**Государственное бюджетное общеобразовательное учреждение "Президентский физико-математический лицей № 239"**

**«Поиск максимального пересечения окружностей и прямоугольников»**

Годовой проект по информатике

Автор: Борисенко Николай, 10-7 класс

Санкт-Петербург

2021

**Содержание**

**Постановка задачи** 3

**Уточнение исходных и выходных данных** 3

**Входные данные** 3

**Выходные данные** 5

**Математическая модель** 5

**Анализ используемой структуры данных** 6

**Выбор метода решения** 7

**Блок-схема алгоритма** 10

**ВНН. Вектора не нужны!** 10

**Комментированный листинг** 11

**Пример работы программы** 12

**Анализ правильности решения** 13

**Будущее проекта** 14

**Постановка задачи**

Задача, значит, вот такая:

На плоскости задано множество прямоугольников и множество окружностей. Найти такую пару прямоугольник-окружность, что фигура, находящаяся внутри прямоугольника и окружности, имеет максимальную площадь. В качестве ответа: выделить найденные прямоугольник и окружность, выделить контур фигуры, которая ограничивает точки внутри найденного прямоугольника и окружности, желательно выделить внутреннее пространство фигуры ("залить цветом").

То есть, взял пользователь мою программку, запустил, задал какие-то окружности и прямоугольники или создал случайные и нажимает кнопочку «решить». Что хочет пользователь увидеть? Увидеть хочет выделенные окружность и прямоугольник, которые дают максимальную площадь пересечения по сравнению с остальными возможными пересечениями других окружностей и прямоугольников. То есть пользователь убеждается в том, что если он возьмёт другие окружность и прямоугольник, изображённые на плоскости, то их площадь пересечения будет меньше результативных.

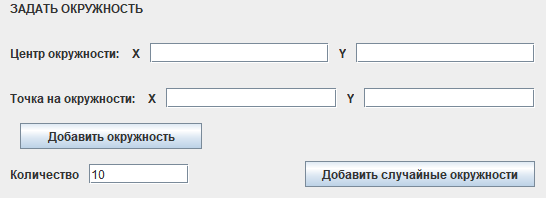
**Уточнение исходных и выходных данных**

**Входные данные**

Есть нюанс - используемые соглашения и определения. Окружность, например, легче, удобней и естественней задать через точку центра и радиус. А теперь читаем соглашения:

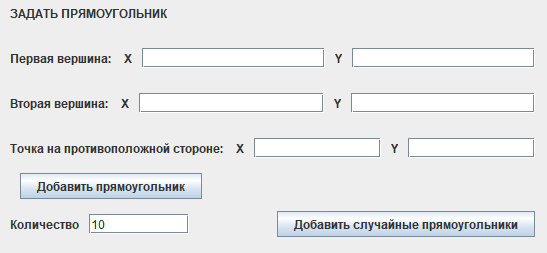
«Окружность. Задаётся точкой центра и точкой на окружности. Точки не совпадают.»

В форме программы реализован второй способ определения окружности:



Аналогично и с прямоугольниками. Удобней - через четыре точки, которые являются вершинами прямоугольника. В соглашении: «Задаётся двумя вершинами одной из сторон. А также точкой, лежащей на прямой, проходящей через две другие вершины.»

В форме программы реализован способ из соглашения:



В связи с этим я реализовал для класса окружности и прямоугольника 2 конструктора, чтобы можно было задавать их обоими способами. Также загрузка из файла осуществляется по двум типам:

1. type 1 - окружность задаётся центром и радиусом, а прямоугольник - через 4 вершины. Пример:

type 1

dSize 20.0

QuadsList

(3,0; 4,0), (5,0; -2,0), (10,1; -0,3), (8,1; 5,7)

(-1,0; -3,0), (-3,0; -4,0), (-6,4; 2,8), (-4,4; 3,8)

(-10,0; -11,0), (-5,0; 12,0), (-9,5; 13,0), (-14,5; -10,0)

(0,1; -15,0), (15,6; -3,3), (18,8; -7,6), (3,3; -19,3)

