

**Dietrich Boles, Daniel Jasper**

# **Hamster-Simulator**

**Version 2.6.1**

## **Benutzungshandbuch**

**04.05.2009**

## Inhaltsverzeichnis

1	Einleitung.....	6
1.1	Das Hamster-Modell.....	6
1.2	Der Hamster-Simulator.....	7
1.3	Änderungen in Version 2.6 gegenüber 2.5.....	7
1.4	Änderungen in Version 2.5 gegenüber 2.4.....	8
1.5	Änderungen in Version 2.4 gegenüber 2.3.....	8
1.6	Änderungen in Version 2.3 gegenüber 2.2.....	9
1.7	Änderungen in Version 2.2 gegenüber 2.1.....	9
1.7.1	Erstellen von Hamster-Programmen unabhängig vom Editor des Simulators 10	
1.7.2	Hamstern mit BlueJ.....	10
1.8	Änderungen in Version 2.1 gegenüber 2.0.....	11
1.9	Anmerkungen zur alten Version 1 des Hamster-Simulators.....	12
2	Installation und Starten des Hamster-Simulators.....	14
2.1	Laden und Installation einer Java-Laufzeitumgebung.....	14
2.2	Laden und Installation des Hamster-Simulators.....	14
2.3	Hamster-Simulator unter JDK 1.4 und JDK 1.5.....	15
2.4	Starten des Hamster-Simulators.....	15
3	Ihr erstes Hamster-Programm.....	17
3.1	Gestaltung eines Hamster-Territoriums.....	18
3.2	Eingeben eines Hamster-Programms.....	20
3.3	Compilieren eines Hamster-Programms.....	21
3.4	Ausführen eines Hamster-Programms.....	22
3.5	Debuggen eines Hamster-Programms.....	23
3.6	Zusammenfassung.....	24
4	Bedienung des Hamster-Simulators.....	25
4.1	Grundfunktionen.....	26
4.1.1	Anklicken.....	26
4.1.2	Tooltipps.....	27
4.1.3	Button.....	27
4.1.4	Menü.....	27
4.1.5	Toolbar.....	28
4.1.6	Popup-Menü.....	28
4.1.7	Eingabefeld.....	29
4.1.8	Dialogbox.....	29

4.1.9	Dateiauswahl-Dialogbox .....	30
4.1.10	Dateibaum .....	31
4.2	Verwalten und Editieren von Hamster-Programmen .....	31
4.2.1	Schreiben eines neuen Hamster-Programms .....	33
4.2.2	Ändern eines existierenden Hamster-Programms .....	34
4.2.3	Löschen eines existierenden Hamster-Programms .....	35
4.2.4	Umbenennen eines existierenden Hamster-Programms .....	35
4.2.5	Verschieben eines existierenden Hamster-Programms in einen anderen Ordner	35
4.2.6	Kopieren eines existierenden Hamster-Programms in einen anderen Ordner	35
4.2.7	Drucken eines Hamster-Programms.....	36
4.2.8	Schließen eines geöffneten Hamster-Programms .....	36
4.2.9	Editier-Funktionen.....	36
4.2.10	Verwaltung von Ordnern.....	37
4.3	Compilieren von Hamster-Programmen .....	37
4.3.1	Compilieren.....	38
4.3.2	Beseitigen von Fehlern .....	38
4.3.3	Setzen des CLASSPATH.....	39
4.4	Verwalten und Gestalten von Hamster-Territorien.....	39
4.4.1	Verändern der Größe des Hamster-Territoriums .....	40
4.4.2	Platzieren des Standard-Hamsters im Hamster-Territorium .....	40
4.4.3	Setzen der Blickrichtung des Standard-Hamsters.....	40
4.4.4	Abfragen und Festlegen der Körneranzahl im Maul des Standard-Hamsters	40
4.4.5	Platzieren von Körnern auf Kacheln des Hamster-Territorium .....	40
4.4.6	Platzieren von Mauern auf Kacheln des Hamster-Territorium .....	41
4.4.7	Löschen von Kacheln des Hamster-Territorium .....	41
4.4.8	Abspeichern eines Hamster-Territoriums.....	42
4.4.9	Wiederherstellen eines abgespeicherten Hamster-Territoriums .....	42
4.4.10	Umbenennen eines abgespeicherten Hamster-Territoriums .....	42
4.4.11	Löschen und Verschieben einer Datei mit einem Hamster-Territorium in einen anderen Ordner .....	42
4.4.12	Verändern der Größendarstellung des Hamster-Territoriums.....	43
4.5	Ausführen von Hamster-Programmen .....	43
4.5.1	Starten eines Hamster-Programms.....	43

4.5.2	Stoppen eines Hamster-Programms.....	44
4.5.3	Pausieren eines Hamster-Programms .....	44
4.5.4	Während der Ausführung eines Hamster-Programms .....	44
4.5.5	Einstellen der Geschwindigkeit .....	45
4.5.6	Wiederherstellen eines Hamster-Territoriums.....	45
4.5.7	Mögliche Fehlerquellen.....	45
4.6	Debuggen von Hamster-Programmen.....	46
4.6.1	Aktivieren bzw. deaktivieren des Debuggers .....	46
4.6.2	Beobachten der Programmausführung .....	47
4.6.3	Schrittweise Programmausführung .....	48
4.7	3D-Simulationsfenster und Sound .....	48
4.7.1	Steuerung mittels der Toolbar.....	49
4.7.2	Steuerung mittels der Maus .....	50
4.8	Dateiverwaltung auf Betriebssystemebene .....	50
5	Properties .....	51
5.1	Vorhandene Properties.....	51
5.1.1	security .....	51
5.1.2	workspace.....	51
5.1.3	logfolder .....	52
5.1.4	scheme .....	52
5.1.5	runlocally.....	53
5.1.6	language .....	53
5.1.7	indent .....	53
5.1.8	color .....	54
5.1.9	3D .....	54
5.1.10	lego.....	55
5.1.11	prolog.....	55
5.2	Mehrbenutzerfähigkeit .....	55
6	Englischsprachiger Hamster.....	56
7	Scheme .....	58
7.1	Funktionale Programmiersprachen.....	58
7.2	Die Programmiersprache Scheme.....	59
7.3	Scheme-Hamster-Programme.....	59
7.4	Grundlagen und Befehle.....	60
7.4.1	Territoriumsliste .....	60

7.4.3	Scheme-Hamster-Programme .....	62
7.5	Beispiele.....	63
7.6	Scheme-Konsole .....	64
7.7	Implementierungshinweise .....	65
8	Prolog .....	66
8.0	Voraussetzungen.....	66
8.1	Logikbasierte Programmierung .....	66
8.2	Die Programmiersprache Prolog .....	67
8.2.1	Syntax von Prolog.....	68
8.2.2	Operationale Semantik .....	72
8.2.3	Systemprädikate .....	74
8.2.4	Trace/Boxenmodell .....	78
8.2.5	Kontrollfluss .....	79
8.2.6	Informationen zu Prolog im WWW.....	80
8.3	Prolog-Hamster-Modell.....	81
8.4	Prolog-Hamster-Programme .....	84
8.5	Prolog-Konsole .....	84
8.6	Beispiele.....	85
9	Noch Fragen?.....	87

# 1 Einleitung

Programmieranfänger haben häufig Schwierigkeiten damit, dass sie beim Programmieren ihre normale Gedankenwelt verlassen und in eher technisch-orientierten Kategorien denken müssen, die ihnen von den Programmiersprachen vorgegeben werden. Gerade am Anfang strömen oft so viele inhaltliche und methodische Neuigkeiten auf sie ein, dass sie das Wesentliche der Programmierung, nämlich das Lösen von Problemen, aus den Augen verlieren.

## 1.1 Das Hamster-Modell

Das Hamster-Modell ist mit dem Ziel entwickelt worden, dieses Problem zu lösen. Mit dem Hamster-Modell wird Programmieranfängern ein einfaches, aber mächtiges Modell zur Verfügung gestellt, mit dessen Hilfe Grundkonzepte der imperativen und objektorientierten Programmierung auf spielerische Art und Weise erlernt werden können. Programmierer entwickeln so genannte Hamster-Programme, mit denen sie virtuelle Hamster durch eine virtuelle Landschaft steuern und bestimmte Aufgaben lösen lassen. Die Anzahl der gleichzeitig zu berücksichtigenden Konzepte wird im Hamster-Modell stark eingeschränkt und nach und nach erweitert.

Prinzipiell ist das Hamster-Modell programmiersprachenunabhängig. Zum praktischen Umgang mit dem Modell wurde jedoch bewusst die Programmiersprache Java als Grundlage gewählt. Java – auch als „Sprache des Internet“ bezeichnet – ist eine moderne Programmiersprache, die sich in den letzten Jahren sowohl im Ausbildungsbereich als auch im industriellen Umfeld durchgesetzt hat.

Zum Hamster-Modell existieren drei Bücher. In dem ersten Buch „Programmieren spielend gelernt mit dem Java-Hamster-Modell“ werden allgemeine Grundlagen der Programmierung erläutert sowie Konzepte der imperativen Programmierung (Anweisungen, Schleifen, Prozeduren, Typen, Variablen, Parameter, Rekursion, ...) eingeführt. Darauf aufbauend behandelt das zweite Buch „Objektorientierte Programmierung spielend gelernt mit dem Java-Hamster-Modell“ alle wichtigen Konzepte der objektorientierten Programmierung (Objekte, Klassen, Vererbung, Polymorphie, Interfaces, Exceptions, Zugriffsrechte, Pakete, ...). Im dritten Band „Parallele Programmierung spielend gelernt mit dem Java-Hamster-Modell“ geht es um die parallele Programmierung mit Java-Threads (Threads, Kommunikation, Synchronisation, Deadlocks, ...).

Alle drei Bücher sind insbesondere für Schüler und Studierende ohne Programmiererfahrung empfehlenswert. Die Bücher sind als Grundlage für Programmierkurse sowie zum Selbststudium geeignet. Dazu enthalten sie viele Beispielprogramme und Übungsaufgaben.

Ergänzende Materialien werden im Internet unter [www.java-hamster-modell.de](http://www.java-hamster-modell.de) bereitgestellt.

## 1.2 Der Hamster-Simulator

Auf der Website **[www.java-hamster-modell.de](http://www.java-hamster-modell.de)** steht auch der „Hamster-Simulator“ kostenlos zur Verfügung; ein Programm, mit dem Hamster-Programme erstellt und ausgeführt werden können. Neben den drei Büchern kommt dem Hamster-Simulator dabei eine ganz wichtige Bedeutung zu, denn Programmieren lernt man nicht durch Lesen. Man muss üben, üben, üben. Und genau dazu dient der Simulator.

Sie halten hier in Händen bzw. sehen am Bildschirm das Benutzungshandbuch des Hamster-Simulators. Im Prinzip ist dieser sehr einfach zu bedienen. Wenn Sie bereits etwas Erfahrung mit Computern haben, sollte Ihnen seine Handhabung keine Probleme bereiten. Trotzdem gibt es sicher Situationen, in denen Sie nicht weiterkommen oder in denen Ihnen nicht bewusst ist, welche Möglichkeiten Sie überhaupt haben. Für genau diese Fälle ist dieses Benutzungshandbuch konzipiert. Nach dieser Einleitung wird im zweiten Abschnitt erläutert, wie Sie den Simulator auf Ihrem Rechner installieren und starten. Im dritten Abschnitt wird kurz und knapp erklärt, was Sie tun müssen, um Ihr erstes Hamster-Programm zu schreiben und ausführen zu lassen. Abschnitt 4 behandelt dann im Detail die einzelnen Funktionen des Simulators.

Über Properties können Sie bestimmte Eigenschaften des Hamster-Simulators beeinflussen. Außerdem können Sie über Properties die Eigenschaft der Mehrbenutzer-Fähigkeit des Hamster-Simulators steuern. Mehr dazu in Abschnitt 5.

Seit Version 2.4 ist der Hamster-Simulator auch in die englisch-sprachige Welt integriert worden. Das betrifft nicht nur die Benutzungsoberfläche sondern auch das Modell selbst. Werfen Sie dazu einen Blick in Abschnitt 6.

Standardmäßig werden Sie mit dem Hamster-Simulator Java-Programme entwickeln. Seit Version 2.3 ist es jedoch auch möglich, Programme in der funktionalen Programmiersprache Scheme zu entwickeln und zu testen. Wie Sie dies tun können, erfahren Sie in Abschnitt 7.

## 1.3 Änderungen in Version 2.6 gegenüber 2.5

Wiederum wurden ein paar kleine Fehler korrigiert. Wichtige Änderungen sind:

- Farbliche Umgestaltung des Simulators
- Im Modus `runlocally=true` (siehe Abschnitt 5) gibt es nun eine Console, die Ein- und Ausgaben über `System.in` bzw. `System.out` und `System.err` verarbeitet. Konkret bedeutet das: Enthält ein Hamster-Programm bspw. den Befehl `„System.out.println(hallo);“` und wird das Programm ausgeführt, öffnet sich das Consolen-Fenster und die Zeichenkette „hallo“ wird in das Fenster geschrieben. Im Standard-Modus `runlocally=false` ändert sich nichts: Ausgabeanweisungen werden weiterhin in die Dateien `sysout.txt` bzw. `syserr.txt` geschrieben.
- Es gibt ein neues Menü „Extras“, über das unter anderem die Fontgröße geändert werden kann.

- Die größte Änderung war die Integration der logikbasierten Programmiersprache Prolog in den Hamster-Simulator (siehe Abschnitt 8).

## 1.4 Änderungen in Version 2.5 gegenüber 2.4

Neben der Korrektur einiger kleiner Fehler sind folgende wesentliche Erweiterungen am Hamster-Simulator vorgenommen worden:

- Hamster-Programme können zusätzlich in einer 3D-Welt ausgeführt werden (siehe Abschnitt 4.7). Die 3D-Ansicht funktioniert momentan aber leider nur unter Windows.
- Es gibt ein Property namens „color“, über das die Farbe des Standard-Hamsters angegeben werden kann. Auch die Farbgestaltung neu erzeugter Hamster kann manipuliert werden. Einzelheiten entnehmen Sie bitte Abschnitt 5.
- Es gibt ein Property namens „logfolder“, über das der Ordner, in dem die beiden Dateien „sysout.txt“ und „syserr.txt“ erzeugt werden, geändert werden kann. Einzelheiten entnehmen Sie bitte Abschnitt 5.
- Mitte des Jahres 2008 erscheint im Teubner-Verlag ein drittes Buch zum Java-Hamster-Modell, in dem in die parallele Programmierung mit Java-Threads eingeführt wird. Der Hamster-Simulator ab Version 2.5 erlaubt die Ausführung solcher paralleler Hamster-Programme. Die Beispielprogramme des dritten Bandes sind bereits unter „beispielprogramme“ einseh- und ausführbar.
- Prinzipiell ist es möglich, mit dem Hamster-Simulator einen Lego-Mindstorms-Roboter zu steuern. Da der Roboter jedoch oft noch ein wenig planlos herumläuft, wird die endgültige Freischaltung dieses Features auf die Version 2.6 des Hamster-Simulators verschoben.

## 1.5 Änderungen in Version 2.4 gegenüber 2.3

Neben der Korrektur einiger kleiner Fehler sind zwei wesentliche Erweiterungen am Hamster-Simulator vorgenommen worden:

- Es wurde das Property *runlocally* eingeführt. Standardmäßig ist dies auf *false* gesetzt, so dass sich gegenüber den alten Versionen des Hamster-Simulator nichts ändert: Hamster-Programme werden in einer neu gestarteten JVM ausgeführt. Setzt man das Property jedoch auf *true*, passiert folgendes: Hamster-Programme werden in derselben JVM ausgeführt, wie der Simulator selbst. Hintergrund für diese Erweiterung ist der, dass auf einigen Linux- und Macintosh-Rechnern Fehler beim Starten eines Hamster-Programms auftraten bzw. es nach dem Drücken des Run-Buttons bis zu einer Minute dauerte, bis der Hamster tatsächlich loslief. Die Hamster-Programmierer, bei denen dieses Problem auftritt, müssen also einfach dieses Property auf *false* setzen, womit



sich das Problem gelöst hat. Nachteil: Im Modus `runlocally=true` ist es nicht möglich, den Debugger zu nutzen und die Benutzung des `CLASSPATH` ist ebenfalls nicht möglich. Weitere Infos siehe in Abschnitt 5.

- Der Hamster-Simulator ist nun auch komplett in englischer Sprache nutzbar. Komplett bedeutet, sowohl die Oberfläche des Simulators als auch das Hamster-Modell selbst wurden an die englische Sprache angelehnt. Hierzu muss nur das Property *language* auf den Wert *en* gesetzt werden. Standardmäßig steht der Wert auf *de* (Deutsch). Weitere Infos finden sich in Abschnitt 6.

## 1.6 Änderungen in Version 2.3 gegenüber 2.2

Die Version 2.3 des Hamster-Simulators enthält folgende Änderungen bzw. Erweiterungen gegenüber Version 2.2:

- Es ist nun auch möglich, Hamster-Programme in der funktionalen Programmiersprache Scheme zu schreiben. Genauereres siehe im Abschnitt 7.
- Die Properties wurden erweitert (siehe Abschnitt 5).
- Der Hamster-Simulator ist nun Mehrbenutzer-fähig, d.h. er kann einmal auf einem Server installiert und dann von mehreren Nutzern gleichzeitig genutzt werden, wobei die Programme der Nutzer in unterschiedlichen Verzeichnissen abgespeichert werden können (siehe Abschnitt 5.2).
- In der oberen Menüleiste des Editor-Fensters gibt es ein neues Menü „Fenster“. Über dieses Menü ist es möglich, das Simulation-Fenster sowie die Scheme-Konsole sichtbar bzw. unsichtbar zu machen.
- Einige kleine Fehler wurden beseitigt.

## 1.7 Änderungen in Version 2.2 gegenüber 2.1

An der generellen Funktionalität des Hamster-Simulators wurde nichts geändert. Die Änderungen beziehen sich nur auf den internen Programmcode. Was jedoch mit dieser neuen Version möglich ist, sind zwei Dinge:

- Hamster-Programme lassen sich nun auch ohne den Editor des Hamster-Simulators erstellen und ausführen.
- Der Hamster-Simulator wurde so angepasst, dass eine Integration in die Entwicklungsumgebung BlueJ möglich ist.

### 1.7.1 Erstellen von Hamster-Programmen unabhängig vom Editor des Simulators

Ab dieser Version des Hamster-Simulators können Sie Hamster-Programme unabhängig vom Hamster-Editor erzeugen und ausführen. Dazu müssen Sie folgendermaßen vorgehen:

- Erstellen Sie ein Hamster-Programm mit einem beliebigen Editor. Speichern Sie dies in einer Datei ab (bspw. test/sammler.ham) Dabei ist folgendes zu beachten: Wenn die Datei ein objektorientiertes Hamster-Programm enthält, muss sie mit folgendem Kommentar beginnen: `/*object-oriented program*/` Wenn die Datei eine (Hamster-)Klasse enthält, muss sie mit folgendem Kommentar beginnen: `/*class*/` Wenn die Datei ein imperatives Hamster-Programm enthält, ist nichts weiter zu beachten.
- Aus der .ham-Datei muss zunächst eine gültige .java-Datei erzeugt werden. Das geht durch folgenden Aufruf:  

```
java -classpath hamstersimulator.jar;tools.jar de.hamster.ham2java <ham-Datei>.
```

  
Im konkreten Beispiel:  

```
java -classpath hamstersimulator.jar;tools.jar de.hamster.ham2java test/sammler.ham
```
- Die erzeugte .java-Datei muss kompiliert werden. Das geht durch folgenden Aufruf:  

```
javac -classpath hamstersimulator.jar;tools.jar <java-Datei>
```

  
Im konkreten Beispiel:  

```
javac -classpath hamstersimulator.jar;tools.jar test/sammler.java
```

  
Insofern das Programm keine Fehler enthält, wird eine .class-Datei mit dem Java-Byte-Code erzeugt.
- Es muss mit dem Hamster-Simulator eine Datei mit einem Territorium erzeugt und gespeichert werden (in unserem Beispiel in die Datei test/sammler.ter)
- Nun kann das Hamster-Programm in dem Territorium ausgeführt werden. Das geht durch folgenden Aufruf:  

```
java -classpath hamstersimulator.jar;tools.jar de.hamster.run <class-Datei> <ter-Datei>
```

  
Im konkreten Beispiel:  

```
java -classpath hamstersimulator.jar;tools.jar de.hamster.run test/sammler.class test/sammler.ter
```
- Dann erscheint das Hamster-Territorium und man muss nur noch auf den Start-Button drücken.

### 1.7.2 Hamstern mit BlueJ

BlueJ ([www.bluej.org](http://www.bluej.org)) ist eine Entwicklungsumgebung für objektorientierte Java-Programme, die speziell für Programmieranfänger entworfen wurde. BlueJ richtet sich also an dieselbe Zielgruppe wie das Java-Hamster-Modell. Mit der Entwicklungsumgebung einher geht eine didaktische Methode zur Einführung in die objektorientierte Programmierung. Ihr zugrunde liegt ein iteratives Vorgehen bei der Einführung der Konzepte der objektorientierten Programmierung, das unter dem Motto

„Objekte zuerst“ steht. Eine der großen Stärken von BlueJ ist die Möglichkeit des interaktiven Erzeugens von Objekten und des interaktiven Umgangs mit diesen. Eine weitere Stärke ist die Visualisierung der Programmstruktur durch Diagramme. Dagegen liegt die besondere Stärke des Java-Hamster-Modells insbesondere in Verbindung mit dem Hamster-Simulator in der Visualisierung der Ausführung eines Programms. Der Programmierer sieht von Anfang an in einer ansprechenden Umgebung, was seine Programme bewirken.

Was ab Version 2.2 des Hamster-Simulators nun möglich ist, ist seine Integration in BlueJ. Hamster-Programme können mit den Werkzeugen und Möglichkeiten, die BlueJ bietet, entwickelt und im Hamster-Simulator ausgeführt werden. Konkret bedeutet das an Vorteilen für Programmieranfänger, BlueJ visualisiert die Programmstruktur und erlaubt insbesondere die interaktive Erzeugung von Hamstern und den interaktiven Aufruf von Hamster-Befehlen und der Hamster-Simulator visualisiert die Programmausführung, d.h. der Programmierer sieht unmittelbar in einer graphischen Umgebung, was seine Anweisungen bzw. Programme bewirken.

Genauere Informationen zum „Hamstern mit BlueJ“ können Sie dem PDF-Dokument `HamsternMitBlueJ.pdf` entnehmen, das Sie auf der Website zum Java-Hamster-Modell finden.

## 1.8 Änderungen in Version 2.1 gegenüber 2.0

Gegenüber der Version 2.0 des Hamster-Simulators enthält Version 2.1 folgende Änderungen:

- Ein paar Fehler wurden behoben, bspw. der Fehler beim Schließen des Simulators, wenn eine oder mehrere Dateien noch nicht gespeichert wurden.
- Das Simulation-Fenster erscheint nun beim Start des Simulators größer.
- Linkshänder können nun im Editor auch `<ctrl><Einf>` zum Kopieren und `<Shift><Einf>` zum Einfügen nutzen.
- Über eine so genannte Property-Datei können sie bestimmte Voreinstellungen überlagern. Die Datei muss den Namen „hamster.properties“ haben und sich in dem Ordner befinden, wo sich auch die Dateien „hamstersimulator.jar“ bzw. „hamstersimulator.bat“ befinden. Momentan sind folgende Einstellungen möglich:
- **security:** Befindet sich in der Datei „hamster.properties“ eine Zeile mit dem Text `security=false`, wird der so genannte Security-Manager ausgeschaltet. Das bedeutet, Hamster-Programme dürfen auf die Festplatte zugreifen und dürfen Dateien lesen und in Dateien schreiben. Damit ist es bspw. auch möglich, aus Hamster-Programmen heraus Sounds abzuspielen. Aber Vorsicht, sollten sie diese Option gesetzt haben, empfehle ich Ihnen dringend, keine fremden Hamster-Programme auszuführen. Sind diese bspw. böswillig geschrieben, könnten sie Ihnen prinzipiell die gesamte Festplatte löschen. Standardmäßig steht in der Property-Datei `security=true`.

- **workspace:** Standardmäßig erscheint im Dateibaum als oberster Ordner ein Ordner `Programme`, der so genannte *Workspace-Ordner*. Er repräsentiert den Unterordner `Programme` des Ordners, in dem sich die Dateien „hamstersimulator.jar“ bzw. „hamstersimulator.bat“ befinden. In diesem Ordner werden alle Hamster-Programme und Hamster-Territorien abgespeichert. Durch Setzen der Property `workspace`<sup>1</sup> kann ein anderer Ordner als Workspace-Ordner festgelegt werden. Befindet sich in der Datei eine Zeile, die mit dem Text `workspace=` beginnt, wird der dahinter angegebene Ordner als Workspace-Ordner gesetzt, bspw.

`workspace=C:/Dokumente und Einstellungen/karl` oder  
`workspace=C:/Dokumente und Einstellungen/heidi/Eigene Dateien`. Der angegebene Ordner muss existieren und er muss lesbar und beschreibbar sein! Achten Sie bitte darauf, dass in dem Ordner-Namen keine Sonderzeichen vorkommen (bspw. ein Ausrufezeichen), da die aktuelle Java-Version (5.0) im Moment nicht damit zurecht kommt. Für Windows-Nutzer ist es wichtig zu wissen, dass die \-Zeichen in den Ordner-Namen durch ein /-Zeichen ersetzt werden müssen. Alternativ kann auch jedes \-Zeichen verdoppelt werden. Standardmäßig steht in der Property-Datei `workspace=Programme`.

Mehr Informationen zu Properties finden Sie im Abschnitt 5.

## 1.9 Anmerkungen zur alten Version 1 des Hamster-Simulators

Version 1 des Hamster-Simulators wird nicht weiter unterstützt!

Die aktuelle Version 2 des Simulators hat nur noch wenige Gemeinsamkeiten mit Version 1. Die Benutzungsoberfläche wurde vollkommen redesigned und die Funktionalität stark erweitert (zum Beispiel Kapselung der Dateiverwaltung, Integration eines Debuggers, ...). Benutzer können deutlich weniger Fehler bei der Installation und Bedienung machen, als dies noch in Version 1 der Fall war.

Wenn Sie trotzdem mit Version 1 des Simulators arbeiten möchten, weil Sie sich bspw. an die Version gewöhnt haben, sei Ihnen gesagt: Version 1 unterstützt nicht die Entwicklung objektorientierter Hamster-Programme, dient also lediglich als Begleitprogramm zum ersten Band der beiden Hamster-Bücher.

Hamster-Programme, die mit der Version 1 des Hamster-Simulators erstellt wurden, können auf zweierlei Art und Weise in die neue Version 2 übernommen werden:

- Sie übertragen den alten Sourcecode mittels Copy-Paste in den Editor des neuen Hamster-Simulators, speichern diesen und kompilieren.

---

<sup>1</sup>Aus Kompatibilität zu früheren Versionen des Hamster-Simulators kann diese Property auch **home** genannt werden.

- In dem Ordner, in dem die Datei `hamstersimulator.jar` liegt, befindet sich ein Unter-Ordner namens `Programme`. Kopieren Sie die alten „.ham“-Dateien einfach in diesen Ordner oder in Unter-Ordner des Ordners. Anschließend können Sie die Dateien über den Editor des neuen Hamster-Simulators öffnen und kompilieren. Achtung: Sie müssen die alten „.ham“-Dateien auf jeden Fall neu kompilieren. Die alten ausführbaren „.class“-Dateien funktionieren nicht mehr!

Hamster-Territorien, die mit Version 1 des Hamster-Simulators erstellt und abgespeichert wurden, können leider in Version 2 des Hamster-Simulators nicht mehr benutzt werden. Sie müssen sie im Simulation-Fenster von Version 2 des Hamster-Simulators neu erstellen und abspeichern.

## 2 Installation und Starten des Hamster-Simulators

Der Hamster-Simulator läuft zur Zeit auf Windows-, Macintosh-, Linux- und Solaris-Rechnern. Dort haben wir ihn auch getestet.

Da er in Java geschrieben ist, müsste er eigentlich auch auf allen anderen Rechnern laufen, für die eine Java JVM existiert.

### 2.1 Laden und Installation einer Java-Laufzeitumgebung

Der Hamster-Simulator ist ein in Java geschriebenes Programm. Um es ausführen zu können, muss auf Ihrem Rechner eine Java-Laufzeitumgebung installiert werden. In Java gibt es nun zwei Varianten, dies zu tun.

