# Aufgabe 4: Krocket

Team-ID: 00178

Team-Name: Team-Name

Bearbeiter/-innen dieser Aufgabe:
Matthew Greiner

November 18, 2024

## Contents



## 1 Lösungsidee / Ansatz

### 1.1 Annahme: Radius = 0

Um diese Aufgabe zu lösen, kann der Garten als zweidimensionales Koordinatensystem interpretiert werden. Die Krockettore sind dabei Strecken, die mit einem Start- und Endpunkt mit x- und y-Koordinaten dargestellt werden. Um herauszufinden, ob es möglich ist mit nur einem Schlag alle Tore in der richtigen Reihenfolge zu durchqueren, muss man schauen, ob es eine Gerade gibt, die alle gegebenen Tore bzw. Strecken schneidet. Wenn eine solche Gerade existert, dann entspricht diese der Bahn, die der Ball rollen müste, um alle Tore zu durchqueren. Der Startpunkt des Balls muss also auf dieser Gerade liegen (z.B. der Schnittpunkt von der Gerade mit dem ersten Tor), und die Schlagrichtung entspricht der Steigung der Gerade.
Um zu schauen, ob so eine Gerade existert, reicht es, nur Geraden anzuschauen, die durch einen Punkt auf dem ersten Tors und einen Punkt auf zweiten Tors verlaufen. Wenn keine dieser Geraden alle Tore in schneidet, ist es unmöglich, die Aufgabe mit einem einzigen Schlag zu lösen. Jede Gerade kann eindeutig durch zwei Punkte definiert werden: einen Punkt auf dem ersten Tor und einen Punkt auf dem zweiten Tor. Deswegen werden die Geraden gebildet, indem alle möglichen Punktkombinationen zwischen diesen beiden Toren geprüft werden.
Dann wird für jede dieser Geraden überprüft, ob sie alle Tore schneidet, wenn nein, wird die nächste Gerade überprüft, und wenn es keine Gerade gibt, gibt es keine Lösung für diese Aufgabe. Wenn eine Gerade gefunden wird, die alle Tore schneidet, ist diese Gerade der Weg des Balls, und Startpunkt und Schlagrichtung können direkt abgeleitet werden.

### 1.2 Annahme: Radius $\neq$ 0

Wenn der Radius $\neq 0$ ist, dann bleibt der Ansatz von (1.1) mit den Geraden gleich. Nur muss hier die Überprüfung, ob eine Gerade ein Tor schneidet, verändert werden. Denn wenn der Ball einen Radius hat, würde der Ball bei einigen Stellen nicht durch das Tor passen, auch wenn die aufgestellte Gerade das Tor schneidet. Um dieses Problem zu lösen, können Kreisgleichungen aufgestellt werden, die ihren Mittelpunkt bei dem Start und Endpunkt eines Tores, und den Radius von dem Ball haben. Wenn die zu überprüfende Gerade das Tor schneidet, und die beiden Kreise um die Pfosten des Tores nicht schneidet, dann passt der Ball durch das Tor. Wenn aber die Gerade das Tor schneidet, und auch mindestens einen der Kreise schneidet und/oder berührt, dann prallt der Ball an einem Posten ab und durchquert das Tor nicht. Somit kann diese Gerade als Lösung ausgeschlossen werden.

Ein simples Python Skript mit Matplotlib kann verwendet werden, um die Tore und die Lösungsgerade in einem Koordinatensystem darzustellen.

## 2 Umsetzung

### 2.1 Einlesung der Daten

Um die Daten aus einer Datei zu lesen, wird der Inhalt der Datei als ein String gespeichert, an den newline Zeichen ("\n") gespalten und jede Zeile als ein Element in einem String Array gespeichert. Jedes Element wird an den Leerzeichen gespalten, um an die Zahlenwerte zu kommen, die als Attribute in einem Gate Objekt gespeichert werden. Ein Gate Objekt hat jeweils einen Start und Endpunkt als Attribute. Punkte sind Objekte mit einer x- und y-Koordinate.

