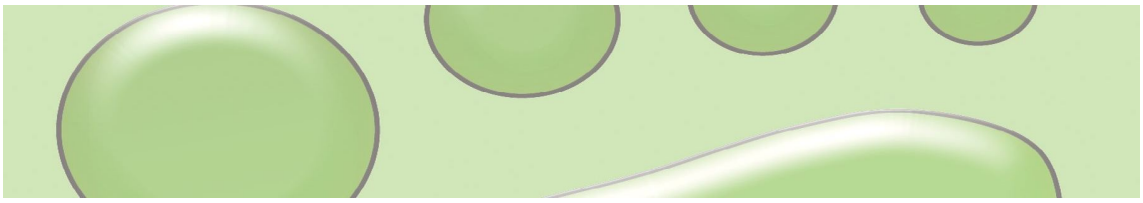

Greenfoot Tutorial

german/deutsch, passend zum Greenfoot System, Version 2.3

Autor: Michael Kölling

Deutsche Übersetzung: Claus Eikemeier
Angepasst von: Nicolas Ruh

KENT
UNIVERSITY OF KENT

A) Die Benutzeroberfläche

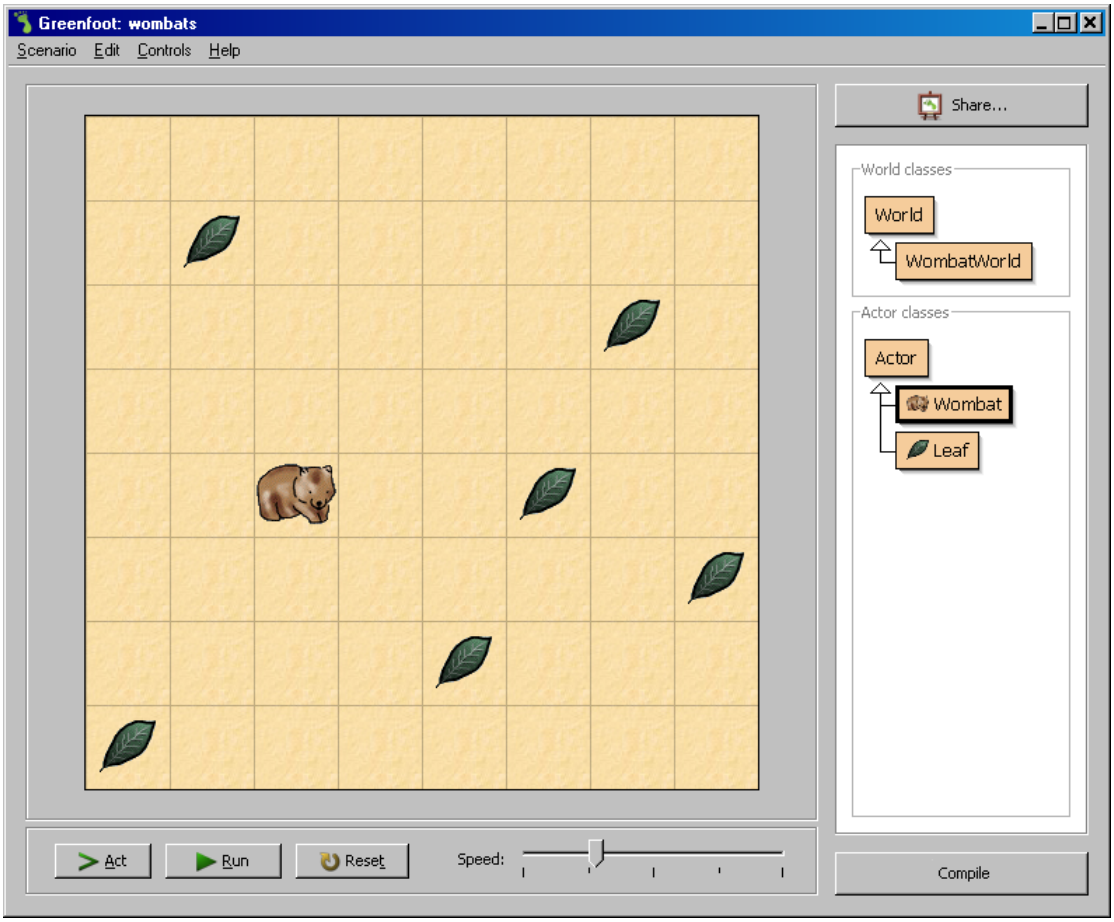

Abbildung 1: Das Wombat-Szenario in Greenfoot

1. *Ein Greenfoot-Projekt öffnen*

In diesem Kurs werden Sie häufig mit vorgegebenen Beispiel-Programmen arbeiten. Das Vorgehen, um ein solches sogenanntes „Szenario" in Greenfoot zu öffnen, ist immer gleich - dementsprechend sollten Sie es sich für zukünftige Lektionen merken:

***Schritt 1**: Das Beispielszenario herunterladen, entpacken und an einem angemessenen Ort speichern.*

Genauer: Sie finden die Beispielszenarien als gezippte Ordner auf dem Wiki, das heutige heisst **wombats.zip**. Diesen Ordner müssen Sie herunterladen und dann entzippen, am einfachsten durch *Rechtsklick → alles entpacken*. Wichtig ist auch, dass die entpackten Dateien dann in Ihrem persönlichen Ordner des Klassenlaufwerks abgelegt werden.

Hinweis: Aufgrund der Einstellungen des Schulnetzwerks gibt es während des Herunterladens u.U. eine Sicherheitswarnung. Ticken sie in diesem Fall „Für alle Dateien.." und bestätigen sie mit „Ja".

***Schritt 2:** Das Beispielszenario in Greenfoot öffnen und kompilieren.*

Weg A: Navigieren in den heruntergeladenen und entpackten Ordner. Sie können das Szenario durch einen Doppelklick auf die Datei „project.greenfoot" öffnen.

