Комитет по образованию г. Санкт-Петербург

ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБЩЕОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ

ПРЕЗИДЕНТСКИЙ ФИЗИКО-МАТЕМАТИЧЕСКИЙ ЛИЦЕЙ №239

Отчет о практике «Создание графических приложений на языке Java»

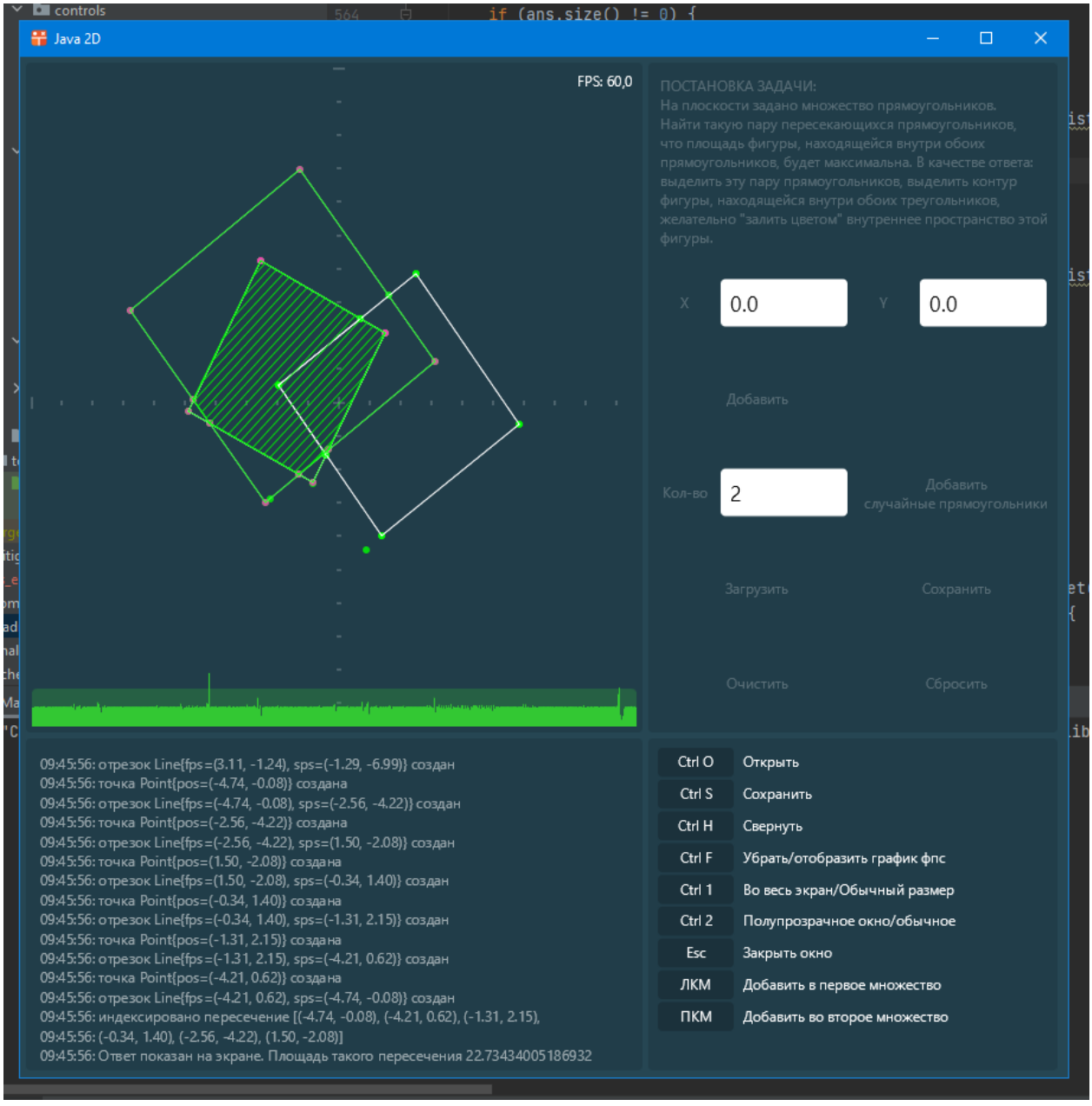
Учащийся 10-2 класса
Фролов Т.С.

Преподаватель:
Клюнин А.О.

