

Benutzungshandbuch

Hamster-Simulator-Light

Version 2.0 (24.08.2010)

Dietrich Boles
Universität Oldenburg

Inhaltsverzeichnis

1	Einleitung	10
1.1	Das Hamster-Modell.....	10
1.2	Der Hamster-Simulator.....	10
1.3	Voraussetzungen.....	11
1.4	Anmerkungen	11
1.5	Änderungen von Version 2.0 gegenüber Version 1.0	11
1.6	Aufbau des Benutzerhandbuch	12
2	Installation.....	13
2.1	Voraussetzungen.....	13
2.2	Download, Installation und Start.....	13
3	Das Java-Hamster-Modell.....	14
3.1	Landschaft.....	14
3.2	Hamster.....	15
3.3	Befehle	15
4	Imperative Programmierkonzepte der Programmiersprache Java	17
4.1	Grundlagen.....	17
4.2	Anweisungen und Programme	17
4.2.1	Hamster-Befehle	18
4.2.2	Anweisungen	18

4.2.3	Programme	19
4.2.4	Kommentare	19
4.2.5	Beispielprogramm	19
4.3	Prozeduren	20
4.3.1	Prozedurdefinition	20
4.3.2	Prozeduraufruf	21
4.3.3	Beispielprogramm	21
4.4	Auswahanweisungen	22
4.4.1	Testbefehle	22
4.4.2	Boolesche Operatoren und Ausdrücke	23
4.4.3	Blockanweisung	23
4.4.4	Bedingte Anweisung	23
4.4.5	Alternativanweisung	24
4.4.6	Beispielprogramm	24
4.5	Wiederholungsanweisungen	25
4.5.1	while-Anweisung	25
4.5.2	do-Anweisung	25
4.5.3	Beispielprogramm	26
4.6	Boolesche Funktionen	27
4.6.1	Boolesche return-Anweisung	27
4.6.2	Definition boolescher Funktionen	27
4.6.3	Aufruf boolescher Funktionen	27

4.6.4	Beispielprogramm	28
4.7	Variablen und Ausdrücke	28
4.7.1	Datentypen	29
4.7.2	Variablen.....	29
4.7.3	Ausdrücke	29
4.7.4	Zuweisung	30
4.7.5	Vergleichsausdrücke.....	31
4.7.6	Gültigkeitsbereiche von Variablen.....	31
4.7.7	Lebensdauer von Variablen	31
4.7.8	Beispielprogramm	32
4.8	int-Funktionen.....	32
4.8.1	Funktionsdefinition	32
4.8.2	Funktionsaufruf	33
4.8.3	Verallgemeinerung des Funktionskonzeptes	33
4.8.4	Beispielprogramm	33
4.9	Funktionsparameter.....	34
4.9.1	Formale Parameter	35
4.9.2	Aktuelle Parameter	35
4.9.3	Parameterübergabe	35
4.9.4	Überladen von Funktionen	36
4.9.5	Beispielprogramm	36
4.10	Rekursion	37

5	Ihr erstes Hamster-Programm.....	38
5.1	Ausprobieren der Hamster-Befehle	39
5.2	Gestaltung eines Hamster-Territoriums.....	39
5.3	Eingeben eines Hamster-Programms.....	42
5.4	Compilieren eines Hamster-Programms.....	43
5.5	Ausführen eines Hamster-Programms	44
5.6	Debuggen eines Hamster-Programms	45
5.7	Zusammenfassung	46
6	Bedienung des Hamster-Simulators	47
6.1	Grundfunktionen	48
6.1.1	Anklicken	48
6.1.2	Tooltips	48
6.1.3	Button	48
6.1.4	Menü.....	49
6.1.5	Toolbar.....	50
6.1.6	Popup-Menü	50
6.1.7	Eingabefeld.....	50
6.1.8	Dialogbox.....	50
6.1.9	Dateiauswahl-Dialogbox	51
6.1.10	Elementbaum	52
6.1.11	Split-Pane	53
6.2	Verwalten und Editieren von Hamster-Programmen	53

6.2.1	Schreiben eines neuen Hamster-Programms	55
6.2.2	Ändern des aktuellen Hamster-Programms	55
6.2.3	Löschen des aktuellen Hamster-Programms	55
6.2.4	Abspeichern des aktuellen Hamster-Programms	55
6.2.5	Öffnen eines einmal abgespeicherten Hamster-Programms	55
6.2.6	Drucken eines Hamster-Programms	56
6.2.7	Editier-Funktionen	56
6.3	Compilieren von Hamster-Programmen	57
6.3.1	Compilieren	57
6.3.2	Beseitigen von Fehlern	57
6.4	Verwalten und Gestalten von Hamster-Territorien	58
6.4.1	Verändern der Größe des Hamster-Territoriums	59
6.4.2	Umplatzieren des Hamsters im Hamster-Territorium	59
6.4.3	Setzen der Blickrichtung des Hamsters	59
6.4.4	Abfragen und Festlegen der Körneranzahl im Maul des Hamsters	60
6.4.5	Platzieren von Körnern auf Kacheln des Hamster-Territorium	60
6.4.6	Platzieren von Mauern auf Kacheln des Hamster-Territorium	60
6.4.7	Löschen von Kacheln des Hamster-Territorium	61
6.4.8	Abspeichern eines Hamster-Territoriums	61
6.4.9	Wiederherstellen eines abgespeicherten Hamster-Territoriums	61
6.5	Interaktives Ausführen von Hamster-Befehlen	62
6.5.1	Befehlsfenster	62

6.5.2	Parameter	63
6.5.3	Rückgabewerte von Funktionen.....	63
6.5.4	Befehls-Popup-Menü	63
6.6	Ausführen von Hamster-Programmen	63
6.6.1	Starten eines Hamster-Programms.....	64
6.6.2	Stoppen eines Hamster-Programms.....	64
6.6.3	Pausieren eines Hamster-Programms	64
6.6.4	Während der Ausführung eines Hamster-Programms	64
6.6.5	Einstellen der Geschwindigkeit	64
6.6.6	Wiederherstellen eines Hamster-Territoriums.....	65
6.7	Debuggen von Hamster-Programmen.....	65
6.7.1	Beobachten der Programmausführung	66
6.7.2	Schrittweise Programmausführung	67
6.7.3	Breakpoints	67
6.7.4	Debugger-Fenster	68
6.8	Konsole	68
7	Beispielprogramme	71
7.1	Feld abgrasen	71
7.2	Berg erklimmen	72
7.3	Berg erklimmen (objektorientierte Variante)	73
8	Literatur zum Erlernen der Programmierung.....	75

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) Programmierung auf spielerische Art und Weise erlernt werden können. Programmierer entwickeln so genannte „Hamster-Programme“, mit denen sie einen virtuellen 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.

Das Hamster-Modell wird ausführlich in dem Buch „Programmieren spielend gelernt mit dem Java-Hamster-Modell“, erschienen im Vieweg-Teubner-Verlag vorgestellt. Viele weitere Informationen finden Sie auf der Hamster-Website im WWW unter www.java-hamster-modell.de.

1.2 *Der Hamster-Simulator*

Beim Hamster-Modell steht nicht so sehr das "Learning-by-Listening" bzw. „Learning-by-Reading“ im Vordergrund, sondern vielmehr das „Learning-by-Doing“, also das praktische Üben. Genau das ist mit dem Hamster-Simulator möglich. Dieser stellt eine Reihe von Werkzeugen zum Erstellen und Ausführen von Hamster-Programmen zur Verfügung: einen Editor zum Eingeben und Verwalten von Hamster-Programmen, einen Compiler zum Übersetzen von Hamster-Programmen, einen Territoriumsgestalter zum Gestalten und Verwalten von Hamster-Territorien, einen Interpreter zum Ausführen von Hamster-Programmen und einen Debugger zum Testen von Hamster-Programmen. Der Hamster-Simulator ist einfach zu bedienen, wurde aber funktional und bedienungsmäßig bewusst an professionelle

Entwicklungsumgebungen für Java (z.B. Eclipse) angelehnt, um einen späteren Umstieg auf diese zu erleichtern.

Der vorliegende Simulator wird als Light-Version bezeichnet, weil es noch einen weiteren Hamster-Simulator gibt, der noch viele weitere Funktionen anbietet (die jedoch für wirkliche Programmieranfänger nicht unbedingt notwendig sind). Diesen finden Sie auf der Hamster-Website www.java-hamster-modell.de.

1.3 Voraussetzungen

Zielgruppe des Hamster-Modells sind Schüler oder Studierende ohne Programmiererfahrung, die die Grundlagen der (imperativen) Programmierung erlernen und dabei ein wenig Spaß haben wollen. Kenntnisse im Umgang mit Computern sind wünschenswert. Der Hamster-Simulator ist dabei kein Lehrbuch sondern ein Werkzeug, das das Erlernen der Programmierung unterstützt. Auf gute Lehrbücher zum Erlernen der Programmierung wird in Kapitel 7 (Literatur zum Programmieren lernen) hingewiesen.

1.4 Anmerkungen

Der Hamster-Simulator-Light wurde mit dem Tool „Solist“ erstellt. Solist ist eine spezielle Entwicklungsumgebung für Miniprogrammierwelt-Simulatoren. Solist ist ein Werkzeug der Werkzeugfamilie des „Programmier-Theaters“, eine Menge von Werkzeugen zum Erlernen der Programmierung. Metapher aller dieser Werkzeuge ist die Theaterwelt. Schauspieler (im Falle von Solist „Solisten“) agieren auf einer Bühne, auf der zusätzlich Requisiten platziert werden können. Eine Aufführung entspricht der Ausführung eines Programms.

Im Falle des Hamster-Simulators ist die Bühne das Hamster-Territorium, auf dem der Hamster als Solist (es gibt nur einen Hamster) agiert. Requisiten sind Mauern und Körner. Wenn Ihnen also beim Umgang mit dem Hamster-Simulator der Begriff „Solist“ begegnet, kennen Sie nun hiermit den Hintergrund dieser Namenswahl.

Mehr Informationen zu Solist finden Sie im WWW unter www.programmierkurs-java.de/solist.

1.5 Änderungen von Version 2.0 gegenüber Version 1.0

Wesentliche Änderung gegenüber Version 1.0 des Hamster-Simulators-Light ist, dass es nun im Programm-Menü bzw. in der Toolbar spezielle Menü-Items zum Neu-Erstellen, Laden und Speichern von Hamster-Programmen gibt.

1.6 *Aufbau des Benutzerhandbuch*

Das Benutzungshandbuch ist in 8 Kapitel gegliedert. Nach dieser Einleitung wird in Kapitel 2 die Installation des Hamster-Simulators beschrieben. Kapitel 3 stellt ausführlich das Hamster-Modell vor. Eine Übersicht über die Programmierkonzepte der imperativen Programmierung mit dem Hamster-Modell bzw. Java gibt Kapitel 4. Wie Sie Ihr erstes Hamster-Programm zum Laufen bringen, erfahren Sie kurz und knapp in Kapitel 5. Dieses enthält eine knappe Einführung in die Elemente und die Bedienung des Hamster-Simulators. Sehr viel ausführlicher geht dann Kapitel 6 auf die einzelnen Elemente und die Bedienung des Hamster-Simulators ein. Kapitel 7 demonstriert an einigen Beispielprogrammen die Programmierung mit dem Java-Hamster. Kapitel 8 enthält letztendlich Hinweise zu guter Begleitliteratur zum Erlernen der Programmierung mit Java.

2 Installation

2.1 Voraussetzungen

Voraussetzung zum Starten des Hamster-Simulators ist ein Java Development Kit SE (JDK) der Version 6 oder höher. Ein Java Runtime Environment SE (JRE) reicht nicht aus. Das jeweils aktuelle JDK kann über die Website <http://java.sun.com/javase/downloads/index.jsp> bezogen werden und muss anschließend installiert werden.

2.2 Download, Installation und Start

Der Hamster-Simulator-Light kann von der Website <http://www.programmierkurs-java.de/solist> oder www.java-hamster-modell.de kostenlos herunter geladen werden. Er befindet sich in einer Datei namens *hamstersimulator-light-2.0.zip*. Diese muss zunächst entpackt werden. Es entsteht ein Ordner namens *hamstersimulator-light-2.0* (der so genannte *Simulator-Ordner*), in dem sich eine Datei *simulator.jar* befindet. Durch Doppelklick auf diese Datei wird der Hamster-Simulator gestartet. Alternativ lässt er sich auch durch Eingabe des Befehls `java -jar simulator.jar` in einer Console starten.

Beim ersten Start sucht der Hamster-Simulator nach der JDK-Installation auf Ihrem Computer. Sollte diese nicht gefunden werden, werden Sie aufgefordert, den entsprechenden Pfad einzugeben. Der Pfad wird anschließend in einer Datei namens *solist.properties* im Simulator-Ordner gespeichert, wo er später wieder geändert werden kann, wenn Sie bspw. eine aktuellere JDK-Version auf Ihrem Rechner installieren sollten. In der Property-Datei können Sie weiterhin die Sprache angeben, mit der der Hamster-Simulator arbeitet. In der aktuellen Version wird allerdings nur deutsch unterstützt.

3 Das Java-Hamster-Modell

Computer können heutzutage zum Lösen vielfältiger Aufgaben genutzt werden. Die Arbeitsanleitungen zum Bearbeiten der Aufgaben werden ihnen in Form von Programmen mitgeteilt. Diese Programme, die von Programmierern entwickelt werden, bestehen aus einer Menge von Befehlen bzw. Anweisungen, die der Computer ausführen kann. Die Entwicklung solcher Programme bezeichnet man als Programmierung.

Das Hamster-Modell ist ein spezielles didaktisches Modell zum Erlernen der Programmierung. Im Hamster-Modell nimmt ein virtueller Hamster die Rolle des Computers ein. Diesem Hamster können ähnlich wie einem Computer Befehle erteilt werden, die dieser ausführt.

Ihnen als Programmierer werden bestimmte Aufgaben gestellt, die sie durch die Steuerung des Hamsters zu lösen haben. Derartige Aufgaben werden im Folgenden Hamster-Aufgaben genannt. Zu diesen Aufgaben müssen Sie in der Hamster-Sprache - eine Programmiersprache, die fast vollständig der Programmiersprache Java entspricht - Programme - Hamster-Programme genannt -- entwickeln, die die Aufgaben korrekt und vollständig lösen. Die Aufgaben werden dabei nach und nach komplexer. Zum Lösen der Aufgaben müssen bestimmte Programmierkonzepte eingesetzt werden, die im Hamster-Modell inkrementell eingeführt werden.

Die Grundidee des Hamster-Modells ist ausgesprochen einfach: Sie als Programmierer müssen einen (virtuellen) Hamster durch eine (virtuelle) Landschaft steuern und ihn gegebene Aufgaben lösen lassen.

3.1 *Landschaft*

Die Welt, in der der Hamster lebt, wird durch eine gekachelte Ebene repräsentiert. Abbildung 3.1 zeigt eine typische Hamsterlandschaft - auch Hamster-Territorium genannt. Die Größe der Landschaft, d.h. die Anzahl der Kacheln, ist dabei nicht explizit vorgegeben. Die Landschaft ist beliebig aber nie unendlich groß.

Auf einzelnen Kacheln können ein oder mehrere Körner liegen. Kacheln, auf denen sich Körner befinden, sind in den Landschaftsskizzen durch ein spezielles Korn-Symbol gekennzeichnet.

Auf den Kacheln des Hamster-Territoriums können weiterhin auch Mauern stehen, was bedeutet, dass diese Kacheln blockiert sind. Der Hamster kann sie nicht betreten. Es ist nicht möglich, dass sich auf einer Kachel sowohl eine Mauer als auch Körner befinden. Der Hamster kann das Territorium nicht verlassen. Stellen Sie sich also vor, dass das Territorium immer vollständig von Mauern umgeben ist.

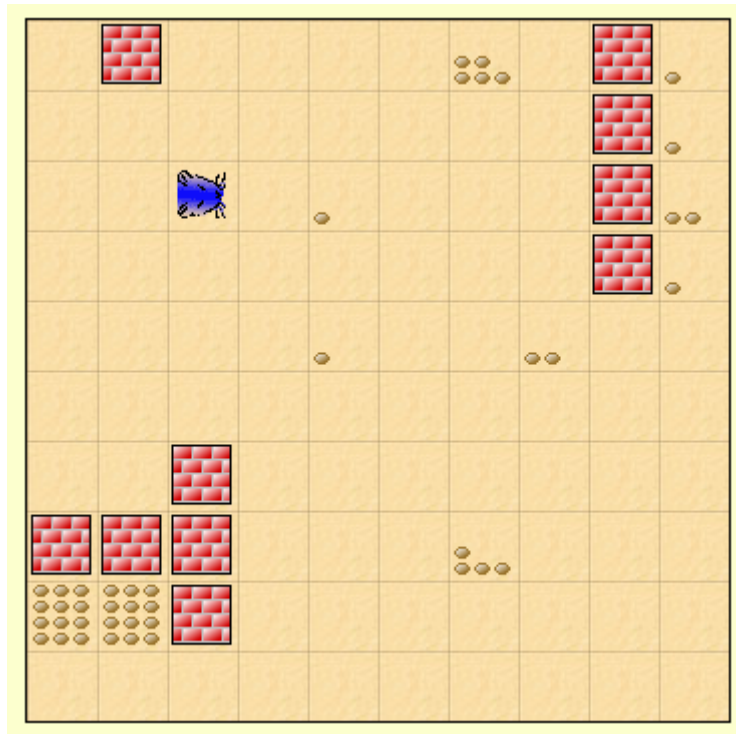


Abb. 3.1: Komponenten des Hamster-Modells

3.2 *Hamster*

Im Hamster-Modell existiert immer genau ein Hamster. Der Hamster steht dabei auf einer der Kacheln des Hamster-Territoriums. Diese Kachel darf nicht durch eine Mauer blockiert sein, sie kann jedoch Körner enthalten. Der Hamster kann in vier unterschiedlichen Blickrichtungen (Nord, Süd, West, Ost) auf den Kacheln stehen. Je nach Blickrichtung wird der Hamster durch unterschiedliche Symbole repräsentiert. In Abbildung 3.1 schaut der Hamster nach Osten.

Körner können sich nicht nur auf einzelnen Kacheln, sondern auch im Maul des Hamsters befinden.

3.3 *Befehle*

Der Hamster kennt vier Befehle und drei Testbefehle, durch deren Aufruf ein Programmierer den Hamster durch ein gegebenes Hamster-Territorium steuern kann. Sie können sich den Hamster quasi als einen virtuellen Prozessor vorstellen, der im Gegensatz zu realen Computer-Prozessoren (zunächst) keine arithmetischen und logischen Operationen ausführen kann, sondern in der Lage ist, mit einem kleinen Grundvorrat an Befehlen ein Hamster-Territorium zu "erkunden". Diese

Befehle werden im Folgenden aufgelistet und im kommenden Kapitel genauer vorgestellt.

