KAPITEL

13 Weitere Szenarien

Lernziele

Themen: Ideen für weitere Szenarien

Konzepte: (es werden keine neuen Konzepte vorgestellt)

Damit kommen wir zum letzten Kapitel dieses Buches. Es unterscheidet sich von den anderen Kapiteln, da es nicht versucht, neue Konzepte oder Programmiertechniken einzuführen. Stattdessen werden dir kurz ein paar weitere Szenarien vorgestellt, die dich inspirieren sollen, eigene Ideen zu entwickeln und zu implementieren.

Alle hier beschriebenen Szenarien sind auch als Greenfoot-Projekte mit Quelltext in den Buchszenarien zu finden. Bei den meisten Implementierungen wurden die Ideen nur teilweise umgesetzt.

Einige Szenarien sind fast vollständig und bieten sich zum Selbststudium an, um neue Techniken kennenzulernen und nachzuvollziehen, wie bestimmte Effekte erzielt werden. Andere implementieren die Ideen nur in Ansätzen, d.h., es sind partielle Implementierungen, die du als Ausgangsbasis für deine eigenen Projekte verwenden kannst. Noch andere veranschaulichen lediglich ein bestimmtes Konzept, das als Anregung dienen soll, etwas Ähnliches in deinem eigenen Szenario einzubauen.

Kurz und gut, betrachte diese Beispiele als eine Sammlung von Ideen für zukünftige Projekte, deren Studium sich lohnt, weil sie dir zeigen, was ein tüchtiger Programmierer sonst noch alles erreichen kann.

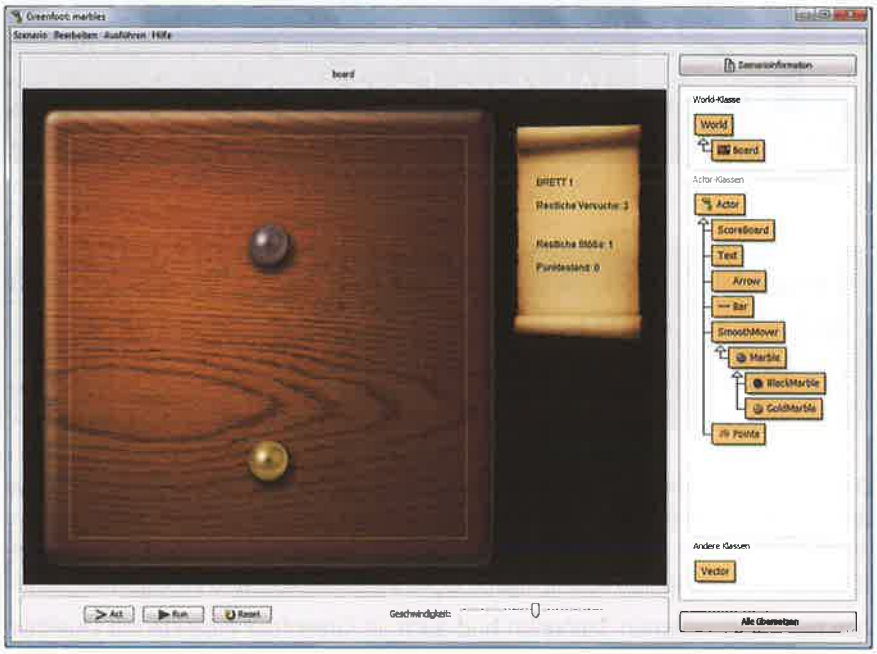
13.1 Murmeln

Das Szenario *marbles* (Abbildung 13.1) implementiert ein Spiel, bei dem du eine goldene Kugel über ein Spielbrett rollst, mit dem Ziel, alle silbernen Kugeln mit einer begrenzten Anzahl von „Stößen" vom Brett zu schubsen. Das Spiel ist mehr oder weniger fertig.

