LANDESLEHRERPRÜFUNGSAMT Außenstelle beim Regierungspräsidium Karlsruhe	STAATLICHES SEMINAR FÜR DIDAKTIK UND LEHRERBILDUNG (Gymnasien) KARLSRUHE

Zweite Staatsprüfung für die Laufbahn des höheren Schuldienstes an Gymnasien

Schriftliche Arbeit

Fach:	Informationstechnische Grundbildung (ITG)
Thema:	**Einstieg in die Programmierung mit dem Java-Hamster-Modell in einer 8. Klasse (ITG)**
Klasse(nstufe):	8
Verfasser:	**Tim Fruth**
Fachleiter:	**StD Theo Heußer**

Versicherung:
Ich versichere, dass ich diese schriftliche Prüfungsarbeit selbstständig und nur mit den angegebenen Hilfsmitteln angefertigt habe und dass ich alle Stellen, die dem Wortlaut oder dem Sinn nach anderen Werken entnommen sind, durch Angabe der Quellen als Entlehnung kenntlich gemacht habe.

....
(Ort, Datum) (Unterschrift)

Im Falle der Aufbewahrung meiner Arbeit im Archiv des Staatlichen Seminars für Didaktik und Lehrerbildung bzw. im Staatsarchiv erkläre ich mein Einverständnis, dass die Arbeit Benutzern zugänglich gemacht werden kann.

....
(Ort, Datum) (Unterschrift)

„Bei all dem Wehgeschrei über den Zustand der Bildung
wird ein Bildungsort, vielleicht der wichtigste überhaupt,
häufig vergessen oder nur gestreift: **Die Familie**“

Daniel Goeudevert, Der Horizont hat Flügel, 2001

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

Albert Einstein sagte einst: „Ich habe keine besondere Begabung, sondern bin einfach nur *leidenschaftlich neugierig!*“ Ich kann und will an dieser Stelle meine Bewunderung für Albert Einstein nicht verbergen, denn ich halte ihn für den größten Denker des letzten Jahrtausends. Ich möchte auch nicht darüber urteilen, ob der erste Teil dieses Zitats wahr ist, also ob Einstein wirklich nicht sonderlich begabt war. Nach recht intensiver Lektüre von Einsteins Leben, bin ich mir jedoch sicher, dass der zweite Teil („leidenschaftliche Neugier“) absolut zutrifft. Und zu welchen wissenschaftlichen Erkenntnissen diese leidenschaftliche Neugier führte ist allgemein bekannt.

Was hat dieses Zitat nun mit Schule und insbesondere mit dieser Arbeit zu tun? Nun, das Verständnis vom Lehren und Lernen und die Auswirkungen auf die Gestaltung des schulischen Unterrichts haben sich in den letzten Jahren drastisch verändert. Es ist zunehmend von der Stärkung der *Kompetenzen* der Schüler die Rede. Neben der *fachlichen* Kompetenz ist verstärkt von *personaler*, *sozialer* und insbesondere von *methodischer* Kompetenz die Rede [Bil04] [Mat03]. Vor allem seit dem Einzug der *konstruktivistischen Didaktik* (Kersten Reich) in den 1990er–Jahren, „[...] kippt die traditionelle Lehrer– und Inhaltsdidaktik um in eine Lerner– und Beziehungsdidaktik“ [Huw04a]. (Es soll im Folgenden jedoch nicht der Eindruck vermittelt werden, dass der konstruktivistische Ansatz der einzig richtige ist. Huwendiek propagiert in [Huw04a] vielmehr eine Mischform aus unterschiedlichen didaktischen Ansätzen, die sich „[...] zum Teil einander ergänzen und entsprechend kombiniert werden können.“) Die Schüler sollen sich aktiv mit dem Lehrstoff auseinandersetzen und sich Problemlösemethoden selbst erschließen (Prinzip des *aktiven Lernens* [Bau96]). Dabei wird Wissen konstruiert, anstatt (wie es bisher war und oftmals immer noch ist) vorwiegend rezeptiv und passiv gelernt [Hub00]. Auch die Rolle des Lehrers ändert sich. Dieser fungiert immer mehr als Berater, Organisator und Moderator anstatt als Präsentator, Stoffdarbieter bzw. Wissensvermittler [Kli04]. Empirische Untersuchungen geben diesem didaktischen Ansatz insofern recht, als dass die Konstruktion von Wissen effektiver ist als beispielsweise die rein visuelle oder auditive Aneignung von Wissen [Kli04].

Mir stellt sich in diesem Zusammenhang nun die Frage, wie man den Schüler zum eigenverantwortlichen Lernen und zur aktiven Auseinandersetzung mit dem Lernstoff bewegt? Und an dieser Stelle schließt sich für mich der Kreis mit dem einleitenden Zitat Einsteins: Man muss in ihm eine „leidenschaftliche Neugier“ wecken, die ihn dazu bewegt, sich mit den gestellten Problemen auseinander setzen *zu wollen*. Die Weckung dieser Neugier ist Aufgabe des Lehrers.

Dieser muss zum einen seine eigene Neugier und sein Interesse am Lernstoff an die Schüler weitervermitteln (sie quasi damit euphorisieren), und zum anderen muss er Themen ansprechen und behandeln, die einerseits durch den Bildungsplan abgedeckt werden, andererseits jedoch unbedingt im Erfahrungs– und Interessensbereich der Schüler liegen (persönliche Meinung und außerdem beschrieben in [Hub00]).

Wozu nun dieser Diskurs? Diese Pädagogische Arbeit wurde im Fach *Informationstechnische Grundbildung (ITG)* durchgeführt. Aus meinen eigenen Erfahrungen kann ich berichten, dass es gerade in diesem Fach besonders einfach ist, bei den Schülern „Neugier“ zu wecken. Es besteht ein natürliches Interesse an informationstechnischen Fragestellungen. Die meisten Schüler beschäftigen sich gerne mit dem Computer, wenngleich oftmals leider sehr unsystematisch und oberflächlich. Auch Themen aus ihrem Erfahrungs– und insbesondere Interessensbereich sind schnell gefunden: Handy–Programmierung, Computerviren, Gestaltung einer Homepage, MP3-Format sowie Aufbau und Funktionsweise von lokalen Netzwerken sind nur einige Beispiele. Diese durchaus positiven Voraussetzungen sollten unbedingt bei der didaktischen und methodischen Analyse von Unterrichtseinheiten im Fach ITG berücksichtigt werden.

Inhalt dieser Arbeit ist die Durchführung einer zwölfstündigen Unterrichtseinheit (UE) über den Einstieg in die Programmierung mit einer Lern– bzw. Programmierumgebung, dem sogenannten *Java–Hamster–Modell* [Bol02] [Ham]. Durchgeführt wurde die UE in der Klasse 8e des Ottheinrich–Gymnasiums in Wiesloch im Zeitraum April 2006 – Mai 2006. Ziel der Arbeit ist der spielerische Zugang zur Programmierung (mit der Programmiersprache Java [Sun]). Dieser soll einerseits nicht zu oberflächlich und intuitiv geschehen. Der Entwurf von Algorithmen mit dem Top–Down–Verfahren sowie die theoretische und praktische Untersuchung der algorithmischen Grundkonstrukte (Sequenz, Wiederholung und Alternative) ist integraler Bestandteil der UE. Andererseits soll jedoch eine geeignete *didaktische Reduktion* erfolgen, um eine Abschweifung in wissenschaftliche Details zu verhindern.

Das vorliegende Manuskript ist in drei Teile gegliedert. Der erste Teil befasst sich mit der Planung der durchgeführten UE. Diese besteht hauptsächlich aus einer fachlichen sowie einer didaktischen und methodischen Analyse. In Teil Zwei der Arbeit werden (unter Angabe der jeweiligen Lernziele und –inhalte) die Unterrichtsverläufe der einzelnen Stunden dokumentiert. Jede Unterrichtsstunde wird mit einer kurzen Reflexion abgeschlossen. Im dritten und abschließenden Teil wird eine kritische Gesamtreflexion durchgeführt. Außerdem werden Verbesserungsvorschläge bezüglich des Verlaufs und der ausgearbeiteten Materialien angegeben. Im Anhang befinden sich die erstellten Arbeitsblätter sowohl abgedruckt als auch in elektronischer Form (auf der Begleit–CD).

Im Hinblick auf eine übersichtliche und kompakte Darstellung wird im weiteren Verlauf dieser Arbeit die maskuline Form bei Personen verwendet. Dabei sind aber immer „Schülerinnen und Schüler“ sowie „Lehrerinnen und Lehrer“ gemeint.

2 Planung und Vorbereitung

Das folgende Kapitel beschreibt die Vorbereitung und Planung der durchgeführten Unterrichtseinheit (UE). Neben der Situation und den Lernvoraussetzungen der Klasse folgt zunächst eine fachwissenschaftliche Betrachtung und anschließend eine didaktische Analyse sowie methodische Vorüberlegungen.

2.1 Situation und Lernvoraussetzungen der Klasse

Wie in Kapitel 1 beschrieben, wurde die UE im Zeitraum April 2006 – Mai 2006 in der Klasse 8e des Ottheinrich–Gymnasiums in Wiesloch durchgeführt. Ich habe die Klasse im November 2005 übernommen, und nachdem der Fachlehrer zum Halbjahr die Schule verließ, habe ich den Unterricht (unter Zuteilung eines neuen Fachlehrers) bis zum Schuljahresende fortgeführt.

Die Informationstechnische Grundbildung (ITG) wurde im Schuljahr 2005/06 in den 8. Klassen einstündig unterrichtet, wobei in je einer Klassenhälfte 14–tägig jeweils eine Doppelstunde abgehalten wurde. Für den Zeitraum der Pädagogischen Arbeit wurde dieser Rhythmus geändert, sodass je eine Klassenhälfte insgesamt zwölf Stunden durchgehend (also wöchentlich) unterrichtet wurde.

In der Klassenhälfte, in welcher die hier beschriebene UE durchgeführt wurde, befanden sich neun Jungen und fünf Mädchen. Sowohl den Leistungsstand als auch Verhalten und Mitarbeit würde ich als gut bezeichnen. Allerdings fielen ab und zu einige Jungen beim Umgang mit dem Computer durch leicht überhebliches Verhalten auf.

Zu Beginn des Schuljahres beschäftigten die Schüler sich mit der Navigation im Dateisystem und der Textverarbeitung. Als ich die Klasse übernahm, behandelte ich ausführlich die Themen „Tabellenkalkulation“ und „Internet“. Im Rahmen der UE „Internet“ beschäftigten die Schüler sich u.a. eine Doppelstunde mit der Sprache HTML. Somit kamen sie zum ersten Mal mit einer zwar nicht streng formalen Programmiersprache, aber zumindest mit einer sogenannten *Auszeichnungssprache* in Kontakt. Programmiererfahrungen besaß nach Auskunft der Schüler noch niemand.

Sowohl bei dem zuerst unterrichtenden Lehrer der Klasse als auch bei den Kollegen in den Parallelklassen wurden im Fach ITG weder Leistungsmessungen erhoben noch wurden Hausaufgaben aufgegeben. Aus diesem Grund hielt ich mich ebenfalls an diese Vorgehensweise. Deshalb wurde am Ende der UE keine unmittelbare Leistungsmessung durchgeführt.

2.2 Fachwissenschaftliche Betrachtungen

Besonders im Hinblick auf didaktische und methodische Überlegungen ist es zunächst einmal sinnvoll und notwendig, die Leitvorstellungen und unterrichtlichen Inhalte sowie eventuelle Probleme und Besonderheiten des Fachs ITG zu erörtern und darzustellen. Um jedoch nicht den Rahmen dieser Ausarbeitung zu sprengen, werden nur die wichtigsten Punkte genannt. Detailliertere Informationen findet man beispielsweise in [Bau96], [Bil04] oder [BLK87].

Des Weiteren muss die Durchführung der UE im Hinblick auf den Bildungsplan 2004 des Landes Baden-Württemberg ([Bil04]) sinnvoll begründet werden.

2.2.1 Die Informationstechnische Grundbildung

Die ITG zählt zweifelsfrei zu den jüngeren Fächern in der Sekundarstufe I im schulischen Bereich. In den 1980er–Jahren fanden verstärkt Diskussionen statt, dass „[...] den Schülern eine sogenannte *informationstechnische Grundbildung* [...] zu vermitteln sei.“ [Bau96]. Daraus resultierte ein „Rahmenkonzept für die informationstechnische Bildung in Schule und Ausbildung“, welches im Jahr 1984 von der *Bund–Länder–Kommission für Bildungsplanung und Forschungsförderung* veröffentlicht wurde (nachzulesen in [BLK87]). Dieses Konzept enthielt mehrere, durch die ITG zu vermittelnden Inhalte, welche weitestgehend auch im Bildungsplan 2004 des Landes Baden-Württemberg ([Bil04]) Einzug fanden. Darin wird die Notwendigkeit der Informationstechnischen Grundbildung u.a. mit den Stichworten „Medienerziehung“, „reflektierter und verantwortungsvoller Umgang mit den neuen Medien“ und „Stärkung der methodischen und sozialen Kompetenz“ begründet.

Probleme der ITG

Das Schulfach ITG wird mit einigen Problemen konfrontiert, mit denen sich „herkömmliche“ Fächer nicht beschäftigen müssen. Einige Probleme müssen bei der Unterrichtsvorbereitung unbedingt berücksichtigt werden, z.B.:

- Die Schülergruppe ist meist äußerst inhomogen in Bezug auf informationstechnisches Vorwissen [SS04]. Eine *innere Differenzierung* wird damit umso wichtiger.
- Aus meinen *bisherigen* Erfahrungen weiß ich, dass die ITG bei einigen Schülern nicht ernst genommen wird, z.B. Schülerzitat: „ITG ist doch nur ein Spaßfach!“.
- Schüler arbeiten meist sehr intuitiv am Rechner, d.h. sehr unsystematisch und mit einer nicht reproduzierbaren Arbeits– und Vorgehensweise.
- Jungen neigen des Öfteren zur Selbstüberschätzung und Überheblichkeit. Mädchen geraten dadurch nicht selten ins Abseits, da sie denken, dass die Jungen es wirklich besser könnten [SS04].

- Für die ITG existieren im gymnasialen Bereich in Baden-Württemberg zwar Bildungsstandards, jedoch werden dem Fach keine Unterrichtsstunden zugewiesen. Folglich werden dafür Poolstunden verwendet, die wiederum einzelnen Fächern (z.B. Deutsch oder Mathematik) zugeordnet sind. Das kann schließlich dazu führen, dass auch Lehrkräfte mit relativ geringer informationstechnischer Vorbildung das Fach ITG unterrichten.

Besonderheiten und Chancen der ITG

Ebenso wie die Probleme, müssen auch Besonderheiten und damit verbundene Chancen der ITG in jedweder unterrichtlichen Planung beachtet werden. Einige Chancen sind beispielsweise:

- Die ITG stößt bei vielen Schülern auf Begeisterung und Interesse.
- Die behandelten Themen haben meist einen relativ großen praktischen Wert, sodass den Schülern bewusst wird, warum sie diese lernen.
- Man kann — und das ist die große Herausforderung — viele Beispiele und Aufgaben aus dem Erfahrungsbereich der Schüler finden. Dies führt zu einer zusätzlichen Motivation. Auch Hubwieser bemerkt, dass „[...] Motivation das vordringlichste Ziel didaktischen Handelns“ sei [Hub00].
- Die von vielen Pädagogen geforderte *innere Differenzierung* lässt sich meines Erachtens bei der Arbeit am Computer besser realisieren als in anderen Fächern. Dies kann durch die Angabe *weiterführender Aufgaben* erreicht werden.

2.2.2 Begründung des Themas

Zur Begründung und Rechtfertigung der UE können primär der Bildungsplan 2004 des Landes Baden-Württemberg ([Bil04]) sowie das schuleigene Curriculum herangezogen werden. Sekundär ist ebenfalls die Einbeziehung der Fachdidaktik-Literatur möglich. Da es am Ottheinrich–Gymnasium noch kein Schulcurriculum für den ITG–Unterricht der 8. Jahrgangsstufe gibt, beschränken sich die Betrachtungen somit auf den Bildungsplan und die Fachliteratur.

In dieser Unterrichtseinheit geht es vorwiegend (aber nicht ausschließlich) darum, Probleme zu lösen. Ein Problem ist nach Hubwieser in [Hub00] durch drei Komponenten gekennzeichnet: durch einen unerwünschten *Anfangszustand*, einen erwünschten *Zielzustand* und einer „Barriere, die die Überführung des Anfangszustands in den Zielzustand im Augenblick verhindert“. Des Weiteren könne die Informatik (in diesem Kontext gilt diese Aussage ebenso für die ITG) „besonders wertvolle Beiträge zur Problemlösefähigkeit der Schüler leisten“. Dies stärkt die methodischen Kompetenzen der Schüler, wie es in den Leitgedanken der ITG im Bildungsplan 2004 gefordert wird. Des Weiteren heißt es darin: „Die Schülerinnen und Schüler können...“

1. „Programme oder Programmiersprachen zur Berechnung und Lösung entsprechender Probleme einsetzen [...]“,

2. „verschiedene Strategien anwenden, um mit informationstechnischen Methoden angemessene Probleme zu lösen, und diese beurteilen“.

Somit werden die Inhalte der UE, die in Abschnitt 2.3.4 genannt werden, durch den aktuellen Bildungsplan gerechtfertigt.

Da in der Orientierungsstufe üblicherweise auch ITG unterrichtet wird, halte ich es noch für erwähnenswert, dass dort ein Einstieg in die Programmierung nur mit weitaus größeren Einschränkungen möglich ist, da die dazu notwendigen formalen Operationen nach Piaget (1975) frühstens in der 7. Jahrgangsstufe erfolgreich vermittelt werden könnten [Hub00]. Diese UE ist demnach nicht ohne Weiteres auf die 5. oder 6. Klassenstufe übertragbar. Einen frühen Einstieg in die Programmierung (für die 6. Klassenstufe) wird in [PB04] anhand der Lern–/Programmierumgebung *Karol* beschrieben.

2.3 Didaktische Analyse

Die didaktische Analyse vollzieht sich nach Baumann ([Bau96]) in zwei Schritten: Zuerst muss ein *didaktischer Handlungsspielraum* ermittelt werden. Dieser besteht aus einer „Menge mehrerer Alternativen, aus der eine optimale Wahl zu treffen ist“. In der folgenden *Sachanalyse* werden dann mögliche Lernziele und –inhalte ermittelt, die „nun im Hinblick auf die konkrete Unterrichtssituation bewertet werden“. Das Ergebnis dieses Prozesses ist dann die begründete Auswahl einer Alternative des Handlungsspielraums.
Das weitere Vorgehen in dieser Arbeit müsste demnach wie folgt aussehen:

1. didaktischen Handlungsspielraum klären, d.h. mehrere Zugänge zur Programmierung herausarbeiten,

2. sämtliche Lernziele und –inhalte der UE im Rahmen einer ausführlichen Sachanalyse detailliert erarbeiten,

3. sich aufgrund der resultierenden Lernziele und –inhalte für einen bestimmten Zugang und damit für ein konkretes Vorgehen entscheiden.

An dieser Stelle sollte jedoch angemerkt werden, dass sich diese didaktische Vorgehensweise eher in der Stundenplanung als in der Planung einer gesamten UE anwenden lässt. Dort können nämlich auch noch andere Faktoren eine Rolle spielen, die diesen strengen Analyseprozess u.U. etwas „aufweichen“. So habe ich mich beispielsweise von vornherein für einen konkreten Zugang zur Programmierung entschieden (Hamster–Modell), ohne vorher sämtliche Lern–/Programmierumgebungen *detailliert* zu vergleichen. Die Gründe dafür waren:

- Ich habe (durch Hospitation) bereits positive Erfahrungen mit diesem Modell gesammelt.
- Es gibt für einige Themen bereits sinnvolles und unterrichtserprobtes Material. Und dieser Aspekt ist fundamental wichtig, denn man kann davon ausgehen „[...] dass der Lernerfolg ganz wesentlich von der Wahl der richtigen Aufgaben für die Schülertätigkeiten abhängt“ [SS04]. Außerdem ist „besonderes Gewicht auf prägnante Beispiele zu legen“.
- Ich wollte dieses Modell einmal selbst im Unterricht ausprobieren und die existierenden Unterrichtsmaterialien sinnvoll ergänzen, um im Anschluss eine didaktische Gesamtbewertung des Modells durchführen zu können.

Aus diesem Grund werde ich nun wie folgt vorgehen. Zunächst werde ich verschiedene Zugänge zur Programmierung vorstellen. Ich halte mich dabei eng an [SS04]. Dann wird das Java–Hamster–Modell kurz vorgestellt. Anschließend werde ich mehrere Lern-/Programmierumgebungen miteinander vergleichen. Zuletzt werden sämtliche Lernziele und –inhalte sowie der Stoffverteilungsplan erarbeitet und genannt. Da ich mich, wie oben beschrieben, bereits von vornherein für das Hamster–Modell entschieden habe, sind die Ausführungen in den Abschnitten 2.3.1 und 2.3.3 auf ein Minimum reduziert.

Am Schluss sei noch angemerkt, dass die Reihenfolge des hier durchgeführten Analyseprozesses (zuerst didaktische Analyse und anschließend methodische Vorüberlegungen) gerade im Fach ITG kritisch ist, da die Umsetzbarkeit der herausgearbeiteten Lerninhalte „stark von der geplanten Methodik“ abhängt [Hub00]. Bei Umkehrung dieser Reihenfolge würde man jedoch vor einem ähnlichen Problem stehen. Aus diesem Grund fließen bei der didaktischen Analyse bereits methodische Vorüberlegungen ein.

2.3.1 Verschiedene Zugänge zur Programmierung

In [SS04] werden folgende Zugänge zur Programmierung genannt:

Programmiersprachlich Kontinuierliche (bottom–up–) Aneignung immer komplexerer Sprachkonstrukte der Programmiersprache. Fokus auf Programmiersprache selbst und nicht auf Problemstellungen. *Vorteile*: systematisch, „viel Sprache in kurzer Zeit“. *Nachteile*: fehlende praktische Relevanz, reiner „Sprachenunterricht“, „Lernen auf Vorrat“.

Systemanalytisch Studium eines komplexen Softwaresystems, bestehend aus: Bedienung, Identifikation einzelner Softwaremodule, Analyse des Zusammenwirkens dieser Module und Modifikation des Systems. *Vorteile*: System steht im Vordergrund, „kommerzielle“ Vorgehensweise, projektorientiertes Vorgehen und Teamarbeit. *Nachteile*: intellektuell anspruchsvoll, enorme Vorarbeit des Lehrers in Bezug auf das zu analysierende System nötig.

Lern–/Programmierumgebungen Programmierung beschränkt sich meist auf die Steuerung virtueller Objekte, die auf dem Bildschirm visualisiert werden. *Vorteile*: stufenweiser Einstieg in die Programmierung, didaktische Reduktion sehr leicht möglich, berücksichtigt kognitive Voraussetzungen von Anfängern, „man sieht was man tut“, „es bewegt sich etwas“. *Nachteile*: späterer Umstieg auf eine „richtige“ Programmiersprache notwendig (jedoch nur im Rahmen der Informatik und nicht der ITG!).

Projektorientiert Für die eigentliche Programmierung irrelevant, nachzulesen in [SS04].

2.3.2 Das Java–Hamster–Modell

Dr. Dietrich Boles beschreibt sein an der Universität Oldenburg entwickeltes Hamster–Modell sehr treffend in nur drei Sätzen:
„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.“
Die Funktionsweise des Hamster–Modells ist ganz einfach. Zuerst konstruiert man ein (leeres) Territorium und platziert darin Mauern, Körner und den Hamster. Das Territorium befindet sich nun in seinem Initialzustand (siehe Abbildung 2.1).

