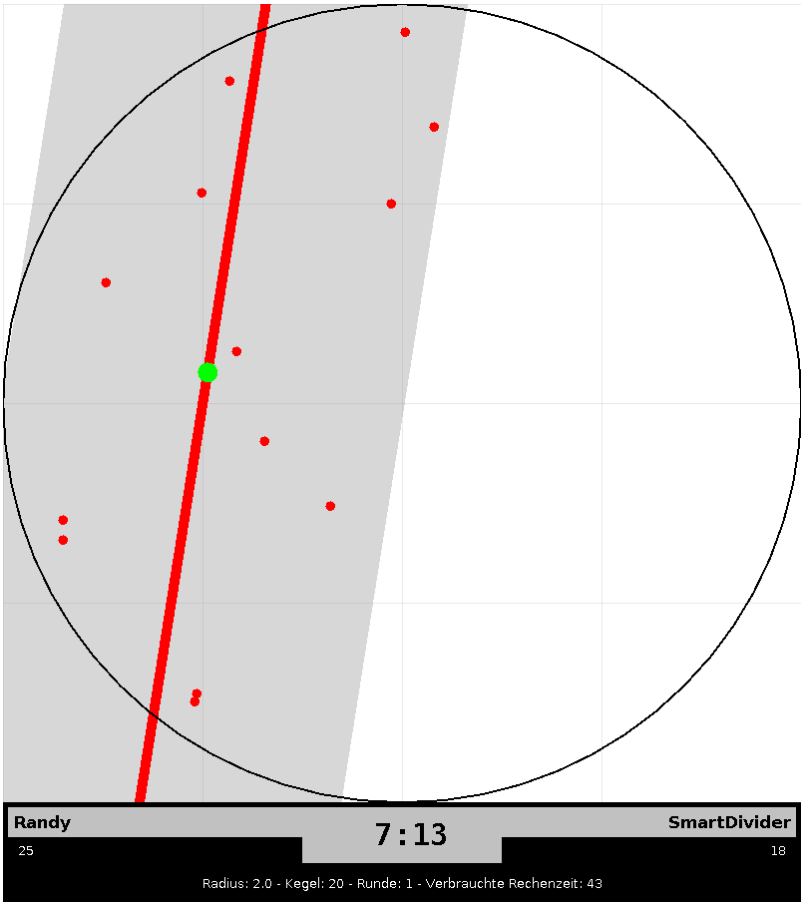
Aufgabe 2 - Panorama-Kegeln

Dominic S. Meiser

33. Bundeswettbewerb Informatik Runde 2



Inhaltsverzeichnis

1 Lösungsidee

Diese Aufgabe besteht aus mehreren Teilaufgaben; zum einen die Simulation des Spiels, die geforderte übersichtliche Ausgabe, und natürlich die KI(s), die dieses Spiel spielen.

1.1 Die Simulation des Spiels

Um das Spiel simulieren zu können, braucht man zunächst eine interne Representation der Kegel und weiteren spielrelevanten Daten. Es reicht für dieses Spiel, die Kegel in einem Array abzuspeichern und darüber hinaus den Radius und die mitspielenden KIs inklusive den von der KI umgeworfenen Kegeln und eventuell der verbrauchten Laufzeit zu speichern. Dadurch, dass die Kegel in einem Array abgespeichert werden, braucht man die Anzahl der Kegel nicht noch einmal separat abzuspeichern, da man einfach die Größe dieses Arrays benutzen kann. Als nächstes müssen die Kegel in den Kreis eingesetzt werden. Zunächst hatte ich zwei Ideen:

1. Die Kegel solange mit zufällig ausgewählten kartesischen Koordinaten versehen, bis diese im Kreis liegen.
2. Jedem Kegel Polarkoordinaten zuteilen.

Ob ein Kegel im Kreis liegt kann man ganz einfach mit dem Satz des Pythagoras, $x^2 + y^2 = r^2$, überprüfen. Nach ein paar Tests hat sich die erste Idee als die deutlich schnellere erwiesen. Man benötigt zwar mehr Durchläufe, die Benutzung der trigonometrischen Funktionen scheint jedoch länger zu dauern (Test mit Intel i3 Dual-Core 2,50 GHz).

Für die weitere Simulation sollte man zuerst abspeichern wie viele Kegel noch stehen. Dass sind am Anfang alle Kegel. Dann muss jede der beiden KIs abwechselnd den aktuellen Zustand erhalten. Ich bezeichne hier den Benutzer, insofern dieser mitspielt, auch als KI, weil er sich für das Programm genauso verhält wie eine KI. Die bereits umgeworfenen Kegel kann die KI herausfiltern, d.h. es reicht, wenn man den Kegel als umgeworfen speichert. Anschließend muss, insofern die KI einen Zug gemacht hat, die beiden Geraden herausgefunden werden, die den Bereich der Kegel, die umgeworfen werden, begrenzen. Da die Kugel, die geworfen wird, den Radius 1 hat, haben die beiden Geraden einen Abstand von 2. Somit kann man die Funktionen mit $f(x) = tan(\alpha) * x + y0 + z/cos(\alpha) - tan(\alpha) * x0$ bestimmt werden, wobei $x0$ und $y0$ der Punkt ist, an dem die Kugel geworfen wird, α der Winkel ist, mit dem die Kugel geworfen wird, wobei 0 eine waagerechte Gerade ist, und z der Abstand zur Mittelgeraden ist, im Fall der oberen Funktion (ich nenne diese $o(x)$) also 1 und im Fall der unteren Funktion (ich nenne diese $u(x)$) -1. Um jetzt die richtigen Kegel als umgeworfen zu markieren, muss man nur noch für jeden Kegel, der noch steht, überprüfen, ob $y \leqslant u(x)$ und $y \geqslant o(x)$ ist. Ist dies der Fall, kann ich den Kegel als umgeworfen markieren und die Anzahl der umgeworfenen Kegel dieser KI um 1 erhöhen und gleichzeitig die Anzahl der noch stehenden Kegel um 1 reduzieren. Wenn die Anzahl der noch stehenden Kegel bei 0 angekommen ist, ist das Spiel vorbei. Gewonnen hat selbstverständlich die KI, die die meisten Kegel umgeworfen hat.

1.2 Die Ausgabe des Spielzustands

Die Ausgabe des aktuellen Spielzustands sollte mit einer GUI, einer grafischen Oberfläche, erfolgen. Dabei muss man zuerst die beiden KIs auswählen, die mitspielen. Hier muss es auch möglich sein auszuwählen, dass der Benutzer spielt. Der Einfachheit halber kann man die beiden KIs in einer Zeile auswählen lassen, wobei immer die linke KI anfängt. Dann muss die GUI natürlich den aktuellen Zustand, also den Kreis, die darin stehenden Kegel, und den aktuellen Punktestand enthalten. Zudem sollte man anhand eines Rechtecks den von der KI abgegebenen Zug sehen können. Dabei muss dieser Bereich über den ganzen Bildschirm gehen, nicht so wie auf dem Turnierserver, wo teilweise nur der halbe Kreis mit dem Rechteck ausgestattet wird. Der Benutzerfreundlichkeit wegen sollte man auch nicht für jedes Spiel das Programm neu starten müssen. Darüber hinaus muss die GUI die Möglichkeit bieten, den Benutzer mitspielen zu lassen, und sich damit quasi wie eine KI zu verhalten.

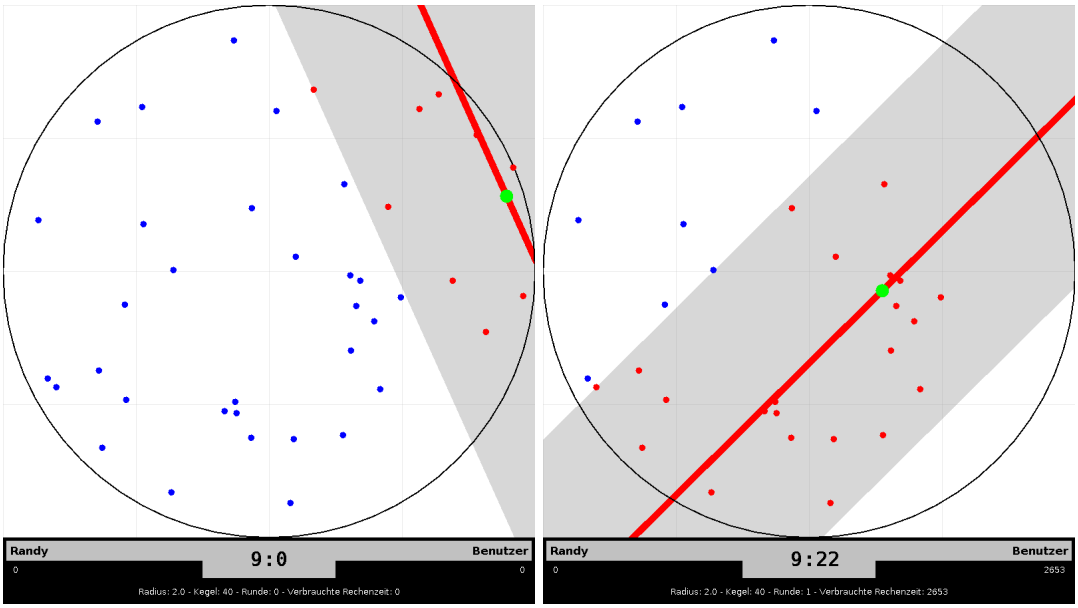
1.3 Die KI(s)

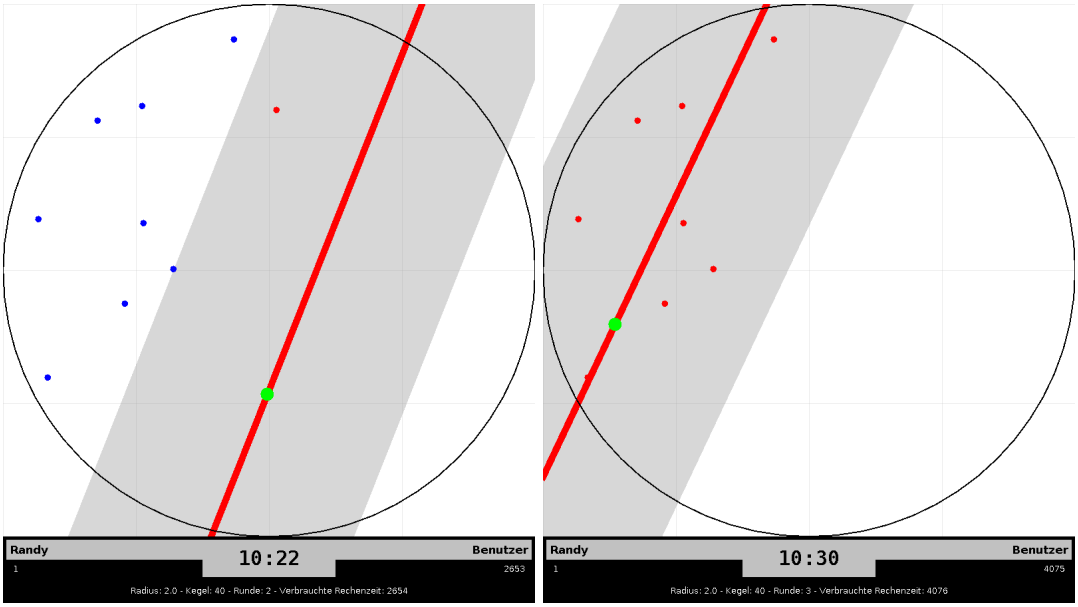
Zunächst muss man natürlich die KI „Randy“ schreiben. Diese ist in der Aufgabenstellung sehr klar beschrieben: Die KI muss einen gleichmäßig verteilten Punkt und einen gleichmäßig verteilten Winkel bestimmen und zurückgeben. Dabei muss dieser natürlich im Kreis liegen. Den Punkt kann man genauso wie die Kegel wählen, hierbei allerdings nur mit kartesischen Koordinaten, da, wenn ich zufällige Polarkoordinaten in kartesische Koordinaten umwandle, diese nicht gleichverteilt sind. Den Winkel kann ich einfach zwischen 0 und $180°$ bzw. π wählen. Da die Richtung, in die der Ball geworfen wird, nicht entscheidend ist, ist ein Winkel von $181°$ gleichbedeutend wie ein Winkel von $1°$.

Neben der KI „Randy“ war noch gefordert, sich eine Strategie zu überlegen, um gegen Randy zu gewinnen, und diese in einer KI zu implementieren. Meine erste Idee war natürlich ein Bruteforce, der dumm die Punkte in einem bestimmten Abstand und dazu einen Winkel mit einem bestimmten Abstand ausprobiert und die Punkt-Winkel-Kombination zurückgibt, bei der am meisten Kegel umgeworfen werden. Mir wurde jedoch schnell klar, dass diese Strategie für einen Computer zu langsam ist.

1.3.1 Die Strategie

Also habe ich mir überlegt was ich als Mensch mache, um gegen Randy zu gewinnen: Ich schaue, wo viele Kegel auf einem Haufen sind, bzw. wo viele Kegel in einer Reihe sind, und werfe dort die Kugel hin. Insofern Randy nicht beim ersten Zug mehr als die Hälfte aller Kegel umwirft, habe ich damit auch eine gute Chance zu gewinnen, wie man hier sieht:





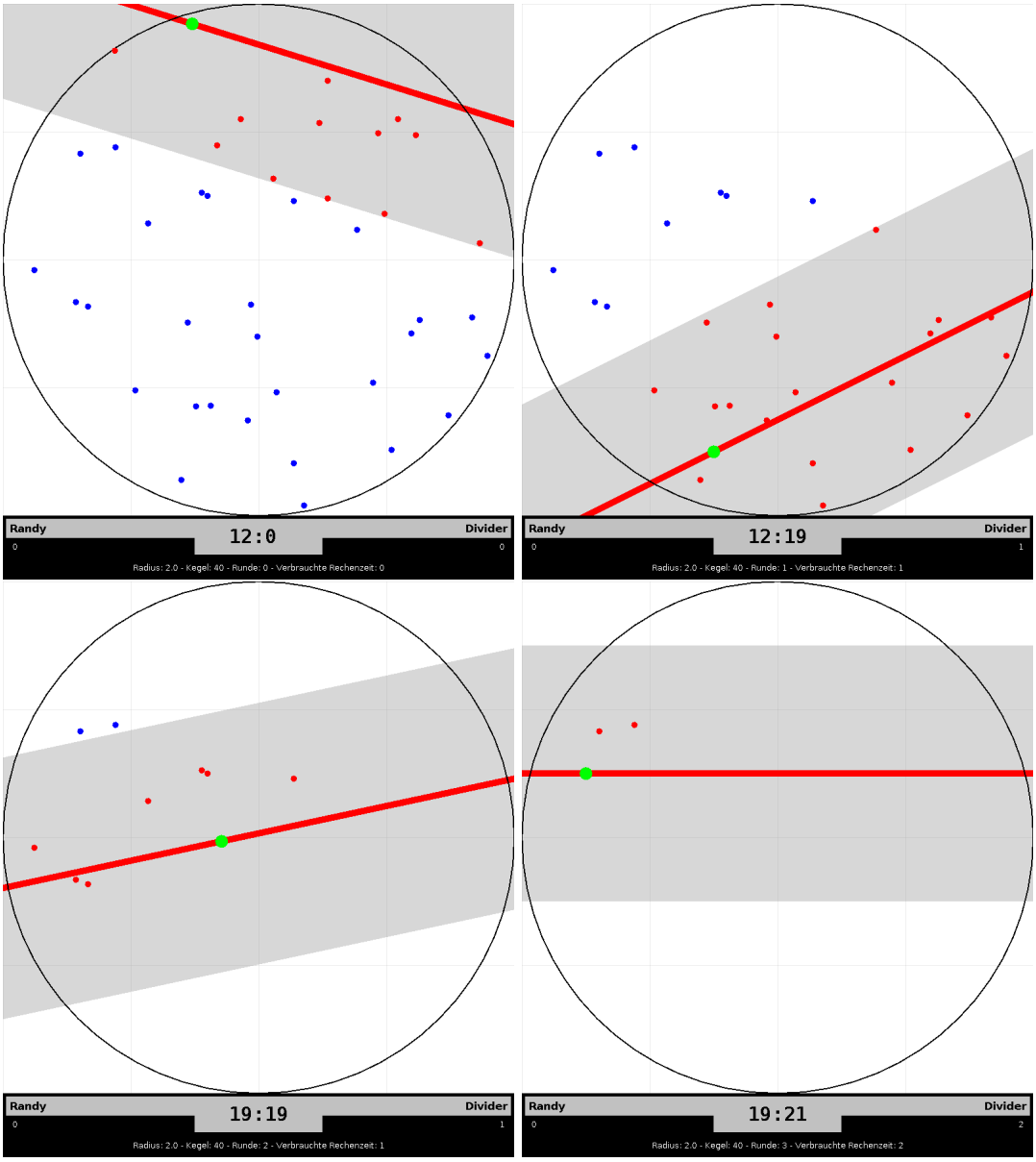
Meine KI sollte also nach Möglichkeit nach vielen Kegeln auf einem Haufen oder in einer Reihe suchen und dorthin die Kugel werfen.

