33. Bundeswettbewerb Informatik

Pong (Aufgabe 5)

Johannes Heinrich, Marian Dietz

Lösungsidee

Da es nicht möglich ist, über die API abzufragen, in welche Richtung der Ball fliegt, muss diese selbst errechnet werden.

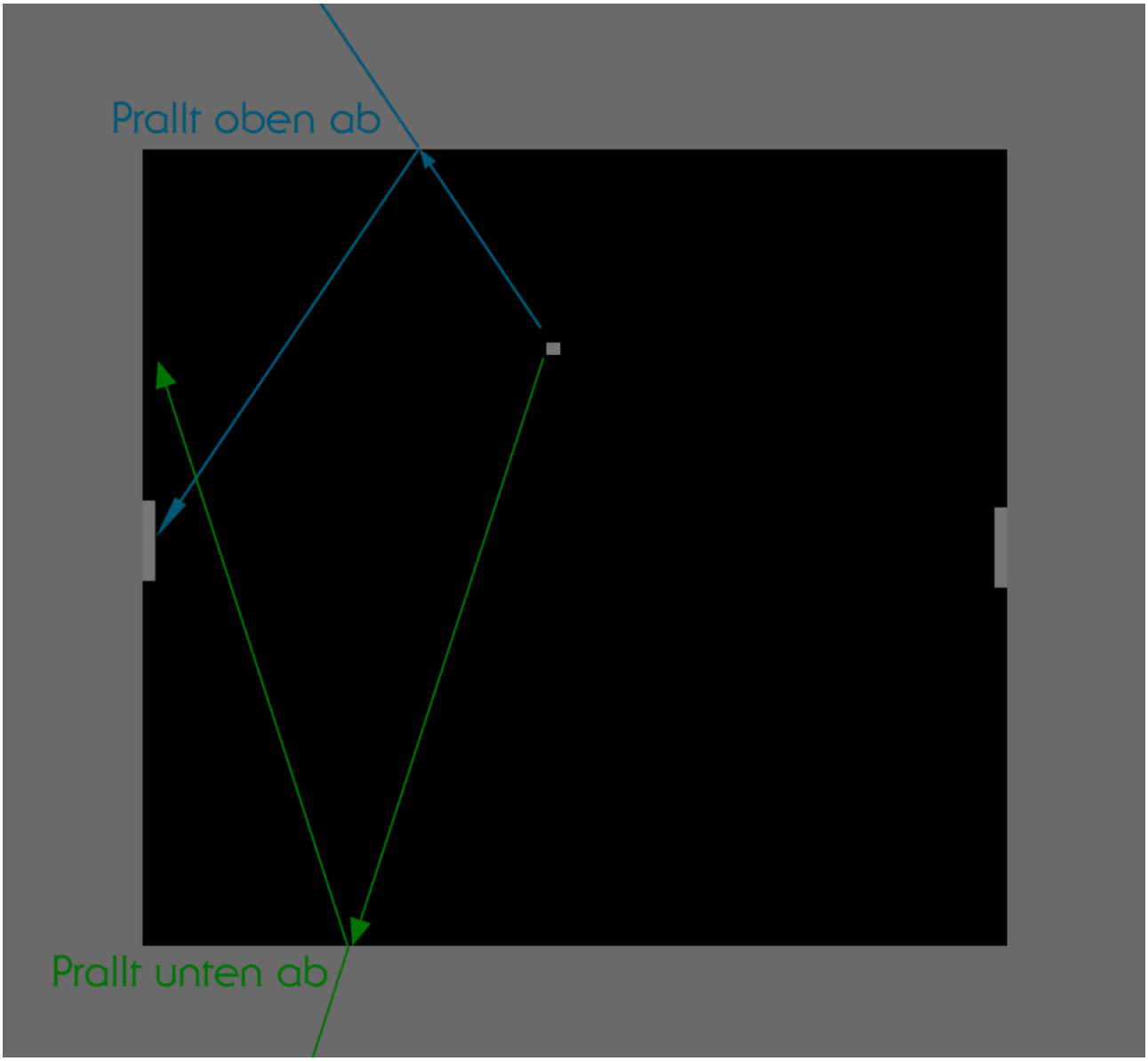
Die Position des eigenen Schlägers ist S. Sie ist die oberste Zelle des Schlägers. Die unterste Zelle des Schlägers ist $s + 5$, da der Schläger 6 Zellen lang ist.
Die aktuellen Koordinaten des Balles nennen wir x_i und y_i. Die allererste Position wird also durch x_1 und y_1 dargestellt. Noch ist es nicht möglich, zu bestimmen, wo der Ball aufprallen wird, da wir nur eine Position haben. Kommt der nächste Zug, dann kommen die weiteren Variablen x_2 und y_2 hinzu. Dadurch ist es möglich, den Aufprallort zu bestimmen.