### 2.2 Finden einer Gerade

Um eine Gerade zu finden, die alle eingelesenen Tore schneidet, kann man iterativ arbeiten, indem man über alle möglichen Geraden iteriert und prüft, ob sie eine Lösung des Problems ist. Dies kann man mit drei verschachelten for-Schleifen machen. Ich gehe wie folgt vor:

1. Schrittweise über alle möglichen Startpunkte entlang des ersten Tors iterierieren (erste for-Schleife). Die Positionen auf dem ersten Tor sind in bestimmten Abständen (als Bruchteile der Länge des Tors) plaziert.
2. Für jeden dieser Punkte wird über alle möglichen Punkte (hier genannt Richtungspunkte) entlang des zweiten Tors iteriert (zweite for-Schleife), welche ebenfalls mit einem Abstand, das ein Bruchteil der Länge des Tors ist, entfernt sind.
3. Nun wird für jede Kombination aus Startpunkt und Richtungspunkt eine Gerade erstellt und überprüft, ob diese Gerade alle Tore schneidet. Es wird über alle Tore iteriert (dritte for-Schleife) und überprüft, ob **ein bestimmtes Tor geschnitten** wird oder nicht. Wenn das Tor geschnitten wird, wird mit dem nächsten Tor weitergemacht. Wenn nicht, dann wird aus dieser Schleife (dritte for-Schleife) gesprungen und mit dem nächsten Paar aus Start- und Richtungspunkt weitergemacht.

#### 2.2.1 Schnittpunktüberprüfung - Gerade mit einer Strecke

Um zu überprüfen, ob eine Strecke und eine Gerade sich schneiden, kann so vorgegangen werden: Wenn die Gerade und Strecke parallel ist, dann gibt es keinen Schnittpunkt. Wenn sie nicht parallel sind, und die Strecke als Gerade angesehen wird, existert ein Schnittpunkt. Daher muss nur überprüft werden, ob dieser Schnittpunkt innerhalb den Begrenzungen der Strecke (Pfosten des Tors) liegt.

Eine Gerade die durch den Punkt $(x_1, y_1)$ verläuft, und den Richtungsvektor $(dx, dy)$ hat, kann in Parameterform mit Parameter $t$ wie folgt dargestellt werden ($x$ und $y$ separat dargestellt, anstatt als Vektor):

$$x = x_1 + t \cdot dx \tag{1}$$

$$y = y_1 + t \cdot dy \tag{2}$$

In der Parameterdarstellung sieht die Strecke mit Startpunkt $(x_3, y_3)$ und Endpunkt $(x_4, y_4)$ und $u$ als Linearfaktor so aus:

$$x = x_3 + u \cdot (x_4 - x_3) \quad ; 0 \leq u \leq 1 \tag{3}$$
$$y = y_3 + u \cdot (y_4 - y_3) \quad ; 0 \leq u \leq 1 \tag{4}$$

Um nun die Schnittpunkte der Strecke und Gerade zu bekommen, werden die Gleichungen (1 und 3; 2 und 4) gleichgesetzt, wodurch sich ein Lineares Gleichungssystem bildet:

$$x_1 + t \cdot dx = x_3 + u \cdot (x_4 - x_3)$$
$$y_1 + t \cdot dy = y_3 + u \cdot (y_4 - y_3)$$

Durch umstellen, dass $t$ und $u$ auf der linken Seite sind und umformen in eine Matrixform, ergibt sich die Matrix $M$:

$$M = \begin{pmatrix} dx & -(x_4 - x_3) \\ dy & -(y_4 - y_3) \end{pmatrix} \cdot \begin{pmatrix} t \\ u \end{pmatrix} = \begin{pmatrix} x_3 - x_1 \\ y_3 - y_1 \end{pmatrix} \tag{5}$$

Daraus ergibt sich die Determinante der 2x2 Matrix als:

$$det(M) = dx \cdot (y_4 - y_3) - dy \cdot (x_4 - x_3)$$

Wenn $det(M) = 0$, dann ist die Gerade und die Strecke parallel oder kollinear zueinander (da die Richtungsvektoren linear abhängig sind). Somit kann es **keinen Schnittpunkt** geben.
Andernfalls, müssen $t$ (wo auf der Gerade der Schnittpunkt ist) und $u$ (zeigt ob der Schnittpunkt auf der Strecke liegt) berechnet werden.
Um die Parameter zu berechnen kann man mit dem Gauß-Verfahren vorgehen oder (wie ich jetzt) mit der Cramerschen Regel für 2x2 Matrixen:

Somit ergibt sich $t$ aus:

$$t = \frac{det(\begin{pmatrix} (x_3 - x_1) & -(x_4 - x_3) \\ (y_3 - y_1) & -(y_4 - y_3) \end{pmatrix})}{det(M)} = \frac{(x_3 - x_1) \cdot (y_4 - y_3) - (y_3 - y_1) \cdot (x_4 - x_3)}{det(M)}$$

Analog dazu ergibt sich $u$ folgendermaßen:

$$u = \frac{(x_3 - x_1) \cdot dy - (y_3 - y_1) \cdot dx}{det(M)}$$

Falls $0 \leq u \leq 1$, dann befindet sich der Schnittpunkt auf der Strecke. Falls das nicht der Fall ist, dann liegt er nicht auf der Strecke und ist für uns unrelevant.