Die erste Variante ist die Installation eines „Java SE Development Kit (JDK)“. Das JDK steht in verschiedenen Versionen zur Verfügung. Aktuell (Stand 04.05.2009) ist die Version 6 (manchmal wird sie auch als Version 1.6 bezeichnet). Sie sollten möglichst immer die aktuellste Version installieren. Ich empfehle dringend die Nutzung der Version 6. Der Hamster-Simulator sollte jedoch auch noch in der Java-Version 5 laufen.

Neben der benötigten Java-Laufzeitumgebung beinhaltet ein JDK noch weitere Werkzeuge zur Entwicklung von Java-Programmen, wie bspw. einen Compiler. Sie sollten also ein JDK installieren, wenn Sie außer Hamster-Programmen noch „richtige“ Java-Programme entwickeln möchten. Problem beim JDK ist: Es ist sehr groß (ca. 70 MByte). Es kann über den folgenden URL kostenlos aus dem WWW geladen werden: <http://java.sun.com/javase/downloads/index.jsp>

Die zweite Variante ist die Installation eines „Java SE Runtime Environment (JRE)“. Dieses stellt genau die benötigte Laufzeitumgebung dar. JREs sind in JDKs enthalten, existieren also in denselben Versionen. Die Vorteile von JREs sind: Sie sind weit weniger groß (ca. 20 MBytes). Auch JREs können über den folgenden URL kostenlos aus dem WWW geladen werden <http://java.sun.com/javase/downloads/index.jsp>.

Sowohl das JDK als auch das JRE werden standardmäßig von der Firma SUN für die Betriebssysteme Windows, Linux und Solaris zur Verfügung gestellt, leider nicht für Macintosh-Betriebssysteme.

Nachdem Sie entweder ein JDK oder ein JRE auf Ihren Rechner geladen haben, müssen Sie es installieren. Das geschieht normalerweise durch Ausführen der geladenen Datei (in Windows Doppelklick auf die .exe-Datei). Sie werden dann durch die weitere Installation geführt.

### 2.2 Laden und Installation des Hamster-Simulators

Auf der Hamster-Website (<http://www.java-hamster-modell.de>) befindet sich im Bereich „Simulator“ eine Datei namens „hamstersimulator-v26-01.zip“. .zip-Dateien sind

Dateien, die mehrere andere Dateien komprimiert speichern. Die Datei „hamstersimulator-v26-01.zip“ müssen Sie auf Ihren Rechner laden und anschließend entpacken. Die Datei enthält eine Reihe von Dateien und vier Ordner. Die wichtigsten sind:

- Datei `hamstersimulator.jar`: Enthält den ausführbaren Hamster-Simulator.
- Datei `hamstersimulator.bat`: Alternative zum Starten des Hamster-Simulators unter Windows.
- Datei `handbuch.pdf`: Das Handbuch zum Hamster-Simulator im PDF-Format.
- Datei `hamster.properties`: Datei zum Einstellen bestimmter Eigenschaften.
- Ordner `Programme`: Enthält standardmäßig alle Hamster-Programme der zwei Hamster-Bücher. In diesem Ordner werden von Ihnen entwickelte Hamster-Programme abgespeichert. Dieser Ordner muss sich immer in demselben Ordner befinden, in dem auch die Dateien `hamstersimulator.jar` und `hamstersimulator.bat` liegen!
- Ordner `lib`: Enthält diverse benötigte Java-Bibliotheken. Insbesondere die Datei `tools.jar`, in dem der Java-Compiler steckt.
- Ordner `handbuch`: Das Handbuch im HTML-Format.
- Ordner `HamsternMitBlueJ`: Anleitung wie man mit der Entwicklungsumgebung BlueJ Hamster-Programme entwickeln und ausführen kann, inkl. einiger Beispielprojekte.
- Ordner `pseudo-hamsterklassen`: Hier liegen die Klassen aus Anhang A von Band 2 des Java-Hamster-Buches. Sie sind pseudomäßig implementiert und sollen dazu dienen, dass fortgeschrittene Hamster-Programmierer auch andere Entwicklungsumgebungen, wie bspw. Eclipse, zum Eintippen und Compilieren von Hamster-Programmen nutzen können.

## 2.3 Hamster-Simulator unter JDK 1.4

Die veraltete Java-Version 1.4 wird nicht weiter unterstützt!

## 2.4 Starten des Hamster-Simulators

Nachdem Sie eine Java-Laufzeitumgebung sowie den Hamster-Simulator wie oben beschrieben auf Ihrem Rechner installiert haben, können Sie den Simulator starten. Dies geschieht folgendermaßen.

- Unter Windows: Führen Sie mit der Maus einen Doppelklick auf die Datei `hamstersimulator.jar` oder die Datei `hamstersimulator.bat` aus.<sup>2</sup>
- Unter Linux und Solaris: Rufen Sie in dem Ordner, in dem sich die Datei `hamstersimulator.jar` befindet, folgenden Befehl auf:
- `java -jar hamstersimulator.jar`.
- Unter Macintosh (OS X): Führen Sie mit der Maus einen Doppelklick auf die Datei `hamstersimulator.jar` aus.

Anschließend öffnen sich zwei Fenster, die mit `Editor` und `Simulation` betitelt sind.

Herzlichen Glückwunsch! Sie können mit der Entwicklung von Hamster-Programmen beginnen!

---

<sup>2</sup>Eine weitere Alternative besteht darin, ein Eingabeaufforderung-Fenster zu öffnen, sich in den Ordner zu begeben, in dem sich die Datei `hamstersimulator.jar` befindet, und dort folgenden Befehl einzugeben: `java -jar hamstersimulator.jar`.



### 3 Ihr erstes Hamster-Programm

Nachdem Sie den Hamster-Simulator gestartet haben, öffnen sich auf dem Bildschirm zwei neue Fenster: das *Editor-Fenster* (siehe auch Abbildung 1) und das *Simulation-Fenster* (siehe auch Abbildung 2). Sie erkennen die beiden Fenster an ihren Titeln Editor bzw. Simulation. Im Großen und Ganzen kann man sagen: Im Editor-Fenster entwickeln Sie Hamster-Programme und im Simulation-Fenster führen Sie Hamster-Programme aus.

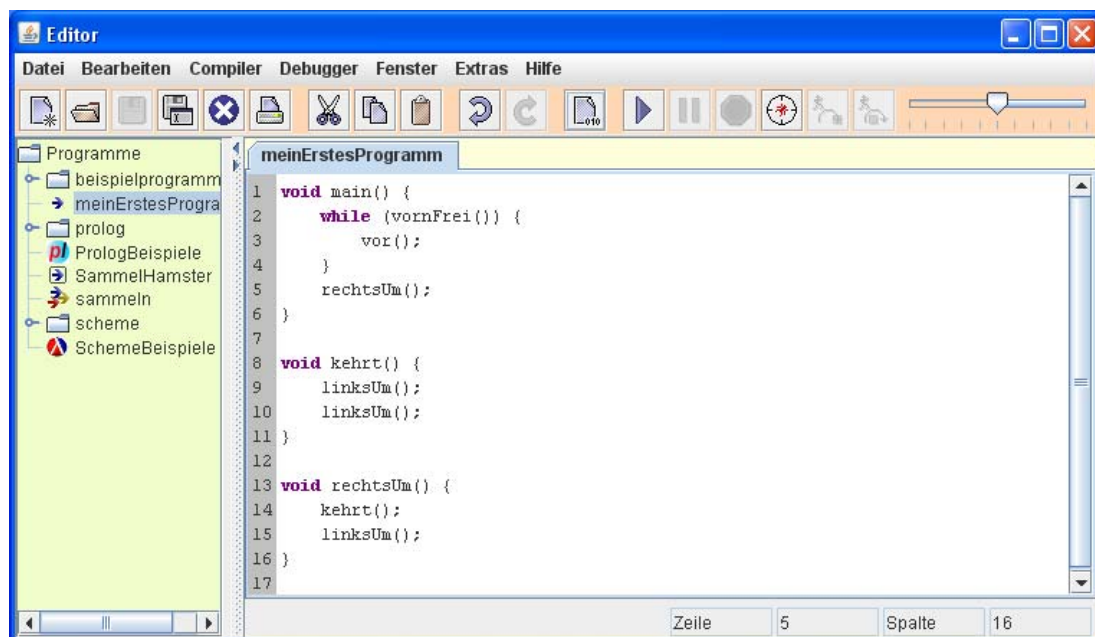


Abbildung 1: Editor-Fenster

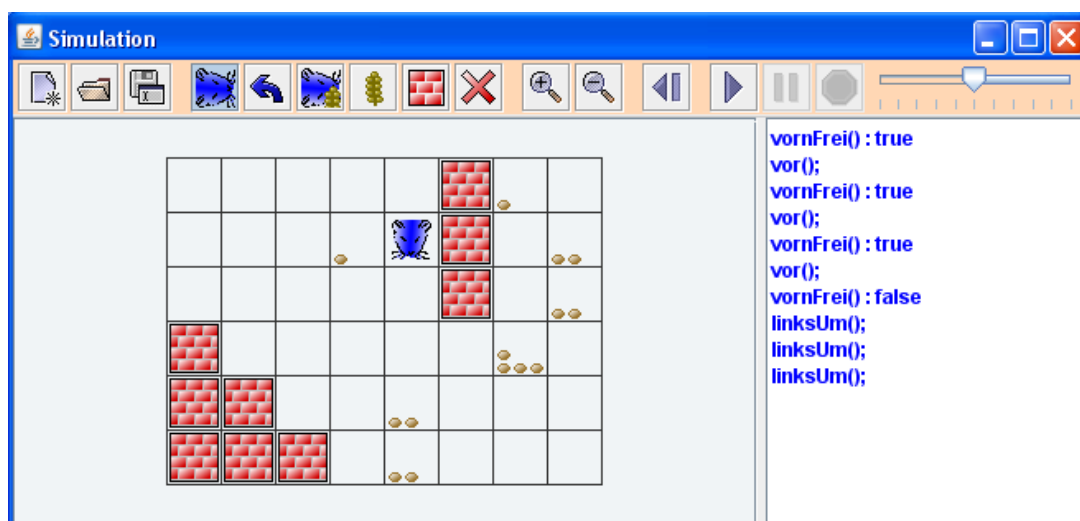


Abbildung 2: Simulation-Fenster

Im Folgenden wird im Detail beschrieben, was Sie machen müssen, um Ihr erstes Hamster-Programm zu schreiben und auszuführen. Insgesamt müssen/können fünf Stationen durchlaufen werden:

- Gestaltung eines Hamster-Territoriums
- Eingeben eines Hamster-Programms
- Compilieren eines Hamster-Programms
- Ausführen eines Hamster-Programms
- Debuggen eines Hamster-Programms

### 3.1 Gestaltung eines Hamster-Territoriums

Als erstes wollen wir ein Hamster-Territorium aufbauen, in dem unser Programm ablaufen soll. Das geschieht im Simulation-Fenster (siehe auch Abbildung 2). In der Mitte sehen wir das Hamster-Territorium, darüber eine so genannte *Toolbar* mit graphischen Buttons. Fahren Sie einfach mal mit der Maus über die einzelnen Buttons der Toolbar, dann erscheint jeweils ein Tooltipp, der beschreibt, wozu dieser Button dient.

Zunächst werden wir die Größe des Territoriums anpassen. Klicken Sie dazu auf den Button „Neues Territorium“ (erster Button von links). Es erscheint eine Dialogbox, in der Sie die gewünschte Anzahl an Reihen und Spalten eingeben können. Um die dort erscheinenden Werte (jeweils 10) ändern zu können, klicken Sie mit der Maus auf das entsprechende Eingabefeld. Anschließend können Sie den Wert mit der Tastatur eingeben. Nach der Eingabe der Werte klicken Sie bitte auf den OK-Button. Die Dialogbox schliesst sich und das Territorium erscheint in der angegebenen Größe. Achtung: Wenn Sie die Größe des Territoriums ändern, wird das Territorium automatisch komplett geleert!

Nun werden wir den Hamster, der immer im Territorium sitzt, – im objektorientierten Hamster-Modell wird er *Standard-Hamster* genannt – umplatzieren. Dazu klicken wir mit der Maus auf den Button „Hamster versetzen“ (vierter Button von links). Dieser Button ist nun aktiviert. Das erkennt man an dem etwas dunkleren Hintergrund. Solange er aktiviert ist, kann der Standard-Hamster im Territorium auf eine andere Kachel gesetzt werden. Klicken Sie dazu einfach auf die entsprechende Kachel.

Standardmäßig schaut der Standard-Hamster nach Osten. Mit dem Button „Hamster drehen“ (fünfter Button von links) können Sie jedoch seine Blickrichtung ändern. Jedes Mal, wenn Sie auf den Button klicken, dreht er sich um 90 Grad nach links.

Normalerweise hat der Standard-Hamster 0 Körner im Maul. Mit Hilfe des Buttons „Körner im Maul“ (sechster Button von links) lässt sich dies ändern. Wenn Sie auf den Button klicken, erscheint eine Dialogbox. Sie sehen eine Zahl, die die aktuelle Anzahl an Körnern im Maul des Hamsters angibt. Wenn Sie diese Anzahl ändern wollen, tippen Sie einfach über die Tastatur die gewünschte Zahl ein und klicken Sie anschließend auf den OK-Button in der Dialogbox. Die Dialogbox wird anschließend automatisch wieder geschlossen.

Nun wollen wir auf einigen Kacheln Körner platzieren. Hierzu dient der Button „Körner setzen“ (siebter Button von links). Wenn Sie ihn mit der Maus anklicken, wird er aktiviert.

Der bis jetzt aktivierte „Hamster versetzen“-Button wird automatisch deaktiviert. Es kann immer nur ein Button aktiviert sein. Solange der „Körner setzen“-Button aktiviert ist, können Sie nun auf die Kacheln des Territoriums Körner legen. Klicken Sie dazu mit der Maus auf die entsprechende Kachel. Es erscheint eine Dialogbox, in der Sie die gewünschte Anzahl an Körnern eingeben können. Anschließend müssen Sie auf den OK-Button in der Dialogbox klicken. Diese schließt sich und auf der Kachel sehen Sie die angegebene Anzahl an Körnern. Dabei gilt: Es werden maximal 12 Körner angezeigt, auch wenn dort mehr Körner liegen. Wenn Sie die genaue Anzahl an Körnern auf einer Kachel ermitteln möchten, fahren Sie einfach mit der Maus über die Kachel. Es erscheint ein Tooltipp, in dem die Anzahl an Körnern auf der Kachel angegeben wird.

Sie können übrigens auch die Körneranzahl auf mehreren Kacheln gleichzeitig festlegen. Klicken Sie dazu mit der Maus auf eine Kachel und ziehen Sie die Maus bei gedrückter Maustaste über die Kacheln, deren Körneranzahl Sie ändern möchten. Erst, wenn Sie die Maustaste loslassen, erscheint die Dialogbox. Geben Sie dann die Anzahl ein und klicken Sie den OK-Button. Die Körneranzahl wird auf allen Kacheln verändert, die Sie vorher markiert hatten.

Mauern werden ähnlich wie Körner auf Kacheln platziert. Aktivieren Sie zunächst den „Mauer setzen“-Button (achter Button von links). Klicken Sie anschließend auf die Kacheln, die durch eine Mauer blockiert werden sollen.

Möchten Sie bestimmte Kacheln im Territorium wieder leeren, so dass weder eine Mauer noch Körner auf ihnen platziert sind, so aktivieren Sie den „Kachel löschen“-Button (neunter Button von links). Klicken Sie anschließend auf die Kacheln, die geleert werden sollen.

So, jetzt wissen Sie eigentlich alles, was notwendig ist, um das Hamster-Territorium nach Ihren Wünschen zu gestalten. Bevor Sie weiterlesen, erzeugen Sie als nächstes das in Abbildung 2 skizzierte Territorium.

Sie können ein bestimmtes Territorium auch in einer Datei abspeichern, wenn Sie es irgendwann noch einmal benutzen möchten, ohne alle Eingaben erneut zu tätigen. Drücken Sie einfach auf den „Territorium speichern“-Button (dritter Button von links) und geben Sie in der sich öffnenden Dateiauswahl-Dialogbox einen Namen an, zum Beispiel `MeinErstesTerritorium`. Wenn Sie dann auf den OK-Button klicken, wird das Territorium in einer Datei mit diesem Namen gespeichert. Der Dateiname erhält übrigens automatisch die Endung „.ter“.

Zum Wiederherstellen eines gespeicherten Territoriums, klicken Sie auf den „Territorium öffnen“-Button (zweiter Button von links). Es erscheint eine Dateiauswahl-Dialogbox. In der Mitte werden die Namen der existierenden Dateien mit abgespeicherten Territorien angezeigt. Klicken Sie mit der Maus auf den Dateinamen, in dem das Territorium abgespeichert ist, das Sie laden möchten. Wenn Sie danach auf den OK-Button klicken, schließt sich die Dateiauswahl-Dialogbox und das abgespeicherte Territorium wird wieder hergestellt.

## 3.2 Eingeben eines Hamster-Programms

Nachdem wir unser erstes Hamster-Territorium im Simulation-Fenster gestaltet haben, begeben wir uns nun in das Editor-Fenster. Dort werden wir unser erstes Hamster-Programm schreiben.

Im Editor-Fenster befindet sich ganz oben eine Menüleiste und darunter eine Toolbar mit graphischen Buttons. Links sehen wir den Dateibaum und das große Feld rechts ist der Eingabebereich für den Sourcecode.

Bevor wir ein Hamster-Programm eintippen, müssen wir zunächst einen neuen Programmrahmen erzeugen. Dazu klicken wir auf den „Neu“-Button (erster Button von links in der Toolbar). Es erscheint eine Dialogbox, in der wir uns für den Typ des Programms (imperatives Programm, objektorientiertes Programm, Klasse oder Scheme-Programm) entscheiden müssen. Unser erstes Programm soll ein imperatives Hamster-Programm gemäß Band 1 der zwei Hamster-Bücher werden. Daher wählen wir dies aus und klicken den OK-Button. Der Eingabebereich wird heller und es erscheint ein Programmrahmen für imperative Hamster-Programme:

```
void main() {  
  
}
```

Unser erstes Programm soll bewirken, dass der Hamster in dem gerade von uns gestalteten Territorium zwei Körner frisst. Wir klicken in die zweite Reihe des Eingabebereiches und tippen dort wie in einem normalen Editor bzw. Textverarbeitungsprogramm, wie Microsoft Word, die entsprechenden Hamster-Befehle ein, so dass letztlich folgendes im Eingabebereich steht:

```
void main() {  
    vor();  
    vor();  
    nimm();  
    rechtsUm();  
    vor();  
    vor();  
    nimm();  
}  
  
void rechtsUm() {  
    linksUm();  
    linksUm();  
    linksUm();  
}
```

Das ist unser erstes Hamster-Programm. Wir müssen es als nächstes in einer Datei abspeichern. Dazu klicken wir den „Speichern“-Button (dritter Button von links). Es erscheint eine Dateiauswahl-Dialogbox. Hier geben wir den gewünschten Dateinamen ein. Dies muss ein gültiger Java-Bezeichner sein, zum Beispiel `MeinErstesHamster-Programm`. Der Dateiname erhält übrigens automatisch die Endung `„.ham“`. Anschließend klicken wir den OK-Button. Damit ist unser Programm in der entsprechenden Datei abgespeichert.

Ihnen sicher von anderen Editoren bzw. Textverarbeitungsprogrammen bekannte Funktionen, wie „Ausschneiden“, „Kopieren“, „Einfügen“, „Rückgängig“ und „Wiederherstellen“ können Sie über das „Bearbeiten“-Menü bzw. die entsprechenden Buttons in der Toolbar ausführen (siebter bis elfter Button von links).

Weiterhin gibt es einen „Öffnen“-Button zum Öffnen von Dateien, die irgendwann einmal abgespeichert worden sind (zweiter Button von links). Es erscheint eine Dateiauswahl-Dialogbox, in der Sie die entsprechende Datei durch Mausklick auswählen. Nach dem Anklicken des OK-Buttons erscheint das Programm, das die Datei enthält, im Eingabebereich. Eine Alternative zum „Öffnen“-Button ist das Anklicken des entsprechenden Dateinamens im Dateibaum auf der linken Seite.

Wenn Sie ein Programm in einer anderen Datei abspeichern möchten, nutzen Sie den „Speichern Als“-Button (vierter Button von links). Mit dem „Schließen“-Button (fünfter Button von links) können Sie eine Datei wieder schließen. Das entsprechende Programm verschwindet dann aus dem Eingabebereich.

Zu guter Letzt gibt es noch den „Drucken“-Button (sechster Button von links) zum Ausdrucken eines Hamster-Programms.

Alle gerade erläuterten Funktionen zum Verwalten von Dateien mit Hamster-Programmen finden Sie auch im Menü „Datei“.

### **3.3 Compilieren eines Hamster-Programms**

Nachdem wir unser Hamster-Programm geschrieben und in einer Datei abgespeichert haben, müssen wir es kompilieren. Der Compiler überprüft den Sourcecode auf syntaktische Korrektheit und transformiert ihn – wenn er korrekt ist – in ein ausführbares Programm. Zum Kompilieren drücken Sie einfach auf den „Kompilieren“-Button (zwölfter Button von links oder „Kompilieren“-Menü). Kompiliert wird dann das Programm, das gerade im Eingabebereich sichtbar ist.

Wenn das Programm korrekt ist, erscheint eine Dialogbox mit der Nachricht „Kompilierung erfolgreich“. Zur Bestätigung müssen Sie anschließend noch den OK-Button drücken. Das Programm kann nun ausgeführt werden. Merken Sie sich bitte: Immer, wenn Sie Änderungen am Sourcecode Ihres Programms vorgenommen haben, müssen Sie es zunächst abspeichern und dann neu kompilieren. Sonst werden die Änderungen nicht berücksichtigt!

Wenn das Programm syntaktische Fehler enthält – wenn Sie sich bspw. bei der Eingabe des obigen Programms vertippt haben –, werden unter dem Eingabebereich die Fehlermeldungen des Compilers eingeblendet. Diese erscheinen in englischer Sprache.

Weiterhin wird die Zeile angegeben, in der der Fehler entdeckt wurde. Wenn Sie mit der Maus auf die Fehlermeldung klicken, springt der Cursor im Eingabebereich automatisch in die angegebene Zeile.

Vorsicht: Die Fehlermeldungen sowie die Zeilenangabe eines Compilers sind nicht immer wirklich exakt. Das Interpretieren der Meldungen ist für Programmieranfänger häufig nicht einfach und bedarf einiger Erfahrungen. Deshalb machen Sie ruhig am Anfang mal absichtlich Fehler und versuchen Sie, die Meldungen des Compilers zu verstehen.

Tipp: Arbeiten Sie die Fehler, die der Compiler entdeckt hat, immer von oben nach unten ab. Wenn Sie eine Meldung dann überhaupt nicht verstehen, speichern Sie ruhig erst mal ab und kompilieren Sie erneut. Häufig ist es (leider) so, dass der Compiler für einen einzelnen Fehler mehrere Fehlermeldungen ausgibt, was Anfänger leicht verwirren kann.

Nachdem Sie die Fehler korrigiert haben, müssen Sie das Programm zunächst erst wieder speichern und dann erneut kompilieren. Wiederholen Sie dies so lange, bis der Compiler die Meldung „Kompilierung erfolgreich“ ausgibt. Erst dann können Sie das Programm ausführen!

### **3.4 Ausführen eines Hamster-Programms**

Nach dem erfolgreichen Kompilieren ist es endlich soweit: Wir können den Hamster bei der Arbeit beobachten. Macht er wirklich das, was wir ihm durch unser Programm beigebracht haben?

Zum Ausführen eines Programms begeben wir uns wieder in das Simulation-Fenster. Zum Steuern der Programmausführung dienen dort die drei rechten Buttons rechts in der Toolbar. Durch Anklicken des „Ausführen“-Buttons (dritter Button von rechts) starten wir das Programm. Ausgeführt wird übrigens automatisch das Programm, das sich im Editor-Fenster gerade im Eingabebereich befindet. Wenn Sie bis hierhin alles richtig gemacht haben, sollte der Hamster loslaufen und wie im Programm beschrieben, zwei Körner einsammeln. Herzlichen Glückwunsch zu Ihrem ersten Hamster-Programm!

Wollen Sie die Programmausführung anhalten, können Sie dies durch Anklicken des „Pause“-Buttons (zweiter Button von rechts) erreichen. Der Hamster stoppt so lange, bis Sie wieder den „Ausführen“-Button anklicken. Dann fährt der Hamster mit seiner Arbeit fort. Das Programm vorzeitig komplett abbrechen, können Sie mit Hilfe des „Stopp“-Buttons (erster Button von rechts).

Rechts neben dem Hamster-Territorium werden übrigens während der Programmausführung jeweils die Hamster-Befehle angezeigt, die der Hamster gerade ausführt.

Wenn Sie ein Programm mehrmals hintereinander im gleichen Territorium ausführen, können Sie mit dem „Rücksetzen“-Button (vierter Button von rechts) den Zustand des Territoriums wieder herstellen, der vor Ausführen des Programms bestand.

Der Schieberegler ganz rechts in der Menüleiste dient zur Steuerung der Geschwindigkeit der Programmausführung. Je weiter Sie den Knopf nach links verschieben, umso langsamer erledigt der Hamster seine Arbeit. Je weiter Sie ihn nach rechts verschieben, umso schneller flitzt der Hamster durchs Territorium.

Die Bedienelemente zum Steuern der Programmausführung („Ausführen“-Button, „Pause“-Button, „Stopp“-Button und Geschwindigkeitsregler) finden Sie übrigens auch im Editor-Fenster sowie im „Debugger“-Menü des Editor-Fensters. Welche Sie nutzen, ist Ihnen überlassen.

### 3.5 Debuggen eines Hamster-Programms

„Debuggen eines Programms“ eines Programms bedeutet, dass Sie bei der Ausführung eines Programms zusätzliche Möglichkeiten zur Steuerung besitzen und sich den Zustand des Programms (welche Zeile des Sourcecodes wird gerade ausgeführt, welche Werte besitzen aktuell die Variablen) in bestimmten Situationen anzeigen lassen können. Den Debugger können Sie im Editor-Fenster mit dem „Debugger aktivieren“-Button (dritter Button der Menüleiste von rechts) aktivieren und wieder deaktivieren. Wenn er aktiviert ist, erscheint der Button etwas dunkler.

Wenn der Debugger aktiviert ist und Sie über den „Ausführen“-Button ein Hamster-Programm starten, öffnen sich oberhalb des Eingabebereichs im Editor-Fenster zwei neue Bereiche. Im linken Bereich wird angezeigt, in welcher Funktion sich der Programmablauf gerade befindet. Im rechten Bereich werden die Variablen und ihre aktuellen Werte dargestellt. Außerdem wird im Eingabebereich durch einen blauen Balken gekennzeichnet, welche Zeile des Programms ausgeführt wird.

Bei aktiviertem Debugger haben Sie die Möglichkeit, das Programm schrittweise, d.h. Anweisung für Anweisung, auszuführen. Das können Sie mit Hilfe der beiden rechten Buttons in der Menüleiste des Editor-Fensters. Der linke Button heißt „Schritt hinein“-Button, der rechte „Schritt über“-Button. Normalerweise bewirken die beiden Buttons das gleiche: die nächste Anweisung – und nur die – wird ausgeführt. Wenn die nächste auszuführende Anweisung jedoch der Aufruf einer von Ihnen definierten Prozedur oder Funktion ist, bewirkt der „Schritt über“-Button die Ausführung der kompletten Prozedur (ohne zwischendurch anzuhalten), während durch das Anklicken des „Schritt hinein“-Buttons zur ersten Anweisung des entsprechenden Funktionsrumpfs verzweigt wird und Sie dadurch die Möglichkeit haben, auch die Ausführung der Funktion schrittweise zu tätigen.

Sie können bei aktiviertem Debugger zunächst auch einfach das Programm durch Anklicken des „Ausführen“-Buttons starten und beobachten. Wenn Sie dann den „Pause“-Button drücken, haben Sie anschließend ebenfalls die Möglichkeit der schrittweisen Ausführung ab der aktuellen Position.

Den „Pause“-Zustand mit der Möglichkeit der schrittweisen Ausführung eines Programms können Sie jederzeit wieder durch Anklicken des „Ausführen“-Buttons beenden. Das Programm läuft dann selbstständig wieder weiter.

## **3.6 Zusammenfassung**

Herzlichen Glückwunsch! Wenn Sie bis hierhin gekommen sind, haben Sie Ihr erstes Hamster-Programm erstellt und ausgeführt. Sie sehen, die Bedienung des Hamster-Simulators ist gar nicht so kompliziert.