Санкт-Петербург – 2023 год

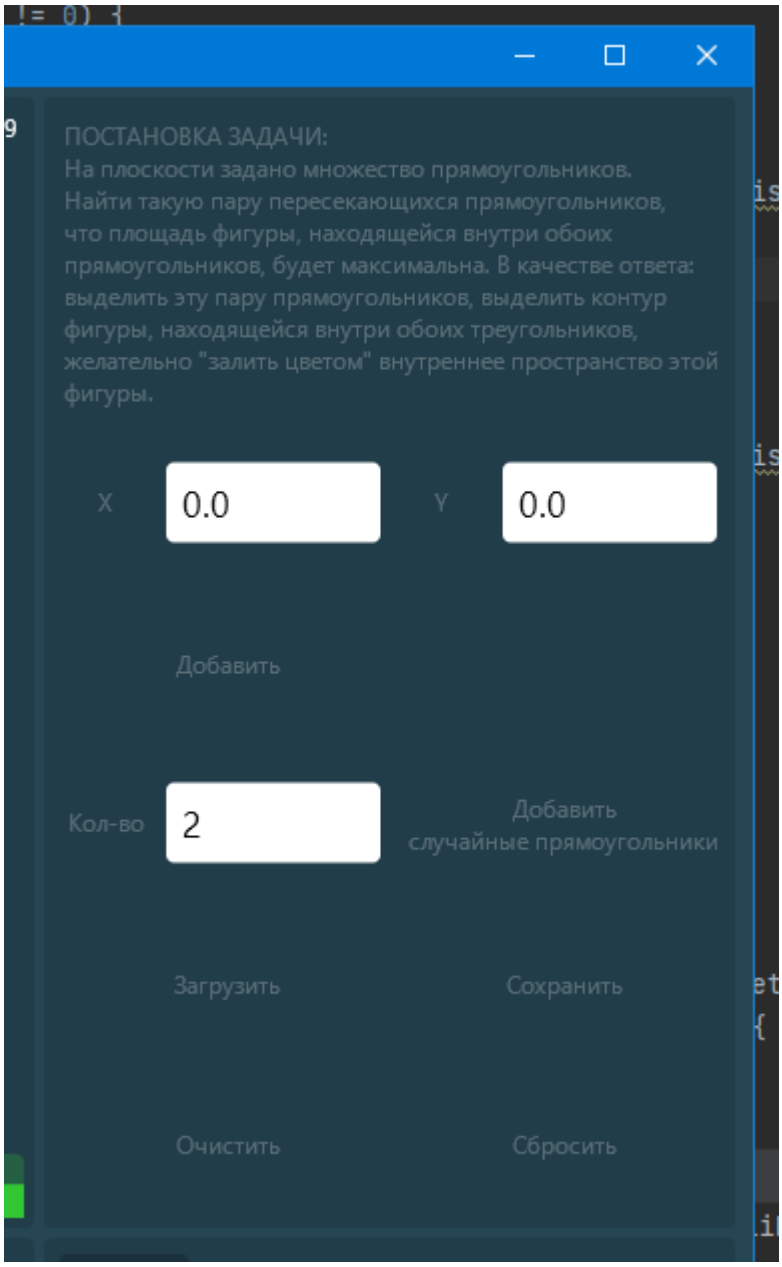
1. Постановка задачи

На плоскости задано множество прямоугольников. Найти такую пару пересекающихся прямоугольников, что площадь фигуры, находящейся внутри обоих прямоугольников, будет максимальна. В качестве ответа: выделить эту пару прямоугольников, выделить контур фигуры, находящейся внутри обоих прямоугольников, желательно "залить цветом" внутреннее пространство этой фигуры.



2. Элементы управления

В рамках данной задачи необходимо было реализовать следующие элементы управления:

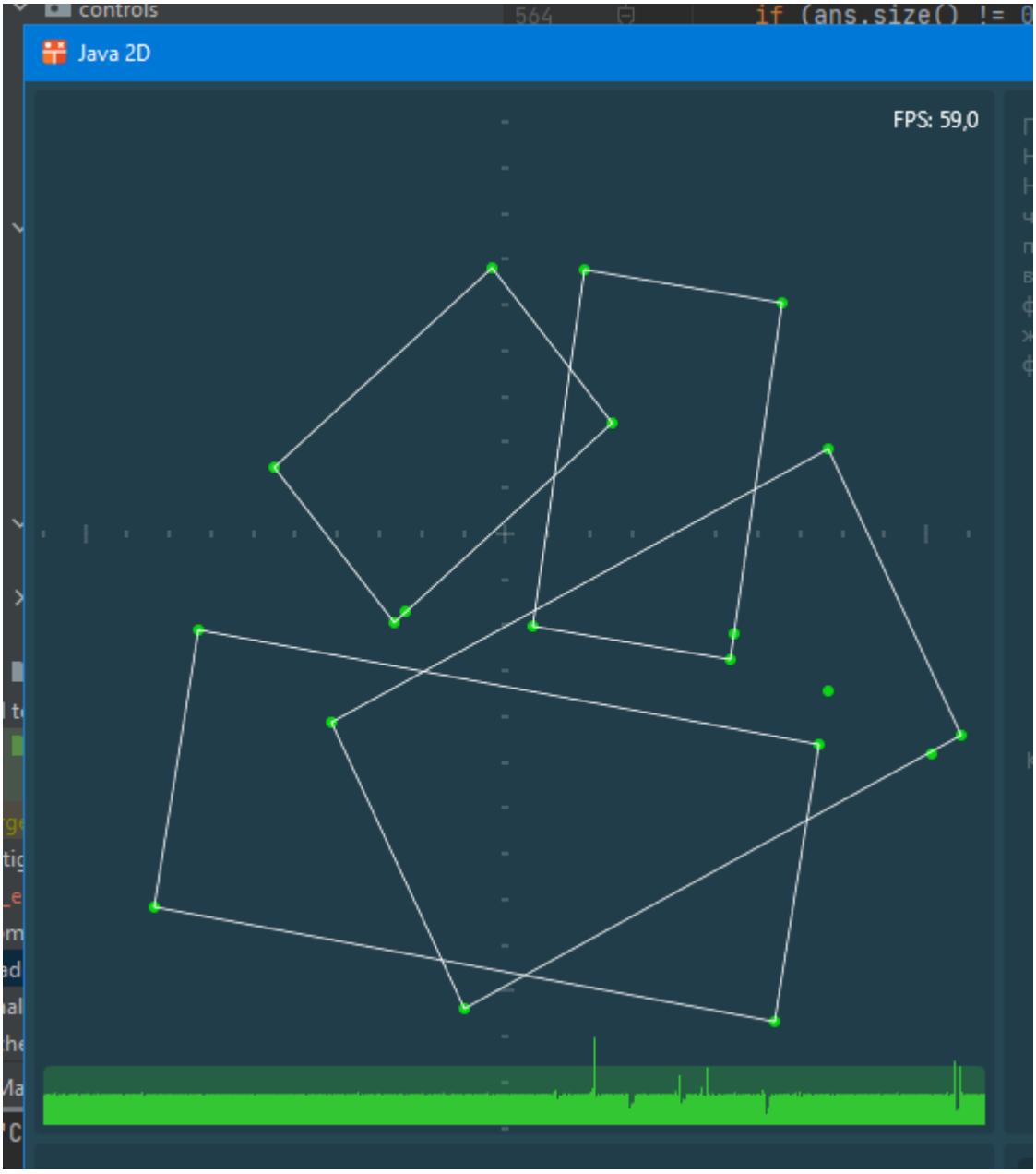


Для добавления прямоугольника необходимо ввести три точки: первые две определяют опорный отрезок, одна из сторон прямоугольника, третья определяет две остальные вершины, лежащими на одной прямой с ней.