CirclesList

(5,0; 3,0) R = 1,4

(0,0; -6,0) R = 5,0

(-11,0; -11,0) R = 3,0

(10,0; 3,0) R = 10,0

(6,9; -15,4) R = 0,4

1. type 2 - окружность задаётся через центр и точку, лежащую на окружности, а прямоугольник задаётся двумя вершинами одной из сторон и точкой, лежащей на прямой, проходящей через две другие вершины. То есть тип для соглашения. Пример:

type 2

dSize 30.0

QuadsList

(3,0; 4,0), (5,0; -2,0), (8,1; 5,0)

(-1,0; -3,0), (-3,0; -4,0), (-4,4; 3,2)

(-10,0; -11,0), (-5,0; 12,0), (-14,0; -8,0)

CirclesList

(5,0; 3,0), (-9,0; 3,0)

(0,0; -6,0), (1,0; 13,0)

(-11,0; -11,0), (-2,1; 8,3)

(10,0; 3,0), (12,2; -14,7)

Похожие примеры лежат в текстовом виде в папке с проектом под названиями «solve type 1 example.txt» и «solve type 2 example.txt».

Координаты точек, через которые задаются фигуры, имеют тип double, т. е. диапазон их значений: от ±5,0 \* 10^(-324) до ±1,7 \* 10^308.

Что такое dSize - будет объяснено в главе «Математическая модель», но у величины dSize так же тип double.  
  
**Выходные данные**

Выходные данные - всего одна окружность и один прямоугольник, которые образовывают максимальную площадь пересечения. Эти окружность и прямоугольник находятся среди заданных, а потому на них накладываются те же ограничения. В решении я выдаю окружность и прямоугольник, определяя их по первому типу, потому что в этом меня никто не ограничивает, да и такое определение более удобное и однозначное, на мой взгляд.  
  
**Математическая модель**

1. Точка внутри окружности или на ней.

Для того, чтобы определить, находится ли точка внутри окружности, достаточно подставить её координаты в уравнение окружности.

Пусть я взял на плоскости точку А. Тогда, если точка А лежит внутри окружности или на ней, то верно следующее неравенство:

(xА - x0)2 + (yА - y0)2 ≤ R2

Если точка снаружи окружности, то приведённое выше неравенство не выполняется.

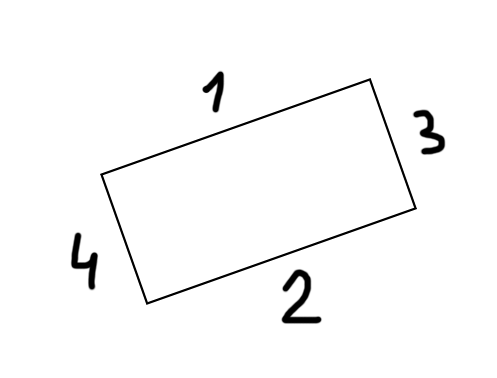
1. Точка внутри прямоугольника или на нём.

Определить, находится ли точка внутри прямоугольника или на нём, так просто, как с окружностью не выйдет. Приходится идти на хитрость и выкручиваться, находя положение точки относительно тех прямых, на которых лежат стороны прямоугольника.

Пусть y = kx + b - уравнение прямой какой-либо стороны прямоугольника. Возьмём точку А на плоскости:

1. yA = kxA + b - точка лежит на прямой
2. yA < kxA + b - точка ниже прямой
3. yA > kxA + b - точка выше прямой

Взглянем на рисунок:



Сторона 1 выше стороны 2. Это можно понять, сравнив коэффициенты b прямых, на которых они лежат. Прямая 1 выше прямой 2, потому что b1 > b2.

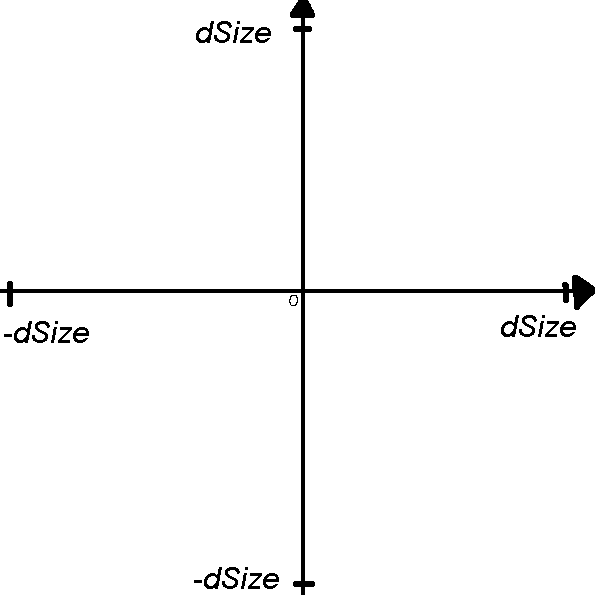
Сторона 3 выше стороны 4 по тем же самым причинам.

