Министерство образования Республики Беларусь

Учреждение образования

«Брестский государственный технический университет»

Кафедра ИИТ

Лабораторная работа №1, 2

за 1 семестр

По дисциплине: «ОСИСП»

Выполнил:

Студент 3 курса

Группы ПО-5

Лозейко П. А.

Проверила:

Дрябко А. В.

2021

**Игра “Лабиринт”**

**game.cpp**

#include "maze.h"

#include "ui\_maze.h"

void Maze::**InitGame**()

{

field = generate(30, 30);

game\_win->setupUi(this);

game\_win->centralwidget->*setVisible*(false);

this->setWindowTitle("Maze");

man.clear();

wall.clear();

gener();

srand(time(0));

isPause=false;

}

void Maze::**gener**() {

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

for (int j = 0; j < field.at(i).size(); j++) {

int cur = field.at(i).at(j);

if (cur == 0) {qMakePair(i, j); wall.push\_back(QPoint(i, j));}

if (cur == 4) {qMakePair(i, j); place\_finish\_x = i, place\_finish\_y = j; qDebug() << place\_finish\_x << place\_finish\_y;}

else if (cur == 3) {qMakePair(i, j); man.push\_back(QPoint(i, j));}

}

}

}

void Maze::***paintEvent***(QPaintEvent \*event)

{

if(flag\_game) {

Q\_UNUSED(event);

QPainter painter(this);

painter.setBrush(Qt::black);

painter.drawRect(MARGIN,MARGIN,AREA\_COL\*BLOCK\_SIZE,AREA\_ROW\*BLOCK\_SIZE);

painter.setBrush(Qt::blue);

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

painter.drawRect(MARGIN+wall[i].x()\*BLOCK\_SIZE,MARGIN+wall[i].y()\*BLOCK\_SIZE,BLOCK\_SIZE,BLOCK\_SIZE);

painter.setBrush(Qt::red);

painter.drawRect(MARGIN+man[0].x()\*BLOCK\_SIZE,MARGIN+man[0].y()\*BLOCK\_SIZE,BLOCK\_SIZE,BLOCK\_SIZE);

}

}

void Maze::***keyPressEvent***(QKeyEvent \*event)

{

if(flag\_game) {

switch(event->key())

{

case Qt::Key\_Up:

SnakeUpdate(Qt::Key\_Up);

break;

case Qt::Key\_Down:

SnakeUpdate(Qt::Key\_Down);

break;

case Qt::Key\_Left:

SnakeUpdate(Qt::Key\_Left);

break;

case Qt::Key\_Right:

SnakeUpdate(Qt::Key\_Right);

break;

case Qt::Key\_Escape:

PauseResumeGame();

break;

default:

break;

}

}

}

bool Maze::**IsGameOver**()

{

int x=man.front().x();

int y=man.front().y();

if(x<0||x>AREA\_COL-1||y<0||y>AREA\_ROW-1)

return true;

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

if(wall[i]==man.front())

return true;

return false;

}

void Maze::**SnakeUpdate**(int key)

{

if(key == Qt::Key\_Up) man.push\_front(QPoint(man.front().x(),man.front().y()-1));

if(key == Qt::Key\_Down) man.push\_front(QPoint(man.front().x(),man.front().y()+1));

if(key == Qt::Key\_Left) man.push\_front(QPoint(man.front().x()-1,man.front().y()));

if(key == Qt::Key\_Right) man.push\_front(QPoint(man.front().x()+1,man.front().y()));

if(IsGameOver())

{

man.clear();

man.push\_back(QPoint(place\_x, place\_y));

}

else {

man.pop\_back();

place\_x = man.front().x();

place\_y = man.front().y();

qDebug() << place\_x << place\_y;

update();

}

if(place\_x == place\_finish\_x && place\_y == place\_finish\_y) {

QMessageBox::information(this, "Информация", "Вы прошли уровень");

flag\_game = false;

InitMenu();