Weg B: Starten Sie das Greenfoot-Programm. Gehen Sie zu *Datei → öffnen...* und wählen Sie den heruntergeladenen und entpackten Ordner auf der Festplatte.

Das System sollte nun ähnlich wie in Abb. 1 auf dem Bildschirm aussehen. Sollte der zentrale Bereich grau sein, müssen Sie zuerst noch auf „Compile" rechts unten klicken, damit das Gitternetz der „Welt" erscheint.

Normalerweise sollten zu Beginn noch keine Blätter und keine Wombats in der „Welt" existieren.

Das grosse hinterlegte Gitter, welches den grössten Teil des Fensters überdeckt, stellt „die Welt“ (World) dar. Da wir es in diesem Projektbeispiel mit Wombats zu tun haben (das sind australische Beuteltiere), sieht man hier die „WombatWelt“ (WombatWorld).

Rechts vom Fenster sieht man die Klassen-Anzeige. Hier werden alle Java-Klassen dargestellt, die am Projekt beteiligt sind. Die Klassen „World“ (Welt) und „Actor“ (Akteur) sind in jedem Projekt vorhanden – sie werden automatisch vom Greenfoot-System angelegt und angezeigt. Alle anderen (Unter-)Klassen gehören speziell zum Wombat-Projekt, und sind daher in anderen Projekten in der Regel auch anders (andere Anzahl, Struktur und Bezeichnung).

Unter der Welt sind die Steuer-Elemente (Execution Controls) angeordnet: dieses ist der Bereich mit der „Act“-, der „Run“- und der „Reset“-Schaltfläche und dem Schieberegler („Speed“/ Geschwindigkeit).

