Dietrich Boles, Daniel Jasper

Hamster-Simulator

Version 2.9.4

Benutzungshandbuch

02.08.2016

Inhaltsverzeichnis

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** bereitgestellt.

1.2 Der Hamster-Simulator

Auf der Website **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 2.6 bzw. 2.8 ist es jedoch auch möglich, Programme in der funktionalen Programmiersprache Scheme, der logikorientierten Programmiersprache Prolog, der Programmiersprache Python, der Programmiersprache Ruby und der visuellen Programmiersprache Scratch zu entwickeln und zu testen. Wie Sie dies tun können, erfahren Sie in den Abschnitten 7 bis 11. In der Version 2.9 wurde das „Hamstern“ mit endlichen Automaten und Programmablaufplänen ergänzt (siehe die Abschnitt 12 und 13).

1.3 Änderungen in Release 4 von Version 2.9

- Hamster-Programme können nun auch in der Programmiersprache JavaScript geschrieben werden (siehe Kapitel 14)

1.4 Änderungen in Release 3 von Version 2.9

- Es wurden ein paar Fehler beseitigt, die sich bei der Umstellung auf das Java SE 8 eingeschlichen hatten.

- Außerdem sollte die 3D-Ansicht nun wieder auf allen Rechnern funktionieren (Danke an Carsten Noeske!)
- Wenn das Workspace-Verzeichnis nicht existiert, wird es angelegt
- Die Property "workspace" in hamster.properties kann jetzt Umgebungsvariablen enthalten, z.B. workspace=$(HOMEDRIVE)/$(HOMEPATH)/HamsterProgramme

1.5 Änderungen in Version 2.9 gegenüber Version 2.8

Im Wesentlichen wurde der Hamster-Simulator um zwei Erweiterungen ergänzt:

- Es ist nun möglich, Hamster-Programme durch endliche Automaten auszudrücken (siehe Abschnitt 12).
- Es ist nun möglich, Hamster-Programme durch Programmablaufpläne zu beschreiben und auszuführen (siehe Abschnitt 13).

1.6 Änderungen in Release 3 von Version 2.8

Es wurden ein paar Fehler beseitigt, die sich bei der Umstellung auf das JDK 1.7 eingeschlichen hatten.

1.7 Änderungen in Release 2 von Version 2.8

Wiederum wurden ein paar kleine Fehler korrigiert. Wichtige Erweiterungen betreffen die Scratch-Programmierung (siehe Kapitel 11):

- Scratch-Hamster-Programme lassen sich nun ausdrucken.
- Aus Scratch-Hamster-Programmen können nun Java-Hamster-Programme generiert werden.
- Die Beschriftungen der Scratch-Blöcke lassen sich über Properties ändern.

1.8 Änderungen in Version 2.8 gegenüber 2.7

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

- Hamster-Programme können nun auch in der Programmiersprache Python geschrieben werden (siehe Kapitel 9)
- Hamster-Programme können nun auch in der Programmiersprache Ruby geschrieben werden (siehe Kapitel 10)

- Hamster-Programme können nun auch in der visuellen Programmiersprache Scratch geschrieben werden (siehe Kapitel 11)

1.9 Änderungen in Version 2.7 gegenüber 2.6

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

- Im Dateibaum des Editor-Fensters werden nun auch die Territorium-Dateien angezeigt. Beim Anklicken wird das entsprechende Territorium in das Simulation-Fenster geladen.
- Existiert in einem Ordner zu einem Hamster-Programm ein gleichnamiges Territorium, wird beim Öffnen bzw. Aktivieren des Hamster-Programms das Territorium automatisch in das Simulation-Fenster geladen. Im Editor-Fenster gibt es nun entsprechende Funktionen („Sichern mit Territorium“, …) durch dessen Aktivierung das gerade aktive Programm und gleichzeitig auch das aktuelle Territorium in eine gleichnamige Datei gespeichert werden.
- Über ein Property namens `laf` kann das Look-And-Feel (Erscheinungsbild) des Hamster-Simulators angepasst werden (siehe Abschnitt 5.1.13).
- Beim Starten des Hamster-Simulators wird ein SplashScreen angezeigt.

1.10 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).

1.11 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.

1.12 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.

1.13 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. Genaueres 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.

1.14 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.

1.14.1 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 compiliert 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.

1.14.2 Hamstern mit BlueJ

BlueJ (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.

1.15 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><Einfg>` zum Kopieren und `<Shift><Einfg>` 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 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`[1] 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

[1]Aus Kompatibilität zu früheren Versionen des Hamster-Simulators kann diese Property auch **home** genannt werden.

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.

1.16 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.
- 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 02.08.2016) ist die Version 8 (manchmal wird sie auch als Version 1.8 bezeichnet). Sie sollten möglichst immer die aktuellste Version installieren. Notwendige Voraussetzung für die Nutzung des Hamster-Simulator ist die Nutzung mindestens der Version 8. Für Java SE 6 und 7 gibt es noch eine etwas veraltete Version.

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://www.oracle.com/technetwork/java/javase/downloads/index.html

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://www.oracle.com/technetwork/java/javase/downloads/index.html

Sowohl das JDK als auch das JRE werden standardmäßig von der Firma Oracle 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-v29-04-java8.zip“. `.zip`-Dateien sind Dateien, die mehrere andere Dateien komprimiert speichern. Die Datei „hamstersimulator-v29-04-java8.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 drei 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 `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 Java SE 6 und 7

Wenn Sie noch die veraltete Java-Version Java SE 6 bzw. 7 benutzen, laden Sie sich bitte anstelle der Datei „hamstersimulator-v29-03-java8.zip“ die Datei „hamstersimulator-v29-01-java7.zip“ herunter. Das sollte prinzipiell auch noch funktionieren, wird aber in Zukunft nicht weiter unterstützt.

2.4 Hamster-Simulator unter Java SE 4 und 5

Die veraltete Java-Version 1.4 und auch die Version 5 werden nicht weiter unterstützt!

2.5 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. [2]
- 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!

Hinweis:

Wenn es Probleme mit dem Doppelklick unter Windows gibt, kann es sein, dass die Endung .jar dem falschen Programm bzw. Java-Interpreter zugeordnet ist. Testen Sie dies, indem Sie in einem Eingabeaufforderungsfenster folgendes eingeben und in etwa die folgenden Ergebnisse angezeigt bekommen:

Eingabe: `assoc .jar`

Ausgabe: `.jar=jarfile`

Eingabe: `ftype jarfile`

Ausgabe: `jarfile="C:\Program Files\Java\jre\bin\javaw.exe" -jar "%1" %*`

Den Java-Interpreter, der standardmäßig beim Doppelklick auf eine .jar-Datei aufgerufen werden soll, können Sie durch folgende Eingabe in einem Eingabeaufforderungsfenster ändern:

```
ftype jarfile="pfad\javaw.exe" -jar "%1" %*
```

wobei „pfad“ entsprechend Ihrer Java-Installation geändert werden muss, bspw. so:

```
jarfile="C:\Program Files\Java\jdk1.7.0\bin\javaw.exe" -jar "%1" %*
```

---

[2] 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.

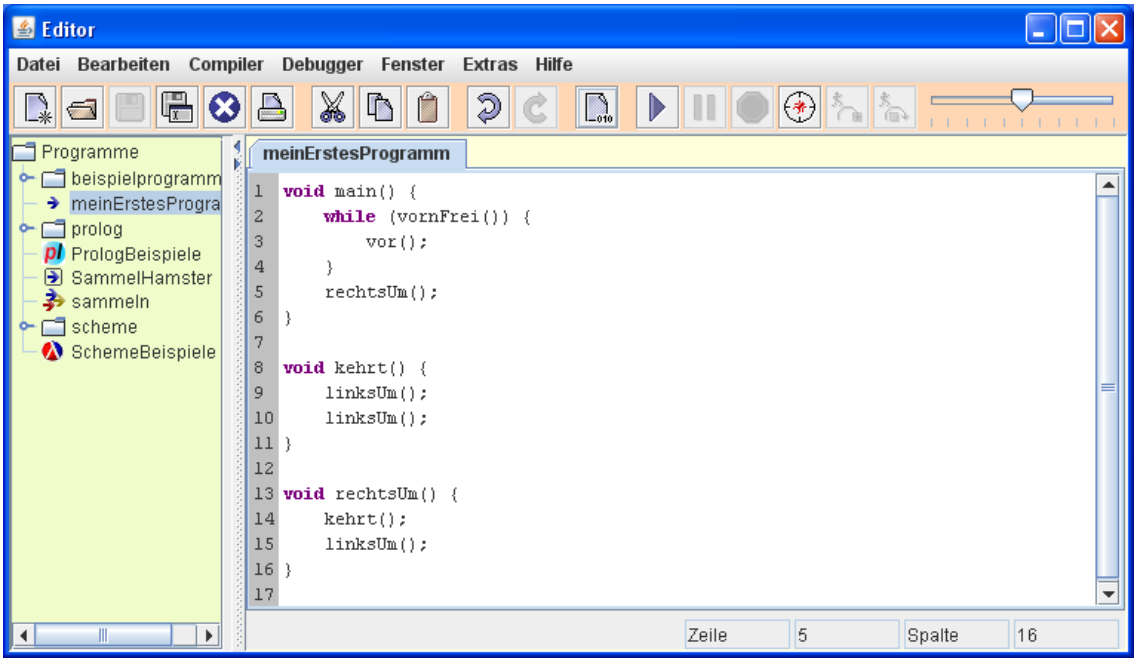

Abbildung 1: Editor-Fenster

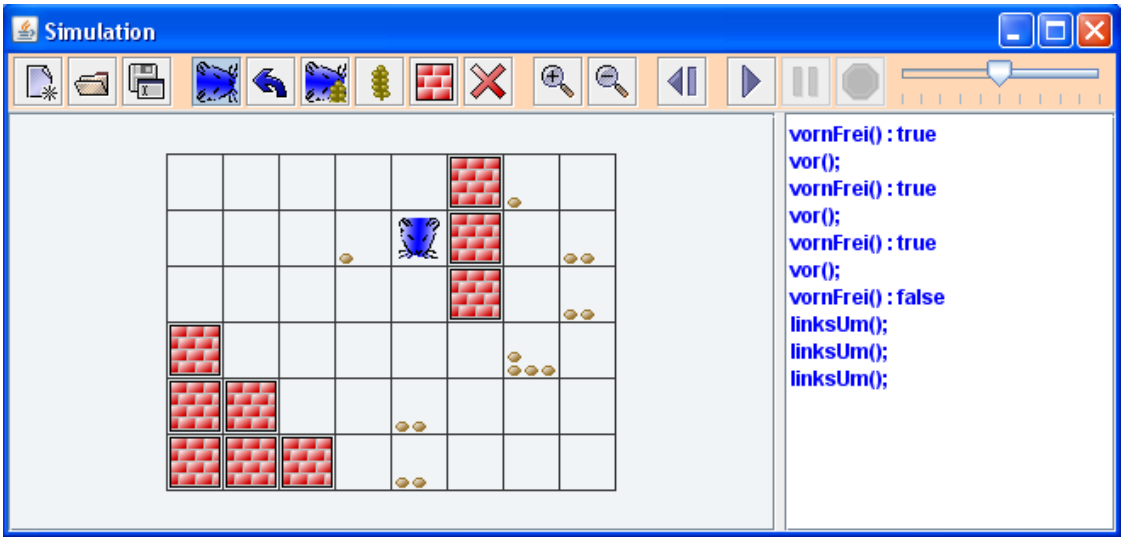

Abbildung 2: Simulation-Fenster

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.

Seit Version 2.7 des Hamster-Simulators gibt es eine Alternative zum Wiederherstellen gespeicherter Territorien: Seit dieser Version werden nämlich Territorium-Dateien auch im Dateibaum des Editor-Fensters angezeigt. Beim Anklicken einer Territorium-Datei im Dateibaum wird das entsprechende Territorium im Simulation-Fenster geladen.

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();
  linksUm();
  vor();
  vor();
  nimm4();
}