1.3.2 Das Programm

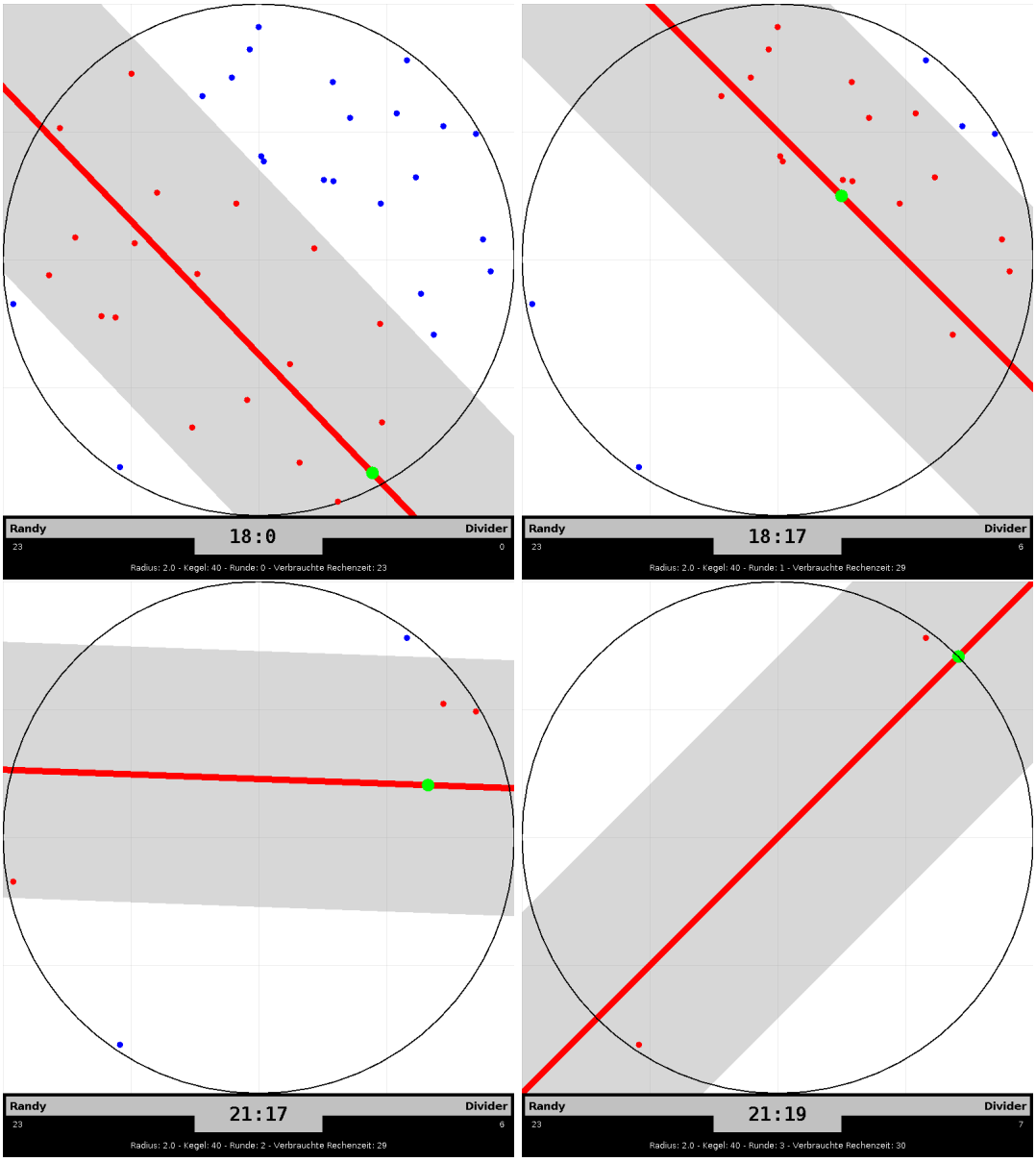
Meine erste Idee war es, über das Spielfeld ein Raster zu legen, wie ich es in der grafischen Oberfläche wegen Benutzerfreundlichkeit auch implementiert hatte, die entstehenden Kästchen, ich nenne diese Parts, nach der Anzahl der Kegel, die darin liegen, zu sortieren, den Punkt als Mitte des besten Parts zu wählen und den Winkel so zu bestimmen, dass die Kugel den Mittelpunkt des zweitbesten Parts trifft. Da die Kugel einen Radius von 1 hat und somit das Rechteck mit den umgeworfenen Kegeln eine Breite von 2 hat, bietet es sich an, die Parts 1x1 groß zu machen, da die Kugel dann immer den kompletten Part mitnimmt (für $45°$ ist immernoch die Diagonale kleiner als 2: $\sqrt{1^2+1^2} \leqslant 2 \Leftrightarrow \sqrt{2} \leqslant 2$). Die Parts kleiner zu machen hat keinen Sinn, da ich dann ungenauere Ergebnisse bekomme, weil ich dadurch die Warscheinlichkeit erhöhe, dass ein Haufen von Kegeln in mehrere Parts aufgesplittet wird.

1.3.3 Das Ergebnis

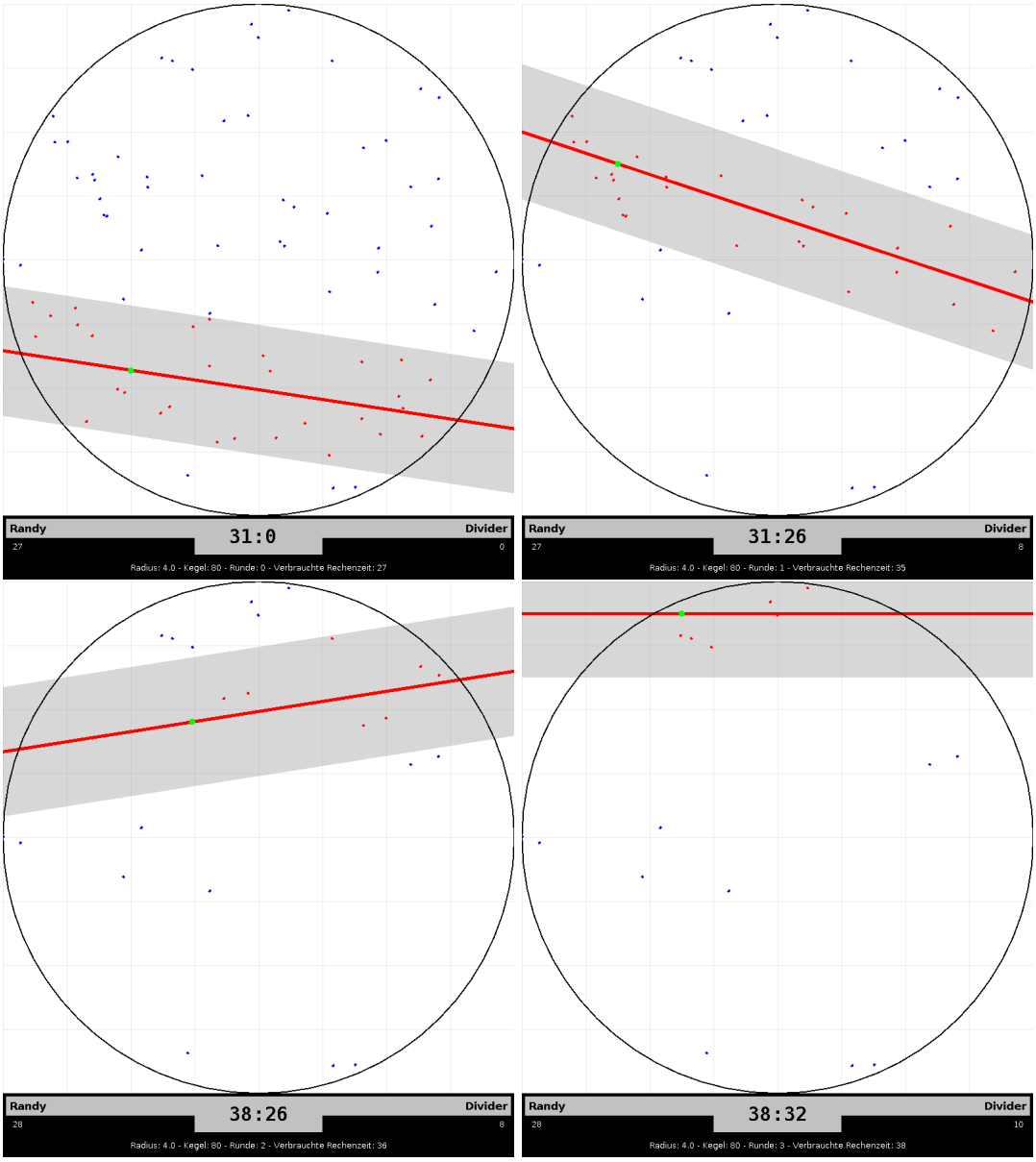
Dieser Algorithmus kann ganz gut funktionieren:

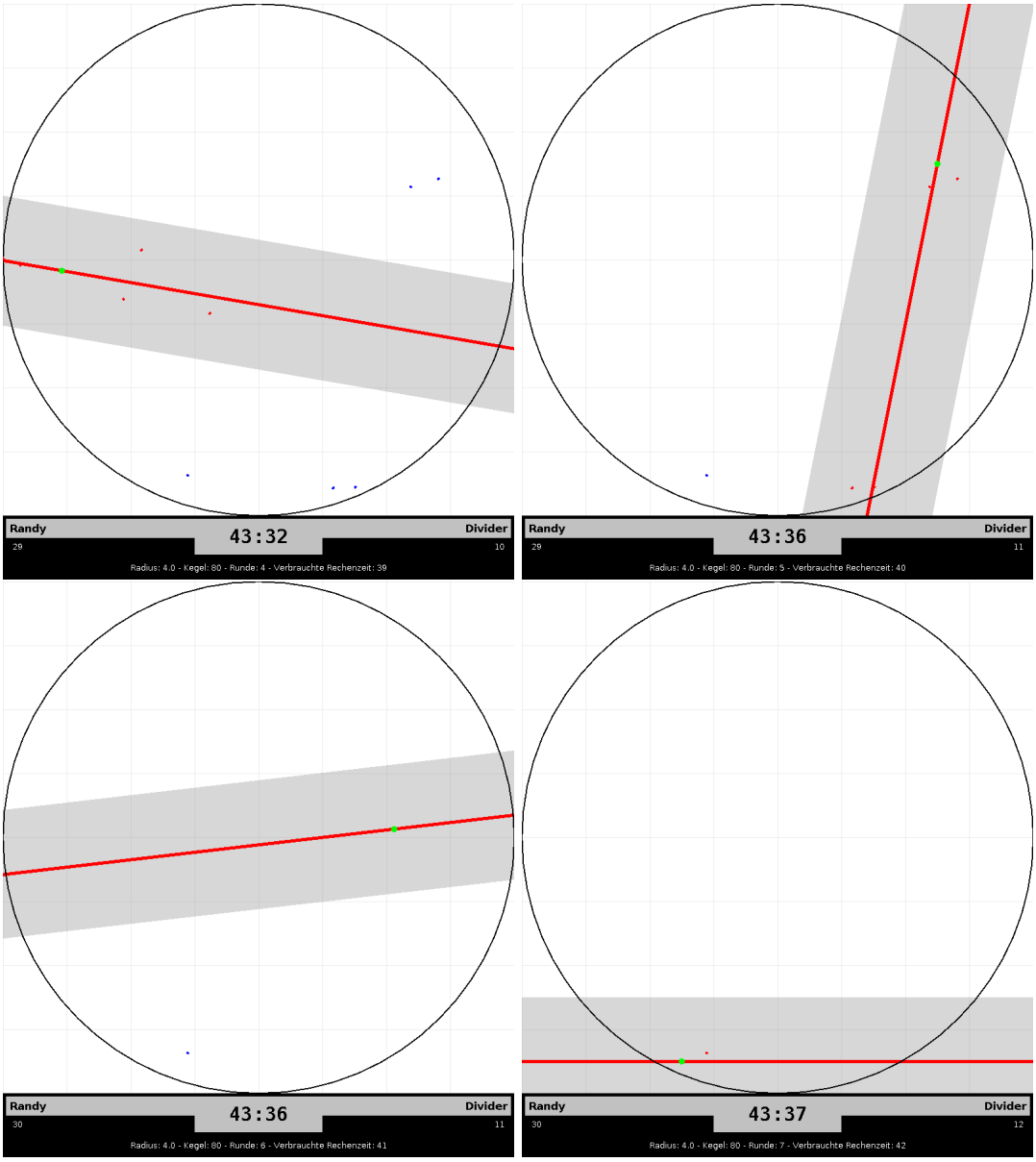


Aber ganz zuverlässig ist dieser Algorithmus nicht:



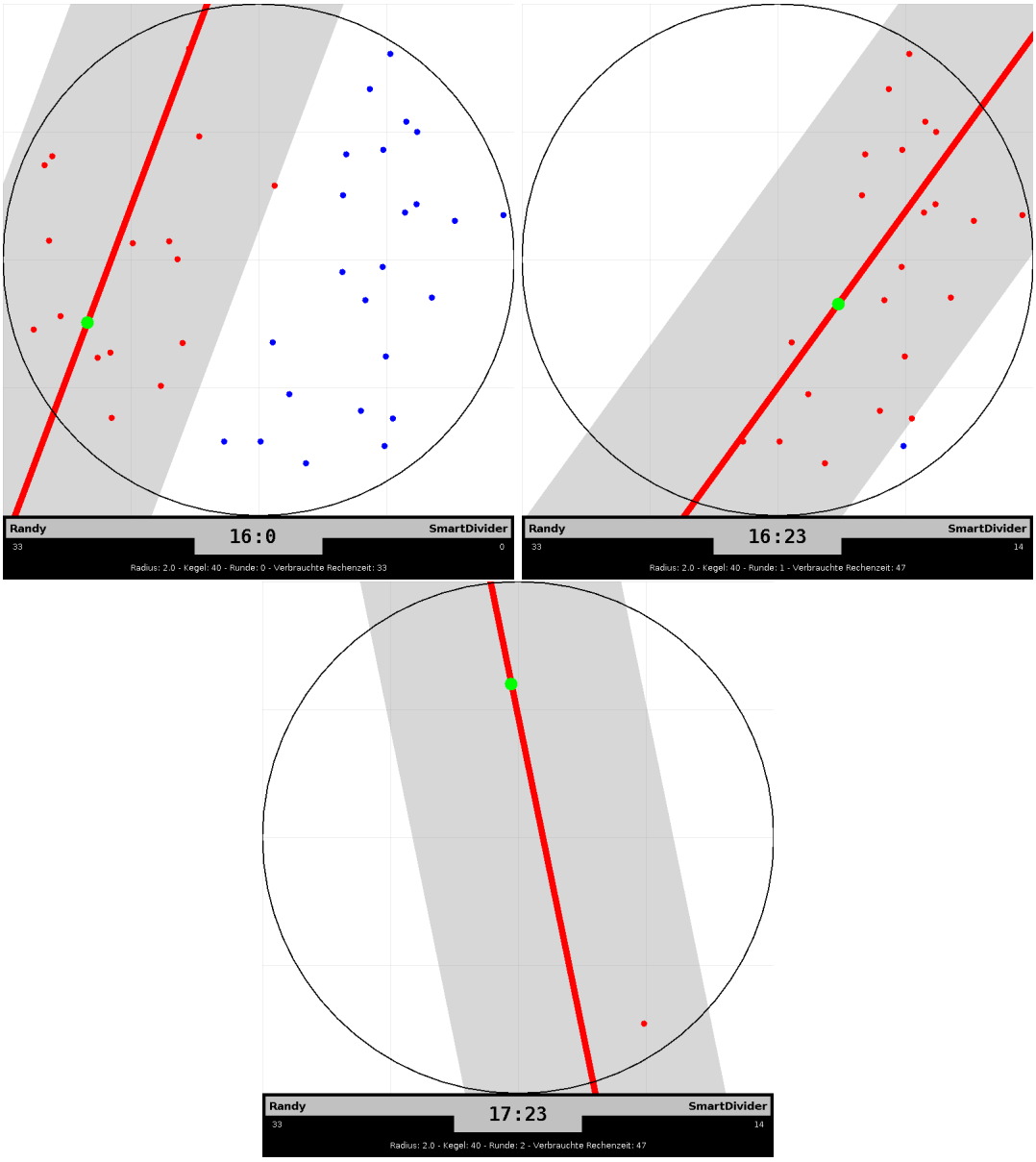
Mit größeren Radien hat der Algorithmus noch mehr Probleme:


Randy
31:0
Divider
27
0
Radius: 4.0 - Kegel: 80 - Runde: 0 - Verbrauchte Rechenzeit: 27
Randy
31:26
Divider
27
8
Radius: 4.0 - Kegel: 80 - Runde: 1 - Verbrauchte Rechenzeit: 35
Randy
38:26
Divider
28
8
Radius: 4.0 - Kegel: 80 - Runde: 2 - Verbrauchte Rechenzeit: 36
Randy
38:32
Divider
28
10
Radius: 4.0 - Kegel: 80 - Runde: 3 - Verbrauchte Rechenzeit: 38

