МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра САПР

ОТЧЕТ
по курсовой работе
по дисциплине «Алгоритмы и структуры данных»
тема «Визуализация работы дерева квадрантов»

Студент гр. 2301 _____ Комиссаров П.Е.

Преподаватель: _____ Пестерев Д.О.

Санкт-Петербург
2023

1. Постановка задачи

1. Реализовать визуализацию работы квадратичного дерева

2. Реализация должна включать себя экран, разделенный на 2 области:

- Область с экраном, на котором можно добавлять, удалять, выделять точки. При наличии 5 точек в квадранте (соответствующей части экрана), квадрант делится на 4 равные области.
- Область с визуализацией дерева квадрантов. Визуализация содержит в себе связанное узлами дерево, в котором узлами являются соответствующие квадранты (деления экрана). Точки, находящиеся в соответствующем квадранте, отображаются в виде прямоугольника (все точки включены в прямоугольник)

3. Связать оба экрана между собой. При выделении точки на экране, на дереве выделяется соответствующий прямоугольник (в соответствующем узле) на дереве зеленым цветом. Если квадрант не содержит в себе точек, то прямоугольник выделяется белым цветом (в ином случае - черным). Если на экране выбрать квадрант (на соответствующую клавишу на клавиатуре), то в дереве узел выделится зеленым цветом.

2. Описание алгоритмов

Дерево квадрантов (Quadtree) — это древовидная структура данных, используемая для организации пространственных данных в двумерном пространстве. Оно используется для хранения данных, которые могут быть представлены в виде точек в двумерном пространстве.

Каждый узел дерева квадрантов представляет собой прямоугольник, который делит пространство на четыре равные части. Каждый узел также может иметь до четырех дочерних узлов, которые также являются прямоугольниками. Это позволяет эффективно разбивать пространство на более мелкие части, что упрощает поиск и обработку данных.

Плюсы и минусы использования дерева квадрантов:

Плюсы:

1. Эффективность при хранении и поиске точек:
   - Квадрантовые деревья обеспечивают эффективное разделение и хранение точек в двумерном пространстве, что делает их подходящими для задач хранения геопространственных данных.
2. Быстрый поиск и фильтрация:
   - Поиск в деревьях квадрантов выполняется быстро, особенно когда необходимо быстро определить точки, попадающие в определенные области.
3. Поддержка динамических данных:
   - Деревья квадрантов могут быть легко адаптированы для вставки и удаления точек, что делает их подходящими для задач с динамически изменяющимися данными.

Минусы:

1. Чувствительность к распределению данных:
   - Деревья квадрантов могут быть менее эффективными, если данные сосредоточены в небольших областях, что может привести к увеличенной глубине.
2. Дополнительные затраты на хранение:
   - Деревья квадрантов могут потреблять дополнительные ресурсы для хранения указателей и данных в узлах, что может быть проблемой при работе с большими объем

Применение дерева квадрантов

Деревья квадрантов широко применяются в различных областях, где необходимо организовать и эффективно обрабатывать данные в двумерном пространстве.

1. ***Геоинформационные системы (ГИС)***: Деревья квадрантов являются популярным выбором для индексации и поиска географических данных. Они позволяют быстро находить объекты в заданных пространственных областях, таких как географические карты, снимки со спутников и другие географические данные.

2. ***Компьютерная графика:*** В графических приложениях, таких как обработка изображений и визуализация данных, деревья квадрантов могут быть использованы для ускорения поиска и обработки точек в изображении

3. ***Обработка сенсорных данных:*** В задачах, связанных с обработкой данных от сенсоров (например, в сенсорных сетях или IoT-устройствах), деревья квадрантов могут помочь эффективно организовывать и искать данные в пространстве

4. ***Симуляции и визуализации в компьютерных играх:*** Деревья квадрантов могут быть применены для организации и быстрого поиска объектов в игровом мире, упрощая процессы обнаружения столкновений и взаимодействия объектов

5. ***Астрономия и космология:*** В областях, где анализируются и обрабатываются данные о распределении звезд, галактик и других объектов в космосе, деревья квадрантов могут использоваться для оптимизации поиска и анализа