}

}

**generator.cpp**

#include "generator.h"

void **markCell**(QVector<QVector<int> >& lens, QStack<QPair<int, int> >& stack, QVector<QVector<int> > field);

QVector<QVector<int> > **generate**(int width, int height) {

srand(time(0));

QVector<QVector<int> > field;

//убедимся, что длина и ширина нечетны для корректной генерации

width = (width % 2 == 1) ? width : width + 1;

height = (height % 2 == 1) ? height : height + 1;

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

field.append(QVector<int>());

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

if (i % 2 != 0 && j % 2 != 0 && (i < height - 1 && j < width - 1)) {

field[i].append(1); //1 - клетка непосещенная

}

else field[i].append(0); //0 -стена, есть еще 2 - дорожка

}

}

//алгоритм генерации лабиринта

for (int i = 1; i < field.at(0).size() - 1; i++) {

field[1][i] = 2;

}

for (int i = 3; i < field.size() - 1; i+=2) {

int j = 1;

QVector<int> set;

while (j < field.at(i).size() - 1) {

set.append(j);

int r = rand() % 2; //0 - не идти вправо, 1 - идти

if (r == 1 && j < field.at(i).size() - 2) {

field[i][j] = 2;

field[i][j+1] = 2;

field[i][j+2] = 2;

j += 2;

}

else {

int randCel = set.at(rand() % set.size());

field[i][randCel] = 2;

field[i-1][randCel] = 2;

if (j < field.at(i).size() - 2) j += 2;

else break;

set.clear();

}

}

}

//выбор клетки входа

bool selected = false;

int startI, startJ;

while (!selected) {

int randij = rand() % 2; //0 - бегаем по строкам, 1 - бегаем по столбцам

if (randij == 0) {

int randj = rand() % 2; //0 - 0, 1 - field[0].size() - 1

randj = (randj == 0) ? 0 : field.at(0).size() - 1;

int randi = 1 + rand() % field.size() - 1;

int nearj = (randj == 0) ? 1 : field.at(0).size() - 2;

if (field.at(randi).at(nearj) == 2) {

field[randi][randj] = 3; //3 - вход

startI = randi, startJ = nearj;

selected = true;

}

} else {

int randi = rand() % 2; //0 - 0, 1 - field.size() - 1

randi = (randi == 0) ? 0 : field.size() - 1;

int randj = 1 + rand() % field.at(0).size() - 1;

int neari = (randi == 0) ? 1 : field.size() - 2;

if (field.at(neari).at(randj) == 2) {

field[randi][randj] = 3; //3 - вход

startI = neari, startJ = randj;

selected = true;

}

}

}

//создание матрицы расстояний (для поиска наиболее отдаленного выхода)

QVector<QVector<int> > lens = QVector<QVector<int> >(field);

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

for (int j = 0; j < field.at(i).size(); j++) {

lens[i][j] = -1; //-1 длина еще не задана

}

}

lens[startI][startJ] = 0;

QStack<QPair<int, int> > stack;

stack.push(qMakePair(startI, startJ));

while (stack.size() > 0) {

markCell(*lens*, *stack*, field);

}

QPair<int, int> exitCell;

int maxDist = 0;

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

if (i == 0) {

for (int j = 1; j < field.at(i).size() - 1; j++) {

if (lens.at(i+1).at(j) > maxDist) {

maxDist = lens.at(i+1).at(j);

exitCell = qMakePair(i, j);

}

}

} else if (i == field.size() - 1) {

for (int j = 1; j < field.at(i).size() - 1; j++) {

if (lens.at(i-1).at(j) > maxDist) {

maxDist = lens.at(i-1).at(j);

exitCell = qMakePair(i, j);

}

}

} else {

if (lens.at(i).at(1) > maxDist) {

maxDist = lens.at(i).at(1);

exitCell = qMakePair(i, 0);

}

if (lens.at(i).at(lens.at(i).size() - 2) > maxDist) {

maxDist = lens.at(i).at(lens.at(i).size() - 2);

exitCell = qMakePair(i, lens.at(i).size() - 1);