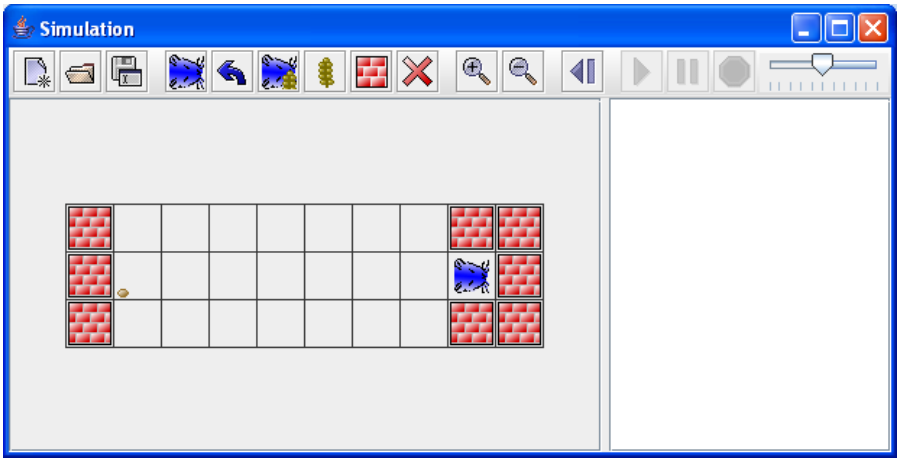


Abbildung 2.1: Beispiel eines Territoriums im Initialzustand

Anschließend überlegt man sich eine Aufgabe für den Hamsters bzw. den gewünschten Endzustand des Territoriums. In unserem Beispiel soll der Hamster das Korn finden und aufnehmen (siehe Abbildung 2.2).

Gesucht ist nun ein Programm, das die Aufgabe des Hamsters löst bzw. das Territorium von seinem Initialzustand in den Endzustand überführt. Dazu stehen dem Programmierer die vier,

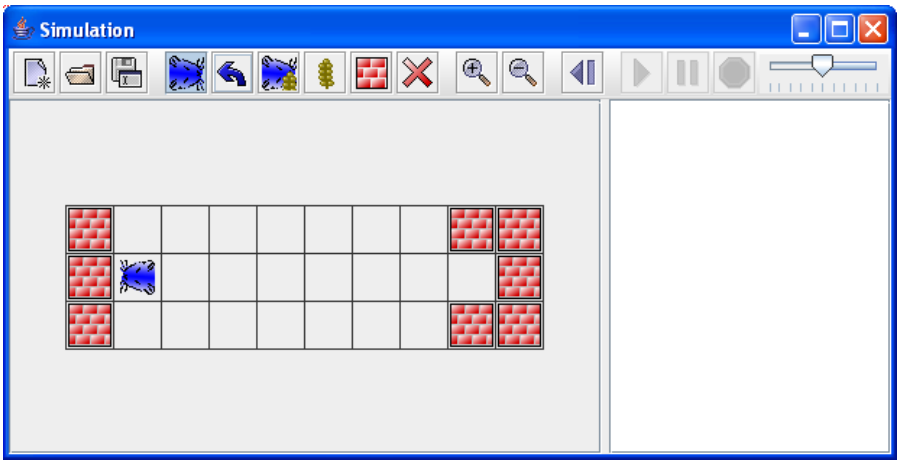


Abbildung 2.2: Territorium im gewünschten Endzustand

in Tabelle 2.1 dargestellten Befehle zur Verfügung sowie diverse Anweisungen der Programmiersprache JAVA.

Hamster–Befehl	Beschreibung
vor()	Hamster bewegt sich *genau* ein Feld vorwärts
linksUm()	Hamster dreht sich um 90° nach links
nimm()	Hamster nimmt *genau* ein Korn von aktuellem Feld auf
gib()	Hamster legt *genau* ein Korn auf aktuellem Feld ab

Tabelle 2.1: Die vier Hamster–Befehle

Ein Programm, welches die Aufgabe des Hamsters in diesem Beispiel löst, zeigt Abbildung 2.3.

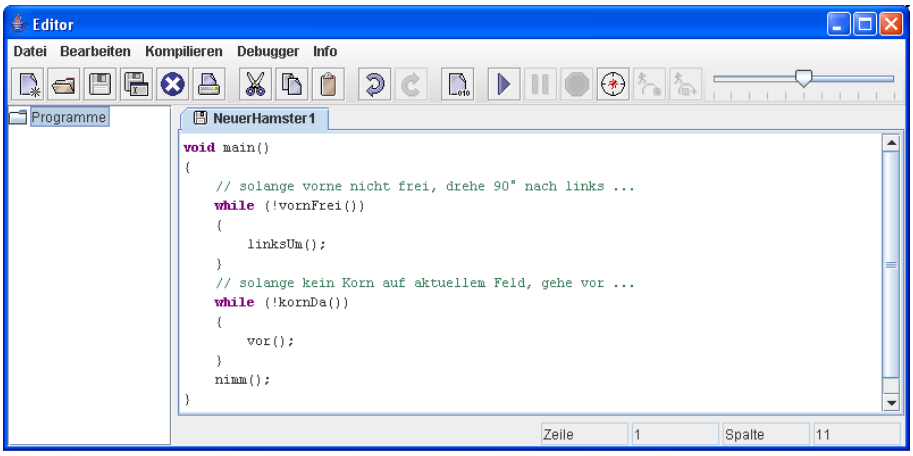


Abbildung 2.3: Programm, welches die Aufgabe des Hamsters löst

2.3.3 Vergleich verschiedener Lern–/Programmierumgebungen

Es gibt eine Vielzahl an Lern–/Programmierumgebungen. Diese alle zu beschreiben sowie deren Vor– und Nachteile gegeneinander abzuwägen, würde den Rahmen dieses Manuskripts

vollends sprengen. Außerdem ist die Entscheidung für die Verwendung des Java–Hamster–Modells bereits getroffen, sodass Tabelle 2.2 für eine grobe Übersicht einiger gängiger Lern–/Programmierumgebungen an dieser Stelle ausreichen soll.

Lern–/Programmierumgebungen		
	Einstieg in die Programmierung	Entdeckendes Lernen mit informatischen Methoden (u.a. Programmierung)
älter	Turtle (Delphi) Niki der Roboter (Pascal) Robi der Roboter (PRO) Karel the Robot (Pascal) Karel++ (C++, Java)	—
neuer	Java–Hamster (Java) [Bol02] JavaKara (Java) [RR04] Karol der Roboter (—) [PB04]	Kara [RR04] Squeak [BAC03]

Tabelle 2.2: Gängige Lern–/Programmierumgebungen

Sicherlich ist die Existenz der hier aufgeführten Lern–/Programmierumgebungen berechtigt. Man sollte jedoch unbedingt hinsichtlich der Zielgruppe differenzieren. Während man beispielsweise die Umgebung *Kara* in der Mittel– und Oberstufe einsetzen kann, weil sie u.a. die Themen *Programmierung*, *Automatentheorie* und *Turingmaschinen* abdeckt, ist *Karol der Roboter* eher für die Unterstufe konzipiert.

2.3.4 Lernziele und –inhalte

Bei der Ausarbeitung der Lernziele und –inhalte habe ich zunächst die Niveaukonkretisierungen des Bildungsplans 2004 (aufgeführt in [HPK]) zu Rate gezogen. Diese sind bis jetzt jedoch nur für die Klassenstufe 6 ausgearbeitet. Beim anschließenden Studium des „alten Lehrplans“ Baden-Württembergs von 1994/95 ([Bil94]) fiel mir zunächst auf, dass die ITG als eigenständiges Fach überhaupt nicht aufgeführt wurde. Die Ausführungen bezüglich der informationstechnischen Grundkenntnisse im Lehrplan Mathematik der 8. Klassenstufe brachten mich in meinen Überlegungen ebenfalls nicht voran. Lediglich die Begriffe „Algorithmus“, „Programmiersprache“ und „Programm“ wurden genannt. Zuletzt habe ich mir die alten Lehrpläne und den neuen Bildungsplan 2004 für das Fach Informatik betrachtet. Auch hier gab es keine nennenswerten Erkenntnisse, welche man ohne Weiteres auf die ITG übertragen könnte.

Aus diesen Gründen habe ich den resultierenden pädagogischen Freiraum genutzt und Lernziele sowie –inhalte selbst ausgearbeitet. Diese werden im Folgenden genannt.

Allgemeine Lernziele

Die einzelnen Lernziele sind bereits den, vom Bildungsplan 2004 ([Bil04]) genannten Kompetenzen (fachliche, methodische, personale und soziale Kompetenz), zugeordnet. Dabei ist zu

beachten, dass einige Lernziele prinzipiell mehreren Kompetenzen zugeordnet werden können. Die Anwendung des *Top–Down–Verfahrens* beispielsweise stärkt sowohl die fachliche als auch die methodische Kompetenz der Schüler. Im Rahmen des ITG–Unterrichts würde ich das Top-Down–Verfahren jedoch eher dem methodischen Bereich zuordnen. (Im Informatik–Unterricht verschiebt sich diese Gewichtung u.U. eher in Richtung des fachlichen Bereichs.)

Des Weiteren sind die folgenden Lernziele sehr allgemein gehalten. Sie müssen deshalb in den jeweiligen Stundenentwürfen unbedingt präzisiert werden.

1. Fachliche Kompetenzen

   - Die Schüler sind mit den Grundzügen einer imperativen Programmiersprache vertraut und in der Lage, einfache (Hamster–)Programme zu entwickeln. Außerdem verstehen sie die Notwendigkeit der Übersetzung (*Compilation*) eines Programms in Maschinencode.
   - Die Schüler lernen die fundamentale Bedeutung des Algorithmen–Begriffs kennen und verstehen, dass jeder Algorithmus aus den *Kontrollstrukturen* Sequenz, Wiederholung und Alternative aufgebaut ist. Des Weiteren können sie Algorithmen in unterschiedlichen Darstellungsformen formulieren.
   - Die Schüler können sich in unbekannte Anwendungen (Hamster–Simulator) einarbeiten und diese bedienen.

2. Methodische Kompetenzen

   - Die Schüler verbessern ihre Fähigkeiten Probleme zu lösen und verstehen, dass die Lösung eines Problems (zumindest im informatischen Sinne) aus den Schritten Analyse, Entwurf, Implementierung und Test besteht.
   - Die Schüler sind mit der Top–Down–Methode vertraut und wenden diese im Algorithmenentwurf an.
   - Die Schüler lernen ihre Gedanken zu strukturieren, sodass sie durch systematisches Denken reproduzierbare Ergebnisse am Rechner erhalten.

3. Personale Kompetenzen

   - Durch die strenge Einhaltung von Formalismen (der Programmiersprache) und vereinbarten Regeln lernen die Schüler sorgfältiges und konzentriertes Arbeiten.
   - Der Problemlöse–Prozess erhöht die Kreativität und Ausdauer der Schüler.

4. Soziale Kompetenzen

   - Durch die Arbeit in Kleingruppen erhöht sich die Teamfähigkeit der Schüler, und sie tolerieren Ideen, Vorschläge und Lösungen von Mitschülern.

Lerninhalte

In Tabelle 2.3 sind sämtliche Lerninhalte der UE aufgeführt. Die Auflistung wird ergänzt durch *wünschenswerte* Inhalte, die eventuell noch eingeschoben werden können oder zur Vertiefung nützlich sind. Außerdem werden Inhalte genannt, die zwar thematisch sinnvoll wären, jedoch keinesfalls behandelt werden sollen.

Lerninhalte	**unbedingt**	**wünschenswert**	**auf keinen Fall**
EINARBEITUNG IN DEN HAMSTER–SIMULATOR			
Konstruktion und Verwaltung von Hamster–Territorien	★		
Erstellung, Verwaltung, Compilation und Ausführung von Programmen	★		
ALGORITHMEN			
Beispiele für Algorithmen	★		
Grundstrukturen von Algorithmen	★		
Definition des Algorithmen-Begriffs	★		
Algorithmen umgangssprachlich formulieren	★		
PROGRAMMIERUNG			
Befehlssequenzen mit Hamster-Befehlen	★		
Prozedurkonzept	★		
WHILE-Schleife als Beispiel für eine Wiederholungsanweisung	★		
FOR-Schleife			★
DO..WHILE-Schleife		★	
geschachtelte WHILE-Schleifen		★	
IF-ELSE-Anweisung als Beispiel für eine Alternativanweisung	★		
syntaktische und semantische Fehler		★	
Variablenkonzept, Datentypen			★
Prozeduren mit formalen Parametern			★
Debugging			★
FORMALISMEN			
Einrücken des Quellcodes	★		
ausführliche Dokumentation			★
Prozeduren kommentieren	★		
Speichern von Territorien/Programmen (Speicherort & Namensgebung)	★		
Compilation bei der Programmerstellung alle 5-10 Zeilen	★		
PROBLEME LÖSEN			
Problemlöse–Prozess (Analyse, Entwurf, Implementierung, Test)	★		
Top–Down–Methode	★		
Struktogramme		★	

Tabelle 2.3: Lerninhalte der UE

2.3.5 (Geplanter) Stoffverteilungsplan

Nachdem die allgemeinen Lernziele und –inhalte herausgearbeitet wurden, müssen diese noch in Form eines Stoffverteilungsplans *sequentialisiert* werden. Auch Baumann bemerkt in [Bau96], dass „[...] das zentrale Problem der Inhaltsplanung [...] in der *Sequentialisierung*, d.h. der zeitlichen Strukturierung der (in den fachspezifischen Richtzielen benannten) Lerninhalte“, bestünde.

Es sei hier darauf hingewiesen, dass es sich in Tabelle 2.4 um den *geplanten* Stoffverteilungsplan der UE handelt. In Kapitel 4 wird dieser mit dem in Abschnitt 3.1 angegebenen *tatsächlichen* Stoffverteilungsplan verglichen, um Verbesserungen bezüglich des Vorgehens herauszuarbeiten.

Stunde	Datum	Inhalte
1/2	26.04.06	Algorithmen, Compilation, Einführung in den Hamster–Simulator
3/4	03.05.06	Entwurf einfacher Hamster–Programme, Prozedurkonzept, Top–Down–Methode
5/6	10.05.06	Prozedurkonzept, Top–Down–Methode, WHILE–Schleife
7/8	N.N.	WHILE–Schleife
9/10	17.05.06	IF–Anweisung
11/12	24.05.06	Wiederholung & vermischte Übungen

Tabelle 2.4: (Geplanter) Stoffverteilungsplan

2.4 Methodische Vorüberlegungen

Nachdem nun geklärt wurde, *was* den Schülern vermittelt werden soll (Lerninhalte), stellt sich nun die Frage nach dem *Wie*: Auf welche Art und Weise bzw. mit welchen Methoden können diese Lerninhalte, unter Berücksichtigung des fachlichen und methodischen Vorwissens der Schüler sowie der kognitiven Voraussetzungen derselben, vermittelt werden?

Die Reformpädagogik fordert hier den Einsatz einer Vielfalt an Methoden, die den klassischen lehrerzentrierten Klassenunterricht zwar nicht ersetzen, jedoch sinnvoll ergänzen. Durch Variation der *Sozialformen* (Klassenunterricht, Einzelarbeit, Gruppenarbeit) und *Aktionsformen* (darbietend, erarbeitend, entdecken lassend) sowie der Verwendung methodischer Großformen (Freiarbeit, Projektarbeit, ...) sollen u.a. die Selbstätigkeit und Entscheidungsfähigkeit der Schüler gestärkt, ganzheitliches Lernen gefördert und eine sinnvolle innere Differenzierung vorgenommen werden. (Weiterführende Informationen findet man u.a. in [Huw04b], [Kli04] und [Mat03].)

Eine präzise Auswahl einzelner Methoden für die UE wäre an dieser Stelle sicherlich nicht sinnvoll. Das detaillierte methodische Vorgehen und die Angabe konkreter Unterrichtsmethoden ist Teil der einzelnen Stundenentwürfe und wird deshalb in Kapitel 3 angegeben. Im Folgenden beschreibe ich daher methodische Entscheidungen, die entweder die gesamte UE betreffen oder die sich meiner Meinung nach für den ITG-Unterricht allgemein bewährt haben.

2.4.1 Grundlegende Sozial– und Arbeitsformen

Ich wende in dieser UE drei verschiedene Unterrichtsformen an:

Klassenunterricht (ohne Computer) – lehrerzentriert – darbietend, gelenkt oder entwickelnd Diese Unterrichtsform verwende ich in erster Linie in Situationen, in denen eine innere Differenzierung des Klassenverbands entweder nicht sinnvoll oder überaus schwierig ist. Beispiele:

1. Einführung in eine neue Problematik,
2. Erarbeitung grundlegender theoretischer Kenntnisse,
3. Wiederholung zentraler Inhalte,
4. Gemeinsame (exemplarische) Bearbeitung von (Programmier–)Aufgaben,
5. Besprechung von (durch die Schüler bereits gelösten) Aufgaben.

Erläuterungen zu 1-5: Für `(1)` verwende ich meist den Lehrerrechner und Beamer, um ein vorbereitetes Problem zu demonstrieren. Auch die Tafel kommt hierbei zum Einsatz. Für `(2)` benutze ich fast ausschließlich den Overheadprojektor. Die halbfertigen Arbeitsblätter (siehe Abschnitt 2.4.2) kopiere ich auf Folie und fülle sie mit den Schülern gemeinsam aus. Punkt `(3)` geschieht entweder mündlich oder mit der ausgefüllten Folie der letzten Stunde. Für `(4)` verwende ich die Tafel und nicht den Lehrerrechner. Wenn man einfach nur Programmcode in den Lehrerrechner eingibt, schalten die Schüler aus meinen Erfahrungen schnell ab. Die Bearbeitung an der Tafel ist viel dynamischer, und man kann auch besser auf Schülerfragen eingehen oder Varianten aufzeigen. Der Nachteil ist jedoch, dass man das entwickelte Programm nicht ausführen kann. Punkt `(5)` erfolgt entweder durch einen Schüler, der seine Lösung vorstellt, oder durch den Lehrer, der wiederum seine eigene oder eine Schülerlösung vorführt.

Gruppenarbeit/Partnerarbeit (ohne Computer) – schülerzentriert – entdeckend Diese Form wende ich an, wenn sich *alle* Schüler etwas tiefer in die Thematik einarbeiten sollen. Meistens werden sie dabei mit neuen Problemen konfrontiert, oder sie haben noch gar nicht alle nötigen theoretischen Erkenntnisse zur Lösung des Problems. Die Schüler dürfen sich deshalb in Gruppen organisieren und selbstständig, auf entdeckende Art und Weise, arbeiten.

Einzelarbeit am Computer – reproduzierend, problemorientiert und entdeckend Diese Arbeitsform ist in der ITG wahrscheinlich am Häufigsten anzutreffen. Ich versuche sie auf ein Mindestmaß (ca. 50%–60% der Unterrichtszeit) zu reduzieren. Natürlich sollen die Schüler möglichst viel mit dem Computer arbeiten. Es darf jedoch nicht in einem reinen *trial and error* enden. Diese, von vielen Schülern praktizierte Arbeitsweise, ist, in geringen Maßen gehalten,

zwar wünschenswert, kann jedoch sehr schnell in einem unsystematischen und nicht reflektierenden Umgang mit dem Rechner führen, in welchem die Ergebnisse und Verfahrensweisen der Schüler nicht reproduzierbar sind. Aus diesem Grund muss man die Schüler zu größter Sorgfalt anleiten und auf die Einhaltung von vereinbarten Regeln (bzgl. der Arbeitstechniken und –prozesse) bestehen, ohne ihnen jedoch dabei den Spaß an der Sache zu nehmen. Und dies kann (für den Klassenverband) nur sehr bedingt am Rechner geschehen.

Des Weiteren möchte ich anmerken, dass ich Partnerarbeit am Computer nicht zulasse. Die Schüler dürfen jederzeit ihren Nachbarn um Rat fragen oder Aufgaben mit ihm besprechen. Jeder Schüler muss jedoch ein eigenes Programm entwickeln und in der Lage sein, dieses zu erklären.

2.4.2 Die Arbeitsmappe

Ich habe mich entschieden, für diese UE eine Arbeitsmappe anlegen zu lassen, in welche sämtliche ausgeteilten Unterlagen von den Schülern abgeheftet werden. Die Gründe für diese Entscheidung waren:

1. Bei den Vorbereitungen dieser UE habe ich sehr schnell bemerkt, dass ich relativ viel Stoff in wenig Zeit durchnehmen will. Das darf natürlich niemals zu Lasten der Schüler geschehen. Deshalb habe ich eine Vielzahl von Arbeitsmaterialien erstellt und diese sorgfältig didaktisch aufbereitet. Die Schüler müssen somit keine aufwendigen Tafelbilder mehr abschreiben, sondern nur noch vereinzelt halbfertige Arbeitsblätter ausfüllen. Dies spart wertvolle Zeit, ohne dass das Verständnis oder die Vollständigkeit darunter leiden.

2. Das Arbeitsmaterial beläuft sich auf insgesamt 52 DIN–A4–Seiten. Ohne Arbeitsmappe würden diese schnell verloren gehen.

3. Die Schüler erarbeiten sich ein *Produkt*, welches man „anfassen kann“ und sich „zu Hause ins Regal stellen kann“. Sie sehen, dass die Arbeit zu einem Ergebnis führt, was zudem noch optisch ansprechend ist.

Zur Arbeitsmappe:

- Der Aufbau gleicht dem eines Buches. Sie besteht aus einem Deckblatt, Inhaltsverzeichnis, den ausgeteilten Blättern, einem Glossar und einem Literaturverzeichnis. Ich erhoffe mir dadurch, dass dieser Umstand den Stellenwert der Mappe bei den Schülern erhöht.

- Die Seiten sind von vornherein nicht durchnummeriert, weil ich es mir offen halten wollte, jederzeit Blätter zu ergänzen bzw. die Reihenfolge zu ändern. Ich ergänze jedoch bei jedem neu ausgeteilten Blatt das (zunächst leere) Inhaltsverzeichnis um den Titel dieses Blattes. Dadurch ist eine eindeutige Reihenfolge gewährt.

- Es wird streng differenziert in:
  - *Arbeitsblätter*: Einführung in eine neue Problematik/Thema. Die Blätter sind lückenhaft und werden im Lehrer–Schülergespräch (auf dem OHP) ergänzt und ausgefüllt.
  - *Übungsblätter*: Meist Programmieraufgaben, die am Rechner gelöst werden sollen.
  - *Merkblätter*: Faktenwissen — kurz und übersichtlich.
  - *Tutorials*: Selbstständige Einarbeitung in den Hamster–Simulator.
  - *Musterbeispiele*: Diese dienen als Vorlage für die allgemeine Vorgehensweise und die Syntax der Programmiersprache, und ich halte sie für sehr wichtig, damit die Schüler etwas haben, an dem sie sich orientieren können. Dadurch werden viele Erläuterungen des Lehrers hinfällig, und die Schüler können eigenverantwortlicher und selbstständiger arbeiten.
  - *Vertiefende Aufgaben*: Sind für besonders leistungsstarke und schnelle Schüler gedacht. Wichtig hierbei ist: Es handelt sich um *weiterführende*, die Thematik *vertiefende* Aufgaben, die dem Schüler einen tieferen Einblick in das Thema gewähren. Damit soll zusätzliches Interesse geweckt werden. Reine *Zusatzaufgaben* erwecken beim Schüler jedoch den Eindruck: „Je schneller und je besser ich bin, desto mehr muss ich arbeiten!“.
  - *Zusatzaufgaben*: Zusätzliche Aufgaben, die jedoch nur dann gestellt werden, wenn sich Vertiefungsaufgaben aus thematischer Sicht nicht anbieten.

3 Unterrichtspraktische Umsetzung

In diesem Kapitel wird die unterrichtspraktische Umsetzung der Unterrichtseinheit (UE) beschrieben. Zunächst wird der tatsächliche Stoffverteilungsplan angegeben. Diesem folgt die Dokumentation der durchgeführten Unterrichtsstunden.

