|  |  |  |
| --- | --- | --- |
| Министерство науки и высшего образования  Российской Федерации | | |
| Федеральное государственное бюджетное  образовательное учреждение высшего образования | | |
| «Новосибирский государственный технический университет» | | |
|  | | |
|  | | |
|  | | |
| Расчетно – графическое задание | | |
| по дисциплине «Методы построения и анализа алгоритмов» | | |
|  | | |
| **кафедра теоретической и прикладной информатики** | | |
|  | | |
|  | Факультет: | ПМИ |
| Группа: | ПМИ-03 |
| Бригада: | Место для ввода текста. |
| Студент: | Сидоров Даниил |
|  |  |
|  |  |
| Преподаватель: | Щукин Георгий Анатольевич |
|  |  |
|  | | |
| Новосибирск | | |
| 2022 | | |

**Задача**

Мобильный робот движется по плоскости. Робот имеет форму круга радиусом r. Робот может двигаться вверх, вниз, вправо, влево. Также на плоскости имеются препятствия (Круги). Заданы начальное положение робота и конечный пункт, куда робот должен попасть, обходя препятствия.

Задача: найти кратчайший путь из начального положения в конечное с учётом препятствий на поле.

Вариант 8:

Алгоритм - Rapidly Exploring Random Trees (RRT);

Тип препятствий – круги;

Без промежуточных координат.

**Алгоритм**

Реализован алгоритм RRT и его улучшение - RRT\*, в котором постоянно уточняются сформированные пути для достижения оптимальной конфигурации пути.

В данном методе из начальной конфигурации qInit строится дерево, покрывающее Cfree. После построения дерева в него добавляется qGoal и производится поиск кратчайшего пути из начальной конфигурации в конечную. Данный метод ориентирован на поиск ответа на один запрос с конкретными qInit и qGoal.

**Подпрограммы**

getInput() – ввод данных через консоль.

fileInput() – загрузка сцены из файла.

save() – сохранение текущей сцены.

prepareInput() – подготовка данных для GUI.

draw(sf::RenderWindow &window) – отрисовка действий RRT.

randomCoordinate(T low, T high) – возвращает случайную координату.

isEdgeObstacleFree(Point a, Point b) – проверка на встречу с препятствием. pickRandomPoint() – случайная точка в сторону конечной точки.

checkDestinationReached() – проверка на нахождение пути.

rewire() – перевязка родителей.

RRT() – сам алгоритм.

checkCollision(Point lineFrom, Point lineTo, Point location, ftype radius) – проверка на коллизию с конечной точкой.

lineSegmentIntersectsCircle(Point lineFrom, Point lineTo, circ O) – проверка на коллизию линии с препятствием.

stepNear(Point &p1, Point &p2, ftype DELTA) – возвращает точку в сторону вектора.

steer(const Point &t, ftype DELTA) – возвращает точку на отрезке XY.

**Спецификация программы**

Программа реализована на С++ в Visual studio с использованием графической библиотеки SFML.

Все данные задаются через консоль. Доступен ввод через файл. Начало координат в верхнем левом углу. Чем больше x, тем правее фигура. Чем больше y, тем ниже фигура.

Все выходные данные о длине пути или его отсутствии так же выводятся в консоли.

Каждые 500 итераций появляется актуальная информация о пути.

После завершения программы, сцена сохраняется в файле.

Пример ввода данных:

1 // Количество сцен

2 //выбор RRT - 1 или RRT\* - 2

5 // радиус робота

50 50 // Координаты начальной точки

550 550 // Координаты конечной точки

2 //Количество препятствий

200 200 //Координаты 1 препятствия

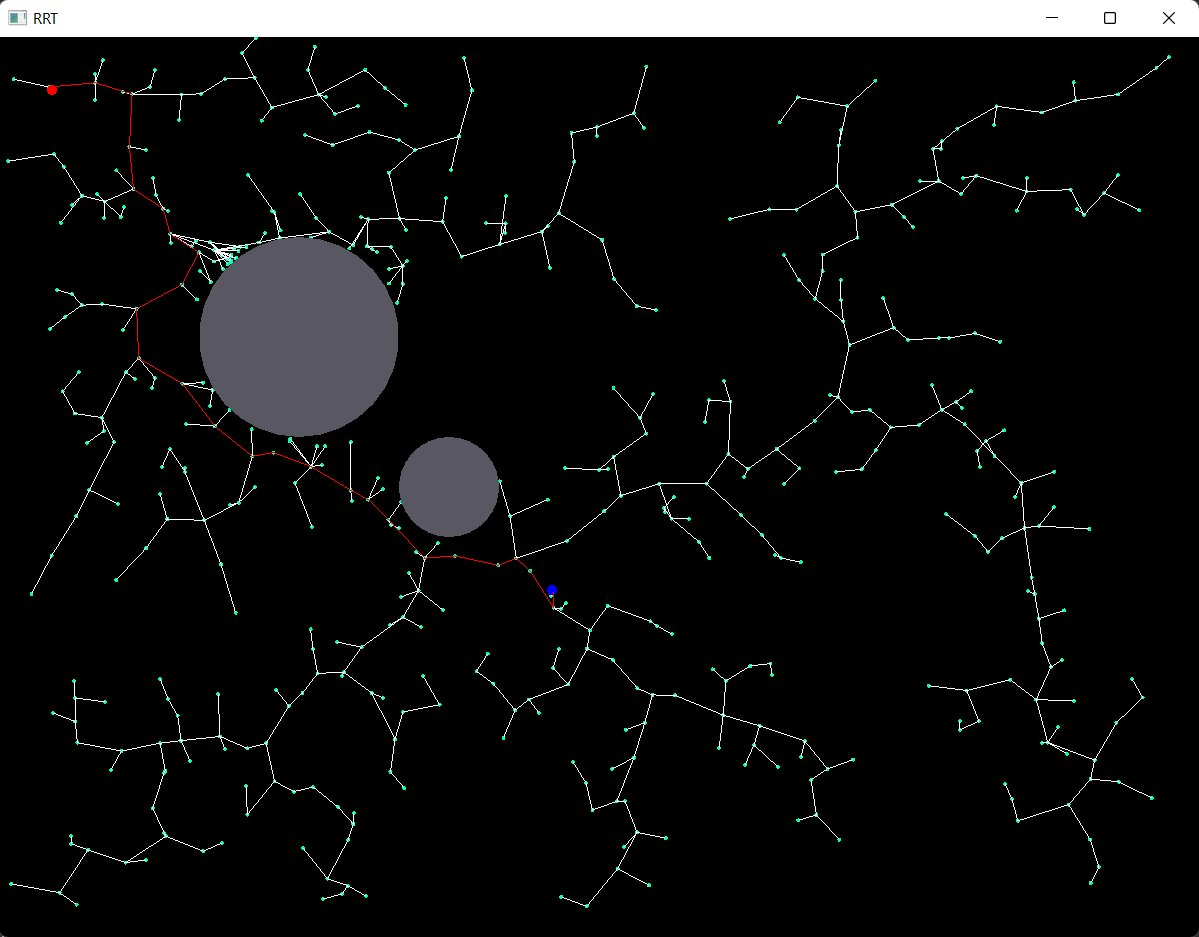
