

fillArray(parts, input\_conf.squad\_types);

// read squad\_speed

line = in.readLine();

parts = line.split(" ");

fillArray(parts, input\_conf.squad\_speeds);

// read squad\_fuel\_consumption

line = in.readLine();

parts = line.split(" ");

fillArray(parts, input\_conf.squad\_fuel\_consumption);

// read roads\_cnt

line = in.readLine();

input\_conf.roads\_cnt = line.toInt();

// read roads\_cords

line = in.readLine();

parts = line.split(" ");

fillArray(parts, input\_conf.roads\_cords);

ui->startPushButton->setEnabled(true);

}

else

{

QMessageBox::critical(this, "Ошибка при импорте", "Не удалось открыть выбранный файл");

}

}

MainForm::~MainForm()

{

delete field\_dlg;

delete ui;

}

# main\_form.h

#ifndef MAIN\_FORM\_H

#define MAIN\_FORM\_H

#include <QDialog>

#include <QString>

#include <QDebug>

#include <QFileDialog>

#include <QVector>

#include <QLine>

#include <QPoint>

#include <QIODevice>

#include <QMessageBox>

#include <QTextStream>

#include <QStringList>

#include "field.h"

#include "main\_form.h"

#include "ui\_main\_form.h"

#include "fill\_utils.h"

#include "PathFinding.h"

namespace Ui {

class MainForm;

}

struct InputFile

{

int squads\_cnt;

QPoint dest\_cords;

QVector<QPoint> squad\_cords;

QVector<int> squad\_types;

QVector<int> squad\_speeds;

QVector<int> squad\_fuel\_consumption;

int roads\_cnt;

QVector<QLine> roads\_cords;

};

class MainForm : public QDialog

{

Q\_OBJECT

public:

explicit MainForm(QWidget \*parent = nullptr);

~MainForm();

private slots:

void handleStartButtonClicked();

void handleImportButtonClicked();

private:

Ui::MainForm \*ui;

Field \*field\_dlg;

InputFile input\_conf;

QVector<int> GetFuelConsumption(QVector<int> &path\_lengths);

float GetMaxTime(QVector<int> &path\_lengths);

void WriteAnswers(QVector<int> &fuel\_consumptions, int max\_time);

};

#endif // MAIN\_FORM\_H

# main\_form.ui

<?xml version="1.0" encoding="UTF-8"?>

<ui version="4.0">

<class>MainForm</class>

<widget class="QDialog" name="MainForm">

<property name="geometry">

<rect>

<x>0</x>

<y>0</y>

<width>303</width>

<height>74</height>

</rect>

</property>

<property name="windowTitle">

<string>Задание №3</string>

</property>

<layout class="QHBoxLayout" name="horizontalLayout">

<item>

<layout class="QVBoxLayout" name="verticalLayout">

<item>

<widget class="QPushButton" name="importPushButton">

<property name="text">

<string>Импорт входных данных</string>

</property>

</widget>

</item>

<item>

<widget class="QPushButton" name="startPushButton">

<property name="enabled">

<bool>false</bool>

</property>

<property name="text">

<string>Запуск</string>

</property>

</widget>

</item>

</layout>

</item>

</layout>

</widget>

<resources/>

<connections/>

</ui>

# PathFinding.cpp

#define \_CRT\_SECURE\_NO\_WARNINGS

#include "PathFinding.h"

PathAlgoResult WaveAlgo( int N\_p, QPoint finish\_pos, QVector<QPoint> start\_pos, int N\_r, QVector<QLine> roads)

{

int x1 = 0, y1 = 0, x2 = 0, y2 = 0;

int paths\_matrix[maxcoord][maxcoord] = { 0 };

QVector <QLine> shortest\_paths;

QVector <int> path\_lengths;

generate\_paths\_matrix(roads, paths\_matrix);

for( int i = 0; i < N\_p; i++ )

{

path\_lengths.push\_back(find\_path( paths\_matrix, start\_pos[i], finish\_pos, shortest\_paths ));

}

return std::make\_tuple( shortest\_paths, path\_lengths );;

}

void generate\_paths\_matrix( QVector <QLine>& roads, int paths[][maxcoord] )

{

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

{

for( int j = roads[i].x1(); j <= roads[i].x2(); j++ )