Для добавления точки по координатам было создано два поля ввода: «X» и «Y». Чтобы добавить точку, была создана кнопка «Добавить».

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 точки, отрезки и прямоугольники с помощью клика мышью по области рисования



При клике кнопкой мыши по области рисования в месте клика создаётся точка, при втором клике – вторая точка, а, следовательно, первый отрезок прямоугольника. Ну и последняя точка задаёт оставшуюся часть прямоугольника

3. Структуры данных

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

В него были добавлены поле **pos**, соответствующее положению точки в пространстве задачи**.**

Для того чтобы хранить отрезки, был разработан класс **Line.java.** Его листинг приведён в приложении Б.

В него были добавлены поля **pointA и pointB**, соответствующее положению точек отрезка в пространстве задачи и коэффициенты прямой, содержащей этот отрезок **a, b** и **c.**

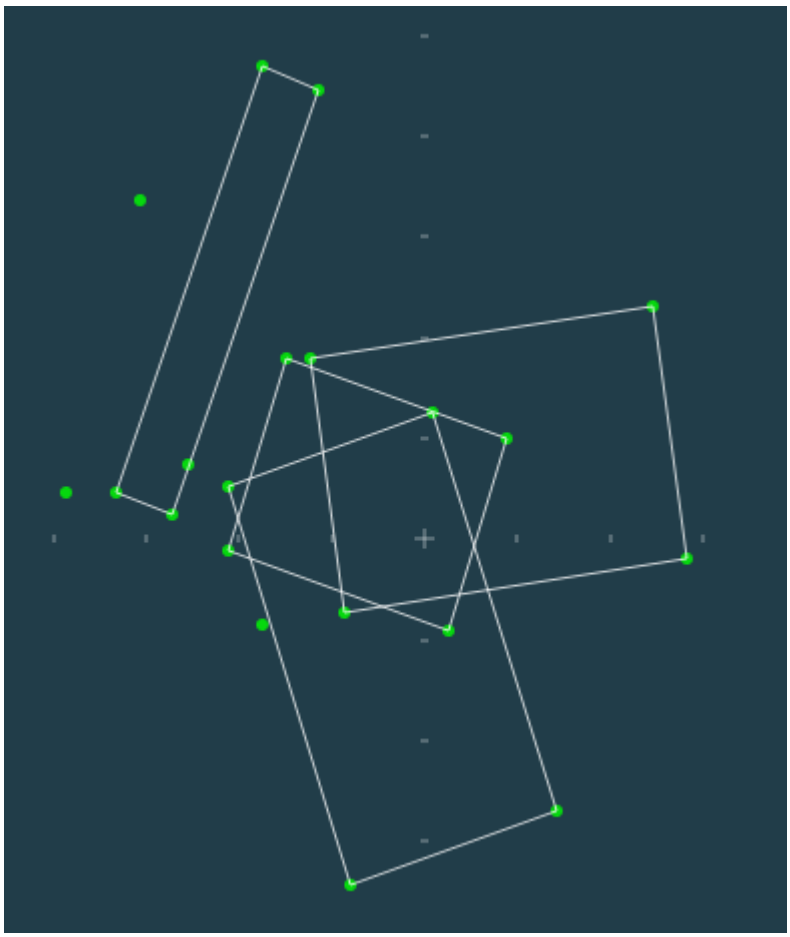
Для того чтобы хранить прямоугольники, был разработан класс **Rectangle.java.** Его листинг приведён в приложении В.

В него были добавлены поля **pointA, pointB, pointC** и **pointD**, соответствующее положению точек прямоугольника в пространстве задачи и его отрезки **AB, BC, CD** и **DA.**

4. Рисование

Чтобы нарисовать точку, использовалась команда рисования кругов **canvas.drawRRect().**

Чтобы нарисовать отрезок, использовалась команда рисования отрезков **canvas.drawLine().**



5. Решение задачи

Для решения поставленной задачи в классе **Task** был разработан метод **solve().** Его листинг приведён в приложении Г.

В нём перебираются пары прямоугольников и, если площадь их пересечения больше максимальной (по умолчанию 0), то обе эти прямоугольники добавляются в ответ.

Потом идёт их выделение и рисование пересечения, используя штриховку.

Для вычисления площади пересечения в классе **Task** был разработан метод **getAreaIntersRect().**

```
/**
 * нахождение площади пересечения
 */
7 usages  frolovts.24
public double getAreaIntersRect(Rectangle first, Rectangle second, int mode) {
    double s = 0;
    ArrayList<Vector2d> intersection = getIntersRect(first, second, mode);
    if (intersection.size() != 0) {
        intersection.add(intersection.get(0));
        //Вычисление площади пересечения по методу Гаусса
        for (int i = 0; i < intersection.size() - 1; i++) {
            s += intersection.get(i).x * -intersection.get(i + 1).y - intersection.get(i + 1).x * -intersection.get(i).y;
        }
    }

    return Math.abs(s / 2);
}
```

Он вычисляет площадь фигуры по методу Гаусса. Для этого ему нужно строго отсортированное множество точек, которое он получает от метода **getIntersRect()**, разработанный в классе **Task.** Его листинг приведен в приложении Д.

Этот метод находит множество точек: вершины прямоугольника, лежащие в другом прямоугольнике (точки объединения) и точки пересечения сторон прямоугольников (точки пересечения). В конце он возвращает отсортированное множество (т.к. порядок точек в методе Гаусса и в методе рисования фигуры пересечения важен) точек, используя метод **sortForCF()**, разработанный в классе **Task**. Его листинг приведен в приложении Е.

Этот же метод сортирует точки по хитрой схеме: сначала находится самая левая точка (с наименьшим x), потом все точки сортируются по тангенсу угла между вертикалью и прямой, соединяющей самую левую и обрабатываемую точку. Таким образом получается множество точек выпуклой фигуры.

6. Проверка

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

Тест 1

Этот тест проверяет, правильно ли метод **getIntersLines()** находит пересечение прямых.

Прямая 1: {(1, -4), (1, 2)}

Прямая 2: {(0, 0), (2, 2)}

Точка пересечения: (1, 1)

Тест 2

Этот тест проверяет, правильно ли метод **getIntersLines()** находит пересечение прямых.

