Министерство образования и науки Российской Федерации  
Федеральное‌ ‌государственное‌ ‌бюджетное‌ ‌образовательное‌ ‌учреждение‌

высшего‌ ‌образования‌

**«Пермский национальный исследовательский  
политехнический университет»**

Кафедра «Информационные технологии и автоматизированные системы»

**О Т Ч Ё Т**

**по лабораторной работе №22**

Дисциплина: основы алгоритмизации и программирования

Тема: графы

Вариант 5

Выполнил работу

студент группы РИС-20-1б

Зверев А.Д.

Проверила

Доцент кафедры ИТАС

Полякова О.А.

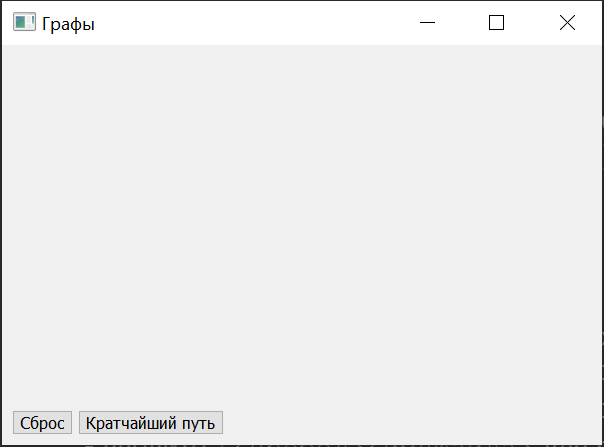
Пермь, 2021

**Постановка задачи**

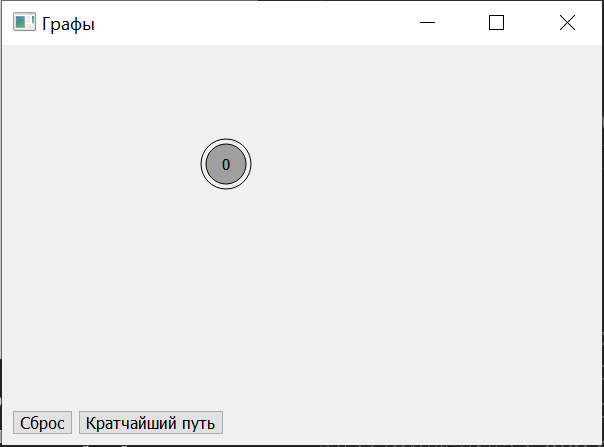
1. Реализовать алгоритм Дейкстры для графа, соответствующего выбранному варианту.
   1. Без использования QT.
   2. Интерфейс на усмотрение разработчика.
   3. Визуализация графа в OpenGL.
   4. Реализуется только метод Дейкстры.

**Анализ задачи**

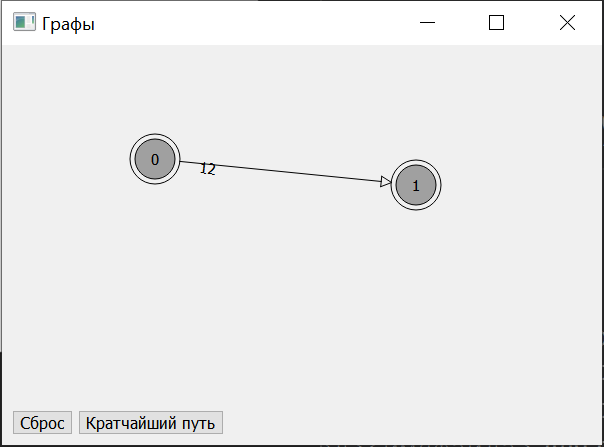
1. Для решения задачи необходимо…
   1. Проработать интерфейс для пользователя. Интерфейс должен быть понятным и удобным.