void nimm4() {
  nimm();
  nimm();
  nimm();
  nimm();
}
```

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 Eingabebreich.

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 Button 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.

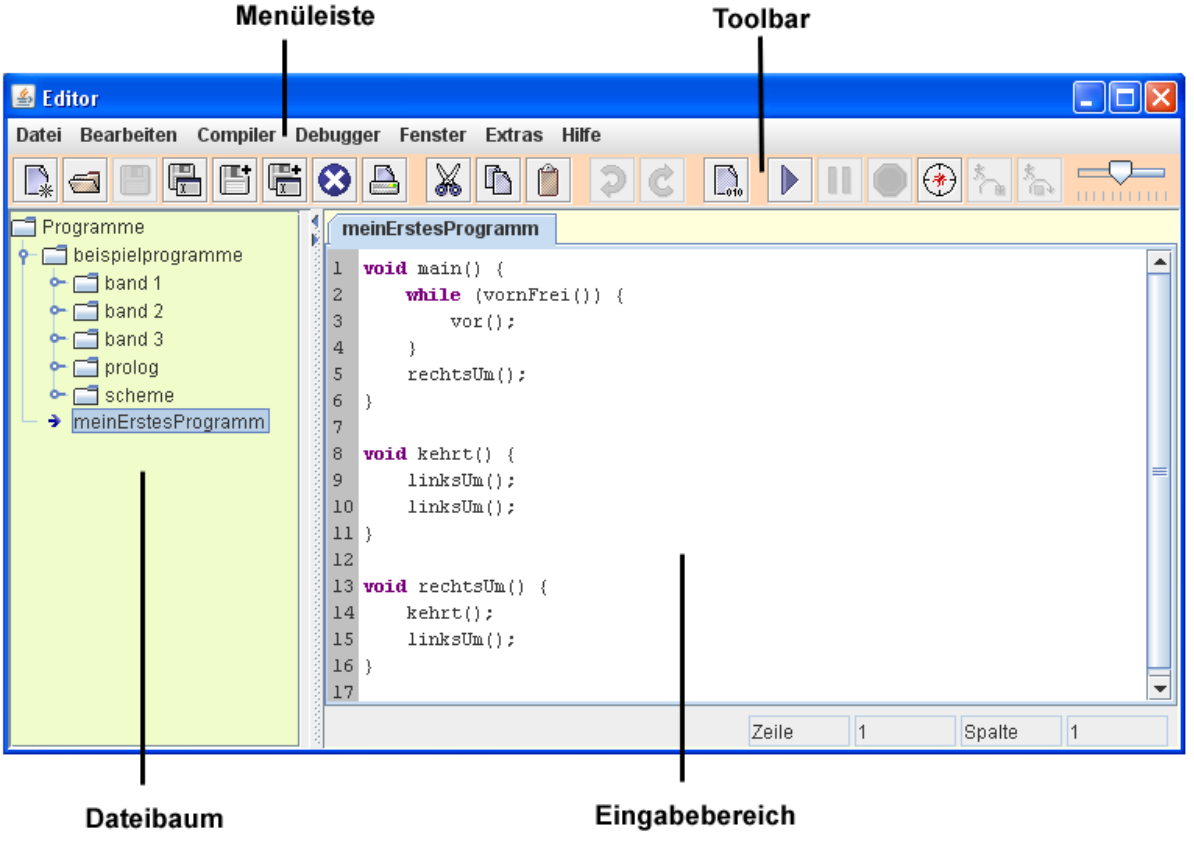

Abbildung 3: Komponenten des Editor-Fensters

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.

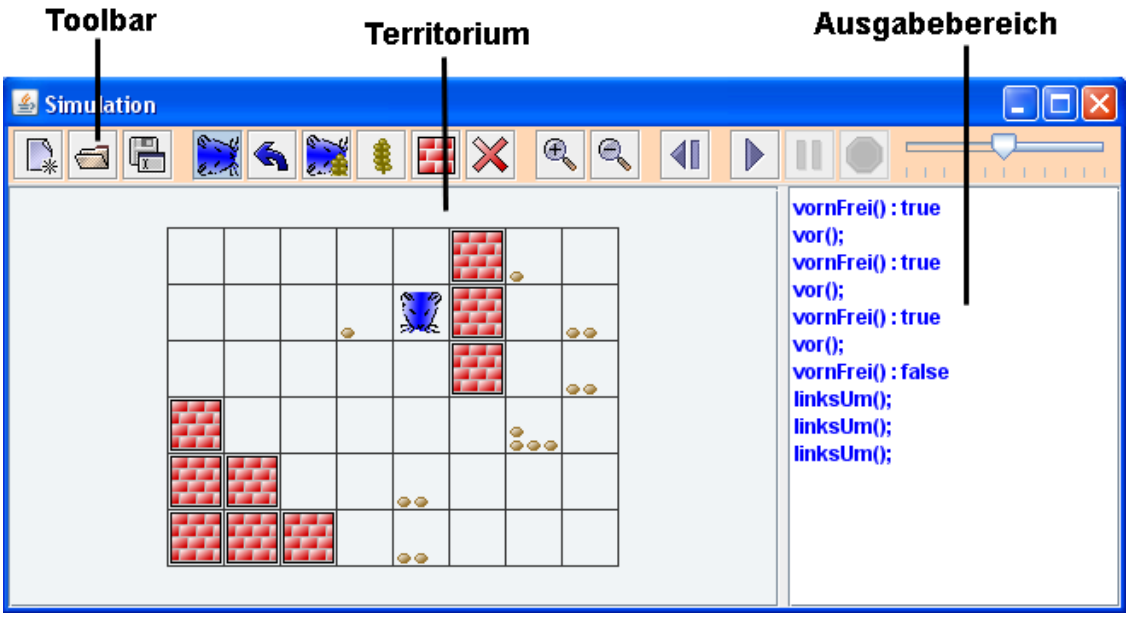

Abbildung 4: Komponenten des Simulation-Fensters

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 Tooltipps

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

Abbildung 5: Tooltipp

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 Tooltipps 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.

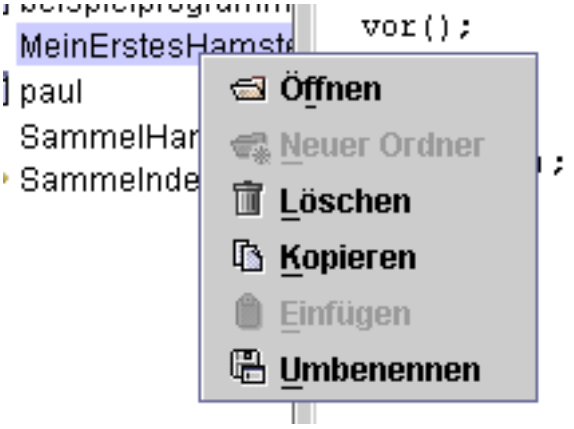

Abbildung 9: Popup-Menü

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 Dialoxbox 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.

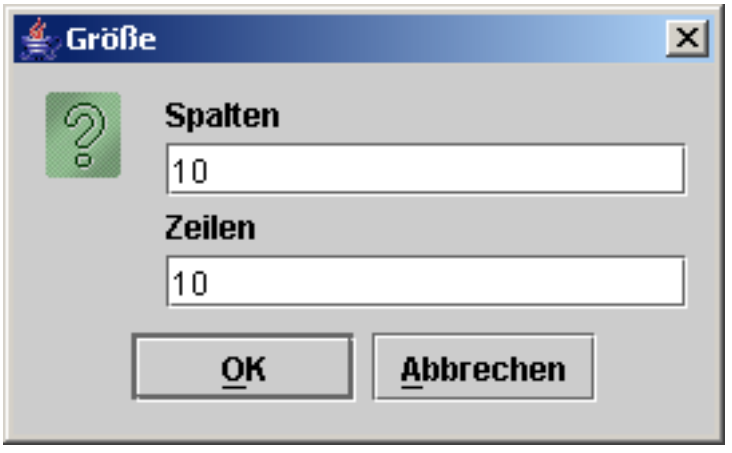

Abbildung 10: Dialogbox mit Eingabefeldern

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.

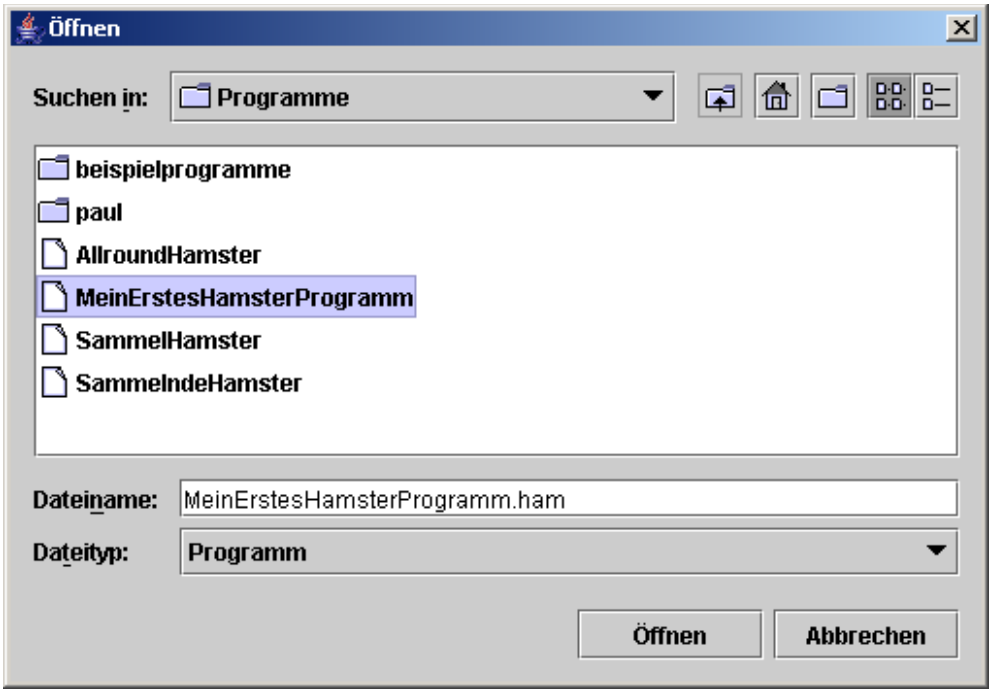

Abbildung 11: Dateiauswahl-Dialogbox

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).

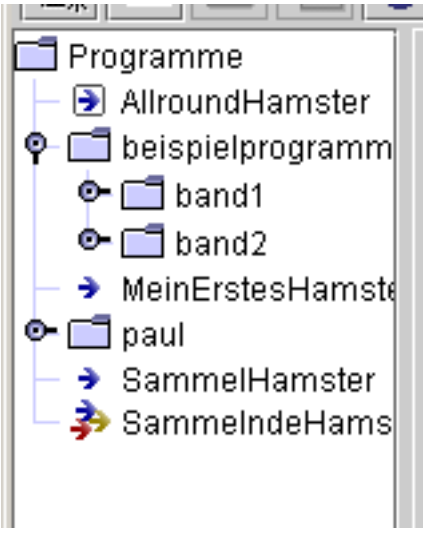

Abbildung 12: Dateibaum

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).

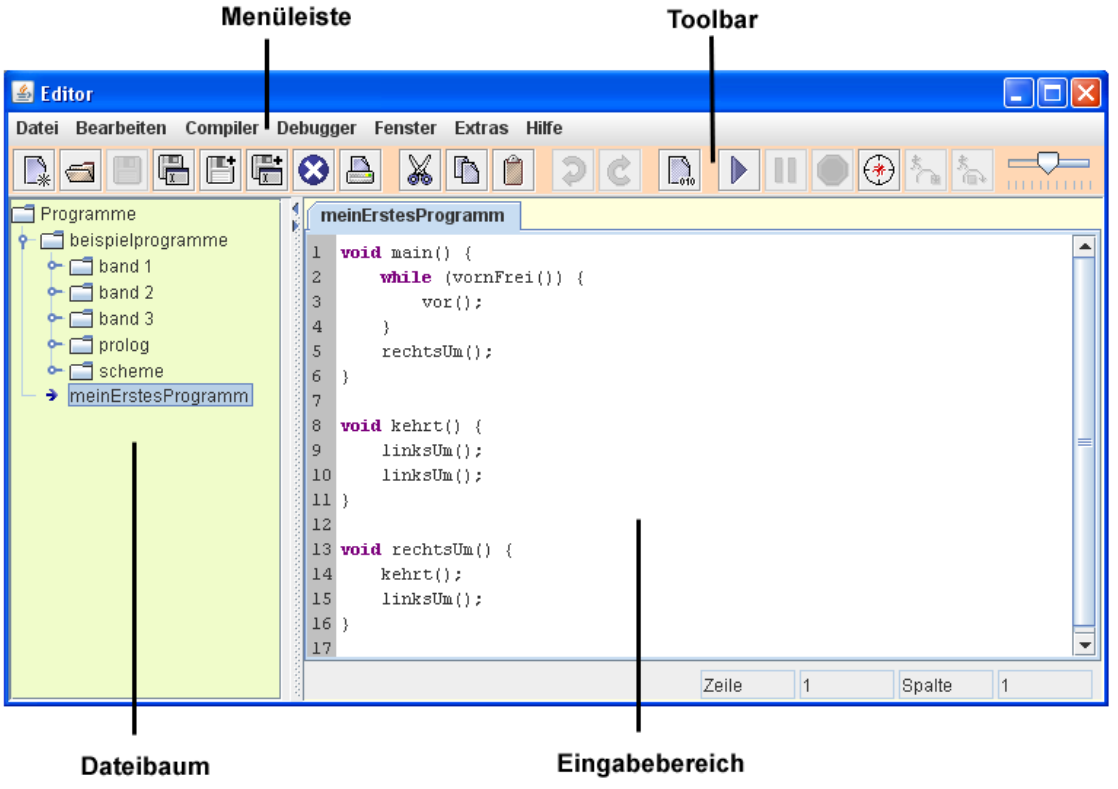

Abbildung 13: Editorfenster

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).

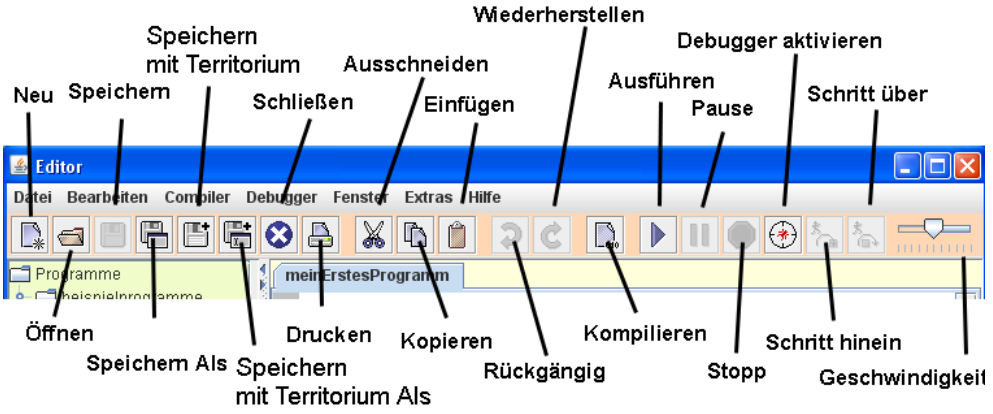

Abbildung 14: Toolbar des Editor-Fensters

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 Stringliterale 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.

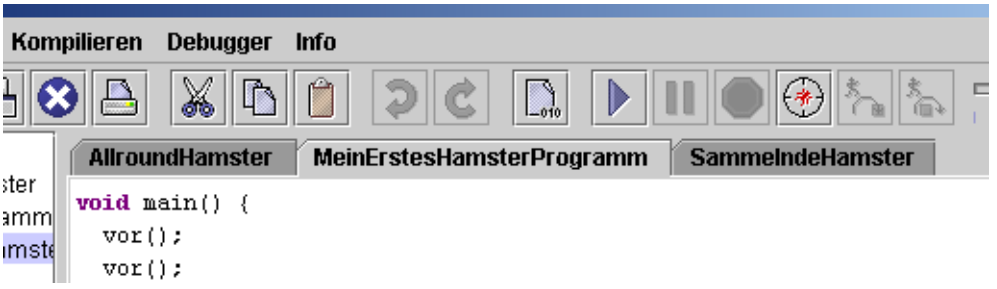

Abbildung 15: Karteireiter

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-Dialog-box, 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 Öffnen eines existierenden Hamster-Programms

Zum Öffnen eines abgespeicherten Hamster-Programms 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 wird das Hamster-Programm angezeigt.

4.2.3 Ändern eines existierenden Hamster-Programms

Um ein Hamster-Programm zu ändern, müssen Sie es zuvor öffnen, so dass der Sourcecode im Eingabebereich erscheint. Im Eingabebereich können Sie nun die gewünschten Änderungen am Sourcecode vornehmen und anschließend mit Hilfe des „Speichern“-Buttons oder des „Speichern Als“-Buttons abspeichern.

4.2.4 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.

4.2.5 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!

4.2.6 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!

4.2.7 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!

4.2.8 Drucken eines Hamster-Programms

Über den „Drucken“-Button (achter 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.

4.2.9 Schließen eines geöffneten Hamster-Programms

Mit dem „Schließen“-Button (siebter 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.

4.2.10 Editier-Funktionen

Im Eingabereich 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 (neunter 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 (zehnter 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 (elfter 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 (zwöfter 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 (dreizehnter 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.

4.2.11 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-Menu des Ordners öffnet. Im Popup-Menü haben Sie nun folgende Funktionalitäten zur Verwaltung von Ordern 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.

4.2.12 Speichern zusammen mit einem Territorium

Mit dem „Speichern mit Territorium“-Button (fünfter Toolbar-Button von links) können Sie das aktuelle Hamster-Programm zusammen mit dem aktuellen Territorium speichern. Das Territorium bekommt dabei denselben Namen wie das Hamster-Programm. Analoges gilt für den „Speichern mit Territorium Als“-Button (sechster Toolbar-Button von links).

4.2.13 Gleichzeitiges Öffnen eines Hamster-Programms und Territoriums

Wenn Sie ein Hamster-Programm öffnen bzw. aktivieren, wird automatisch überprüft, ob es im gleichen Verzeichnis eine zum Hamster-Programm gleichnamige Territorium-Datei gibt. In diesem Fall wird gleichzeitig mit dem Hamster-Programm auch das entsprechende Territorium geladen und im Simulation-Fenster angezeigt.

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 (vierzehnter 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.

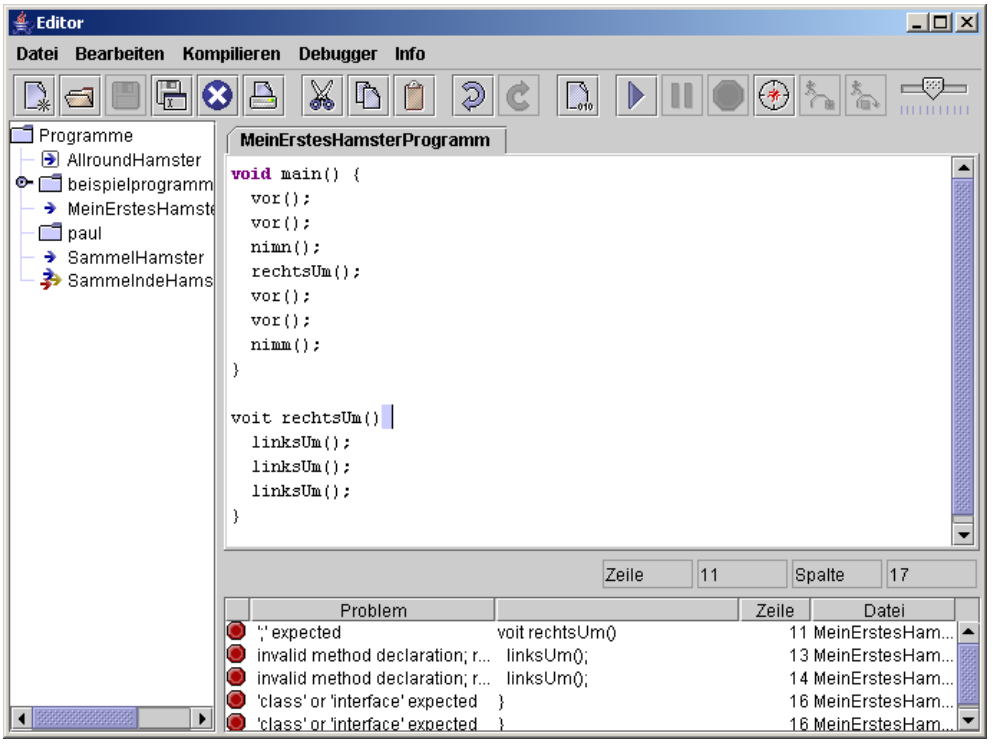

Abbildung 16: Fehlermeldungen

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.

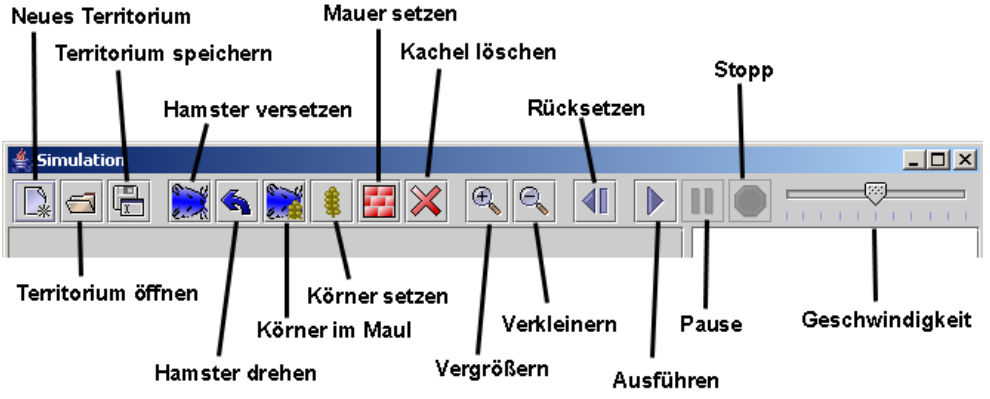

Abbildung 17: Toolbar des Simulation-Fensters

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 Kachel 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 nicht 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`“.

Mit dem „Speichern mit Territorium“-Button des Editor-Fensters haben Sie auch die Möglichkeit, das aktuelle Hamster-Programm zusammen mit dem aktuellen Territorium abzuspeichern. Das Territorium wird dabei in einer dem Hamster-Programm gleichnamigen Territorium-Datei abgespeichert.

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.

Seit Version 2.7 des Hamster-Simulators gibt es eine Alternative zum Wiederherstellen gespeicherter Territorien: Seit dieser Version werden nämlich Territorium-Dateien auch im Dateibaum des Editor-Fensters angezeigt. Beim Anklicken einer Territorium-Datei im Dateibaum wird das entsprechende Territorium im Simulation-Fenster geladen.

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.

Alternativ können Sie auch im Dateibaum des Editor-Fensters ein Popup-Menü oberhalb einer Territorium-Datei aktivieren und hierin die Funktion „Umbenennen“ auswählen.

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 analog zu Hamster-Programmen über den Dateibaum im Editor-Fenster möglich.

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 der 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.

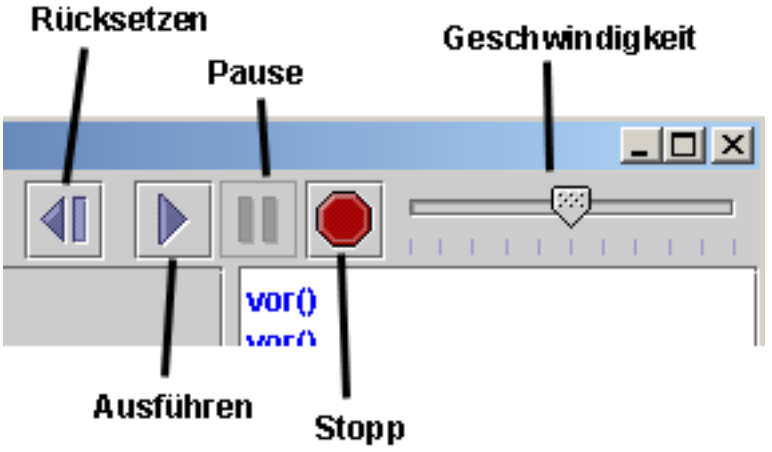

Abbildung 18: Toolbar des Simulation-Fensters

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).

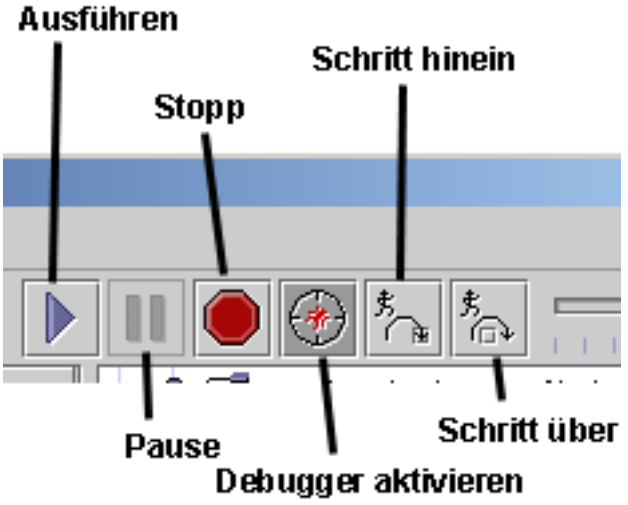

Abbildung 19: Toolbar des Editor-Fensters

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).

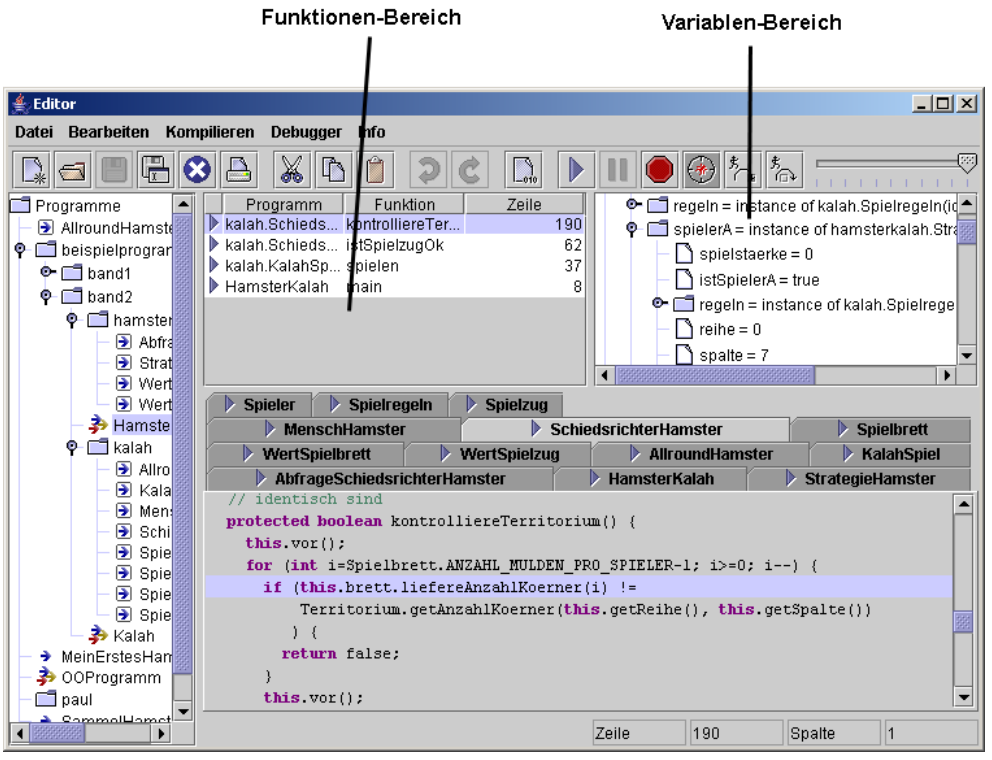

Abbildung 20: Debugging-Fenster

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 gegebenfalls 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 und herzlichen Dank an Carsten Noeske, der Probleme mit der 3D-Simulation in Version 2.9.3 des Hamster-Simulators behoben hat.

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

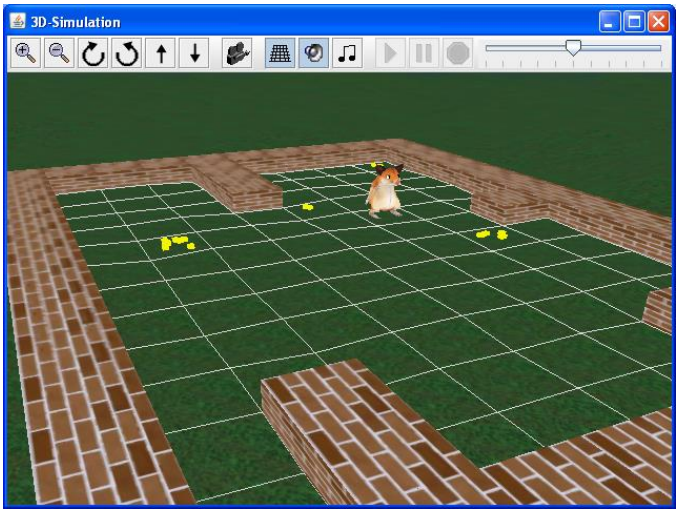

Abbildung 21: 3D-Simulationsfenster

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 hein- 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 Mausrad 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. Properties bestehen immer aus einem Namen und einem Wert, die durch ein Gleichheitszeichen (=) getrennt werden und in derselben Zeile stehen müssen. Tritt ein Name mehrfach in der Property-Datei auf, ist das unterste Vorkommen gültig. Steht am Anfang einer Zeile ein #-Zeichen, wird die Zeile ignoriert.

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`[3] 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*.

Seit Version 2.9.3 des Hamster-Simulators kann die Property "workspace"auch Umgebungsvariablen enthalten, z.B.
workspace=$(HOMEDRIVE)/$(HOMEPATH)/HamsterProgramme

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

[3]Aus Kompatibilität zu früheren Versionen des Hamster-Simulators kann diese Property auch **home** genannt werden.

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.

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 zusaetzlicher 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 vermutlich erst in Version 2.8 des Hamster-Simulators erläutert. Die Funktionalität ist zwar bereits integriert, enthält aber leider noch ein paar Fehler.

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.

5.1.12 plcon

Über die Property „plcon“ kann der Pfad zum SWIProlog-Interpreter angegeben werden. Standardmäßig ist dort der Wert „swipl.exe“ angegeben. Wurde SWIProlog installiert und die PATH-Umgebungsvariable entsprechend angepasst, sollte das unter WINDOWS funktionieren. Für andere Betriebssysteme (LINUX, …) muss hier eventuell der vollständige Pfad zum SWIProlog-Interpreter angegeben werden, bspw. in der Form

`plcon=C:/Programme/pl/bin/swipl.exe` oder

`plcon=C:\\Programme\\pl\\bin\\swipl.exe`

Achtung: In älteren Versionen von SWI-Prolog hieß der SWI-Interpreter nicht „swipl“ sondern „plcon“! Dementsprechend müssten Sie „swipl.exe“ durch „plcon.exe“ ersetzen.

5.1.13 laf

Über die Property „laf“ kann (seit der Version 2.7) das Look-And-Feel des Hamster-Simulators, d.h. sein Erscheinungsbild, verändert werden. Folgende Einstellungen sind dabei möglich:

