**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 10-3 класса

Белхассен А.

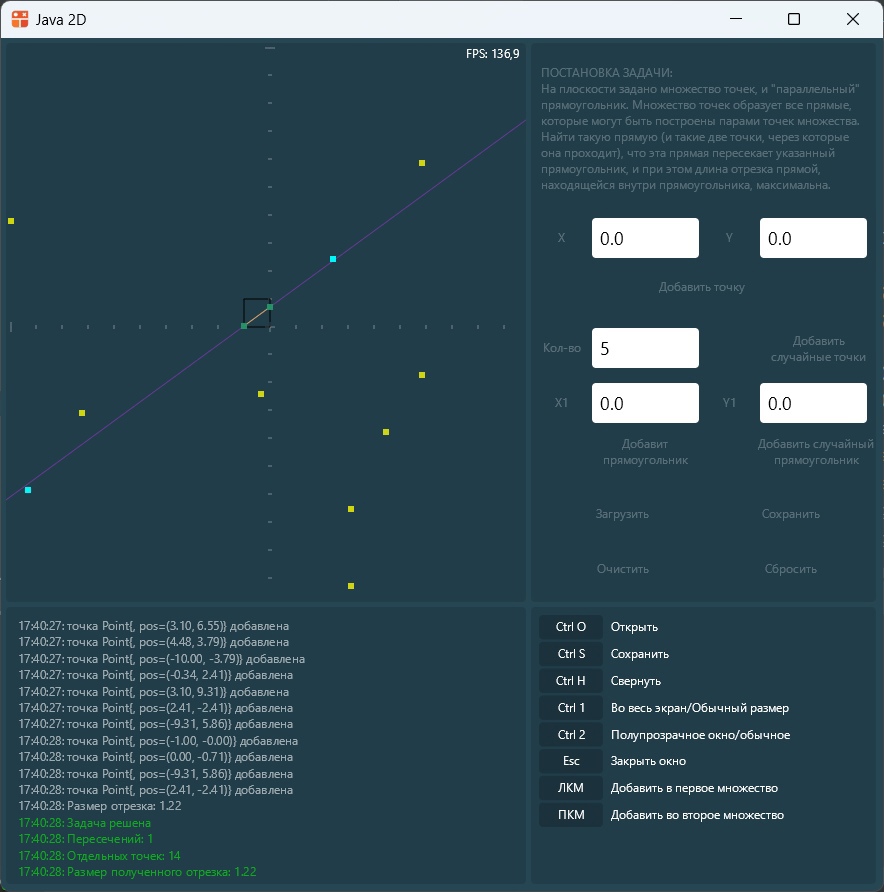
Преподаватель:

Клюнин А.О.

Санкт-Петербург – 2023 год

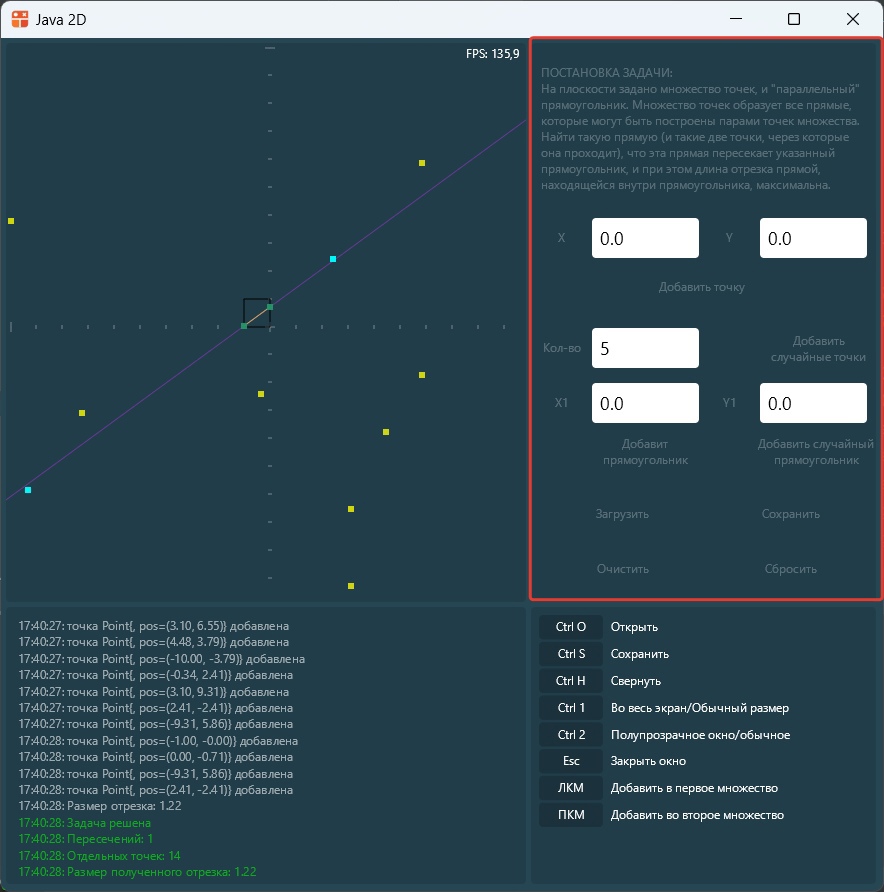
# 1. Постановка задачи

Задано множество точек, и "параллельный" прямоугольник. Множество точек образует все прямые, которые могут быть построены парами точек множества. Найти прямую (и такие две точки, через которые она проходит), что эта прямая пересекает указанный прямоугольник, и при этом длина отрезка прямой, находящейся внутри прямоугольника, максимальна. Требуется выделить найденные две точки, нарисовать прямую, которая через них проходит, а также выделить на этой прямой отрезок между двумя найденными точками пересечения.