**Unter der Annahme, dass der Ballradius null ist,** genügt es an dieser Stelle zu sagen, dass das Tor von dem Ball durchquert wird und es kann mit dem nächsten Tor weitergemacht werden.

**Wenn der Ballradius aber ungleich null ist,** muss überprüft werden, ob die Gerade die Kreise mit $r_{Kreis} = r_{Ball}$ um die Posten schneidet (Siehe **1.2**).

#### 2.2.2 Schnittpunktüberprüfung - Gerade mit einem Kreis

Ein Kreis mit dem Mittelpunkt $(c_x, c_y)$ und dem Radius $r$ wird mathematisch so beschrieben:

$$(x - c_x)^2 + (y - c_y)^2 = r^2 \tag{6}$$

Um die Schnittpunkte von einem Kreis und einer Gerade zu berechnen, muss die parametrische Form der Geradengleichung (1)/(2) in die Kreisgleichung (6) eingesetzt werden:

$$((x_1 + t \cdot dx) - c_x)^2 + ((y_1 + t \cdot dy) - c_y)^2 = r^2$$

Nachdem man das vereinfacht und ausmultipliziert, hat man eine quadratische Gleichung in $t$:

$$a \cdot t^2 + b \cdot t + c = 0$$

mit

$$\begin{aligned} a &= dx^2 + dy^2 \\ b &= 2 \cdot (dx \cdot (x_1 - c_x) + dy \cdot (y_1 - c_y)) \\ c &= (x_1 - c_x)^2 + (y_1 - c_y)^2 - r^2 \end{aligned}$$

die mit der quadratischen Lösungsformel berechnet werden kann:

$$t = \frac{-b \pm \sqrt{\text{Diskriminante}}}{2a}$$

Falls hierbei die **Diskriminante** $<$ **0** ist, schneidet die Gerade den Kreis nicht (für diese Aufgabe bedeutet das, dass der Ball das Tor durchqueren kann).
Wenn die Diskriminante $= 0$ ist, berührt die Gerade den Kreis und wenn sie $> 0$ ist, gibt es zwei Schnittpunkte von der Gerade und dem Kreis. In den letzteren zwei Fällen durchquert der Ball das Tor nicht, sondern prallt an einem Posten ab.
Die Diskriminante berechnet sich wie folgt:

$$\text{Diskriminante} = b^2 - 4ac$$

4. Wenn eine Gerade gefunden wurde, die alle Tore schneidet, wird sie als Resultat zurückgegeben. Der Startpunkt des Schlags von Laura ist der Schnittpunkt der Geraden mit dem ersten Tor und die Richtung des Schlags entspricht der Steigung der Gerade. Da aber eine Gerade, die parallel zur y-Achse ist, nicht existert und somit die Steigung davon nicht definiert wird, wird als Richtung des Schlags zusätzlich der normierte Richtungsvektor der Gerade ausgegeben.