***Код визуализации*:**

```
#include <cmath>
#include <iostream>
#include <SFML/Graphics.hpp>
#include <vector>
#include<string>
#include <thread>
#include <chrono>

using namespace std;
using namespace sf;

#define SCREEN_W 1580
#define SCREEN_H 900

#define SCREEN_TREE_W 600
#define SCREEN_TREE_H 800

#define SCREEN_TREE_NODE_W 500
#define SCREEN_TREE_NODE_H 400

#define SCREEN_SPACE 50

#define MAX_LEVEL 4
#define CAPACITY 4

#define MAX_SCROLL 30
#define MIN_SCROLL 2

#define SCALE_STEP 1
#define SCALE_MAX 3

#define STEP 20

#define RADIUS 4
#define THICKNESS 2

#define TREE_SIZE_X 1100
#define TREE_SIZE_Y 800

#define TEXT_SIZE_BIG 24
#define TEXT_SIZE_STANDART 18

#define SCREEN_COLOR  31, 31, 31

class Point {
public:
    int x;
    int y;
    bool highlighted;
    Point(int _x, int _y, bool hltd = false)
    {
        x = _x;
        y = _y;
        highlighted = hltd;
    }
    Point()
    {
        x = 0;
        y = 0;
        highlighted = false;
```

```
    }
    void ColorHighlighted(Point* p) {
        p->highlighted = true;
    }
};

class Rectangle {
public:
    float x, y, w, h;
    Rectangle(){}

    Rectangle(float _x, float _y, float _w, float _h) : x(_x), y(_y), w(_w), h(_h) {
}



    bool contains(const Point& p) const {
        return p.x >= x - w && p.x <= x + w     //проверяем, содержится ли точка в п
рямоугольнике
            && p.y >= y - h && p.y <= y + h;
    }

};

class cursorRectangle {
public:
    float x, y, w, h;
    cursorRectangle() {}

    cursorRectangle(float _x, float _y, float _w, float _h) : x(_x), y(_y), w(_w), h
(_h) {}

    bool contains(const Point& p) const {
        return p.x >= x - w && p.x <= x + w     //проверяем, содержится ли точка в п
рямоугольнике
            && p.y >= y - h && p.y <= y + h;
    }

    void draw(RenderTarget& t, bool cursorInArea) {
        static Vertex vertices[5];
        Color color;
        if (cursorInArea) {
            color = Color::White;
            color.a = 200;
        }
        else {
            color = Color::Transparent;
            color.a = 0;
        }

        vertices[0] = Vertex(Vector2f(x - w, y - h), color);
        vertices[1] = Vertex(Vector2f(x + w, y - h), color);
        vertices[2] = Vertex(Vector2f(x + w, y + h), color);
        vertices[3] = Vertex(Vector2f(x - w, y + h), color);
        vertices[4] = Vertex(Vector2f(x - w, y - h), color);
        t.draw(vertices, 5, LinesStrip);
    }


    bool intersects(Rectangle& other) {
        return !(x - w > other.x + other.w || x + w < other.x - other.w || y - h>oth
er.y + other.h || y + h < other.y - other.h);
    }
```

```
};

class NodeRectangle {

    sf::RectangleShape rectangle;
    float OutlineThickness= THICKNESS;
    public:
        NodeRectangle(float radiusCircle) {
            rectangle.setSize(sf::Vector2f(radiusCircle, radiusCircle*2));

            rectangle.setOutlineColor(Color::Black);
        }

        bool havePoints = false;
        bool selectPoint = false;
        void setOutlineThickness(float scale) {
            rectangle.setOutlineThickness(OutlineThickness*scale);
        }

        void setPosition(float x, float y) {
            rectangle.setPosition(x, y);
        }
        void drawRectangle(RenderTarget& t) {
            if (havePoints) rectangle.setFillColor(Color::Black);

            else  rectangle.setFillColor(Color::White);
            if (selectPoint) rectangle.setFillColor(Color::Green);

            t.draw(rectangle);
            havePoints = false;
            selectPoint = false;
        }

};

class Circle {
public:
    float screenPosX;
    float screenSizeX;
    float screenSizeY;
    float screenPosY;
    float scale;
    float radius;
    bool select = false;
    CircleShape CircleTree;

     Circle( float radius, float scale, float sizeX, float sizeY, float posX, float
posY) {

        this->radius = radius;
        this->scale = scale;
        this->screenSizeX = sizeX;
        this->screenSizeY = sizeY;
        this->screenPosX = posX;
        this->screenPosY = posY;
        CircleTree.setPosition(posX, posY);
        CircleTree.setRadius(radius);
    }
     void updateSize() {
         screenSizeX*=scale;
         screenSizeY *= scale;
     }
     void setRadius(float radius) {
```

```
            CircleTree.setRadius(radius*scale);
    }
      float GetRadius() {
          return radius * scale;
      }
      void setPosition(float posX, float posY) {
          CircleTree.setPosition(posX, posY);
      }


     void draw(RenderTarget& t) {
         if (select) CircleTree.setFillColor(Color::Green);
         else CircleTree.setFillColor(Color::White);
         CircleTree.setOutlineThickness(THICKNESS * scale);
         CircleTree.setOutlineColor(Color::Black);
         t.draw(CircleTree);
         select = false;

     }
};

class Quad {

        Quad* topLeftTree;
        Quad* topRightTree;
        Quad* botLeftTree;
        Quad* botRightTree;


        Rectangle boundaries;
        size_t capacity;
        size_t level;
        bool divided;

        vector<Point*> children;

        void subdivide(int &levelPresent) {
            static Vector2f halfSize;
            halfSize.x = boundaries.w / 2.0f;
            halfSize.y = boundaries.h / 2.0f;
            topLeftTree = new Quad(Rectangle(boundaries.x - halfSize.x, boundaries.y
 - halfSize.y, halfSize.x, halfSize.y), capacity, level + 1);
            topRightTree = new Quad(Rectangle(boundaries.x + halfSize.x, boundaries.
y - halfSize.y, halfSize.x, halfSize.y), capacity, level + 1);

            botLeftTree = new Quad(Rectangle(boundaries.x - halfSize.x, boundaries.y
 + halfSize.y, halfSize.x, halfSize.y), capacity, level + 1);
            botRightTree= new Quad(Rectangle(boundaries.x + halfSize.x, boundaries.y
 + halfSize.y, halfSize.x, halfSize.y), capacity, level + 1);

            if (levelPresent < level + 1) {
                levelPresent = level + 1;
            }

            divided = true;
        }

        int countChildren(int &count) {
            if (divided) {
                if (topLeftTree->divided)
                    count += topLeftTree->countChildren(count);
                if (topRightTree->divided)
                    count += topRightTree->countChildren(count);
```

```
            if (botRightTree->divided)
                count += botRightTree->countChildren(count);
            if (botLeftTree->divided)
                count += botLeftTree->countChildren(count);

            count += topLeftTree->children.size();
            count += topRightTree->children.size();
            count += botRightTree->children.size();
            count += botLeftTree->children.size();
        }

        return count;
    }

public:

    Quad(const Rectangle& _boundaries, size_t _capacity, size_t _level) {
        topLeftTree = NULL;
        topRightTree = NULL;
        botLeftTree = NULL;
        botRightTree = NULL;
        boundaries = _boundaries;
        divided = false;
        capacity = _capacity;
        level = _level;
        if (level >= MAX_LEVEL)
            capacity = 0;

    }
    ~Quad()
    {
        if (divided) {
            delete topLeftTree;
            delete topRightTree;
            delete botLeftTree;
            delete botRightTree;
        }
    }


    bool insert(Point* p, int& levelPresent) {
        if (!boundaries.contains(*p))
            return false;
        if (!divided) {
            children.push_back(p);
            if (children.size() > capacity && capacity != 0) {
                subdivide(levelPresent);
                vector<Point*>::iterator it = children.begin();
                while (it != children.end()) {
                    if (topLeftTree->insert(*it, levelPresent));
                    else if (topRightTree->insert(*it, levelPresent));
                    else if (botLeftTree->insert(*it, levelPresent));
                    else if (botRightTree->insert(*it, levelPresent));

                    it = children.erase(it);
                }
            }

            return true;
        }
        else {
            if (topLeftTree->insert(p, levelPresent))
                return true;
```

```
                else if (topRightTree->insert(p, levelPresent))
                    return true;
                if (botLeftTree->insert(p, levelPresent))
                    return true;
                if (botRightTree->insert(p, levelPresent))
                    return true;
                return false;
            }
        }


        void drawCircles(RenderTarget& t, Circle circle, float radius, float stepLev
elX, float stepLevelY, float posNode1, float posNode2, float ySpace, float xPosPast,
 float yPosPast, NodeRectangle rect, Point* point, Point circleFind) {
            int otstup = (((posNode2 - posNode1) - 8 * radius) / 8);



            if (divided) {