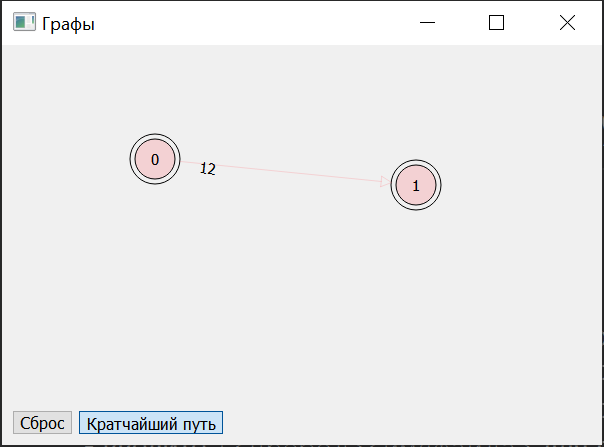
Как видно на данной картинке, интерфейс достаточно примитивен, чтобы быть понятным любому пользователю. Любое нажатие на пустую область экрана добавляет пункт.



Соединение двух пунктов последовательным нажатием на них мышью создает путь из одного города в другой.

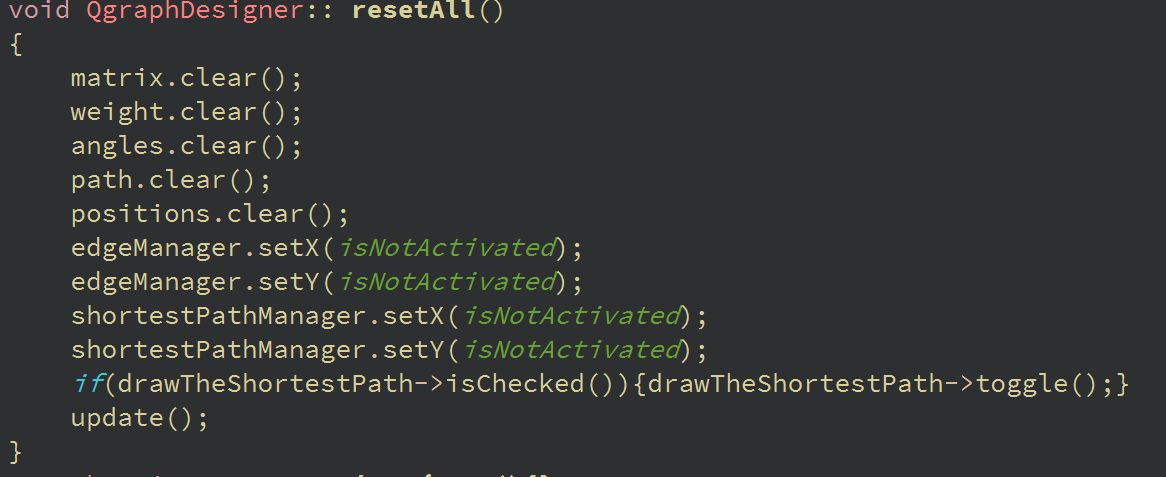


В этой миниигре пользователь сам волен выбирать кратчайший путь, программа лишь с помощью заложенных в ней функций подстраивается под его желания. Нажатие на кнопку «кратчайший путь», а затем последовательное выделение городов подсвечивает путь, выбранный пользователем.

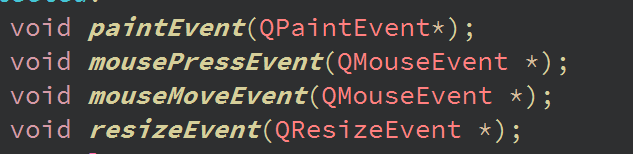


Кнопка сброс очищает окно программы, позволяя пользователю начать строить все с начала.