Der Ball fliegt vom eigenen Schläger weg, wenn x_i größer als x_1 ist, da die X-Koordinate beim eigenen Schläger null ist und bis zum anderen Schläger (X-Koordinate 65) immer größer wird. Sollte dies der Fall sein, dann geht der eigene Schläger einfach zurück in die Mitte, um den später ankommenden Ball von jeder Seite aus schnell abfangen zu können. Noch kann nicht berechnet werden, wo der Ball später an der eigenen Seite ankommen wird, denn wenn der Gegner den Ball mit dem oberen oder unteren Drittel annimmt, kann der Ball seinen Winkel ändern. Da die Höhe des Spielfeldes 60 Zellen beträgt und alle Zellen der obersten Zeile die X-Koordinate null haben, muss $s = 27$ wenn der Schläger zurück in die Mitte geht, damit die *Mitte* des Schlägers auch in der Mitte des Spielfeldes steht.

Wenn der Ball aber zur eigenen Seite hinfliegt ($x_1 > x_i$), dann muss sich der Schläger schnell zur Aufprallstelle bewegen, um den Ball abzufangen. Da sich die Koordinaten des Balles aber seiner genauen Position nur annähern, kann das Ziel nie genau berechnet werden. Zunächst wird die bisherige Steigung des Balles berechnet: $m = \frac{y}{x} = \frac{y_1 - y_i}{x_1 - x_i}$. $x_1 - x_i$ kann nicht negativ sein, da $x_1 > x_i$ ist. $y_1 - y_i$ kann jedoch negativ sein, wenn der Ball nach unten fliegt. In diesem Fall ist $y_i > y_1$. Die Steigung m ist also positiv, wenn der Ball nach oben fliegt, negativ, wenn der Ball nach unten fliegt.
Dabei gibt es einen Sonderfall: wenn $x_1 - x_i = 0$, kann der oben genannte Bruch nicht ausgerechnet werden. Das bedeutet, dass sich der Ball sehr steil nach oben oder unten bewegt und sich deshalb die X-Koordinate noch nicht geändert hat. Dann wird einfach davon ausgegangen, dass der Ball den größte Winkel (70°) besitzt: $m = tan(70°)$. Der Tangens ist im rechtwinkligen Dreieck definiert als Gegenkathete durch Ankathete, also genau die Steigung. Sollte die Richtung des Balles nach unten zeigen, dann wird der Tangens von $-70°$ genommen, damit das Vorzeichen der Steigung wieder korrekt ist.

Nachdem dies getan ist, wird die Länge berechnet, welche der Ball auf der Y-Koordinate zurücklegen wird, bis er auf der eigenen Seite aufprallt: $h = x_i \cdot m$. Dabei wird einfach die Steigung mit dem auf der X-Koordinate zurückzulegendem Weg multipliziert, um die Strecke auf der Y-Koordinate herauszubekommen. h ist auch wieder wegen dem Vorzeichen der Steigung positiv, wenn der Ball nach oben fliegt und negativ, wenn der Ball nach unten fliegt.