# 2. Элементы управления

В рамках данной задачи необходимо было реализовать следующие элементы управления:



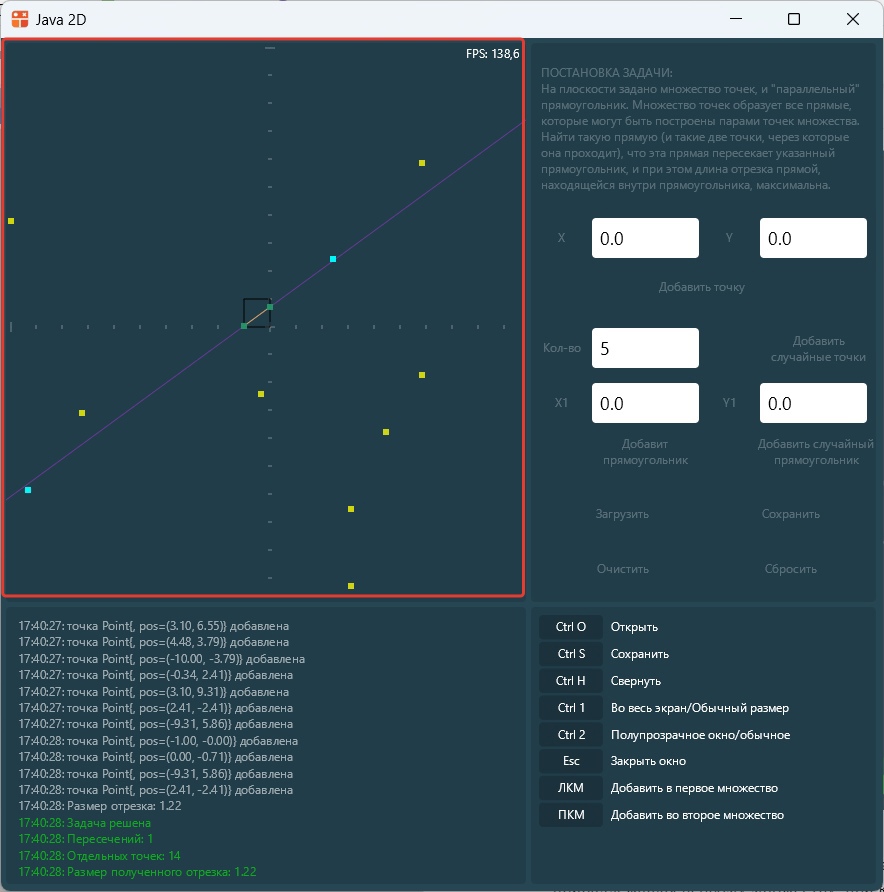
Для добавления точки по координатам было создано два поля ввода: «X» и «Y».

Т.к. задача предполагает множество точек, то следует создать поле для добавления случайных точек. В него вводится количество, которое будет добавлено.

Для добавления «параллельного» прямоугольника по координатам (задаются с помощью двух противоположных вершин) было создано два дополнительных поля ввода: «X1» и «Y1», так как противоположная вершина задается в уже существующих «X» и «Y».

Также предусмотрено создание случайного «параллельного» прямоугольника с помощью ещё одной кнопки.

Программа позволяет добавлять точки с помощью клика левой мышью по области рисования



При клике правой кнопкой мыши по области рисования в 2 местах, создаётся «параллельный» прямоугольник противоположными вершинами которого являются выбранные точки.

# 3. Структуры данных

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

В него были добавлены поля **Vector2d pos**, соответствующее положению точки в пространстве задачи**.**

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

В него были добавлены поля **Point а** и **Point с**, которые задают противоположные вершины в «параллельном» прямоугольнике.