* 1. Написать функцию QgraphDesigner типа void, которая очищает окно вывода;



* 1. Написать несколько функций типа void, рассматривающие всякие события, происходящие в программе, в том числе и взаимодействие пользователя с интерфейсом при помощи мыши;



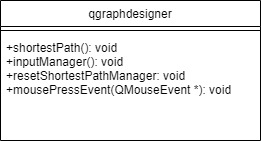
1. В программе были использованы следующие типы данных:
   1. Тип данных int для обозначения расстояний между городами;



* 1. Тип данных void для различных функций;



**Диаграмма классов**



**Код программы**

**Main.cpp**

#include "qgraphdesigner.h"

#include <QApplication>

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

{

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

QgraphDesigner w;

w.show();

*return* a.exec();

}

**qgraphdesigner.cpp**

#include "qgraphdesigner.h"

#include <QtGui>

#include <QtCore>

QgraphDesigner::**QgraphDesigner**(QWidget \*parent)

: QWidget(*parent*)

{

setWindowTitle ("Графы");

setAutoFillBackground(*true*);

setSizePolicy(QSizePolicy::*Expanding*, QSizePolicy::*Expanding*);

setFocusPolicy(Qt::*StrongFocus*);

*//initialise* *Managers*

edgeManager.setX(*isNotActivated*);

edgeManager.setY(*isNotActivated*);

shortestPathManager.setX(*isNotActivated*);

shortestPathManager.setY(*isNotActivated*);

*//create* *and* *configure* *buttons*

drawTheShortestPath = *new* QToolButton(*this*);

drawTheShortestPath->setText("Кратчайший путь");

drawTheShortestPath->setCheckable(*true*);

drawTheShortestPath->adjustSize();

connect(drawTheShortestPath,SIGNAL(toggled(bool)),*this*,SLOT(resetShortestPathManager()));

reset = *new* QToolButton(*this*);

reset->setText("Сброс");

reset->adjustSize();

connect(reset,SIGNAL(clicked(bool)),*this*,SLOT(resetAll()));

*//create* *and* *configure* *the* *reader* *object*

reader = *new* QLineEdit(*this*);

reader->hide();

reader->setFixedWidth(20);

connect(reader,SIGNAL(returnPressed()),*this*,SLOT(inputManager()));

}

QSize QgraphDesigner::***minimumSizeHint***() *const*

{

*return* QSize(300,200);

}

QSize QgraphDesigner::***sizeHint***() *const*

{

*return* QSize(600,400);

}

void QgraphDesigner::***resizeEvent***(QResizeEvent\*)

{

int y = height() - drawTheShortestPath->height() - 10 ;

reset->move(10,y);

drawTheShortestPath->move(10+reset->width()+5,y);

update();

}

void QgraphDesigner::***paintEvent***(QPaintEvent\*)

{

QPainter Painter(*this*);

Painter.setRenderHint(QPainter::*Antialiasing*);

QPen pen( Qt::*black* );

pen.setWidth(1);

pen.setStyle( Qt::*SolidLine* );

Painter.setPen( pen );

QFont font;

font.setPixelSize(15);

font.setWeight(4);

Painter.setFont(font);

QColor\* darkGreen = *new* QColor(255,84,92,50);

*//paint* *the* *outer* *cercle* *of* *all* *vertecies*

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

{

*if*(positions[i].x()==*Erased*)

{

*continue*;

}

Painter.drawEllipse(positions[i],*Radius*,*Radius*);

}

Painter.setBrush(palette().dark());

*//paint* *the* *inner* *disc* *and* *id* *number* *of* *all* *verticies*

bool vertexFromThePath=*false* ;

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

{

*if*(positions[i].x()==*Erased*){*continue*;}

*if*(drawTheShortestPath->isChecked())

{

*for*(int j=0 ; j<path.size() ; j++)

{

*if*(path[j]==i){vertexFromThePath=*true*;*break*;}

}

*if*(vertexFromThePath)

{

Painter.setBrush(\*darkGreen);

}

}

Painter.drawEllipse(positions[i],*Radius* - *Radius*/5,*Radius* - *Radius*/5);

Painter.drawText(positions[i].x()-*Radius*, positions[i].y()-*Radius*, 2\**Radius*, 2\**Radius*, Qt::*AlignCenter*, QString::number(i));

*if*(vertexFromThePath)