2. Objekte in der Welt platzieren

Nun werden Objekte in der Welt platziert: Machen Sie einen Rechts-Klick auf die Wombat-Klasse in der Klassenanzeige auf der rechten Seite. Ein Pop-Up-Menü erscheint. Aus diesem wählt man „new Wombat()“ aus und kann dann irgendwo in die „Welt“ klicken. Dieser Vorgang hat ein neues Objekt (eine Instanz) vom Typ „Wombat“ erzeugt und dieses dann in der Welt platziert.

Wombats essen Blätter (Leaf), daher kann man jetzt auch ein paar Blätter in gleicher Weise in der Welt platzieren: Ein Rechts-Klick auf die „Leaf“-Klasse erzeugt ein neues Objekt und ein (Links-)Klick in die Welt legt das Objekt an der Stelle ab. Hält man während des Klickens die Umschalt-Taste (SHIFT) gedrückt, kann man mehrere Objekte zu ablegen.

3. Objekte „zum Leben erwecken“

Klicken Sie nun auf die „Act“-Schaltfläche in der Steuer-Elemente-Leiste. Sie sehen, dass einige Objekte agieren. Dies bedeutet: jedes Objekt macht das, was ihm als Aktion vorgegeben wurde.

Was genau von einem Objekt gemacht wird, hängt davon ab, wie das Objekt definiert wurde – dazu kommen wir später. In unserem Beispiel ist den Blättern (Klasse „Leaf“) nichts als Aktion zugeordnet, Objekte der Klasse „Wombat“ bewegen sich im Gegensatz dazu in der „Welt“. Falls noch nicht gemacht, platzieren Sie jetzt ein paar Wombats, d.h. Objekte der Klasse „Wombat“ auf der Welt und drücken Sie die „Act“-Schaltfläche noch einmal. Alle Wombat-Objekte bewegen sich.

Wombats essen gerne Blätter. Wenn also ein Blatt auf ihrem Weg liegt, werden sie es essen. Beobachten Sie dieses Verhalten. (Eventuell müssten Sie noch ein Blatt vor einem Wombat positionieren, damit Sie sehen können, wie es wieder verschwindet.)

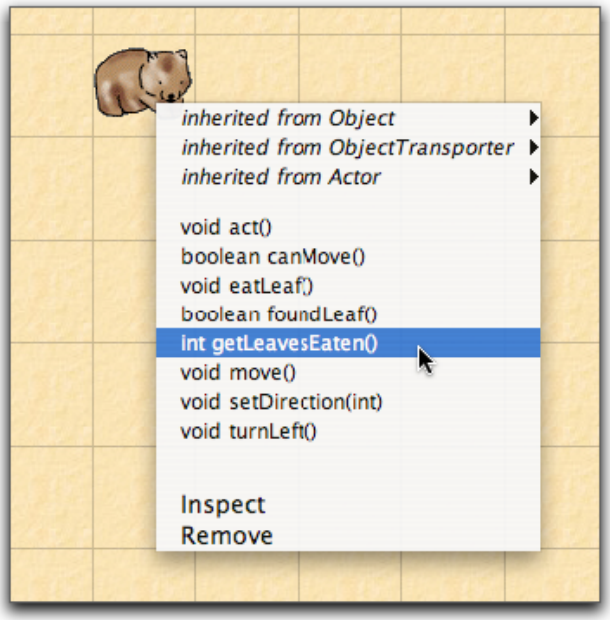

Abbildung 2: Kontextmenue des Wombats.

4. Ein Szenario ablaufen lassen

Klicken Sie auf die „Run“-Schaltfläche. Das ist äquivalent zum wiederholten schnellen Anklicken der „Act“-Schaltfläche. Man stellt fest, dass die Bezeichnung der „Run“-Schaltfläche sich zu „Pause“ verändert. Anklicken der „Pause“-Schaltfläche stoppt den Ablauf wieder.

Der Schieberegler neben den Schaltflächen „Act“ und „Run“ steuert die Geschwindigkeit der Animation. Wenn man die Animation mittels der „Run“-Schaltfläche startet und dann den Schieberegler verändert, sieht man die Auswirkung sofort.