```
laf=com.sun.java.swing.plaf.nimbus.NimbusLookAndFeel
laf=com.sun.java.swing.plaf.windows.WindowsLookAndFeel
laf=com.sun.java.swing.plaf.motif.MotifLookAndFeel
laf=javax.swing.plaf.metal.MetalLookAndFeel
laf=org.jvnet.substance.skin.SubstanceBusinessLookAndFeel
laf=org.jvnet.substance.skin.SubstanceBusinessBlueSteelLookAndFeel
laf=org.jvnet.substance.skin.SubstanceBusinessBlackSteelLookAndFeel
laf=org.jvnet.substance.skin.SubstanceCremeLookAndFeel
laf=org.jvnet.substance.skin.SubstanceCremeCoffeeLookAndFeel
laf=org.jvnet.substance.skin.SubstanceSaharaLookAndFeel
laf=org.jvnet.substance.skin.SubstanceModerateLookAndFeel
laf=org.jvnet.substance.skin.SubstanceOfficeSilver2007LookAndFeel
laf=org.jvnet.substance.skin.SubstanceOfficeBlue2007LookAndFeel
laf=org.jvnet.substance.skin.SubstanceNebulaLookAndFeel
laf=org.jvnet.substance.skin.SubstanceNebulaBrickWallLookAndFeel
laf=org.jvnet.substance.skin.SubstanceAutumnLookAndFeel
laf=org.jvnet.substance.skin.SubstanceMistSilverLookAndFeel
laf=org.jvnet.substance.skin.SubstanceMistAquaLookAndFeel
laf=org.jvnet.substance.skin.SubstanceDustLookAndFeel
laf=org.jvnet.substance.skin.SubstanceDustCoffeeLookAndFeel
laf=org.jvnet.substance.api.skin.SubstanceGeminiLookAndFeel
laf=org.jvnet.substance.skin.SubstanceRavenGraphiteLookAndFeel
laf=default
```

Standardeinstellung ist „default“. Probieren Sie doch einfach mal das ein oder andere LAF aus. Die meisten LAFs stammen übrigens aus dem Projekt *substance* (https://substance.dev.java.net/)

5.1.14 python

Über die Property „python“ kann die Fähigkeit des Hamster-Simulators Python-Programme zu entwickeln bzw. auszuführen ein- bzw. ausgeschaltet werden (siehe Kapitel 9). Befindet sich in der Datei „hamster.properties“ eine Zeile mit dem Text `python=false`, ist der Python-Modus ausgeschaltet. Befindet sich in der Datei „hamster.properties“ eine Zeile mit dem Text `python=true`, ist der Python-Modus eingeschaltet. Standardmäßig ist der Python-Modus ausgeschaltet.

5.1.15 ruby

Über die Property „ruby“ kann die Fähigkeit des Hamster-Simulators Ruby-Programme zu entwickeln bzw. auszuführen ein- bzw. ausgeschaltet werden (siehe Kapitel 10). Befindet sich in der Datei „hamster.properties“ eine Zeile mit dem Text `ruby=false`, ist der Ruby-Modus ausgeschaltet. Befindet sich in der Datei „hamster.properties“ eine Zeile mit dem Text `ruby=true`, ist der Ruby-Modus eingeschaltet. Standardmäßig ist der Ruby-Modus ausgeschaltet.

5.1.16 scratch

Über die Property „scratch“ kann die Fähigkeit des Hamster-Simulators Scratch-Programme zu entwickeln bzw. auszuführen ein- bzw. ausgeschaltet werden (siehe Kapitel 11). Befindet sich in der Datei „hamster.properties“ eine Zeile mit dem Text `scratch=false`, ist der Scratch-Modus ausgeschaltet. Befindet sich in der Datei „hamster.properties“ eine Zeile mit dem Text `scratch=true`, ist der Scratch-Modus eingeschaltet. Standardmäßig ist der Scratch-Modus eingeschaltet.

Seit Version 2.8.2 des Hamster-Simulators lassen sich die Beschriftungen der Scratch-Blöcke über Properties ändern. Folgende Properties können dabei gesetzt werden (dahinter steht jeweils der Standard-Wert):

```
scratch_new_procedure=Neue Prozedur
scratch_new_function=Neue Funktion
scratch_void_return=verlasse
scratch_bool_return=liefere
scratch_true=wahr
scratch_false=falsch
scratch_and=und
scratch_or=oder
scratch_not=nicht
scratch_if=falls
scratch_else_if=falls
scratch_else=sonst
scratch_while=solange
scratch_do=wiederhole
scratch_do_while=solange
```

5.1.17 fsm

Über die Property „fsm“ kann die Fähigkeit des Hamster-Simulators Hamster-Automaten zu entwickeln bzw. auszuführen ein- bzw. ausgeschaltet werden (siehe Kapitel 12). Befindet sich in der Datei „hamster.properties“ eine Zeile mit dem Text `fsm=false`, ist der Automaten-Modus ausgeschaltet. Befindet sich in der Datei „hamster.properties“ eine Zeile mit dem Text `fsm=true`, ist der Automaten-Modus eingeschaltet. Standardmäßig ist der Ruby-Modus ausgeschaltet.

5.1.18 flowchart

Über die Property „flowchart“ kann die Fähigkeit des Hamster-Simulators Hamster-Programmablaufpläne (PAP) zu entwickeln bzw. auszuführen ein- bzw. ausgeschaltet werden (siehe Kapitel 13). Befindet sich in der Datei „hamster.properties“ eine Zeile mit dem Text `flowchart=false`, ist der PAP-Modus ausgeschaltet. Befindet sich in der Datei „hamster.properties“ eine Zeile mit dem Text `flowchart=true`, ist der PAP-Modus eingeschaltet. Standardmäßig ist der PAP-Modus ausgeschaltet.

5.1.19 javascript

Über die Property „javascript“ kann die Fähigkeit des Hamster-Simulators JavaScript-Programme zu entwickeln bzw. auszuführen ein- bzw. ausgeschaltet werden (siehe Kapitel 14). Befindet sich in der Datei „hamster.properties“ eine Zeile mit dem Text `javascript=false`, ist der JavaScript-Modus ausgeschaltet. Befindet sich in der Datei „hamster.properties“ eine Zeile mit dem Text `javascript=true`, ist der JavaScript-Modus eingeschaltet. Standardmäßig ist der JavaScript-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:

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

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`

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.wwc.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 (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 nicht 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 Listenrepäsentation 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))
)
```

Vergeichen 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 runtergeladen 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 bzw. swipl.exe). Die dafür verantwortliche Anwendungsdatei befindet sich nach der Installation im Unterverzeichnis \bin\ des Installationsverzeichnisses 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 und unter Umständen muss das Property `plcon` auf den vollständigen Pfad zum SWIProlog-Interpreter gesetzt werden (siehe Abschnitt 5.1.11 und 5.1.12).

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. *PROgrammation en LOGique*) 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 weicht 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 gedachtes 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 nach wie vor die Grundlage für die meisten PROLOG-Implementierungen dar. Ein in den 70er Jahren stattgefundenes japanisches Projekt am Forschungsinstitut ICOT[4] 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 traditioneller Rechnerarchitektur, wie es bei den anderen Sprachen der Fall ist, zu

[4] Institute for New Generation Computer Technology

orientieren. Dabei ändert sich der Programmierstil für PROLOG im Vergleich zu den Letzteren sehr stark: Hier wird das eigentliche Problem in den Vordergrund gestellt. 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 setzten 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 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[5]. 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.

---

[5]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

```
Klauselkopf :- 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.

$$A :- B_1, \ldots, B_n.$$

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 (←). 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 allgemeinster 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 σ={*C*/[*e*,*f*],*D*/[*a*,*b*]}.
- t_1 = [*a*,*b*,*c*] und t_2 = [*A*,*B*|*Z*] sind unifizierbar mit σ={*A*/*a*,*B*/*b*,*Z*/*c*}.
- t_1 = [*a*] und t_2 = [*A*|*B*] sind unifizierbar mit σ={*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,\s\do5()) 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 Weiten 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 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 der 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 \== 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 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 =\= 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 =< 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 >= Y	Ist wahr, wenn der Wert von X größer gleich dem Wert von Y ist.
V is X	V wird mit dem Wert von X unifiziert. Schägt fehl, wenn X keine Zahl oder V keine Variable ist.
sin(X),cos(X)…	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 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 zweiter 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 prozedurallen 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) augewählt 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) forfahren. 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 war 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 Auszudrücken 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

Prolog Tutorial:
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 Korns 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 Python

Die Programmiersprache *Python* wurde Anfang der 1990er Jahre von Guido van Rossum am Centrum Wiskunde & Informatica (Zentrum für Mathematik und Informatik) in Amsterdam als Nachfolger für die Programmier-Lehrsprache ABC entwickelt und war ursprünglich für das verteilte Betriebssystem Amoeba gedacht. Der Name geht nicht etwa auf die gleichnamige Schlangengattung (Pythons) zurück, sondern bezog sich ursprünglich auf die englische Komikertruppe Monty Python. Trotzdem etablierte sich die Assoziation zur Schlange.

Python wurde mit dem Ziel entworfen, möglichst einfach und übersichtlich zu sein. Dies soll durch zwei Maßnahmen erreicht werden: Zum einen kommt die Sprache mit relativ wenigen Schlüsselwörtern aus, zum anderen ist die Syntax reduziert und auf Übersichtlichkeit optimiert. Dies führt dazu, dass Python eine Sprache ist, in der schnell und einfach programmiert werden kann. Sie ist daher besonders dort geeignet, wo Übersichtlichkeit und Lesbarkeit des Codes eine herausragende Rolle spielen – zum Beispiel in der Teamarbeit, bei Beschäftigung mit dem Quelltext nach längeren Pausen oder bei Programmieranfängern.

Python ist eine Multiparadigmensprache. Das heißt, Python zwingt den Programmierer nicht zu einem einzigen bestimmten Programmierparadigma, sondern erlaubt es, das für die jeweilige Aufgabe am besten geeignete Paradigma zu wählen. Objektorientierte und strukturierte Programmierung werden vollständig unterstützt, weiterhin gibt es Spracheigenschaften für funktionale und aspektorientierte Programmierung (aus Wikipedia).

9.1 Die Programmiersprache Python

Im Internet finden Sie viele Informationen rund um die Programmiersprache Python. Schauen Sie einfach mal unter folgenden Links nach:

- http://www.python.org/ (offizielle Webseite)
- http://de.wikipedia.org/wiki/Python_(Programmiersprache) (Wikipedia)
- http://openbook.galileocomputing.de/python/ (Online-Buch)

9.2 Python-Hamster-Programme

Um mit dem Hamster-Simulator Python-Programme entwickeln und ausführen zu können, muss die Property `python` auf `true` gesetzt sein. Standardmäßig ist das nicht der Fall. Öffnen Sie also die Datei `hamster.properties` und ersetzen Sie dort die Zeile `python=false` durch die Zeile `python=true`.

Starten Sie dann den Hamster-Simulator. Wenn Sie ein neues Programm erstellen wollen, wählen Sie in der erscheinenden Dialogbox den Eintrag „Python-Programm“. Im Editor-Bereich erscheint dann ein Feld mit dem Inhalt „`if vornFrei(): vor()`“.

Ersetzen Sie diesen Inhalt durch Ihr Python-Programm und speichern Sie die Datei dann ab. Python-Programme brauchen nicht compiliert zu werden. Nach dem Abspeichern haben Sie direkt die Möglichkeit, das Programm auszuführen. Syntaxfehler werden erst zur Laufzeit angezeigt.

Im Hamster-Simulator lassen sich sowohl imperative als auch objektorientierte Python-Programme entwickeln. Auch gemischte Programme sind möglich. Der (imperative) Hamster kennt (wie üblich) die Befehle

```
vor()
linksUm()
gib()
nimm()
vornFrei() (-> bool)
kornDa() (-> bool)
maulLeer() (-> bool)
```

Eine vordefinierte Klasse namens `Hamster` kann für objektorientierte Python-Programme genutzt werden. Die Klasse stellt folgende Methoden zur Verfügung:

```
vor(self)
linksUm(self)
gib(self)
nimm(self)
vornFrei(self) (-> bool)
kornDa(self) (-> bool)
maulLeer(self) (-> bool)
schreib(self, string)
liesZeichenkette(self, string) (-> string)
liesZahl(self, string) (-> int)
getReihe(self) (-> int)
getSpalte(self) (-> int)
getBlickrichtung(self) (-> int: Hamster.NORD, Hamster.SUED,
Hamster.OST, Hamster.WEST)
getAnzahlKoerner(self) (-> int)
getStandardHamster() (-> Hamster)
```

Die Klasse definiert 2 Konstruktoren:

```
Hamster(self, reihe, spalte, blickrichtung, anzahlKoerner)
Hamster(self, hamster)
```

9.2.1 Python-Bibliothek einbinden

Wenn Sie Module aus der Standard-Bibliothek von Python nutzen und importieren wollen, müssen Sie zusätzlich folgendes tun:

1. Sie müssen Jython downloaden: http://www.jython.org/

2. Sie müssen Jython installieren, bspw. ins Verzeichnis „C:\Program Files\jython2.5.1“
3. Jetzt gibt es zwei Alternativen:
    a. Entweder verschieben/kopieren Sie dann das Jython-Lib-Verzeichnis „C:\Program Files\jython2.5.1\Lib“ in das Unterverzeichnis "lib" des Hamster-Simulator-Ordners (so dass es in "lib" ein Unterverzeichnis "Lib" gibt)
    b. Oder Sie schreiben die folgende Zeile in die Datei "hamstersimulator.bat" (im Hamster-Simulator-Ordner):
    `java -Dpython.home="C:\Program Files\jython2.5.1" -jar hamstersimulator.jar`
    und starten den Hamster-Simulator dann durch Doppelklick auf diese Datei "hamstersimulator.bat"

9.2.2 Eigene Module definieren

Sie können auch eigene Module definieren und nutzen. Wenn Sie bspw. einen vordefinierten Hamster-Befehl „rechtsUm“ zur Verfügung stellen wollen, gehen Sie folgendermaßen vor:

1. Legen Sie im Jython-Lib-Verzeichnis bspw. eine Datei mit dem Namen hamstererweiterung.py“ an. Füllen Sie die Datei mit folgendem Inhalt:

```
from de.hamster.python.model import PythonHamster

vor = PythonHamster.getStandardHamsterIntern().vor;
linksUm = PythonHamster.getStandardHamsterIntern().linksUm;
gib = PythonHamster.getStandardHamsterIntern().gib;
nimm = PythonHamster.getStandardHamsterIntern().nimm;
vornFrei = PythonHamster.getStandardHamsterIntern().vornFrei;
kornDa = PythonHamster.getStandardHamsterIntern().kornDa;
maulLeer = PythonHamster.getStandardHamsterIntern().maulLeer;

def rechtsUm():
     linksUm()
     linksUm()
     linksUm()
```

2. Starten Sie dann den Hamster-Simulator neu.
3. Im Hamster-Simulator können Sie nun bspw. folgendes Programm eingeben und ausführen:

```
from hamstererweiterung import rechtsUm

for i in range(5):
     vor()
     rechtsUm()
```

4. Wenn Sie die definierten Befehle nicht einzeln importieren möchten, müssen Sie den Modulnamen vor die Befehle platzieren, bspw.

```
import hamstererweiterung

for i in range(5):
     vor()
     hamstererweiterung.rechtsUm()
```

5. Wenn Sie die Datei "hamstererweiterung.py" um neue Befehle ergänzen, scheint es manchmal notwendig zu sein, den Hamster-Simulator neu zu starten, damit dieser auch die neuen Befehle ausführen kann.

Eine weitere Möglichkeit eigene Module zu schreiben und zu nutzen, ist folgende:

Gegeben folgende Datei „my.py“:

```
from de.hamster.python.model import PythonHamster


vor = PythonHamster.getStandardHamsterIntern().vor;
linksUm = PythonHamster.getStandardHamsterIntern().linksUm;
gib = PythonHamster.getStandardHamsterIntern().gib;
nimm = PythonHamster.getStandardHamsterIntern().nimm;
vornFrei = PythonHamster.getStandardHamsterIntern().vornFrei;
kornDa = PythonHamster.getStandardHamsterIntern().kornDa;
maulLeer = PythonHamster.getStandardHamsterIntern().maulLeer;


def rechtsUm():
     linksUm()
     linksUm()
     linksUm()


def zurWand():
     while vornFrei():
          vor()
```

Sei "Programme" das Verzeichnis, in dem Sie Ihre Hamster-Python-Programme speichern. Können Sie auch Python-Module dort abspeichern, bspw. die Datei "my.py". Anschließend können Sie in Ihren Python-Hamster-Programmen folgendermaßen diese Module nutzen:

```
import sys
sys.path.append("Programme")
from my import rechtsUm

if vornFrei():
    vor()
    rechtsUm()
```

9.3 Python-Beispielprogramme

Die folgenden Unterabschnitte demonstrieren an Beispielen das Schreiben von Python-Hamster-Programmen.

9.3.1 Territorium leeren

Das folgende Python-Programm ist ein einfaches imperatives Beispielprogramm, bei dem der Standard-Hamster ein beliebiges Territorium ohne innere Mauern leert:

```
# Aufgabe:
# der Hamster steht irgendwo in einem beliebigen Territorium
# ohne innere Mauern; er soll alle Körner fressen

def kehrt():
    linksUm()
    linksUm()

def rechtsUm():
    kehrt()
    linksUm()

def laufeZurueck():
    while vornFrei():
        vor()

def sammle():
    while kornDa():
      nimm()

def laufeZurWand():
    while vornFrei():
        vor()

def laufeInEcke():
    laufeZurWand()
    linksUm()
    laufeZurWand()
    linksUm()
```

```
def ernteEineReihe():
    sammle()
    while vornFrei():
        vor()
        sammle()

def ernteEineReiheUndLaufeZurueck():
    ernteEineReihe()
    kehrt()
    laufeZurueck()

laufeInEcke()
ernteEineReiheUndLaufeZurueck()
rechtsUm()
while vornFrei():
    vor()
    rechtsUm()
    ernteEineReiheUndLaufeZurueck()
    rechtsUm()
```

9.3.2 Territorium leeren 2

Auch das folgende Python-Programm ist ein einfaches imperatives Beispielprogramm, bei dem der Standard-Hamster ein beliebiges Territorium ohne innere Mauern leert:

```
# Aufgabe:
# der Hamster steht irgendwo in einem beliebigen Territorium
# ohne innere Mauern; er soll alle Körner fressen

# drehe dich um 180 Grad
def kehrt():
    linksUm()
    linksUm()

# drehe dich um 90 Grad nach rechts
def rechtsUm():
    kehrt()
    linksUm()

# der Hamster sammelt alle Koerner eines Feldes ein
def sammle():
    while kornDa():
        nimm()

# Ueberpruefung, ob sich links vom Hamster
# eine Mauer befindet
def linksFrei():
    linksUm()
    if vornFrei():
        rechtsUm();
        return True
    else:
        rechtsUm()
        return False
```

```
# Ueberpruefung, ob sich rechts vom Hamster eine
# Mauer befindet
def rechtsFrei():
    rechtsUm()
    if vornFrei():
        linksUm()
        return True
    else:
        linksUm()
        return False

# der Hamster laeuft bis zur naechsten Wand
def laufeZurWand():
    while vornFrei():
        vor()

# der Hamster laeuft in eine Ecke
def laufeInEcke():
    laufeZurWand()
    linksUm()
    laufeZurWand()
    linksUm()

# der Hamster soll sich in die naechste Reihe in noerdlicher
# Richtung begeben; vom Hamster aus gesehen, liegt diese Reihe
# links von ihm
def begibDichLinksUmInNaechsteReihe():
    linksUm()
    vor()
    linksUm()

# der Hamster soll sich in die naechste Reihe in noerdlicher
# Richtung begeben; vom Hamster aus gesehen, liegt diese Reihe
# rechts von ihm
def begibDichRechtsUmInNaechsteReihe():
    rechtsUm()
    vor()
    rechtsUm()


# Ueberpruefung, ob in noerdlicher Richtung (vom Hamster aus
# gesehen links) eine weitere nicht mit Mauern besetzte
# Reihe existiert
def weitereReiheLinksVomHamsterExistiert():
    return linksFrei()

# Ueberpruefung, ob in noerdlicher Richtung (vom Hamster aus
# gesehen rechts) eine weitere nicht mit Mauern besetzte
# Reihe existiert
def weitereReiheRechtsVomHamsterExistiert():
    return rechtsFrei()

# der Hamster soll alle Koerner in einer Reihe einsammeln
def ernteEineReihe():
    sammle()
    while vornFrei():
        vor()
```

```
        sammle()

# der Hamster soll alle Koerner in einer Reihe einsammeln;
# er laeuft dabei von Westen nach Osten
def ernteEineReiheNachOsten():
    ernteEineReihe()

# der Hamster soll alle Koerner in einer Reihe einsammeln;
# er laeuft dabei von Osten nach Westen
def ernteEineReiheNachWesten():
    ernteEineReihe()