Der Hamster-Simulator bietet jedoch noch weitere Möglichkeiten. Diese können Sie nun durch einfaches Ausprobieren selbst erkunden oder im nächsten Abschnitt nachlesen.



## 4 Bedienung des Hamster-Simulators

Im letzten Abschnitt haben Sie eine kurze Einführung in die Funktionalität des Hamster-Simulators erhalten. In diesem Abschnitt werden die einzelnen Funktionen des Simulators nun im Detail vorgestellt. Dabei wird sich natürlich einiges auch wiederholen.

Wenn Sie den Hamster-Simulator starten, öffnen sich zwei Fenster. Das eine heißt *Editor-Fenster*, das andere *Simulation-Fenster*. Sie erkennen die beiden Fenster an ihren Titeln: Editor bzw. Simulation. Abbildung 3 skizziert die einzelnen Komponenten des Editor-Fensters, Abbildung 4 die des Simulation-Fensters.

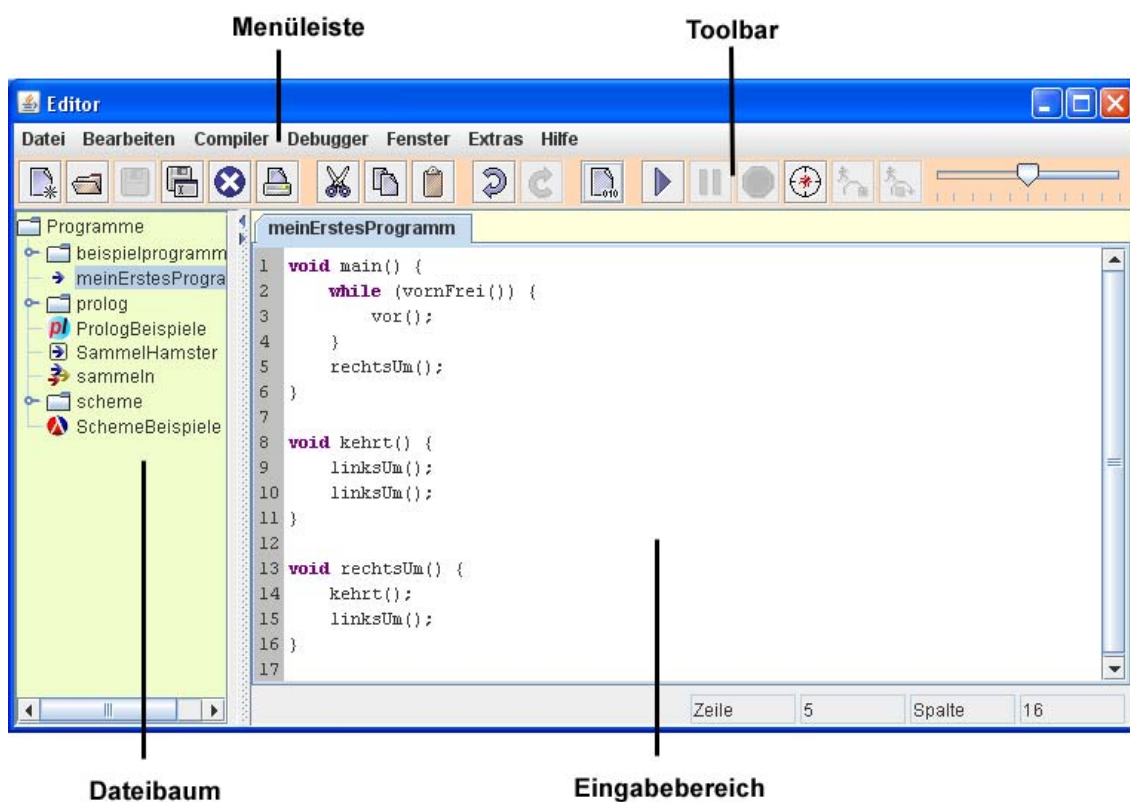


Abbildung 3: Komponenten des Editor-Fensters

Im Großen und Ganzen kann man sagen, dient das Editor-Fenster zum Editieren, Compilieren und Debuggen von Hamster-Programmen und das Simulation-Fenster zur Gestaltung des Hamster-Territoriums und zum Ausführen von Hamster-Programmen.

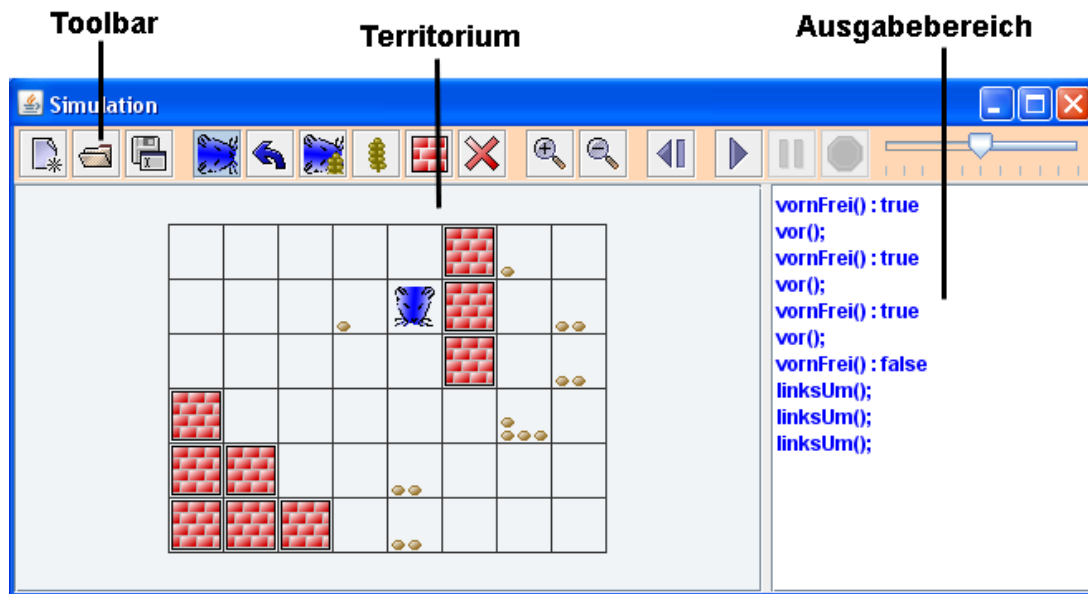


Abbildung 4: Komponenten des Simulation-Fensters

Damit wurden auch bereits die Hauptfunktionsbereiche des Hamster-Simulators genannt:

- Verwalten und Editieren von Hamster-Programmen
- Compilieren von Hamster-Programmen
- Verwalten und Gestalten von Hamster-Territorien
- Ausführen von Hamster-Programmen
- Debuggen von Hamster-Programmen

Bevor im Folgenden anhand dieser Funktionsbereiche der Simulator im Detail vorgestellt wird, werden zuvor noch einige Grundfunktionen graphischer Benutzungsoberflächen erläutert.

## 4.1 Grundfunktionen

In diesem Unterabschnitt werden einige wichtige Grundfunktionalitäten graphischer Benutzungsoberflächen beschrieben. Der Abschnitt ist für diejenigen unter Ihnen gedacht, die bisher kaum Erfahrungen mit Computern haben. Diejenigen von Ihnen, die schon längere Zeit einen Computer haben und ihn regelmäßig benutzen, können diesen Abschnitt ruhig überspringen.

### 4.1.1 Anklicken

Wenn im Folgenden von „Anklicken eines Objektes“ oder „Anklicken eines Objektes mit der Maus“ gesprochen wird, bedeutet das, dass Sie den Mauscursor auf dem Bildschirm

durch Verschieben der Maus auf dem Tisch über das Objekt platzieren und dann die – im Allgemeinen linke – Maustaste drücken.

### 4.1.2 Tooltips

Als *Tooltips* werden kleine Rechtecke bezeichnet, die automatisch auf dem Bildschirm erscheinen, wenn man den Mauscursor auf entsprechende Objekte platziert (siehe Abbildung 5). In den Tooltips werden bestimmte Informationen ausgegeben.



Abbildung 5: Tooltip

### 4.1.3 Button

*Buttons* sind Objekte der Benutzungsoberfläche, die man anklicken kann und die daraufhin eine bestimmte Aktion auslösen (siehe Abbildung 6). Buttons besitzen eine textuelle Beschreibung (z.B. „OK“) oder eine Graphik, die etwas über die Aktion aussagen. Sie erkennen Buttons an der etwas hervorgehobenen Darstellung. Graphik-Buttons sind in der Regel Tooltips zugeordnet, die die zugeordnete Aktion beschreiben.

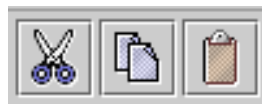


Abbildung 6: Buttons

### 4.1.4 Menü

*Menüs* befinden sich ganz oben in einem Fenster in der so genannten *Menüleiste* (siehe Abbildung 7). Sie werden durch einen Text beschrieben (Datei, Bearbeiten, ...). Klickt man die Texte an, öffnet sich eine Box mit so genannten *Menüitems*. Diese bestehen wiederum aus Texten, die man anklicken kann. Durch Anklicken von Menüitems werden genauso wie bei Buttons Aktionen ausgelöst, die im Allgemeinen durch die Texte beschrieben werden (Speichern, Kopieren, ...). Nach dem Anklicken eines Menüitems wird die Aktion gestartet und die Box schließt sich automatisch wieder. Klickt man irgendwo außerhalb der Box ins Fenster schließt sich die Box ebenfalls und es wird keine Aktion ausgelöst.

Häufig steht hinter den Menüitems ein weiterer Text, wie z.B. „Strg-O“ oder „Alt-N“. Diese Texte kennzeichnen Tastenkombinationen. Drückt man die entsprechenden Tasten, wird dieselbe Aktion ausgelöst, die man auch durch Anklicken des Menüitems auslösen würde.

Manchmal erscheinen bestimmte Menüitems etwas heller. Man sagt auch, sie sind ausgegraut. In diesem Fall kann man das Menüitem nicht anklicken und die zugeordnete Aktion nicht auslösen. Das Programm befindet sich in einem Zustand, in dem die Aktion keinen Sinn machen würde.



Abbildung 7: Menü

#### 4.1.5 Toolbar

Direkt unterhalb der Menüleiste ist die so genannte *Toolbar* angeordnet (siehe Abbildung 8). Sie besteht aus einer Menge an Graphik-Buttons, die Alternativen zu den am häufigsten benutzten Menüitems darstellen.

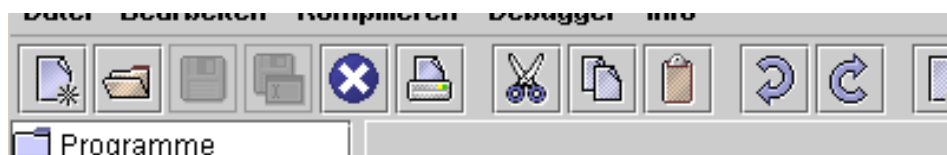


Abbildung 8: Toolbar

#### 4.1.6 Popup-Menü

*Popup-Menüs* sind spezielle Menüs, die bestimmten Elementen auf dem Bildschirm zugeordnet sind (siehe Abbildung 9). Man öffnet sie dadurch, dass man das Objekt zunächst anklickt und danach nochmal die rechte Maustaste drückt. Genauso wie bei normalen Menüs erscheint dann eine Box mit Menüitems.

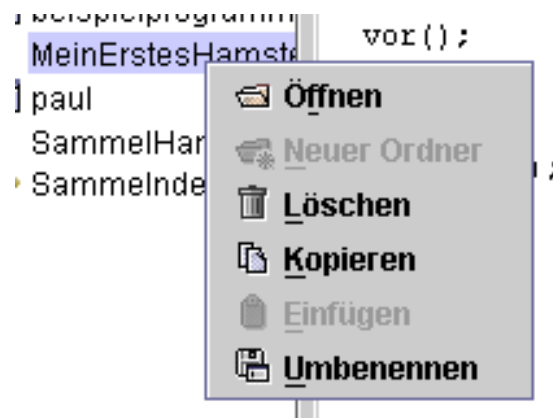


Abbildung 9: Popup-Menü

#### 4.1.7 Eingabefeld

*Eingabefelder* dienen zur Eingabe von Zeichen (siehe Abbildung 10). Positionieren Sie dazu den Mauscursor auf das Eingabefeld und klicken Sie die Maus. Anschließend können Sie über die Tastatur Zeichen eingeben, die im Eingabefeld erscheinen.

#### 4.1.8 Dialogbox

Beim Auslösen bestimmter Aktionen erscheinen so genannte *Dialogboxen* auf dem Bildschirm (siehe Abbildung 10). Sie enthalten in der Regel eine Menge von graphischen Objekten, wie textuelle Informationen, Eingabefelder und Buttons. Wenn eine Dialogbox auf dem Bildschirm erscheint, sind alle anderen Fenster des Programms für Texteingaben oder Mausklicks gesperrt. Zum Schließen einer Dialogbox, d.h. um die Dialogbox wieder vom Bildschirm verschwinden zu lassen, dienen in der Regel eine Menge an Buttons, die unten in der Dialogbox angeordnet sind. Durch Anklicken eines „OK-Buttons“ wird dabei die der Dialogbox zugeordnete Aktion ausgelöst. Durch Anklicken des „Abbrechen-Buttons“ wird eine Dialogbox geschlossen, ohne dass irgendwelche Aktionen ausgelöst werden.

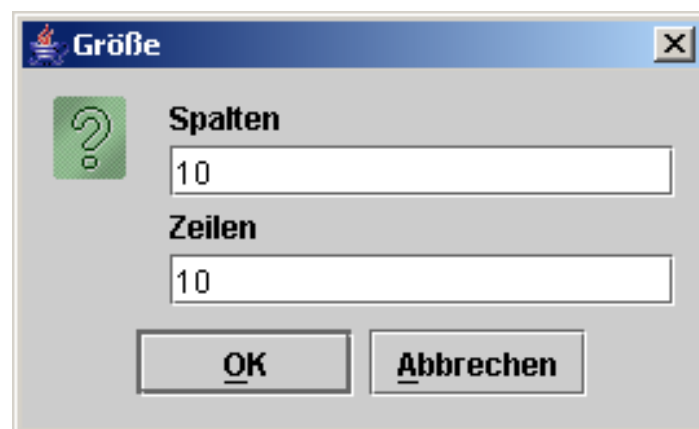


Abbildung 10: Dialogbox mit Eingabefeldern

### 4.1.9 Dateiauswahl-Dialogbox

*Dateiauswahl-Dialogboxen* sind spezielle Dialogboxen, die zum Speichern und Öffnen von Dateien benutzt werden (siehe Abbildung 11). Sie spiegeln im Prinzip das Dateisystem wider und enthalten Funktionalitäten zum Verwalten von Dateien und Ordnern.

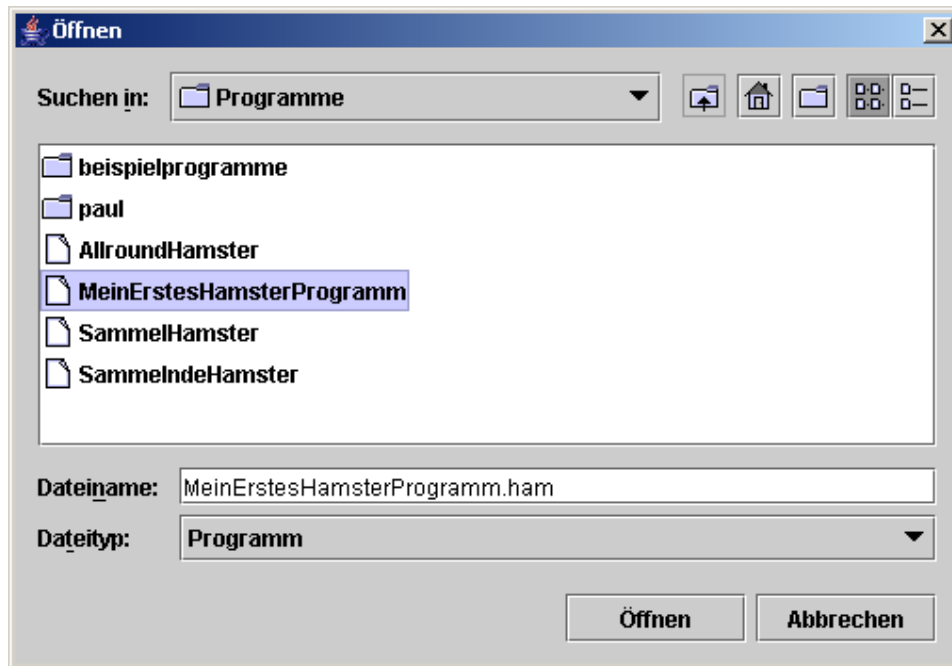


Abbildung 11: Dateiauswahl-Dialogbox

Im mittleren Bereich einer Dateiauswahl-Dialogbox erscheinen alle Dateien und Unterordner des aktuellen Ordners. Sie sind durch unterschiedliche Symbole repräsentiert. Der eigentliche Zweck von Dateiauswahl-Dialogboxen ist – wie der Name schon sagt – die Auswahl einer Datei. Klickt man auf eine Datei, erscheint der Name automatisch im Eingabefeld „Dateiname“. Dort kann man auch über die Tastatur einen Dateinamen eingeben. Anschließend wird nach Drücken des OK-Buttons die entsprechende Datei geöffnet bzw. gespeichert.

Dateiauswahl-Dialogboxen stellen jedoch noch zusätzliche Funktionalitäten bereit. Durch Doppelklick auf einen Ordner kann man in den entsprechenden Ordner wechseln. Es werden dann anschließend die Dateien und Unterordner dieses Ordners im mittleren Bereich angezeigt. Um zu einem übergeordneten Ordner zurück zu gelangen, bedient man sich des Menüs „Suchen in“, in dem man den entsprechenden Ordner auswählen kann.

Neben dem „Suchen in“-Menü sind noch fünf Graphik-Buttons angeordnet. Durch Anklicken des linken Buttons kommt man im Ordnerbaum eine Ebene höher. Durch Anklicken des zweiten Buttons von links gelangt man zur Wurzel des Ordnerbaumes. Mit dem mittleren Button kann man im aktuellen Ordner einen neuen Unterordner anlegen. Mit den beiden rechten Buttons kann man die Darstellung im mittleren Bereich verändern.

Möchte man einen Ordner oder eine Datei umbenennen, muss man im mittleren Bereich der Dateiauswahl-Dialogbox zweimal – mit Pause zwischendurch – auf den Namen des Ordners oder der Datei klicken. Die textuelle Darstellung des Namens wird dann zu einem Eingabefeld, in der man über die Tastatur den Namen verändern kann.

#### 4.1.10 Dateibaum

Ein *Dateibaum* repräsentiert die Ordner und Dateien des Dateisystems (siehe Abbildung 12).

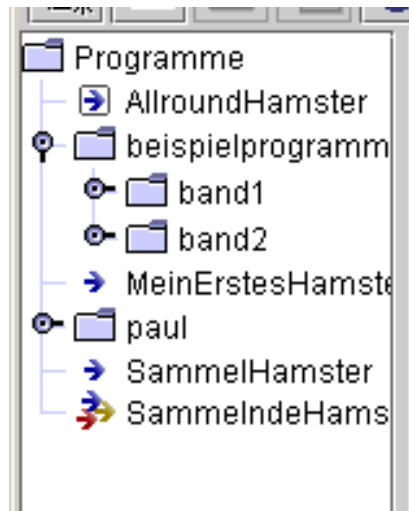


Abbildung 12: Dateibaum

Ordner und Dateien werden dabei durch unterschiedliche Symbole dargestellt, hinter denen die Namen erscheinen. Durch Anklicken des Zeigers vor einem Ordnersymbol kann man den Ordner öffnen und schließen. Bei einem geöffneten Ordner werden die darin enthaltenen Unterordner und Dateien (genauer ihre Namen) angezeigt.

Das Anklicken eines Dateinamens im Dateibaum entspricht im Hamster-Simulator dem Öffnen einer Datei. Der entsprechende Inhalt wird im Eingabebereich des Fensters dargestellt.

Den Ordnern und Dateien sind Popup-Menüs zugeordnet. Um diese zu öffnen, muss man zunächst den Ordner bzw. die Datei mit der Maus anklicken. Der Name wird dann durch einen blauen Balken hinterlegt. Anschließend muss man die rechte Maustaste drücken. Dann öffnet sich das Popup-Menü. Die Popup-Menüs enthalten bspw. Menüitems zum Löschen und Umbenennen des entsprechenden Ordners bzw. der entsprechenden Datei.

## 4.2 Verwalten und Editieren von Hamster-Programmen

Das Schreiben von Programmen bzw. genauer gesagt das Schreiben des Sourcecodes von Programmen bezeichnet man als *Editieren*. Im Hamster-Simulator dient das Editor-Fenster zum Editieren von Hamster-Programmen.

Schauen Sie sich das Editor-Fenster einmal an (siehe Abbildung 13).

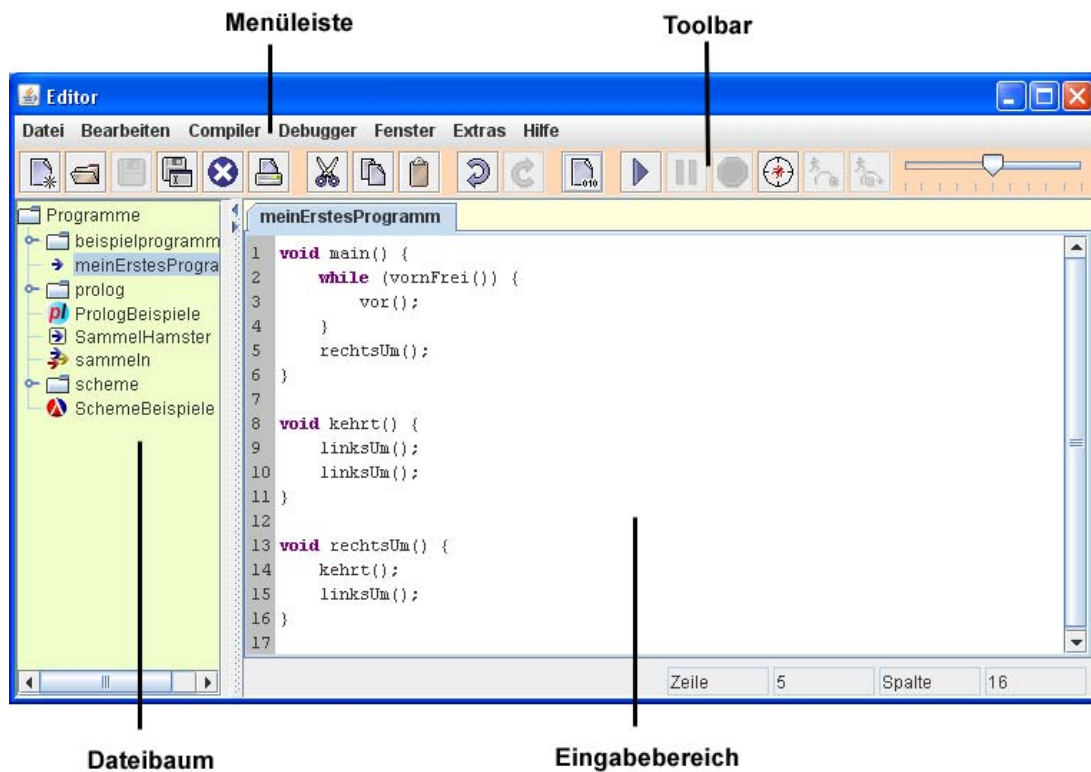


Abbildung 13: Editorfenster

Im linken Bereich sehen Sie den Dateibaum. Den Sourcecode von Hamster-Programmen müssen Sie in solchen Dateien abspeichern. Dateien sind dabei logische Speicherbehälter auf der Festplatte Ihres Computers. Der Dateibaum zeigt Ihnen an, welche Dateien bereits existieren. Neben Dateien enthält der Dateibaum auch Ordner. Ordner sind spezielle Ablagebereiche, um Dateien strukturiert abspeichern zu können. Den Hauptteil des Editor-Fensters nimmt der Eingabebereich ein. Hier können Sie Programme eintippen. Unten werden dabei die aktuelle Zeile und Spalte eingeblendet. Ganz oben im Editor-Fenster gibt es eine Menüleiste.

Für das Verwalten und Editieren von Programmen sind die beiden Menüs „Dateien“ und „Bearbeiten“ wichtig. Unterhalb der Menüleiste ist eine spezielle Toolbar zu sehen, über die Sie alle Funktionen der Menüs auch schneller erreichen und ausführen können. Schieben Sie einfach mal die Maus über die Buttons. Dann erscheint jeweils ein Tooltipp, der die Funktionalität des Buttons anzeigt (siehe auch Abbildung 14).



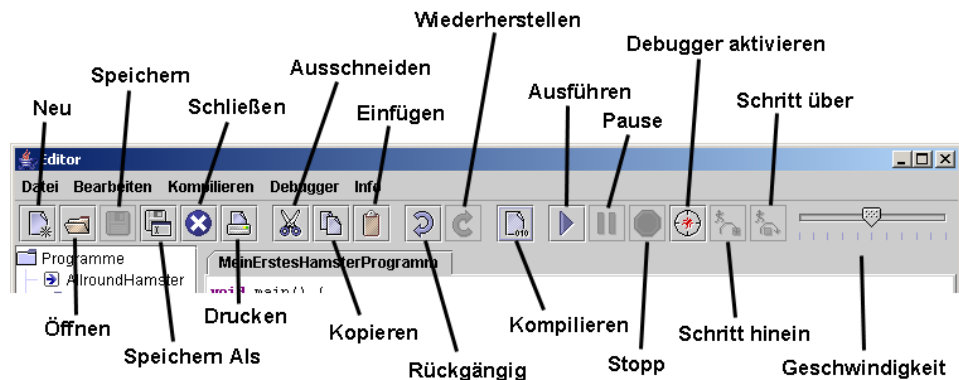


Abbildung 14: Toolbar des Editor-Fensters

### 4.2.1 Schreiben eines neuen Hamster-Programms

Wenn Sie ein neues Hamster-Programm schreiben möchten, klicken Sie bitte den „Neu“-Button an (erster Toolbar-Button von links). Es öffnet sich eine Dialogbox. In dieser Dialogbox müssen Sie auswählen, welchen Typ von Programm Sie schreiben möchten. Es existieren drei Alternativen:

- Imperatives Programm: Imperative Programme sind Programme, wie sie in Band 1 des Java-Hamster-Buches „Programmieren spielend gelernt“ eingeführt werden. Sie bestehen aus einer `main`-Funktion sowie weiteren Funktionen.
- Objektorientiertes Programm: Objektorientierte Programme sind Programme, wie sie in Band 2 des Java-Hamster-Buches „Objektorientierte Programmierung spielend gelernt“ eingeführt werden. Sie bestehen aus einer `main`-Funktion sowie Funktionen und Klassen.
- Klasse: Wollen Sie eine separate Klasse ohne `main`-Funktion schreiben, bspw. eine erweiterte Hamster-Klasse, müssen Sie diesen Typ von Programm auswählen. Wählen Sie den Programmtyp „Klasse“ auch, wenn Sie separate Interfaces definieren möchten.
- Scheme-Programm: hierzu siehe Abschnitt 7.

**Achtung:** Es ist nicht möglich, den Typ eines Programms nachträglich zu ändern!

Nach der Auswahl des Programmtyps drücken Sie bitte den OK-Button. Die Dialogbox schließt sich und oben im Eingabebereich des Editor-Fenster erscheint ein Karteireiter mit der Bezeichnung „NeuerHamster“ sowie einem Diskettensymbol. Das Diskettensymbol im Karteireiter deutet an, dass es sich um eine Datei handelt, in der Änderungen durchgeführt wurden, die noch nicht abgespeichert worden sind. Im Eingabebereich erscheint ein voreingestellter Programmrahmen.

Im Eingabebereich können Sie nun – wie in anderen Editoren auch – mit Hilfe der Tastatur Zeichen – also Sourcecode – eingeben. Der Editor unterstützt übrigens Syntax-Highlighting, d.h. normaler Sourcecode wird schwarz, Java-Schlüsselwörter werden violett, Kommentare grün und Stringlitterale blau dargestellt.

