Комитет по образованию г. Санкт-Петербург

ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБЩЕОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ

ПРЕЗИДЕНТСКИЙ ФИЗИКО-МАТЕМАТИЧЕСКИЙ ЛИЦЕЙ №239

Отчет о практике «Создание графических приложений на языке Java»

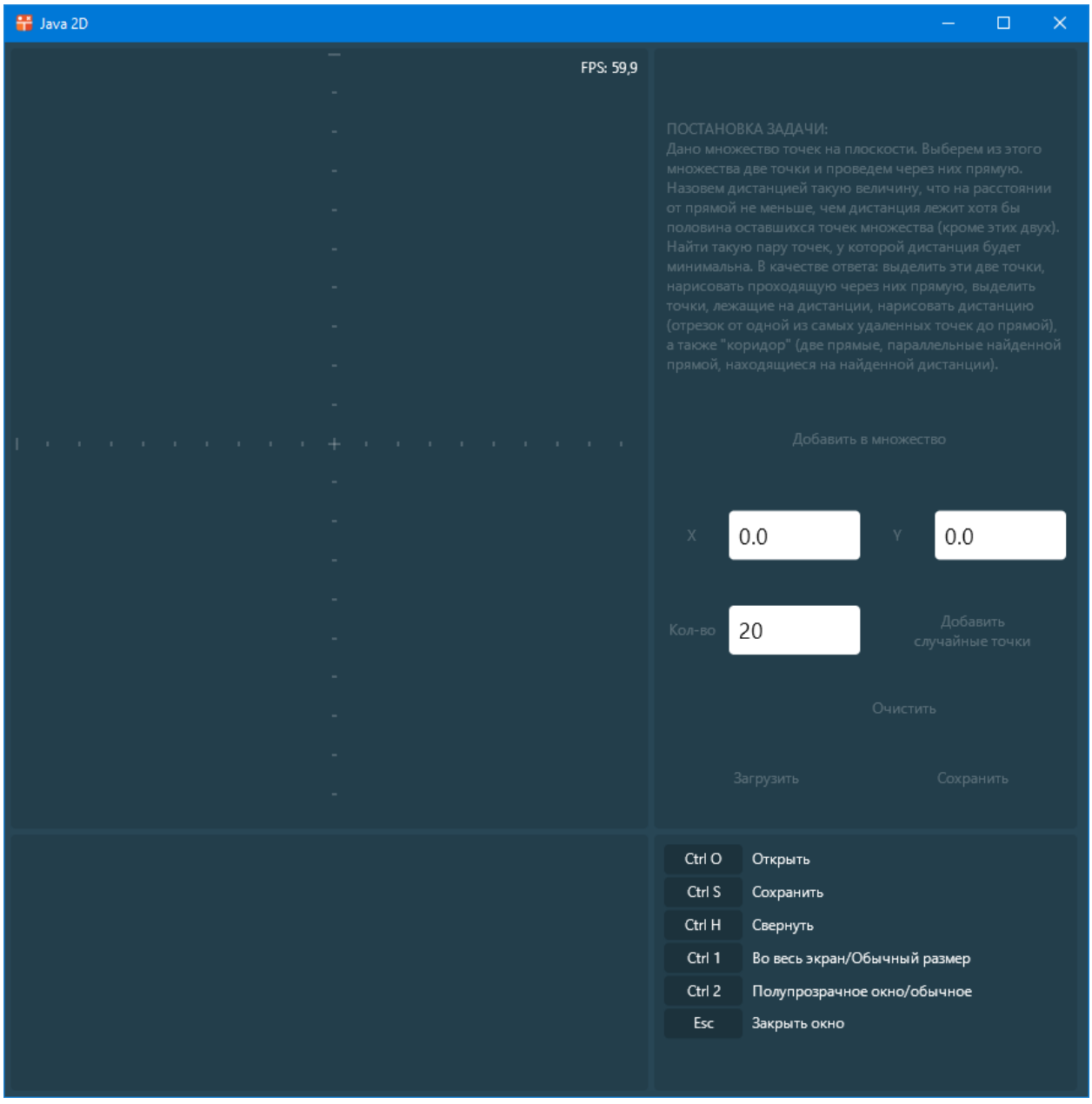
Учащаяся 10-3 класса
Нагорнова К.А.