Nun kann der Aufprall berechnet werden: $a = y_i - h$. Dabei wird von der aktuellen Y-Koordinate des Balles die noch auf der Y-Koordinate zurückzulegenden Strecke subtrahiert. Wenn der Ball nach oben fliegt ($h > 0$), ist der Aufprall weiter oben als die aktuelle Y-Koordinate. Wenn der Ball nach unten fliegt ($h < 0$), ist der Aufprall weiter unten als die aktuelle Y-Koordinate.
Auf dem Weg zum eigenen Schläger kann der Ball jedoch noch an der oberen bzw. unteren Kante des Spielfeldes abprallen:



Diese Grafik veranschaulicht das Abprallen des Balles am oberen oder unteren Spielfeldrand. Der Winkel des Balles verändert sich in einem solchen Fall *nicht*.
Wenn der Ball oben abprallt, ist a negativ. Deshalb ist der eigentliche Aufprall einfach der Betrag von a. a wird also auf $-a$ gesetzt. Um diese Vorgehensweise zu begründen: der Weg, den der

Ball „über“ dem Spielfeld zurücklegen würde, wird auf dem Spielfeld von der oberen Kante aus ausgeführt, wodurch die Position am oberen Spielfeldrand „gespiegelt“ wird.

Wenn der Ball unten abprallt (grüne Pfeile) ist $a > 60$. Um den korrekten Aufprall zu ermitteln, wird zunächst der Abstand zwischen der Spielfeldhöhe (60) und dem vorher berechneten Aufprall „unter“ dem Spielfeld berechnet. Dieser Abstand wird von der Spielfeldhöhe abgezogen. Dadurch wird der Aufprall auch wieder am unteren Spielfeldrand „gespiegelt“.
Der Aufprall wird also auf folgenden Wert gesetzt: $60 - (a - 60)$.
Da der Ball mehrmals vom Rand abprallen kann, bis er an der eigenen Seite auftrifft, müssen diese Vorgänge so oft ausgeführt werden, bis $0 \leq a \leq 60$. Dies ist dann er ungefähre Aufprallort.

Wenn der Aufprall des Balles über dem mittleren Drittel des Schlägers ist ($a < s + 2$), dann bewegt sich der Schläger weiter nach oben.
Wenn der Aufprall des Balles unter dem mittleren Drittel des Schlägers ist ($a > s + 3$), dann bewegt sich der Schläger weiter nach unten.
Ansonsten steht der Schläger bereits an der richtigen Stelle. Dann wird berechnet, um wie viele Zellen sich der Ball durchschnittlich auf der X-Koordinate weiterbewegt: $d = \frac{x_a}{z_a}$. Dafür wird die Bewegung auf der X-Koordinate seit dem letzten Abprallen durch die Anzahl der Bewegungen seit dem letzten Abprall dividiert. Wenn $\frac{x_i}{d} \leq 1$, dann wird der Ball im nächsten Zug am Schläger ankommen. Durch $\frac{x_i}{d}$ wird berechnet, wie viele Züge der Ball noch benötigt, um aufzuprallen.

Nur wenn der Ball im nächsten Zug aufprallen wird und der Schläger bereits an der richtigen Stelle steht, wird fortgefahren. Ansonsten bewegt sich der Schläger einfach gar nicht.
Zunächst wird berechnet, ob der Gegner den Ball bekommen kann, wenn der eigene Schläger ihn mit dem mittleren Drittel abprallen lässt, sich der Winkel also nicht verändert. Dafür wird wieder der Algorithmus oben verwendet, durch welchen sich die Aufprallstelle des Balles berechnen lässt. Der einzige Unterschied zu vorher ist, dass der „Startpunkt“ des Balles die Y-Koordinate am zuvor errechneten Aufprall am eigenen Schläger ist und der zurückzulegende Weg auf der X-Koordinate beträgt 65, also die gesamte Breite des Spielfeldes. Die Steigung m von vorher wird einfach wiederverwendet, wird aber negiert, da der Ball schließlich in die entgegengesetzte Richtung abprallt.
Wenn die Aufprallstelle a' also so weit vom gegnerischen Schläger entfernt ist, dass er den Ball nicht mehr bekommen kann, bleibt der eigene Schläger stehen, um ihn mit dem mittleren Drittel abzufangem. Dafür wird die Anzahl der Züge berechnet, die der Ball benötigt, um einmal über das ganze Spielfeld zu fliegen: $z = \frac{65}{d}$. d ist dabei wieder die zuvor berechnete durchschnittliche Bewegung auf der X-Koordinate. Der gegnerische Schläger s' bekommt also den Ball, wenn:

- $a' < s'$ und $s' - a' < z$, die Aufprallstelle also über dem gegnerischen Schläger ist und die Differenz zwischen Aufprallstelle und Schläger kleiner ist als die Anzahl der Züge, welche der Ball benötigt *oder*
- $a' > s' + 5$ und $a' - s' + 5 < z$, die Aufprallstelle also unter dem untersten Punkt des gegnerischen Schlägers ist und die Differenz zwischen Aufprallstelle und unterster Zelle des Schlägers kleiner ist als die Anzahl der Züge, welche der Ball benötigen wird.

Dabei kann die Anzahl der Züge nicht genau berechnet werden, da der Ball nach dem Abprall am eigenen Schläger und gegebenenfalls nach mehreren Abprällen am oberen bzw. unteren Spielfeldrand schneller wird.

Sollte der Gegner den Ball bekommen können, dann bewegt sich der eigene Schläger folgendermaßen:

- Wenn $a = s + 2$, dann bewegt sich der Schläger nach unten, um den Ball mit dem oberen Drittel abzufangen.
- Wenn $a = s + 3$, dann bewegt sich der Schläger nach oben, um den Ball mit dem unteren Drittel abzufangen.

Vor jedem Zug, also bevor die Aufprallstelle etc. berechnet wird, wird überprüft, ob der Ball inzwischen irgendwo abgeprallt ist.

- Der Ball ist an der Y-Koordinate, also am oberen bzw. unteren Rand abgeprallt, wenn die Vorzeichen (Signum) von der vorigen Y-Richtung und der jetzigen Y-Richtung unterschiedlich sind (eines davon positiv, das andere negativ): $|sgn(y_1 - y_{i-1}) - sgn(y_{i-1} - y_i)| = 2$
- Der Ball ist an der X-Koordinate, also an einem Schläger abgeprallt, wenn die Vorzeichen (Signum) von der vorigen X-Richtung und der jetzigen X-Richtung unterschiedlich sind (eines davon positiv, das andere negativ): $|sgn(x_1 - x_{i-1}) - sgn(x_{i-1} - x_i)| = 2$

Dabei muss y_i-1 bzw. x_i-1 einen Wert haben.
Wenn ein Spieler einen Punkt bekommt und der Ball neu „eingeworfen“ wird, dann wird auch erkannt, dass der Ball „abgeprallt“ ist, da sich dann in jedem Fall der Ball an der X-Koordinate an einer anderen Stelle befindet und er sich deshalb, wenn auch nur für einen Zug, in eine andere Richtung bewegt hat.
Wenn der Ball an der X-Koordinate abgeprallt ist, dann werden alle vorherigen Koordinaten des Balles zurückgesetzt, da er seinen Winkel geändert haben kann.
Wenn sich jedoch *nur* die Y-Richtung änderte, dann kann sich der Winkel nicht verändert haben. In einem solchen Fall wird einfach so getan, als wären die vorherigen Positionen außerhalb des Spielfeldes:

- Wenn der Ball oben abgeprallt ist ($y_{i-1} < y_i$, Richtung zeigt jetzt nach unten) und y_1 angepasst werden soll, wird y_1 einfach negiert, um am oberen Spielfeldrand „gespiegelt“ zu werden.
- Wenn der Ball unten abgeprallt ist ($y_{i-1} > y_i$, Richtung zeigt jetzt nach oben), dann wird y_1 auf den folgenden Wert gesetzt: $60 + (60 - y_1)$. Dafür wird der Abstand zwischen der Y-Koordinate und dem unteren Rand gemessen und zur Spielfeldhöhe addiert. Auch das „spiegelt“ die Koordinate am unteren Spielfeldrand.

