

Informatik *Java-Hamster-Modell*

1.1 Definition des Begriffs Algorithmus

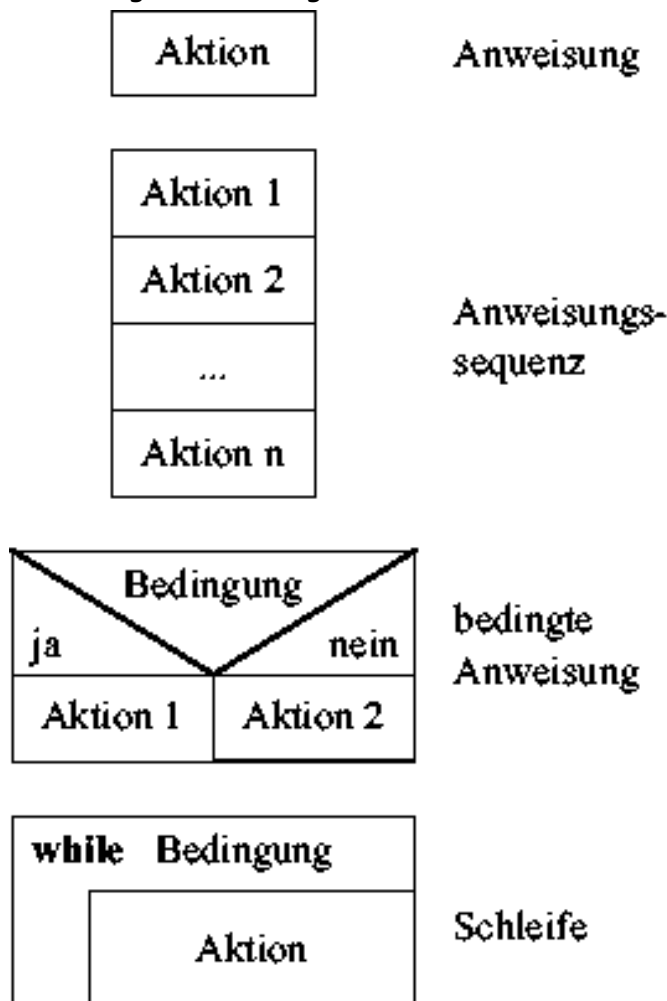
Anleitungen, z. B. Kochrezepte, werden von Menschen ausgeführt, um unter bestimmten Voraussetzungen zu einem bestimmten Ergebnis zu gelangen. Genauso wie Menschen benötigen auch Computer Arbeitsanleitungen, um Probleme zu lösen. Arbeitsanleitungen für einen Computer bezeichnet man als Algorithmen. Sie bestehen aus Anweisungen, können bedingte Anweisungen und Schleifen enthalten und operieren auf vorgegebenen Materialien, den Daten. Sie unterscheiden sich gegenüber Kochrezepten jedoch darin, dass sie wesentlich exakter formuliert sein müssen, da Computer keine Intelligenz besitzen, um mehrdeutige Formulierungen selbständig interpretieren zu können.

Damit kann der Begriff Algorithmus folgendermaßen definiert werden: Ein Algorithmus ist eine Arbeitsanleitung zum Lösen eines Problems bzw. einer Aufgabe, die so präzise formuliert ist, dass sie von einem Computer ausgeführt werden kann.

1.2 Struktogramme

Struktogramme (Nassi-Shneiderman-Diagramme) bieten eine graphische Notation zur Darstellung von Algorithmen. Die wichtigsten Elemente, aus denen sich Struktogramme zusammensetzen, sind Abbildung 1 zu entnehmen

Abbildung 1: Struktogramme



1.3 Programme

Als Programm wird ein in einer Programmiersprache formulierter Algorithmus bezeichnet. Die Formulierung von Programmen erfolgt in sehr konkreter und eingeschränkter Form:

Programme werden im exakt definierten und eindeutigen Formalismus einer Programmiersprache verfasst.

Daten, auf denen Programme operieren, unterliegen einer festgelegten Darstellungsform.

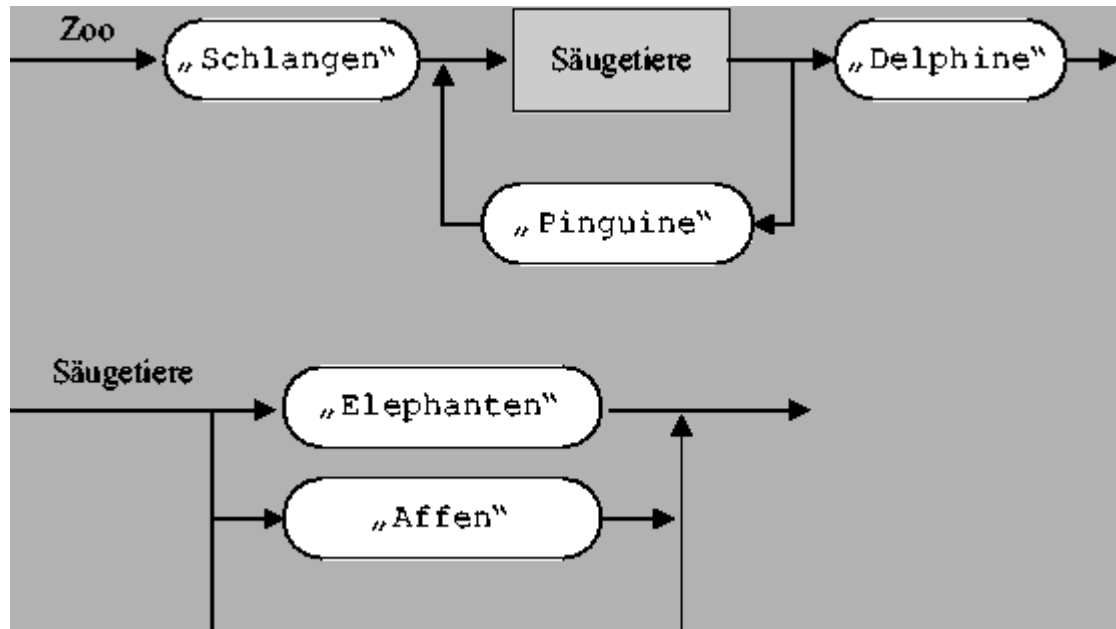
Unter Programmieren versteht man das Erstellen von Programmen. Zuständig für die Entwicklung von Programmen ist der Programmierer. Die Formulierung von Programmen erfolgt in der Regel unter Verwendung der uns bekannten Zeichen wie Buchstaben, Ziffern und Sonderzeichen. Eine Programmbeschreibung wird auch als Programmcode, Quellcode oder Source-Code bezeichnet. Der Programmcode selbst kann im Allgemeinen noch nicht direkt von einem Computer ausgeführt werden, er muss zuvor noch mit Hilfe eines Compilers in eine maschinenverständliche Form - ein ausführbares Programm - transformiert werden. Die Programmausführung wird durch den Aufruf des ausführbaren Programms gestartet.

Informatik *Java-Hamster-Modell*

2. Syntaxdiagramme

Bei den Syntaxdiagrammen handelt es sich um eine graphische und daher sehr übersichtliche Notation zur Definition der Syntax einer Programmiersprache. Syntaxdiagramme sind folgendermaßen definiert:

Abbildung 2: Wegeplan im Zoo als Syntaxdiagramm



Mögliche Bildsequenzen sind zum Beispiel:

- Schlangen Delphine
- Schlangen Elephanten Pinguine Delphine
- Schlangen Elephanten Pinguine Affen Delphine
- Schlangen Elephanten Pinguine Elephanten Pinguine Delphine

Dahingegen sind folgende Bildsequenzen nicht möglich:

- Elephanten Delphine (weil der Fotograf anfangs auf jeden Fall zuerst am Schlangengehege vorbeikommt)
- Schlangen Pinguine (weil der Fotograf am Ende seines Zoobesuchs auf jeden Fall am Delphingehege vorbeikommt)
- Schlangen Elephanten Affen Delphine (weil der Fotograf zwischen dem Besuch des Elefanten- und Affengeheges auf jeden Fall einmal am Pinguingehege vorbeigehen muss)
- Schlangen Pinguine Schlangen Delphine (weil der Fotograf nur am Anfang seines Zoobesuchs am Schlangengehege vorbeikommt)

Informatik *Java-Hamster-Modell*

3. Analyse

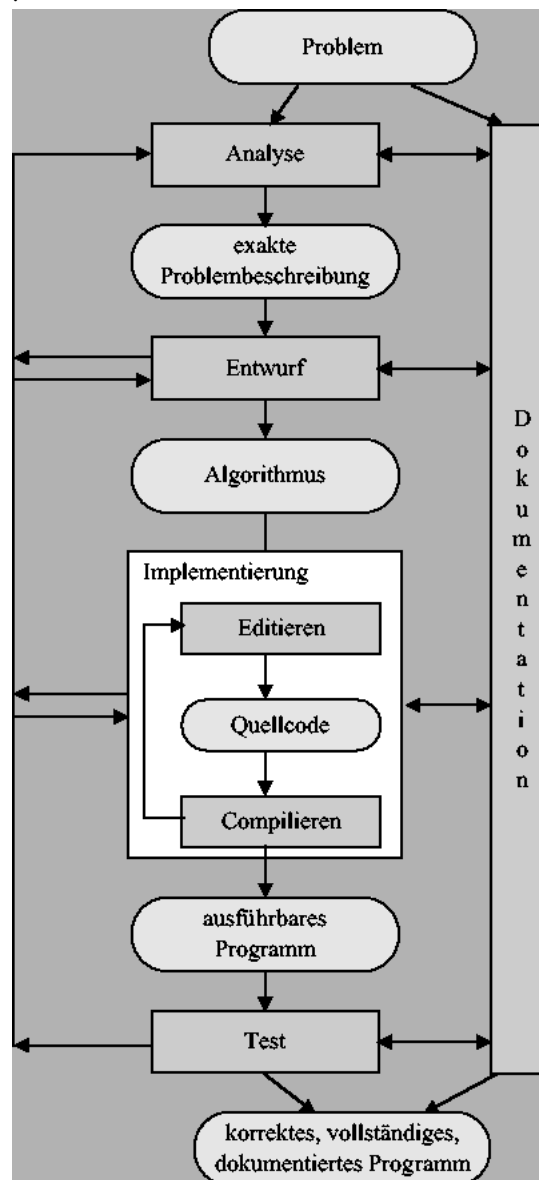
In der Analysephase wird das zu lösende Problem bzw. das Umfeld des Problems genauer untersucht. Insbesondere folgende Fragestellungen sollten bei der Analyse ins Auge gefasst und auch mit anderen Personen diskutiert werden:

Ist die Problemstellung exakt und vollständig beschrieben?

Was sind mögliche Initialzustände bzw. Eingabewerte (Parameter) für das Problem?

Welches Ergebnis wird genau erwartet, wie sehen der gewünschte Endzustand bzw. die gesuchten Ausgabewerte aus?

Abbildung 3: Programmentwicklungsphasen



Informatik *Java-Hamster-Modell*

3.1 Entwurf

Nachdem die Problemstellung präzisiert und verstanden worden ist, muss in der Entwurfsphase ein Algorithmus zum Lösen des Problems entworfen werden. Der Entwurfsprozess kann im Allgemeinen nicht mechanisch durchgeführt werden, vor allen Dingen ist er nicht automatisierbar. Vielmehr kann man ihn als einen kreativen Prozess bezeichnen, bei dem Auffassungsgabe, Intelligenz und vor allem Erfahrung des Programmierers eine wichtige Rolle spielen. Diese Erfahrung kann insbesondere durch fleißiges Üben erworben werden.

Trotzdem können die folgenden Ratschläge beim Entwurf eines Algorithmus nützlich sein:

Du solltest Dich nach bereits existierenden Lösungen für vergleichbare Probleme erkundigen und diese nutzen.

Du solltest Dich nach allgemeineren Problemen umschauen, und überprüfen, ob Dein Problem als Spezialfall des allgemeinen Problems behandelt werden kann.

Du solltest versuchen, das Problem in einfachere Teilprobleme aufzuteilen. Wenn eine Aufteilung möglich ist, solltest Du den hier skizzierten Programmentwicklungsprozess zunächst für die einzelnen Teilprobleme anwenden und anschließend die Lösungen der einzelnen Teilprobleme zu einer Lösung für das Gesamtproblem zusammensetzen.