Преподаватель:
Клюнин А.О.

Санкт-Петербург – 2023 год

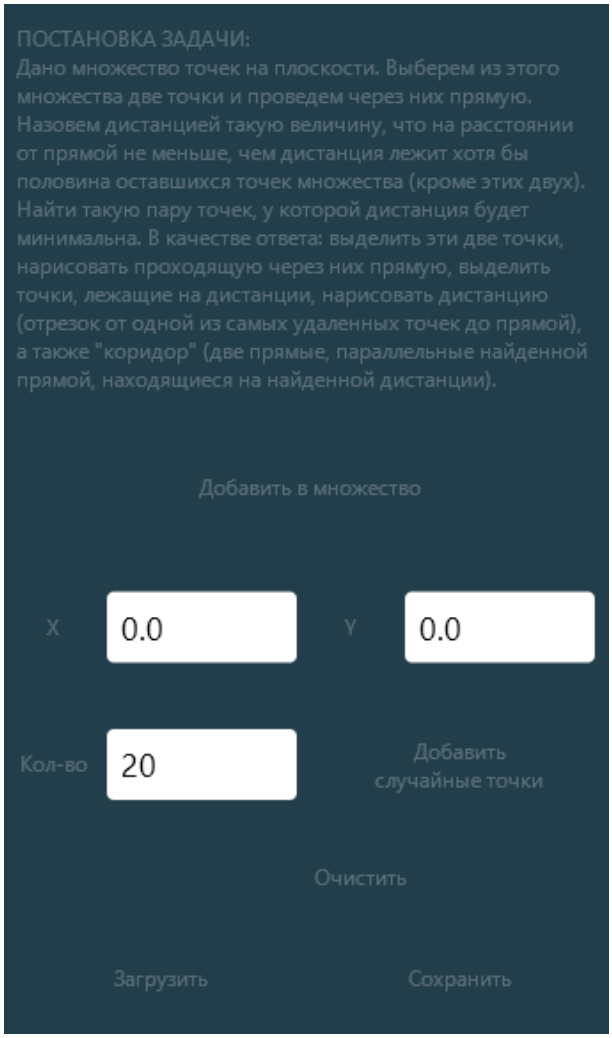
1. Постановка задачи

Дано множество точек на плоскости. Выберем из этого множества две точки и проведем через них прямую. Назовем дистанцией такую величину, что на расстоянии от прямой не меньше, чем дистанция лежит хотя бы половина оставшихся точек множества (кроме этих двух). Найти такую пару точек, у которой дистанция будет минимальна. В качестве ответа: выделить эти две точки, нарисовать проходящую через них прямую, выделить точки, лежащие на дистанции, нарисовать дистанцию (отрезок от одной из самых удаленных точек до прямой), а также "коридор" (две прямые, параллельные найденной прямой, находящиеся на найденной дистанции).



2. Элементы управления

В рамках данной задачи необходимо было реализовать следующие элементы управления:



Для добавления точки по координатам было создано два поля ввода: «X» и «Y».

Т.к. задача предполагает только один вид геометрических объектов, то для добавления случайных элементов достаточно одного поля ввода. В него вводится количество случайных точек, которые будут добавлены.

Также программа позволяет добавлять точки с помощью клика мышью по области рисования. Клик левой копкой добавляет точку, клик правой – выбирает точку для прямой.

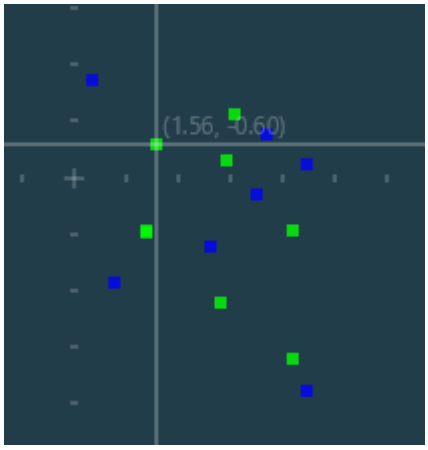
3. Структуры данных

Для того чтобы хранить точки, был разработан класс **Point.java**. Его листинг приведён в приложении А. В него были добавлены поля **pos**, соответствующее положению точки в пространстве задачи.

