# Schritt 4: Actor-Objekte zum Leben erwecken

## Der Ball

### Bewegung in x-Richtung

Der Ball soll sich zunächst nur in x-Richtung bewegen.

Öffne die Klasse `Ball` im Editor. Ergänze die Methode `act()` um folgende Zeile:

```
public void act() {
    setLocation(getX() + 1, getY());
}
```

- Erzeuge wieder einen Ball auf dem Spielfeld. Lasse dir den Objektinspector anzeigen. Rufe die Methode `act()` des Balls wiederholt auf. Beobachte sowohl auf dem Spielfeld als auch im Objektinspektor was passiert.
- Klicke dann auf die Schaltfläche Run. Beobachte dabei den Ball und seinen Objektinspektor.
- Wenn der Ball sich nicht mehr bewegt, klicke auf die Schaltfläche Pause.
- Erläutere die Funktionsweisen der drei Schaltflächen Act, Run und Pause.

Ändere nun in der Methode `setLocation(…, …)` den ersten Parameter folgendermaßen ab:

```
setLocation(getX() + 2, getY());
```

Kompiliere dein Programm. Erzeuge einen Ball, klicke auf Run und beobachte wieder das Verhalten des Balls auf dem Spielfeld und die Änderung seiner Werte im Objektinspektor. Was geschieht, wenn du den ersten Parameter in `setLocation` änderst in

- `getX() + 5`
- `getX() + 10`
- `getX() – 1`
- `getX() – 2`
- `getX() – 10`?

Du hast gesehen, dass die Zahl, die zu `getX()` hinzu addiert wird, ein Maß für die Geschwindigkeit des Balls in x-Richtung ist. Ist die Zahl positiv, bewegt sich der Ball nach rechts, ist sie negativ, bewegt er sich nach links. Denke an das Greenfoot Koordinatensystem!

# Die Geschwindigkeit als Attribut (Eigenschaft) des Objekts "Ball"

Damit das Spiel interessanter wird, sollte jeder neue Ball eine jeweils andere Anfangsgeschwindigkeit erhalten. Das realisiert man, indem man ihm jeweils andere Änderung in x-Richtung (dx genannt)

zuweist. Dazu benötigt der Ball ein neues Attribut dx.

Die Klasse Ball sollte dann so anfangen:

```
public class Ball extends Actor {
    private int dx;
...
```

private bedeutet, dass die Variable dx von außen nicht sichtbar ist. Das Paddle oder ein Block können also dx nicht verändern. Man vermeidet dadurch ungewollte Programmierfehler: Die Geschwindigkeit des Balls darf nur der Ball selbst kontrollieren. Dies bezeichnet man als Geheimnisprinzip (dx wird geheim gehalten). Das Gegenteil wäre public: alle Objekte können den Wert des Attributs verändern.

Die Objektvariable dx ist nun zwar deklariert, sie hat aber noch keinen Anfangswert. Denke daran das Attribut im Konstruktor zu initialisieren, d.h. einen Startwert zuzuweisen.

Der Konstruktor für die Klasse Ball sieht also so aus:

```
public Ball() {
    dx = 10 – Greenfoot.getRandomNumber(21);
}
```

`Greenfoot.getRandomNumber(21)` liefert eine Zufallszahl zwischen 0 (einschließlich) und 21 (ausschließlich). Subtrahiert man diese Zufallszahl von 10, erhält man einen zufälligen Wert zwischen –10 und +10 jeweils einschließlich.

Nun muss noch die Methode act() geändert werden:

```
public void act() {
    setLocation(getX() + this.dx, getY());
}
```

# Bitte auch in y-Richtung...

Jetzt soll sich der Ball auch in y-Richtung bewegen. Dabei soll der Ball zu Beginn nur nach oben fliegen.

- Welches Vorzeichen muss dann der Änderungswert dy in y-Richtung haben?
- Erstelle eine Objektvariable dy.
- Initialisiere dy im Konstruktor. Der Anfangswert soll eine Zufallszahl zwischen –10 und –5 jeweils einschließlich sein.
- Ergänze den zweiten Parameter in setLocation(int x, int y) in der Methode act() entsprechend.

Teste das Programm mit „Step“ und „Run“.

# Reflexion am Spielfeldrand

Du hast bemerkt, dass der Ball am Rand „kleben bleibt“. Es wäre schöner, wenn er an den Spielfeldrändern seine Flugrichtung umkehren (reflektiert) würde. Bei der Reflexion an den Spielfeldrändern ändert einer der Werte von dx oder dy sein Vorzeichen.

Die Umkehrung des Vorzeichens kann man folgendermaßen erreichen:

```
// dx umkehren
dx = -dx;
// dy umkehren
dy = -dy;
```

- Fertige eine Skizze für die Reflexion am linken/rechten Rand an.
- Wie ändern sich die Werte von dx und dy bei der Reflexion am
  - oberen Rand?
  - linken Rand?
  - unteren Rand?
  - rechten Rand?