3.2 Implementierung

Der Entwurf eines Algorithmus sollte unabhängig von einer konkreten Programmiersprache erfolgen. Die anschließende Überführung des Algorithmus in ein in einer Programmiersprache verfasstes Programm wird als Implementierung bezeichnet. Anders als der Entwurf eines Algorithmus ist die Implementierung in der Regel ein eher mechanischer Prozess.

Die Implementierungsphase besteht selbst wieder aus zwei Teilphasen:

Editieren: Zunächst wird der Programmcode mit Hilfe eines Editors eingegeben und in einer Datei dauerhaft abgespeichert.

Kompilieren: Anschließend wird der Programmcode mit Hilfe eines Compilers auf syntaktische Korrektheit überprüft und - falls keine Fehler vorhanden sind - in eine ausführbare Form (ausführbares Programm) überführt. Liefert der Compiler eine Fehlermeldung, so muss in die Editierphase zurückgesprungen werden.

Ist die Kompilation erfolgreich, kann das erzeugte Programm ausgeführt werden. Je nach Sprache und Compiler ist die Ausführung entweder mit Hilfe des Betriebssystems durch den Rechner selbst oder aber durch die Benutzung eines Interpreters möglich.

Informatik *Java-Hamster-Modell*

4.1 Programmieren lernen

Das Erlernen einer Programmiersprache ist in der Tat nicht besonders schwierig. Was sehr viel schwieriger ist, ist das Programmieren lernen, d.h. das Erlernen des Programmentwicklungsprozesses:

Wie komme ich von einem gegebenen Problem hin zu einem Programm, das das Problem korrekt und vollständig löst?

Wie finde ich eine Lösungsidee bzw. einen Algorithmus, der das Problem löst?

Wie setze ich den Algorithmus in ein Programm um?

4.2 Sinn und Zweck des Hamster-Modells

Das Hamster-Modell ist ein einfaches Modell, bei dem nicht primär das Erlernen einer Programmiersprache sondern das Erlernen der Programmierung im Vordergrund steht, d.h. das Erlernen grundlegender Programmierkonzepte und das Erlernen des Problemlösungs- bzw. des Programmentwicklungsprozesses. Der Lernprozess wird im Hamster-Modell durch spielerische Elemente unterstützt. Der Programmierer steuert einen virtuellen Hamster durch eine virtuelle Landschaft und lässt ihn bestimmte Aufgaben lösen.

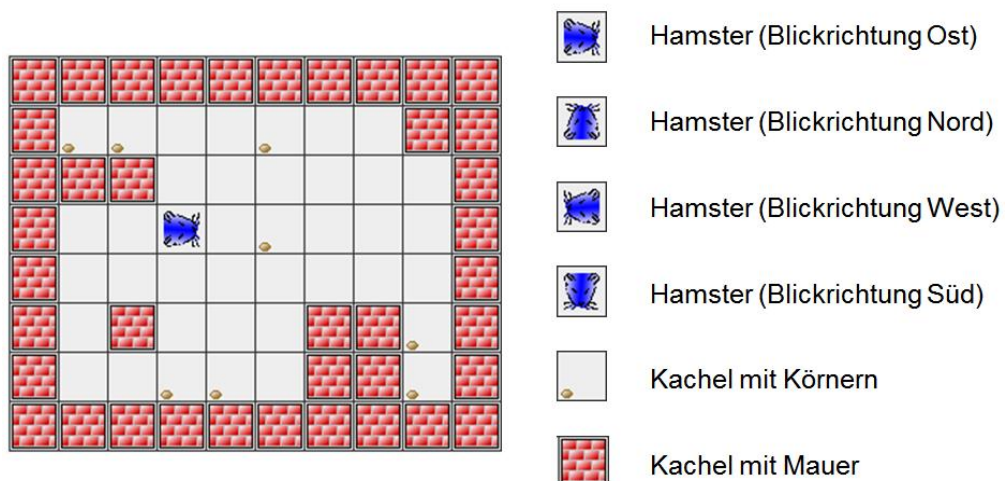
4.3 Komponenten des Hamster-Modells

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

4.4 Landschaft

Die Welt, in der der Hamster lebt, wird durch eine gekachelte Ebene repräsentiert. Abbildung 4 zeigt eine typische Hamsterlandschaft - auch Hamster-Territorium genannt - plus Legende. Die Größe der Landschaft, d.h. die Anzahl der Kacheln, ist dabei nicht explizit vorgegeben. Die Landschaft kann prinzipiell unendlich groß sein.

Abbildung 4: Komponenten des Hamster-Modells



Informatik *Java-Hamster-Modell*

Auf einzelnen Kacheln können ein oder mehrere Körner liegen. Kacheln, auf denen sich Körner befinden, sind in den Landschaftsskizzen durch ein spezielles Symbol gekennzeichnet. Dabei sagt das Symbol nur aus, dass auf der Kachel mindestens ein Korn liegt. Die genaue Anzahl an Körnern auf einem Feld geht aus der Landschaftsskizze nicht direkt hervor.

Auf den Kacheln der Hamsterlandschaft können weiterhin auch Mauern stehen, das 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

4.5 Hamster

Im imperativen Hamster-Modell existiert immer genau ein Hamster. Der Hamster steht dabei auf einer der Kacheln der Hamsterlandschaft. 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 Zeichen repräsentiert.

Wenn der Hamster auf einer Kachel steht, auf der auch Körner liegen, wird in der Skizze das Kornsymbol nicht angezeigt, d.h. es kann aus der Skizze nicht direkt abgelesen werden, ob sich der Hamster auf einer Körnerkachel befindet.

Körner können sich nicht nur auf einzelnen Kacheln, sondern auch im Maul des Hamsters befinden. Ob der Hamster Körner im Maul hat und wenn ja, wie viele, ist ebenfalls nicht direkt aus der Landschaftsskizze ersichtlich.

Mit Hilfe bestimmter Befehle, die unten genauer erläutert werden, kann ein Programmierer den Hamster durch eine gegebene Hamsterlandschaft steuern. Der Hamster kann dabei von Kachel zu Kachel hüpfen, er kann sich drehen, Körner fressen und Körner wieder ablegen. Du kannst dich den Hamster quasi als einen virtuellen Prozessor vorstellen, der im Gegensatz zu realen Prozessoren (zunächst) keine arithmetischen und logischen Operationen ausführen kann, sondern in der Lage ist, mit einem kleinen Grundvorrat an Befehlen eine Hamsterlandschaft zu „erforschen“.

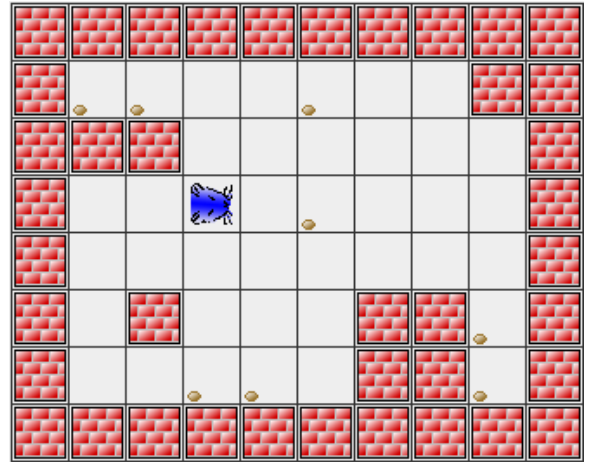
4.6 Hamsteraufgaben

Dir als Hamsterprogrammierer werden nun bestimmte Aufgaben gestellt, die Du durch die Steuerung des Hamsters durch eine Landschaft zu lösen hast. Diese Aufgabe wird im folgenden Hamsteraufgabe und die entstehenden Lösungsprogramme Hamsterprogramme genannt. Zusätzlich zu einer Aufgabe werden dabei zunächst bestimmte Hamsterlandschaften fest vorgegeben, in denen der Hamster die Aufgabe zu lösen hat.

Beispiel für eine Hamsteraufgabe: Gegeben sei die Landschaft in Abbildung 5. Der Hamster soll zwei beliebige Körner fressen.

Informatik *Java-Hamster-Modell*

Abbildung 5: Hamsterlandschaft zur Beispiel-Hamsteraufgabe



Später werden die Landschaften zu einer Hamsteraufgabe nicht mehr fest vorgegeben. Sie werden dann nur noch durch vorgegebene Eigenschaften charakterisiert, d.h. der Hamster wird in der Lage sein, eine bestimmte Aufgabe in unterschiedlichen (aber gleichartigen) Landschaft zu lösen.

5.1 Hamsterbefehle

Die Aufgabe eines Hamsterprogrammierers besteht darin, den Hamster durch eine Landschaft zu steuern, um dadurch gegebene Hamsteraufgaben 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();  
linksUm();  
nimm();  
gib();
```

Die Zeichenfolgen **vor**, **linksUm**, **nimm** und **gib** dürfen so und nur so geschrieben werden. In der Hamstersprache werden Klein- und Großbuchstaben unterschieden.

Die Zeichenfolgen **vor**, **linksUm**, **nimm** und **gib** stellen jeweils ein Token dar, d.h. die Zeichenfolgen müssen immer als Ganzes auftreten, sie dürfen nicht durch Trennzeichen (Leerzeichen, Tabulator, Zeilenwechsel) unterbrochen werden.

Vor, hinter und zwischen den runden Klammern können beliebig viele Trennzeichen stehen. Das Semikolon gehört zum Befehl dazu. Es darf nicht weggelassen werden.

5.2 Semantik

Während die Syntax das Vokabular und die Grammatik einer Programmiersprache definiert, wird durch die Semantik die Bedeutung eines syntaktisch korrekten Programmes angegeben, d.h. es wird festgelegt, was das Programm bewirkt. Im Folgenden wird dazu zunächst die Semantik der vier Grundbefehle des Hamster-Modells verbal beschrieben. Im Falle des Hamster-Modells bedeutet das, es wird definiert, wie der Hamster reagiert, wenn ihm ein Befehl ``mitgeteilt'' wird:

```
vor();      Der Hamster hüpft eine Kachel in seiner aktuellen Blickrichtung nach vorn.  
linksUm();  Der Hamster dreht sich auf der Kachel, auf der er gerade steht, um 90 Grad nach  
            links.
```


Informatik *Java-Hamster-Modell*

- 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.

Syntaktisch nicht korrekt sind folgende Befehle:

- n imm(); (kein Leerzeichen erlaubt)
- Gib(); (großes „G“ nicht korrekt)
- linksum(); (kleines „u“ nicht korrekt)
- vor() (Semikolon fehlt)
- gib; (Klammern fehlen)

Informatik *Java-Hamster-Modell*

6.0 Hamstern mit Scratch

Scratch ist eine Programmierumgebung bzw. Programmiersprache für echte Programmieranfänger. Anders als bei anderen Programmiersprachen müssen hier die Programmierer keinen textuellen Sourcecode schreiben. Vielmehr setzen sich Scratch-Programme aus graphischen Bausteinen zusammen (siehe Abbildung 6.1). Die Programmierumgebung unterstützt dabei das Erstellen von Programmen durch einfache Drag-and-Drop-Aktionen mit der Maus.

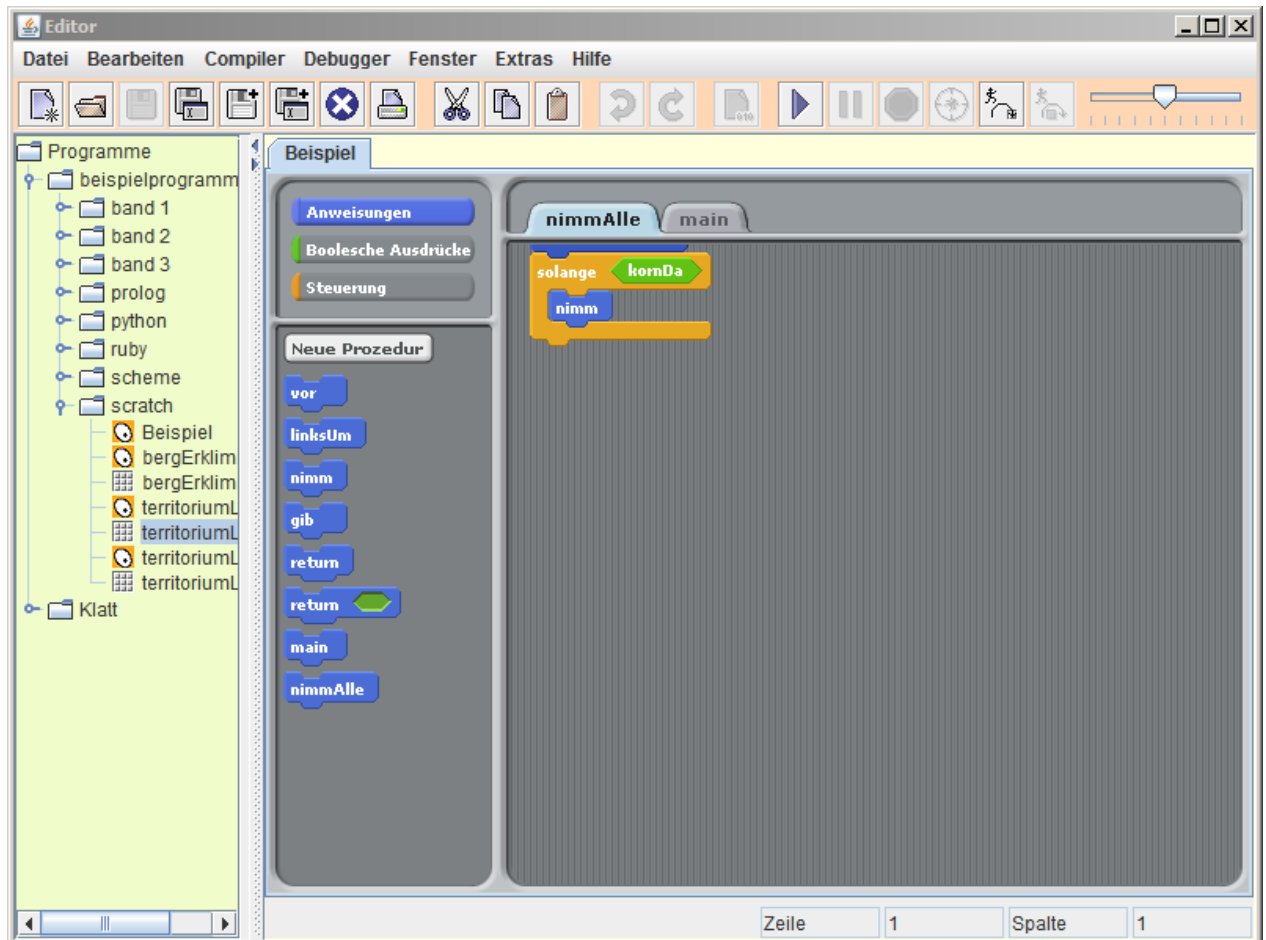


Abbildung 6.1: Scratch

Ein Vorteil von Scratch für Programmieranfänger ist, dass man keine syntaktischen Fehler machen kann und dass sich sehr schnell kleine Programme „zusammenklicken“ lassen.

6.1 Die vier Grundbefehle

- Die vier Grundbefehle des Hamsters (vor, linksUm, gib und nimm) sind als Anweisungsblöcke vordefiniert.

6.1.1 Ein erstes kleines Beispiel

Um einen ersten kleinen Eindruck vom Scratch-Hamster-Modell zu bekommen, öffne bitte im Ordner „beispielprogramme“ den Unterordner „scratch“ und lade dann das Programm „Beispiel“. Es erscheint das in Abbildung 6.1 dargestellte Scratch-Editorfenster. Weiterhin

Informatik *Java-Hamster-Modell*

ist das Territorium „territoriumLeeren“ zu öffnen. Führe das Programm durch Anklicken des Run-Buttons in der Toolbar aus: Der Hamster läuft bis zur nächsten Wand und sammelt dabei alle Körner ein.

6.2.0 Erstellen von Scratch-Programmen

Um ein neues Scratch-Hamster-Programm zu erstellen, klicke bitte in der Toolbar des Editor-Fensters auf den Neu-Button (1. Toolbar-Button von links). Er erscheint ein Fenster, in dem der Programmtyp ausgewählt werden muss. Wähle hier den Programmtyp „Scratch-Programm“. Nach dem Drücken des OK-Buttons erscheint das *Scratch-Editorfenster*. Es besteht aus drei Bereichen (siehe auch Abbildung 6.3): der *Kategorienauswahl*, der *Blockpalette* und dem *Programmbereich*.

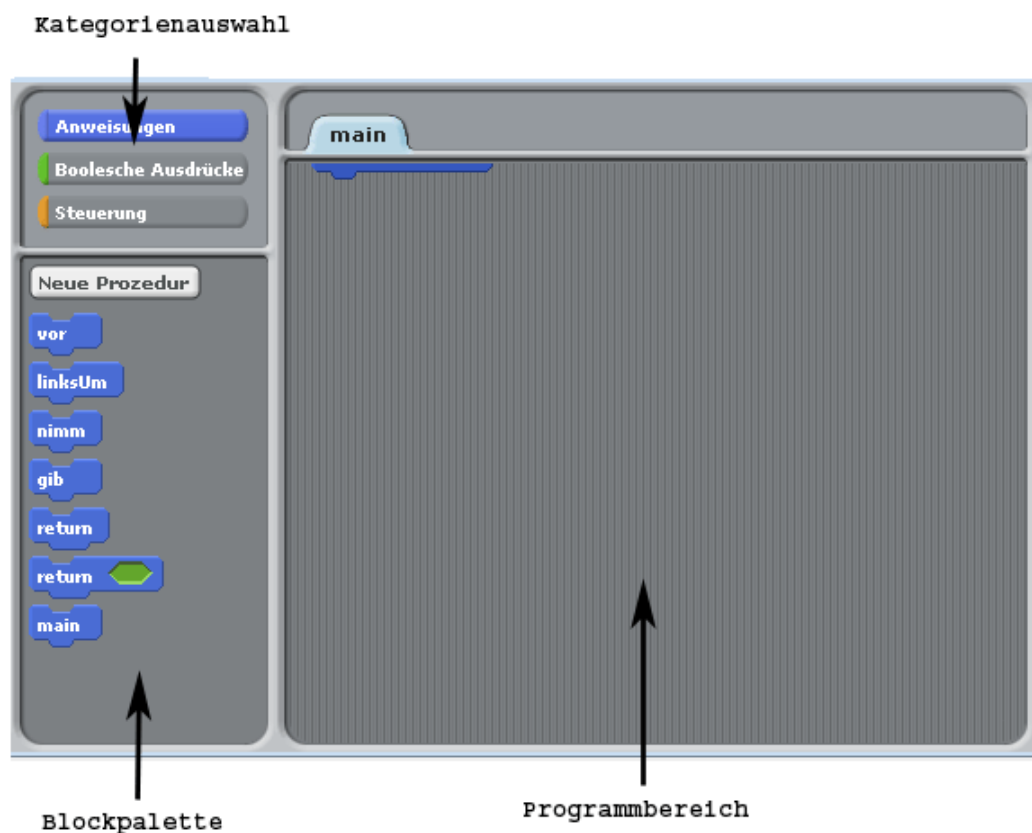


Abbildung 6.3: Scratch-Editorfenster

6.2.1 Kategorienauswahl

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

6.2.2 Blockpalette

In der Blockpalette werden die Blöcke der in der Kategorienauswahl aktuell ausgewählten Kategorie angezeigt. Diese – genauer gesagt Kopien hiervon – lassen sich per Drag-und-Drop

Informatik *Java-Hamster-Modell*

mit der Maus in den Programmbereich ziehen, um Programme zu erstellen. Wähle hierzu den entsprechenden Block, klicke ihn mit der Maus (linke Taste) an, ziehen Sie die Maus bei gedrückter linker Taste in den Programmbereich und lassen die Maustaste los.

Folgende Blöcke werden standardmäßig in der Blockpalette angezeigt

6.3.0 Anweisungsblöcke (Kategorie „Anweisungen“):

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

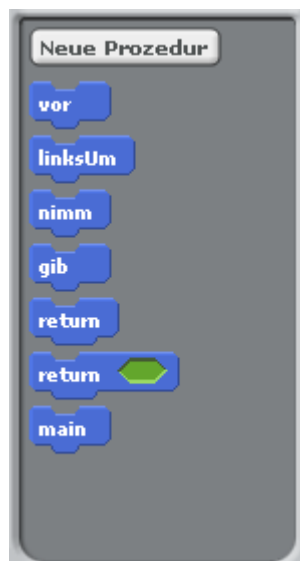


Abbildung 6.4: Blockpalette der Kategorie „Anweisungen“

6.3.1 Programmbereich

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

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

Informatik *Java-Hamster-Modell*

6.3.2 Anweisungssequenzen

Befindet sich im Programmbereich ein Anweisungsblock A und zieht man einen anderen Anweisungsblock B in die Nähe (ober- und unterhalb), dann erscheint irgendwann ein transparenter grauer Bereich (siehe Abbildung 6.7). Lässt man B nun los, schnappen Zapfen (unten an den Anweisungsblöcken) und Mulde (oben an den Anweisungsblöcken) der beiden Blöcke ein und A und B bilden nun eine Anweisungssequenz (in Scratch wird der Begriff „Stapel“ verwendet). Zieht man im Folgenden den obersten Block einer Anweisungssequenz im Programmbereich hin und her, wird automatisch der gesamte Block mit verschoben. Eine Anweisungssequenz kann man wieder trennen, indem man einen mittleren Block verschiebt. Darunter liegende Blöcke werden dann mit verschoben, darüber liegende Blöcke bleiben liegen und werden abgetrennt.

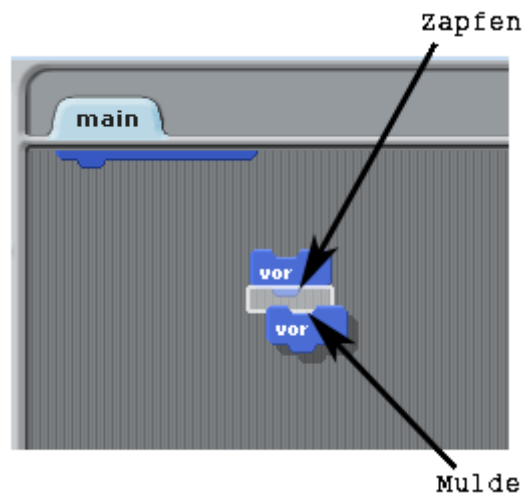


Abbildung 6.7: Bilden von Anweisungssequenzen

6.3.3 Programme

Scratch-Hamster-Programme starten immer durch Ausführung der main-Prozedur. Ausgeführt wird dabei die Anweisungssequenz, die mit der main-Prozedur verbunden ist. Hierzu besitzt die Prozedur am oberen Rand des Programmbereichs einen Zapfen (den sogenannten *Prozedurzapfen*). Hier muss man die entsprechende Anweisungssequenz andocken (siehe Abbildung 6.8).

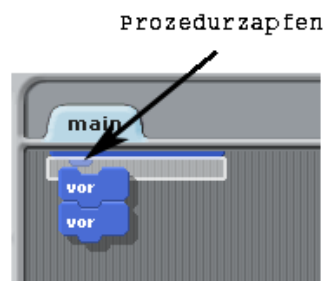


Abbildung 6.8: Definition von Prozeduren

Nun sind wir so weit, dass wir ein erstes kleines Scratch-Hamster-Programm selbst erstellen und testen können. Erstellen Sie zum Beispiel ein Programm, in dem sich der Hamster zwei Kacheln nach vorne bewegt, sich dann linksum dreht und wiederum zwei Kacheln nach vorne springt (siehe Abbildung 6.9).



Abbildung 6.9: Erstes kleines Scratch-Hamster-Programm

6.4 Steuerung

- Zur Programmsteuerung sind die if-Anweisung, die if-else-Anweisung, die while-Schleife und die do-while-Schleife als spezielle Anweisungsblöcke („falls“, „falls-sonst“, „solange“, „wiederhole-solange“) integriert. Diesen können Boolesche Ausdruck-Blöcke als Bedingungen und Anweisungsblöcke als innere Anweisungen zugeordnet werden.
- Steuerungsblöcke (Kategorie „Steuerung“):
 - falls: Repräsentiert die if-Anweisung.
 - falls-sonst: Repräsentiert die if-else-Anweisung.
 - solange <boolescher Ausdruck>: Repräsentiert die while-Schleife.
 - wiederhole-solange <boolescher Ausdruck>: Repräsentiert die do-while-Schleife.



Abbildung 6.6: Blockpalette der Kategorie „Steuerung“

6.5 Testbefehle (Boolesche Ausdrücke)

Informatik *Java-Hamster-Modell*

- Die drei Testbefehle des Hamsters (vornFrei, kornDa und maulLeer) sind als Boolesche-Ausdruck-Blöcke vordefiniert. Weiterhin gibt es die Boolesche-Ausdruck-Blöcke „true“ und „false“.
- Mittels entsprechender Boolesche-Ausdruck-Blöcke für die booleschen Operatoren „nicht“, „und“ und „oder“ können komplexe boolesche Ausdrücke erstellt werden. Als Operanden sind dabei beliebige andere Boolesche-Ausdruck-Blöcke einsetzbar.
- Boolesche-Ausdrucks-Blöcke (Kategorie „Boolesche Ausdrücke“):
 - vornFrei: Repräsentiert den Hamster-Testbefehl vornFrei.
 - kornDa: Repräsentiert den Hamster-Testbefehl kornDa.
 - maulLeer: Repräsentiert den Hamster-Testbefehl maulLeer.
 - wahr: Repräsentiert das boolesche Literal „true“.
 - falsch: Repräsentiert das boolesche Literal „false“.
 - <boolescher Ausdruck> und <boolescher Ausdruck>: Repräsentiert den booleschen Und-Operator.
 - <boolescher Ausdruck> oder <boolescher Ausdruck>: Repräsentiert den booleschen Oder-Operator.
 - nicht <boolescher Ausdruck>: Repräsentiert den booleschen Nicht-Operator.
 - Über den Button „Neue Funktion“ lassen sich neue boolesche Funktionen definieren.



Abbildung 6.5: Blockpalette der Kategorie „Boolesche Ausdrücke“

6.6 Steuerungsblöcke und Testbefehle anwenden

Informatik *Java-Hamster-Modell*

Die Blöcke, die die Kontrollstrukturen repräsentieren, lassen sich analog zu den Anweisungsblöcken handhaben. Sie besitzen jedoch als weitere Unterkomponenten einen Bedingungsbe-
reich (grün) und einen Innenbereich (siehe Abbildung 6.10).

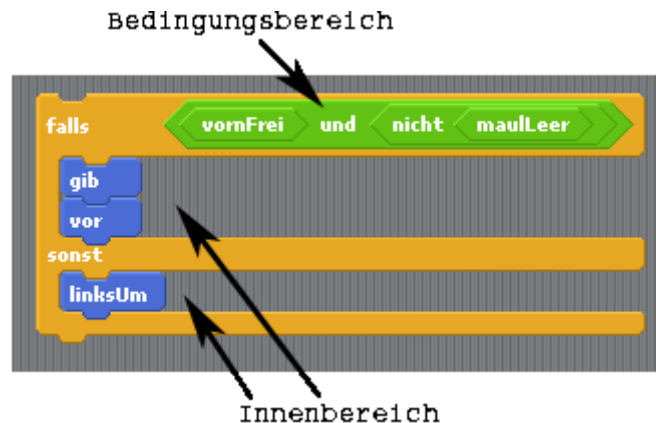


Abbildung 6.10: Steuerungsblöcke

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

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

6.7 Neue Prozeduren oder boolesche Funktionen erstellen

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

Aufruf

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

Informatik *Java-Hamster-Modell*

Handhabung

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

Zu jeder Prozedur bzw. Funktion existiert ein Popup-Menü, das sich durch Anklicken mit der Maus (rechte Maustaste) über dem entsprechenden Block in der Blockpalette oder dem Tab im Programmbereich öffnen lässt. Im Popup-Menü existieren zwei Menü-Items zum Umbenennen und Löschen von Prozeduren und Funktionen.

Besonderheiten

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

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

6.8 Ausführen von Scratch-Hamster-Programmen

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

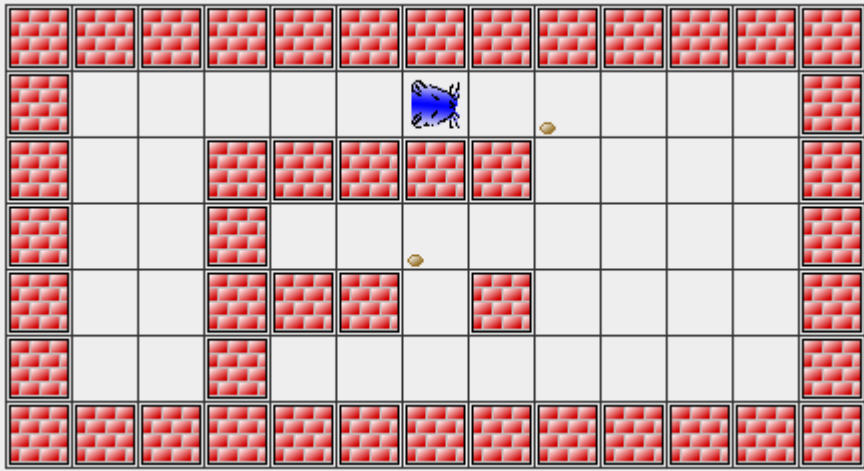
Weiterhin ist es möglich, ein Scratch-Programm Schritt für Schritt, d.h. Anweisung für Anweisung, auszuführen. Hierzu dient der „Schritt-hinein“-Button der Toolbar. Der Block der im nächsten Schritt auszuführenden Anweisung wird dabei im Programmbereich durch eine rote Farbe markiert. Es wird automatisch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

Informatik *Java-Hamster-Modell*

7.0 Methoden (Prozeduren)

In Abbildung 7 ist ein Territorium gegeben, und der Hamster soll zwei Körner einsammeln.

Abb.: 7. Territorium zur Einführung von Methoden (Prozeduren)



Die bisherige Lösung ist umständlich, da sich Befehle wiederholen, z.B. weil der Hamster den Anweisungsblock **rechtsUm** nicht kennt:

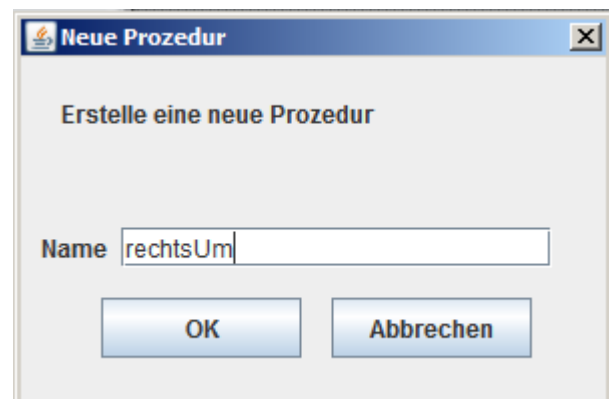


Abb.: 8

Wir wollen den Hamster daher den Anweisungsblock **rechtsUm** neu beibringen. Der Befehl wird in einer Methode (Prozedur) definiert, die den Namen des neuen Anweisungsblocks bekommt. Dazu klicken wir in der Kategorie „Anweisungen“ auf „Neue Prozedur“.

Die Lösung lautet:

Wir vergeben einen neuen Namen für den Anweisungsblock.



Informatik *Java-Hamster-Modell*

Im Programmbereich fügen wir der Methode (Prozedur) die Grundbefehle hinzu.



Ein weiterer neuer Anweisungsblock soll zweiVor heißen, und den Hamster um zwei Kacheln weiter bewegen zu lassen:

Die Lösung lautet:



Jetzt können wir das Programm für die Abb. 8 vereinfachen:

Bisherige Lösung	Neue Lösung

Die neue Lösung umfasst neben dem main-Programm die beiden Definitionen für die neuen Anweisungsblöcke rechtsUm und zweiVor, die sich in der Kategorie Anweisungen jetzt finden.

Innerhalb des main-Programmes werden die neuen Anweisungsblöcke, wie die bisherigen Anweisungsblöcke verwendet.

Informatik *Java-Hamster-Modell*

8.0 Auswahl (bedingte Anweisungen/Verarbeitung, Verzweigung)

In Abbildung 9a ist eine „Zweifache Auswahl“ in der Form eines Struktogramms abgebildet. Wird die Bedingung „wahr“ wird in den Block der Aktion 1 verzweigt, ansonsten bei „falsch“ in den Block der Aktion 2. In der einseitigen bedingten Anweisung, Abb. 9b, entfällt der falsch -Zweig.

Abb. 9a: Zweifache Auswahl

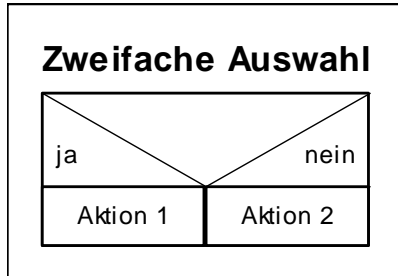
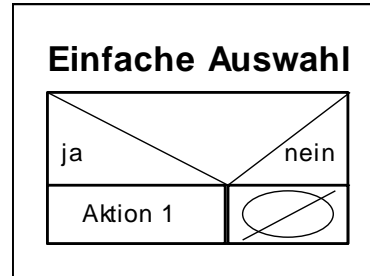


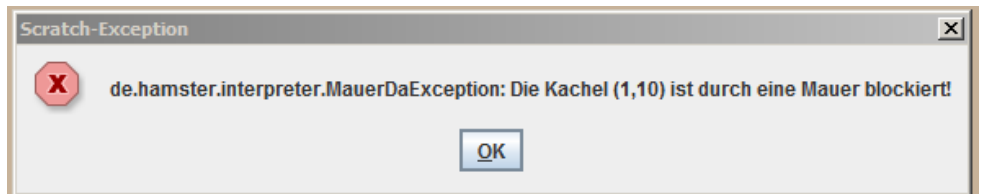
Abb. 9b: Einfache Auswahl



8.1 Hamster-Bedingungen

Läuft der Hamster gegen eine Wand, so würde bisher das Programm abgebrochen werden:

Laufzeitfehler und Programmabbruch



Damit Laufzeitfehler vermieden werden können bringt der Hamster-Simulator folgende Testbefehle mit (s. Abschnitt 6.5)

vornFrei; wahr, falls sich in Blickrichtung keine Mauer befindet, ansonsten falsch
maulLeer; wahr, falls der Hamster keine Körner im Maul hat, ansonsten falsch
kornDa; wahr; falls auf der Kachel auf der der Hamster steht, Körner liegen, ansonsten falsch

8.2 Zweifache Auswahl

Um die Testbefehle anwenden können, verwenden wir die Anweisungsblöcke der Kategorie Steuerung (siehe Abschnitt 6.4). Wir wollen hier testen, ob sich vor dem Hamster eine Mauer befindet. In Abb. 10a die Lösung als Struktogramm (Achtung ja=wahr, nein=falsch).

Informatik *Java-Hamster-Modell*

In Abb. 10b die Lösung in Java-Hamster.

Abb. 10a: Struktogramm „zweifache Auswahl“	Abb. 10b: Lösung im Java-Hamster-Modell

Gehen wir die Lösung schrittweise durch, wird der Aufbau des falls-Anweisungsblocks erkennbar:

	Schlüsselwort „falls“ gefolgt vom Testbefehl „vornFrei“
	Beginn des wahr-Blocks Hamster-Befehle.
	Ende des wahr -Blocks.
	Schlüsselwort „sonst“ für den falsch-Block.
	Beginn des falsch-Blocks. Hamster-Befehle.
	Ende des falsch-Blocks.

Allgemeiner Aufbau der „Zweifachen Auswahl“:

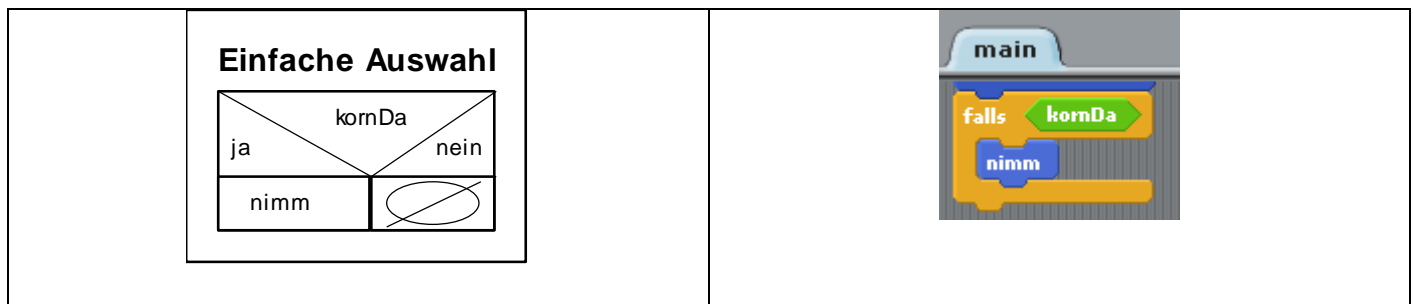


8.3 Einfache Auswahl

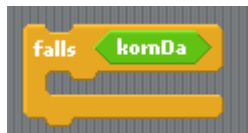
Nehmen wir an, wir wollen testen, ob die Kachel Körner enthält. Der Hamster soll nur etwas tun, wenn Körner vorhanden sind, ansonsten ist nichts zu tun. Hier lösen wir die Aufgabe mit einer „einfachen Auswahl“:

Abb. 11a: Struktogramm „einfache Auswahl“	Abb. 11b: Lösung im Java-Hamster-Modell

Informatik *Java-Hamster-Modell*



Allgemeiner Aufbau der „Einfachen Auswahl“:

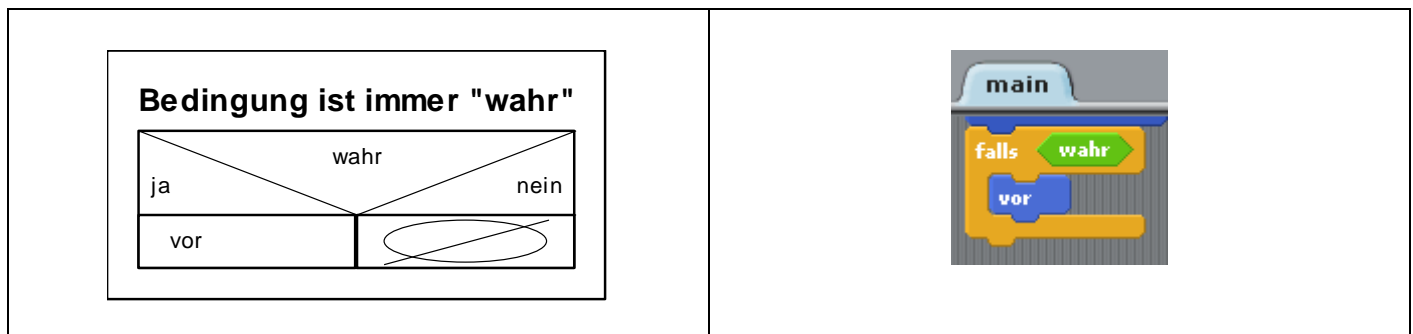


8.4 Boolesche Operatoren

Die Testbefehle `vornFrei`, `kornDa`, und `maulLeer` bekommen in Abhängigkeit von der Situation einen Wert, sie liefern **wahr** oder **falsch**. (siehe auch Abschnitt 6.6).

Innerhalb des Bedingungsereichs steht letztendlich, dann wenn das Programm ausgeführt wird, das Ergebnis **wahr** oder **falsch**.

Hierzu ein sinnloses Beispiel, da der `wahr`-Block immer ausgeführt wird:



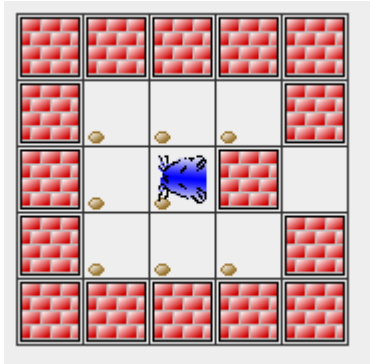
Nun kann es vorkommen, dass eine Aktion ausgeführt werden soll, wenn zwei/mehrere Testbefehle ausgeführt werden sollen.

8.4.1 UND-Operator

In der folgenden Aufgabe (Abb. 12) soll der Hamster das Korn aufnehmen und gleichzeitig vor einer Wand stehen bleiben. Alle Kacheln sind mit Körnern belegt.

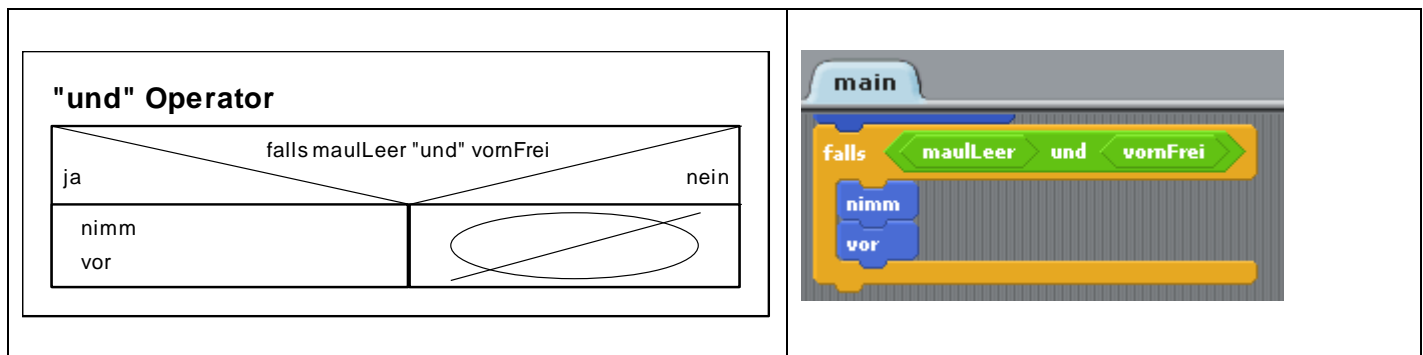
Abb. 12a: Boolesche Operatoren

Informatik *Java-Hamster-Modell*



Hier müssen zwei Testbefehle gleichzeitig ausgewertet werden:

Ist die Bedingung: vornFrei **UND** maulLeer wahr, soll der Hamster das Korn aufnehmen. Ist zusätzlich eine Mauer vorhanden, soll der Hamster nichts tun.



Die Wirkung des UND-Operators lässt sich in einer „Wahrheitstafel“ darstellen:

maulLeer	vornFrei	Wert der Testbefehle	Wirkung
wahr	wahr	wahr	wahr -Zweig
wahr	falsch	falsch	falsch -Zweig
falsch	wahr	falsch	falsch -Zweig
falsch	falsch	falsch	falsch -Zweig

(im obigen Beispiel ist der falsch-Zweig leer).

Beim und-Operator müssen beide Testbefehle wahr sein, soll der wahr-Zweig ausgeführt werden.

8.4.2 ODER-Operator

Die Wirkung des ODER-Operators lässt sich in einer „Wahrheitstafel“ darstellen:

maulLeer	vornFrei	Wert der Testbefehle	Wirkung
wahr	wahr	wahr	wahr -Zweig
wahr	falsch	wahr	wahr -Zweig
falsch	wahr	wahr	wahr -Zweig
falsch	falsch	falsch	falsch -Zweig

Beim „oder-Operator“ muss einer der beiden Testbefehle wahr sein, soll der wahr-Zweig ausgeführt werden.

8.4.3 NICHT-Operator

Der NICHT-Operators steht immer vor einem Testbefehl und kehrt das Ergebnis um.
In der „Wahrheitstafel“ ergibt sich:

	Wert	Negation	Negierter Wert
maulLeer	wahr	!maulLeer	falsch
maulLeer	falsch	!maulLeer	wahr

Informatik *Java-Hamster-Modell*

9.0 Schleifen (Wiederholung, Iteration)

Die bisherigen Hamster-Programme waren auf bestimmte Territorien abgestimmt. Nun kann aber eine Mauer eine unterschiedliche Anzahl von Kacheln vom Hamster entfernt sein, siehe Abb. 13a und 13b. Mit Schleifen können wir die Programme flexibler gestalten.

Der Hamster soll in Blickrichtung zur Mauer laufen.

Abb. 13a: Territorium 1

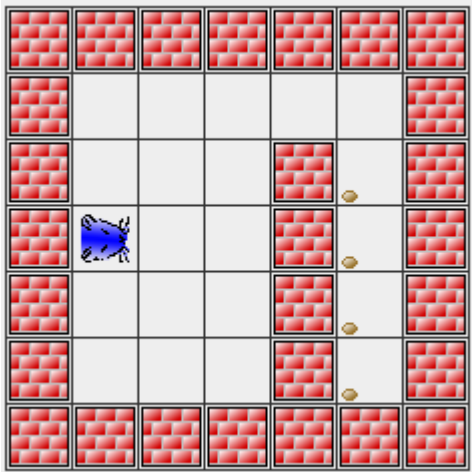
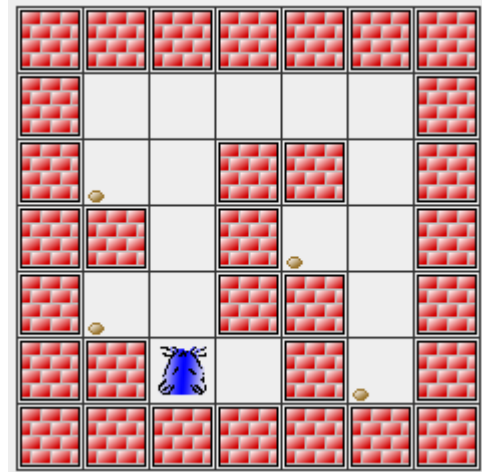


Abb.13b: Territorium 2

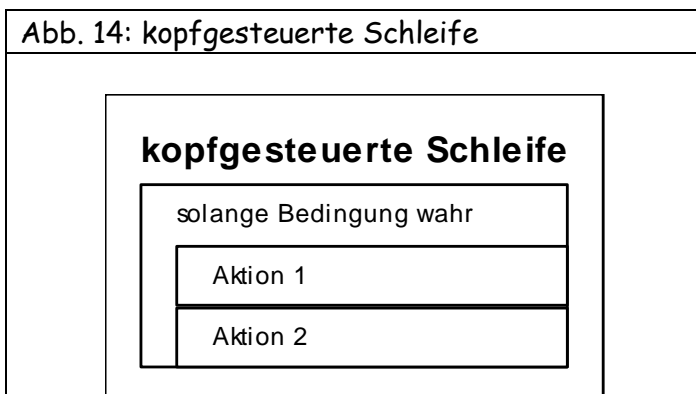


Beide Programme lösen die Aufgabe für das jeweilige Territorium.

9.1 solange-Steuerungsblock (kopfgesteuerte Schleife)

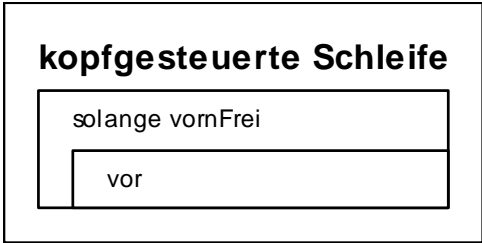

Hinter dem Schlüsselwort *solange* steht die Bedingung. Die Bedingung selbst ist eine bekannte Hamster-Bedingung, wie *vornFrei*. Solange dieser Testbefehl wahr liefert, wird der Schleifenblock mit seinen Aktionen ausgeführt. Eine Besonderheit der kopfgesteuerten Schleife ist, dass falls der Testbefehl bereits beim ersten Durchlauf falsch ergibt, die Schleife nicht ein einziges Mal ausgeführt wird.

Abb. 14: kopfgesteuerte Schleife

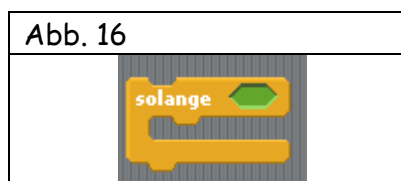


Informatik *Java-Hamster-Modell*

Die in Abb. 13a und 13b gezeigten Aufgaben können jetzt mit einem Hamster-Programm gelöst werden.

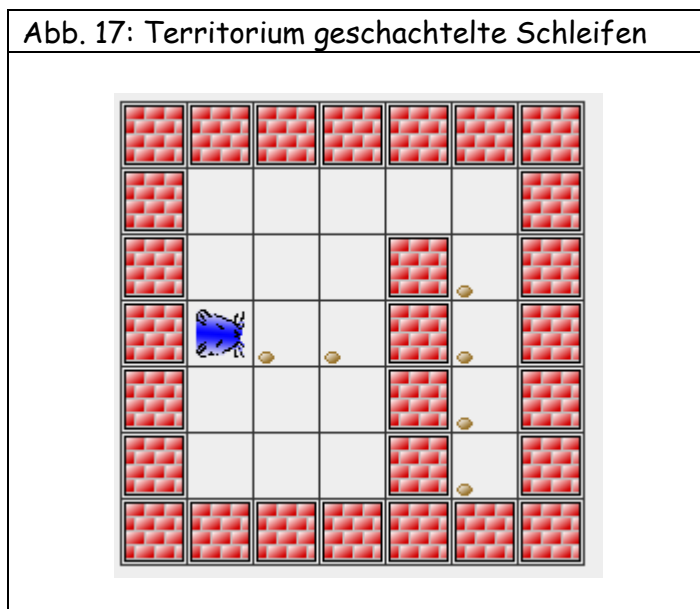
Abb. 15a: Struktogramm zur Abb. 13a	Abb. 15b: Lösung im Java-Hamster-Modell
	

Allgemeiner Aufbau einer kopfgesteuerten solange-Schleife



9.2 Geschachtelte Schleifen

Die Aufgabe des Territoriums in der Abb. 17 besteht darin, erstens bis zur Wand zu laufen und zweitens die Körner aufzunehmen



Eine Lösungsidee besteht darin, in der 1. Schleife die Bedingung vornFrei zu prüfen, und innerhalb des 1. Schleifenblocks eine 2. Schleife zum Aufnehmen der Körner aufzunehmen. In Abb. 18a zeigt das Struktogramm diese Lösung.

Abb. 18a: Struktogramm zur Abb. 17	Abb. 18b: Lösung im Java-Hamster-Modell
------------------------------------	---

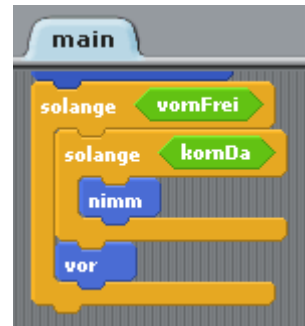
geschachtelte Schleife

solange vornFrei

solange kornDa

nimm

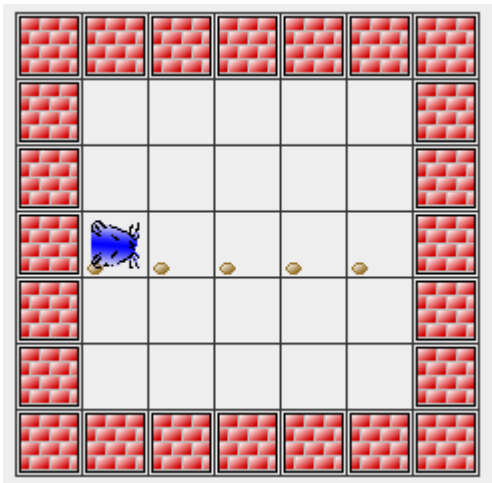
vor



9.3 wiederhole-solange-Steuerungsblock (fußgesteuerte Schleife)

Die Aufgabe des Territoriums in der Abb. 19 besteht darin, alle Körner aufzunehmen. Vorausgesetzt wird, dass alle Kacheln, die der Hamster betritt, Körner enthalten.

Abb. 19: Territorium fußgesteuerte Schleife



Eine Lösung besteht darin den Testbefehl kornDa erst am Ende der Schleife anzuwenden, so dass die Schleife mindestens einmal durchlaufen wird.

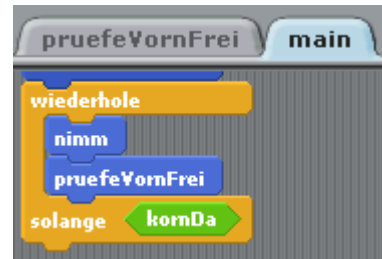
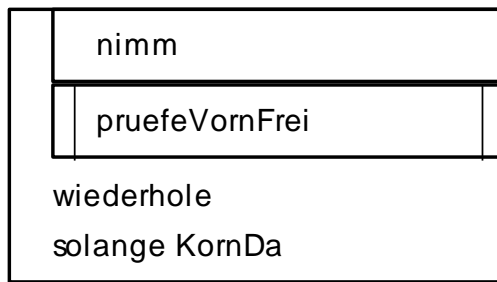
Im Java-Hamster-Modell beginnt die -jetzt- fußgesteuerte Schleife mit dem wiederhole-Schlüsselwort.

Das auflaufen auf der Wand wird mit der neuen Prozedur pruefeVornFrei verhindert. Das Struktogramm in der Abb. 20a zeigt eine Lösung.

Abb. 20a: Struktogramm zur Abb. 19

Abb. 20b: Lösung im Java-Hamster-Modell

fußgesteuerte Schleife



Neue Prozedur/Anweisung „pruefeVornFrei“



Allgemeiner Aufbau einer fußgesteuerten wiederhole-solange-Schleife

Abb. 21



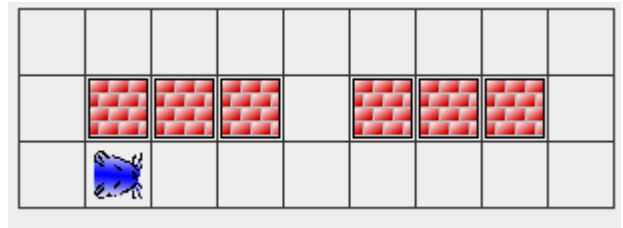
Informatik *Java-Hamster-Modell*

10.0 Boolesche Funktionen

(Siehe einleitend dazu das Kapitel 6.7)

Problem:

Der Hamster soll im gezeigten Territorium die Nische finden.






Die bekannten Testbefehle vornFrei, maulLeer und kornDa machen den unmittelbaren Test, ob links vom Hamster ein Durchbruch ist, nicht möglich. Es können jedoch neue Testbefehle, neue Funktionen, erstellt werden. Der Testbefehl soll „wahr“ oder „falsch“ ergeben.

Der Testbefehl kann dann als Bedingung in einer Auswahl oder Schleife benutzt werden.

Achtung: Ein Testbefehl darf den Ausgangszustand nicht verändern.

In Abb. 22 die Lösung in Java-Hamster.

Abb. 22a:	Abb. 22b: Lösung im Java-Hamster-Modell
<p>Das Programm lässt den Hamster in einer äußeren Schleife vorwärts laufen.</p> <p>Die zweifache Auswahl nutzt den neuen Testbefehl „linksFrei“, um die Nische zu finden.</p>	
<p>Die neue Funktion bzw. der neue Testbefehl.</p> <p>Die Anweisungen „return wahr“ und „return falsch“ verursachen einen sofortigen Aussprung aus der zweifachen Auswahl und geben den booleschen Wert wahr oder falsch zurück.</p> <p>Die Prozedur rechtsUm stellt den Ausgangszustand wieder her.</p>	
<p>Die Prozedur/Methode rechtsUm zur Verbesserung der Übersicht.</p>	

Informatik *Java-Hamster-Modell*

11.0 Imperative Programmierung

11.1 Anweisungen

In imperativen Programmiersprachen werden Verarbeitungsvorschriften durch sogenannte Anweisungen ausgedrückt. Anweisungen, die nicht weiter zerlegt werden können, werden elementare Anweisungen genannt. In der Hamstersprache sind die vier Grundbefehle elementare Anweisungen. Eine Folge von Anweisungen, die nacheinander ausgeführt werden, wird als Anweisungssequenz bezeichnet.

11.2 Beispiele

Das folgende Beispiel enthält eine syntaktisch korrekte Anweisungssequenz:

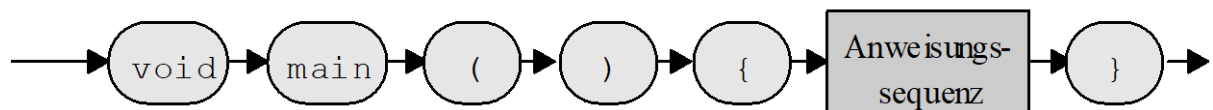
```
vor();  
linksUm(); vor(); nimm(); vor(); gib();  
vor(); linksUm();
```

11.3 Syntax

Die Syntax eines Hamsterprogramms wird in Abbildung 23 definiert. Danach setzt sich ein Hamsterprogramm aus den Schlüsselwörtern **void**, gefolgt von **main**, einem **runden Klammerpaar** und einem geschweiften **Klammerpaar**, das eine Anweisungssequenz umschließt, zusammen.

Abbildung 23: Syntaxdiagramm: Programm

Programm (1)



11.4 Beispiele

Das folgende Beispiel stellt ein syntaktisch korrektes Hamsterprogramm dar:

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

Informatik *Java-Hamster-Modell*

11.5 Kommentare

Ziel der Hamster-Programmierung ist es, Hamsterprogramme zu entwickeln, die gegebene Hamsteraufgaben lösen. Neben ihren Eigenschaften, korrekt und vollständig zu sein, sollten sich Hamsterprogramme 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 sogenannte Kommentare. Sie haben auf die Steuerung des Hamsters keinerlei Auswirkungen. Alles, was sie bewirken, ist eine bessere Lesbarkeit des Programms. In der Hamstersprache gibt es zwei Typen von Kommentaren: Zeilenkommentare und Bereichskommentare.

11.6 Beispiele

Das folgende Programm enthält einen korrekten Bereichskommentar:

```
void main()
{
    /* der Hamster soll sich einmal
       im Kreis drehen
    */
    linksUm(); linksUm(); linksUm(); linksUm();
}
```

Im folgenden Programm wird der Bereichskommentar aus dem obigen Beispiel durch einen Zeilenkommentar ersetzt:

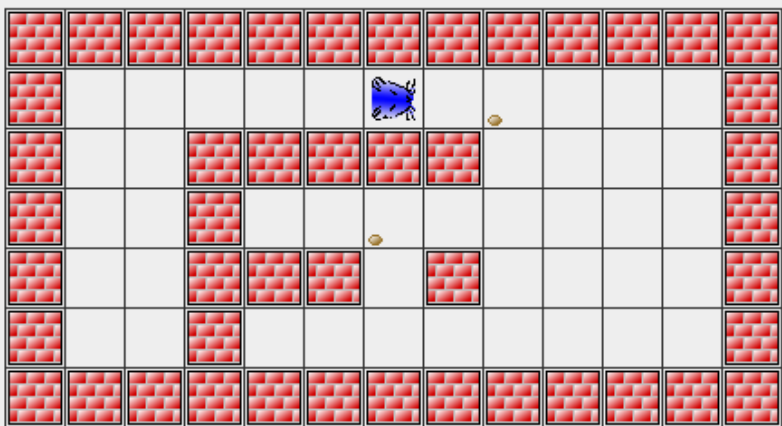
```
void main()
{
    // der Hamster soll sich einmal im Kreis drehen
    linksUm(); linksUm(); linksUm(); linksUm();
}
```

Informatik *Java-Hamster-Modell*

12.1 Methoden (Prozeduren)

In Abbildung 24 ist ein Territorium gegeben und der Hamster soll zwei Körner einsammeln.

Abb: 24. Territorium zur Einführung von Prozeduren



Die bisherige Lösung ist umständlich, da sich Befehle wiederholen, z.B. weil der Hamster den Befehl **rechtsUm()** nicht kennt:

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

Wir wollen den Hamster daher den Befehl **rechtsUm()** neu beibringen.

Der Befehl wird in einer Methode (Prozedur) definiert, die den Namen des neuen Befehls bekommt.

Die Lösung lautet:

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

Ein weiterer neuer Befehl soll **zweiVor()** heißen, und den Hamster um zwei Kacheln weiter bewegen zu lassen:

Die Lösung lautet:

```
void zweiVor()
{
    vor();
    vor();
}
```

Informatik *Java-Hamster-Modell*

Jetzt können wir das Programm für die Abb. 24 vereinfachen:

Bisherige Lösung	Neue Lösung
<pre>{ vor(); vor(); nimm(); linksUm(); linksUm(); linksUm(); vor(); vor(); linksUm(); linksUm(); linksUm(); vor(); vor(); nimm(); }</pre>	<pre>void main() { zweiVor(); nimm(); rechtsUm(); zweiVor(); rechtsUm(); zweiVor(); nimm(); }</pre>
	<pre>void rechtsUm() { linksUm(); linksUm(); linksUm(); }</pre>
	<pre>void zweiVor() { vor(); vor(); }</pre>

Die neue Lösung umfasst neben dem main-Programm die beiden Definitionen für die neuen Befehle rechtsUm() und zweiVor().

Innerhalb des main-Programmes werden die neuen Befehle, wie die bisherigen Befehle verwendet.

Informatik *Java-Hamster-Modell*

13.0 Auswahl (bedingte Anweisungen)

In Abbildung 25a ist eine bedingte Anweisung in der Form eines Struktogramms abgebildet. Wird die Bedingung „true“ (wir verwenden „true“, da „wahr“ kein Schlüsselwort ist), wird in den Block der Aktion 1 verzweigt, ansonsten bei „false“ (da „falsch“ kein Schlüsselwort ist) in den Block der Aktion 2. In der einseitigen bedingten Anweisung, Abb. 25b, entfällt der false-Zweig.

Abb. 25a: Zweifache Auswahl

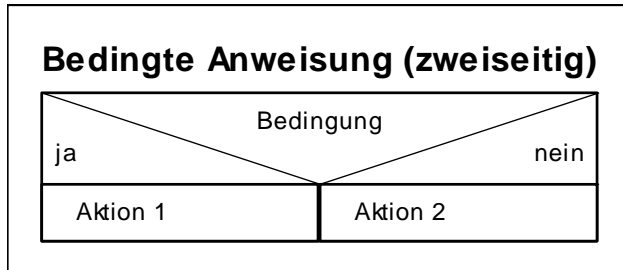
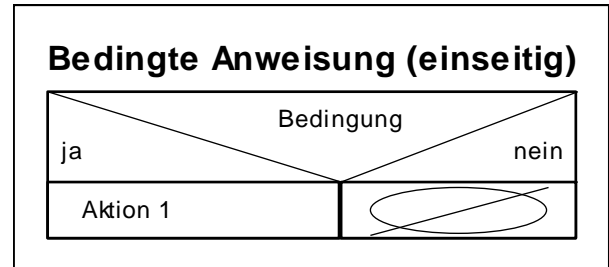


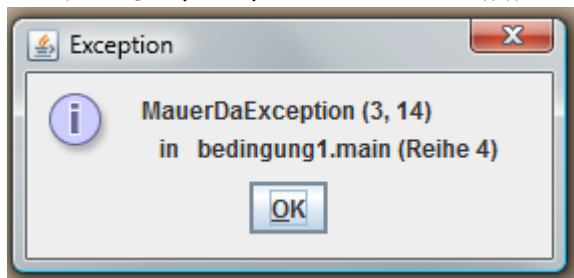
Abb. 25b: Einfache Auswahl



13.1 Hamster-Bedingungen

Läuft der Hamster gegen eine Wand, so würde bisher das Programm abgebrochen werden:

Abb. 26: Laufzeitfehler und Programmabbruch



Damit Laufzeitfehler vermieden werden können bringt der Hamster-Simulator folgende Testbefehle mit:

```
vornFrei();    //true (wahr), falls sich in Blickrichtung keine Mauer befindet, ansonsten false
maulLeer();    //true (wahr), falls der Hamster keine Körner im Maul hat, ansonsten false
kornDa();      //true(wahr); falls auf der Kachel auf der der Hamster steht, Körner liegen, ansonsten false
```

13.2 Zweifache Auswahl

Um die Testbefehle anwenden können, verwenden wir die if-Anweisung. Wir wollen hier testen, ob sich vor dem Hamster eine Mauer befindet. In Abb. 27a die Lösung als Struktogramm (Achtung ja=true, nein=false).

In Abb. 27b die Lösung in Java-Hamster.

Informatik *Java-Hamster-Modell*

Abb. 27a: Struktogramm „Zweifache Auswahl“

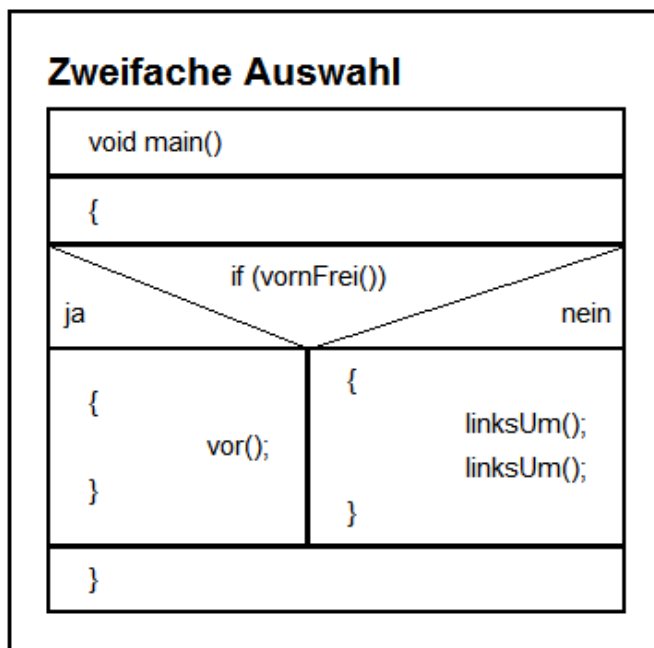


Abb. 27b: Lösung im Java-Hamster-Modell

