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 10-2 класса
Скаленко Ю.П.

Преподаватель:
Клюнин А.О.

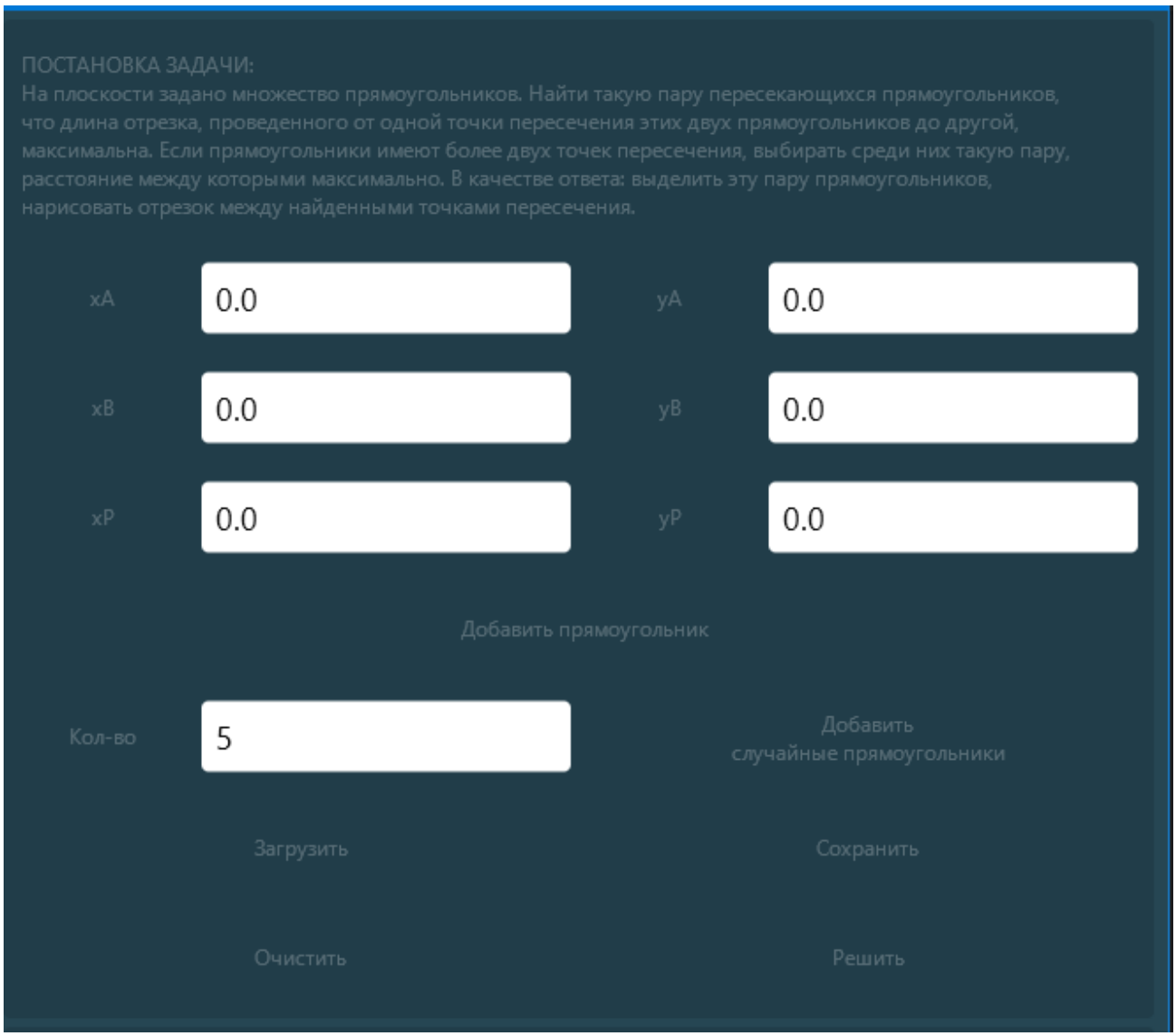
Санкт-Петербург – 2023 год

1. Постановка задачи

На плоскости задано множество прямоугольников. Найти такую пару пересекающихся прямоугольников, что длина отрезка, проведенного от одной точки пересечения этих двух прямоугольников до другой, максимальна. Если прямоугольники имеют более двух точек пересечения, выбирать среди них такую пару, расстояние между которыми максимально. В качестве ответа: выделить эту пару прямоугольников, нарисовать отрезок между найденными точками пересечения.