An diesem Szenario sind mehrere Dinge bemerkenswert. Zuerst fällt auf, dass das Spiel optisch nett gestaltet ist. Das hat sehr wenig mit der Java- oder Greenfoot-Programmierung zu tun, sondern basiert zum großen Teil auf der Verwendung ansprechender Grafiken. Ansprechend aussehende Grafiken und gute Sounds können einen großen Unterschied machen, was die Attraktivität eines Spiels angeht.

Marbles verwendet ein attraktives Hintergrundbild (das Spielbrett und die Schriftrolle für die Anzeige des Textes) und Akteure mit halbtransparenten Schlagschatten (die Murmeln und die Hindernisse).

Abbildung 13.1
Das Murmelspiel.



Der andere interessante Aspekt, der eine Untersuchung lohnt, ist die Kollisionserkennung. Die Murmeln verwenden keine der vorgegebenen Greenfoot-Methoden für die Kollisionserkennung, da diese sich nur auf rechteckige Akteur-Bilder anwenden lassen. Die Murmeln sind jedoch rund und hierfür benötigen wir eine genaue Kollision.

Zum Glück ist dies bei runden Akteuren nicht besonders schwer. Zwei Murmeln stoßen zusammen, wenn ihr Abstand (gemessen von ihren Mittelpunkten) kleiner als ihr Durchmesser ist. Den Durchmesser kennen wir und der Abstand ist mithilfe des Satzes von Pythagoras relativ leicht zu berechnen.

Der nächste Aspekt von Interesse ist die Berechnung der neuen Bewegungsrichtung einer angestoßenen Murmel. Hierbei werden ein paar Trigonomie-Kenntnisse vorausgesetzt, doch wenn du damit vertraut bist, solltest du keine allzu großen Verständnisschwierigkeiten haben.

Die Kollisionen mit festen Hindernissen sind leichter, da es sich bei den Hindernissen immer um vertikal oder horizontal angeordnete Rechtecke handelt. Wenn eine Murmel also auf eines dieser Hindernisse trifft, ändert sie einfach ihre Richtung entlang einer dieser Achsen (x oder y).

Die Kollisionslogik der Murmeln lässt sich auf alle möglichen anderen Spiele übertragen, bei denen runde Objekte zusammenstoßen.

13.2 Fahrstühle

Das Szenario *lifts* simuliert ein einfaches Fahrstuhl- oder Aufzugsystem (Abbildung 13.2). Es zeigt mehrere Etagen eines mehrstöckigen Gebäudes und drei Aufzüge, die rauf- und runterfahren. Personen erscheinen auf den Etagen, drücken die Aufzugschalter und betreten den Aufzug, wenn er kommt.

Abbildung 13.2
Eine teilweise implementierte Aufzugsimulation.

Diese Implementierung ist jedoch sehr rudimentär. Viele Teile der Simulation auf dem Bildschirm sind nur Attrappen. Sie simulieren nicht das, was eigentlich geschieht, sondern dienen nur der Darstellung.

So steigen die Personen zum Beispiel nicht, wie man erwarten würde, in den Aufzug, sondern sie werden einfach gelöscht, wenn einer der Aufzüge auf der Etage hält. Die Anzahl der Personen im Aufzug wird durch eine Zufallszahl vorgegeben. Außerdem reagieren die Aufzüge nicht auf das Drücken der Aufzugsschalter – sie fahren einfach willkürlich rauf und runter. Für die Aufzüge ist noch kein Steuerungsalgorithmus implementiert worden.

Dies ist also mehr oder weniger eine Demo-Version, die nur die Ideen und die Grafiken liefert. Um das Projekt zu vervollständigen, müsstest du erst einmal die Bewegungen der Personen (beim Betreten und Verlassen der Aufzüge) ordentlich modellieren. Anschließend könntest du dann dazu übergehen, die verschiedenen Algorithmen zur Aufzugsteuerung zu implementieren und zu testen.

13.3 Boids

Das Beispiel *boids* simuliert das Schwarmverhalten von Vögeln (Abbildung 13.3).

