Bitte lösen Sie Aufgabe 27-28 bis zum 28.06.2020 **und geben Aufgabe 27 ab!**

Aufgabe 27

ein Billardspiel

Wir programmieren ein Billardspiel mit zunächst drei Kugeln, das wir später erweitern.

Erweiterung von Spielfigur

Erweitern Sie ***SpielFigur*** um die Instanzvariablen ***daempfung*** und ***masse*** mit den **Anfangswerten 1.0**.

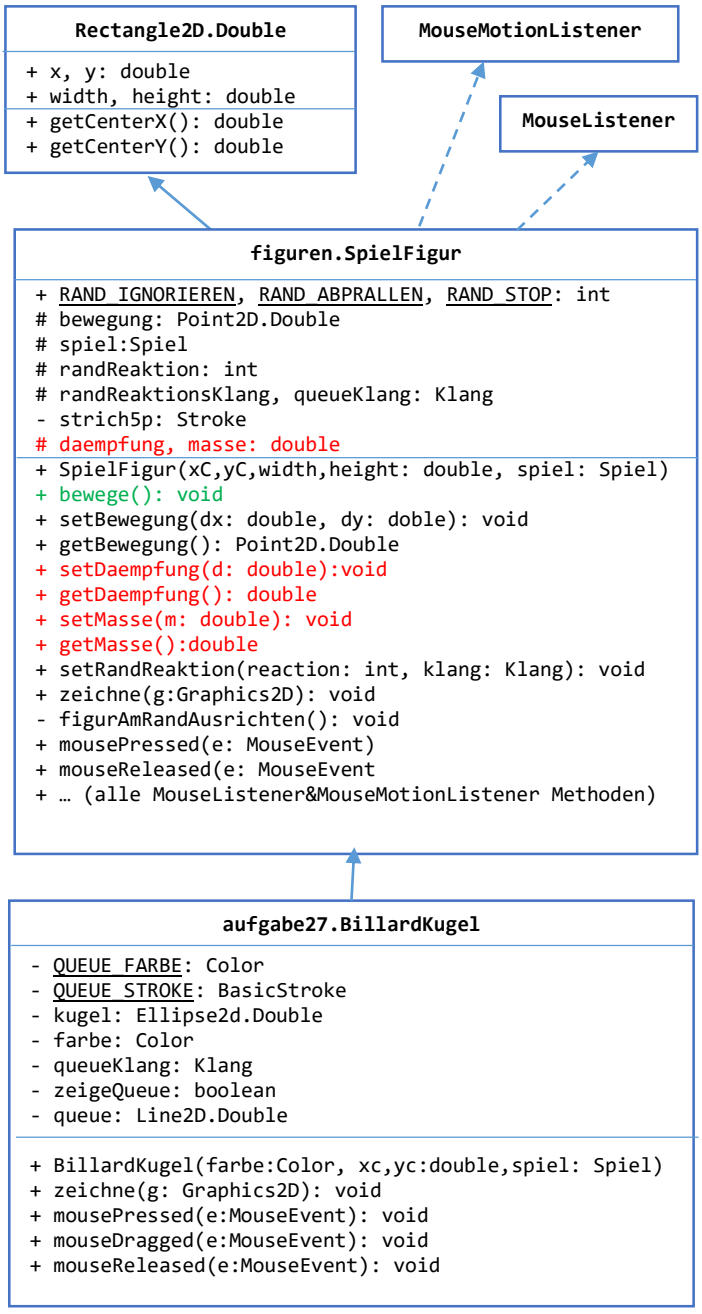
Fügen Sie die Instanzmethoden ***setDaempfung*** und ***setMasse*** sowie ***getDaempfung*** und ***getMasse*** hinzu, mit denen man diese Werte setzen und abfragen kann.

Am Ende der Methode ***bewege*** (hinter der if-Anweisung) multiplizieren Sie den Bewegungsvektor mit *daempfung*:

```
bewegung.x=bewegung.x*daempfung;
bewegung.y=bewegung.y*daempfung;
```

Bei einem Dämpfungswert kleiner 1 werden die Spielfiguren jetzt allmählich langsamer, bei Werten größer 1 beschleunigen sie.

Die Masse brauchen wir erst später.



Die *Billard-Kugel*

Zunächst legen Sie bitte das neue Paket ***aufgabe27*** an. Darin erzeugen Sie die von *SpielFigur* abgeleitete öffentlichen Klasse ***aufgabe27.BillardKugel***.

Eine Billard-Kugel soll bei gedrückter Maustaste den Queue als Strich zwischen Cursor und Mittelpunkt der Kugel zeigen. Beim Loslassen der Taste soll die Kugel in Richtung des Queue angestoßen werden.