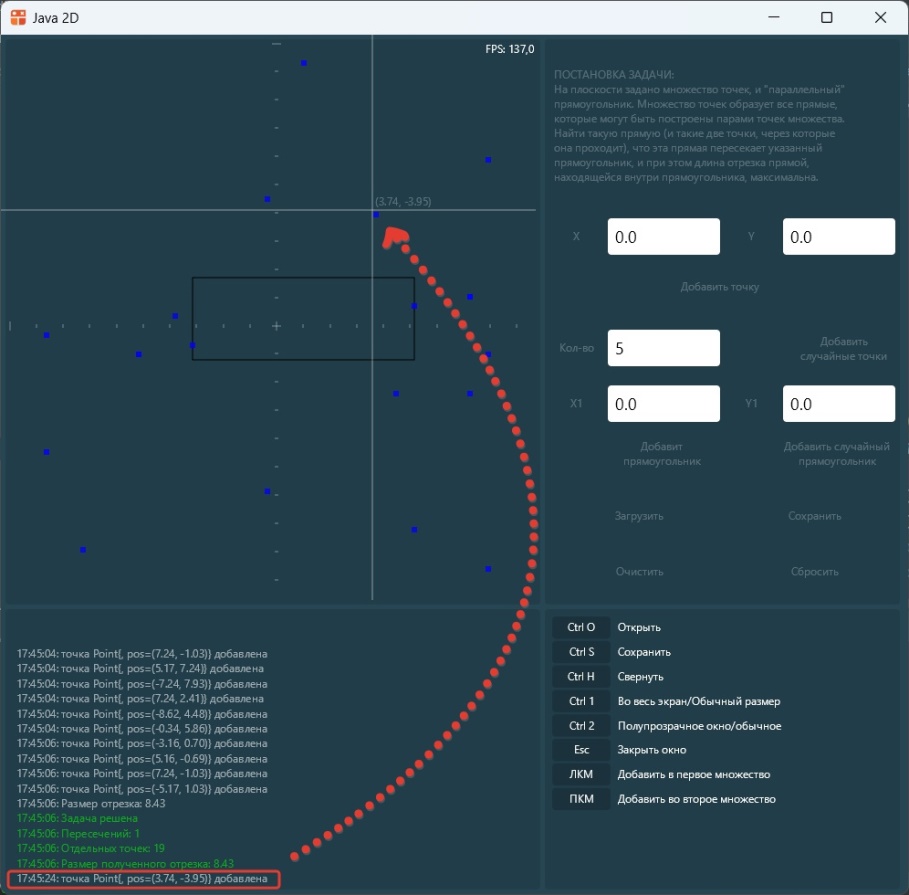
Для того чтобы хранить прямую, был разработан класс **Line.java.**

Его листинг приведён в приложении В.

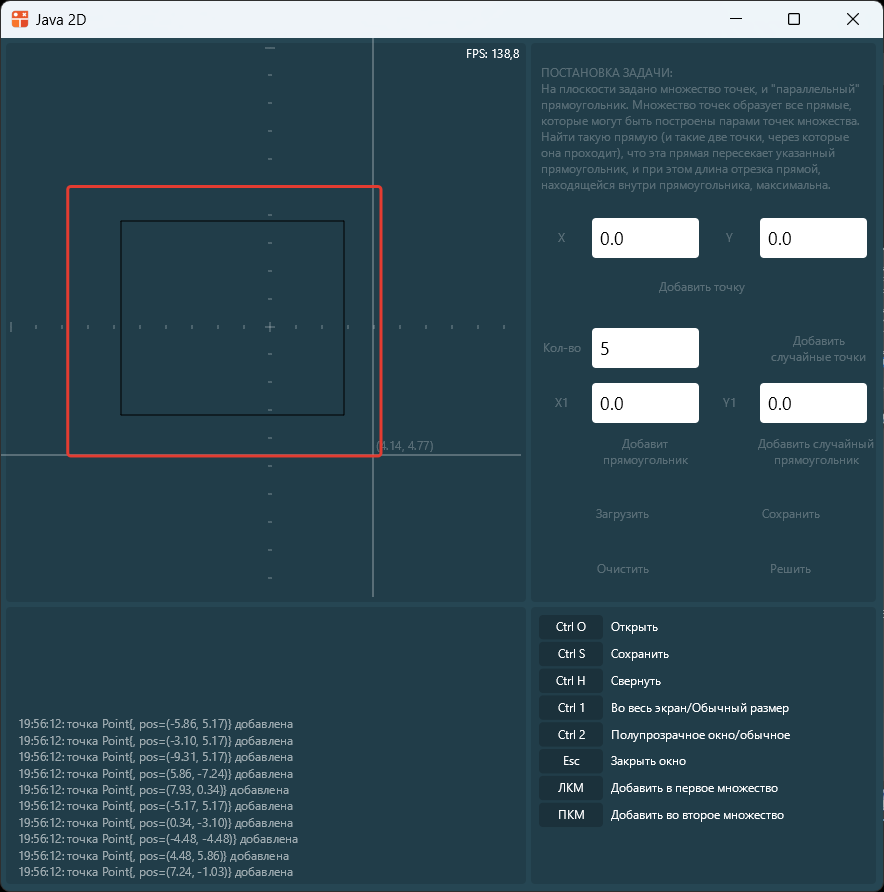
В него были добавлены поля **Vector2d** **a** и **Vector2d b**, которые задают прямую двумя точками, которые мы задали, как вектора.