- `void vor()`: Der Hamster hüpft eine Kachel in seiner aktuellen Blickrichtung nach vorne.
- `void linksUm()`: Der Hamster dreht sich auf der Kachel, auf der er gerade steht, um 90 Grad entgegen dem Uhrzeigersinn.
- `void nimm()`: Der Hamster frisst von der Kachel, auf der er sich gerade befindet, genau ein Korn, d.h. anschließend hat der Hamster ein Korn mehr im Maul und auf der Kachel liegt ein Korn weniger als vorher.
- `void gib()`: Der Hamster legt auf der Kachel, auf der er sich gerade befindet, genau ein Korn aus seinem Maul ab, d.h. er hat anschließend ein Korn weniger im Maul, und auf der Kachel liegt ein Korn mehr als vorher.
- `boolean vornFrei()`: Liefert den Wert *true*, falls sich auf der Kachel in Blickrichtung vor dem Hamster keine Mauer befindet. Ist die Kachel durch eine Mauer blockiert, dann wird der Wert *false* geliefert.
- `boolean maulLeer()`: Liefert den Wert *false*, falls der Hamster ein oder mehrere Körner im Maul hat. Befinden sich keine Körner im Maul des Hamsters, dann wird der Wert *true* geliefert.
- `boolean kornDa()`: Liefert den Wert *true*, falls auf der Kachel, auf der der Hamster gerade steht, ein oder mehrere Körner liegen. Befindet sich kein Korn auf der Kachel, dann wird der Wert *false* geliefert.

4 Imperative Programmierkonzepte der Programmiersprache Java

In diesem Kapitel werden die Befehle, die der Hamster kennt, genauer erläutert. Außerdem enthält dieses Kapitel eine Zusammenfassung der wichtigsten Programmierkonzepte der imperativen Programmierung mit Java. Im Detail und sehr ausführlich werden diese Konzepte im Buch „Programmieren spielend gelernt mit dem Java-Hamster-Modell“, erschienen im Vieweg-Teubner-Verlag vorgestellt. Wenn Sie noch überhaupt keine Kenntnisse der Programmierung haben, empfehle ich Ihnen, sich Informationen zu den allgemeinen Grundlagen der Programmierung auf der folgenden Website durchzulesen (Leseprobe des Java-Hamster-Buches): <http://www-is.informatik.uni-oldenburg.de/~dibo/hamster/leseprobe/node3.html>.

4.1 Grundlagen

Der Zeichenvorrat (die Lexikalik), den Sie beim Erstellen von Hamster-Programmen verwenden dürfen, entspricht dem 16-Bit-Zeichensatz Unicode.

Die Token einer Sprache, auch lexikalische Einheiten genannt, sind die Wörter, auf denen sie basiert. Wenn Sie Ihr Programm compilieren, teilt der Compiler Ihren Quellcode in Token auf und versucht herauszufinden, welche Anweisungen, Bezeichner und andere Elemente der Quellcode enthält. Token müssen in Java durch Wortzwischenräume voneinander getrennt werden. Zu den Wortzwischenräumen zählen Leerzeichen, Tabulatoren, Zeilenvorschub- und Seitenvorschubzeichen. Diese im Folgenden kurz als Trennzeichen bezeichneten Zeichen haben ansonsten keine Bedeutung.

Bezeichner, die zur Benennung von deklarierten Elementen (wie Prozeduren oder Variablen) verwendet werden, müssen in Java mit einem Buchstaben, einem Unterstrich (`_`) oder einem Dollarzeichen (`$`) beginnen, dem weitere Buchstaben, Unterstriche und Ziffern folgen können. Bezeichner dürfen beliebig lang sein.

In Java wird streng zwischen Groß- und Kleinbuchstaben unterschieden, d.h. dass bspw. die Bezeichner „rechts“ und „Rechts“ unterschiedliche Bezeichner sind.

Schlüsselwörter sind reservierte Wörter einer Programmiersprache. Sie dürfen nicht als Bezeichner verwendet werden.

4.2 Anweisungen und Programme

Programme setzen sich aus einer Menge von Befehlen bzw. Anweisungen zusammen.

4.2.1 Hamster-Befehle

Die Aufgabe eines Hamster-Programmierers besteht darin, den Hamster durch eine Landschaft zu steuern, um dadurch gegebene Hamster-Aufgaben zu lösen. Zur Steuerung des Hamsters müssen ihm Anweisungen in Form von Befehlen gegeben werden. Der Hamster besitzt dabei die Fähigkeit, vier verschiedene Befehle zu verstehen und auszuführen:

- `vor()`; Der Hamster hüpft eine Kachel in seiner aktuellen Blickrichtung nach vorne.
- `linksUm()`; Der Hamster dreht sich auf der Kachel, auf der er gerade steht, um 90 Grad entgegen dem Uhrzeigersinn.
- `nimm()`; Der Hamster frisst von der Kachel, auf der er sich gerade befindet, genau ein Korn, d.h. anschließend hat der Hamster ein Korn mehr im Maul und auf der Kachel liegt ein Korn weniger als vorher.
- `gib()`; Der Hamster legt auf der Kachel, auf der er sich gerade befindet, genau ein Korn aus seinem Maul ab, d.h. er hat anschließend ein Korn weniger im Maul, und auf der Kachel liegt ein Korn mehr als vorher.

Wie Sie vielleicht schon festgestellt haben, können bei den Befehlen `vor`, `nimm` und `gib` Probleme auftreten:

- Der Hamster bekommt den Befehl `vor()`; und die Kachel in Blickrichtung vor ihm ist durch eine Mauer blockiert.
- Der Hamster bekommt den Befehl `nimm()`; und auf der Kachel, auf der er sich gerade befindet, liegt kein einziges Korn.
- Der Hamster bekommt den Befehl `gib()`; und er hat kein einziges Korn im Maul.

Bringen Sie den Hamster in diese für ihn unlösbaren Situationen, dann ist der Hamster derart von Ihnen enttäuscht, dass er im Folgenden nicht mehr bereit ist, weitere Befehle auszuführen. Derartige Fehler werden Laufzeitfehler genannt. Laufzeitfehler können im Allgemeinen nicht schon durch den Compiler entdeckt werden, sondern treten erst während der Ausführung eines Programmes auf. Programme, die zu Laufzeitfehlern führen können, sind nicht korrekt!

4.2.2 Anweisungen

In imperativen Programmiersprachen werden Verarbeitungsvorschriften durch so genannte Anweisungen ausgedrückt. Anweisungen, die nicht weiter zerlegt werden können, werden elementare Anweisungen genannt. In der Hamster-Sprache sind die vier Grundbefehle elementare Anweisungen. Eine Folge von Anweisungen, die nacheinander ausgeführt werden, wird als Anweisungssequenz bezeichnet. Die

einzelnen Anweisungen einer Anweisungssequenz werden in der angegebenen Reihenfolge hintereinander ausgeführt.

4.2.3 Programme

Ein Hamster-Programm setzt sich aus dem Schlüsselwort `void`, gefolgt von `main`, einem runden Klammernpaar und einem geschweiften Klammernpaar, das eine Anweisungssequenz umschließt, zusammen. Beim Aufruf bzw. Start des Programms werden die Anweisungen der Anweisungssequenz innerhalb der geschweiften Klammern hintereinander ausgeführt.

4.2.4 Kommentare

Ziel der Programmierung ist es, Programme zu entwickeln, die gegebene Aufgaben lösen. Neben ihren Eigenschaften, korrekt und vollständig zu sein, sollten sich Programme durch eine weitere Eigenschaft auszeichnen; sie sollten gut verständlich sein. Das bedeutet, die Lösungsidee und die Realisierung sollte auch von anderen Programmierern mühelos verstanden und nachvollzogen werden können, um bspw. das Programm später noch zu erweitern oder in anderen Zusammenhängen wiederverwenden zu können.

Diesem Zweck der Dokumentation eines Programms dienen so genannte Kommentare. Sie haben auf die Steuerung des Hamsters keinerlei Auswirkungen. Alles, was sie bewirken, ist eine bessere Lesbarkeit des Programms. In Java gibt es zwei Typen von Kommentaren: Zeilenkommentare und Bereichskommentare.

Zeilenkommentare beginnen mit zwei Schrägstrichen (`//`) und enden am nächsten Zeilenende. Den Schrägstrichen können beliebige Zeichen folgen.

Bereichskommentare beginnen mit der Zeichenkombination `/*` und enden mit der Zeichenkombination `*/`. Dazwischen können beliebige Zeichen stehen. Bereichskommentare können sich auch über mehrere Zeilen erstrecken.

4.2.5 Beispielprogramm

Das folgende Hamster-Programm bewirkt, dass der Hamster in dem in Abbildung 4.2 skizzierten Territorium zwei Körner frisst:

```
void main() {  
  
    // friss erstes Korn  
    vor();  
    vor();  
    nimm();  
  
    // friss zweites Korn  
    linksUm();  
}
```

```

vor();
vor();
nimm();
}

```

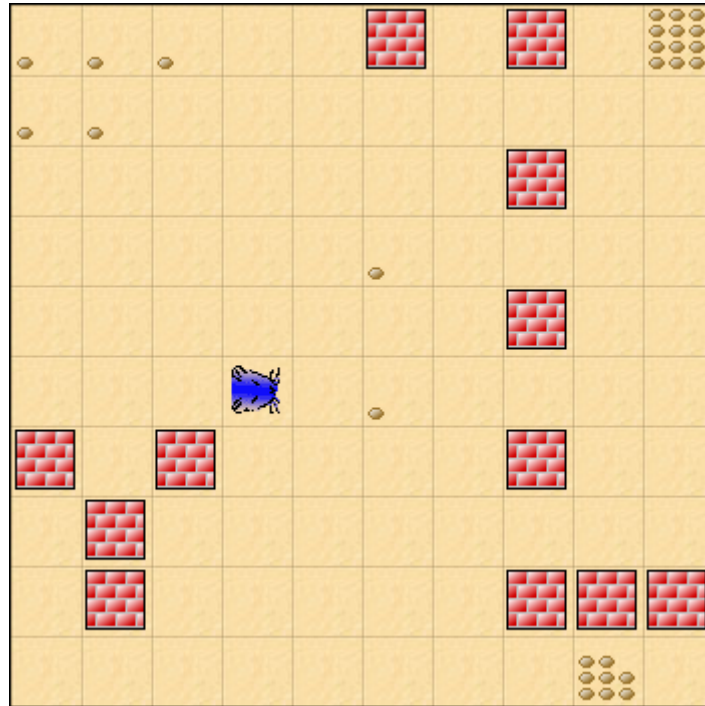


Abb. 4.2: Hamster-Territorium zum Beispielprogramm

4.3 Prozeduren

Prozeduren dienen zur Vereinbarung neuer Befehle. Diesbezüglich sind zwei Aspekte zu betrachten: die Definition von Prozeduren und deren Aufruf, d.h. Ausführung.

4.3.1 Prozedurdefinition

Durch eine Prozedurdefinition wird ein neuer Befehl vereinbart. In der Definition muss zum einen angegeben werden, wie der Befehl heißt (Prozedurname), und zum anderen muss festgelegt werden, was der Hamster tun soll, wenn er den neuen Befehl erhält. Ersteres erfolgt im so genannten Prozedurkopf, letzteres im so genannten Prozedurrumpf.

Im Prozedurkopf muss zunächst das Schlüsselwort `void` angegeben werden. Anschließend folgt ein Bezeichner, der Prozedurname bzw. der Name des neuen Befehls. Nach dem Prozedurnamen folgt ein rundes Klammernpaar, das den

Prozedurkopf beendet. Der Prozedurrumpf beginnt mit einer öffnenden geschweiften Klammer, der eine Anweisungssequenz folgt. Der Prozedurrumpf und damit die Prozedurdefinition endet mit einer schließenden geschweiften Klammer.

Auch das oben eingeführte main-Konstrukt ist im Prinzip eine Prozedur. Sie wird automatisch beim Start eines Programms durch das Laufzeitsystem aufgerufen. Die Definition einer neuen Prozedur darf vor oder nach der Definition der main-Prozedur erfolgen.

4.3.2 Prozeduraufruf

Durch eine Prozedurdefinition wird ein neuer Befehl eingeführt. Ein Aufruf des neuen Befehls wird Prozeduraufruf genannt. Ein Prozeduraufruf entspricht syntaktisch dem Aufruf eines der vier Grundbefehle des Hamsters. Er beginnt mit dem Prozedurnamen. Anschließend folgen eine öffnende und eine schließende runde Klammer und ein Semikolon.

Wird irgendwo in einem Programm eine Prozedur aufgerufen, so werden bei der Ausführung des Programms an dieser Stelle die Anweisung(en) des Prozedurrumpfes ausgeführt. Der Kontrollfluss des Programms verzweigt beim Prozeduraufruf in den Rumpf der Prozedur, führt die dortigen Anweisungen aus und kehrt nach der Abarbeitung der letzten Anweisung des Rumpfes an die Stelle des Prozeduraufrufs zurück. Durch Aufruf der return-Anweisung (`return;`) kann eine Prozedur vorzeitig verlassen werden.

4.3.3 Beispielprogramm

Im folgenden Hamster-Programm wird eine Prozedur `rechtsUm` definiert und aufgerufen, die bewirkt, dass sich der Hamster dreimal nach links dreht, was einer Rechtsdrehung um 90 Grad entspricht. Der Hamster frisst in dem in Abbildung 4.3 skizzierten Territorium zwei Körner:

```
void rechtsUm() { // Prozedurdefinition
    linksUm();
    linksUm();
    linksUm();
}

void main() { // main - Prozedur
    // friss erstes Korn
    rechtsUm(); // Prozeduraufruf
    vor();
    vor();
    nimm();

    // friss zweites Korn
    rechtsUm (); // Prozeduraufruf
    vor();
    vor();
}
```

```
nimm();
}
```

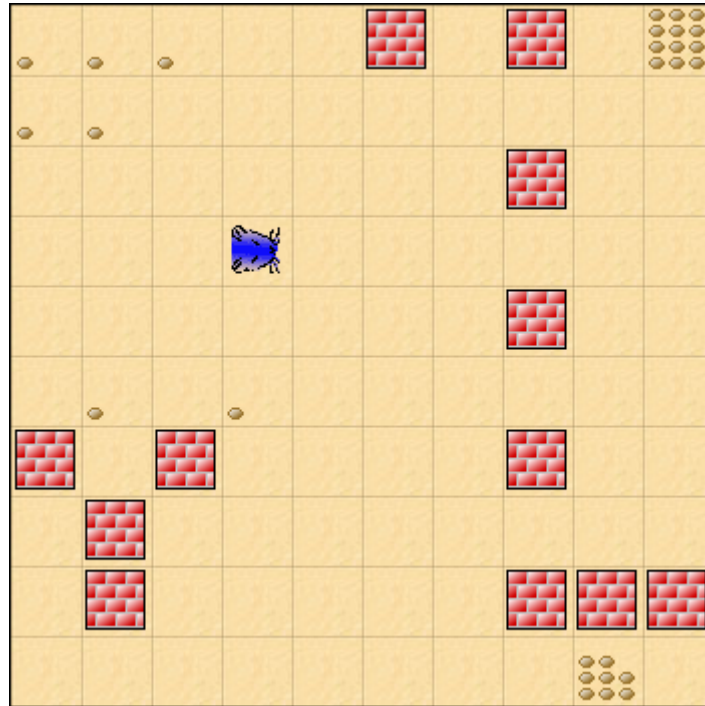


Abb. 4.3: Hamster-Territorium zum Beispielprogramm

4.4 Auswahlanweisungen

Auswahlanweisungen dienen dazu, bestimmte Anweisungen nur unter bestimmten Bedingungen ausführen zu lassen.

4.4.1 Testbefehle

Um Laufzeitfehler zu vermeiden, die ja bspw. dadurch entstehen können, dass Sie dem Hamster den Befehl `vor` geben, obwohl er vor einer Mauer steht, existieren drei so genannte Testbefehle. Testbefehle liefern boolesche Werte, also *wahr* (`true`) oder *falsch* (`false`):

- `vornFrei()`: Liefert den Wert `true`, falls sich auf der Kachel in Blickrichtung vor dem Hamster keine Mauer befindet. Ist die Kachel durch eine Mauer blockiert, dann wird der Wert `false` geliefert.
- `maulLeer()`: Liefert den Wert `false`, falls der Hamster ein oder mehrere Körner im Maul hat. Befinden sich keine Körner im Maul des Hamsters, dann wird der Wert `true` geliefert.

- `kornDa()`: Liefert den Wert `true`, falls auf der Kachel, auf der der Hamster gerade steht, ein oder mehrere Körner liegen. Befindet sich kein Korn auf der Kachel, dann wird der Wert `false` geliefert.

4.4.2 Boolesche Operatoren und Ausdrücke

Die drei Testbefehle stellen einfache boolesche Ausdrücke dar. Ausdrücke sind Verarbeitungsvorschriften, die einen Wert berechnen und liefern. Boolesche Ausdrücke liefern einen booleschen Wert. Durch die Verknüpfung boolescher Ausdrücke mittels boolescher Operatoren lassen sich zusammengesetzte boolesche Ausdrücke bilden. Die Programmiersprache Java stellt drei boolesche Operatoren zur Verfügung:

- Der unäre Operator `!` (Negation) negiert den Wahrheitswert seines Operanden, d.h. er dreht ihn um. Liefert ein boolescher Ausdruck `bA` den Wert `true`, dann liefert der boolesche Ausdruck `!bA` den Wert `false`. Umgekehrt gilt, liefert `bA` den Wert `false`, dann liefert `!bA` den Wert `true`.
- Der binäre Operator `&&` (Konjunktion) konjugiert die Wahrheitswerte seiner beiden Operanden, d.h. er liefert genau dann den Wahrheitswert `true`, wenn beide Operanden den Wert `true` liefern.
- Der binäre Operator `||` (Disjunktion) disjunkt die Wahrheitswerte seiner beiden Operanden, d.h. er liefert genau dann den Wahrheitswert `true`, wenn einer seiner Operanden oder seine beiden Operanden den Wert `true` liefern.

Der Operator `!` hat eine höhere Priorität als der Operator `&&`, der wiederum eine höhere Priorität als der Operator `||` besitzt. Durch Einschließen von booleschen Ausdrücken in runde Klammern können Sie die Abarbeitungsfolge der Operatoren beeinflussen.