                topLeftTree-
>drawCircles(t, circle, radius, stepLevelX, stepLevelY, posNode1, posNode1 + 2 * ots
tup + 2 * radius, ySpace, posNode1 + otstup + stepLevelX, circle.screenPosY + 10 * c
ircle.GetRadius() * (level + 1) + stepLevelY + ySpace, rect, point, circleFind);
                topRightTree-
>drawCircles(t, circle, radius, stepLevelX, stepLevelY, posNode1 + 2 * otstup + 2 *
radius, posNode1 + 4 * otstup + 4 * radius, ySpace, posNode1 + 3 * otstup + 2 * radi
us + stepLevelX, circle.screenPosY + 10 * circle.GetRadius() * (level + 1) + stepLev
elY + ySpace, rect, point, circleFind);
                botLeftTree-
>drawCircles(t, circle, radius, stepLevelX, stepLevelY, posNode1 + 4 * otstup + 4 *
radius, posNode1 + 6 * otstup + 6 * radius, ySpace, posNode1 + 5 * otstup + 4 * radi
us + stepLevelX, circle.screenPosY + 10 * circle.GetRadius() * (level + 1) + stepLev
elY + ySpace, rect, point, circleFind);
                botRightTree-
>drawCircles(t, circle, radius, stepLevelX, stepLevelY, posNode1 + 6 * otstup + 6 *
radius, posNode2, ySpace, posNode1 + 7 * otstup + 6 * radius + stepLevelX, circle.sc
reenPosY + 10 * circle.GetRadius() * (level + 1) + stepLevelY + ySpace, rect, point,
 circleFind);
            }
            if (topLeftTree != NULL || topRightTree != NULL || botLeftTree != NULL |
| botRightTree != NULL) {

                circle.setPosition(posNode1 + otstup + stepLevelX, circle.screenPosY
 + 10 * circle.GetRadius() * (level + 1) + stepLevelY + ySpace);


                if (!topLeftTree->divided)
                {
                    if (topLeftTree->boundaries.contains(circleFind)) {
                        circle.select = true;
                    }

                    if (topLeftTree->children.size() == 0) {
                        rect.havePoints = false;
                    }
                    else
                    {
                        rect.havePoints = true;
                        for (int i = 0; i < topLeftTree->children.size(); i++) {
                            if (point->x == topLeftTree->children[i]->x && point-
>y == topLeftTree->children[i]->y) rect.selectPoint = true;
```

```
                    }
                }
                rect.setPosition(posNode1 + otstup + stepLevelX, circle.screenPo
sY + 10 * circle.GetRadius() * (level + 2) + stepLevelY + ySpace);
                rect.drawRectangle(t);
                sf::VertexArray line(sf::Lines, 2);
                line[0].position = sf::Vector2f(posNode1 + otstup + stepLevelX +
 circle.GetRadius(), circle.screenPosY + 10 * circle.GetRadius() * (level + 2) + ste
pLevelY + ySpace);
                line[1].position = sf::Vector2f(posNode1 + otstup + stepLevelX +
 circle.GetRadius(), circle.screenPosY + 10 * circle.GetRadius() * (level + 1) + ste
pLevelY + ySpace);
                t.draw(line);

            }
            circle.draw(t);


            circle.setPosition(posNode1 + 3 * otstup + 2 * radius + stepLevelX,
circle.screenPosY + 10 * circle.GetRadius() * (level + 1) + stepLevelY + ySpace);


            if (!topRightTree->divided)
            {
                if (topRightTree->boundaries.contains(circleFind)) {
                    circle.select = true;
                }

                if (topRightTree->children.size() == 0) {
                    rect.havePoints = false;
                }
                else
                {
                    rect.havePoints = true;
                    for (int i = 0; i < topRightTree->children.size(); i++) {
                        if (point->x == topRightTree->children[i]->x && point-
>y == topRightTree->children[i]->y) rect.selectPoint = true;
                    }
                }
                rect.setPosition(posNode1 + 3 * otstup + 2 * radius + stepLevelX
, circle.screenPosY + 10 * circle.GetRadius() * (level + 2) + stepLevelY + ySpace);
                rect.drawRectangle(t);
                sf::VertexArray line(sf::Lines, 2);
                line[0].position = sf::Vector2f(posNode1 + 3 * otstup + 2 * radi
us + stepLevelX + circle.GetRadius(), circle.screenPosY + 10 * circle.GetRadius() *
(level + 2) + stepLevelY + ySpace);
                line[1].position = sf::Vector2f(posNode1 + 3 * otstup + 2 * radi
us + stepLevelX + circle.GetRadius(), circle.screenPosY + 10 * circle.GetRadius() *
(level + 1) + stepLevelY + ySpace);
                t.draw(line);

            }
            circle.draw(t);


            circle.setPosition(posNode1 + 5 * otstup + 4 * radius + stepLevelX,
circle.screenPosY + 10 * circle.GetRadius() * (level + 1) + stepLevelY + ySpace);

            if (!botLeftTree->divided)
            {
                if (botLeftTree->boundaries.contains(circleFind)) {
                    circle.select = true;
                }
```

```
                    if (botLeftTree->children.size() == 0) {
                        rect.havePoints = false;
                    }
                    else
                    {
                        rect.havePoints = true;
                        for (int i = 0; i < botLeftTree->children.size(); i++) {
                            if (point->x == botLeftTree->children[i]->x && point-
>y == botLeftTree->children[i]->y) rect.selectPoint = true;
                        }
                    }
                    rect.setPosition(posNode1 + 5 * otstup + 4 * radius + stepLevelX
, circle.screenPosY + 10 * circle.GetRadius() * (level + 2) + stepLevelY + ySpace);
                    rect.drawRectangle(t);
                    sf::VertexArray line(sf::Lines, 2);
                    line[0].position = sf::Vector2f(posNode1 + 5 * otstup + 4 * radi
us + stepLevelX + circle.GetRadius(), circle.screenPosY + 10 * circle.GetRadius() *
(level + 2) + stepLevelY + ySpace);
                    line[1].position = sf::Vector2f(posNode1 + 5 * otstup + 4 * radi
us + stepLevelX + circle.GetRadius(), circle.screenPosY + 10 * circle.GetRadius() *
(level + 1) + stepLevelY + ySpace);
                    t.draw(line);

                }
                circle.draw(t);

                circle.setPosition(posNode1 + 7 * otstup + 6 * radius + stepLevelX,
circle.screenPosY + 10 * circle.GetRadius() * (level + 1) + stepLevelY + ySpace);

                if (!botRightTree->divided)
                {
                    if (botRightTree->boundaries.contains(circleFind)) {
                        circle.select = true;
                    }
                    if (botRightTree->children.size() == 0) {
                        rect.havePoints = false;
                    }
                    else
                    {
                        rect.havePoints = true;
                        for (int i = 0; i < botRightTree->children.size(); i++) {
                            if (point->x == botRightTree->children[i]->x && point-
>y == botRightTree->children[i]->y) rect.selectPoint = true;
                        }
                    }
                    rect.setPosition(posNode1 + 7 * otstup + 6 * radius + stepLevelX
, circle.screenPosY + 10 * circle.GetRadius() * (level + 2) + stepLevelY + ySpace);

                    rect.drawRectangle(t);
                    sf::VertexArray line(sf::Lines, 2);
                    line[0].position = sf::Vector2f(posNode1 + 7 * otstup + 6 * radi
us + stepLevelX + circle.GetRadius(), circle.screenPosY + 10 * circle.GetRadius() *
(level + 2) + stepLevelY + ySpace);
                    line[1].position = sf::Vector2f(posNode1 + 7 * otstup + 6 * radi
us + stepLevelX + circle.GetRadius(), circle.screenPosY + 10 * circle.GetRadius() *
(level + 1) + stepLevelY + ySpace);
                    t.draw(line);
                    //circle.draw(t);
                }
                circle.draw(t);
                sf::VertexArray lines(sf::Lines, 8);
```

```
                lines[0].position = sf::Vector2f(posNode1 + otstup + stepLevelX + ci
rcle.GetRadius(), circle.screenPosY + 10 * circle.GetRadius() * (level + 1) + stepLe
velY + ySpace + circle.GetRadius());
                lines[1].position = sf::Vector2f(xPosPast + circle.GetRadius(), yPos
Past + circle.GetRadius());
                lines[2].position = sf::Vector2f(posNode1 + 3 * otstup + 2 * radius
+ stepLevelX + circle.GetRadius(), circle.screenPosY + 10 * circle.GetRadius() * (le
vel + 1) + stepLevelY + ySpace + circle.GetRadius());
                lines[3].position = sf::Vector2f(xPosPast + circle.GetRadius(), yPos
Past + circle.GetRadius());
                lines[4].position = sf::Vector2f(posNode1 + 5 * otstup + 4 * radius
+ stepLevelX + circle.GetRadius(), circle.screenPosY + 10 * circle.GetRadius() * (le
vel + 1) + stepLevelY + ySpace + circle.GetRadius());
                lines[5].position = sf::Vector2f(xPosPast + circle.GetRadius(), yPos
Past + circle.GetRadius());
                lines[6].position = sf::Vector2f(posNode1 + 7 * otstup + 6 * radius