Тогда, точка будет находится в прямоугольнике, если:

1. Точка ниже стороны 1
2. Точка выше стороны 2
3. Точка ниже стороны 3
4. Точка выше стороны 4

А как это определять, я показал выше.

В моей реализации проекта подразумевается, что пользователь может масштабировать плоскость, на которой он рассматривает задачу. Это осуществимо через переменную dSize. Чертёж всё должен сказать за меня:

Но я прокомментирую: dSize - крайнее правое значение х и крайнее верхнее значение у, которые видны пользователю.

Координаты точек в моей реализации ограничены лишь типом double. В остальном - ограничений нет. Но opengl рендерит точки лишь в координатах ([-1; 1]; [-1; 1]). Соответственно, чтобы перевести координаты в те, которые opengl может отрисовать, я делаю следующее:

double x = p.x / Problem.dSize, y = p.y / Problem.dSize;

**Анализ используемой структуры данных**

Хранить нужно окружности и прямоугольники. Их я буду добавлять в динамический массив ArrayList. С ним удобно работать. Хранить «на лету» списки окружностей и прямоугольников не получится, так как в моей реализации проекта предполагается, что пользователь может масштабировать задачу, добавлять к уже существующему списку фигур новые фигуры, поэтому хочется всё-таки иметь возможность фигуры перерисовывать. В процессе я храню значение maxArea, чтобы сравнивать его с полученными значениями nowArea, которые показывают, сколько точек лежит в пересечении прямоугольника и окружности, рассматриваемых в данной итерации. nowArea не хранится, maxArea - хранится, чтобы было с чем сравнивать.

Также созданы следующие ссылки и динамический массив:

public Circle resultCircle = null;

public Quad resultQuad = null;

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

Первое - ссылка на результативный круг. Его нужно сохранить, чтобы в методе render отрисовать. Второе - ссылка на результативный прямоугольник. Его нужно сохранить по тем же причинам. Динамический массив сохраняет точки, которые лежат одновременно и внутри окружности, и внутри прямоугольника. Они нужны так же для отрисовки.

**Выбор метода решения**

Первоначально я думал об использовании интегральных формул. То есть, например, окружность можно задать уравнением:

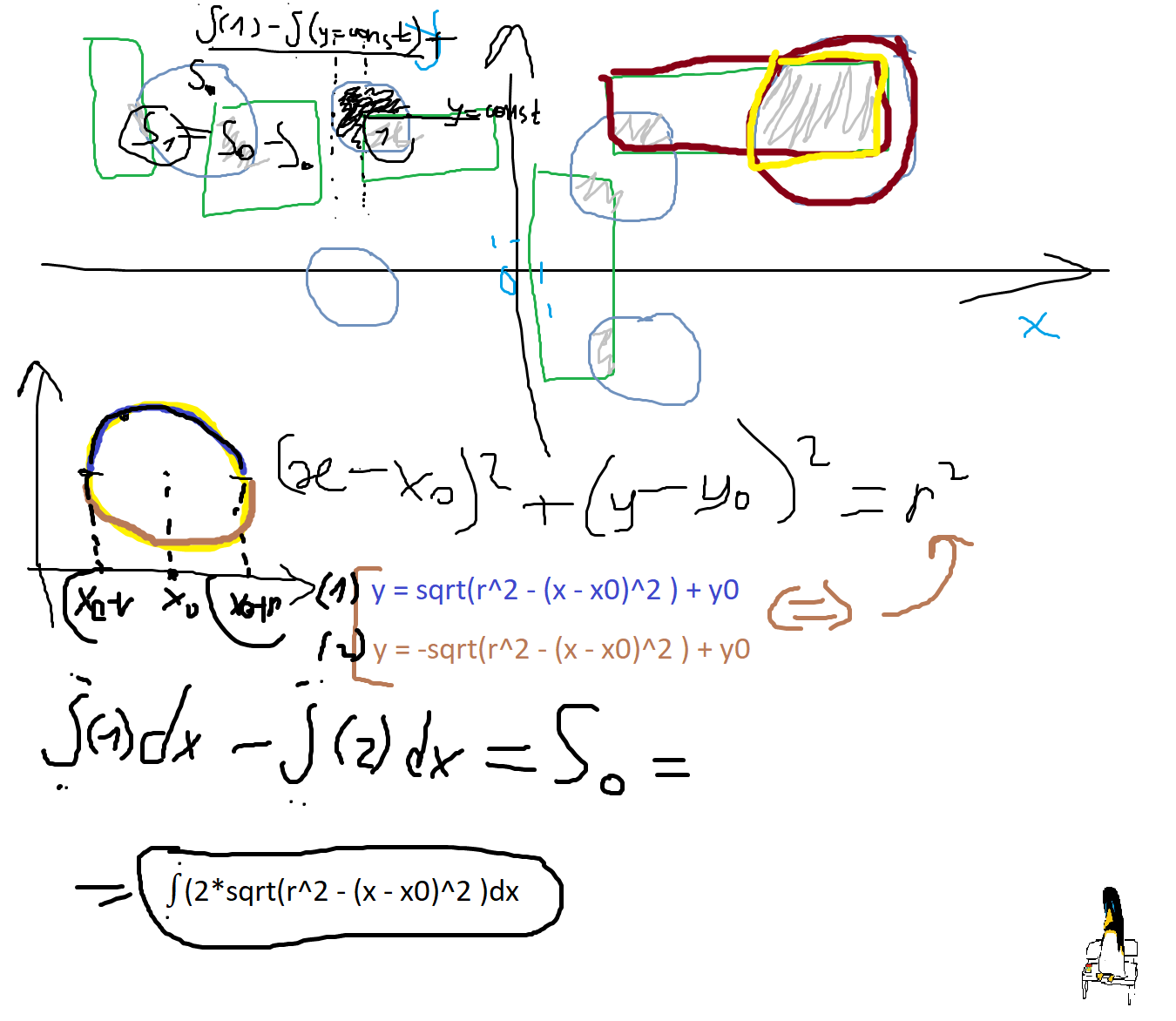
(x - x0)2 + (y - y0)2 = R2

где x0 и y0 - координаты центра, а R - радиус окружности.

Отсюда можно выразить y, задать окружность через две функции. Прямоугольник же можно было бы задать 4 прямыми, на которых лежат стороны, искать точки пересечения, а там через интегралы считать площади. Но:

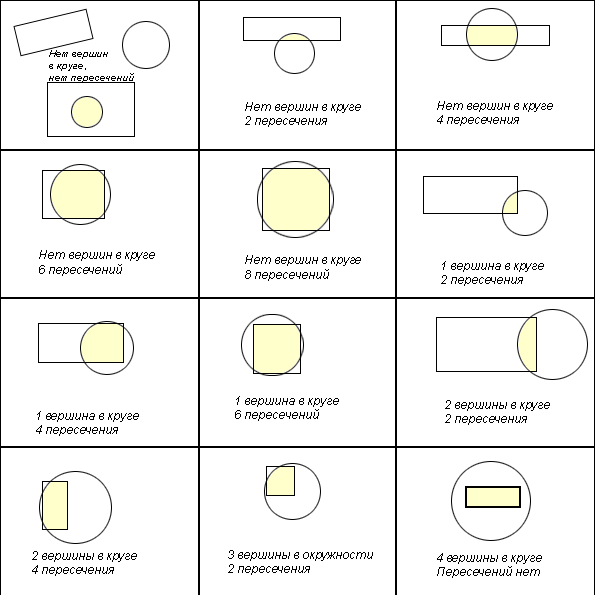
1. Случаев оказалось довольно много (порядка 12-13 и немного больше)
2. Использование этих формул оказалось сложным в плане вычислений

Вот один из первых чертежей, попытка использовать придуманный мною метод:



Тут изображён первичный бульон моих рассуждений, и получена формула для частного случая пересечения без учёта нахождения точек пересечения окружности и прямоугольника. А таких случаев неприлично много (~20). Поэтому от этой затеи я отказался.

В табличке снизу хоть и показаны многие случаи, однако не все. Здесь не рассмотрены случаи касания. Но уже представленных случаев довольно много.



Решил действовать по-другому, используя перебор.