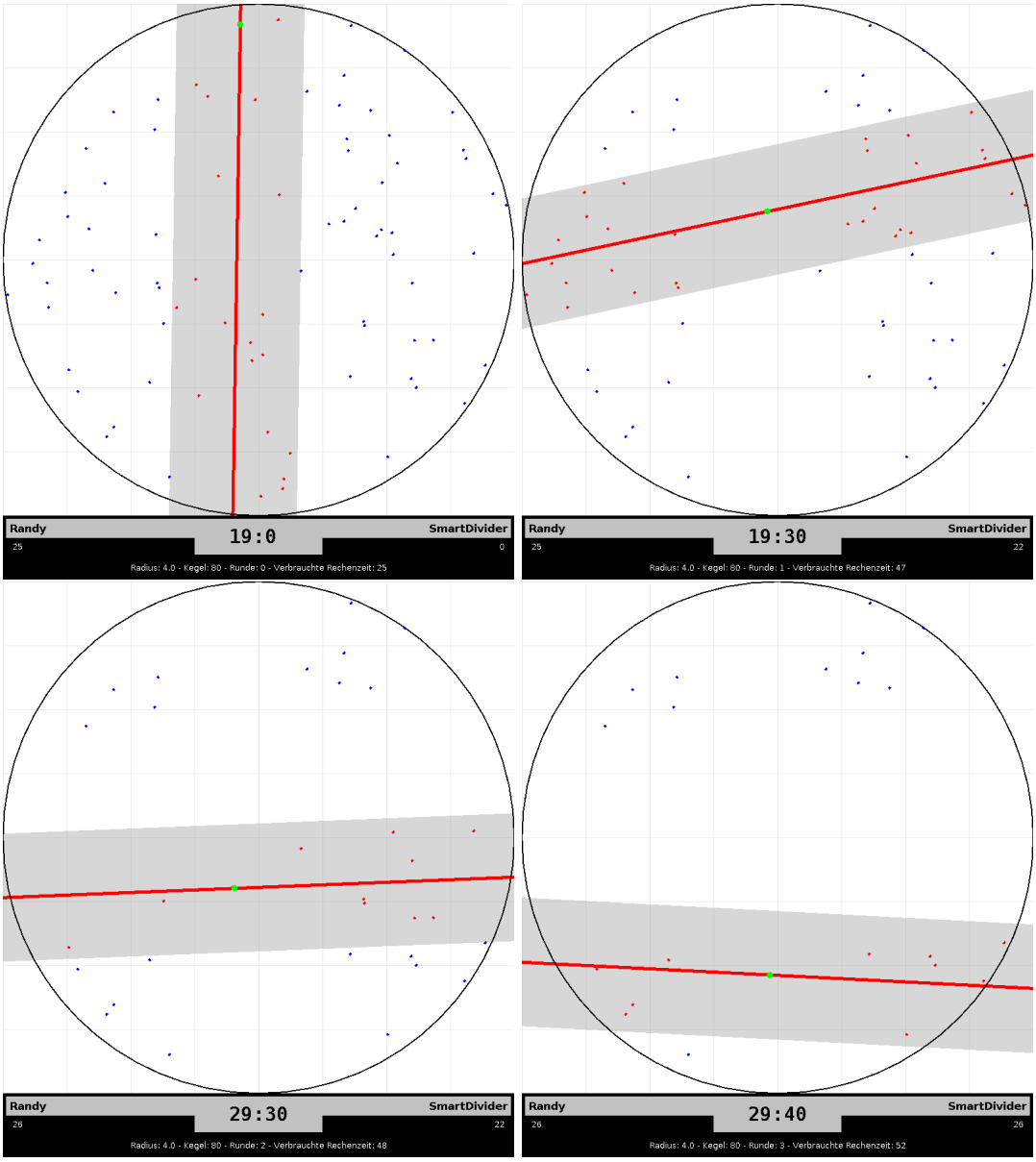


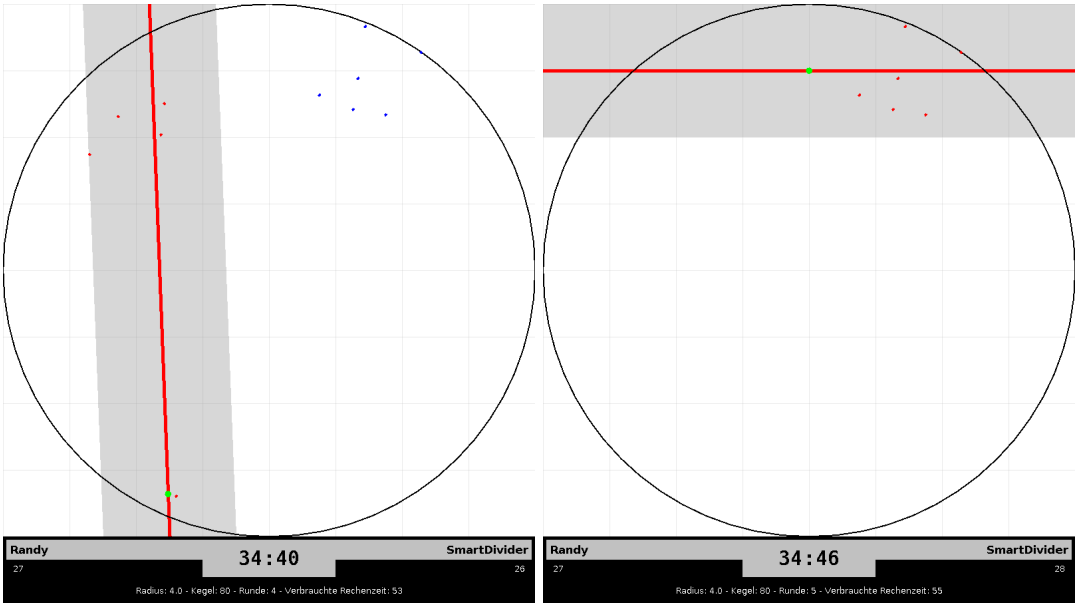
1.3.4 Die Verbesserung

Um mein Programm etwas zu verbessern, aber immernoch effizienter als Bruteforce zu bleiben, habe ich mir überlegt, man könnte, wie zuvor auch, die Kegel in Parts aufteilen und die Parts nach der Anzahl der Kegel sortieren. Jetzt kann man die besten Parts anschauen und, wie bei Bruteforce, alle möglichen Winkel ausprobieren. Dies ist deutlich effizienter als Bruteforce und liefert bessere Ergebnisse als vorher:



Auch bei größeren Radien hat diese Verbesserung noch eine Chance:


Randy
19:0
SmartDivider
25
0
Radius: 4.0 - Kegel: 80 - Runde: 0 - Verbrauchte Rechenzeit: 25
Randy
19:30
SmartDivider
25
22
Radius: 4.0 - Kegel: 80 - Runde: 1 - Verbrauchte Rechenzeit: 47
Randy
29:30
SmartDivider
26
22
Radius: 4.0 - Kegel: 80 - Runde: 2 - Verbrauchte Rechenzeit: 48
Randy
29:40
SmartDivider
26
26
Radius: 4.0 - Kegel: 80 - Runde: 3 - Verbrauchte Rechenzeit: 52



Ein letztes Problem habe ich noch festgestellt: Der Punkt liegt nicht zwingend im Kreis. Deshalb sollte die KI, um die Rundungsfehler des Turnierserver, der aus mir unbekannten Gründen mit float rechnet, zu umgehen, einfach den mittigsten Punkt der Gerade zurückgeben. Das geht ganz einfach, indem ich die Differenz des x-Werts der beiden Schnittpunkte mit dem Kreis nehme und dazu den y-Wert ausrechne. Wie ich die Funktionsgleichung für die Gerade aufstelle habe ich oben schon erläutert. Jetzt kann ich die Schnittpunkte mit dem Kreis ganz einfach berechnen: $x_{1/2} = \frac{\pm\sqrt{m^2r^2-b^2+r^2}-m*b}{m^2+1}$

2 Umsetzung

Ich habe die Lösungsidee in ein Java-Programm umgesetzt, damit ich die KI problemlos auf den Turnierserver hochladen kann. Im folgenden werde ich die wesentlichen Bestandteile meines Programms erklären. Insgesamt hat mein Programm 3 Klassen mit `main()`-Methode:

- `Main`: Diese Klasse startet das Spiel ganz normal, wie es die Aufgabenstellung fordert. Es nimmt als Kommandozeilenargumente die Anzahl der Kegel (`-c`) und den Radius (`-r`). Default ist Radius=2 und Kegel=20. Außerdem erstellt die Klasse am Ende jedes Spiels eine gif-Animation des Spiels.
- `AiObfuscator`: Diese Klasse enthält einen Haufen reguläre Ausdrücke (Regexe), die eine KI für den Server „obfuscaten“. Außerdem ruft sie delombok auf, sodass ich Lombok-Annotations für die KIs benutzen kann. Diese Klasse nimmt den Namen der KI-Klasse als Kommandozeilenargument entgegen.
- `AiTester`: Diese Klasse füttert alle bekannten KIs mit denselben Ausgangszuständen und erstellt damit quasi eine „Rangordnung“ unter den KIs. Außerdem erstellt die Klasse ganz am Ende eine gif-Animation mit den Ausgaben aller KIs.

2.1 Die Spielinitialisierung

Nachdem die Argumente mithilfe von Apache CLI geparst wurden, sucht mein Programm zunächst nach den KIs. Für die KIs habe ich ein eigenes Interface namens `AI` angelegt, dass eine `move()`- und eine `getName()`-Methode enthält. Die `getName()`-Methode gibt by default den Namen der Klasse zurück. Das Interface sieht damit so aus:

```
public interface AI
{
    public Move move (SceneData data);
    public default String getName () { return getClass().getSimpleName(); }
}
```

Zudem sind alle KIs im Paket `bwinf33_2.aufgabe2.ai.impl`, sodass ich die KIs ganz einfach mithilfe von Reflections auffinden kann:

```
Reflections p = new Reflections("bwinf33_2.aufgabe2.ai.impl");
Set<Class<? extends AI>> kiset = p.getSubTypesOf(AI.class);
```

Der Benutzerfreundlichkeit wegen sortiere ich diese Klassen anschließend noch nach Name, und füge an erster Stelle die benutzergesteuerte KI ein. Als nächstes erstelle ich ein `SceneData`-Objekt und füttere es mit den Kommandozeilenargumenten bzw. dem default-Wert (s.o.), woraufhin diese Klasse die Kegel mithilfe der in der Lösungsidee beschriebenen Variante mit den kartesischen Koordinaten erstellt:

```
for (int i = 0; i < cones; i++)
{
    Cone cone;

    // Kartesische Koordinaten zufällig in einem Rechteck wählen und überprüfen,
    // dass sie im Kreis liegen.
    do
    {
        cone = new Cone(Math.random() * 2 * radius - radius,
                        Math.random() * 2 * radius - radius);
    }
    // Mithilfe des Satz des Phytagoras überprüfen dass der Punkt im Kreis liegt
    while (cone.x * cone.x + cone.y * cone.y > radius * radius);
```

```
    data[i] = cone;
}
```

Anschließend erstelle ich mithilfe von Swing und `SpringLayout` die grafische Oberfläche, die den Benutzer die beiden KIs auswählen lässt. Sobald der Benutzer auf „Spiel starten“ clickt, starte ich die Simulation und übergebe dem Simulator die wichtigen Teile der GUI damit dieser diese automatisch updaten kann. Sobald der Simulator beendet, kann der Benutzer auf „Neues Spiel“ clicken, woraufhin ein neues `SceneData`-Objekt erstellt wird, der Benutzer andere KIs auswählen kann wenn er will, und das Spiel erneut starten kann:

```
startGame.addActionListener((ActionEvent e) -> new Thread(() -> {
    if (data.isNull())
    {
        data.randomCones(radius, cones);
        scene.repaint();
        result0.setBackground(Color.WHITE); result1.setBackground(Color.WHITE);
        result0.setText("0"); result1.setText("0");
        player0.setEnabled(true); player1.setEnabled(true);
        startGame.setText("Spiel starten");
    }
    else
    {
        startGame.setEnabled(false);
        player0.setEnabled(false); player1.setEnabled(false);
        try
        {
            AI ai0 = (player0.getSelectedIndex() == 0
                      ? scene.createUserAi()
                      : kiclasses[player0.getSelectedIndex() - 1].newInstance());
            AI ai1 = (player1.getSelectedIndex() == 0
                      ? scene.createUserAi()
                      : kiclasses[player1.getSelectedIndex() - 1].newInstance());
            Simulator s = new Simulator(data, ai0, ai1);
            s.startSimulation(scene, result0, result1);
        }
        catch (InstantiationException | IllegalAccessException | InterruptedException ex)
        {
            ex.printStackTrace();
        }
        data.clear();
        startGame.setText("Neues Spiel");
        startGame.setEnabled(true);
    }
}, "Simulator").start());
```

2.2 Die Spielsimulation

Für die Spielsimulation ist hauptsächlich die Klasse `Simulator` zuständig. Die Klasse nimmt im Konstruktor das `SceneData`-Objekt sowie eine Instanz der beiden KIs entgegen. Außerdem speichert die Klasse noch für jede KI die umgeworfenen Kegel und die verbrauchte Rechenzeit. Die Methode `startSimulation()` nimmt dann noch die entsprechenden GUI-Objekte entgegen und startet die Simulation im selben Thread. Neben der grafischen Oberfläche wird pro Spiel noch eine gif-Animation mithilfe von ImageMagick erstellt, dafür muss das Commandozeilenprogramm `convert` installiert sein.

Als erstes definiert die Klasse einen `boolean`, welche KI als nächstes dran ist. Bei `true` ist die erste KI, bei `false` die zweite KI dran. Danach loope ich solange, bis alle Kegel umgeworfen wurden, und

lasse die KI einen Zug machen. Dann speichere ich alle Kegel, die dabei umgeworfen wurden, in einem `Set`. Diese Kegel finde ich, indem ich, wie in der Lösungsidee beschrieben, die obere und untere Funktion des abgegebenen Wurfs herausfinde und die y-Werte vergleiche:

```
// die beiden Funktionen für den Zug herausfinden
Function<Double, Double> upper = SceneRenderer.getFunction(move, SceneRenderer.UPPER_FKT);
Function<Double, Double> lower = SceneRenderer.getFunction(move, SceneRenderer.LOWER_FKT);

// durch jeden Kegel durchloopen und evtl. umschmeißen
for (Cone cone : data.getCones())
{
    if (cone.isOverthrown()) continue;
    if ((cone.getY() <= upper.apply(cone.getX()))
     && (cone.getY() >= lower.apply(cone.getX())))
        overthrown.add(cone);
}
```

Anschließend muss ich nur noch die Punkte zu den Punkten der entsprechenden KI addieren und die Laufzeit speichern. Als Letztes wird noch die grafische Oberfläche geupdated; dann ist entweder das Spiel vorbei oder die nächste KI ist dran.

2.3 Die Spieldarstellung

Für die Spieldarstellung sind hauptsächlich zwei Klassen verantwortlich: `SceneRenderer` und `ScenePane`. Die Klasse `SceneRenderer` stellt verschiedene Methoden zur Verfügung, um ein `SceneData`-Objekt in ein `Graphics2D`-Objekt zu rendern; wohingegen die Klasse `ScenePane`, die die Klasse `JComponent` erweitert, ein `SceneData`-Objekt mithilfe der Methoden aus der Klasse `SceneRenderer` in eine Swing-GUI malt.

2.3.1 Die Klasse `SceneRenderer`

Als erstes definiert die Klasse `SceneRenderer` eine Methode namens `getFunction()`, die eine `Function<Double,Double>` für einen angegebenen `Move` und einen angegebenen z-Wert wie in der Lösungsidee beschrieben berechnet. Da ich hierfür trigonometrische Funktionen benutzen muss, muss der Winkel zwischen $\frac{\pi}{2}$ und $-\frac{\pi}{2}$ liegen, damit keine falschen oder negativen Werte herauskommen. Dann kann ich die Funktion berechnen:

```
double a = Math.tan(angle);
double b = move.getY() + z / Math.cos(angle) - a * move.getX();
return (Double x) -> a * x + b;
```

Danach definiert die Klasse noch jede Menge Methoden, die dazu dienen, ein `SceneData`-Objekt in ein `Graphics2D`-Objekt oder ein `BufferedImage` zu rendern. Dabei wird als erstes ein Gitter mit der Größe r auf den weißen Hintergrund gezeichnet. Dies ist sehr hilfreich, wenn man als Benutzer spielt, aber auch, um die Divider-KIs nachvollziehen zu können. Als nächstes wird, sofern dieser angegeben wurde, der `Move` auf das Gitter gezeichnet. Danach kommt der Kreis und als letztes noch die Kegel. Insgesamt sieht die `render()`-Methode, die von den restlichen Methoden aufgerufen wird, so aus:

```
public Point2D.Double render (@NonNull Graphics2D g, Move m,
                              double x0, double y0, double width, double height,
                              double zoom)
{
    // Hintergrund weiß malen
    g.setColor(Color.WHITE);
    g.fill(new Rectangle2D.Double(x0, y0, width, height));
```

```
    // Clippingbereich setzen
    Shape oldclip = g.getClip();
    g.setClip(new Rectangle2D.Double(x0, y0, width, height));

    // Werte berechnen
    double r = data.getRadius();
    double conesize = Math.max(5.0, zoom / 20.0);
    double xborder = (width - zoom * 2 * r) / 2.0;
    double yborder = (height - zoom * 2 * r) / 2.0;

    // ein Gitter mit dem Abstand des Radius zeichnen
    g.setColor(new Color(200, 200, 200, 100));
    for (int i = -(int)Math.floor(r); i <= r; i++)
    {
        double x = zoom(i, zoom, r, xborder);
        double y = zoom(i, zoom, r, yborder);
        g.draw(new Line2D.Double(x0 + x, y0 + yborder,
                                 x0 + x, y0 + yborder + r * 2 * zoom));
        g.draw(new Line2D.Double(x0 + xborder, y0 + y,
                                 x0 + xborder + r * 2 * zoom, y0 + y));
    }

    // wenn ein Move angegeben wurde, diesen zeichenen
    if (m != null)
    {
        // die Funktionen der drei Geraden herausfinden
        Function<Double, Double> upper = getFunction(m, UPPER_FKT);
        Function<Double, Double> middle = getFunction(m, MIDDLE_FKT);
        Function<Double, Double> lower = getFunction(m, LOWER_FKT);

        // Den grauen Bereich malen
        GeneralPath path = new GeneralPath();
        path.moveTo(x0 + xborder, y0 + zoom(upper.apply(-r), zoom, r, yborder));
        path.lineTo(x0 + xborder + r * zoom * 2,
                    y0 + zoom(upper.apply((double)r), zoom, r, yborder));
        path.lineTo(x0 + xborder + r * zoom * 2,
                    y0 + zoom(lower.apply((double)r), zoom, r, yborder));
        path.lineTo(x0 + xborder, y0 + zoom(lower.apply(-(double)r), zoom, r, yborder));
        path.lineTo(x0 + xborder, y0 + zoom(upper.apply(-(double)r), zoom, r, yborder));
        g.setColor(new Color(150, 150, 150, 100));
        g.fill(path);

        // Die Linie durch den Punkt, wo die Kugel hingeworfen wird, zeichnen
        g.setColor(Color.RED);
        g.setStroke(new BasicStroke((float)conesize));
        g.draw(new Line2D.Double(x0 + xborder,
                                 y0 + zoom(middle.apply(-(double)r), zoom, r, yborder),
                                 x0 + xborder + r * zoom * 2,
                                 y0 + zoom(middle.apply((double)r), zoom, r, yborder)));

        // Den Punkt, wo die Kugel hingeworfen wird, zeichnen
        g.setColor(Color.GREEN);
        g.fill(new Ellipse2D.Double(x0 + zoom(m.getX(), zoom, r, xborder) - conesize,
                                    y0 + zoom(m.getY(), zoom, r, yborder) - conesize,
                                    conesize * 2, conesize * 2));
    }

    // den Kreis zeichnen
```

```
    g.setColor(Color.BLACK);
    g.setStroke(new BasicStroke(2f));
    g.draw(new Ellipse2D.Double(x0 + xborder, y0 + yborder,
                                2 * data.getRadius() * zoom,
                                2 * data.getRadius() * zoom));

    // die Kegel zeichnen
    for (Cone d : data.getCones())
    {
        switch (d.getState())
        {
            case Cone.STANDING:
                g.setColor(Color.BLUE);
                break;
            case Cone.OVERTHROWING:
                g.setColor(Color.RED);
                break;
            default:
                continue;
        }
        g.fill(new Ellipse2D.Double(
                x0 + zoom(d.x, zoom, data.getRadius(), xborder) - conesize / 2.0,
                y0 + zoom(d.y, zoom, data.getRadius(), yborder) - conesize / 2.0,
                conesize, conesize));
    }

    // den alten Clippingbereich wiederherstellen
    g.setClip(oldclip);

    // den Mittelpunkt des Kreises zurückgeben
    return new Point2D.Double(x0 + xborder + r * zoom, y0 + yborder + r * zoom);
}

/**
 * Diese Methode zoomt den Punkt d und verschiebt den Koordinatenursprung an die
 * richtige Stelle.
 */
private double zoom (double d, double zoom, double radius, double border)
{
    return (d * zoom + radius * zoom + border);
}
```