Для того чтобы хранить линии, был разработан класс **Line.java**. Его листинг приведён в приложении Б. Был создан список линий, и в решении я могла обращаться к линии по её индексу.

4. Рисование

Чтобы нарисовать точку, использовалась команда рисования прямоугольников **canvas.drawRect().**



5. Решение задачи

Для решения поставленной задачи в классе **Task** был разработан метод **solve().**

В нём находятся дистанции до выбранной линии от всех точек множества; выбираются 2 (красные) точки с минимальными дистанциями, и по ним строится прямая (красная). Затем через точку с максимальной дистанцией (зелёная) строится прямая (серая), параллельная второй прямой, и такая же прямая (серая) через точку, симметричную точке с максимальной дистанцией.

```
/**
 * Решить
 */
public void solve() {
    this.lines.add(new Line(
            selected.get(0),
            selected.get(1)
    ));

    //создаём массив выживших точек
    for (Point s: points) {
        if (!selected.contains(s)) survived.add(s);
    }
    for (Point t: survived) {
        Line l = lines.get(0);
        distSurv.add(l.getDistance(t));

    }
    //сортировка и вывод дистанций выживших точек
    Collections.sort(distSurv);
```

```
    for (double ds: distSurv) {
        System.out.println(ds);
    }
    for (Point u:survived){
        Line l = lines.get(0);
        if (l.getDistance(u)==distSurv.get(0) ||
l.getDistance(u)==distSurv.get(1))    {
            minDist.add(u);
        }
    }
    //точки на дистанции
    for (Point u:survived){
        Line l = lines.get(0);
        if (l.getDistance(u)>=distSurv.get(2) &&
l.getDistance(u)<distSurv.get(distSurv.size()/2))    {
            dist.add(u);
        }
    }
    //нарисуем вторую линию по точкам с мин. дистанциями
    this.lines.add(new Line(
            minDist.get(0),
            minDist.get(1)
    ));
    // w2 проекция w1 на вторую прямую
    Point w1 = findPoint(); //зелёная точка
    Line l2 = lines.get(1);
    double d = l2.getDistance(w1);
    double x11 = l2.pointA.pos.x;
    double y11 = l2.pointA.pos.y;
    double x12 = l2.pointB.pos.x;
    double y12 = l2.pointB.pos.y;
    double x2 = w1.pos.x;
    double y2 = w1.pos.y;
    double ex1 = -1;
    double ey1 = -1;
    double a = y12-y11;
    if (a==0) a=0.000001;
    double b = x11-x12;
    if (b==0) b=0.000001;
    double c = y11*x12-y12*x11;
    Point w2 = new Point(new Vector2d(0,0));

    for (int i = 0; i < 2; i++) {
        ex1=-ex1;
        for (int j = 0; j < 2; j++) {
            ey1=-ey1;
            w2 = new Point(new Vector2d(
                    ex1*d*(y11-y12)/Math.sqrt(a*a+b*b)+x2,
                    ey1*d*(x12-x11)/Math.sqrt(a*a+b*b)+y2
            ));
            if (Math.abs(a*w2.pos.x+b*w2.pos.y+c)<=0.5){
                //нарисуем максимальную дистанцию
                this.lines.add(new Line(w1, w2));
                i=j=2; //break
            }
        }
    }
    //w3 точка симметричная отн-но найденной прямой
    Point w3 = new Point(new Vector2d(
            w1.pos.x+2*(w2.pos.x-w1.pos.x),
            w1.pos.y+2*(w2.pos.y-w1.pos.y)

    ));
    this.wlist.add(w1);
```

```
    this.wlist.add(w2);
    this.wlist.add(w3);
    //this.lines.add(new Line(w2, w3));
    double c2 = -x2*a-y2*b;
    //рандомная точка для первой линии коридора
    double xw12=0;
    double yw12=0;
    if (a==0.000001) {
        xw12=w1.pos.x+1;
        yw12=w1.pos.y;
    } else if (b==0.000001) {
        xw12=w1.pos.x;
        yw12=w1.pos.y+1;
    } else {
        xw12=w1.pos.x+1;
        yw12=-c2/b-a*xw12/b;
    }

    Point w12 = new Point(new Vector2d(
            xw12,
            yw12
    ));
    this.lines.add(new Line(w1, w12));

    double c3 = -w3.pos.x*a-w3.pos.y*b;
    //рандомная точка для второй линии коридора
    double xw32=0;
    double yw32=0;
    if (a==0.000001) {
        xw32=w3.pos.x+1;
        yw32=w3.pos.y;
    } else if (b==0.000001) {
        xw32=w3.pos.x;
        yw32=w3.pos.y+1;
    } else {
        xw32=w3.pos.x+1;
        yw32=-c3/b-a*xw32/b;
    }

    Point w32 = new Point(new Vector2d(
            xw32,
            yw32
    ));
    this.lines.add(new Line(w3, w32));
}
```

