Aufgabe 4: Krocket

Team-ID: **00078**

Team: Timon

Bearbeiter dieser Aufgabe:
Timon Retzlaff

Inhaltsverzeichnis

Lösungsidee

Das Programm iteriert mit einer festgelegten Schrittanzahl über die Strecke des ersten und letzten Tores. Dabei wird für jede mögliche Kombination von Start- und Endpunkt getestet, ob diese eine Lösung ist. Der Test funktioniert mittels Vektorrechnung im zweidimensionalen Raum.

Der Radius des Balls wird berücksichtigt, indem für drei Geraden getestet wird. Von denen repräsentiert eine den Mittelpunkt des Balles, die anderen beiden verlaufen parallel dazu im Abstand des Radius des Balles.

Wenn alle drei Geraden kollisionsfrei die Tore passieren, ist der Schuss gültig. Für die Berücksichtigung der richtigen Reihenfolge wird der Mittelpunkt des Balls betrachtet.

Umsetzung

Einlesen der Datei

Die Implementierung beginnt damit, dass die Eingabedaten aus einer Datei gelesen und in einer Liste gespeichert werden. Dann werden sie in einem Gate-Array gespeichert.

Gate-Klasse

Jedes Tor wird durch ein G*ate*-Objekt repräsentiert, in dem der Start- und Endpunkt gespeichert wird. Diese bietet verschiedene Funktionen, wie *getLength, getDist* oder *getDiffX*.

Iterieren über Möglichkeiten

In der *findShot*-Methode wird zuerst die Schrittgröße in x- und y-Richtung für das erste und letzte Tor berechnet. Dann iteriere ich in einer äußeren Schleife über das erste Tor und in einer inneren über das letzte Tor. Für jede Kombination wird ein *Shot* erstellt, der die Punkte speichert. Danach wird der *Shot* getestet.

TestShot-Methode – Möglichkeit überprüfen

Die Methode geht für den Schuss alle Tore durch und ruft die *getParam*-Methode auf. Dann überprüft sie, ob die Parameter immer größer werden und durch die *isLegit*-Methode, ob der Ball durch alle Tore passt.

getParam-Methode

Die *getParam*-Methode rechnet für die Gerade des Shots mit dem Stützvektor (x1, y1) und Richtungsvektor (x2-x1, y2-y1) den Parameter aus, bei dem das Gate geschnitten wird. Wenn die Geraden sich nicht treffen, wird -1 zurückgegeben.

isLegit - Methode

Die *isLegit*-Methode überprüft den Schuss und ruft die *checkRadius*-Funktion auf, die die zwei parallelen Geraden berechnet und diese überprüft. Dazu wird der orthogonale Vektor zum Schuss berechnet und auf den Schuss addiert.

Beispiele

Beispiel 1: Ein Tor

In einem Fall mit nur einem Tor prüft das Programm, ob das Tor groß genug für den Ball ist. Falls ja rechnet es den orthogonalen Vektor aus und ermittelt so den Schuss.

Eingabe:

- 1 1
- 5 5 5 10

Ausgabe:

x1=6.0;x2=5.0;angle:-0.0°

Beispiel 2: Mehrere Tore

Hier ein Beispiel mit mehreren Toren.

Eingabe:

- <siehe gegebene Beispiel aus krocket1.txt>

Ausgabe:

x1=11.2736;x2=238.943;angle:25.975357705716817°

Beispiel 3: Viele Tore

Eingabe:

- <siehe gegebene Beispiel aus krocket5.txt>

Ausgabe:

x1=1302.1857;x2=48968.296;angle:-14.00509684897943°

Beispiel 4

Eingabe:

- <siehe gegebene Beispiel aus krocket4.txt>

Ausgabe:

x1=6.144300000000001;x2=8946.9584;angle:26.535976854632914°

Beispiel 5

Eingabe:

- <siehe gegebene Beispiel aus krocket3.txt>

Ausgabe:

Null

Beispiel 6

Eingabe:

- <siehe gegebene Beispiel aus krocket2.txt>

Ausgabe:

null

Quellcode

```
private static Shot findShot(final java.util.List<String> inputLines) {

    […]

    final Gate firstGate = gates[0];
    final Gate lastGate = gates[gates.length - 1];
    final double stepSizeX1 = firstGate.getDiffX() / STEP_COUNT;
    final double stepSizeY1 = firstGate.getDiffY() / STEP_COUNT;
    final double stepSizeX2 = lastGate.getDiffX() / STEP_COUNT;
    final double stepSizeY2 = lastGate.getDiffY() / STEP_COUNT;

    for (int i = 0; i < STEP_COUNT; i++) {
        double x1 = firstGate.x1 + stepSizeX1 * i;
        double y1 = firstGate.y1 + stepSizeY1 * i;
        for (int j = 0; j < STEP_COUNT; j++) {
            double x2 = lastGate.x1 + stepSizeX2 * j;
            double y2 = lastGate.y1 + stepSizeY2 * j;
            Shot toTest = new Shot(x1, y1, x2, y2);
            if (testShot(toTest, gates, radius)) {
                return toTest;
            }
        }
    }
    return null;
}

private static boolean testShot(final Shot shot, final Gate[] gates, final
double radius) {
    double lastS = -0.5;
    for (Gate gate : gates) {
        double newS = getParam(shot, gate);
        if (newS < lastS || !isLegit(newS, shot, gate, radius, true)) {
            return false;
        } else {
            lastS = newS;
```

```
        }
    }
    return true;
}

private static boolean isLegit(final double s, final Shot shot, final Gate gate,
final double radius, boolean checkRadius) {
    final double x = shot.getDiffX() * s + shot.x1;
    final double y = shot.getDiffY() * s + shot.y1;

    if (x <= Math.max(gate.x1, gate.x2) && x >= Math.min(gate.x1, gate.x2)
            && y <= Math.max(gate.y1, gate.y2) && y >= Math.min(gate.y1,
gate.y2)) {
        return !checkRadius || checkRadius(shot, gate, radius);
    } else {
        return false;
    }
}
```