100 //Радиус 1 препятствия

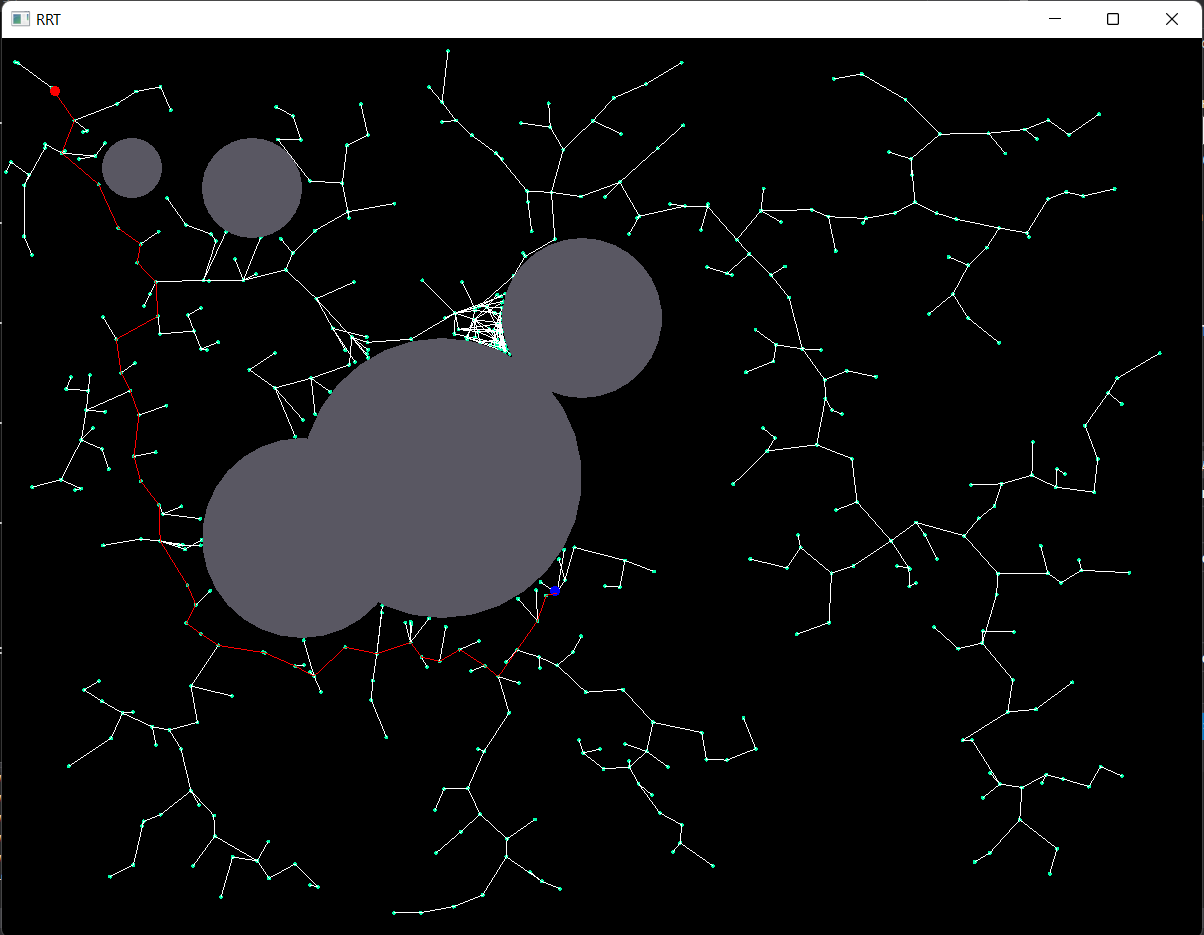
400 400 //Координаты 2 препятствия

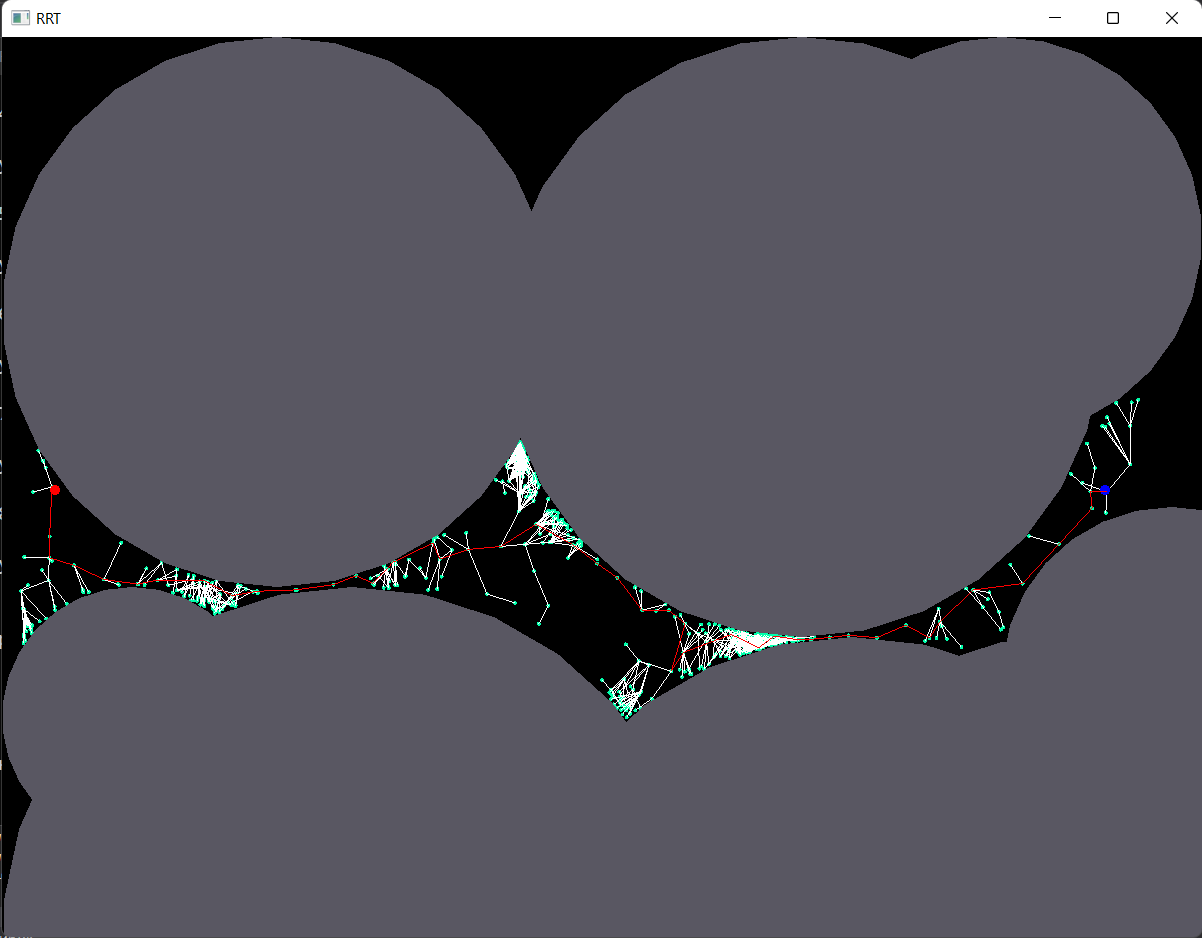
50 //Радиус 2 препятствия

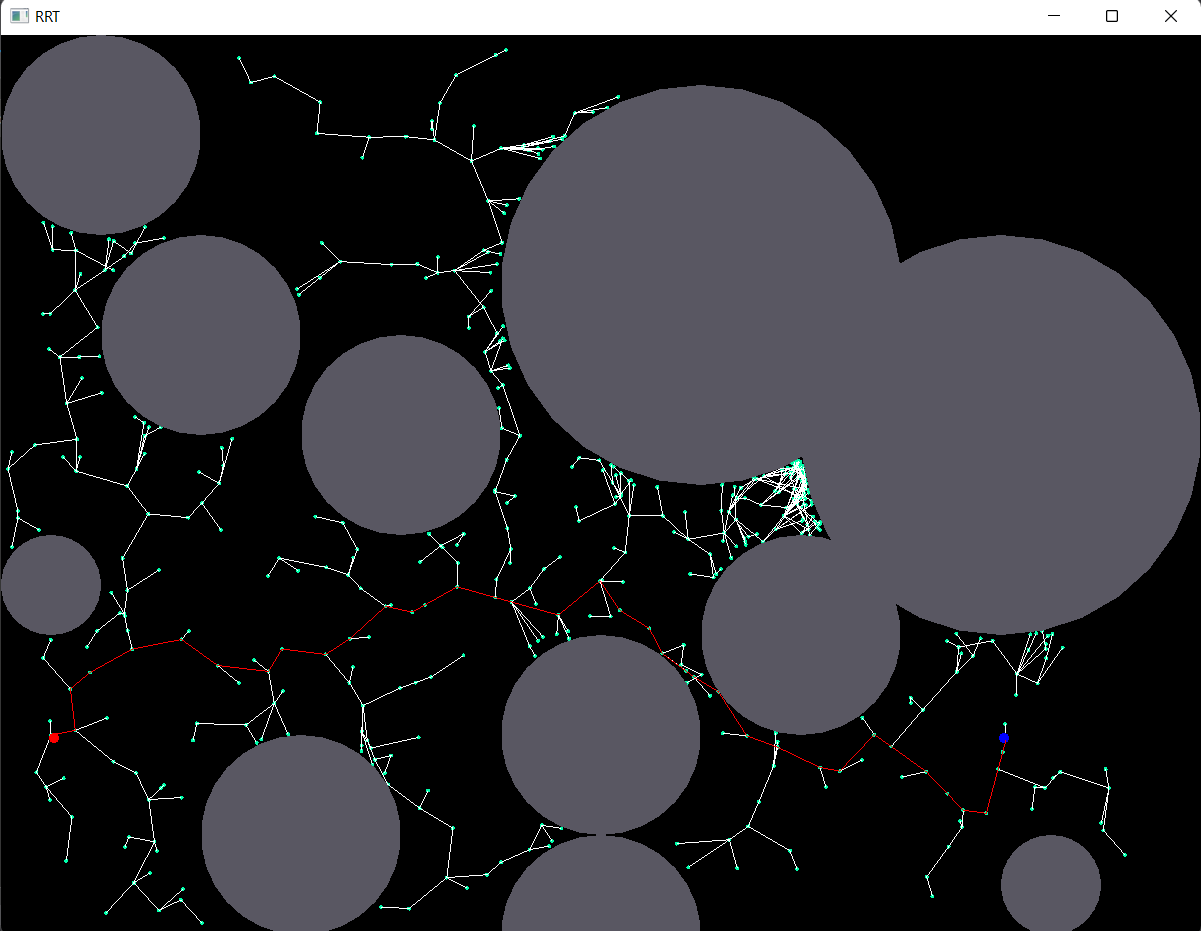