Der Begriff „Boids" basiert auf einem Programm, das 1986 von Craig Reynolds entwickelt wurde und diesen Schwarmalgorithmus das erste Mal implementierte. Dem Algorithmus zufolge berücksichtigt jeder Vogel bei seinem Flug die folgenden drei Regeln:

- Separation: Gehe auf Abstand zu anderen Vögeln, wenn du ihnen zu nahe kommst.
- Angleichung: Richte deinen Flug an die mittlere Flugrichtung der anderen Vögel in der Nähe aus.
- Zusammenhalt: Bewege dich auf die mittlere Position der anderen Vögel in deiner Nähe zu.

Bei diesem Algorithmus ergibt sich für das Flugverhalten der Vögel ein sehr interessantes Bewegungsmuster. Außerdem wurde in diesem Szenario eine Hinderniserkennung implementiert, d.h., die Vögel versuchen, nicht in die Bäume zu fliegen.

Benutzt wurde dieser Algorithmus zum Beispiel in Tim Burtons Film „Batmans Rückkehr" von 1992 für die Animationen der computererzeugten Fledermausschwärme und Pinguinherden sowie bei den „Herr der Ringe"-Verfilmungen, um die Bewegung der Orks-Armee zu simulieren.

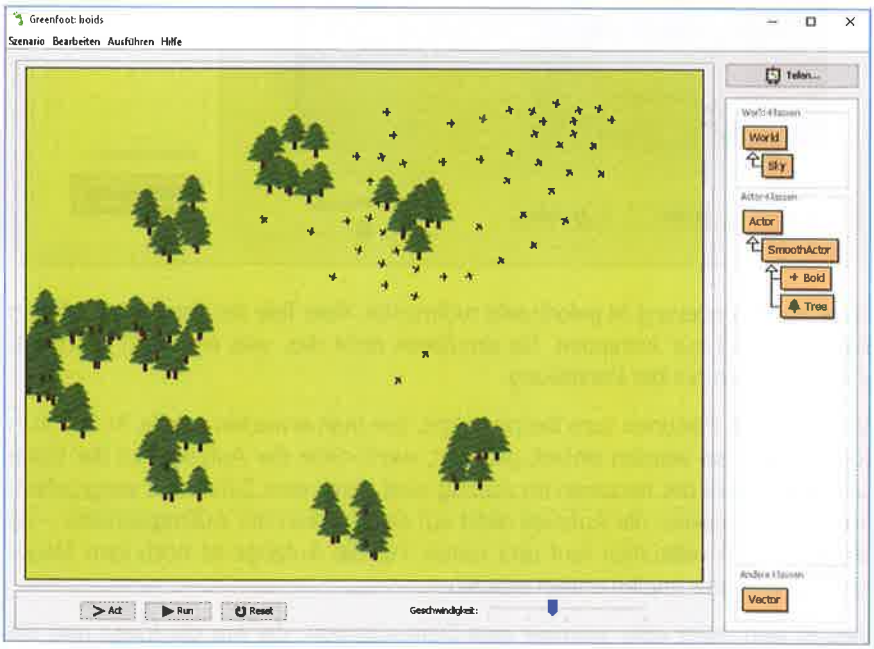


Abbildung 13.3
Boids: Eine Simulation des Schwarmverhaltens.

Die Version für dieses Buch wurde von Poul Henriksen geschrieben.

Wenn du nach „Boids" suchst, findest du im Internet eine Unmenge an weiteren Informationen. Und obwohl dieses Szenario eigentlich nur die Bewegung zeigt, lässt einen das Gefühl nicht los, dass da irgendwo ein Spiel drin steckt.

13.4 Explosionen

Das Szenario *explosion* (Abbildung 13.4) zeigt, wie wir einen spektakuläreren Explosionseffekt implementieren können. Das explodierende Objekt ist in unserem Fall ein einfacher Gesteinsbrocken, den wir bereits aus anderen Szenarien kennen (er wurde zum Beispiel als Asteroid in unserem *asteroids*-Szenario verwendet). Wir können jedoch alles Mögliche explodieren lassen.