+ stepLevelX + circle.GetRadius(), circle.screenPosY + 10 * circle.GetRadius() * (le
vel + 1) + stepLevelY + ySpace + circle.GetRadius());
                lines[7].position = sf::Vector2f(xPosPast + circle.GetRadius(), yPos
Past + circle.GetRadius());

                t.draw(lines);
            }
        }

        void draw(RenderTarget& t) {
            if (divided) {

                static Vertex vertices[4];
                vertices[0] = Vertex(Vector2f(boundaries.x, boundaries.y - boundarie
s.h), Color::White);
                vertices[1] = Vertex(Vector2f(boundaries.x, boundaries.y + boundarie
s.h), Color::White);
                vertices[2] = Vertex(Vector2f(boundaries.x - boundaries.w, boundarie
s.y), Color::White);
                vertices[3] = Vertex(Vector2f(boundaries.x + boundaries.w, boundarie
s.y), Color::White);
                t.draw(vertices, 4, Lines);
                topLeftTree->draw(t);
                topRightTree->draw(t);
                botLeftTree->draw(t);
                botRightTree->draw(t);

            }
        }

        void query(cursorRectangle& area, Point* &found, Point*& foundSave) {
            if (!area.intersects(boundaries))
                return;
            if (divided) {
                topLeftTree->query(area, found, foundSave);
                topRightTree->query(area, found, foundSave);
                botLeftTree->query(area, found,foundSave);
                botRightTree->query(area, found, foundSave);
            }
            else {
                for (size_t i = 0; i < children.size(); i++) {
                    if (area.contains(*children[i])) {
                        found = children[i];
                        foundSave->x = children[i]->x;
                        foundSave->y = children[i]->y;
                    }
```

```
                }
            }
        }

        void deletePoint(Point*& found, bool& check) {

            if (!boundaries.contains(*found)) {
                return;
            }


            if (divided) {
                topLeftTree->deletePoint(found, check);
                topRightTree->deletePoint(found, check);
                botLeftTree->deletePoint(found, check );
                botRightTree->deletePoint(found, check);
            }
            else {
                vector<Point*>::iterator it = children.begin();
                for (size_t i = 0; i < children.size(); i++) {
                    if ((found->x == children[i]->x)and (found->y == children[i]-
>y)) {

                        children.erase(it);
                        check = true;

                        return;
                    }
                    it++;
                }
            }
            int count = 0;
            if ((countChildren(count) <= 4) and (divided) and (countChildren(count)>
0) and (check)) {
                children.insert(children.begin(), topLeftTree-
>children.begin(), topLeftTree->children.end());
                children.insert(children.begin(), topRightTree-
>children.begin(), topRightTree->children.end());
                children.insert(children.begin(), botRightTree-
>children.begin(), botRightTree->children.end());
                children.insert(children.begin(), botLeftTree-
>children.begin(), botLeftTree->children.end());
                topLeftTree = NULL;
                topRightTree = NULL;
                botLeftTree = NULL;
                botRightTree = NULL;

                divided = false;
                check = true;
                return;
            }
            check = false;
        }

        void findMaxLevel(int& levelMax) {
            if (divided) {
                topLeftTree->findMaxLevel(levelMax);
                topRightTree->findMaxLevel(levelMax);
                botLeftTree->findMaxLevel(levelMax);
                botRightTree->findMaxLevel(levelMax);
            }
            if (levelMax < level) {
                levelMax = level;
            }
```

```
        }

};

class Settings {
    RectangleShape rectangleTextSettings;
    Text text;
    float xSize;
    float ySize;
public:
    Settings(RectangleShape rectangleTextSetting, Text text) {
        this->rectangleTextSettings = rectangleTextSettings;
        rectangleTextSettings.setFillColor(sf::Color{ SCREEN_COLOR });
        rectangleTextSettings.setOutlineColor(sf::Color::White);
        rectangleTextSettings.setOutlineThickness(1);
        this->text = text;

        text.setPosition(SCREEN_SPACE, SCREEN_SPACE / 4); // Помещаем текст в левый
верхний угол
    };
    void setTextFontSize(float size, Font font) {
        text.setFont(font);
        text.setCharacterSize(size);
    }
    void  setSize(float xSize, float ySize) {
        rectangleTextSettings.setSize(sf::Vector2f(xSize, ySize));
        this->xSize = xSize;
        this->ySize = ySize;
     }
   void setPosition(float xPos, float yPos) {
       rectangleTextSettings.setPosition(xPos-xSize, yPos-ySize);
   }

   void setTextPosition(float xPos, float yPos) {
       text.setPosition(xPos - xSize, yPos - ySize);
   }

   void setString(string s) {
       text.setString(s);
   }
   void draw(RenderTarget& t) {
       t.draw(rectangleTextSettings);
       //t.draw(text);
   }

   void hide(bool set) {
       if (set == false) rectangleTextSettings.setSize(sf::Vector2f(0, 0));
   }

};

    int main()
    {

        setlocale(LC_ALL, "Russian");

        float stepLevelX=0;                 //сдвиг при стрелочках по x
        float stepLevelY = 0;               //сдвиг при стрелочках по y
        int levelPresent = 0;               //глубина
        bool cursorInArea = false;          //нахождение курсора в области
        float cursorSize = MIN_SCROLL;      //изначальный размер курсора
        float radius = RADIUS;              //изначальный радиус
```

```
        int xPosCircle = SCREEN_SPACE + SCREEN_TREE_W + 200;        //позиция для на
        int yPosCircle = 1.5 * SCREEN_SPACE + 300;                  //позиция для на
        float scale = 1;                    //масштаб
        bool set=false;
        float rectangleTextParamsPosX = SCREEN_SPACE + SCREEN_TREE_W - 20;  //позици

        float treeNodeSizeX = SCREEN_W - SCREEN_SPACE - SCREEN_TREE_W - 20;     //ра
змер экрана дерева по x
        float treeNodePosX = SCREEN_SPACE + SCREEN_TREE_W - 20;                 //по
зиция экрана дерева по x
        float treeNodeSizeY = SCREEN_TREE_H - 1.5 * SCREEN_SPACE - 110;         //ра
змер экрана дерева по y
        float treeNodePosY = 1.5 * SCREEN_SPACE + 110;                          //по
зиция экрана дерева по y

        bool xPressed = false;

        RenderWindow window(VideoMode(SCREEN_W, SCREEN_H), "Quad Tree ");       //со
здание окна

        Quad tree(Rectangle((SCREEN_TREE_W+ SCREEN_SPACE) / 2, (SCREEN_TREE_H+ SCREE
N_SPACE)/2 , (SCREEN_TREE_W- SCREEN_SPACE) / 2, (SCREEN_TREE_H- SCREEN_SPACE) / 2),
CAPACITY, 0);
            //создание корня дерева



                                                                /*СОЗДАНИЕ ТОЧЕК ДЛЯ ДЕР
ЕВА*/


        Point* po;                          //обычная точка
        vector<Point*> points;              //вектор точек
        Point* found = new Point();         //найденная точка
        Point* foundSave = new Point(-1, -
1); //точка, которую необходимо отдельно хранить(выбранная точка)
        Point circleFind(-1, -
1);             //точка, по которой находится нужный узел


        CircleShape shape;              //точка на экране
        shape.setRadius(RADIUS);
        shape.setOrigin(1, 1);
        cursorRectangle cursor(200,200,4,4);

        Circle CircleSTree(radius, scale, TREE_SIZE_X, TREE_SIZE_Y, treeNodePosX-
(abs(TREE_SIZE_X- treeNodeSizeX)/2), 1.5 * SCREEN_SPACE + 110);

        Text textPoint;
        Text textScroll;
        Text textParams;
        Text textPresentParams;
        Text textInformation;
        Text textSettings;
        Text textCursor;

        sf::Font font;
        if (!font.loadFromFile("times.ttf")) {
            cout << "MOOO";
        }

        textCursor.setFont(font);
        textCursor.setCharacterSize(TEXT_SIZE_BIG);
```

```
        textCursor.setPosition(rectangleTextParamsPosX + 450 + SCREEN_SPACE+30, SCRE
EN_SPACE+10);

        textSettings.setCharacterSize(TEXT_SIZE_BIG);
        textSettings.setFont(font);
        textSettings.setString(L"Нажмите X, чтобы открыть управление");
        sf::FloatRect textRect = textSettings.getLocalBounds();
        textSettings.setPosition(SCREEN_W - textRect.width- SCREEN_SPACE, SCREEN_H -
 SCREEN_SPACE - textRect.height);


        textPoint.setCharacterSize(TEXT_SIZE_BIG);
        textPoint.setFont(font);
        textPoint.setPosition(SCREEN_SPACE, SCREEN_SPACE / 4); // Помещаем текст в л
евый верхний угол

        textScroll.setCharacterSize(TEXT_SIZE_BIG);
        textScroll.setFont(font);