# 4. Рисование

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

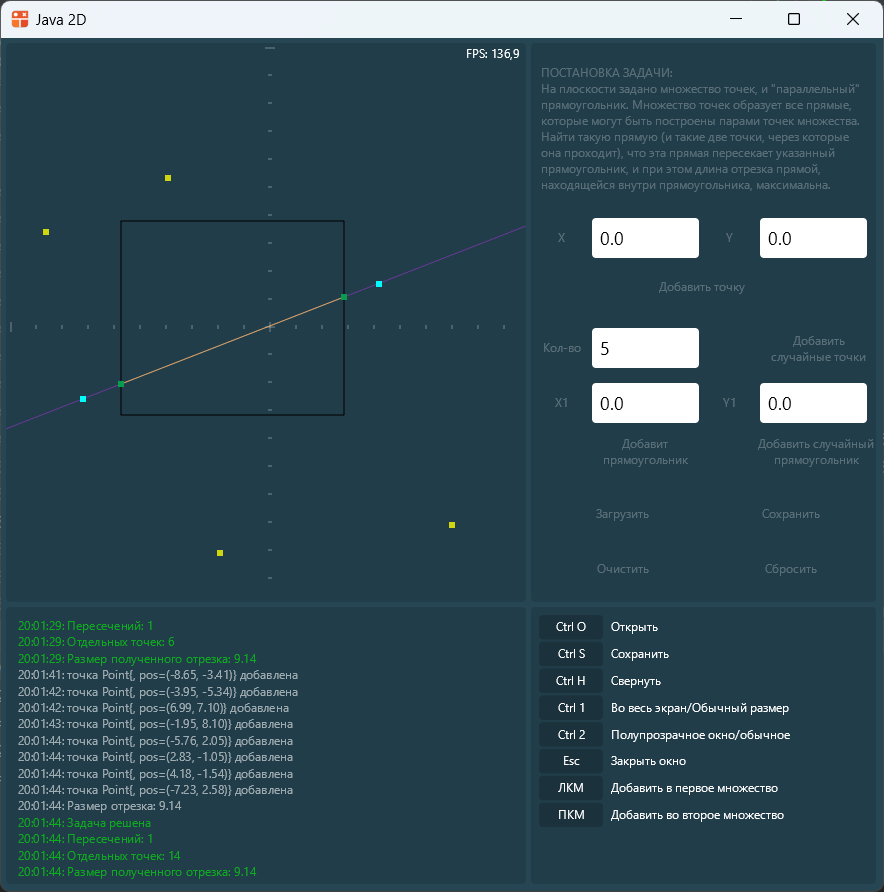


Чтобы нарисовать прямоугольник, использовалась команда рисования прямоугольников **paint(CoordinateSystem2i windowCS, CoordinateSystem2d taskCS, Canvas canvas).**



Чтобы нарисовать отрезок, использовалась команда рисования линии **canvas.drawLine(a.x, a.y, b.x, b.y, paint),** где a и b вектора, задающие точки, которые являются концами отрезка.

Чтобы нарисовать прямую, использовалась команда рисования **renderLine(canvas, windowCS).**



