
Programmieren in Java<http://proglang.informatik.uni-freiburg.de/teaching/java/2017/>

simple-pool*Pool-Billard mit einer Kugel*

Woche 11 Aufgabe 3/3

Herausgabe: 2017-07-12

Abgabe: 2017-07-31

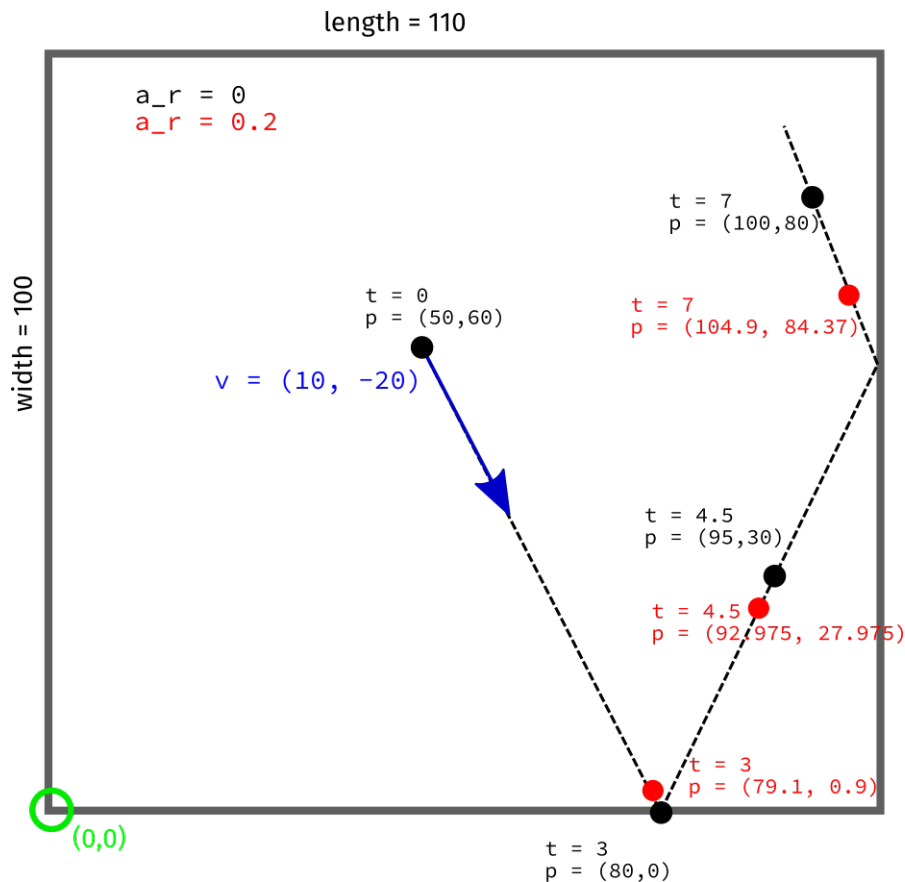
Achtung: beachten Sie unbedingt die allgemeinen Hinweise zur Abgabe auf der Homepage.

Project `simple-pool`
Package `simplepool`
Klassen

In dieser Aufgaben implementieren Sie eine vereinfachte Welt für zweidimensionales Pool-Billard. Die Pool-Welt enthält einen rechteckigen Tisch mit Breite w und Länge l . Ferner enthält die Welt eine Kugel, die sich auf dem Tisch befindet. Die Kugel wird in ihrer initialen Position $p = (p_x, p_y)$ angestoßen. Nach dem Stoß (den wir hier nicht modellieren) hat die Kugel einen initialen Geschwindigkeitsvektor $v = (v_x, v_y)$. Durch die Rollreibung wird die Kugel mit der Beschleunigung a_r gebremst. Prallt die Kugel an eine Wand, wird sie von ihr ohne Verlust reflektiert; d.h. Aufprallwinkel und Abstoßwinkel sind gleich und der Betrag des Geschwindigkeitsvektors ändert sich nicht.