        textScroll.setPosition(SCREEN_SPACE, SCREEN_TREE_H+10); // Помещаем текст в
левый верхний угол

        textParams.setCharacterSize(TEXT_SIZE_STANDART);
        textParams.setFont(font);
        textParams.setPosition(SCREEN_SPACE + SCREEN_TREE_W , SCREEN_SPACE+5); // По
мещаем текст в левый верхний угол
        textParams.setString(L"Максимальная глубина: " + to_string(MAX_LEVEL) + L"\n
Максимальное кол-во точек в квадранте: "
            + to_string(CAPACITY) + L"\nМаксимальный размер курсора: " + to_string(M
AX_SCROLL)
            + L"\nМинимальный размер курсора: " + to_string(MIN_SCROLL));

        textPresentParams.setCharacterSize(TEXT_SIZE_BIG);
        textPresentParams.setFont(font);
        textPresentParams.setPosition(SCREEN_SPACE + 300, SCREEN_TREE_H + 10); // По
мещаем текст в левый верхний угол




                                                                    /*СОЗДАН
ИЕ ПРЯМОУГОЛЬНКОВ*/

        sf::RectangleShape tectangleTextSettings;

        sf::RectangleShape rectangleTextParams(sf::Vector2f(450, 110));         //дл
я текста
        rectangleTextParams.setFillColor(sf::Color{ SCREEN_COLOR });
        rectangleTextParams.setPosition(rectangleTextParamsPosX, SCREEN_SPACE);
        rectangleTextParams.setOutlineColor(sf::Color::White);
        rectangleTextParams.setOutlineThickness(1);

        sf::RectangleShape borderTree(sf::Vector2f(SCREEN_TREE_W - SCREEN_SPACE, SCR
EEN_TREE_H - SCREEN_SPACE));
        borderTree.setFillColor(sf::Color{ SCREEN_COLOR });                      //
область работы с экраном точек
        borderTree.setOutlineColor(Color::White);
        borderTree.setOutlineThickness(1);
        borderTree.setPosition(SCREEN_SPACE, SCREEN_SPACE);

        sf::RectangleShape treeNode(sf::Vector2f(treeNodeSizeX, treeNodeSizeY));
        treeNode.setFillColor(sf::Color{ SCREEN_COLOR });                       //об
ласть работы с экраном дерева
        treeNode.setOutlineColor(Color::White);
```

```
        treeNode.setOutlineThickness(1);
        treeNode.setPosition(treeNodePosX, treeNodePosY);


        sf::RectangleShape hideCircles(sf::Vector2f(treeNodeSizeX +2, treeNodeSizeY
+2));
        hideCircles.setFillColor(sf::Color::Transparent);                       //ма
ска экрана с деревом
        hideCircles.setOutlineColor(Color::Black);
        hideCircles.setOutlineThickness(1200);
        hideCircles.setPosition(treeNodePosX -1, treeNodePosY -1);

        sf::RectangleShape rectangleCursorPos(sf::Vector2f(350, 50));         //для
текста
        rectangleCursorPos.setFillColor(sf::Color{ SCREEN_COLOR });
        rectangleCursorPos.setPosition(rectangleTextParamsPosX+450+ SCREEN_SPACE, SC
REEN_SPACE);
        rectangleCursorPos.setOutlineColor(sf::Color::White);
        rectangleCursorPos.setOutlineThickness(1);

        Settings* settings = new Settings(tectangleTextSettings, textSettings);


                                                                /*ОБРАБОТКА ДЕЙС
ТВИЙ ПОЛЬЗОВАТЕЛЯ*/


        while (window.isOpen()) {
            Event e;
            while (window.pollEvent(e)) {
                if (e.type == Event::Closed)        //кнопка закрытия
                    window.close();

                                                                /*ОБРАБОТКА ЛЕВОЙ
 КНОПКИ МЫШИ*/


                else if (e.type == Event::MouseButtonPressed && e.mouseButton.button
 == Mouse::Left) {  //левая кнопка мыши
                    circleFind.x = -1;
                    circleFind.y = -1;
                    foundSave->x = -1;
                    foundSave->y = -1;
                    po = new Point(Mouse::getPosition(window).x, Mouse::getPosition(
window).y); //точка клика
                    if (po->x > SCREEN_SPACE && po->y > SCREEN_SPACE && po-
>x < SCREEN_TREE_W && po->y < SCREEN_TREE_H) {
                        points.push_back(po);   //если точка внцтри экрана, то запис
ываем в массив
                        string s = "Добавлена точка с координатами: " + to_string(po
->x) + ":" + to_string(po->y);
                        cout << s << endl;
                        textPoint.setString(L"Добавлена точка с координатами: " + to
_string(po->x) + ":" + to_string(po->y));
                    }
                    tree.insert(po, levelPresent);      //вставка точки в дерево

                    for (size_t i = 0; i < points.size(); i++)      //убираем подсве
тку точек
                        points[i]->highlighted = false;

                }
```

```
                                                                /*ОБРАБОТКА ПРАВОЙ
КНОПКИ МЫШИ*/


                else if (e.type == Event::MouseButtonPressed && e.mouseButton.button
 == Mouse::Right) { //правая кнопка мыши
                    circleFind.x = -1;
                    circleFind.y = -1;
                    for (size_t i = 0; i < points.size(); i++)
                        points[i]->highlighted = false;
                    foundSave->x = -1;
                    foundSave->y = -1;
                    found = new Point();
                    tree.query(cursor, found, foundSave); //находим точку, которая н
аходится в области
                    if (foundSave-
>x >= 0) {    //если точка найдена, подсвечиваем ее
                        found->highlighted = true;
                        string s = "Выделена точка с координатами: " + to_string(fou
ndSave->x) + ":" + to_string(foundSave->y);
                        cout << s << endl;
                        textPoint.setString(L"Выделена точка с координатами: " + to_
string(foundSave->x) + ":" + to_string(foundSave->y));
                    }

                }



/*ОБРАБОТКА ДВИЖЕНИЯ МЫШИ*/



                else if (e.type == Event::MouseMoved) {     //движение мыши
                    cursor.x = Mouse::getPosition(window).x + 2;  //двигаем курсор к
вадрат
                    cursor.y = Mouse::getPosition(window).y;
                    if (Mouse::getPosition(window).x > SCREEN_SPACE && Mouse::getPos
ition(window).x < SCREEN_TREE_W  && //если курсор в экране
                        Mouse::getPosition(window).y> SCREEN_SPACE && Mouse::getPosi
tion(window).y < SCREEN_TREE_H ) {
                        window.setMouseCursorVisible(false);
                        cursorInArea = true;        //то виден квадратный курсор
                        cursor.h = cursorSize;
                        cursor.w = cursorSize;
                    }
                    else if (Mouse::getPosition(window).x > SCREEN_SPACE + SCREEN_TR
EE_W - 20 && Mouse::getPosition(window).x < SCREEN_W - SCREEN_SPACE + 20 && //если к
урсор в экране
                        Mouse::getPosition(window).y> 1.5 * SCREEN_SPACE + 110 && Mo
use::getPosition(window).y < SCREEN_TREE_H )
                    {
                        window.setMouseCursorVisible(false);
                        cursorInArea = true;
                        cursor.h = MIN_SCROLL * 2;
                        cursor.w = MIN_SCROLL * 2;
                    }
                    else {  //иначе обычный
                        window.setMouseCursorVisible(true);
                        cursorInArea = false;
                    }
```

```
            }


                                                                                /*О
БРАБОТКА КОЛЕСИКА МЫШИ*/


            else if (e.type == Event::MouseWheelMoved) {    //колесико мыши
                if (Mouse::getPosition(window).x > SCREEN_SPACE && Mouse::getPos
ition(window).x < SCREEN_TREE_W && //если курсор в экране
                    Mouse::getPosition(window).y> SCREEN_SPACE && Mouse::getPosi
tion(window).y < SCREEN_TREE_H) {
                    if (e.mouseWheel.delta < 0) {   //если вниз, то уменьшаем ра
змер курсора
                        if (cursorSize > MIN_SCROLL) {
                            cursorSize -= 2;

                        }
                    }
                    else if (e.mouseWheel.delta > 0)
                        if (cursorSize < MAX_SCROLL) {
                            cursorSize += 2;


                        }
                    cursor.h = cursorSize;
                    cursor.w = cursorSize;
                    string s = "Размер курсора: " + to_string(cursor.h);
                    cout << s << endl;
                }
                else if (Mouse::getPosition(window).x > SCREEN_SPACE + SCREEN_TR
EE_W - 20 && Mouse::getPosition(window).x < SCREEN_W - SCREEN_SPACE + 20 && //если к
урсор в экране
                    Mouse::getPosition(window).y> 1.5 * SCREEN_SPACE + 110 && Mo
use::getPosition(window).y < SCREEN_TREE_H)
                {
                    if (e.mouseWheel.delta < 0) {   //если вниз, то уменьшаем ра
змер курсора
                        if (scale >1) {
                            if (scale <=2) {
                                scale -= 0.5;
                            } else{ scale -= SCALE_STEP; }