{

Painter.setBrush(palette().dark());

vertexFromThePath=*false* ;

}

}

*//paint* *activated* *vertex* *if* *there* *is* *one*

*if*(edgeManager.x()==*isActivated*)

{

Painter.setBrush(palette().window());

Painter.drawEllipse(positions[edgeManager.y()],*Radius*,*Radius*);

Painter.setBrush(Qt::*red*);

Painter.drawEllipse(positions[edgeManager.y()],*Radius* - *Radius*/5,*Radius* - *Radius*/5);

Painter.drawText(positions[edgeManager.y()].x()-*Radius*, positions[edgeManager.y()].y()-*Radius*, 2\**Radius*,2\**Radius*, Qt::*AlignCenter*, QString::number(edgeManager.y()));

}

*//paint* *source* *vertex* *for* *a* *shortest* *path* *search* *if* *there* *is* *one*

*if*((shortestPathManager.x()!=*isNotActivated*)&&(shortestPathManager.y()==*isNotActivated*))

{

Painter.setBrush(\*darkGreen);

Painter.drawEllipse(positions[shortestPathManager.x()],*Radius* - *Radius*/5,*Radius* - *Radius*/5);

Painter.drawText(positions[shortestPathManager.x()].x()-*Radius*, positions[shortestPathManager.x()].y()-*Radius*, 2\**Radius*,2\**Radius*, Qt::*AlignCenter*, QString::number(shortestPathManager.x()));

}

*//paint* *edges*

QLineF one,tow,three,neoLine,arrowBase;

QPointF prime;

bool bothVerticiesAreIn=*false*,oneVertexIsIn=*false*;

int vertex1positionInPath,vertex2PositionInPath;

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

{

*if*(positions[i].x()==*Erased*){*continue*;}

*for*(int j=0 ; j<path.size() ; j++)

{

*if*(path[j]==i){oneVertexIsIn=*true*;vertex1positionInPath=j;*break*;}

}

*for*(int j=0 ; j< matrix[i].size() ; j++)

{

*if*(positions[matrix[i][j]].x()==*Erased*){*continue*;}

*for*(int k=0 ; k<path.size() ; k++)

{

*if*((path[k]==matrix[i][j])&&(oneVertexIsIn)){bothVerticiesAreIn=*true*;vertex2PositionInPath=k;*break*;}

}

*if*((bothVerticiesAreIn)&&((vertex1positionInPath==vertex2PositionInPath+1)||(vertex2PositionInPath==vertex1positionInPath+1)))

{

Painter.setPen(\*darkGreen);

}

one.setP1(positions[i]);

one.setP2(positions[matrix[i][j]]);

one.setLength(one.length() - *Radius*);

tow.setP2(positions[i]);

tow.setP1(positions[matrix[i][j]]);

tow.setLength(tow.length() - *Radius*);

prime.setX(one.p2().x());

prime.setY(one.p2().y());

neoLine.setP1(tow.p2());

neoLine.setP2(prime);

neoLine.setLength(neoLine.length()-10);

arrowBase = neoLine.normalVector();

arrowBase.translate(neoLine.dx(),neoLine.dy());

arrowBase.setLength(5);

three.setP1(arrowBase.p2());

three.setP2(neoLine.p2());

three.setLength(10);

Painter.drawLine(neoLine);

Painter.drawLine(three.p1(), three.p2());

Painter.drawLine(three.p1(), prime);

Painter.drawLine(three.p2(), prime);

*if*(bothVerticiesAreIn)

{

Painter.setPen(Qt::*black*);

bothVerticiesAreIn=*false* ;

}

}

oneVertexIsIn=*false* ;

}

*//draw* *weights*

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

{

*if*(positions[i].x()==*Erased*){*continue*;}

*for*(int j=0 ; j< weight[i].size() ; j++)

{

*if*(positions[matrix[i][j]].x()==*Erased*)

{

*continue*;

}

Painter.save();

Painter.translate(positions[i]);

Painter.rotate(-angles[i][j]);