2. Элементы управления

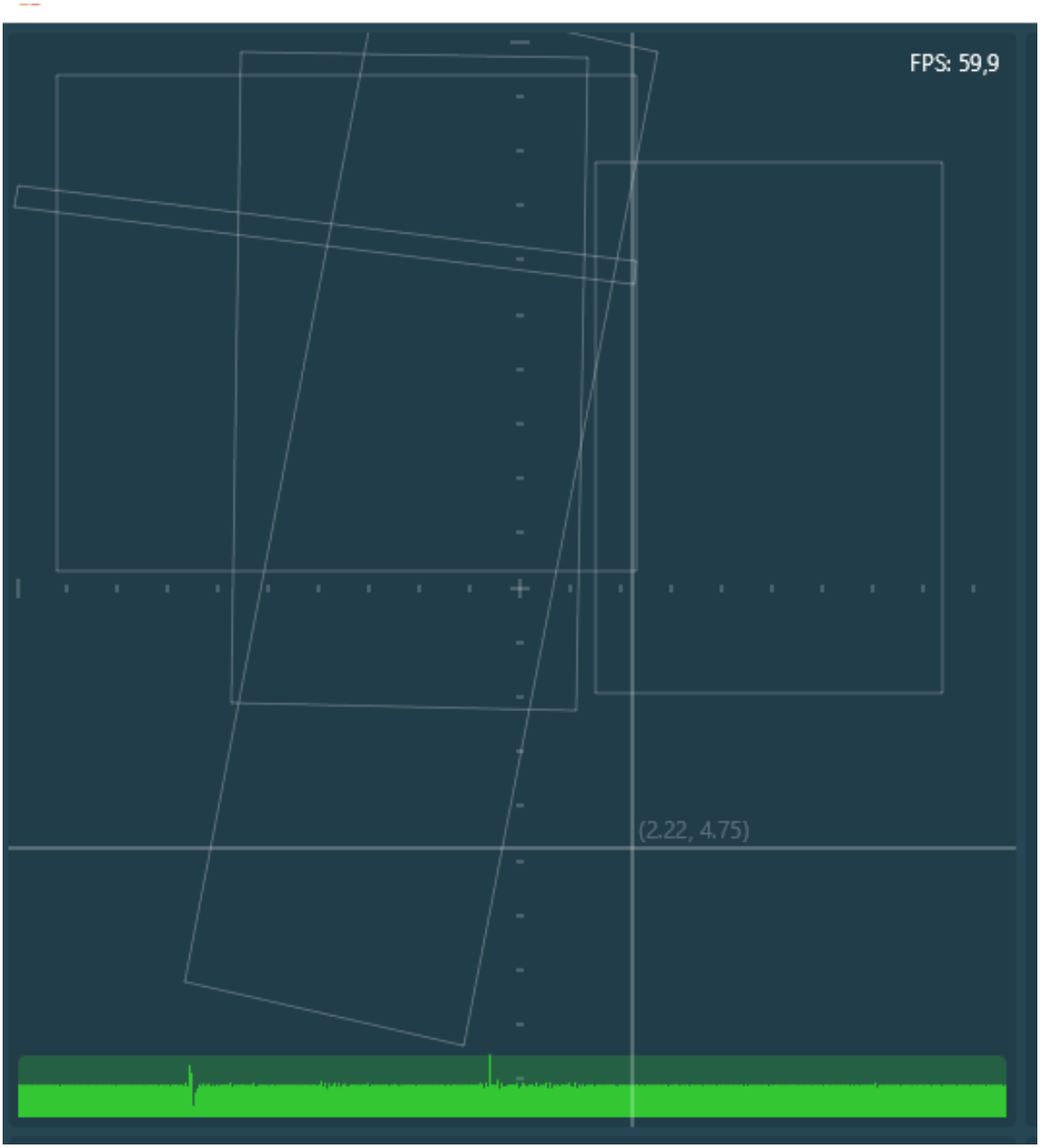
В рамках данной задачи необходимо было реализовать следующие элементы управления:



Для добавления прямоугольника по трем координатам было создано шесть полей ввода: «xA», «yA», «xB», «yB», «xP», «yP».