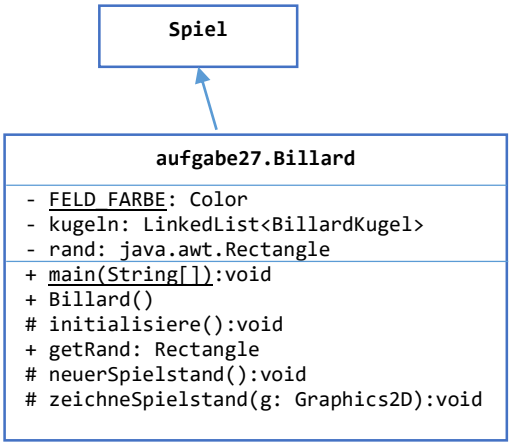
Die Instanzvariablen ***kugel*** und ***queue*** initialisieren Sie mit dem Standardkonstruktor, ***QUEUE_STROKE*** als Strich der Stärke 9 und ***QUEUE_FARBE*** als Braun, z.B. `(89,62,26)`.
Für ***queueKlang*** finden Sie die Datei *billard_kugel_queue.mp3* im online Kurs, ***zeigeQueue*** wird auf *false* gesetzt.

Im **Konstruktor** geben Sie *xC, yC, spiel* sowie die Größe 40,40 an den Basisklassenkonstruktor weiter und speichern *farbe* in der Instanzvariable. Dann rufen Sie *this.setrandReaktion* mit *RAND_ABPRALLEN*, für den Sound beim Abprallen finden Sie *billard_kugel_rand.mp3* im online Kurs. Damit die Kugeln abgebremst werden rufen Sie `this.setDaempfung(0.99)`.

Die Methode ***zeichne*** übernehmen Sie von *ZufallsFigur*.

Die Methoden *mousePressed*, *mouseDragged* und *mouseReleased* bleiben zunächst leer, sie übernehmen später das Anstoßen der Kugel mit dem Queue.

Klasse aufgabe27.*Billard*



Im **Konstruktor** legen Sie die Größe auf 1200*900 Pixel und die Hintergrundfarbe auf ein Braun, das den Rand des Billard-Tisches darstellen soll, z.B. new Color(139,69,19*)* fest. Für *FELD_FARBE* wählen Sie z.B. *Color.GREEN.darker()*.

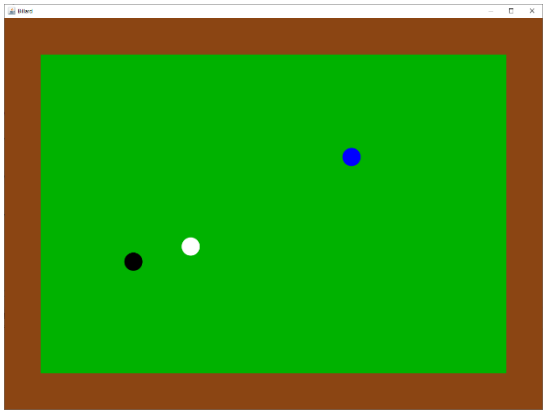
In ***initialisiere*** erzeugen Sie *rand* und *kugeln* mit dem Standardkonstruktor. Dann fügen Sie zu *kugeln* drei Billardkugeln mit unterschiedlichen Positionen, Farben und Geschwindigkeiten hinzu.

Die Methode ***getRand*** übernehmen Sie aus *Aufgabe26*.

in ***zeichneSpielstand*** stellen Sie zunächst das Innere des Billard-Tischs in der Farbe *FELD_FARBE* dar (siehe Aufgabe 26) und rufen in einer *for*-Schleife für jede Kugel die Methode *zeichne*.

In ***neuerSpielstand*** rufen Sie für jede Kugel *bewege*.

Jetzt sollten sich drei Billardkugeln über das Spielfeld bewegen, am Rand abprallen, abbremsen und nach einiger Zeit zum Stehen kommen.

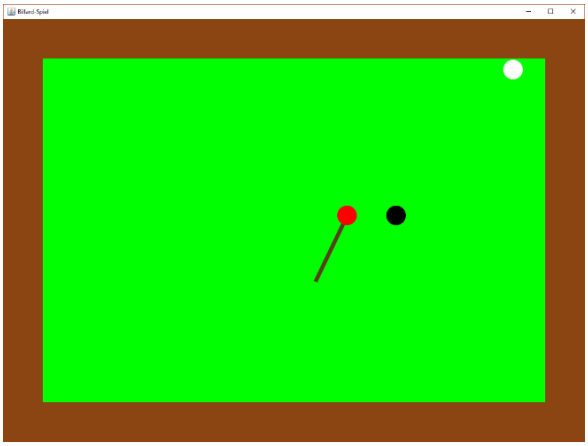


Darstellung des Billard-Queue

Sobald man auf eine Kugel klickt und die Maus bei gedrückter Taste zieht, soll eine Linie zwischen dem Mittelpunk der Kugel und dem Cursor gezeichnet werden.