2.3.2 Die Klasse `ScenePane`

Diese Klasse speichert ein `SceneRenderer`-Objekt sowie zoom und den Koordinatenursprung und stellt zum einen die für einen Swing-Komponenten wichtigen Methoden zur Verfügung und enthält zum anderen die Benutzer-KI. Die beiden Methoden für die GUI, `paintComponent()` und `getPreferredSize()`, sind eigentlich selbsterklärend:

```
@Override
protected void paintComponent (Graphics g1d)
{
    super.paintComponent(g1d);
    Graphics2D g = (Graphics2D)g1d;

    // Hintergrund weiß zeichnen
    g.setColor(Color.WHITE);
```

```
    g.fillRect(0, 0, getWidth(), getHeight());

    // SceneRenderer den Rest machen lassen
    synchronized (origin)
    {
        zoom = (Math.min(getWidth(), getHeight()) - 2 * defborder)
               / (2 * renderer.getData().getRadius());
        origin = renderer.render(g, move, getWidth(), getHeight(), zoom);
    }
}

@Override
public Dimension getPreferredSize ()
{
    return new Dimension(
            Math.round(2 * renderer.getData().getRadius() * defzoom + 2 * defborder),
            Math.round(2 * renderer.getData().getRadius() * defzoom + 2 * defborder));
}
```

Die Benutzer-KI funktioniert über einen `MouseListener`, der den zuletzt angeklickten Punkt speichert und sobald sich dieser ändert `notifyAll()` aufruft und somit der KI sagt, dass der Benutzer einen Punkt angeklickt hat. Nachdem der Benutzer zwei Punkte angeklickt hat, wird eine Gerade durch diese beiden Punkte gelegt und diese zurückgegeben. Das Ganze sieht so aus:

```
@SneakyThrows private Point getMouseClick ()
{
    if (mouseAdapter == null)
    {
        mouseAdapter = new MouseAdapter()
        {
            @Override public void mouseClicked (MouseEvent e)
            {
                synchronized (ScenePane.this)
                {
                    lastClick = new Point(e.getX(), e.getY());
                    ScenePane.this.notifyAll();
                }
            }
        };
        addMouseListener(mouseAdapter);
    }

    synchronized (this)
    {
        lastClick = null;
        while (lastClick == null) wait();
        Graphics g = getGraphics();
        g.setColor(Color.CYAN);
        g.fillOval(lastClick.x - 5, lastClick.y - 5, 11, 11);
        return lastClick;
    }
}

public AI createUserAi ()
{
    return new AI()
    {
        @Override public Move move (SceneData data)
        {
```

```
            Point point  = ScenePane.this.getMouseClick();
            Point second = ScenePane.this.getMouseClick();
            double x, y, sx, sy;
            synchronized (origin)
            {
                x  = (point.x  - origin.x) / zoom;
                y  = (point.y  - origin.y) / zoom;
                sx = (second.x - origin.x) / zoom;
                sy = (second.y - origin.y) / zoom;
            }
            return new Move(Math.atan((y - sy) / (x - sx)), x, y);
        }
        @Override public String getName () { return "Benutzer"; }
    };
}
```

2.3.3 Die Swing-GUI

Die Swing-GUI wird in der `main()`-Methode der Klasse `Main` erstellt. Dafür erstelle ich alle Komponenten und sortiere diese mithilfe von `SpringLayout`:

```
JFrame f = new JFrame("Panorama-Kegeln");

JPanel cp = new JPanel();
SpringLayout layout = new SpringLayout();
cp.setLayout(layout);
f.setContentPane(cp);

JButton startGame = new JButton("Spiel␣starten");
cp.add(startGame);
layout.putConstraint(SOUTH, startGame, -10, SOUTH, cp);
layout.putConstraint(HORIZONTAL_CENTER, startGame, 0, HORIZONTAL_CENTER, cp);

JLabel result0 = new JLabel("0");
result0.setOpaque(true);
result0.setHorizontalAlignment(SwingConstants.CENTER);
cp.add(result0);
layout.putConstraint(NORTH, result0, 0, NORTH, startGame);
layout.putConstraint(SOUTH, result0, 0, SOUTH, startGame);
layout.putConstraint(WEST, result0, 10, WEST, cp);
layout.putConstraint(EAST, result0, -10, WEST, startGame);

JLabel result1 = new JLabel("0");
result1.setOpaque(true);
result1.setHorizontalAlignment(SwingConstants.CENTER);
cp.add(result1);
layout.putConstraint(NORTH, result1, 0, NORTH, startGame);
layout.putConstraint(SOUTH, result1, 0, SOUTH, startGame);
layout.putConstraint(WEST, result1, 10, EAST, startGame);
layout.putConstraint(EAST, result1, -10, EAST, cp);

JComboBox<String> player0 = new JComboBox<>(kinames);
player0.setSelectedItem("Randy");
cp.add(player0);
layout.putConstraint(SOUTH, player0, -10, NORTH, startGame);
layout.putConstraint(WEST, player0, 10, WEST, cp);
layout.putConstraint(EAST, player0, -20, HORIZONTAL_CENTER, cp);
```

```
JComboBox<String> player1 = new JComboBox<>(kinames);
player1.setSelectedItem("Benutzer");
cp.add(player1);
layout.putConstraint(SOUTH, player1, 0, SOUTH, player0);
layout.putConstraint(EAST, player1, -10, EAST, cp);
layout.putConstraint(WEST, player1, 20, HORIZONTAL_CENTER, cp);

ScenePane scene = new ScenePane(data);
cp.add(scene);
layout.putConstraint(NORTH, scene, 10, NORTH, cp);
layout.putConstraint(SOUTH, scene, -10, NORTH, player0);
layout.putConstraint(WEST, scene, 10, WEST, cp);
layout.putConstraint(EAST, scene, -10, EAST, cp);
```

2.4 Die KIs

Ich habe, wie ich bereits erwähnt habe, mehrere KIs, die teilweise aufeinander aufbauen. Alle KIs sind im Paket `bwinf33_2.aufgabe2.ai.impl` enthalten. Dort sind auch weitere Klassen, die von den KIs teilweise benutzt werden. Die wichtigste Klasse davon ist `LinearFunction`, die ähnlich wie `SceneRenderer.getFunction()` eine Funktion speichert, die wie oben besprochen aufgebaut wird:

```
/**
 * Berechnet die Geradengleichung aus dem angegebenen Punkt und dem Winkel. Der Winkel
 * sollte dabei zwischen -PI/2 und PI/2 liegen.
 */
public LinearFunction (double x, double y, double angle)
{
    m = Math.tan(correctAngle(angle));
    b = y - m * x;
}

/**
 * Berechnet die Geradengleichung aus dem angegebenen Punkt und dem Winkel. Der Winkel
 * sollte dabei zwischen -PI/2 und PI/2 liegen.
 */
public LinearFunction (double x, double y, double angle, FunctionType z)
{
    angle = correctAngle(angle);
    m = Math.tan(angle);
    b = y + z.getZ() / Math.cos(angle) - m * x;
}

/**
 * Berechnet die Geradengleichung aus den angegebenen beiden Punkten.
 */
public LinearFunction (double x0, double y0, double x1, double y1)
{
    m = (y1 - y0) / (x1 - x0);
    b = y0 - m * x0;
}

/**
 * Berechnet die Geradengleichung aus den angegebenen beiden Punkten.
 */
public LinearFunction (@NonNull Point2D.Double p0, @NonNull Point2D.Double p1)
{
```

```
    this(p0.x, p0.y, p1.x, p1.y);
}
```

Das enum `FunctionType` orientiert sich dabei an der Lösungsidee und sieht wie folgt aus:

```
@AllArgsConstructor @ToString enum FunctionType
{
    UPPER_FCT(1),
    MIDDLE_FCT(0),
    LOWER_FCT(-1);

    @Getter private double z;
}
```

Neben natürlich der `y(x)`-Methode, die einfach $mx + b$ zurückgibt, enthält die Klasse noch eine Funktion, um die Schnittpunkte mit dem Kreis zu berechnen, wie ich es auch in der Lösungsidee beschrieben habe:

```
public Set<Point2D.Double> intersections (double r)
{
    Set<Point2D.Double> intersections = new HashSet<>();

    // Diskriminante der ABC-Formel ausrechnen
    double discriminant = m * m * r * r - b * b + r * r;

    // wenn die Diskriminante positiv ist gibt es mindestens eine Lösung
    if (discriminant >= 0)
    {
        discriminant = Math.sqrt(discriminant);
        double x = (discriminant - m * b) / (m * m + 1);
        intersections.add(new Point2D.Double(x, y(x)));

        // wenn die Diskriminante größer als 0 ist gibt es zwei Lösungen
        if (discriminant > 0)
        {
            x = (-discriminant - m * b) / (m * m + 1);
            intersections.add(new Point2D.Double(x, y(x)));
        }
    }

    return intersections;
}
```

Darüber hinaus gibt es noch die `Area`-Klasse, die nicht nur eine Funktion speichert, sondern nach Bedarf alle Funktionen berechnet. Interessant an diese Klasse ist aber die Möglichkeit auszurechnen, wieviele Kegel in der Area liegen. Dies mache ich nach demselben Prinzip wie auch bei der Simulation:

```
public int countCones (@NonNull SceneData data)
{
    int count = 0;
    for (Cone c : data.getCones())
    {
        if (!c.isOverthrown()
            && (c.getY() <= getUpper().y(c.getX()))
            && (c.getY() >= getLower().y(c.getX())))
            count++;
    }
    return count;
}
```

Die letzte interessante Klasse ist `Part`. Sie wird von den beiden Divider-KIs benutzt und speichert die Koordinaten, die Größe und die Anzahl der Kegel des Parts. Darüber hinaus verschiebt die Funktion `getMiddle()` evtl. den Mittelpunkt mithilfe von `LinearFunction.intersections()` so, dass er im Kreis liegt:

```
public Point2D.Double getMiddle (@NonNull LinearFunction fct, double r)
{
    double mx = x + width / 2.0, my = y + height / 2.0;
    if (mx * mx + my * my <= r * r)
        return new Point2D.Double(mx, my);

    Set<Point2D.Double> intersections = fct.intersections(r);
    for (Point2D.Double intersection : intersections)
        if (contains(intersection))
            return intersection;
    throw new IllegalPartException(/* … */);
}
```

2.4.1 Randy

Die KI „Randy“ ist die trivialste KI von allen. Wie in der Aufgabenstellung gefordert, sucht sich die KI per Zufall einen Punkt und einen Winkel aus und gibt diesen zurück. Das Ganze sieht dann ohne den sysout so aus:

```
@Override public Move move (SceneData data)
{
    double x, y;
    do
    {
        x = Math.random() * 2 * data.getRadius() - data.getRadius();
        y = Math.random() * 2 * data.getRadius() - data.getRadius();
    }
    while (x * x + y * y > data.getRadius() * data.getRadius());

    double angle = Math.random() * Math.PI;
    return new Move(angle, x, y);
}
```

2.4.2 Bruteforce

Die KI „Bruteforce“ macht ihrem Namen alle Ehre und „bruteforct“ nach dem besten Wurf. Die genesteten for-Schleifen sind selbsterklärend:

```
// Schrittweite bestimmen
double step = 1.0 / 3.0;
double angle_step = 180.0 / 2.0;

for (double x = -data.getRadius(); x <= data.getRadius(); x += step)
{
    for (double y = -data.getRadius(); y <= data.getRadius(); y += step)
    {
        for (double angle = 0; angle <= Math.PI; angle += Math.PI / angle_step)
        {
            int cones0 = new Area(x, y, angle).countCones(data);

            // evtl. diesen Wurf als besten speichern
```

```
            if (cones0 > cones)
            {
                move_x = x;
                move_y = y;
                move_angle = angle;
                cones = cones0;
            }
        }
    }
}
```

Anschließend wird der Wurf noch möglichst nahe an die Kreismitte verschoben, wie ich es in der Lösungsidee beschrieben habe:

```
LinearFunction fct = new LinearFunction(move_x, move_y, move_angle);
Set<Point2D.Double> intersections = fct.intersections(data.getRadius());
Point2D.Double result = null;
for (Point2D.Double p : intersections)
{
    System.out.println(Formatter.format(p));
    if (result == null)
        result = p;
    else
    {
        result.x = Math.min(result.x, p.x) + Math.abs(p.x - result.x) / 2.0;
        result.y = fct.y(result.x);
    }
}
```

2.4.3 Divider

Die KI „Divider“ teilt, wie ich in der Lösungsidee beschrieben habe, das Spielfeld in Parts ein und berechnet daraus den besten Wurf. Dabei handelt es sich um die hauptsächliche Idee aus der Lösungsidee ohne die Verbesserung. Die Parts erstelle ich iterativ, d.h. ich nehme einen größeren Part und viertele ihn, bis er so klein wie gewünscht ist. Dies geschieht so:

```
List<Part> parts[] = new List[DEPTH + 1];
parts[0] = Arrays.asList(
        new Part(-data.getRadius(), -data.getRadius(),
                 2 * data.getRadius(), 2 * data.getRadius(),
                 data.getCones()));
for (int depth = 1; depth <= DEPTH; depth++)
{
    parts[depth] = new ArrayList<>();
    for (Part part : parts[depth - 1])
    {
        List<Part> subs = new ArrayList<>(4);
        subs.add(new Part(part.x, part.y,
                          part.width / 2.0, part.height / 2.0));
        subs.add(new Part(part.x + part.width / 2.0, part.y,
                          part.width / 2.0, part.height / 2.0));
        subs.add(new Part(part.x, part.y + part.height / 2.0,
                          part.width / 2.0, part.height / 2.0));
        subs.add(new Part(part.x + part.width / 2.0, part.y + part.height / 2.0,
                          part.width / 2.0, part.height / 2.0));

        for (Cone c : part.cones)
        {
```

```
            if (c.isOverthrown()) continue;
            for (Part sub : subs)
                if ((c.getX() >= sub.x) && (c.getX() <= sub.x + sub.width)
                 && (c.getY() >= sub.y) && (c.getY() <= sub.y + sub.height))
                    sub.cones.add(c);
        }

        for (Part sub : subs)
            if (!sub.cones.isEmpty())
                parts[depth].add(sub);
    }
}
```

Anschließend sortiere ich die Parts mithilfe von `Collections.sort()` und hole mir, sofern diese existieren, die beiden besten Parts. Einen besten Part gibt es immer da sonst das Spiel vorbei wäre. Dann lege ich noch eine Gerade dadurch und kann dann das Ergebnis von `best.getMiddle(fct, data.getRadius())` zurückgeben:

```
Part best = parts[DEPTH].get(0);
Part second = (parts[DEPTH].size() > 1 ? parts[DEPTH].get(1) : null);
double x = best.x + best.width / 2.0;
double y = best.y + best.height / 2.0;
LinearFunction fct = new LinearFunction(x, y, 0);
if (second != null)
{
    double sx = second.x + second.width / 2.0;
    double sy = second.y + second.height / 2.0;
    fct = new LinearFunction(x, y, sx, sy);
}
```