# 5. Решение задачи

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

public void solve() {  
 // очищаем списки  
 crossed.clear();  
 single.clear();  
 Point p1 = rect.a;  
 Point p2 = new Point(new Vector2d(rect.a.pos.x, rect.c.pos.y));  
 Point p3 = rect.c;  
 Point p4 = new Point(new Vector2d(rect.c.pos.x, rect.a.pos.y));  
 Point middle = new Point(new Vector2d((rect.a.pos.x + rect.c.pos.x) / 2, (rect.a.pos.y + rect.a.pos.x) / 2));  
 // перебираем пары точек  
 if (p1.pos.x == p3.pos.x || p1.pos.y == p3.pos.y) {  
 solved = false;  
 return;  
 }  
 while (p1.pos.x < p3.pos.x || p1.pos.y < p3.pos.y) {  
 Point temp = p1;  
 p1 = p2;  
 p2 = p3;  
 p3 = p4;  
 p4 = temp;  
 }  
 if (p4.pos.y > p2.pos.y) {  
 Point temp = p2;  
 p2 = p4;  
 p4 = temp;  
 }  
 int m = points.size();  
 *length* = 0;  
 for (int i = 0; i < m - 1; i++) {  
 for (int j = i + 1; j < m; j++) {  
 // сохраняем точки  
 Point a = points.get(i);  
 Point b = points.get(j);  
 Point c, d;  
 if (!(*line*(a, b, p1, p3) == (null)) && (*line*(a, b, p2, p4) == (null))) {  
 Point n = new Point(Objects.*requireNonNull*(*line*(a, b, p1, p3)).getPos());  
 if (n.pos.x > middle.pos.x) {  
 c = new Point(new Vector2d(*line*(a, b, p1, p2).pos.x, *line*(a, b, p1, p2).pos.y));  
 d = new Point(new Vector2d(*line*(a, b, p1, p4).pos.x, *line*(a, b, p1, p4).pos.y));  
// PanelRendering.task.addPoint(new Vector2d(line(a, b, p1, p2).pos.x, line(a, b, p1, p2).pos.y));  
// PanelRendering.task.addPoint(new Vector2d(line(a, b, p1, p4).pos.x, line(a, b, p1, p4).pos.y));  
 } else {  
 c = new Point(new Vector2d(*line*(a, b, p3, p2).pos.x, *line*(a, b, p3, p2).pos.y));  
 d = new Point(new Vector2d(*line*(a, b, p3, p4).pos.x, *line*(a, b, p3, p4).pos.y));  
// PanelRendering.task.addPoint(new Vector2d(line(a, b, p3, p2).pos.x, line(a, b, p3, p2).pos.y));  
// PanelRendering.task.addPoint(new Vector2d(line(a, b, p3, p4).pos.x, line(a, b, p3, p4).pos.y));  
 }  
 if (*distance*(c, d) > *length*) {  
 *max*[0] = c;  
 *max*[1] = d;  
 *max*[2] = a;  
 *max*[3] = b;  
 *length* = *distance*(c, d);  
 }  
 } else if ((*line*(a, b, p1, p3) == (null)) && !(*line*(a, b, p2, p4) == (null))) {  
 Point n = new Point(Objects.*requireNonNull*(*line*(a, b, p2, p4)).getPos());  
 if (n.pos.x > middle.pos.x) {  
 c = new Point(new Vector2d(*line*(a, b, p3, p4).pos.x, *line*(a, b, p3, p4).pos.y));  
 d = new Point(new Vector2d(*line*(a, b, p1, p4).pos.x, *line*(a, b, p1, p4).pos.y));  
// PanelRendering.task.addPoint(new Vector2d(line(a, b, p3, p4).pos.x, line(a, b, p3, p4).pos.y));  
// PanelRendering.task.addPoint(new Vector2d(line(a, b, p1, p4).pos.x, line(a, b, p1, p4).pos.y));  
 } else {  
 c = new Point(new Vector2d(*line*(a, b, p3, p2).pos.x, *line*(a, b, p3, p2).pos.y));  
 d = new Point(new Vector2d(*line*(a, b, p1, p2).pos.x, *line*(a, b, p1, p2).pos.y));  
// PanelRendering.task.addPoint(new Vector2d(line(a, b, p3, p2).pos.x, line(a, b, p3, p2).pos.y));  
// PanelRendering.task.addPoint(new Vector2d(line(a, b, p1, p2).pos.x, line(a, b, p1, p2).pos.y));  
 }  
 if (*distance*(c, d) > *length*) {  
 *max*[0] = c;  
 *max*[1] = d;  
 *max*[2] = a;  
 *max*[3] = b;  
 *length* = *distance*(c, d);  
 }  
 } else if (!(*line*(a, b, p1, p3) == (null)) && !(*line*(a, b, p2, p4) == (null))) {  
 if ((*line*(a, b, p1, p2) == null)) {  
 c = new Point(new Vector2d(*line*(a, b, p3, p2).pos.x, *line*(a, b, p3, p2).pos.y));  
 d = new Point(new Vector2d(*line*(a, b, p1, p4).pos.x, *line*(a, b, p1, p4).pos.y));  
// PanelRendering.task.addPoint(new Vector2d(line(a, b, p3, p2).pos.x, line(a, b, p3, p2).pos.y));  
// PanelRendering.task.addPoint(new Vector2d(line(a, b, p1, p4).pos.x, line(a, b, p1, p4).pos.y));  
 } else {  
 c = new Point(new Vector2d(*line*(a, b, p3, p4).pos.x, *line*(a, b, p3, p4).pos.y));  
 d = new Point(new Vector2d(*line*(a, b, p1, p2).pos.x, *line*(a, b, p1, p2).pos.y));  
// PanelRendering.task.addPoint(new Vector2d(line(a, b, p3, p4).pos.x, line(a, b, p3, p4).pos.y));  
// PanelRendering.task.addPoint(new Vector2d(line(a, b, p1, p2).pos.x, line(a, b, p1, p2).pos.y));  
 }  
 if (*distance*(c, d) > *length*) {  
 *max*[0] = c;  
 *max*[1] = d;  
 *max*[2] = a;  
 *max*[3] = b;  
 *length* = *distance*(c, d);  
 }  
 }  
 }  
 }  
 *length* = ((double)Math.*round*(*length*\*100))/100;  
 PanelRendering.*task*.addPoint(*max*[0].getPos());  
 PanelRendering.*task*.addPoint(*max*[1].getPos());  
 PanelRendering.*task*.addPoint(*max*[3].getPos());  
 PanelRendering.*task*.addPoint(*max*[2].getPos());  
 line = new Line(new Vector2d(*max*[0].pos.x, *max*[0].pos.y), new Vector2d(*max*[1].pos.x, *max*[1].pos.y), this);  
 PanelLog.*info*("Размер отрезка: "+ *length*);  
 System.*out*.println(*max*[0] + " " + *max*[1]);  
 lines.add(*max*[0]);  
 lines.add(*max*[1]);  
 crossed.add(*max*[2]);  
 crossed.add(*max*[3]);  
 System.*out*.println(crossed);  
 /// добавляем вс  
 for (Point point : points)  
 if (!crossed.contains(point) || !lines.contains(point))  
 single.add(point);  
  
 // задача решена  
 solved = true;  
 }

В начале решения мы «правильно» ориентируем прямоугольник, чтобы точка **p1** являлась правой нижней точкой в прямоугольнике и **p3** – вершина напротив, **p2** – левой нижней,  **p4** – напротив неё. Это мы делаем для того, чтобы не разбирать случаи.