Kommt Sie auf dem Tisch *zum Stillstand*, ist dieses spannende Spiel vorbei. Die Kugel *steht still*, wenn der Betrag ihrer Geschwindigkeit den Schwellwert ε unterschreitet.

Die folgende Zeichnung illustriert die Bewegung einer Kugel mit initialer Position $(50, 60)$ und initialer Geschwindigkeit $(10, -20)$. Die Kugelpositionen sind zu verschiedenen Zeitpunkten t eingezeichnet. Zeiten werden in Sekunden gemessen, Strecken in Millimetern. Die schwarze Kugel stellt die Bewegung bei einer Bremsbeschleunigung von $a_R = 0$ dar, die rote bei einer Bremsbeschleunigung von $a_R = 0.2\text{mm/s}^2$.



Um die Poolwelt zu modellieren finden Sie im Skelett zu dieser Aufgabe einige Hilfsmittel:

- Die Klasse **V2** zur Darstellung von 2D-Vektoren. Sie implementiert typische Operationen auf Vektoren.
- Die Klasse **Geometry** enthält Fabrikmethoden für Vektoren .
- Die Klasse **Physics** enthält Funktionen zur Berechnung der gleichförmig-gebremsten Bewegungen, die hier benötigt werden.

Die Details entnehmen Sie bitte den Javadoc-Kommentaren in den Klassen.

Ihre Aufgabe: implementieren Sie das folgende Interface **IPoolWorld** mit einer Klasse **PoolWorld**.

```

1 package simplepool;
2
3 public interface IPoolWorld {
4
5     /**
6      * Return a world representing this world "seconds" seconds advanced into
7      * the future.
8      */
9     public IPoolWorld step(double seconds);
10
11     /**

```

```

12     * Return true iff the ball is still isMoving
13     */
14     public boolean isMoving();
15
16     public V2 getCurrentBallPosition();
17
18     public double getLength();
19     public double getWidth();
20
21
22 }

```

Beachten Sie:

- die Methode `step` gibt eine *neue* `IPoolWorld` zurück. Die alte soll unverändert bleiben.
- die Kugel sollte sich immer innerhalb der Grenzen des Tisches befinden. Testen Sie dies im Konstruktor zu `PoolWorld` und brechen sie gegebenenfalls mit einer `IllegalArgumentException` ab. Die Kugel darf auch genau *auf* einer Tischkante liegen. Für die x -Koordinate der Kugel gilt z.B.: $x_{\text{Kugel}} \geq 0$ und $x_{\text{Kugel}} \leq \text{length}$.
- Die Kugel muss nicht unbedingt eine Wand berühren, bevor sie stehenbleibt.
- Wenn die Kugel einmal steht (nach der obigen Definition), dann soll sie sich ihre Position in Zukunft auch nicht mehr verändern.