4.4.3 Blockanweisung

Mit Hilfe der Blockanweisung lassen sich mehrere Anweisungen zu einer Einheit zusammenfassen. Syntaktisch gesehen handelt es sich bei einer Blockanweisung um eine zusammengesetzte Anweisung. Innerhalb von geschweiften Klammern steht eine andere Anweisung – im Allgemeinen eine Anweisungssequenz.

Beim Ausführen einer Blockanweisung werden die innerhalb der geschweiften Klammern stehenden Anweisungen ausgeführt.

4.4.4 Bedingte Anweisung

Die bedingte Anweisung, die auch `if`-Anweisung genannt wird, ist eine zusammengesetzte Anweisung. Die bedingte Anweisung wird eingeleitet durch das

Schlüsselwort `if`. Anschließend folgen innerhalb eines runden Klammernpaares ein boolescher Ausdruck und danach eine Anweisung. Bei dieser Anweisung, die im Folgenden `true`-Anweisung genannt wird, handelt es sich im Allgemeinen um eine Blockanweisung.

Beim Ausführen einer bedingten Anweisung wird zunächst der boolesche Ausdruck innerhalb der runden Klammern ausgewertet. Falls dieser Ausdruck den Wert `true` liefert, d.h. die Bedingung erfüllt ist, wird die `true`-Anweisung (daher der Name) ausgeführt. Liefert der boolesche Ausdruck den Wert `false`, dann wird die `true`-Anweisung nicht ausgeführt.

4.4.5 Alternativanweisung

Die Alternativanweisung ist eine bedingte Anweisung mit einem angehängten so genannten `else`-Teil. Dieser besteht aus dem Schlüsselwort `else` und einer Anweisung (`false`-Anweisung) – im Allgemeinen eine Blockanweisung. Die Alternativanweisung ist wie die bedingte Anweisung eine Auswahlanweisung.

Wird eine Alternativanweisung ausgeführt, dann wird zunächst der Wert der Bedingung (boolescher Ausdruck) ermittelt. Ist die Bedingung erfüllt, d.h. liefert der boolesche Ausdruck den Wert `true`, dann wird die `true`-Anweisung nicht aber die `false`-Anweisung ausgeführt. Liefert der boolesche Ausdruck den Wert `false`, dann wird die `false`-Anweisung nicht aber die `true`-Anweisung ausgeführt.

4.4.6 Beispielprogramm

Im folgenden Beispielprogramm führt der Hamster die Blockanweisung mit dem `vor`- und dem `linksUm`-Befehl nur aus, wenn sich vor ihm keine Mauer befindet. Anschließend überprüft er, ob sich auf der aktuellen Kachel ein Korn befindet und gleichzeitig die Kachel vor ihm frei ist. Dann und nur dann nimmt er das Korn und springt eine Kachel nach vorne. Im anderen Fall dreht er sich um 90 Grad nach links.

```
void main () {
    if (vornFrei()) { // bedingte Anweisung
        vor();
        linksUm();
    }

    if (kornDa() && vornFrei()) { // Alternativanweisung
        nimm();
        vor();
    } else {
        linksUm();
    }
}
```


4.5 Wiederholungsanweisungen

Wiederholungsanweisungen – auch Schleifenanweisungen genannt – dienen dazu, bestimmte Anweisungen mehrmals ausführen zu lassen, solange eine bestimmte Bedingung erfüllt wird.

4.5.1 while-Anweisung

Die `while`-Anweisung ist eine zusammengesetzte Anweisung. Nach dem Schlüsselwort `while` steht in runden Klammern ein boolescher Ausdruck, die so genannte Schleifenbedingung. Anschließend folgt die Anweisung, die eventuell mehrfach ausgeführt werden soll. Sie wird auch Iterationsanweisung genannt. Hierbei handelt es sich im Allgemeinen um eine Blockanweisung.

Bei der Ausführung einer `while`-Anweisung wird zunächst überprüft, ob die Schleifenbedingung erfüllt ist, d.h. ob der boolesche Ausdruck den Wert `true` liefert. Falls dies nicht der Fall ist, ist die `while`-Anweisung unmittelbar beendet. Falls die Bedingung erfüllt ist, wird die Iterationsanweisung einmal ausgeführt. Anschließend wird die Schleifenbedingung erneut ausgewertet. Falls sie immer noch erfüllt ist, wird die Iterationsanweisung ein weiteres Mal ausgeführt. Dieser Prozess (Überprüfung der Schleifenbedingung und falls diese erfüllt ist, Ausführung der Iterationsanweisung) wiederholt sich so lange, bis (hoffentlich) irgendwann einmal die Bedingung nicht mehr erfüllt ist.

4.5.2 do-Anweisung

Bei Ausführung der `while`-Anweisung kann es vorkommen, dass die Iterationsanweisung kein einziges Mal ausgeführt wird; nämlich genau dann, wenn die Schleifenbedingung direkt beim ersten Test nicht erfüllt ist. Für solche Fälle, bei denen die Iterationsanweisung auf jeden Fall mindestens einmal ausgeführt werden soll, existiert die `do`-Anweisung – auch `do`-Schleife genannt.

Dem Schlüsselwort `do`, von dem die Anweisung ihren Namen hat, folgt die Iterationsanweisung. Hinter der Iterationsanweisung muss das Schlüsselwort `while` stehen. Anschließend folgt in runden Klammern ein boolescher Ausdruck – die Schleifenbedingung. Abgeschlossen wird die `do`-Anweisung durch ein Semikolon.

Bei der Ausführung einer `do`-Anweisung wird zunächst einmal die Iterationsanweisung ausgeführt. Anschließend wird die Schleifenbedingung überprüft. Ist sie nicht erfüllt, d.h. liefert der boolesche Ausdruck den Wert `false`, dann endet die `do`-Anweisung. Ist die Bedingung erfüllt, wird die Iterationsanweisung ein zweites Mal ausgeführt und danach erneut die Schleifenbedingung ausgewertet. Dieser Prozess wiederholt sich so lange, bis irgendwann einmal die Schleifenbedingung nicht mehr erfüllt ist.

4.5.3 Beispielprogramm

Der Hamster steht – wie im Beispiel in Abbildung 4.4 skizziert – vor einem regelmäßigen Berg unbekannter Höhe. Er soll den Gipfel erklimmen und dort anhalten.

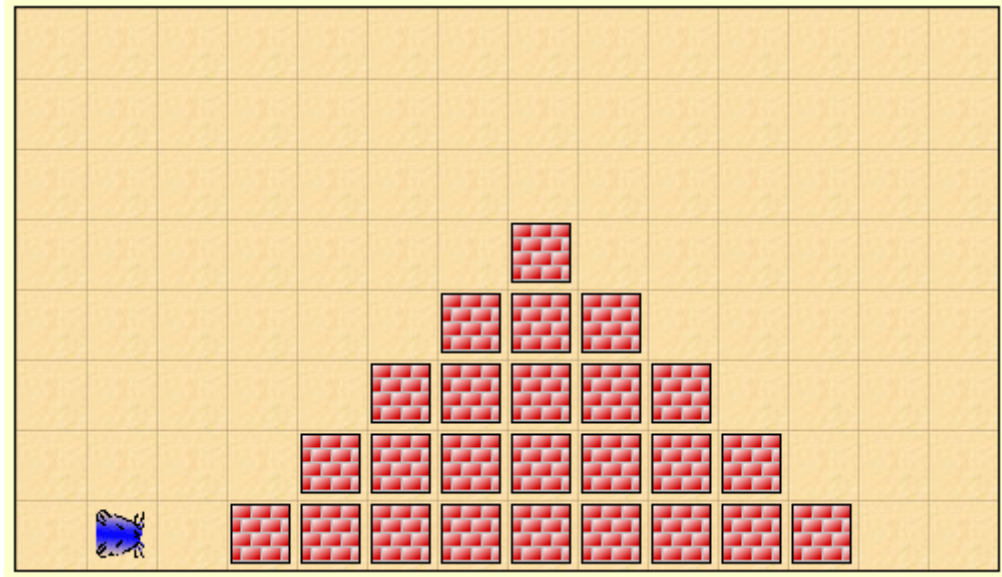


Abb. 2.4: Hamster-Territorien zum Beispielprogramm

```
void main () {
    laufeZumBerg();
    erklimmeGipfel();
}

void laufeZumBerg() {
    while (vornFrei()) { // while - Schleife
        vor();
    }
}

void erklimmeGipfel() {
    do { // do - Schleife
        erklimmeEineStufe ();
    } while (!vornFrei());
}

void erklimmeEineStufe() {
    linksUm();
    vor();
    rechtsUm();
    vor();
}

void rechtsUm() {
    linksUm();
    linksUm();
}
```

```
linksUm();  
}
```

4.6 Boolesche Funktionen

Während Prozeduren dazu dienen, den Befehlsvorrat des Hamsters zu erweitern, dienen boolesche Funktionen dazu, neue Testbefehle einzuführen.

4.6.1 Boolesche return-Anweisung

Boolesche return-Anweisungen werden in booleschen Funktionen zum Liefern eines booleschen Wertes benötigt.

Die Syntax der booleschen return-Anweisung ist sehr einfach: Dem Schlüsselwort `return` folgt ein boolescher Ausdruck und ein abschließendes Semikolon. Boolesche return-Anweisungen sind spezielle Anweisungen, die ausschließlich im Funktionsrumpf boolescher Funktionen verwendet werden dürfen.

Die Ausführung einer booleschen return-Anweisung während der Ausführung einer booleschen Funktion führt zur unmittelbaren Beendigung der Funktionsausführung. Dabei wird der Wert des booleschen Ausdrucks als so genannter Funktionswert zurückgegeben.

4.6.2 Definition boolescher Funktionen

Die Syntax der Definition einer booleschen Funktion unterscheidet sich nur geringfügig von der Definition einer Prozedur. Statt Prozedurkopf, -name und -rumpf spricht man hier von Funktionskopf, Funktionsname und Funktionsrumpf.

Anstelle des Schlüsselwortes `void` bei der Definition einer Prozedur muss bei der Definition einer booleschen Funktion das Schlüsselwort `boolean` am Anfang des Funktionskopfes stehen. Ganz wichtig bei der Definition boolescher Funktionen ist jedoch folgende Zusatzbedingung: In jedem möglichen Weg durch die Funktion muss eine boolesche return-Anweisung auftreten!

Boolesche Funktionen können überall dort in einem Hamster-Programm definiert werden, wo auch Prozeduren definiert werden können.

4.6.3 Aufruf boolescher Funktionen

Eine boolesche Funktion darf überall dort aufgerufen werden, wo auch einer der drei vordefinierten Testbefehle aufgerufen werden darf. Der Aufruf einer booleschen Funktion gilt also als ein spezieller boolescher Ausdruck. Der Funktionsaufruf erfolgt

syntaktisch durch die Angabe des Funktionsnamens gefolgt von einem runden Klammernpaar.

Wird bei der Berechnung eines booleschen Ausdrucks eine boolesche Funktion aufgerufen, so wird in deren Funktionsrumpf verzweigt und es werden die dortigen Anweisungen aufgerufen. Wird dabei eine boolesche return-Anweisung ausgeführt, so wird der Funktionsrumpf unmittelbar verlassen und an die Stelle des Funktionsaufrufs zurückgesprungen. Der von der booleschen return-Anweisung gelieferte Wert (also der Funktionswert) wird dabei zur Berechnung des booleschen Ausdrucks weiterverwendet.

4.6.4 Beispielprogramm

Im folgenden Programm sucht der Hamster auf der rechten Seite eine Nische. Findet er eine, begibt er sich hinein. Beim Suchen benutzt er eine boolesche Funktion `rechtsFrei`.

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

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

boolean rechtsFrei () { // Definition einer booleschen Funktion
    rechtsUm ();
    if (vornFrei()) {
        linksUm();
        return true; // boolesche return - Anweisung
    } else {
        linksUm();
        return false; // boolesche return - Anweisung
    }
}

void main () {
    while (vornFrei() &&
           !rechtsFrei()) { // Aufruf einer booleschen Funktion
        vor ();
    }
    if (rechtsFrei()) { // Aufruf einer booleschen Funktion
        rechtsUm();
        vor();
    }
}
```

4.7 Variablen und Ausdrücke

Durch die Einführung von Variablen und Ausdrücken bekommt der Hamster ein „Gedächtnis“ und lernt rechnen.

4.7.1 Datentypen

Ein Datentyp repräsentiert Werte eines bestimmten Typs. Im Hamster-Modell werden die Datentypen `boolean` und `int` genutzt. Der Datentyp `boolean` repräsentiert boolesche Werte, also `true` und `false`. Der Datentyp `int` repräsentiert ganze Zahlen zwischen -2^{31} und $2^{31} - 1$.

4.7.2 Variablen

Variablen sind Speicherbereiche („Behälter“), in denen Werte abgespeichert werden können. Vor ihrer Benutzung müssen sie definiert werden. Bei der Definition einer Variablen wird ihr ein Name – ein beliebiger Bezeichner – zugeordnet. Außerdem wird durch die Angabe eines Datentyps festgelegt, welche Werte die Variable speichern kann.

Die folgenden Anweisungen definieren eine Variable `frei` zum Speichern von booleschen Werten sowie eine Variable `anzahlKoerner` zum Speichern von ganzen Zahlen. Der Variablen `frei` wird der Initialwert `false` zugewiesen, der Variablen `anzahlKoerner` der Wert 13.

```
boolean frei = false;
int anzahlKoerner = 13;
```

4.7.3 Ausdrücke

Ausdrücke sind spezielle Programmierkonstrukte zum Berechnen und Liefern eines Wertes. Boolesche Ausdrücke haben Sie bereits kennen gelernt. Sie liefern boolesche Werte. Zur Berechnung boolescher Ausdrücke können auch `boolean`-Variablen hinzugezogen werden. Enthält ein boolescher Ausdruck den Namen einer booleschen Variablen, dann wird bei der Auswertung des booleschen Ausdrucks an der entsprechenden Stelle der Wert berücksichtigt, der aktuell in der Variablen gespeichert ist.

Einen weiteren Typ von Ausdrücken stellen arithmetische Ausdrücke dar. Sie berechnen und liefern Werte vom Typ `int`, also ganze Zahlen. Arithmetische Ausdrücke lassen sich auf folgende Art und Weise bilden:

- **int-Literale:** `int`-Literale werden durch Zeichenfolgen beschrieben, die aus dezimalen Ziffern (0, 1, 2, 3, 4, 5, 6, 7, 8, 9) bestehen. Dabei gilt die Einschränkung, dass einer Zahl ungleich 0 keine „0“ vorangestellt werden darf. Gültige `int`-Literale sind also: 0, 2, 4711, 1234560789, ...
- **int-Variablenname:** Der Name einer `int`-Variablen in einem arithmetischen Ausdruck repräsentiert den aktuell in der Variablen gespeicherten Wert.

- Unäre arithmetische Operatoren: Die Zeichen „+“ und „-“ kennzeichnen Vorzeichen von arithmetischen Ausdrücken. Die unären arithmetischen Operatoren sind rechtsassoziativ und besitzen die höchste Priorität aller arithmetischen Operatoren.
- Binäre arithmetische Operatoren: Es existieren insgesamt fünf binäre arithmetische Operatoren, mit denen jeweils zwei andere arithmetische Ausdrücke (die Operanden) verknüpft werden:
 - „+“: liefert als Wert die Summe seiner beiden Operanden (Addition)
 - „-“: liefert als Wert die Differenz seiner beiden Operanden (Subtraktion)
 - „*“: liefert als Wert das Produkt seiner beiden Operanden (Produkt)
 - „/“: liefert als Wert den Quotient seiner beiden Operanden; dabei werden entstehende Nachkommastellen ignoriert (ganzzahlige Division); z.B. $7/3 = 2$
 - „%“: liefert als Wert den Rest einer ganzzahligen Division (Modulo-Operator); z.B. $7\%3 = 1$. Zwischen der Ganzzahldivision und der Restbildung besteht folgende Beziehung: Seien x und y arithmetische Ausdrücke, dann gilt $((x/y) * y) + (x\%y) = x$.

Die binären arithmetischen Operatoren sind linksassoziativ. Die Operatoren „*“, „/“ und „%“ besitzen eine höhere Priorität als die Operatoren „+“ und „-“ („Punkt-vor-Strich-Rechnung“).

- Klammern: Zum Bilden von (komplexen) arithmetischen Ausdrücken können arithmetische Ausdrücke in Paare von runden Klammern gesetzt werden. Dadurch lässt sich die Priorität von arithmetischen Operatoren beeinflussen.

4.7.4 Zuweisung

Mit Hilfe einer Zuweisungsanweisung – kurz auch Zuweisung genannt – können Variablen neue Werte zugewiesen werden. Die alten Werte gehen dabei verloren.

Syntaktisch wird die Zuweisung so gebildet, dass dem Namen der betroffenen Variablen das Zeichen = – der Zuweisungsoperator – und anschließend ein Ausdruck folgt. Bei booleschen Variablen muss das ein boolescher Ausdruck sein, bei int-Variablen ein arithmetischer Ausdruck.

Bei der Ausführung einer Zuweisung wird zunächst der Ausdruck ausgewertet und anschließend der berechnete Wert der Variablen auf der linken Seite des Zuweisungsoperators zugewiesen. Die Zuweisungsanweisung `zahl = zahl + 1;` erhöht bspw. den Wert einer int-Variablen `zahl` um den Wert 1.

4.7.5 Vergleichsausdrücke

Vergleichsausdrücke sind boolesche Ausdrücke, die zum Vergleichen arithmetischer Ausdrücke dienen. Sie liefern boolesche Werte nach folgenden Gesetzmäßigkeiten: Seien x und y zwei arithmetische Ausdrücke, dann gilt:

- $x == y$ liefert genau dann den Wert true, falls der Wert, den x liefert, gleich dem Wert ist, den y liefert (Gleichheitsoperator).
- $x != y$ liefert genau dann den Wert true, falls der Wert, den x liefert, ungleich dem Wert ist, den y liefert (Ungleichheitsoperator).
- $x < y$ liefert genau dann den Wert true, falls der Wert, den x liefert, kleiner ist als der Wert, den y liefert (Kleineroperator).
- $x <= y$ liefert genau dann den Wert true, falls der Wert, den x liefert, kleiner oder gleich dem Wert ist, den y liefert (Kleinervergleichoperator).
- $x > y$ liefert genau dann den Wert true, falls der Wert, den x liefert, größer ist als der Wert, den y liefert (Größeroperator).
- $x >= y$ liefert genau dann den Wert true, falls der Wert, den x liefert, größer oder gleich dem Wert ist, den y liefert (Größervergleichoperator).

Die Vergleichsoperatoren sind linksassoziativ. Die Operatoren $<$, $<=$, $>$ und $>=$ haben eine höhere Priorität als die Operatoren $==$ und $!=$. Weiterhin ist zu beachten, dass die Vergleichsoperatoren eine niedrigere Priorität besitzen als die arithmetischen Operatoren und eine höhere Priorität als der Zuweisungsoperator.

4.7.6 Gültigkeitsbereiche von Variablen

Variablen lassen sich innerhalb von Prozeduren und Funktionen definieren. In diesem Fall nennt man sie lokale Variablen. Variablen, die außerhalb von Prozeduren oder Funktionen definiert werden, heißen globale Variablen.

Als Gültigkeitsbereich einer Variablen wird der Teil eines Programmes bezeichnet, in dem eine Variable genutzt werden kann, d.h. in dem der Name einer Variablen verwendet werden darf. Der Gültigkeitsbereich einer lokalen Variablen erstreckt sich von der Variablendefinition folgenden Anweisung bis zum Ende desselben Blockes und umschließt alle inneren Blöcke. Der Gültigkeitsbereich einer globalen Variablen umfasst das gesamte Hamster-Programm. Im Gültigkeitsbereich einer Variablen darf keine weitere Variable mit demselben Namen definiert werden.

4.7.7 Lebensdauer von Variablen

Während der Gültigkeitsbereich einer booleschen Variablen zur Compilierzeit von Bedeutung ist, ist die Lebensdauer einer booleschen Variablen eine Größe, die zur Laufzeit Relevanz besitzt. Sie ist definiert als die Zeitspanne, während der im Hauptspeicher Speicherplatz für eine Variable reserviert ist. Es gilt dabei: Die

Lebensdauer einer globalen Variablen umfasst die gesamte Ausführungszeit eines Hamster-Programms. Die Lebensdauer einer lokalen Variablen beginnt bei ihrer Definition und endet nach der vollständigen Abarbeitung des Blocks, in dem sie definiert wurde.

4.7.8 Beispielprogramm

Im folgenden Programm läuft der Hamster bis zur nächsten Wand. Anschließend dreht er sich um und läuft zum Ausgangsfeld zurück. Die Anzahl der gelaufenen Schritte merkt er sich dabei in einer globalen Variablen `anzahl`.