3.1 (Tatsächlicher) Stoffverteilungsplan

Um eine Ausdehnung der UE über Pfingsten hinaus zu vermeiden, wurde mit den Schülern vereinbart, dass am 10.05.06 und am 17.05.06 jeweils drei Stunden ITG unterrichtet werden — die ersten beiden und die siebte Stunde. Diese „Triplestunden“ wurden bei den folgenden Betrachtungen zusammenhängend (also wie eine Doppelstunde) dokumentiert.

Stunde	Datum	Inhalte
1/2	26.04.06	Algorithmen, Compilation, Einführung in den Hamster–Simulator
3/4	03.05.06	Entwurf einfacher Hamster–Programme, Prozedurkonzept, Top–Down–Methode
5/6/7	10.05.06	Prozedurkonzept, Top–Down–Methode, WHILE–Schleife
8/9/10	17.05.06	WHILE-Schleife, geschachtelte WHILE–Schleifen, IF–Anweisung
11/12	24.05.06	IF–Anweisung, Struktogramme, Wiederholung & vermischte Übungen

Tabelle 3.1: (Tatsächlicher) Stoffverteilungsplan

3.2 Durchführung der Unterrichtseinheit

Im Folgenden werden die durchgeführten Stunden detailliert beschrieben. Neben Angabe der verwendeten Literatur werden für jede Stunde die Lernziele und der Unterrichtsverlauf angegeben. Jede Stundendokumentation schließt mit einer kurzen Reflexion ab, die sich jedoch nur auf diese Unterrichtsstunde bezieht. Die Unterrichtsverläufe sind bereits in einzelne Phasen gegliedert und mit den zusätzlichen Angaben *(Lernziel dieser Phase, Sozialform, Arbeitsform, eingesetzte Medien)* versehen.

3.2.1 Stunde 1 & 2

`Literatur:` [KB06], [PB04], [Bol02], [UHe], [Bau92].

Lernziele

Die Schüler

- LZ1 erkennen, auf welche Art und Weise sie bisher Probleme mit dem Computer gelöst haben (nämlich mit Anwender–Software),
- LZ2 kennen den Begriff und die Definition des Algorithmus' und sind in der Lage, Algorithmen sowohl umgangssprachlich als auch in einer vorgegebenen Normsprache zu formulieren,
- LZ3 wissen, dass die eingeführte Normsprache nur ein Kompromiss zwischen Mensch und Maschine darstellt und können die Notwendigkeit der Compilation begründen,
- LZ4 sind mit der Bedienung des Simulatorfensters des Hamster–Modells vertraut und können Territorien konstruieren und verwalten.

Unterrichtsverlauf

Erarbeitungsphase 1: Probleme lösen mit dem Computer
(LZ1–LZ2, `Klassenunterricht, entwickelnd, Tafel als Plakatwand`)

Zu Beginn der Stunde bekamen die Schüler je ein farbiges DIN–A5–Blatt, auf dem sie in zwei bis drei Stichworten ein Thema, welches im Laufe des Jahres in ITG behandelt wurde, notierten. Die Blätter wurden mit Magnetpins an die Tafel geheftet und von mir stillschweigend (je nach Thematik) sortiert, wobei ich den Begriff HTML separierte. Ein paar Schüler sollten nun aus einem beliebigen Themenbereich (außer HTML), ein im Unterricht behandeltes Problem bzw. eine Aufgabe nennen und das Verfahren beschreiben, welches zur Lösung des Problems führte. Die Schüler erkannten schließlich das bisher angewandte Verfahren: Die schrittweise und symbolische Interaktion (durch Buttons, Menüs, etc.) mit dem Computer führt iterativ zur Lösung (LZ1). Nun forderte ich einen Schüler auf, die Erstellung einer Website (mit der Sprache HTML) zu skizzieren. Dabei erkannten die Schüler, dass das eigentliche „Lösungsverfahren" aus der Erstellung eines HTML-Skripts besteht, welches im Anschluss „komplett an den Rechner übergeben und ausgeführt wird". Der Lösungsweg ist also nicht mehr rein symbolisch und iterativ. Diese Vorgehensweise (Aneinanderreihung textueller Befehle) führte schließlich auf den Algorithmen–Begriff, welchen ich zunächst nur mündlich nannte und beschrieb (LZ2).

Erarbeitungsphase 2 & Übungsphase 1: Algorithmen formulieren
(LZ2, `Klassenunterricht, Rollenspiel, schülerzentriert, Tafel`)

Als nächstes sollten die Schüler einfache Algorithmen umgangssprachlich formulieren. Dies geschah sofort im Kontext des später eingeführten Hamster–Modells. Da auf der Tafel bereits Karos vorhanden waren, konstruierte ich mit Magnetpins unterschiedlicher Farbe eine Hamsterlandschaft, bestehend aus Mauern, Körnern und dem Hamster. Ein Schüler übernahm nun

die Rolle des Computers und ließ sich von der Klasse (den Programmierern) Instruktionen erteilen (Lz2), die er schrittweise ausführte. Ich schrieb die genannten Instruktionen an die Tafel. Genannt wurden beispielsweise: „gehe vor“, „drehe dich rechts“, „gehe fünf Schritte vor“, „gehe vor bis zur Mauer“, „nimm zwei Körner“ und viele andere. Als nächstes schränkte ich die Menge aller erlaubten Instruktionen ein. Somit erhielt ich eine *Normsprache*, die nur noch aus den Anweisungen *vor*, *linksUm*, *nimm* und *gib* bestand. Damit sollten die Schüler nun das gleiche Problem erneut lösen (Lz2), was auch ohne Probleme funktionierte. Ich erklärte den Schülern den Vorteil einer normierten Sprache — dass diese nämlich eindeutig interpretierbar ist und somit von einem Computer prinzipiell verarbeitet werden könnte. Des Weiteren machte ich sie (in Anlehnung an HTML) darauf aufmerksam, dass nicht jede geschriebene Anweisung sofort vom Computer ausgeführt wird, sondern dass dieser den gesamten Algorithmus „am Stück“ ausführt.

Erarbeitungsphase 3 & Übungsphase 2: Notwendigkeit der Compilation
(Lz3, `Klassenunterricht, Rollenspiel, schülerzentriert, Tafel`)

Ich forderte nun zwei Schüler — ein Computer und (wie sich später herausstellte) den Compiler — auf, den Raum zu verlassen. Die Klasse (Programmierer) entwickelte nun gemeinsam einen Algorithmus (in der Normsprache) für ein neues Hamster–Problem. Dazu ergänzte jeder Schüler der Reihe nach eine Anweisung des Algorithmus. Das Ergebnis war ein Blatt mit dem Lösungsalgorithmus. Ich übertrug diesen Algorithmus an die Tafel, übersetzte ihn dabei jedoch in die englische Sprache. Anschließend betraten die beiden Schüler wieder das Klassenzimmer. Ich erklärte, dass der Computer selbst die entwickelte Normsprache nicht verstehen kann. Aus diesem Grund benötigt er einen Übersetzer, der den formulierten Algorithmus in die Maschinensprache („Nullen und Einsen“) übersetzt. Dabei wird zusätzlich der in Normsprache formulierte Algorithmus auf korrekte „Rechtschreibung und Grammatik“ überprüft (Lz3). Der Compiler (Schüler 2) übersetzte nun den in Englisch formulierten Algorithmus in die (deutsche) Normsprache. Anschließend führte der Computer (Schüler 1) diesen aus.

Sicherungsphase 1: Bearbeitung von Arbeitsblatt 1
(Lz1–Lz3, `Klassenunterricht, fragend-gelenkt, Arbeitsblatt & OHP`)

Nun wurden die bisherigen Ergebnisse auf dem `Arbeitsblatt 1` (Anhang D, Seite 55 ff.) gesichert.

Erarbeitungsphase 4: Einführung in den Hamster–Simulator
(Lz4, `Klassenunterricht & Einzelarbeit, Lehrervortrag & entdeckend, Tutorial & Computer`)

Danach demonstrierte ich den Hamster–Simulator auf dem Lehrerrechner. Dazu konstruierte ich das Territorium des Tafelbeispiels und schrieb das entsprechende Programm dafür. Dieses compilierte ich und führte es im Simulator aus.

Den Rest der Stunde bearbeiteten die Schüler selbstständig den ersten Teil des `Tutorials` (Anhang D, Seite 59 ff.) für den Hamster–Simulator. Darin wurde ihnen die Konstruktion und Verwaltung (öffnen, speichern, Speicherort) von Territorien erläutert (Lz4).

Reflexion der Stunde

Erarbeitungsphase 1: Die Schrift auf den farbigen DIN–A5–Blättern war bei fast allen Schülern viel zu klein. Man müsste die Schüler künftig darauf hinweisen, größer zu schreiben.

Erarbeitungsphase 2 & Übungsphase 1: Das Rollenspiel hat bei den Schülern für Aufmerksamkeit gesorgt. Sie wollten unbedingt wissen, was es damit auf sich hat und beteiligten sich rege daran. Ein Mal schweiften wir jedoch etwas vom Thema ab, weil ich zu viel aus den Schülern herausholen wollte. Bemerkenswert fand ich auch den Kommentar einer Schülerin: „Warum verwenden wir nicht *solange vorne frei ist, gehe vor* statt fünf Mal *vor*? Dann wäre es nämlich egal, wie weit die Mauer vom Hamster entfernt ist!“.

Sicherungsphase 1: Die gemeinsame Bearbeitung von Arbeitsblatt 1, Seite 1 war etwas problematisch, da ich wieder versucht habe, mehr aus den Schülern herauszuholen als möglich war. Dies erkannte ich jedoch schnell und wechselte von einem fragend–gelenkten Gespräch in die Vortragsform. Die Seiten 2 und 3 konnten wieder problemlos von den Schülern bearbeitet werden, da auf diesen Seiten keine Lücken vorhanden waren. Seite 4 habe ich aufgrund des hohen Detaillierungsgrads nicht besprochen. (Aus diesem Grund steht dort auch die Anmerkung „Für Spezialisten“.) Außerdem stieg aufgrund der längeren einführenden Theoriephase der Unruhepegel. Dieser legte sich jedoch schnell wieder, als ich ankündigte, dass die Schüler gleich an die Rechner dürften.

Erarbeitungsphase 4: Bei der Ausarbeitung der Tutorials habe ich mir sehr viel Zeit genommen, weil ich von den Schülern (auch im Hinblick auf die späteren Übungen) ein systematisches Vorgehen verlangte. Um so mehr war ich erfreut, dass *ausnahmslos alle* Schüler das Tutorial Schritt für Schritt bearbeiteten und sich darüber hinaus an die geforderten Konventionen hielten. Aus diesem Grund versuche ich die restlichen Arbeitsmaterialien für diese UE mindestens genauso gut auszuarbeiten.

Insgesamt war ich mit dem Verlauf dieser Stunde zufrieden. Ich nehme mir für die nächsten Stunden allerdings vor, im Voraus noch detaillierter zu analysieren, was die Schüler wissen können und was nicht. In jenen Situationen, in denen ein fragend–gelenktes Lehrer–Schülergespräch in einem (etwas überspitzt formulierten) „Ratespiel“ enden könnte, in welchem Schülerantworten und –fragen u.U. von der eigentlichen Thematik ablenken, werde ich mich künftig für einen kurzen und prägnanten Lehrervortrag entscheiden.

3.2.2 Stunde 3 & 4

`Literatur:` [Bol02], [MS99].

Lernziele

Die Schüler

- LZ1 können einfache Hamster–Programme mit den vier Grundbefehlen *vor, linksUm, nimm* und *gib* schreiben,
- LZ2 halten sich an die vereinbarten Konventionen beim Erstellen (des Quellcodes) von Programmen,
- LZ3 sind in der Lage, ihre Programme zu verwalten — wissen also wie und wo man die Programme speichert (und sie auch wieder findet!),
- LZ4 kennen das Prozedurkonzept und können es beim Programmentwurf anwenden,
- LZ5 sind vertraut mit der Top–Down–Methode und wenden diese bei der Problemlösung an.

Unterrichtsverlauf

ORGANISATIONS– UND WIEDERHOLUNGSPHASE: ALGORITHMEN, ARBEITSMAPPE
(LZ2–LZ4 d. letzten Std., `Klassenunterricht,fragend-gelenkt,Tafel & Arbeitsmappe`)

Zu Beginn der Stunde teilte ich den Schülern die Arbeitsmappen (blauer Leitz-Ordner) aus. Diese enthielten bereits das `Deckblatt` (Anhang D, Seite 52), das `Rahmendatenblatt` (Anhang D, Seite 53), das leere `Inhaltsverzeichnis` (Anhang D, Seite 54) sowie das leere `Glossar` (Anhang D, Seite 101 ff.) und das leere `Literaturverzeichnis` (Anhang D, Seite 103). Die Schüler waren begeistert von der Mappe. Einige fingen prompt an, den Hamster auf dem Deckblatt farbig anzumalen. Wir sortierten zunächst alle bis dahin ausgeteilten Blätter ein und ergänzten das Glossar (im Rahmen einer mündlichen Wiederholung der letzten Stunde) um den Begriff „Algorithmus“. Außerdem führte ich noch mal die Konstruktion eines Territoriums vor und ließ mir von den Schülern die diesbezüglich vereinbarten Konventionen nennen.

ERARBEITUNGSPHASE 1 & ÜBUNGSPHASE 1: DAS ERSTE HAMSTER–PROGRAMM
(LZ1–LZ3, `Einzelarbeit, reproduzierend & entdeckend, Computer & Tutorial`)

In dieser Phase bearbeiteten die Schüler den zweiten Teil des `Tutorials` (Anhang D, Seite 64 ff.). Im Rahmen dieser Einzelarbeit mussten die Schüler bereits ihre erste (im Tutorial gestellte) Programmieraufgabe selbstständig lösen (LZ1–LZ3). Dafür reichten die vier Hamster–Grundbefehle aus. Für besonders schnelle Schüler teilte ich die `Zusatzaufgabe 1` (Anhang D, Seite 71) aus. Außerdem erhielt jeder Schüler `Merkblatt 1` (Anhang D, Seite 70) mit den vier Grundbefehlen des Hamster–Modells.

Sicherungsphase 1 & Besprechungsphase 1: Konventionen, Besprechung
(Lz1–Lz3, `Klassenunterricht, schülerzentriert & Schülervortrag, Computer & Merkblatt`)

Wir versammelten uns wieder an den computerfreien Arbeitsplätzen, und ich teilte `Merkblatt 2` (Anhang D, Seite 72) aus, welches ein Schüler laut vorlas. Darin ging es ein letztes Mal um sämtliche bis dahin vereinbarten Konventionen bezüglich des Umgangs mit dem Simulator und der Programmentwicklung (Lz2, Lz3). Anschließend führte ein Schüler seine Lösung der Übungsaufgabe vor, und ich kommentierte an den wichtigen Stellen noch mal das allgemeine Vorgehen (Lz1).

Erarbeitungsphase 2: Prozedurkonzept und Top–Down–Methode
(Lz4–Lz5, `Klassenunterricht,fragend gelenkt & Lehrervortrag,Computer & Tafel`)

Diese Phase leitete ich mit der Frage: „Was könnte man an dem eben vorgestellten Programm noch verbessern bzw. was ist hier noch etwas umständlich?“ ein. Sofort meldeten sich einige Schüler, und nach und nach wussten immer mehr Schüler die Antwort auf die Frage, nämlich, dass es sinnvoll wäre, statt drei Mal *linksUm* besser ein Mal *rechtsUm* zu schreiben oder statt fünf Mal hintereinander *vor* besser *5vor*. Sofort führte ich den Vorschlag (*rechtsUm* statt drei Mal *linksUm*) am Lehrerrechner aus, was natürlich zu einer Fehlermeldung führte. Auch hier kamen nach kurzer Zeit Wortmeldungen: „Der Computer weiß noch gar nicht, wie *rechtsUm* eigentlich funktioniert. Er kann damit nichts anfangen.“ Daraufhin erklärte ich den Schülern anhand des `Musterbeispiels 1` (Anhang D, Seite 75) das Prozedurkonzept (Lz4). Des Weiteren vereinbarte ich mit ihnen ein neues Vorgehen für den künftigen Programmentwurf: Zuerst sollen sie geeignete (Makro–)Befehle herausarbeiten, die sie schneller zur Lösung des Problems führen — diese werden dann jeweils in Form einer Prozedur realisiert. Somit führte ich sie in die Top–Down–Methode (ohne jedoch den Begriff zu nennen) ein (Lz5).

Übungsphase 2: Programmentwurf mit Prozeduren
(Lz4–Lz5, `Einzelarbeit, schülerzentriert & problemorientiert, Computer & Übungsblatt`)

Den Rest der Stunde bearbeiteten die Schüler das `Übungsblatt 1` (Anhang D, Seite 76 ff.), welches aus zwei Aufgaben bestand. Dabei musste ich die Schüler des Öfteren darauf hinweisen, dass sie nach der Top–Down–Methode verfahren sollen, dass sie sich also zuerst sinnvolle (Makro–)Befehle überlegen, diese durch Prozeduren realisieren und erst anschließend das Hauptprogramm schreiben. Wenige Schüler schafften in der (geringen) verbleibenden Zeit beide Aufgaben, aber fast alle konnten die erste Aufgabe erfolgreich bearbeiten. Während der Übungsphase wurde auch die von mir erhoffte Frage: „Kann man Prozeduren auch ineinander schachteln?“ gestellt. Diese Frage gab ich an die Klasse zurück, und sie wurde schließlich geklärt.

Reflexion der Stunde

Erarbeitungsphase 2: Bereits hier kam schon die Frage eines Schülers: „Könnten wir nicht eine Prozedur *nimmAlle* schreiben?“ (führt auf WHILE-Schleife). Ich habe den Schüler damit vertröstet, dass wir diese Prozedur noch nicht schreiben könnten, dass der Einwand jedoch sehr gut und berechtigt war. An dieser Stelle wäre ein anderes Vorgehen vielleicht besser gewesen. Ich könnte den Schüler auffordern, die Prozedur *nimmAlle* zu schreiben, sodass die Klasse sieht, dass die bisherigen Konzepte nicht ausreichen und (zu einem späteren Zeitpunkt) ergänzt werden müssen. Des Weiteren haben sich die abschließenden Überlegungen der letzten Stunde bewährt: Zuerst habe ich im Lehrer–Schülergespräch die Schüler an das Thema herangeführt und habe in einem folgenden kurzen Lehrervortrag das Prozedurkonzept verdeutlicht. Dadurch schweiften wir im Gegensatz zur letzten Stunde nicht so sehr vom eigentlichen Thema ab.

Übungsphase 2: Vor dieser Übungsphase wollte ich ursprünglich eigentlich noch eine (theoretische) Sicherungsphase einbauen, habe im Unterricht jedoch gemerkt, dass die Schüler aufgrund der längeren Theoriephase wieder unruhig wurden. Deshalb habe ich die Übungsphase nach vorne geschoben und die Sicherungsphase auf die nächste Stunde verlegt.

Insgesamt bin ich mit dieser Stunde sehr zufrieden. Die Schüler waren fast alle in der Lage, die Übungsaufgaben *vollständig* zu lösen, und haben sich darüber hinaus auch an sämtliche eingeführten Konventionen gehalten.

3.2.3 Stunde 5 & 6 & 7

`Literatur:` [Bol02], [MS99].

Lernziele

Die Schüler

- Lz1 haben ihr Verständnis für die Top–Down–Methode und das Prozedurkonzept vertieft und können diese gezielt beim Problemlöse–Prozess einsetzen,
- Lz2 kennen die WHILE-Schleife als Beispiel einer Wiederholungsanweisung und haben (zunächst nur) den Vorteil (und noch nicht die Notwendigkeit) dieser Schleife erkannt,
- Lz3 sind in der Lage, die WHILE–Schleife beim Programmentwurf sinnvoll einzusetzen.

Unterrichtsverlauf

Organisations– und Wiederholungsphase: Prozedurkonzept,Glossar
(Lz4–Lz5 d. letzten Std., `Klassenunterricht, entwickelnd, Tafel & Lehrerrechner`)

Zunächst habe ich 15–20 Minuten zur Organisation der Arbeitsmappe verwendet. Wir haben das

Inhaltsverzeichnis und das Glossar um die Begriffe *Programm*, *Programmiersprache*, *Programmierer*, *Compiler*, *Binärcode* und *Maschinencode* ergänzt. Dabei diskutierten wir die Begriffe und verdeutlichten zudem deren starken Zusammenhang. Außerdem ergänzten wir das Literaturverzeichnis. Danach wiederholten wir anhand `Musterbeispiel 1` (Anhang D, Seite 75) gemeinsam das Prozedurkonzept und die Top–Down–Methode.

Sicherungsphase 1: Prozeduren
(Lz1, `Klassenunterricht, entwickelnd & darbietend, Arbeitsblatt & OHP`)
Nun füllten wir `Arbeitsblatt 2` (Anhang D, Seite 73 ff.) aus. Die erste Seite des Arbeitsblatts sowie die Vorteile von Prozeduren (auf Seite 2) ließ ich mündlich von den Schülern bearbeiten und hielt die Ergebnisse auf dem OHP fest. Um den Fehler der ersten Doppelstunde (mehr aus den Schülern herausholen, als möglich ist) nicht erneut zu begehen, erklärte ich Seite 2 des Arbeitsblattes und füllte sie währenddessen (ohne Schülerinteraktion) aus.

Besprechungsphase 1: Übungsblatt 1, Aufgabe 1
(Lz1, `Klassenunterricht,fragend-gelenkt & Schülervortrag,Tafel & Lehrerrechner`)
Bei der Besprechung von Aufgabe 1 des `Übungsblatts 1` (Anhang D, Seite 76 ff.) forderte ich die Schüler zunächst auf, sinnvolle (Makro–)Befehle zu nennen, die die Lösung des Problems vereinfachten. Diese schrieb ich an die Tafel. Es wurden genannt: *rechtsUm*, *zweiHoch2rechts*, *StufeHoch* und *zweiMalLinks*. Wir diskutierten zunächst die Befehle. Sehr schnell bemerkten einige Schüler, dass es nicht sinnvoll wäre, eine Prozedur *zweiMalLinks* zu definieren, weil es stattdessen besser wäre, zwei Mal *linksUm* zu schreiben. Alle waren sich einig, dass *rechtsUm* unbedingt verwendet werden sollte. Bei den anderen zwei Befehlen spaltete sich die Meinung — einige verwendeten *zweiHoch2rechts*, andere *StufeHoch*, und wiederum andere verwendeten keine von beiden Befehlen.
Als nächstes entwickelten wir gemeinsam die entsprechenden Prozeduren der drei verbleibenden Befehle, indem drei Schüler gleichzeitig jeweils eine Prozedur an die Tafel schrieben. Da ich die Verwendung von *rechtsUm* und *StufeHoch* als sinnvoll erachtete, wählte ich nun einen Schüler aus, der diese Prozeduren ebenfalls verwendete. Dieser stellte seine Lösung am Lehrerrechner vor.

Übungsphase 1: Übungsblatt 1, Aufgabe 2
(Lz1, `Einzelarbeit,schülerzentriert & problemorientiert,Computer & Übungsblatt`)
Nun folgte eine Übungsphase, in der die Schüler `Übungsblatt 1`, Aufgabe 2 bearbeiteten. Diejenigen, die diese Aufgabe bereits in der letzten Stunde gelöst hatten, bearbeiteten die Zusatzaufgabe auf demselben Übungsblatt. Dabei fiel mir auf, dass nun deutlich mehr Schüler von alleine mit der Top–Down–Methode arbeiteten. Die restlichen Schüler motivierte ich weiterhin, diese Methode zu verwenden.
Da fast alle Schüler die zweite Aufgabe sinnvoll lösten, verzichtete ich auf eine gemeinsame

Besprechung und fuhr direkt mit Erarbeitungsphase 1 fort.