Прямая 1: {(1, 1), (2, 2)}

Прямая 2: {(2, 1), (1, 2)}

Точка пересечения: (1.5, 1.5)

Тест 3

Этот тест проверяет, правильно ли метод **getDistance()** находит расстояние от точки до прямой.

Точка: (0, 0)

Прямая: {(2, 1), (1, 2)}

Расстояние: 2.1213

Тест 4

Этот тест проверяет, правильно ли метод **getIntersLines()** находит пересечение прямых.

Прямая 1: {(0, 2), (5, 0)}

Прямая 2: {(0, 0), (8, 4)}

Точка пересечения: (2.222, 1.111)

Тест 5

Этот тест проверяет, правильно ли метод **isIntersRect()** определяет принадлежность точки к прямоугольнику.

Точка: (0, 0)

Прямоугольник: {(-2, -1), (-2, 3), (5, 3), (5, -1)}

Принадлежит? **ДА**

Тест 6

Этот тест проверяет, правильно ли метод **isIntersRect()** определяет принадлежность точки к прямоугольнику.

Точка: (0, 0)

Прямоугольник: {(-0.39, -2.19), (3.89, 0.27), (2.20, 3.22), (-2.08, 0.76)}

Принадлежит? **ДА**

Тест 7

Этот тест проверяет, правильно ли метод **getAreaIntersRect()** находит площадь пересечения прямоугольников.

Прямоугольник 1: {(3.69, 1.46), (2.65, -3.18), (-0.78, -2.41), (0.26, 2.23)}

Прямоугольник 2: {(3.04, -5.37), (0.17, -2.22), (2.38, -0.20), (5.25, -3.35)}

Площадь пересечения: 4.98297

Тест 8

Этот тест проверяет, правильно ли решена задача.

Прямоугольник 1: {(3.69, 1.46), (2.65, -3.18), (-0.78, -2.41), (0.26, 2.23)}

Прямоугольник 2: {(3.04, -5.37), (0.17, -2.22), (2.38, -0.20), (5.25, -3.35)}

Прямоугольник 3: {(-1.56, -0.60), (-1.16, 4.13), (5.84, 3.54), (5.44, -1.19)}

Точка пересечения 1 и 2: 4.98297

Точка пересечения 2 и 3: 0.53269

Точка пересечения 1 и 3: 9.7005

Ответ:

фигура: {(-0.396, -0.698), (3.139, -0.997), (3.69, 1.46), (0.26, 2.23)}

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
 */
23 usages  frolovts.24
public class Point {
    /**
     * Координаты точки
     */
    6 usages
    public final Vector2d pos;
    /**
     * выделение
     */
    4 usages
    @JsonIgnore
    public boolean isAns;

    /**
     * Конструктор точки
     *
     * @param pos      положение точки
     */
    14 usages  frolovts.24
    @JsonCreator
    public Point(@JsonProperty("pos") Vector2d pos) {
        this.pos = pos;
        isAns = false;
    }
```

```java
/**
 * Получить положение
 * (нужен для json)
 *
 * @return положение
 */
frolovts.24
public Vector2d getPos() { return pos; }

/**
 * Строковое представление объекта
 *
 * @return строковое представление объекта
 */
frolovts.24
@Override
public String toString() {
    return "Point{" +
            "pos=" + pos +
            '}';
}

/**
 * Получить хэш-код объекта
 *
 * @return хэш-код объекта
 */
frolovts.24
@Override
public int hashCode() { return Objects.hash(pos); }
/**
 * Получить цвет точки
 *
 * @return цвет точки
 */
1 usage  frolovts.24
@JsonIgnore
public int getColor() {
    return isAns ? Misc.getColor( a: 0xCC,  r: 0xF8,  g: 0x3B,  b: 0xBB)
            : Misc.getColor( a: 0xCC,  r: 0x00,  g: 0xFF,  b: 0x0);
}
```