```
int anzahl = 0; // globale Variable

void laufeBisZurWand() {
    while (vornFrei()) {
        vor();
        anzahl = anzahl + 1; // Zuweisung
    }
}

void laufeZurueckZumAusgangspunkt() {
    while (anzahl > 0) { // Vergleichsausdruck
        vor ();
        anzahl = anzahl - 1;
    }
}

void main() {
    laufeBisZurWand();
    linksUm();
    linksUm();
    laufeZurueckZumAusgangspunkt ();
}
```

4.8 *int-Funktionen*

Boolesche Funktionen liefern einen Wert vom Typ `boolean`. Dementsprechend sind `int-Funktionen` Funktionen, die einen Wert vom Typ `int`, also eine ganze Zahl, liefern.

4.8.1 Funktionsdefinition

Die Definition einer `int-Funktion` unterscheidet sich nur dadurch von der Definition einer booleschen Funktion, dass im Funktionskopf das Schlüsselwort `boolean` gegen das Schlüsselwort `int` ausgetauscht werden muss und die im Funktionsrumpf obligatorischen `return`-Anweisungen anstelle eines booleschen Wertes einen Wert vom Typ `int` liefern müssen.

4.8.2 Funktionsaufruf

Der Aufruf von int-Funktionen entspricht einem speziellen arithmetischen Ausdruck. int-Funktionen dürfen also überall dort aufgerufen werden, wo arithmetische Ausdrücke stehen dürfen. Der Aufruf einer int-Funktion erfolgt wie der Aufruf einer booleschen Funktion syntaktisch durch die Angabe des Funktionsnamens gefolgt von einem runden Klammernpaar. Beim Aufruf einer int-Funktion wird in den Funktionsrumpf verzweigt und die dortigen Anweisungen werden ausgeführt. Als Wert des arithmetischen Ausdrucks wird in diesem Fall der Wert genommen, den die Funktion berechnet und mit Hilfe einer return-Anweisung geliefert hat.

4.8.3 Verallgemeinerung des Funktionskonzeptes

Neben den Datentypen `boolean` und `int` gibt es in Java und in anderen Programmiersprachen weitere Datentypen, auf deren Einführung im Hamster-Modell zunächst verzichtet wird, für die das Funktionskonzept aber entsprechend gilt.

Funktionen können prinzipiell wie Prozeduren auch in Form von Anweisungen aufgerufen werden. Der gelieferte Funktionswert wird dann einfach ignoriert.

Prozeduren werden ab jetzt als Spezialform von Funktionen aufgefasst. Sie liefern keinen Wert, was durch das Schlüsselwort `void` bei ihrer Definition ausgedrückt wird.

Der Gültigkeitsbereich von Funktionen erstreckt sich über ein gesamtes Programm. Insbesondere dürfen Funktionen auch schon vor ihrer Definition aufgerufen werden.

Wenn sich nach der Ausführung einer Funktion der Programmzustand verändert hat (bspw. die Position oder Blickrichtung des Hamsters), spricht man von einem Seiteneffekt, den die Funktion produziert hat.

4.8.4 Beispielprogramm

Der Hamster bekommt die Aufgabe, so viele Körner abzulegen, wie insgesamt auf seinen vier Nachbarfeldern liegen (natürlich nur, wenn er auch genügend Körner im Maul hat). Er nutzt dabei eine Funktion `koernerVorne`, die ohne Seiteneffekte die Anzahl an Körnern liefert, die sich auf der Kachel vor dem Hamster befindet.

```
// liefert die Anzahl an Koernern, die sich auf dem Feld vor dem
// Hamster befindet; produziert dabei keine Seiteneffekte
int koernerVorne() { // Definition einer int - Funktion
    if (! vornFrei ()) {
        return 0;
    }
    vor();
    int anzahl = 0; // lokale Variable
    while (kornDa()) {
```

```

        nimm();
        anzahl = anzahl + 1;
    }

    // um Seiteneffekte zu vermeiden, muessen nun noch die Koerner
    // wieder abgelegt werden und der Hamster muss auf
    // seine alte Kachel zurueckkehren
    int zurueck = anzahl ;
    while (zurueck > 0) {
        gib();
        zurueck = zurueck - 1;
    }
    kehrt();
    vor();
    kehrt();

    // nun kann die Anzahl an Koernern zurueckgeliefert werden
    return anzahl;
}

void kehrt() {
    linksUm();
    linksUm();
}

void main() {

    // ermittelt die Anzahl an Koernern vor sich
    int nachbarKoerner = koernerVorne(); // Aufruf einer int - Funktion
    int drehungen = 0;

    // dreht sich in die anderen Richtungen und addiert die
    // entsprechenden Koerneranzahlen
    while (drehungen < 3) {
        linksUm ();
        nachbarKoerner = nachbarKoerner +
            koernerVorne (); // Aufruf einer int - Funktion
        drehungen = drehungen + 1;
    }

    // legt entsprechend viele Koerner ab, solange er noch
    // welche im Maul hat
    while (nachbarKoerner > 0 && !maulLeer()) {
        gib();
        nachbarKoerner = nachbarKoerner - 1;
    }
}

```

4.9 Funktionsparameter

Das Parameterkonzept erhöht die Flexibilität von Prozeduren und Funktionen.

4.9.1 Formale Parameter

Parameter sind lokale Variablen von Funktionen, die dadurch initialisiert werden, dass der Funktion bei ihrem Aufruf ein entsprechender Initialisierungswert für die Variable übergeben wird.

Parameter werden im Funktionskopf definiert. Zwischen die beiden runden Klammern wird eine so genannte formale Parameterliste eingeschoben. Die Parameterliste besteht aus keiner, einer oder mehreren durch Kommata getrennten Parameterdefinitionen. Eine Parameterdefinition hat dabei eine ähnliche Gestalt wie die Variablendefinition. Was fehlt, ist ein expliziter Initialisierungsausdruck. In der Tat handelt es sich bei einem Parameter auch um eine ganz normale Variable. Sie ist lokal bezüglich des Funktionsrumpfes. Ihr können im Funktionsrumpf ihrem Typ entsprechend Werte zugewiesen werden und sie kann bei der Bildung von typkonformen Ausdrücken innerhalb des Funktionsrumpfes eingesetzt werden. Man nennt die Parameter innerhalb der Funktionsdefinition auch formale Parameter oder Parametervariablen.

4.9.2 Aktuelle Parameter

Der Funktionsaufruf wird durch die Angabe einer aktuellen Parameterliste zwischen den runden Klammern erweitert. Die durch Kommata getrennten Elemente dieser Liste werden als aktuelle Parameter bezeichnet. Hierbei handelt es sich um Ausdrücke.

4.9.3 Parameterübergabe

Bezüglich der Definition von Funktionen mit (formalen) Parametern und dem Aufruf von Funktionen mit (aktuellen) Parametern sind folgende zusätzliche Bedingungen zu beachten:

- Die Anzahl der aktuellen Parameter beim Aufruf einer Funktion muss gleich der Anzahl der formalen Parameter der Funktionsdefinition sein. (Ausnahme: so genannte varargs-Parameter)
- Für alle Parameter in der angegebenen Reihenfolge muss gelten: Der Typ eines aktuellen Parameters muss konform sein zum Typ des entsprechenden formalen Parameters (boolesche Ausdrücke sind konform zum Typ `boolean`, arithmetische Ausdrücke sind konform zum Typ `int`).

Wird eine Funktion mit Parametern aufgerufen, so passiert folgendes: Die aktuellen Parameter – hierbei handelt es sich ja um Ausdrücke – werden berechnet, und zwar immer von links nach rechts, falls es sich um mehr als einen Parameter handelt. Für jeden formalen Parameter der formalen Parameterliste wird im Funktionsrumpf eine lokale Variable angelegt. Diese Variablen werden anschließend – bei Beachtung der

Reihenfolge innerhalb der Parameterlisten – mit dem Wert des entsprechenden aktuellen Parameters initialisiert. Man spricht in diesem Zusammenhang auch von Parameterübergabe: Der Wert eines aktuellen Parameters wird beim Aufruf einer Funktion einem formalen Parameter der Funktion als Initialisierungswert übergeben.

4.9.4 Überladen von Funktionen

Eigentlich müssen Funktionsnamen in einem Programm eindeutig sein. Zwei oder mehrere Funktionen dürfen jedoch denselben Namen besitzen, falls sich ihre formalen Parameterlisten durch die Anzahl an Parametern oder die Typen der Parameter unterscheiden. Man nennt dieses Prinzip auch Überladen von Funktionen. Die tatsächlich aufgerufene Funktion wird dann beim Funktionsaufruf anhand der Anzahl bzw. Typen der aktuellen Parameterliste bestimmt.

4.9.5 Beispielprogramm

Im folgenden Beispielprogramm wird eine Funktion `vor` mit einem formalen Parameter `anzahl` vom Typ `int` definiert. Die Funktion überlädt dabei die Funktion, die den Hamster-Befehl `vor` repräsentiert. Der Wert von `anzahl` gibt an, um wie viele Kacheln der Hamster nach vorne springen soll. Der Wert braucht dabei erst zur Laufzeit beim Aufruf der Funktion angegeben werden. In der `main`-Funktion wird die Funktion `vor` einmal mit dem Wert 4 aufgerufen, d.h. der Hamster soll vier Kacheln nach vorne springen. Beim zweiten Aufruf wird als Wert die Anzahl an Körnern, die sich im Maul des Hamsters befinden, übergeben. Entsprechend oft hüpfet der Hamster nach vorne.

```
// der Hamster huepft anzahl - mal vor, maximal aber bis zur Mauer
void vor(int anzahl) { // formaler Parameter
    while (vornFrei() && anzahl > 0) {
        vor();
        anzahl = anzahl - 1;
    }
}

// liefert seiteneffektfrei die Koerneranzahl im Maul des Hamsters
int koernerImMaul() {
    int anzahl = 0;
    while (!maulLeer()) {
        gib();
        anzahl = anzahl + 1;
    }
    int ergebnis = anzahl;

    // Koerner wieder fressen
    while (anzahl > 0) {
        nimm();
        anzahl = anzahl - 1;
    }

    return ergebnis ;
}
```

```

void main() {
    vor(4); // aktueller Parameter
    linksUm();
    vor(koernerImMaul()); // aktueller Parameter
}

```

4.10 Rekursion

Funktionen, die sich selbst aufrufen, bezeichnet man als rekursive Funktionen. Die rekursive Funktion `hinUndZurueck` im folgenden Programm bewirkt, dass der Hamster bis zur nächsten Wand und anschließend zurück zu seinem Ausgangspunkt läuft.

```

void hinUndZurueck() {
    if (vornFrei()) {
        vor();
        hinUndZurueck(); // rekursiver Funktionsaufruf
        vor();
    } else {
        kehrt();
    }
}

void kehrt() {
    linksUm();
    linksUm();
}

void main() {
    // der Hamster laeuft zur naechsten Wand und zurueck
    hinUndZurueck();
}

```

5 Ihr erstes Hamster-Programm

Nachdem Sie den Hamster-Simulator gestartet haben, öffnet sich ein Fenster, das in etwa dem in Abbildung 5.1 dargestellten Fenster entspricht. Ganz oben enthält es eine Menüleiste mit 5 Menüs, darunter eine Toolbar mit einer Reihe von Buttons und ganz unten einen Meldungsbereich, in dem Meldungen ausgegeben werden. Im linken Teil befindet sich der Editor-Bereich, im rechten Teil der Simulationsbereich. Im Editor-Bereich geben Sie Hamster-Programme ein und im Simulationsbereich führen Sie Hamster-Programme aus.

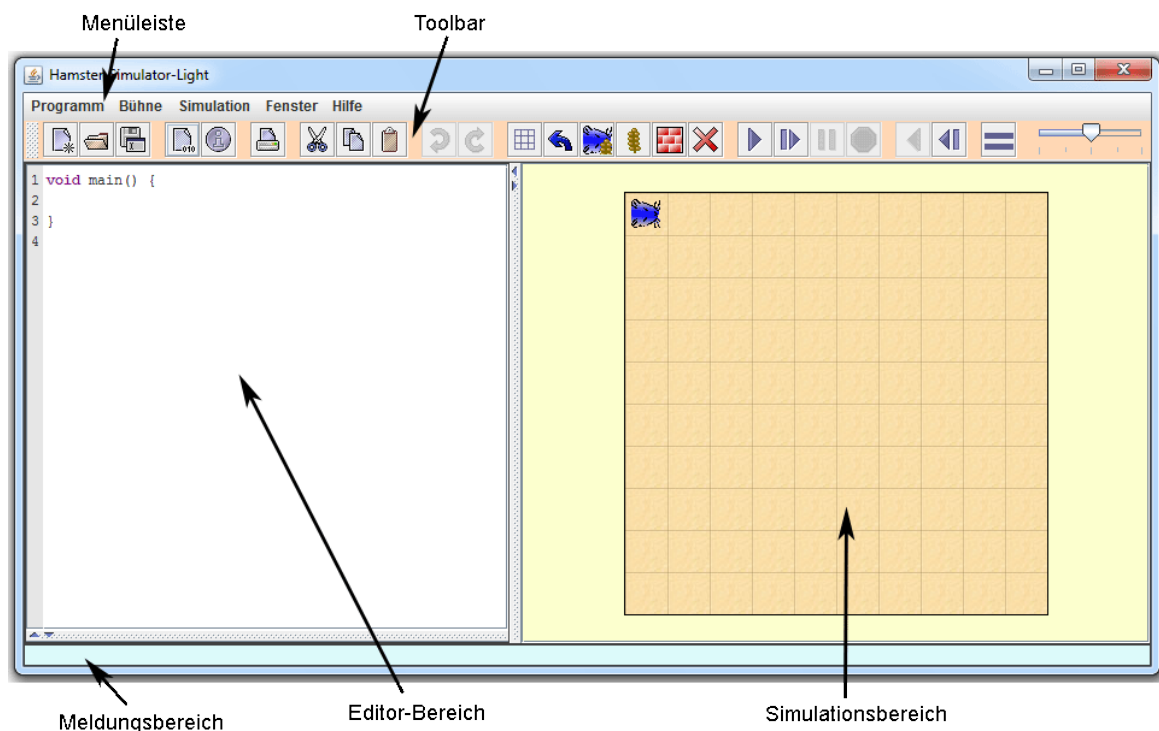


Abb. 5.1: Hamster-Simulator nach dem Öffnen

Um den Hamster ein wenig näher kennen zu lernen, empfehle ich Ihnen, als erstes zunächst die Hamster-Befehle einmal auszuprobieren. Wie das geht, wird in Abschnitt 5.1 erläutert. Anschließend wird in den darauf folgenden Abschnitten 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

5.1 Ausprobieren der Hamster-Befehle

Klicken Sie im Menü „Fenster“ das Menü-Item „Befehlsfenster“ an. Es öffnet sich ein Fenster mit dem Titel „Befehlsfenster“. In diesem Fenster erscheinen alle Befehle, die der Hamster kennt.

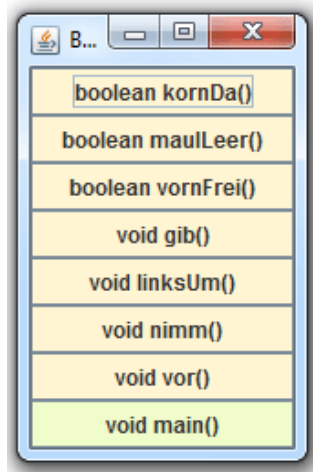


Abb. 5.2: Befehlsfenster

Sie können die jeweiligen Befehle ausführen, indem Sie den Maus-Cursor auf den entsprechenden Button verschieben und diesen anklicken. Klicken Sie bspw. auf den Button „void vor()“, dann hüpfet der Hamster in seiner aktuellen Blickrichtung eine Kachel nach vorne (oder es erscheint eine Fehlermeldung, wenn der Hamster vor einer Mauer steht).

Wenn Sie Programme mit Prozeduren oder Funktionen schreiben und erfolgreich kompilieren, werden die Prozeduren und Funktionen übrigens auch im Befehlsfenster angezeigt und können interaktiv ausgeführt werden. Dabei werden keine Zwischenzustände im Simulationsbereich ausgegeben, sondern immer der jeweilige Endzustand nach Abarbeitung der Funktion.

5.2 Gestaltung eines Hamster-Territoriums

Nun wollen wir ein Hamster-Territorium aufbauen, in dem unser erstes Programm ablaufen soll. Das geschieht im Simulationsbereich. Hier wird das Hamster-Territorium dargestellt. Zur Gestaltung des Territoriums müssen Sie die Buttons 12 – 17 (von links) in der so genannten *Toolbar* benutzen. Diese werden auch als *Gestaltungsbuttons* bezeichnet. 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 „Territoriumsgröße ändern“ (12. 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.

Nun werden wir den Hamster, der immer im Territorium sitzt, umplatziert. Dazu klicken wir den Hamster an und ziehen (man spricht auch von „dragen“) ihn mit gedrückter Maustaste, auf die gewünschte Kachel. Dann lassen wir die Maustaste los.

Standardmäßig schaut der Hamster nach Osten. Mit dem Button „linksum drehen“ (13. 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 Hamster 0 Körner im Maul. Mit Hilfe des Buttons „Körner ins Maul legen“ (14. 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 „Korn setzen“ (15. Button von links). Wenn Sie ihn mit der Maus anklicken, erscheint am Maus-Cursor ein Korn-Symbol. Bewegen Sie die Maus nun über die Kachel, auf der ein Korn platziert werden soll, und klicken Sie dort die Maus. Auf der Kachel erscheint nun ein Korn-Symbol. Liegen auf der Kachel bereits ein oder mehrere Körner, wird deren Anzahl um eins erhöht. Aber Achtung: Es werden maximal 12 Körner angezeigt, auch wenn auf der Kachel mehr als 12 Körner liegen!

Wenn Sie die Shift-Taste Ihrer Tastatur drücken und gedrückt halten und anschließend den Button „Korn setzen“ der Toolbar anklicken, haben Sie die Möglichkeit, mehrere Körner im Territorium zu platzieren, ohne vorher jedes Mal erneut den Button anklicken zu müssen. Solange Sie die Shift-Taste gedrückt halten und eine Kachel anklicken, wird dort ein Korn hinzugefügt.

Mauern werden ähnlich wie Körner auf Kacheln platziert. Nutzen Sie dazu den „Mauer setzen“-Button (16. Button von links). Mauern können natürlich nicht auf

Kacheln platziert werden, auf denen bereits eine Mauer existiert oder auf der sich der Hamster befindet. Wird eine Mauer auf eine Kachel platziert, auf der sich Körner befinden, werden diese gelöscht.

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 (17. Button von links). Klicken Sie anschließend auf die Kacheln, die geleert werden sollen. Der Button ist so lange aktiviert, bis er erneut oder ein anderer Gestaltungsbutton angeklickt wird.

Genauso wie der Hamster können auch Mauern und Körner und Anklicken und Bewegen der Maus bei gedrückter linker Maustaste im Territorium umplatziert werden.

Über das Menü „Bühne“ können Territorien auch in Datei gespeichert und später wieder geladen werden.

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 5.3 skizzierte Territorium.

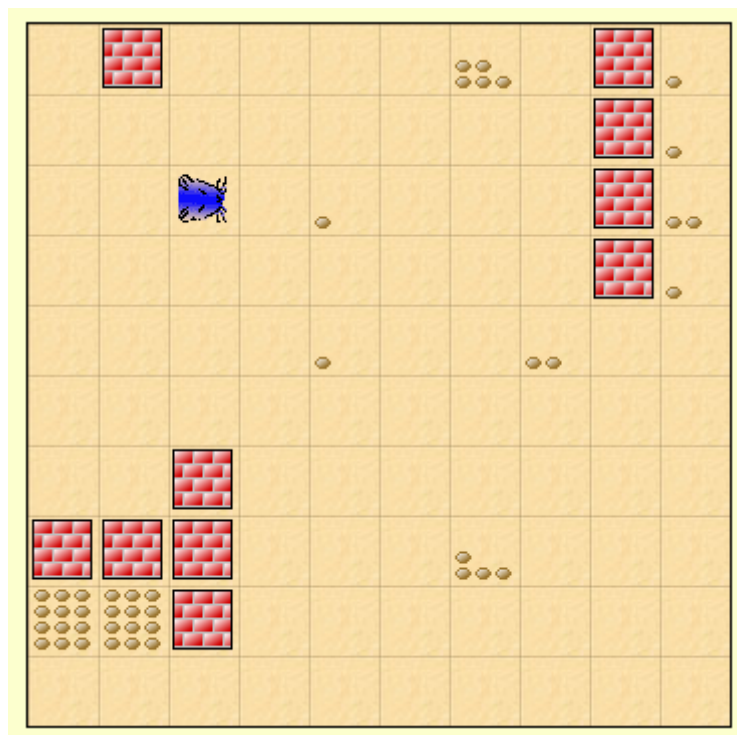


Abb. 5.3: Beispiel-Territorium

5.3 Eingeben eines Hamster-Programms

Nachdem wir unser erstes Hamster-Territorium im Simulationsbereich gestaltet haben, begeben wir uns nun in den Editor-Bereich. Dort werden wir unser erstes Hamster-Programm schreiben.

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

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

void rechtsUm() {
    linksUm();
    linksUm();
    linksUm();
}
```

