Java Tutorial anhand eines kleinen 2D-Spiels in Greenfoot

Simon Teiwes

20. November 2017

Das Spiel

Es soll ein einfaches 2D-Spiel entworfen werden, bei dem der Spieler einen Akteur (z.B. eine Rakete) steuert, welcher vom einen Ende der Welt zum anderen (z.B. von links nach rechts) gelangen soll. Das Spiel soll schrittweise mit Gegnern, einem Inventar und Gegenständen erweitert werden. Du kannst gerne eigene Ideen einarbeiten und Grafiken gestalten.

1 Aufgabe - Die Welt

Erstelle zunächst ein neues Projekt mit Greenfoot (Neues Java-Szenario). Dein Szenario sollte nun wie in Abb. 1 aussehen. Rechts siehst du das Klassendiagramm. Greenfoot stellt schon die Klassen `World` und `Actor` bereit, die viele nützliche Methoden enthalten, mit denen man sein Spiel gestalten kann. Mache dich kurz mit der Greenfoot-API vertraut, indem du zunächst die Dokumentation der Klasse `World` anschaust (Diese öffnest du durch Doppelklick auf die `World`-Klasse im Klassendiagramm). Dort sind einige Methoden aufgelistet, mit denen du deine eigene Welt gestalten und verändern kannst. So ist es möglich die Zellengröße zu ändern (Orte an denen sich ein Actor befinden kann) und das Hintergrundbild zu setzen.

a)

Erstelle eine Unterklasse von `World`, namens `SpaceWorld` welche die Welt repräsentiert, in der der spätere Akteur existieren soll. Im neuen Projekt existiert bereits eine Unterklasse `MyWorld`, diese kann im Quellcode (Öffnen ebenfalls durch Doppelklick) einfach umbenannt werden, dort musst du allerdings auch den Konstruktor anpassen. Schreibe Klassennamen immer mit einem großen Anfangsbuchstaben, Variablen- oder Objektnamen und Methoden immer mit kleinem Anfangsbuchstaben. Verwende die Camel-Case Notation (`http://www.datenbanken-verstehen.de/namenskonventionen-datenbank/camel-case-notation/`)

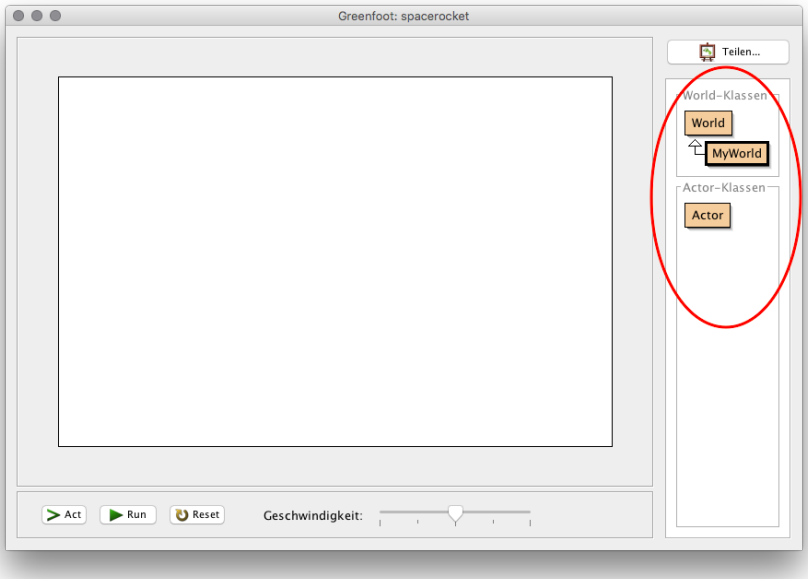

Abbildung 1: Initiales Szenario mit Klassendiagramm

und orientiere dich an dem mitgeschickten Dokument zum Style-Guide von der Universität Bielefeld.