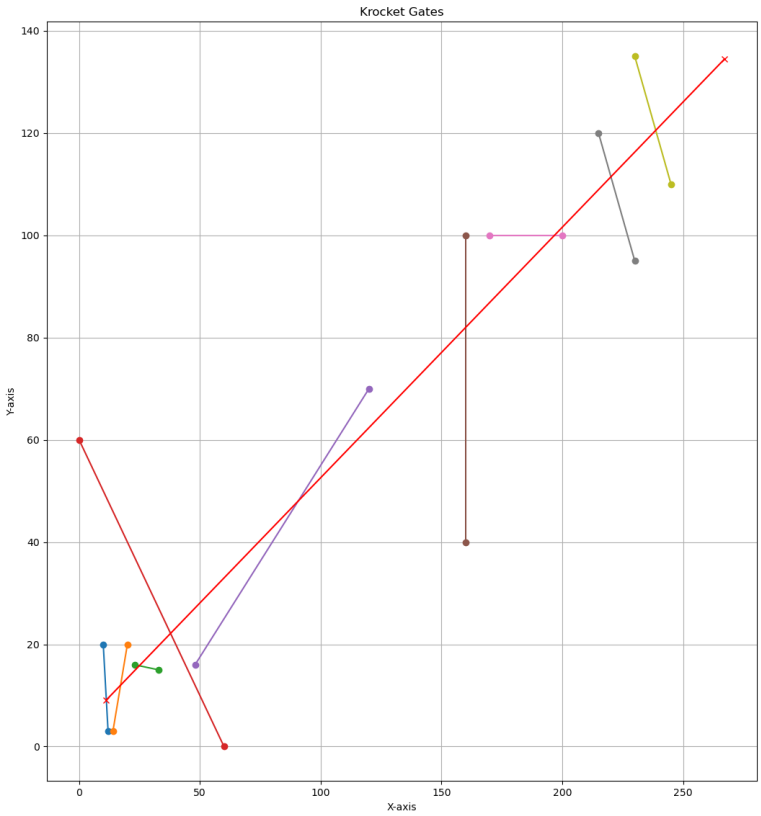
# 3 Beispiele

## 3.1 Beispiele der BwInf-Webseite

Hier die Ausgaben des Programmes zu den Beispielen auf der BwInf-Webseite: Zusätzlich wird das Ergebnis mit dem beigelegem python-skript und matplotlib visuell dargestellt. Hierbei ist die Rote Linie die Strecke des Balls von Laura, die alle Tore durchquert. (Falls es ein Ergebnis gibt).

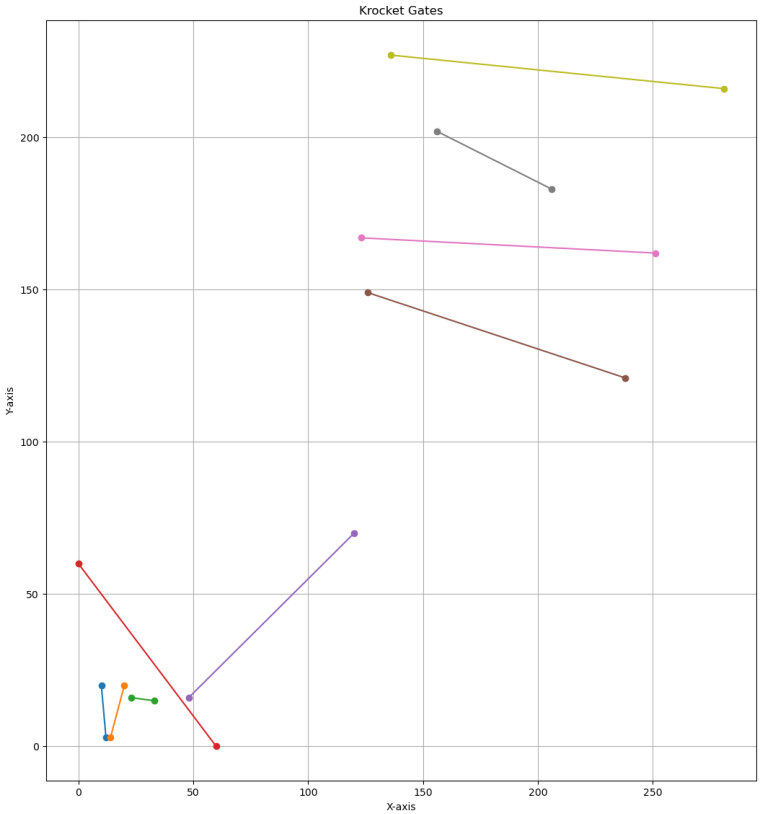
1. **krocket1.txt**
   Es ist möglich mit einem Schlag alle Tore zu durchqueren.
   Der Schlag kann als Halbgerade mit dem Startpunkt: (11.280000000000001|9.119999999999994), die durch den Punkt (17.18|12.009999999999994) verläuft, beschrieben werden.
   Normierter Richtungsvektor: (0.8980504366047625, 0.43989250199792623), bzw. Steigung der Halbgeraden: 0.4898305084745765

   Darstellung (python-skript)