5. Objekt-Methoden direkt aufrufen

Anstatt das gesamte Szenario ablaufen zu lassen, kann man auch einzelne Objekt-Methoden aufrufen. Eine Methode ist eine einzelne Aktion, die ein Objekt ausführen kann:

Stoppen Sie die Animation, falls sie noch läuft. Dann klicken Sie mit der rechten Maustaste auf den Wombat. Man sieht, dass jedes Objekt der „Wombat-Welt“ ein eigenes Kontextmenü besitzt. (Abb. 2).

Nun können Sie eine der angegebenen Methoden im Kontext-Menü anwählen und das Objekt führt die zugehörige Aktion aus. Wählen Sie zum Beispiel turnLeft() durch Anklicken aus. Diese Methode lässt den Wombat eine Linksdrehung ausführen. Versuchen Sie nun die Methode move().

Einige Methoden geben Rückgabewerte aus. Probieren Sie zum Beispiel die Methode getLeavesEaten(). Diese Methode gibt aus, wie viele Blätter der Wombat bis jetzt gefressen hat.

Abbildung 3: Das Kontext-Menü von Wombat

Vielleicht haben Sie bemerkt, dass auch eine Methode act() vorhanden ist. Diese Methode wird jedes Mal aufgerufen, wenn man auf die „Act“-Schaltfläche klickt. Wenn man also die act()-Methode nur eines Objektes aufrufen möchte, kann man das durch direkten Aufruf im Kontext-Menü des jeweiligen Objektes tun.

6. Eine neue Welt erzeugen

Falls man mehrere Objekte in der Welt platziert hat und diese Objekte nicht mehr haben möchte, d.h. man von ganz vorne anfangen möchte, gibt es einen einfachen Weg: man kann die ganze Welt „aufgeben“ und einfach eine neue erzeugen. Dieses wird in der Regel durch Anklicken der „Reset“-Schaltfläche (unten) erreicht. Die alte Welt wird automatisch verworfen (und damit auch alle Objekte, die darin waren), und eine neue Welt wird erstellt.

7. Eine Methode der Welt aufrufen

Wir haben eben gesehen, dass die Objekte in der Welt eigene Methoden besitzen, welche über das jeweilige Kontextmenü aufgerufen werden können. Die Welt selber ist auch ein Objekt mit eigenen Methoden, die Sie auch direkt aufrufen können – dazu genügt ein Rechts-Klick irgendwo in der Welt, wo sich kein anderes Objekt befindet.

Eine der Methoden in diesem Menü ist populate(). Probieren Sie die Methode einfach mal aus. Es ist eine Methode, die mehrere Blatt- und mehrere Wombat-Objekte erzeugt und sie auf der Fläche, sprich in der Wombat-Welt, platziert. Man kann nun die Animation wieder mittels der „Run“-Schaltfläche starten.

Eine andere Methode des World-Objektes ist ‘randomLeaves(int howMany)’. Diese Methode setzt einige Blätter an zufällig gewählten Positionen auf die Welt. Man beachte, dass der Methodenaufruf einen sog. Parameter („howMany“ / wie viele) vom Typ int (= Ganze Zahl / Integer) besitzt. Dieses bedeutet, dass man zusätzliche Informationen angeben muss, wenn man die Methode aufrufen will. Der „Name“ des Parameters – hier „howMany“ („wie viele“) - deutet an, dass man eingeben muss, wie viele Blätter in der Welt positioniert werden sollen – probieren Sie es aus.

Hinweis: Eventuell stellen Sie fest, dass die Anzahl der erschaffenen Blätter nicht genau mit der eingegebenen übereinstimmt. Das kommt daher, dass einige Blätter an gleichem Ort abgelegt wurden und sich somit überdecken.