6. Проверка

Для проверки правильности решённой задачи были разработаны unit-тесты. Их листинг приведён в приложении В.

Тест 1

Все точки: $\{(1,1);(-1,1);(-5,1);(2,1);(1,2);(2,2);\}$

Выбранные точки: $\{(-1,1);(1,2)\}$

Первая точка красной прямой: $\{(1,1)\}$

Вторая точка красной прямой: $\{(2,2)\}$

Зелёная точка: $\{(2,1)\}$

Проекция зелёной точки на красную прямую: $\{(1.5,1.5)\}$

Тест 2

Все точки: $\{(1,1);(-1,1);(-5,1);(4,1);(1,8);(2,2);\}$

Выбранные точки: $\{(4,1);(-1,1)\}$

Первая точка красной прямой: $\{(1,1)\}$

Вторая точка красной прямой: $\{(-5,1)\}$

Зелёная точка: $\{(2,2)\}$

Проекция зелёной точки на красную прямую: $\{(2,1)\}$

Тест 3

Все точки: $\{(1,1);(-1,1);(-5,2);(2,1);(1,2);(2,2);\}$

Выбранные точки: $\{(1,2);(-5,2)\}$

Первая точка красной прямой: $\{(2,1)\}$

Вторая точка красной прямой: $\{(2,2)\}$

Зелёная точка: $\{(1,1)\}$

Проекция зелёной точки на красную прямую: $\{(2,1)\}$

7. Заключение

В рамках выполнения поставленной задачи было создано графическое приложение с требуемым функционалом. Правильность решения задачи проверена с помощью юнит-тестов.

Приложение А. Point.java

```java
package app;

import com.fasterxml.jackson.annotation.JsonCreator;
import com.fasterxml.jackson.annotation.JsonIgnore;
import com.fasterxml.jackson.annotation.JsonProperty;
import misc.Misc;
import misc.Vector2d;
import java.util.Objects;

/**
 * Класс точки
 */
public class Point {
    /**
     * Координаты точки
     */
    public final Vector2d pos;

    /**
     * Конструктор точки
     *
     * @param pos      положение точки
     */
    @JsonCreator
    public Point(@JsonProperty("pos") Vector2d pos) {
        this.pos = pos;
    }

    /**
     * Получить цвет точки по её множеству
     *
     * @return цвет точки
     */
    @JsonIgnore
    public int getColor() {
        return  Misc.getColor(0xCC, 0x00, 0x00, 0xFF);
    }

    /**
     * Получить положение (нужен для json)
     *
     * @return положение
     */
    public Vector2d getPos() {
        return pos;
    }

    /**
     * Строковое представление объекта
     *
     * @return строковое представление объекта
     */
    @Override
    public String toString() {
        return "Point{" +
                ", pos=" + pos +
                '}';
    }

    /**
```

```
     * Проверка двух объектов на равенство
     *
     * @param o объект, с которым сравниваем текущий
     * @return флаг, равны ли два объекта
     */
    @Override
    public boolean equals(Object o) {
        // если объект сравнивается сам с собой, тогда объекты равны
        if (this == o) return true;
        // если в аргументе передан null или классы не совпадают, тогда
объекты не равны
        if (o == null || getClass() != o.getClass()) return false;
        // приводим переданный в параметрах объект к текущему классу
        Point point = (Point) o;
        return Objects.equals(pos, point.pos);
    }

    /**
     * Получить хэш-код объекта
     *
     * @return хэш-код объекта
     */
    @Override
    public int hashCode() {
        return Objects.hash(pos);
    }
}
```