**Скриншоты работы программы**

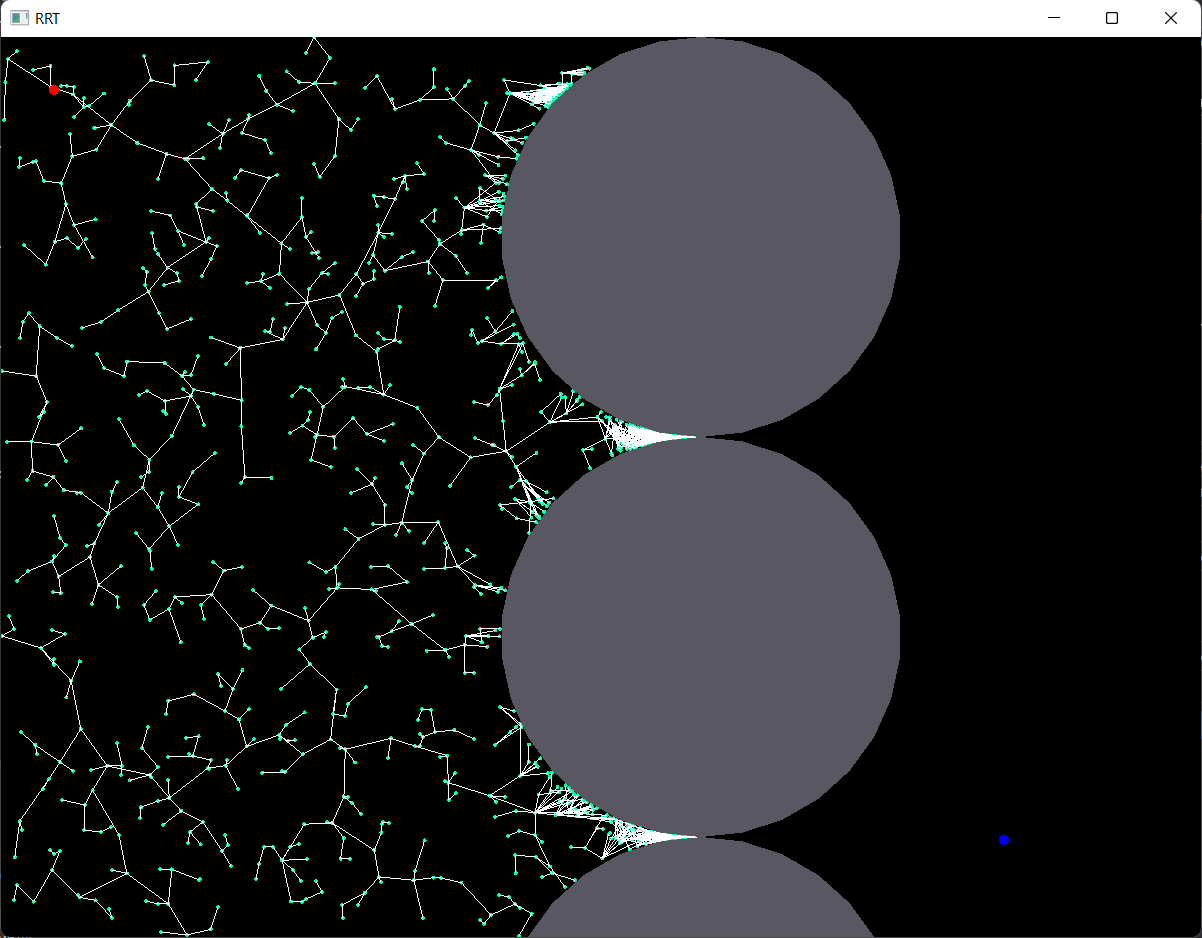
**Алгоритм RRT:**

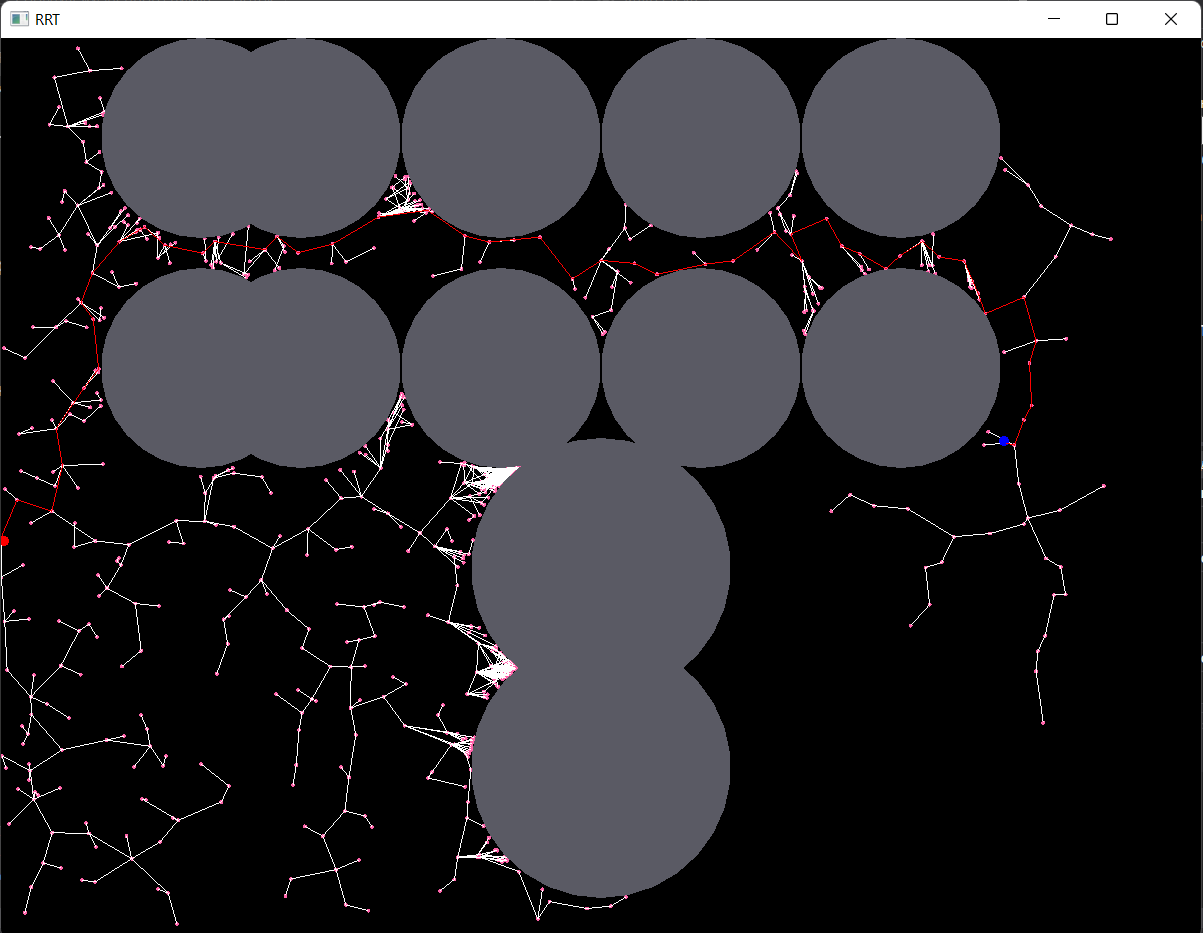




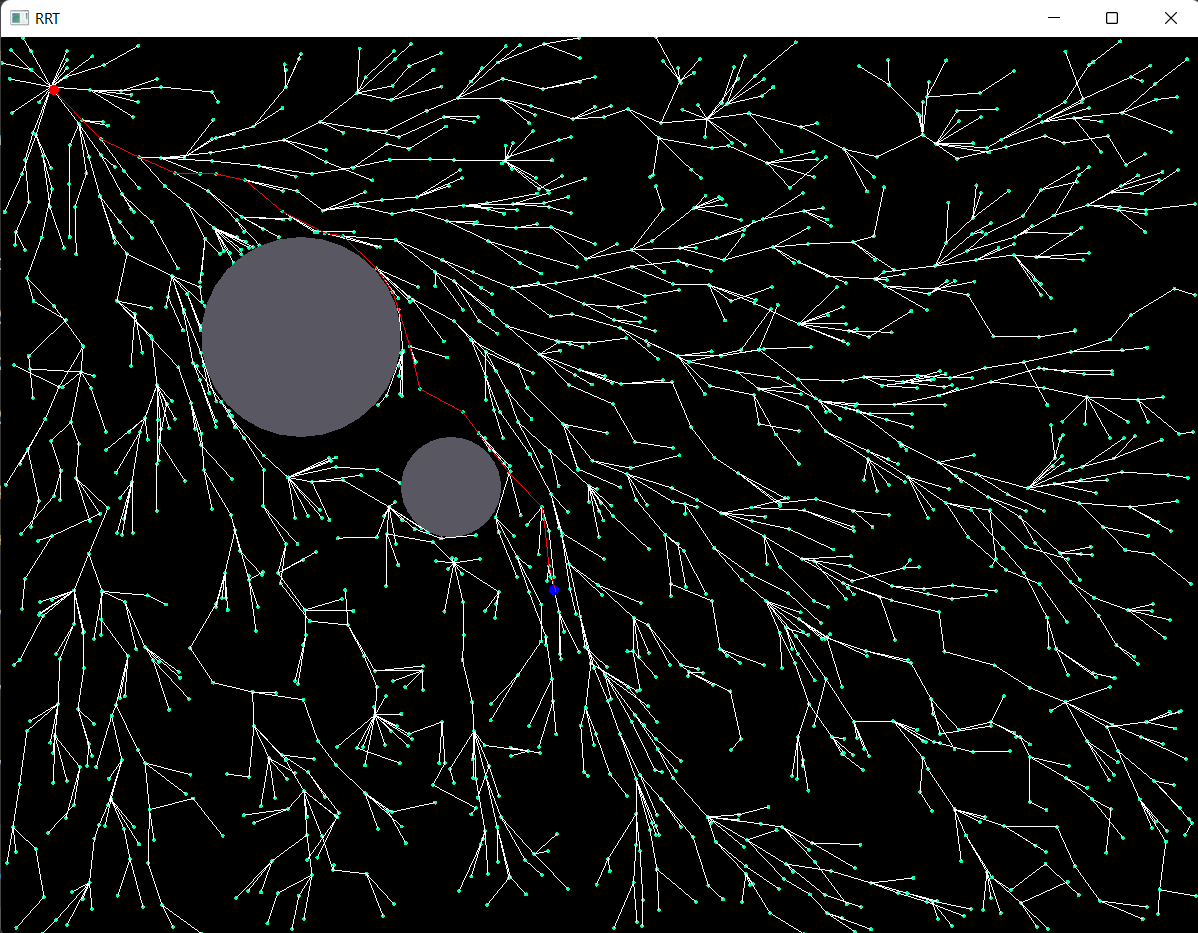


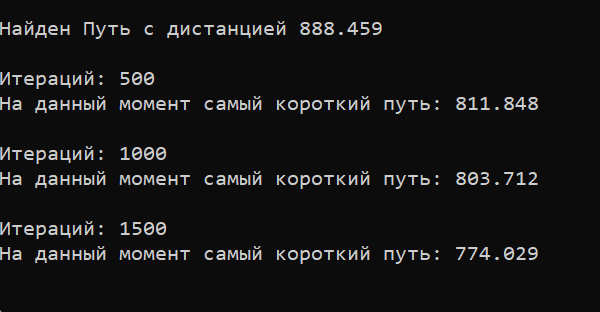


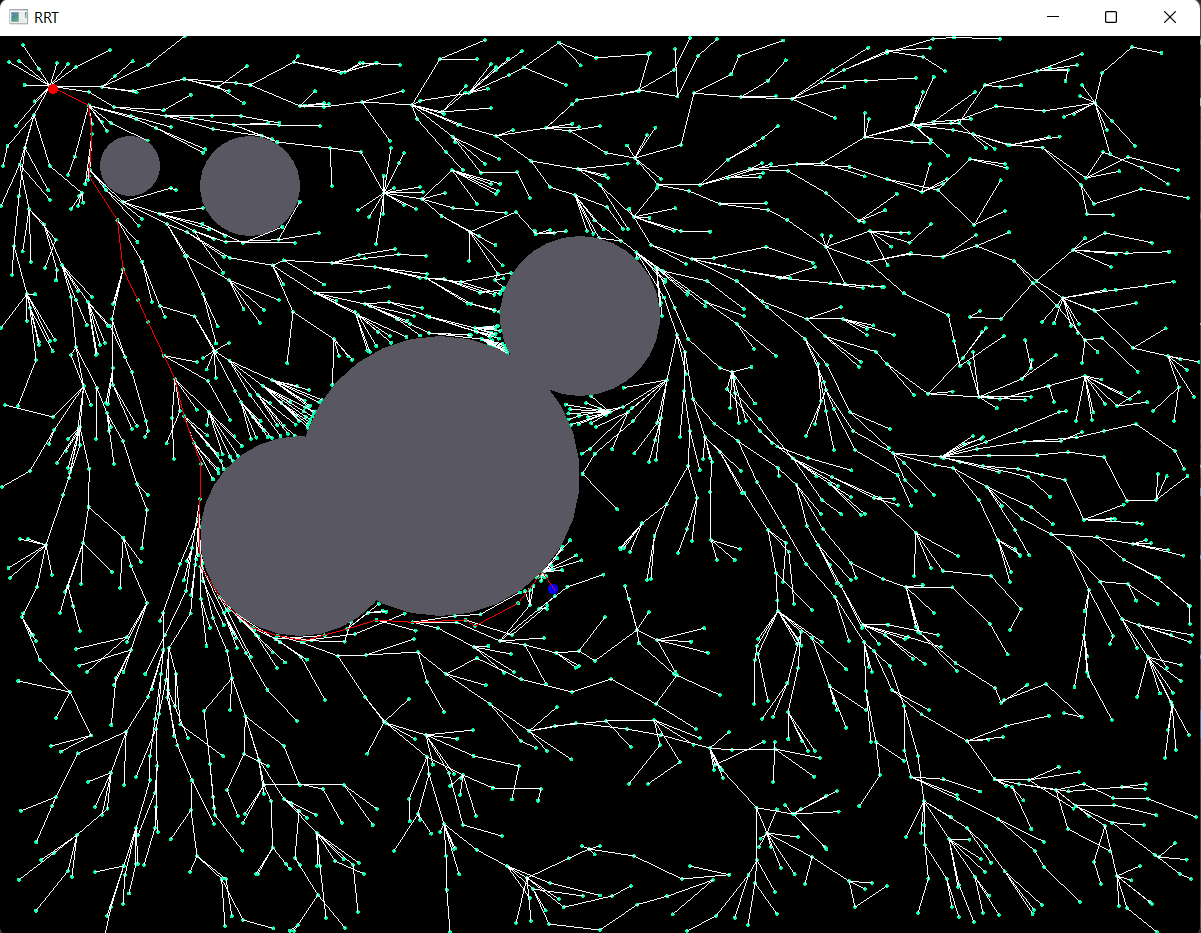