Gut, jetzt haben Sie vielleicht genug davon, den Wombats zuzuschauen, wie sie letztlich einfach im Kreis laufen (und ggf. Blätter fressen). Wir kommen jetzt zu wirklich interessanteren Dingen: der Programmierung von interessantem Verhalten für unsere eigenen Wombats.

B) Programmieren

8. Verändern des Verhaltens von Objekten

Man kann eigene Objekte – Wombats, oder irgendetwas anderes, was man möchte – einfach durch das Schreiben von etwas Java Quellcode für die Klasse dieses Objektes programmieren. Man kann aber auch bereits bestehenden Objekte der Beispiel-Szenarios verändern, und damit wollen wir anfangen.

Klicken Sie die Wombat-Klasse in der Klassenanzeige doppelt an. Ein Texteditor erscheint und man sieht den Quellcode der Wombat-Klasse. Dieser Code ist geschrieben in einer Programmiersprache namens Java, hier muss der Programmierer (also Sie) in einer für den Computer verständlichen Weise ausdrücken, wie das Programm funktionieren soll – also wie sich das Wombat verhalten soll.

Verändern wir also das Verhalten des Wombats. Wir wollen, dass er nicht immer nach links dreht wenn er nicht weiter kann - statt dessen soll er in eine zufällig ausgewählte Richtung weiter laufen Um dies zu erreichen, fügen wir zu den Methoden der Wombat-Klasse eine Methode hinzu, die wir „turnRandom()“ (drehe in zufällig gewählte Richtung) nennen. Öffnen Sie dafür die Wombat-Klasse im Editor und fügen Sie den folgenden Code an einer geeigneten Stelle ein – wenn Sie die grobe Struktur des neuen Codes mit dem bereits existierenden vergleichen, sollten Sie sehen können, wo der neue Code hin passt:

```
/**
* Turn in a random direction.
*/
public void turnRandom() { // get a random number between 0 and 3...
    int turns = Greenfoot.getRandomNumber(4); // ...an turn left that many times.
    for(int i=0; i<turns; i++) {
          turnLeft();
    }
}
```

Hinweis: Beim Kopieren gehen evtl. die Zeilenumbrüche verloren. Diese müssen Sie ggf. von Hand wieder einfügen. Sorgen Sie dabei bitte auch dafür, dass der Code korrekt eingerückt ist und damit übersichtlich bleibt – unter „Edit“ gibt es dafür übrigens eine Auto-layout Funktion (Shortcut: SHIFT+CTRL+I). Da die Einrückung von den geschweiften Klammern bestimmt wird, funktioniert das automatische Layout nur dann richtig, wenn die geschweiften Klammern korrekt gesetzt sind.

Jetzt verändern wir noch die Methode „act()“, welche die neu erstellte Methode turnRandom() nun verwenden soll. Die act() Methode sieht derzeit so aus:

```
public void act() {
    if(foundLeaf()) {
          eatLeaf();
    }
    else if(canMove()) {
          move();
    }
    else {
          turnLeft();
    }
}
```

Ersetzen Sie den Aufruf von „turnLeft()“ am Ende durch den Aufruf der neuen Methode „turnRandom()“.

Bevor Sie es ausprobieren können, braucht es allerdings noch einen weiteren Schritt: Der Computer versteht den soeben geschriebenen Code nicht direkt, er muss zuerst in eine spezielle „Maschinensprache“ übersetzt

werden (das ist bei den meisten Programmiersprachen so). Diesen Übersetzungsprozess nennt man „Kompilierung“, wenn der Code noch nicht kompiliert wurde, sind die entsprechenden Klassen in der Greenfoot-Oberfläche mit einer blauen Schraffur hinterlegt. Klicken Sie also unten rechts auf die Schaltfläche „Compile“ um den Code zu übersetzen, dann sollte die Schraffur verschwinden. Falls Fehler angezeigt werden, beheben Sie diese (achten Sie auf die Fehlermeldung) und übersetzen Sie das Programm noch einmal. Wiederholen Sie dies so lange, bis sich die Klasse ohne Fehler übersetzen lässt. Nach der erfolgreichen Kompilierung wird automatisch eine neue Welt erzeugt, die Sie jetzt ausprobieren können.