Erarbeitungsphase 1: Die WHILE–Schleife
(Lz2, `Klassenunterricht & Partnerarbeit, schülerzentriert & entwickelnd, Tafel & Papier & Lehrerrechner`)

Ich konstruierte auf dem Lehrerrechner ein einfaches Territorium, bestehend aus einem einzigen Feld, dem Hamster und fünf Körnern. Dann bekamen die Schüler den Auftrag, in Partnerarbeit eine Prozedur *nimmAlle* zu schreiben. Alle Gruppen formulierten die gesuchte Prozedur als Sequenz aus fünf Mal *nimm*. Einige Schüler fragten ihre Nachbarn bzw. andere Gruppen (jedoch nicht mich!) was passiere, falls mehr oder weniger als fünf Körner auf dem Feld liegen. Als fast alle Gruppen fertig waren, modifizierte ich das Territorium und legte zwölf statt fünf Körner auf das Feld. (Bemerkung: Ich wollte an dieser Stelle noch nicht die Notwendigkeit, sondern nur den Vorteil einer Wiederholungsanweisung behandeln, um die Schüler nicht gleich zu überfordern. Ansonsten hätte ich nämlich weniger als fünf Körner auf das Feld gelegt, sodass die ursprüngliche Prozedur *nimmAlle* zu einem Laufzeitfehler geführt hätte.) Ich erteilte erneut den Auftrag, für dieses Szenario eine Prozedur *nimmAlle* zu schreiben. Nun protestierten (glücklicherweise) die Schüler: „Das ist doch das Gleiche, nur dass wir jetzt zwölf Mal *nimm* schreiben müssen!“ Eine Schülerin schlug vor, dass man zwei Mal die ursprüngliche Prozedur verwenden könne, und dann noch zwei Mal *nimm* ergänzen könne. Ich lobte sie für diese richtige und gute Bemerkung, kündigte jedoch an, dass es auch noch einfacher ginge. Dazu schrieb ich das Wort „solange“ an die Tafel und forderte die Schüler auf, die Prozedur *nimmAlle* umgangssprachlich, unter Verwendung dieses Wortes, zu formulieren. Dabei entstanden die aufregendsten Formulierungen — die richtige und gewünschte Lösung war auch dabei: „*Solange* noch Körner da sind, nimm eins!“ Ich erläuterte, dass wir jetzt nur noch diese Formulierung übersetzen müssten und schrieb die Lösung an die Tafel: „**while** (kornDa()) { nimm(); }“ (Lz2).
Anschließend bearbeiteten wir gemeinsam die Seiten 1 und 2 des `Arbeitsblattes 3` (Anhang D, Seite 78 ff.). Dabei sollten die Schüler für verschiedene Territorien die Prozeduren *nimmAlle, vorBisMauer* und *vorBisKorn* zuerst umgangssprachlich (mit „solange“) formulieren und die Formulierung anschließend in die Hamster–Sprache übersetzen (Lz2).

Übungsphase 2: Programmentwurf mit WHILE–Schleife
(Lz3, `Einzelarbeit,schülerzentriert & problemorientiert,Computer & Übungsblatt`)

Den Rest der Stunde bearbeiteten die Schüler Aufgabe 1 von `Übungsblatt 2` (Anhang D, Seite 82 ff.). Dabei kam es zum ersten Mal zu größeren Verständnisproblemen. Ein paar Schüler kamen ohne Hilfestellung meinerseits nicht weiter, andere brachten Prozeduren und WHILE–Schleifen durcheinander und verstanden nicht, dass beides völlig unabhängig voneinander existieren kann. Deshalb versammelte ich die Schüler noch einmal in der Mitte des Raumes, und wir besprachen zunächst die Aufgabe. Ich gab ihnen den Tipp, wieder streng nach der Top–Down–Methode vorzugehen und Arbeitsblatt 3 zu verwenden, da wir auf diesem Blatt bereits die

Prozedur *vorBisMauer* ausgearbeitet hatten. Daraufhin schafften es mehr Schüler, die Aufgabe am Rechner zügig und vollständig zu lösen (Lz3). Manche begannen schon mit der Bearbeitung der Aufgabe 2.

Reflexion der Stunde

Organisations– und Wiederholungsphase: Einmal mehr bemerkte ich, dass es den Schülern sichtlich Spaß machte, die Arbeitsmappe zu führen, und dass ich mir dadurch viel Arbeit und Probleme bereits vor der UE erspart habe. Dieser Umstand ist in künftigen UEs besonders zu berücksichtigen.

Übungsphase 1: Fast alle Schüler konnten die Aufgaben ohne nennenswerte Hilfestellung lösen. Sie verwendeten die Top–Down–Methode und hielten sich an sämtliche Konventionen. Aus diesem Grund lobte ich die Klasse, dass sie sich an die vereinbarten Regeln bezüglich des Programmentwurfs hielten. Daraufhin entgegnete mir eine Schülerin: „Sie haben es uns ja auch oft genug gesagt!“ (was mich innerlich etwas schmunzeln ließ). Ferner kann ich die These „Erfolg ist die beste Motivation“ nur bestätigen. Dies führte dazu, dass ich in der Übungsphase kaum Hilfestellungen geben und Schülergespräche führen musste.

Erarbeitungsphase 1: In Beispiel 4 auf `Arbeitsblatt 3` (Anhang D, Seite 78 ff.) schlich sich ein Fehler ein, den die Schüler bereits vor mir entdeckten. (Dieser Fehler ist bereits verbessert.) Ohne dieses käme es zu einem Laufzeitfehler.) Im Nachhinein könnte man sagen, dass dieser Fehler auch einen pädagogischen Zweck erfüllte, da wir dadurch schneller ins Gespräch kamen und er vor allem die Aufmerksamkeit der Schüler erhöhte.

Insgesamt war ich mit dem Verlauf dieser Stunde zufrieden. Ich bemerkte jedoch, dass einige Schüler die WHILE–Schleife im Rahmen der gemeinsamen Erarbeitungsphase 1 zwar verstanden, in Übungsphase 2 am Rechner jedoch Probleme damit hatten. Aus diesem Grund sei bei der weiteren Behandlung der WHILE–Schleife in der nächsten Stunde Vorsicht geboten.

3.2.4 Stunde 8 & 9 & 10

`Literatur:` [Bol02], [MS99].

Lernziele

Die Schüler

Lz1 können die Notwendigkeit (und nicht nur den Vorteil) der WHILE–Schleife begründen,

Lz2 wissen, was eine Endlosschleife ist,

Lz3 sind in der Lage, die WHILE–Schleife beim Programmentwurf sinnvoll einzusetzen,

LZ4 wissen, dass man WHILE-Schleifen auch ineinander schachteln kann und haben dies mindestens an einem Beispiel ausprobiert,

LZ5 lernen die IF–Anweisung als Beispiel einer Alternativanweisung kennen und können deren Notwendigkeit begründen,

LZ6 sind in der Lage, die IF–Anweisung beim Programmentwurf sinnvoll einzusetzen.

Unterrichtsverlauf

Wiederholungsphase & Sicherungsphase 1: Die WHILE–Schleife
(LZ1–LZ2, `Klassenunterricht,darbietend & fragend-gelenkt,OHP & Lehrerrechner`)

Zu Beginn der Stunde legte ich (die bereits in der letzten Stunde ausgefüllten) Seiten 1 und 2 von `Arbeitsblatt 3` (Anhang D, Seite 78 ff.) auf den OHP und wiederholte die wichtigsten Aspekte der WHILE–Schleife. Danach füllten wir gemeinsam die Seiten 3 und 4 aus. Dabei erkannten die Schüler nun auch die Notwendigkeit der WHILE–Schleife (LZ1), und sie wurden zum ersten Mal mit einer Endlosschleife konfrontiert (LZ2). Wir ergänzten sogleich das Glossar um diesen neuen Begriff. Im Anschluss besprachen wir ausführlich (die in der letzten Stunde bearbeitete) Aufgabe 1 auf `Übungsblatt 2` (Anhang D, Seite 82 ff.).

Übungsphase 1: Programmentwurf mit WHILE-Schleife
(LZ3, `Einzelarbeit,schülerzentriert & problemorientiert,Computer & Übungsblatt`)

Nun bearbeiteten die Schüler Aufgabe 2 auf `Übungsblatt 2` (LZ3). Dabei kam es bezüglich der WHILE–Schleife vor allem bei einigen Jungen erneut zu Problemen. Das lag daran, dass sie trotz der Wiederholungsphase und dem ausgeteilten `Musterbeispiel 2` (Anhang D, Seite 84) das Prinzip einer Wiederholungsanweisung immer noch nicht verstanden hatten. Daraufhin versuchten sie die Aufgabe ohne Verwendung der WHILE–Schleife zu lösen, womit sie natürlich scheiterten. Diese Erfolglosigkeit wirkte sichtlich demotivierend. Im Gegensatz zu den Jungen hatten die Mädchen das Prinzip der Wiederholungsanweisung besser verstanden, und sie *wollten* die WHILE–Schleife vor allem auch anwenden. Sie kamen sehr schnell zum Erfolg, was sich positiv auf deren Motivation auswirkte.
Um dieses Problem zu beseitigen, schenkte ich einigen Jungen nun mehr Aufmerksamkeit und gab ihnen mehr Hilfestellung, damit auch sie zum Erfolg kamen. Dem Rest der Klasse teilte ich währenddessen `Vertiefungsaufgabe 1` (Anhang D, Seite 85) aus, deren Inhalte die DO..WHILE–Schleife und die Unterschiede zur WHILE–Schleife waren.

Besprechungsphase 1: `Übungsblatt 2`, Aufgabe 2
(LZ3, `Klassenunterricht, fragend-gelenkt, Lehrerrechner & Tafel`)

Wir besprachen nun Aufgabe 2, indem wir zuerst gemeinsam alle nötigen Prozeduren formulierten. Danach führte ich eine gute Schülerlösung am Lehrerrechner vor.

Übungsphase 2: Programmentwurf mit WHILE-Schleife
(Lz3, `Einzelarbeit,schülerzentriert & problemorientiert,Computer & Übungsblatt`)

Danach bearbeiteten die Schüler Aufgabe 4 auf `Übungsblatt 2` (Anhang D, Seite 82 ff.). Dabei handelte es sich um die bis dahin schwerste Aufgabe. Deshalb gab ich den Schülern, die bereits bei der vorherigen Aufgabe Probleme hatten, sofort Hilfestellung, indem ich ihnen die benötigten Prozeduren nannte, sodass sie sich einzig und allein auf die Anwendung der WHILE–Schleife konzentrieren konnten. Aber auch damit hatten einige Schüler immer noch Probleme. Den Rest der Klasse ließ ich arbeiten und verwies auf Aufgabe 3 desselben Übungsblattes als Zusatzaufgabe.

Erarbeitungsphase 1: Geschachtelte WHILE–Schleifen
(Lz4, `Gruppenarbeit,schülerzentriert & problemorientiert,Papier & Übungsblatt`)

Die Schüler versammelten sich nun wieder in der Mitte des Raumes und bildeten Zweier– oder Dreiergruppen. Ich zeigte ihnen am Lehrerrechner die Aufgabenstellung der Aufgabe 1 auf `Übungsblatt 3` (Anhang D, Seite 86 ff.). Die Schüler sollten nun auf einem Stück Papier das entsprechende Programm entwickeln. Diese Gruppenübung sollte in der gesamten UE die einzige *vertiefende Übung* für alle Schüler sein. Nachdem nach ein paar Minuten die Frage „Kann man auch WHILE–Schleifen ineinander schachteln?“ gestellt (und von mir auch beantwortet) wurde, war zumindest garantiert, dass alle Gruppen den richtigen Weg einschlugen (Lz4). Allerdings gab es (aufgrund des Schwierigkeitsgrades) nur eine Mädchen–Gruppen, die die Aufgabe vollständig und korrekt lösen konnte. Außerdem hatten noch eine weitere Mädchen– und zwei Jungen–Gruppen akzeptable Lösungen.

Besprechungsphase 2: `Übungsblatt 3`, Aufgabe 1
(Lz4, `Klassenunterricht, entwickelnd, Lehrerrechner & Tafel`)

Nun besprachen wir (schülerzentriert) die gestellte Aufgabe, entwickelten die Lösung an der Tafel, und ich führte sie schließlich auf dem Lehrerrechner vor.
Aufgabe 2 auf `Übungsblatt 3` (Anhang D, Seite 86 ff.) ließ ich nicht bearbeiten, weil ich das Gefühl hatte, dass ich nun etwas Abstand von der WHILE–Schleife nehmen sollte, um einige Schüler nicht „zu verlieren“. Auch Arbeitsblatt 4 (Anhang D, Seite 88) übersprang ich, was mir jedoch schon vor der Stunde bewusst war, da ich dieses Blatt vor der UE lediglich als eventuellen „Lückenfüller“ verwenden wollte. Ich teilte es den Schülern dennoch aus, um die Chronologie und Vollständigkeit der Arbeitsmappe zu wahren.

Erarbeitungsphase 2: Die IF–Anweisung
(Lz5, `Klassenunterricht,fragend-erarbeitend & entwickelnd,Papier & Tafel & OHP`)

Ich legte die Aufgabenstellung des `Musterbeispiels 3` auf und wies die Schüler an, ein Programm (auf Papier) zu schreiben, welches das Problem löst. Nachdem die Schüler nach zwei bis drei Minuten nicht weiter kamen (verständlicherweise, da ihnen die IF–Anweisung nicht bekannt

war), erkundigte ich mich nach deren Problem, und die Schüler erkannten, dass weder die vier Hamster–Grundbefehle, noch Prozeduren oder die WHILE–Schleife zur Lösung des Problems ausreichend sind. Sie erkannten die Notwendigkeit einer Alternativanweisung (Lz5). Ohne die Lösung anzugeben, bearbeiteten wir zunächst die Seiten 1 und 2 von `Arbeitsblatt 5` (Anhang D, Seite 89 ff.). Erst danach führte ich die Lösung des Ausgangsproblems vor. (Grund: Ich wollte mit den Schülern auf `Arbeitsblatt 5` zunächst die IF–Anweisung separat üben, bevor ich diese in den Kontext eines vollständigen Programms aufnehme.)

Übungsphase 3: Übungsblatt 4, Aufgabe 2
(Lz6, `Einzelarbeit,schülerzentriert & problemorientiert,Computer & Übungsblatt`)
Die Schüler bearbeiteten nun die Aufgaben 1 und 2 auf `Übungsblatt 4` (Anhang D, Seite 93). Erstaunlicherweise lösten nun alle Schüler (die einen besser, die anderen schlechter) zumindest Aufgabe 1. Der Grund hierfür dürfte gewesen sein, dass keine WHILE–Schleifen benötigt wurden und die Schüler sich somit ganz auf die neu eingeführte IF–Anweisung konzentrieren konnten. Eine Besprechung der Aufgabe wurde damit hinfällig.

Reflexion der Stunde

In dieser Stunde war deutlich zu erkennen, dass die Mädchen in den Übungsphasen viel erfolgreicher als die Jungen waren. Sie waren offen für neue Konzepte und wollten diese auch ausprobieren und anwenden. Viele Jungen wollten jedoch nur schnell zur Lösung kommen und sahen die WHILE–Schleife eher als „Behinderung“ an, weil die theoretischen Grundlagen schwerer als bei den Prozeduren zu begreifen waren. Bei Einführung der IF–Anweisung holten diese Jungen jedoch wieder schlagartig auf, weil das Konzept der Alternative offenbar viel einfacher zu verstehen ist. Ich erkundigte mich bei den Schülern auch, warum die IF–Anweisung verständlicher ist. Die Antwort: „Das ist ja eigentlich ganz einfach — entweder man führt den einen Teil (IF–Zweig) oder den anderen (ELSE–Zweig) aus!“. Ich vermute, dass vor allem der Umstand, dass die IF–Anweisung nicht wiederholt ausgeführt wird, zum leichteren Verständnis führt („...entweder der eine Teil oder der andere, *und dann ist der IF–Block beendet!*“).
Es stellt sich nun die Frage, ob man die Alternativanweisung eventuell *vor* der Wiederholungsanweisung behandelt und ob man insbesondere geschachtelte Wiederholungsanweisungen komplett übergeht. Diese Frage wird jedoch in Kapitel 4 geklärt, weil sie die gesamte UE betrifft.

Insgesamt bin ich mit dem Verlauf dieser Stunde nur teilweise zufrieden. Die Behandlung der IF–Anweisung war problemlos. Die Verständnisschwierigkeiten mit der WHILE–Schleife und die damit verbundene Arbeitsunwilligkeit einiger Schüler stellten mich jedoch keineswegs zufrieden. Ich halte diese Stunde für den „Knackpunkt“ der gesamten UE. Das Problem der WHILE–Schleife muss unbedingt tiefergehend in Kapitel 4 reflektiert werden.

3.2.5 Stunde 11 & 12

`Literatur:` [KB06], [Bau92].

Lernziele

Die Schüler

Lz1 lernen Struktogramme als weitere Darstellungsmöglichkeit für Algorithmen kennen,

Lz2 sind in der Lage, ein Struktogramm in ein Hamster–Programm zu überführen,

Lz3 haben das Prozedurkonzept sowie die WHILE–Schleife und die IF–Anweisung wiederholt und sind in der Lage, diese sinnvoll zu kombinieren.

Unterrichtsverlauf

Organisations– und Wiederholungsphase: IF–Anweisung, Arbeitsmappe
(Lz5–Lz6 d. letzten Stunde, `Klassenunterricht,fragend-gelenkt & entwickelnd,OHP`)

Auch zu Beginn dieser Stunde ordneten wir zuerst fliegende Blätter in die Mappe ein und ergänzten das Inhaltsverzeichnis. Zur Wiederholung legte ich die Seiten 1 und 2 von `Arbeitsblatt 5` (Anhang D, Seite 89 ff.) auf den OHP und erklärte noch einmal anhand beider Beispiele die Funktionsweise der IF–Anweisung. Danach füllten wir gemeinsam Seite 3 desselben Arbeitsblattes aus. Anschließend stellte ich die Frage, welche bisherigen Konzepte zur Formulierung von Algorithmen unbedingt notwendig sind. Wie vermutet, nannten die Schüler „Prozeduren“, „WHILE–Schleife“ und „IF–Anweisung“. In einer sehr schülerzentrierten Diskussion fanden wir jedoch heraus, dass Wiederholungs– und Alternativanweisungen absolut notwendig und Prozeduren überaus sinnvoll, aber nicht zwingend notwendig sind. Ich ergänzte noch die *Befehlssequenz*, also die Aneinanderreihung von einzelnen Befehlen.

Erarbeitungsphase 1 & Übungsphase 1: Struktogramme
(Lz1–Lz2, `Einzelarbeit, schülerzentriert & problemorientiert, Computer & Übungsblatt`)

Die folgende Unterrichtsphase leitete ich mit der Bemerkung ein, dass es noch eine weitere Form der Algorithmendarstellung gibt. Ohne die nun eingeführten Struktogramme zu erwähnen, forderte ich die Schüler auf, das Struktogramm in Aufgabe 1 auf `Übungsblatt 5` (Anhang D, Seite 94) in ein Hamster–Prgoramm zu überführen (Lz1, Lz2). Fast alle Schüler verstanden die (von mir bewusst) sehr spärlich formulierte Aufgabenstellung. Die meisten konnten die Aufgabe lösen — einige hatten jedoch Probleme mit der Klammernsetzung bei den verschachtelten Ausdrücken. Schnellen Schülern teilte ich `Arbeitsblatt 6` (Anhang D, Seite 95 ff.) aus, und sie bearbeiteten `Vertiefungsaufgabe 2` (Anhang D, Seite 97).

Besprechungsphase 1 & Sicherungsphase 1: Übungsblatt 5, Aufgabe 1
(Lz1–Lz2, `Klassenunterricht, fragend gelenkt, Arbeitsblatt`)

In der Mitte des Raumes versammelt, fragte ich die Schüler nach dem Sinn und Zweck von Struktogrammen. Nach einigem Hin und Her fielen die richtigen Antworten: „Struktogramme sind sehr einfach in Programme überführbar“ und „Struktogramme beschreiben Algorithmen“. Ich ergänzte die Ausführungen noch durch: „Struktogramme helfen beim Problemlöse–Prozess, weil man gezwungen ist, zuerst über die Struktur des zu entwickelnden Algorithmus nachzudenken.“ Diese Ausführungen notierten wir schließlich auf Seite 2 des `Arbeitsblattes 6` (Anhang D, Seite 95 ff.).

Übungsphase 2: Vermischte Aufgaben
(Lz3, `Einzelarbeit,schülerzentriert & problemorientiert,Computer & Übungsblatt`)

In der abschließenden Übung der UE wies ich jedem Schüler individuell eine Aufgabe des `Übungsblattes 6` (Anhang D, Seite 98 ff.) zu (Lz3), ohne dieses Vorgehen jedoch genauer zu kommentieren. (Sinn und Zweck: Ich wies jedem Schüler eine Aufgabe gemäß seinem Leistungsstand zu.) Bemerkenswert war, dass (wieder einmal) einige Mädchen die mit Abstand schwersten Aufgaben Drei und Vier lösen konnten.

Abschliessende Demonstration: Das Weltraumspiel *Absolute Space*

Den Abschluss der gesamten UE bildete die Demonstration des 2D–Weltraumspiels *Absolute Space* [Spa]. Bei diesem handelt es sich um Open–Source–Software, geschrieben in der Programmiersprache Java. Das Prinzip des Spiels ist einfach. Man kann mit den Cursortasten ein Raumschiff steuern und mit der Space–Taste die Meteoriten, die auf das Schiff zusteuern, abschießen. Zunächst wählte ich einen Schüler aus, der das Spiel am Lehrerrechner demonstrieren sollte. Danach zeigte ich der Klasse den Quellcode dieses Spiels. Er enthielt viele bekannte Anweisungen und Schlüsselwörter, wie beispielsweise *WHILE*, *IF* oder *VOID*. Ich modifizierte nun einige Stellen, und die Schüler staunten nicht schlecht, als das Raumschiff plötzlich unverwundbar war. Bei der nächsten Modifikation hatte das Raumschiff unendlich viel Munition, und schließlich tat sich ein Schüler sehr schwer, als er die Cursor–links–Taste betätigte und das Raumschiff plötzlich nach rechts flog. Schließlich kamen auch auf Schülerseite Fragen der Art: „Herr Fruth, ändern Sie doch mal die Farbe des Raumschiffs!“ oder „Kann man auch die Anzahl der Meteoriten erhöhen?“. Da der Quelltext relativ gut überschaubar war, konnte ich den Schülern auch diese Wünsche erfüllen.

Ich demonstrierte den Schülern somit, wie die PC–Spiele funktionieren, die einige von ihnen zu Hause spielen. Ich erhoffte mir dadurch, dass die Schüler über die Funktionsweise von Software nachdenken und diese nicht als „gottgegeben“ hinnehmen. Außerdem wurde die praktische Relevanz der Programmierung verdeutlicht, was u.U. dazu führte, dass einige Schüler sich weiterhin mit der Programmierung beschäftigten.

Reflexion der Stunde

Der Ablauf der letzten Doppelstunde war völlig reibungslos, und ich war auch sehr zufrieden damit. Die Demonstration des Computerspiels war ein voller Erfolg. Nur selten habe ich Schüler derart neugierig und ruhig erlebt. Leider kam mir der Gedanke für diese Demonstration erst kurz vor der letzten Doppelstunde. Andernfalls hätte man diese Demonstration auch als Einstieg in die UE nehmen können — oder mehrmals zur Motivation der einzelnen Konzepte (Prozeduren, Wiederholung und Alternative). Man kann damit eventuell auch die *leidenschaftliche Neugier* wecken, über die in Kapitel 1 diskutiert wurde.

4 Reflexion und Auswertung

Gegenstand dieses Kapitels ist die kritische Reflexion der in Kapitel 3 beschriebenen Unterrichtseinheit (UE). Dabei soll weniger geklärt werden was gut bzw. schlecht lief — dieser Aspekt wurde bereits in den Stundendokumentationen und der anschließenden Reflexion angesprochen. Vielmehr soll erörtert werden, inwiefern ich mein Vorgehen bei der nächsten UE *Einstieg in die Programmierung* ändern oder beibehalten würde.