Es ist im Eingabebereich möglich, mehrere Dateien gleichzeitig zu bearbeiten. Für jede geöffnete Datei existiert ein Karteireiter (siehe auch Abbildung 15). Welche Datei sich aktuell im Eingabebereich in Bearbeitung befindet, erkennen Sie an dem etwas helleren Karteireiter. Durch Anklicken eines Karteireiters können Sie den entsprechenden Sourcecode in den Eingabebereich laden. Bei einem derartigen Wechsel der Datei wird nicht automatisch gespeichert.

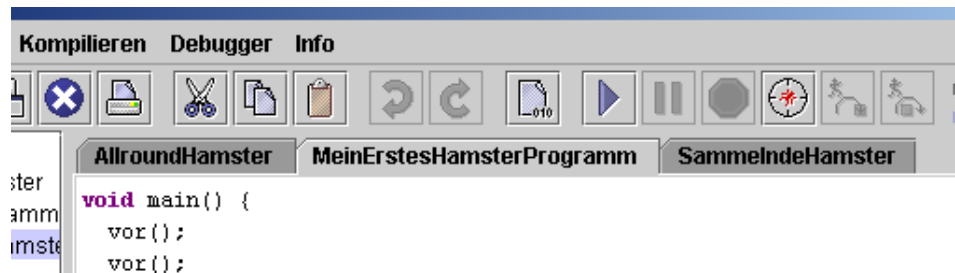


Abbildung 15: Karteireiter

Zum Abspeichern des von Ihnen editierten Textes klicken Sie bitte den „Speichern“-Button an (dritter Toolbar-Button von links). Handelt es sich um eine bereits angelegte Datei wird automatisch in diese Datei gespeichert. Handelt es sich um eine neu angelegte Datei, erscheint eine Dateiauswahl-Dialogbox. Hier müssen Sie nun den Namen Ihres Programms angeben. Achtung: Der Name muss ein gültiger Java-Bezeichner sein, ansonsten erscheint eine Fehlermeldung. Dateien mit Hamster-Programmen haben immer die Endung „.ham“. Die können Sie, müssen Sie aber nicht beim Dateinamen eingeben. Im letzteren Fall wird die Endung automatisch ergänzt.

Nachdem Sie ein neues Programm abgespeichert haben, erscheint der angegebene Name im Dateibaum des Editorfensters. Außerdem zeigt nun der Karteireiter anstelle von „NeuerHamster“ den Namen an. Das Diskettensymbol ist verschwunden, d.h. alle Änderungen sind gespeichert. Wenn an einer Datei keine Änderungen vorgenommen wurden, ist ein Speichern übrigens nicht möglich, der „Speichern“-Button ist ausgegraut.

Eine Alternative zum „Speichern“-Button stellt der „Speichern Als“-Button dar (vierter Toolbar-Button von links). Hiermit können Sie den eingegebenen bzw. geänderten Text in eine andere als die aktuelle Datei speichern. Dazu erscheint eine Dateiauswahl-Dialogbox, in der Sie den gewünschten Namen eingeben müssen.

Im Dateibaum ist übrigens auch der Typ eines Programms ersichtlich: Bei imperativen Programmen ist dem Namen ein einzelner blauer Pfeil vorangestellt, bei objektorientierten Programmen drei Pfeile in den Farben blau, gelb und rot und Klassen erkennen Sie an einem blauen Pfeil in einem Rechteck.

#### 4.2.2 Ändern eines existierenden Hamster-Programms

Um ein Hamster-Programm zu ändern, müssen Sie es zuvor öffnen, so dass der Sourcecode im Eingabebereich erscheint. Hierfür haben Sie mehrere Möglichkeiten. Zum einen können Sie den „Öffnen“-Button nutzen (zweiter Toolbar-Button von links). Es erscheint eine Dateiauswahl-Dialogbox, in der sie die entsprechende Datei auswählen können. Eine (schnellere) Alternative stellt das Anklicken des entsprechenden Namens im

Dateibaum dar. Im Eingabebereich können Sie nun die gewünschten Änderungen am Sourcecode vornehmen und anschließend mit Hilfe des „Speichern“-Buttons oder des „Speichern Als“-Buttons abspeichern.

#### **4.2.3 Löschen eines existierenden Hamster-Programms**

Möchten Sie ein Hamster-Programm wieder löschen, klicken Sie zunächst den entsprechenden Namen im Dateibaum an. Er wird durch ein blaues Rechteck hinterlegt. Klicken Sie anschließend die rechte Maustaste. Es erscheint ein Popup-Menü, in dem Sie das Menüitem „Löschen“ anklicken. Damit ist das Programm unwiderruflich gelöscht, der Name verschwindet aus dem Dateibaum und falls das Programm geöffnet war, verschwindet auch der Sourcecode inklusive Dateireiter aus dem Eingabebereich.

#### **4.2.4 Umbenennen eines existierenden Hamster-Programms**

Möchten Sie ein Hamster-Programm umbenennen, klicken Sie zunächst den entsprechenden Namen im Dateibaum an. Er wird durch ein blaues Rechteck hinterlegt. Klicken Sie anschließend die rechte Maustaste. Es erscheint ein Popup-Menü, in dem Sie das Menüitem „Umbenennen“ anklicken. Es öffnet sich eine Dialogbox, in der Sie den neuen Namen angeben können.

Wenn Sie eine Datei mit einem Hamster-Programm umbenannt haben, müssen Sie sie neu kompilieren!

#### **4.2.5 Verschieben eines existierenden Hamster-Programms in einen anderen Ordner**

Möchten Sie eine Datei mit einem Hamster-Programm (oder auch einen kompletten Ordner) in einen anderen Ordner verschieben, klicken Sie den entsprechenden Namen im Dateibaum an und verschieben Sie den Mauscursor bei gedrückter Maustaste über den Namen des Ordners, in den die Datei bzw. der Ordner verschoben werden soll. Der Name verschwindet aus dem vorherigen Ordner und erscheint im neuen Ordner, falls dieser geöffnet ist.

Wenn Sie eine Datei mit einem Hamster-Programm in einen anderen Ordner verschoben haben, müssen Sie sie neu kompilieren!

#### **4.2.6 Kopieren eines existierenden Hamster-Programms in einen anderen Ordner**

Möchten Sie ein Hamster-Programm in einen anderen Ordner kopieren, klicken Sie zunächst den entsprechenden Namen im Dateibaum an. Er wird durch ein blaues Rechteck hinterlegt. Klicken Sie anschließend die rechte Maustaste. Es erscheint ein Popup-Menü, in dem Sie das Menüitem „Kopieren“ anklicken. Klicken Sie nun im Dateibaum den Namen des Ordners an, in den die Datei kopiert werden soll. Klicken Sie danach die rechte Maustaste. Es erscheint ein Popup-Menü, in dem Sie das Menüitem „Einfügen“ anklicken.

Eine Alternative hierzu sieht folgendermaßen aus: Möchten Sie eine Datei mit einem Hamster-Programm (oder auch einen kompletten Ordner! ) in einen anderen Ordner

kopieren, klicken Sie den entsprechenden Namen im Dateibaum an und verschieben Sie den Mauscursor bei gedrückter Maustaste und gedrückter „Strg“-Taste Ihrer Tastatur über den Namen des Ordners, in den die Datei bzw. der Ordner verschoben werden soll.

Zumindest unter Windows funktioniert die Alternative übrigens auch in Kombination mit dem Betriebssystem, d.h. Sie können Dateien bspw. auf den Desktop kopieren und umgekehrt!

Wenn Sie eine Datei mit einem Hamster-Programm in einen anderen Ordner kopiert haben, müssen Sie die neue Datei noch kompilieren!

#### **4.2.7 Drucken eines Hamster-Programms**

Über den „Drucken“-Button (sechster Toolbar-Button von links) können Sie die aktuell im Eingabebereich geöffnete Datei drucken. Es öffnet sich eine Dialogbox, in der Sie die entsprechenden Druckeinstellungen vornehmen und den Druck starten können. Aktuell funktioniert das Drucken nur unter Windows.

#### **4.2.8 Schließen eines geöffneten Hamster-Programms**

Mit dem „Schließen“-Button (fünfter Toolbar-Button von links) können Sie die aktuell im Eingabebereich geöffnete Datei schließen, d.h. den Sourcecode inklusive Karteireiter aus dem Eingabebereich entfernen. Wenn Sie den Button anklicken und die aktuelle Datei noch nicht gespeicherte Änderungen enthält, wird über eine Dialogbox nachgefragt, ob diese Änderungen gespeichert werden sollen oder nicht.

#### **4.2.9 Editier-Funktionen**

Im Eingabebereich können Sie – wie bei anderen Editoren auch – über die Tastatur Zeichen eingeben bzw. wieder löschen. Darüber hinaus stellt der Editor ein paar weitere Funktionalitäten zur Verfügung, die über das „Bearbeiten“-Menü bzw. die entsprechenden Buttons in der Toolbar des Editor-Fensters aktiviert werden können.

- „Ausschneiden“-Button (siebter Toolbar-Button von links): Hiermit können Sie komplette Passagen des Eingabebereichs in einem Schritt löschen. Markieren Sie die zu löschende Passage mit der Maus und klicken Sie dann den Button an. Der markierte Text verschwindet.
- „Kopieren“-Button (achter Toolbar-Button von links): Hiermit können Sie komplette Passagen des Eingabebereichs in einen Zwischenpuffer kopieren. Markieren Sie die zu kopierende Passage mit der Maus und klicken Sie dann den Button an.
- „Einfügen“-Button (neunter Toolbar-Button von links): Hiermit können Sie den Inhalt des Zwischenpuffers an die aktuelle Cursorposition einfügen. Wählen Sie zunächst die entsprechende Position aus und klicken Sie dann den Button an. Der Text des Zwischenpuffers wird eingefügt.
- „Rückgängig“-Button (zehnter Toolbar-Button von links): Wenn Sie durchgeführte Änderungen des Sourcecode – aus welchem Grund auch immer – wieder rückgängig machen wollen, können Sie dies durch Anklicken des

Buttons bewirken. Das Rückgängigmachen bezieht sich dabei immer auf die aktuell im Eingabereich erscheinende Datei.

- „Wiederherstellen“-Button (elfter Toolbar-Button von links): Rückgängig gemachte Änderungen können Sie mit Hilfe dieses Buttons wieder herstellen.

Die Funktionalitäten „Kopieren“ und „Einfügen“ funktionieren übrigens auch über einzelne Programme hinaus. Es ist sogar möglich, mit Hilfe der Betriebssystem-Kopieren-Funktion Text aus anderen Programmen (bspw. Microsoft Word) zu kopieren und hier einzufügen.

#### **4.2.10 Verwaltung von Ordnern**

Um Funktionen auf einem Ordner durchzuführen, klicken Sie zunächst den entsprechenden Ordner im Dateibaum an. Der Name des Ordners wird dadurch durch ein blaues Rechteck hinterlegt. Klicken Sie anschließend die rechte Maustaste, so dass sich das Popup-Menü des Ordners öffnet. Im Popup-Menü haben Sie nun folgende Funktionalitäten zur Verwaltung von Ordnern zur Verfügung:

- Menüitem „Neuer Ordner“: Durch Anklicken dieses Items können Sie einen neuen Unterordner anlegen. Es öffnet sich eine Dialogbox, in der Sie den Namen des Unterordners angeben müssen.
- Menüitem „Löschen“: Durch Anklicken dieses Items können Sie den entsprechenden Ordner löschen. Achtung: Es werden automatisch auch alle Dateien und Unterordner des Ordners unwiderruflich gelöscht!
- Menüitem „Einfügen“: Wenn Sie zuvor eine Datei mit dessen Popup-Menüitem „Kopieren“ kopiert haben, können Sie durch Anklicken dieses Items die entsprechende Datei in den aktuellen Ordner einfügen, d.h. die Datei wurde kopiert.
- Menüitem „Umbenennen“: Mit Hilfe dieses Items können Sie den Namen des Ordners ändern. Es öffnet sich eine Dialogbox, in der Sie den gewünschten Namen eingeben können.

Möchten Sie einen Ordner samt aller seiner Unterordner und Dateien in einen anderen Ordner verschieben, klicken Sie den entsprechenden Ordner im Dateibaum an und verschieben Sie den Mauscursor bei gedrückter Maustaste über den Namen des Ordners, in den der Ordner verschoben werden soll. Der Name verschwindet aus dem vorherigen Ordner und erscheint im neuen Ordner, falls dieser geöffnet ist. Führen Sie die Funktion bei gedrückter „Strg“-Taste durch, wird der Ordner samt Inhalt kopiert anstelle von verschoben.

### **4.3 Compilieren von Hamster-Programmen**

Beim Kompilieren werden Programme – genauer gesagt der Sourcecode – auf ihre (syntaktische) Korrektheit überprüft und im Erfolgsfall ausführbare Programme erzeugt. Zum Kompilieren von Programmen dient im Editor-Fenster das „Kompilieren“-Menü.

### 4.3.1 Compilieren

Wenn Sie das „Kompilieren“-Menüitem im „Kompilieren“-Menü oder in der Toolbar des Editor-Fensters den „Kompilieren“-Button (zwölfter Toolbar-Button von links) anklicken, wird das Programm, das gerade im Eingabebereich des Editor-Fensters sichtbar ist, kompiliert. Wenn Sie zuvor Änderungen am Sourcecode vorgenommen und noch nicht abgespeichert haben, werden Sie durch eine Dialogbox gefragt, ob das Programm vor dem Kompilieren gespeichert werden soll oder nicht.

Wenn Ihr Programm korrekt ist, erscheint nach ein paar Sekunden eine Dialogbox mit einer entsprechenden Meldung. Es wurde ein (neues) ausführbares Programm erzeugt.

### 4.3.2 Beseitigen von Fehlern

Wenn Ihr Programm Fehler enthält, öffnet sich unterhalb des Eingabebereichs ein neuer Bereich, der die Fehlermeldungen des Compilers anzeigt (siehe Abbildung 16). Es wurde kein (neues) ausführbares Programm erzeugt! Jede Fehlermeldung erscheint in einer eigenen Zeile. Jede Zeile enthält den (wahrscheinlichen) Fehler, die Anweisung, die den Fehler enthält, die Zeile der Anweisung im Programm und den Dateinamen. Wenn Sie eine Fehlermeldung anklicken, wird die entsprechende Anweisung im Eingabebereich blau markiert und der Mauscursor an die entsprechende Stelle gesetzt. Sie müssen nun die einzelnen Fehler beseitigen und dann erneut speichern und kompilieren, bis Ihr Programm keine Fehler mehr enthält. Der Fehlermeldungsbereich schließt sich dann automatisch wieder.

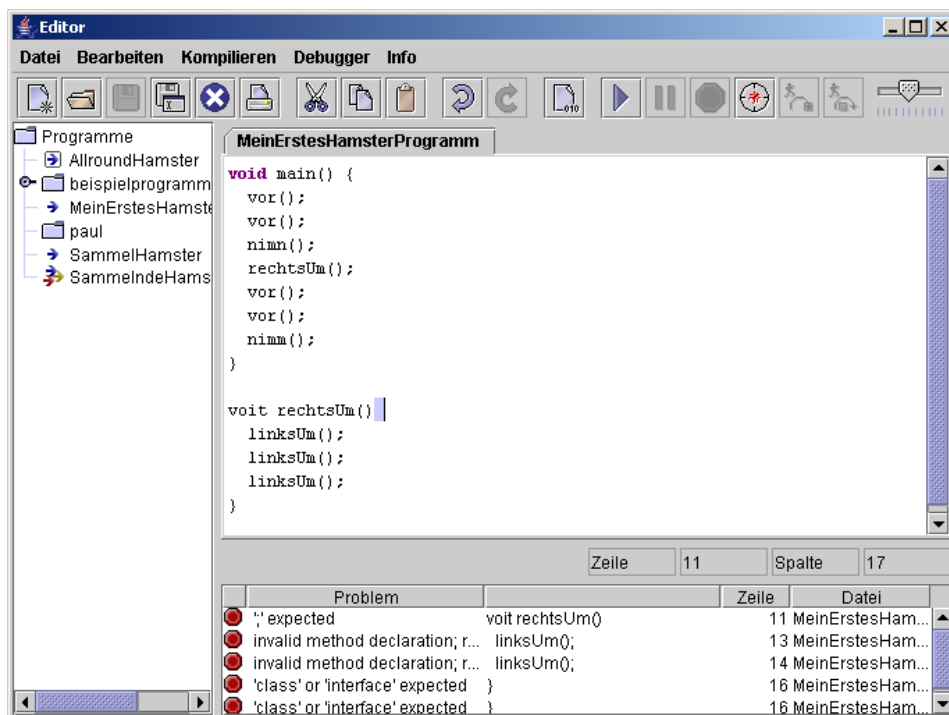


Abbildung 16: Fehlermeldungen

Achtung: Die Interpretation von Fehlermeldungen, die der Compiler ausgibt, ist nicht trivial. Die Meldungen sind nicht immer besonders präzise und oft auch irreführend. Häufig gibt der Compiler mehrere Fehlermeldungen aus, obwohl es sich nur um einen einzelnen Fehler handelt. Deshalb beherzigen Sie gerade am Anfang folgende Hinweise: Arbeiten Sie die Fehlermeldungen immer von oben nach unten ab. Wenn der Compiler eine große Menge von Fehlermeldungen liefert, korrigieren Sie zunächst nur eine Teilmenge und speichern und kompilieren Sie danach erneut. Bauen Sie – gerade als Programmieranfänger – auch mal absichtlich Fehler in Ihre Programme ein und schauen Sie sich dann die Fehlermeldungen des Compilers an.

### 4.3.3 Setzen des CLASSPATH

Als zweites Menüitem enthält das „Kompilieren“-Menü ein Menüitem mit der Bezeichnung „CLASSPATH setzen“. Was es damit auf sich hat, entnehmen Sie bitte Kapitel 14 von Band 2 des Java-Hamster-Buches. Als Anfänger müssen Sie sich hiermit nicht auseinandersetzen.

## 4.4 Verwalten und Gestalten von Hamster-Territorien

Das Hamster-Territorium befindet sich im Simulation-Fenster. Es umfasst standardmäßig 10 Reihen und 10 Spalten. Der Standard-Hamster – das ist der blaue Hamster – steht auf der Kachel ganz oben links, also der Kachel mit den Koordinaten (0/0). Er hat 0 Körner im Maul und schaut nach Osten.

Oberhalb des Hamster-Territoriums befindet sich eine Toolbar mit Graphik-Buttons (siehe auch Abbildung 17). Die ersten drei Buttons von links dienen zum Verwalten von Hamster-Territorien. Mit den Buttons vier bis neun kann das Hamster-Territorium umgestaltet werden. Mit dem zehnten und elften Button lässt sich das Erscheinungsbild des Territoriums verändern. Die restlichen Buttons sowie der Schieberegler haben beim Ausführen von Hamster-Programmen eine Bedeutung.

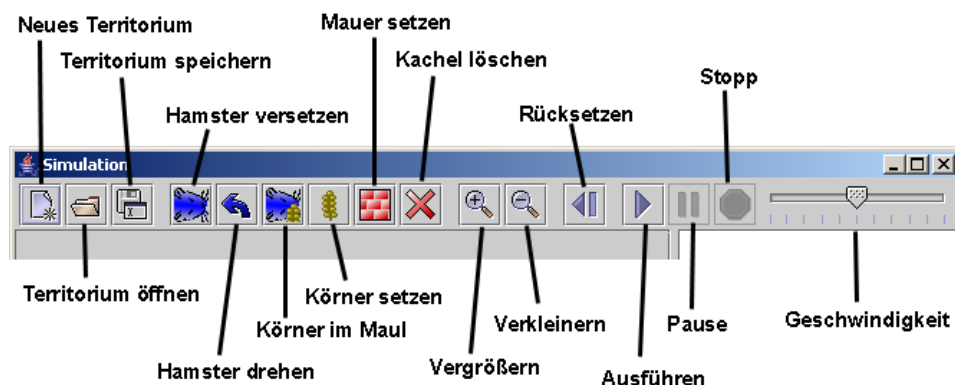


Abbildung 17: Toolbar des Simulation-Fensters

#### **4.4.1 Verändern der Größe des Hamster-Territoriums**

Durch Anklicken des „Neues Territorium“-Buttons (erster Toolbar-Button von links) können Sie die Größe des Territoriums verändern. Es öffnet sich eine Dialogbox mit zwei Eingabefelder, in denen Sie die gewünschte Reihen- und Spaltenanzahl eingeben können. Nach Drücken des OK-Buttons schließt sich die Dialogbox und das Territorium erscheint in der angegebenen Größe. Aber Achtung: Nach dem Ändern der Größe des Territoriums sind alle Kacheln leer und der Standard-Hamster nimmt seinen Standardzustand ein.

#### **4.4.2 Platzieren des Standard-Hamsters im Hamster-Territorium**

Um den Standard-Hamster im Hamster-Territorium auf eine andere Kachel zu platzieren, müssen Sie zunächst im Simulation-Fenster in der Toolbar den „Hamster versetzen“-Button (vierter Toolbar-Button von links) anklicken. Dadurch aktivieren Sie die Hamster-Versetzen-Funktion. Sie erkennen dies daran, dass der Hintergrund des Buttons nun dunkler erscheint. Solange die Funktion aktiviert ist, können Sie nun durch Anklicken einer Kachel den Standard-Hamster auf diese Kachel platzieren.

Eine Deaktivierung der Hamster-Versetzen-Funktion ist durch Anklicken des „Körner setzen“- , „Mauer setzen“- oder „Kachel löschen“-Buttons und der damit verbundenen Aktivierung der entsprechenden Funktion möglich.

#### **4.4.3 Setzen der Blickrichtung des Standard-Hamsters**

Um die Blickrichtung des Standard-Hamsters zu ändern, klicken Sie bitte den „Hamster drehen“-Button (fünfter Toolbar-Button von links) an. Bei jedem Klick auf diesen Button dreht sich der Standard-Hamster um 90 Grad linksum.

#### **4.4.4 Abfragen und Festlegen der Körneranzahl im Maul des Standard-Hamsters**

Um die Anzahl an Körnern im Maul des Standard-Hamster festzulegen, klicken Sie bitte den „Körner im Maul“-Button (sechster Toolbar-Button von links) an. Es öffnet sich eine Dialogbox mit einem Eingabefeld. In diesem Eingabefeld erscheint die aktuelle Anzahl an Körnern im Maul des Standard-Hamsters. Sie können nun in das Eingabefeld die gewünschte Anzahl eingeben. Klicken Sie anschließend den OK-Button, um die Eingabe zu bestätigen. Die Dialogbox schließt sich und der Standard-Hamster hat die eingegebene Anzahl an Körnern im Maul.

#### **4.4.5 Platzieren von Körnern auf Kacheln des Hamster-Territorium**

Um auf einzelnen Kacheln des Hamster-Territoriums Körner zu platzieren, müssen Sie zunächst im Simulation-Fenster in der Toolbar den „Körner setzen“-Button (siebter Toolbar-Button von links) anklicken. Dadurch aktivieren Sie die Körner-Setzen-Funktion. Sie erkennen dies daran, dass der Hintergrund des Buttons nun dunkler erscheint. Solange die Funktion aktiviert ist, können Sie nun durch Anklicken einer Kachel die Körneranzahl auf dieser Kachel festlegen. Nach Anklicken der entsprechenden Kachel erscheint eine Dialogbox mit einem Eingabefeld, in das Sie die gewünschte Anzahl an



Körnern eingeben können. Nach Klicken des OK-Buttons schließt sich die Dialogbox und die Körner erscheinen auf der Kachel.

Dabei gilt: Bei einer Körneranzahl bis 12 werden entsprechend viele Körner auf der Kachel angezeigt. Bei einer Körneranzahl größer als 12 werden immer nur 12 Körner angezeigt. Wie viele Körner tatsächlich auf der Kachel liegen, können Sie ermitteln, wenn Sie den Mauscursor auf die entsprechende Kachel verschieben. Es erscheint ein Tooltipp, in dem die Koordinaten der Kachel sowie die genaue Körneranzahl ersichtlich sind.

Es ist auch möglich, die Anzahl an Körnern auf mehreren Kacheln gleichzeitig festzulegen. Klicken Sie dazu die Maus auf einer der Kacheln und ziehen Sie den Mauscursor bei gedrückter Maustaste über die anderen Kacheln. Alle betroffenen Kacheln werden zunächst durch ein Korn in der Mitte markiert. Wenn Sie dann die Maustaste loslassen, erscheint die Dialogbox zur Eingabe der Körneranzahl. Die Anzahl an Körnern, die sie jetzt eingeben, wird dann auf allen markierten Kacheln abgelegt.

Eine Deaktivierung der Körner-Setzen-Funktion ist durch Anklicken des „Hamster versetzen“- , „Mauer setzen“- oder „Kachel löschen“-Buttons und der damit verbundenen Aktivierung der entsprechenden Funktion möglich.

#### **4.4.6 Platzieren von Mauern auf Kacheln des Hamster-Territorium**

Um auf einzelnen Kacheln des Hamster-Territoriums Mauern zu platzieren, müssen Sie zunächst im Simulation-Fenster in der Toolbar den „Mauer setzen“-Button (achter Toolbar-Button von links) anklicken. Dadurch aktivieren Sie die Mauer-Setzen-Funktion. Sie erkennen dies daran, dass der Hintergrund des Buttons nun dunkler erscheint. Solange die Funktion aktiviert ist, können Sie nun durch Anklicken einer Kachel auf dieser Kachel eine Mauer platzieren.

Dabei gilt: Es ist nicht möglich, auf einer Kachel, auf der sich aktuell ein Hamster befindet, eine Mauer zu platzieren. Liegen auf einer angeklickten Kachel Körner, werden diese gelöscht.

Es ist auch möglich, auf mehreren Kacheln gleichzeitig Mauern zu platzieren. Klicken Sie dazu die Maus auf einer der Kacheln und ziehen Sie den Mauscursor bei gedrückter Maustaste über die anderen Kacheln. Auf allen Kacheln werden unmittelbar Mauern gesetzt.

Eine Deaktivierung der Mauer-Setzen-Funktion ist durch Anklicken des „Hamster versetzen“- , „Körner setzen“ oder „Kachel löschen“-Buttons und der damit verbundenen Aktivierung der entsprechenden Funktion möglich.

#### **4.4.7 Löschen von Kacheln des Hamster-Territorium**

Um einzelne Kacheln des Hamster-Territoriums zu löschen, d.h. gegebenenfalls vorhandene Körner bzw. Mauern zu entfernen, müssen Sie zunächst im Simulation-Fenster in der Toolbar den „Kachel löschen“-Button (neunter Toolbar-Button von links) anklicken. Dadurch aktivieren Sie die Kachel-Löschen-Funktion. Sie erkennen dies daran, dass der Hintergrund des Buttons nun dunkler erscheint. Solange die Funktion aktiviert ist, können Sie nun durch Anklicken einer Kachel diese Kachel löschen.

Es ist auch möglich, mehrere Kacheln gleichzeitig zu löschen. Klicken Sie dazu die Maus auf einer der Kacheln und ziehen Sie den Mauscursor bei gedrückter Maustaste über die anderen Kacheln. Alle Kacheln, die nach Loslassen der Maustaste gelöscht werden, werden durch ein rotes X gekennzeichnet.

Eine Deaktivierung der Kachel-Löschen-Funktion ist durch Anklicken des „Hamster versetzen“- „Körner setzen“ oder „Mauer setzen“-Buttons und der damit verbundenen Aktivierung der entsprechenden Funktion möglich.

#### **4.4.8 Abspeichern eines Hamster-Territoriums**

Sie können einmal gestaltete Hamster-Territorien in einer Datei abspeichern und später wieder laden. Zum Abspeichern des aktuellen Territoriums drücken Sie bitte den „Territorium speichern“-Button (dritter Toolbar-Button von links). Es öffnet sich eine Dateiauswahl-Dialogbox. Hierin können Sie den Ordner auswählen und den Namen einer Datei eingeben, in die das aktuelle Territorium gespeichert werden soll. Namen von Dateien mit Hamster-Territorien bekommen übrigens automatisch die Endung „.ter“.

#### **4.4.9 Wiederherstellen eines abgespeicherten Hamster-Territoriums**

Abgespeicherte Hamster-Territorien können mit dem „Territorium öffnen“-Button (zweiter Toolbar-Button von links) wieder geladen werden. Klicken Sie hierzu den Button. Es erscheint eine Dateiauswahl-Dialogbox, in der Sie die zu ladende Datei auswählen können. Nach dem Anklicken des OK-Buttons schließt sich die Dialogbox und das entsprechende Hamster-Territorium ist wiederhergestellt.

Achtung: Der Zustand des Hamster-Territoriums, der vor dem Ausführen der Territorium-Öffnen-Funktion Gültigkeit hatte, ist unwiderruflich verloren. Speichern Sie ihn daher gegebenenfalls vorher ab.