Um diesen Effekt zu erzielen, haben wir eine Klasse **`Debris`** (Bruchstücke) eingerichtet, die einen Teil von einem Gesteinsbrocken repräsentiert. Wenn der Gesteinsbrocken explodiert, entfernen wir ihn aus der Welt und platzieren stattdessen an der Stelle 40 Bruchstücke. Um jedem Bruchstück etwas Einzigartiges zu verleihen, wird es per Zufall gestaucht oder gestreckt und gedreht und erhält am Anfang einen Bewegungsvektor in eine willkürliche Richtung. Bei jedem Schritt ergänzen wir eine Abwärtsbewegung, um die Schwerkraft zu simulieren. Das Ergebnis der Explosion kannst du sehen, wenn du dieses Szenario ausführst.

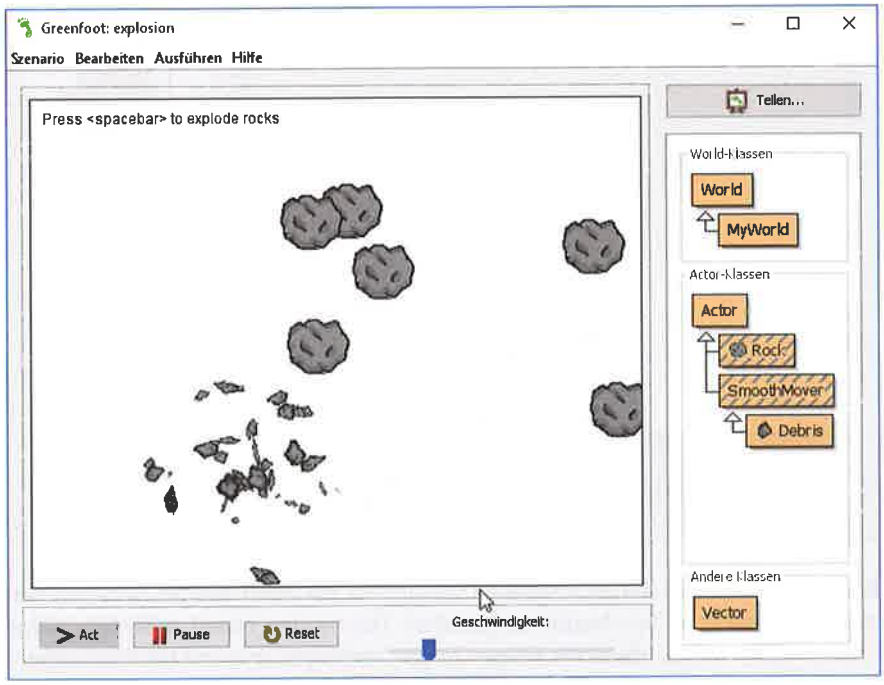


Abbildung 13.4
Ein Explosionseffekt.

Im Greenfoot-YouTube-Kanal unter *https://www.youtube.com/user/18km* findest du ein Video-Tutorial, das dieses Szenario ausführlich erläutert.

13.5 Breakout

„Breakout" ist ein klassisches Computerspiel, bei dem der Spieler mithilfe eines Schlägers unten im Bildschirm einen auf und ab hüpfenden Ball steuert, um Ziegel aus einer Mauer zu entfernen. Wenn du das Spiel nicht kennst, recherchiere ein wenig im Internet.

Das *breakout*-Szenario (Abbildung 13.5) implementiert dieses Spiel nur zum Teil. Der hier verwendete Ball weist einen Rauchfahneneffekt auf, den wir bereits in ***Kapitel 10*** besprochen haben. Zur Steuerung des Balls dient dem Spieler ein Schläger. Was jedoch noch fehlt, sind die Ziegel, die „abgebaut" werden sollen; das bedeutet, dass das Spiel in seiner jetzigen Form nicht besonders interessant ist.