2.4.4 SmartDivider

Die KI „SmartDivider“ implementiert die Verbesserung für die „Divider“-KI wie in der Lösungsidee beschrieben. Die KI ist unter dem Namen **power** von **meiserdo** auf dem Turnierserver hochgeladen. Das Aufteilen der Kegel in die Parts bzw. generell das erzeugen der Parts habe ich hier nicht iterativ vorgenommen, sondern ich erstelle direkt ein Array mit allen Parts und fülle diese dann mit den Kegeln:

```
// die Parts erstellen
Part parts[] = new Part[PARTS * PARTS];
for (int x = 0; x < PARTS; x++)
    for (int y = 0; y < PARTS; y++)
        parts[x * PARTS + y] = new Part(x - r, y - r);
// die Kegel in die Parts aufteilen
divide(parts, data, PARTS, r);
```

Die `divide()`-Methode sieht dabei so aus:

```
private void divide (Part parts[], SceneData data, int PARTS, int r)
{
    for (Cone c : data.getCones())
    {
        if (c.isOverthrown()) continue;
        parts[Math.min(PARTS - 1, (int)Math.floor(c.getX()) + r) * PARTS
              + Math.min(PARTS - 1, (int)Math.floor(c.getY()) + r)]
                .getCones().add(c);
    }
}
```

Anschließend ermittle ich die Zahl der sinvollen Parts, wobei diese maximal 7 betragen darf. Die sinvollen Parts finde ich indem ich schaue dass diese erstens mindestens einen Kegel enthalten und zweitens mindestens 70% der Kegelanzahl der besten Parts enthalten. Dann kann ich den Winkel „bruteforcen“:

```
double x = 0, y = 0, angle = 0;
int best = -1;
for (int i = 0; i < usefull; i++)
{
    for (double angle0 = 0; angle0 <= Math.PI; angle0 += Math.PI / 60.0) // Abstand: 3°
    {
        Area a = new Area(parts[i].getX(), parts[i].getY(), angle0);
        int conecount = a.countCones(data);
        if (conecount > best)
        {
            x = parts[i].getX();
            y = parts[i].getY();
            angle = angle0;
            best = conecount;
        }
    }
}
```

Dann kann ich die Gerade berechnen und den zurückgegebenen Punkt wie bei der „Bruteforce“-KI auch korrigieren:

```
LinearFunction fct = new LinearFunction(x, y, angle);
Set<Point2D.Double> intersections = fct.intersections(data.getRadius());
Point2D.Double result = null;
for (Point2D.Double p : intersections)
{
    System.out.println(Formatter.format(p));
    if (result == null)
        result = p;
    else
    {
        result.x = Math.min(result.x, p.x) + Math.abs(p.x - result.x) / 2.0;
        result.y = fct.y(result.x);
    }
}
```

Vorrausgesetzt `result` ist nicht `null` kann ich diesen Punkt zurückgeben, ansonsten einfach die vorherigen Werte zurückgeben.

2.5 Utilities

Als letztes gibt es noch die anderen beiden Klassen mit `main()`-Methode, die ich im Folgenden erklären werde:

2.5.1 `AiObfuscator`

Diese Klasse dient dazu, eine KI, die mit meinem Framework läuft, so umzuschreiben, dass der Turnierserver in der Lage ist, die KI zu kompilieren und korrekt auszuführen. Dazu gehört als erster Schritt delombok. Anschließend wird der gedelombokte Code genommen und mithilfe von einigen regulären Ausdrücken an den Turnierserver angepasst. Hier ist der gut kommentierte Code:

```
public class AiObfuscator
{
```

```
    private HashSet<String> aiimports = new HashSet<>();

    public File obfuscate (File dir, String file, boolean ai) throws IOException
    {
        // delombok aufrufen
        Delombok delombok = new Delombok();
        delombok.setOutput(new File("/tmp"));
        delombok.addFile(dir, file);
        delombok.delombok();

        // IO-Zeugs
        BufferedReader in = new BufferedReader(new FileReader(new File("/tmp", file)));
        File tmp = File.createTempFile("ai-", ".java");
        PrintWriter out = new PrintWriter(new FileWriter(tmp));
        if (ai)
            out.println("//␣Obfuscated␣for␣the␣BwInf␣Turnier␣Server␣by␣AiObfuscator");

        // bitte nix kompletterbulshitnonsense nennen ;)
        String className = "kompletterbulshitnonsense";
        String dataVarName = "kompletterbulshitnonsense";
        boolean inComment = false;
        String line;
        for (int linecount = 1; (line = in.readLine()) != null; linecount++)
        {
            //System.out.println("> " + line);
            //out.println("// " + file + ":" + linecount + ": " + line);

            // weitere benötigte Klassen einbinden
            Matcher m = Pattern.compile("\\s*//\\s*ai-import\\s+([a-zA-Z0-9_]+)")
                               .matcher(line);
            if (m.matches() && !aiimports.contains(m.group(1)))
            {
                aiimports.add(m.group(1));
                File f = obfuscate(dir, m.group(1) + ".java", false);
                Reader r = new FileReader(f);
                char buf[] = new char[8192];
                int read;
                while ((read = r.read(buf)) > 0)
                {
                    out.write(buf, 0, read);
                    out.flush();
                }

                r.close();
                f.delete();

                continue;
            }

            // Kommentare sind unnötig
            if (inComment)
            {
                m = Pattern.compile(".*\\*/(.*)").matcher(line);
                if (m.matches())
                {
                    line = m.group(1);
                    inComment = false;
                }
```

```
        else
            continue;
    }
    line = line.replaceAll("/\\*.*\\*/", "");
    m = Pattern.compile("(.*)/\\*.*$").matcher(line);
    if (m.matches())
    {
        line = m.group(1);
        inComment = true;
    }

    // nicht existierendes bzw. unnötiges Zeugs und singe-line Kommentare
    // ignorieren
    line = line.replaceAll("package\\s+[^;]*;", "");
    line = line.replaceAll("import\\s+(bwinf33_2|lombok)[^;]*;", "");
    line = line.replaceAll("implements\\s+AI", "");
    line = line.replaceAll("(^|\\s+|\\(|@)java.(lang|beans).([A-Z])", "$1$3");
    line = line.replaceAll("@(Override|NonNull)", "");
    if (!ai || !line.contains("delombok")) // delombok-Kommentar leben lassen
        line = line.replaceAll("//.*$", "");

    // Klassen- und Methodennamen korrigieren
    line = line.replaceAll("System.out.println\\((.+)\\);",
                           "zug.ausgabe(($1).toString());");
    line = line.replaceAll("(^|\\s+|\\(|<)" + className,
                           "$1AI");
    line = line.replaceAll("(^|\\s+|\\(|<)Cone",
                           "$1Spiel.Zustand.Kegel");
    line = line.replaceAll("(\\.|\\s+|\\()getX\\s*\\(\\s*\\)",
                           "$1xKoordinate()");
    line = line.replaceAll("(\\.|\\s+|\\()getY\\s*\\(\\s*\\)",
                           "$1yKoordinate()");
    line = line.replaceAll("(\\.|\\s+|\\()isOverthrown\\s*\\(\\s*\\)",
                           "$1umgeworfen()");
    line = line.replaceAll(dataVarName + "\\.getCones\\(\\)",
                           "zustand.listeKegel()");
    line = line.replaceAll(dataVarName + "\\.getRadius\\(\\)",
                           "zustand.listeKreis().get(0).radius()");
    line = line.replaceAll("(\\s+|\\(|,)" + dataVarName + "(\\s+|,|\\.|\\))",
                           "$1zustand$2");
    line = line.replaceAll("(\\.|\\s+|\\()getCones\\s*\\(\\s*\\)",
                           "$1listeKegel()");
    line = line.replaceAll(
            "(\\s*)return\\s*\\(?\\s*new\\s+Move\\s*\\("
            + "\\s*([a-zA-Z0-9_\\.]+)\\s*,\\s*([a-zA-Z0-9_\\.]+)\\s*,"
            + "\\s*([a-zA-Z0-9_\\.]+)\\s*\\)\\s*\\)?\\s*;",
            "$1{\n"
            + "$1\tdouble␣_degree_angle␣=␣Math.toDegrees($2);\n"
            + "$1\twhile␣(_degree_angle␣<=␣0)\n$1\t{\n"
              + "$1\t\t_degree_angle␣+=␣180.0;\n$1\t}\n"
            + "$1\twhile␣(_degree_angle␣>␣180)\n$1\t{\n"
              + "$1\t\t_degree_angle␣-=␣180.0;\n$1\t}\n"
            + "$1\tint␣_degree_angle_int␣=␣(int)Math.round(_degree_angle);\n"
            + "$1\tzug.ausgabe(\"zug.werfen(\"␣+␣_degree_angle_int␣+␣\","
              + "␣\"␣+␣$3␣+␣\",␣\"␣+␣$4␣+␣\");\");\n"
            + "$1\tzug.werfen(_degree_angle_int,␣(float)$3,␣(float)$4);\n"
            + "$1\treturn;\n"
            + "$1}");
```

```
        // die Klasse muss AI heißen
        m = Pattern.compile("(.*)public\\s+class\\s+([a-zA-Z0-9_]+)(\\s|\\{)(.*)$")
                .matcher(line);
        if (ai && m.matches())
        {
            className = m.group(2);
            line = m.group(1) + "public class AI /* obfuscated from " + className
                   + " */" + m.group(3) + m.group(4);
        }

        // die Methode muss zug heißen
        m = Pattern.compile("(\\s*)public\\s+Move\\s+move\\s*"
                            + "\\(\\s*SceneData\\s+([a-zA-Z0-9_]+)\\s*\\)(.*)")
                .matcher(line);
        if (ai && m.matches())
        {
            dataVarName = m.group(2);
            out.println(m.group(1)
                + "public void zug(int id, Spiel.Zustand zustand, Spiel.Zug zug)\n"
                + m.group(1) + "{\n"
                + m.group(1) + "\tlong time = System.currentTimeMillis();\n"
                + m.group(1) + "\tzug0(id, zustand, zug);\n"
                + m.group(1) + "\tzug.ausgabe(\"Zug beendet, time consumed: \""
                  + " + (System.currentTimeMillis() - time) + \" ms\");\n"
                + m.group(1) + "}\n"
                + m.group(1) + "@SuppressWarnings(\"all\")");
            line = m.group(1)
                   + "public void zug0 (int id, Spiel.Zustand zustand, Spiel.Zug zug)"
                   + m.group(3);
        }

        // alle weiteren SceneData-Onjekte ummatchen
        line = line.replaceAll("(^|\\s+|\\(|<)SceneData", "$1Spiel.Zustand");

        // Code-Sytle wiederherstellen
        if (line.trim().length() > 1)
        {
            line = line.replaceAll(
                    "^(\\s*)(if|else if|else|for|while)"
                    + "\\s+(\\([^\\(\\)]*(\\([^\\(\\)]*(\\([^\\(\\)]*\\)[^\\(\\)]*)*"
                    + "[^\\(\\)]*\\)[^\\(\\)]*)*[^\\(\\)]*\\))\\s*([^;]*;)\\s*$",
                    "$1$2 $3\n$1{\n$1\t$6\n$1}");
            line = line.replaceAll("^(\\s*)([^\\n]+)\\{([^\\n]+)\\}\\s*$",
                                   "$1$2\n$1{\n$1\t$3\n$1}");
            line = line.replaceAll("^(\\s*)([^\n]+)\\{\\s*$", "$1$2\n$1{");
        }

        // Leerzeile ignorieren
        if (!line.trim().isEmpty())
        {
            out.println(line);
            out.flush();
            //System.out.println("< " + line);
        }
    }

    System.err.println("Source file " + file + " successfully obfuscated.");
```

```
        out.close();
        in.close();

        return tmp;
    }

    public static void main (String args[]) throws IOException, InterruptedException
    {
        long time = System.currentTimeMillis();

        // KI-Klasse abfragen und obfuscaten
        String ai = args.length > 0
                    ? args[0]
                    : JOptionPane.showInputDialog(null, "AI Class Name:", "Choose AI …",
                                                  JOptionPane.QUESTION_MESSAGE);
        File tmp0 = new AiObfuscator().obfuscate(
                new File(System.getProperty("user.dir")
                + "/src/bwinf33_2/aufgabe2/ai/impl/"), ai + ".java", true);

        // imports korrigieren
        HashSet<String> imports = new HashSet<>();
        imports.add("java.beans"); // @ConstructorProperties
        {
            @Cleanup
            BufferedReader in = new BufferedReader(new FileReader(tmp0));
            String line;
            Pattern p = Pattern.compile("\\s*import\\s+([^;]+)\\.([a-zA-Z0-9_]+|\\*);");
            while ((line = in.readLine()) != null)
            {
                Matcher m = p.matcher(line);
                if (m.matches())
                    imports.add(m.group(1));
            }
        }
        File tmp = File.createTempFile("ai-", ".java");
        {
            @Cleanup
            BufferedReader in = new BufferedReader(new FileReader(tmp0));
            @Cleanup
            PrintWriter out = new PrintWriter(tmp);

            // die imports schreiben
            for (String pkg : imports)
            {
                out.println("import " + pkg + ".*;");
            }

            // den Rest schreiben
            String line;
            while ((line = in.readLine()) != null)
            {
                if (!line.trim().startsWith("import"))
                    out.println(line);
            }
        }

        System.out.println("Obfuscating finished in "
```

```
                            + (System.currentTimeMillis() - time) + "␣ms");

        // KI-Code öffnen
        Process p = new ProcessBuilder("xdg-open", tmp.getAbsolutePath()).start();
        p.waitFor();
    }
}
```

2.5.2 `AiTester`

Die Klasse `AiTester` dient, wie früher schon erwähnt, dazu, alle KIs mit demselben Ausgangszustand zu füttern und dadurch zu schauen, welche KI die beste ist. Anschließend werden alle Ausgangszustände und die Würfe der KIs in eine gif-Animation geschrieben. Dazu implementiert die Klasse selbst das Interface `AI` und spielt dabei die im Konstruktor übergebene KI, speichert aber noch weitere Informationen ab. Hier ist der kommentierte Quelltext:

```
@RequiredArgsConstructor @EqualsAndHashCode
@ToString(of = { "points", "time" })
public class AiTester implements AI, Comparable<AiTester>
{
    @NonNull @Getter private AI ai;
    @Getter private int points;
    @Getter @Setter private int lastpoints;
    @Getter private long time;
    @Getter @Setter private long lasttime;
    @Getter @Setter private Move lastmove;

    @Override public Move move (SceneData data)
    {
        return getAi().move(data);
    }

    @Override public String getName ()
    {
        return getAi().getName();
    }

    public int compareTo (@NonNull AiTester other)
    {
        return other.getPoints() - getPoints();
    }

    public static void main (String args[]) throws ReflectiveOperationException
    {
        // Die KIs suchen
        Reflections reflections = new Reflections("bwinf33_2.aufgabe2.ai.impl");
        Set<Class<? extends AI>> kiset = reflections.getSubTypesOf(AI.class);
        @SuppressWarnings("unchecked")
        Class<? extends AI>[] kiclasses = new Class[kiset.size()];
        int i = 0;
        for (Class<? extends AI> c : kiset)
        {
            kiclasses[i++] = c;
        }

        // Die KIs instantiieren
        AiTester ais[] = new AiTester[kiclasses.length];
```

```
        for (i = 0; i < kiclasses.length; i++)
        {
            ais[i] = new AiTester(kiclasses[i].newInstance());
        }

        // die Bilder in eine GIF-Datei schreiben
        LinkedList<String> images = new LinkedList<>();

        // KIs testen
        for (int r = 2; r <= 8; r++)
        {
            for (int c = 20; c <= 200; c += 20)
            {
                SceneData data = new SceneData(r, c);
                images.addLast(test(ais, data));

                Move m = new Move(Math.random() * Math.PI, 0, 0);
                Function<Double, Double> upper =
                        SceneRenderer.getFunction(m, SceneRenderer.UPPER_FKT);
                Function<Double, Double> lower =
                        SceneRenderer.getFunction(m, SceneRenderer.LOWER_FKT);
                data.getCones()
                    .stream()
                    .filter(cone -> (cone.getY() >= lower.apply(cone.getX()))
                                 && (cone.getY() <= upper.apply(cone.getX())))
                    .forEach(cone -> cone.setState(Cone.OVERTHROWN));
                images.addLast(test(ais, data));
            }
        }

        // Ergebnis ausgeben
        Arrays.sort(ais);
        for (AiTester ai : ais)
        {
            System.err.println("[AITESTER] »" + ai.getName() + "«: " + ai);
        }

        // den kompletten Test in einer GIF-Animation speichern
        try
        {
            String cmd[] = new String[images.size() + 6];
            cmd[0] = "convert";
            cmd[1] = "-loop";
            cmd[2] = "1";
            cmd[3] = "-delay";
            cmd[4] = "300";
            for (int j = 0; j < images.size(); j++)
            {
                cmd[j + 5] = images.get(j);
            }
            cmd[cmd.length - 1] = System.getProperty("user.home") + "/aitest-"
                                + System.currentTimeMillis() + ".gif";
            System.out.println(Arrays.toString(cmd));
            Process p = new ProcessBuilder(cmd).start();
            System.out.println(p.waitFor());
            for (String image : images)
            {
                new File(image).delete();
```