#### **4.4.10 Umbenennen eines abgespeicherten Hamster-Territoriums**

Um eine Datei mit einem abgespeicherten Hamster-Territorium umzubenennen, klicken Sie bitte den „Territorium öffnen“-Button an (zweiter Toolbar-Button von links). Es öffnet sich eine Dateiauswahl-Dialogbox. Klicken Sie dann im mittleren Bereich der Dateiauswahl-Dialogbox zweimal – mit Pause zwischendurch – auf den Namen der Datei. Die textuelle Darstellung des Namens wird dann zu einem Eingabefeld, in der man über die Tastatur den Namen verändern kann. Klicken Sie anschließend den OK-Button, wenn Sie die umbenannte Datei gleich öffnen wollen, oder den Abbrechen-Button, wenn Sie nur eine Umbenennung vornehmen wollen.

#### **4.4.11 Löschen und Verschieben einer Datei mit einem Hamster-Territorium in einen anderen Ordner**

Das Löschen und Verschieben einer Datei mit einem abgespeicherten Territorium ist nur mittels des Betriebssystems möglich (siehe Abschnitt 4.8).

#### 4.4.12 Verändern der Größendarstellung des Hamster-Territoriums

Durch Anklicken des „Vergrößern“- bzw. „Verkleinern“-Buttons (zehnter und elfter Toolbar-Button von links) können Sie die Darstellung des Hamster-Territoriums manipulieren. Bei jedem Klick auf einen dieser Buttons erscheint es vergrößert bzw. verkleinert. Eine Verkleinerung ist dabei nur bis zu einem bestimmten Maß möglich, so dass man auch noch etwas erkennen kann.

### 4.5 Ausführen von Hamster-Programmen

Ausgeführt werden Hamster-Programme im Simulation-Fenster. Zur Steuerung dienen dabei die Graphik-Buttons sowie der Schieberegler im rechten Teil der Toolbar oberhalb des Hamster-Territoriums (siehe auch Abbildung 18). Alle Steuerelemente befinden sich zusätzlich auch im Editor-Fenster.

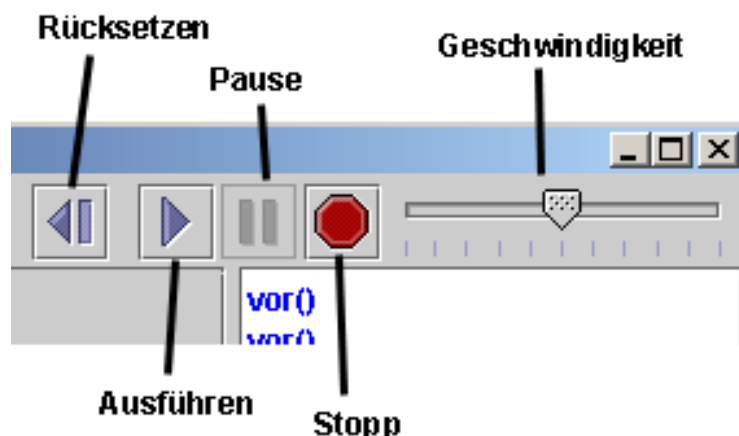


Abbildung 18: Toolbar des Simulation-Fensters

#### 4.5.1 Starten eines Hamster-Programms

Bevor ein Hamster-Programm ausgeführt werden kann, muss es im Editor-Fenster geöffnet werden und sein Sourcecode im Eingabebereich des Editor-Fensters erscheinen. Außerdem muss es natürlich erfolgreich kompiliert worden sein. Gestartet werden kann dieses Programm dann durch Anklicken des „Ausführen“-Buttons (dritter Toolbar-Button des Simulation-Fensters von rechts).

Es können nur solche Programme ausgeführt werden, die in Dateien vom Typ „imperatives Programm“ oder „objektorientiertes Programm“ abgespeichert sind, also keine separaten Klassen. Befindet sich aktuell eine Datei mit einer separaten Klasse im Eingabebereich, erscheint der „Ausführen“-Button auch ausgegraut und kann nicht angeklickt werden.

Nach dem Starten eines Hamster-Programms werden die Hamster im Hamster-Territorium aktiv und tun das, was das Programm ihnen befiehlt. Während des Ausführens eines Hamster-Programms wird der Eingabebereich im Editor-Fenster ausgegraut, d.h. es können während der Ausführung eines Programms keine Änderungen am Sourcecode durchgeführt werden.

Wenn Sie vor dem Anklicken des „Ausführen“-Buttons die Datei im Eingabebereich geändert, aber noch nicht gespeichert und/oder kompiliert haben, werden Sie über entsprechende Dialogboxen gefragt, ob das Abspeichern und Kompilieren noch vor dem Ausführen erledigt werden soll oder nicht.

#### **4.5.2 Stoppen eines Hamster-Programms**

Die Ausführung eines Hamster-Programms kann durch Anklicken des „Stopp“-Buttons (erster Toolbar-Button des Simulation-Fensters von rechts) jederzeit abgebrochen werden.

#### **4.5.3 Pausieren eines Hamster-Programms**

Möchten Sie ein in Ausführung befindliches Programm anhalten, können Sie dies durch Anklicken des „Pause“-Buttons (zweiter Toolbar-Button des Simulation-Fensters von rechts) tun. Wenn Sie anschließend auf den „Ausführen“-Button klicken, wird das Programm fortgeführt.

#### **4.5.4 Während der Ausführung eines Hamster-Programms**

Der Standard-Hamster wird immer in blau dargestellt. Wenn Sie (in objektorientierten Programmen) weitere Hamster erzeugen, erhalten diese zur Unterscheidung andere Farben. Der erste erzeugte Hamster ist rot, der zweite grün, der dritte gelb, der vierte pink und der fünfte violett. Alle weiteren Hamster haben ein graues Erscheinungsbild.

Führt ein Hamster einen Hamster-Befehl aus, wird dieser im Ausgabebereich des Simulation-Fensters ausgegeben. Zur Unterscheidung, welcher Hamster den Befehl ausgeführt hat, erfolgt die Ausgabe in der Farbe des entsprechenden Hamsters.

Führt in objektorientierten Programmen ein Hamster einen `schreib`- oder `lies`-Befehl aus, öffnet sich eine Dialogbox. Der Ausgabe- bzw. Aufforderungsstring erscheint darin wiederum in der Farbe des entsprechenden Hamsters. Beim Befehl `schreib` pausiert das Programm so lange, bis der Benutzer den OK-Button der Dialogbox angeklickt hat. Bei einem `liesZahl`- oder `liesZeichenkette`-Befehl kann der Benutzer eine Zahl bzw. eine Zeichenkette eingeben und muss anschließend den OK-Button drücken. Dann wird der eingegebene Wert an das Programm weitergegeben. Gibt der Benutzer bei einem `liesZahl`-Befehl keine gültige Zahl ein (bspw. `a2d`), liefert der Befehl den Wert 0.

Normalerweise sind Dialogboxen exklusive Fenster, die es, wenn sie geöffnet sind, nicht erlauben, in anderen Fenster Mausklicks zu tätigen. Die Dialogboxen der `schreib`- und `lies`-Befehle sind in diesem Sinne keine „richtigen“ Dialogboxen. Während sie geöffnet sind, können auch andere Funktionalitäten (bspw. Abbruch der Programmausführung durch Anklicken des „Stopp“-Buttons) ausgeführt werden.

Treten bei der Ausführung eines Programms Laufzeitfehler auf (in objektorientierten Programmen entsprechen diese dem Werfen von Exceptions), z.B. wenn ein Hamster gegen eine Mauer donnert, wird eine Dialogbox geöffnet, die eine entsprechende Fehlermeldung enthält. Nach dem Anklicken des OK-Buttons in der Dialogbox wird das Hamster-Programm beendet.

#### **4.5.5 Einstellen der Geschwindigkeit**

Mit dem Schieberegler ganz rechts in der Toolbar des Simulation-Fenster und der Toolbar des Editor-Fensters können Sie die Geschwindigkeit der Programmausführung beeinflussen. Je weiter links der Regler steht, desto langsamer wird das Programm ausgeführt. Je weiter Sie den Regler nach rechts verschieben, umso schneller flitzen die Hamster durchs Territorium.

#### **4.5.6 Wiederherstellen eines Hamster-Territoriums**

Beim Testen eines Programms recht hilfreich ist der „Rücksetzen“-Button (vierter Toolbar-Button des Simulation-Fensters von rechts). Sein Anklicken bewirkt, dass das Hamster-Territorium in den Zustand zurückversetzt wird, den es vor dem letzten Start eines Programms inne hatte. Außerdem verschwinden während der Ausführung eines objektorientierten Programms erzeugte Hamster aus dem Territorium.

#### **4.5.7 Mögliche Fehlerquellen**

Im Folgenden werden die häufigsten Fehlerquellen genannt, die bei der Ausführung eines Programms auftreten können:

- Sie haben ein neues Programm geschrieben und auch abgespeichert, aber nicht kompiliert oder der Compiler hat Fehler gemeldet. In diesem Fall erscheint beim Starten des Programms die Laufzeitfehler-Dialogbox mit der Fehlermeldung „ClassNotFoundException“.
- Sie haben den Sourcecode eines Programms verändert, eventuell auch noch abgespeichert, aber nicht neu kompiliert. Oder Sie haben zwar kompiliert, der Compiler hat jedoch Fehlermeldungen geliefert. In diesem Fall wird das alte Programm ausgeführt.
- Bei objektorientierten Programmen müssen Sie darauf achten, dass, wenn Sie Werte von Konstanten im Sourcecode ändern, alle Dateien, die diese Konstanten benutzen, neu kompiliert werden müssen. Ansonsten arbeiten die nicht kompilierten Dateien noch mit den alten Werten.
- Wenn in einem objektorientierten Programm eine Klasse X eine Klasse Y referenziert und umgekehrt, müssen Sie zunächst X kompilieren, dann Y und anschließend nochmal X.

## 4.6 Debuggen von Hamster-Programmen

*Debugger* sind Hilfsmittel zum Testen von Programmen. Sie erlauben es, während der Programmausführung den Zustand des Programms zu beobachten und gegebenenfalls sogar interaktiv zu ändern. Damit sind Debugger sehr hilfreich, wenn es um das Entdecken von Laufzeitfehlern und logischen Programmfehlern geht.

Der Debugger des Hamster-Simulator ermöglicht während der Ausführung eines Hamster-Programms das Beobachten des Programmzustands. Sie können sich während der Ausführung eines Hamster-Programms anzeigen lassen, welche Anweisung des Sourcecodes gerade ausgeführt wird und welche Werte die Variablen aktuell speichern. Die interaktive Änderung von Variablenwerten wird aktuell nicht unterstützt.

Der Debugger ist im Hamster-Simulator dem Editor-Fenster zugeordnet. Seine Funktionen sind eng mit den Funktionen zur Programmausführung verknüpft. Sie finden die Funktionen im Menü „Debugger“. Es bietet sich jedoch an, die entsprechenden Graphik-Buttons der Toolbar zu verwenden. Neben dem „Ausführen“- , dem „Pause“- und dem „Stopp“-Button gehören die drei rechten Buttons „Debugger aktivieren“, „Schritt hinein“ und „Schritt über“ zu den Debugger-Funktionen (siehe auch Abbildung 19).

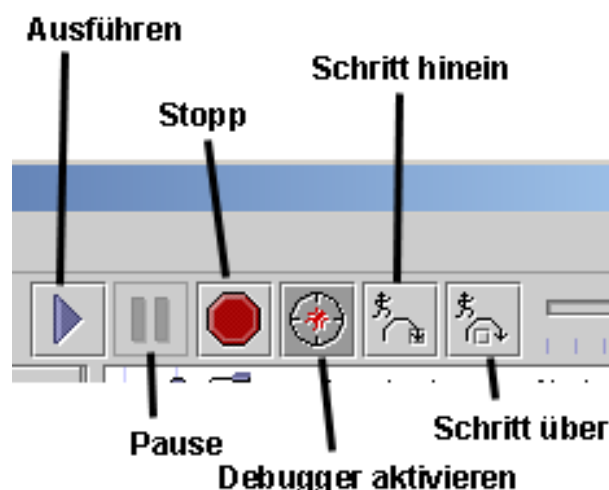


Abbildung 19: Toolbar des Editor-Fensters

### 4.6.1 Aktivieren bzw. deaktivieren des Debuggers

Sie können den Debugger durch Anklicken des „Debugger aktivieren“-Buttons in der Toolbar (dritter Toolbar-Button von rechts) aktivieren bzw. durch erneutes Anklicken wieder deaktivieren. Der Debugger ist aktiviert, wenn der Hintergrund des Buttons dunkler erscheint.

Das Aktivieren bzw. Deaktivieren des Debuggers ist vor, aber auch noch während der Ausführung eines Programms möglich.

## 4.6.2 Beobachten der Programmausführung

Wenn der Debugger aktiviert ist und Sie ein Programm mit dem „Ausführen“-Button starten, öffnen sich im Editor-Fenster oberhalb des Eingabebereiches zwei neue Bereiche. Der linke dieser beiden Bereiche heißt *Funktionen-Bereich*, der rechte *Variablen-Bereich* (siehe auch Abbildung 20).

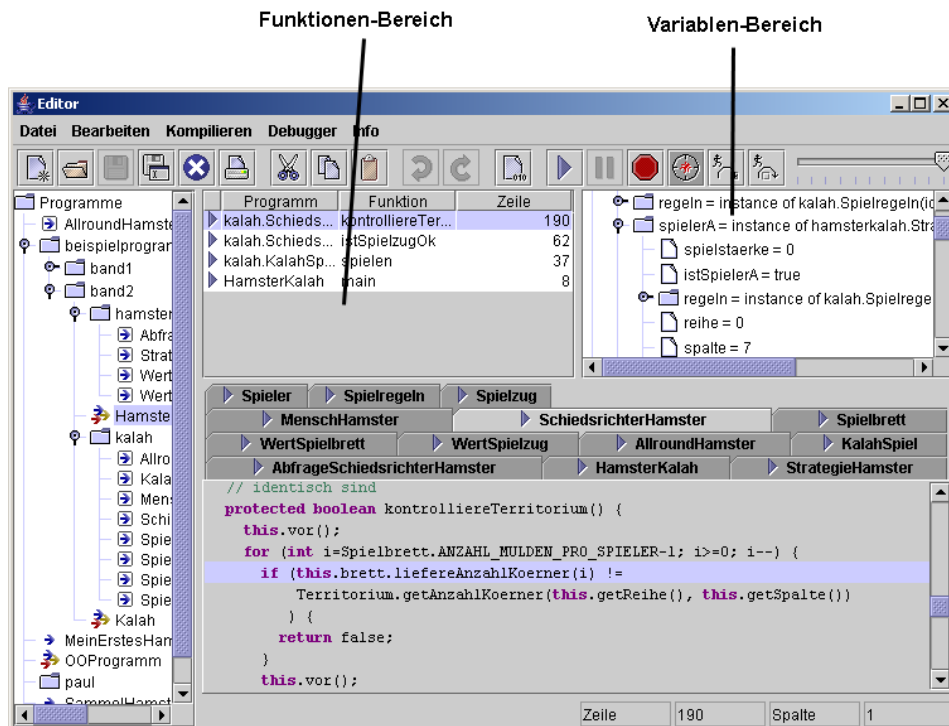


Abbildung 20: Debugging-Fenster

Im Funktionen-Bereich werden Informationen zu den aktiven Funktionen bzw. Methoden angezeigt, und zwar jeweils von links nach rechts der Programm- bzw. Klassenname, der Funktions- bzw. Methodenname und die aktuelle Zeile. Ganz oben erscheint die aktuell aktive Funktion, darunter gegebenenfalls die Funktion, die diese Funktion aufgerufen hat, usw. Ganz unten steht also immer die `main`-Funktion.

Im Variablen-Bereich werden die aktiven Variablen und ihre aktuellen Werte angezeigt. Die Darstellung erfolgt dabei analog zu einem Dateibaum, d.h. bei komplexen Variablen, wie Objekten oder Arrays, können Sie durch Anklicken des Symbols vor dem Variablennamen die Attribute bzw. Komponenten einsehen.

Im Eingabebereich selbst wird jeweils der Sourcecode eingeblendet, der gerade ausgeführt wird. Die Zeile mit der gerade aktiven Anweisung wird durch einen blauen Balken hinterlegt. Bei (objektorientierten) Programmen, die aus mehreren Dateien bestehen, werden gegebenenfalls Dateien automatisch geöffnet.

Auch während der Debugger aktiviert ist, können Sie die Programmausführung durch Anklicken des „Pause“-Buttons anhalten und durch anschließendes Anklicken des „Ausführen“-Buttons wieder fortfahren lassen. Auch die Geschwindigkeit der

Programmausführung lässt sich mit dem Schieberegler anpassen. Bei Anklicken des „Stopp“-Buttons wird das Programm abgebrochen und der Funktionen- und Variablen-Bereich verschwinden.

### 4.6.3 Schrittweise Programmausführung

Mit den beiden Buttons „Schritt hinein“ (zweiter Toolbar-Button von rechts) und „Schritt über“ (erster Toolbar-Button von rechts) ist es möglich, ein Programm schrittweise, d.h. Anweisung für Anweisung auszuführen. Immer, wenn Sie einen der beiden Buttons anklicken, wird die nächste Anweisung – und nur die! – ausgeführt.

Die beiden Buttons unterscheiden sich genau dann, wenn die nächste Anweisung ein Prozedur-, Funktions- oder Methodenaufruf ist. Das Anklicken des „Schritt hinein“-Buttons bewirkt in diesem Fall, dass in den entsprechenden Rumpf der Prozedur, Funktion oder Methode verzweigt wird, so dass man die dortigen Anweisungen ebenfalls Schritt für Schritt weiter ausführen kann. Beim Anklicken des „Schritt über“-Buttons wird die komplette Prozedur, Funktion oder Methode in einem Schritt ausgeführt.

Beachten Sie bitte, dass man die Ausführung eines Programms auch mit dem „Schritt hinein“-Button starten kann. Ist der Debugger aktiviert, führt ein Anklicken des „Schritt hinein“-Buttons dazu, dass in die `main`-Funktion gesprungen wird. Von hieraus können Sie dann ein komplettes Programm schrittweise ausführen.

Sie können die „Schritt hinein“- und „Schritt über“-Buttons auch nutzen, wenn der Debugger aktiv ist und die Programmausführung durch Anklicken des „Pause“-Buttons angehalten wurde. Wenn Sie also die Programmausführung erst ab einer bestimmten Stelle beobachten möchten, können Sie das Programm zunächst einfach starten, dann anhalten, den Debugger aktivieren und dann schrittweise weiter ausführen.

Wenn Sie irgendwann ein Programm nicht weiter schrittweise ausführen möchten, können Sie durch Anklicken des Ausführen-Buttons die automatische Programmausführung wieder aktivieren.

## 4.7 3D-Simulationsfenster und Sound

Seit der Version 2.5 des Hamster-Simulators ist es möglich, sich die Aktivitäten der Hamster (zusätzlich) in einem 3-dimensionalen Territorium anzuschauen. Herzlichen Dank an Christoph Meyer, der die 3D-Simulation im Rahmen einer Studienarbeit implementiert hat.

Öffnen Sie dazu im Editor-Fenster in der Menüleiste das Menü „Fenster“ und aktivieren Sie den Eintrag „3D-Simulation“. Wenn sich der Eintrag nicht aktivieren lässt, ist eine 3D-Ansicht auf Ihrem Rechner leider nicht möglich. Aktuell funktioniert die 3D-Ansicht leider nur unter Windows. Nach ein paar Sekunden Wartezeit öffnet sich das in Abbildung 21 gezeigte Fenster.



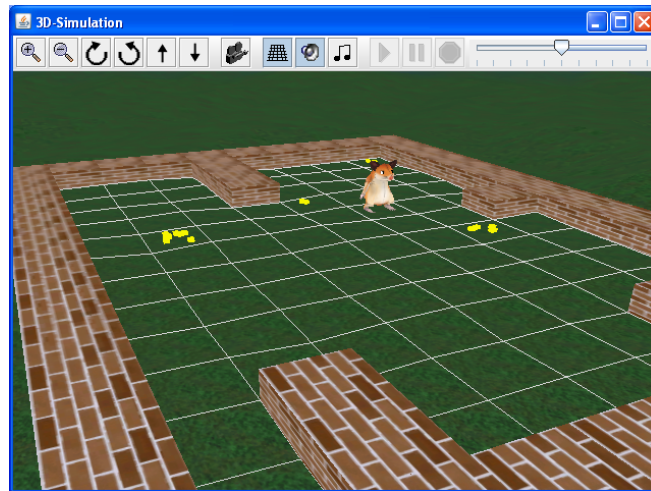


Abbildung 21: 3D-Simulationsfenster

Das 3D-Simulationsfenster enthält keine Buttons zur Gestaltung eines Territoriums. Die Territoriumsgestaltung findet ausschließlich im 2D-Simulationsfenster statt. Das 3D-Simulationsfenster wird aber immer automatisch entsprechend aktualisiert.

Am oberen Rand des 3D-Simulationsfenster befindet sich eine Toolbar. Diese enthält Buttons zur Manipulation der 3D-Ansicht und Steuerbefehle für den Ablauf der Simulation. Alternativ zur Nutzung dieser Buttons kann auch die Maus zur Manipulation der 3D-Ansicht verwendet werden.

#### 4.7.1 Steuerung mittels der Toolbar

Mit den ersten beiden Buttons können Sie in das Territorium hinein- und hinauszoomen. Die nächsten beiden Buttons erlauben das Drehen der Hamsterwelt. Gedreht wird dabei immer um den Punkt, auf den die Kamera schaut. Der Neigungswinkel der Kamera lässt sich mit dem nächsten Buttonpaar einstellen. Hier ist keine völlig freie Bewegung möglich; so lässt sich die Welt von einem recht flachen Winkel bis hin zur Vogelperspektive betrachten. Über 90 Grad lässt sich die Kamera nicht neigen; die Szene kann also nicht auf dem Kopf stehen.

Mit dem siebten Button der Toolbar kann die Ansicht in eine Ich-Perspektive umgeschaltet werden. In dieser Ansicht wird die Welt aus der Ansicht des Standard-Hamsters gezeigt und man folgt diesem auf seinem Weg durch das Territorium. Durch wiederholtes Drücken des Buttons wird die Ansicht wieder gewechselt.

Über den achten Button können die Gitterlinien entfernt bzw. eingeblendet werden.

Mit dem neunten Button kann ein Sound aktiviert bzw. deaktiviert werden, der ertönt, wenn der Hamster läuft. Auch der zehnte Button dient dem Aktivieren bzw. Deaktivieren von Sound, der fortwährend in einer Schleife abgespielt wird. Wenn Sie einen anderen Sound wünschen, wechseln Sie einfach im Unterverzeichnis „data“ die Datei „music.mid“ gegen eine andere MIDI-Datei aus.

Die weiteren Elemente der Toolbar kennen Sie bereits aus dem 2D-Simulationsfenster. Sie dienen zum Starten, Pausieren und Stoppen der Ausführung eines Hamster-Programms sowie zum Einstellen der Ausführungsgeschwindigkeit.

## 4.7.2 Steuerung mittels der Maus

Alternativ zur Benutzung der Toolbar-Buttons kann die 3D-Ansicht auch auf intuitive Weise mit der Maus manipuliert werden. So kann das Zoomen über das Mousrad vorgenommen werden. Bei gedrückter linker Maustaste kann die Hamsterwelt mit der Maus verschoben werden. Bei gedrückter rechter Maustaste wird die Welt mit Seitwärtsbewegungen rotiert und mit Aufwärtsbewegungen geneigt.

## 4.8 Dateiverwaltung auf Betriebssystemebene

Die von Ihnen entwickelten Hamster-Programme sowie die Hamster-Territorien werden auf Betriebssystemebene in Dateien abgespeichert. Diese Dateien finden Sie im so genannten *Workspace-Ordner*. Dies ist standardmäßig ein Ordner namens „Programme“ in demselben Ordner, in dem auch die Dateien zum Starten des Hamster-Simulators – `hamstersimulator.jar` bzw. `hamstersimulator.bat` – liegen. Über ein entsprechendes Property kann auch ein anderer Workspace-Ordner als der Ordner `Programme` verwendet werden (siehe Abschnitt 5). Funktionen zum Verwalten dieser Dateien (Umbenennen, Kopieren, Verschieben, neuer Ordner, ...) können Sie auch auf Betriebssystemebene durchführen. Sie sollten dies jedoch nicht tun, wenn der Hamster-Simulator gestartet ist, da es ansonsten zu Inkonsistenzen kommen kann.

Dateien mit Sourcecode von Hamster-Programmen haben die Endung „.ham“. Dateien mit Hamster-Territorien haben die Endung „.ter“. Ansonsten gibt es noch Dateien mit der Endung „.java“ und „.class“. Diese werden beim Kompilieren generiert und enthalten Java-Sourcecode („.java“) bzw. ausführbaren Java-Bytecode („.class“).

Sie können Hamster-Programme auch mit anderen Editoren entwickeln. Dabei gilt es allerdings folgendes zu beachten:

- Objektorientierten Hamster-Programmen muss folgender Kommentar unmittelbar vorangestellt werden: `/*object-oriented program*/`
- Separaten Klassen und Interfaces muss der Kommentar `/*class*/` unmittelbar vorangestellt werden.
- Imperativen Hamster-Programmen sollte (muss aber nicht) folgender Kommentar unmittelbar vorangestellt werden: `/*imperative program*/`

Die Kommentare kennzeichnen den Typ der Programme. Sie werden vom Editor des Hamster-Simulators automatisch generiert.

Weiterhin können Sie in Hamster-Programmen auch die Java-Anweisungen `System.out.println` und `System.err.println` benutzen. Die Ausgaben erfolgen bei Ausführung des Programms in Dateien namens `sysout.txt` bzw. `syserr.txt` in dem Ordner, in dem auch die Datei `hamstersimulator.jar` liegt.

Im Workspace-Ordner befindet sich eine Datei namens `settings.properties`, in der der aktuelle CLASSPATH (siehe Abschnitt 4.3.3) gespeichert wird.

## 5 Properties

Über so genannte Properties können sie bestimmte Voreinstellungen des Simulators überlagern.

### 5.1 Vorhandene Properties

Die Properties werden in einer Datei namens „hamster.properties“ definiert, die sich in dem Ordner befinden muss, wo sich auch die Dateien „hamstersimulator.jar“ bzw. „hamstersimulator.bat“ befinden. Momentan sind folgende Properties möglich:

#### 5.1.1 security

Befindet sich in der Datei „hamster.properties“ eine Zeile mit dem folgenden Text `security=false`, wird der so genannte Security-Manager ausgeschaltet. Das bedeutet, Hamster-Programme dürfen auf die Festplatte zugreifen und dürfen Dateien lesen und in Dateien schreiben. Aber Vorsicht, sollten sie diese Option gesetzt haben, empfehle ich Ihnen dringend, keine fremden Hamster-Programme auszuführen. Sind diese bspw. böswillig geschrieben, könnten sie Ihnen prinzipiell die gesamte Festplatte löschen. Standardmäßig steht in der Property-Datei `security=true`.

Durch Setzen der security-Property auf `false` ist es bspw. möglich, aus Hamster-Programmen heraus Sounds abzuspielen. Im folgenden Hamster-Programm wird ein Sound aus der angegebenen Datei abgespielt, während der Hamster zur Mauer läuft:

```
void main() {
    try {
        java.io.File f = new java.io.File( "C:\\\\fanfare.wav" );
        java.applet.AudioClip audioClip =
            java.applet.Applet.newAudioClip(f.toURL() );
        audioClip.play();
    } catch (Exception exc) { }

    while (vornFrei()) vor();
}
```

#### 5.1.2 workspace

Standardmäßig erscheint im Dateibaum als oberster Ordner ein Ordner names Programme, der so genannte *Workspace-Ordner*. Er repräsentiert den Unterordner Programme des Ordners, in dem sich die Dateien „hamstersimulator.jar“ bzw. „hamstersimulator.bat“ befinden. In diesem Ordner werden alle Hamster-Programme und

Hamster-Territorien abgespeichert. Durch Setzen der Property `workspace`<sup>3</sup> kann ein anderer Ordner als Workspace-Ordner festgelegt werden. Befindet sich in der Datei eine Zeile, die mit dem Text `workspace=` beginnt, wird der dahinter angegebene Ordner als Workspace-Ordner gesetzt, bspw.