Implementieren Sie weiterhin die folgende Factorymethode in der Klasse `simplepool.PoolWorlds`:

```

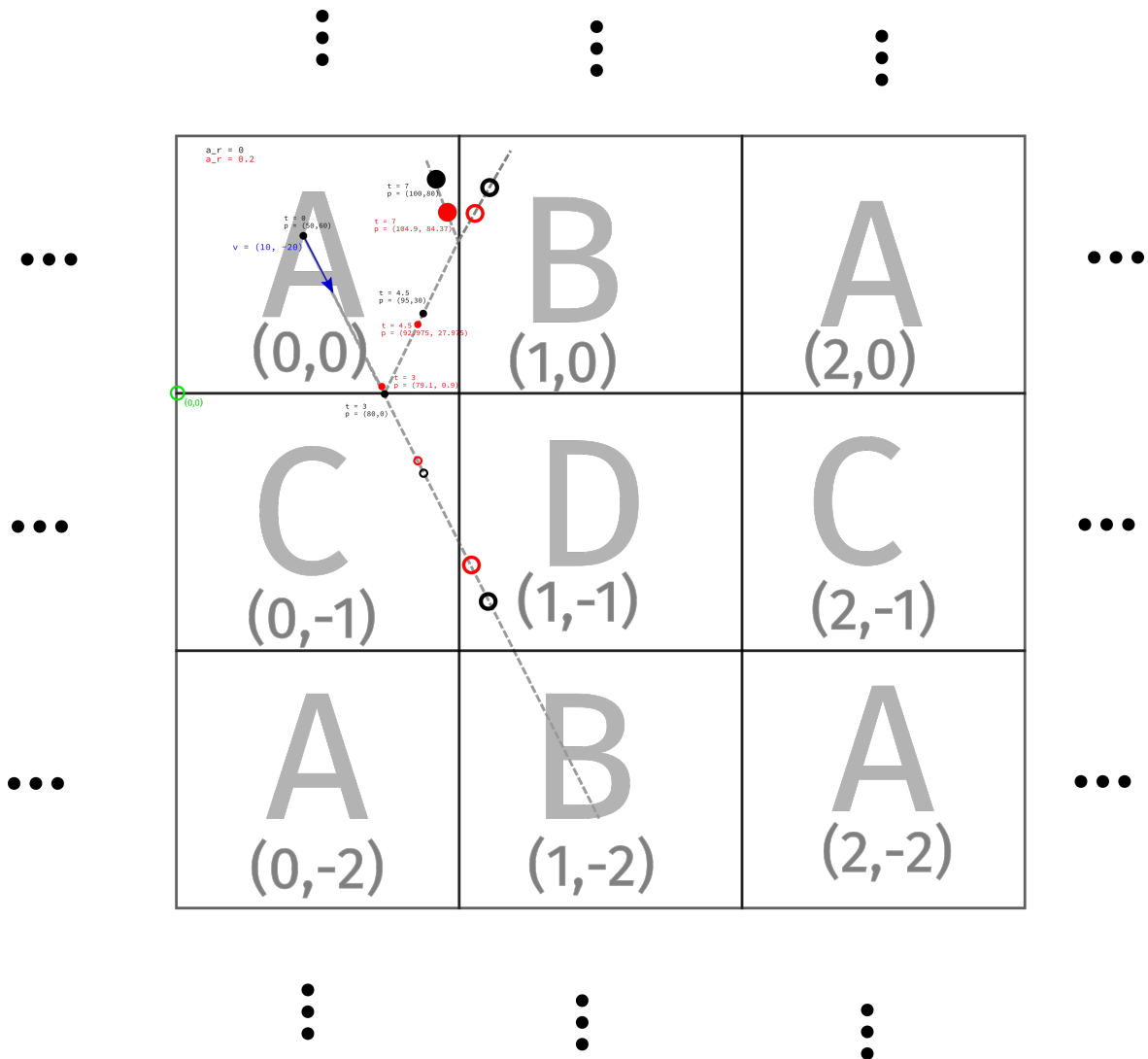
1  public class PoolWorlds {
2      /**
3       *
4       * @param initialBallPosition
5       * @param initialBallVelocity
6       * @param width
7       * @param length
8       * @param brakeAcceleration
9       * @return
10     */
11     public static IPoolWorld makePoolWorld(
12         V2 initialBallPosition,
13         V2 initialBallVelocity,
14         double length,
15         double width,
16         double brakeAcceleration,
17         double epsilon) {
18         /* Ihr Code hier */
19     }
20
21 }

```

Lösungsansatz

Zunächst sollten sie mit den Funktionen aus **Physics** die *lineare Bewegung* der Kugel berechnen, die sie bis zum gefragten Zeitpunkt ausführen würde, ohne die Reflexionen an den Tischkanten zu berücksichtigen. Dadurch erhalten Sie eine Punkt Z und einen Geschwindigkeitsvektor w .