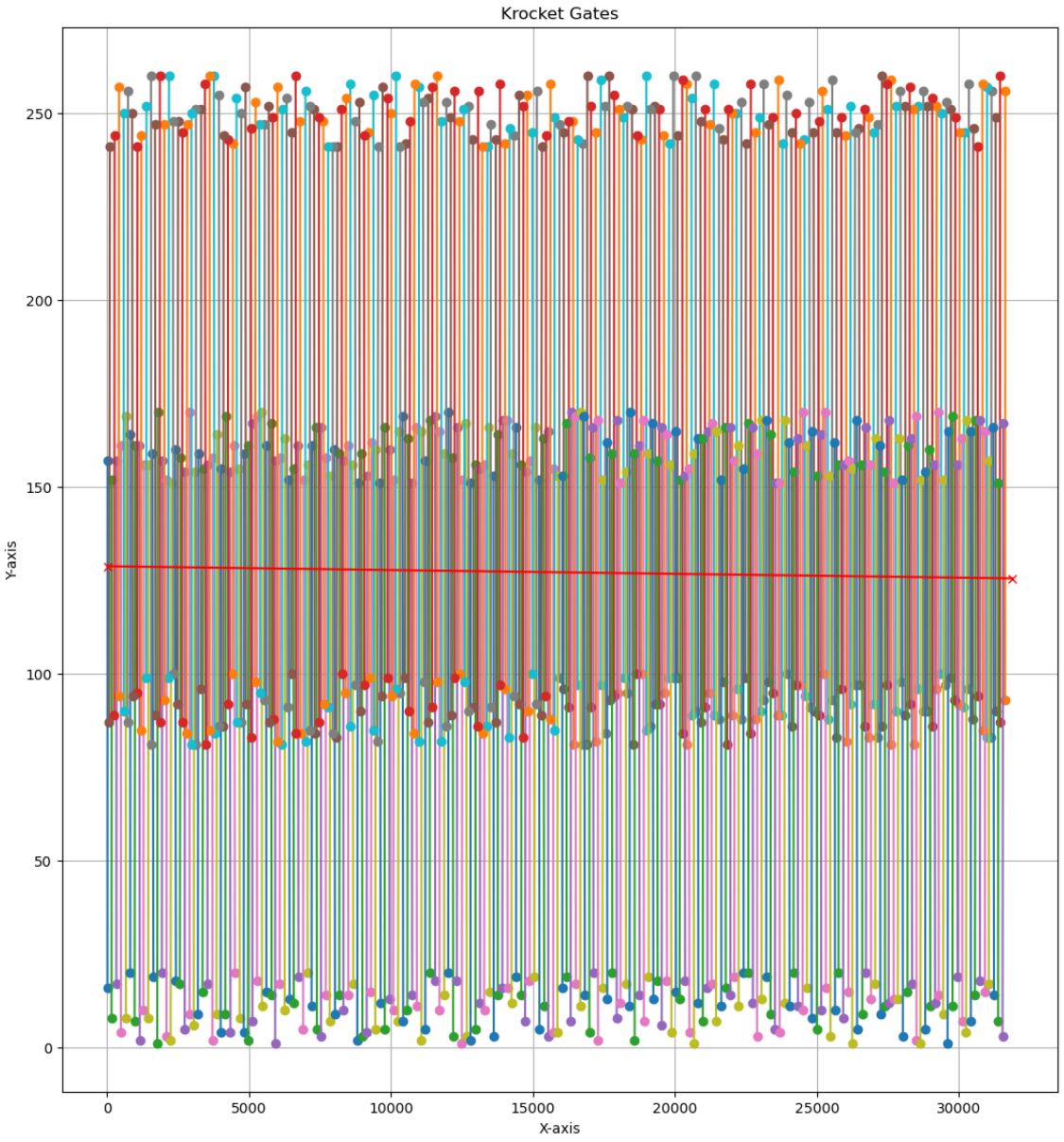

2. **krocket2.txt**
   Es ist nicht möglich, alle Tore in der richtigen Reihenfolge mit nur einem Schlag zu durchqueren.

   Darstellung (python-skript)