Zunächst wird die UE bezüglich der Schülerleistungen evaluiert. Danach möchte ich noch einmal die Problematik der WHILE–Schleife diskutieren, weil sie ein zentrales Problem darstellte. Anschließend beschreibe ich eventuelle künftige Änderungen meiner Vorgehensweise bezüglich dieser UE. Diese Überlegungen führen schließlich in Abschnitt 4.4 auf die Adaption und Modifikation der gesamten UE, bestehend aus den überarbeiteten Lerninhalten sowie einer Anpassung des Stoffverteilungsplans. Der Aspekt „Jungen und Mädchen in der ITG“ wird abschließend in Abschnitt 4.5 diskutiert, weil mich das Leistungsgefälle zwischen Mädchen und Jungen in dieser UE sehr überraschte.

4.1 Evaluation der Schülerleistungen

Wie bereits angedeutet, wurde aus dem in Abschnitt 2.1 genannten Grund keine unmittelbare Leistungsmessung am Ende der UE durchgeführt. Meine persönliche Bewertung, die den Erfolg der UE bezüglich der Schülerleistungen widerspiegelt, ist in Tabelle 4.1 angegeben. Aus Gründen des Datenschutzes wurden die Namen der Schüler verfälscht — lediglich das Geschlecht stimmt mit den jeweiligen Personen überein.

Interessant ist die Beobachtung, dass (bis auf eine Ausnahme) die Noten der PC–Arbeit durchgängig schlechter sind als die mündlichen Noten. Daran erkennt man meines Erachtens, dass die Schüler zwar eine zufriedenstellende bis gute Auffassungsgabe haben und auch denken, dass sie die Thematik verstanden hätten. Bei der Umsetzung der erlernten Konzepte am PC ergeben sich jedoch meistens mehr Probleme als zunächst erwartet.

4.2 Die WHILE–Problematik

Meiner Meinung nach gibt es zwei Gründe, die zu den Verständnisproblemen mit der WHILE–Schleife führten. Zum einen merkte man einigen Jungen an, dass sie sich nicht mit weiterer

Name	MÜ	PC	Gesamt
Thorsten	3	4	3/4
Tobias	3/4	4	4
Anna	1	1/2	1
Nils	1/2	2	2
Melanie	1	1	1
Claudia	1	1/2	1
Johannes	3/4	4	4
Tanja	2/3	3	3
Martin	3	3/4	3
Carsten	2	2/3	2
Jan	3	4	3/4
Lara	3	2	2/3
Andreas	2	2	2
Henrik	1/2	1/2	1/2

1	1/2	2	2/3	3	3/4	4	4/5	5	5/6	6
3	1	3	1	2	2	2	0	0	0	0

Schnitt = 2,4

Tabelle 4.1: Evaluation der Schülerleistungen. *Links:* Schülerleistungen, [MÜ]...Qualität der mündlichen Beiträge, [PC]...Arbeit am PC. *Rechts:* Notenspiegel und Durchschnitt.

Theorie beschäftigen wollten und die WHILE–Schleife somit quasi stillschweigend boykottierten. Sie bevorzugten *zu diesem Zeitpunkt* stattdessen die spielerische Beschäftigung mit dem Hamster–Simulator und versuchten, mit den Hamster–Grundbefehlen und dem Prozedurkonzept auszukommen. Vielleicht kam die WHILE–Schleife deshalb einfach nur zur falschen Zeit — nicht aufgrund falscher Planung, sondern weil die Jungen sich lieber noch etwas am Rechner „austoben“ wollten — denn bis dahin gab es nur wenige Probleme, und bei der folgenden IF–Anweisung gab es ebenfalls keine nennenswerten Schwierigkeiten.

Zum anderen denke ich, dass die Probleme einiger Schüler nicht bei dem allgemeinen Prinzip der Wiederholungsanweisung liegen, sondern ganz speziell bei der WHILE–Schleife zu suchen sind. In der Lern–/Programmierumgebung *Karol* (siehe Abschnitt 2.3.3) gibt es beispielsweise die Wiederholungsanweisung:

```
wiederhole 9 mal
   Schritt
*wiederhole
```

Ich bin mir sicher, dass diese *feste Wiederholung* kein Problem darstellen würde, sondern die Schwierigkeiten einzig und allein in der *bedingten Wiederholung* bestehen. Es gibt im Hamster–Modell ebenfalls die Möglichkeit einer festen Wiederholung:

```
for (int i=0; i<=8; i++)
   vor();
```

Die Verwendung der FOR–Schleife setzt jedoch die Kenntnis des Variablenkonzepts voraus, welches ich nicht behandelte. Außerdem würde die Formulierung des Schleifenkopfes zu erheblichen Problemen auf Schülerseite führen. Und letztlich ist es nicht sinnvoll, nur die FOR–Schleife

zu behandeln, weil es keine Notwendigkeit für diese gibt. Sie bietet lediglich Vorteile. Im Gegensatz dazu ist die bedingte Wiederholung bei der WHILE–Schleife absolut notwendig und bietet darüber hinaus Vorteile — und dieser Aspekt ist absolut zentral für die Programmierung. (Anmerkung: Ich habe bei meinen Ausführungen davon abgesehen, dass man auch mit der FOR–Schleife bedingte Wiederholungen formulieren kann, der Art „**for** (;vornFrei();)“. Diese Möglichkeit ist jedoch keinesfalls zu empfehlen und widerspricht dem Sinn dieser Schleife.)

Aus diesen Gründen werde ich künftig weiterhin (nur) die WHILE–Schleife behandeln. Ich werde dabei jedoch noch mehr betonen, dass diese aus zwei Teilen besteht. Im ersten Teil (Schleifenkopf) wird die Schleifenbedingung ausgewertet. Ist diese falsch, wird der folgende Block (Schleifenrumpf) komplett übersprungen. Andernfalls (und jetzt kommt der wichtige Aspekt) wird der Schleifenrumpf *genau ein Mal* ausgeführt (nicht mehrmals!). Anschließend wird die Schleifenbedingung erneut ausgewertet. Dieser Prozess wiederholt sich, *solange* die Schleifenbedingung zutrifft.

4.3 Künftige Vorgehensweise

Diese Vorgehensweise hat sich bewährt, und ich werde sie (vorerst) beibehalten:

- Ich werde weiterhin auf vielen Formalismen und Konventionen bezüglich des Programmentwurfs und der –entwicklung beharren. Das kostet zwar viel Mühe und Kraft, und die Schüler fühlen sich anfangs dadurch gestört. Nach zwei bis drei (Einzel–)Stunden haben die Schüler diese Vorgehensweisen jedoch akzeptiert, und es „stört sie auch nicht weiter“.

- Die detaillierte Ausarbeitung der Arbeitsmaterialien (auch schon im Hinblick auf die jeweils nächste und übernächste Doppelstunde) hat sich absolut bewährt. Es kostete zwar viel Vorbereitungszeit, aber einerseits profitierten die Schüler davon, weil sie ohne große Rückfragen effizient arbeiten konnten. Und andererseits profitierte ich davon, weil ich zum einen viele Lerninhalte behandeln und mich zum anderen im Unterricht etwas zurücknehmen konnte.

- Die Unterrichtsmethodik hat sich meines Erachtens vollends bewährt, und ich bin sehr zufrieden damit. Zusätzliche methodische Großformen, wie beispielsweise Planarbeit, Lernzirkel, Gruppenpuzzle oder ähnliches, wären im Hinblick auf die Thematik einfach nicht angemessen gewesen, und ich sah keine Möglichkeit, diese *sinnvoll* einzusetzen.

- Ich werde weiterhin die WHILE–Schleife *vor* der IF–Anweisung behandeln. Rein thematisch wäre es zwar durchaus sinnvoll diese Reihenfolge umzukehren, weil die Komplexität der IF–Anweisung meines Erachtens geringer als die der WHILE–Schleife ist. Es gibt jedoch — und das ist der wichtige Aspekt — im Rahmen und den Möglichkeiten des

Hamster–Modells kaum sinnvolle Übungsaufgaben, in denen die Alternative separat, also ohne die Wiederholung, zur Anwendung kommt. (Ich habe nach Lektüre von [Bol02] und reichlich eigenen Überlegungen zumindest nur wenige gefunden bzw. konstruieren können.)

- Übungsaufgaben bespreche ich weiterhin nur dann im Plenum, wenn sich hinsichtlich der Lösung oder des Lösungsprozesses Schwierigkeiten ergeben haben. In diesem Kontext ist eine Reduktion des Klassenunterrichts wirklich sinnvoll.

- Das Hamster–Modell hat sich für den Einstieg in die Programmierung absolut bewährt. Die Schüler beschäftigen sich gerne mit dem Simulator und können damit ihre Lösungen visualisieren und vor allem kontrollieren. Deshalb werde ich das Modell prinzipiell weiterhin im Unterricht verwenden, versuche jedoch, auch die Lern–/Programmierumgebung *Kara* im Unterricht zu testen.

Ich werde meine künftige Vorgehensweise in folgenden Punkten ändern:

- Im Rahmen der einstündig unterrichteten ITG ist es nicht sinnvoll, sich sechs Doppelstunden mit dem gleichen Thema zu beschäftigen — obwohl das Thema natürlich durchaus sinnvoll ist. Das entspräche einem Drittel eines Schuljahres. Andere, ebenfalls wichtige Themen, würden darunter leiden. Aus diesem Grund tendiere ich, unter Streichung einiger Lerninhalte (siehe dazu Abschnitt 4.4), zu einer Reduktion der UE auf vier bis maximal fünf Doppelstunden.

- Im Rahmen dieser angesprochenen Reduktion würde ich nicht mehr behandeln (obwohl sinnvoll und berechtigt):

  – geschachtelte WHILE–Schleifen (diese nur noch zur Vertiefung anbieten),
  – die Compilation in dieser Ausführlichkeit (künftig besser: „Compiler ist dieser Knopf im Simulator, und er überprüft, ob ihr euch an die Rechtschreibe– und Grammatikregeln der Hamster–Sprache gehalten habt.“ — Ende),
  – den Algorithmenbegriff in diesem Detaillierungsgrad (aber dennoch kurz behandeln und notieren).

  Weitere Themen kann man nicht streichen — höchstens noch Struktogramme, die ich jedoch weiterhin behandeln möchte. (Anmerkung: Bei der Planung der UE habe ich Struktogramme nur als *wünschenswerten Inhalt* deklariert. Nach der Durchführung der UE bin ich anderer Meinung. Die Schüler sollten zumindest in der Lage sein, Struktogramme in Hamster–Programme zu überführen, um deren Sinn und Zweck zu verstehen.) Aber um eine Einführung in den Hamster–Simulator, den Hamster–Grundbefehlen, dem Prozedurkonzept und der Top–Down–Methode sowie der WHILE–Schleife und der IF–Anweisung kommt man nicht herum.

- Das Computerspiel *Absolute Space* würde ich öfters in den Unterricht einbinden. Wenn beispielsweise die WHILE–Schleife behandelt wird, kann man im Quellcode dieses Spiels ganz gezielt einzelne WHILE–Schleifen ansteuern und modifizieren. Diese Vorgehensweise könnte die Schüler zusätzlich motivieren.

- Ich werde künftig versuchen, die Denkprozesse der Schüler genauer zu analysieren, um meine Vorgehensweise sinnvoll darauf abzustimmen. Dies könnte durch gezieltes Fragen während der Einzelarbeit oder dem Klassenunterricht geschehen. Fragemöglichkeiten: „Warum ist es schwierig?“, „Beschreibe mal was, du *verstanden* und was du *nicht verstanden* hast!“ oder „Wenn du das nicht nachvollziehen kannst, wie würdest du es denn machen?“.

4.4 Adaption und Modifikation der Unterrichtseinheit

Im Hinblick auf die im vorherigen Abschnitt gesammelten Erfahrungen dieser UE und einem in diesem Abschnitt vorgestellten Vergleich der Planung mit dem tatsächlichen Verlauf (bzgl. Lerninhalte und Stoffverteilung), versuche ich nun, die UE für eine erneute Durchführung sinnvoll zu modifizieren. Aus dem im vorherigen Abschnitt genannten Grund werde ich den Zeitraum auf fünf Doppelstunden reduzieren.

In Anhang A (Seite 43) sind noch einmal die in Abschnitt 2.3.4 aufgeführten Lernziele dieser UE aufgeführt. Zusätzlich wird angegeben, inwiefern sie meiner Meinung nach erreicht wurden. Diese Lernziele würde ich für eine künftige UE nicht modifizieren, zumal sie sowieso sehr allgemein gehalten sind. Dies erkennt man u.a. daran, dass die tatsächliche Realisierung einiger Lernziele nur sehr schwer verifizierbar ist.

Des Weiteren werden in den Tabellen B.1 und B.2 (Anhang B, Seite 46 ff.) die vor der UE geplanten Lerninhalte mit den tatsächlich vermittelteten Inhalten verglichen. Der Tabelle 4.2 kann man die geplanten Lerninhalte der modifizierten UE entnehmen.

Zuletzt wird der Stoffverteilungsplan auf fünf Doppelstunden angepasst. In den Tabellen C.1 und C.2 (Anhang C, Seite 49) wird zunächst der vor der UE geplante, mit dem tatsächlichen Stoffverteilungsplan verglichen. Tabelle 4.3 enthält einen Vorschlag für den Stoffverteilungsplan der modifizierten UE.

4.5 Jungen und Mädchen in der ITG

Mich überraschte nicht unbedingt, dass die Mädchen in dieser UE klar die Besseren waren. Vielmehr überraschte es mich jedoch, dass sie *ausgerechnet* und *nur* in dieser UE leistungsstärker waren und dass viele Jungen ihren vorherigen Leistungsstand nicht abrufen und halten konnten. Warum also hat sich ausgerechnet in dieser UE das Leistungsgefälle zwischen Jungen und Mädchen umgekehrt?

Lerninhalte	**unbedingt**	**wünschenswert**	**auf keinen Fall**
EINARBEITUNG IN DEN HAMSTER–SIMULATOR			
Konstruktion und Verwaltung von Hamster–Territorien	★		
Erstellung, Verwaltung und Ausführung von Programmen	★		
ALGORITHMEN			
Beispiele für Algorithmen	★		
Grundstrukturen von Algorithmen		★	
Definition des Algorithmen-Begriffs	★		
Algorithmen umgangssprachlich formulieren		★	
PROGRAMMIERUNG			
Befehlssequenzen mit Hamster-Befehlen	★		
Prozedurkonzept	★		
WHILE-Schleife als Beispiel für eine Wiederholungsanweisung	★		
FOR-Schleife			★
DO..WHILE-Schleife			★
geschachtelte WHILE-Schleifen		★	
IF-ELSE-Anweisung als Beispiel für eine Selektionsanweisung	★		
syntaktische und semantische Fehler			★
Variablenkonzept, Datentypen			★
Prozeduren mit formalen Parametern			★
Debugging			★
FORMALISMEN			
Einrücken des Quellcodes	★		
ausführliche Dokumentation			★
Prozeduren kommentieren	★		
Speichern von Territorien/Programmen (Speicherort & Namensgebung)	★		
Compilation bei der Programmerstellung alle 5-10 Zeilen	★		
PROBLEME LÖSEN			
Problemlöse–Prozess (Analyse, Entwurf, Implementierung, Test)	★		
Top–Down–Methode	★		
Struktogramme		★	

Tabelle 4.2: Modifizierte Lerninhalte für künftige UEs

Stunde	Inhalte
1/2	Einführung in den Hamster–Simulator, Entwurf einfacher Hamster–Programme, Algorithmen (kurz)
3/4	Prozedurkonzept, Top–Down–Methode
5/6	WHILE–Schleife
7/8	WHILE–Schleife, IF–Anweisung
9/10	Struktogramme, Wiederholung & vermischte Übungen

Tabelle 4.3: Modifizierter Stoffverteilungsplan für künftige UEs

Ich suchte in der Fachliteratur nach Erklärungsansätzen. In [SS04] wurde ich fündig. Dort heißt es, dass „sich Mädchen theoretischen Zugängen weniger verschließen als Jungen“. Des Weiteren „arbeiten sie [die Mädchen] systematischer und damit informatischer und bringen mehr Fähigkeiten zu Team– und Projektarbeit mit [...]“. Auch Baumann bemerkt in [Bau96], dass der Umgang von Mädchen mit der Informationstechnik „weniger experimentierend, sondern eher planend ist“.

Und genau diese Vorgehensweise „systematisches Arbeiten“ und „weniger experimentierend, sondern eher planend“ sind die Grundvoraussetzungen für den Problemlöse–Prozess und insbesondere einen erfolgreichen Programmentwurf.

Natürlich will ich meine Beobachtungen in diesem Zusammenhang nicht zu einem Gesetz formulieren. Wie bereits erwähnt, ist es lediglich ein Erklärungsansatz, warum die Mädchen erfolgreicher waren. In der nächsten UE kann sich dieses Ungleichgewicht zwischen Jungen und Mädchen selbstverständlich wieder ausgleichen oder gar umkehren.

Literaturverzeichnis

[BAC03] B.J. Allen-Conn, Kim Rose: *Powerful Ideas in the Classroom — Using Squeak to Enhance Math and Science Learning.* 1. Auflage. –, –, 2003.

[Bau92] Baumann, Rüdeger: *Informatik — für die Sekundarstufe II.* 1. Auflage. Klett–Verlag, Stuttgart, 1992.

[Bau96] Baumann, Rüdeger: *Didaktik der Informatik.* 2. Auflage. Ernst Klett-Verlag, Stuttgart, 1996.

[Bil94] *Bildungsplan für das Gymnasium.* Amtsblatt des Ministeriums für Kultus und Sport, Lehrplanheft 4, Baden–Württemberg, 1994.

[Bil04] *Bildungsplan 2004.* Ministerium für Kultus, Jugend und Sport des Landes Baden-Württemberg, 2004.

[BLK87] *Gesamtkonzept für die informationstechnische Bildung.* Bund–Länder–Komission für Bildungsplanung und Forschungsförderung, 1987.

[Bol02] Boles, Dietrich: *Programmieren spielend gelernt.* 2. Auflage. Teubner–Verlag, Stuttgart/Leipzig/Wiesbaden, 2002.

[Ham] *http://www.java-hamster-modell.de.* Java–Hamster–Modell.

[HPK] *http://www.bildung–staerkt–menschen.de.* Ministerium für Kultus, Jugend und Sport, Baden–Württemberg.

[Hub00] Hubwieser, Peter: *Didaktik der Informatik.* 1. Auflage. Springer–Verlag, Berlin, 2000.

[Huw04a] Huwendiek, Volker: *„Didaktische Modelle", in: Leitfaden Schulpraxis – Pädagogik und Psychologie für den Lehrberuf.* 4. Auflage. Cornelsen Scriptor, Berlin, 2004.

[Huw04b] Huwendiek, Volker: *„Unterrichtsmethoden", in: Leitfaden Schulpraxis – Pädagogik und Psychologie für den Lehrberuf.* 4. Auflage. Cornelsen Scriptor, Berlin, 2004.

[KB06] Klaus Buck, Dieter Haas: *enter2 — Informationstechnische Grundbildung.* 1. Auflage. Schroedel–Verlag, Braunschweig, 2006.

[Kli04] Klippert, Heinz: *Methoden–Training.* 14. Auflage. Beltz Praxis, Weinheim, 2004.

[Mat03] Mattes, Wolfgang: *Methoden für den Unterricht.* 1. Auflage. Schöningh–Verlag, Paderborn, 2003.

[MS99] Martin Schader, Lars Schmidt-Thieme: *Java — eine Einführung.* 2. Auflage. Springer–Verlag, Berlin, Heidelberg, New York, 1999.

[PB04] Peter Brichzen, Ulrich Freiberger: *Ikarus – Natur und Technik, Schwerpunkt Informatik 6/7.* 1. Auflage. Oldenbourg–Verlag, München, 2004.

[RR04] Raimond Reichert, Jürg Nievergelt: *Programmieren mit Kara — Ein spielerischer Zugang zur Informatik.* 2. Auflage. Springer–Verlag, Berlin, 2004.

[Spa] *http://www.xtreme-fun.de/files/onlinegames/weltraum.* Absolute Space.

[SS04] Sigrid Schubert, Andreas Schwill: *Didaktik der Informatik.* 1. Auflage. Spektrum Lehrbuch–Verlag, Heidelberg, 2004.

[Sun] *http://www.sun.com.* Sun Microsystems.

[UHe] *http://www.u-helmich.de/inf/java2h/seite01.html.* Ulrich Helmich.

A Erreichte Lernziele

(Zeichenerklärung (++, +, 0, –) auf nächster Seite.)

1. Fachliche Kompetenzen

   \+ Die Schüler sind mit den Grundzügen einer imperativen Programmiersprache vertraut und in der Lage, einfache (Hamster–)Programme zu entwickeln. Außerdem verstehen sie die Notwendigkeit der Übersetzung (*Compilation*) eines Programms in Maschinencode.

   \+ Die Schüler lernen die fundamentale Bedeutung des Algorithmen–Begriffs kennen und verstehen, dass jeder Algorithmus aus den *Kontrollstrukturen* Sequenz, Wiederholung und Alternative aufgebaut ist. Des Weiteren können sie Algorithmen in unterschiedlichen Darstellungsformen formulieren.

   ++ Die Schüler können sich in unbekannte Anwendungen (Hamster–Simulator) einarbeiten und diese bedienen.

2. Methodische Kompetenzen

   \+ Die Schüler verbessern ihre Fähigkeiten Probleme zu lösen und verstehen, dass die Lösung eines Problems (zumindest im informatischen Sinne) aus den Schritten Analyse, Entwurf, Implementierung und Test besteht.

   ++ Die Schüler sind mit der Top–Down–Methode vertraut und wenden diese im Algorithmenentwurf an.

   (+) Die Schüler lernen, ihre Gedanken zu strukturieren, sodass sie durch systematisches Denken reproduzierbare Ergebnisse am Rechner erhalten.

3. Personale Kompetenzen

   ++ Durch die strenge Einhaltung von Formalismen (der Programmiersprache) und vereinbarten Regeln lernen die Schüler sorgfältiges und konzentriertes Arbeiten.

   (0) Der Problemlöse–Prozess erhöht die Kreativität und Ausdauer der Schüler.

4. Soziale Kompetenzen

   (0) Durch die Arbeit in Kleingruppen erhöht sich die Teamfähigkeit der Schüler, und sie tolerieren Ideen, Vorschläge und Lösungen von Mitschülern.

Abkürzungen: ++ ... Lernziel voll erreicht
+ ... Lernziel zufriedenstellend erreicht
0 ... Lernziel mit einzelnen Mängeln erreicht
– ... Lernziel verfehlt
( ) ... aufgrund des eigentlichen Lernziels oder der allgemeinen Formulierung nicht oder nur schwer beurteilbar

B Geplante und vermittelte Lerninhalte

Auf den nächsten beiden Seiten werden die in Abschnitt 2.3.4 erarbeiteten geplanten Lerninhalte mit den tatsächlich vermittelten Inhalten verglichen.