# der Hamster soll einzelne Koernerreihen abgrasen,
# so lange noch weitere Reihen existieren;
# er unterscheidet dabei, ob er die Reihen von
# Osten oder von Westen aus abgrast
laufeInEcke()
ernteEineReiheNachOsten()
while weitereReiheLinksVomHamsterExistiert():
    begibDichLinksUmInNaechsteReihe()
    ernteEineReiheNachWesten()
    if weitereReiheRechtsVomHamsterExistiert():
        begibDichRechtsUmInNaechsteReihe()
        ernteEineReiheNachOsten()
    else:
        kehrt()
```

9.3.3 Berg erklimmen

Im folgenden imperativen Python-Programm erklimmt der Standard-Hamster einen Berg:

```
# Aufgabe:
# der Hamster soll den Gipfel eines vor ihm stehenden
# Berges erklimmen

# der Hamster soll zum Berg laufen
def laufeZumBerg():
    while vornFrei():
        vor()

# der Hamster soll den Berg erklimmen
def erklimmeDenBerg():
    while not gipfelErreicht():
        erklimmeEineStufe()

# der Hamster soll eine Stufe erklimmen
def erklimmeEineStufe():
    linksUm()  # nun schaut der Hamster nach oben
    vor()      # der Hamster erklimmt die Mauer
    rechtsUm() # der Hamster wendet sich wieder dem Berg zu
    vor()      # der Hamster springt auf den Vorsprung

# der Hamster dreht sich nach rechts um
```

```
def rechtsUm():
    linksUm()
    linksUm()
    linksUm()

# hat der Hamster den Gipfel erreicht?
def gipfelErreicht():
    return vornFrei()

# der Hamster soll zunaechst bis zum Berg laufen
# und dann den Berg erklimmen
laufeZumBerg()
erklimmeDenBerg()
```

9.3.4 Wettlauf

Im folgenden objektorientierten Python-Programm liefern sich 3 Hamster einen Wettlauf bis zur nächsten Mauer:

```
paul = Hamster.getStandardHamster()
willi = Hamster(paul)
maria = Hamster(paul.getReihe()+1, paul.getSpalte(), paul.getBlickrichtung(), 0)
while paul.vornFrei() and willi.vornFrei() and maria.vornFrei():
    paul.vor()
    willi.vor()
    maria.vor()
paul.schreib("Fertig!")
```

9.3.5 Objektorientiertes Territorium leeren

Im folgenden Python-Programm werden objektorientierte Konzepte genutzt, um einen Hamster das Territorium leeren zu lassen:

```
# Ein neu erzeugter Hamster grast das Territorium ab und
# zaehlt dabei die gesammelten Koerner

class AbgrasHamster(Hamster):

    def __init__(self, reihe, spalte, blickrichtung, anzahlKoerner):
        Hamster.__init__(self, reihe, spalte, blickrichtung, anzahlKoerner)
        self.__Gesammelt = 0

    def kehrt(self):
        self.linksUm()
        self.linksUm()

    def rechtsUm(self):
        self.kehrt()
        self.linksUm()

    def laufeZurueck(self):
        while self.vornFrei():
            self.vor()
```

```
    def sammle(self):
        while self.kornDa():
          self.nimm()
          self.__Gesammelt += 1

    def laufeZurWand(self):
        while self.vornFrei():
            self.vor()

    def laufeInEcke(self):
        self.laufeZurWand()
        self.linksUm()
        self.laufeZurWand()
        self.linksUm()

    def ernteEineReihe(self):
        self.sammle()
        while self.vornFrei():
            self.vor()
            self.sammle()

    def ernteEineReiheUndLaufeZurueck(self):
        self.ernteEineReihe()
        self.kehrt()
        self.laufeZurueck()

    def gesammelteKoerner(self):
        return self.__Gesammelt

# Hauptprogramm
paul = AbgrasHamster(2, 3, Hamster.WEST, 0)
paul.laufeInEcke()
paul.ernteEineReiheUndLaufeZurueck()
paul.rechtsUm()
while paul.vornFrei():
    paul.vor()
    paul.rechtsUm()
    paul.ernteEineReiheUndLaufeZurueck()
    paul.rechtsUm()
paul.schreib("Gesammelte Koerner = " + str(paul.gesammelteKoerner()))
```

9.4 Python-Konsole

Mit Hilfe der Python-Konsole ist es möglich, interaktiv Python- und Python-Hamster-Befehle auszuführen. Die Python-Konsole können Sie über das Menü „Fenster“ im Editor-Fenster öffnen. Er erscheint ein neues Fenster auf dem Bildschirm. Die Hamster-Python-Konsole funktioniert dabei wie eine normale Python-Konsole. Zusätzlich zu normalen Python-Befehlen können Sie jedoch auch Hamster-Befehle eingeben.

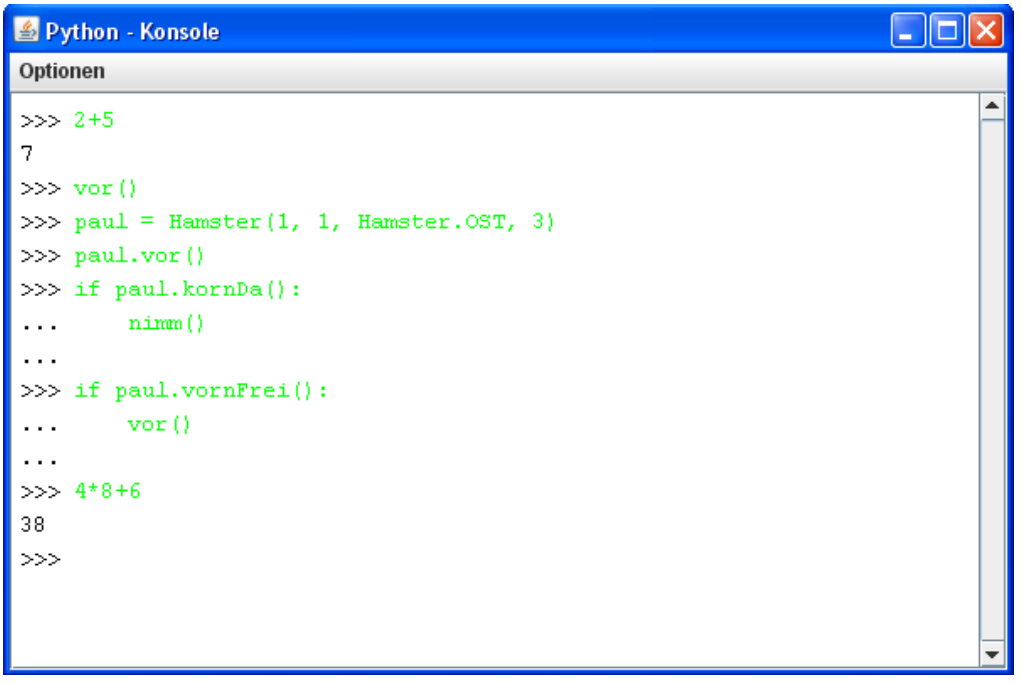

Abbildung 9.1: Python-Konsole

Hinweis: Die Python-Konsole und Python-Programme arbeiten mit unterschiedlichen Python-Interpretern, d.h. nachdem Sie ein Python-Programm ausgeführt haben, sind unter Umständen durch das Programm angelegte Variablen nicht in der Python-Konsole (und auch nicht in anderen Python-Programmen) verfügbar. Weiterhin ist anzumerken, dass nach der Ausführung eines Python-Programms unter Umständen vorher in der Konsole erzeugte Hamster anschließend nicht mehr zugreifbar sind.

9.5 Implementierung

Die Anbindung von Python an den Hamster-Simulator wurde mit Hilfe von *Jython* realisiert (http://www.jython.org/). Hier die Lizenzbedingungen:

PYTHON SOFTWARE FOUNDATION LICENSE VERSION 2

1. This LICENSE AGREEMENT is between the Python Software Foundation ("PSF"), and the Individual or Organization ("Licensee") accessing and otherwise using this software ("Jython") in source or binary form and its associated documentation.

2. Subject to the terms and conditions of this License Agreement, PSF hereby grants Licensee a nonexclusive, royalty-free, world-wide license to reproduce, analyze, test, perform and/or display publicly, prepare derivative works, distribute, and otherwise use Jython alone or in any derivative version, provided, however, that PSF's License Agreement and PSF's notice of copyright, i.e., "Copyright (c) 2007 Python Software Foundation; All Rights Reserved" are retained in Jython alone or in any derivative version prepared by Licensee.

3. In the event Licensee prepares a derivative work that is based on or incorporates Jython or any part thereof, and wants to make the derivative work available to others as provided herein, then Licensee hereby agrees to include in any such work a brief summary of the changes made to Jython.

4. PSF is making Jython available to Licensee on an "AS IS" basis. PSF MAKES NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED. BY WAY OF EXAMPLE, BUT NOT LIMITATION, PSF MAKES NO AND DISCLAIMS ANY REPRESENTATION OR WARRANTY OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE OR THAT THE USE OF JYTHON WILL NOT INFRINGE ANY THIRD PARTY RIGHTS.

5. PSF SHALL NOT BE LIABLE TO LICENSEE OR ANY OTHER USERS OF JYTHON FOR ANY INCIDENTAL, SPECIAL, OR CONSEQUENTIAL DAMAGES OR LOSS AS A RESULT OF MODIFYING, DISTRIBUTING, OR OTHERWISE USING JYTHON, OR ANY DERIVATIVE THEREOF, EVEN IF ADVISED OF THE POSSIBILITY THEREOF.

6. This License Agreement will automatically terminate upon a material breach of its terms and conditions.

7. Nothing in this License Agreement shall be deemed to create any relationship of agency, partnership, or joint venture between PSF and Licensee. This License Agreement does not grant permission to use PSF trademarks or trade name in a trademark sense to endorse or promote products or services of Licensee, or any third party.

8. By copying, installing or otherwise using Jython, Licensee agrees to be bound by the terms and conditions of this License Agreement.

Jython 2.0, 2.1 License

Copyright (c) 2000, 2001, 2002, 2003, 2004, 2005, 2006, 2007 Jython Developers All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- Neither the name of the Jython Developers nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE REGENTS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT

OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

10 Ruby

Ruby (engl. für Rubin) ist eine moderne, vielseitige höhere Programmiersprache, die Mitte der Neunziger Jahre von Yukihiro Matsumoto entworfen wurde. Ruby ist interpretiert und objektorientiert, unterstützt aber mehrere weitere Programmierparadigmen (unter anderem Prozedurale und Funktionale Programmierung sowie Nebenläufigkeit), bietet dynamische Typisierung, Reflexion und Automatische Speicherbereinigung.

Aus Unzufriedenheit über verfügbare Skriptsprachen begann Yukihiro „Matz“ Matsumoto 1993, an einer eigenen Sprache zu arbeiten, und gab am 21. Dezember 1995 die erste Version von Ruby, 0.95, frei. Den Namen, hergeleitet vom Edelstein Rubin, wählte er als Anspielung auf die Programmiersprache Perl. Zunächst wurde Ruby mangels englischsprachiger Dokumentation fast ausschließlich in Japan benutzt, wo es einen ähnlichen Stellenwert erlangte wie Perl und Python in Europa und Amerika. Um das Jahr 2000 wurden Aktivitäten gestartet, um Ruby auch außerhalb Japans bekannt zu machen, woraufhin mit der Zeit auch englische Dokumentationen entstanden. Inzwischen gibt es auch dutzende deutschsprachige Bücher zu Ruby im Allgemeinen und speziellen Themen. Heute wird die Sprache als Open-Source-Projekt weitergepflegt (aus Wikipedia).

10.1 Die Programmiersprache Ruby

Im Internet finden Sie viele Informationen rund um die Programmiersprache Ruby. Schauen Sie einfach mal unter folgenden Links nach:

- http://www.ruby-lang.org/de/ (offizielle Webseite)
- http://de.wikipedia.org/wiki/Ruby_(Programmiersprache) (Wikipedia)

10.2 Ruby-Hamster-Programme

Um mit dem Hamster-Simulator Ruby-Programme entwickeln und ausführen zu können, muss die Property `ruby` auf `true` gesetzt sein. Standardmäßig ist das nicht der Fall. Öffnen Sie also die Datei `hamster.properties` und ersetzen Sie dort die Zeile `ruby=false` durch die Zeile `ruby=true`.

Starten Sie dann den Hamster-Simulator. Wenn Sie ein neues Programm erstellen wollen, wählen Sie in der erscheinenden Dialogbox den Eintrag „Ruby-Programm“. Im Editor-Bereich erscheint dann ein Feld mit dem Inhalt „`if vornFrei vor end`“. Ersetzen Sie diesen Inhalt durch Ihr Ruby-Programm und speichern Sie die Datei dann ab. Ruby-Programme brauchen nicht compiliert zu werden. Nach dem Abspeichern haben Sie direkt die Möglichkeit, das Programm auszuführen. Syntaxfehler werden erst zur Laufzeit angezeigt.

Im Hamster-Simulator lassen sich sowohl imperative als auch objektorientierte Ruby-Programme entwickeln. Auch gemischte Programme sind möglich. Der (imperative) Hamster kennt (wie üblich) die Befehle

```
vor()
linksUm()
gib()
nimm()
vornFrei() (-> bool)
kornDa() (-> bool)
maulLeer() (-> bool)
```

Eine vordefinierte Klasse namens `Hamster` kann für objektorientierte Ruby-Programme genutzt werden. Die Klasse stellt folgende Methoden zur Verfügung:

```
vor()
linksUm()
gib()
nimm()
vornFrei() (-> bool)
kornDa() (-> bool)
maulLeer() (-> bool)
schreib(string)
liesZeichenkette(string) (-> string)
liesZahl(string) (-> int)
getReihe() (-> int)
getSpalte() (-> int)
getBlickrichtung() (-> int: Hamster.NORD, Hamster.SUED, Hamster.OST,
Hamster.WEST)
getAnzahlKoerner() (-> int)
getStandardHamster() (-> Hamster)
```

Die Klasse definiert 2 Konstruktoren:

```
Hamster(reihe, spalte, blickrichtung, anzahlKoerner)
Hamster(hamster)
```

10.3 Ruby-Beispielprogramme

Die folgenden Unterabschnitte demonstrieren an Beispielen das Schreiben von Ruby-Hamster-Programmen.

10.3.1 Territorium leeren

Das folgende Ruby-Programm ist ein einfaches imperatives Beispielprogramm, bei dem der Standard-Hamster ein beliebiges Territorium ohne innere Mauern leert:

```
# Aufgabe:
# der Hamster steht irgendwo in einem beliebigen Territorium
# ohne innere Mauern; er soll alle Körner fressen

def kehrt
```

```
    linksUm
    linksUm
end

def rechtsUm
    kehrt
    linksUm
end

def laufeZurueck
    while vornFrei
        vor
    end
end

def sammle
    while kornDa
        nimm
    end
end

def laufeZurWand
    while vornFrei
        vor
    end
end

def laufeInEcke
    laufeZurWand
    linksUm
    laufeZurWand
    linksUm
end

def ernteEineReihe
    sammle
    while vornFrei
        vor
        sammle
    end
end

def ernteEineReiheUndLaufeZurueck
    ernteEineReihe
    kehrt
    laufeZurueck
end

# Hauptprogramm
laufeInEcke
ernteEineReiheUndLaufeZurueck
rechtsUm
while vornFrei
    vor
    rechtsUm
    ernteEineReiheUndLaufeZurueck
    rechtsUm
end
```

10.3.2 Territorium leeren 2

Auch das folgende Ruby-Programm ist ein einfaches imperatives Beispielprogramm, bei dem der Standard-Hamster ein beliebiges Territorium ohne innere Mauern leert:

```
# Aufgabe:
# der Hamster steht irgendwo in einem beliebigen Territorium
# ohne innere Mauern; er soll alle Körner fressen

# drehe dich um 180 Grad
def kehrt
    linksUm
    linksUm
end

# drehe dich um 90 Grad nach rechts
def rechtsUm
    kehrt
    linksUm
end

# der Hamster sammelt alle Koerner eines Feldes ein
def sammle
    while kornDa
        nimm
    end
end

# Ueberpruefung, ob sich links vom Hamster
# eine Mauer befindet
def linksFrei
    linksUm
    if vornFrei
        rechtsUm
        return true
    else
        rechtsUm
        return false
    end
end

# Ueberpruefung, ob sich rechts vom Hamster eine
# Mauer befindet
def rechtsFrei
    rechtsUm
    if vornFrei
        linksUm
        return true
    else
        linksUm
        return false
    end
end

# der Hamster laeuft bis zur naechsten Wand
def laufeZurWand
```

```
    while vornFrei
        vor
    end
end

# der Hamster laeuft in eine Ecke
def laufeInEcke
    laufeZurWand
    linksUm
    laufeZurWand
    linksUm
end

# der Hamster soll sich in die naechste Reihe in noerdlicher
# Richtung begeben; vom Hamster aus gesehen, liegt diese Reihe
# links von ihm
def begibDichLinksUmInNaechsteReihe
    linksUm
    vor
    linksUm
end

# der Hamster soll sich in die naechste Reihe in noerdlicher
# Richtung begeben; vom Hamster aus gesehen, liegt diese Reihe
# rechts von ihm
def begibDichRechtsUmInNaechsteReihe
    rechtsUm
    vor
    rechtsUm
end


# Ueberpruefung, ob in noerdlicher Richtung (vom Hamster aus
# gesehen links) eine weitere nicht mit Mauern besetzte
# Reihe existiert
def weitereReiheLinksVomHamsterExistiert
    return linksFrei
end

# Ueberpruefung, ob in noerdlicher Richtung (vom Hamster aus
# gesehen rechts) eine weitere nicht mit Mauern besetzte
# Reihe existiert
def weitereReiheRechtsVomHamsterExistiert
    return rechtsFrei
end

# der Hamster soll alle Koerner in einer Reihe einsammeln
def ernteEineReihe
    sammle
    while vornFrei
        vor
        sammle
    end
end

# der Hamster soll alle Koerner in einer Reihe einsammeln;
# er laeuft dabei von Westen nach Osten
def ernteEineReiheNachOsten
```

```
    ernteEineReihe
end

# der Hamster soll alle Koerner in einer Reihe einsammeln;
# er laeuft dabei von Osten nach Westen
def ernteEineReiheNachWesten
    ernteEineReihe
end

# der Hamster soll einzelne Koernerreihen abgrasen,
# so lange noch weitere Reihen existieren;
# er unterscheidet dabei, ob er die Reihen von
# Osten oder von Westen aus abgrast
laufeInEcke
ernteEineReiheNachOsten
while weitereReiheLinksVomHamsterExistiert
    begibDichLinksUmInNaechsteReihe
    ernteEineReiheNachWesten
    if weitereReiheRechtsVomHamsterExistiert
        begibDichRechtsUmInNaechsteReihe
        ernteEineReiheNachOsten
    else
        kehrt
    end
end
```

10.3.3 Berg erklimmen

Im folgenden imperativen Ruby-Programm erklimmt der Standard-Hamster einen Berg:

```
# Aufgabe:
# der Hamster soll den Gipfel eines vor ihm stehenden
# Berges erklimmen

# der Hamster soll zum Berg laufen
def laufeZumBerg
    while vornFrei
        vor
    end
end

# der Hamster soll den Berg erklimmen
def erklimmeDenBerg
    while not gipfelErreicht
        erklimmeEineStufe
    end
end

# der Hamster soll eine Stufe erklimmen
def erklimmeEineStufe
    linksUm  # nun schaut der Hamster nach oben
    vor      # der Hamster erklimmt die Mauer
    rechtsUm # der Hamster wendet sich wieder dem Berg zu
    vor      # der Hamster springt auf den Vorsprung
end
```

```
# der Hamster dreht sich nach rechts um
def rechtsUm
    linksUm
    linksUm
    linksUm
end

# hat der Hamster den Gipfel erreicht?
def gipfelErreicht
    return vornFrei
end

# der Hamster soll zunaechst bis zum Berg laufen
# und dann den Berg erklimmen
laufeZumBerg
erklimmeDenBerg
```

10.3.4 Wettlauf

Im folgenden objektorientierten Ruby-Programm liefern sich 3 Hamster einen Wettlauf bis zur nächsten Mauer:

```
paul = Hamster.getStandardHamster
willi = Hamster.new(paul)
maria = Hamster.new(paul.getReihe + 1, paul.getSpalte, paul.getBlickrichtung, 0)
while paul.vornFrei and willi.vornFrei and maria.vornFrei
    paul.vor
    willi.vor
    maria.vor
end
paul.schreib("Fertig!")
```

10.3.5 Objektorientiertes Territorium leeren

Im folgenden Ruby-Programm werden objektorientierte Konzepte genutzt, um einen Hamster das Territorium leeren zu lassen:

```
# Ein neu erzeugter Hamster grast das Territorium ab und
# zaehlt dabei die gesammelten Koerner

class AbgrasHamster < Hamster

    def initialize(reihe, spalte, blickrichtung, anzahlKoerner)
        super(reihe, spalte, blickrichtung, anzahlKoerner)
        @gesammelt = 0
    end

    def kehrt
        linksUm
        linksUm
    end
```

```
    def rechtsUm
        kehrt
        linksUm
    end

    def laufeZurueck
        while vornFrei
            vor
        end
    end

    def sammle
        while kornDa
            nimm
            @gesammelt += 1
        end
    end

    def laufeZurWand
        while vornFrei
            vor
        end
    end

    def laufeInEcke
        laufeZurWand
        linksUm
        laufeZurWand
        linksUm
    end

    def ernteEineReihe
        sammle
        while vornFrei
            vor
            sammle
        end
    end

    def ernteEineReiheUndLaufeZurueck
        ernteEineReihe
        kehrt
        laufeZurueck
    end

    def gesammelteKoerner
        return @gesammelt
    end
end

# Hauptprogramm
paul = AbgrasHamster.new(2, 3, Hamster.WEST, 0)
paul.laufeInEcke
paul.ernteEineReiheUndLaufeZurueck
paul.rechtsUm
while paul.vornFrei
    paul.vor
    paul.rechtsUm
```

```
    paul.ernteEineReiheUndLaufeZurueck
    paul.rechtsUm
end
paul.schreib("Gesammelte Koerner = " + paul.gesammelteKoerner.to_s)
```

10.4 Ruby-Konsole

Mit Hilfe der Ruby-Konsole ist es möglich, interaktiv Ruby- und Ruby-Hamster-Befehle auszuführen. Die Ruby-Konsole können Sie über das Menü „Fenster“ im Editor-Fenster öffnen. Er erscheint ein neues Fenster auf dem Bildschirm. Die Hamster-Ruby-Konsole funktioniert dabei wie eine normale Python-Konsole. Zusätzlich zu normalen Ruby-Befehlen können Sie jedoch auch Hamster-Befehle eingeben.

Abbildung 10.1: Ruby-Konsole

Hinweis: Die Ruby-Konsole und Ruby-Programme arbeiten mit unterschiedlichen Ruby-Interpretern, d.h. nachdem Sie ein Ruby-Programm ausgeführt haben, sind unter Umständen durch das Programm angelegte Variablen nicht in der Ruby-Konsole (und auch nicht in anderen Ruby-Programmen) verfügbar. Weiterhin ist anzumerken, dass nach der Ausführung eines Ruby-Programms unter Umständen vorher in der Konsole erzeugte Hamster anschließend nicht mehr zugreifbar sind.

10.5 Implementierung

Die Anbindung von Ruby an den Hamster-Simulator wurde mit Hilfe von *JRuby* realisiert (http://jruby.org/). Lizenz: CPL, GPL und LGPL

```
JRuby distributes some ruby modules which are distributed under Ruby
license:

Ruby is copyrighted free software by Yukihiro Matsumoto <matz@netlab.jp>.
You can redistribute it and/or modify it under either the terms of the GPL
(see the file GPL), or the conditions below:

  1. You may make and give away verbatim copies of the source form of the
     software without restriction, provided that you duplicate all of the
     original copyright notices and associated disclaimers.