```java
    /**
     * пометить точку
     */
    1 usage   frolovts.24
    public void addToAns() { isAns = true; }
    /**
     * убрать из ответа
     */
    1 usage   frolovts.24
    public void RemoveFromAns() { isAns = false; }
}
```

Приложение Б. Line.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 panels.PanelLog;

import static java.lang.Math.max;
import static java.lang.Math.min;

/**
 * Класс линии
 */
112 usages  frolovts.24
public class Line {
    /**
     * Первая опорная точка
     */
    18 usages
    public final Vector2d pointA;
    /**
     * Вторая опорная точка
     */
    18 usages
    public final Vector2d pointB;
    /**
     * Первый коэффициент канонического уравнения прямой
     */
    20 usages
    private final double a;
    /**
     * Второй коэффициент канонического уравнения прямой
     */
    14 usages
    private final double c;
```

```java
/**
 * Конструктор линии
 *
 * @param pointA первая опорная точка
 * @param pointB вторая опорная точка
 */
63 usages  frolovts.24 *
@JsonCreator
public Line(@JsonProperty("pointA") Vector2d pointA,
            @JsonProperty("pointB") Vector2d pointB) {
    this.pointA = pointA;
    this.pointB = pointB;
    this.isAns = false;

    a = pointA.y - pointB.y;
    b = pointB.x - pointA.x;
    c = pointA.x * pointB.y - pointB.x * pointA.y;
}

/**
 * Получить расстояние до точки
 *
 * @param pos координаты точки
 * @return расстояние
 */
7 usages  frolovts.24 *
public double getDistance(Vector2d pos) {
    return
            Math.abs(a * pos.x + b * pos.y + c)
                    / Math.sqrt(a * a + b * b);
}
```

```java
/**
 * Строковое представление объекта
 *
 * @return строковое представление объекта
 */
```
frolovts.24
```java
@Override
public String toString() {
    return "Line{" +
            "fps=" + pointA +
            ", sps=" + pointB +
            '}';
}
/**
 * добавить к ответу
 */
```
6 usages frolovts.24
```java
public void addToAns() { isAns = true; }

/**
 * убрать из ответа
 */
```
1 usage frolovts.24
```java
public void RemoveFromAns() { isAns = false; }
/**
 * Получить цвет отрезка по его множеству
 *
 * @return цвет отрезка
 */
```
2 usages frolovts.24 *
```java
@JsonIgnore
public int getColor() {
    return (isAns ? Misc.getColor( a: 0xCC,  r: 0x00,  g: 0xFF,  b: 0x00)
            : Misc.getColor( a: 0xCC,  r: 0xFF,  g: 0xFF,  b: 0xFF));
}
```

```java
/**
 * пересекаются ли прямые
 */
1 usage   frolovts.24
public boolean isIntersLines(Line other) {
    if (b != 0 && other.b != 0) {
        return a / b != other.a / other.b;
    }
    else if (b == 0 && other.b == 0) {
        return a == other.a;
    }
    return true;
}
/**
 * получить пересечение прямых
 */
7 usages   frolovts.24 *
public Vector2d getIntersLines(Line other) {
    if (b != 0 && other.b != 0) {
        double x = (other.c * b - c * other.b) /
                (a * other.b - other.a * b);
        return new Vector2d(x,  y: -a * x / b - c / b);
    }
    else if (b == 0 && other.b != 0) {
        double x = -c / a;
        return new Vector2d(x,  y: -other.a * x /
                other.b - other.c / other.b);
    }
    double x = -other.c / other.a;
    return new Vector2d(x,  y: -a * x / b - c / b);
}
```

```java
    /**
     * вспомогательный метод к нижнему
     */
    1 usage   frolovts.24
    public boolean isInterLines(double aOther, double bOther, double cOther) {
        double x;
        if (b != 0) {
            x = (cOther * b - c * bOther) / (a * bOther - aOther * b);
            return min(pointA.x, pointB.x) < x && x < max(pointA.x, pointB.x);
        }
        x = -c / a;
        double y = -(aOther * x + cOther) / bOther;
        return min(pointA.y, pointB.y) < y && y < max(pointA.y, pointB.y);
    }
    /**
     * получить пересечение отрезка и прямой через коэффициенты
     *
     */
    2 usages   frolovts.24
    public Vector2d getInterLines(double aOther, double bOther, double cOther) {
        double x;
        if (b != 0) {
            x = (cOther * b - c * bOther) / (a * bOther - aOther * b);
            return new Vector2d(x,  y: -a * x / b - c / b);
        }
        x = -c / a;
        double y = -(aOther * x + cOther) / bOther;
        return new Vector2d(x, y);
    }
}
```

Приложение B. Rectangle.java

```java
package app;

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

import java.util.ArrayList;

/**
 * Класс линии
 */
38 usages  frolovts.24
public class Rectangle {
    /**
     * Первая опорная точка
     */
    7 usages
    public final Vector2d pointA;
    /**
     * Вторая опорная точка
     */
    11 usages
    public final Vector2d pointB;
    /**
     * Третья опорная точка
     */
    15 usages
    public final Vector2d pointC;
    /**
     * Четвертая опорная точка
     */
    11 usages
    public final Vector2d pointD;
```

```java
/**
 * Первый отрезок
 */
5 usages
public final Line AB;
/**
 * Второй отрезок
 */
5 usages
public final Line BC;
/**
 * Третий отрезок
 */
5 usages
public final Line CD;
/**
 * Четвертый отрезок
 */
5 usages
public final Line DA;
```

```java
/**
 * Конструктор прямоугольника
 *
 * @param pointA первая опорная точка
 * @param pointB вторая опорная точка
 * @param pointC третья опорная точка
 * @param pointD четвёртая опорная точка
 * @param AB первый отрезок
 * @param BC второй отрезок
 * @param CD третий отрезок
 * @param DA четвёртый отрезок
 */
9 usages  frolovts.24 *
@JsonCreator
public Rectangle(@JsonProperty("pointA") Vector2d pointA, @JsonProperty("pointB") Vector2d pointB,
                 @JsonProperty("pointC") Vector2d pointC, @JsonProperty("pointD") Vector2d pointD,
                 @JsonProperty("AB") Line AB, @JsonProperty("BC") Line BC,
                 @JsonProperty("CD") Line CD, @JsonProperty("DA") Line DA) {
    this.pointA = pointA;
    this.pointB = pointB;
    this.pointC = pointC;
    this.pointD = pointD;

    this.AB = AB;
    this.BC = BC;
    this.CD = CD;
    this.DA = DA;


}
```

```java
/**
 * Строковое представление объекта
 *
 * @return строковое представление объекта
 */
```
frolovts.24
```java
@Override
public String toString() {
    return "Rectangle{" +
            "A=" + pointA +
            ", B=" + pointB +
            ", C=" + pointC +
            ", D=" + pointD +
            ", AB=" + AB +
            ", BC=" + BC +
            ", CD=" + CD +
            ", DA=" + DA +
            '}';
}
/**
 * добавлен в решение
 */
```
frolovts.24
```java
public void addtoansR () {
    AB.addToAns(); BC.addToAns(); CD.addToAns(); DA.addToAns();
}
/**
 * получить список отрезков
 */
```
2 usages frolovts.24
```java
@JsonIgnore
public ArrayList<Line> getListLines() {
    ArrayList<Line> ans = new ArrayList<>();
    ans.add(AB); ans.add(BC); ans.add(CD); ans.add(DA);
    return ans;
}
```

```java
    /**
     * получить список точек
     */
    4 usages  frolovts.24
    @JsonIgnore
    public ArrayList<Vector2d> getListPoints() {
        ArrayList<Vector2d> ans = new ArrayList<>();
        ans.add(pointA); ans.add(pointB); ans.add(pointC); ans.add(pointD);
        return ans;
    }
    /**
     * получить нулевой прямоугольник
     */
    6 usages  frolovts.24 *
    public static Rectangle nullRectangle() {
        return new Rectangle(new Vector2d( x: 0,  y: 0), new Vector2d( x: 0,  y: 0),
                new Vector2d( x: 0,  y: 0), new Vector2d( x: 0,  y: 0),
                new Line(new Vector2d( x: 0,  y: 0), new Vector2d( x: 0,  y: 0)),
                new Line(new Vector2d( x: 0,  y: 0), new Vector2d( x: 0,  y: 0)),
                new Line(new Vector2d( x: 0,  y: 0), new Vector2d( x: 0,  y: 0)),
                new Line(new Vector2d( x: 0,  y: 0), new Vector2d( x: 0,  y: 0)));
    }
}
```