Painter.drawText(*Radius* + 20, 10, QString::number(weight[i][j]));

Painter.restore();

}

}

}

void QgraphDesigner::***mousePressEvent***(QMouseEvent\* event)

{

*if*(reader->isVisible())

{

reader->setFocus();

*return*;

}

QRect rect(*Margin*+*Radius*, *Margin*+*Radius*, width() - 2 \* *Margin* - 2 \* *Radius*, height() - 2 \* *Margin* - 2 \* *Radius*);

*if* (event->button() == Qt::*LeftButton*) *//add* *a* *vertex* *||* *select* *a* *vertex*

{

*if*((shortestPathManager.x()!=*isNotActivated*)&&(shortestPathManager.y()!=*isNotActivated*)){drawTheShortestPath->setFocus(); *return* ;}

*if* (rect.contains(event->pos())) *//valid* *click*

{

*if*(positions.isEmpty())

{

positions.append(event->pos()); *//first* *vertex*

}

*else*

{

bool AVertexIsClicked = *false* ;

int theClickedVertex=0;

QRect vertexSafeArea ;

*while*(!AVertexIsClicked) *//check* *if* *the* *user* *clicked* *a* *vertex*

{

vertexSafeArea.setX( positions[theClickedVertex].x()- 2 \* *Radius* );

vertexSafeArea.setY( positions[theClickedVertex].y()- 2 \* *Radius* );

vertexSafeArea.setWidth(4 \* *Radius*);

vertexSafeArea.setHeight(4 \* *Radius*);

*if*(vertexSafeArea.contains(event->pos())){AVertexIsClicked=*true*;*break*;}

theClickedVertex++;

*if*(theClickedVertex==positions.size()){*break*;}

}

*if*(AVertexIsClicked==*false*) *//No* *vertex* *clicked* *->* *add* *a* *new* *vertex* *by* *saving* *the* *click's* *postion* *and* *adding* *a* *new* *empty* *edge* *vector* *in* *the* *main* *edge* *matrix*

{

*if*(drawTheShortestPath->isChecked()){*return*;}

*if*(edgeManager.x()==*isActivated*){*return*;}

positions.append(event->pos());

QVector<int> vect ;

QVector<qreal> vect2 ;

matrix.insert(positions.size(),vect);

weight.insert(positions.size(),vect);

angles.insert(positions.size(),vect2);

}

*else* *//* *Some* *vertex* *was* *clicked* *->* *activate* *it* *if* *no* *one* *else* *is* *activated* *or* *add* *an* *new* *adge* *end* *desactive* *it* *if* *an* *other* *vertex* *is* *all* *ready* *activated*

{

vertexSafeArea.setX( positions[theClickedVertex].x()- *Radius* ); *//make* *the* *area* *smaller* *for* *optimisation* *purposes*

vertexSafeArea.setY( positions[theClickedVertex].y()- *Radius* );

vertexSafeArea.setWidth(2 \* *Radius*);

vertexSafeArea.setHeight(2 \* *Radius*);

*if*(vertexSafeArea.contains(event->pos()))

{

*if*(drawTheShortestPath->isChecked()) *//if* *we* *are* *looking* *for* *the* *shortest* *path* *handle* *the* *shortest* *path* *manager*

{

*if*(shortestPathManager.x()==*isNotActivated*)

{

shortestPathManager.setX(theClickedVertex);

}

*else* *if*(shortestPathManager.x()==theClickedVertex)

{

shortestPathManager.setX(*isNotActivated*);

}

*else*

{

shortestPathManager.setY(theClickedVertex);

shortestPath();

}

}

*else* *//else* *activate* *the* *vertex* *or* *complete* *the* *addition* *of* *an* *edge*

{

*if*(edgeManager.x()==*isNotActivated*) *//activate* *the* *edge*

{

edgeManager.setX(*isActivated*);

edgeManager.setY(theClickedVertex);

}

*else* *if*(edgeManager.x()==*isActivated*)