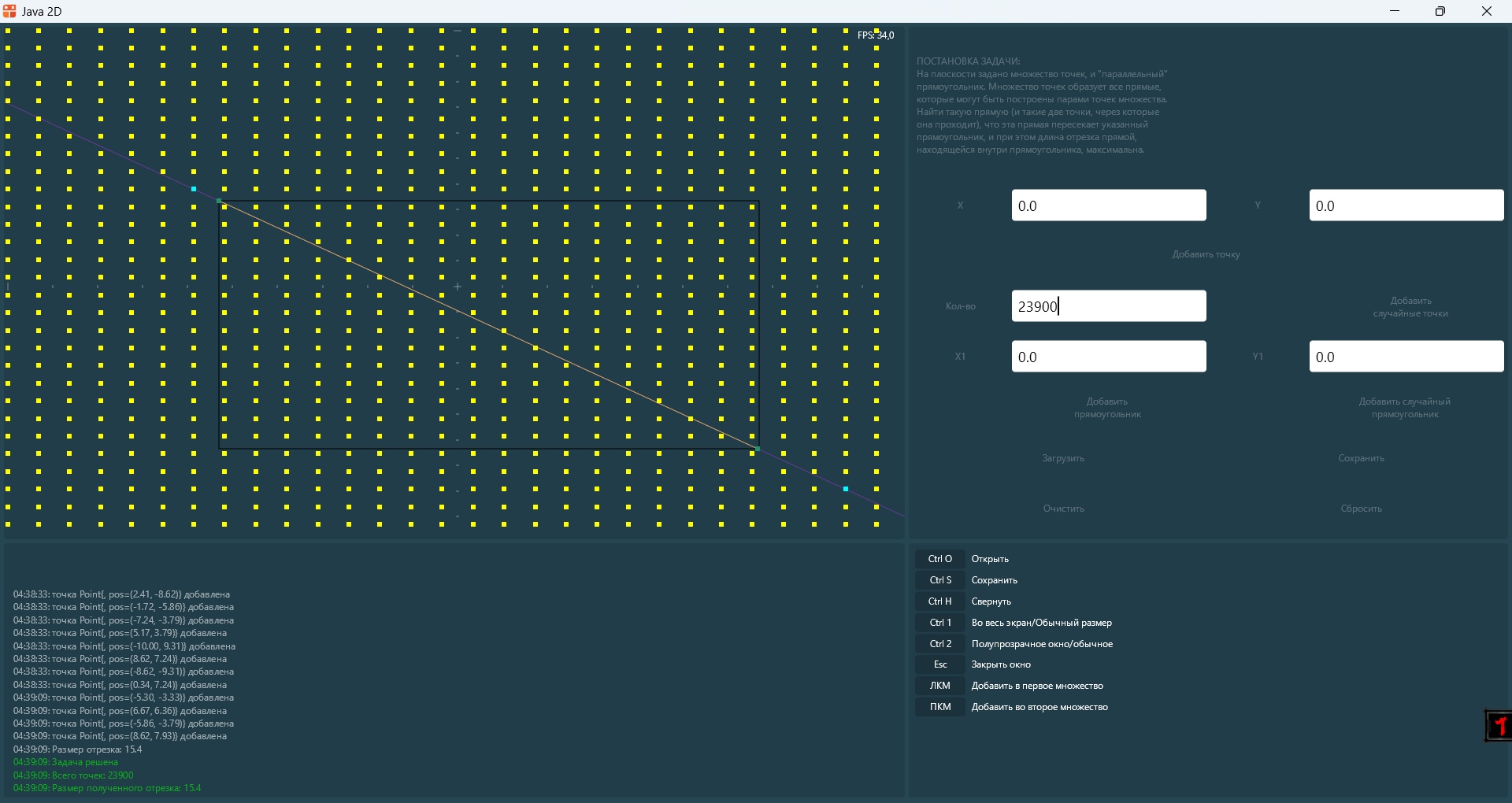
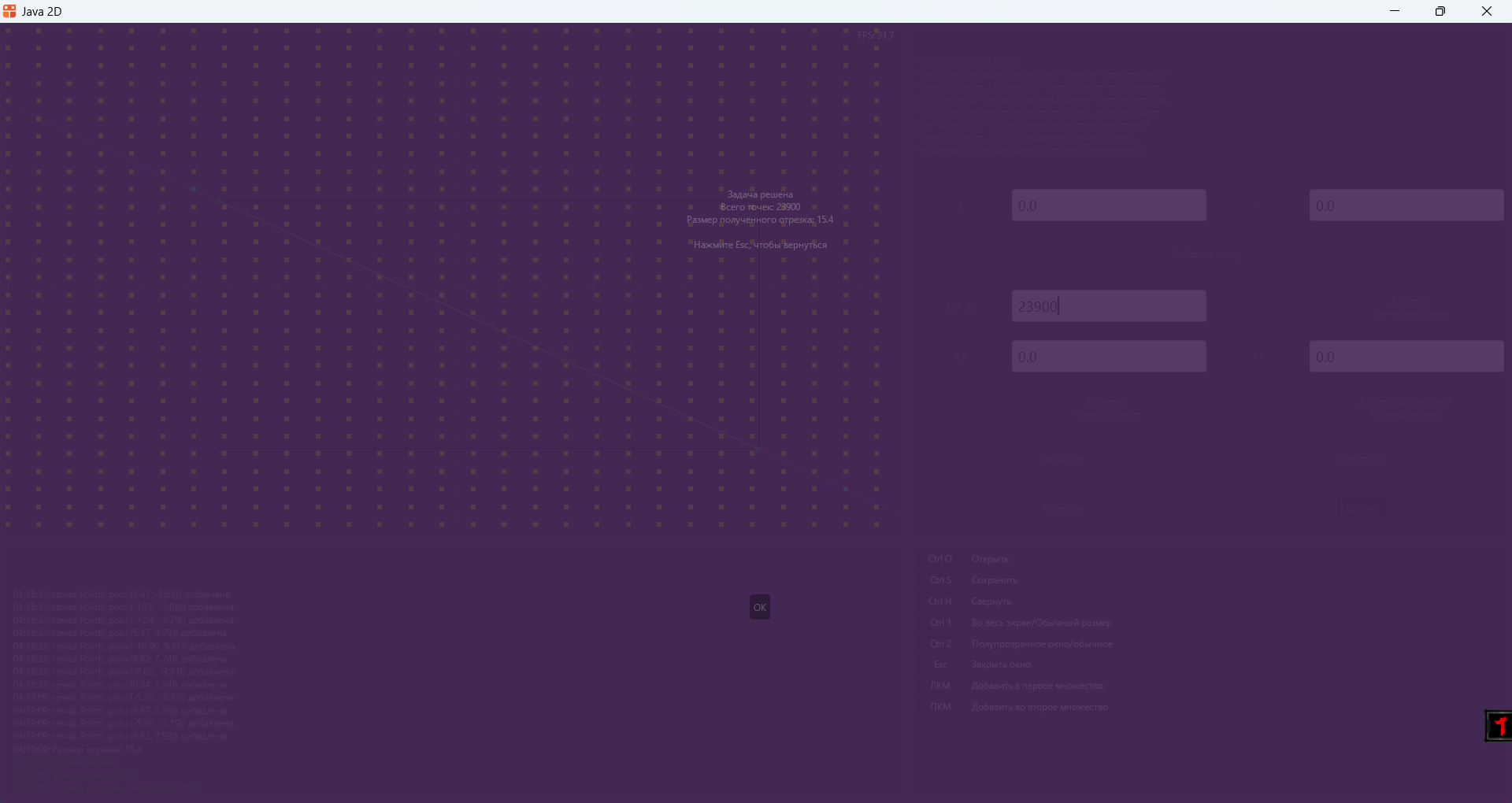
Далее мы перебираем наш записанный заранее массив точек и для каждой пары смотрим, есть ли у линии, проведённой через 2 точки, пересечения с диагоналями прямоугольника с помощью функции **line**(касание не считается пересечением, так как в задаче чётко сказано, что отрезок лежит внутри прямоугольника). Если их нет, то нет и точек пересечения с самим прямоугольником. По пересечению с определёнными диагоналями и сравнением координаты центра прямоугольника с точкой пересечения можно понять, какие стороны прямоугольника пересекает данная линия. С помощью функции **distance** проверяем расстояние между этими точками, и если оно больше, чем предыдущее максимальное, мы сохраняем это расстояние в публичную переменную **length**, точки пересечения записываем в массив типа **Point: max,** где элементы под индексом **0** и **1** – точки пересечения линии с прямой, а эл-ты под индексом **2** и **3** – точки, через которые была проведена линия. Если прямоугольник вырожденный, то мы ничего не решаем.