Lerninhalte	**unbedingt**	**wünschenswert**	**auf keinen Fall**
EINARBEITUNG IN DEN HAMSTER–SIMULATOR			
Konstruktion und Verwaltung von Hamster–Territorien	★		
Erstellung, Verwaltung, Compilation und Ausführung von Programmen	★		
ALGORITHMEN			
Beispiele für Algorithmen	★		
Grundstrukturen von Algorithmen	★		
Definition des Algorithmen-Begriffs	★		
Algorithmen umgangssprachlich formulieren	★		
PROGRAMMIERUNG			
Befehlssequenzen mit Hamster-Befehlen	★		
Prozedurkonzept	★		
WHILE-Schleife als Beispiel für eine Wiederholungsanweisung	★		
FOR-Schleife			★
DO..WHILE-Schleife		★	
geschachtelte WHILE-Schleifen		★	
IF-ELSE-Anweisung als Beispiel für eine Alternativanweisung	★		
syntaktische und semantische Fehler		★	
Variablenkonzept, Datentypen			★
Prozeduren mit formalen Parametern			★
Debugging			★
FORMALISMEN			
Einrücken des Quellcodes	★		
ausführliche Dokumentation			★
Prozeduren kommentieren	★		
Speichern von Territorien/Programmen (Speicherort & Namensgebung)	★		
Compilation bei der Programmerstellung alle 5-10 Zeilen	★		
PROBLEME LÖSEN			
Problemlöse–Prozess (Analyse, Entwurf, Implementierung, Test)	★		
Top–Down–Methode	★		
Struktogramme		★	

Tabelle B.1: Geplante Lerninhalte der UE

Lerninhalte	**alle Schüler**	**einige Schüler**	**kein Schüler**
EINARBEITUNG IN DEN HAMSTER–SIMULATOR			
Konstruktion und Verwaltung von Hamster–Territorien	★		
Erstellung, Verwaltung, Compilation und Ausführung von Programmen	★		
ALGORITHMEN			
Beispiele für Algorithmen	★		
Grundstrukturen von Algorithmen	★		
Definition des Algorithmen-Begriffs	★		
Algorithmen umgangssprachlich formulieren	★		
PROGRAMMIERUNG			
Befehlssequenzen mit Hamster-Befehlen	★		
Prozedurkonzept	★		
WHILE-Schleife als Beispiel für eine Wiederholungsanweisung	★		
FOR-Schleife			★
DO..WHILE-Schleife		★	
geschachtelte WHILE-Schleifen	☆		
IF-ELSE-Anweisung als Beispiel für eine Selektionsanweisung	★		
syntaktische und semantische Fehler			☆
Variablenkonzept, Datentypen			★
Prozeduren mit formalen Parametern			★
Debugging			★
FORMALISMEN			
Einrücken des Quellcodes	★		
ausführliche Dokumentation			★
Prozeduren kommentieren	★		
Speichern von Territorien/Programmen (Speicherort & Namensgebung)	★		
Compilation bei der Programmerstellung alle 5-10 Zeilen	★		
PROBLEME LÖSEN			
Problemlöse–Prozess (Analyse, Entwurf, Implementierung, Test)	★		
Top–Down–Methode	★		
Struktogramme	☆		

Tabelle B.2: Vermittelte Lerninhalte der UE. Die ☆–Einträge deuten auf eine Änderung hinsichtlich der ursprünglich geplanten Lerninhalte hin.

C Geplanter und tatsächlicher Stoffverteilungsplan

Stunde	Datum	Inhalte
1/2	26.04.06	Algorithmen, Compilation, Einführung in den Hamster–Simulator
3/4	03.05.06	Entwurf einfacher Hamster–Programme, Prozedurkonzept, Top–Down–Methode
5/6	10.05.06	Prozedurkonzept, Top–Down–Methode, WHILE–Schleife
7/8	N.N.	WHILE–Schleife
9/10	17.05.06	IF–Anweisung
11/12	24.05.06	Wiederholung & vermischte Übungen

Tabelle C.1: (Geplanter) Stoffverteilungsplan

Stunde	Datum	Inhalte
1/2	26.04.06	Algorithmen, Compilation, Einführung in den Hamster–Simulator
3/4	03.05.06	Entwurf einfacher Hamster–Programme, Prozedurkonzept, Top–Down–Methode
5/6/7	10.05.06	Prozedurkonzept, Top–Down–Methode, WHILE–Schleife
8/9/10	17.05.06	WHILE-Schleife, geschachtelte WHILE–Schleifen, IF–Anweisung
11/12	24.05.06	IF–Anweisung, Struktogramme, Wiederholung & vermischte Übungen

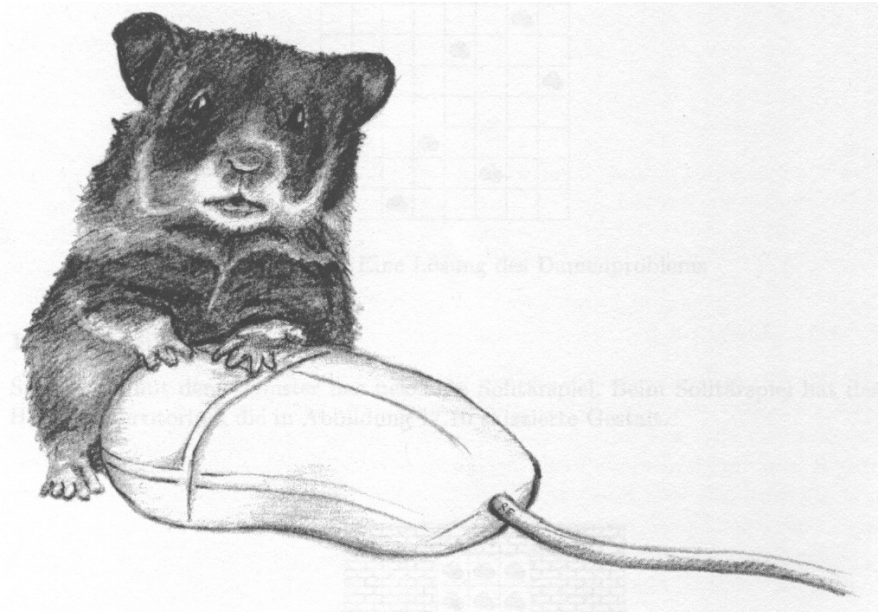
Tabelle C.2: (Tatsächlicher) Stoffverteilungsplan

D Arbeitsmaterialien der Unterrichtseinheit

Es folgt nun die Angabe sämtlicher Arbeitsmaterialien der Arbeitsmappe. Diese wurden alle von mir erstellt. Lediglich einige Aufgaben habe ich aus Büchern bzw. Internetquellen zitiert bzw. modifiziert. Sie sind in den Arbeitsmaterialien daher mit einer Quellenangabe versehen. Die restlichen Aufgaben habe ich selbst konzipiert.

Sämtliche Unterrichtsmaterialien befinden sich außerdem auf der Begleit–CD. (Dort wurde jedoch auf die Quellenangabe der übernommenen Aufgaben verzichtet, um sie direkt für den Unterricht nutzbar zu machen.) Die Materialien sind sowohl im *Portable–Document–Format* (PDF) als auch im *Open–Document–Textformat* (ODT) verfügbar und können daher geändert und individuell angepasst werden.

Einstieg in die ***Programmierung***



mit dem Java-Hamster-Modell

Rahmendaten

Schüler: _____

Klasse: _____

Schule: _____

Fach: _____

Zeitraum: _____

Lehrer: _____

Inhaltsverzeichnis

Arbeitsblatt 1

Probleme lösen mit dem Computer

Von der symbolischen Interaktion zu textuellen Befehlssequenzen

Bisher: Probleme lösen mit *Anwendersoftware* (MS Word, Paintbrush, HTML-Editor, ...) – meist durch symbolische Interaktion mit dem Rechner (z.B. Menüsteuerungen, Buttons, etc.).

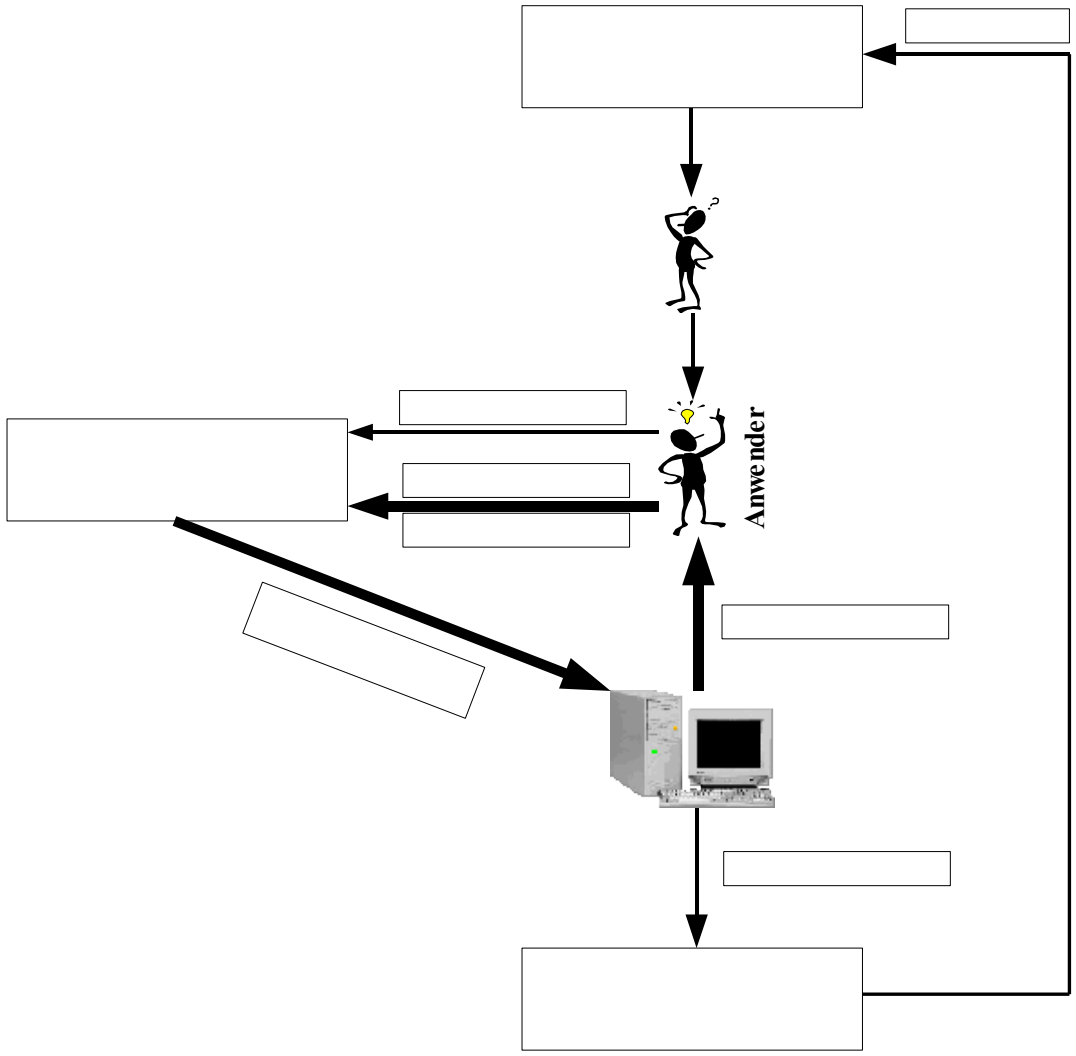


Abb. 1: Probleme lösen mit Anwendersoftware

Bemerkung:
Bei der Problemlösung mit Anwendersoftware (Abb. 1) benutzt der *Anwender* eine Software, mit der er (meist symbolische) Befehle an den Computer gibt. Der Computer gibt ständig Rückmeldungen an den Anwender, worauf dieser mit dem nächsten Befehl fortfährt. Dieser Prozess wird solange wiederholt, bis die gewünschte Lösung erarbeitet wurde.

Beispiel (Erzeugung eines Textdokuments in MS Word):

Problembeschreibung:
Es soll ein Textdokument mit MS Word erzeugt werden. Der Inhalt des Dokuments soll „Probleme lösen mit dem **Computer**“ lauten. Das Dokument soll unter C:\Texte gespeichert werden.

Lösungsverfahren (Vorbedingung: Benutzer ist am Rechner angemeldet):
- MS Word starten
- neues Dokument erstellen
- Text „Probleme lösen mit dem“ schreiben
- „Fett-Modus“ aktivieren
- Text „Computer“ schreiben
- „Fett-Modus“ deaktivieren
- …

Ausführung (des Lösungsverfahrens):

Schritt	Andwenderaktion	Computeraktion
MS Word starten	Doppelklick mit linker Maustaste (lM) auf Desktopsymbol „Word“	startet MS Word
neues Dokument erstellen	Klick mit lM auf Button „Neu“	erzeugt neues Dokument
Text „Probleme lösen mit dem“ schreiben	eine Tastenfolge drücken, sodass der gewünschte Text entsteht	zeigt nach jedem Tastendruck ein neues Zeichen auf dem Bildschirm an
„Fett-Modus“ aktivieren	Klick mit lM auf Button „Fett“	aktiviert „Fett-Modus“ - jedoch keine sichtbare Aktion auf Bildschirm
Text „Computer“ schreiben	Tastenfolge drücken, sodass Text „Computer“ entsteht	zeigt nach jedem Tastendruck ein neues Zeichen auf dem Bildschirm an
„Fett-Modus“ deaktivieren	Klick mit lM auf Button „Fett“	aktiviert „Fett-Modus“ - jedoch keine sichtbare Aktion auf Bildschirm
…	**…**	**…**

Merke:
- Bei dem Lösungsverfahren handelt es sich um eine kleinschrittige Handlungsvorschrift, die der Anwender ausführt, um die gewünschte Lösung zu erhalten.
- Eine Handlungsvorschrift zur Lösung eines Problems oder einer bestimmten Art von Problemen bezeichnet man als *Algorithmus*.

Neu: Probleme lösen mit (selbst erstellten) Programmen

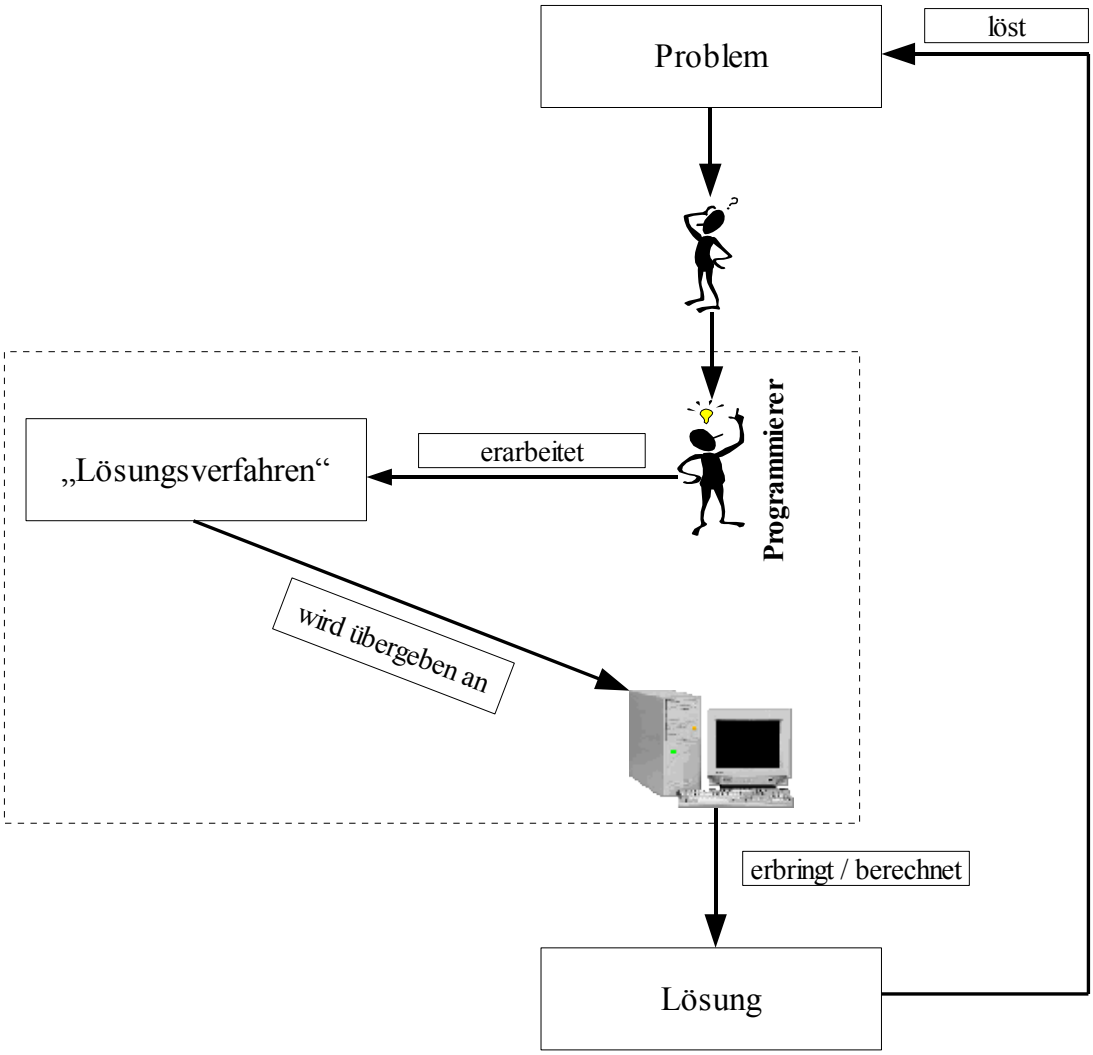


Abb. 2: Probleme lösen mit Programmen

Bemerkung:
Im Gegensatz zur Problemlösung mit Anwendersoftware entwickelt der *Programmierer* hier ein „Lösungsverfahren“, welches aus einer Aneinanderreihung textueller Befehle (textuelle Befehlssequenz oder *Programm*) besteht. Die Ausführung des Programms löst schließlich das gestellte Problem. (Details auf nächster Seite.)

Detailliert (für die Spezialisten und jene, die es ganz genau wissen wollen):

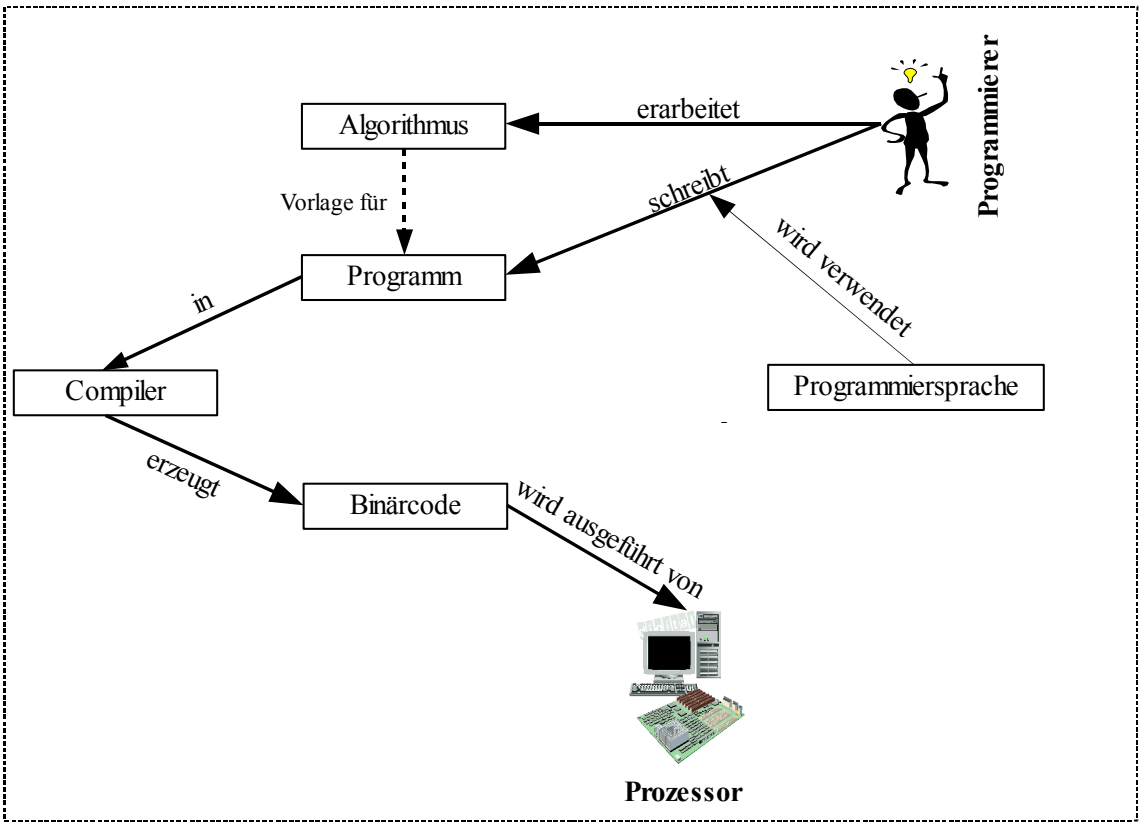


Abb. 3: Prozess der Programmentwicklung und -ausführung

Bemerkung:
Der *Programmierer* entwirft zunächst einen *Algorithmus* zur Lösung seines Problems. Danach „übersetzt“ er diesen Algorithmus mit einer *Programmiersprache* in ein *Programm*. Diese Übersetzung ist notwendig, weil der Computer den meist umgangssprachlich formulierten Algorithmus nicht versteht.
Die Programmiersprache ist eine Art Kompromiss zwischen Programmierer und Computer. Sie ist relativ computernah – kann jedoch vom Menschen noch recht gut verstanden werden.
Weil das Programm aber noch aus Buchstaben und Zahlen besteht, wird es vom *Compiler* (dem Übersetzer, *engl.* to compile = übersetzen) in einen *Binärcode* übersetzt.
Der Binärcode besteht nur noch aus „Nullen“ und „Einsen“ und kann somit vom Prozessor verarbeitet werden. (Der Binärcode ist also die Übersetzung des Programms in „Nullen“ und „Einsen“.) Außerdem prüft der Compiler das Programm auf „Rechtschreibe– und Grammatikfehler“ (bezüglich der Programmiersprache). Somit können „Tipp– und Leichtsinnsfehler“ bereits vor Programmstart beseitigt werden.

Tutorial

„Der Hamster–Simulator“

Teil I: Installation & Konstruktion von Hamster–Territorien

Installation des Hamster–Simulators

Kopiere den Ordner „JavaHamster“ vom Tauschordner (Laufwerk T) in dein Homeverzeichnis (Laufwerk H) bzw. in deine „Eigene Dateien“.

Starten des Hamster–Simulators

Der Hamster–Simulator benötigt eine sogenannte *Java Runtime Environment* (kurz *JRE*), um korrekt arbeiten zu können. Diese kannst du installieren, indem du auf der Taskleiste das Fenster „Novell delivered applications“ maximierst und danach auf das Symbol mit dem roten Herz („Java“) doppelklickst.

Öffne anschließend den Ordner H:\JavaHamster. Darin findest du die Dateien „tools.jar“ und „hamstersimulator.jar“. Starte den Simulator, indem du die Datei „hamstersimulator.jar“ ausführst. Daraufhin werden zwei Fenster geöffnet: der *Editor* und der *Simulator*.

Der Simulator

Um unserem Hamster Befehle zu erteilen, müssen wir zunächst ein Territorium konstruieren und den Hamster darin platzieren. Die Verwaltung von Territorien (also das erzeugen, speichern und öffnen) geschieht im Simulator. Die (vorerst) wichtigsten Funktionen des Simulators werden in Abbildung 1 genannt.