`workspace=C:/Dokumente und Einstellungen/karl oder`

`workspace=C:/Dokumente und Einstellungen/heidi/Eigene Dateien oder  
workspace=../test.` Der angegebene Ordner muss existieren und er muss lesbar und beschreibbar sein! Achten Sie bitte darauf, dass in dem Ordner-Namen keine Sonderzeichen vorkommen (bspw. ein Ausrufezeichen), da die einige Java-Versionen nicht damit zurecht kommen. Für Windows-Nutzer ist es wichtig zu wissen, dass die \-Zeichen in den Ordner-Namen durch ein /-Zeichen ersetzt werden müssen. Standardmäßig steht in der Property-Datei `workspace=Programme`

Wenn Sie den Workspace-Ordner verändern und mit Paketen arbeiten, muss im CLASSPATH anstelle von „Programme“ der String angegeben werden, den Sie der Property `workspace` zugewiesen haben, also bspw. `C:\DokumenteundEinstellungen\karl` oder `../test`.

### 5.1.3 logfolder

Über diese Property kann der Ordner gewählt werden, in dem die beiden Dateien „sysout.txt“ und „syserr.txt“ erzeugt werden sollen. In diese Dateien werden Ausgaben auf Standard-Output (System.out) und Standard-Error (System.err) umgelenkt (nur im Modus `runlocally=false`).

Befindet sich in der Datei eine Zeile, die mit dem Text `logfolder=` beginnt, wird der dahinter angegebene Ordner als Logfolder-Ordner gesetzt, bspw.

`logfolder=C:/Dokumente und Einstellungen/karl oder`

`logfolder=C:/Dokumente und Einstellungen/heidi/Eigene Dateien oder  
logfolder=../test.` Der angegebene Ordner muss existieren und er muss lesbar und beschreibbar sein! Achten Sie bitte darauf, dass in dem Ordner-Namen keine Sonderzeichen vorkommen (bspw. ein Ausrufezeichen), da die einige Java-Versionen nicht damit zurecht kommen. Für Windows-Nutzer ist es wichtig zu wissen, dass die \-Zeichen in den Ordner-Namen durch ein /-Zeichen ersetzt werden müssen. Standardmäßig steht in der Property-Datei `logfolder=.`

### 5.1.4 scheme

Über die Property „scheme“ kann die Fähigkeit des Hamster-Simulators Scheme-Programme zu entwickeln bzw. auszuführen ein- bzw. ausgeschaltet werden (siehe Abschnitt 7). Befindet sich in der Datei „hamster.properties“ eine Zeile mit dem Text `scheme=false`, ist der Scheme-Modus ausgeschaltet. Befindet sich in der Datei „hamster.properties“ eine Zeile mit dem Text `scheme=true`, ist der Scheme-Modus eingeschaltet. Standardmäßig ist der Scheme-Modus ausgeschaltet.

---

<sup>3</sup>Aus Kompatibilität zu früheren Versionen des Hamster-Simulators kann diese Property auch **home** genannt werden.

### 5.1.5 runlocally

Über die Property „runlocally“ kann eingestellt werden, ob die Ausführung eines Hamster-Programms in einer neuen JVM oder in der JVM des Hamster-Simulators erfolgen soll. Befindet sich in der Datei „hamster.properties“ eine Zeile mit dem Text `runlocally=false`, werden Hamster-Programme in einer neuen JVM ausgeführt. Standardmäßig ist dies der Fall. Befindet sich in der Datei „hamster.properties“ eine Zeile mit dem Text `runlocally=true`, werden Hamster-Programme in derselben JVM wie der Hamster-Simulator selbst ausgeführt.

Normalerweise muss man sich nicht um dieses Property kümmern. Leider kam es jedoch bei einigen Macintosh- und Linux-Nutzer zu dem Problem, dass beim Starten eines Hamster-Programms Fehler auftraten oder sich der Start der Ausführung um viele Sekunden verzögerte. Im runlocally-Modus *true* kann das nicht mehr passieren. Nachteile dieses Modus: Eine Nutzung des Debuggers ist nicht möglich. Auch die Nutzung des CLASSPATH ist nicht möglich (siehe Kapitel 14 von Band 2 des Java-Hamster-Buches).

Im Modus `runlocally=true` gibt es ab Version 2.6.1 eine Console, die Ein- und Ausgaben über `System.in` bzw. `System.out` und `System.err` verarbeitet. Konkret bedeutet das: Enthält ein Hamster-Programm bspw. den Befehl „`System.out.println(hallo);`“ und wird das Programm ausgeführt, öffnet sich das Consolen-Fenster und die Zeichenkette „hallo“ wird in das Fenster geschrieben. Im Standard-Modus `runlocally=false` ändert sich nichts: Ausgabeanweisungen werden weiterhin in die Dateien `sysout.txt` bzw. `syserr.txt` geschrieben.

### 5.1.6 language

Über die Property „language“ kann die Sprache des Hamster-Simulators eingestellt werden. Befindet sich in der Datei „hamster.properties“ eine Zeile mit dem Text `language=de`, ist als Sprache Deutsch eingestellt. Das ist der Standard. Befindet sich in der Datei „hamster.properties“ eine Zeile mit dem Text `language=en`, ist als Sprache Englisch eingestellt. Weitere Infos zu englischen Hamster-Programmen siehe auch in Abschnitt 6.

### 5.1.7 indent

Über die Property „indent“ kann eingestellt werden, ob im Editor beim Zeilenumbruch die Cursorposition in der neuen Zeile anhand der ersten beschriebenen Spalte der vorhergehenden Zeile ausgerichtet wird. Befindet sich in der Datei „hamster.properties“ eine Zeile mit dem Text `indent=true`, ist die Spaltenausrichtung eingeschaltet. Das ist der Standard. Befindet sich in der Datei „hamster.properties“ eine Zeile mit dem Text `indent=false`, ist die Spaltenausrichtung ausgeschaltet.

Über das Menu „Extras“ kann der indent-Modus auch während der Programmausführung noch geändert werden.

### 5.1.8 color

Über die Property „color“ kann die Farbe des Standard-Hamsters geändert werden. Voreingestellt ist BLAU. Möglich sind folgende Farben: BLAU, BLUE, ROT, RED, GRUEN, GREEN, GELB, YELLOW, CYAN, MAGENTA, ORANGE, PINK, GRAU, GRAY, WEISS und WHITE.

Weiterhin wurde für objektorientierte Programme in der Klasse `Hamster` ein zusätzlicher Konstruktor eingeführt, bei dem als fünften Parameter die Farbe des Hamsters angegeben werden kann. Die Klasse `Hamster` stellt hierfür entsprechenden Konstanten zur Verfügung:

```
public final static int BLAU = 0;
public final static int BLUE = 0;

public final static int ROT = 1;
public final static int RED = 1;

public final static int GRUEN = 2;
public final static int GREEN = 2;

public final static int GELB = 3;
public final static int YELLOW = 3;

public final static int CYAN = 4;

public final static int MAGENTA = 5;

public final static int ORANGE = 6;

public final static int PINK = 7;

public final static int GRAU = 8;
public final static int GRAY = 8;

public final static int WEISS = 9;
public final static int WHITE = 9;

// neuer zusätzlicher Konstruktor
public Hamster(int reihe, int spalte,
               int blickrichtung, int anzahlKoerner,
               int farbe)
```

### 5.1.9 3D

Über die Property „3D“ kann das 3D-Simulationsfenster ein- oder ausgeschaltet werden (siehe Abschnitt 4.7). Befindet sich in der Datei „hamster.properties“ eine Zeile mit dem

Text `3D=false`, ist der 3D-Modus ausgeschaltet. Befindet sich in der Datei „hamster.properties“ eine Zeile mit dem Text `3D=true`, ist der 3D-Modus eingeschaltet. Standardmäßig ist der 3D-Modus eingeschaltet.

### 5.1.10 lego

Über die Property „lego“ kann der Lego-Modus ein- oder ausgeschaltet werden. Befindet sich in der Datei „hamster.properties“ eine Zeile mit dem Text `lego=false`, ist der Lego-Modus ausgeschaltet. Befindet sich in der Datei „hamster.properties“ eine Zeile mit dem Text `lego=true`, ist der Lego-Modus eingeschaltet. Standardmäßig ist der Lego-Modus ausgeschaltet. Die Möglichkeit, über den Hamster-Simulator einen Lego-Mindstorms-Roboter zu steuern, wird erst in Version 2.7 des Hamster-Simulators erläutert.

### 5.1.11 prolog

Über die Property „prolog“ kann die Fähigkeit des Hamster-Simulators Prolog-Programme zu entwickeln bzw. auszuführen ein- bzw. ausgeschaltet werden (siehe Abschnitt 8). Befindet sich in der Datei „hamster.properties“ eine Zeile mit dem Text `prolog=false`, ist der Prolog-Modus ausgeschaltet. Befindet sich in der Datei „hamster.properties“ eine Zeile mit dem Text `prolog=true`, ist der Prolog-Modus eingeschaltet. Standardmäßig ist der Prolog-Modus ausgeschaltet.

## 5.2 Mehrbenutzerfähigkeit

Seit Version 2.3 ist der Hamster-Simulator Mehrbenutzer-fähig, d.h. er kann einmal auf einem Server installiert und dann von mehreren Nutzern (gleichzeitig) genutzt werden. Jeder Nutzer hat dabei seinen eigenen Ordner für die Hamster-Programme.

Um die Mehrbenutzer-Fähigkeit des Hamster-Simulators zu nutzen, muss ein Nutzer eine Datei namens „hamster.properties“ in seinem HOME-Verzeichnis anlegen (bspw. durch Kopieren der gleichnamigen Datei aus dem Ordner, wo sich auch die Dateien „hamstersimulator.jar“ bzw. „hamstersimulator.bat“ befinden). Die Property-Einstellungen in der Datei im HOME-Verzeichnis haben dabei die höchste Priorität bezogen auf den Nutzer!

In der Datei „hamster.properties“ in seinem HOME-Verzeichnis sollte dann jeder Nutzer die Properties **workspace** und **logfolder** entsprechend seinen Wünschen setzen, d.h. dort sollten die Ordner eingetragen werden, in dem die Hamster-Programme dieses Nutzers gespeichert bzw. in dem die Log-Dateien `sysout.txt` und `syserr.txt` erzeugt werden sollen.

## 6 Englischsprachiger Hamster

Seit Version 2.4 des Hamster-Simulators ist der Hamster-Simulator in die englisch-sprachige Welt integriert worden. Das betrifft zunächst die Benutzungsoberfläche. Durch Einstellen der Property *language* auf den Wert *en* erscheinen alle Ausgaben in englischer Sprache.

Aber nicht nur die Oberfläche auch das Hamster-Modell selbst wurde angepasst. Bswp. versteht der Hamster ab sofort nicht mehr nur den Befehl `vor()` ;, sondern auch den Befehl `move()` ;, der dasselbe bewirkt: Der Hamster springt eine Kachel in Blickrichtung nach vorne.

Im Folgenden wird ein Überblick über die entsprechenden englischen Befehle bzw. Klassen des Hamster-Modells gegeben:

<code>vor</code>	<code>move</code>
<code>linksUm</code>	<code>turnLeft</code>
<code>nimm</code>	<code>pickGrain</code>
<code>gib</code>	<code>putGrain</code>
<code>vornFrei</code>	<code>frontIsClear</code>
<code>kornDa</code>	<code>grainAvailable</code>
<code>maulLeer</code>	<code>mouthEmpty</code>
<code>liesZahl</code>	<code>readNumber</code>
<code>liesZeichenkette</code>	<code>readString</code>
<code>schreib</code>	<code>write</code>
<code>init</code>	<code>init</code>
<code>getReihe</code>	<code>getRow</code>
<code>getSpalte</code>	<code>getColumn</code>
<code>getBlickrichtung</code>	<code>getDirection</code>
<code>getStandardHamster</code>	<code>getDefaultHamster</code>
<code>getAnzahlKoerner</code>	<code>getNumberOfGrains</code>
<code>getAnzahlHamster</code>	<code>getNumberOfHamsters</code>
<code>Territorium</code>	<code>Territory</code>
<code>HamsterInitialisierungsException</code>	<code>HamsterInitializationExce</code>
<code>ption</code>	
<code>HamsterNichtInitialisiertException</code>	<code>HamsterNotInitializedExce</code>
<code>ption</code>	
<code>KachelLeerException</code>	<code>TileEmptyException</code>
<code>MauerDaException</code>	<code>WallInFrontException</code>
<code>MaulLeerException</code>	<code>MouthEmptyException</code>



Ein imperatives englisches Hamster-Programm, bei dem der Hamster alle vor ihm liegenden Körner einsammeln soll, sieht damit folgendermaßen aus:

```
void main() {
    pickAll();
    while (frontIsClear()) {
        move();
        pickAll();
    }
}

void pickAll() {
    while (grainAvailable())
        pickGrain();
}
```

Das folgende Programm skizziert ein objektorientiertes englisches Hamster-Programm:

```
class MyHamster extends Hamster {
    MyHamster(Hamster h) {
        super(h.getRow(), h.getColumn(),
              h.getDirection(), h.getNumberOfGrains());
    }

    void turnRight() {
        this.turnLeft();
        this.turnLeft();
        this.turnLeft();
    }
}

void main() {
    MyHamster paul = new MyHamster(Hamster.getDefaultHamster());
    try {
        while (true) {
            paul.move();
        }
    } catch (WallInFrontException exc) {
    }
    paul.turnRight();
    paul.write("Number of Hamsters: " +
               Territory.getNumberOfHamsters());
}
```

Prinzipiell kann man übrigens auch die deutschen und englischen Befehle mischen.

## 7 Scheme

In Band 1 des Hamster-Buches (Programmieren spielend gelernt mit dem Java-Hamster-Modell) werden die Programmiersprachen verschiedenen *Programmierparadigmen* zugeordnet. Java wird dabei in die Klasse der imperativen objektorientierten Sprachen eingeordnet. Ich versichere Ihnen, wenn Sie die imperativen Konzepte von Java verstanden haben (Prozeduren, Anweisungen, Schleifen, Variablen, ...) werden Sie ohne große Probleme auch andere imperative Programmiersprachen, wie Pascal oder Modula-2 erlernen können. Im Prinzip unterscheiden diese sich nur durch eine andere Syntax von der Programmiersprache Java.

Anders sieht dies jedoch auch, wenn Sie Programmiersprachen anderer Programmierparadigmen lernen wollen. Die zugrunde liegenden Konzepte der einzelnen Programmierparadigmen weichen stark voneinander ab. Seit Version 2.3 unterstützt der Hamster-Simulator das funktionale Programmierparadigma: Es ist möglich, in der funktionalen Programmiersprache *Scheme* Hamster-Programme zu entwickeln und auszuführen. Herzlichen Dank an Martin Kramer, der im Rahmen einer Studienarbeit die Integration von Scheme in den Hamster-Simulator vorgenommen hat.

Dieses Benutzerhandbuch enthält keine Einführung in die funktionale Programmierung und auch keine Einführung in die Programmiersprache Scheme. Hierzu wird auf die im folgenden genannte Literatur verwiesen. Wenn Sie also Scheme lernen wollen, sollten Sie sich eines der genannten Bücher beschaffen oder die online-verfügbare Literatur sichten.

Ein Problem vieler Anfängerbücher für Scheme ist, dass nahezu alle Beispiele aus der Welt der Mathematik stammen, was Schüler bzw. Studierende, die keinen großen Bezug zur Mathematik haben, häufig abschreckt. An dieser Stelle setzt das Hamster-Modell an. Sie können Scheme quasi unabhängig von Ihren mathematischen Fähigkeiten lernen, in dem Sie einen Hamster durch ein Territorium steuern und ihn bestimmte (nicht-mathematische) Aufgaben lösen lassen.

### 7.1 Funktionale Programmiersprachen

Die wichtigsten Unterschiede funktionaler Programmiersprachen gegenüber imperativen Programmiersprachen sind:

- Programme funktionaler Programmiersprachen werden als mathematische Funktionen betrachtet.
- Funktionen werden als Daten behandelt.
- Seiteneffekte von Funktionen werden stark eingeschränkt.
- Es gibt keine Variablen.
- Es gibt keine Schleifen.
- Rekursion spielt in der funktionalen Programmierung eine entscheidende Rolle.
- Zentrale Datenstruktur der funktionalen Programmierung ist die Liste.

Funktionale Programmiersprachen werden insbesondere im Bereich der Künstlichen Intelligenz, für mathematische Beweissysteme und für Logikanwendungen eingesetzt. Weitere Eigenschaften der funktionalen Programmierung finden Sie bspw. unter [http://de.wikipedia.org/wiki/Funktionale\\_Programmierung](http://de.wikipedia.org/wiki/Funktionale_Programmierung)

## 7.2 Die Programmiersprache Scheme

Die erste funktionale Programmiersprache, die in den 60er Jahren entwickelt wurde, hieß *LISP*. Von Lisp wurden viele Dialekte entwickelt. Die beiden Dialekte, die zum Standard geworden sind, heißen *Common Lisp* und eben *Scheme*, d.h. Scheme ist ein standardisierter Lisp-Dialekt.

Scheme ist keine rein-funktionale Programmiersprache. Vielmehr enthält sie auch Konzepte der imperativen Programmierung (Schleifen, Variablen). Um funktional programmieren zu lernen, sollten Sie sich jedoch auf die funktionalen Konzepte beschränken und die imperativen Konzepte nicht benutzen!

Im Internet finden sich eine Reihe von Informationen über und Online-Tutorials zu Scheme:

- <http://de.wikipedia.org/wiki/Scheme>
- <http://www.htdp.org/> (How to design programs)
- <http://www.ccs.neu.edu/home/dorai/t-y-scheme/t-y-scheme-Z-H-1.html> (Teach Yourself Scheme in Fixnum Days)
- <http://www.cs.hut.fi/Studies/T-93.210/schemetutorial/schemetutorial.html> (ausführliches Scheme-Tutorial)
- <http://cs.wvc.edu/KU/PR/Scheme.html> (kurzes Scheme-Tutorial)
- <http://www.scheme.com/tspl2d/index.html> (The Scheme Programming Language)
- [http://www-pu.informatik.uni-tuebingen.de/pfg-2001/scheme/schintro-v14/schintro\\_toc.html](http://www-pu.informatik.uni-tuebingen.de/pfg-2001/scheme/schintro-v14/schintro_toc.html) (An Introduction to Scheme and its Implementation)

## 7.3 Scheme-Hamster-Programme

Um Scheme-Hamster-Programme zu entwickeln und zu testen, müssen Sie die Property **scheme** auf **true** gesetzt haben (siehe Abschnitt

5). Dies ist standardmäßig der Fall.

Anschließend müssen Sie eine neue Datei öffnen (Menü „Datei“, Item „Neu“) und in der Auswahl „Scheme-Programm“ auswählen. Es erscheint eine neue Datei mit folgendem Grundgerüst:

```
(define (start Territorium)
  ()
)
```

Die zweite Zeile ersetzen Sie dabei durch entsprechende Scheme-Anweisungen. Bspw. lässt das folgende Scheme-Programm den Hamster zwei Schritte vorlaufen:

```
(define (start Territorium)
  (vor (vor Territorium))
)
```

Anschließend müssen Sie die neue Datei abspeichern. Nach dem Speichern können Sie Ihr Scheme-Programm durch Drücken des Start-Buttons ausführen. Kompilieren ist nicht notwendig, Scheme-Programme werden interpretiert, d.h. wenn sie Fehler enthalten, werden Sie darüber zur Ausführungszeit informiert.

## 7.4 Grundlagen und Befehle

### 7.4.1 Territoriumsliste

Die grundlegende Datenstruktur des Scheme-Hamster-Modells ist die Territoriumsliste. Sie spiegelt das aktuelle Territorium des Simulation-Fensters wieder. Sie ist folgendermaßen aufgebaut (in EBNF):

```
<territorium>      ::= "(" <feld-liste> <hamster-liste> ")"
<feld-liste>       ::= "(" { <reihe-liste> } ")"
<reihe-liste>      ::= "(" { <kachel> } ")"
<kachel>           ::= "(" "Kachel" <koerner-auf-kachel> ")"
                   | "(" "Mauer" ")"
<hamster-liste>    ::= "(" "Hamster"
                        <reihe>
                        <spalte>
                        <koerner-im-maul>
                        <blickrichtung>
                        ")"
<blickrichtung>    ::= "Nord"
                   | "Ost"
```

```
| "Sued"
| "West"
```

```
<koerner-auf-kachel> ist positiver int-Wert
<reihe> ist positiver int-Wert oder 0
<spalte> ist positiver int-Wert oder 0
<koerner-im-maul> ist positiver int-Wert
```

Der Ursprung des Territoriums befindet sich in der linken oberen Ecke. Die Nummerierung von Spalten und Zeilen beginnt bei 0.

Ein Beispiel: Das folgende Territorium besteht aus zwei Reihen und drei Spalten. In der ersten Spalte liegen auf beiden Kacheln keine Körner. In der zweiten Spalte liegen auf beiden Kacheln drei Körner. In der dritten Spalte befinden sich ausschließlich Mauern. Der Hamster steht mit Blickrichtung Nord und 3 Körnern im Maul auf der Kachel in der linken unteren Ecke.

```
(
  (
    ( ("Kachel" 0) ("Kachel" 3) ("Mauer") )
    ( ("Kachel" 0) ("Kachel" 3) ("Mauer") )
  )
  ("Hamster" 1 0 3 "Nord")
)
```

#### 7.4.2 Hamster-Befehle

Die Hamster-Befehle des Java-Hamster-Modells sind als Funktionen implementiert, die ein Territorium auf ein neues Territorium abbilden, und zwar mit der bekannten Semantik der Hamster-Grundbefehle:

- `(vor Territorium)` liefert ein Territorium, in dem der Hamster gegenüber dem als Parameter übergebenen Territorium eine Kachel in Blickrichtung gelaufen ist
- `(linksUm Territorium)` liefert ein Territorium, in dem sich der Hamster gegenüber dem als Parameter übergebenen Territorium um 90 Grad nach links umgedreht hat
- `(nimm Territorium)` liefert ein Territorium, in dem der Hamster gegenüber dem als Parameter übergebenen Territorium ein Korn mehr im Maul hat und sich auf der entsprechenden Kachel ein Korn weniger befindet
- `(gib Territorium)` liefert ein Territorium, in dem der Hamster gegenüber dem als Parameter übergebenen Territorium ein Korn weniger im Maul hat und sich auf der entsprechenden Kachel ein Korn mehr befindet
- `(vornFrei? Territorium)` liefert `true`, wenn sich in dem als Parameter übergebenen Territorium keine Mauer vor dem Hamster befindet

- (maulLeer? Territorium) liefert true, wenn in dem als Parameter übergebenen Territorium der Hamster keine Körner im Maul hat
- (kornDa? Territorium) liefert true, wenn sich in dem als Parameter übergebenen Territorium auf der Kachel, auf der der Hamster steht, mindestens ein Korn befindet

Bei den Befehlen `vor`, `nimm` und `gib` können die bekannten Fehler auftreten.

Im Hamster-Simulator geschieht nach dem Ausführen eines der vier Hamster-Grundbefehle folgendes: Das von der entsprechenden Funktion gelieferte Territorium wird im Simulation-Fenster angezeigt!

### 7.4.3 Scheme-Hamster-Programme

Ein Scheme-Hamster-Programm hat immer folgende Gestalt:

```
(define (start Territorium)
  <Funktionsaufruf>
)
```

**start** ist die Funktion, die beim Ausführen eines Hamster-Programms aufgerufen wird. Ihr wird als Parameter die entsprechende Listenrepräsentation des aktuell im Simulation-Fenster angezeigten Territoriums übergeben.

Im folgenden Hamster-Programm hüpft der Hamster eine Kachel nach vorne:

```
(define (start Territorium)
  (vor Territorium)
)
```

Im folgenden Hamster-Programm hüpft der Hamster eine Kachel nach vorne und dreht sich anschließend nach links:

```
(define (start Territorium)
  (linksUm (vor Territorium))
)
```

Vergleichen Sie mit diesem Programm bitte das folgende Hamster-Programm:

```
(define (start Territorium)
  (vor Territorium)
  (linksUm Territorium)
)
```

Hier hüpft der Hamster zunächst eine Kachel nach vorne. Anschließend wird der Befehl `linksUm` jedoch auf dem anfänglichen Territorium, dem Parameter, ausgeführt, und nicht auf dem durch den vor-Befehl geänderten Territorium, d.h. das Ergebnis ist, dass sich der Hamster nach der Ausführung des Programms auf der selben Kachel befindet, auf der er sich vor Start des Programms befand, allerdings mit einer anderen Blickrichtung. Grund hierfür ist der folgende: Im Hamster-Simulator geschieht nach dem Ausführen eines der vier Hamster-Grundbefehle in Scheme folgendes: Das von der entsprechenden Funktion gelieferte Territorium wird im Simulation-Fenster angezeigt. Es wird nicht wie im imperativen Hamster-Modell der Befehl auf dem aktuell sichtbaren Territorium ausgeführt.

Ein Hinweis für erfahrenere Scheme-Programmierer: Ausgaben bzw. Eingaben über die Scheme-Funktionen `display` bzw. `read` erfolgen über Standard-Out bzw. Standard-In, d.h. Sie sollten den Hamster-Simulator entsprechend über den Befehl „`java -jar hamstersimulator.jar`“ und nicht via Doppelklick auf die Datei „`hamstersimulator.jar`“ starten, damit Sie eine Konsole haben.

## 7.5 Beispiele

Es folgen ein paar Beispiele für nützliche Scheme-Hamster-Funktionen:

```
(define (kehrt T)
  (linksUm (linksUm T))
)

(define (rechtsUm T)
  (kehrt (linksUm T))
)

(define (nimmAlle T)
  (if (kornDa? T)
      (nimmAlle (nimm T))
      T
  )
)

(define (gibAlle T)
  (if (not (maulLeer? T))
      (gibAlle (gib T))
      T
  )
)

(define (vorwaerts n T)
  (if (and (vornFrei? T) (> n 0))
      (vorwaerts (- n 1) (vor T))
      T
  )
)
```

```

(define (zurMauer T)
  (if (vornFrei? T)
      (zurMauer (vor T))
      T)
  )
)

(define (hinUndZurueck T)
  (if (vornFrei? T)
      (vor (hinUndZurueck (vor T)))
      (kehrt T)
  )
)

(define (reiheBearbeiten T)
  (if (vornFrei? T)
      (reiheBearbeiten (vor (nimmAlle T)))
      (nimmAlle T)
  )
)

(define (linksFrei? T)
  (let ((T1 (linksUm T)))
    (if (vornFrei? T1)
        (begin (rechtsUm T1) #t)
        (begin (rechtsUm T1) #f)
    )
  )
)

(define (rechtsFrei? T)
  (let ((T1 (rechtsUm T)))
    (if (vornFrei? T1)
        (begin (linksUm T1) #t)
        (begin (linksUm T1) #f)
    )
  )
)

```

## 7.6 Scheme-Konsole

Neben dem Schreiben kompletter Scheme-Hamster-Programme ist es auch möglich, einzelne Befehle interaktiv ausführen zu lassen. Hierzu dient die Scheme-Konsole. Diese können Sie über das Menü „Fenster“ des Editor-Fensters öffnen. In den Eingabebereich



können Sie nun Scheme-Befehle eingeben und durch Anklicken des „Ausführen“-Buttons ausführen. Ausgaben erscheinen im Ausgabebereich der Konsole.

Tippen Sie bspw. mal in den Eingabebereich der Scheme-Konsole folgendes ein: `(vor (getTerritorium))`. Der Hamster hüpft eine Kachel nach vorne. Die Funktion `getTerritorium` liefert dabei die entsprechende Territoriumsliste des aktuell im Simulation-Fenster sichtbaren Territoriums.

Mit Hilfe der Buttons „vor“ und „zurück“ können Sie in Ihren bereits ausgeführten Befehlen navigieren, um bspw. einen Befehl wiederholt auszuführen. Mit dem Button „Löschen“ löschen Sie Eingaben im Eingabebereich.

Aktuell ist es in der Konsole nur möglich, Befehle einzeln auszuführen. Wenn Sie versuchen, mehrere Befehle gleichzeitig ausführen zu lassen, wird nur der letzte tatsächlich ausgeführt.

Sowohl der Eingabe- als auch der Ausgabebereich der Konsole unterstützen „Copy und Paste“, und zwar über die Tastatureingaben „Strg-c“ bzw. „Strg-v“.

Bei Ausführung der Scheme-Funktion `read` in der Scheme-Konsole, erscheint eine Dialogbox, in der Sie die Eingabe tätigen müssen.