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

Также программа позволяет добавлять прямоугольники с помощью трех кликов мышью по области рисования



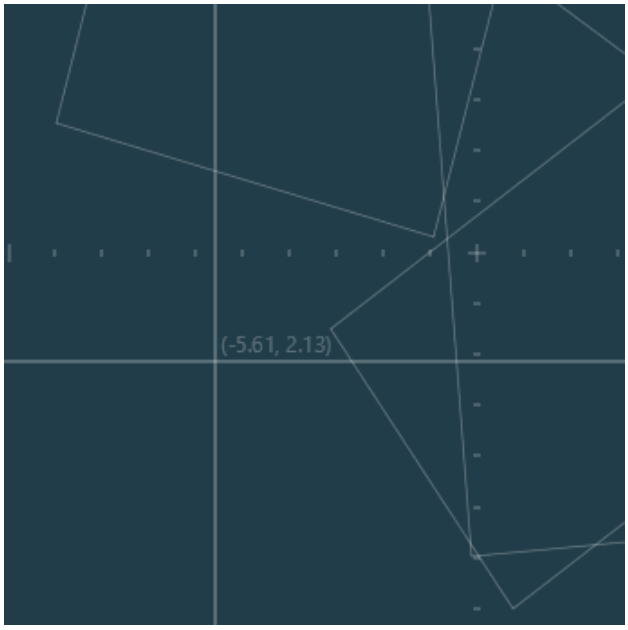
3. Структуры данных

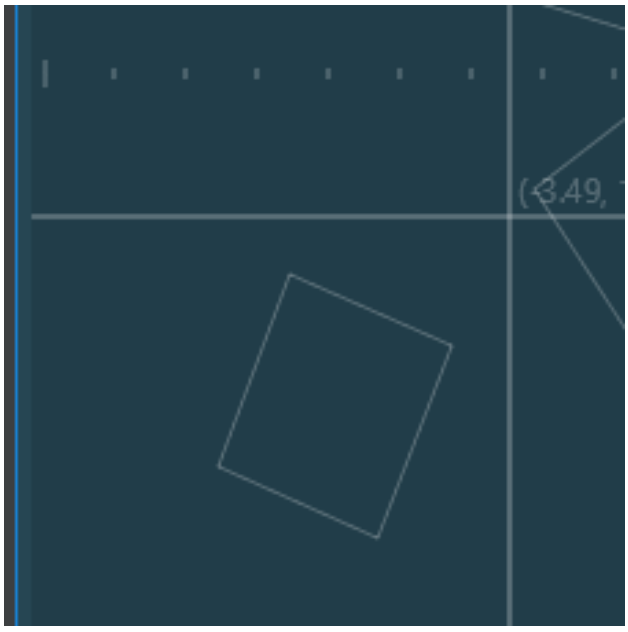
Для того чтобы хранить точки прямоугольника, был разработан класс **Rect.java.** Его листинг приведён в приложении А.

В него были добавлены поля **pointA**, **pointB**, **pointP**, **pointC**, **pointD**, соответствующие положениям пяти точек прямоугольника в пространстве задачи.

4. Рисование

Чтобы нарисовать точку, использовались команды рисования четырех сторон прямоугольника **canvas.drawLine()**





5. Решение задачи

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

```
public void solve() {
    // очищаем списки
    InterPoints.clear();
    segments.clear();
    answer = false;
    double maxlength = 0.0;  //максимальная длина отрезка
    for (int i = 0; i < rects.size(); i++) {
        for (int j = i + 1; j < rects.size(); j++) {  //перебираем все пары
прямоугольников
            app.Rect first = rects.get(i);
            app.Rect second = rects.get(j);
            Line[] lines = new Line[8];
            lines[0] = new Line(first.pointA, first.pointB);
            lines[1] = new Line(first.pointA, first.pointD);
            lines[2] = new Line(first.pointB, first.pointC);
            lines[3] = new Line(first.pointC, first.pointD);
            lines[4] = new Line(second.pointA, second.pointB);
            lines[5] = new Line(second.pointA, second.pointD);
            lines[6] = new Line(second.pointB, second.pointC);
            lines[7] = new Line(second.pointC, second.pointD);  //8 прямых,
построенных по сторонам прямоугольников
            for (int k = 0; k < 4; k++) {
                for (int l = 4; l < 8; l++) {  //перебираем все пары прямых
                    Vector2d InterCandidate =
lines[k].intersection(lines[l]);  //точка пересечения прямых, она может
лежать либо на отрезках, либо вне их
                    if (InterCandidate != null) {  //если точка пересечения
есть
                        if ((((InterCandidate.x >= lines[k].pointA.x) &&
(InterCandidate.x <= lines[k].pointB.x))
                                || (InterCandidate.x <= lines[k].pointA.x) &&
(InterCandidate.x >= lines[k].pointB.x))
                                && (((InterCandidate.y >= lines[k].pointA.y)
&& (InterCandidate.y <= lines[k].pointB.y))
                                || (InterCandidate.y <= lines[k].pointA.y) &&
(InterCandidate.y >= lines[k].pointB.y)))  //принадлежит ли точка первому
отрезку
                        {
                            if ((((InterCandidate.x >= lines[l].pointA.x) &&
(InterCandidate.x <= lines[l].pointB.x))
                                    || (InterCandidate.x <=
lines[l].pointA.x) && (InterCandidate.x >= lines[l].pointB.x))
                                    && (((InterCandidate.y >=
lines[l].pointA.y) && (InterCandidate.y <= lines[l].pointB.y))
                                    || (InterCandidate.y <=
lines[l].pointA.y) && (InterCandidate.y >= lines[l].pointB.y))) //принадлежит
ли точка второму отрезку
                            {
                                InterPoints.add(InterCandidate);  //если
лежит на обоих отрезках, добавляем в список точек пересечения
                            }
                        }
                    }
                }
            }
            for (int k = 0; k < InterPoints.size(); k++) {
                for (int l = k; l < InterPoints.size(); l++) {  //перебираем
```

```
все точки пересечения (для данной пары прямоугольников)
                    segments.add(new Segment(InterPoints.get(k),
InterPoints.get(l), first, second));  //добавляем все отрезки в список
                }
            }
            InterPoints.clear();
        }
    }
    for (Segment segment : segments) {
        if (segment.length() > maxlength) {
            maxlength = segment.length();
            result = segment;  //ищем отрезок максимальной длины
        }
    }
    if (result != null)
        answer = true;
    // задача решена
    solved = true;
}
```