9. Bilder / Icons verändern

Es gibt zwei verschiedene Arten, wie man die Bilder von Objekten verändern kann: Man kann das Bild einer Klasse verändern und somit das normale Aussehen der Bilder für alle Objekte der Klasse. Oder ein Objekt kann per Code das eigene Bild verändern. Dieses verändert dann das Bild von nur diesem einen Objekt. Jedes Objekt kann sein Bild verändern, so oft es möchte.

Um das Bild für eine ganze Klasse zu setzen, wählen Sie „Set Image...“ aus dem Kontextmenü der Klasse aus. Sie können dieses mit der Klasse „Leaf“ (Blatt) ausprobieren – verändern Sie das Bild z.B. in eine Banane, und der Wombat wird die Bananen essen (so, wie er vorher die Blätter gefressen hat). Greenfoot hat eine Sammlung von Bildern eingebaut, die hier genutzt werden können. Sie können aber auch eigene Bilder in den „images“-Ordner im aktuellen Projekt ablegen (hier: „wombats“) und dann verwenden.

Die zweite Möglichkeit besteht darin, dass das Objekt selbst das Bild durch Ausführen eines Programmstückes tauscht. Dazu muss die Methode 'setImage()', die das Objekt von der Klasse 'Actor' erbt, aufgerufen werden. Man könnte beispielsweise die Zeile

```
setImage("wombat-left.gif");
```

benutzen, um das Bild eines nach links schauenden Wombats für ein Objekt festzulegen.

10. Weiteres zu den Klassen des Greenfoot-Systems

Um Veränderungen am Verhalten der Objekte zu machen, muss man häufig auf einige Standardklassen des Greenfoot-Systems zugreifen. Das Greenfoot-System hat vier wichtige Grundklassen, die Sie kennen sollten:

1. die Welt ('World'),
2. den Akteur ('Actor'),
3. das Bild ('GreenfootImage'),
4. und das System selbst ('Greenfoot').

Prägen Sie sich hier am besten die englischen Begriffe ein, da wir sie im Text und natürlich in den Programmstücken verwenden werden.

Die ersten beiden Klassen, d.h. 'World' und 'Actor', haben Sie bereits in der graphischen Benutzeroberfläche gesehen. Sie bilden die Oberklassen für die Welt und die darin enthaltenen Objekte. Die Klasse 'GreenfootImage' wird verwendet, wenn man mit Bildern arbeitet. Die Klasse 'Greenfoot' bietet Zugang zu den Interna des Greenfoot-Systems. Der einfachste Weg, diese Klassen kennen zu lernen, ist ein Blick in die Dokumentation/API (Anwendungs-Programmier-Schnittstelle) von Greenfoot:

- http://www.greenfoot.org/doc/javadoc/ - online, englisch, aktuellste Version
- Rechts-Klick auf Actor oder World – offline, englisch, passend zur installierten Version
- http://www.greenfoot-center.de/fuer-programmierer/klassendokumentation.html - online, deutsch, Version 2.0 (d.h. hier fehlen einige wenige Methoden, die es erst seit der Version 2.1 gibt)

Klassendokumentationen sind sehr wichtig, denn sie sagen dem Programmierer, was die Bausteine (eben, die Klassen) können, die er zur Verfügung hat. Greenfoot stellt die oben beschriebenen Klassen zur Verfügung, und Sie werden in Zukunft viel mit diesen Klassen und deren Methoden arbeiten.