}

}

}

field[exitCell.first][exitCell.second] = 4; //4 - выход

return field;

}

void **markCell**(QVector<QVector<int> > &lens, QStack<QPair<int, int> >& stack, QVector<QVector<int> > field) {

QPair<int, int> pair = stack.pop();

int i = pair.first, j = pair.second;

if (i < lens.size() - 1) {

if (field.at(i + 1).at(j) == 2 && lens.at(i + 1).at(j) == -1) {

lens[i+1][j] = lens.at(i).at(j) + 1;

stack.push(qMakePair(i+1, j));

}

}

if (i > 0) {

if (field.at(i - 1).at(j) == 2 && lens.at(i - 1).at(j) == -1) {

lens[i-1][j] = lens.at(i).at(j) + 1;

stack.push(qMakePair(i-1, j));

}

}

if (j < lens.at(i).size() - 1) {

if (field.at(i).at(j + 1) == 2 && lens.at(i).at(j + 1) == -1) {

lens[i][j+1] = lens.at(i).at(j) + 1;

stack.push(qMakePair(i, j+1));

}

}

if (j > 0) {

if (field.at(i).at(j - 1) == 2 && lens.at(i).at(j -1) == -1) {

lens[i][j-1] = lens.at(i).at(j) + 1;

stack.push(qMakePair(i, j-1));

}

}

}

**main.cpp**

#include "maze.h"

#include <QApplication>

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

{

QApplication a(*argc*, argv);

Maze w;

w.show();

return a.exec();

}

**maze.cpp**

Maze::**Maze**(QWidget \*parent): QMainWindow(parent), ui(new Ui::Maze)

{

ui->setupUi(this);

setFixedSize(QSize(MARGIN\*2+(AREA\_COL+1)\*BLOCK\_SIZE,MARGIN\*2+AREA\_ROW\*BLOCK\_SIZE));

this->setWindowTitle("Maze");

ui->btn\_start->setText("Играть");

QMenu\* menu = new QMenu("О программе");

ui->menubar->addMenu(menu);

menu->addAction("about.dll", this,SLOT(aboutQt()), Qt::CTRL + Qt::Key\_Q);

}

void Maze::**InitMenu**() {

ui->setupUi(this);

setFixedSize(QSize(MARGIN\*2+(AREA\_COL+1)\*BLOCK\_SIZE,MARGIN\*2+AREA\_ROW\*BLOCK\_SIZE));

connect(ui->btn\_start, SIGNAL(clicked()), this, SLOT(on\_btn\_start\_clicked()));

connect(ui->btn\_exit, SIGNAL(clicked()), this, SLOT(on\_btn\_exit\_clicked()));

this->setWindowTitle("Maze");

ui->btn\_start->setText("Играть");

QMenu\* menu = new QMenu("О программе");

ui->menubar->addMenu(menu);

menu->addAction("about.dll", this,SLOT(aboutQt()), Qt::CTRL + Qt::Key\_Q);

}

void Maze::**PauseResumeGame**()

{

if(!isPause)

{

isPause=!isPause;

InitPause();

}

else isPause=!isPause;

}

void Maze::**InitPause**()

{

QMessageBox::StandardButton pause;

pause = QMessageBox::question(this, "Pause", "Продолжить?",

QMessageBox::Yes|QMessageBox::No);

if (pause == QMessageBox::Yes) {

PauseResumeGame();

}

else {

flag\_game = false;

InitMenu();

}

}

void Maze::**on\_btn\_start\_clicked**()

{

ui->centralwidget->*setVisible*(false);

flag\_game = true;

InitGame();

}

**menu.cpp**

#include "maze.h"

#include "ui\_maze.h"

void Maze::**on\_btn\_exit\_clicked**()

{

exit(0);

}

void Maze::**aboutQt**()

{

QMessageBox msgBox;

msgBox.setWindowTitle("About programm");

msgBox.setText("Эту программу сделал Лозейко Павел");

msgBox.*exec*();