  2. You may modify your copy of the software in any way, provided that
     you do at least ONE of the following:

       a) place your modifications in the Public Domain or otherwise
          make them Freely Available, such as by posting said
        modifications to Usenet or an equivalent medium, or by allowing
        the author to include your modifications in the software.

       b) use the modified software only within your corporation or
          organization.

       c) give non-standard binaries non-standard names, with
          instructions on where to get the original software distribution.

       d) make other distribution arrangements with the author.

  3. You may distribute the software in object code or binary form,
     provided that you do at least ONE of the following:

       a) distribute the binaries and library files of the software,
        together with instructions (in the manual page or equivalent)
        on where to get the original distribution.

       b) accompany the distribution with the machine-readable source of
        the software.

       c) give non-standard binaries non-standard names, with
          instructions on where to get the original software distribution.

       d) make other distribution arrangements with the author.

  4. You may modify and include the part of the software into any other
     software (possibly commercial).  But some files in the distribution
     are not written by the author, so that they are not under these terms.

     For the list of those files and their copying conditions, see the
     file LEGAL.

  5. The scripts and library files supplied as input to or produced as
     output from the software do not automatically fall under the
     copyright of the software, but belong to whomever generated them,
     and may be sold commercially, and may be aggregated with this
     software.

  6. THIS SOFTWARE IS PROVIDED "AS IS" AND WITHOUT ANY EXPRESS OR
     IMPLIED WARRANTIES, INCLUDING, WITHOUT LIMITATION, THE IMPLIED
     WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
     PURPOSE.
```

11 Hamstern mit Scratch

Scratch ist eine Programmierumgebung bzw. Programmiersprache für echte Programmieranfänger. Anders als bei anderen Programmiersprachen müssen hier die Programmierer keinen textuellen Sourcecode schreiben. Vielmehr setzen sich Scratch-Programme aus graphischen Bausteinen zusammen (siehe Abbildung 11.1). Die Programmierumgebung unterstützt dabei das Erstellen von Programmen durch einfache Drag-and-Drop-Aktionen mit der Maus. Mehr Informationen zu Scratch findet man im Internet unter http://scratch.mit.edu/

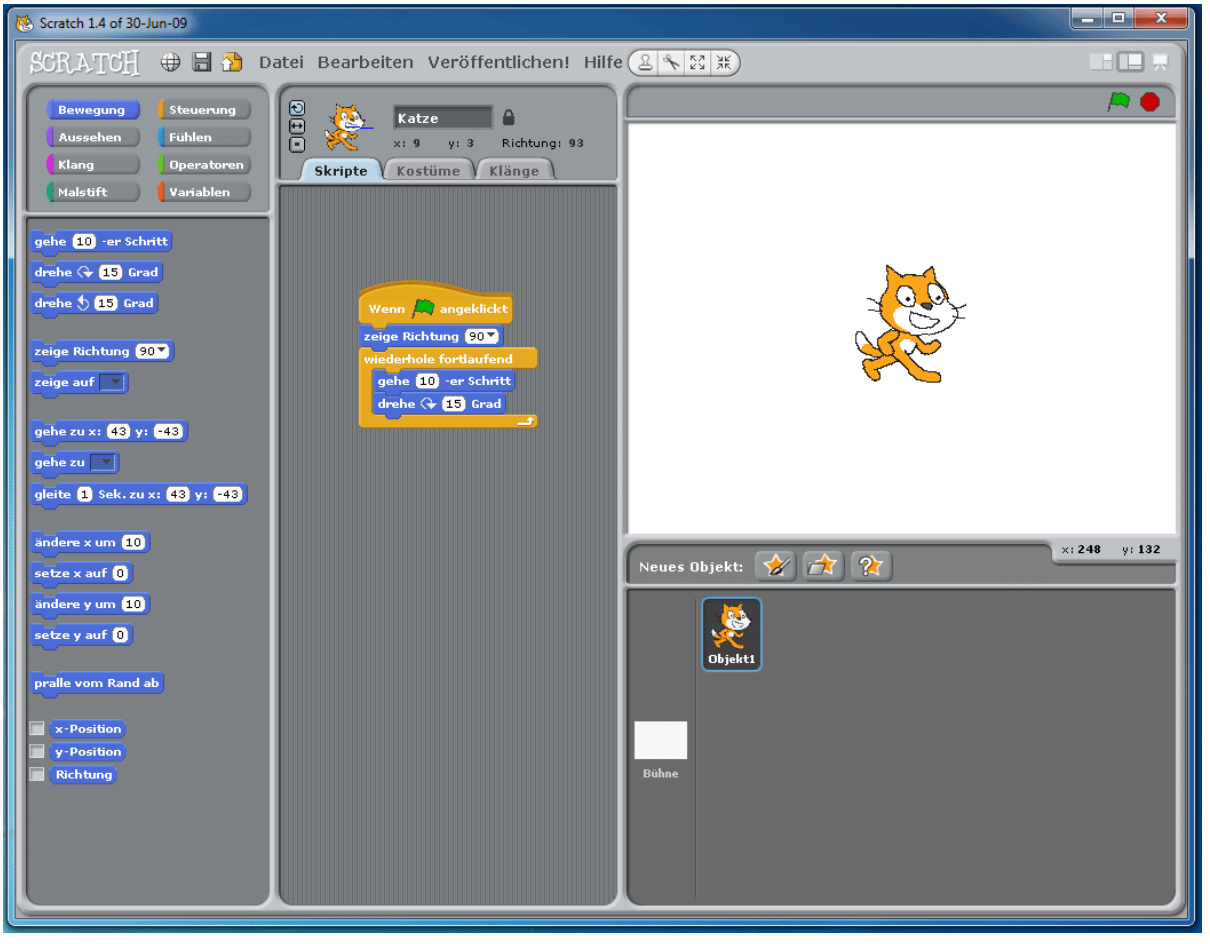

Abbildung 11.1: Scratch

Ein Vorteil von Scratch für Programmieranfänger ist, dass man keine syntaktischen Fehler machen kann und dass sich sehr schnell kleine Programme „zusammenklicken“ lassen. Aus diesem Grund haben wir einige Ideen und Konzepte von Scratch in den Hamster-Simulator integriert. Das „Look-and-Feel“ ist dabei bis auf wenige Ausnahmen identisch mit dem „Look-and-Feel“ des Original-Scratch (siehe Abbildung 11.2). Herzlichen Dank an Jürgen Boger, der die Scratch-Integration im Rahmen seiner Bachelor-Arbeit implementiert hat.

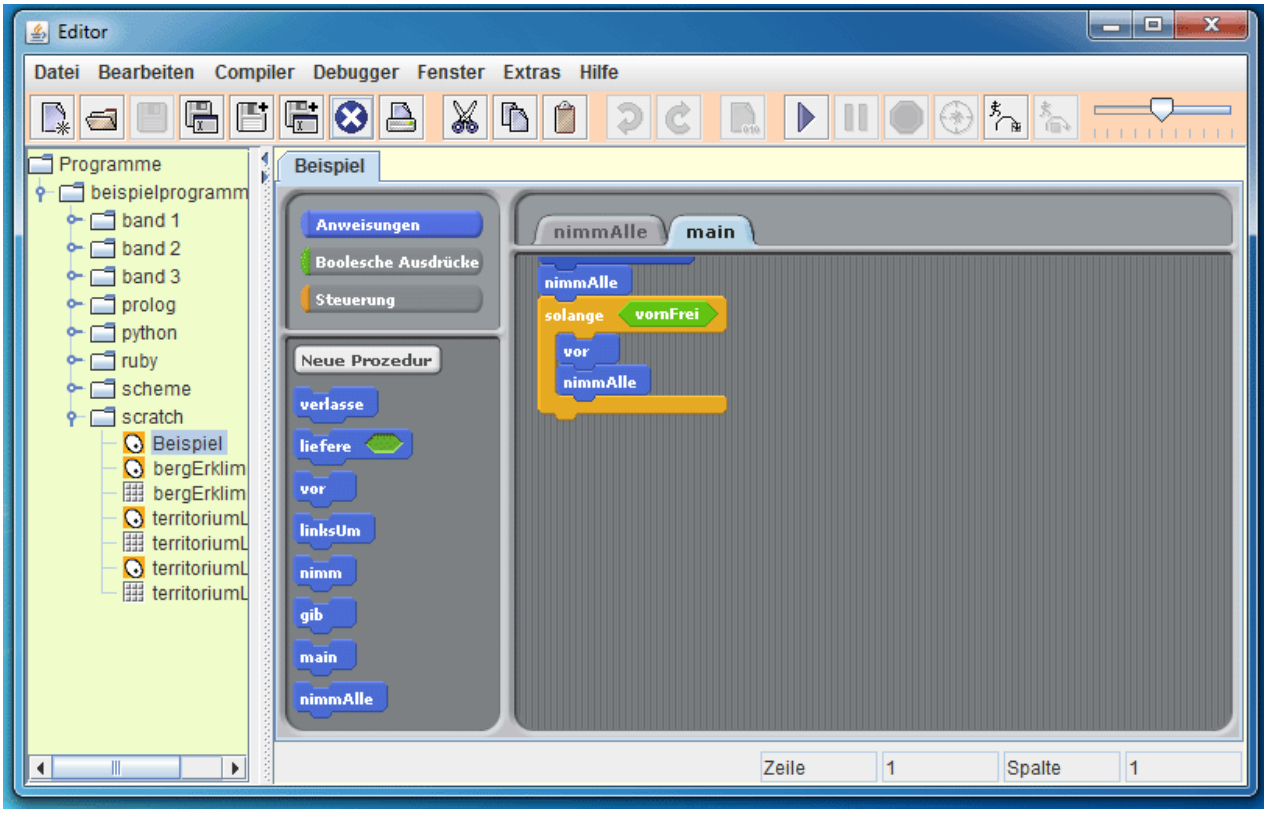

Abbildung 11.2: Scratch-Hamster-Simulator

11.1 Überblick

Das Scratch-Hamster-Modell unterstützt folgende Komponenten bzw. Konzepte:

- Die vier Grundbefehle des Hamsters (vor, linksUm, gib und nimm) sind als Anweisungsblöcke vordefiniert.
- Es lassen sich neue Befehle als Prozeduren definieren, die als neue Anweisungsblöcke repräsentiert werden.
- Die drei Testbefehle des Hamsters (vornFrei, kornDa und maulLeer) sind als Boolesche-Ausdruck-Blöcke vordefiniert. Weiterhin gibt es die Boolesche-Ausdruck-Blöcke „true“ und „false“.
- Mittels entsprechender Boolesche-Ausdruck-Blöcke für die booleschen Operatoren „nicht“, „und“ und „oder“ können komplexe boolesche Ausdrücke erstellt werden. Als Operanden sind dabei beliebige andere Boolesche-Ausdruck-Blöcke einsetzbar.
- Es lassen sich neue Testbefehle als boolesche Funktionen definieren, die als neue Boolesche-Ausdruck-Blöcke repräsentiert werden.
- Als Kontrollstrukturen zur Programmsteuerung sind die if-Anweisung, die if-else-Anweisung, die while-Schleife und die do-while-Schleife als spezielle Anweisungsblöcke („falls“, „falls-sonst“, „solange“, „wiederhole-solange“) integriert.

Diesen können Boolesche-Ausdruck-Blöcke als Bedingungen und Anweisungsblöcke als innere Anweisungen zugeordnet werden.

Variablen werden durch das Scratch-Hamster-Modell nicht unterstützt, um es nicht unnötig zu verkomplizieren. Intention des Scratch-Hamster-Modells ist es, Programmieranfängern einen ersten Eindruck von der Programmierung zu geben. Das Scratch-Hamster-Modell deckt dazu die ersten 12 Kapitel des Java-Hamster-Buches (erster Band) ab. Danach sollte unserer Meinung nach ein Programmieranfänger so weit sein, dass er ohne große Probleme auf die textuelle Programmierung umschwenken kann.

11.2 Voraussetzungen

Um mit dem Hamster-Simulator Scratch-Hamster-Programme entwickeln und ausführen zu können, muss die Property `scratch` auf `true` gesetzt sein. Standardmäßig ist das der Fall. Um den Scratch-Modus auszuschalten, öffnen Sie bitte die Datei `hamster.properties` und ersetzen Sie dort die Zeile `scratch=true` durch die Zeile `scratch=false`. Mit Hilfe von Properties lassen sich seit Version 2.8.2 des Hamster-Simulators auch die Beschriftungen der Standard-Blöcke verändern (sie Kapitel 5.1.16).

11.3 Ein erstes kleines Beispiel

Um einen ersten kleinen Eindruck vom Scratch-Hamster-Modell zu bekommen, öffnen Sie bitte im Ordner „beispielprogramme“ den Unterordner „scratch“ und laden Sie dann das Programm „Beispiel“. Es erscheint das in Abbildung 11.2 dargestellte Scratch-Editorfenster. Führen Sie das Programm durch Anklicken des Run-Buttons in der Toolbar aus: Der Hamster läuft bis zur nächsten Wand und sammelt dabei alle Körner ein.

11.4 Erstellen von Scratch-Programmen

Um ein neues Scratch-Hamster-Programm zu erstellen, klicken Sie bitte in der Toolbar des Editor-Fensters auf den Neu-Button (1. Toolbar-Button von links). Er erscheint ein Fenster, in dem der Programmtyp ausgewählt werden muss. Wählen Sie hier den Programmtyp „Scratch-Programm“. Nach dem Drücken des OK-Buttons erscheint im Editor-Fenster (anstelle eines normalen textuellen Editors) ein so genanntes *Scratch-Editorfenster*. Es besteht aus drei Bereichen (siehe auch Abbildung 11.3): der *Kategorienauswahl*, der *Blockpalette* und dem *Programmbereich*.

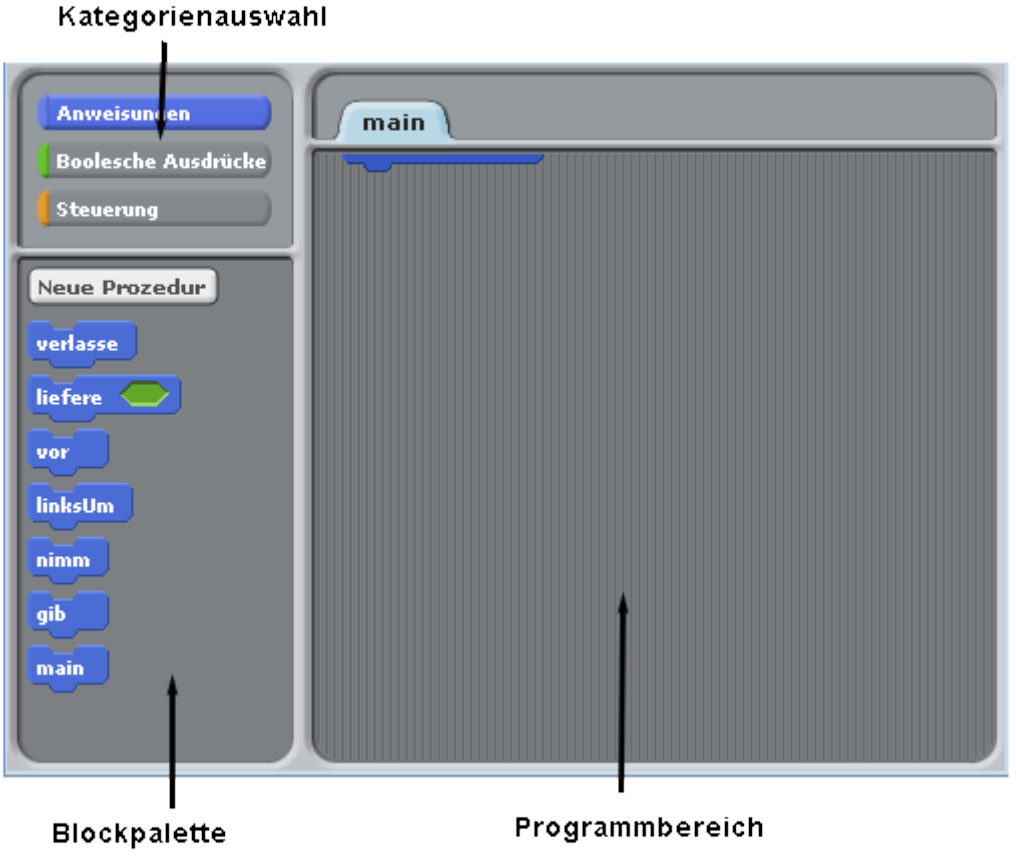

Abbildung 11.3: Scratch-Editorfenster

11.4.1 Kategorienauswahl

In der Kategorienauswahl finden sich die drei Buttons „Anweisungen“, „Boolesche Ausdrücke“ und „Steuerung“. Durch Klick auf einen der Buttons erscheinen entsprechende Blöcke der jeweiligen Kategorie in der Blockpalette.

11.4.2 Blockpalette

In der Blockpalette werden die Blöcke der in der Kategorienauswahl aktuell ausgewählten Kategorie angezeigt. Diese – genauer gesagt Kopien hiervon – lassen sich per Drag-und-Drop mit der Maus in den Programmbereich ziehen, um Programme zu erstellen. Wählen Sie hierzu den entsprechenden Block, klicken Sie ihn mit der Maus (linke Taste) an, ziehen Sie die Maus bei gedrückter linker Taste in den Programmbereich und lassen Sie die Maustaste los.

Folgende Blöcke werden standardmäßig in der Blockpalette angezeigt (siehe auch die Abbildungen 11.4, 11.5 und 11.6):

- Anweisungsblöcke (Kategorie „Anweisungen“):
  - vor: Repräsentiert den Hamster-Grundbefehl vor.
  - linksUm: Repräsentiert den Hamster-Grundbefehl linksUm.

- nimm: Repräsentiert den Hamster-Grundbefehl nimm.
- gib: Repräsentiert den Hamster-Grundbefehl gib.
- verlasse: Repräsentiert die return-Anweisung zum Verlassen einer Prozedur.
- liefere <boolescher Ausdruck>: Repräsentiert die boolesche-return-Anweisung zum Verlassen einer booleschen Funktion.
- main: Repräsentiert die main-Prozedur.
- Über den Button „Neue Prozedur“ lassen sich neue Prozeduren definieren.

Abbildung 11.4: Blockpalette der Kategorie „Anweisungen“

- Boolesche-Ausdrucks-Blöcke (Kategorie „Boolesche Ausdrücke“):
  - vornFrei: Repräsentiert den Hamster-Testbefehl vornFrei.
  - kornDa: Repräsentiert den Hamster-Testbefehl kornDa.
  - maulLeer: Repräsentiert den Hamster-Testbefehl maulLeer.
  - wahr: Repräsentiert das boolesche Literal „true“.
  - falsch: Repräsentiert das boolesche Literal „false“.
  - <boolescher Ausdruck> und <boolescher Ausdruck>: Repräsentiert den booleschen Und-Operator.
  - <boolescher Ausdruck> oder <boolescher Ausdruck>: Repräsentiert den booleschen Oder-Operator.
  - nicht <boolescher Ausdruck >: Repräsentiert den booleschen Nicht-Operator.
  - Über den Button „Neue Funktion“ lassen sich neue boolesche Funktionen definieren.

Abbildung 11.5: Blockpalette der Kategorie „Boolesche Ausdrücke“

- Steuerungsblöcke (Kategorie „Steuerung“):

  Bei den Steuerungsblöcken handelt sich um spezielle Anweisungsblöcke.

  - falls: Repräsentiert die if-Anweisung.
  - falls-sonst: Repräsentiert die if-else-Anweisung.
  - solange <boolescher Ausdruck>: Repräsentiert die while-Schleife.
  - wiederhole-solange <boolescher Ausdruck>: Repräsentiert die do-while-Schleife.

Abbildung 11.6: Blockpalette der Kategorie „Steuerung“

11.4.3 Programmbereich

Im Programmbereich liegen die Scratch-Hamster-Programme. Standardmäßig ist hier die main-Prozedur geöffnet.

Blöcke, die aus der Blockpalette in den Programmbereich gezogen wurden, lassen sich hier per Drag-and-Drop-Aktion weiter hin und herschieben. Blöcke lassen sich aus dem Programmbereich wieder entfernen, indem man sie in die Kategorienauswahl oder die Blockpalette zieht. Alternativ kann man auch oberhalb eines zu entfernenden Blocks ein Popup-Menü aktivieren und hierin das Menü-Item „entfernen“ anklicken.

11.4.4 Anweisungssequenzen

Befindet sich im Programmbereich ein Anweisungsblock A und zieht man einen anderen Anweisungsblock B in die Nähe (ober- und unterhalb), dann erscheint irgendwann ein transparenter grauer Bereich (siehe Abbildung 11.7). Lässt man B nun los, schnappen Zapfen (unten an den Anweisungsblöcken) und Mulde (oben an den Anweisungsblöcken) der beiden Blöcke ein und A und B bilden nun eine Anweisungssequenz (in Scratch wird der Begriff „Stapel“ verwendet). Zieht man im Folgenden den obersten Block einer Anweisungssequenz im Programmbereich hin und her, wird automatisch der gesamte Block mit verschoben. Eine Anweisungssequenz kann man wieder trennen, indem man einen mittleren Block verschiebt. Darunter liegende Blöcke werden dann mit verschoben, darüber liegende Blöcke bleiben liegen und werden abgetrennt.

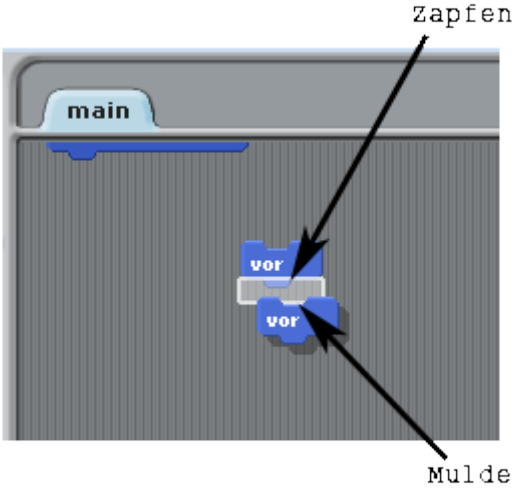

Abbildung 11.7: Bilden von Anweisungssequenzen

11.4.5 Programme

Scratch-Hamster-Programme starten immer durch Ausführung der main-Prozedur. Ausgeführt wird dabei die Anweisungssequenz, die mit der main-Prozedur verbunden ist. Hierzu besitzt die Prozedur am oberen Rand des Programmbereichs einen Zapfen (den sogenannten *Prozedurzapfen*). Hier muss man die entsprechende Anweisungssequenz andocken (siehe Abbildung 11.8).

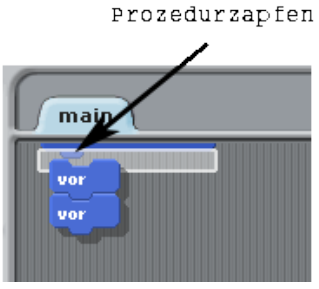

Abbildung 11.8: Definition von Prozeduren

Zwischenbemerkung:

Nun sind wir so weit, dass wir ein erstes kleines Scratch-Hamster-Programm selbst erstellen und testen können. Erstellen Sie zum Beispiel ein Programm, in dem sich der Hamster zwei Kacheln nach vorne bewegt, sich dann linksum dreht und wiederum zwei Kacheln nach vorne springt (siehe Abbildung 11.9).

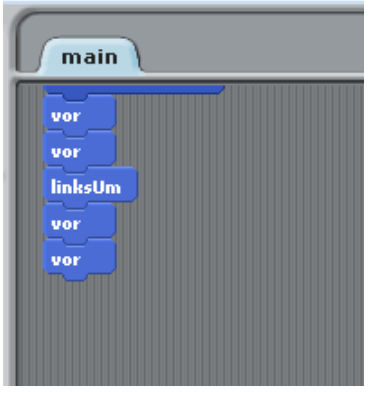

Abbildung 11.9: Erstes kleines Scratch-Hamster-Programm

11.4.6 Kontrollstrukturen

Die Blöcke, die die Kontrollstrukturen repräsentieren, lassen sich analog zu den Anweisungsblöcken handhaben. Sie besitzen jedoch als weitere Unterkomponenten einen Bedingungsbereich (grün) und einen Innenbereich (siehe Abbildung 11.10).

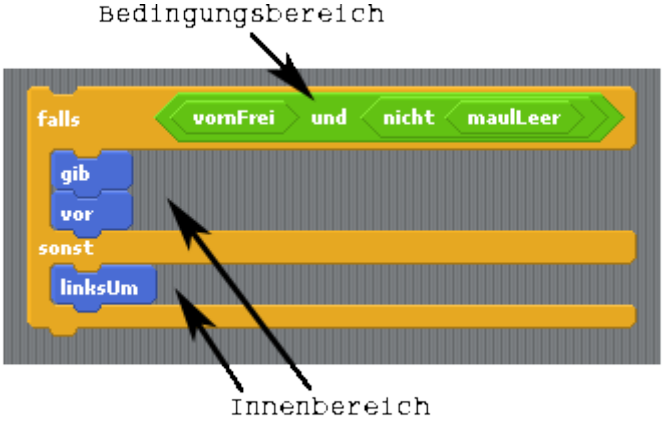

Abbildung 11.10: Steuerungsblöcke

In den Bedingungsbereich lassen sich beliebige Boolesche-Ausdrucks-Blöcke ziehen, die die Bedingung der entsprechenden Kontrollstruktur repräsentieren. Ein leerer Bedingungsbereich entspricht implizit dem Wert „wahr“. Auch die Boolesche-Ausdruck-Blöcke, die die booleschen Operatoren repräsentieren, enthalten analog zu handhabende Bedingungsbereiche. Damit lassen sich komplexe Bedingungen realisieren.

In den Innenbereich eines Kontrollstrukturblockes lassen sich wiederum Anweisungsblöcke und Stapel ziehen. Diese entsprechen dabei der true- oder false-Anweisung einer if -Anweisung oder der Iterationsanweisung einer Schleife.

11.4.7 Prozeduren und boolesche Funktionen

Das Konzept der Definition neuer Prozeduren und Funktionen existiert im Original-Scratch (leider) nicht. Wir haben versucht, es möglichst harmonisch zu integrieren.

Definition

Bei Auswahl von „Anweisungen“ in der Kategorienauswahl erscheint in der Blockpalette ein Button „Neue Prozedur“, über den eine neue Prozedur definiert werden kann. Bei Auswahl von „Boolesche Ausdrücke“ in der Kategorienauswahl erscheint in der Blockpalette ein Button „Neue Funktion“, über den eine neue boolesche Funktion definiert werden kann. Nach Drücken eines der beiden Buttons wird der Programmierer nach dem Namen der neuen Prozedur bzw. Funktion gefragt. Der anzugebende Name muss dabei den Java-Bezeichner-Konventionen entsprechen. Nach Drücken des OK-Buttons erscheint in der Blockpalette ein neuer Block, der die entsprechende Prozedur bzw. Funktion repräsentiert. Weiterhin erscheint oben im Programmbereich einer neuer Tab. Durch Anklicken eines Tabs wird der entsprechende Prozedur- bzw. Funktionsrumpf im Programmbereich angezeigt. Die aktuelle Prozedur wird durch einen hellblauen Tab angezeigt.

Die Definition einer Prozedur bzw. Funktion ist völlig analog zu der Definition der main-Prozedur durch Hineinziehen entsprechender Blöcke möglich (siehe oben). Wird eine Prozedur bzw. Funktion aufgerufen, wird dabei die Anweisungssequenz ausgeführt, die an den Prozedurrapfen der entsprechenden Prozedur bzw. Funktion angedockt ist.

Aufruf

Die neuen Blöcke, die bei der Definition einer Prozedur bzw. Funktion in die Blockpalette integriert wurden, können analog zu anderen Blöcken verwendet werden. Prozedurblöcke sind dabei spezielle Anweisungsblöcke, Funktionsblöcke sind spezielle Boolesche-Ausdruck-Blöcke.

Handhabung

Durch Doppelklick auf einen Prozedur- bzw. Funktionsblock in der Blockpalette wird die entsprechende Funktion im Programmbereich geöffnet. Durch Anklicken des entsprechenden Tabs im Programmbereich lässt sich zwischen verschiedenen Prozeduren und Funktionen hin und her wechseln.