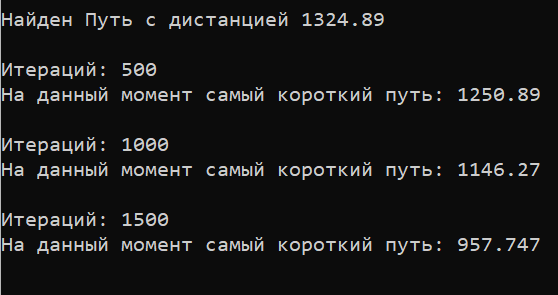


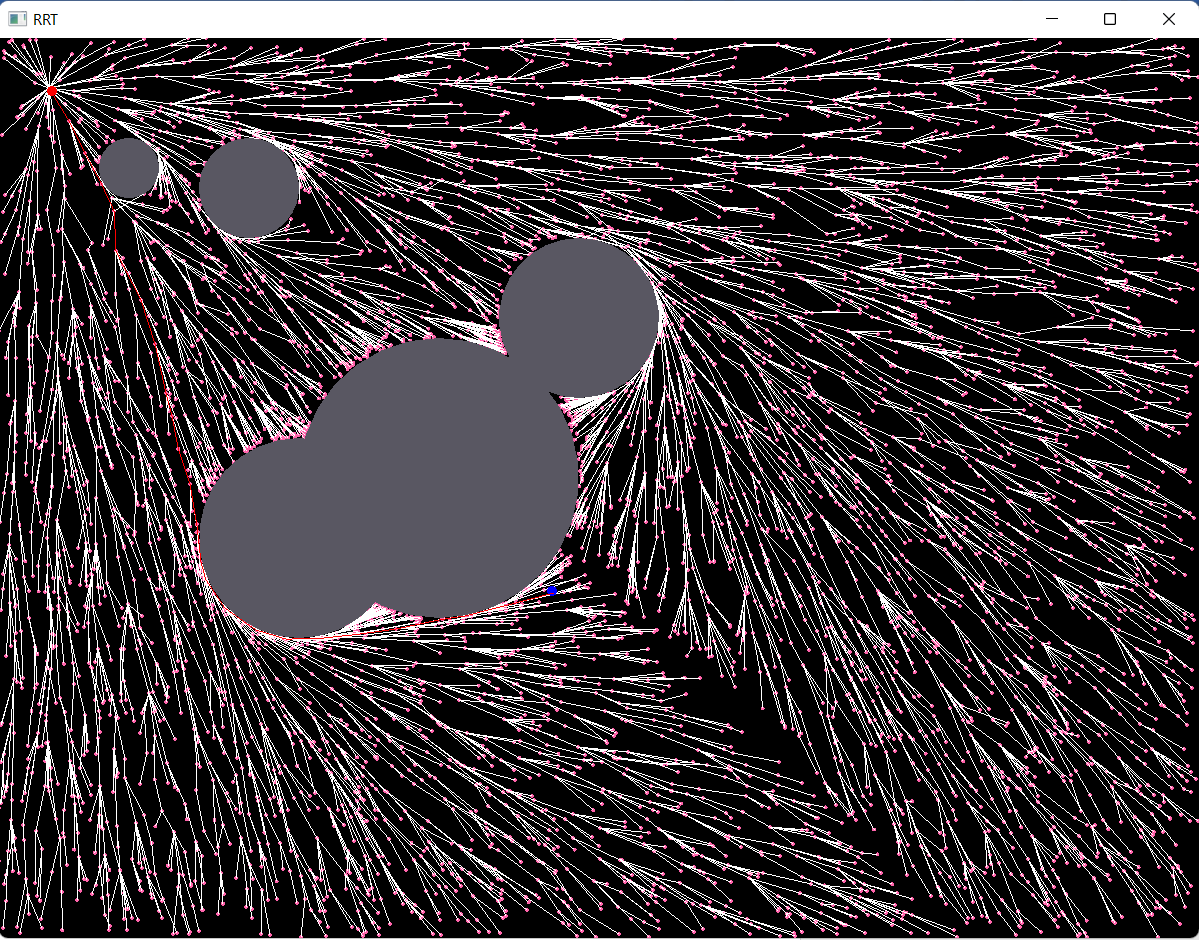
**Алгоритм RRT\*:**

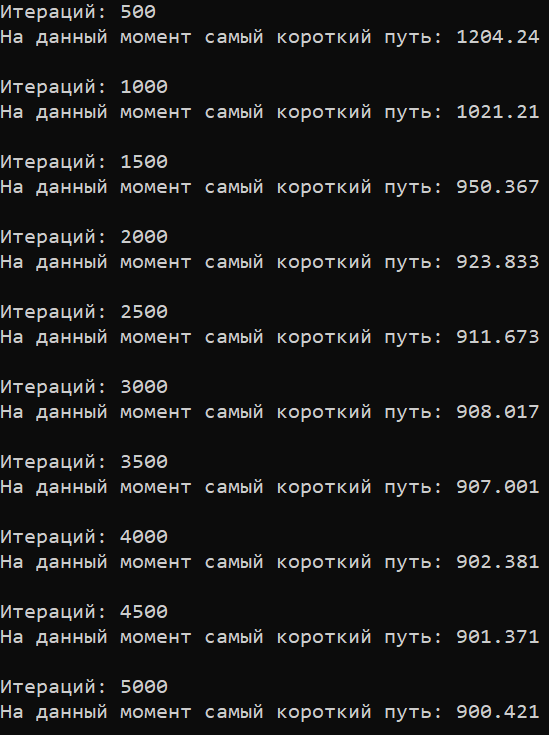


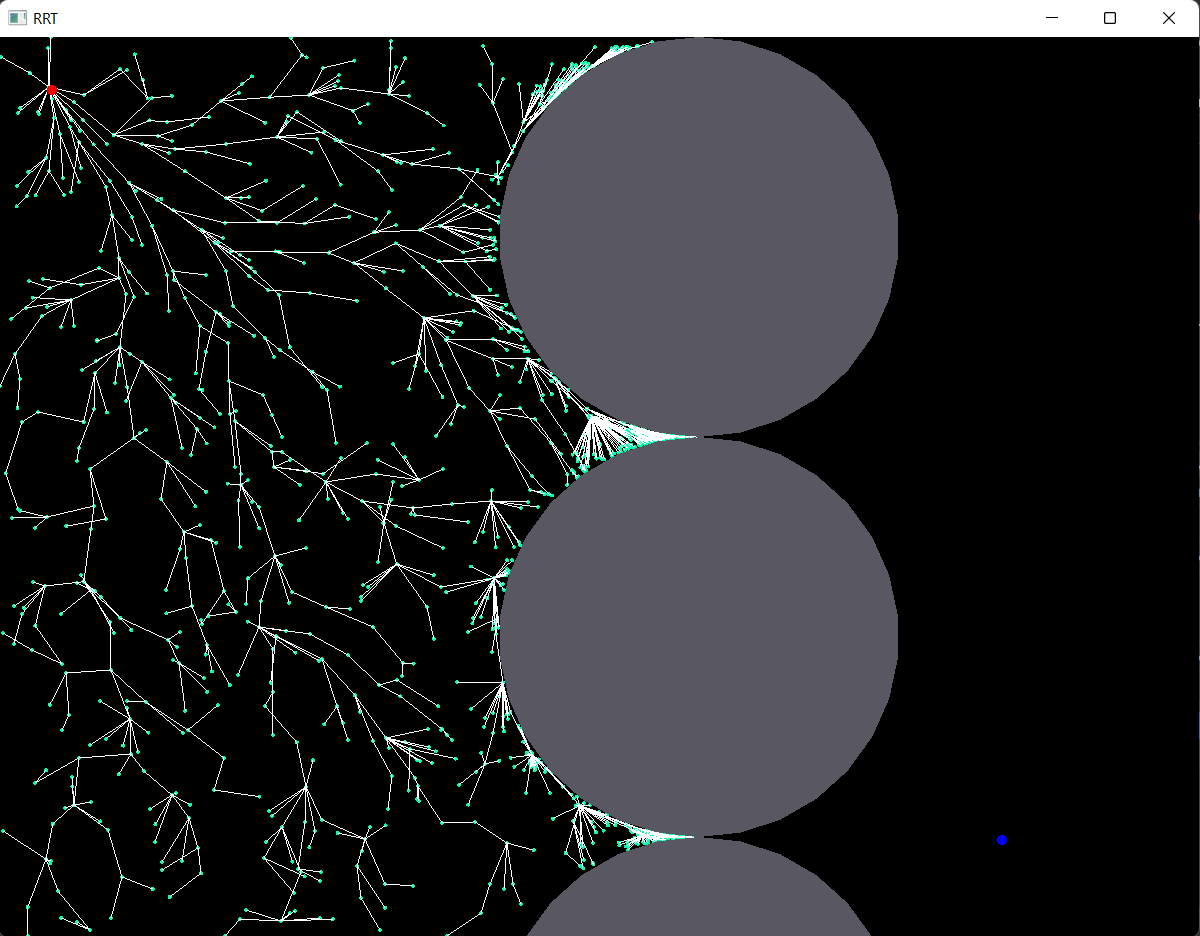


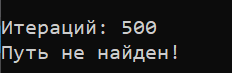


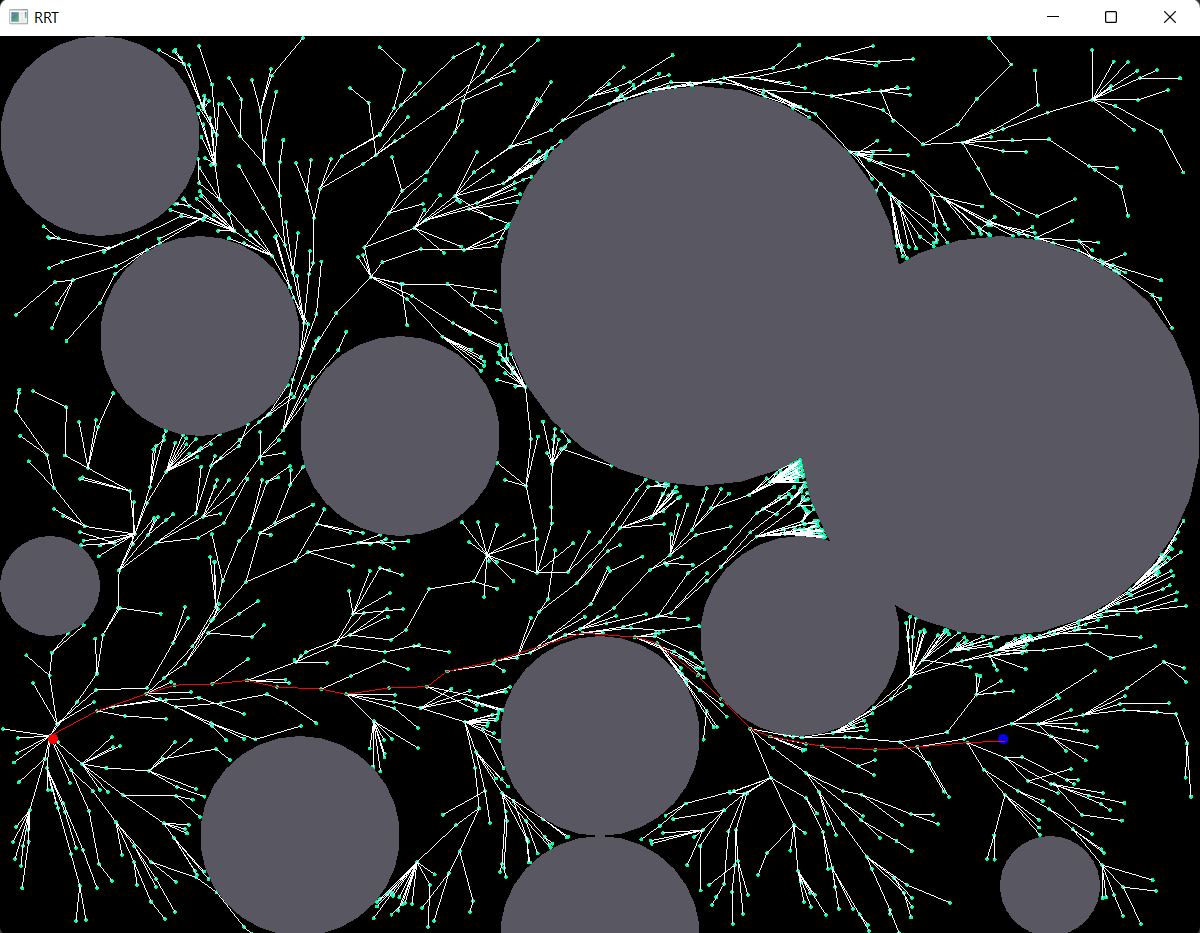