Zur Berechnung der tatsächlichen Positions- und Geschwindigkeitsvektoren hilft dann folgender Trick: Stellen Sie sich eine unendliche Wiederholung der Tische in x und y Richtung vor, wie im folgenden Bild illustriert:



Die Tische sind mit den Labels A, B, C, D versehen, die verschiedene Orientierungen des Tisches darstellen:

- A ist der Tisch in normaler Orientierung
- B ist an der x -Achse gespiegelt (verglichen mit der normalen Orientierung)

- C ist an der y -Achse gespiegelt
- D ist an x -Achse und y -Achse gespiegelt

Jetzt müssen Sie bestimmen auf welchem Tisch sich Z befindet; also auf welchem Tisch sich die Kugel nach der linearen Bewegung befinden würde. Wie in der Zeichnung zu sehen, befindet sich z.B. die schwarze Kugel nach $t = 7s$ auf Tisch $(1, -1)$ mit Label D .

Mit Hilfe der Tischindices und des Tischlabels können Sie nun den Punkt Z und die Geschwindigkeit w in die tatsächliche Position und Geschwindigkeit auf Tisch $(0,0)$ umwandeln.

Beispieltests:

```

1 package simplepool;
2
3 import org.junit.Test;
4
5 import java.util.Optional;
6
7 import static org.junit.Assert.*;
8
9 public class ExampleTests {
10
11     @Test
12     public void testWithoutBrake() {
13         IPoolWorld w = PoolWorlds.makePoolWorld(Geometry.v2(50, 60),
14                                                     Geometry.v2(10, -20),
15                                                     110,
16                                                     100,
17                                                     0,
18                                                     0.01);
19         assertV2Equals(Geometry.v2(50, 60), w.getCurrentBallPosition(), 0.001);
20         assertV2Equals(Geometry.v2(80, 0), w.step(3).getCurrentBallPosition(), 0.001);
21
22         w = w.step(4.5);
23         assertV2Equals(Geometry.v2(95, 30), w.getCurrentBallPosition(), 0.001);
24
25         assertV2Equals(Geometry.v2(100, 80), w.step(2.5).getCurrentBallPosition(), 0.001);
26     }
27
28     @Test
29     public void testWithBrake() {
30         IPoolWorld w = PoolWorlds.makePoolWorld(Geometry.v2(50, 60),
31                                                     Geometry.v2(10, -20),
32                                                     110,
33                                                     100,
34                                                     0.2,
35                                                     0.01);

```

```

36     assertV2Equals(Geometry.v2(50, 60), w.getCurrentBallPosition(), 0.001);
37     assertV2Equals(Geometry.v2(79.1, 0.9), w.step(3).getCurrentBallPosition(), 0.001);
38
39     w = w.step(4.5);
40     assertV2Equals(Geometry.v2(92.975, 27.975), w.getCurrentBallPosition(), 0.001);
41
42     assertV2Equals(Geometry.v2(104.9, 75.1), w.step(2.5).getCurrentBallPosition(), 0.001);
43 }
44
45
46 @Test
47 public void testDone() {
48     IPoolWorld w = PoolWorlds.makePoolWorld(Geometry.v2(6, 5),
49                                             Geometry.v2(0, 0.009),
50                                             150,
51                                             100,
52                                             0,
53                                             0.01);
54     assertFalse(w.isMoving());
55 }
56
57
58 public static void assertV2Equals(V2 v1, V2 v2, double delta) {
59     String vectorMsg = String.format("v1=%s; v2=%s", v1, v2);
60     assertEquals("x components differ: " + vectorMsg, v1.getX(), v2.getX(),
61                 delta);
62     assertEquals("y components differ: " + vectorMsg, v1.getY(), v2.getY(),
63                 delta);
64 }
65
66 }
67 }

```