Wähle für deine Welt eine Zellengröße von 1x1 Pixel und Maße von 900 Pixeln Breite und 400 Pixeln Höhe. Gib deiner Welt ein Hintergrundbild (Z.B. `spacebg.png` aus dem Bilder-Ordner).

Dokumentiere jede Methode, indem du nach folgendem Beispiel einen Kommentar über ihr einfügst:

```
/**
 * Zeigt beispielhaft, wie man ordentlich Dokumentiert
 *
 * @param ersterParameter eine ganzzahlige Zahl
 * @return eine Zahl als Wort
 */
public String beispielMethode(int ersterParameter) {
    // tu was die Methode tun soll
}
```

Die Wörter `@param` und `@return` sind Schlüsselwörter für den Javadoc-Compiler, welcher aus den Quellcode-Kommentaren HTML- oder PDF-Dokumente erstellt. Wenn du in Greenfoot den Quellcode einer Klasse geöffnet hast, kannst du oben rechts im Drop-Down-Menü auf *Dokumentation* umstellen, dann siehst du

das Ergebnis von Javadoc. Für weitere Informationen zum Java-Dokumentationsstil siehe `http://www.oracle.com/technetwork/java/javase/documentation/index-137868.html#styleguide`.

2 Aufgabe - Massige Aktoren

a)

Die Welt soll nun von Aktoren bevölkert werden. Dabei soll zunächst eine Klasse für Dinge entworfen werden, welche eine genaue Position und Geschwindigkeit besitzen (z.B. `MassActor`). Ebenso sollen die Objekte geschwindigkeitsabhängig gebremst werden, ähnlich wie beim Luftwiderstand. Die neue Klasse soll von `Actor` erben und `abstract` sein, da von ihr nicht direkt Instanzen erstellt werden sollen. Dementsprechend benötigt die Klasse auch keinen Konstruktor. Sollen später Instanzen erstellt werden (Z.B. unsere Rakete), so müssen diese durch Unterklassen spezifiziert werden.

Beginne zunächst mit der Deklaration der benötigten Variablen. Um den beschriebenen Eigenschaften gerecht zu werden, brauchen wir zwei Fließkommawerte für die Position, zwei für die Geschwindigkeit und zwei für die Reibung. Die Positions- und Geschwindigkeitswerte sind in `World`-Feldgrößen, also in Pixeln angegeben. Die Reibungskoeffizienten sollen einen Wertebereich zwischen 0 (keine Reibung) und 1 (volle Reibung, also keine Bewegung mehr möglich) haben. Nutze den Datentyp `double` und mache die Variablen `protected`, damit nur die Unterklassen direkten Zugriff auf sie haben. Andere Klassen, wie z.B. `World` werden mit getter-Methoden auf diese Werte zugreifen. Des weiteren brauchen wir für spätere Zwecke eine Referenz auf die Welt, in der eine Instanz lebt. Auch dafür soll eine Variable erstellt werden.

Implementiere nun die `act`-Methode, welche das zeitliche Verhalten beschreibt. `act` wird in bestimmten Zeitschritten von der Greenfoot-Umgebung aufgerufen, also müssen dort die Geschwindigkeiten auf die Position addiert werden, um eine Bewegung zu realisieren. Anschließend müssen die Geschwindigkeiten noch mit dem Reibungskoeffizienten vermindert werden. Schlussendlich muss die neue Position noch der Greenfoot-Umgebung mitgeteilt werden, dafür nutze die `Actor`-Methode `setLocation(int x, int y)`. Beachte, dass du die Fließkommazahlen zu Ganzzahlen umwandeln musst. Nutze dafür die Rundungsfunkion von `java.lang.Math`. Dafür musst du bei den import-Statements ganz oben `import java.lang.Math;` hinzufügen. Nutze die Java-Dokumentation für die `Math`-Klasse aus dem Internet.

Weiterhin benötigen wir eine neue Methode zum setzen der Position und Geschwindigkeit (`setPose()`). Die Methode muss nichts zurückgeben und bekommt die vier Parameter als `double`. Auch hier soll wieder `setLocation(int x, int y)` aufgerufen werden.