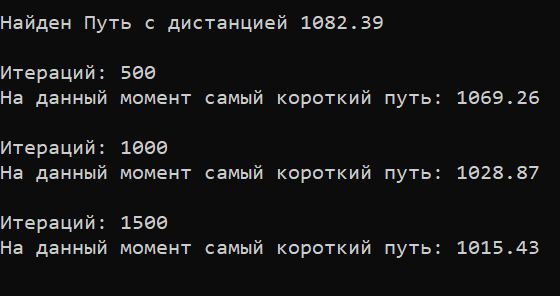


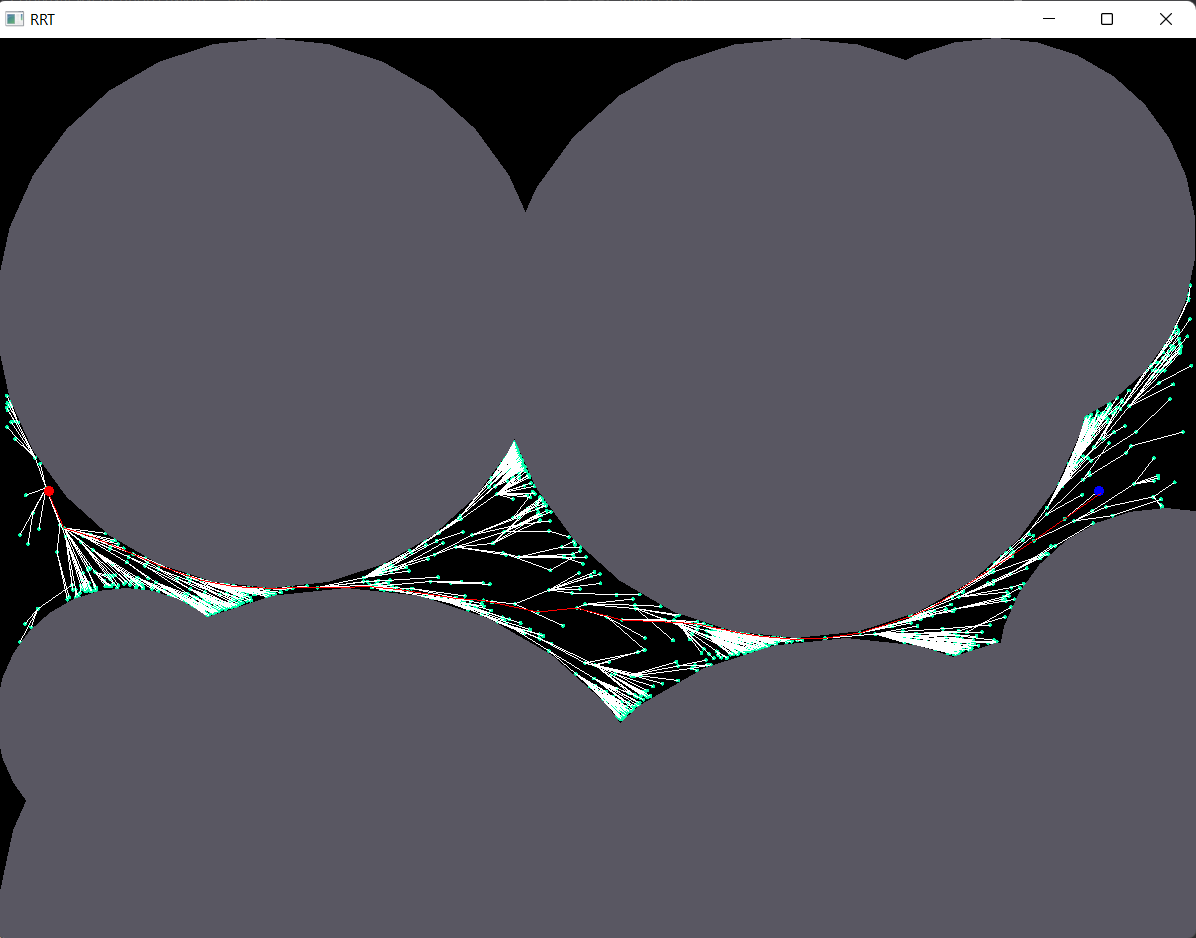


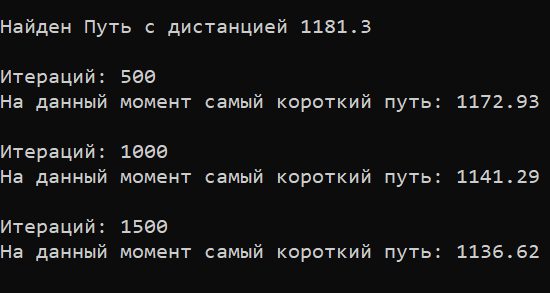


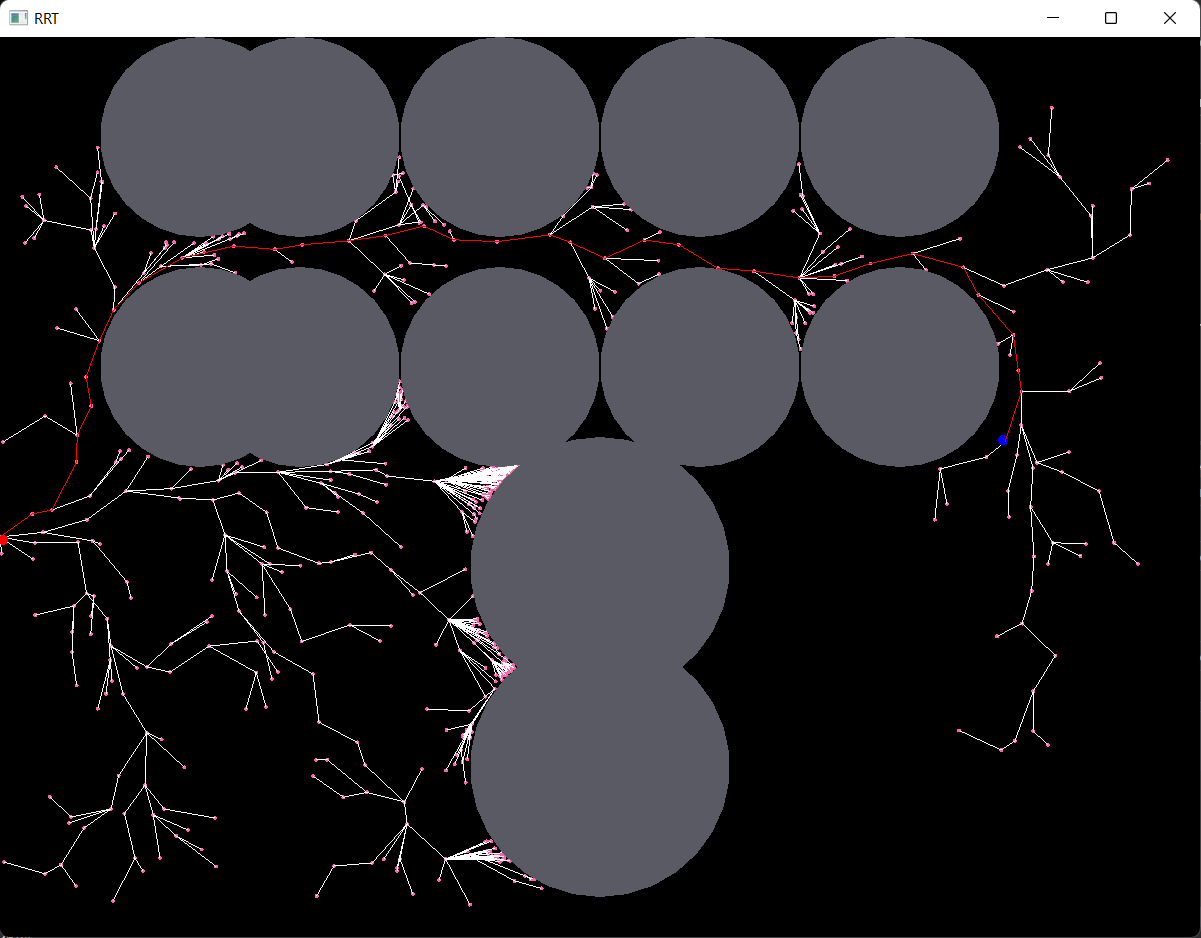


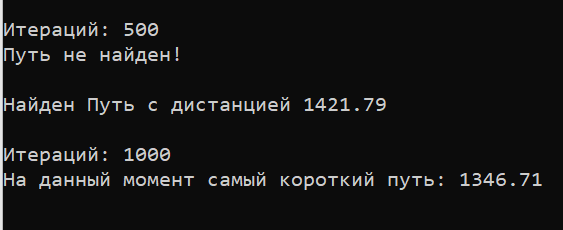












Код программы:

**geometry.h:**

#ifndef GEOMETRY\_H

#include <math.h>

#include <iostream>

#include <fstream>

#include <random>

using namespace std;

#define ftype double

const ftype EPS = 1e-6;//точность

struct Point {

ftype x, y;

Point() {}

Point(ftype x, ftype y) : x(x), y(y) {}

Point &operator+=(const Point &t) {

x += t.x;

y += t.y;

return \*this;

}

Point &operator-=(const Point &t) {

x -= t.x;

y -= t.y;

return \*this;

}

Point &operator\*=(ftype t) {

x \*= t;

y \*= t;

return \*this;

}

Point &operator/=(ftype t) {

x /= t;

y /= t;

return \*this;

}

Point operator+(const Point &t) const {

return Point(\*this) += t;

}

Point operator-(const Point &t) const {

return Point(\*this) -= t;

}

Point operator\*(ftype t) const {

return Point(\*this) \*= t;

}

Point operator/(ftype t) const {

return Point(\*this) /= t;

}

ftype dot(const Point &t) const {

return (x \* t.x + y \* t.y);

}

ftype cross(const Point &t) const {

return x \* t.y - y \* t.x;

}

ftype cross(const Point &a, const Point &b) const {

return (a - \*this).cross(b - \*this);

}

ftype distance(const Point &t) const {

const double x\_diff = x - t.x, y\_diff = y - t.y;

return sqrt(x\_diff \* x\_diff + y\_diff \* y\_diff);

}

Point steer(const Point &t, ftype DELTA) {

if (this->distance(t) < DELTA) {

return t;

}

else {

double theta = atan2(t.y - y, t.x - x);

return Point(x + DELTA \* cos(theta), y + DELTA \* sin(theta));

}

}

bool operator==(const Point &rhs) const

{

return fabs(x - rhs.x) < EPS and fabs(y - rhs.y) < EPS;

}

};

Point operator\*(ftype a, Point b) {

return b \* a;

}

ftype distance(Point &a, Point &b) {

const ftype x\_diff = a.x - b.x, y\_diff = a.y - b.y;

return sqrt(x\_diff \* x\_diff + y\_diff \* y\_diff);

}

ftype dot(Point a, Point b) {

return (a.x \* b.x + a.y \* b.y);

}

ftype cross(Point a, Point b) {

return (a.x \* b.y - b.x \* a.y);

}

Point stepNear(Point &p1, Point &p2, ftype DELTA)//возвращает точку в направлении вектора p2p1

{

if ((distance(p1, p2) - DELTA) <= EPS)

return p2;

else {

ftype theta = atan2(p2.y - p1.y, p2.x - p1.x);

return Point(p1.x + DELTA \* cos(theta), p1.y + DELTA \* sin(theta));

}

}

struct circ

{

Point points;

int radius;

void addPoint(const Point pnt) {

points.x=pnt.x;

points.y = pnt.y;

}

void addRadius(const int r) {

radius=r;

}

int returnRad()

{

return radius;

}

Point returnPoint()

{

return points;

}

};

bool checkCollision(Point lineFrom, Point lineTo, Point location, ftype radius)

{

location += Point(radius, radius);

ftype ab2, acab, h2;

Point ac = location - lineFrom;

Point ab = lineTo - lineFrom;

ab2 = dot(ab, ab); acab = dot(ac, ab);

ftype t = acab / ab2;

if (t < 0) t = 0;

else if (t > 1) t = 1;

Point h = ((ab \* t) + lineFrom) - location;

h2 = dot(h, h);

return (h2 <= (radius \* radius));

}

bool lineSegmentIntersectsCircle(Point lineFrom, Point lineTo, circ O) {

O.points += Point(O.radius, O.radius);

ftype ab2, acab, h2;

Point ac = O.points - lineFrom;

Point ab = lineTo - lineFrom;

ab2 = dot(ab, ab); acab = dot(ac, ab);

ftype t = acab / ab2;

if (t < 0) t = 0;

else if (t > 1) t = 1;

Point h = ((ab \* t) + lineFrom) - O.points;

h2 = dot(h, h);

return (h2 <= (O.radius \* O.radius));

}

#endif

**Main.cpp:**

#include "geometry.h"

#include <SFML/Graphics.hpp>

using namespace std;

const int WIDTH = 1200;

const int HEIGHT = 900;

int RADIUS;

const double GOAL\_SAMPLING\_PROB = 0.05;

const double INF = 1e18;

const double JUMP\_SIZE = (WIDTH / 100.0 \* HEIGHT / 100.0)/2;

const double DISK\_SIZE = JUMP\_SIZE;

int whichRRT;

vector < circ > obstacles;

Point start, stop;

int obstacle\_cnt;

vector < Point > nodes;

vector < int > parent, nearby;

vector < double > cost, jumps;

int nodeCnt = 0, goalIndex = -1;

vector <sf::CircleShape> circles;

sf::CircleShape startingPoint, endingPoint;

bool pathFound = 0;

struct file

{

int RRT;

Point start;

Point end;

int pntrad;

int count;

vector<Point> XY;

vector<int> radius;

void addpntrad(int r)

{

pntrad = r;

}

void addRRT(int rrt)

{

RRT = rrt;

}

void addC(int c)

{

count = c;

}

void addRad(int r)

{

radius.push\_back(r);

}

void addXY(Point xy)

{

XY.push\_back(xy);

}

void addS(Point xy)

{

start = xy;

}

void addE(Point xy)

{

end = xy;

}

};

Point temp;

vector<file> File;

void getInput() {

cout << "Ширина экрана: " << WIDTH << endl;

cout << "Высота экрана: " << HEIGHT << endl;

cout << "RRT - 1, RRT\* - 2" << endl;

cin >> whichRRT;

if (whichRRT < 1 || whichRRT>2) throw 0;

cout << "Радиус робота" << endl;

cin >> RADIUS;

if (RADIUS < 1) throw 1;

cout << "Введите координаты двух точек X1 Y1 X2 Y2" << endl;

cin >> start.x >> start.y >> stop.x >> stop.y;

if (start.x < 0 || start.y<0 || start.y>HEIGHT || start.x > WIDTH) throw 2;

if (stop.x < 0 || stop.y<0 || stop.y>HEIGHT || stop.x > WIDTH) throw 3;

cout << "Как много кругов?" << endl;

cin >> obstacle\_cnt;

if (obstacle\_cnt < 0) throw 4;

Point xy;

int r;

obstacles.resize(obstacle\_cnt);

circles.resize(obstacle\_cnt);

for (int i = 0; i < obstacle\_cnt; i++)

{

cout << "Введите координаты " << i + 1 << " круга" << endl;

cin >> xy.x >> xy.y;

if (xy.x < 0 || xy.y<0 || xy.y>HEIGHT || xy.x > WIDTH) throw 5;

obstacles[i].addPoint(xy);

cout << "Введите радиус " << i + 1 << " круга" << endl;

cin >> r;

if (r < 1) throw 6;

obstacles[i].addRadius(r);

}

}

void fileInput()

{

ifstream fin("set.txt");

fin >> temp.x;

if (temp.x < 1) throw -1;

File.resize(temp.x);

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

{

fin >> temp.x;

if (temp.x < 1 || temp.x>2) throw 0;

File[i].addRRT(temp.x);//RRT

fin >> temp.x;

if (temp.x < 1) throw 1;

File[i].addpntrad(temp.x);

fin >> temp.x;

fin >> temp.y;

if (temp.x < 0 || temp.y<0 || temp.y>HEIGHT || temp.x > WIDTH) throw 2;

File[i].addS(temp);//start

fin >> temp.x;

fin >> temp.y;

if (temp.x < 0 || temp.y<0 || temp.y>HEIGHT || temp.x > WIDTH) throw 3;

File[i].addE(temp);//end

fin >> temp.x;

if (temp.x < 0) throw 4;

File[i].addC(temp.x);

for (int j=0; j < File[i].count; j++)