Wenn Sie über die Konsole Funktionsdefinitionen ausführen, geben Sie diese dem kompletten System bekannt. D.h. Sie können die entsprechenden Funktionen danach auch in Scheme-Programmen nutzen, ohne sie erneut definieren zu müssen. Umgekehrt gilt dies genauso! Bei Beendigung des Hamster-Simulators gehen jedoch alle entsprechenden Definitionen „verloren“.

## 7.7 Implementierungshinweise

Für die Implementierung des Scheme-Hamster-Modells wurde JScheme benutzt, siehe <http://jscheme.sourceforge.net/jscheme/mainwebpage.html>.

## 8 Prolog

Ab der Version 2.6 deckt der Hamster-Simulator nach der imperativen, objektorientierten und parallelen Programmierung mit Java und der funktionalen Programmierung mit Scheme mit der logikbasierten Programmierung mit Prolog alle wichtigen Programmierparadigmen ab. Herzlichen Dank an Andreas Schäfer, der im Rahmen einer Studienarbeit die Integration von Prolog in den Hamster-Simulator vorgenommen hat.

### 8.0 Voraussetzungen

Für den Einsatz von PROLOG innerhalb des Hamster-Simulators muss auf dem Zielsystem ein SWIProlog-Interpreter installiert werden. Die dafür notwendigen Installationsdateien können direkt von der SWIProlog-Homepage heruntergeladen werden. Die SWIProlog-Homepage ist unter dem Link <http://www.swi-prolog.org/> erreichbar. Bei der Entwicklung der PROLOG-Komponente wurde die Version 5.6.61 des Interpreters verwendet. Auch andere Versionen sind aber durchaus verwendbar, da keine zusätzlichen Bibliotheken verwendet werden.

Die PROLOG-Komponente verwendet die konsolenbasierte Version des Interpreters (plcon.exe). Die dafür verantwortliche Anwendungsdatei befindet sich nach der Installation im Unterverzeichnis \bin\ des Installationsverzeichnis von SWIProlog.. Damit die Anwendungsdatei von der Java Virtual Machine gefunden werden kann, muss die Umgebungsvariable PATH um den Pfad, standardmäßig C:\Programme\pl\bin\ erweitert werden.

Weiterhin muss in der Datei „hamster.properties“ das Property `prolog` auf `true` gesetzt sein (siehe Abschnitt 5.1.11)

### 8.1 Logikbasierte Programmierung

Aus Wikipedia: Logische oder logikbasierte Programmierung ist ein Programmierparadigma, das auf der mathematischen Logik beruht. Anders als bei der imperativen Programmierung besteht ein Logik-Programm nicht aus einer Folge von Anweisungen, sondern aus einer Menge von Axiomen, welche hier als eine reine Ansammlung von Fakten oder Annahmen zu verstehen sind. Stellt der Benutzer eines Logik-Programms eine Anfrage, so versucht der Interpreter die Lösungsaussage allein aus den Axiomen zu berechnen.

Dazu werden eine Menge von so genannten Regeln und Anweisungen, die der Syntax gemäß aufgebaut sind, zusammen mit der Information, welche Lösungsmethode vorgesehen ist, in den Programmcode eingefügt. Logische Programmiersprachen gehören zu den deklarativen Programmiersprachen und haben ihre Ursprünge im Forschungsgebiet Künstliche Intelligenz.

In einem imperativen Programm wird genau beschrieben, wie und in welcher Reihenfolge ein Problem zu lösen ist. Im Gegensatz dazu wird in einem logikbasierten Programm

idealerweise nur beschrieben, was gilt. Das "wie" ist bereits durch die Lösungsmethode vorgegeben. Die Lösung wird aus den vorhandenen Regeln hergeleitet. Meistens wird schon nur eine Menge von Regeln als "das Programm" bezeichnet, wenn klar ist, welche Lösungsmethode dazugehört: Nämlich die (einzige) in der vom regelbasierten System bereit gestellten Inferenzmaschine verwirklichte.

Die bekannteste logische Programmiersprache ist Prolog.

## 8.2 Die Programmiersprache Prolog

Die Programmiersprache PROLOG (fr. *PRO*grammation en *LOG*ique) wurde Anfang 70er Jahre *Alain Colmerauer* und *Philippe Roussel* an der Universität von Marseille entwickelt und erstmalig implementiert. Die zahlreichen, bereits gemachten Erfahrungen auf dem Gebiet der automatischen Beweisführung mathematischer Sätze, deren Entwicklung in dem Moment vorerst nur innerhalb der Logik vorangetrieben wurde, lieferten die Ausgangsbasis zur Entwicklung der Sprache. Die Aufstellung der wesentlichen theoretischen Grundlagen zur logischen Programmierung reicht bis in die Mitte des 19. Jahrhunderts zurück. Die Syntax der Prädikatenlogik erster Stufe, aufgestellt von Gottlob Frege, geprägt und weiterentwickelt durch einer Vielzahl weiterer Wissenschaftler, bildet die Grundlage für die Syntax der meisten heutigen logischen Programmiersprachen.

Das ursprünglich gedachte Anwendungsfeld für PROLOG, als Werkzeug für Sprachwissenschaftler zur Erfassung und Verarbeitung natürlichsprachlicher Sätze auf logischer Ebene wurde im späteren Verlauf erweitert. Es wurde schnell erkannt, dass PROLOG auch als eigenständige Programmiersprache genutzt werden kann. Eine zunehmende Verbreitung von PROLOG begünstigte Mitte der 70er Jahren die Entwicklung der sogenannten „Warren Abstract Machine“ (WAM) von *David Warren*, die eine effizientere und schnellere Implementierung von PROLOG darstellte. Die WAM stellt auch heute noch wie vor die Grundlage für die meisten PROLOG-Implementierungen dar. Ein in den 70er Jahren stattgefundenes japanisches Projekt am Forschungsinstitut ICOT<sup>4</sup> zur Entwicklung der fünften Rechnergeneration wählte ebenfalls die logische Programmierung als Grundlage für die Entwicklung. Dies begünstigte ebenfalls die Verbreitung und Entwicklung dieser Programmierung. Seit Ende der 80er Jahre führte die Entwicklung zur Entstehung von sogenannten constraint-logischen Programmiersprachen (*Constraint Logic Programming*), die als Erweiterung der logischen Programmierung angesehen werden können. Diese Sprachen gewinnen heute eine immer größere Bedeutung in der Forschung und in der Anwendung.

Gegenüber den traditionellen, prozeduralen Programmiersprachen wie beispielsweise JAVA oder C++ hat die Programmierung in PROLOG einen ganz anderen Charakter. Wie bereits erwähnt, orientiert sich PROLOG an den Möglichkeiten einer eher abstrakteren Wissensdarstellung mit Hilfe des Prädikatenkalküls erster Stufe, anstatt sich bei der traditionellen Rechnerarchitektur, wie es bei den anderen Sprachen der Fall ist, zu orientieren. Dabei ändert sich der Programmierstil für PROLOG im Vergleich zu den Letzteren sehr stark: Hier wird das eigentliche Problem in den Vordergrund gestellt.

---

<sup>4</sup>Institute for New Generation Computer Technology

Anstatt sich beim Entwurf dem algorithmischen Ablauf des Programms zu widmen und dieses danach zu strukturieren, beschreiben die PROLOG-Programme im Wesentlichen nur „was“ das Ziel eines Programms bzw. Programmteils darstellt. Dabei wird also nicht unbedingt extra spezifiziert, „wie“ dieses Ziel erreicht werden soll. Ein PROLOG-Programm repräsentiert im Allgemeinen lediglich nur das Wissen, das dazu notwendig wäre, mit Hilfe logischer Schlussfolgerungen an die Lösung des Problems bzw. der an PROLOG-Interpreter gestellten Anfrage zu kommen. Dank dieser Eigenschaft sind die PROLOG-Programme sehr kompakt und der Entwicklungsaufwand wird dadurch verringert.

Eine andere Eigenschaft von PROLOG liegt an der einfachen Möglichkeit der Programme, sich während der Laufzeit zu verändern. Dabei kann Programm-Wissen dazugewonnen oder auch verworfen werden. Das Programm kann dabei als eine Art Datenbank angesehen werden, die sich nach Belieben kürzen oder erweitern lässt. Aufgrund der gleichen Struktur der Daten und Prozeduren, worauf im späteren Verlauf etwas näher eingegangen wird, kann die Erweiterung des Programms sehr leicht vollzogen werden.

Die grundlegende Vorgehensweise im Umgang mit PROLOG gestaltet sich wie folgt: Ähnlich wie es im Umgang mit einer Datenbank der Fall ist, kann der Benutzer unterschiedliche Anfragen an das PROLOG-System stellen. Jede Anfrage wird vom PROLOG-System als eine neue These bzw. ein Ziel interpretiert, die das System nachfolgend versucht mit Hilfe der vorliegenden Fakten und Prozeduren im Programm zu beweisen. Hierzu wird der sogenannte *Resolutionsalgorithmus*, um vom bekannten Informationen eine Lösung für das angeforderte Ziel abzuleiten, verwendet. Nebenbei werden unter anderem Verfahren wie das *Pattern Matching* (Mustervergleich oder Unifikation), und das *Backtracking* (Rückverfolgung) bei der Suche nach einer Lösung im Resolutionsalgorithmus angewandt.

## 8.2.1 Syntax von Prolog

Dieser Abschnitt beschreibt die Regeln für die syntaktische Zusammensetzung der PROLOG-Programme. Im ersten Teil erfolgt die Vorstellung der grundlegenden Datentypen der Sprache. Im zweiten Part wird der generelle Aufbau eines PROLOG-Programms beschrieben.

### 8.2.1.1 Datentypen von Prolog

Wie auch jede andere Programmiersprache hat auch PROLOG zur Beschreibung und Charakterisierung von Informationen primitive Datentypen. Im Vergleich zu JAVA gibt es aber in PROLOG nur schwache Typisierung. Der endgültige Datentyp zu einem Ausdruck wird in PROLOG erst zur Laufzeit dynamisch ermittelt und zugewiesen. Nachfolgend werden die wichtigsten Datentypen von PROLOG beschrieben.

#### Term

Ein Term ist der grundlegende syntaktische Baustein eines PROLOG-Programms. Bei den Termen unterscheidet man zwischen einfachen und komplexen Termen. Einfache Terme können entweder *konstant* (Atome und Zahlen) oder auch *variabel* (Variablen) sein. Komplexe Terme, im Weiteren auch *Strukturen* genannt, entstehen durch Zusammensetzung mehrerer einfacher Terme und haben einen besonderen Aufbau.

## Einfache Terme

Zu den **einfachen Termen** gehören **Atome**, **Zahlen** und **Variablen**. Die Atome sind beliebige Zeichenfolgen, die mit einem Kleinbuchstaben anfangen. Diese setzen sich dabei zusammen aus Kleinbuchstaben, Ziffern oder dem Unterstreichungszeichen. Weiterhin können auch Atome gebildet werden, indem beliebige, in Anführungszeichen eingeschlossene Zeichenfolgen angegeben werden. In diesem Fall können auch Sonderzeichen und Großbuchstaben bei der Definition eines Atoms verwendet werden. Nachfolgend sind einige Beispiele für Atom-Definitionen aufgeführt:

**Atom-Beispiele:** `kornfeld`, `hamster`, `123ham`, `'ROGGEN'`, `'Weizen'`

Die **Zahlen** werden in PROLOG ebenfalls als konstante Terme dargestellt. Diese können als Integerzahlen (12, 3500, ...) oder Dezimalzahlen (mit Dezimalpunkt- (3.1415) oder Exponentenschreibweise (1.34e10) geschrieben werden. PROLOG definiert eine Reihe arithmetischer Operatoren und vordefinierter Systemprädikate, die in Verbindung mit Zahlen-Termen eingesetzt werden können.

**Variablen** sind spezielle Terme, die als Platzhalter für beliebige andere PROLOG-Terme dienen. Bei den Variablen unterscheidet man zwischen *anonymen* und *nicht anonymen* bzw. *normalen* Variablen. Der Variablen-Term besitzt in PROLOG zum leichteren Erkennen eine besondere Schreibweise:

- Die *normale* Variable beginnt immer mit einem Großbuchstaben. Der Rest der Variable besteht aus Kleinbuchstaben, Ziffern oder dem Unterstreichungszeichen.

Nicht anonyme Variablen: `A`, `B`, `HamsterXYZ`.

- Die *anonyme* Variable beginnt im Unterschied dazu mit einem Unterstrich. Dabei kann diese unter Umständen auch nur aus einem einzigen Unterstrich bestehen. Die *anonyme* Variable spielt hier eine besondere Rolle: Mit dem Einsatz der anonymen Variablen bietet PROLOG die Möglichkeit, die sogenannten „Dont-Care“-Variablen zu markieren. Dies sind Variablen, deren späteren Wertebelegungen den Programmierer einfach nicht interessieren. Hierbei ist eine sehr wichtige Eigenschaft der anonymen Variable zu beachten: Jedes Vorkommen der anonymen Variable „`_`“ innerhalb eines PROLOG-Prädikats referenziert jeweils eine andere Variablen-Instanz. Dadurch kann sich der Programmierer den Aufwand des Ausdenkens einen eindeutigen Namen für Variable ersparen, sofern ihn die Variable an sich nicht interessiert.

Anonyme Variablen: `_a`, `_myHam`, `_`

## Komplexe Terme (Strukturen)

Ein komplexer Term kann im Allgemeinen durch ein Ausdruck der Form: `p(a1, ..., an)` beschrieben werden. Er besteht aus einem Funktor `p` und einer beliebigen Anzahl von Argumenten (`a1, ..., an`). Beim Funktor handelt es sich um ein Atom, welches in der Regel in der Präfixnotation mit der in runde Klammern eingeschlossenen Argumentenliste

geschrieben wird<sup>5</sup>. An Stelle der Argumente können beliebige PROLOG-Terme, also Konstanten, Variablen oder auch komplexe Terme, verwendet werden. Die Anzahl der Argumente des komplexen Terms definiert seine Stelligkeit. Bei dem Term mit der Stelligkeit 0 handelt es sich in diesem Sinne um ein einfaches Atom. Nachfolgend sind einige Beispiele der komplexen Terme aufgelistet.

```
Tier(hamster)
kornFeld(X, Y)
liebt(hamster, koerner)
```

Der erste Term drückt die Beziehung aus, dass der Hamster ein Tier ist. Die Stelligkeit des Terms `tier(hamster)` ist 1. Die Stelligkeit des zweiten Terms, `kornfeld(X,Y)` ist 2. Dabei sind Argumente des Terms noch nicht genau spezifiziert. Dabei kann es sich an Stelle von X und Y noch um beliebige, auch nicht numerische Terme handeln. Der dritte Term definiert eine Beziehung `liebt/2` zwischen den Atomen `hamster` und `koerner`. Die Reihenfolge der Angabe der Atome ist dabei sehr wichtig, die ein mehrstelliger PROLOG-Term nicht kommutativ ist.

Die komplexen Terme werden in PROLOG, wie bereits angedeutet, dazu verwendet, Beziehungen zwischen Objekten und deren Eigenschaften herzustellen und den vorliegenden Daten eine bessere Strukturierung zu verleihen. In diesem Fall können komplexe Terme auch einfach als Prädikate bezeichnet. Zusätzlich können komplexe Terme aber auch dazu verwendet, Anfragen an das PROLOG-System zu formulieren. Einige Argumente eines komplexen Terms werden dabei als Variablen deklariert, mit der Aufgabe an PROLOG, entsprechende Belegungen für diese zu finden. Wird ein komplexer Term zur Abfrage an PROLOG verwendet, so handelt es sich dabei um einen (Prozedur-) Aufruf.

## Listen

Eine der wichtigsten und meist genutzten Datenstrukturen in PROLOG ist die Liste. Dabei handelt es sich um eine rekursiv definierte Struktur, die zur Darstellung einer beliebig langen, geordneten Menge von Termen verwendet wird. Prinzipiell sind Listen nur spezielle komplexe Terme und können in PROLOG durch die übliche, allgemeine Notation eines Terms ausgedrückt werden. Eine Liste aus drei Elementen a, b und c kann wie folgt geschrieben werden: `.(a,.(b,.(c,.[ ])))`. Die Konstante `[ ]` bezeichnet dabei eine leere Liste. Als Funktor bei einem Listen-Term wird das „.“-Zeichen verwendet. Zu einer besseren Lesbarkeit und Handhabung gibt es in PROLOG für die Liste eine besondere Schreibweise. Die vorherige Liste kann dabei wie folgt geschrieben werden: `[a,b,c]`. Die Elemente der Liste werden voneinander mit Komma getrennt und in eckige Klammern eingeschlossen.

Nachfolgend werden einige Beispiele zur Definition von Listen angegeben.

---

<sup>5</sup>Mit dem vordefinierten Operator `op/3` ist es aber auch möglich alternative Notationen, wie zweistellige Infix-, oder einstellige Präfix-/Postfix-Operatoren zu definieren. In diesem Fall können die Klammern auch weggelassen werden.

L1 = [a,b,[c,d],e,f]} oder L1 = .(a,. (b,. (c,. (d,. []), .(e,. (f))))).

L2 = [[hamster,[liebt,koerner]], [koerner sind braun]].

L3 = [eine, mauer, ist, [vorne,hinten,rechts,links]].

### 8.2.1.2 Aufbau des Programms

Ein PROLOG-Programm besteht aus einer Menge von *Fakten*, *Regeln* und *Abfragen*. Jedes dieser Elemente kann durch eine spezielle Form einer PROLOG-Klausel ausgedrückt werden. Die allgemeine Form einer PROLOG-Klausel hat die folgende Struktur

$$\text{Klauselkopf} \text{ :- Klauselkörper.}$$

Eine PROLOG-Klausel in genau dieser Form, mit sowohl einem Klauselkopf als auch einem Klauselkörper, wird als *Regel* bezeichnet. Bei einer Klausel ohne Klauselkörper handelt es sich um ein *Fakt*. Sollte hingegen der Klauselkopf fehlen, so handelt es sich bei einer solchen Definition um eine *Anfrage* an das PROLOG-System. Die *Abfrage*-Klausel wird dabei auch *Zielklausel* genannt.

Sehen wir uns mal als Nächstes die Zusammensetzung einer Klausel am Beispiel einer abstrakten *Regel*-Klausel an. Eine Regel-Klausel kann in PROLOG wie folgt geschrieben werden.

$$A \text{ :- } B_1, \dots, B_n.$$

Der Kopf einer Regel definiert ein einzelnes Prädikat, welches im Nachfolgenden mittels dieser Regel abgeleitet werden kann. Der Regelkörper umfasst eine Liste von Prädikaten, die für die Erfüllung dieser Regel ebenfalls erfüllt werden müssen. Das „:-“-Symbol fungiert dabei als ein umgekehrtes Implikationszeichen ( $\leftarrow$ ). Dies bedeutet, dass zur Erfüllung von A, alle Prädikate von  $B_1$  bis  $B_n$  erfüllt werden müssen.

### Deklarative und prozedurale Sicht auf das Programm

Bei der Interpretation der oben dargestellten Regel können in PROLOG zwei unterschiedliche Sichtweisen eingesetzt werden. Die *deklarative* Sichtweise besagt, dass das Prädikat A nur dann erfüllt ist, wenn alle *Zielprädikate*  $B_1$  bis  $B_n$  erfüllt werden können. Hierbei ist die Reihenfolge der Erfüllung der  $B_i$ -Prädikate nicht von Bedeutung. Die deklarative Sicht hat einen rein informellen Charakter. Aus der anderen, der *prozeduralen* Sichtweise, kann das Prädikat A nur dann erfüllt werden, wenn zunächst  $B_1$ , dann  $B_2$ , und dann schließlich die  $B_n$ -Klausel erfüllt werden können. Sollte also eines der  $B_i$  Prädikate nicht zu erfüllen sein, so braucht der Rest der  $B_i$ -Prädikate gar nicht mehr

betrachtet werden. Die Auswertung der Regel würde in diesem Fall mit einer negativen Antwort (*false*) beendet werden. Bei der Abarbeitung einer Regel während der Programmausführung setzt PROLOG für die Definition der Regel-Klauseln die deklarative Sichtweise voraus.

## 8.2.2 Operationale Semantik

Bei der Ausführung eines Programms führt das PROLOG-System für eine zuvor formulierte Zielklausel mehrere Berechnungsschritte durch. Die angegebene Zielklausel wird vorerst nur als These betrachtet, die im Nachfolgenden entweder zu beweisen oder zu widerlegen ist. Das PROLOG-System versucht daraufhin zu prüfen, ob die angegebene Zielklausel sich logisch aus den Prädikaten des vorliegenden Programms ableiten lässt.

Der erste Schritt beim Abarbeiten einer Abfrage besteht darin, die einzelnen Subterme der Zielklausel zur erfüllen. Dafür kommt in PROLOG der Unifikationsalgorithmus zum Einsatz. Der nachfolgende Abschnitt beschreibt das Wesen und die Funktionsweise der Unifikation.

### 8.2.2.1 Unifikation

Die Unifikation bezeichnet ein Vorgehen, bei welchem versucht wird, zwei Terme durch Auswahl gültiger Substitutionen zur Deckung zu bringen. Zwei Terme  $t_1$  und  $t_2$  sind dabei unifizierbar, wenn es eine Substitution  $s$  gibt, sodass gilt:  $s(t_1) = s(t_2)$ . Das Ziel einer Unifikation liegt in der Suche nach einer solchen Substitution, die am wenigsten einschränkend auf die beiden Terme  $t_1$  und  $t_2$  wirkt. Das heißt, alle Variablen innerhalb eines Terms bleiben weiterhin unbelegt, sofern dies nicht unbedingt für die Deckung der Terme notwendig ist. Eine nach diesem Vorgehen gefundene Lösung wird als allgemeinsten Unifikator bezeichnet.

### Beispiele

- Die Terme  $t_1 = f(X, b)$  und  $t_2 = f(a, Y)$  sind unifizierbar, wenn  $X$  durch  $a$  und  $Y$  durch  $b$  ersetzt wird. Die dabei gesuchte Substitution lautet  $\sigma = \{X/a, Y/b\}$ .
- Die Terme  $t_1 = f(a, b)$  und  $t_2 = f(c, Y)$  sind nicht unifizierbar, da keines der ersten Argumente der Terme eine Variable beinhaltet.
- Die Terme  $t_1 = F$  und  $t_2 = f(m(W, d), Y)$  sind unifizierbar, wenn  $F = f(m(W, d), Y)$ ,  $\sigma = \{F/f(m(W, d), Y)\}$ .
- Die Terme  $t_1 = f(m(X), X, Y)$  und  $t_2 = f(Z, a, p(Z))$  sind unifizierbar, wenn  $\sigma = \{X/a, Z/m(a), Y/p(m(a))\}$ .

Auf eine äquivalente Art und Weise erfolgt die Unifikation von Listen. Nachfolgend werden einige Beispiele der Unifikation mit den Listen dargestellt. Bei der Schreibweise der Listen wird dabei das — Symbol zur Trennung vom Kopf und dem Rest der Liste verwendet.



### Beispiele

- $t_1 = [a, b|C]$  und  $t_2 = [D|e, f]$  sind unifizierbar mit  $\sigma = \{C/[e, f], D/[a, b]\}$ .
- $t_1 = [a, b, c]$  und  $t_2 = [A, B|Z]$  sind unifizierbar mit  $\sigma = \{A/a, B/b, Z/c\}$ .
- $t_1 = [a]$  und  $t_2 = [A|B]$  sind unifizierbar mit  $\sigma = \{A/a, B/[]\}$ .

#### 8.2.2.2 Resolutionsalgorithmus

Der Resolutionsalgorithmus bildet die Grundlage für die automatische Beweisführung von PROLOG-Klauseln. Der Algorithmus verwendet das Verfahren der Unifikation sowie das Verfahren der automatischen Rücksetzung (das Backtracking).

Zur Erfüllung einer Zielklausel wird versucht, alle Teilziele einer Klausel nacheinander von links nach rechts abzuarbeiten und zu erfüllen. Für jedes der Teilziele wird versucht, dieses mit den in dem Programm befindlichen Klauseln zu unifizieren. Die Programm-Datenbank wird dabei in einer festen Reihenfolge von oben nach unten durchlaufen. Die Reihenfolge der einzelnen Klauseln im Programm spielt dabei also eine wichtige Rolle.

Beim erfolgreichen Unifizieren eines Teilziels der Zielklausel mit einem Faktum wird nachfolgend mit der Bearbeitung des nächsten Teilziels fortgesetzt. Bei der Unifizierung eines Teilziels mit einer Regel-Klausel wird der gesamte Inhalt des Regelkörpers der Regel an die Stelle des Teilziels geschrieben. Dabei fährt PROLOG mit der Bearbeitung des ersten Teilziels der gerade ausgewählten Regel fort. Im Falle eines Fehlers, sofern kein einziger passender Eintrag in der PROLOG-Datenbank für das aktuelle Ziel gefunden werden konnte, legt das PROLOG-System ein Backtracking-Schritt ein. Beim Backtracking geht das Programm zum letzten bekannten Abzweigungspunkt, an dem weitere Auswahlmöglichkeiten bei der Unifikation der Klauseln bestanden. Dabei werden die alten Variablenbelegungen aufgehoben und die nächste Alternativ-Lösung gewählt.

Ist es möglich, alle Teilziele der Zielklausel auf diese Weise zu erfüllen, so liefert das PROLOG-System die ermittelten Variablenbelegungen zurück und beantwortet die Anfrage mit einem *true*. Andernfalls, nach dem Erschöpfen aller verfügbarer Backtracking-Punkte, schlägt die Bearbeitung der Abfrage fehl und das PROLOG-System liefert ein *false* zurück.

Während der Bearbeitung eines PROLOG-Programms führt das System bei der Suche nach einer Lösung eine Reihe von Unifizierungs- und Backtrackingschritten durch.

#### 8.2.2.3 Backtracking

Das Prinzip des Backtracking-Verfahrens ist, wie bereits umrissen, relativ einfach. Während der Abarbeitung der einzelnen Schritte im Resolutionsalgorithmus werden einzelne Punkte mit alternativen Lösungen gemerkt. Beim nächsten Vorkommen eines Fehlers bei der Unifikation kehrt das Programm zum letzten gewählten Auswahl-Punkt zurück. Die alten Variablenbelegungen werden wieder aufgehoben und die nächste Lösung, falls vorhanden, gewählt.

### Beispiel

```
tier(hamster)
tier(tiger)
tier(fuchs)
tier(antilope)
tier(maus)

jagt(tiger,antilope)
jagt(fuchs,maus)
jagt(fuchs,hamster)

raeuber(X) :- tier(X),jagt(X,_).
```

Die obere Definition eines Räubers ist sehr allgemein gehalten. Jedes Tier, das jagen kann, ist ein Räuber. Bei der Anfrage an das PROLOG-System

```
?- raeuber(Tier).
```

würde das System wie folgt vorgehen: Zur Erfüllung des Prädikats `raeuber(X)` ist es notwendig die Prädikate `tier(X)` und `jagt(X,_)` zu erfüllen. Das erste im Programm vorkommende tier-Prädikat ist das `tier(hamster)`. Daher wird nachfolgend geprüft, ob das Prädikat `jagt(hamster,_)` unifizierbar ist. Da dieses offensichtlich nicht möglich ist und die Unifikation zu einem Fehler führt, wird daraufhin ein Backtracking-Schritt eingeleitet. Das System hat sich zuvor gemerkt, dass bei der Unifikation von `tier(X)` weitere Lösungen möglich waren. Daher kehrt das PROLOG-System zurück an diese Stelle und wählt die nächste verfügbare Belegung für die Variable `X`. In diesem Fall wird `X` mit `tiger` unifiziert. Da es für den Tiger ein entsprechendes Prädikat `jagt(tiger,antilope)` gibt, unterbricht das PROLOG-System die Bearbeitung der Anfrage und liefert die gefundene Lösung an den Benutzer.

```
Tier = tiger ?
```

Mit der Eingabe von `;` kann der Benutzer das PROLOG-System zur Suche nach weiteren Lösungen für die aktuelle Abfrage bewegen. Das System legt dabei ebenfalls ein oder mehrere Backtracking-Schritte ein und versucht weitere gültige Variablenbelegungen zu ermitteln. Die Ausgabe für das aktuelle Beispiel könnte dabei wie folgt aussehen.

```
Tier = tiger ? ;
Tier = fuchs ? ;
false.
```

### 8.2.3 Systemprädikate

PROLOG ist eine moderne, sich für praktische Aufgaben eignende Programmiersprache. Neben den grundlegenden Funktionalitäten bietet PROLOG wie auch viele andere Programmiersprachen eine Menge zusätzlicher Steuerungsmechanismen zur Unterstützung des Programmierers bei der Umsetzung von notwendigen und oft wiederkehrenden Aufgaben und Funktionen. Für diesen Zweck bietet PROLOG eine Reihe vordefinierter Systemprädikate, auch *Built-In-Prädikate* genannt, die jeweils eine

bestimmte Funktionalität innerhalb des PROLOG-Systems realisieren. Es handelt sich dabei beispielsweise um die grundlegenden Möglichkeiten der Interaktion des Benutzers mit dem System, die Möglichkeit zur Ein- und Ausgabe von Daten, grundlegende Manipulationsmöglichkeiten in der PROLOG-Datenbank, Steuerung der Programmausführung und ähnliche Funktionen. Die Systemprädikate können vom Programmierer zwar verwendet aber nicht mehr umdefiniert werden. Neben den Systemprädikaten eines „Standard“-PROLOG-Systems, welche von den meisten PROLOG-Systemen angeboten werden, bieten einzelne PROLOG-Implementierungen auch noch weitere vordefinierten Prädikate. Auf diese speziellen Prädikate wird aber im Weiteren nicht näher eingegangen.