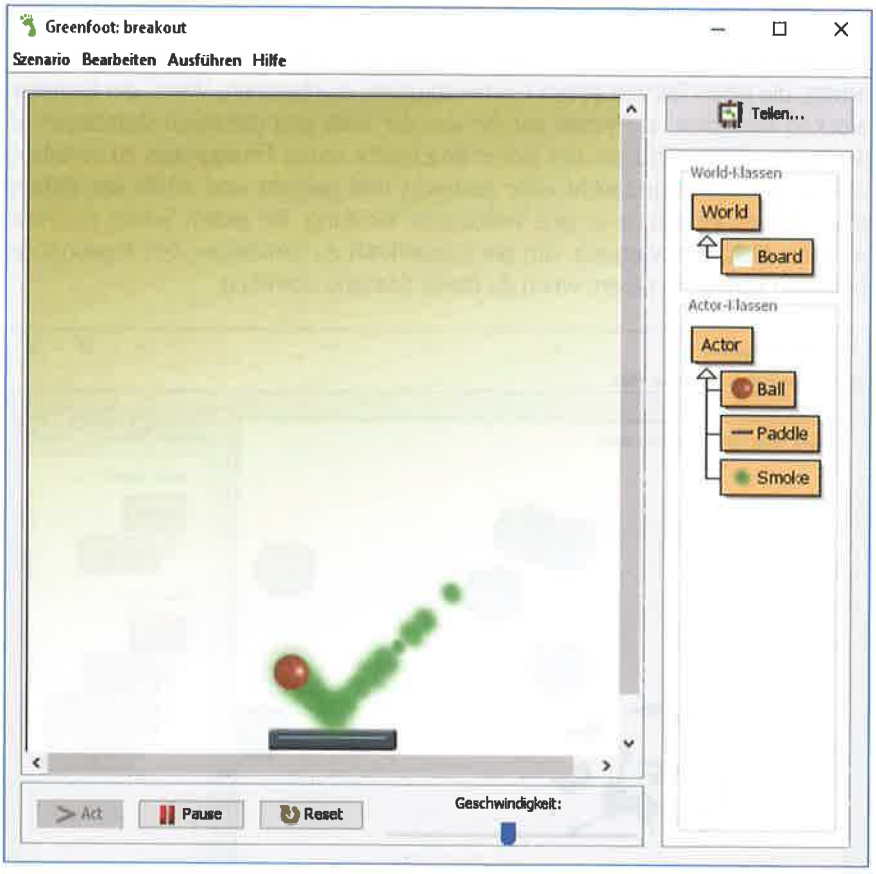


Abbildung 13.5
Das Anfangsstadium eines „Breakout"-Spiels.

Breakout gibt es in vielen verschiedenen Variationen. Bei vielen sind die Ziegel auf den einzelnen Ebenen unterschiedlich angeordnet. Die meisten bieten außerdem einige „Power-ups" – Extras, die hinter einigen Ziegeln verborgen sind und herabschweben, wenn der Ziegel zerstört wird. Wenn du sie fängst, wird das Spiel noch interessanter (ein Zusatzball, höhere Geschwindigkeit, größere oder kleinere Balken usw.).

Dieses Spiel auf interessante Art und Weise zu vervollständigen, bietet sich praktisch als Projekt an. Es kann auch mit zwei Schlägern (links und rechts) implementiert werden, sodass das Spiel eher dem klassischen „Pong"-Spiel ähnelt.

13.6 Plattform-Springer

Weite Verbreitung haben auch die sogenannten „Plattform"-Spiele gefunden. Hierbei steuert ein Spieler in der Regel eine Spielfigur, die von einem Bildschirmbereich zu einem anderen gelangen muss und dabei etliche Hindernisse zu überwinden hat. Ein solches Hindernis könnte eine Kluft im Boden sein, auf dem die Spielfigur wandert. Es sollte jedoch immer eine Möglichkeit geben, diese Kluft zu überbrücken.

Das Szenario *pengu* implementiert ausschnittsweise ein solches Spiel (Abbildung 13.6). Es gibt zur Linken und Rechten des Bildschirms zwei Felsen, die durch eine Kluft getrennt sind. Der Pinguin kann die Kluft überqueren, indem er auf eine sich hin und her bewegende Wolke springt.