Приложение Б.Line.java

```
package app;

import java.util.Objects;

public class Line {
    Point pointA;
    Point pointB;

    public Line(Point pointA, Point pointB) {
        this.pointA = pointA;
        this.pointB = pointB;
        System.out.println(pointA.pos.x + " " + pointA.pos.y);
        System.out.println(pointB.pos.x + " " + pointB.pos.y);
    }

    public double getDistance(Point pointC){
        double xA = this.pointA.pos.x;
        double xB = this.pointB.pos.x;
        double yA = this.pointA.pos.y;
        double yB = this.pointB.pos.y;
        double xC = pointC.pos.x;
        double yC = pointC.pos.y;
        double a = yB-yA;
        if (a==0) a=0.0000000000001;
        double b = xA-xB;
        if (b==0) b=0.0000000000001;
        double c = yA*xB-yB*xA;
        double d = Math.abs(a*xC+b*yC+c)/Math.sqrt(a*a+b*b);
        return d;
    }

    @Override
```

```
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        Line line = (Line) o;
        return Objects.equals(pointA, line.pointA) && Objects.equals(pointB,
line.pointB);
    }

    @Override
    public int hashCode() {
        return Objects.hash(pointA, pointB);
    }
}
```

Приложение B. UnitTest.java

```
import app.Line;
import app.Point;
import app.Task;
import misc.CoordinateSystem2d;
import misc.Vector2d;
import org.junit.Test;

import java.util.ArrayList;

/**
 * Класс тестирования
 */
public class UnitTest {

    /**
     * Тест
     *
     * @param points        список точек
*/
    private static void test(ArrayList<Point> points, ArrayList<Point>
selected, ArrayList<Line> lines) {
        Task task = new Task(new CoordinateSystem2d(10, 10, 20, 20), points,
selected);
        task.solve();
        // проверяем, что выбраны точки из общего списка точек
        for (Point p: task.getSelected()) {
            assert points.contains(p);
        }
        // проверяем, что выбранные точки не лежат в списке выживших точек
        for (Point p: selected) {
            assert !task.getSurvived().contains(p);
        }
        // проверяем, что первая линия проходит через выбранные точки
        assert task.getLines().get(0).equals(new
Line(selected.get(0),selected.get(1)));
        // проверяем, что обе красные точки выбраны правильно
        assert
lines.get(0).getDistance(task.getMinDist().get(0))==task.getDistSurv().get(0)
||
lines.get(0).getDistance(task.getMinDist().get(0))==task.getDistSurv().get(1)
;
        assert
lines.get(0).getDistance(task.getMinDist().get(1))==task.getDistSurv().get(1)
||
lines.get(0).getDistance(task.getMinDist().get(1))==task.getDistSurv().get(0)
;
```

```
        // проверяем, что через них проходит красная прямая
        Line lred = lines.get(1);
        Point pr1 = task.getMinDist().get(0);
        Point pr2 = task.getMinDist().get(1);
        assert lred.getDistance(pr1)<0.001&&lred.getDistance(pr2)<0.001;
        // проверяем, что зелёный отрезок правильно проведен
        Line lgreen = lines.get(2);
        Point pg1 = task.getWlist().get(0);
        Point pg2 = task.getWlist().get(1);
        assert lgreen.getDistance(pg1)<0.001&&lgreen.getDistance(pg2)<0.001;
        //проверяем, что точка, симметричная зелёной точке отн-но красной
прямой, лежит на зелёной прямой
        Point pg3 = task.getWlist().get(2);
        assert lgreen.getDistance(pg3)<0.001;

    }

    /**
     * Первый тест
     */
    @Test
    public void test1() {