Zu jeder Prozedur bzw. Funktion existiert ein Popup-Menü, das sich durch Anklicken mit der Maus (rechte Maustaste) über dem entsprechenden Block in der Blockpalette oder

dem Tab im Programmbereich öffnen lässt. Im Popup-Menü existieren zwei Menü-Items zum Umbenennen und Löschen von Prozeduren und Funktionen.

Besonderheiten

Fehlt am Ende der Ausführung einer booleschen Funktion eine boolesche-return-Anweisung, so wird automatisch der Wert „true“ geliefert. Dasselbe gilt für den Fall der Ausführung einer (normalen) return-Anweisung innerhalb einer booleschen Funktion.

Wird innerhalb einer Prozedur eine boolesche-return-Anweisung ausgeführt, wird der gelieferte Wert ignoriert.

Auslagern von Blöcken

Möchte man ganze Stapel oder Teile von Stapeln in Prozeduren bzw. Funktionen auslagern, kann man das erreichen, indem man oberhalb eines Blockes ein Popup-Menü aktiviert und hierin das Menü-Item „Auslagern in Prozedur“ bzw. „Auslagern in Funktion“ anklickt. Es wird dann eine neue Prozedur bzw. Funktion definiert, in die alle Blöcke des Stapels unterhalb des betroffenen Blockes ausgelagert werden. Die ausgelagerten Blöcke werden automatisch durch einen Block der neuen Prozedur bzw. Funktion ersetzt.

Auch Boolescher-Ausdruck-Blöcke lassen sich in Funktionen auslagern. In die neue Funktion wird dann automatisch eine entsprechende Boolesche-Return-Anweisung integriert.

11.5 Ausführen von Scratch-Hamster-Programmen

Zum Ausführen von Scratch-Hamster-Programmen dienen die entsprechenden Buttons (Start bzw. Fortführen, Pause und Stopp) der Toolbar. Scratch-Hamster-Programme werden dabei interpretativ ausgeführt, müssen also nicht (und können auch nicht) compiliert werden. Ein Abspeichern ist vor der Ausführung nicht unbedingt notwendig. Während der Ausführung eines Programms (was man daran erkennt, dass der Stopp-Button enabled (also anklickbar) ist), ist keine Änderung an dem Programm möglich.

Weiterhin ist es möglich, ein Scratch-Programm Schritt für Schritt, d.h. Anweisung für Anweisung, auszuführen. Hierzu dient der „Schritt-hinein“-Button der Toolbar. Der Block der im nächsten Schritt auszuführenden Anweisung wird dabei im Programmbereich durch eine rote Farbe markiert. Es wird automatisch in neu definierte Prozeduren bzw. Funktionen verzweigt und die entsprechenden Tabs werden automatisch geöffnet. Das schrittweise Ausführen eines Scratch-Hamster-Programms ist von Anfang an oder auch während der Ausführung nach Anklicken des Pause-Buttons möglich.

11.6 Generieren von Java-Hamster-Programmen

Seit der Version 2.8.2 des Hamster-Simulators ist es möglich, aus Scratch-Hamster-Programmen äquivalente imperative Java-Hamster-Programme zu generieren. Öffnen Sie dazu das umzuwandelnde Scratch-Hamster-Programm und wählen Sie im Menü „Datei“ den Eintrag „Generieren“. Es öffnet sich im Eingabebereich des Editor-Fensters ein neuer Karteireiter mit dem generierten Java-Hamster-Programm.

Es ist zu beachten, dass bei der Generierung unter Umständen Java-Programme entstehen, die sich nicht compilieren lassen. Hintergrund hierfür ist, dass return-Anweisungen in Scratch anders behandelt werden als in Java. Sie müssen also unter Umständen noch fehlende return-Anweisungen in das generierte Java-Programm integrieren.

11.7 Beispielprogramme

Im Unterordner „scratch“ des Ordners „beispielprogramme“ finden Sie einige Scratch-Hamster-Programme als Beispiele.

12 Hamstern mit endlichen Automaten

Seit der Version 2.9 unterstützt der Hamster-Simulator das Hamstern mit endlichen Automaten. Die Idee hierzu wurde von „Kara dem Marienkäfer“ übernommen (siehe http://www.swisseduc.ch/informatik/karatojava/kara/). Herzlichen Dank an Raffaela Ferrari, die im Rahmen ihrer Bachelorarbeit die Integration dieses Konzeptes in den Hamster-Simulator vorgenommen hat.

12.1 Endliche Automaten

Die grundlegende Theorie endlicher Automaten entnehmen Sie am besten dem entsprechenden Wikipedia-Eintrag: http://de.wikipedia.org/wiki/Endlicher_Automat. Ein endlicher Automat (EA, auch Zustandsmaschine, Zustandsautomat; englisch: finite state machine (FSM)) ist danach „ein Modell eines Verhaltens, bestehend aus Zuständen, Zustandsübergängen und Aktionen“.

12.2 Hamster-Automaten

Die im Hamster-Modell verwendete Form eines endlichen Automaten ist von den Mealy-Automaten (siehe http://de.wikipedia.org/wiki/Mealy-Automat) abgeleitet und wird im Folgenden als „Hamster-Automat“ bezeichnet.

Ein konkreter Hamster-Automat besteht aus *Zuständen* und *Verbindungen* zwischen Zuständen (so genannte *Transitionen*). Zustände werden grafisch durch Kreise repräsentiert, Verbindungen durch Pfeile, die von einem Zustand zu einem anderen Zustand (oder auch dem gleichen) führen.

Den Pfeilen zugeordnet sind Beschriftungen, die jeweils aus zwei Teilen bestehen. Der erste Teil ist ein boolescher Ausdruck, in dem die Hamster-Testbefehle benutzt werden können. Der zweite Teil ist eine Sequenz von Hamster-Befehlen.

Genau ein Zustand eines Hamster-Automaten ist als *Startzustand* ausgezeichnet (zu erkennen an einem eingehenden Pfeil). Bestimmte Zustände können als *Endzustände* markiert sein (Kreise mit doppelter Umrandung).

Wird ein Hamster-Automat ausgeführt, passiert folgendes:

- Genau ein Zustand ist jeweils aktiv. Anfangs ist dies der Startzustand.
- Es wird überprüft, ob vom aktiven Zustand eine Transition ausgeht, deren boolescher Ausdruck den Wert *true* liefert. Ist dies der Fall, wird die zugehörige Sequenz von Hamster-Befehlen ausgeführt. Der Zustand, in den die Transition führt, ist anschließend der neue aktive Zustand.

Gibt es vom aktiven Zustand ausgehend keine gültige Transition, dann gibt es zwei Fälle:

- Handelt es sich bei dem Zustand um einen Endzustand, dann ist das Programm erfolgreich beendet.

- Handelt es sich beim dem Zustand um keinen Endzustand, dann wird das Programm mit einem Fehler beendet.

Weiterhin ist zu beachten, dass unterschieden wird zwischen *deterministischen* und *nicht-deterministischen* Hamster-Automaten. Ist ein Hamster-Automat deterministisch, dann darf es zu jedem Zeitpunkt immer nur eine gültige Transition geben, die aus dem aktiven Zustand herausführt. Das wird zur Laufzeit überprüft und gegebenenfalls ein Fehler gemeldet. Ist ein Hamster-Automat nicht-deterministisch, dann darf es mehrere gültige Transitionen geben. Welche dann ausgeführt wird, ist vom Zufall abhängig.

Abbildung 12.1 zeigt ein Beispiel für einen Hamster-Automaten. Führt man das Programm aus, läuft der Hamster zur nächsten Wand und dreht sich dort um.

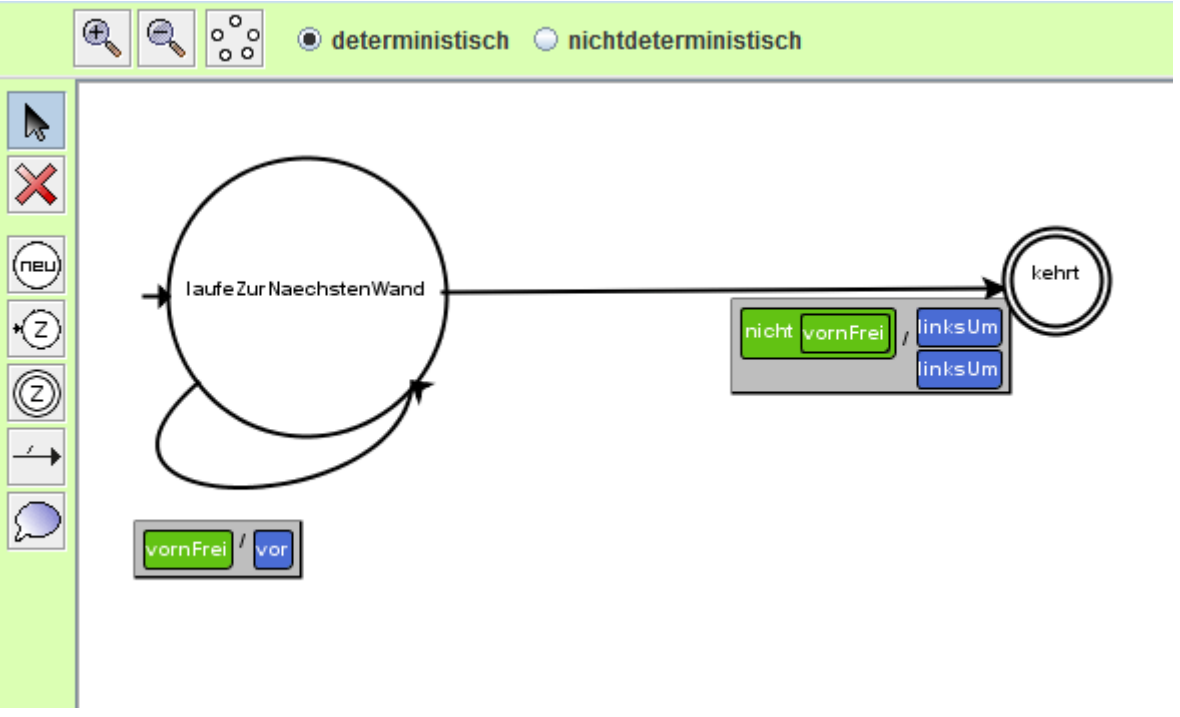

Abbildung 12.1: Beispiel Hamster-Automat

In diesem Automaten, der als deterministisch eingestellt ist, existieren zwei Zustände. Einer trägt den Namen *laufeZurNaechstenWand* und ist der Startzustand, der andere den Namen *kehrt*. Der Zustand *kehrt* ist ein Endzustand.

Außerdem existieren zwei Transitionen. Die erste geht vom Zustand *laufeZurNaechstenWand* aus und führt auch wieder in diesen Zustand hinein. Die zweite geht vom Zustand *laufeZurNaechstenWand* aus und führt zum Zustand *kehrt*. Die erste Transition besitzt den booleschen Ausdruck `vornFrei` und die Befehlssequenz `vor`. Die zweite Transition besitzt den booleschen Ausdruck `nicht vornFrei` und die Befehlssequenz `linksUm, linksUm`.

Bei der Ausführung startet das Programm im Startzustand *laufeZurNaechstenWand*. Es werden die beiden ausgehenden Transitionen überprüft. Steht der Hamster vor einem freien Feld, wird die erste Transition ausgewählt (weil der boolesche Ausdruck `vornFrei` den Wert *true* liefert) und der Befehl `vor` wird ausgeführt. Der nächste aktive Zustand ist

wiederum der Zustand *laufeZurNaechstenWand*. Steht der Hamster vor einer Wand, wird die zweite Transition ausgewählt und die Befehlssequenz `linksUm, linksUm` ausgeführt. Der nächste aktive Zustand ist nun der Zustand *kehrt*. Da vom diesem Zustand keine Transitionen ausgehen, ist das Programm beendet; und zwar erfolgreich, weil der Zustand *kehrt* ein Endzustand ist.

12.3 Voraussetzungen

Um mit dem Hamster-Simulator Hamster-Automaten entwickeln und ausführen zu können, muss die Property `fsm` auf `true` gesetzt sein. Standardmäßig ist das der Fall. Um den Automaten-Modus auszuschalten, öffnen Sie bitte die Datei `hamster.properties` und ersetzen Sie dort die Zeile `fsm=true` durch die Zeile `fsm=false`.

12.4 Erstellen von Hamster-Automaten

Um einen neuen Hamster-Automatenzu erstellen, klicken Sie bitte in der Toolbar des Editor-Fensters auf den Neu-Button (1. Toolbar-Button von links). Er erscheint ein Fenster, in dem der Programmtyp ausgewählt werden muss. Wählen Sie hier den Programmtyp „Endlicher Automat“. Nach dem Drücken des OK-Buttons erscheint im Editor-Fenster (anstelle eines normalen textuellen Editors) ein so genanntes *Automat-Editorfenster*. Es besteht aus drei Bereichen (siehe auch Abbildung 12.1): dem *Zeichnen-Menü* auf der linken Seite, der *Automaten-Menü* oben und dem *Programmbereich*.

12.4.1 Zeichnen-Menü

Das Zeichnen-Menü an der linken Seite beinhaltet folgende Funktionalität (von oben nach unten) durch Anklicken des entsprechenden Buttons, wobei immer nur ein Button zur gleichen Zeit aktiv sein kann:

- [Editieren] Durch die vielfältigen Möglichkeiten in diesem Modus wird dieser Punkt im nächsten Abschnitt behandelt.
- [Löschen] Durch Klicken auf ein Element (Zustand, Transition) werden dieses und alle davon abhängigen Elemente gelöscht. Dabei gilt bei übereinanderliegenden Elementen, dass stets das oberste Element gelöscht wird. Der Startzustand kann nicht gelöscht werden. Beschriftungen von Transitionen können nur gelöscht werden, wenn für diese Transition eine weitere Beschriftung vorhanden ist. Anmerkung: Werden von einem Zustand zu einem anderen Zustand zwei Transitionen definiert, wird nur ein Pfeil gezeichnet. Die beiden Transitionen werden durch so genannte *Beschriftungen* dargestellt.
- [Zustand erzeugen] Durch Klicken an einen bestimmten Punkt im Programmbereich wird an dieser Stelle der Mittelpunkt des neu erzeugten

Zustands gesetzt. Der Zustand bekommt dabei eine Standardbenennung (z+<Zahl>). Der erste erzeugte Zustand wird als Startzustand definiert.

- [Startzustand markieren] Durch Klicken auf einen Zustand wird dieser als Startzustand definiert. Der vorher als Startzustand markierte Zustand wird zu einem normalen Zustand abgeändert.
- [Endzustand markieren] Durch Klicken auf einen Zustand wird dieser als Endzustand definiert, wenn es vorher keiner war, oder als Nicht-Endzustand definiert, wenn es schon ein Endzustand gewesen ist.
- [Transition erzeugen] Klicken Sie zunächst den Zustand an, von dem die Transition ausgehen soll, und ziehen sie die Maus bei gehaltener Maustaste zu dem Zustand, bei dem die Transition enden soll. Nach Loslassen der Maus, wird ein neues Fenster geöffnet wird, indem die Transition genauer definiert werden kann. Darauf wird später eingegangen.
- [Kommentar erzeugen] Durch Klicken an einen bestimmten Punkt im Programmbereich wird die linke obere Ecke des neu erzeugten Kommentar gesetzt. Zunächst aber wird ein Textfeld an dieser Stelle gesetzt, in dem der Kommentar eingegeben werden kann. Dieser kann durch Enter oder klicken an einen Punkt außerhalb der Eingabe bestätigt werden. Durch Drücken der Taste „Esc“ wird das Textfeld verlassen, ohne einen Kommentar zu erzeugen.

Nach Ausführung eines Menüpunktes wird jedes Mal automatisch wieder in den Editieren-Modus gewechselt.

12.4.2 Editieren-Modus

In diesem Modus können Kommentare und Zustände verschoben werden, indem das entsprechende Element mit der Maus angeklickt und per Drag-and-Drop an die gewünschte Stelle gezogen wird.

Dabei ist es bei Zuständen möglich, mehrere gleichzeitig zu verschieben. Dazu werden diese angeklickt, während die Strg-Taste gedrückt wird. Nach Loslassen dieser werden die entsprechenden Zustände hervorgehoben und sie können nun verschoben werden, indem einer dieser Zustände an die gewünschte Stelle gezogen wird.

Weiterhin können Zustände und Kommentare umbenannt bzw. der Text geändert werden, indem ein Textfeld durch einen Doppelklick auf das entsprechende Element geöffnet wird. Bei Zuständen ist die Besonderheit, dass Zustände nicht in z + Zahl umbenannt werden dürfen, weil es sich um die Standardbenennung der Zustände handelt. Außerdem müssen sich die Namen aller Zustände unterscheiden. Die Namen müssen dabei aus Buchstaben und Ziffern bestehen.

Transitionen können an andere Zielzustände umgelegt wird, indem die Pfeilspitze der Transition auf den gewünschten Zielzustand gezogen wird. Außerdem ist es möglich Transitionen zu biegen. Dazu wird auf den Transitionspfeil doppelgeklickt und der erscheinende rote Punkt so gezogen, bis die Transition die gewünschte Biegung besitzt.

Eine Beschriftung einer Transition kann geändert werden, indem durch einen Doppelklick auf die Beschriftung das Fenster zur Änderung geöffnet wird, welches im folgenden Abschnitt beschrieben ist.

Jedes Element besitzt in diesem Modus ein Kontextmenü bzw. Popup-Menü, indem die für das Element möglichen Funktionen aufgeführt werden. Das Kontextmenü wird im Allgemeinen geöffnet, wenn die rechte Maustaste gedrückt wird, während sich die Maus auf dem entsprechenden Element befindet.

Auch der Programmbereich selber besitzt ein Kontextmenü, welches sich auf einer freien Stelle öffnet. Darin ist es möglich, den Graphen zu vergrößern oder zu verkleinern, zu layouten und neue Kommentare, Zustände oder Transitionen zu erstellen. Bei letzteren wird ein spezielles Auswahl-Fenster geöffnet, in dem der Start- und Endzustand festgelegt werden kann. Danach öffnet sich das Fenster zur Festlegung der einzelnen Bestandteile dieser Transition.

12.4.3 Definition einer Transition

Das Fenster für die Definition einer Transition ist in zwei Bereiche aufgeteilt, wie Abbildung 12.2 zeigt. Im oberen Bereich kann der boolesche Ausdruck (auch Bedingung oder Input genannt) der Transition, im unteren Bereich die Befehlssequenz (auch Output genannt) festgelegt werden. Die Vorgehensweise in beiden Bereichen ist aber die Gleiche. Im linken Feld des jeweiligen Bereiches befinden sich die zur Auswahl stehenden Elemente. Um ein Element auszuwählen, wird dieses per Drag-and-Drop in das jeweils rechte Feld gezogen. Dort muss es an eine hervorgehobene Stelle innerhalb des grünen bzw. blauen Feldes platziert werden. Diese Stellen werden sichtbar (graue Hervorhebung), sobald die Maus über den entsprechenden Bereich fährt. Während bei der Bedingung genau ein Element ausgewählt wird, in das gegebenenfalls Kinderelemente gezogen werden können, so ist es bei der Befehlssequenz möglich, eine Ausgabeliste zu erstellen, die aus mehreren Elementen bestehen kann.