Beim nächsten Zug wird wieder y_2 auf die dann aktuelle Y-Koordinate des Balles gesetzt.
Wenn der Ball in X-Richtung abgeprallt ist, darf sich der Schläger noch nicht bewegen, da sich der Winkel geändert haben könnte.

Umsetzung

Das Programm wurde in der Programmiersprache Java geschrieben. Dabei ist der gesamte Code in der Klasse `AI` enthalten. Jeder Zug wird in der Methode `zug()` ausgeführt.
Die Breite und die Höhe sind als Konstanten (`final static`) namens `WIDTH` und `HEIGHT` in `AI` enthalten.
Dann gibt es das Attribut `zuegeSeitAbprall`. Dieses ist ein `int` und besagt, wie oft sich der Ball seit dem letzten Abprall (ob an der X- oder Y-Koordinate ist egal) bewegt hat.
Die Variablen `firstX` und `firstY` geben die erste Position des Balles an, sie sind also vergleichbar mit x_1 und y_1.

`lastX` und `lastY` stellen die vorherige Position des Balles dar.
Zu guter Letzt gibt es noch das Attribut `abgepralltX`. Es zeigt an, wo die X-Koordinate des letzten Abpralls war.
Alle Variablen, welche Koordinaten darstellen, haben am Anfang den Wert -1.

In der Methode `zug()` wird zunächst überprüft, ob der Ball irgendwo abgeprallt ist bzw. ob das Spiel gerade gestartet wurde (dies ist dann der Fall, wenn `abgepralltX` -1 ist, da dies die einzige Variable ist, die nur während des ersten Zuges sicher -1 ist). Wenn der Ball abgeprallt ist, wird die Methode `abgeprallt()` aufgerufen, welche diese Information verarbeitet. Dabei bekommen `firstX` und `firstY` möglicherweise neue Werte. `abgepralltX` wird auf die aktuelle X-Koordinate gesetzt, `zuegeSeitAbprall` wird auf 0 und die Koordinaten `lastX` und `lastY` werden auf -1 zurückgesetzt. Diese Methode gibt `true` zurück, wenn der Schläger sich bewegen darf, `false`, wenn nicht.

Wenn sich der Schläger bewegen soll, wird in `zug()` die Methode `move()` aufgerufen. Wenn der Ball vom eigenen Schläger wegfliegt, geht der Schläger zurück in die Mitte mit den Methoden `Spiel.Zug.nachOben()` bzw. `Spiel.zug.nachUnten()`.
Ansonsten wird die Steigung berechnet. Dann wird der Aufprall mithilfe der Methode `aufprall()` errechnet und zurückgegeben. Wenn sich der Schläger noch an der falschen Position befindet, bewegt er sich dorthin.
Wenn er schon dort ist, wird berechnet, ob der Ball im nächsten Zug aufprallen wird. Ist dies der Fall, wird mit der Methode `gegnerBekommtBall()` überprüft, ob der Gegner den Ball bekommen wird. Diese ruft wieder die Methode `aufprall()` mit der negierten Steigung auf. Wenn der Gegner den Ball bekommt kann, geht der Schläger nach oben bzw. unten, sodass der Ball am oberen bzw. unteren Drittel abprallt.
Ist all das getan, werden `lastX` und `lastY` auf die akutelle Position gesetzt, um im nächsten Zug dann als vorherige Position zu dienen. Außerdem wird `zuegeSeitAbprall` hochgezählt.