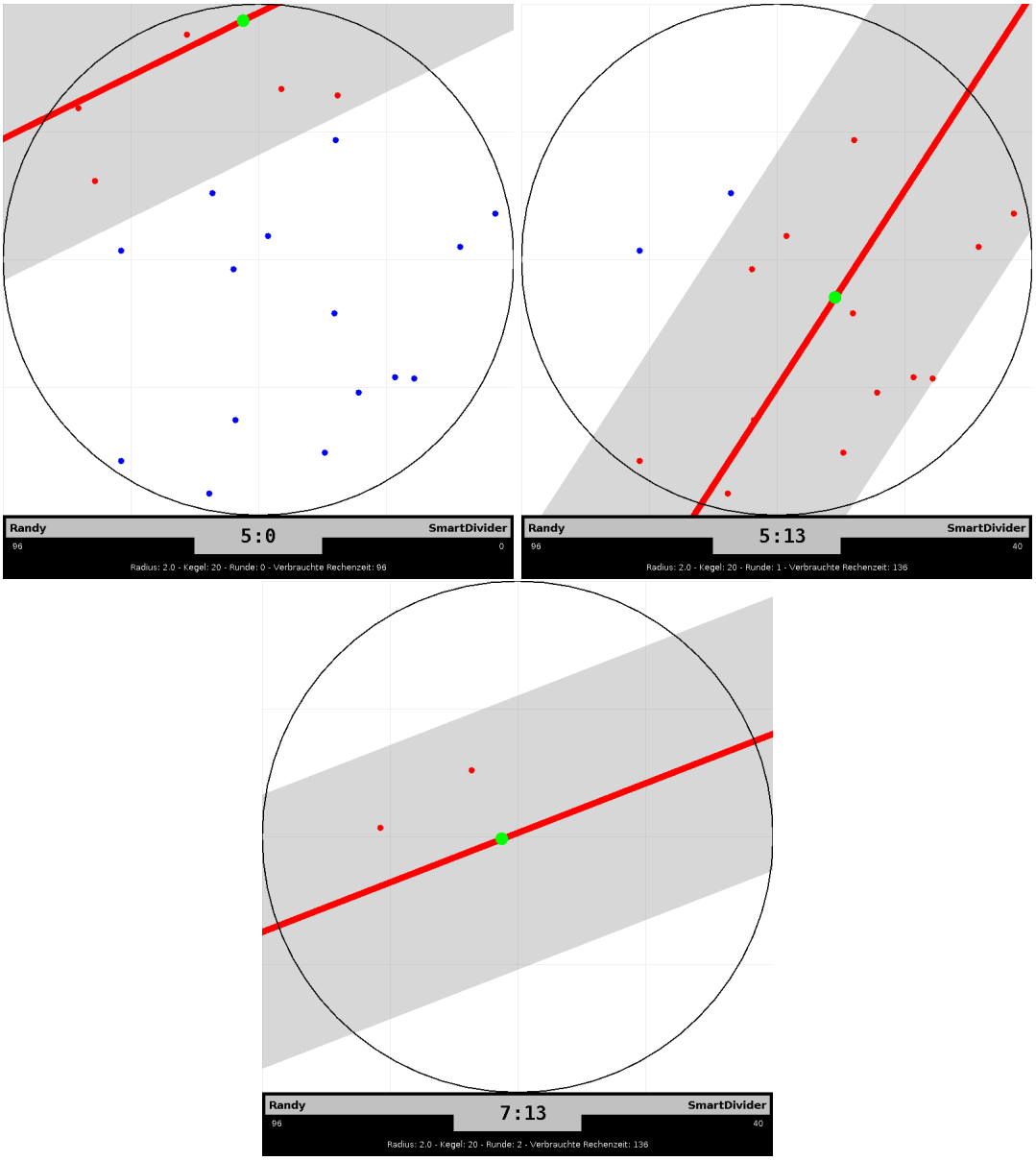
```
            }
            new ProcessBuilder("xdg-open", cmd[cmd.length - 1]).start();
        }
        catch (IOException | InterruptedException e)
        {
            e.printStackTrace();
        }
    }

    private static String test (AiTester ais[], SceneData data)
    {
        Arrays.asList(ais).parallelStream().forEach(ai ->
        {
            Stopwatch stopwatch = new Stopwatch();
            Move m = null;
            try
            {
                stopwatch.start();
                m = ai.move(data);
                stopwatch.stop();
            }
            catch (Exception e)
            {
                System.out.println("[AITESTER] Die KI »" + ai.getName()
                                   + "« hat eine Exception geworfen:");
                e.printStackTrace(System.out);
            }
            int overthrown = 0;
            if (m != null)
            {
                Function<Double, Double> upper =
                        SceneRenderer.getFunction(m, SceneRenderer.UPPER_FKT);
                Function<Double, Double> lower =
                        SceneRenderer.getFunction(m, SceneRenderer.LOWER_FKT);

                for (Cone cone : data.getCones())
                {
                    if (!cone.isOverthrown()
                        && (cone.getY() >= lower.apply(cone.getX()))
                        && (cone.getY() <= upper.apply(cone.getX())))
                        overthrown++;
                }
            }
            System.err.println("[AITESTER] Die KI »" + ai.getName()
                               + "« hat den Zug " + m
                               + " in " + stopwatch + " berechnet."
                               + " Umgeworfene Kegel: " + overthrown);
            ai.setLasttime(stopwatch.elapsedMillis());
            ai.setLastpoints(overthrown);
            ai.setLastmove(m);
            ai.time += stopwatch.elapsedMillis();
            ai.points += overthrown;
        });
        return drawPicture(data, ais);
    }

    @SneakyThrows
    private static String drawPicture (SceneData data, AiTester ais[])
```

```
    {
        SceneRenderer renderer = new SceneRenderer(data);
        File tmp = File.createTempFile("aitest-", ".png");

        int gridsize = (int)Math.ceil(Math.sqrt(ais.length));
        BufferedImage img = new BufferedImage(
                gridsize * 300, gridsize * 350,
                BufferedImage.TYPE_INT_ARGB);
        Graphics2D g = img.createGraphics();
        g.setColor(Color.LIGHT_GRAY);
        g.fillRect(0, 0, img.getWidth(), img.getHeight());

        Font f = new Font("SansSerif", Font.BOLD, 13);
        FontMetrics fm = g.getFontMetrics(f);

        for (int i = 0; i < ais.length; i++)
        {
            int row = i / gridsize;
            int col = i - row * gridsize;

            renderer.render(g, ais[i].getLastmove(), row * 300, col * 350, 300, 300);
            g.setFont(f);
            g.setColor(Color.BLACK);
            g.drawString(ais[i].getName(),
                         row * 300 + 10,
                         col * 350 + 310 + fm.getAscent());
            g.drawString(Integer.toString(ais[i].getLastpoints()),
                         row * 300 + 150 - fm.stringWidth(
                                 Integer.toString(ais[i].getLastpoints())) / 2f,
                         col * 350 + 310 + fm.getAscent());
            g.drawString(Long.toString(ais[i].getLasttime()),
                         row * 300 + 290 - fm.stringWidth(
                                 Long.toString(ais[i].getLasttime())),
                         col * 350 + 310 + fm.getAscent());
        }

        // Bild speichern
        ImageIO.write(img, "png", tmp);

        // Pfad zurückgeben
        return tmp.getAbsolutePath();
    }
}
```

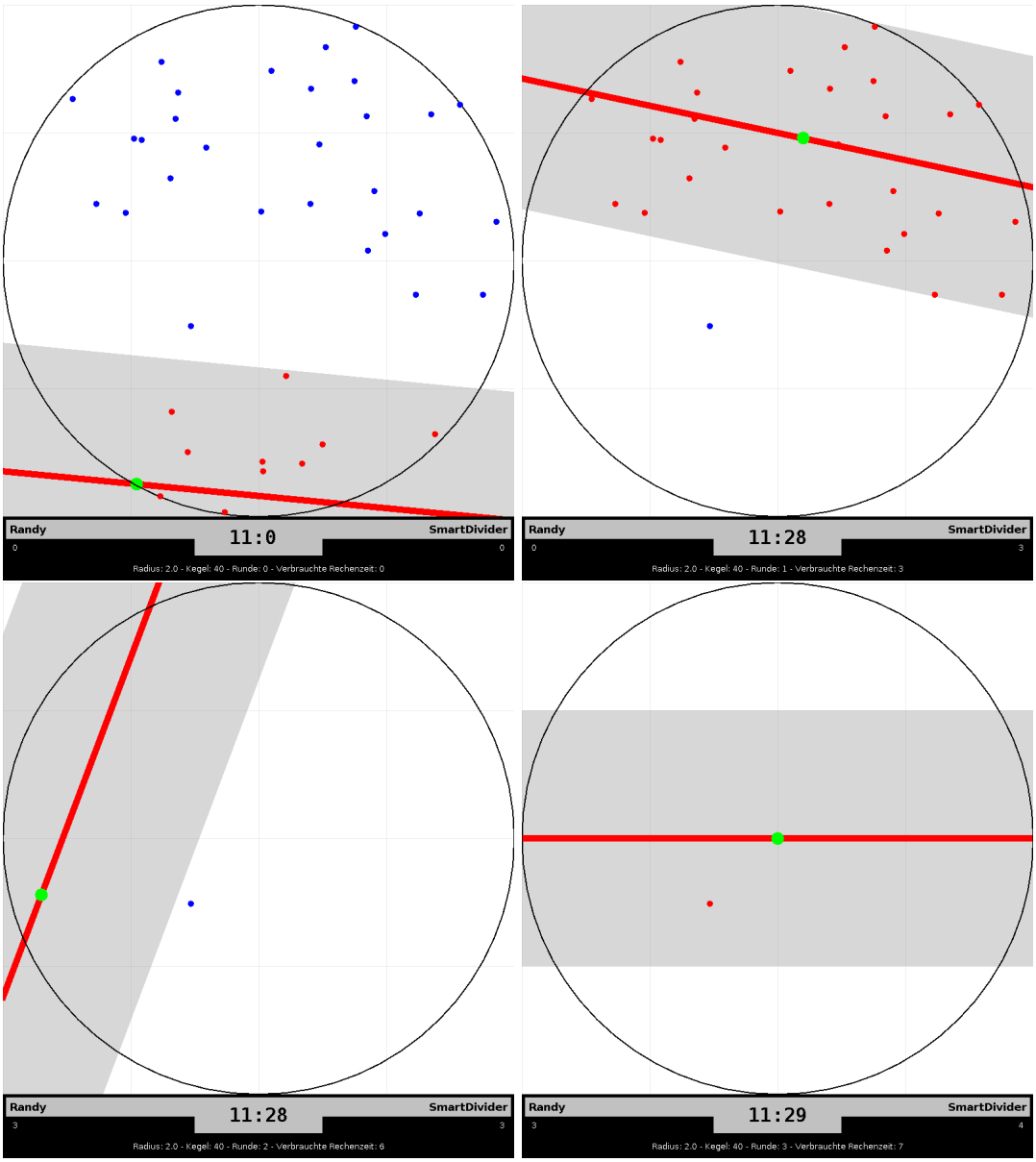
3 Beispiele

Hier habe ich ein paar Beispiele von `SmartDivider` gegen `Randy` mit verschieden Radien und Kegelzahlen gesammelt.