Um ein Element wieder aus der Auswahl-Liste in den rechten Bereichen zu entfernen, muss der Menüpunkt „lösch**e**n“ im Kontextmenü des Elementes ausgewählt werden. Dabei werden gegebenenfalls auch untergeordnete Elemente gelöscht. Das Kontextmenü für ein Element wird dabei dadurch geöffnet, dass der rechte Mausbutton geklickt wird, während sich der Mauszeiger auf dem entsprechenden Element befindet.

Durch Anklicken des „fertig“-Buttons wird die Definition der Transition bestätigt. Wird das Fenster geschlossen oder die Auswahl abgebrochen, so werden Standardwerte als Ein- und Ausgabe genommen. Gleiches geschieht, wenn in der Auswahl nichts festgelegt wurde. Dies soll ungültige Transitionen vermeiden.

Im Input-Bereich gibt es außer den Hamster-Testbefehlen und den booleschen Operatoren (und, oder, nicht) noch das Epsilon. Das ist gleichbedeutend zum Wert true. Im Output-Bereich steht das Epsilon für die leere Anweisung (die nichts bewirkt).

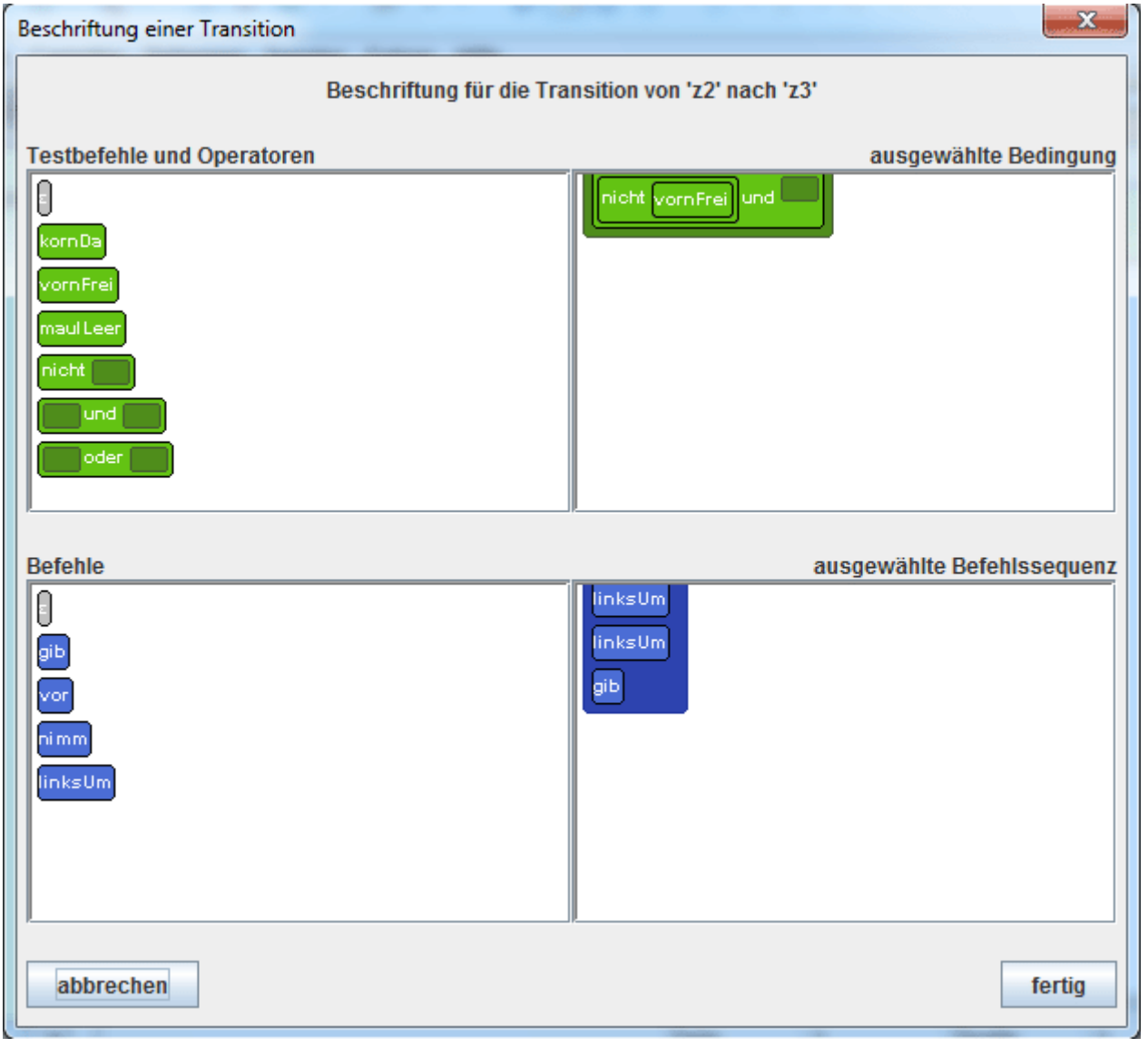

Abbildung 12.2: Transitionsfenster

Nach ihrer Definition im Transitionsfenster wird die Transition entsprechend beschriftet. Bestand vorher schon eine Transition zwischen dem ausgehenden Zustand zum Folgezustand, wird nur eine zusätzliche Beschriftung an den schon existierenden Transitionspfeil angehängt. Der Input und Output werden abgeschnitten, wenn der Input zu lang ist bzw. der Output mehr als zwei Befehle enthält. In jedem Zeichnen-Modus kann aber die komplette Beschriftung betrachtet werden, wenn die Maus über den entsprechenden Bereich fährt.

12.4.4 Automaten-Menü

Das Automaten-Menü oberhalb des Programmbereiches beinhaltet folgende Funktionalität durch Auswahl des entsprechenden Menüpunktes:

[Vergrößern] Durch Klicken auf diesen Menüpunkt wird der Graph im Programmbereich um eine Stufe vergrößert.

[Verkleinern] Durch Klicken auf diesen Menüpunkt wird der Graph um eine Stufe verkleinert.

[Layouten] Durch Klicken auf diesen Menüpunkt werden die Zustände im Graphen des Hamster-Automaten in einem Kreis neu angeordnet.

[Auswahl der Art des Hamster-Automaten] Durch Auswahl einer der beiden Punkte deterministisch und nicht-deterministisch wird der entsprechende Typ dem Hamster-Automaten zugeordnet. Dies wirkt sich bei der Ausführung des Hamster-Automaten-Programmes aus, da bei einem deterministischen Hamster-Automaten Fehlermeldungen geworfen werden, wenn es einen Zustand mit mehreren ausführbaren Transitionen gibt.

12.5 Weitere Funktionen

Ausführung: Wie andere Programme auch können Hamster-Automaten gestartet, gestoppt, pausiert und fortgesetzt werden. Während der Ausführung werden die gerade aktiven Elemente rot hervorgehoben. Auch eine schrittweise Ausführung ist möglich (Funktion „Schritt hinein“).

Speichern/Laden: Hamster-Automaten können (bspw. über das Menü „Datei“) gespeichert und später wieder geladen werden. Im Dateibaum im Editorfenster werden entsprechende Dateien durch das Symbol „gelbes Kästchen mit rotem Punkt“ dargestellt.

Drucken: Hamster-Automaten können (bspw. über das Menü „Datei) gedruckt werden.

Generieren: Über das Menü-Item „Generieren“ im Menü „Datei“ lässt sich aus einem Hamster-Automaten ein gleichbedeutendes imperatives Java-Hamster-Programm generieren.

13 Hamstern mit Programmablaufplänen

Seit der Version 2.9 unterstützt der Hamster-Simulator das Hamstern mit Programmablaufplänen (Flowcharts). Herzlichen Dank an Gerrit Apeler, der im Rahmen seiner Bachelorarbeit die Integration dieses Konzeptes in den Hamster-Simulator vorgenommen hat.

13.1 Programmablaufpläne

Die grundlegende Theorie von Programmablaufplänen entnehmen Sie am besten dem entsprechenden Wikipedia-Eintrag: http://de.wikipedia.org/wiki/Programmablaufplan. Ein Programmablaufplan (Abkürzung: PAP) ist danach ein Ablaufdiagramm für ein Computerprogramm, das auch als Flussdiagramm (engl. flowchart) oder Programmstrukturplan bezeichnet wird. Es ist eine graphische Darstellung zur Umsetzung eines Algorithmus in einem Programm und beschreibt die Folge von Operationen zur Lösung einer Aufgabe.

13.2 Hamster-PAPs

Als Elemente enthalten Hamster-PAPs Start/Stop-Elemente, Operationen, Unterprogramme, Verzweigungen, Kommentare und Pfeile.

- Start-Element: Jedes Hamster-PAP-Programm und jedes selbst definierte Unterprogramm muss mit einem Start-Element beginnen.
- Stop-Element: Jedes Hamster-PAP-Programm und jedes selbst definierte Unterprogramm muss bei einem Stop-Element enden.
- Operationen: Als vordefinierte Operationen stehen die vier Hamster-Befehle vor, linksUm, gib und nimm zur Verfügung.
- Unterprogramme: Es ist möglich, Unterprogramme zu definieren und im Programm zu verwenden.
- Verzweigungen: Als vordefinierte Verzweigungen stehen die drei Hamster-Testbefehle vornFrei, kornDa und maulLeer zur Verfügung.
- Kommentare: Es ist möglich Kommentare zu definieren und diese mit Elementen zu verbinden.
- Pfeile: Elemente lassen sich untereinander mit Pfeilen verbinden, um somit den Programmfluss zu definieren.

Abbildung 13.1 zeigt beispielhaft einen Hamster-PAP.

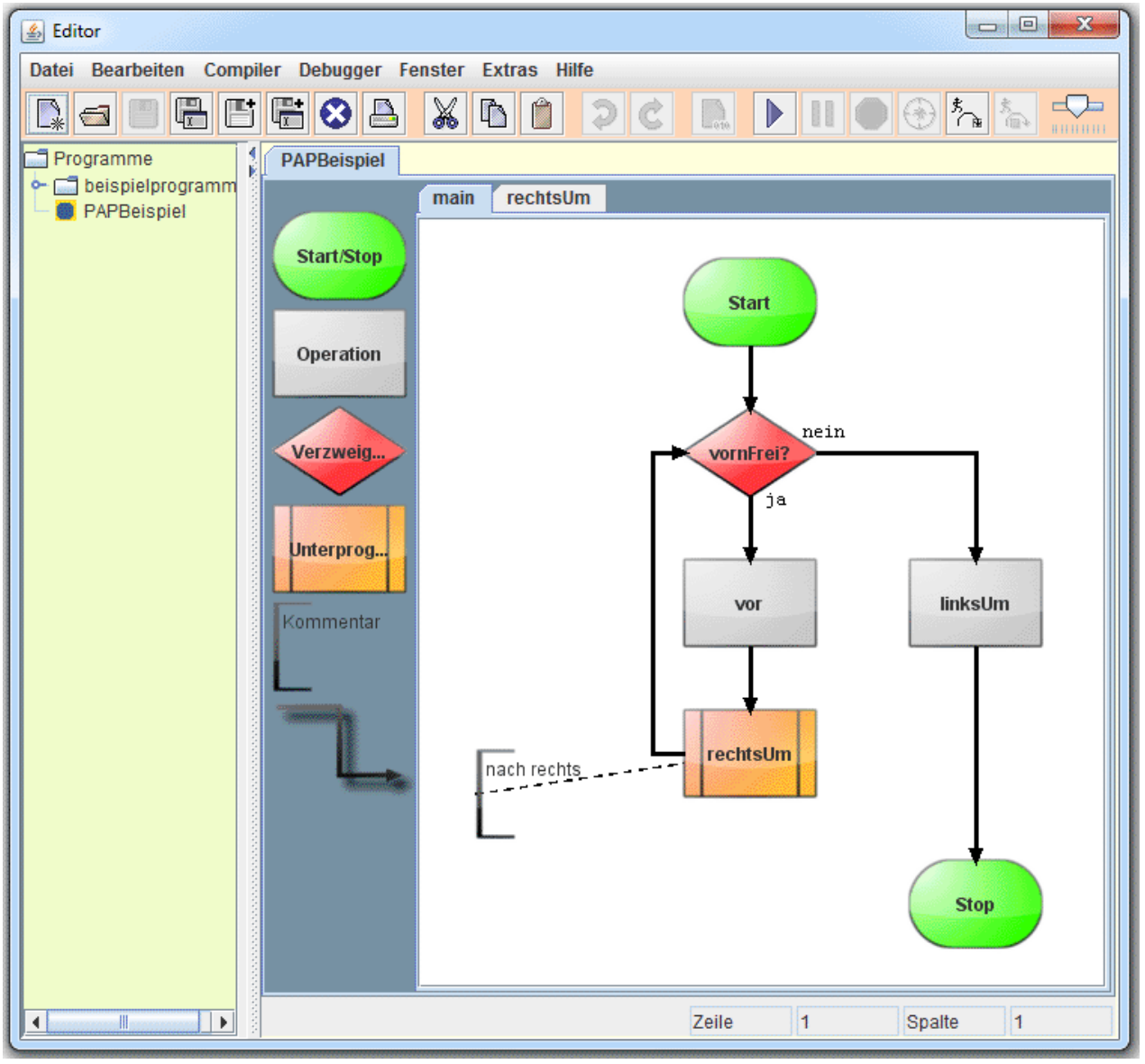

Abbildung 13.1: Elemente von Hamster-PAPs

13.3 Voraussetzungen

Um mit dem Hamster-Simulator Hamster-PAPs entwickeln und ausführen zu können, muss die Property `flowchart` auf `true` gesetzt sein. Standardmäßig ist das der Fall. Um den PAP-Modus auszuschalten, öffnen Sie bitte die Datei `hamster.properties` und ersetzen Sie dort die Zeile `flowchart=true` durch die Zeile `flowchart=false`.

13.4 Erstellen von Hamster-PAPs

Um einen neuen Hamster-PAP zu erstellen, klicken Sie bitte in der Toolbar des Editor-Fensters auf den Neu-Button (1. Toolbar-Button von links). Er erscheint ein Fenster, in dem der Programmtyp ausgewählt werden muss. Wählen Sie hier den Programmtyp „Programmablaufplan“. Nach dem Drücken des OK-Buttons erscheint im Editor-Fenster (anstelle eines normalen textuellen Editors) ein so genanntes *PAP-Editorfenster*. Es besteht aus zwei Bereichen (siehe auch Abbildung 13.1): dem *Auswahl-Menü* auf der linken Seite und dem *Programmbereich*.

13.4.1 Auswahl-Menü

Das Auswahl-Menü an der linken Seite beinhaltet folgende Funktionalität (von oben nach unten):

- [Start-Stop-Element] Mit diesem grünen Symbol kann ein Start- oder Stop-Element in den Programmbereich gebracht werden.
- [Operation-Element] Mit diesem grauen Symbol kann ein Operation-Element, d.h. ein Hamster-Befehl, in den Programmbereich gebracht werden.
- [Verzweigung-Element] Mit diesem roten Symbol kann ein Verzweigung-Element, d.h. ein Hamster-Testbefehl, in den Programmbereich gebracht werden.
- [Unterprogramm-Element] Mit diesem orangenem Element kann ein neues Unterprogramm-Element in den Programmbereich gebracht werden.
- [Kommentar-Element] Mit dem Kommentar-Element kann ein neues Kommentar-Element in den Programmbereich gebracht werden.
- [Pfeil-Element] Durch Klicken auf den Pfeil lässt sich der Pfeil-Zeichnen-Modus ein- und ausschalten (siehe Abschnitt 13.4.4).

Die Erzeugung eines Elementes (mit Ausnahme von Pfeilen) erfolgt dabei auf die Art und Weise, dass das entsprechende Element im Auswahl-Menü angeklickt und die Maus bei gedrückter Maustaste in den Programmbereich gezogen wird („Draggen“). Lässt man die Maustaste los, erscheint eine Elementtyp-spezifische Abfrage und anschließend wird das Element im Programmbereich angezeigt. Aus Gründen einer besseren Übersichtlichkeit ist dem Programmbereich ein unsichtbares Raster hinterlegt, das dafür sorgt, dass Elemente nur an bestimmten Stellen stehen können. In dieses Raster „schnappen“ die Elemente ein.

13.4.2 Programmbereich

Der Programmbereich enthält Hamster-PAPs. Oberhalb des Programmbereichs sind für das Hauptprogramm (main) sowie definierte Unterprogramme Tabs angebracht. Durch Anklicken eines Tabs können Sie in den entsprechenden Bereich wechseln.

13.4.3 Elemente

Start/Stop-Elemente: Das Hauptprogramm und alle Unterprogramme müssen jeweils genau ein Start- und einen Stop-Element besitzen. Beim Start-Element startet das (Unter-)Programm, beim Stop-Element endet es. Bei der Erzeugung eines Start/Stop-Elementes erscheint nach dem Draggen ein Menü, über das man auswählen kann, ob es sich bei dem neu erzeugten Element um ein Start- oder ein Stop-Element handeln soll.

Operation-Elemente: Bei der Erzeugung eines Operation-Elementes erscheint nach dem Draggen ein Menü, über das man auswählen kann, welchen Hamster-Befehl die neue Operation repräsentieren soll (vor, linksUm, gib, nimm).

Verzweigung-Elemente: Bei der Erzeugung eines Verzweigung-Elementes erscheint nach dem Draggen ein Menü, über das man auswählen kann, welchen Hamster-Testbefehl die neue Operation repräsentieren soll (vornFrei, maulLeer, kornDa).

Unterprogramm-Elemente: Bei der Erzeugung eines Unterprogramm-Elementes erscheint nach dem Draggen ein Menü, über das man auswählen kann, ob ein bereits definiertes Unterprogramm aufgerufen werden oder ein neues Unterprogramm definiert werden soll. Im zweiten Fall muss man in einem kleinen erscheinenden Fenster den Namen des neuen Unterprogramms eintippen und anschließend die RETURN-Taste drücken. Im Programmbereich wird dann ein neuer Tab erzeugt und automatisch aktiviert.

Kommentar-Elemente: Nach der Erzeugung eines Kommentar-Elementes kann man nach dem Draggen den Kommentartext eingeben und die Eingabe durch Drücken der RETURN-Taste abschließen.

Elemente lassen sich bei deaktiviertem Pfeil-Zeichnen-Modus (siehe Abschnitt 13.4.4) durch Anklicken und Verschieben bei gedrückter Maustaste („Dragging“) im Programmbereich an andere Stellen verschieben. Alle Elemente besitzen ein Kontext-Menü, das man durch Klicken der rechten Maustaste oberhalb des Elementes aktivieren kann. Hierüber ist es möglich, Elemente umzubenennen, zu entfernen oder vom Element ausgehende Verbindungen zu entfernen.

13.4.4 Pfeile

Durch Klicken des Pfeil-Symbols im Auswahl-Menü lässt sich der Pfeil-Zeichnen-Modus an- bzw. wieder ausschalten. Ist der Modus aktiv, lassen sich Verbindungen zwischen den Elementen erzeugen. Während der Modus aktiv ist, können Elemente im Programmbereich nicht verschoben werden.

Bewegt man bei aktiviertem Pfeil-Zeichnen-Modus die Maustaste über ein Element im Programmbereich, erscheinen kleine rote Kreise, die mögliche Ausgangspunkte eines Pfeils andeuten. Klickt man einen solchen Kreis an und zieht die Maus bei gedrückter Maustaste auf ein (anderes) Element („Draggen“), erscheinen dort ebenfalls kleine rote Kreise. Lässt man die Maus dann los, wird ein Pfeil zwischen den beiden Elementen gezeichnet. Es wird versucht, die Pfeile optisch möglichst übersichtlich zu positionieren. Verschiebungen von Pfeilen oder Teilen von Pfeilen sind möglich, in dem erscheinende kleine rote Pfeile entsprechend an andere Stellen gedraggt werden.

Pfeile lassen sich über das Kontext-Menü des ausgehenden Elementes löschen („Verbindung entfernen“). Von jedem Element (außer einem Verzweigung-Element) kann nur ein Pfeil ausgehen. Zeichnet man einen weiteren Pfeil, wird der andere automatisch entfernt.

Aus einem Verzweigung-Element können zwei Pfeile ausgehen. Einer kennzeichnet den Weg, sollte die Auswertung des Verzweigung-Elementes zur Laufzeit den Wert wahr (true, ja) ergeben, der andere den anderen Fall (falsch, false, nein). Zeichnet man einen Pfeil, der von einem Verzweigung-Element ausgeht, wird man gefragt, ob es sich um den ja- oder nein-Pfeil handeln soll.

13.5 Weitere Funktionen

Ausführung: Wie andere Programme auch können Hamster-PAPs gestartet, gestoppt, pausiert und fortgesetzt werden. Auch eine schrittweise Ausführung ist möglich (Funktion „Schritt hinein“). Während der schrittweisen Ausführung werden die gerade aktiven Elemente rot hervorgehoben.

Speichern/Laden: Hamster-PAPs können (bspw. über das Menü „Datei“) gespeichert und später wieder geladen werden. Im Dateibaum im Editorfenster werden entsprechende Dateien durch das Symbol „gelbes Kästchen mit blauem Punkt“ dargestellt.

Drucken: Hamster-Automaten können (bspw. über das Menü „Datei) gedruckt werden.

Generieren: Über das Menü-Item „Generieren“ im Menü „Datei“ lässt sich aus einem Hamster-PAP ein gleichbedeutendes imperatives Java-Hamster-Programm generieren. Die Funktion sollte allerdings nur bei fehlerfreien Hamster-PAPs ausgeführt werden. Ist das Hamster-PAP fehlerhaft, ist nicht definiert, was das generierte Programm tut.

14 JavaScript

JavaScript (kurz JS) ist eine Skriptsprache, die ursprünglich für dynamisches HTML in Webbrowsern entwickelt wurde, um Benutzerinteraktionen auszuwerten, Inhalte zu verändern, nachzuladen oder zu generieren und so die Möglichkeiten von HTML und CSS zu erweitern. Heute findet JavaScript auch außerhalb von Browsern Anwendung, so etwa auf Servern und in Microcontrollern.

Der als ECMAScript (ECMA 262) standardisierte Sprachkern von JavaScript beschreibt eine dynamisch typisierte, objektorientierte, aber klassenlose Skriptsprache. Sie wird allen objektorientierten Programmierparadigmen unter anderem auf der Basis von Prototypen gerecht. In JavaScript lässt sich objektorientiert und sowohl prozedural als auch funktional programmieren (aus Wikipedia).

14.1 Die Programmiersprache JavaScript

Im Internet finden Sie viele Informationen rund um die Programmiersprache JavaScript. Schauen Sie einfach mal unter folgenden Links nach:

- https://de.wikipedia.org/wiki/JavaScript (Wikipedia)
- http://openbook.rheinwerk-verlag.de/javascript_ajax/ (Online-Buch)

14.2 JavaScript-Hamster-Programme

Um mit dem Hamster-Simulator JavaScript-Programme entwickeln und ausführen zu können, muss die Property `javascript` auf `true` gesetzt sein. Standardmäßig ist das nicht der Fall. Öffnen Sie also die Datei `hamster.properties` und ersetzen Sie dort die Zeile `javascript=false` durch die Zeile `javascript=true`.

Starten Sie dann den Hamster-Simulator. Wenn Sie ein neues Programm erstellen wollen, wählen Sie in der erscheinenden Dialogbox den Eintrag „JavaScript-Programm“. Im Editor-Bereich erscheint dann ein Feld mit dem Inhalt „`if (vornFrei()) vor();`“. Ersetzen Sie diesen Inhalt durch Ihr JavaScript-Programm und speichern Sie die Datei dann ab. JavaScript-Programme brauchen nicht compiliert zu werden. Nach dem Abspeichern haben Sie direkt die Möglichkeit, das Programm auszuführen. Syntaxfehler werden erst zur Laufzeit angezeigt.

Im Hamster-Simulator lassen sich sowohl imperative als auch objektorientierte Python-Programme entwickeln. Auch gemischte Programme sind möglich. Der (imperative) Hamster kennt (wie üblich) die Befehle