Beispiele

Beispiele hier einzufügen ist wegen der Ansicht leider nicht möglich. Um den Schläger in Aktion zu sehen, können die Spiele unserer KI namens „qPong“ im Turnierstystem des Bundeswettbewerbes Informatik angesehen werden:
http://turnier.bundeswettbewerb-informatik.de/ai/profile/2211/

Quellcode

AI

```
public class AI {

    /**
     * Breite / Höhe des Spielfeldes
     */
    private final static int WIDTH = 65, HEIGHT = 60;

    /**
     * lastX und lastY: Koordinaten für die vorherige Position des Balles
     * firstX und firstY: Koordinaten für die erste Position des Balles seit dem
     * letzten Ändern des Winkels. Wenn diese -1 sind, bedeutet das, dass sie
     * noch nie verändert wurden und deswegen das Spiel noch nicht gestartet
     * wurde bzw. gerade der erste Zug gemacht wird.
     * abgepralltX: X-Koordinate des Balles als er das letzte Mal abgeprallt ist
     * Alle Werte sind standardmäßig auf -1 gesetzt.
     */
    private int lastX = -1, lastY = -1, firstX = -1, firstY = -1, abgepralltX = -1;

    /**
     * Anzahl der Bewegungen des Balles seit dem letzten Abprall.
     */
    private int zuegeSeitAbprall;

    // ...

}
```

AI.zug()

```
public void zug(int id, Spiel.Zustand zustand, Spiel.Zug zug) {

    Spiel.Zustand.Ball ball = zustand.listeBall().get(0);
    int x = ball.xKoordinate();
    int y = ball.yKoordinate();

    // Prüfen, ob der Ball irgendwo Schläger abgeprallt ist:
    boolean yAbgeprallt = lastX != -1 &&
            Math.abs(Math.signum(firstY - lastY) - Math.signum(lastY - y)) == 2;
    // Wenn Y-Richtung vorher (firstY - lastY) anders als jetzt (lastY - y) ist.
    boolean xAbgeprallt = lastX != -1 &&
            Math.abs(Math.signum(firstX - lastX) - Math.signum(lastX - x)) == 2;
    // Wenn X-Richtung vorher (firstX - lastX) anders als jetzt (lastX - x) ist.

    // Dabei muss die Differenz der Vorzeichen 2 sein, um sicherzustellen,
    // dass der Ball in die entgegengesetzte Richtung fliegt.

    boolean move = true; // Ob der Schläger sich bewegen soll.

    // abgeprallt() aufrufen, wenn der allererste Zug ausgeführt wird
    // (abgepralltX == -1) oder wenn der Ball irgendwo abgeprallt ist.
    if (abgepralltX == -1 || yAbgeprallt || xAbgeprallt)
        move = abgeprallt(xAbgeprallt, x, y);

    // Hier bewegt sich der Schläger:
    if (move)
        move(x, y, zug, zustand, id);
}
```

AI.abgepralltl()

```
/**
 * Verarbeitet die Informationen, wenn der Ball abgeprallt ist.
 *
 * @param xAbgeprallt ob der Ball an der X-Koordinate abgeprallt ist
 * @param x           aktuelle X-Koordinate des Balles
 * @param y           aktuelle Y-Koordinate des Balles
 * @return {@code true} wenn sich der Schläger bewegen soll,
 * {@code false} wenn nicht
 */
private boolean abgeprallt(boolean xAbgeprallt, int x, int y) {

    // Normalerweise kann sich der Schläger bewegen.
    boolean move = true;

    // Wenn der Ball NUR am oberen oder unteren Rand (dafür müssen die anderen
    // Bedingungen zur Zurücksetzung alle falsch sein) abgeprallt ist, sonst
    // könnte es z. B. mit xAbgeprallt zusammen wahr sein.
    // In diesem Fall hat der Ball seinen Winkel NICHT verändert.
    if (abgepralltX != -1 && !xAbgeprallt) {
        // Da der erste Punkt nach dem Abprall die Berechnung verfälschen würde,
        // wird der erste Punkt an der oberen bzw. unteren Seite "gespiegelt":

        // Wenn der Ball oben abgeprallt ist (y muss dafür größer sein als
        // vorher), Punkt einfach negieren:
        if (lastY < y)
            firstY = -firstY;
        // Wenn der Ball unten abgeprallt ist (y kleiner als vorher), muss der
        // Punkt unter dem Spielfield sein. Dazu wird zur Spielfeldhöhe der
        // Abstand zwischen der Y-Koordinate und der Spielfeldhöhe addiert:
        else
            firstY = HEIGHT + (HEIGHT - firstY);
    }

    // Ansonsten ist dies der erste Zug (abgepralltX == -1) oder der Ball ist an
    // einem Schläger abgeprallt. In diesem Fall ist der Winkel noch nicht
    // bekannt bzw. kann sich geändert haben. Deswegen werden die
    // Startkoordinaten firstX und firstY auf die aktuelle Position gesetzt.
    else {
        // Neue Messung muss vorgommen werden, der erste Punkt wird auf aktuelle
        // Koordinate aktualisiert.
        firstX = x;
        firstY = y;

        // Bewegen kann sich der Schläger noch nicht, da der Winkel
        // möglicherweise verändert wurde.
        move = false;
    }

    // Folgendes tritt IMMER ein, wenn der Ball irgendwo abgeprallt ist bzw. das
    // Spiel gestartet wurde:

    // Die X-Koordinate des letzten Abpralls auf die aktuelle X-Koordinate
    // aktualisieren und zuegeSeitAbprall zurücksetzen.
    abgepralltX = x;
    zuegeSeitAbprall = 0;

    // Die letzten Koordinaten zurücksetzen.
    lastX = -1;
    lastY = -1;

    return move;
}
```