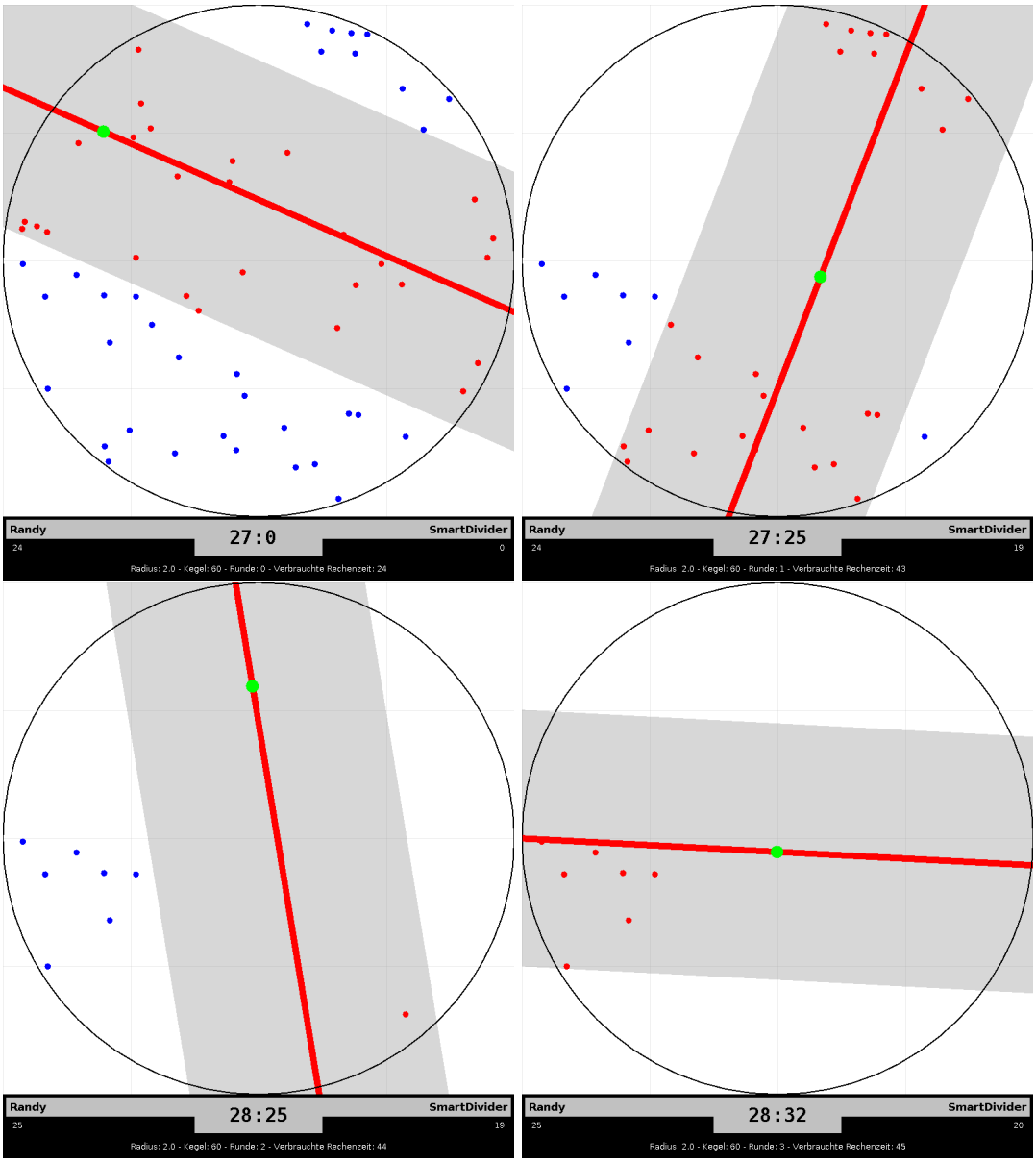
3.1 Beispiel R:2 K:20



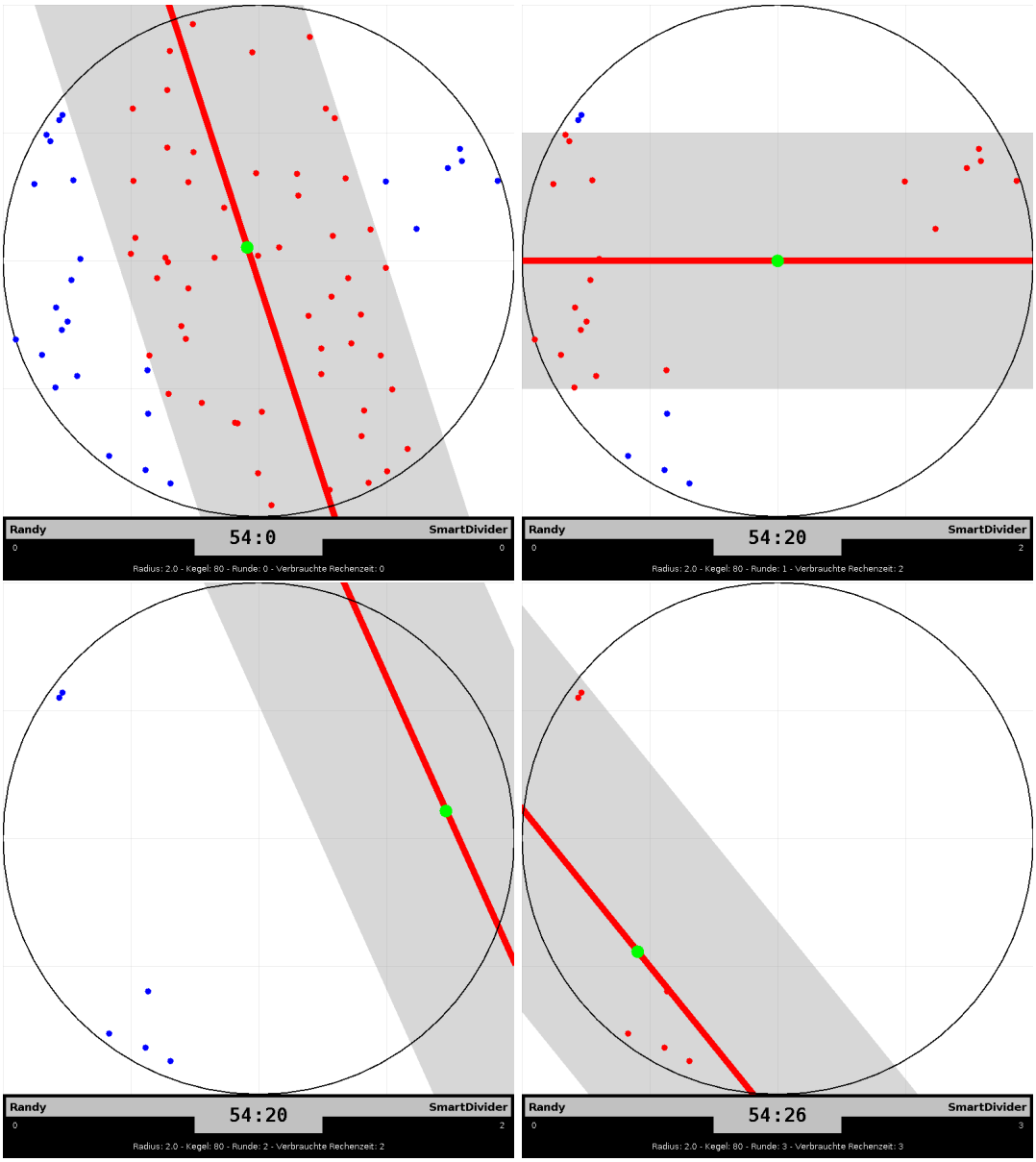
3.2 Beispiel R:2 K:40



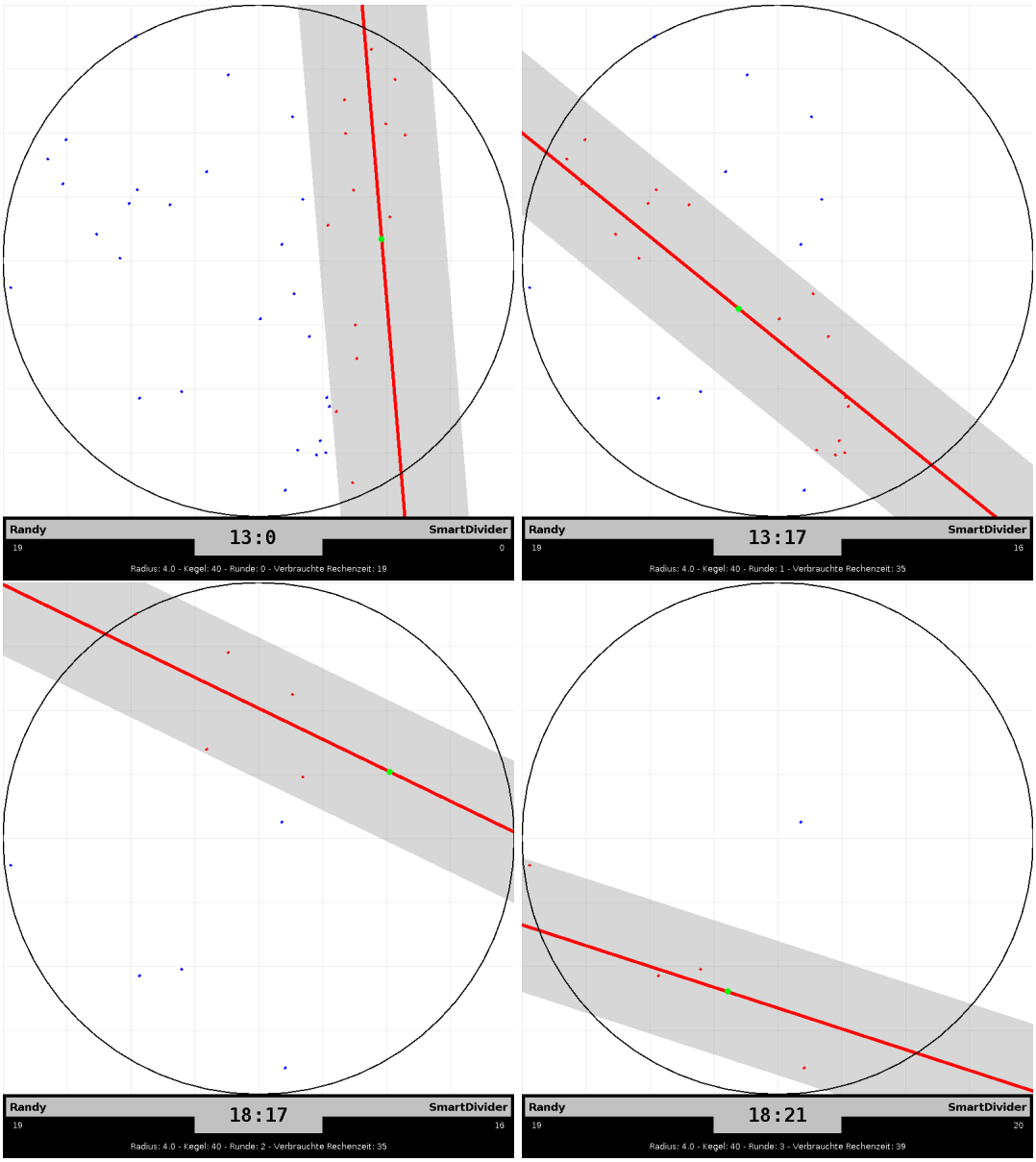
3.3 Beispiel R:2 K:60

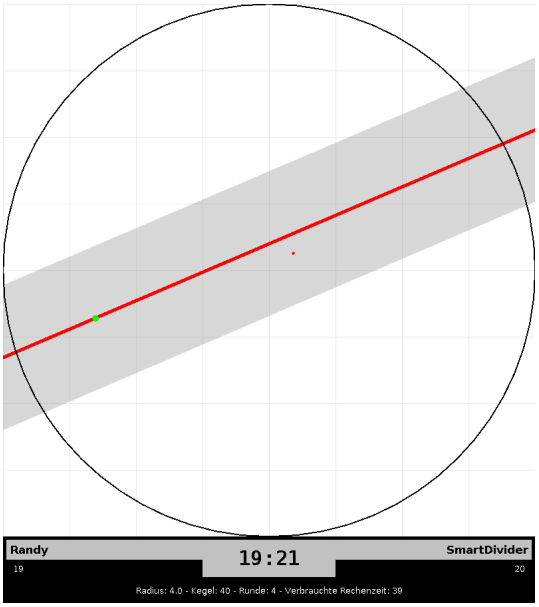


3.4 Beispiel R:2 K:80

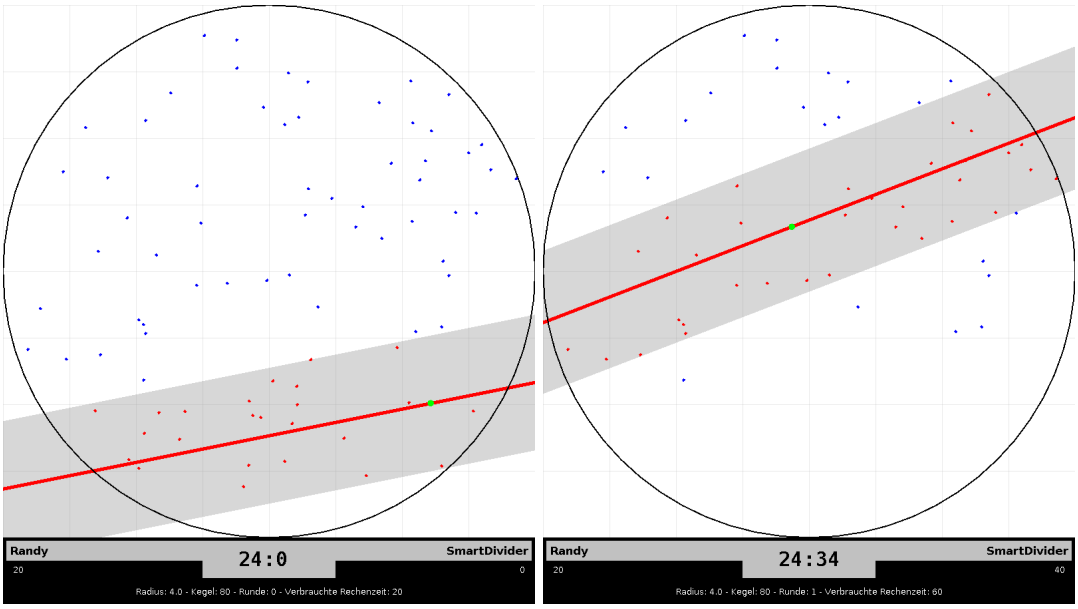


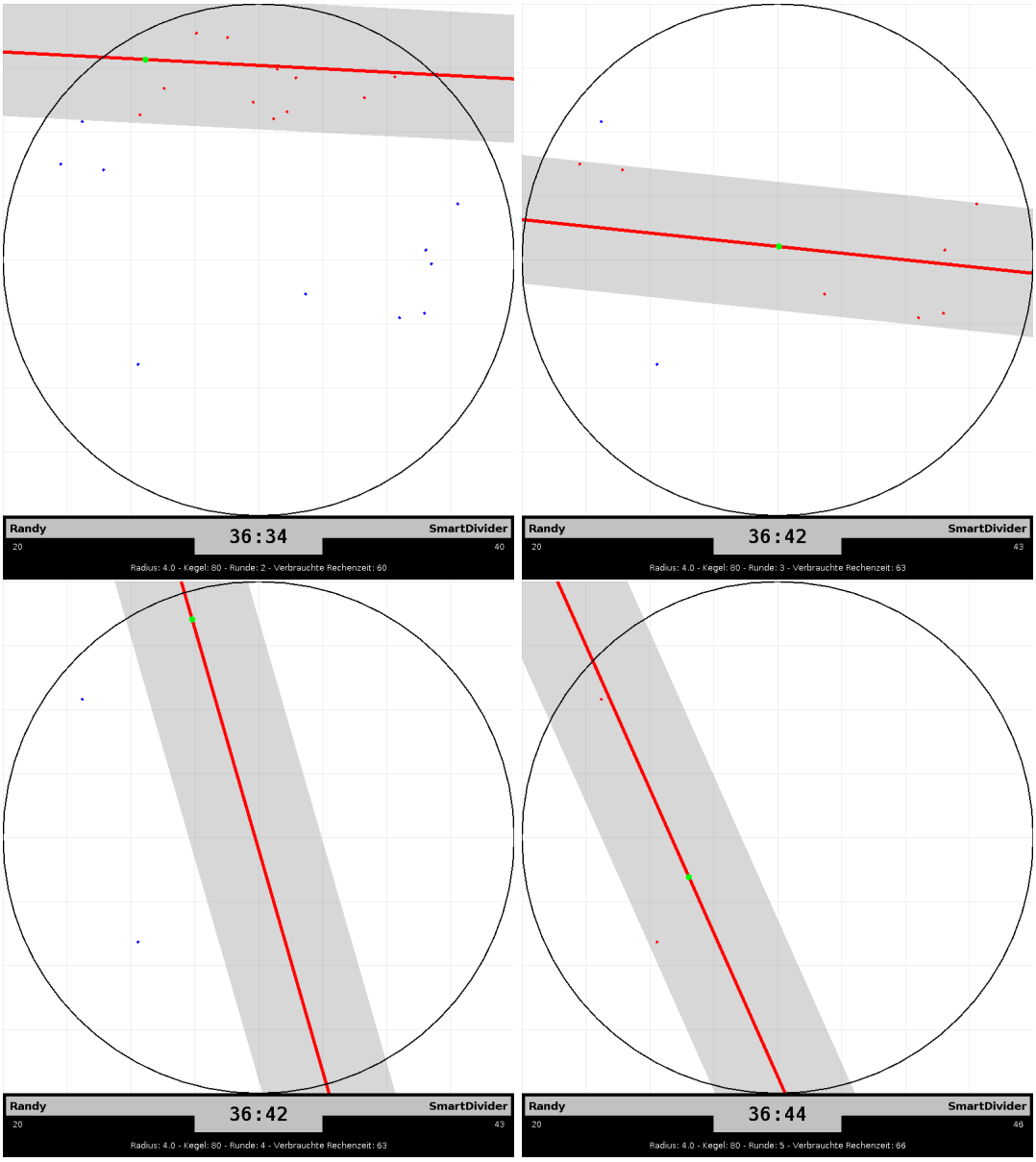
3.5 Beispiel R:4 K:40



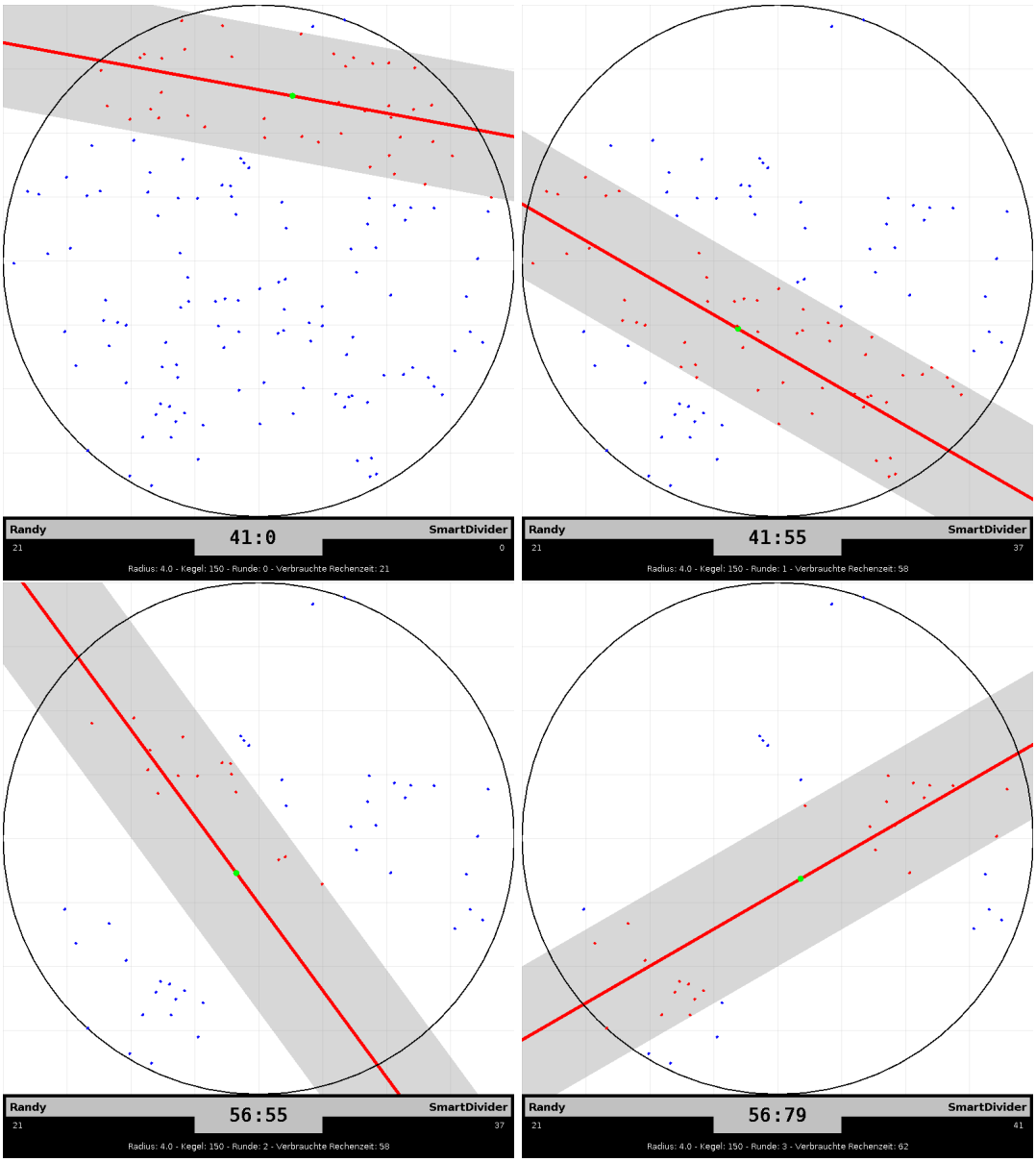


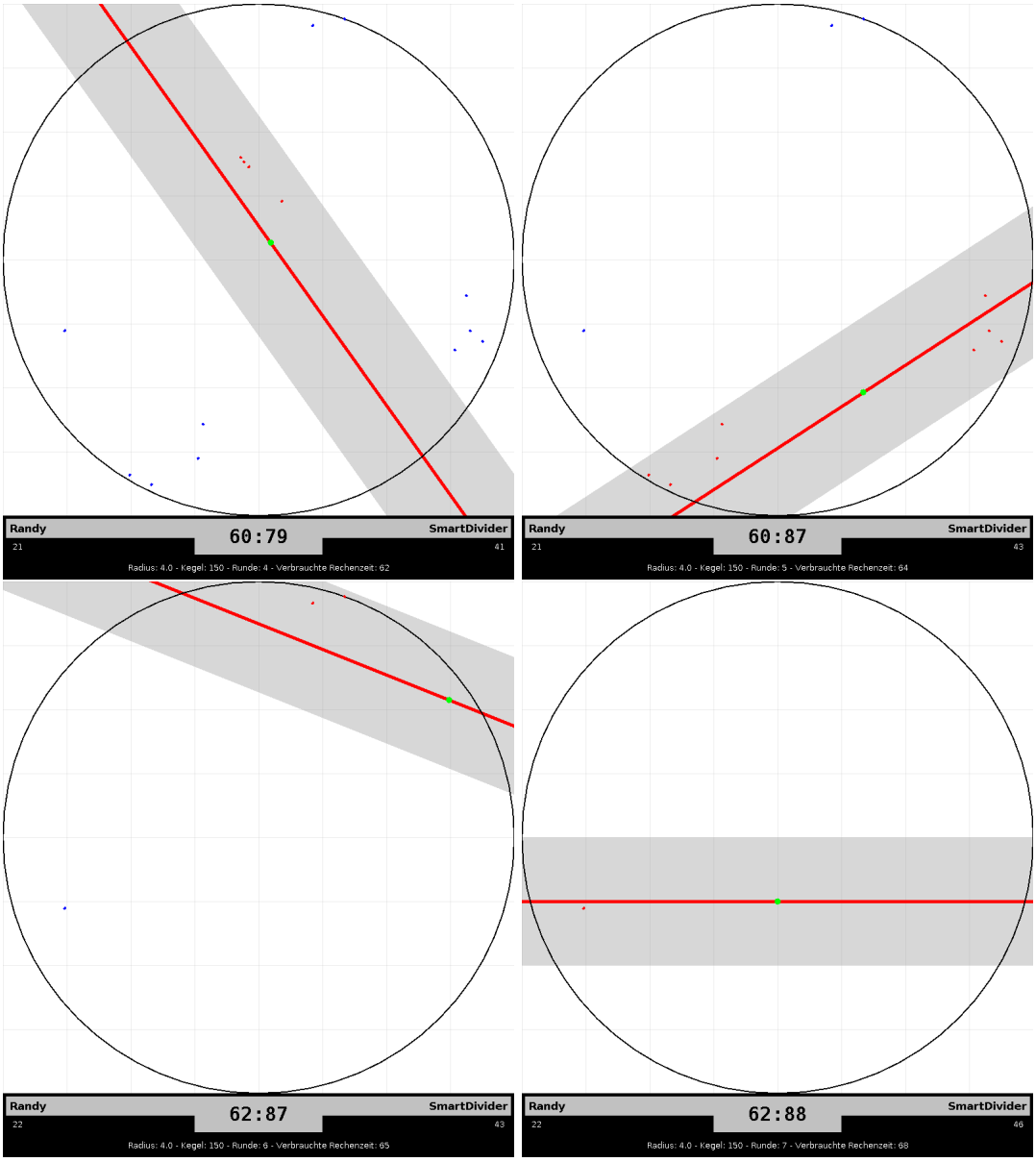
3.6 Beispiel R:4 K:80



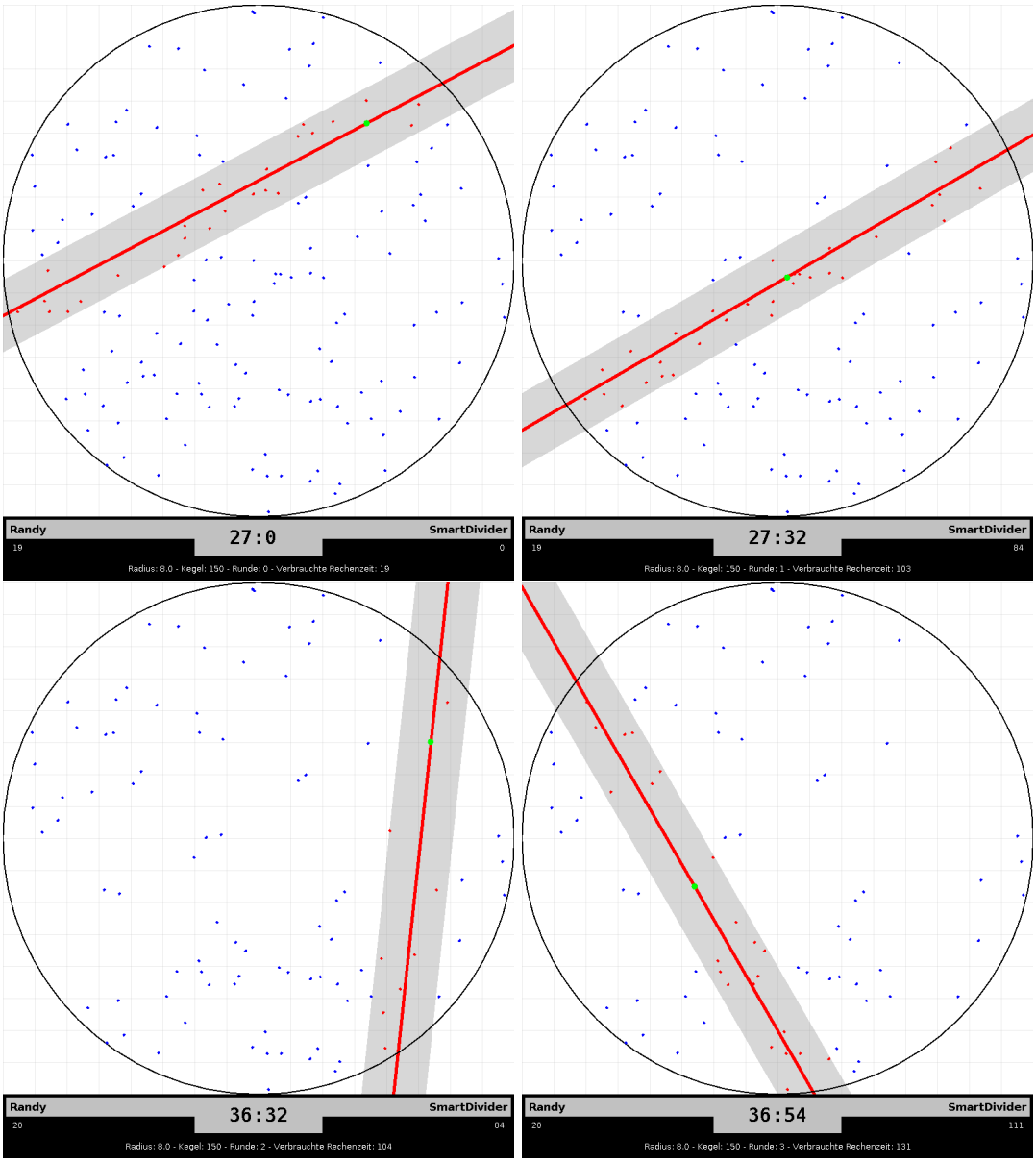

Randy
36:34
SmartDivider
20
40
Radius: 4.0 - Kegel: 80 - Runde: 2 - Verbrauchte Rechenzeit: 60
Randy
36:42
SmartDivider
20
43
Radius: 4.0 - Kegel: 80 - Runde: 3 - Verbrauchte Rechenzeit: 63
Randy
36:42
SmartDivider
20
43
Radius: 4.0 - Kegel: 80 - Runde: 4 - Verbrauchte Rechenzeit: 63
Randy
36:44
SmartDivider
20
46
Radius: 4.0 - Kegel: 80 - Runde: 5 - Verbrauchte Rechenzeit: 66

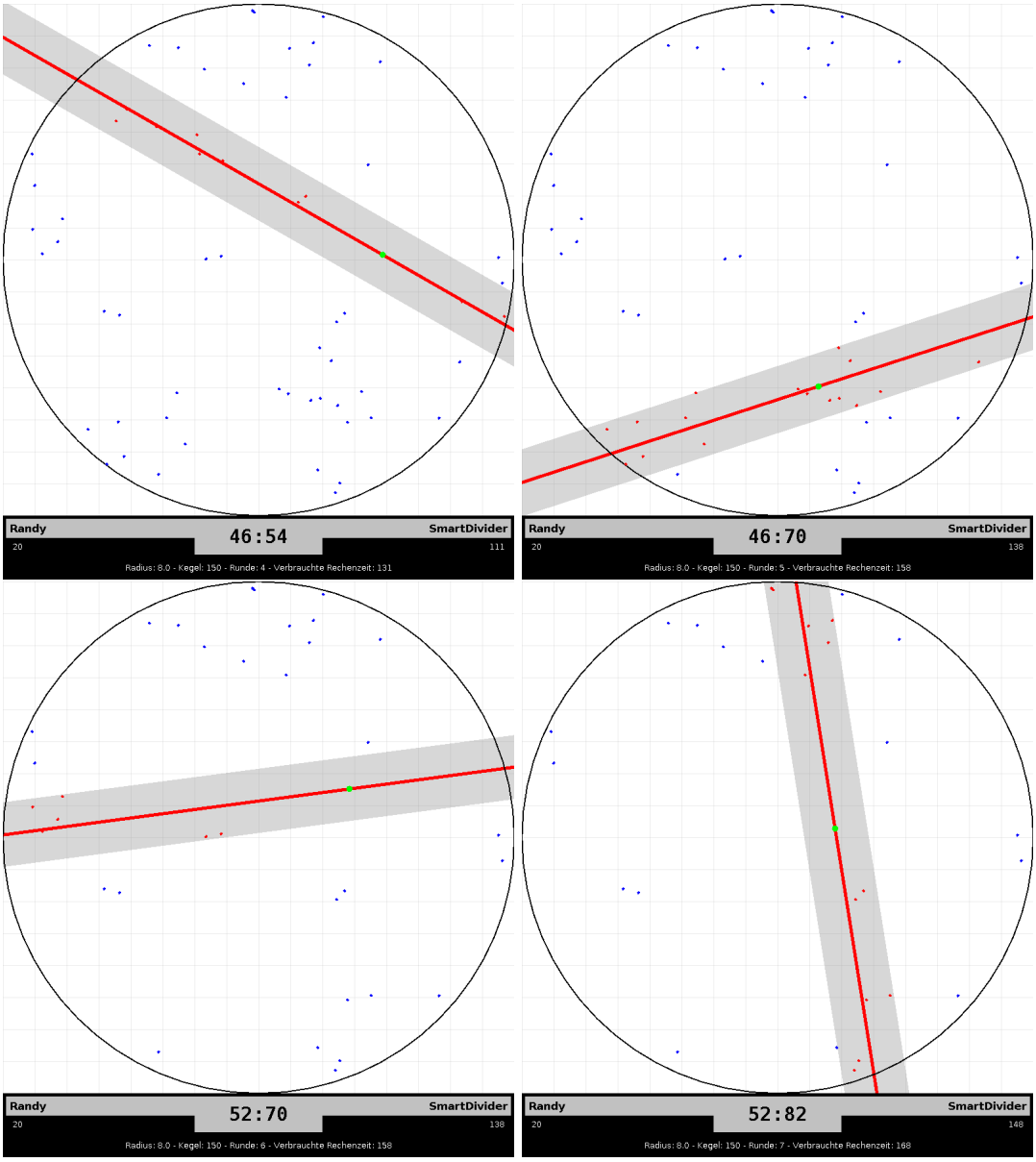
3.7 Beispiel R:4 K:150

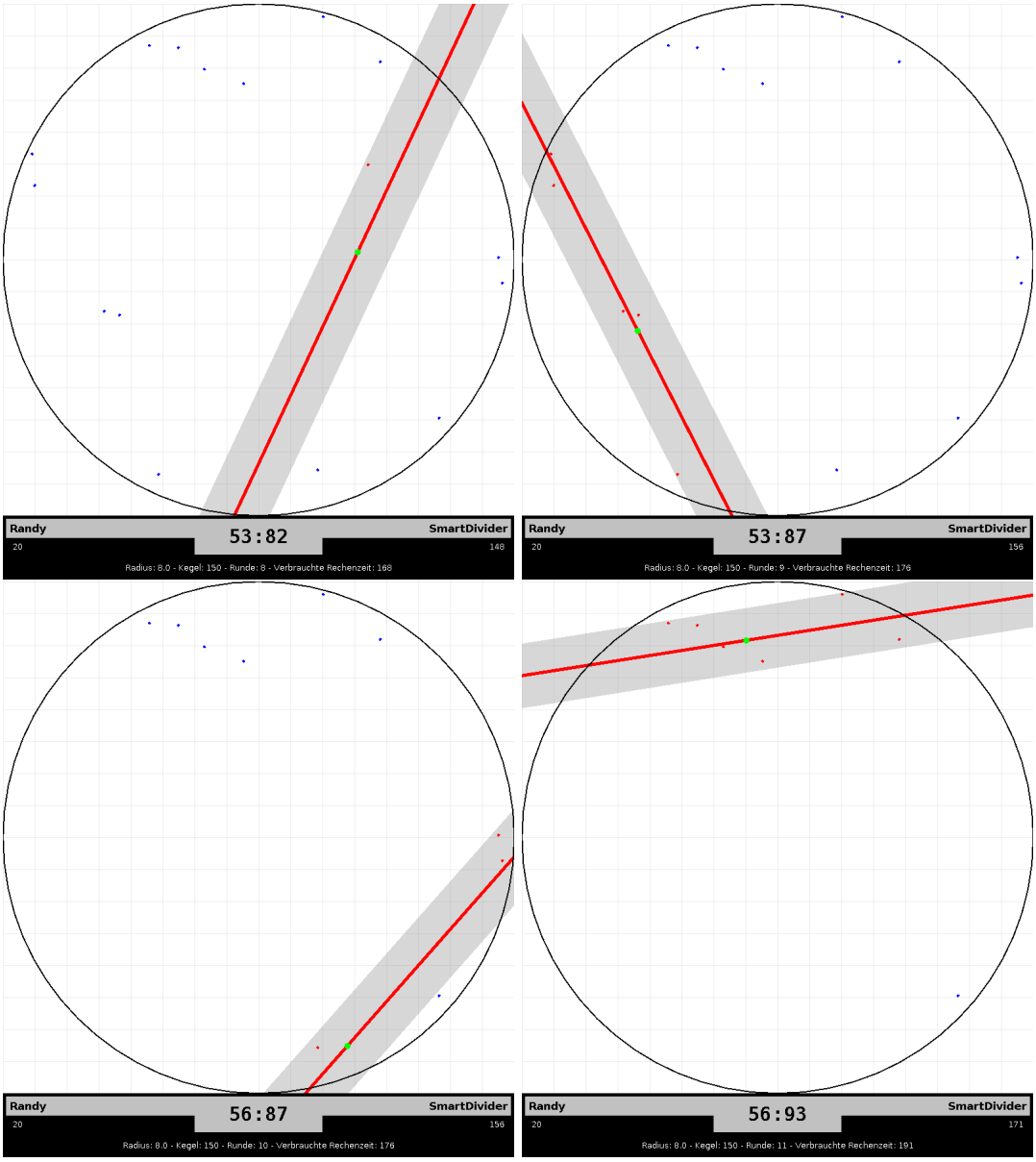


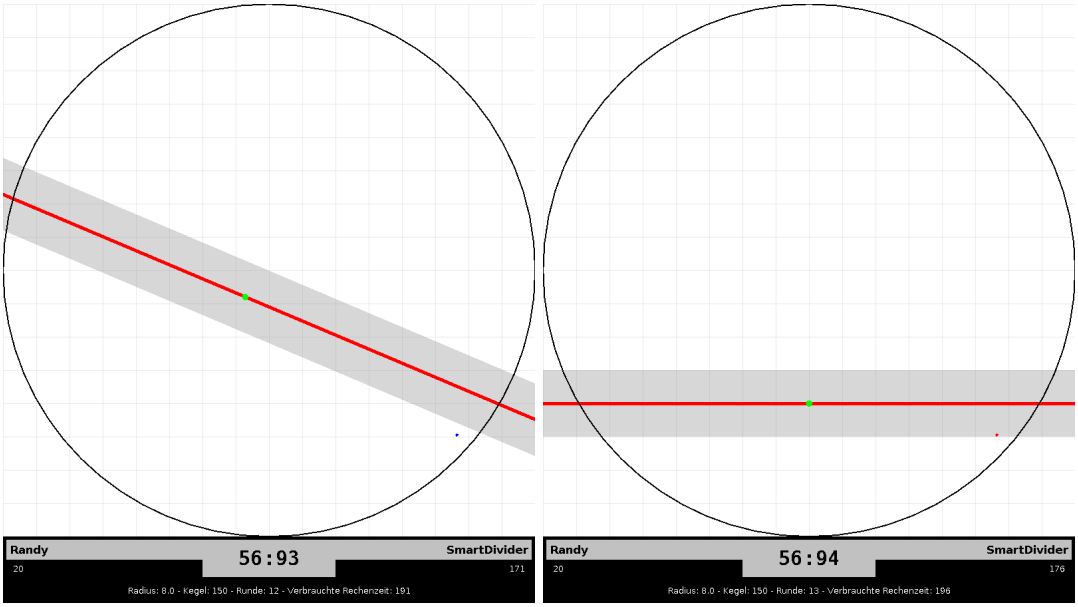

Randy
60:79
SmartDivider
21
41
Radius: 4.0 - Kegel: 150 - Runde: 4 - Verbrauchte Rechenzeit: 62
Randy
60:87
SmartDivider
21
43
Radius: 4.0 - Kegel: 150 - Runde: 5 - Verbrauchte Rechenzeit: 64