Das ist unser erstes Hamster-Programm.

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

Weiterhin gibt es im Menü „Programm“ zwei Menü-Items zum Speichern von Programmen in Dateien und zum wieder Laden von abgespeicherten Programmen. Bei ihrer Aktivierung erscheint eine Dateiauswahl-Dialogbox, in der Sie die entsprechende Auswahl der jeweiligen Datei vornehmen müssen. Achtung: Wenn Sie eine abgespeicherte Datei laden, geht der Programmtext, der sich aktuell im Editorbereich befindet, verloren (wenn er nicht zuvor in einer Datei abgespeichert wurde).

Im Menü „Programm“ finden Sie darüber hinaus weitere nützliche Funktionen (Schriftgröße ändern, Drucken, ...).

5.4 Compilieren eines Hamster-Programms

Nachdem wir unser Hamster-Programm geschrieben haben, müssen wir es compilieren. Der Compiler überprüft den Sourcecode auf syntaktische Korrektheit und transformiert ihn – wenn er korrekt ist – in ein ausführbares Programm. Zum Compilieren drücken Sie einfach auf den „Compilieren“-Button (erster Toolbar-Button von links oder „Compilieren“-Menü-Item im „Programm“-Menü). Compiliert wird dann das Programm, das gerade im Eingabebereich sichtbar ist. Es wird zuvor automatisch in einer (temporären) Datei (namens Solist.java) abgespeichert.

Wenn das Programm korrekt ist, erscheint eine Dialogbox mit der Nachricht „Compilierung 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 neu kompilieren.

Wenn das Programm syntaktische Fehler enthält – wenn Sie sich bspw. bei der Eingabe des obigen Programms vertippt haben –, werden unter dem Editor-Bereich 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 Maus-Cursor im Editor-Bereich automatisch in die angegebene Zeile (siehe Abbildung 5.4).

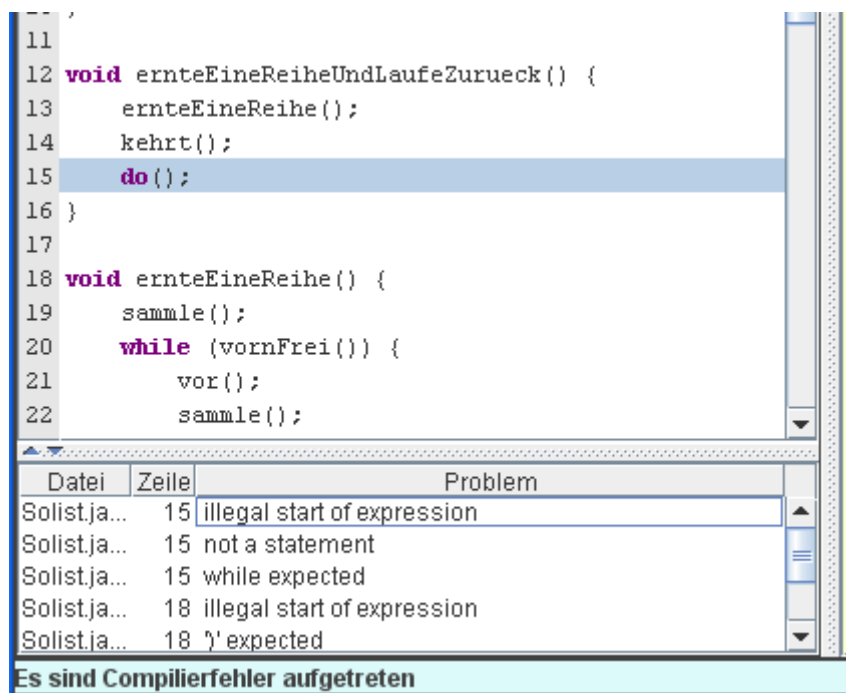


Abb. 5.4: Fehlermeldungen des Compilers

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, compilieren Sie ruhig erst mal 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 erneut compilieren. Wiederholen Sie dies so lange, bis der Compiler die Meldung „Compilierung erfolgreich“ ausgibt. Erst dann können Sie das Programm ausführen!

5.5 Ausführen eines Hamster-Programms

Nach dem erfolgreichen Compilieren 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 Steuern der Programmausführung dienen die Buttons rechts in der Toolbar. Durch Anklicken des „Simulation starten“-Buttons (18. Button von links) starten wir das Programm. 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 „Simulation pausieren“-Buttons (20. Button von links) erreichen. Der Hamster pausiert so lange, bis Sie wieder den „Simulation starten/fortsetzen“-Button (18. Button von links) anklicken. Dann fährt der Hamster mit seiner Arbeit fort. Das Programm vorzeitig komplett abbrechen, können Sie mit Hilfe des „Simulation beenden“-Buttons (21. Button von links).

Wenn Sie ein Programm mehrmals hintereinander im gleichen Territorium ausführen wollen, können Sie mit dem „Rücksetzen“-Button (19. Button von links) den Zustand des Territoriums wieder herstellen, der vor dem letzten Ausführen des Programms Bestand hatte. Eine komplette Zurücksetzung des Territoriums auf den Zustand beim Öffnen des Hamster-Simulators ist mit dem „Komplett zurücksetzen“-Button möglich (23. Button von links).

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 finden Sie übrigens auch im Menü „Simulation“.

5.6 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. Das ist ganz nützlich, wenn Ihr Programm nicht das tut, was es soll, und sie herausfinden wollen, woran der Fehler liegt.

Klicken Sie zum Debuggen zunächst den „Ablaufverfolgung aktivieren“-Button in der Toolbar an (24. Button von links). Es wird das so genannte Debugger-Fenster mit dem Titel „Debugger“ geöffnet. Im linken Bereich des Debugger-Fensters wird der Prozedur-Stack angezeigt, das ist die aktuelle Prozedur sowie die Prozeduren, aus denen die Prozedur heraus aufgerufen wurde. Im rechten Bereich werden – falls vorhanden – die gültigen Variablen und ihre aktuellen Werte dargestellt (siehe Abb. 5.5).

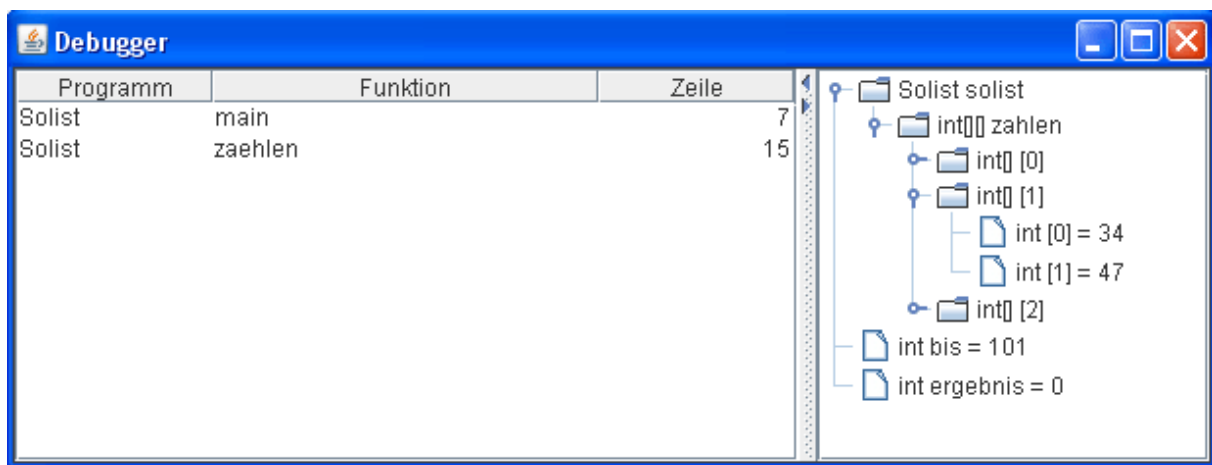


Abb. 5.5: Debugger-Fenster

Jetzt können Sie über den „Schritt“-Button (19. Toolbar-Button von links) jeden Befehl einzeln ausführen und bekommen im Editorbereich durch einen blauen Balken angezeigt, welcher Befehl bzw. welche Zeile als nächstes ausgeführt wird. Bei einem Prozeduraufruf wird dabei auch automatisch in die entsprechende Prozedur verzweigt. Im Debugger-Fenster wird zusätzlich der Prozedur-Stack

angezeigt, d.h. die aktuelle Prozedur sowie die Prozeduren, aus denen die Prozedur heraus aufgerufen wurde.

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

Die Ablaufverfolgung kann übrigens jederzeit durch erneutes Klicken des „Ablaufverfolgung“-Buttons deaktiviert bzw. reaktiviert werden.

Wenn Sie möchten, dass der Programmablauf beim Erreichen einer bestimmten Zeile automatisch in den Pausenzustand gelangt, können Sie vor oder während des Programmablaufs in der entsprechenden Zeile einen Breakpoint setzen. Führen Sie dazu im Editorbereich auf der entsprechenden Zeilennummer einen Doppelklick mit der Maus aus. Breakpoints werden durch eine violette Hinterlegung der Zeilennummer kenntlich gemacht. Durch erneuten Doppelklick oder über ein Popup-Menü, das oberhalb der Zeilennummern aktiviert werden kann, kann ein Breakpoint oder auch alle Breakpoints wieder gelöscht werden. Breakpoints nutzt man häufig dazu, dass man ein Programm bis zu einer bestimmten Stelle normal ablaufen lässt und ab dort dann die Möglichkeit der zeilenweisen Ausführung nutzt.

5.7 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 im Detail nachlesen.

6 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 ein Fenster mit dem Titel „Hamster-Simulator-Light“. Abbildung 6.1 skizziert die einzelnen Komponenten des Fensters.

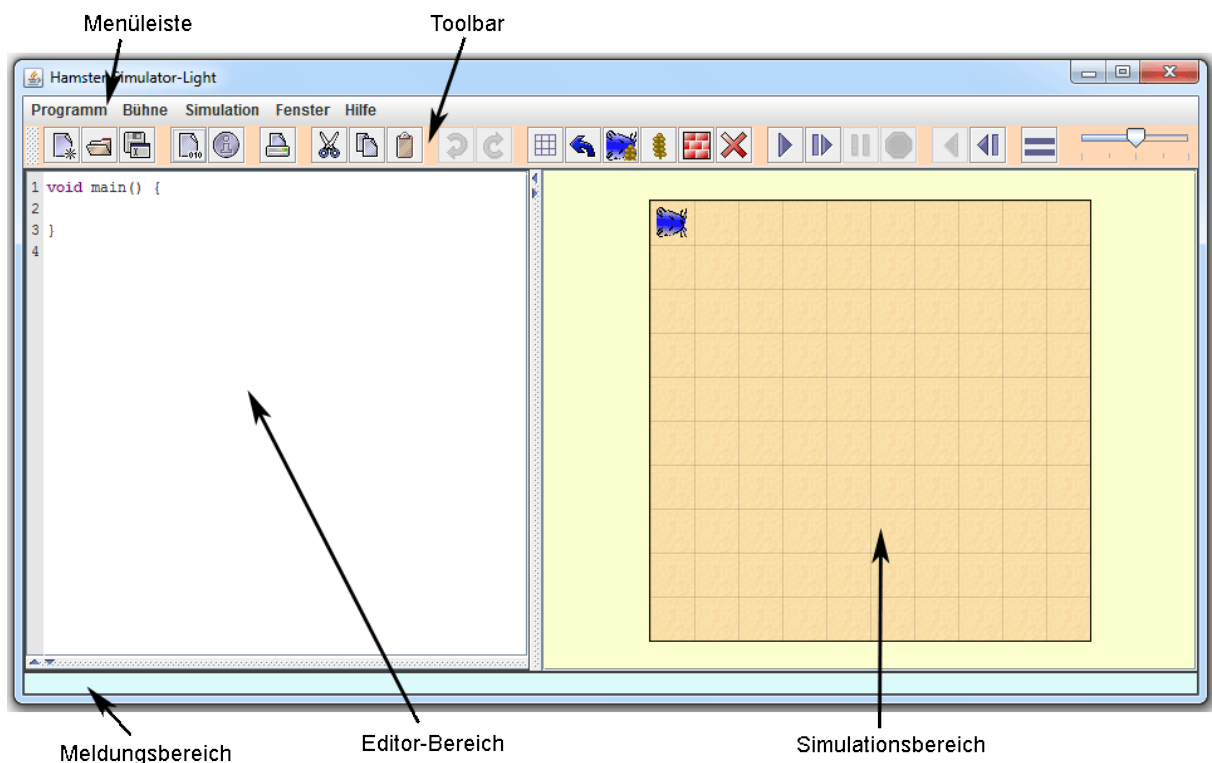


Abb. 6.1: Komponenten des Simulator-Fensters

Die Menüleiste oben im Fenster beinhaltet 5 Menüs. Darunter ist eine Toolbar mit Buttons platziert, über die die wichtigsten Funktionen der Menüs durch Anklicken eines Buttons schneller ausgeführt werden können. Ganz unten im Meldungsbereich werden wichtige Meldungen ausgegeben. Die Eingabe von Hamster-Programmen erfolgt im Editor-Bereich links und die Ausführung von Hamster-Programmen wird im Simulationsbereich (auch „Bühne“ genannt) visualisiert.

Als Hauptfunktionsbereiche des Hamster-Simulators lassen sich identifizieren:

- Verwalten und Editieren von Hamster-Programmen
- Compilieren von Hamster-Programmen
- Verwalten und Gestalten von Hamster-Territorien
- Interaktives Ausführen von Hamster-Befehlen
- 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.

6.1 Grundfunktionen

In diesem Unterabschnitt werden einige wichtige Grundfunktionalitäten graphischer Benutzungsoberflächen beschrieben. Der Abschnitt ist für diejenigen von 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.

6.1.1 Anklicken

Wenn im Folgenden von „Anklicken eines Objektes“ oder „Anklicken eines Objektes mit der Maus“ gesprochen wird, bedeutet das, dass Sie den Maus-Cursor auf dem Bildschirm durch Verschieben der Maus auf dem Tisch über das Objekt platzieren und dann die – im Allgemeinen linke – Maustaste drücken.