Außerdem sollen noch zwei Methoden zum Beschleunigen implementiert werden. Dafür verwenden wir den gleichen Namen, überladen also die Methode. Eine Methode `accelerate(double acc)` soll den `MassActor` in Richtung seiner Ausrichtung beschleunigen (nützlich bei der Rakete, die nur ein Triebwerk besitzt), und die andere Methode `accelerate(double accX, double accY)` soll die Beschleunigung in X- und Y-Richtung festlegen. Mache dir dafür noch einmal klar, wie die zeitlichen Zusammenhänge zwischen Position, Geschwindigkeit und Beschleunigung sind.

Um die Beschleunigung in Richtung der aktuellen Ausrichtung zu realisieren, musst du mit trigonometrischen Funktionen die X- und Y-Komponente errechnen. Die `Actor`-Methode `getRotation()` gibt dir die Ausrichtung in Grad zurück.

Schlussendlich musst du noch die Methode `addedToWorld(World world)` implementieren. Diese wird aufgerufen, sobald eine Instanziierung zur Welt hinzugefügt wurde. Hier speichern wir uns die übergebene Welt ab, indem wir sie der anfänglich erstellten Variable zuweisen. Wichtig ist hier auch, dass wir die initialen Koordinaten aus der Greenfoot-Umgebung in unserer doppelten Buchführung übernehmen. Diese bekommen wir mit den `Actor`-Methoden `getX()` und `getY()`. Auch hier ist wieder Vorsicht geboten, da wir Fließkommazahlen benötigen.

Da Aktoren in Greenfoot nur mit ganzzahligen Werten beschrieben werden können, betreiben wir doppelte Buchführung mit Fließkommazahlen. Die Berechnung findet dadurch sehr genau statt und erzeugt ein realistischeres Verhalten.

b)

Um deine Implementierung nun endlich nutzen zu können, erstellst du eine Unterklasse von `MassActor` namens `Rocket` und wählst das Raketenbild aus den Greenfoot-Beispielbildern. Eine Instanziierung dieser Klasse soll später der Hauptcharakter unseres Spiels werden. `Rocket` ist keine abstrakte Klasse, da wir ja Instanzen erstellen wollen. Dafür benötigen wir auch einen Konstruktor. Im Konstruktor werden die Reibungskoeffizienten und die Beschleunigung, sowie die anfängliche Geschwindigkeit der Rakete festgelegt. Die Beschleunigung soll als Objekt-Attribut einmal festgelegt werden, soll allerdings im späteren Spielverlauf verändert werden können. Du kannst dir zunächst sinnvolle Werte aussuchen und schauen zu welchem Verhalten sie führen.

Steuerung

Die Rakete soll mit den Pfeiltasten steuerbar sein: Mit rechts und links soll sie gedreht werden können und oben beschleunigt die Rakete in Richtung ihrer Ausrichtung. Implementiere dafür zwei neue Methoden `turnOnKeyPress(String key, int angle)` und `accelerateOnKeyPress(String key, double acceleration)`. In den Methoden soll mittels der `Greenfoot`-Klasse, welche Funktionen zur Interaktion mit dem System bereitstellt, geprüft werden ob die übergebene Taste gedrückt ist. Anschließend soll die Rakete entsprechend beschleunigt, oder gedreht werden.

Sind die Methoden fertig implementiert, so sollen sie in der `act`-Methode der `Rocket` aufgerufen werden. Beispiel: `turnOnKeyPress("left", -5);`.

Nun solltest du die Rakete in der Welt umherfliegen können. Du kannst mit Rechtsklick auf die Raketenklasse eine neue Instanz erstellen und sie in der Welt platzieren.

Tipp: Vergiss nicht, am Ende der `act`-Methode der Rakete auch das `act` der Überklasse aufzurufen! Falls etwas nicht läuft, kannst du auch debug-Ausgaben einfügen.