{

fin >> temp.x;

fin >> temp.y;

if (temp.x < 0 || temp.y<0 || temp.y>HEIGHT || temp.x > WIDTH) throw 5;

File[i].addXY(temp);

fin >> temp.x;

if (temp.x < 1) throw 6;

File[i].addRad(temp.x);

}

}

int choose(-1);

while (choose < 0 || choose >= File.size())

{

cout << "Выберите сцену, доступно: " << File.size() << endl;

cin >> choose;

choose--;

}

whichRRT = File[choose].RRT;

RADIUS= File[choose].pntrad;

start.x = File[choose].start.x;

start.y= File[choose].start.y;

stop.x= File[choose].end.x;

stop.y= File[choose].end.y;

obstacle\_cnt= File[choose].count;

obstacles.resize(obstacle\_cnt);

circles.resize(obstacle\_cnt);

for (int i = 0; i < obstacle\_cnt; i++)

{

obstacles[i].addPoint(File[choose].XY[i]);

obstacles[i].addRadius(File[choose].radius[i]);

}

}

void save()

{

ofstream fout("save.txt");

fout << whichRRT<<endl;

fout << start.x <<" "<< start.y << endl;

fout << stop.x << " " << stop.y << endl;

fout << obstacle\_cnt << endl;

for (int i = 0; i < obstacle\_cnt; i++)

{

Point temp=obstacles[i].returnPoint();

fout << temp.x << " " << temp.y<<endl;

fout << obstacles[i].returnRad()<<endl;

}

}

// Подготовка препятствий и точек для SFML

void prepareInput() {

startingPoint.setRadius(RADIUS); endingPoint.setRadius(RADIUS);

startingPoint.setFillColor(sf::Color::Red); endingPoint.setFillColor(sf::Color::Blue);

startingPoint.setPosition(start.x, start.y); endingPoint.setPosition(stop.x, stop.y);

startingPoint.setOrigin(RADIUS / 2, RADIUS / 2); endingPoint.setOrigin(RADIUS / 2, RADIUS / 2);

for (int i = 0; i < obstacle\_cnt; i++) {

circles[i].setFillColor(sf::Color(90, 90, 100));

circles[i].setRadius(obstacles[i].radius);

circles[i].setPosition(obstacles[i].points.x, obstacles[i].points.y);

}

}

void draw(sf::RenderWindow &window) {

sf::Vertex line[2]; sf::CircleShape nodeCircle;

for(auto& node: nodes) {

nodeCircle.setRadius(2);

nodeCircle.setOrigin(2, 2);

nodeCircle.setFillColor(sf::Color(255, 100, 170)); nodeCircle.setPosition(node.x, node.y);

window.draw(nodeCircle);

}

for (auto &circle : circles) window.draw(circle);

for (int i = (int)nodes.size() - 1; i; i--) {

Point par = nodes[parent[i]];

line[0] = sf::Vertex(sf::Vector2f(par.x, par.y));

line[1] = sf::Vertex(sf::Vector2f(nodes[i].x, nodes[i].y));

window.draw(line, 2, sf::Lines);

}

window.draw(startingPoint); window.draw(endingPoint);

// Если путь найден, красим в красный

if (pathFound) {

int node = goalIndex;

while (parent[node] != node) {

int par = parent[node];

line[0] = sf::Vertex(sf::Vector2f(nodes[par].x, nodes[par].y));

line[1] = sf::Vertex(sf::Vector2f(nodes[node].x, nodes[node].y));

line[0].color = line[1].color = sf::Color::Red;

window.draw(line, 2, sf::Lines);

node = par;

}

}

}

template <typename T>

T randomCoordinate(T low, T high) {

random\_device random\_device;

mt19937 engine{ random\_device() };

uniform\_real\_distribution<double> dist(low, high);

return dist(engine);

}

// Проверка на встречу с препятствием

bool isEdgeObstacleFree(Point a, Point b) {

for (auto &circle : obstacles)

if (lineSegmentIntersectsCircle(a, b, circle)) return false ;

return true;

}

// Случайная точка в сторону цели

Point pickRandomPoint() {

double random\_sample = randomCoordinate(0.0, 1.0);

if ((random\_sample - GOAL\_SAMPLING\_PROB) <= EPS and !pathFound) return stop + Point(RADIUS, RADIUS);

return Point(randomCoordinate(0, WIDTH), randomCoordinate(0, HEIGHT));

}

//Проверка на нахождение пути

void checkDestinationReached() {

sf::Vector2f position = endingPoint.getPosition();

if (checkCollision(nodes[parent[nodeCnt - 1]], nodes.back(), Point(position.x, position.y), RADIUS)) {

pathFound = 1;

goalIndex = nodeCnt - 1;

cout << "Найден Путь с дистанцией " << cost.back() << endl << endl;

}

}

void rewire() {

int lastInserted = nodeCnt - 1;

for (auto nodeIndex : nearby) {

int par = lastInserted, cur = nodeIndex;

while (((cost[par] + distance(nodes[par], nodes[cur])) - cost[cur]) <= EPS) {

int oldParent = parent[cur];

parent[cur] = par; cost[cur] = cost[par] + distance(nodes[par], nodes[cur]);

par = cur, cur = oldParent;

}

}

}

void RRT() {

Point newPoint, nearestPoint, nextPoint; bool updated = false; int cnt = 0;

int nearestIndex = 0; double minCost = INF; nearby.clear(); jumps.resize(nodeCnt);

while (!updated) {

newPoint = pickRandomPoint();

nearestPoint = \*nodes.begin(); nearestIndex = 0;

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

if (pathFound && randomCoordinate(0.0, 1.0) < 0.25)

cost[i] = cost[parent[i]] + distance(nodes[parent[i]], nodes[i]);

// Для прохождения препятствий делаем прыжки меньше

jumps[i] =randomCoordinate(0.3, 1.0) \* JUMP\_SIZE;

auto pnt = nodes[i];

if ((pnt.distance(newPoint) - nearestPoint.distance(newPoint)) <= EPS and isEdgeObstacleFree(pnt, pnt.steer(newPoint, jumps[i])))

nearestPoint = pnt, nearestIndex = i;

}

nextPoint = stepNear(nearestPoint, newPoint, jumps[nearestIndex]);

if (!isEdgeObstacleFree(nearestPoint, nextPoint)) continue;

if (whichRRT == 1)

{

updated = true;

nodes.push\_back(nextPoint); nodeCnt++;

parent.push\_back(nearestIndex);

cost.push\_back(cost[nearestIndex] + distance(nearestPoint, nextPoint));

if (!pathFound) checkDestinationReached();

continue;

}

for (int i = 0; i < nodeCnt; i++)

if ((nodes[i].distance(nextPoint) - DISK\_SIZE) <= EPS and isEdgeObstacleFree(nodes[i], nextPoint))

nearby.push\_back(i);

int par = nearestIndex; minCost = cost[par] + distance(nodes[par], nextPoint);

for (auto nodeIndex : nearby) {

if (((cost[nodeIndex] + distance(nodes[nodeIndex], nextPoint)) - minCost) <= EPS)

minCost = cost[nodeIndex] + distance(nodes[nodeIndex], nextPoint), par = nodeIndex;

}

parent.push\_back(par); cost.push\_back(minCost);

nodes.push\_back(nextPoint); nodeCnt++;

updated = true;

if (!pathFound) checkDestinationReached();

rewire();

}

}

int main() {

setlocale(LC\_ALL, "rus");

bool flag = 1;

while (flag)

{

int select(-1);

int iter;

while (select < 1 || select >2)

{

cout << "Ввод из файла - 1, через консоль - 2" << endl;

cin >> select;

}

try

{

if (select == 1) fileInput(); else getInput();

cout << "Кол-во итераций" << endl;

cin >> iter;

if (iter < 1) throw;

}

catch (...)

{

cout << "Ошибка ввода данных!" << endl;

system("pause");

exit(0);

}

prepareInput();

nodeCnt = 1;

nodes.push\_back(start);

int iterations = 0;

parent.push\_back(0);

cost.push\_back(0);

sf::RenderWindow window(sf::VideoMode(WIDTH, HEIGHT), "RRT");

sf::Time delayTime = sf::milliseconds(10);

cout << endl << "Стартовая точка - красная, конечная - синия" << endl << endl;

while (window.isOpen()&&iter>=iterations)

{

sf::Event event;

while (window.pollEvent(event))

{

if (event.type == sf::Event::Closed)

{

window.close();

//return 0; exit(0);

}

}

RRT(); iterations++;

if (iterations % 500 == 0||iter== iterations) {

cout << "Итераций: " << iterations << endl;

if (!pathFound) cout << "Путь не найден! " << endl;

else cout << "На данный момент самый короткий путь: " << cost[goalIndex] << endl;

cout << endl;

}

window.clear();

draw(window);

window.display();

}

save();

nodeCnt = 0;

goalIndex = -1;

pathFound = 0;

nodes.clear();

parent.clear();

cost.clear();

nearby.clear();

jumps.clear();

}

}