В нём перебираются пары прямоугольников, далее перебираются 8 прямых, построенных по сторонам прямоугольника. Если точка пересечения прямых лежит на отрезках двух прямоугольников, то она добавляется в список точек пересечения. Далее перебираются все точки пересечения для данной пары прямоугольников, и добавляются все отрезки в список. Потом выбирается максимальный отрезок.

6. Проверка

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

Тест 1

Точки первого прямоугольника: $\{(1,1);(8,0);\ (3,3)\}$

Точки второго прямоугольника:$\{(-9,-9);(-8,-9);\ (-8,-5)\}$

Точки третьего прямоугольника $\{(9,-5);(6,-3);\ (7,-7)\}$

эти прямоугольники не пересекаются, ответа быть не должно

Тест 2

Точки первого прямоугольника: $\{(0,-9);(0,9);\ (2,0)\}$

Точки второго прямоугольника:$\{(1,-8);(1,8);\ (-2,0)\}$

Крайние точки отрезка пересечения: $\{(0,8);(0,-8)\}$

Тест 3

Точки первого прямоугольника: $\{(-5,0);(0,5);\ (2.5,-2.5)\}$

Точки второго прямоугольника:$\{(-1,8);(8,9);\ (-2,-7)\}$

Крайние точки отрезка пересечения: $\{(-0{,}6;\ 4{,}4);(-0{,}1;\ -0{,}1)\}$

7. Заключение

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

Приложение А. Rect.java

```
package app;

import com.fasterxml.jackson.annotation.JsonCreator;
import com.fasterxml.jackson.annotation.JsonProperty;
import misc.Vector2d;
import misc.Vector2i;

import java.util.Objects;

public class Rect { //Прямоугольник задается двумя вершинами одной из сторон,
а также точкой, лежащей на прямой, проходящей через две другие вершины.

    /**
     * Вершины, прямоугольника и точка, нужная для его задания
     */
    public final Vector2d pointA;
    public final Vector2d pointB;
    public final Vector2d pointC;
    public final Vector2d pointD;
    public final Vector2d pointP;
    /**
     * Конструктор прямоугольника
     *
     * @param pointA положение первой вершины прямоугольника
     * @param pointB положение второй вершины прямоугольника
     * @param pointP положение точки на противоположной стороне
прямоугольника
     */
    @JsonCreator
    public Rect(@JsonProperty("pointA") Vector2d pointA,
@JsonProperty("pointB") Vector2d pointB, @JsonProperty("pointP") Vector2d
pointP) {
        this.pointA = pointA;
        this.pointB = pointB;
        this.pointP = pointP;
        Line line = new Line(this.pointA, this.pointB);
        // рассчитываем расстояние от прямой до точки
        double dist = line.getDistance(this.pointP);
        // рассчитываем векторы для векторного умножения
        Vector2d AB = Vector2d.subtract(this.pointB, this.pointA);
        Vector2d AP = Vector2d.subtract(this.pointP, this.pointA);
        // определяем направление смещения
        double direction = Math.signum(AB.cross(AP));
        // получаем вектор смещения
        Vector2d offset = AB.rotated(Math.PI / 2 *
direction).norm().mult(dist);
        this.pointC = Vector2d.sum(this.pointB, offset);
        this.pointD = Vector2d.sum(this.pointA, offset);
    }
    /**
     * Получить положение первой вершины прямоугольника
     *
     * @return положение первой вершины прямоугольника
     */
    public Vector2d getpointA(){
        return pointA;
    }
    /**
     * Получить положение второй вершины прямоугольника
     *
```

```
     * @return положение второй вершины прямоугольника
     */
    public Vector2d getpointB(){
        return pointB;
    }
    /**
     * Получить положение точки на противоположной стороне прямоугольника
     *
     * @return положение точки на противоположной стороне прямоугольника
     */
    public Vector2d getpointP(){
        return pointP;
    }

    /**
     * Строковое представление объекта
     *
     * @return строковое представление объекта
     */
    @Override
    public String toString() {
        return "Rectangle{" +
                "pointA=" + pointA +
                ", pointB=" + pointB +
                ", pointP=" + pointP +
                '}';
    }
    /**
     * Получить хэш-код объекта
     *
     * @return хэш-код объекта
     */
    @Override
    public int hashCode() {
        return Objects.hash(pointA, pointB, pointP);
    }
}
```