                            CircleSTree.scale = scale;

                            CircleSTree.setRadius(radius);
                            CircleSTree.setPosition(xPosCircle- radius * scale /
 2, yPosCircle - radius * scale / 2);

                        }
                    }
                    else if (e.mouseWheel.delta > 0)
                        if (scale < SCALE_MAX) {
                            if (scale <= 2) {
                                scale += 0.5;
                            }
                            else { scale += SCALE_STEP; }
                            CircleSTree.scale = scale;

                            CircleSTree.setRadius(radius);
```

```
                                CircleSTree.setPosition(xPosCircle- radius * scale/2
, yPosCircle - radius * scale / 2);
                            }
                        }

                    //textScroll.setString(L"Размер курсора: " + to_string(area.h));
                }



                                                                        /*ОБРАБ
ОТКА НАЖАТИЯ LSHIFT*/



                else if (Keyboard::isKeyPressed(sf::Keyboard::LShift)) {    //нажати
е левого шифта
                circleFind.x = -1;
                circleFind.y = -1;

                if (foundSave->x != -1) {       //если точка выделена
                        bool check = false;
                        tree.deletePoint(foundSave, check); //удаляем точку
                        int i=0;
                        while(!(points[i]->x == foundSave->x && points[i]-
>y == foundSave->y)){ //удаляем точку из массива точек
                            i++;
                        }
                        points.erase(points.begin() + i);
                        string s = "Удалена точка с координатами: " + to_string(foun
dSave->x) + ":" + to_string(foundSave->y);
                        cout << s<<endl;
                        textPoint.setString(L"Удалена точка с координатами: " + to_s
tring(foundSave->x) + ":" + to_string(foundSave->y));
                        foundSave->x = -1;
                        foundSave->y = -1;
                    }

                }


                else if (Keyboard::isKeyPressed(sf::Keyboard::Z)) {    //нажатие лев
ого шифта

                circleFind.x = Mouse::getPosition(window).x;
                circleFind.y = Mouse::getPosition(window).y;

                }


                else if (Keyboard::isKeyPressed(sf::Keyboard::X)) {
                float xSize = 350;
                float ySize = 200;
                if (!xPressed) {
                        if (set == true) set = false;
                        else set = true;
                        xPressed = true;
                    }
                    textSettings.setPosition(SCREEN_W - SCREEN_SPACE- xSize+10, SCRE
EN_H - SCREEN_SPACE- ySize+5);


                settings->setSize(xSize, ySize);
```

```
                settings-
>setPosition(SCREEN_W - SCREEN_SPACE, SCREEN_H - SCREEN_SPACE);


                if (set == true) {
                    textSettings.setCharacterSize(TEXT_SIZE_STANDART);
                    textSettings.setString(L"Управление:\nЛКМ - добавитть точку\nПКМ
 - выделить точку\nLShist - удалить выделенную точку\nZ - выделить узел\nX - скрыть
управление\nКолесико мыши - изменение масштаба\n   экрана/курсора\nСтрелочки - движе
ние дерева");

                }
                else {
                    textSettings.setCharacterSize(TEXT_SIZE_BIG);
                    textSettings.setString(L"Нажмите X, чтобы открыть управление");
                    textSettings.setPosition(SCREEN_W - textRect.width - SCREEN_SPAC
E, SCREEN_H - SCREEN_SPACE - textRect.height);
                }

                settings->hide(set);


                }
                if (e.type == sf::Event::KeyReleased)
                {
                    if (e.key.code == sf::Keyboard::X)
                    {
                        xPressed = false;

                    }
                }

                                                                                /*ОБРАБО
ТКА НАЖАТИЯ СТРЕЛОЧЕК*/


                else if (e.type == sf::Event::KeyPressed) {
                    if (e.key.code == Keyboard::Left) {
                        xPosCircle += STEP * scale;
                        stepLevelX += STEP * scale;
                        CircleSTree.setPosition(xPosCircle, yPosCircle);
                    }
                    if (e.key.code == Keyboard::Right) {
                        xPosCircle -= STEP*scale;
                        stepLevelX -= STEP;
                        CircleSTree.setPosition(xPosCircle, yPosCircle);
                    }
                    if (e.key.code == Keyboard::Up) {
                        yPosCircle -= STEP * scale;
                        stepLevelY -= STEP * scale;
                        CircleSTree.setPosition(xPosCircle, yPosCircle);
                    }
                    if (e.key.code == Keyboard::Down) {

                        yPosCircle += STEP * scale;
                        stepLevelY += STEP * scale;
                        CircleSTree.setPosition(xPosCircle, yPosCircle);
                    }
                }

                sf::Vector2i mousePos = sf::Mouse::getPosition(window);
```

```
                textCursor.setString(L"Позиция курсора: (" + std::to_string(mousePos
.x) + ", " + std::to_string(mousePos.y) + ")");
            }



            textScroll.setString(L"Размер курсора: " + to_string(cursor.h));
            levelPresent = 0;
            tree.findMaxLevel(levelPresent);        //находим максимальную глубину в
 дереве
            textPresentParams.setString(L"Глубина: " + to_string(levelPresent) + L"\
nКол-во точек(всего): " + to_string(points.size()));

            window.clear(sf::Color::Black);     //очищаем экран



       /*ОТРИСОВКА*/



            window.draw(treeNode);
            CircleSTree.setPosition(CircleSTree.screenPosX + CircleSTree.screenSizeX
 / 2+stepLevelX, CircleSTree.screenPosY+50+ + stepLevelY);

            tree.findMaxLevel(levelPresent);

            NodeRectangle rect(radius*2 * scale);
            rect.setOutlineThickness(scale);
            if (levelPresent == 0) {
                rect.setPosition(CircleSTree.screenPosX + CircleSTree.screenSizeX /
2 + stepLevelX, CircleSTree.screenPosY + 50 + +stepLevelY+4*CircleSTree.GetRadius())
;
                if (points.size() > 0) {
                    rect.havePoints = true;
                    for (int i = 0; i < points.size(); i++) {
                        if (foundSave->x == points[i]->x && foundSave-
>y == points[i]->y) rect.selectPoint = true;
                    }
                }

                rect.drawRectangle(window);
                if (levelPresent == 0) {
                    if (circleFind.x< SCREEN_SPACE + SCREEN_TREE_W && circleFind.x >
 SCREEN_SPACE && circleFind.y > SCREEN_SPACE && circleFind.y < SCREEN_TREE_H) {
                        CircleSTree.select = true;
                    }
                }
                sf::VertexArray line(sf::Lines, 2);
                line[0].position = sf::Vector2f(CircleSTree.screenPosX + CircleSTree
.screenSizeX / 2 + stepLevelX+ CircleSTree.GetRadius(), CircleSTree.screenPosY + 50
+ +stepLevelY);
                line[1].position = sf::Vector2f(CircleSTree.screenPosX + CircleSTree
.screenSizeX / 2 + stepLevelX+ CircleSTree.GetRadius(), CircleSTree.screenPosY + 50
+ +stepLevelY + 4 * CircleSTree.GetRadius());
                window.draw(line);

            }
            CircleSTree.draw(window);
            tree.drawCircles(window, CircleSTree, CircleSTree.radius, stepLevelX, st
epLevelY, CircleSTree.screenPosX - CircleSTree.screenPosX * (scale - 1), (CircleSTre
e.screenPosX + TREE_SIZE_X) * scale, CircleSTree.screenPosY / 2, CircleSTree.screenP
```

```
osX + CircleSTree.screenSizeX / 2 + stepLevelX, CircleSTree.screenPosY + 50 + +stepL
evelY, rect, foundSave, circleFind);

            window.draw(hideCircles);
            settings->draw(window);
            window.draw(textSettings);
            window.draw(borderTree);            //экран
            for (Point* p : points) {           //точки
                shape.setPosition(p->x, p->y);
                shape.setFillColor(p->highlighted ? Color::Green : Color::White);
                window.draw(shape);
            }

            tree.draw(window);                  //дерево(границы)

            cursor.draw(window, cursorInArea);  //отрисовка курсора квадратного
            window.draw(textPoint);
            window.draw(textScroll);
            window.draw(rectangleTextParams);
            window.draw(textParams);
            window.draw(textPresentParams);
            window.draw(rectangleCursorPos);
            window.draw(textCursor);

            window.display();
        }

        return 0;
    }
```