Randy
62:87
SmartDivider
22
43
Radius: 4.0 - Kegel: 150 - Runde: 6 - Verbrauchte Rechenzeit: 65
Randy
62:88
SmartDivider
22
46
Radius: 4.0 - Kegel: 150 - Runde: 7 - Verbrauchte Rechenzeit: 68

3.8 Beispiel R:8 K:150




Randy
46:54
SmartDivider
20
111
Radius: 8.0 - Kegel: 150 - Runde: 4 - Verbrauchte Rechenzeit: 131
Randy
46:70
SmartDivider
20
138
Radius: 8.0 - Kegel: 150 - Runde: 5 - Verbrauchte Rechenzeit: 158
Randy
52:70
SmartDivider
20
138
Radius: 8.0 - Kegel: 150 - Runde: 6 - Verbrauchte Rechenzeit: 158
Randy
52:82
SmartDivider
20
148
Radius: 8.0 - Kegel: 150 - Runde: 7 - Verbrauchte Rechenzeit: 168


Randy
53:82
SmartDivider
20
148
Radius: 8.0 - Kegel: 150 - Runde: 8 - Verbrauchte Rechenzeit: 168
Randy
53:87
SmartDivider
20
156
Radius: 8.0 - Kegel: 150 - Runde: 9 - Verbrauchte Rechenzeit: 176
Randy
56:87
SmartDivider
20
156
Radius: 8.0 - Kegel: 150 - Runde: 10 - Verbrauchte Rechenzeit: 176
Randy
56:93
SmartDivider
20
171
Radius: 8.0 - Kegel: 150 - Runde: 11 - Verbrauchte Rechenzeit: 191


Randy
56:93
SmartDivider
20
171
Radius: 8.0 - Kegel: 150 - Runde: 12 - Verbrauchte Rechenzeit: 191
Randy
56:94
SmartDivider
20
176
Radius: 8.0 - Kegel: 150 - Runde: 13 - Verbrauchte Rechenzeit: 196