}

Maze::~***Maze***(){

delete ui;

}

**generator.h**

#ifndef GENERATOR\_H

#define GENERATOR\_H

#include <QVector>

#include <QPair>

#include <QStack>

#include <ctime>

QVector<QVector<int> > **generate**(int width, int height);

void **markCell**(QVector<QVector<int> > &lens, QStack<QPair<int, int> >& stack, QVector<QVector<int> > field);

#endif // GENERATOR\_H

**maze.h**

#ifndef MAZE\_H

#define MAZE\_H

#include <QMainWindow>

#include <QPaintEvent>

#include <QKeyEvent>

#include <QWidget>

#include <QComboBox>

#include <QPainter>

#include <time.h>

#include <QString>

#include <QtWidgets>

#include <QMessageBox>

#include "generator.h"

const int BLOCK\_SIZE=15;

const int MARGIN=22;

const int AREA\_ROW=31;

const int AREA\_COL=31;

enum **moved**

{

UP,

DOWN,

LEFT,

RIGHT,

};

QT\_BEGIN\_NAMESPACE

namespace **Ui** { class **Maze**; }

QT\_END\_NAMESPACE

class **Maze** : public QMainWindow

{

Q\_OBJECT

public:

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

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

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

~***Maze***();

int place\_x;

int place\_y;

int place\_finish\_x, place\_finish\_y;

QVector<QVector<int> > field;

public:

void **InitGame**();

void **InitPause**();

void **PauseResumeGame**();

bool **IsGameOver**();

void **InitMenu**();

void **gener**();

public slots:

void **on\_btn\_exit\_clicked**();

private slots:

void **SnakeUpdate**(int key);

void **aboutQt**();

void **on\_btn\_start\_clicked**();

private:

Ui::Maze \*ui;

Ui::Maze \*game\_win; // окно для игры

bool flag\_game = false;

bool isPause;

QList<QPoint> wall;

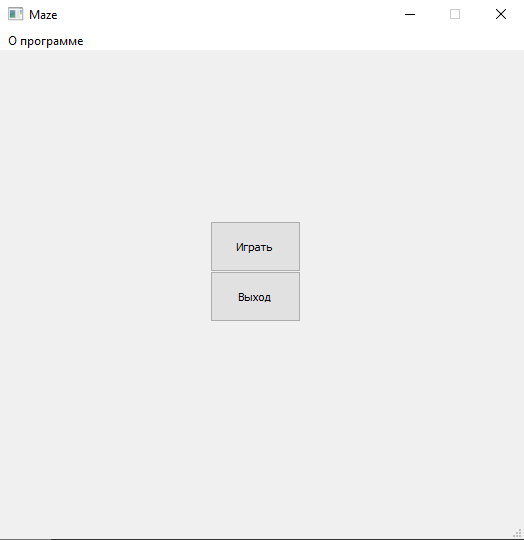
QList<QPoint> man;

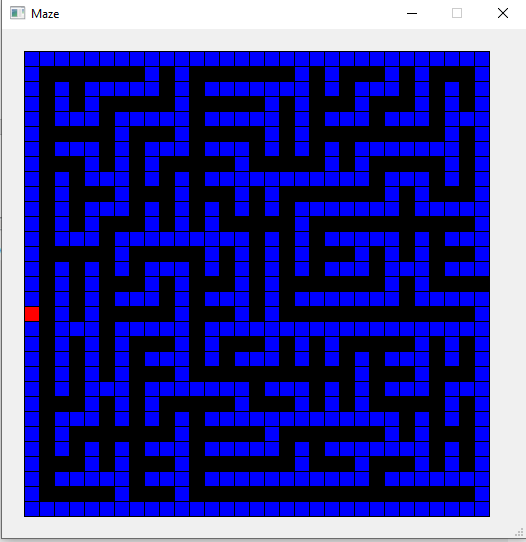
moved dir;

};

#endif // MAZE\_H

**Результат:**

****

****