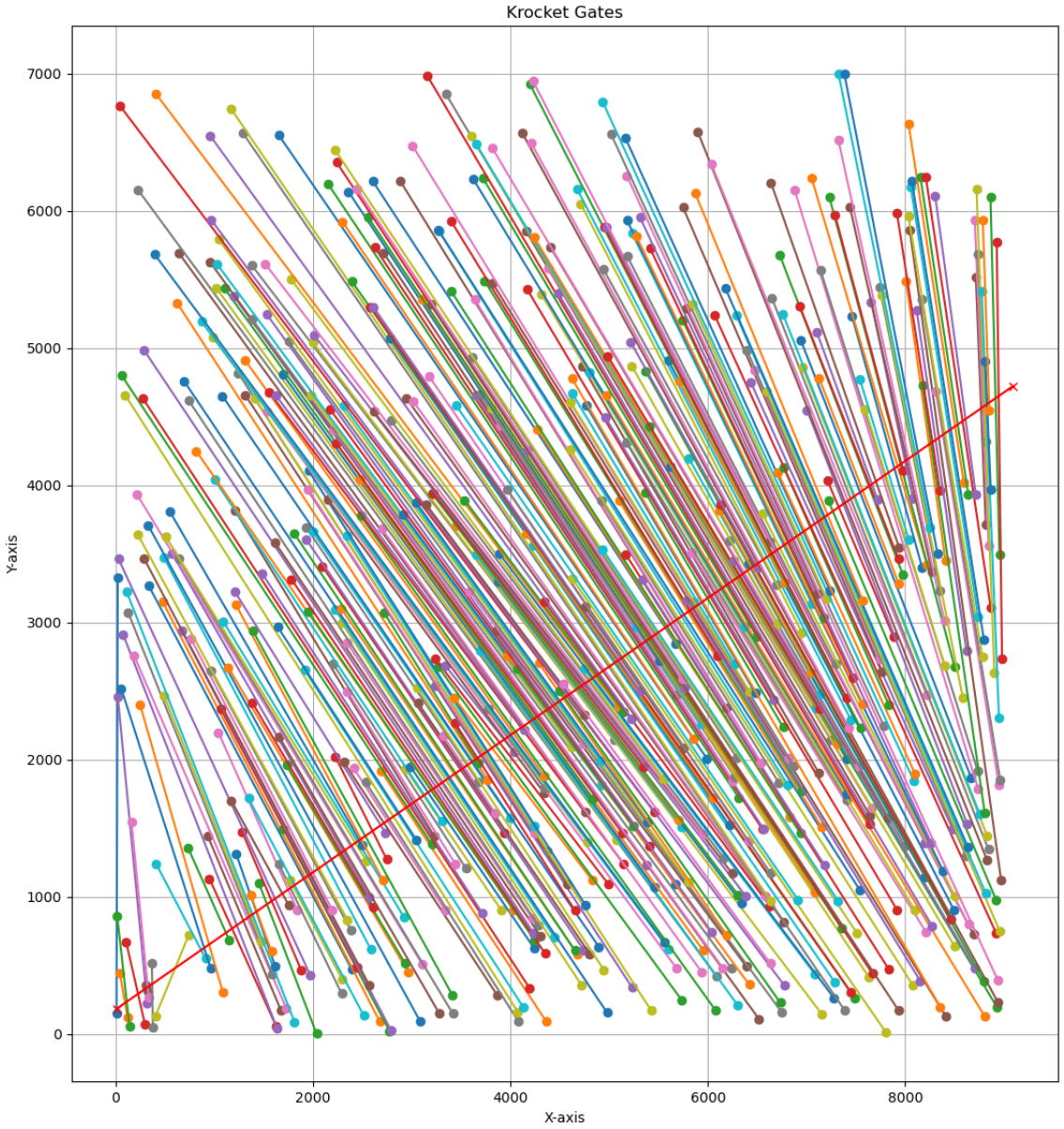
3. **krocket3.txt**
   Es ist möglich mit einem Schlag alle Tore zu durchqueren.
   Der Schlag kann als Halbgerade mit dem Startpunkt: (22.8|128.8), die durch den Punkt (31628.4|125.60000000000002) verläuft, beschrieben werden.
   Normierter Richtungsvektor: (0.9999999948744334, -1.0124787960355681E-4), bzw. Steigung der Halbgeraden: -1.0124788012250957E-4

   Darstellung (python-skript)