{

*if*(edgeManager.y()==theClickedVertex) *//if* *user* *click* *the* *same* *vetex* *twice* *then* *deactivate* *it*

{

edgeManager.setX(*isNotActivated*);

}

*else* *//add* *the* *edge,* *its* *angle* *and* *actiate* *the* *reader* *only* *if* *it* *doesn't* *already* *existe.*

{

int k = 0 ; bool theEdgeAllreadyExists =*false* ;

*while*(k<matrix[edgeManager.y()].size())

{

*if*(matrix[edgeManager.y()][k]==theClickedVertex){theEdgeAllreadyExists=*true* ; *break* ;}

k++;

}

*if*(!theEdgeAllreadyExists)

{

QLineF li(positions[edgeManager.y()],positions[theClickedVertex]);

matrix[edgeManager.y()].append(theClickedVertex);

angles[edgeManager.y()].append(li.angle());

reader->move( (positions[edgeManager.y()].x()+positions[theClickedVertex].x())/2,

(positions[edgeManager.y()].y()+positions[theClickedVertex].y())/2 );

reader->setEnabled(*true*);

reader->*setVisible*(*true*);

reader->setFocus();

edgeManager.setX(*isNotActivated*);

}

*else*

{

edgeManager.setX(*isNotActivated*);

}

}

}

}

}

}

}

}

}

*else* *if*(event->button() == Qt::*RightButton*) *//erase* *a* *vertex*

{

bool AVertexIsClicked = *false* ;

int theClickedVertex=0;

QRect vertexSafeArea ;

*while*(!AVertexIsClicked) *//check* *if* *some* *vertex* *is* *clicked*

{

vertexSafeArea.setX( positions[theClickedVertex].x()- *Radius* );

vertexSafeArea.setY( positions[theClickedVertex].y()- *Radius* );

vertexSafeArea.setWidth(2 \* *Radius*);

vertexSafeArea.setHeight(2 \* *Radius*);

*if*(vertexSafeArea.contains(event->pos())){AVertexIsClicked=*true*;*break*;}

theClickedVertex++;

*if*(theClickedVertex==positions.size()){*break*;}

}

*if*(AVertexIsClicked==*true*)

{

positions[theClickedVertex].setX(*Erased*);

}

*if*(drawTheShortestPath->isChecked())

{

shortestPath();

}

}

update();

}

void QgraphDesigner::***mouseMoveEvent***(QMouseEvent \* event)

{

*if*(reader->isVisible())

{

reader->setFocus();

*return*;

}

QRect rect(*Radius* + *Margin*/2, *Radius* + *Margin*/2, width() - 2 \* *Radius* - *Margin*, height() - 2 \* *Radius* - *Margin*);

*if* (rect.contains(event->pos())) *//valid* *click*

{

bool AVertexIsClicked = *false* ;

int theClickedVertex=0;

QRect vertexSafeArea ;

*while*(!AVertexIsClicked) *//check* *if* *the* *user* *clicked* *a* *vertex*

{

vertexSafeArea.setX( positions[theClickedVertex].x()- *Radius* );

vertexSafeArea.setY( positions[theClickedVertex].y()- *Radius* );

vertexSafeArea.setWidth(2 \* *Radius*);

vertexSafeArea.setHeight(2 \* *Radius*);

*if*(vertexSafeArea.contains(event->pos())){AVertexIsClicked=*true*;*break*;}

theClickedVertex++;

*if*(theClickedVertex==positions.size()){*break*;}

}

*if*(AVertexIsClicked)

{

positions[theClickedVertex]=event->pos();

*for*(int i =0 ; i<angles[theClickedVertex].size() ; i++)

{

QLineF li(positions[theClickedVertex],positions[matrix[theClickedVertex][i]]);

angles[theClickedVertex][i]=li.angle();

}

edgeManager.setX(*isNotActivated*);

update();

}

}

}

void QgraphDesigner::**inputManager**()

{

weight[edgeManager.y()].append(reader->text().toInt());

reader->clear();

reader->setDisabled(*true*);

reader->hide();

update();