# 6. Проверка

Для проверки правильности решённой задачи не было разработано никаких unit-тестов, но было многократно проведено множество проверок программы с помощью ручного ввода точек и прямоугольника / случайной генерации точек и прямоугольника / считывания с файла точек и прямоугольника.

Также был проведён стресс-тест программы. Она перестаёт работать, когда увеличение поля составляет **х1000** от начального размера, нижней границы нет. При количестве точек **23900** программа нашла лучший отрезок спустя всего лишь **29.8 секунд**, что, несомненно, является хорошим результатом, данный результат был записан в





# 7. Заключение

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

# Приложение А. Point.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.ArrayList;  
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. MyRect.java

package app;  
  
import io.github.humbleui.skija.Canvas;  
import io.github.humbleui.skija.Paint;  
import misc.CoordinateSystem2d;  
import misc.CoordinateSystem2i;  
import misc.Vector2i;  
  
import java.util.Objects;  
  
public class MyRect {  
 Point a;  
 Point c;  
  
 public MyRect(Point a, Point c) {  
 this.a = a;  
 this.c = c;  
 }  
  
 void paint(CoordinateSystem2i windowCS, CoordinateSystem2d taskCS, Canvas canvas){  
 // создаём перо  
 try (var p = new Paint()) {  
 // левая верхняя вершина  
 Vector2i pointA = windowCS.getCoords(a.pos,taskCS);  
 // правая нижняя  
 Vector2i pointC = windowCS.getCoords(c.pos,taskCS);  
  
 // рассчитываем опорные точки прямоугольника  
 Vector2i pointB = new Vector2i(pointA.x, pointC.y);  
 Vector2i pointD = new Vector2i(pointC.x, pointA.y);  
  
 // рисуем его стороны  
 canvas.drawLine(pointA.x, pointA.y, pointB.x, pointB.y, p);  
 canvas.drawLine(pointB.x, pointB.y, pointC.x, pointC.y, p);  
 canvas.drawLine(pointC.x, pointC.y, pointD.x, pointD.y, p);  
 canvas.drawLine(pointD.x, pointD.y, pointA.x, pointA.y, p);  
 }  
 }  
 void section(CoordinateSystem2i windowCS, CoordinateSystem2d taskCS, Canvas canvas){  
 // создаём перо  
 try (var p = new Paint()) {  
 // левая верхняя вершина  
 Vector2i pointA = windowCS.getCoords(a.pos,taskCS);  
 // правая нижняя  
 Vector2i pointC = windowCS.getCoords(c.pos,taskCS);  
  
 // рисуем его стороны  
 canvas.drawLine(pointA.x, pointA.y, pointC.x, pointC.y, p);  
 }  
 }  
  
 @Override  
 public boolean equals(Object o) {  
 if (this == o) return true;  
 if (o == null || getClass() != o.getClass()) return false;  
 MyRect myRect = (MyRect) o;  
 return Objects.*equals*(a, myRect.a) && Objects.*equals*(c, myRect.c);  
 }  
  
 @Override  
 public int hashCode() {  
 return Objects.*hash*(a, c);  
 }  
}

# Приложение В. Line.java

package app;  
  
import io.github.humbleui.skija.Canvas;  
import io.github.humbleui.skija.Paint;  
import io.github.humbleui.skija.RRect;  
import io.github.humbleui.skija.Rect;  
import misc.CoordinateSystem2i;  
import misc.Misc;  
import misc.Vector2d;  
import misc.Vector2i;  
  
public class Line {  
 public Vector2d a;  
 public Vector2d b;  
  
 Task task;  
  
 public Line(Vector2d a, Vector2d b, Task task) {  
 this.a = a;  
 this.b = b;  
 this.task = task;  
 }  
  
  
 void renderLine(Canvas canvas, CoordinateSystem2i windowCS) {  
 // опорные точки линии  
 Vector2i pointA = windowCS.getCoords(a, task.getOwnCS());  
 Vector2i pointB = windowCS.getCoords(b, task.getOwnCS());  
 // вектор, ведущий из точки A в точку B  
 Vector2i delta = Vector2i.*subtract*(pointA, pointB);  
 // получаем максимальную длину отрезка на экране, как длину диагонали экрана  
 int maxDistance = (int) windowCS.getSize().length();  
 // получаем новые точки для рисования, которые гарантируют, что линия  
 // будет нарисована до границ экрана  
 Vector2i renderPointA = Vector2i.*sum*(pointA, Vector2i.*mult*(delta, maxDistance));  
 Vector2i renderPointB = Vector2i.*sum*(pointA, Vector2i.*mult*(delta, -maxDistance));  
 // рисуем линию  
 try (Paint p = new Paint()) {  
 p.setColor(Misc.*getColor*(100, 200, 50, 255));  
 canvas.drawLine(renderPointA.x, renderPointA.y, renderPointB.x, renderPointB.y, p);  
 }  
 try (Paint p = new Paint()) {  
 p.setColor(Misc.*getColor*(200, 200, 100, 255));  
 canvas.drawRect(Rect.*makeXYWH*(pointA.x - 3, pointA.y - 3, 6, 6), p);  
 canvas.drawRect(Rect.*makeXYWH*(pointB.x - 3, pointB.y - 3, 6, 6), p);  
 }  
  
 }  
 public Point cross(Line line){  
 double a1 = b.y - a.y;  
 double b1 = a.x - b.x;  
 double c1 = a1\*(a.x) + b1\*(a.y);  
  
 double a2 = line.b.y - line.a.y;  
 double b2 = line.a.x - line.b.x;  
 double c2 = a2\*(line.a.x)+ b2\*(line.a.y);  
  
 double determinant = a1\*b2 - a2\*b1;  
  
 if (determinant == 0)  
 {  
 return null;  
 }  
 else  
 {  
 double x = (b2\*c1 - b1\*c2)/determinant;  
 double y = (a1\*c2 - a2\*c1)/determinant;  
 return new Point(new Vector2d(x, y));  
 }  
  
  
 }  
  
}