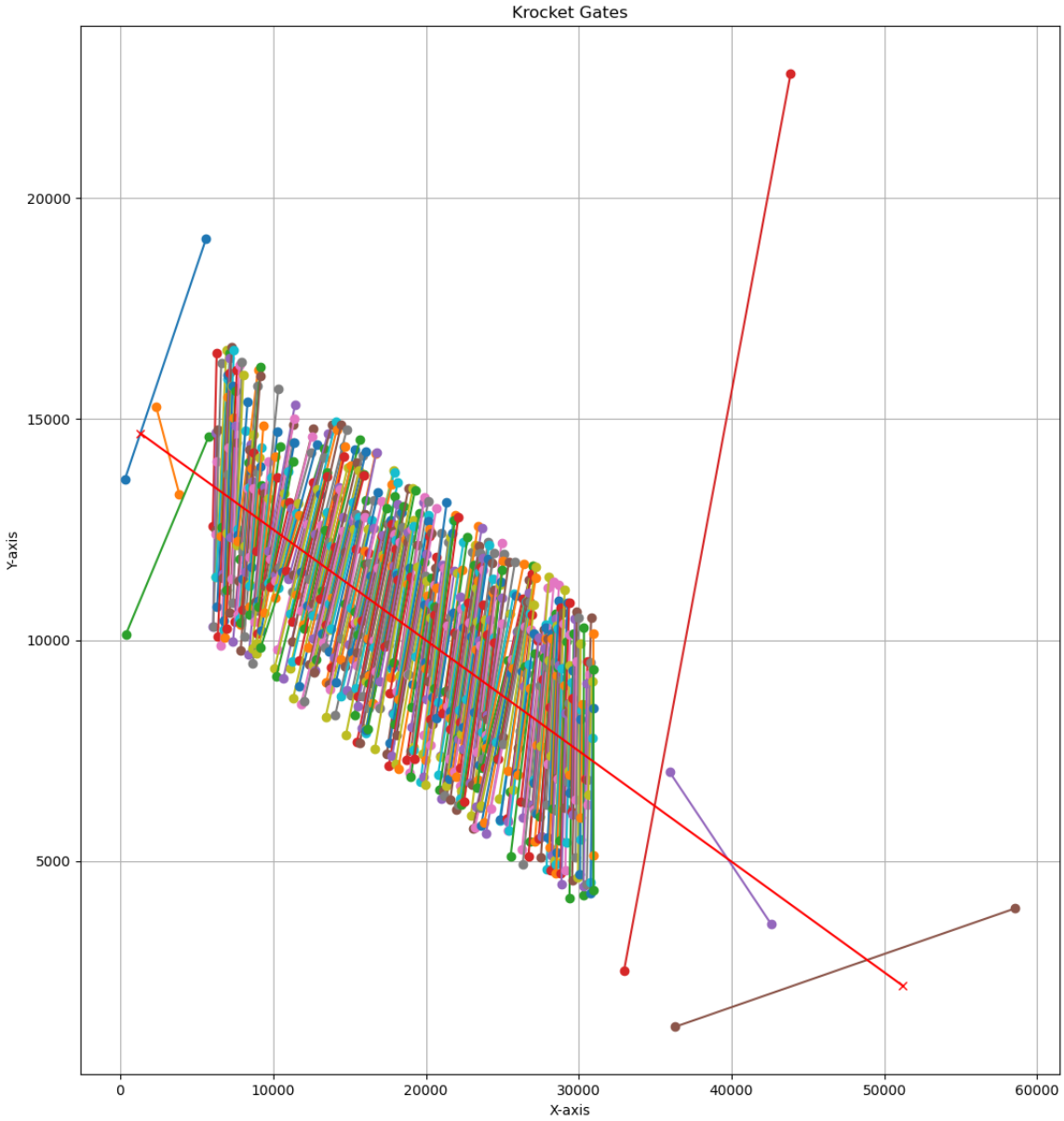


4. **krocket4.txt**
   Es ist möglich mit einem Schlag alle Tore zu durchqueren. Der Schlag kann als Halbgerade mit dem Startpunkt: (6.14299999999999|182.9909999999977), die durch den
   Punkt (92.69300000000004|226.22499999999985) verläuft, beschrieben werden.
   Normierter Richtungsvektor: (0.8945966401484378, 0.4468745365705309), bzw. Steigung der Halbgeraden: 0.4995262853841954

   Darstellung (python-skript)



5. **krocket5.txt**
   Es ist möglich mit einem Schlag alle Tore zu durchqueren.
   Der Schlag kann als Halbgerade mit dem Startpunkt: (1314.4010000000007|14678.04), die durch den Punkt (3160.4020000000005|14215.655999999999) verläuft, beschrieben werden.
   Normierter Richtungsvektor: (0.9700331331504777, -0.24297267457528587), bzw. Steigung der Halbgeraden: -0.25047873755214756




# 4 Quellcode

**Finden einer Gerade, die alle Tore durchquert**

```
/**
* Calculates a line that intersects all given gates
* The line intersects the starting point (placed on the first gate)
* Endpoint (direction of the line) for calculation of the line is on the second
* gate
*
* @param gates    - List of gates the line has to intersect
* @param stepSize - Step size (0 to 1) percentage of the first gate's
*                  length, specifies how finely the starting points
*                  are iterated along the first gate
```

```
* @return Line - line that intersects all gates, null if no intersection
*/
public Line findRayIntersectAllGates(List<Gate> gates, double stepSize, double
    ballRadius) {
    Gate firstGate = gates.get(0);
    Gate secondGate = gates.get(1);

    // Iterate over positions on first gate in increments defined by step size
    // For every startpoint on the first gate, the loop is executed
    for (double stepStart = 0; stepStart <= 1; stepStart += stepSize) {
        // Create start point for the line on the first gate
        Point startPoint = new Point(
                firstGate.getStart().getX()
                        + stepStart * (firstGate.getEnd().getX() -
    firstGate.getStart().getX()),
                firstGate.getStart().getY()
                        + stepStart * (firstGate.getEnd().getY() -
    firstGate.getStart().getY()));

        // Iterate over positions on second gate in increments defined by step size
        for (double stepDir = 0; stepDir <= 1; stepDir += stepSize) {
            Point directionPoint = new Point(
                    secondGate.getStart().getX()
                            + stepDir * (secondGate.getEnd().getX() -
    secondGate.getStart().getX()),
                    secondGate.getStart().getY()
                            + stepDir * (secondGate.getEnd().getY() -
    secondGate.getStart().getY()));
            // Create line, that intersects startpoint and directionpoint
            Line gerade = new Line(startPoint, directionPoint);

            boolean intersectsAll = true;
            // Checks if line intersects all gates
            for (Gate gate : gates) {
                // Intersection between Segment and line
                Point intersectionSegmentGerade = calculateLineIntersectSegment(gerade,
    gate, ballRadius);
                // Does not intersect
                if (intersectionSegmentGerade == null) {
                    intersectsAll = false;
                    break;
                }
            }
            if (intersectsAll) {
                return gerade;
            }
        }
    }
    return null;
}
```