AI.move()

```
/**
 * Bewegt den Schläger.
 *
 * @param x       X-Koordinate des Balles
 * @param y       Y-Koordinate des Balles
 * @param zug     Spielzug
 * @param zustand Spielzustand
 * @param id      Identifikation des eigenen Schlägers
 */
private void move(int x, int y, Spiel.Zug zug, Spiel.Zustand zustand, int id) {

    int schlaeger = getOwnSchlaeger(zustand, id).yKoordinate();

    // Wenn der Ball sich zu der anderen Seite hinbewegt, geht der Schläger
    // zurück in die Mitte (Höhe / 2 - 3, da die Variable schlaeger der
    // oberste Punkt des Schlägers ist, er soll sich aber mit der Mitte des
    // Schlägers in die Mitte stellen):
    if (firstX < x) {
        if (schlaeger < HEIGHT / 2 - 3) zug.nachUnten();
        else if (schlaeger > HEIGHT / 2 - 3) zug.nachOben();
    }

    // Der Ball bewegt sich auf den eigenen Schläger zu; Aufprall berechnen und
    // dorthin bewegen:
    else {
        // Steigung des Balles berechnen (y / x). Wenn er sich auf der
        // X-Koordinate noch nicht bewegt hat, wird der Tangenz von (-)70 Grad
        // berechnet, da der größtmögliche Winkel des Balles (-)70 Grad ist.
        // Positiv, wenn der Ball nach oben fliegt, negativ, wenn die Richtung
        // nach unten zeigt (wegen firstY - y).
        double m = (firstX - x) == 0 ?
                (Math.tan(Math.toRadians(firstY - y > 0 ? 70 : -70))) :
                (firstY - y) / (double) (firstX - x);

        // Aufprall des Balles berechnen:
        int aufprall = aufprall(m, x, y);

        // Ball kommt über dem mittleren Drittel auf; nach oben bewegen:
        if (aufprall < schlaeger + 2)
            zug.nachOben();
        // Ball kommt unter dem mittlere Drittel auf; nach unten bewegen:
        else if (aufprall > schlaeger + 3)
            zug.nachUnten();
        // Schläger steht an der richtigen Stelle, weitere Aktion berechnen:
        else {
            // Durchschnittliche Bewegung des Balles auf X pro Zug berechnen:
            double durchschnittX = (abgepralltX - x) / (double) zuegeSeitAbprall;

            // Wenn der Ball im nächsten Zug aufprallen sollte:
            if (x / durchschnittX <= 1) {
                boolean gegnerFaengtAb = gegnerBekommtBall(-m, aufprall,
                        getOtherSchlaeger(zustand, id).yKoordinate(),
                        durchschnittX);

                // Nach unten gehen, wenn der Schläger den Ball mit dem
                // mittleren Drittel oben bekommen und wenn der Gegner den Ball
                // bei normaler Position nicht bekommen würde.
                if (gegnerFaengtAb && aufprall == schlaeger + 2)
                    zug.nachUnten();
                // Nach oben gehen, wenn der Schläger den Ball mit dem
                // mittleren
                // Drittel unten bekommen und wenn der Gegner den Ball bei
                // normaler Position nicht bekommen würde.
                else if (gegnerFaengtAb && aufprall == schlaeger + 3)
                    zug.nachOben();
            }
        }
    }

    // Die letzten Koordinaten aktualisieren und Anzahl der Züge erhöhen.
    lastX = x;
    lastY = y;
    zuegeSeitAbprall++;
}
```