Описание кода визуализации:

Первым делом объявляются все константы, связанные с цветами, с размерами окна и всех экранов, коэффициентами прокрутки и масштабирования, максимальным уровнем глубины и максимальное кол-во точек в квадранте (до его разделения)

Описание класса Point

Класс необходим для создания точек на экране, он содержит координаты по x и y. Также есть bool переменная, которая отвечает за выделение точки.

Описание класса Rectangle

Класс отвечает за квадранты.

Метод contains отвечает за наличие точки внутри квадранта

Описание класса NodeRectangle

Класс необходим для визуализации прямоугольника, отвечающего за точки на экране. (Вставляется после узла в дереве)

Описание класса cursorRectangle

Класс необходим для работы с квадратным курсором, отвечающего за точки на экране. (Вставляется после узла в дереве)

Метод contains отвечает за наличие точки внутри квадранта

Метод draw отвечает за отрисовку прямоугольного курсора. Также есть проверка на прозрачность курсора. Прямоугольный курсор становится прозрачным тогда, когда он выходит за границы 2 экранов (он становится обычным).

Метод intersects необходим, чтобы найти, в каком квадранте находится курсор (чтобы в будущем найти точку внутри квадратного курсора)

Описание класса Quad

Класс необходим для работы с квадратичным деревом. Это ключевой класс в программе.

Экземпляр класса содержит ссылки на 4 потомков (деление экрана на 4 части). Если потомков нет, то они указывают на Null. Также есть

прямоугольник с размерами квадранта.

Метод insert нужен для добавления точки. Сначала находим, в какой квадрант нужно добавить точку. При нахождении добавляем ее в вектор точек. Если размер вектора точек больше 4 (максимального числа точек в квадранте), то выполняется метод деления экрана, после этого вектор точек распределяется по потомкам

Метод subdivide нужен для деления квадранта на подквадранты.

Метод countChildren нужен для подсчета всех детей у квадранта (сумма всех точек у потомков квадранта). Метод нужен для удаления точек

Метод drawCircles нужен для отрисовки узлов на экране визуализации дерева. Сначала доходим до конца дерева. Потом определяем, выделен ли данный узел и имеет ли узел точки. Если узел не делится, то выводится прямоугольник точек. После чего определяется положение узла и рисуется круг (узел). После этого круг соединяется со своим родителем линией.

Итог: отрисовка осуществляется с отрисовки нижних узлов. Отрисовка каждых 4 узлов осуществляется с помощью отступов. Отступы изображены на Рисунок 1. По рисунку можно найти размер отступа = (Размер области – 8*радиус окружности (4 круга))/8 (8 областей отступа X)

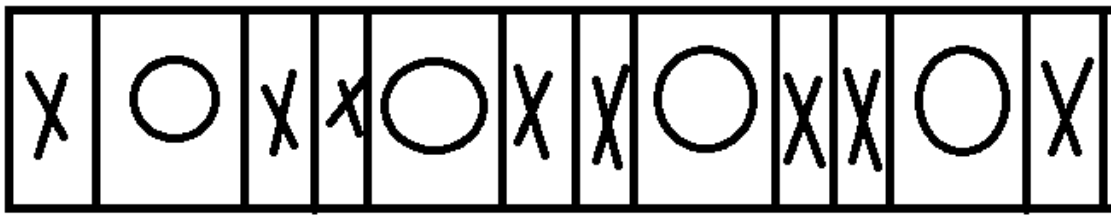

Рисунок 1 – нахождение отступа между узлами

Метод draw нужен для отрисовки деления экрана

Метод query нужен для нахождения точек внутри области (квадратного курсора). Сначала находим, где находится область (в каком квадранте). Потом смотрим все точки квадранта и если хотя бы одна из них находится в области, то она запоминается в соответствующей переменной. (запоминается последняя подходящая точка в векторе)

Метод deletePoint нужен для удаления выбранной точки. Сначала находим нужную точку, удаляем ее. Потом проверяем, сколько точек содержит родитель узла, в котом удалена точка. Если родитель имеет менее 5 точек (сумма всех точек в потомках). То потомки удаляются (деление экрана убирается, так как он содержит менее 5 точек)

Метод findMaxLevel нужен для нахождения текущей глубины дерева

Описание класса Settings

Класс необходим для работы с окном настроек. Появление и скрытие настроек. Установка параметров текста и размера прямоугольника, на котором находится текст (окно настроек)

Описание main

Инициализация всех фигур, переменных. Создание окна

Обработка действий пользователя и отрисовка фигур

3. Руководство оператора

Если курсор входит в экраны визуализации, он меняет форму на квадратную. Если курсор в экране визуализации деления экрана, то с помощью колесика мыши можно менять размеры курсора.

Действия внутри экрана визуализации деления экрана:

- На лкм создается точка
- На пкм можно выделить точку (точка выделяется если находится внутри области курсора). Выделяется последняя добавленная точка (если в области курсора находится больше 1 точки)
- На LShift можно удалить выделенную точку. Если удаляется точка и внутри родительского квадранта становится точек меньше, чем 5, деление родительского квадранта убирается. Это правило не работает только в том случае, если достигнута максимальная глубина (при максимальной глубине можно делать больше 4 точек без разделения экрана).
- На Z можно выбрать квадрант (узел в экране визуализации дерева, меняет цвет на зеленый). Выделить можно только последние узлы (после них нет других узлов)

Экран визуализации дерева:

- На стрелочки можно двигать экран визуализации дерева.
- С помощью колесика мыши, можно менять масштаб дерева
- Если в квадранте есть точки, то соответствующий прямоугольник точек имеет черный цвет, в ином случае белый цвет. Если точка в квадранте выделена, то прямоугольник имеет зеленый цвет
- Если выделяется квадрант, то соответствующий ему узел имеет зеленый цвет.

- Когда экран визуализации деления делится на квадранты, на экране визуализации дерева создаются 4 новых узла, которые соединены с соответствующим родительским узлом
- Если после удаления точки деление экрана убирается, то и все 4 узла удаляются

На X можно открыть и закрыть меню управления
Также в окне выводится информация:

- Текущее положение курсора
- Текущая глубина дерева
- Кол-во добавленных точек
- Последнее действие с удалением, добавлением, выделением точек
- Настройки программы
- Размер курсора