```

void main()
{
    if ( vornFrei() )
    {
        vor();
    }
    else
    {
        linksUm();
        linksUm();
    }
}

```

Gehen wir die Lösung schrittweise durch, wird der Aufbau der if-Anweisung erkennbar:

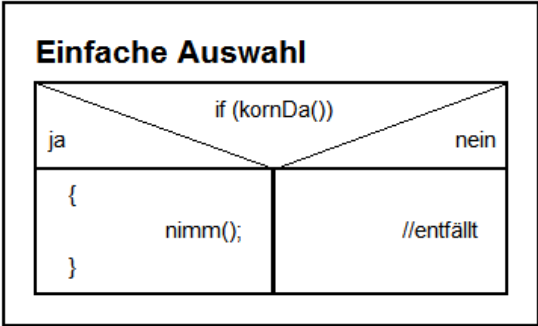
if (vornFrei())	Schlüsselwort if. Gefolgt von einem Klammerpaar. Innerhalb des Klammerpaars der Testbefehl vornFrei().
{	Beginn des true-Blocks
vor();	Hamster-Befehle.
}	Ende des true-Blocks.
else	Schlüsselwort else für den false-Block.
{	Beginn des false-Blocks.
linksUm(); linksUm();	Hamster-Befehle.
}	Ende des false-Blocks.

Allgemeiner Aufbau der „Zweifachen Auswahl“:

Abb. 28	
if (Testbefehl)	// Schlüsselwort
{	// true-Block
...	
}	
else	// Schlüsselwort
{	// false Block
...	
}	

13.3 Einfache Auswahl

Nehmen wir an, wir wollen testen, ob die Kachel Körner enthält. Der Hamster soll nur etwas tun, wenn Körner vorhanden sind, ansonsten ist nichts zu tun. Hier lösen wir die Aufgabe mit einer „einfachen“ einseitigen Bedingung:

Abb. 29a: Struktogramm „Einfache Auswahl“	Abb. 29b: Lösung im Java-Hamster-Modell
	<pre>void main() { if (kornDa()) { nimm(); } vor(); }</pre>

Allgemeiner Aufbau der „Einfachen Auswahl“:

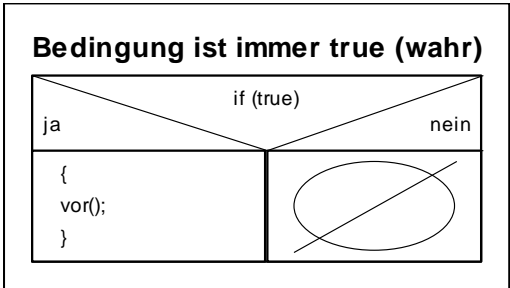
Abb. 30
<pre>if (Testbefehl) // Schlüsselwort { // true-Block ... }</pre>

13.4 Boolesche Operatoren

Die Testbefehle `vornFrei()`, `kornDa()`, und `maulLeer()` bekommen in Abhängigkeit von der Situation einen Wert, sie liefern **true** oder **false**.

Innerhalb der runden Klammern der `if`-Anweisung steht letztendlich, dann wenn das Programm ausgeführt wird, das Ergebnis `true` oder `false`.

Hierzu ein sinnloses Beispiel, da der `true`-Block immer ausgeführt wird:

	<pre>void main() { if (true) { vor(); } }</pre>
---	---

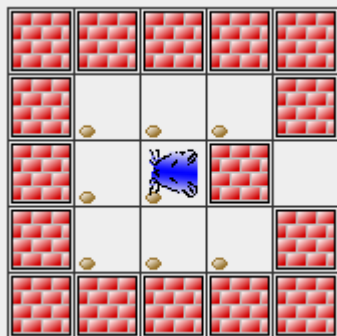
Informatik *Java-Hamster-Modell*

Nun kann es vorkommen, dass eine Aktion ausgeführt werden soll, wenn zwei/mehrere Testbefehle ausgeführt werden sollen.

13.4.1 UND-Operator

In der folgenden Aufgabe (Abb. 31) soll der Hamster das Korn aufnehmen und gleichzeitig vor einer Wand stehen bleiben. Alle Kacheln sind mit Körnern belegt.

Abb. 31: Boolesche Operatoren



Hier müssen zwei Testbefehle gleichzeitig ausgewertet werden:

Ist die Bedingung: vornFrei() **UND** maulLeer() wahr, soll der Hamster das Korn aufnehmen. Ist zusätzlich eine Mauer vorhanden, soll der Hamster nichts tun. Das logische **UND** wird in Java mit den Zeichen „&&“ dargestellt.

UND-Operator

if (maulLeer() && vornFrei())	
ja	nein
<pre>{ nimm(); vor(); }</pre>	