Der Ball hat den Spielfeldrand erreicht, wenn sein Mittelpunkt weniger als die Hälfte seiner Breite bzw. Höhe vom Spielfeldrand entfernt ist. Um z.B. zu überprüfen, ob der rechte Rand erreicht ist, kann man die Methode `reflektiereRandRechts()` implementieren.

Beachte dabei, dass das Spielfeld 640 Pixel breit ist und der Ball einen Durchmesser von 28 Pixel hat. (Wenn du eine andere Ballgrafik verwendest, musst du das evtl. anpassen!)

```
private void reflektiereRandRechts() {
   if (getX() >= 626) {
       dx = -dx;
   }
}
```

Hier wird geprüft, ob die x-Koordinate des Balls – ermittelt mit `getX()` – größer oder gleich der Breite des Spielfeldes vermindert um die Hälfte der Ballbreite ist. Das Gleichheitszeichen ist zu setzen, weil die Zählung der Pixel-Nummern bei Null beginnt.

Wenn diese Bedingung zutrifft, wird die x-Richtung des Balls umgedreht (`dx = -dx;`). Die Methode `reflektiereRandRechts()` wird in der Methode `act()` aufgerufen. Damit ist sicher gestellt, dass die Überprüfung auf Kontakt mit dem rechten Rand in jedem Frame einmal stattfindet[1)].

- Erstelle die Methode `reflektiereRandRechts()`.
- Füge den Methodenaufruf deiner neuen Methode in die Methode `act()` vor der Zeile `setLocation(getX() + dx, getY() + dy);` ein.

Übersetze dein Programm und teste, ob der Ball tatsächlich am rechten Rand reflektiert wird.

- Schreibe die Methoden

    - `reflektiereRandLinks()`
    - `reflektiereRandOben()` und
    - `reflektiereRandUnten()`[2)]
- Füge die Methodenaufrufe deiner neu erstellten Methoden an geeigneter Stelle in `act()` ein.

Übersetze dein Programm und teste, ob der Ball nun an allen Rändern reflektiert wird.

# Der Ball soll am unteren Rand verschwinden

In der nächsten Stufe soll der Ball nun nicht mehr am unteren Spielfeldrand reflektiert werden, sondern dort verschwinden.

- Ersetze die Methode reflektiereRandUnten durch verschwindeRandUnten()mit der Anweisung zum Entfernen des Balls aus der Welt: `getWorld().removeObject(this);`

Überprüfe, ob deine Bälle am unteren Rand korrekt verschwinden.

**Hinweis:** Die Abmessungen des Balls müssen am unteren Rand nun nicht mehr beachtet werden. Der Mittelpunkt des Balls darf aber höchstens bis zur letzten Pixel-Reihe wandern. Denke auch daran, dass die Zählung der Pixel-Reihen bei Null beginnt.

**Erklärung von `getWorld().removeObject(this)`**

Das Spielfeld kann Objekte entfernen durch seine Methode `removeObject(Actor object)`. Der Ball muss dem Spielfeld diesen Befehl geben. Dazu fragt der Ball durch `getWorld()` erst nach, wer sein Spielfeld ist. Das zu entfernende Objekt ist der Ball selbst `this`. Umgangssprachlich formuliert sendet der Ball an das Spielfeld folgende Botschaft: „Hallo Spielfeld. Entferne ein Objekt, nämlich mich selbst!“

Wenn dein Ball tatsächlich verschwindet, es aber eine Fehlermeldung gibt, weil nach dem Entfernen des Balls für diesen Ball in der Methode `act()` noch weitere Anweisungen durchgeführt werden sollen, obwohl er gar nicht mehr vorhanden ist, musst du sicherstellen, dass nach dem Entfernen des Balls keine weiteren Methodenaufrufe in den Ball `act()` erfolgen. Verschiebe die Methode `verschwindeRandUnten()` an die letzte Stelle in `act()`. Überprüfe erneut.

---

<<< Zurück zu Schritt 3 | **Breakout: Schritt 4** | Kapitelübersicht | Weiter zu Schritt 5 >>>

---

Alle Anleitungen in diesem Namensraum basieren auf den Materialien der Fachberatergruppe Informatik am RP Karlsruhe.

1) Ein *Frame* ist bei Greenfoot der Durchlauf durch alle `act()`-Methoden aller Klassen - bevor das dann wieder von vorne beginnt, um den nächsten Frame zu erzeugen

2) Beachte dabei die Höhe des Spielfelds und die Abmessungen des Balls.

From:
https://wiki.qg-moessingen.de/ - **QG Wiki**

Permanent link:
**https://wiki.qg-moessingen.de/faecher:informatik:mittelstufe:breakout:breakout04:start?rev=1633011474**

Last update: **30.09.2021 16:17**