{

for( int k = roads[i].y1(); k <= roads[i].y2(); k++ )

{

paths[k][j] = 1;

}

}

}

}

int find\_path( int paths\_matrix[][maxcoord], struct QPoint start\_pos, struct QPoint finish\_point, QVector<QLine> &shortest\_paths )

{

int DRP[maxcoord][maxcoord] = { 0 };

int waving = 1;

int start\_x = start\_pos.x(), start\_y = start\_pos.y(), end\_x = finish\_point.x(), end\_y = finish\_point.y();

int x, y;

int wave = 1;

int LEN=0;

int x\_prev, y\_prev;

QLine direction;

DRP[start\_y][start\_x] = 1;

while( waving )

{

waving = 0;

for( int i = 0; ( i < maxcoord ) && ( ( end\_x + end\_y ) < 2 \* maxcoord - 2 ); i++ )

{

for( int j = 0; ( j < maxcoord ) && ( ( end\_x + end\_y ) < 2 \* maxcoord - 2 ); j++ )

{

if( DRP[i][j] == wave )

{

waving = 1;

if( ( i == end\_y ) && ( j == end\_x ) )

{

printf( "Waves to reach finish: %d\n", DRP[i][j] - 1 );

LEN = DRP[i][j] - 1;

waving = 0;

break;

}

if( ( ( i + 1 ) < maxcoord ) && ( DRP[i + 1][j] == 0 ) && ( paths\_matrix[i + 1][j] == 1 ) ) DRP[i + 1][j] = wave + 1;

if( ( ( i - 1 ) >= 0 ) && ( DRP[i - 1][j] == 0 ) && ( paths\_matrix[i - 1][j] == 1 ) ) DRP[i - 1][j] = wave + 1;

if( ( ( j + 1 ) < maxcoord ) && ( DRP[i][j + 1] == 0 ) && ( paths\_matrix[i][j+1] == 1 ) ) DRP[i][j + 1] = wave + 1;

if( ( ( j - 1 ) >= 0 ) && ( DRP[i][j - 1] == 0 ) && ( paths\_matrix[i][j-1] == 1 ) ) DRP[i][j - 1] = wave + 1;

}

}

}

wave++;

}

x = end\_x;

y = end\_y;

if( x + y > 2 \* maxcoord - 2 ) return 0;

wave = DRP[end\_y][end\_x];

x\_prev = end\_x;

y\_prev = end\_y;

while( 1 )

{

if( ( ( y + 1 ) < maxcoord ) && ( DRP[y + 1][x] == wave - 1 ) )

{

if( ( x == start\_x ) && ( (y) == start\_y ) ) break;

direction = { x\_prev, y\_prev, x, y + 1 };

y++;

}

else if( ( ( y - 1 ) >= 0 ) && ( DRP[y - 1][x] == wave - 1 ) )

{

if( ( x == start\_x ) && ( (y) == start\_y ) ) break;

direction = { x\_prev, y\_prev, x, y - 1 };

y--;

}

else if( ( ( x + 1 ) < maxcoord ) && ( DRP[y][x + 1] == wave - 1 ) )

{

if( ( (x) == start\_x ) && ( y == start\_y ) ) break;

direction = { x\_prev, y\_prev, x+1, y };

x++;

}

else if( ( ( x - 1 ) >= 0 ) && ( DRP[y][x - 1] == wave - 1 ) )

{

if( ( (x) == start\_x ) && ( y == start\_y ) ) break;

direction = { x\_prev, y\_prev, x-1, y };

x--;

}

x\_prev = x;

y\_prev = y;

wave--;

shortest\_paths.push\_back( direction );

}

return LEN;

}

# PathFinding.h

#pragma once

#include <QVector>

#include <QLine>

#include <QPoint>

#include <tuple>

#define maxcoord 101

typedef std::tuple <QVector<QLine>, QVector<int>> PathAlgoResult;

PathAlgoResult WaveAlgo( int , QPoint, QVector<QPoint>, int , QVector<QLine> );

int find\_path( int paths\_matrix[][maxcoord], QPoint start\_pos, QPoint finish\_point, QVector<QLine>& );

void generate\_paths\_matrix( QVector <QLine> &roads, int paths[][maxcoord]);