Приложение Г. Task.solve()

```java
/**
 * Решить задачу
 */
1 usage  frolovts.24
public void solve() {

    cancel();

    double sMax = 0;
    // перебираем пары прямоугольников
    for (int i = 0; i < rectangles.size() - 1; i++) {
        for (int j = i + 1; j < rectangles.size(); j++) {
            double s = getAreaIntersRect(rectangles.get(i), rectangles.get(j),  mode: 1);
            if (s > sMax) {
                sMax = s;
                ansRectF = rectangles.get(i);
                ansRectS = rectangles.get(j);
            }
        }
    }
    ArrayList<Vector2d> ans = sortForCF(getIntersRect(ansRectF, ansRectS,  mode: 1));
    if (ans.size() != 0) {
        ans.add(ans.get(0));

        // выделяем пару
        for (int i = 0; i < 4; i++) { addPointInter(ansRectF.getListPoints().get(i)); }
        addLineInter(ansRectF.pointA, ansRectF.pointB);
        addLineInter(ansRectF.pointB, ansRectF.pointC);
        addLineInter(ansRectF.pointC, ansRectF.pointD);
        addLineInter(ansRectF.pointD, ansRectF.pointA);

        for (int i = 0; i < 4; i++) { addPointInter(ansRectS.getListPoints().get(i)); }
        addLineInter(ansRectS.pointA, ansRectS.pointB);
        addLineInter(ansRectS.pointB, ansRectS.pointC);
        addLineInter(ansRectS.pointC, ansRectS.pointD);
        addLineInter(ansRectS.pointD, ansRectS.pointA);
```

```
        for (int i = 0; i < ans.size() - 1; i++) {
            addPointInter(ans.get(i));
            addLineInter(ans.get(i), ans.get(i + 1));
        }
        //штриховка
        ArrayList<Double> ext = getExtremum(ans);
        double minX = ext.get(0), minY = ext.get(1),
                maxX = ext.get(2), maxY = ext.get(3);
        for (double i = minY + minX; i <= maxY + maxX; i += 0.3) {
            double a = 1, b = 1, c = -i;
            Vector2d firstPoint=null; Vector2d secondPoint=null;
            for (int j = 0; j < ans.size() - 1; j++) {
                Line l = new Line(ans.get(j), ans.get(j + 1));
                if (l.isInterLines(a, b, c)) {
                    if (firstPoint == null) {
                        firstPoint = l.getInterLines(a, b, c);
                    }
                    else{
                        secondPoint = l.getInterLines(a, b, c);
                    }
                }
            }
            if (firstPoint != null && secondPoint != null) {
                addLineHatching(firstPoint, secondPoint);
            }

        }
        PanelLog.info( text: "Ответ показан на экране. Площадь такого пересечения "
                + getAreaIntersRect(ansRectF, ansRectS,  mode: 2));

    } else {
        PanelLog.info( text: "Пересечений прямоугольников нет\n");
    }

    // задача решена
    solved = true;
}
```

Приложение Д. Task.getIntersRect()

```java
/**
 * нахождение фигуры пересечения
 */
3 usages  frolovts.24 *
public ArrayList<Vector2d> getIntersRect(Rectangle first, Rectangle second, int mode) {
    ArrayList<Vector2d> ans = new ArrayList<>();
    for (Line lineF : first.getListLines()) {
        for (Line lineS : second.getListLines()) {
            if (lineF.isIntersLines(lineS) &&
                    lineF.getIntersLines(lineS).isBelongSegment(lineF, lineS)) {
                ans.add(lineF.getIntersLines(lineS));
                if (mode == 1) {
                    PanelLog.info( text: "Точка пересечения "
                            + lineF.getIntersLines(lineS) + " обработана\n");
                    addPoint(lineF.getIntersLines(lineS));
                }
            }
        }
    }
    for (Vector2d point : first.getListPoints()) {
        if (point.isIntersRect(second)) {
            ans.add(point);
            if (mode == 1) {
                PanelLog.info( text: "Точка объединения " + point + " обработана\n");
                addPoint(point);
            }
        }
    }
    for (Vector2d point : second.getListPoints()) {
        if (point.isIntersRect(first)) {
            ans.add(point);
            if (mode == 1) {
                PanelLog.info( text: "Точка объединения " + point + " обработана\n");
                addPoint(point);
            }
        }
    }
    PanelLog.info( text: "индексировано пересечение " + ans);
    return sortForCF(ans);
}
```

Приложение E. Task.sortForCF()

```java
/**
 * сортировка точек для выпуклой фигуры
 *
 * @param mass список точек
 * @return отсортированный список
 */
3 usages  frolovts.24 *
public ArrayList<Vector2d> sortForCF(ArrayList<Vector2d> mass) {
    ArrayList<Vector2d> ans = new ArrayList<>();
    if (mass.size() != 0) {
        Vector2d first = mass.get(0);
        for (Vector2d point : mass) {
            if (point.x < first.x ||
                    point.x == first.x && point.y > first.y) {
                first = point;
            }
        }
        ans.add(first);
        mass.remove(first);
        while (mass.size() != 0) {
            Vector2d Point = mass.get(0);
            for (Vector2d point : mass) {
                double a1 = Math.atan((point.x - first.x) /
                        (first.y - point.y));
                double a2 = Math.atan((Point.x - first.x) /
                        (first.y - Point.y));
                if (a1 < 0) {
                    a1 += Math.PI;
                }
                if (a2 < 0) {
                    a2 += Math.PI;
                }
                if (a1 < a2) {
                    Point = point;
                }
            }
            ans.add(Point);
            mass.remove(Point);
        }
    }
    return ans;
}
```