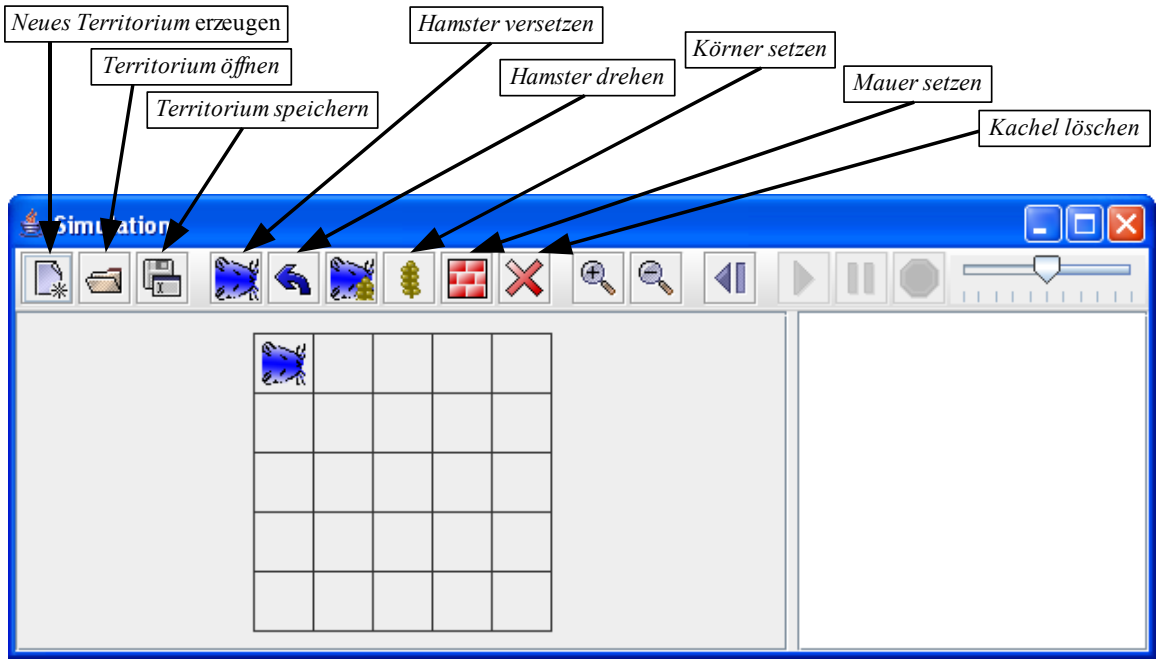


Abb. 1: Der Simulator

Erzeugen eines Hamster–Territoriums
Wir wollen nun das in Abbildung 2 dargestellte Territorium erzeugen.

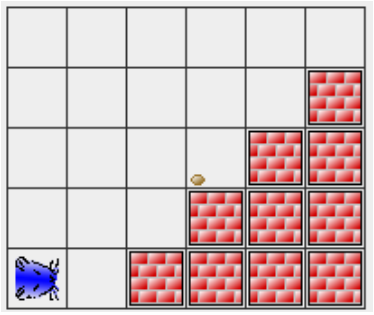


Abb. 2: Territorium *Bergsteiger*

Klicke dazu auf den Button „Neues Territorium“. Es öffnet sich ein Fenster mit der Bezeichnung „Größe“. Gib darin die entsprechende Spalten- und Zeilenzahl an, also:

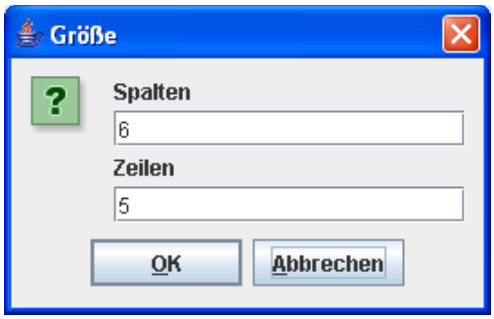


Abb. 3: Größe des Territoriums festlegen

Bilde nun das in Abbildung 2 dargestellte Territorium nach. Ziehe eventuell Abbildung 1 zu Hilfe heran. (Teste auch die Funktionsweise des Buttons „Kachel löschen“.)

Speichern eines Hamster-Territoriums
Wir wollen nun das eben erzeugte Territorium speichern. Klicke dazu auf den Button „Territorium speichern“. Es öffnet sich das in Abbildung 4 dargestellte Fenster.

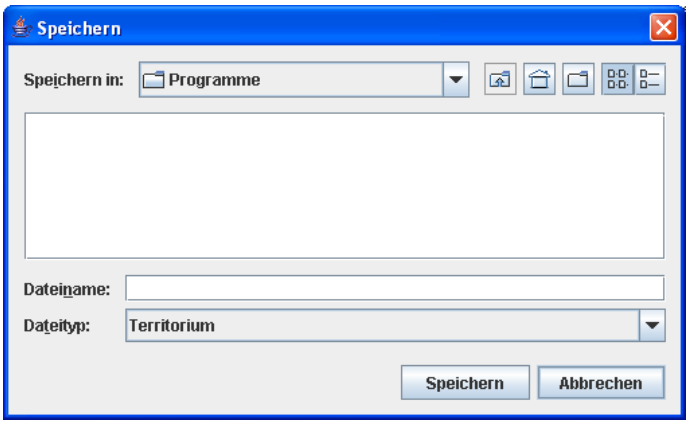


Abb. 4: Territorium speichern

Merke:
Jedes Territorium speichern wir in einem separaten Ordner, der denselben Namen wie das Territorium bekommt.

Gehe deshalb wie folgt vor:
Klicke auf den Button „Neuen Ordner erstellen“ (siehe Abbildung 5).

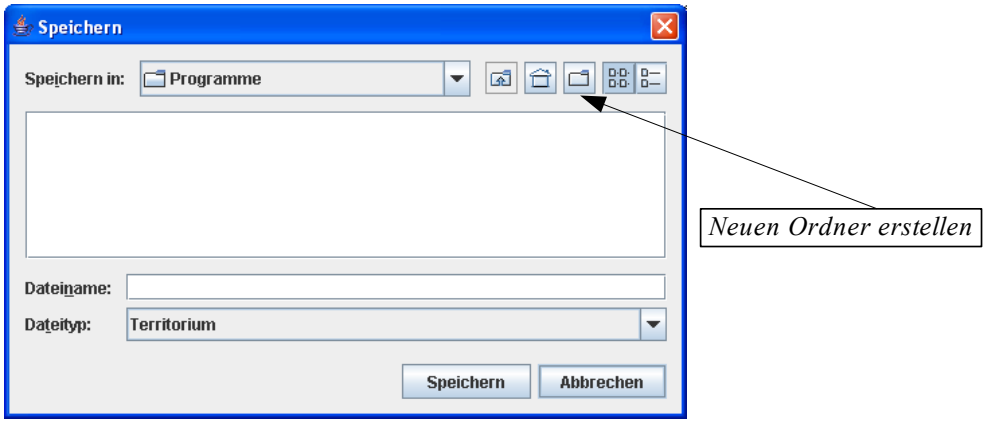


Abb. 5: Neuen Ordner erstellen

Ein neuer Ordner „Neuer Ordner“ wurde erzeugt (siehe Abbildung 6).

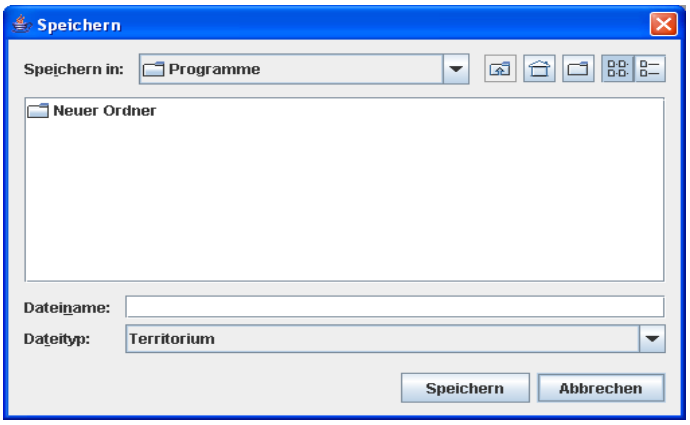


Abb. 6: Neuer Ordner wurde erstellt

Benenne den erstellten Ordner um in „Bergsteiger“, indem du auf den Schriftzug „Neuer Ordner“ (einfach) klickst (siehe Abbildung 7).

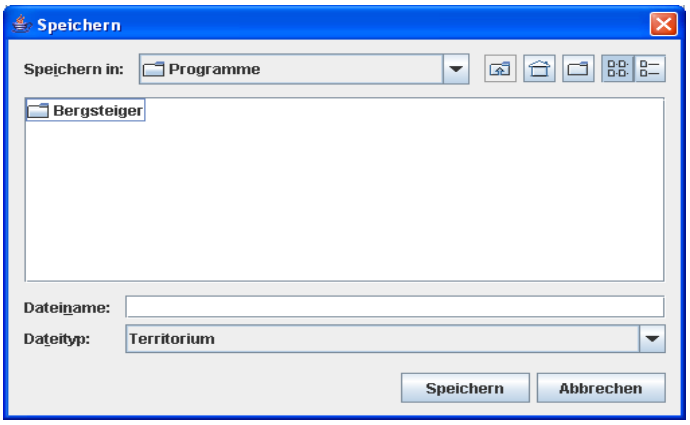


Abb. 7: Ordner *Bergsteiger* wurde erstellt

Öffne diesen Ordner nun mit einem Doppelklick und gib unter dem Punkt „Dateiname“ ebenfalls den Namen „Bergsteiger“ an.

Beachte:
Gib bei der Eingabe eines Dateinamens niemals eine Dateierweiterung an, wie beispielsweise „.ter“ oder „.ham“.

Speichere nun das Territorium durch einen Klick auf den Button „Speichern“.
Schließe den Hamster–Simulator jetzt mit einem Klick auf das rote Kreuz (rechts oben).

Öffnen eines Hamster–Territoriums
Starte den Hamster–Simulator erneut. Klicke auf den Button „Territorium öffnen“, navigiere in den Ordner „Bergsteiger“ und wähle die Datei „Bergsteiger“ aus. Klicke anschließend auf den Button „Öffnen“. Das eben erzeugte Territorium wird geladen und angezeigt.

Übung
Erzeuge nun folgendes Territorium

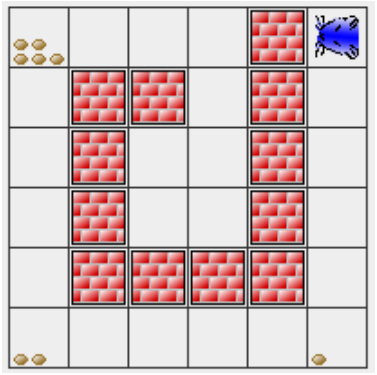


Abb. 8: Territorium *Winterschlaf* [aus [UHe], modifiziert]

und speichere es unter dem Namen „Winterschlaf“. Gehe dabei GENAUSO vor, wie im Tutorial beschrieben.

Tutorial
„Der Hamster–Simulator“
Teil II: Unser erstes Hamster–Programm

In Teil 1 dieses Tutorials haben wir den Simulator kennen gelernt, mit dem wir Hamster–Territorien erzeugen können. Bisher haben wir unserem Hamster jedoch noch keine Befehle in Form eines Hamster–Programms erteilt. Wie man den Hamster nun durch die Hamster–Landschaft dirigiert und wie man ihn Körner einsammeln und wieder fallen lassen kann, lernen wir nun in Teil 2 dieses Tutorials.
(Vereinbarung: Ab jetzt nennen wir (bis auf wenige Ausnahmen) „Hamster–Programme“ nur noch *Programme* und „Hamster–Territorien“ nur noch *Territorien*.)

Der Editor
Wir lernen jetzt zunächst den Editor kennen, in dem wir später unsere Programme schreiben werden. Abbildung 9 verschafft dir einen Überblick über die wichtigsten Funktionen des Editors.

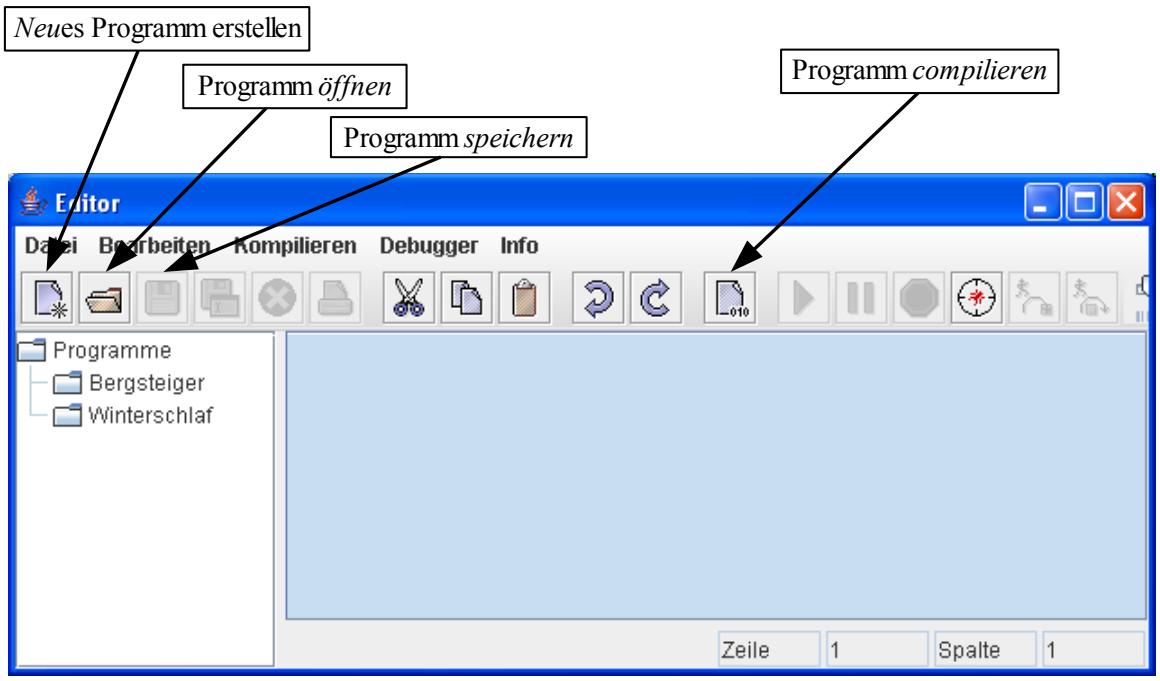
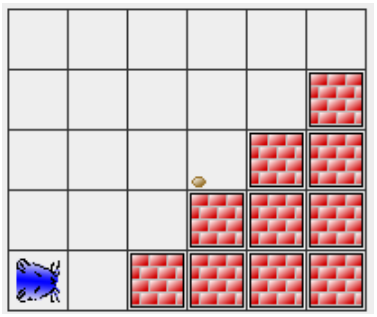


Abb. 9: Der Editor

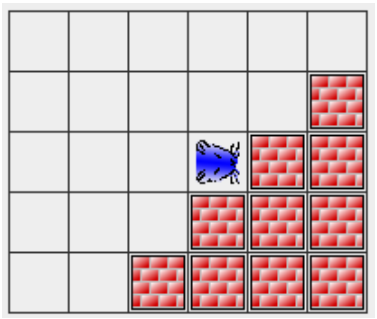
Wir schreiben unser erstes Programm
Um ein Programm zu schreiben, müssen wir zunächst ein Territorium erzeugen bzw. ein bereits gespeichertes Territorium laden. Öffne deshalb das Territorium „Bergsteiger“ im Simulator (Achtung: nicht im Editor sondern im Simulator!).

Wie du siehst, steckt unser Hamster in einer schwierigen Situation. Er will nämlich bis zur Hälfte den Berg vor sich erklimmen und das Korn in der Mitte des Berghangs aufnehmen. Unsere Aufgabe ist es also, dieses Problem mit einem Hamster–Programm zu lösen.

Wir können die Problemstellung auch etwas formaler stellen:
Schreibe ein Programm, welches das Territorium



in das Territorium

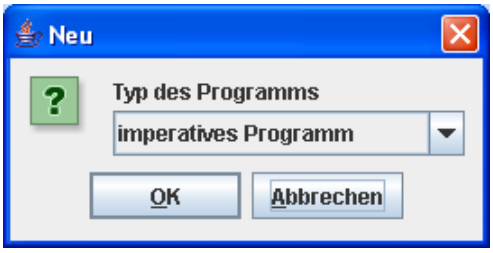


überführt.
(Obwohl formale Problemstellungen äußerst präzise und elegant sind, ist eine kleine Hamster–Geschichte ab und zu auch ganz schön, nicht wahr?)

Du kannst hier also bereits erkennen, dass Hamster–Programme, einen *Zustandswechsel* eines Territoriums herbeiführen:

> *Merke:*
> *Ein* initialer Anfangszustand *eines Territoriums wird durch ein Hamster–Programm in einen gewünschten* Endzustand *überführt.*

Klicke nun auf den Button „Neu“, um ein neues Programm zu erstellen. Es erscheint folgendes Fenster:



Der Begriff *imperatives Programm* soll uns an dieser Stelle nicht weiter interessieren. Klicke deshalb einfach auf „OK“.
Im *Programmfenster* (dem rechten Teilfenster des Editors) erscheinen nun folgende *Programmzeilen*:

```
void main ( ) {

}
```

Diese Zeilen ändern wir (bei jedem neuen Programm) zunächst ab zu:

```
void main ( )
{

}
```

Das ist das Grundgerüst unseres Programms: das *Hauptprogramm* (engl. *main* = Haupt-). Die geschweiften Klammern inklusive aller Befehle, die wir zwischen diese Klammern schreiben, bilden einen sogenannten *Block*. Wir achten stets darauf, dass:

> Die (geschweiften) Klammern eines Blockes stehen immer in derselben Spalte – also auf einer vertikalen Linie.

Für uns ist vorerst nur wichtig zu wissen, dass wir unser Programm zwischen die geschweiften Klammern „{ }“ schreiben. Die Bedeutung des *Schlüsselwortes* „void“, sowie der Name „main“, als auch die Bedeutung der runden Klammern, lassen wir zunächst außen vor.
Ergänze nun das Programm um folgende Programmzeilen:

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

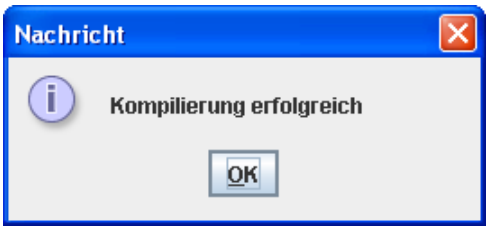
Wir vereinbaren:

Die Programmzeilen eines Blocks rücken wir immer um 4 Leerzeichen bzw. einen Tabulator ein.

Klicke nun auf den Button „Speichern“, und navigiere im sich öffnenden Fenster in den Ordner „Bergsteiger“. Dort gibst du unter dem Punkt „Dateiname“ den Namen „Bergsteiger“ ein und klickst auf den Button „Speichern“.

Merke: Alle Programme zu einem bestimmten Territorium speichern wir in dem gleichen Ordner, in welchem sich auch das Territorium befindet. Werden mehrere Programme zu demselben Territorium geschrieben, so müssen diese eindeutig benannt werden.

Bevor wir das Programm nun ausführen, müssen wir es zunächst (in die Maschinensprache) übersetzen (*compilieren)* lassen. Klicke dazu auf den Button „Compilieren“ (siehe Abbildung 9). Nach einigen Sekunden sollte folgende Nachricht erscheinen:



Falls diese Nachricht erscheint, klicke auf den Button „OK“.

Falls diese Nachricht nicht erscheint, dann erlebst du jetzt zum ersten Mal, dass Computer äußerst kleinlich sind (noch viel kleinlicher als Lehrer!). Dir sind nämlich höchstwahrscheinlich bei der Eingabe des Programms ein oder mehrere Tippfehler unterlaufen. Hast du beispielsweise nur ein Semikolon vergessen, oder einen Buchstaben fälschlicherweise groß geschrieben, dann scheitert der gesamte Übersetzungprozess.
Kontrolliere also noch einmal dein Programm auf Tippfehler, und versuche erneut zu compilieren. Wie man bei etwas größeren Programmen Tippfehler sehr einfach finden kann, lernen wir später gemeinsam.

Unser Programm wurde nun also übersetzt und in den Speicher unseres Computers geladen. Wir können es nun ausführen. Wechsle dazu in den Simulator, und klicke auf den Button „Ausführen“ (siehe Abbildung 10). Außerdem kannst du an dem Schieberegler (ganz rechts im Simulator) die Ausführungsgeschwindigkeit des Programms regeln.

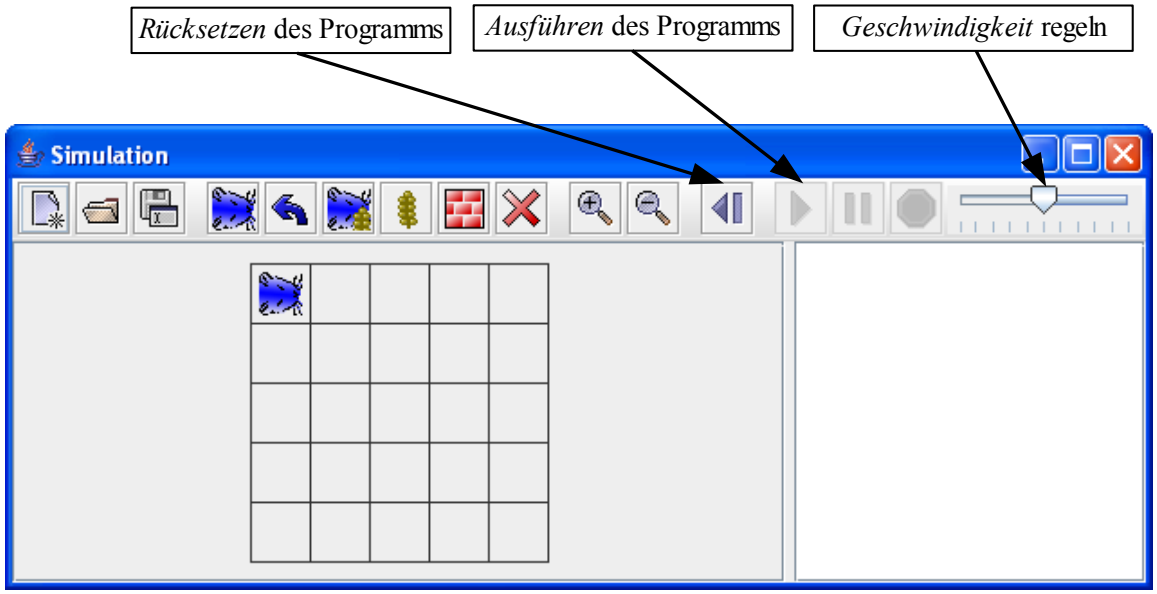


Abb. 10: Programmstart aus dem Simulator heraus

Des Weiteren werden im rechten Teilfenster des Simulators die abgearbeiteten Programmzeilen angezeigt.

Wichtig für den Moment ist nur noch:

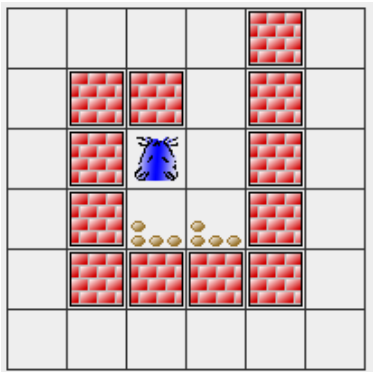
> Immer wenn wir unser Programm erneut ausführen, müssen wir vorher den Simulator mit dem Button „Rücksetzen“ in den initialen Anfangszustand zurücksetzen (siehe Abbildung 10).

Übung

Öffne im Simulator das Territorium „Winterschlaf“.

Der Sommer neigt sich dem Ende, und unser Hamster hat völlig vergessen, für seinen Winterschlaf vorzusorgen. Deshalb rafft er sich auf und will ein paar Körner in seinen Bau bringen. Kannst du ihm dabei helfen?

Schreibe ein Programm, welches das Territorium „Winterschlaf“ in folgendes Territorium überführt:



Gehe dabei GENAUSO vor, wie im Tutorial beschrieben.
Achte insbesondere darauf, die Programmzeilen innerhalb eines Blockes einzurücken, sowie auf den korrekten Speicherort und die korrekte Namensgebung des Programms.

Beachte ferner folgenden verbindlichen Tipp:

> Compiliere ca. alle zehn Programmzeilen, um Tippfehler frühzeitig zu erkennen.