6.1.2 Tooltips

Als *Tooltips* werden kleine Rechtecke bezeichnet, die automatisch auf dem Bildschirm erscheinen, wenn man den Maus-Cursor auf entsprechende Objekte platziert (siehe Abbildung 6.2). In den Tooltips werden bestimmte Informationen ausgegeben.



Abb. 6.2: Tooltip

6.1.3 Button

Buttons sind Objekte der Benutzungsoberfläche, die man anklicken kann und die daraufhin eine bestimmte Aktion auslösen (siehe Abbildung 6.3). Buttons besitzen

eine textuelle Beschreibung (z.B. „OK“) oder eine Graphik, die etwas über die Aktion aussagen. Sie erkennen Buttons an der etwas hervorgehobenen Darstellung. Graphik-Buttons sind in der Regel Tooltips zugeordnet, die die zugeordnete Aktion beschreiben.



Abb. 6.3: Buttons

6.1.4 Menü

Menüs befinden sich ganz oben in einem Fenster in der so genannten *Menüleiste* (siehe Abbildung 6.4). Sie werden durch einen Text beschrieben (Programm, Bühne, Simulation, ...). Klickt man die Texte an, öffnet sich eine Box mit so genannten *Menü-Items*. Diese bestehen wiederum aus Texten, die man anklicken kann. Durch Anklicken von Menü-Items werden genauso wie bei Buttons Aktionen ausgelöst, die im Allgemeinen durch die Texte beschrieben werden (Speichern, Kopieren, ...). Nach dem Anklicken eines Menü-Items 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.



Abb. 6.4: Menüleiste und Menü

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

Manchmal erscheinen bestimmte Menü-Items etwas heller. Man sagt auch, sie sind ausgegraut. In diesem Fall kann man das Menü-Item nicht anklicken und die

zugeordnete Aktion nicht auslösen. Das Programm befindet sich in einem Zustand, in dem die Aktion keinen Sinn machen würde.

6.1.5 Toolbar

Direkt unterhalb der Menüleiste ist die so genannte *Toolbar* angeordnet (siehe Abbildung 6.5). Sie besteht aus einer Menge an Graphik-Buttons, die Alternativen zu den am häufigsten benutzten Menü-Items der Menüs darstellen.



Abb. 6.5: Toolbar

6.1.6 Popup-Menü

Popup-Menüs sind spezielle Menüs, die bestimmten Elementen auf dem Bildschirm zugeordnet sind (siehe Abbildung 6.6). Man öffnet sie dadurch, dass man den Maus-Cursor auf das entsprechende Element verschiebt und danach die rechte Maustaste drückt. Genauso wie bei normalen Menüs erscheint dann eine Box mit Menü-Items.

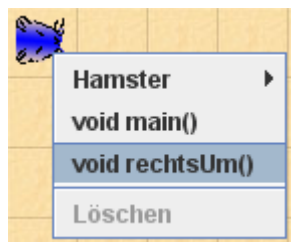


Abb. 6.6: Popup-Menü

6.1.7 Eingabefeld

Eingabefelder dienen zur Eingabe von Zeichen (siehe Abbildung 6.7). Positionieren Sie dazu den Maus-Cursor auf das Eingabefeld und klicken Sie die Maus. Anschließend können Sie über die Tastatur Zeichen eingeben, die im Eingabefeld erscheinen.

6.1.8 Dialogbox

Beim Auslösen bestimmter Aktionen erscheinen so genannte *Dialogboxen* auf dem Bildschirm (siehe Abbildung 6.7). Sie enthalten in der Regel eine Menge von graphischen Objekten, wie textuelle Informationen, Eingabefelder und Buttons. Wenn eine Dialogbox auf dem Bildschirm erscheint, sind alle anderen Fenster des Programms für Texteingaben oder Mausklicks gesperrt. Zum Schließen einer

Dialogbox, d.h. um die Dialogbox wieder vom Bildschirm verschwinden zu lassen, dienen in der Regel eine Menge an Buttons, die unten in der Dialogbox angeordnet sind. Durch Anklicken eines „OK-Buttons“ wird dabei die der Dialogbox zugeordnete Aktion ausgelöst. Durch Anklicken des „Abbrechen-Buttons“ wird eine Dialogbox geschlossen, ohne dass irgendwelche Aktionen ausgelöst werden.

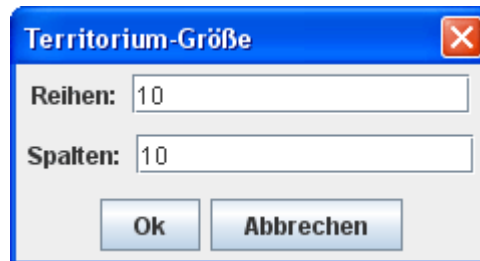


Abb. 6.7: Dialogbox mit Eingabefeldern

6.1.9 Dateiauswahl-Dialogbox

Dateiauswahl-Dialogboxen sind spezielle Dialogboxen, die zum Speichern und Öffnen von Dateien benutzt werden (siehe Abbildung 6.8). Sie spiegeln im Prinzip das Dateisystem wider und enthalten Funktionalitäten zum Verwalten von Dateien und Ordern.

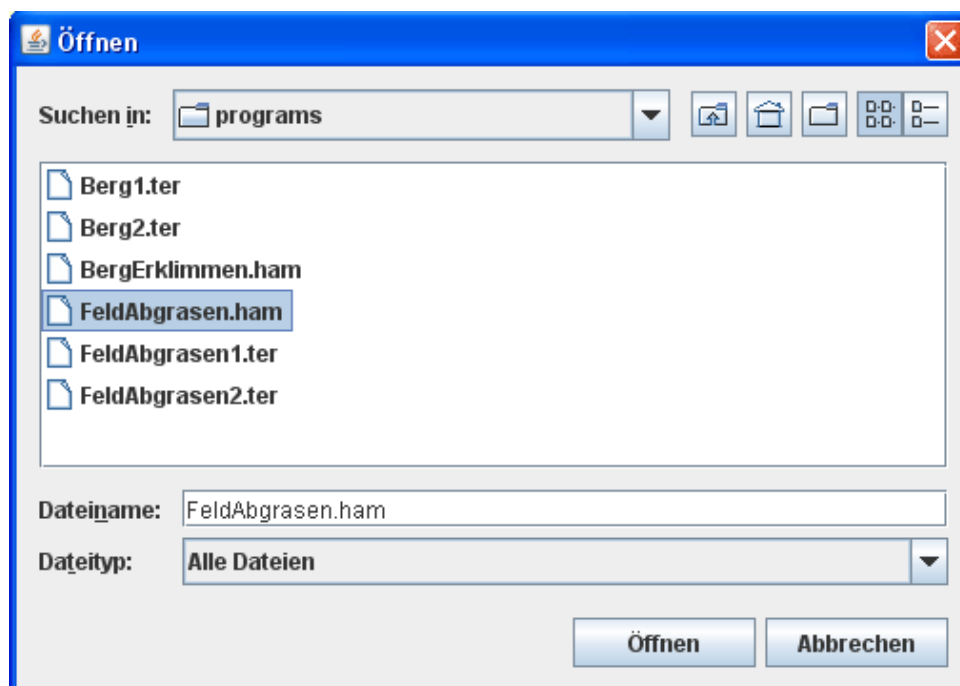


Abb. 6.8: Dateiauswahl-Dialogbox

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.

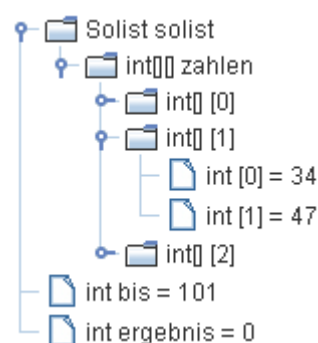
Dateiauswahl-Dialogboxen stellen jedoch noch zusätzliche Funktionalitäten bereit. Durch Doppelklick auf einen Ordner kann man in den entsprechenden Ordner wechseln. Es werden dann anschließend die Dateien und Unterordner dieses Ordners im mittleren Bereich angezeigt. Um zu einem übergeordneten Ordner 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.

6.1.10 Elementbaum

Ein *Elementbaum* repräsentiert Elemente und strukturelle Beziehungen zwischen ihnen, bspw. die Ordner und Dateien des Dateisystems (siehe Abbildung 6.9).



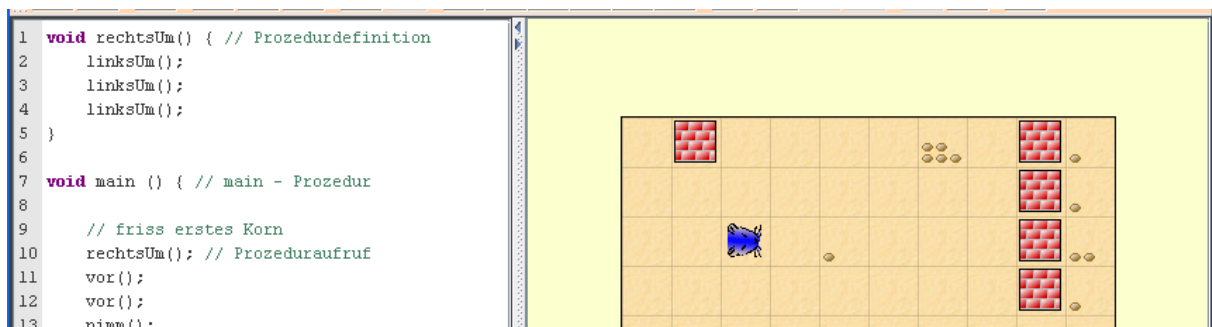
Abbi. 6.9: Elementbaum mit Verzeichnissen und Dateien

Unterschiedliche Elementtypen werden dabei durch unterschiedliche Symbole dargestellt, hinter denen entsprechende Bezeichnungen erscheinen. Durch Anklicken der Symbole auf der linken Seite kann man Strukturen öffnen und schließen, d.h. Unterstrukturen sichtbar bzw. unsichtbar machen.

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. Menü-Items zum Löschen und Umbenennen des entsprechenden Ordners bzw. der entsprechenden Datei.

6.1.11 Split-Pane

Eine Split-Pane ist ein Element, das aus zwei Bereichen und einem Balken besteht. (siehe Abbildung 6.10). Die beiden Bereiche können dabei links und rechts oder oberhalb und unterhalb des Balkens liegen. Wenn Sie den Balken mit der Maus anklicken und bei gedrückter Maustaste nach links oder rechts (bzw. oben oder unten) verschieben, vergrößert sich einer der beiden Bereiche und der andere verkleinert sich. Durch Klicken auf einen der beiden Pfeile auf dem Balken können Sie einen der beiden Bereiche auch ganz verschwinden lassen.



Abbi. 6.10: Split-Pane

6.2 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 der Editor-Bereich zum Editieren von Hamster-Programmen (siehe Abbildung 6.11)

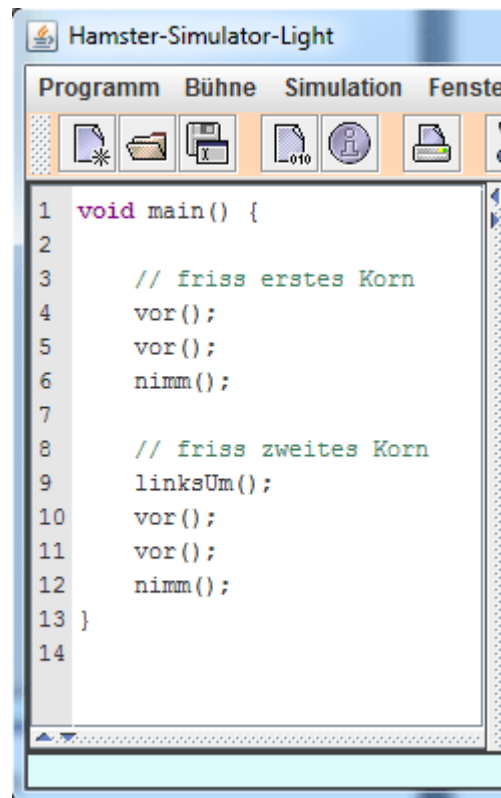


Abb. 6.11: Editor-Bereich des Hamster-Simulators

Im Editor-Bereich können Sie Programme eintippen. Für das Verwalten und Editieren von Programmen ist das Menü „Programm“ wichtig. Unterhalb der Menüleiste ist eine spezielle Toolbar zu sehen, über die Sie die wichtigsten Funktionen der Menüs auch schneller erreichen und ausführen können. Schieben Sie einfach mal die Maus über die Buttons. Dann erscheint jeweils ein Tooltip, der die Funktionalität des Buttons anzeigt. Die für das Editieren von Programmen wichtigen Buttons der Toolbar werden in Abbildung 6.12 skizziert.



Abb. 6.12: Editor-Buttons der Toolbar

6.2.1 Schreiben eines neuen Hamster-Programms

Das Schreiben eines neuen Hamster-Programms ist durch das entsprechende Eintippen des Sourcecodes im Editor-Bereich möglich.

6.2.2 Ändern des aktuellen Hamster-Programms

Möchten Sie Teile des aktuellen Hamster-Programms ändern, klicken Sie im Editor-Bereich einfach an die entsprechende Stelle und fügen dort die entsprechenden Wörter ein oder löschen sie.

6.2.3 Löschen des aktuellen Hamster-Programm

Komplett löschen können Sie das aktuelle Hamster-Programm des Editor-Bereichs, indem Sie den kompletten Sourcecode mit der Maus selektieren und dann in der Toolbar den „Ausschneiden“-Button (7. Button von links) drücken. Alternativ können Sie in der Toolbar den „Neu“-Button anklicken (1. Button von links).

Achtung: Beim Löschen oder Neu-Erstellen geht der aktuelle Inhalt des Editor-Bereichs verloren! Sie müssen ihn also gegebenenfalls vorher in einer Datei abspeichern.

6.2.4 Abspeichern des aktuellen Hamster-Programms

Normalerweise müssen Sie sich nicht um das Speichern des aktuellen Programms kümmern. Es wird automatisch vor dem Compilieren in einer internen Datei abgespeichert. Wenn Sie jedoch ein Programm explizit abspeichern möchten, können Sie in der Toolbar den „Speichern“-Button (3. Button von links) anklicken. Es öffnet sich eine Dateiauswahl-Dialogbox, über die Sie die gewünschte Datei auswählen können.

6.2.5 Öffnen eines einmal abgespeicherten Hamster-Programms

Möchten Sie ein einmal abgespeichertes Hamster-Programm wieder in den Editorbereich laden, können Sie dies über den „Laden“-Button der Toolbar (2. Button von links) tun. Nach dem Anklicken des Buttons erscheint eine Dateiauswahl-Dialogbox, über die Sie die gewünschte Datei auswählen können.

Achtung: Beim Laden einer abgespeicherten Datei geht der aktuelle Inhalt des Editor-Bereichs verloren! Sie müssen ihn also gegebenenfalls vorher in einer Datei abspeichern.

6.2.6 Drucken eines Hamster-Programms

Über den „Drucken“-Button (6. Toolbar-Button von links) können Sie das aktuelle Programm des Editor-Bereichs drucken. Es öffnet sich eine Dialogbox, in der Sie die entsprechenden Druckeinstellungen vornehmen und den Druck starten können.

6.2.7 Editier-Funktionen

Im Editor-Bereich 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 „Programm“-Menü bzw. die entsprechenden Editor-Buttons in der Toolbar aktiviert werden können.

- „Ausschneiden“-Button (7. Toolbar-Button von links): Hiermit können Sie komplette Passagen des Editor-Bereichs 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 (8. Toolbar-Button von links): Hiermit können Sie komplette Passagen des Editor-Bereichs in einen Zwischenpuffer kopieren. Markieren Sie die zu kopierende Passage mit der Maus und klicken Sie dann den Button an.
- „Einfügen“-Button (9. 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 (10. 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.
- „Wiederherstellen“-Button (11. 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.

Die gerade aufgelisteten Funktionen finden Sie auch im „Programm“-Menü. Als zusätzliche Funktionalitäten werden dort angeboten:

- Einrückung: Durch Anklicken des Menü-Items wird der Einrück-Modus aktiviert bzw. deaktiviert. Ist der Einrück-Modus aktiviert, werden beim Eingeben von Text im Editor-Bereich automatisch Einrückungen an die entsprechende Spalte vorgenommen, wenn Sie die Enter-Taste drücken.

- Schriftgröße: Hierüber können Sie die Schriftgröße des Editor-Bereichs anpassen.

6.3 Compilieren von Hamster-Programmen

Beim Compilieren werden Programme – genauer gesagt der Sourcecode – auf ihre (syntaktische) Korrektheit überprüft und im Erfolgsfall ausführbare Programme erzeugt. Zum Compilieren von Programmen dient im Hamster-Simulator der „Compilieren“-Button (4. Button der Toolbar von links) bzw. das Menü-Item „Compilieren“ im „Programm“-Menü (siehe Abbildung 6.13).

An der Farbe des Compiler-Buttons können Sie erkennen, ob Compilieren aktuell notwendig ist oder nicht. Immer wenn Sie Änderungen im Editor-Bereich vorgenommen haben, erscheint der Button rot, und die Änderungen werden erst dann berücksichtigt, wenn Sie (erneut) compiliert haben. Erscheint der Button in einer neutralen Farbe, ist kein Compilieren notwendig.

6.3.1 Compilieren

Wenn Sie den „Compilieren“-Button anklicken, wird das Programm, das gerade im Editor-Bereich sichtbar ist, in einer internen Datei (mit dem Namen „Solist.java“) abgespeichert und kompiliert.

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

6.3.2 Beseitigen von Fehlern

Wenn Ihr Programm Fehler enthält, öffnet sich unterhalb des Editor-Bereichs in einer Scroll-Pane ein neuer Bereich, der die Fehlermeldungen des Compilers anzeigt (siehe Abbildung 6.13). Es wurde kein (neues) ausführbares Programm erzeugt! Jede Fehlermeldung erscheint in einer eigenen Zeile. Jede Zeile enthält eine Beschreibung des (wahrscheinlichen) Fehlers sowie die entsprechende Zeile der Anweisung im Programm. Wenn Sie eine Fehlermeldung anklicken, wird die entsprechende Anweisung im Eingabebereich blau markiert und der Maus-Cursor an die entsprechende Stelle gesetzt. Sie müssen nun die einzelnen Fehler beseitigen und dann erneut speichern und compilieren, bis Ihr Programm keine Fehler mehr enthält. Der Fehlermeldungs-bereich schließt sich dann automatisch wieder.