Перед решением задачи создаётся список всех точек на пространстве (сетка), расположенных друг от друга с некоторым интервалом, который может изменить пользователь (точность измерений). Так же создаются нулевые ссылки на результативные окружность и прямоугольник, чтобы потом туда их сохранить, если они будут. Также создам переменную maxArea = 0, с которой буду сравнивать другие пересечения.

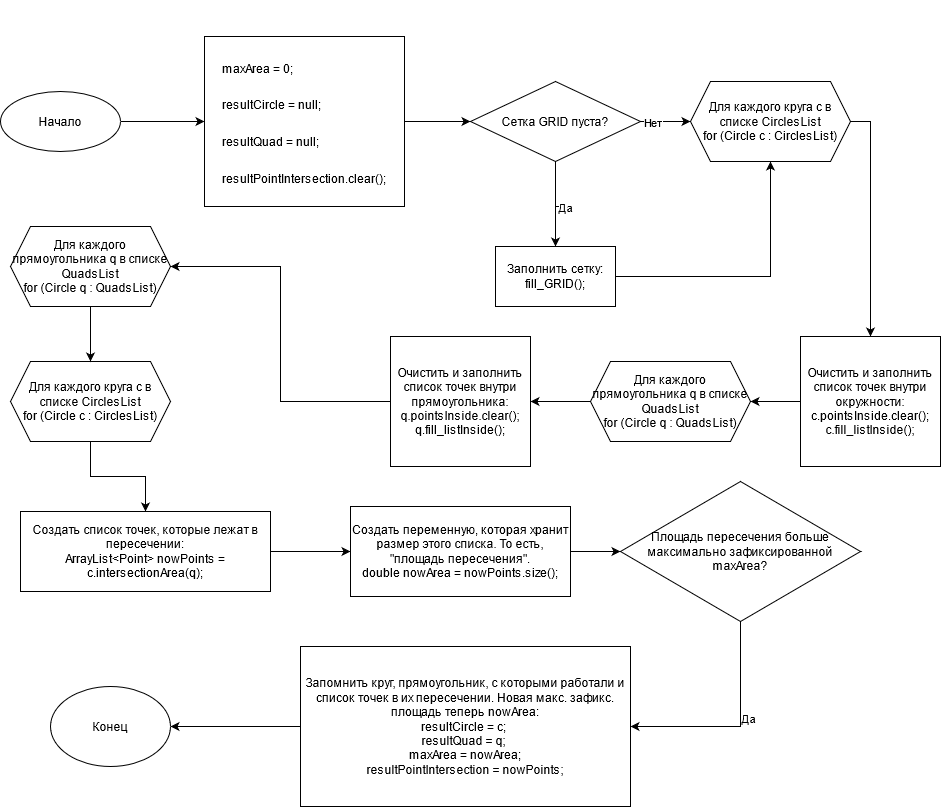
Дальше для каждого прямоугольника перебираем каждую окружность и рассматриваем эту пару отдельно от других. И у окружности, и у прямоугольника есть список для хранения точек, находящихся внутри фигуры и метод, заполняющий этот список.

При работе с конкретной парой вызываем эти методы, заполняя эти списки.

У окружности есть метод, который принимает на вход прямоугольник. Метод называется intersectionArea. Данный метод возвращает список точек, которые лежат и внутри данной окружности, и внутри данного прямоугольника, то есть в пересечении. Размер этого списка - nowArea - количество точек в пересечении, то есть «площадь» пересечения. Если nowArea больше максимально встреченной (maxArea), то nowArea и есть новая максимальная площадь. На этом этапе сохраняем окружность и прямоугольник, с которыми работали.

Задача решена после перебора всех окружностей и прямоугольников.

Ниже предоставлена **блок-схема моего алгоритма**.



**ВНН. Вектора не нужны!** В этом небольшом блоке я хочу объяснить, почему использование векторов в этой задачи бессмысленно и от него нужно отказаться. Я не понимаю, зачем нужно реализовывать класс Vector, если это натурально тоже самое, что и класс точки Point. Нас просят сделать следующее:

«реализовать класс двумерного вектора problem.Vector2 с двумя публичными вещественными полями x и y, а также конструктором с двумя вещественными аргументами, задающий значения этих полей».