AI.aufprall()

```java
/**
 * Berechnet den Aufprall des Balles.
 *
 * @param m Steigung, nach oben positiv, nach unten negativ
 * @param x die zurückzulegende Strecke des Balles auf der X-Koordinate
 * @param y die aktuelle Y-Koordinate des Balles
 * @return die Y-Koordinate, wo der Ball aufkommen wird
 */
private int aufprall(double m, int x, int y) {

    // Länge der Strecke, welche der Ball noch nach OBEN oder UNTEN
    // zurückzulegen hat, um am Schläger anzukommen. Dabei wird die Strecke
    // auf X vernachlässigt.
    // Positiv, wenn der Ball nach oben fliegt, negativ, wenn die Richtung nach
    // unten zeigt (wegen m).
    int h = (int) (x * m);

    // Aufprall des Balles. Strecke nach oben oder nach unten von der aktuellen
    // Position subtrahiert. Wenn h negativ ist (Ball fliegt nach unten), ist der
    // Aufprall auch weiter unten.
    int aufprall = y - h;

    // Wenn der Ball laut der bisherigen Berechnung das Spielfeld nach oben oder
    // unten hin verlässt.
    // while statt if, da der Ball mehrmals abprallen kann.
    while (aufprall < 0 || aufprall > HEIGHT) {
        // Ball prallt oben ab; Aufpall einfach negieren um dies auszugleichen:
        if (aufprall < 0)
            aufprall = -aufprall;
            // Ball prallt unten ab; Abstand des Balles zum Rand von der
            // Spielfeldhöhe abziehen, um korrekten Aufprall zu bekommen.
        else if (aufprall > HEIGHT)
            aufprall = HEIGHT - (aufprall - HEIGHT);
    }

    return aufprall;
}
```

AI.gegnerBekommtBall()

```java
/**
 * @param m             Steigung, nach oben positiv, nach unten negativ
 * @param start         Y-Koordinate des Punktes, bei dem der Ball starten wird
 * @param gegner        aktuelle Y-Koordinate des Gegners
 * @param durchschnittX durchschnittliche Bewegung des Balles auf der
 *                      X-Koordinate
 */
private boolean gegnerBekommtBall(double m, int start, int gegner,
                                  double durchschnittX) {
    int aufprall = aufprall(m, WIDTH, start);
    double zuege = WIDTH / durchschnittX;
    return (aufprall < gegner && gegner - aufprall < zuege) ||
            (aufprall > gegner + 5 && aufprall - gegner + 5 < zuege);
}
```

```
AI.getOwnSchlaeger()
AI.getOtherSchlaeger()
```

```
/**
 * @return den eigenen Schläger
 */
private Spiel.Zustand.Schlaeger getOwnSchlaeger(Spiel.Zustand zustand, int id) {
    if (zustand.listeSchlaeger().get(0).identifikation() == id)
        return zustand.listeSchlaeger().get(0);
    return zustand.listeSchlaeger().get(1);
}

/**
 * @return den gegnerischen Schläger
 */
private Spiel.Zustand.Schlaeger getOtherSchlaeger(Spiel.Zustand zustand, int
        id) {
    if (zustand.listeSchlaeger().get(0).identifikation() == id)
        return zustand.listeSchlaeger().get(1);
    return zustand.listeSchlaeger().get(0);
}
```