}

void QgraphDesigner::**shortestPath**()

{

int AdjMat[25][25];

int p[25][25];

path.clear();

*//initialise* *AdjMat*

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

*for* (int j = 0; j < positions.size(); j++)

AdjMat[i][j] = *Infinity*;

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

{

*if*(positions[i].x()==*Erased*)

{

*continue*;

}

*for*(int j=0 ; j< matrix[i].size() ; j++)

{

*if*(positions[matrix[i][j]].x()==*Erased*){*continue*;}

AdjMat[i][matrix[i][j]]=weight[i][j];

}

}

*//initialise* *Parent* *matrix*

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

*for* (int j = 0; j < positions.size(); j++)

p[i][j] = i;

*//Floyd\_Warshall* *Algorithm* *for* *all* *pairs* *shortest* *paths*

*for* (int k = 0; k < positions.size(); k++)

{

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

{

*for* (int j = 0; j < positions.size(); j++)

{

*if* (AdjMat[i][k] + AdjMat[k][j] < AdjMat[i][j])

{

AdjMat[i][j] = AdjMat[i][k] + AdjMat[k][j];

p[i][j] = p[k][j]; *//* *update* *the* *parent* *matrix*

}

}

}

}

*if*(AdjMat[shortestPathManager.x()][shortestPathManager.y()]!=*Infinity*)

{

int i=shortestPathManager.y();

*while*(i!=shortestPathManager.x())

{

path.append(i);

i=p[shortestPathManager.x()][i];

}

path.append(shortestPathManager.x());

}

*else*

{

path.append(*Infinity*);

}

update();

}

void QgraphDesigner:: **resetShortestPathManager**()

{

shortestPathManager.setX(*isNotActivated*);

shortestPathManager.setY(*isNotActivated*);

path.clear();

update();

}

void QgraphDesigner:: **resetAll**()

{

matrix.clear();

weight.clear();

angles.clear();

path.clear();

positions.clear();

edgeManager.setX(*isNotActivated*);

edgeManager.setY(*isNotActivated*);

shortestPathManager.setX(*isNotActivated*);

shortestPathManager.setY(*isNotActivated*);

*if*(drawTheShortestPath->isChecked()){drawTheShortestPath->toggle();}

update();

}

QgraphDesigner::~***QgraphDesigner***(){}

**qgrapghdesigner.h**

#ifndef QGRAPHDESIGNER\_H

#define QGRAPHDESIGNER\_H

#include <QWidget>

#include <QToolButton>

#include <QMap>

#include <QVector>

#include <QLineEdit>

*//class* *qreal;*

*class* **QgraphDesigner** : *public* QWidget

{

Q\_OBJECT

*public*:

**QgraphDesigner**(QWidget \*parent = 0);

QSize ***minimumSizeHint***() *const*;

QSize ***sizeHint***() *const*;

~***QgraphDesigner***();

*protected*:

void ***paintEvent***(QPaintEvent\*);

void ***mousePressEvent***(QMouseEvent \*);

void ***mouseMoveEvent***(QMouseEvent \*);

void ***resizeEvent***(QResizeEvent \*);

*private* slots:

void **inputManager**() ;

void **resetShortestPathManager**() ;

void **resetAll**() ;

*public* slots:

void **shortestPath**() ;

*private*:

*enum* { *Margin* = 5 , *Radius* = 25 , *isActivated* = -1 , *isNotActivated* = -2 , *Erased* = -99 , *Infinity* = 100000000 };

QPoint edgeManager ;

QPoint shortestPathManager ;

QToolButton\* drawTheShortestPath ;

QToolButton\* reset ;

QLineEdit\* reader ;

QMap< int,QVector<int> > matrix ;

QMap< int,QVector<int> > weight ;

QMap< int,QVector<qreal> > angles ;

QVector<QPointF> positions ;

QVector<int> path ;

};

#endif *//* *QGRAPHDESIGNER\_H*

**Скриншоты**