Um den Queue so darzustellen, erweitern Sie ***BillardKugel*** wie folgt:

- In **mousePressed** prüfen Sie, ob der Cursor innerhalb von *this* liegt. Wenn ja, setzen Sie *zeigeQueue* auf *true* und speichern die Cursorposition in *queue.x1* sowie *queue.y1* (das *Line2D* Objekt *queue* hat zwei Punkte (x1,y1) und (x2,y2) zwischen denen die Linie verläuft).
- In **mouseReleased** setzen Sie *zeigeQueue* auf *false*.
- In ***mouseDragged*** speichern Sie die aktuelle Cursorposition als Anfangspunkt des Queue (*queue.x1*, *queue.y1)*.
- Am Anfang von ***zeichne*** fragen Sie, ob *zeigeQueue true* ist, wenn ja definieren Sie die Strichstärke auf QUEUE_STROKE, die Farbe auf QUEUE_FARBE, die Koordinaten (x2, y2) des Endpunktes von *queue* auf den Mittelpunkt der Kugel und zeichnen den Queue mit *g.draw*.



Jetzt sollte bei Mausbewegung mit gedrückter Taste eine braune Linie zwischen Cursor und Kugel erscheinen, die beim Loslassen der Taste wieder verschwindet.

Anstoßen der Kugeln mit dem Queue

Beim Loslassen der Taste soll die Kugel beschleunigt werden, wobei Richtung und Länge des Queue die Bewegung festlegen.

Das Anstoßen passiert in *mouseReleased,* allerdings nur, wenn *zeigeQueue true* ist*:*

Die durch den Stoß ausgelöste Bewegung berechnen wir aus der Länge der Linie:

```
bewegung.x = 0.12*(queue.x2-queue.x1);
```

und entsprechend für die y-Komponente.

Der willkürlich gewählte Faktor 0.12 hängt unter anderem von der Bildwiederholrate des Spiels ab und sollte ggf. angepasst werden. Am Ende setzt *mouseReleased* wie bisher *zeigeQueue* auf *false*.

Mit dem Billard-Queue können wir jetzt die Kugeln mit variabler Geschwindigkeit anstoßen.

Verwenden Sie die Instanzvariable *queueKlang* welche auf die Sound Datei *billard_kugel_queue.mp3* verweist und spielen sie diesen Klang in *mouseReleased* ab.

Zum Schluß entfernen Sie noch die Anfangsgeschwindigkeit aus dem Konstruktor von *BillardKugel*, so dass die Kugeln am Anfang des Spiels bewegungslos liegen.

Dynamischer Sound

Realistischer wirkt der Sound, wenn wir die Lautstärke dynamisch an die Geschwindigkeit der Kugel anpassen.

Beim Abspielen des Klangs bei der Randreaktion in ***SpielFigur.bewege*** sowie beim Anstoßen der Kugel in ***BillardKugel.mouseReleased*** geben wir der Methode *play* als Lautstärke die Länge des Geschwindigkeitsvektors multipliziert mit 0.03 mit. Die Größe der Geschwindigkeit bekommen Sie durch den folgenden Aufruf: *bewegung.distance(0,0)*. Jetzt sollte sich die Lautstärke der Geschwindigkeit der Kugeln anpassen.

Abgesehen davon, dass die Kugeln noch nicht aneinander abprallen ist unser Billardspiel schon ganz passabel.

Bitte schauen Sie sich die einzelnen Klassen nochmals genau an und versuchen Sie den Zusammenhang zwischen ihnen genau zu verstehen.

Aufgabe 28

Filme enthalten einen Timecode, der jedes Einzelbild eindeutig identifiziert:

hh:mm:ss:ff hh=Stunde, mm=Minute , ss=Sekunde, ff=Frame (Einzelbild)

a) Wieviel Speicherplatz (in Bit) benötigt ein solcher Timecode mindestens (bei optimaler Speicherung), wenn er bei 25 Bildern/Sekunde bis zu maximal 10 Stunden gehen soll?

b) Wie viel Bit Speicherplatz sind mindestens notwendig, wenn Stunde/Minute/Sekunde und Frame jeweils getrennt binär codiert werden?

c) Wie viel Bit sind mindestens notwendig, wenn jede der acht Ziffern einzeln binär codiert wird?

d) Wie groß ist in den Fällen a)-c) die Redundanz?

e) Berechnen Sie Entropie der folgenden Datenquelle:
afedbbbbaaccddddddeeeeedddddddd
(Häufigkeit der Zeichen: 3*a, 4*b, 3*c, 16*d, 6*e, 1*f). Wie groß ist die Entropie pro Zeichen und wieviel Bit Speicherplatz benötigt man, um die gesamte Datenquelle zu speichern?

f) geben Sie einen Huffman Code für diese Datenquelle an. Wie viel Bit benötigt ein Zeichen in dieser Codierung im Durchschnitt?

g) Wie groß ist die Wortlänge, wenn man stattdessen eine Codierung fester Länge verwendet?

h) Wie groß ist in den Fällen f) und g) die Redundanz?

Im online Kurs finden Sie die Datei *EntropieRechner.java*, mit der Sie die Entropie einer Datenquelle berechnen können. Üben Sie das Berechnen der Entropie und das Erzeugen von Huffman Codes für unterschiedliche Zeichenketten. Eine schöne Darstellung für die Erzeugung von Huffman Codes ist https://people.ok.ubc.ca/ylucet/DS/Huffman.html