Natürlich ist das Programmieren in Greenfoot aber nicht auf diese Klassen beschränkt, es stehen ihnen sämtliche Klassen der Programmiersprache Java zur Verfügung. Zwar werden wir noch eine Weile ohne diesen grossen und allgemeinen Werkzeugkasten auskommen, aber Sie können ja mal einen kurzen Blick darauf werfen und sich überzeugen, dass diese Bausteine sehr ähnlich aussehen (und benutzt werden) wie die Greenfoot-spezifischen Klassen:

- http://download.oracle.com/javase/6/docs/api/ Java-API von Oracle, englisch

11. Dokumentation eigener Klassen

Auch für selbst programmierte Klassen ist eine Dokumentation verfügbar – vorausgesetzt man versieht die Methoden und Eigenschaften/Attribute mit entsprechenden Kommentaren. Kommentare werden im Editor in grau (nach //) oder dunkelblau (zwischen /* und */) dargestellt, sie dienen allein dem Verständnis des Programmierers und werden vom Computer nicht ausgeführt. Manche Kommentare werden allerdings dazu benutzt, die Dokumentation einer Klasse automatisch zu generieren.

Im Greenfoot-Editor gibt es rechts oben eine Auswahlliste, mit der man zwischen dem Quellcode und der automatisch generierten Dokumentation einer Klasse wechseln kann. Schauen Sie sich das in der Wombat-Klasse mal an – sehen Sie, aus welchen Informationen die Dokumentation generiert wird?

12. Eine neue Klasse erstellen

Nun ist es Zeit, das Leben unserer Wombats etwas aufregender zu gestalten: wir werden ein paar Hindernisse einführen, nämlich Steine (’rocks’), und das Programm dann so abändern, dass die Wombats um die Steine herumlaufen müssen.

Als erstes müssen wir dafür eine neue 'Actor'-Klasse erstellen. Dazu wählt man den Menüpunkt 'New subclass' (Neue Unterklasse) aus dem Kontextmenü der Klasse 'Actor'. Wenn der Name gefragt wird, geben Sie 'Rock' (=Stein) ein (Klassennamen schreibt man GROSS!). Zusätzlich muss man auch noch ein Bild für die Klasse auswählen, damit sie in der „Welt“ auch sichtbar ist.

Man kann Bilder aus dem Internet herunterladen oder in einem Graphikprogramm erstellen und dann im Unterverzeichnis 'images' (= Bilder) ablegen. Wenn man das vor der Erstellung der Klasse durchführt, wird das Bild auch in diesem Dialog mit angeboten. Für dieses Beispiel haben wir diese Arbeit schon gemacht und ein Bild eines Steines im entsprechenden Verzeichnis abgelegt. Das Bild heisst 'rock.gif'. Damit sollte das Problem für dieses Beispiel erledigt sein: Wählen Sie dieses Bild aus, klicken Sie auf die 'OK'-Schaltfläche und schon wird eine neue Klasse mit Namen 'Rock' erzeugt.

Nun öffnen Sie den Editor für diese Klasse. Sie sehen, dass das Programm-Skelett schon automatisch erzeugt wurde. Tatsächlich müssen wir an dieser Stelle für die Klasse 'Rock' überhaupt keinen Programmcode schreiben, da Steine kein besonderes Verhalten haben, sondern lediglich im Weg liegen.

Schliessen Sie daher den Editor, kompilieren und testen Sie das ganze System: erzeugen Sie zusätzlich zu Wombats und Leaves auch einen Stein und platzieren Sie ihn in der Welt – am besten natürlich so, dass ein Wombat darüber stolpern wird. Klicken Sie dann auf „Run“ – funktioniert alles wie gewünscht?