Приложение Ж. UnitTest.java

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

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

/**
 * Класс тестирования
 */
 frolovts.24
public class UnitTest {


    frolovts.24
    @Test
    public void test1() {


        Line l1 = new Line(new Vector2d( x: 1,  y: -4), new Vector2d( x: 1,  y: 2));
        Line l2 = new Line(new Vector2d( x: 0,  y: 0), new Vector2d( x: 2,  y: 2));

        Vector2d i = l1.getIntersLines(l2);

        assert Math.abs(i.x-1) < 0.001 && Math.abs(i.y-1) < 0.001;
    }
```

```java
@Test
public void test2() {


    Line l1 = new Line(new Vector2d( x: 1,  y: 1), new Vector2d( x: 2,  y: 2));
    Line l2 = new Line(new Vector2d( x: 2,  y: 1), new Vector2d( x: 1,  y: 2));

    Vector2d i = l1.getIntersLines(l2);

    assert Math.abs(i.x-1.5) < 0.001 && Math.abs(i.y-1.5) < 0.001;
}
```

frolovts.24

```java
@Test
public void test3() {



    Vector2d p = new Vector2d( x: 0,  y: 0);
    Line l = new Line(new Vector2d( x: 2,  y: 1), new Vector2d( x: 1,  y: 2));

    double x = l.getDistance(p);

    assert Math.abs(x-2.1213) < 0.001;
}
```

frolovts.24

```java
@Test
public void test4() {



    Line l1 = new Line(new Vector2d( x: 0,  y: 2), new Vector2d( x: 5,  y: 0));
    Line l2 = new Line(new Vector2d( x: 0,  y: 0), new Vector2d( x: 8,  y: 4));


    Vector2d x= l1.getIntersLines(l2);

    assert Math.abs(x.x - 2.2222) < 0.001 && Math.abs(x.y - 1.1111) < 0.001;
}
```

```java
@Test
public void test5() {



    Vector2d p = new Vector2d( x: 0,  y: 0);
    Rectangle rectangle = new Rectangle(new Vector2d( x: -2,  y: -1),
            new Vector2d( x: -2,  y: 3), new Vector2d( x: 5,  y: 3), new Vector2d( x: 5,  y: -1),
            new Line(new Vector2d( x: -2,  y: -1), new Vector2d( x: -2,  y: 3)),
            new Line(new Vector2d( x: -2,  y: 3), new Vector2d( x: 5,  y: 3)),
            new Line(new Vector2d( x: 5,  y: 3), new Vector2d( x: 5,  y: -1)),
            new Line(new Vector2d( x: 5,  y: -1), new Vector2d( x: -2,  y: -1)));

    assert p.isIntersRect(rectangle);
}
```

frolovts.24 *

```java
@Test
public void test6() {



    Vector2d p = new Vector2d( x: 0,  y: 0);
    Rectangle rect = new Rectangle(new Vector2d( x: -0.39,  y: -2.19), new Vector2d( x: 3.89,  y: 0.27),
            new Vector2d( x: 2.20,  y: 3.22), new Vector2d( x: -2.08,  y: 0.76),
            new Line(new Vector2d( x: -0.39,  y: -2.19), new Vector2d( x: 3.89,  y: 0.27)),
            new Line(new Vector2d( x: 3.89,  y: 0.27), new Vector2d( x: 2.20,  y: 3.22)),
            new Line(new Vector2d( x: 2.20,  y: 3.22), new Vector2d( x: -2.08,  y: 0.76)),
            new Line(new Vector2d( x: -2.08,  y: 0.76), new Vector2d( x: -0.39,  y: -2.19)));



    assert p.isIntersRect(rect);
}
```

```java
@Test
public void test7() {



    Rectangle rectangle1 = new Rectangle(new Vector2d( x: 3.69,  y: 1.46),
            new Vector2d( x: 2.65,  y: -3.18), new Vector2d( x: -0.78,  y: -2.41), new Vector2d( x: 0.26,  y: 2.23),
            new Line(new Vector2d( x: 3.69,  y: 1.46), new Vector2d( x: 2.65,  y: -3.18)),
            new Line(new Vector2d( x: 2.65,  y: -3.18), new Vector2d( x: -0.78,  y: -2.41)),
            new Line(new Vector2d( x: -0.78,  y: -2.41), new Vector2d( x: 0.26,  y: 2.23)),
            new Line(new Vector2d( x: 0.26,  y: 2.23), new Vector2d( x: 3.69,  y: 1.46)));

    Rectangle rectangle2 = new Rectangle(new Vector2d( x: 3.04,  y: -5.37), new Vector2d( x: 0.17,  y: -2.22),
            new Vector2d( x: 2.38,  y: -0.20), new Vector2d( x: 5.25,  y: -3.35),
            new Line(new Vector2d( x: 3.04,  y: -5.37), new Vector2d( x: 0.17,  y: -2.22)),
            new Line(new Vector2d( x: 0.17,  y: -2.22), new Vector2d( x: 2.38,  y: -0.20)),
            new Line(new Vector2d( x: 2.38,  y: -0.20), new Vector2d( x: 5.25,  y: -3.35)),
            new Line(new Vector2d( x: 5.25,  y: -3.35), new Vector2d( x: 3.04,  y: -5.37)));



    Task task = new Task(new CoordinateSystem2d( minX: 10,  minY: 10,  sizeX: 20,  sizeY: 20),
            new ArrayList<>(), new ArrayList<>(), new ArrayList<>(), new ArrayList<>());

    assert Math.abs(task.getAreaIntersRect(rectangle1, rectangle2,  mode: 2)- 4.98297) < 0.001;
}
```

```java
@Test
public void test8() {


    Rectangle rectangle1 = new Rectangle(new Vector2d( x: 3.69,  y: 1.46), new Vector2d( x: 2.65,  y: -3.18),
            new Vector2d( x: -0.78,  y: -2.41), new Vector2d( x: 0.26,  y: 2.23),
            new Line(new Vector2d( x: 3.69,  y: 1.46), new Vector2d( x: 2.65,  y: -3.18)),
            new Line(new Vector2d( x: 2.65,  y: -3.18), new Vector2d( x: -0.78,  y: -2.41)),
            new Line(new Vector2d( x: -0.78,  y: -2.41), new Vector2d( x: 0.26,  y: 2.23)),
            new Line(new Vector2d( x: 0.26,  y: 2.23), new Vector2d( x: 3.69,  y: 1.46)));