Зачем делать всё сложнее и работать с точкой, как с вектором? Зачем работать с вектором, интуитивное понимание которого сложнее, если можно работать с точкой? В реализации, по крайней мере, моей задачи, нигде вектора не используются в математическом смысле. Я рассматриваю, например, расположение точки относительно прямой. Можете себе представить такие же рассуждения с использованием вектора? Не думаю. Даже соглашения на моей стороне. Например, посмотрим, как задаётся прямоугольник:

«Прямоугольник. Задаётся двумя вершинами одной из сторон. А также точкой, лежащей на прямой, проходящей через две другие вершины.»

Прямоугольник, хочу заметить, задаётся точками, а не векторами! Ровно как и прямая, треугольник, окружность и все остальные фигуры.

**Комментированный листинг**

//Решить задачу

public void solve() {

//обнулить переменную, хранящую максимальную площадь

maxArea = 0;

//обнулить ссылку на результативный круг

resultCircle = null;

//обнулить ссылку на результативный прямоугольник

resultQuad = null;

//очистить список точек в результативном пересечении

resultPointIntersection.clear();

//если сетка пуста - заполнить заново

if (GRID.size() == 0)

fill\_GRID();

//для каждой окружности заполняем список точек внутри, предварительно очистив, если что-то в этом списке уже было

for (Circle c : CirclesList) {

c.pointsInside.clear();

c.fill\_listInside();

}

//для каждого прямоугольника заполняем список точек внутри, предварительно очистив, если что-то в этом списке уже было

for (Quad q : QuadsList) {

q.pointsInside.clear();

q.fill\_listInside();

}

//для каждого прямоугольника перебираем каждую окружность

for (Quad q : QuadsList) {

for (Circle c : CirclesList) {

//создать список «нынешних» точек, находящихся в пересечении выбранных окружности и прямоугольника в данной итерации

ArrayList<Point> nowPoints = c.intersectionArea(q);

//создать временную переменную, которая хранит размер предыдущего списка, то есть «нынешнюю» площадь

double nowArea = nowPoints.size();

//если «нынешняя» площадь больше максимальной, то

if (nowArea > maxArea) {

//сохранить в ссылки окружность и прямоугольник, с которыми работали на данной итерации

resultCircle = c;

resultQuad = q;

//новая максимальная площадь - это «нынешняя»

maxArea = nowArea;

//сохранить все точки в пересечении

resultPointIntersection = nowPoints;

}

}

}

}

**Пример работы программы**

Входные данные:

type 2

dSize 20.0

QuadsList

(1,2; -16,3), (0,4; 6,6), (5,2; 6,9)

(-10,3; -5,4), (-6,5; -4,0), (-8,4; 1,0)

(15,7; 3,3), (-14,4; 11,1), (-17,3; 0,1)

(15,9; 8,1), (-4,5; -9,4), (-14,1; 2,8)

(5,9; -10,0), (6,4; -3,8), (-14,4; -2,2)

(-9,4; -17,4), (2,5; 14,8), (11,3; 11,3)

(-4,6; -12,5), (-1,8; 5,5), (9,4; 4,0)

CirclesList

(-19,3; 0,7), (-19,0; 0,0)

(16,9; 18,3), (-2,0; 0,3)

(3,9; 13,8), (0,0; 1,5)

(1,1; 15,1), (1,5; 14,0)

(4,7; 3,4), (15,4; 9,8)