        ArrayList<Point> points = new ArrayList<>();
        points.add(new Point(new Vector2d(1, 1)));
        points.add(new Point(new Vector2d(-1, 1)));
        points.add(new Point(new Vector2d(-5, 1)));
        points.add(new Point(new Vector2d(2, 1)));
        points.add(new Point(new Vector2d(1, 2)));
        points.add(new Point(new Vector2d(2, 2)));

        ArrayList<Point> selected = new ArrayList<>();
        selected.add(new Point(new Vector2d(1, 2)));
        selected.add(new Point(new Vector2d(-1, 1)));

        ArrayList<Line> lines = new ArrayList<>();
        lines.add(new Line(selected.get(0),selected.get(1))); //добавляем
первую прямую
        lines.add(new Line(new Point(new Vector2d(1, 1)), //крА
                           new Point(new Vector2d(2, 2)))); //крВ
        lines.add(new Line(new Point(new Vector2d(2, 1)), //зел
                new Point(new Vector2d(1.5, 1.5)))); //проекция зел

        /*1.0 1.0 //крА
        2.0 2.0 //крВ
        2.0 1.0 //зел
        1.5 1.5 //проекция зел*/
        test(points, selected, lines);
    }

    /**
     * Второй тест
     */
    @Test
    public void test2() {
        ArrayList<Point> points = new ArrayList<>();
        points.add(new Point(new Vector2d(1, 1)));
        points.add(new Point(new Vector2d(-1, 1)));
        points.add(new Point(new Vector2d(-5, 1)));
        points.add(new Point(new Vector2d(4, 1)));
        points.add(new Point(new Vector2d(1, 8)));
        points.add(new Point(new Vector2d(2, 2)));

        ArrayList<Point> selected = new ArrayList<>();
        selected.add(new Point(new Vector2d(4, 1)));
```

```
        selected.add(new Point(new Vector2d(-1, 1)));

        ArrayList<Line> lines = new ArrayList<>();
        lines.add(new Line(selected.get(0),selected.get(1))); //добавляем
первую прямую
        lines.add(new Line(new Point(new Vector2d(1, 1)), //крА
                new Point(new Vector2d(-5, 1)))); //крВ
        lines.add(new Line(new Point(new Vector2d(2, 2)), //зел
                new Point(new Vector2d(2, 1)))); //проекция зел

        /*1.0 1.0 //крА
        -5.0 1.0 //крВ
        2.0 2.0 //зел
        2.0 1.0 //проекция зел*/
        test(points, selected, lines);
    }

    /**
     * Третий тест
     */
    @Test
    public void test3() {
        ArrayList<Point> points = new ArrayList<>();
        points.add(new Point(new Vector2d(5, 3)));
        points.add(new Point(new Vector2d(-1, 2)));
        points.add(new Point(new Vector2d(-5, 1)));
        points.add(new Point(new Vector2d(7, 0)));
        points.add(new Point(new Vector2d(-7, 1)));
        points.add(new Point(new Vector2d(3, 9)));
        points.add(new Point(new Vector2d(15, 8)));

        ArrayList<Point> selected = new ArrayList<>();
        selected.add(new Point(new Vector2d(-1, 2)));
        selected.add(new Point(new Vector2d(-7, 1)));

        ArrayList<Line> lines = new ArrayList<>();
        lines.add(new Line(selected.get(0),selected.get(1))); //добавляем
первую прямую
        lines.add(new Line(new Point(new Vector2d(5, 3)), //крА
                new Point(new Vector2d(-5, 1)))); //крВ
        lines.add(new Line(new Point(new Vector2d(7, 0)), //зел
                new Point(new Vector2d(6.3, 3.2)))); //проекция зел
        /*5.0 3.0 //крА
        -5.0 1.0 //крВ
        7.0 0.0 //зел
        6.3 3.2 //проекция зел*/
        test(points, selected, lines);
    }
}
```