    Rectangle rectangle2 = new Rectangle(new Vector2d( x: 3.04,  y: -5.37), new Vector2d( x: 0.17,  y: -2.22),
            new Vector2d( x: 2.38,  y: -0.20), new Vector2d( x: 5.25,  y: -3.35),
            new Line(new Vector2d( x: 3.04,  y: -5.37), new Vector2d( x: 0.17,  y: -2.22)),
            new Line(new Vector2d( x: 0.17,  y: -2.22), new Vector2d( x: 2.38,  y: -0.20)),
            new Line(new Vector2d( x: 2.38,  y: -0.20), new Vector2d( x: 5.25,  y: -3.35)),
            new Line(new Vector2d( x: 5.25,  y: -3.35), new Vector2d( x: 3.04,  y: -5.37)));

    Rectangle rectangle3 = new Rectangle(new Vector2d( x: -1.56,  y: -0.60), new Vector2d( x: -1.16,  y: 4.13),
            new Vector2d( x: 5.84,  y: 3.54), new Vector2d( x: 5.44,  y: -1.19),
            new Line(new Vector2d( x: -1.56,  y: -0.60), new Vector2d( x: -1.16,  y: 4.13)),
            new Line(new Vector2d( x: -1.16,  y: 4.13), new Vector2d( x: 5.84,  y: 3.54)),
            new Line(new Vector2d( x: 5.84,  y: 3.54), new Vector2d( x: 5.44,  y: -1.19)),
            new Line(new Vector2d( x: 5.44,  y: -1.19), new Vector2d( x: -1.56,  y: -0.60)));

    ArrayList<Point> points = new ArrayList<>();

    points.add(new Point(new Vector2d( x: 3.69,  y: 1.46)));
    points.add(new Point(new Vector2d( x: 2.65,  y: -3.18)));
    points.add(new Point(new Vector2d( x: -0.78,  y: -2.41)));
    points.add(new Point(new Vector2d( x: 0.26,  y: 2.23)));

    points.add(new Point(new Vector2d( x: 3.04,  y: -5.37)));
    points.add(new Point(new Vector2d( x: 0.17,  y: -2.22)));
    points.add(new Point(new Vector2d( x: 2.38,  y: -0.20)));
    points.add(new Point(new Vector2d( x: 5.25,  y: -3.35)));

    points.add(new Point(new Vector2d( x: -1.56,  y: -0.60)));
    points.add(new Point(new Vector2d( x: -1.16,  y: 4.13)));
    points.add(new Point(new Vector2d( x: 5.84,  y: 3.54)));
    points.add(new Point(new Vector2d( x: 5.44,  y: -1.19)));
```

```
        ArrayList<Line> lines = new ArrayList<>();
        lines.add(new Line(new Vector2d( x: 3.69,  y: 1.46), new Vector2d( x: 2.65,  y: -3.18)));
        lines.add(new Line(new Vector2d( x: 2.65,  y: -3.18), new Vector2d( x: -0.78,  y: -2.41)));
        lines.add(new Line(new Vector2d( x: -0.78,  y: -2.41), new Vector2d( x: 0.26,  y: 2.23)));
        lines.add(new Line(new Vector2d( x: 0.26,  y: 2.23), new Vector2d( x: 3.69,  y: 1.46)));

        lines.add(new Line(new Vector2d( x: 3.04,  y: -5.37), new Vector2d( x: 0.17,  y: -2.22)));
        lines.add(new Line(new Vector2d( x: 0.17,  y: -2.22), new Vector2d( x: 2.38,  y: -0.20)));
        lines.add(new Line(new Vector2d( x: 2.38,  y: -0.20), new Vector2d( x: 5.25,  y: -3.35)));
        lines.add(new Line(new Vector2d( x: 5.25,  y: -3.35), new Vector2d( x: 3.04,  y: -5.37)));

        lines.add(new Line(new Vector2d( x: -1.56,  y: -0.60), new Vector2d( x: -1.16,  y: 4.13)));
        lines.add(new Line(new Vector2d( x: -1.16,  y: 4.13), new Vector2d( x: 5.84,  y: 3.54)));
        lines.add(new Line(new Vector2d( x: 5.84,  y: 3.54), new Vector2d( x: 5.44,  y: -1.19)));
        lines.add(new Line(new Vector2d( x: 5.44,  y: -1.19), new Vector2d( x: -1.56,  y: -0.60)));

        ArrayList<Rectangle> rectangles = new ArrayList<>();
        rectangles.add(rectangle1);
        rectangles.add(rectangle2);
        rectangles.add(rectangle3);

        Task task = new Task(new CoordinateSystem2d( minX: 10,  minY: 10,  sizeX: 20,  sizeY: 20), points,
                lines, rectangles, new ArrayList<>());

        assert Math.abs(task.getAreaIntersRect(rectangle1, rectangle2,  mode: 2) - 4.98297) < 0.001;

        assert Math.abs(task.getAreaIntersRect(rectangle2, rectangle3,  mode: 2) - 0.53269) < 0.001;

        assert Math.abs(task.getAreaIntersRect(rectangle1, rectangle3,  mode: 2) - 9.7005) < 0.005;

        ArrayList<Vector2d> res = new ArrayList<>();
        res.add(new Vector2d( x: -0.396,  y: -0.698));
        res.add(new Vector2d( x: 3.139,  y: -0.997));
        res.add(new Vector2d( x: 3.69,  y: 1.46));
        res.add(new Vector2d( x: 0.26,  y: 2.23));
        for (int i = 0; i < 4; i++) {
            assert Math.abs(res.get(i).x - task.solve1().get(i).x) < 0.001;
            assert Math.abs(res.get(i).y - task.solve1().get(i).y) < 0.001;
        }
    }

}
```