Dieses Szenario soll veranschaulichen, wie sich ein Akteur auf einem anderen fortbewegen kann (der Pinguin auf dem Felsen) und wie man das Springen und Fallen implementieren kann.

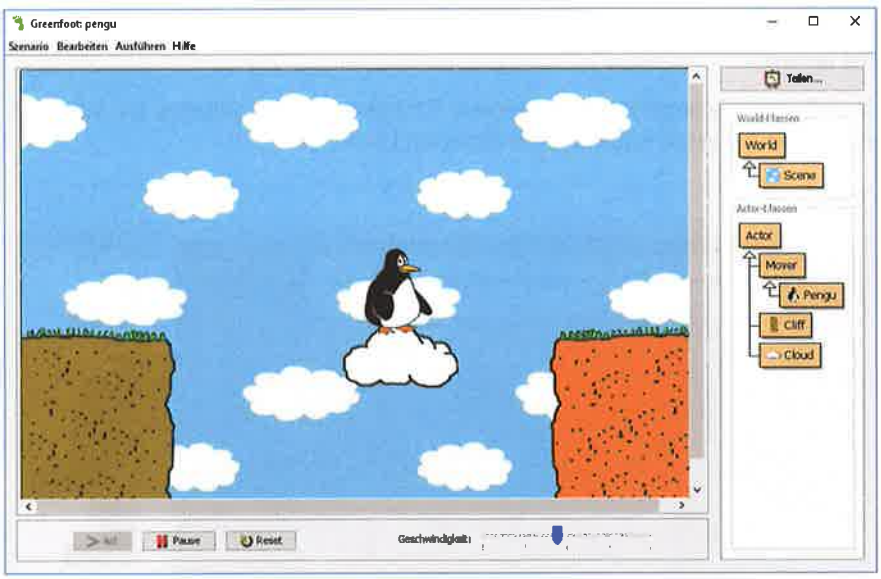


Abbildung 13.6
Der Beginn eines einfachen Spiels mit einem Plattform-Springer.

Im Greenfoot-YouTube-Kanal unter *https://www.youtube.com/user/18km* findest du ein Video-Tutorial mit der Bezeichnung „Running, jumping and falling", das dieses Szenario erläutert.

13.7 Wave

Das nächste Szenario trägt den Namen *wave* (Abbildung 13.7). Es handelt sich dabei um eine einfache Simulation einer Wellenausbreitung in einer Perlenschnur. Spiele ein wenig mit dem Szenario herum, um festzustellen, was du damit alles machen kannst.

Faszinierend an diesem Beispiel ist unter anderem, wie eine relativ einfache Implementierung – bei jedem **act**-Schritt bewegt sich jede Perle einfach in Richtung Mitte ihrer beiden benachbarten Perlen – zu einer relativ anspruchsvollen Simulation von verschiedenen Aspekten der Wellenausbreitung führt.

Wir haben dieses Beispiel hier aufgeführt, um zu zeigen, dass mit entsprechenden Vorüberlegungen und Vorbereitungen verschiedene Verhaltensweisen aus anderen Disziplinen simuliert werden können. In diesem Fall ist es ein einfacher physikalischer Effekt. Ebenso gut könnte man auch chemische Reaktionen, biologische Interaktionen, Interaktionen von subatomaren Teilchen und vieles mehr simulieren. Bei entsprechend sorgfältiger Vorbereitung können wir nicht nur etwas über die Programmierung, sondern auch über andere Anwendungsbereiche lernen.

Zusätzlich implementiert dieses Szenario Schieberegler und Schalter, die sich auch in anderen Projekten nutzbringend einsetzen lassen.