**Schnittpunkt einer Geraden mit einer Strecke**

```
/**
* Calculates point of intersection of a line and a segment (krocket gate)
*
* @param line - line to describe possible path of krocket ball
* @param gate - Krocketgate (segment)
* @return Point - Point of intersection; null if no intersection exists
*/
public Point calculateLineIntersectSegment(Line line, Gate gate, double ballRadius) {
    double lineStartX = line.getStart().getX();
    double lineStartY = line.getStart().getY();
    double dx = line.getDeltaX();
    double dy = line.getDeltaY();
    double gateStartX = gate.getStart().getX();
    double gateStartY = gate.getStart().getY();
    double gateEndX = gate.getEnd().getX();
    double gateEndY = gate.getEnd().getY();

    // More info in the documentation of this task
```

```
    // To calculate the intersection the equation of the line and gate have to be
    // equated
    // A system of linear equation can be set up for that
    // Then solve for unknown (u & t)
    // Which can be interpreted as a 2x2 matrix

    // Calculate determinant of equation system
    double denominator = dx * (gateEndY - gateStartY) - dy * (gateEndX - gateStartX);

    // If determinant is (almost) zero, the line and gate (segment) are parallel or
    // colinear
    // Therefore they cant have an intersection
    if (Math.abs(denominator) < 1e-10) {
        return null;
    }

    // Solve the system of equations
    // (I used gauss algorithm)
    double t = ((gateStartX - lineStartX) * (gateEndY - gateStartY)
            - (gateStartY - lineStartY) * (gateEndX - gateStartX)) / denominator;
    double u = ((gateStartX - lineStartX) * dy - (gateStartY - lineStartY) * dx) /
    denominator;

    // Check if the intersection point is outside of the gate segment
    if (u < 0 || u > 1) {
        return null;
    }

    // Calculate the intersection point on the line
    Point intersectionPoint = line.getPointAt(t);

    // Check if the line intersects circle (collision boundary of start and endpoint
    // of gate)
    if (lineIntersectsCircle(line, gate.getStart(), ballRadius) ||
            lineIntersectsCircle(line, gate.getEnd(), ballRadius)) {
        return null; // Intersection invalid due to collision with gate endpoints
    }

    // Valid intersection point with enough space to gate endpoints
    return intersectionPoint;
}
```

**Schnittpunkte einer Gerade und einem Kreis**

```
/**
* Check if a line intersects a circle
*
* @param line         - line
* @param circleCenter - center of the circle as a Point
* @param radius       - radius of circle as double
* @return true if the line intersects the circle, else false
*/
private boolean lineIntersectsCircle(Line line, Point circleCenter, double radius) {
    double cx = circleCenter.getX();
    double cy = circleCenter.getY();
    double x1 = line.getStart().getX();
    double y1 = line.getStart().getY();
    double dx = line.getDeltaX();
    double dy = line.getDeltaY();

    // More info in the docs of this task
    // circle equation: (x - cx)^2 + (y - cy)^2=r^2
    // line equation: x = x1 + t * dx
    // &&
    // line equation: y = y1 + t * dy
    // Short: plug line equation into circle equation and solve for t
    double a = dx * dx + dy * dy;
    double b = 2 * (dx * (x1 - cx) + dy * (y1 - cy));
    double c = (x1 - cx) * (x1 - cx) + (y1 - cy) * (y1 - cy) - radius * radius;

    // Solve quadratic equation a*t^2 + b*t + c = 0 and just check discriminant
    // If discriminant < 0: no intersection
```

```
    // = 0: one intersection = line is a tangent
    // > 0: two intersections
    double discriminant = b * b - 4 * a * c;

    if (discriminant <= 0) {
        return false; // No real number
    } else {
        return true;
    }
}
```