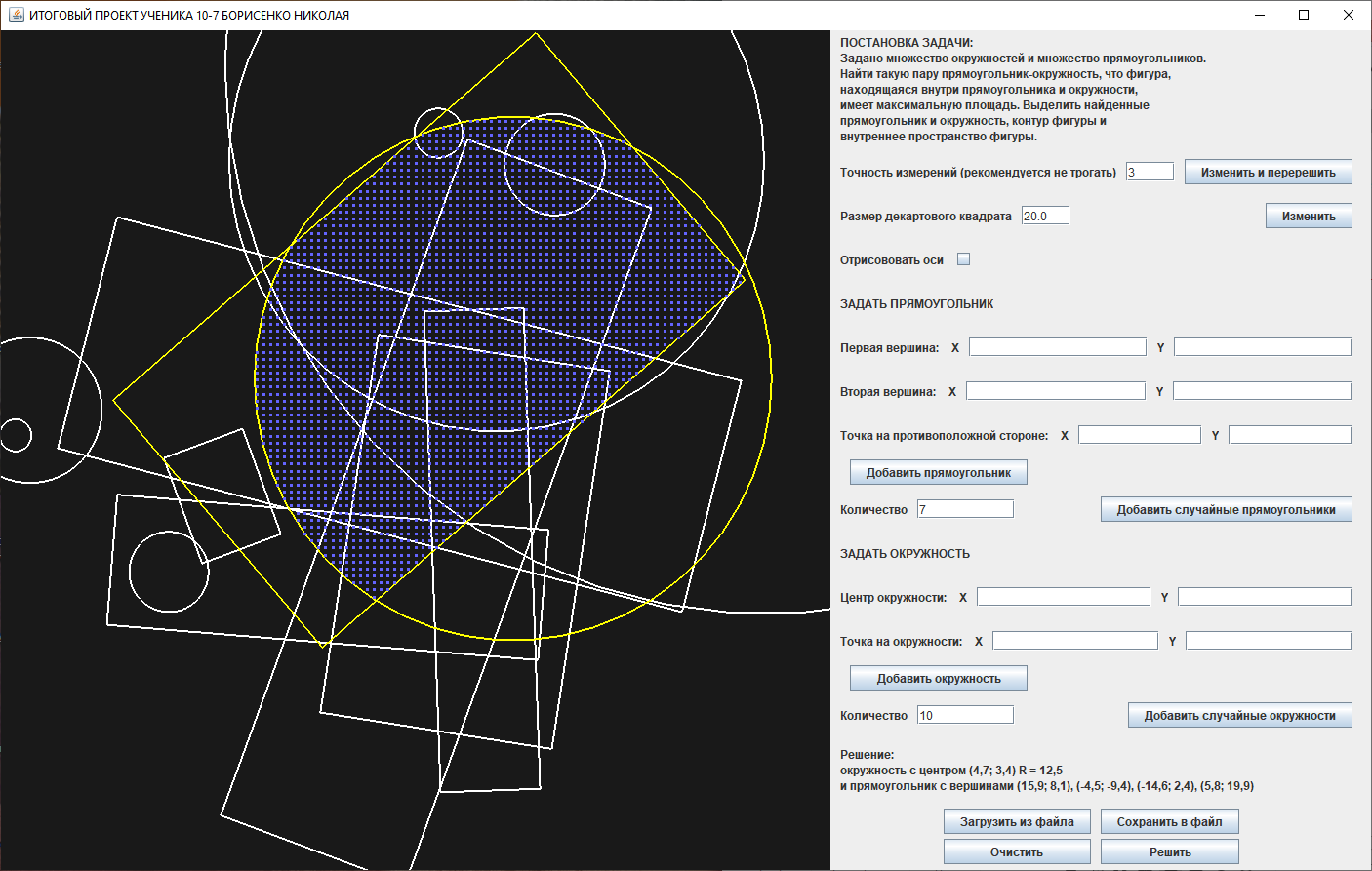
(6,7; 13,6), (4,3; 13,3)

(-18,6; 1,9), (-15,7; 0,0)

(-11,9; -5,8), (-10,0; -6,0)

Выходные данные:

Решение: окружность с центром (4,7; 3,4) R = 12,5 и прямоугольник с вершинами (15,9; 8,1), (-4,5; -9,4), (-14,6; 2,4), (5,8; 19,9)