```
vor();
linksUm();
gib();
nimm();
vornFrei() (-> boolean)
```

```
kornDa() (-> boolean)
maulLeer() (-> boolean)
```

Eine vordefinierte Klasse namens `Hamster` kann für objektorientierte JavaScript-Programme genutzt werden. Die Klasse stellt folgende Methoden zur Verfügung:

```
vor()
linksUm()
gib()
nimm()
vornFrei() (-> boolean)
kornDa() (-> boolean)
maulLeer() (-> boolean)
schreib(String)
liesZeichenkette(String) (-> String)
liesZahl(string) (-> int)
getReihe() (-> int)
getSpalte() (-> int)
getBlickrichtung() (-> int: Hamster.NORD, Hamster.SUED, Hamster.OST,
Hamster.WEST)
getAnzahlKoerner() (-> int)
getStandardHamster() (-> Hamster)
```

Die Klasse definiert 2 Konstruktoren:

```
Hamster(reihe, spalte, blickrichtung, anzahlKoerner)
Hamster(hamster)
```

Leider ist es aus implementierungstechnischen Gründen nicht möglich, Hamster-Objekte um zusätzliche Eigenschaften und Methoden zu erweitern. Weiterhin kann die Klasse Hamster leider auch nicht zur Vererbung über Prototypen genutzt werden.

Aus diesem Grund wurde eine JavaScript-Klasse JSHamster vordefiniert. JSHamster-Objekte können sowohl erweitert werden als auch kann die Klasse zur Vererbung über Prototypen genutzt werden (siehe Beispiel in Abschnitt 14.3.5).

```
function JSHamster(r, s, b, k) {
  this.hamster = new Hamster(r, s, b, k);
  this.vor = function() {
      this.hamster.vor();
  }
  this.linksUm = function() {
      this.hamster.linksUm();
  }
  this.gib = function() {
      this.hamster.gib();
  }
  this.nimm = function() {
      this.hamster.nimm();
  }
```

```
  this.vornFrei = function() {
      return this.hamster.vornFrei();
  }
  this.kornDa = function() {
      return this.hamster.kornDa();
  }
  this.maulLeer = function() {
      return this.hamster.maulLeer();
  }
  this.schreib = function(nachricht) {
      this.hamster.schreib(nachricht);
  }
  this.liesZeichenkette = function(nachricht) {
      return this.hamster.liesZeichenkette(nachricht);
  }
  this.liesZahl = function(nachricht) {
      return this.hamster.liesZahl(nachricht);
  }
  this.getReihe = function() {
      return this.hamster.getReihe();
  }
  this.getSpalte = function() {
      return this.hamster.getSpalte();
  }
  this.getBlickrichtung = function() {
      return this.hamster.getBlickrichtung();
  }
  this.getAnzahlKoerner = function() {
      return this.hamster.getAnzahlKoerner();
  }
}
```

14.3 JavaScript-Beispielprogramme

Die folgenden Unterabschnitte demonstrieren an Beispielen das Schreiben von JavaScript-Hamster-Programmen.

14.3.1 Territorium leeren

Das folgende JavaScript-Programm ist ein einfaches imperatives Beispielprogramm, bei dem der Standard-Hamster ein beliebiges Territorium ohne innere Mauern leert:

```
// Aufgabe:
// der Hamster steht irgendwo in einem beliebigen Territorium
// ohne innere Mauern; er soll alle Körner fressen
```

```
function kehrt() {
    linksUm();
    linksUm();
}

function rechtsUm() {
    kehrt();
    linksUm();
}

function laufeZurueck() {
    while (vornFrei()) {
        vor();
    }
}

function sammle() {
    while (kornDa()) {
        nimm();
    }
}

function laufeZurWand() {
    while (vornFrei()) {
        vor();
    }
}

function laufeInEcke() {
    laufeZurWand();
    linksUm();
    laufeZurWand();
    linksUm();
}

function ernteEineReihe() {
    sammle();
    while (vornFrei()) {
        vor();
        sammle();
    }
}

function ernteEineReiheUndLaufeZurueck() {
    ernteEineReihe();
    kehrt();
    laufeZurueck();
}

laufeInEcke();
ernteEineReiheUndLaufeZurueck();
rechtsUm();
while (vornFrei()) {
    vor();
    rechtsUm();
    ernteEineReiheUndLaufeZurueck();
    rechtsUm();
```

```
}
```

14.3.2 Territorium leeren 2

Auch das folgende JavaScript-Programm ist ein einfaches imperatives Beispielprogramm, bei dem der Standard-Hamster ein beliebiges Territorium ohne innere Mauern leert:

```
// Aufgabe:
// der Hamster steht irgendwo in einem beliebigen Territorium
// ohne innere Mauern; er soll alle Körner fressen

// drehe dich um 180 Grad
function kehrt() {
    linksUm();
    linksUm();
}

// drehe dich um 90 Grad nach rechts
function rechtsUm() {
    kehrt();
    linksUm();
}

// der Hamster sammelt alle Koerner eines Feldes ein
function sammle() {
    while (kornDa()) {
        nimm();
    }
}

// Ueberpruefung, ob sich links vom Hamster
// eine Mauer befindet
function linksFrei() {
    linksUm();
    if (vornFrei()) {
        rechtsUm();
        return true;
    } else {
        rechtsUm();
        return false;
    }
}

// Ueberpruefung, ob sich rechts vom Hamster eine
// Mauer befindet
function rechtsFrei() {
    rechtsUm();
    if (vornFrei()) {
        linksUm();
        return true;
    } else {
        linksUm();
        return false;
    }
}
```

```
// der Hamster laeuft bis zur naechsten Wand
function laufeZurWand() {
    while (vornFrei()) {
        vor();
    }
}

// der Hamster laeuft in eine Ecke
function laufeInEcke() {
    laufeZurWand();
    linksUm();
    laufeZurWand();
    linksUm();
}

// der Hamster soll sich in die naechste Reihe in noerdlicher
// Richtung begeben; vom Hamster aus gesehen, liegt diese Reihe
// links von ihm
function begibDichLinksUmInNaechsteReihe() {
    linksUm();
    vor();
    linksUm();
}

// der Hamster soll sich in die naechste Reihe in noerdlicher
// Richtung begeben; vom Hamster aus gesehen, liegt diese Reihe
// rechts von ihm
function begibDichRechtsUmInNaechsteReihe() {
    rechtsUm();
    vor();
    rechtsUm();
}

// Ueberpruefung, ob in noerdlicher Richtung (vom Hamster aus
// gesehen links) eine weitere nicht mit Mauern besetzte
// Reihe existiert
function weitereReiheLinksVomHamsterExistiert() {
    return linksFrei();
}

// Ueberpruefung, ob in noerdlicher Richtung (vom Hamster aus
// gesehen rechts) eine weitere nicht mit Mauern besetzte
// Reihe existiert
function weitereReiheRechtsVomHamsterExistiert() {
    return rechtsFrei();
}

// der Hamster soll alle Koerner in einer Reihe einsammeln
function ernteEineReihe() {
    sammle();
    while (vornFrei()) {
        vor();
        sammle();
    }
}

// der Hamster soll alle Koerner in einer Reihe einsammeln;
```

```
// er laeuft dabei von Westen nach Osten
function ernteEineReiheNachOsten() {
    ernteEineReihe();
}

// der Hamster soll alle Koerner in einer Reihe einsammeln;
// er laeuft dabei von Osten nach Westen
function ernteEineReiheNachWesten() {
    ernteEineReihe();
}

// der Hamster soll einzelne Koernerreihen abgrasen,
// so lange noch weitere Reihen existieren;
// er unterscheidet dabei, ob er die Reihen von
// Osten oder von Westen aus abgrast
laufeInEcke();
ernteEineReiheNachOsten();
while (weitereReiheLinksVomHamsterExistiert()) {
    begibDichLinksUmInNaechsteReihe();
    ernteEineReiheNachWesten();
    if (weitereReiheRechtsVomHamsterExistiert()) {
        begibDichRechtsUmInNaechsteReihe();
        ernteEineReiheNachOsten();
    } else {
        kehrt();
    }
}
```

14.3.3 Berg erklimmen

Im folgenden imperativen JavaScript-Programm erklimmt der Standard-Hamster einen Berg:

```
// Aufgabe:
// der Hamster soll den Gipfel eines vor ihm stehenden
// Berges erklimmen

// der Hamster soll zum Berg laufen
function laufeZumBerg() {
    while (vornFrei()) {
        vor();
    }
}

// der Hamster soll den Berg erklimmen
function erklimmeDenBerg() {
    while (!gipfelErreicht()) {
        erklimmeEineStufe();
    }
}

// der Hamster soll eine Stufe erklimmen
function erklimmeEineStufe() {
    linksUm();  // nun schaut der Hamster nach oben
```

```
    vor();      // der Hamster erklimmt die Mauer
    rechtsUm(); // der Hamster wendet sich wieder dem Berg zu
    vor();      // der Hamster springt auf den Vorsprung
}

// der Hamster dreht sich nach rechts um
function rechtsUm() {
    linksUm();
    linksUm();
    linksUm();
}

// hat der Hamster den Gipfel erreicht?
function gipfelErreicht() {
    return vornFrei();
}

// der Hamster soll zunaechst bis zum Berg laufen
// und dann den Berg erklimmen
laufeZumBerg();
erklimmeDenBerg();
```

14.3.4 Wettlauf

Im folgenden objektorientierten JavaScript-Programm liefern sich 3 Hamster einen Wettlauf bis zur nächsten Mauer:

```
paul = Hamster.getStandardHamster();
willi = new Hamster(paul);
maria = new Hamster(paul.getReihe()+1, paul.getSpalte(), Hamster.OST, 0);
while (paul.vornFrei() && willi.vornFrei() && maria.vornFrei()) {
    paul.vor();
    willi.vor();
    maria.vor();
}
paul.schreib("Fertig!");
```

14.3.5 Objektorientiertes Territorium leeren

Im folgenden JavaScript-Programm werden objektorientierte Konzepte genutzt, um einen Hamster das Territorium leeren zu lassen:

```
// Ein neu erzeugter Hamster grast das Territorium ab und
// zaehlt dabei die gesammelten Koerner

function AbgrasHamster(r, s, b, k) {
    JSHamster.call(this, r, s, b, k);

    this.gesammelt = 0;

    this.kehrt = function() {
        this.linksUm();
        this.linksUm();
```

```
    }

    this.rechtsUm = function() {
        this.kehrt();
        this.linksUm();
    }

    this.laufeZurueck = function() {
        while (this.vornFrei()) {
            this.vor();
        }
    }

    this.sammle = function() {
        while (this.kornDa()) {
            this.nimm();
            this.gesammelt += 1;
        }
    }

    this.laufeZurWand = function() {
        while (this.vornFrei()) {
            this.vor();
        }
    }

    this.laufeInEcke = function() {
        this.laufeZurWand();
        this.linksUm();
        this.laufeZurWand();
        this.linksUm();
    }

    this.ernteEineReihe = function() {
        this.sammle();
        while (this.vornFrei()) {
            this.vor();
            this.sammle();
        }
    }

    this.ernteEineReiheUndLaufeZurueck = function() {
        this.ernteEineReihe();
        this.kehrt();
        this.laufeZurueck();
    }

    this.gesammelteKoerner = function() {
        return this.gesammelt;
    }
}
AbgrasHamster.prototype = Object.create(JSHamster.prototype);

// Hauptprogramm

paul = new AbgrasHamster(2, 3, Hamster.WEST, 0);
paul.laufeInEcke();
paul.ernteEineReiheUndLaufeZurueck();
```

```
paul.rechtsUm();
while (paul.vornFrei()) {
    paul.vor();
    paul.rechtsUm();
    paul.ernteEineReiheUndLaufeZurueck();
    paul.rechtsUm();
}
paul.schreib("Gesammelte Koerner = " + paul.gesammelteKoerner());
```

15 Noch Fragen?

Eigentlich müsste dieses Benutzungshandbuch alle Fragen im Zusammenhang mit dem Hamster-Simulator klären. Wenn Sie trotzdem noch weitere Fragen oder Probleme haben, schreiben Sie mir einfach eine Email: boles@informatik.uni-oldenburg.de