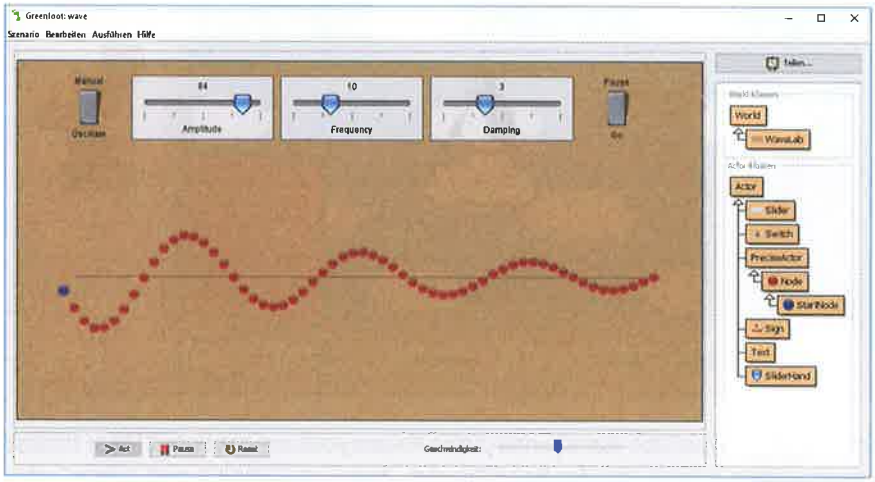


Abbildung 13.7
Simulation einer Wellenausbreitung in einer Perlenschnur.

13.8 Karten

Das letzte Szenario, das wir hier vorstellen, heißt *maps* (Karten, Abbildung 13.8). Es zeigt eine Karte auf dem Welthintergrund an und hat Tasten zum hinein- und herauszoomen.

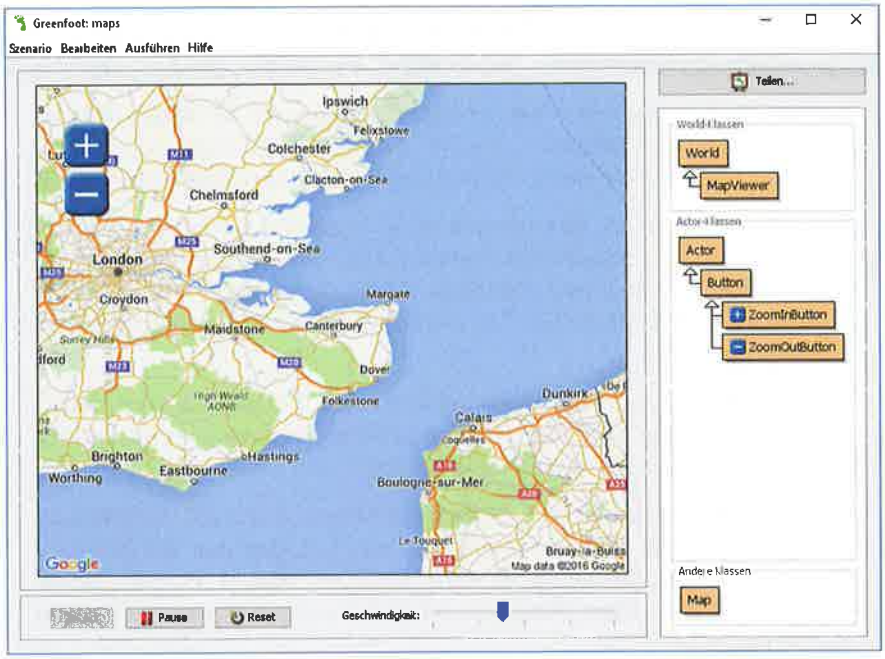


Abbildung 13.8
Ein Greenfoot-Szenario mit Echtzeitzugriff auf die Karte.

Der interessante Aspekt dieser Klasse ist, dass hier Echtzeitdaten benutzt werden, auf die über das Internet zugegriffen wird (in diesem Fall sind es die Kartendaten). Das Bild der Karte ist nicht starr: Es ist kein fester Bestandteil dieses Szenarios, sondern wird zur Laufzeit vom Onlinedienst Google-Maps abgerufen. (Dies bedeutet natürlich auch, dass dieses Szenario nur auf einem Computer läuft, der mit dem Internet verbunden ist.)