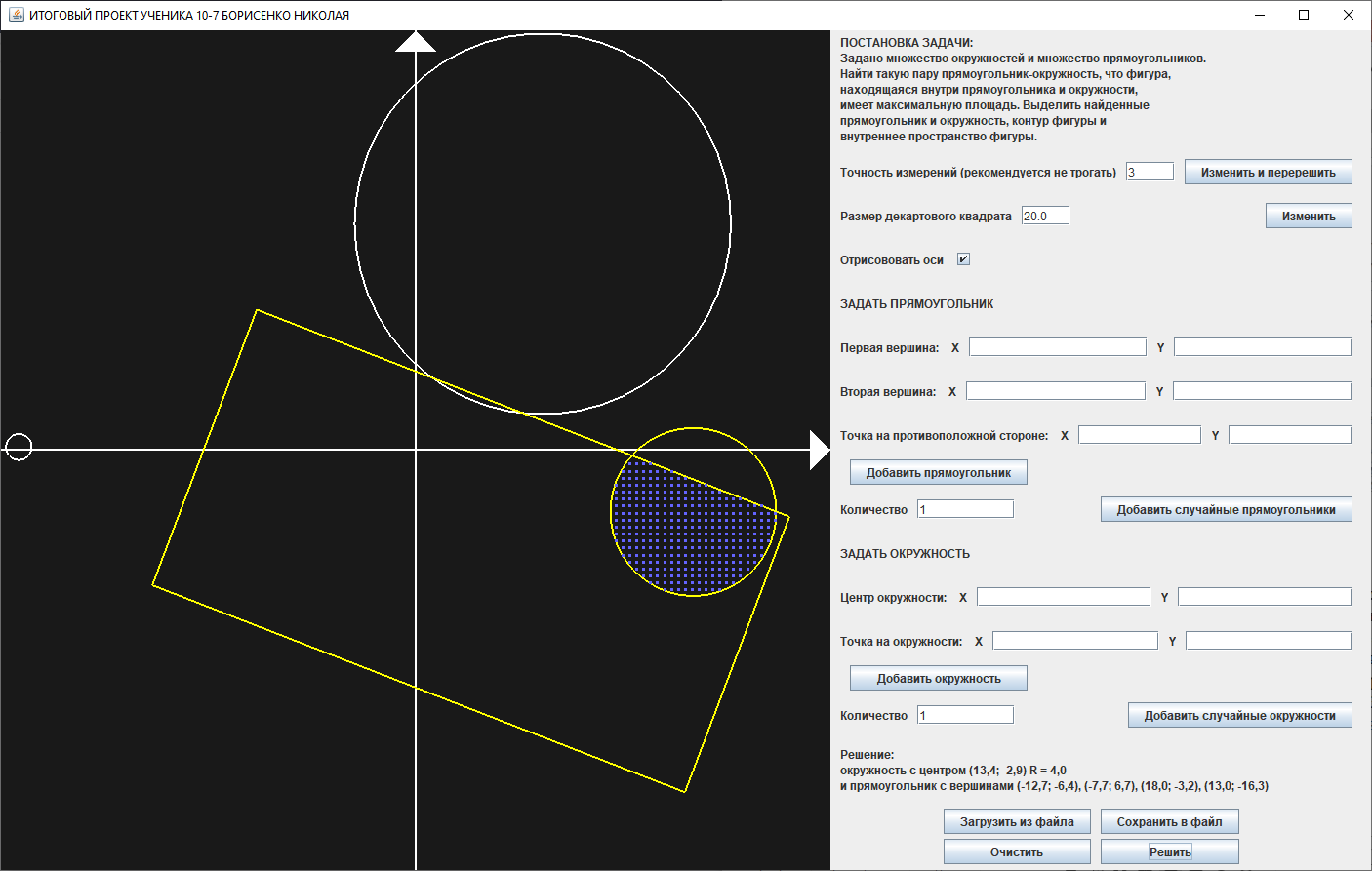


**Анализ правильности решения**

Программа работает правильно. Алгоритм максимально прост, задействован простейший перебор точек, а потому вероятность ошибки крайне мала. Но есть нюансы.

Во-первых, да, мне удалось генерировать случайные окружности строго внутри заданного квадрата, но с прямоугольниками так не получилось. Они чуть-чуть вылезают. Но, я считаю, что это не очень важно, ведь пользователь может увеличить размер декартового квадрата сколь угодно сильно и уместить в экран любую фигуру. Иногда программа может выдать неправильный результат, если точность измерений выставлена низкой. Точность 3 - вполне оптимальная для подавляющего большинства задач. При повышении точности программа работает медленней. Но точнее.

Вот пример очевидно верно решённой задачи:



**Будущее проекта**

Мой проект жив во времени! У него есть своё прошлое, он живёт в настоящем - вы прямо сейчас читаете отчёт по программе. У него есть и будущее, так как проект можно доработать.

Вы могли заметить, что в проекте реализован класс Color, но про него ничего не сказано и нигде он не используется. Я создал этот класс, чтобы заложить фундамент в реализацию кастомизации цвета пользователем. В будущем я хочу сделать отдельные настройки программы, в которых пользователь сможет выбирать: цвет фона, цвет отрисовки окружностей и прямоугольников, цвет отрисовки результативных окружности и прямоугольника, цвет отрисовки внутренних точек и так далее. Это внесёт краски в моей проект и сделает его ярким и интересным для пользователя.