4. Примеры работы программы

Примеры работы программы представлены на Рисунках 2, 3, 4, 5

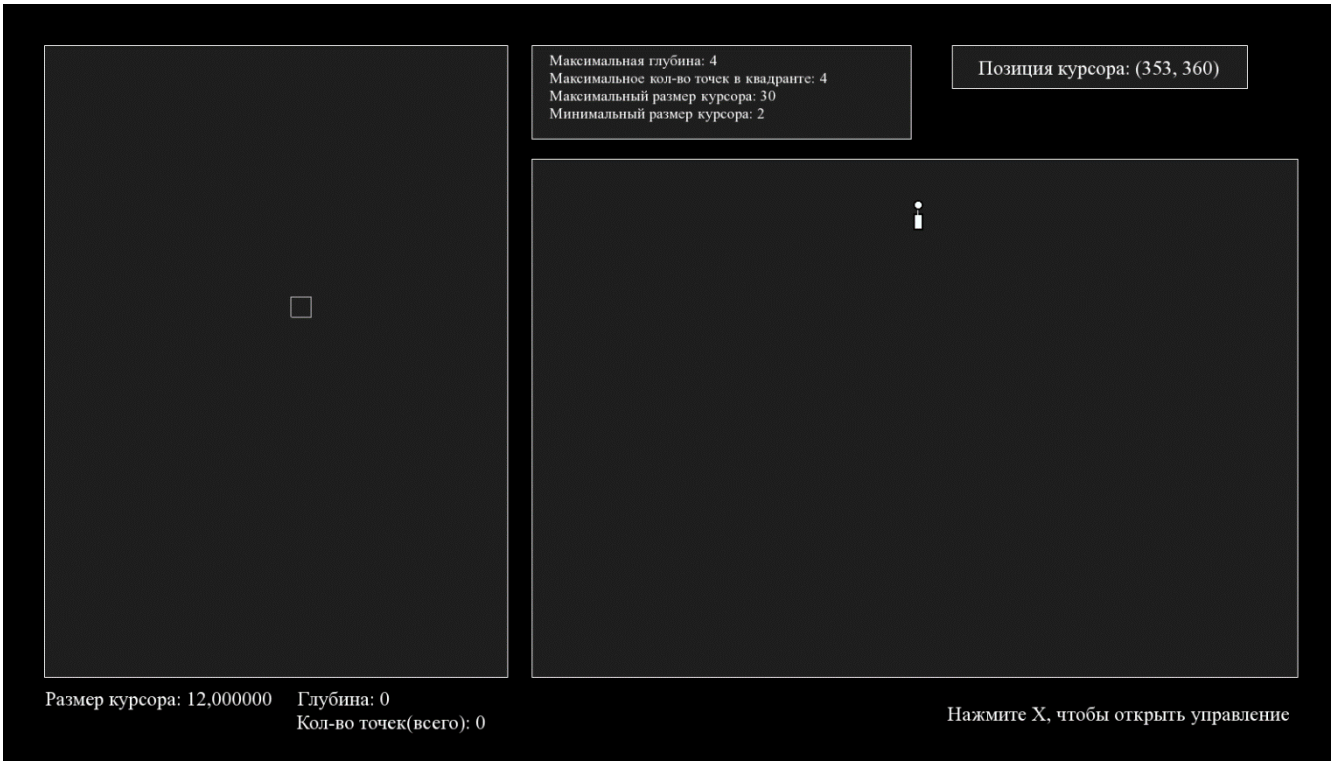

Рисунок 2 – Вид окна программы

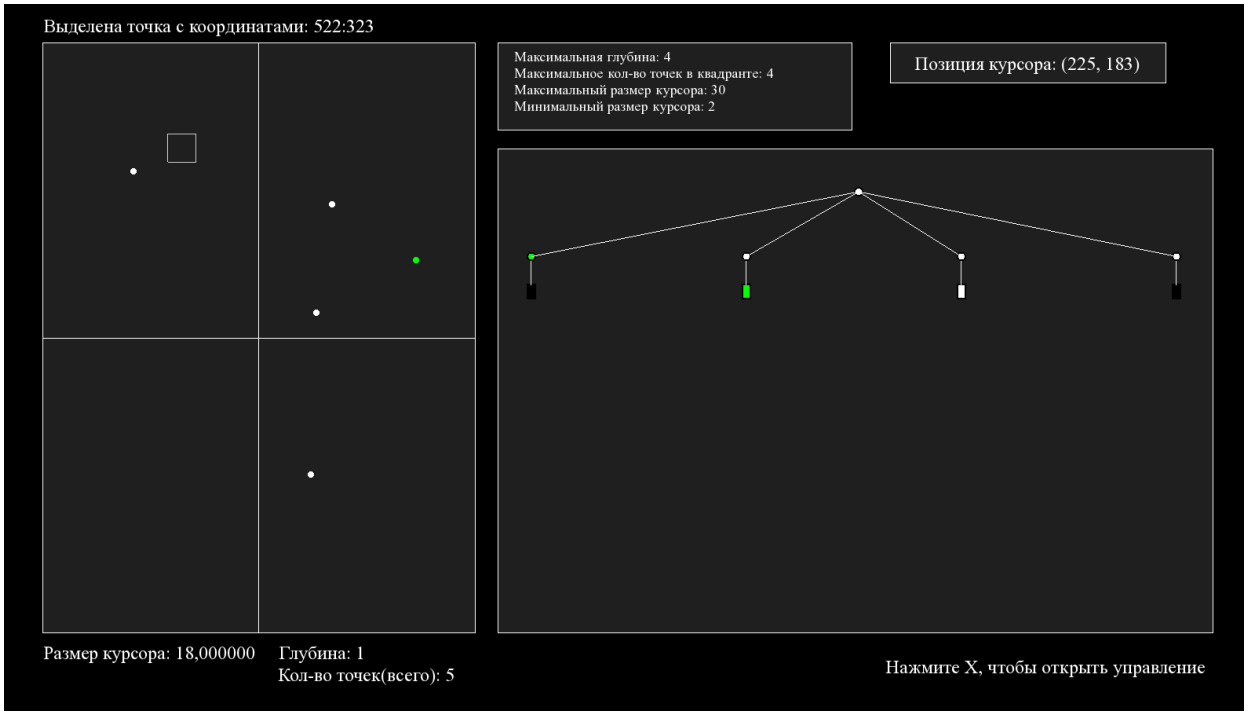

Рисунок 2 – реализация действий программы

Из рисунка 2 видно, что в одном из квадрантов нет точек и его соответствующий прямоугольник имеет белый цвет. Прямоугольник с выделенной точкой имеет зеленый цвет. Выделенный квадрант имеет узел зелёного цвета. Из рисунка таже видно, что позиция курсора выводится корректно и выводится последнее действие с точкой (выделение точки)

Также видно, что размер курсора изменился

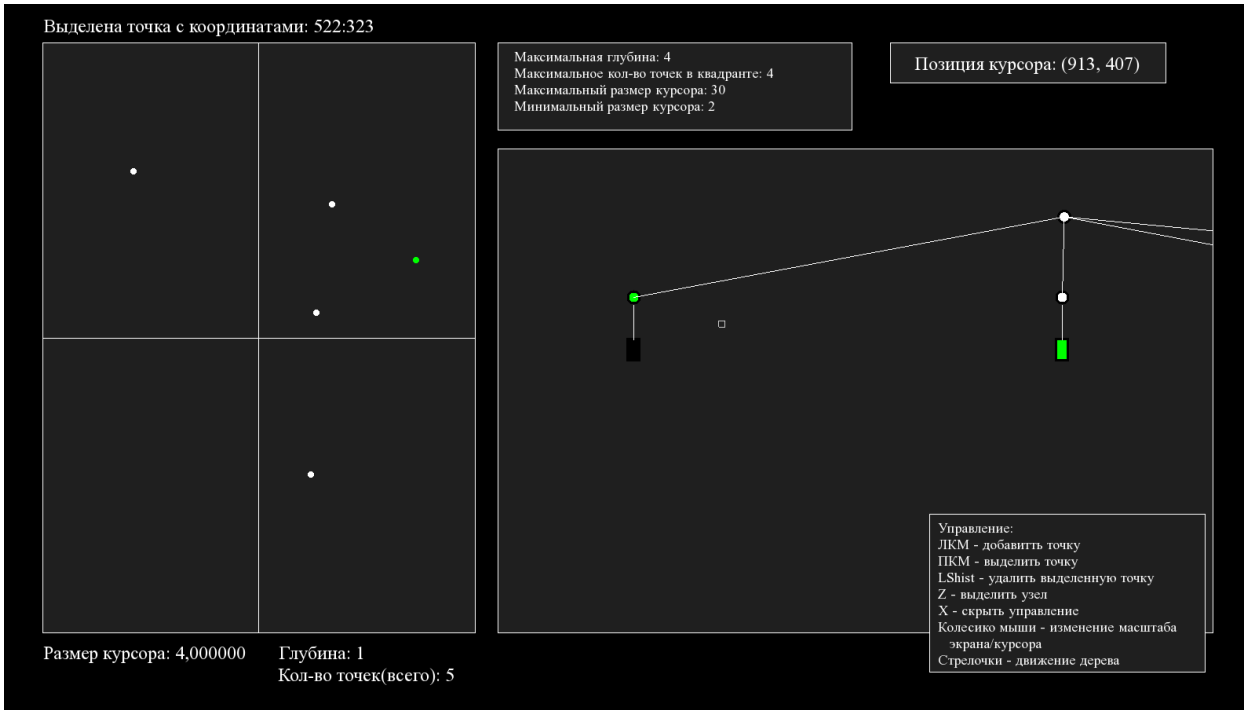

Рисунок 3 – реализация действий программы

Из рисунка 3 видно, что можно масштабировать экран визуализации дерева и менять его положение (прокрутка)

Также можно открывать экран управления

5. Вывод

Визуализация дерева квадрантов имеет такие плюсы:

1. Визуализация может помочь понять процесс работы дерева квадрантов и его алгоритмы.
2. Визуализация может помочь обнаружить проблемы и ошибки в работе дерева квадрантов.
3. Визуализация может помочь продемонстрировать преимущества и недостатки дерева квадрантов по сравнению с другими методами представления данных.
4. Визуализация может помочь сделать тему деревьев поиска более доступной и интересной для широкой аудитории.

В данной работе получилось реализовать визуализацию дерева квадрантов двумя разными представлениями:

1. Визуализация деления экрана на квадранты
2. Визуализация дерева квадрантов в виде узлов

Ссылка на репозиторий
https://github.com/pavlichek121/quadTree.git