In seiner aktuellen Form ist die Universität von Kent der Mittelpunkt der Karte, aber das kann leicht geändert werden. Wirf einen Blick in die Klasse **MapViewer**, dort wirst du in den Kommentaren mehrere weitere Beispiele von Orten finden, die du ausprobieren kannst. Prinzipiell kannst du jeden Ortsnamen benutzen, dessen Name sich hinreichend von anderen unterscheidet – sei es der Name eines Landes, einer Stadt oder einer Landmarke, oder du verwendest Längen- und Breitengrade.

Die Implementierung ist relativ einfach, weil dazu eine fertige **Map**-Helferklasse eingesetzt wird – diese erledigt den größten Teil der Arbeit.

Die **Map**-Helferklasse gehört zu den Klassen, die Greenfoot zur Verfügung stellt: Du kannst über die Funktion IMPORTIERE KLASSE im BEARBEITEN-Menü des Hauptfensters darauf zugreifen.

Ganz allgemein bekommt man durch den Zugriff auf Echtzeitdaten aus dem Internet einige interessante Fälle. Über denselben Menüpunkt (IMPORTIERE KLASSE) kannst du zum Beispiel eine weitere Helferklasse namens **Weather** importieren, welche dir Echtzeit-Wetterdaten liefert, die von einem Onlinedienst stammen.

Versuche, damit deine eigene Wetteranzeige zu programmieren. Oder kombiniere es mit der Karte, um eine Echtzeit-Wetterkarte zu erstellen.

Zusammenfassung der Programmiertechniken

In unserem abschließenden Kapitel haben wir versucht zu zeigen, dass es neben den wenigen Beispielen, die wir in diesem Buch ausführlich besprochen haben, noch viele weitere Richtungen gibt, die du einschlagen kannst.

Je mehr Erfahrungen du sammelst, umso sicherer wirst du und umso besser bist du in der Lage, bei der Programmentwicklung deine Ideen umzusetzen. Als Programmierer liegt eine unendliche Welt kreativer Arbeit vor dir, wobei es keine Rolle spielt, ob du Greenfoot oder irgendeine andere Entwicklungsumgebung einsetzt.

Wenn du in einer anderen Umgebung als Greenfoot programmierst, musst du neue Kenntnisse erwerben und Techniken lernen, aber alles, was du für Greenfoot gelernt hast, ist immer noch nützlich und anwendbar.

Wenn du das Buch bis hierher sorgfältig durchgearbeitet hast, hast du eine ganze Menge über die Java-Programmierung, ja sogar über die Programmierung in objektorientierten Sprachen allgemein gelernt. Programmieren zu lernen ist am Anfang immer am schwersten – und diesen Teil hast du hiermit bereits hinter dir.

Wenn du Unterstützung oder neue Ideen für deine Greenfoot-Programmierung benötigst, nutze die Greenfoot-Website[1]. Hier kannst du deine Szenarien veröffentlichen, einen Blick in die Arbeit anderer Programmierer werfen und dich durch die Ideen anderer inspirieren lassen. Die Video-Tutorials verraten dir viele Tipps und Tricks. Auch ist es sinnvoll, dem Diskussionsforum beizutreten, um sich mit anderen Greenfoot-Programmierern auszutauschen, um Hilfe zu bitten, selbst Hilfe zu geben und neue Ideen zu diskutieren.

Hast du das Gefühl, dass du an die Grenzen von Greenfoot gestoßen bist? Dann ist es vielleicht an der Zeit, einen Blick auf unsere nächste Programmierumgebung zu werfen: BlueJ[2]. Doch das ist eine andere Geschichte, die in einem anderen Buch erzählt wird: Es ist dicker und geht mehr in die Tiefe als dieses und erwartet dich ...

Wir hoffen, dass dir das Programmieren genauso viel Spaß macht wie uns. Wenn ja, liegt eine völlig neue Welt vor dir. Setze deine Ideen in Programme um und lasse deiner Kreativität freien Lauf. Dabei wünschen wir dir viel Vergnügen!

1 *http://www.greenfoot.org*
2 *www.bluej.org*