Die vom PROLOG-System angebotene Systemprädikate lassen sich im Allgemeinen in mehrere Kategorien unterscheiden. Diese unterteilen sich in

- Prädikate ohne logische Bedeutung (Die Seiteneffekte sind entscheidend)
- Testprädikate
- Prädikate zur Auswertung und Vergleich arithmetischer Ausdrücke
- Steuerung- und Metaprädikate
- Prädikate zur Manipulation von Listen

Die Gesamtzahl aller vom PROLOG-System angebotenen Systemprädikate ist sehr groß. Es wäre kaum sinnvoll und auch möglich alle diese Prädikate hier im einzelnen aufzuführen und zu erläutern. Nachfolgend werden nur die einzelnen Kategorien grob beschrieben und einige wichtigsten Repräsentanten dieser Kategorien genannt und erläutert. Eine umfassendere Übersicht über die Systemprädikate in PROLOG kann beispielsweise in der erweiterten Literatur **[Fehler! Verweisquelle konnte nicht gefunden werden., Fehler! Verweisquelle konnte nicht gefunden werden., Fehler! Verweisquelle konnte nicht gefunden werden.]** gewonnen werden.

Zu der ersten Kategorie gehören solche Systemprädikate, die keine explizite logische Bedeutung besitzen und nur durch ihre Ausführung gewisse Seiteneffekte im System hervorrufen. Dazu gehören in erster Linie die Prädikate zur Steuerung der Ein- und Ausgabe in der Konsole, Prädikate zu Erstellung und Verwaltung von Dateien im Betriebssystem sowie einige allgemeinen Funktionen des Interpreters. Nachfolgend werden einige davon aufgelistet.

listing	Auflisten aller bekannten Programmklauseln.
listing(P)	Auflisten aller bekannten Programmklauseln für die Prozedur P
read(T)	Einlesen eines Terms T.
write(T)	Ausgabe des Terms T auf die Konsole.
tab(N)	Ausgabe von N Leerzeichen.
nl	Nächste Ausgabe erfolgt in einer neuen Zeile.
see(F)	Umschalten des Eingabestroms auf die Datei F.
seen	Schließen des aktuellen Eingabestroms. Umschalten auf die

	(Konsole).
tell(F)	Schließen des Ausgabestroms auf die Datei F.
told	Der aktuelle Ausgabestrom wird geschlossen. Umschalten auf die (Konsole)
trace,notrace	Ein/Ausschalten des -tracing-Mode.
debug,nodebug	Ein/Ausschalten des -debugging-Mode.

Die zweite Kategorie beinhaltet Prädikate zur Prüfung der unterschiedlichen Term-Typen sowie der Gleichheit und Ungleichheit (aus der sich die Unifikation) von Termen. Dabei kann geprüft werden, ob es sich bei einem Term T um ein Atom, eine Zahl oder Variable handelt. Weiterhin gibt es Operatoren zur manuellen Unifikationsprüfung von Termen.

atom(T)	Prüft, ob T ein PROLOG-Atom ist.
atomic(T)	Prüft, ob T eine Konstante oder Zahl ist.
number(T)	Prüft, ob T eine Zahl ist.
integer(T)	Prüft, ob T eine ganze Zahl ist.
var(T),nonvar(T)	Prüft, ob T eine Variable bzw. keine Variable ist.
$T == U, T \neq U$	Prüft, ob T und U identisch bzw. nicht identisch sind.
$T = U$	Prüft, ob T und U unifizierbar sind.

Bei der dritten Kategorie handelt es sich um Prädikate zur Berechnung arithmetischer Funktionen sowie Operatoren zum Vergleich und Auswertung von Zahlen- sowie entsprechender Variablen-Terme. Nachfolgend werden einige davon aufgelistet.

$X == Y$	Ist wahr, wenn der Wert von X gleich dem Wert von Y ist.
$X \neq Y$	Ist wahr, wenn der Wert von X ungleich dem Wert von Y ist.
$X < Y$	Ist wahr, wenn der Wert von X kleiner als der Wert von Y ist.
$X \leq Y$	Ist wahr, wenn der Wert von X kleiner gleich als der Wert von Y ist.
$X > Y$	Ist wahr, wenn der Wert von X größer als der Wert von Y ist.
$X \geq Y$	Ist wahr, wenn der Wert von X größer gleich dem Wert von Y ist.
$V \text{ is } X$	V wird mit dem Wert von X unifiziert. Schlägt fehl, wenn X keine Zahl oder V keine Variable ist.
$\sin(X), \cos(X) \dots$	Liefert jeweils den Wert der gewählten arithmetischen Funktion.

Zu der vierten Kategorie – den Steuerungs- und Metaprädikaten – gehören Prädikate zur allgemeinen Programmsteuerung sowie eine Menge von Prädikaten zur Manipulation der PROLOG-Datenbank. Diese Prädikate finden in der Regel den häufigsten Einsatz im Programm. Die nachfolgenden Beispiele sind nur ein kleiner Ausschnitt aus der Gesamtheit aller Steuerungs- und Metaprädikate. Eine vollständigere Auflistung kann beispielsweise im Buch von Ivan Bratko [**Fehler! Verweisquelle konnte nicht gefunden werden.**] eingesehen werden.

consult(F)	Die Datei F wird eingelesen und die PROLOG-Datenbank um die im Programm befindlichen Klauseln erweitert.
reconsult(F)	Analog zur consult-Klauseln mit identischer Signatur werden dabei aber überschrieben (aktualisiert).
assert(C)	Die Klausel C wird am Ende des aktuellen Programms (Wissensbasis) hinzugefügt.
retract(C)	Das erste Vorkommen der Klausel C im Programm (PROLOG-Datenbank) wird gelöscht.
retractall(F)	Alle Vorkommen der Klausel C im Programm werden gelöscht.
abolish(P,N)	Eine Prozedur P mit der Stelligkeit N wird vollständig aus dem Programm gelöscht.
not(G)	Ist wahr, wenn das Prädikat G nicht erfüllbar ist.
true	Ein Prädikat, das immer erfüllt ist.
fail	Schlägt immer fehl. Erzwingt ein Backtracking-Schritt.
!	Das Cut-Prädikat ist immer erfüllt. Verhindert späteres Backtracking.
P ; Q	Oder-Verknüpfung zweier Prozeduren

Weiterhin bieten die meisten PROLOG-Systeme vordefinierte Systemprädikate für den Umgang mit Listen-Konstrukten. Diese Prädikate können aber auch relativ leicht manuell im eigenen Programm nachimplementiert werden. Die am häufigsten verwendeten Prädikate sind

member(X,L)	Prüft, ob X ein Element der Liste L.
append(L1,L2,L3)	Konkateniert die Listen L1 und L2 zusammen in der Ergebnisliste L3.
reverse(L1,L2)	L2 ist die invertierte Liste von L1.
last(X,L)	Liefert X als letztes Element der Liste L.

## 8.2.4 Trace/Boxenmodell

In PROLOG gibt es eine Möglichkeit, sich die Einzelheiten der Ausführung einer Abfrage näher anzuschauen und somit das Programm auf eventuelle Fehler zu untersuchen. Mit Hilfe des *trace*-Prädikats kann das PROLOG-System in den sogenannten „Protokollierungs“-Modus versetzt werden. Während des *tracing*-Mode macht das System für jeden ausgeführten Schritt zusätzliche Ausgaben auf die Konsole. Anhand dieser Ausgaben behält der Programmierer die Übersicht über den aktuellen Ausführungsfortschritt und kann durch Interaktion mit dem System die Abarbeitung des aktuellen Programms in eine von ihm gewünschte Richtung lenken. Die im *tracing*-Modus getätigte Ausgabe des PROLOG-Systems wird durch das Boxenmodell beschrieben.

Das Boxenmodell stellt ein vereinfachtes Modell zur grafischen Visualisierung der Ausführung einer PROLOG-Klausel dar. Eine einzelne Box repräsentiert dabei eine PROLOG-Klausel. Die Box hat mehrere Ein- und Ausgabe-Ports, die vom System während der Bearbeitung der Klausel, abhängig vom aktuellen Zustand und den Eingaben des Nutzers, genommen werden können. Die Bedeutung der einzelnen Ports wird nachfolgend erläutert.

**CALL** Beim Aufruf einer Klausel wird für diese eine neue Box erstellt und über den *CALL*-Port betreten. Jede erstellte Box bekommt eine fortlaufend vergebene Nummer zugewiesen. Besteht die aktuelle Klausel aus mehreren Unterzielen, so werden für diese innerhalb der aktuellen Box weitere Boxen angelegt.

**EXIT** Bei der erfolgreichen Bearbeitung der Klausel verlässt das System die Box über den *EXIT*-Port. Ist die aktuelle Klausel nur ein Teil einer größeren Abfrage, so wird anschließend das nächste Unterziel (Klausel) der Abfrage über den *CALL*-Port aufgerufen.

**FAIL** Konnte die aktuelle Klausel nicht wahr gemacht werden, so verlässt das System über diesen Ausgang die Box. Bei der darüber liegenden Klausel wird dabei beim Backtracking das *REDO*-Port betreten, um eine Neuauswertung der darüber liegenden Klausel zu initiieren.

**REDO** Beim Backtracking kommt das System über diesen Port zu seinem früheren Ziel zurück.

Dem Programmierer stehen zudem mehrere Möglichkeiten zur Verfügung, die Bearbeitung der aktuellen Klausel zu beeinflussen bzw. zu dirigieren.

**creep**-Aktion: Bei der Wahl dieser Aktion setzt das System die Bearbeitung der Klausel fort, bis das nächste Port-Ereignis registriert wird.

**skip**-Aktion: Die Protokollierung der Ausführung der aktuellen Klausel wird übersprungen.

**retry**-Aktion: Mit dieser Anweisung kann der Programmierer ein Backtracking-Schritt erzwingen. Das PROLOG-System kehrt dabei zurück und versucht die aktuelle Klausel erneut zu erfüllen.

**fail**-Aktion: Mit der *fail*-Aktion kann der Programmierer das Fehlschlagen des aktuellen Ziels erzwingen. Dabei wird das PROLOG-System, sofern es möglich ist, ein Backtracking-Schritt einleiten und die nächste Alternativlösung präsentieren.

Ausgehend vom Programm-Beispiel im Abschnitt 8.2.2.3 nachfolgend eine Anfrage im trace-Modus gestartet. Das PROLOG-System erstellt dabei die folgende Ausgabe. An den einzelnen Interaktionspunkten wurde hier stets die *creep*-Aktion gewählt.

```
?- trace, raeuber(fuchs).  
  Call: (8) raeuber(fuchs) ? creep  
  Call: (9) tier(fuchs) ? creep  
  Exit: (9) tier(fuchs) ? creep  
  Call: (9) jagt(fuchs, _L186) ? creep  
  Exit: (9) jagt(fuchs, maus) ? creep  
  Exit: (8) raeuber(fuchs) ? creep  
true.
```

## 8.2.5 Kontrollfluss

Bei den prozeduralen Programmiersprachen gibt es eine Reihe unterschiedlicher Kontrollstrukturen, um Fallunterscheidungen durchführen zu können. Das *if-then-else*-Konstrukt ist eines davon. In PROLOG gibt es ebenfalls eine Möglichkeit, diesen Konstrukt mit Hilfe von Prozeduren nachzubilden. Das nachfolgende Beispiel stellt eine Implementierung dieses Konstrukts dar.

```
if_then_else(P,Q,R) :- P, Q.           % C1  
if_then_else(P,Q,R) :- not(P), R.     % C2
```

Diese Lösung ist aber nicht sehr effizient, da bei der Bearbeitung der Prozedur auch unnötige Berechnungen durchgeführt werden. Nehmen wir an, dass während der Bearbeitung die erste Klausel (C1) ausgewählt und das erste Prädikat P bereits als wahr ausgewertet wurde. Das Prädikat Q schlägt aber nachfolgend fehl. In diesem Fall würde das PROLOG-System im Rahmen des Backtracking zurückgehen und eine alternative Herleitung für das Prädikat P suchen. Dies ist aber unnötig, da wir schon wissen, dass P bereits bewiesen werden kann. Im anderen Fall, sollte die erste Berechnung von P fehlschlagen, würde das PROLOG-System mit der Bearbeitung der zweiten Klausel der Prozedur `if_then_else(P,Q,R)` (C2) fortfahren. Hier findet aber nochmals die Berechnung von `not(P)` statt, was ebenfalls überflüssig ist, da P bereits bei der ersten Klausel als falsch ausgewertet wurde.

Zur Behandlung solcher Probleme gibt es in PROLOG den Cut-Prädikat. Dieser wird im nachfolgenden Abschnitt näher beschrieben.

### 8.2.5.1 Cut-Prädikat

Der Cut ist ein nullstelliges Prädikat, welches in PROLOG im Programm mit einem „!“-Zeichen geschrieben wird. Dieses Prädikat ist immer wahr und es hat die Eigenschaft, das Backtracking im Programm zu verhindern. Die Funktionsweise des Prädikats an sich ist sehr einfach: Für alle Ziele vor einem Cut-Symbol werden keine Alternativlösungen, auch wenn diese vorhanden sind, angenommen. Die Variablenbelegungen vor dem Cut können nicht mehr aufgehoben werden. Das in dem oberen Abschnitt aufgeführte Beispiel kann mit dem Cut-Prädikat wie folgt ergänzt werden:

```
if_then_else(P,Q,R) :- P, !, Q.       % C1
```

```
if_then_else(P,Q,R) :- R.                                % C2
```

Diese Implementierung ist deutlich effizienter. Beim Fehlschlagen des Prädikats Q analog wie im oberen Beispiel findet in diesem Fall kein Backtracking mehr statt. Die Antwort auf die Abfrage würde dabei *false* sein. Im zweiten Fall, wenn das Prädikat P in der Klausel C1 fehlschlägt, wird direkt mit der Bearbeitung von R in der Klausel C2 fortgesetzt. Bei dieser Implementierung finden also keine zusätzlichen Berechnungen statt.

Bei der Verwendung von Cuts unterscheidet man zwischen den *grünen* und *roten* Cuts. Die *grünen* Cuts sind optional und können auch weggelassen werden, ohne dass die Bedeutung des Programms in irgendeiner Weise verändert wird. Die *grünen* Cuts beschleunigen die Abarbeitung des Programms, in dem bestimmte Teile des Suchbaums, die keine Lösungen enthalten, abgeschnitten werden. Die *roten* Cuts sind ein essentieller Bestandteil der Programmlogik und können daher nicht weggelassen werden. Bei den *roten* Cuts werden auch solche Teile des Suchbaums beschnitten, die potentielle Lösungen enthalten. Bei der Programmierung in PROLOG gilt es daher, die Verwendung der *roten* Cuts zu vermeiden.

#### 8.2.5.2 Negation

Das Systemprädikat not/1 bietet dem Anwender die Möglichkeit zum Ausdrücken der Negationen von Aussagen. Das not-Prädikat darf aber dabei keineswegs als logische Verneinung einer Aussage aufgefasst werden. Die einzig korrekte Interpretation der Aussage not(X) ist die folgende: not(X) ist erfüllt, wenn X fehlschlägt. In PROLOG geht man von der Annahme einer „geschlossenen“ Welt. Das heißt, alles was sich nicht aus dem Programm herleiten lässt, ist automatisch nicht wahr. Doch in der realen Welt trifft diese Annahme nicht immer zu. Bei der Übertragung der Probleme der realen Welt in PROLOG hat man also darauf zu achten, dass bei der Programmentwicklung keine logischen Denkfehler einschleichen und die Benutzung des not-Prädikats im Sinne der Aufgabenstellung stets zum korrekten Ergebnis führt.

### 8.2.6 Informationen zu Prolog im WWW

Learn Prolog now!: <http://www.learnprolognow.org/>

Prolog Programming A first course: <http://computing.unn.ac.uk/staff/cgpb4/prologbook/>

Prolog Beispielsammlung:

[http://www.csupomona.edu/~jrfisher/www/prolog\\_tutorial/contents.html](http://www.csupomona.edu/~jrfisher/www/prolog_tutorial/contents.html)

Prolog Tutorial:

[http://www.ifi.uzh.ch/req/courses/logische\\_programmierung/ws03/documents/Prolog\\_Tutorial.pdf](http://www.ifi.uzh.ch/req/courses/logische_programmierung/ws03/documents/Prolog_Tutorial.pdf)



## 8.3 Prolog-Hamster-Modell

Der Zustand und die Befehle der Hamster-Welt werden im Prolog-Hamster-Modell durch entsprechende Fakten und Regeln in einer vordefinierten (und automatisch zur Verfügung stehenden) Datenbasis repräsentiert. Diese hat folgende Gestalt:

```
%%% Modellierung der Basisbefehle des Java-Hamsters.
%%%
%%% Modellierung des Hamsterterritoriums.
%
% territorium(<anzahlReihen>,<anzahlSpalten>)

:- dynamic territorium(_,_).

%%% Modellierung eines Kornes im Hamsterterritorium.
%
% korn(<reihe>,<spalte>,<anzahl>)

:- dynamic korn(_,_,_).

%%% Modellierung der Mauern im Hamsterterritorium
%
% mauer(<reihe>,<spalte>)

:- dynamic mauer(_,_).

%%% Modellierung des Hamsters selbst.
%
% hamster(<reihe>,<spalte>,<blickrichtung>,<anzahlKoernerImMaul>)

:- dynamic hamster(_,_,_,_).

%%%
%%% Standard-Aktionen des Hamsters.
%%%

vor :-
    territorium(TReihen,TSpalten),
    hamster(Reihe,Spalte,Blickrichtung,AnzahlKoerner),
    nextPos(Reihe,Spalte,Blickrichtung,ReiheNeu,SpalteNeu),
    ReiheNeu >= 0, ReiheNeu < TReihen,
    SpalteNeu >= 0, SpalteNeu < TSpalten,
    not(mauer(ReiheNeu,SpalteNeu)),
    %% Aktualisiere die Prolog-Datenbank:
    retract(hamster(Reihe,Spalte,Blickrichtung,AnzahlKoerner)),
    assert(hamster(ReiheNeu,SpalteNeu,Blickrichtung,AnzahlKoerner)),
    %% Rufe vor() beim Hamster auf und warte solange dies ausgeführt wird..
    write('prologhamster:vor'),ttyflush,
    read(Return),
    call(Return),!.

vor :-
    %% Rufe vor() beim Hamster auf und warte solange dies ausgeführt wird..
    write('prologhamster:vor'),
    read(Return),
    call(Return),
    false,!.
```

```

nextPos(Reihe,Spalte,'NORD',ReiheNeu,Spalte) :-
    ReiheNeu is Reihe - 1, !.
nextPos(Reihe,Spalte,'WEST',Reihe,SpalteNeu) :-
    SpalteNeu is Spalte - 1, !.
nextPos(Reihe,Spalte,'SUED',ReiheNeu,Spalte) :-
    ReiheNeu is Reihe + 1, !.
nextPos(Reihe,Spalte,'OST',Reihe,SpalteNeu) :-
    SpalteNeu is Spalte + 1, !.

vornFrei :-
    hamster(Reihe,Spalte,Blickrichtung,_),
    territorium(TReihen,TSpalten),
    nextPos(Reihe,Spalte,Blickrichtung,ReiheNeu,SpalteNeu),
    ReiheNeu >= 0, ReiheNeu < TReihen,
    SpalteNeu >= 0, SpalteNeu < TSpalten,
    not(mauer(ReiheNeu,SpalteNeu)),
    write('prologhamster:vornFrei'),ttyflush,
    read(Return),
    call(Return),!.

linksUm :-
    hamster(Reihe,Spalte,Blickrichtung,AnzahlKoerner),
    dreheHamster(Blickrichtung,BlickrichtungNeu),
    % Aktualisiere die Datenbank:
    retract(hamster(Reihe,Spalte,Blickrichtung,AnzahlKoerner)),
    assert(hamster(Reihe,Spalte,BlickrichtungNeu,AnzahlKoerner)),
    % Rufe linksUm() beim Hamster auf.
    write('prologhamster:linksUm'),ttyflush,
    read(Return),
    call(Return),!.

dreheHamster('NORD','WEST').
dreheHamster('WEST','SUED').
dreheHamster('SUED','OST').
dreheHamster('OST','NORD').

rechtsUm :-
    linksUm,
    linksUm,
    linksUm,!.

nimm :-
    hamster(Reihe,Spalte,Blickrichtung,AnzahlKoernerImMaul),
    korn(Reihe,Spalte,AnzahlKoerner),
    AnzahlKoerner > 0,
    AnzahlKoernerImMaulNeu is AnzahlKoernerImMaul + 1,
    AnzahlKoernerNeu is AnzahlKoerner - 1,
    %% Aktualisiere die Datenbank:
    retract(hamster(Reihe,Spalte,Blickrichtung,AnzahlKoernerImMaul)),
    retract(korn(Reihe,Spalte,AnzahlKoerner)),
    assert(hamster(Reihe,Spalte,Blickrichtung,AnzahlKoernerImMaulNeu)),
    assert(korn(Reihe,Spalte,AnzahlKoernerNeu)),
    %% Rufe nimm() beim Hamster auf.
    write('prologhamster:nimm'),ttyflush,
    read(Return),
    call(Return),!.

nimm :-

```

```

%% Rufe nimm() beim Hamster auf.
write('prologhamster:nimm'),ttyflush,
read(Return),
call(Return),
false, !.

gib :-
    hamster(Reihe,Spalte,Blickrichtung,AnzahlKoernerImMaul),
    korn(Reihe,Spalte,AnzahlKoerner),
    AnzahlKoernerImMaul > 0,
    AnzahlKoernerImMaulNeu is AnzahlKoernerImMaul - 1,
    AnzahlKoernerNeu is AnzahlKoerner + 1,
    %% Aktualisiere die Datenbank:
    retract(hamster(Reihe,Spalte,Blickrichtung,AnzahlKoernerImMaul)),
    retract(korn(Reihe,Spalte,AnzahlKoerner)),
    assert(hamster(Reihe,Spalte,Blickrichtung,AnzahlKoernerImMaulNeu)),
    assert(korn(Reihe,Spalte,AnzahlKoernerNeu)),
    %% Rufe gib() beim Hamster auf.
    write('prologhamster:gib'),ttyflush,
    read(Return),
    call(Return),!.

%% falls es noch keine koerner auf dieser Kachel gibt..
gib :-
    hamster(Reihe,Spalte,Blickrichtung,AnzahlKoernerImMaul),
    AnzahlKoernerImMaul > 0,
    AnzahlKoernerImMaulNeu is AnzahlKoernerImMaul - 1,
    AnzahlKoernerNeu is 1,
    %% Aktualisiere die Datenbank:
    retract(hamster(Reihe,Spalte,Blickrichtung,AnzahlKoernerImMaul)),
    assert(hamster(Reihe,Spalte,Blickrichtung,AnzahlKoernerImMaulNeu)),
    assert(korn(Reihe,Spalte,AnzahlKoernerNeu)),
    %% Rufe gib() beim Hamster auf.
    write('prologhamster:gib'),ttyflush,
    read(Return),
    call(Return),!.

gib :-
    %% Rufe gib() beim Hamster auf.
    write('prologhamster:gib'),ttyflush,
    read(Return),
    call(Return),
    false,!.

kornDa :-
    hamster(Reihe,Spalte,_,_),
    korn(Reihe,Spalte,AnzahlKoerner),
    AnzahlKoerner > 0,
    write('prologhamster:kornDa'),ttyflush,
    read(Return),
    call(Return),!.

maulLeer :-
    hamster(_,_,_,AnzahlKoernerImMaul),
    AnzahlKoernerImMaul = 0,
    write('prologhamster:maulLeer'),ttyflush,
    read(Return),
    call(Return),!.

```

## 8.4 Prolog-Hamster-Programme

Ein Prolog-Hamster-Programm hat immer folgende Gestalt:

```
%% Eintrittspunkt des Programms.  
main :- <ziele>.
```

**main/1** ist das Prädikat, das beim Ausführen eines Prolog-Hamster-Programms aufgerufen wird.

Im folgenden Prolog-Hamster-Programm läuft der Hamster bis zur nächsten Wand.

```
%% Eintrittspunkt des Programms.  
main :- laufeZurWand.  
  
laufeZurWand :-  
    vornFrei, vor, laufeZurWand, !.  
  
laufeZurWand.
```

## 8.5 Prolog-Konsole

Die PrologKonsole bietet einen alternativen Weg zur Steuerung des Hamsters. Sie gibt dabei dem Programmierer die Möglichkeit, direkt mit dem PROLOG-System zu interagieren. Dabei kann über die Benutzung der angebotenen Steuerungsbuttons oder auch durch das direkte Eintippen der Befehle die Suche der (Alternativ-)Lösungen initiiert werden.

Die primäre Aufgabe der PrologKonsole liegt aber an der Fähigkeit, den Programmierer beim Debugging-Prozess der PROLOG-Programme zu unterstützen und diesen Prozess zu visualisieren.

Sie können die Prolog-Konsole über den entsprechenden Menü-Item im Menü „Fenster“ öffnen.

## 8.6 Beispiele

Das folgende Prolog-Hamster-Beispielprogramm zeigt eine Lösung für das Problem, dass der Hamster ein mauerloses Territorium komplett leeren, sprich alle Körner fressen soll. Weitere Beispielprogramme finden Sie in den Beispielprogrammen des Hamster-Simulators.

```
%% Eintrittspunkt des Programms.
main :- leereTerritorium.

%% zusätzliche nützliche Befehle
kehrt :-
    linksUm,
    linksUm.

rechtsUm :-
    kehrt,
    linksUm.

laufeZurWand :-
    vornFrei,
    vor,
    laufeZurWand,
    !.

laufeZurWand.

laufeInEcke :-
    laufeZurWand,
    linksUm,
    laufeZurWand,
    linksUm.

%% zusätzliche Testbefehle (ohne Seiteneffekte)
linksFrei :-
    linksUm,
    vornFrei,
    rechtsUm,
    !.

linksFrei :-
    rechtsUm,
    fail.

rechtsFrei :-
    rechtsUm,
    vornFrei,
```

```

        linksUm,
        !.

rechtsFrei :-
    linksUm,
    fail.

%% Körner sammeln
nimmAlle :-
    kornDa,
    nimm,
    nimmAlle,
    !.

nimmAlle.

%% laufen und sammeln
graseReiheAb :-
    nimmAlle,
    vornFrei,
    vor,
    graseReiheAb,
    !.

graseReiheAb.

leereTerritorium :-
    laufeInEcke,
    graseAlleReihenAb.

leereTerritorium.

graseAlleReihenAb :-
    graseReiheAb,
    kehrt,
    laufeZurWand,
    kehrt,
    linksFrei,
    begibDichInNaechsteReihe,
    !,
    graseAlleReihenAb.

begibDichInNaechsteReihe :-
    linksUm,
    vor,
    rechtsUm.

```

## 9 Noch Fragen?

Eigentlich müsste dieses Benutzungshandbuch alle Fragen im Zusammenhang mit dem Hamster-Simulator klären. Wenn Sie trotzdem noch weitere Fragen oder Probleme haben, schauen Sie doch einfach mal im WWW auf der Website **[www.java-hamster-modell.de](http://www.java-hamster-modell.de)** ins Diskussionsforum. Hier können Sie Ihre Fragen stellen und mit anderen Hamster-Programmierern und den Autoren des Hamster-Buches bzw. dem Entwickler des Hamster-Simulators in Kontakt treten.