```
void main() {  
    if ( maulLeer() && vornFrei() )  
    {  
        nimm();  
        vor();  
    }  
}
```

Die Wirkung des UND-Operators „&&“ lässt sich in einer „Wahrheitstafel“ darstellen:

Abb. 32			
maulLeer()	vornFrei()	Wert der Testbefehle	Wirkung
true	true	true	true-Zweig
true	false	false	false-Zweig
false	true	false	false-Zweig
false	false	false	false-Zweig

(im obigen Beispiel ist der false-Zweig leer).

Informatik *Java-Hamster-Modell*

Beim &&-Operator müssen beide Testbefehle true sein, soll der true-Zweig ausgeführt werden.

13.4.2 ODER-Operator

Die Wirkung des ODER-Operators „||“ lässt sich in einer „Wahrheitstafel“ darstellen:

maulLeer()	vornFrei()	Wert der Testbefehle	Wirkung
true	true	true	true-Zweig
true	false	true	true-Zweig
false	true	true	true-Zweig
false	false	false	false-Zweig

Beim „||-Operator“ muss einer der beiden Testbefehle true sein, soll der true-Zweig ausgeführt werden.

13.4.3 NICHT-Operator

Der NICHT-Operators „!“ steht immer vor einem Testbefehl und kehrt das Ergebnis um.
In der „Wahrheitstafel“ ergibt sich

	Wert	Negation	Negierter Wert
maulLeer()	true	!maulLeer()	false
maulLeer()	false	!maulLeer()	true

Weitere Ausdrücke sind:

	Wert	Negation	Negierter Wert
maulLeer() && kornDa()	true	!(maulLeer() && kornDa())	false
maulLeer() && kornDa()	false	!(maulLeer() && kornDa())	true

Informatik *Java-Hamster-Modell*

14.0 Schleifen (Wiederholung, Iteration)

Die bisherigen Hamster-Programme waren auf bestimmte Territorien abgestimmt. Nun kann aber eine Mauer eine unterschiedliche Anzahl von Kacheln vom Hamster entfernt sein, siehe Abb. 33a und 33b. Mit Schleifen können wir die Programme flexibler gestalten.

Der Hamster soll in Blickrichtung zur Mauer laufen.

Abb. 33a: Territorium 1

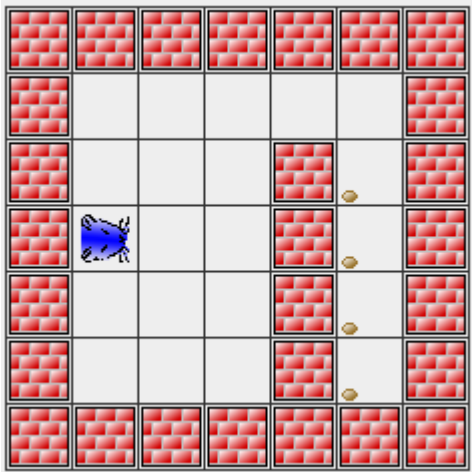
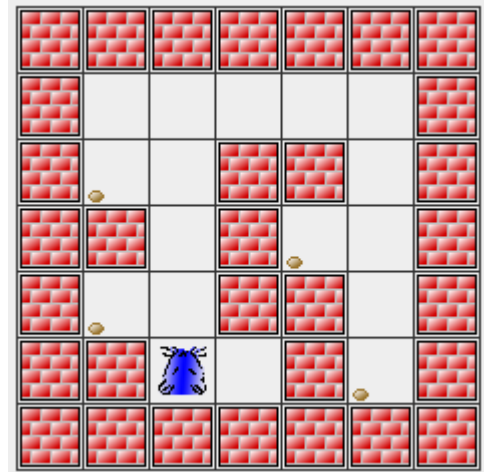


Abb. 33b: Territorium 2



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

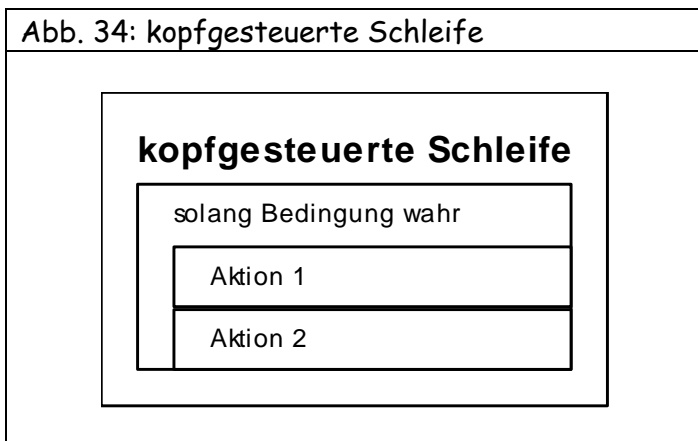
```
void main()
{
    vor();vor();vor();vor();
}
```

Beide Programme lösen die Aufgabe für das jeweilige Territorium.

14.1 while-Anweisung

Hinter dem Schlüsselwort `while` steht die Bedingung. Die Bedingung selbst ist eine bekannte Hamster-Bedingung, wie `vornFrei()`. Solange dieser Testbefehl `true` (wahr) liefert wird der Schleifenblock mit seinen Aktionen ausgeführt. Eine Besonderheit der kopfgesteuerten Schleife ist, dass falls der Testbefehl bereits beim ersten Durchlauf `false` ergibt, die Schleife nicht ein einziges Mal ausgeführt wird.

Abb. 34: kopfgesteuerte Schleife



Informatik *Java-Hamster-Modell*

Die in Abb. 33a und 33b gezeigten Aufgaben können jetzt mit einem Hamster-Programm gelöst werden.

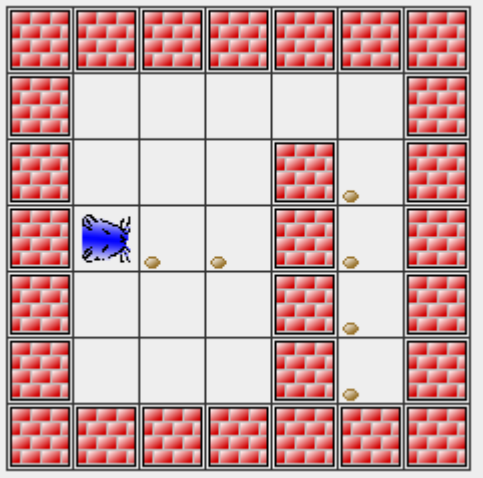
Abb. 35a: Struktogramm zur Abb. 33a	Abb. 35b: Lösung im Java-Hamster-Modell
<div><p>kopfgesteuerte Schleife</p><pre>while (vornFrei()) { vor(); }</pre></div>	<pre>void main() { while (vornFrei()) { vor(); } }</pre>

Allgemeiner Aufbau einer kopfgesteuerten while-Schleife

Abb. 36
<pre>while (Testbefehl) // Schlüsselwort while { // true-Block ... }</pre>

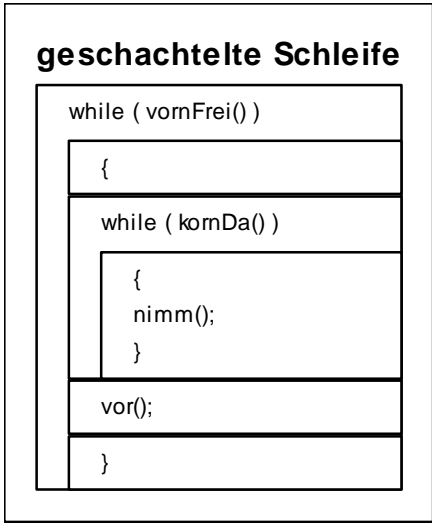
14.2 Geschachtelte Schleifen

Die Aufgabe des Territoriums in der Abb. 37 besteht darin, erstens bis zur Wand zu laufen und zweitens die Körner aufzunehmen

Abb. 37: Territorium geschachtelte Schleifen


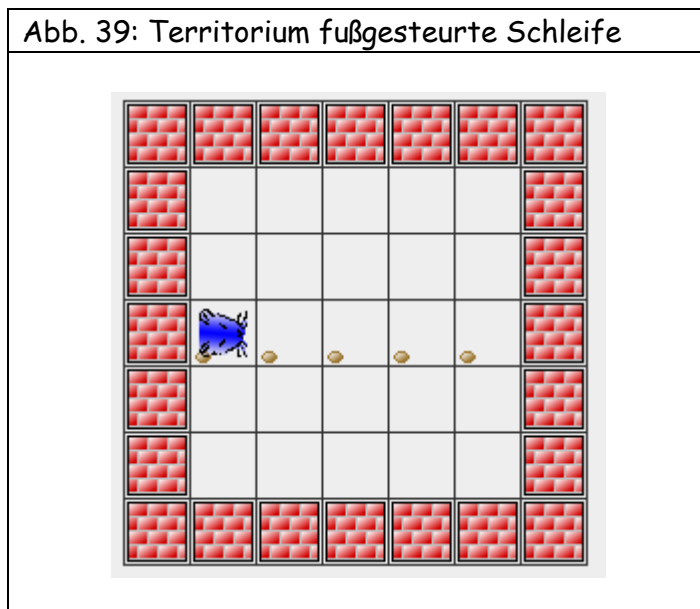
Informatik *Java-Hamster-Modell*

Eine Lösungsidee besteht darin, in der 1. Schleife die Bedingung vornFrei() zu prüfen, und innerhalb des 1. Schleifenblocks eine 2. Schleife zum aufnehmen der Körner aufzunehmen. In Abb. 38a zeigt das Struktogramm diese Lösung.

Abb. 38a: Struktogramm zur Abb. 37	Abb. 38b: Lösung im Java-Hamster-Modell
	<pre>void main() { while (vornFrei()) { while (kornDa()) { nimm(); } vor(); } }</pre>

14.3 Fußgesteuerte Schleife

Die Aufgabe des Territoriums in der Abb. 39 besteht darin, alle Körner aufzunehmen. Vorausgesetzt wird, dass alle Kacheln, die der Hamster betritt, Körner enthalten.



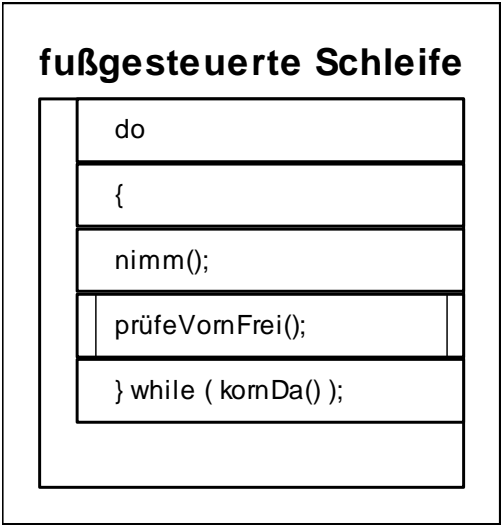
Eine Lösung besteht darin den Testbefehl kornDa() erst am Ende der Schleife anzuwenden, so dass die Schleife mindestens einmal durchlaufen wird.

Im Java-Hamster-Modell beginnt die -jetzt- fußgesteuerte Schleife mit dem do-Schlüsselwort.

Das auflaufen auf die Wand wird mit der neuen Prozedur prüfeVornFrei() verhindert.

Informatik *Java-Hamster-Modell*

Das Struktogramm in der Abb. 40a zeigt eine Lösung.

Abb. 40a: Struktogramm zur Abb. 34a	Abb. 40b: Lösung im Java-Hamster-Modell
	<pre>void main() { do { nimm(); prüfeVornFrei(); }while (kornDa()); } void prüfeVornFrei() { if (vornFrei()) { vor(); } }</pre>

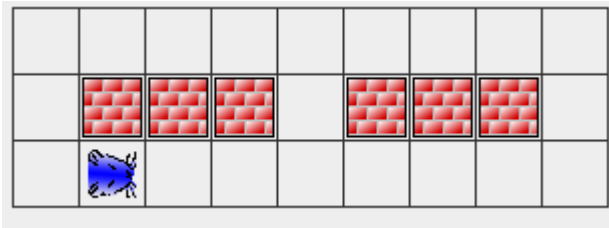
Allgemeiner Aufbau einer fußgesteuerten do-while-Schleife

Abb. 41	
<pre>do { ... } while (Testbefehl);</pre>	<pre>// Schlüsselwort do // true-Block // Schlüsselwort while // Wichtig ist hier ";;"</pre>

Informatik *Java-Hamster-Modell*

15.0 Boolesche Funktionen

Der Hamster soll im gezeigten Territorium die Nische finden.



Lösung:

```
void main()
{
    while ( vornFrei() )
    {
        if ( linksFrei() )
            { linksUm();vor(); }
        else
            {vor();}
    }
}
```

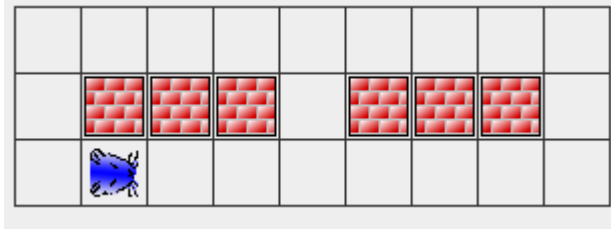
```
boolean linksFrei()
{
linksUm();
    if ( vornFrei() )
        {rechtsUm();
        return true;}
    else
        {rechtsUm();
        return false;}
}
```

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

Informatik *Java-Hamster-Modell*

16.0 Boolesche Variablen

Der Hamster soll im gezeigten Territorium die Nische finden.



Die Verbesserung gegenüber der in Abschnitt 15.0 gezeigten Lösung liegt in der verbesserten boolean-Funktion.

Statt der Auswahlanweisung wird das Ergebnis der Testfrage „vornFrei()“ in einer Variablen gespeichert. Variablen haben einen Namen und sind Platzhalter für Werte. Hier nutzen wir den Variablentyp „boolean“ um eine Variable zu definieren, die die Werte true und false annehmen kann.

Lösung:

```
void main()
{
    while (vornFrei())
    {
        if (linksFrei())
        {
            linksUm();
            vor();
        }
        else
        {
            vor();
        }
    }
}

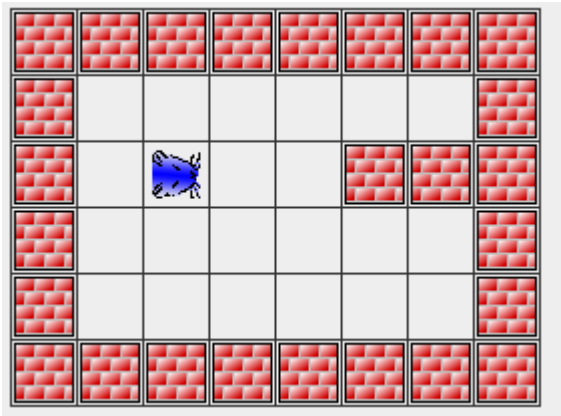
boolean linksFrei()
{
    boolean bol_istfrei;    //Variablen-Deklaration
    linksUm();
    bol_istfrei = vornFrei();
    rechtsUm();
    return bol_istfrei;
}

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


Informatik *Java-Hamster-Modell*

17.0 Zahlen, Variablen, Ausdrücke

Der Hamster soll in einem beliebigen Territorium Kacheln bis zur Mauer zählen. Für jede Kachel soll er dann ein Korn ablegen. Er hat genügend Körner im Maul.



Zum Zählen benötigen wir eine Variable, die die Anzahl Schritte speichert. Da wir ganze Zahlen haben ist der Variablentyp `INTEGER`.

Hier wird auch eine besondere Form der Zuweisung benutzt, eine die in der Mathematik nicht erlaubt ist.

Achtung: Alles was rechts vom Gleichheitszeichen steht - wird berechnet - und dann der links stehenden Variablen zugewiesen.

Beispiele: `schritte = wertA + wertB`
`schritte = schritte + 1`

Lösung:

```
void main()
{
  int schritte;
  schritte = 0;
  while (vornFrei())
  {
    vor();
    schritte = schritte + 1;
  }

  while (schritte > 0)
  {
    if (!maulLeer()) { gib();}
    schritte = schritte - 1;
  }
}
```