Приложение Б. UnitTest.java

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

import java.util.ArrayList;
import java.util.HashSet;
import java.util.Objects;
import java.util.Set;

/**
 * Класс тестирования
 */
public class UnitTest {
    /**
     * Первый тест
     */
    @Test
    public void test1() {
        ArrayList<Rect> rects = new ArrayList<>();
        rects.add(new Rect(new Vector2d(1,1), new Vector2d(8,0), new
Vector2d(3,3)));
        rects.add(new Rect(new Vector2d(-9,-9), new Vector2d(-8,-9), new
Vector2d(-8,-5)));
        rects.add(new Rect(new Vector2d(9,-5), new Vector2d(6,-3), new
Vector2d(7,-7)));
        // эти прямоугольники не пересекаются, ответа быть не должно
        Task task = new Task(new CoordinateSystem2d(10, 10, 20, 20), rects);
        task.solve();
        // проверка, есть ли ответ
        assert !task.isAnswer();
    }

    /**
     * Второй тест
     */
    @Test
    public void test2() {
        ArrayList<Rect> rects = new ArrayList<>();
        rects.add(new Rect(new Vector2d(0,-9), new Vector2d(0,9), new
Vector2d(2,0)));
        rects.add(new Rect(new Vector2d(1,-8), new Vector2d(1,8), new
Vector2d(-2,0)));
        // очевидно, что самое длинное пересечение проходит по оси Y, от
точки (0, -8) до точки (0, 8)
        rects.add(new Rect(new Vector2d(1,1), new Vector2d(8,0), new
Vector2d(3,3)));
        rects.add(new Rect(new Vector2d(-9,-9), new Vector2d(-8,-9), new
Vector2d(-8,-5)));
        rects.add(new Rect(new Vector2d(9,-5), new Vector2d(6,-3), new
Vector2d(7,-7)));
        Task task = new Task(new CoordinateSystem2d(10, 10, 20, 20), rects);
        task.solve();
        // проверка самого длинного пересечения
        assert (Objects.equals(task.getResult().pointA, new Vector2d(0, -8))
|| Objects.equals(task.getResult().pointA, new Vector2d(0, 8)));
        assert (Objects.equals(task.getResult().pointB, new Vector2d(0, -8))
```

```
|| Objects.equals(task.getResult().pointB, new Vector2d(0, 8)));
    }

    /**
     * Третий тест
     */
    @Test
    public void test3() {
        ArrayList<Rect> rects = new ArrayList<>();
        Rect r1 = new Rect(new Vector2d(-5,0), new Vector2d(0,5), new
Vector2d(2.5,-2.5));
        Rect r2 = new Rect(new Vector2d(-1,8), new Vector2d(8,9), new
Vector2d(-2,-7));
        rects.add(r1);
        rects.add(r2);
        // очевидно, что самое длинное пересечение у этих двух
прямоугольников
        rects.add(new Rect(new Vector2d(1,1), new Vector2d(8,0), new
Vector2d(3,3)));
        rects.add(new Rect(new Vector2d(-9,-9), new Vector2d(-8,-9), new
Vector2d(-8,-5)));
        rects.add(new Rect(new Vector2d(9,-5), new Vector2d(6,-3), new
Vector2d(7,-7)));
        Task task = new Task(new CoordinateSystem2d(10, 10, 20, 20), rects);
        task.solve();
        // проверка самого длинного пересечения
        assert (Objects.equals(task.getResult().rect1, r1));
        assert (Objects.equals(task.getResult().rect2, r2));
    }
}
```