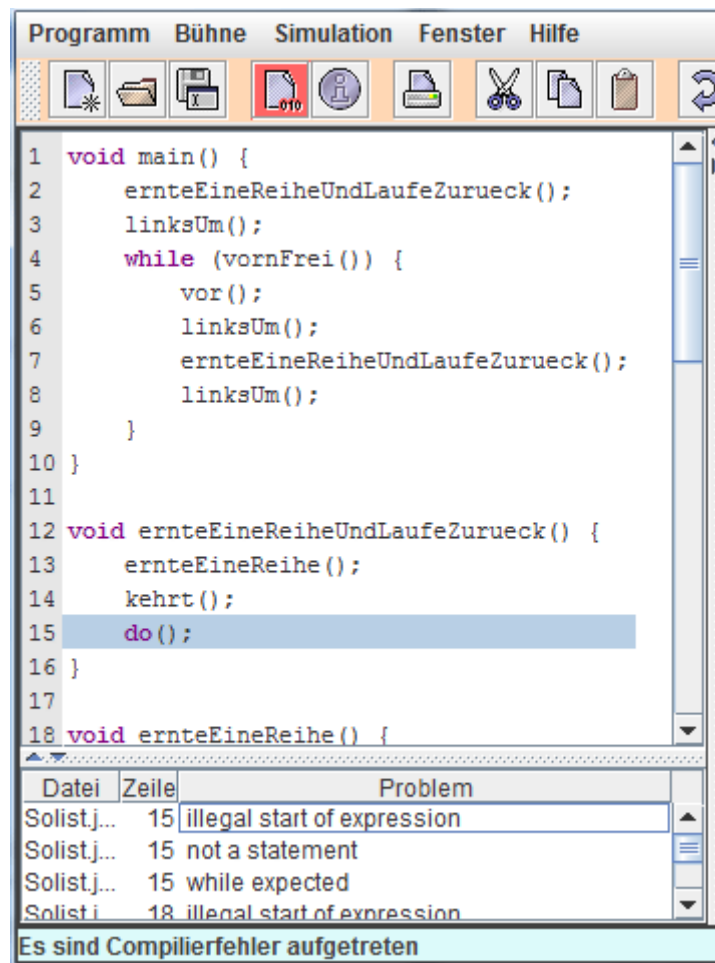


Abb. 6.13: Fehlermeldungen des Compilers

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 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.

6.4 Verwalten und Gestalten von Hamster-Territorien

Das Hamster-Territorium umfasst standardmäßig 10 Reihen und 10 Spalten. Der 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.

In der Toolbar dienen die Buttons 12 – 17 von links zum Gestalten des Hamster-Territoriums. Über das Menü „Bühne“ können Hamster-Territorien verwaltet werden (siehe auch Abbildung 6.14).



Abb. 6.14: Territorium-Buttons der Toolbar

Normalerweise sollten Sie ein Territorium vor der Ausführung eines Programms gestalten. Es ist jedoch auch möglich, während der Programmausführung noch Umgestaltungen vorzunehmen.

6.4.1 Verändern der Größe des Hamster-Territoriums

Durch Anklicken des „Größe ändern“-Buttons (12. 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.

Achtung: Wenn Sie das Territorium verkleinern, werden Mauern und Körner, die sich dann außerhalb des Territoriums befinden würden, gelöscht. Steht der Hamster beim Verkleinern auf einer Kachel außerhalb der neuen Territoriumsgröße, wird er auf die Kachel (0/0) versetzt (eine u.U. sich dort befindende Mauer wird zuvor gelöscht).

6.4.2 Umplatzieren des Hamsters im Hamster-Territorium

Um den Hamster im Hamster-Territorium auf eine andere Kachel zu platzieren, klicken Sie ihn mit der Maus an und ziehen ihn bei gedrückter Maustaste auf die gewünschte Kachel.

6.4.3 Setzen der Blickrichtung des Hamsters

Um die Blickrichtung des Hamsters zu ändern, klicken Sie bitte den „Hamster drehen“-Button (13. Toolbar-Button von links) an. Bei jedem Klick auf diesen Button dreht sich der Hamster um 90 Grad linksrum.

6.4.4 Abfragen und Festlegen der Körneranzahl im Maul des Hamsters

Um die Anzahl an Körnern im Maul des Hamsters festzulegen, klicken Sie bitte den „Körner im Maul“-Button (14. 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 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 Hamster hat die eingegebene Anzahl an Körnern im Maul.

6.4.5 Platzieren von Körnern auf Kacheln des Hamster-Territorium

Um auf einer Kachel des Hamster-Territoriums ein Korn zu platzieren, klicken Sie in der Toolbar den „Korn setzen“-Button (15. Toolbar-Button von links). Klicken Sie anschließend die entsprechende Kachel an. Das Korn wird dort platziert (insofern sich dort keine Mauer befindet). Achtung: Auch wenn sich auf einer Kachel mehr als 12 Körner befinden, werden maximal 12 Körner auf einer Kachel angezeigt.

Um nicht für jedes Platzieren eines Kornes erneut den „Korn setzen“-Button anklicken zu müssen, können Sie auch folgendes tun: Drücken Sie die Shift-Taste der Tastatur und Klicken bei gedrückter Shift-Taste den „Korn setzen“-Button an. Solange Sie nun die Shift-Taste gedrückt halten, können Sie durch Anklicken einer Kachel des Hamster-Territoriums dort ein (weiteres) Korn platzieren.

Körner können auch im Territorium umplaziert werden. Klicken Sie dazu die entsprechende Kachel an und ziehen Sie die dortigen Körner bei gedrückter Maustaste auf die gewünschte Kachel. Die Körner aller Kacheln, die dabei berührt werden, werden dabei eingesammelt.

6.4.6 Platzieren von Mauern auf Kacheln des Hamster-Territorium

Um auf einer Kachel des Hamster-Territoriums eine Mauer zu platzieren, klicken Sie in der Toolbar den „Mauer setzen“-Button (16. Toolbar-Button von links). Klicken Sie anschließend die entsprechende Kachel an. Die Mauer wird dort platziert (insofern sich dort nicht der Hamster befindet). Achtung: Wenn sich auf der Kachel Körner befinden, werden diese gelöscht.

Um nicht für jedes Platzieren einer Mauer erneut den „Mauer setzen“-Button anklicken zu müssen, können Sie auch folgendes tun: Drücken Sie die Shift-Taste der Tastatur und Klicken bei gedrückter Shift-Taste den „Mauer setzen“-Button an. Solange Sie nun die Shift-Taste gedrückt halten, können Sie durch Anklicken einer Kachel des Hamster-Territoriums dort eine Mauer platzieren.

Mauern können auch im Territorium umplaziert werden. Klicken Sie dazu die entsprechende Kachel an und ziehen Sie die dortige Mauer bei gedrückter Maustaste auf die gewünschte Kachel. Körner auf Kacheln, die dabei berührt werden, werden dabei gelöscht.

6.4.7 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 in der Toolbar den „Kachel löschen“-Button (17. 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.

Eine Deaktivierung der Kachel-Löschen-Funktion ist durch erneutes Anklicken des „Kachel löschen“-Buttons möglich. Eine Deaktivierung erfolgt automatisch, wenn ein anderer der Buttons zum Territorium-Gestalten gedrückt wird.

Eine weitere Möglichkeit, Körner oder Mauern im Territorium wieder zu löschen, besteht darin, über den entsprechenden Elementen mit Hilfe der rechten Maustaste ein Popup-Menü zu aktivieren und das darin erscheinende Item „Löschen“ anzuklicken.

6.4.8 Abspeichern eines Hamster-Territoriums

Sie können einmal gestaltete Hamster-Territorien in einer Datei abspeichern und später wieder laden. Zum Abspeichern des aktuellen Territoriums aktivieren Sie im Menü „Bühne“ das Menü-Item „Speichern unter...“. 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.

Weiterhin ist es möglich, das aktuell Territorium als Bild (gif- oder png-Datei) abzuspeichern. Eine entsprechende Funktion findet sich im „Bühne“-Menü.

6.4.9 Wiederherstellen eines abgespeicherten Hamster-Territoriums

Abgespeicherte Hamster-Territorien können mit dem „Laden“-Menü-Item des „Bühne“-Menüs wieder geladen werden. Es erscheint eine Dateiauswahl-Dialogbox, in der Sie die zu ladende Datei auswählen können. Nach dem Anklicken des OK-Buttons schließt sich die Dialogbox und das entsprechende Hamster-Territorium ist wiederhergestellt.

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

6.5 Interaktives Ausführen von Hamster-Befehlen

Sie können einzelne Hamster-Befehle oder selbst definierte Funktionen und Prozeduren bspw. zu Testzwecken auch interaktiv ausführen. Aktivieren Sie dazu im Menü „Fenster“ den Eintrag „Befehlsfenster“. Es öffnet sich das so genannte Befehlsfenster (siehe Abbildung 6.15).

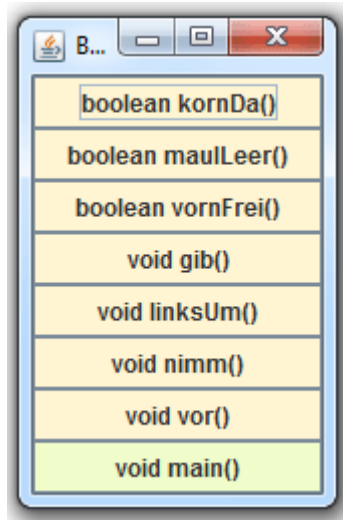


Abb. 6.15: Befehlsfenster

6.5.1 Befehlsfenster

Im Befehlsfenster werden die 7 Hamster-Befehle (mit grünlichem Hintergrund) sowie die Prozeduren und Funktionen dargestellt, die im aktuellen Hamster-Programm beim letztmaligen erfolgreichen Compilieren definiert waren (mit orangenem Hintergrund).

Beim Anklicken mit der Maus werden die entsprechenden Befehle jeweils ausgeführt. Die Ausführung erfolgt dabei ohne Anzeige von Zwischenzuständen. D.h. wird bspw. eine Prozedur `rechtsUm` dadurch definiert, dass dreimal der Befehl `linksUm` aufgerufen wird, und wird diese Prozedur `rechtsUm` im Befehlsfenster aktiviert, dreht sich der Hamster tatsächlich nur einmal um 90 Grad nach rechts.

Dauert die Ausführung eines Befehls mehr als 5 Sekunden (vermutlich enthält dann die Funktion eine Endlosschleife), wird der Befehl abgebrochen und der Hamster-Simulator wird komplett auf seinen Startzustand zurückgesetzt.

6.5.2 Parameter

Besitzt eine Prozedur oder Funktion Parameter, erscheint nach ihrer Aktivierung im Befehlsfenster eine Dialogbox, in der die entsprechenden aktuellen Parameterwerte eingegeben werden müssen. Hinweis: Aktuell wird nur die Eingabe von Werten der Java-Standard-Datentypen (`int`, `boolean`, `float`, ...) sowie die Eingabe von Zeichenketten (Strings) unterstützt.

6.5.3 Rückgabewerte von Funktionen

Bei der Ausführung von Funktionen wird der jeweils gelieferte Wert in einem Dialogfenster dargestellt.

6.5.4 Befehls-Popup-Menü

Alternativ zur Benutzung des Befehlsfensters ist es auch möglich, über ein Popup-Menü die Hamster-Befehle interaktiv auszuführen. Sie können dieses Popup-Menü durch Drücken der rechten Maustaste oberhalb des Hamsters im Territorium aktivieren (siehe Abbildung 6.16).

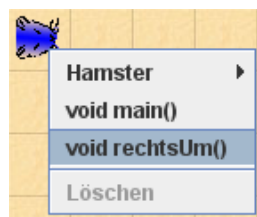


Abb. 6.16: Befehls-Popup-Menü

6.6 Ausführen von Hamster-Programmen

Ausgeführt werden können (erfolgreich compilierte) Hamster-Programme mit Hilfe der in Abbildung 6.17 skizzierten Buttons der Toolbar. Alle Funktionen sind darüber hinaus auch über das Menü „Simulation“ aufrufbar.

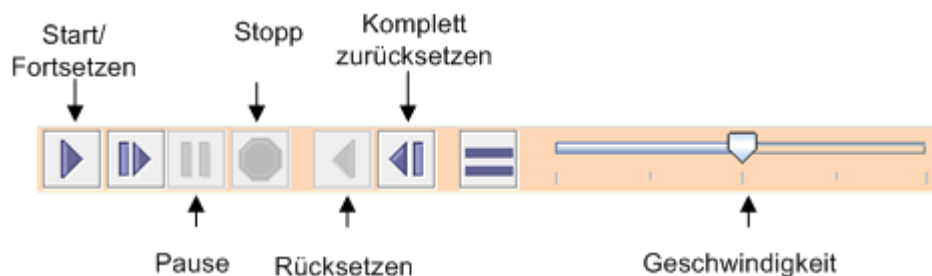


Abb. 6.17: Simulationsbuttons der Toolbar

6.6.1 Starten eines Hamster-Programms

Bevor ein Hamster-Programm ausgeführt werden kann, muss es erfolgreich compiliert worden sein. Gestartet werden kann das aktuelle Hamster-Programm dann durch Anklicken des „Start/Fortsetzen“-Buttons (18. Toolbar-Button von links).

Nach dem Starten eines Hamster-Programms wird der Hamster im Hamster-Territorium aktiv und tut das, was das Programm ihm vorgibt. Während des Ausführens eines Hamster-Programms wird der Editor-Bereich ausgegraut, d.h. es können während der Ausführung eines Programms keine Änderungen am Sourcecode durchgeführt werden.

6.6.2 Stoppen eines Hamster-Programms

Die Ausführung eines Hamster-Programms kann durch Anklicken des „Stopp“-Buttons (21. Button der Toolbar von links) jederzeit abgebrochen werden.

6.6.3 Pausieren eines Hamster-Programms

Möchten Sie ein in Ausführung befindliches Programm (kurzfristig) anhalten, können Sie dies durch Anklicken des „Pause“-Buttons (20. Button der Toolbar von links) tun. Wenn Sie anschließend auf den „Start/Fortsetzen“-Button klicken, wird die Programmausführung fortgesetzt.

6.6.4 Während der Ausführung eines Hamster-Programms

Treten bei der Ausführung eines Programms Laufzeitfehler auf, 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. Weiterhin öffnet sich das Konsolen-Fenster, in dem ebenfalls die Fehlermeldung ausgegeben wird.

Während der Ausführung eines Hamster-Programms ist es durchaus noch möglich, das Territorium umzugestalten, also bspw. neue Körner und Mauern zu platzieren.

6.6.5 Einstellen der Geschwindigkeit

Mit dem Schieberegler ganz rechts in der Toolbar 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 flitzt der Hamster durchs Territorium.

6.6.6 Wiederherstellen eines Hamster-Territoriums

Beim Testen eines Programms recht hilfreich ist der „Rücksetzen“-Button (19. Button der Toolbar von links). Sein Anklicken bewirkt, dass das Hamster-Territorium in den Zustand zurückversetzt wird, den es vor dem letzten Start eines Programms inne hatte.

Über den „Komplett Zurücksetzen“-Button (23. Button der Toolbar von links) ist eine Rücksetzung des Territoriums in den Zustand möglich, der beim Öffnen des Hamster-Simulators gültig war. Sollte es irgendwann einmal bei der Benutzung des Hamster-Simulators zu unerklärlichen Fehlern kommen, ist es mit Hilfe dieses Buttons möglich, den Hamster-Simulator zu reinitialisieren.

6.7 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-Simulators 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.

Die Funktionen des Debuggers sind eng mit den Funktionen zur Programmausführung verknüpft. Sie finden die Funktionen im Menü „Simulation“. Es bietet sich jedoch an, die entsprechenden Buttons der Toolbar zu verwenden. Neben dem „Start/Fortsetzen“-, dem „Pause“- und dem „Stopp“-Button gehören die zwei Buttons „Schrittweise Ausführung“ und „Ablaufverfolgung“ zu den Debugger-Funktionen (siehe auch Abbildung 6.18).

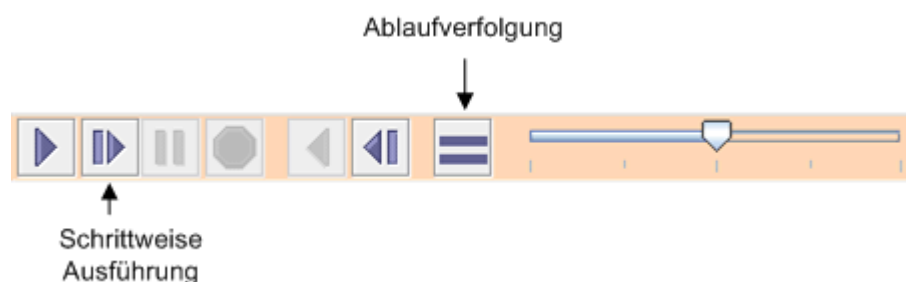


Abb. 6.18: Debugging-Buttons der Toolbar

6.7.1 Beobachten der Programmausführung

Durch Anklicken des Buttons „Ablaufverfolgung“ (24. Button der Toolbar von links) aktivieren bzw. (bei erneuten Anklicken) deaktivieren Sie die Ablaufverfolgung des Debuggers. Bei der Aktivierung öffnet sich dazu das Debugger-Fenster (siehe Abbildung 6.19). Dieses können Sie auch über das Menü „Fenster“ sichtbar bzw. unsichtbar machen.

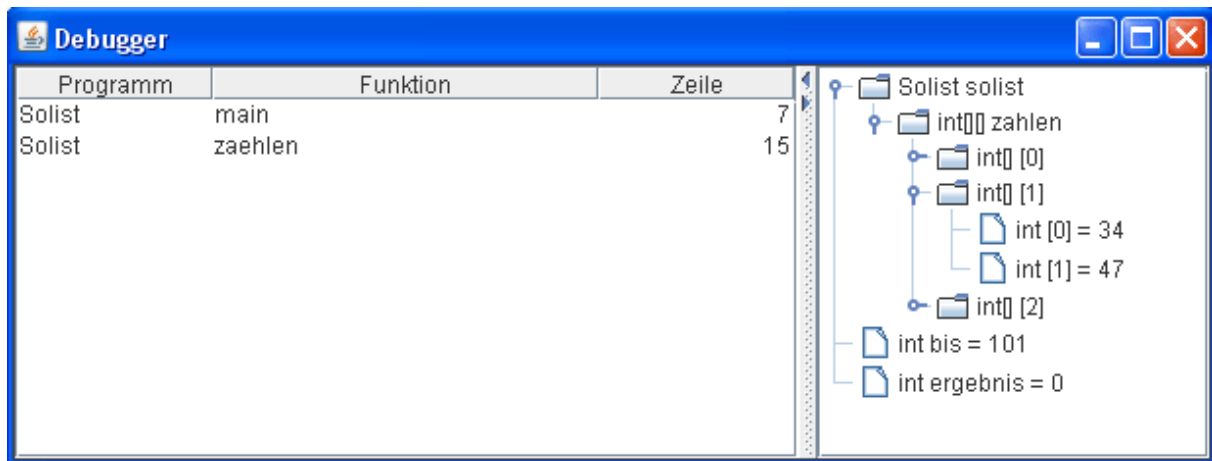


Abb. 6.19: Debugger-Fenster

Ist die Ablaufverfolgung aktiviert, wird bei der Ausführung des Programms im Editor-Bereich der Befehl (bzw. die entsprechende Zeile), der als nächstes ausgeführt wird, blau markiert. Bei einem Prozedur- bzw. Funktionsaufruf wird in die entsprechende Funktion gesprungen. Weiterhin werden im Debugger-Fenster der aktuelle Stack der Funktionsaufrufe (Name der Funktion und aktuelle Position der Ausführung der Funktion) sowie die aktuelle Belegung der Variablen dargestellt.