Sie werden feststellen, dass sich die Wombats derzeit noch nicht um die Steine kümmern: sie laufen einfach über die Steine hinweg, oder besser: durch sie hindurch. Naja, es wäre auch zu schön, wenn es tatsächlich so einfach gewesen wäre. Wir müssen natürlich noch etwas tun, damit die Wombats die Steine erkennen und vor ihnen die Richtung ändern:

Öffnen Sie den Editor mit der 'Wombat'-Klasse und suchen Sie die 'canMove' (= kann sich bewegen) Methode. Derzeit prüft diese Methode einzig und alleine, ob der Wombat am „Rand“ der Welt steht, d.h. ob er beim nächsten Schritt „von der Welt fallen“ würde. Wir müssen diese Methode so verändern, dass sie auch dann den Wert 'false' (=falsch) zurückgibt, wenn ein Stein vor dem Wombat liegt.

Dazu müssen einfach wir die letzte 'return'-Anweisung verändern. Sie lautet derzeit so:

```
return true;
```

und sollte nun ersetzt werden durch das folgende „intelligente“ Programmstück:

```
List rocks = myWorld.getObjectsAt(x, y, Rock.class);
if(rocks.isEmpty()) {  //es liegt kein Stein im Weg
    return true;
}
else {  //es liegt also ein Stein vor dem Wombat
    return false;
}
```

Dieses Programmstück holt eine Liste aller Stein-Objekte in der Zelle vor uns (man kann mehrere Objekte auf einem Feld ablegen). Falls diese Liste leer ist, kann das Wombat sich bewegen, ansonsten nicht.

Das Beispielszenario 'wombats2' enthält alle vorgestellten Änderungen und sollte somit mit dem von Ihnen „verbesserten“ 'wombat'-Projekt übereinstimmen. Sie können auch dort nachschauen, welche Veränderungen an den Programmstücken durchgeführt wurden.

13. Und weiter?

Die wichtigsten Grundlagen des Programmierens in Greenfoot haben Sie nun kennen gelernt – ab jetzt gibt es keine Schritt-für-Schritt Anleitungen mehr, sondern nur noch Problemstellungen, an deren Lösung Sie sich versuchen können.

Beispiele für schon etwas fortgeschrittenere Problemstellung im Kontext des Wombat-Szenatios wären die folgenden:

1. Verändern Sie das Verhalten des Wombats so, dass er ab und zu (z.B. in 30% der Fälle) in eine zufällige Richtung abbiegt, obwohl er auch geradeaus hätte laufen können.
2. Programmieren Sie das Verhalten eines Wombats so, dass *alle Felder nacheinander besucht* werden, und somit alle Blätter garantiert gefressen werden.
   Für die erste Variante können Sie den Wombat an einem *geeigneten Startpunkt* platzieren.
3. (Erweiterung von 2) Finden Sie auch ein Programm, das *von allen Startpunkten aus* funktioniert?
4. (Erweiterung von 2 oder 3) Geht das auch in einer Welt *mit Steinen*?

14. Zusatzaufgabe (schwierig!)

Falls Sie bisher keinerlei Probleme hatten, vielleicht weil Sie schon ein wenig Vorwissen zum Programmieren mit Java mitbringen, hier noch eine echte Knacknuss – auch wenn es nicht so scheint.

Voraussetzung: Eine Welt kann Futter und/oder Hindernisse enthalten, dazu mindesten 2 Wombats, die sich manchmal in eine zufällige Richtung drehen (s. Aufgabe 1 oben). Man weiss nicht, wo die Wombats und die anderen Akteure sich zu Beginn befinden – z.B. weil sie mit der Methode populate() erzeugt wurden.

Aufgabe: Programmieren Sie die Wombats so, dass sich nie zwei Wombats auf demselben Feld aufhalten können. Das ist nicht einfach, weil man nicht weiss (und auch nicht herausfinden kann), welches der Wombats zuerst agiert.

Hinweis: Wenn (oder sollte ich sagen: falls?) Sie eine allgemeine Lösung für diese Aufgabe finden, dann haben Sie höchstwahrscheinlich verstanden, wie ein Betriebssystem verhindert, dass zwei Prozesse (~Programme) gleichzeitig auf die CPU eines Computers zugreifen können.