Im linken Bereich des Debugger-Fensters werden Informationen zu den aktiven Funktionen angezeigt, und zwar jeweils der Name der Funktion und die aktuelle Position der Ausführung der Funktion. Ganz unten erscheint die aktuell aktive Funktion, darüber gegebenenfalls die Funktion, die diese Funktion aufgerufen hat, usw. Ganz oben steht also immer die `main`-Funktion.

Im rechten Bereich des Debugger-Fensters werden die aktiven Variablen und ihre aktuellen Werte angezeigt. Die Darstellung erfolgt dabei in einem Elementbaum, d.h. bei komplexen Variablen, wie Arrays, können Sie durch Anklicken des Symbols vor dem Variablennamen die Komponenten einsehen.

Auch während die Ablaufverfolgung aktiviert ist, können Sie die Programmausführung durch Anklicken des „Pause“-Buttons anhalten und durch anschließendes Anklicken des „Start/Fortsetzen“-Buttons wieder fortfahren lassen.

Auch die Geschwindigkeit der Programmausführung lässt sich mit dem Schieberegler anpassen.

6.7.2 Schrittweise Programmausführung

Mit dem Toolbar-Button „Schrittweise Ausführung“ (19. Button der Toolbar von links) ist es möglich, ein Programm schrittweise, d.h. Anweisung für Anweisung auszuführen. Immer, wenn Sie den Button anklicken, wird die nächste Anweisung (bzw. Zeile) ausgeführt.

Sie können das Programm mit der schrittweisen Ausführung starten. Sie können jedoch auch zunächst das Programm normal starten, dann pausieren und ab der aktuellen Anweisung schrittweise ausführen. Eine normale Programmweiterführung ist jederzeit durch Klicken des „Start“-Buttons möglich.

6.7.3 Breakpoints

Wenn Sie das Programm an einer bestimmten Stelle anhalten möchten, können Sie in der entsprechenden Zeile einen so genannten Breakpoint setzen. Führen Sie dazu vor oder während der Programmausführung im Editor-Bereich mit der Maus einen Doppelklick auf die entsprechende Zeilennummer aus. Breakpoints werden durch violett hinterlegte Zeilennummern dargestellt (siehe Abbildung 6.20). Ein Doppelklick auf einen Breakpoint löscht den Breakpoint wieder. Über der Spalte mit den Zeilennummern lässt sich auch durch Klicken der rechten Maustaste ein Popup-Menü aktivieren, das als Funktionen das Setzen bzw. Entfernen von Breakpoints bzw. das gleichzeitige Löschen aller Breakpoints bietet.

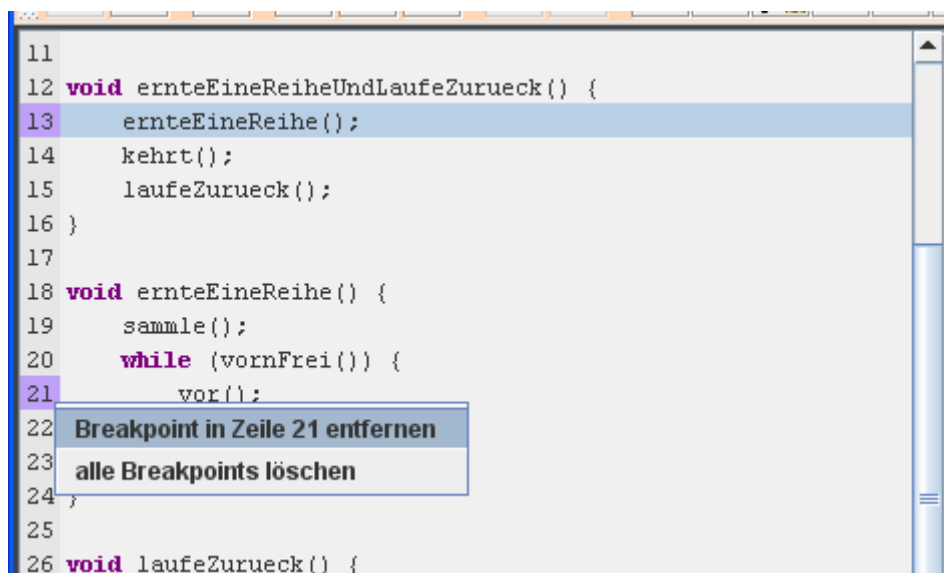


Abb. 6.20: Breakpoints

Wenn Sie ein Programm mit Breakpoints ausführen, wird die Ausführung jedesmal pausiert, wenn eine Zeile mit einem Breakpoint erreicht wird (unabhängig davon, ob die Ablaufverfolgung eingeschaltet ist). Durch Drücken des „Start/Fortsetzen“-Buttons kann die Ausführung des Programms fortgesetzt werden.

6.7.4 Debugger-Fenster

Das Debugger-Fenster wird automatisch bei Aktivierung der Ablaufverfolgung sichtbar gemacht. Über das Menü „Fenster“ lässt es sich jedoch auch explizit sichtbar bzw. unsichtbar machen. Das Fenster besteht aus einem rechten und einem linken Teil innerhalb einer Split-Pane. Der Inhalt des Debugger-Fensters wird nur dann aktualisiert, wenn die Ablaufverfolgung aktiviert ist.

Im linken Bereich des Fensters werden die aktuell aufgerufenen Funktionen und die jeweilige Zeilennummer dargestellt, in der sich die Programmausführung innerhalb der Funktion gerade befindet. Ganz oben steht dabei immer die main-Funktion, ganz unten die zuletzt aufgerufene Funktion.

Im rechten Bereich werden Variablen und deren aktuelle Werte dargestellt. Im Normalfall sind das genau die Variablen, die in der zuletzt aufgerufenen Funktion (also der im linken Bereich ganz unten stehenden Funktion) gültig sind. Die Darstellung erfolgt dabei in einem Elementbaum. Bei strukturierten Variablen kann durch Mausklick auf das Symbol vor dem Variablennamen die Struktur weiter geöffnet (bzw. wieder geschlossen) werden.

Im pausierten Zustand ist es möglich, sich auch die Werte lokaler Variablen anderer Funktionsinkarnationen anzuschauen. Das ist möglich, indem man im linken Bereich auf den entsprechenden Funktionsnamen klickt.

6.8 Konsole

Die Konsole ist ein zusätzliches Fenster, das bei bestimmten Aktionen automatisch geöffnet wird, sich aber über das Menü „Fenster“ auch explizit öffnen bzw. schließen lässt (siehe Abbildung 6.20).

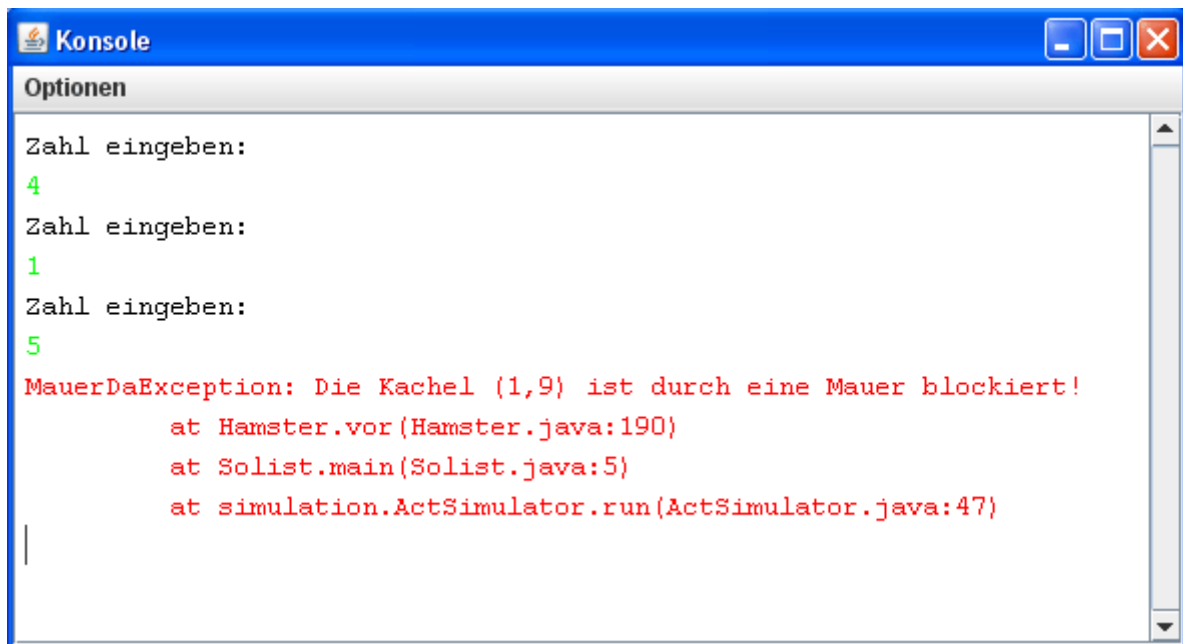


Abb. 6.20: Konsole

Die Konsole ist für die Java-Eingabe mittels `System.in` und die Ausgabe mittels `System.out` und `System.err` zuständig. Wird in Ihrem Programm bspw. der Befehl `System.out.println(„hallo“);` ausgeführt, wird die Zeichenkette `hallo` in der Konsole ausgegeben. `System.out` und `System.err` unterscheiden sich durch eine schwarze (out) bzw. eine rote (err) Ausgabe des entsprechenden Textes. Auch Fehlermeldungen des Hamster-Simulators erscheinen in der Konsole (bspw. Laufzeitfehler, wenn der Hamster gegen eine Mauer rennt).

Ein Eingabebefehl via `System.in` blockiert das Hamster-Programm so lange, bis der Nutzer eine entsprechende Eingabe in der Konsole getätigt und im Allgemeinen durch Drücken der Enter-Taste abgeschlossen hat. Die folgende Funktion `readInt` erwartet bspw. vom Nutzer die Eingabe einer Zahl in der Konsole und liefert den entsprechenden Wert an das Hamster-Programm. Wenn der Nutzer in der Konsole den Wert 4 eingibt, hüpfet der Hamster vier Felder nach vorne (insofern er nicht gegen eine Mauer rennt).

```
void main() {
    System.out.println("Zahl eingeben: ");
    int zahl = readInt();
    while (zahl > 0) {
        vor();
        zahl = zahl - 1;
    }
}

// Einlesen eines int-Wertes
int readInt() {
```

```

try {
    java.io.BufferedReader input =
        new java.io.BufferedReader(
            new java.io.InputStreamReader(System.in));
    String eingabe = input.readLine();
    return new Integer(eingabe);
} catch (Throwable exc) {
    return 0;
}
}

```

Die Konsole enthält in der Menüleiste ein Menü namens „Optionen“. Hierin finden sich drei Menü-Items, über die es möglich ist, den aktuellen Inhalt der Konsole zu löschen, den aktuellen Inhalt der Konsole in einer Datei abzuspeichern bzw. die Konsole zu schließen.

7 Beispielprogramme

Dem Hamster-Simulator-Light sind drei Beispielprogramme beigelegt, die Sie über das Menü „Programm“ laden können (Menü-Item „Laden...“). Prinzipiell können Sie jedoch auch alle Beispielprogramme des Buches „Programmieren spielend gelernt mit dem Java-Hamster-Modell“ unmittelbar laden, compilieren und ausführen. Diese Beispielprogramme können über die Website www.java-hamster-modell.de geladen werden. Allerdings können die dort gespeicherten Hamster-Territorien nicht verwendet werden. Sie wurden mit dem „normalen“ Hamster-Simulator erstellt und sind nicht kompatibel zu diesem Hamster-Simulator-Light.

7.1 Feld abgrasen

Die Aufgabe lautet: Der Hamster steht auf Kachel (0/0) mit Blickrichtung Osten in einem rechteckigen Territorium ohne innere Mauern. Auf den Kacheln können 0, 1 oder mehrere Körner liegen. Der Hamster soll alle Körner einsammeln und dann stehen bleiben.

Dateiname: FeldAbgrasen.ham

Beispiel-Territorien: FeldAbgrasen1.ter und FeldAbgrasen2.ter

```
void main() {
    ernteEineReiheUndLaufeZurueck();
    linksUm();
    while (vornFrei()) {
        vor();
        linksUm();
        ernteEineReiheUndLaufeZurueck();
        linksUm();
    }
}

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

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

void laufeZurueck() {
    while (vornFrei()) {
        vor();
    }
}
```

```

}

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

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

void kehrt() {
    linksUm();
    linksUm();
}

```

7.2 *Berg erklimmen*

Die Aufgabe lautet: Der Hamster steht mit Blickrichtung Osten vor einem Berg mit regelmäßigen jeweils eine Kachel hohen Stufen. Er bekommt die Aufgabe, den Berg zu erklimmen und auf dem Gipfel stehen zu bleiben.

Dateiname: BergErklimmen.ham

Beispiel-Territorien: Berg1.ter und Berg2.ter

```

void main() {
    laufeZumBerg();
    erklimmeGipfel();
}

void laufeZumBerg() {
    while (vornFrei()) {
        vor();
    }
}

void erklimmeGipfel() {
    do {
        erklimmeEineStufe();
    } while (!vornFrei());
}

void erklimmeEineStufe() {
    linksUm();
    vor();
    rechtsUm();
    vor();
}

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

```



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

7.3 *Berg erklimmen (objektorientierte Variante)*

Der Hamster hat übrigens auch einen Namen, nämlich `willi`. Darüber ist es möglich, Befehle an den Hamster auch in einer objektorientierten Notation zu verwenden:

```
willi.vor();
willi.linksUm();
willi.nimm();
willi.gib();
if (willi.vornFrei()) ...
while (willi.kornDa()) ...
while (!willi.maulLeer()) ...
```

Die objektorientierte Notation kann auch bei selbst definierten Prozeduren bzw. Funktionen verwendet werden:

```
void rechtsUm() {
    willi.kehrt();
    willi.linksUm();
}

void kehrt() {
    willi.linksUm();
    willi.linksUm();
}
```

Lösen wir die zweite Beispielaufgabe nun einmal in dieser objektorientierten Variante. Aufgabe: Der Hamster steht mit Blickrichtung Osten vor einem Berg mit regelmäßigen jeweils eine Kachel hohen Stufen. Er bekommt die Aufgabe, den Berg zu erklimmen und auf dem Gipfel stehen zu bleiben.

Dateiname: `OOBergErklimmen.ham`

Beispiel-Territorien: `Berg1.ter` und `Berg2.ter`

```
void main() {
    willi.laufeZumBerg();
    willi.erklimmeGipfel();
}

void laufeZumBerg() {
    while (willi.vornFrei()) {
```

```

        willi.vor();
    }
}

void erklimmeGipfel() {
    do {
        willi.erklimmeEineStufe();
    } while (!willi.vornFrei());
}

void erklimmeEineStufe() {
    willi.linksUm();
    willi.vor();
    willi.rechtsUm();
    willi.vor();
}

void rechtsUm() {
    willi.kehrt();
    willi.linksUm();
}

void kehrt() {
    willi.linksUm();
    willi.linksUm();
}

```

8 Literatur zum Erlernen der Programmierung

Der Hamster-Simulator-Light ist ein Werkzeug, das Programmierern beim praktischen Erlernen der (imperativen) Programmierung hilft. Der Simulator basiert dabei auf dem Buch „**Programmieren spielend gelernt mit dem Java-Hamster-Modell**“ von Dietrich Boles, erschienen im Vieweg+Teubner-Verlag. In diesem Buch werden die grundlegenden Konzepte der Programmierung anhand des Java-Hamster-Modells vorgestellt. Das Buch geht langsam und schrittweise vor. Es enthält viele Beispiele und auch eine Reihe von Aufgaben, an denen Sie unmittelbar ausprobieren können, ob Sie die vorgestellten Konzepte nicht nur verstanden haben, sondern auch selbstständig beim Lösen von Problemen einsetzen können. Das Buch ist insbesondere für solche Programmieranfänger zu empfehlen, die sich beim Erlernen der Programmierung schwer tun. Mehr Informationen und auch Links zu Leseproben gibt es auf der Java-Hamster-Website www.java-hamster-modell.de.

Es gibt heutzutage unzählige Lehrbücher zu Java und es ist schwer zu sagen, für wen welches Buch das geeignetste ist. Viele Bücher stehen bei Amazon oder Google-Books zumindest auszugsweise online zur Verfügung und ich kann nur empfehlen, über diese Online-Angebote selbst einmal in die Bücher hineinzuschnuppern. Neben dem Java-Hamster-Buch kann ich die beiden weiteren Bücher insbesondere für Programmieranfänger empfehlen:

- Ratz, D., Scheffler, J., Seese, D. und Wiesenberger, J.: „Grundkurs Programmieren in Java“, Hanser-Verlag.
- Heinisch, C., Müller, F. und Goll, J.: „Java als erste Programmiersprache“, Vieweg+Teubner.

Zum Java-Hamster-Modell gibt es zwei weitere Bücher:

- Boles, D. und Boles, C.: Objektorientierte Programmierung spielend gelernt mit dem Java-Hamster-Modell, Vieweg+Teubner
- Boles, D.: Parallele Programmierung spielend gelernt mit dem Java-Hamster-Modell – Programmierung mit Java-Threads, Vieweg+Teubner

In diesen beiden Büchern wird das Hamster-Modell erweitert und dazu genutzt, mit spielerischen Elementen eine Einführung in die objektorientierte und parallele Programmierung mit Java zu geben. In diesen Büchern gibt es nicht mehr nur einen Hamster. Vielmehr können weitere Hamster erzeugt werden, die dann versuchen, gemeinsam gegebene Probleme zu lösen. Das Entwickeln und Ausführen objektorientierter bzw. paralleler Hamster-Programme ist mit diesem Hamster-Simulator-Light nicht möglich. Hierfür benötigen Sie den „normalen“ Hamster-Simulator, den Sie über die Website www.java-hamster-modell.de laden können.