Merkblatt 1

Hamster-Befehle

Die vier Grundbefehle im Hamster-Modell

vor ( );	...	Hamster geht einen Schritt nach vorne
linksUm ( );	...	Hamster dreht sich um 90° nach links
nimm ( );	...	Hamster nimmt *genau ein* Korn auf
gib ( );	...	Hamster legt *genau ein* Korn ab

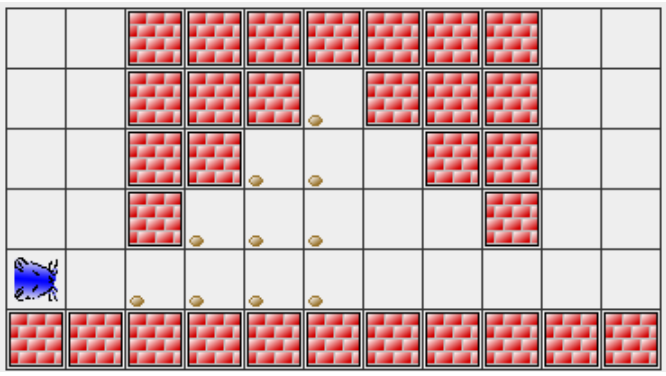
Bemerkung:
Die Semikola sind keine Bestandteile der jeweiligen Befehle, sondern dienen als Trennzeichen zwischen zwei Befehlen. Trotzdem sind sie notwendig, und das Auslassen eines Semikolons führt bei der Compilation zu einer Fehlermeldung.

Einstieg in die Programmierung

Zusatzaufgabe 1
Die ersten Hamster–Programme

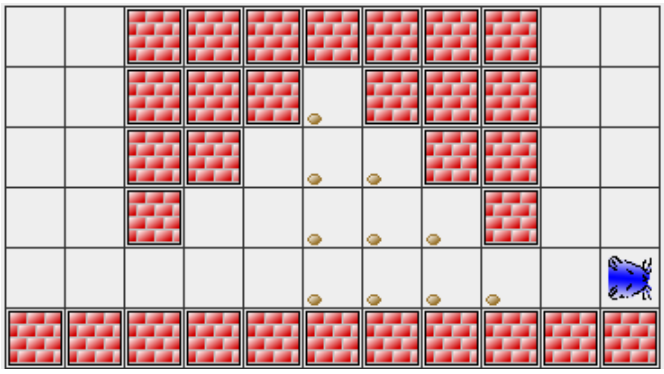
Aufgabe:

a) Gegeben ist folgendes Territorium:



Speichere das Territorium in einem Ordner „Zusatz1“ unter demselben Namen.

Schreibe nun ein Programm, welches das oben dargestellte Territorium in folgendes überführt:



Speichere das Programm ebenfalls im Ordner „Zusatz1“ unter demselben Namen.

b) Überlege dir eine schöne Hamster–Geschichte für das in Aufgabenteil a dargestellte Szenario.

Merkblatt 2

Programme erstellen im Überblick

Zur Verwaltung von Territorien und Programmen ...

- Wir speichern jedes Territorium unter *...\Programme* in einem separaten Ordner, der denselben Namen wie das Territorium bekommt.
- Alle Programme zu einem bestimmten Territorium speichern wir in dem gleichen Ordner, in welchem sich auch das Territorium befindet.
- Wir geben beim Speichern von Territorien und Programmen niemals eine Dateierweiterung, wie beispielsweise *„.ter“* oder *„.ham“* an.

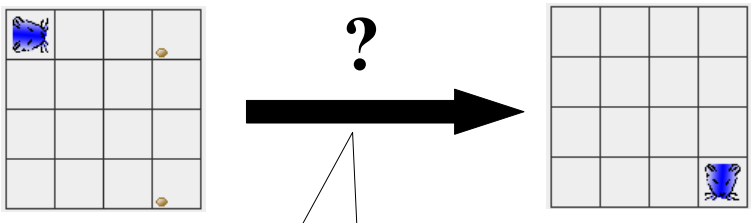
Zur Erstellung von Programmen ...

- Ein *initialer Anfangszustand* eines Territoriums wird durch ein *Hamster-Programm* in einen gewünschten *Endzustand* überführt.
- Die geschweiften Klammern eines Blocks stehen immer in derselben Spalte – also auf einer vertikalen Linie.
- Die Programmzeilen eines Blocks rücken wir immer um vier Leerzeichen bzw. einen Tabulator ein.
- Immer wenn wir unser Programm erneut ausführen, müssen wir vorher den Simulator in den Anfangszustand zurücksetzen.
- Wir compilieren ca. alle zehn Programmzeilen, um Fehler frühzeitig zu erkennen.

Arbeitsblatt 2

Prozeduren

Ein Konzept zur Definition neuer Befehle



bisher

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

Idee: _____

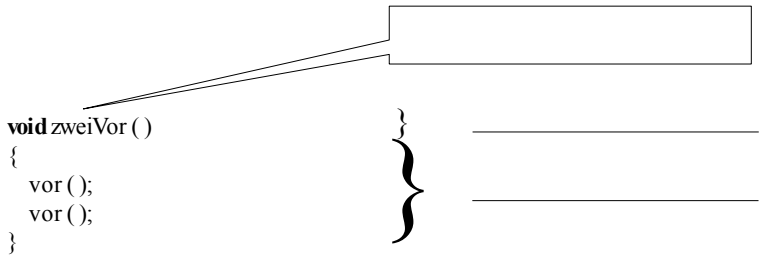
Umsetzung: _____

Befehl:

Befehl:

AB 2: Prozeduren ITG, 8

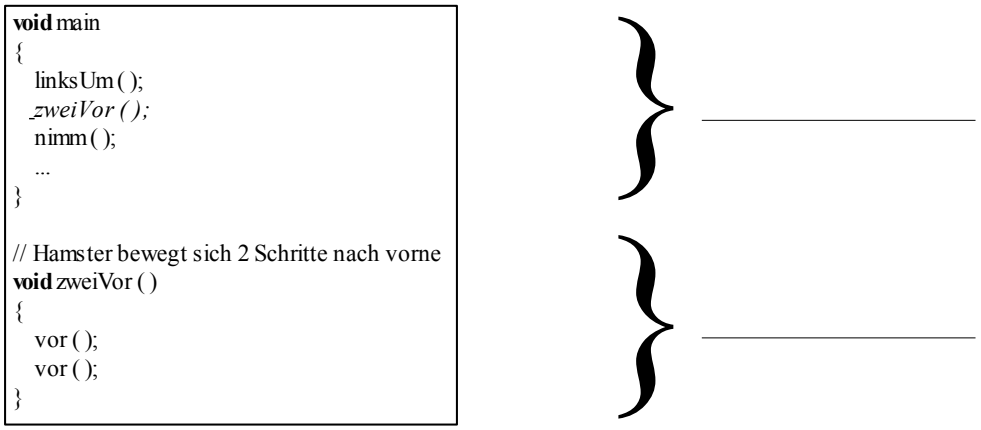
Begriffe:



Beachte:

- Der Prozedurkopf wird *nicht* mit einem Semikolon abgeschlossen.
- Der Prozedurname
  - darf keine Leerzeichen enthalten,
  - darf nicht mit einer Zahl beginnen,
  - sollte nur Buchstaben, Zahlen und „_“ (als Trennzeichen) enthalten,
  - sollte keine Umlaute (ä, ö, ü, ß) enthalten,
  - sollte aussagekräftig sein.
- Jeder Prozedur sollte ein kurzer *Kommentar* vorangestellt werden (siehe Prozedur *zweiVor* im nächsten Abschnitt), der Aufschluss über den Inhalt bzw. die Funktionsweise der Prozedur gibt.

Prozeduraufruf:

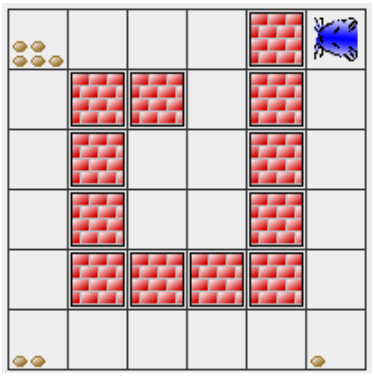


Vorteile von der Verwendung von Prozeduren:

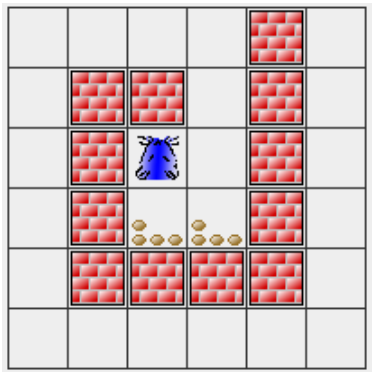
Musterbeispiel 1

Programmentwurf mit Prozeduren

Aufgabe (Winterschlaf)**:**
Schreibe ein Programm, welches das Territorium



in



überführt.

Lösung:

```
void main
{
   linksUm( );
   fuenfVor( );
   nimm( );
   rechtsUm( );
   fuenfVor( );
   nimm2( );
   rechtsUm( );
   fuenfVor( );
   nimm2( );
   nimm2( );
   nimm( );
   rechtsUm( );
   dreiVor( );
   rechtsUm( );
   dreiVor( );
   gib4( );
   rechtsUm( );
   vor( );
   gib4( );
   rechtsUm( );
   vor( );
}
```

```
// Hamster geht 5 Schritte vor
void fuenfVor( )
{
   vor( );
   vor( );
   vor( );
   vor( );
   vor( );
}

// Hamster dreht sich 90° rechts
void rechtsUm( )
{
   linksUm( );
   linksUm( );
   linksUm( );
}

// Hamster nimmt 2 Körner auf
void nimm2( )
{
   nimm( );
   nimm( );
}
```

```
// Hamster geht 3 Schritte vor
void dreiVor( )
{
   vor( );
   vor( );
   vor( );
}

// Hamster legt 4 Körner ab
void gib4( )
{
   gib( );
   gib( );
   gib( );
   gib( );
}
```

Übungsblatt 1

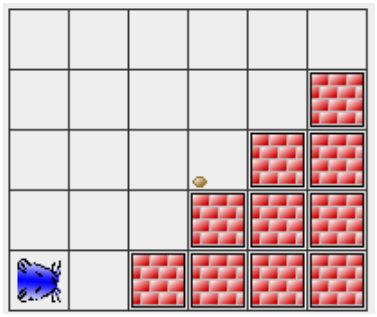
Programmentwurf mit Prozeduren

Aufgabe 1 (Unser Hamster will hoch hinaus):

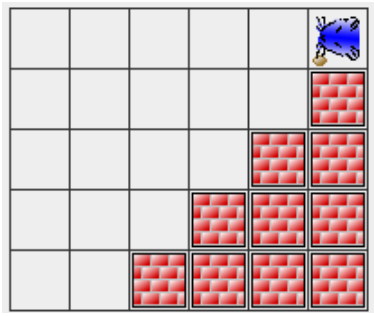
Öffne das Territorium „Bergsteiger“ im Simulator.

Dieses Mal will unser Hamster hoch hinaus. Er will zuerst den Fuß des Bergs erklimmen, dann auf halber Höhe das Korn aufnehmen und es auf dem Berggipfel ablegen, sodass kein Hamster auf dieser Welt es ihm stehlen kann. Nach getaner Arbeit dreht sich unser Hamster um 180°, um den Ausblick auf dem Berggipfel zu genießen.

Schreibe ein Programm, welches das Territorium



in



überführt.

Speichere es im Ordner „Bergsteiger“ unter dem Namen „Gipfelreiter“.

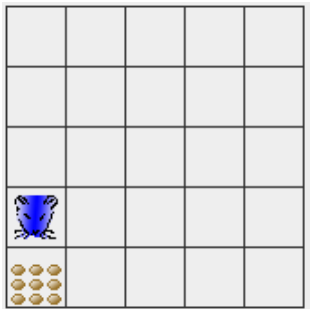
Verwende beim Programmentwurf unbedingt Prozeduren!

Achte auf die Einhaltung der Stichpunkte auf Merkblatt 2.

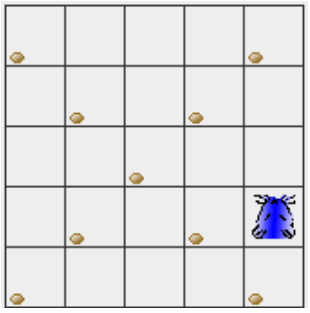
Aufgabe 2 (Unser Hamster lernt Geometrie) [Bol02]:

Unser Hamster steht vor einer neuen Aufgabe. Im Mathematikunterricht soll er in einem Quadrat die Diagonalen einzeichnen. Kannst du ihm dabei helfen?

Schreibe ein Programm, welches das Territorium



in



überführt.

Speichere zunächst das Territorium in einem Ordner „Diagonalen“ unter demselben Namen.

Speichere das Programm ebenfalls unter dem Namen „Diagonalen“.

Zusatzaufgabe (Entwurf einer eigenen Aufgabe):

Bei dieser Aufgabe darfst du nun kreativ werden und dir selbst eine Hamster–Aufgabe überlegen.

Gehe dabei folgendermaßen vor:

- Konstruiere ein Territorium, und speichere es unter einem sinnvollen Namen.
- Überlege dir eine Aufgabe für den Hamster bzw. einfach nur den gewünschten Endzustand des Territoriums.
- Schreibe nun das Programm, welches den initialen Anfangszustand des Territoriums in den gewünschten Endzustand überführt.
- Speichere dein Programm unter demselben Namen wie das Territorium.

Arbeitsblatt 3

Die WHILE-Schleife

als Beispiel für eine Wiederholungsanweisung

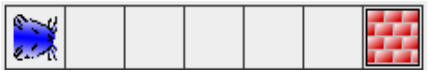
Beispiel 1:

Unser Hamster soll alle Körner aufnehmen, die sich auf dem aktuellen Feld befinden. Schreibe eine Prozedur *nimmAlle( )*.

umgangssprachlich

Prozedur *nimmAlle*

Beispiel 2:

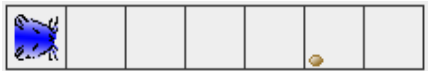


Unser Hamster soll bis zur Mauer gehen, ohne gegen sie zu stoßen. Schreibe eine Prozedur *vorBisMauer( )*.

umgangssprachlich

Prozedur *vorBisMauer*

Beispiel 3:



Unser Hamster soll auf das Feld mit dem Korn laufen. Schreibe eine Prozedur *vorBisKorn( )*.

umgangssprachlich

Prozedur *vorBisKorn*

Beispiel 4:

Was geschieht, wenn auf dem Territorium

das Programm

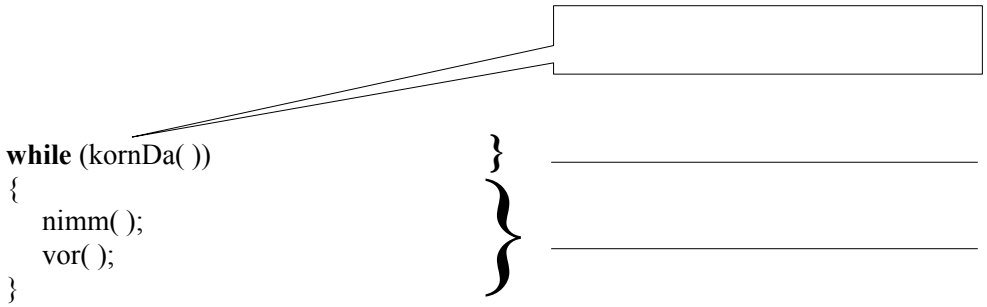
```
void main ( )
{
   vor( );
   while (kornDa( ))
   {
        nimm( );
        vor( );
   }
}
```

ausgeführt wird?

Antwort

AB 3: WHILE-Schleife ITG, 8

Begriffe:



Schleifenbedingungen:

Schleifenbedingung	***Bedeutung***
	auf aktuellem Feld befindet sich mindestens ein Korn
	auf aktuellem Feld befindet sich kein Korn
	auf Feld vor Hamster befindet sich weder eine Mauer, noch das Ende des Territoriums
	auf Feld vor Hamster befindet sich eine Mauer oder das Ende des Territoriums
	Im Maul des Hamsters befinden sich keine Körner.
	Im Maul des Hamsters befindet sich mindestens ein Korn.

Funktionsweise der WHILE-Schleife:

Die WHILE-Schleife ist eine sogenannte *Wiederholungsanweisung*.
Bei ihrer Ausführung wird im Schleifenkopf zunächst überprüft, ob die Schleifenbedingung erfüllt ist. Falls dies nicht der Fall ist, wird der auf die WHILE-Schleife folgende Block {...} übersprungen, und die Schleife ist unmittelbar beendet.

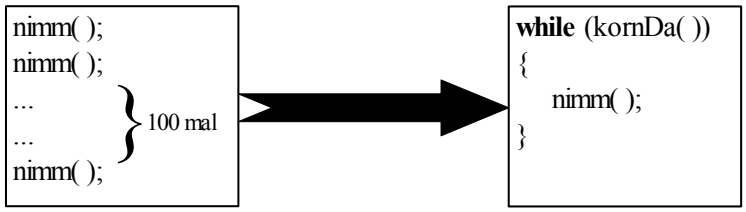
Falls die Bedingung erfüllt ist, wird der Schleifenrumpf *genau ein Mal* ausgeführt. Anschließend wird die Schleifenbedingung erneut ausgewertet. Falls sie immer noch erfüllt ist, wird der Schleifenrumpf ein weiteres Mal ausgeführt. Dieser Prozess wiederholt sich, *solange* die Schleifenbedingung erfüllt ist.

Beachte:

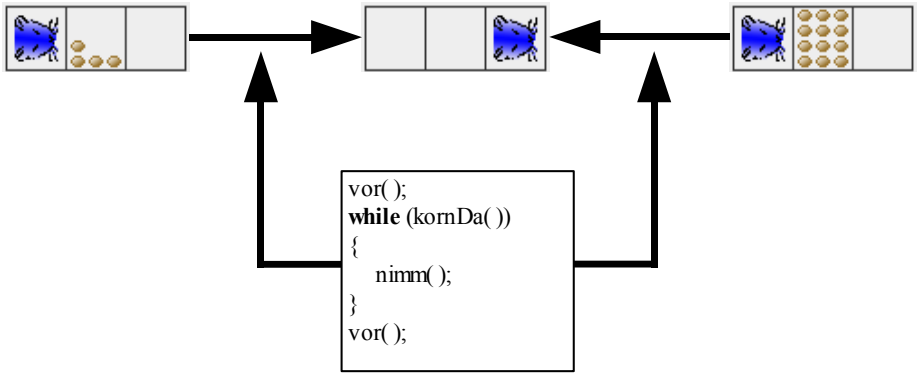
- Der Schleifenkopf wird nicht mit einem Semikolon abgeschlossen.
- Die Befehle im Schleifenrumpf rücken wir um 4 Leerzeichen bzw. einen Tabulator ein.

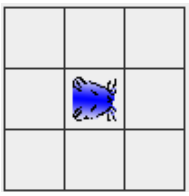
Vorteile & Notwendigkeit

1.



2.



Was geschieht, wenn auf dem Territorium  das Programm

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

ausgeführt wird?

Antwort

Übungsblatt 2
Programmentwurf mit WHILE-Schleifen

Aufgabe 1 (Licht aus!) [Bol02]:

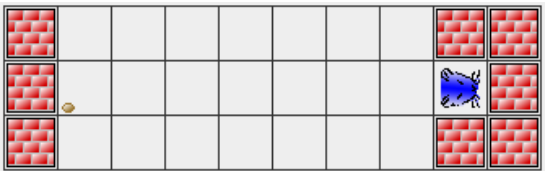
Unser Hamster will sich etwas sportlich betätigen. Dazu sucht er eine Turnhalle beliebiger Größe auf und absolviert seine sportlichen Übungen. Plötzlich fällt das Licht aus, und da es im Freien bereits dunkel ist, weiß unser Hamster nicht, an welcher Stelle er sich in der Halle befindet. Jetzt ist schnelles Handeln angesagt. An jeder Ecke der Turnhalle befindet sich ein Notausgang. Kannst du unserem Hamster helfen, einen dieser Notausgänge zu erreichen?

Vorgehensweise:

1. Konstruiere ein beliebig großes Territorium (Turnhalle) ohne Körner und Mauern. Unser Hamster sitzt an einer beliebigen Stelle und schaut in eine beliebige Richtung.
2. Speichere das Territorium in einem Ordner „LichtAus“ unter demselben Namen.
3. Schreibe ein Programm, welches unseren Hamster in eine beliebige Ecke des Territoriums bewegt. Verwende unbedingt WHILE-Schleifen, aber noch keine Prozeduren! Speichere das Programm im Ordner „LichtAus“ unter dem Namen „LichtAus_oP“.
4. Teste dein Programm nun. Falls das Programm funktioniert, teste es außerdem mit dem Territorium „LichtAus_2“ (Tauschordner). Kopiere „LichtAus_2“ zunächst vom Tauschordner in den von dir erstellten Ordner „LichtAus“ auf deinem Homeverzeichnis.
5. Ändere dein Programm ab, und verwende eine sinnvolle Prozedur. Speichere das modifizierte Programm im Ordner „LichtAus“ unter dem Namen „LichtAus_mP“
6. Teste dein Programm nun mit den Territorien „LichtAus“ und „LichtAus_2“.

Aufgabe 2 (Erblindet!):

Schaut ihn euch an, unseren armen Hamster! Er ist erblindet, sitzt in seinem Bau und weiß nicht, wo sich der Ausgang befindet. Dabei hat er einen so großen Hunger, dass er unbedingt das Korn erreichen will, welches er schon von weitem riecht.

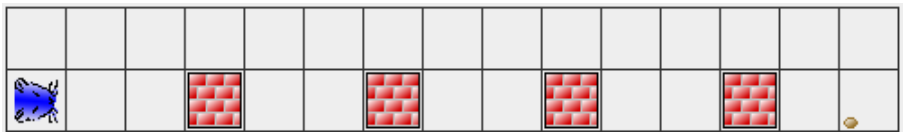


Beispielterritorium

1. Speichere das Territorium in einem Ordner „Erblindet“ unter demselben Namen.
2. Schreibe ein Programm für das oben beschriebene Szenario. Verwende unbedingt WHILE-Schleifen und ggf. sinnvolle Prozeduren. *Beachte, dass unser Hamster in seiner Startposition in eine beliebige Richtung schaut und dass das Korn beliebig weit entfernt sein kann (jedoch auf einer Höhe mit dem Hamster und direkt vor einer Mauer).*
3. Teste dein Programm nun. Falls das Programm funktioniert, teste es außerdem mit dem Territorium „Erblindet_2“ (Tauschordner). Kopiere „Erblindet_2“ zunächst vom Tauschordner in den von dir erstellten Ordner „Erblindet“ auf deinem Homeverzeichnis.

Aufgabe 3 (Hürdenläufer) [aus [Bol02], modifiziert]:

Olympische Sommerspiele 2006 in Wiesloch! Unser Hamster ist natürlich auch dabei und hat sich im 110-m Hürdenlauf qualifiziert. Auf der Ziellinie liegt ein Korn für den Sieger.

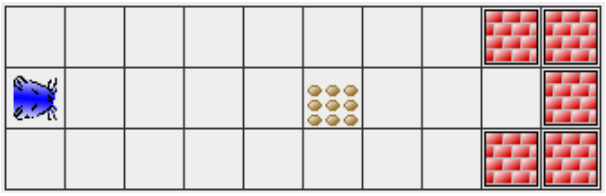


Beispielterritorium

1. Speichere das Territorium im Ordner „Huerdenlaeufer“ unter demselben Namen.
2. Schreibe ein Programm für das oben beschriebene Szenario. *Beachte, dass sowohl der Abstand zwischen den Hürden beliebig variieren kann, als auch der Abstand zwischen der letzten Hürde und dem Korn.*
3. Speichere das Programm im Ordner „Huerdenlaeufer“ unter demselben Namen.
4. Teste dein Programm nun. Falls das Programm funktioniert, teste es außerdem mit dem Territorium „Huerdenlaeufer_2“ (Tauschordner). Kopiere „Huerdenlaeufer_2“ zunächst vom Tauschordner in den von dir erstellten Ordner „Huerdenlaeufer“ auf deinem Homeverzeichnis.

Aufgabe 4 (Heimweg):

Unser Hamster befindet sich gerade auf dem Heimweg in seinen Bau. Plötzlich traut er seinen Augen nicht, als er in weiter Ferne einen großen Haufen Körner erspäht. Diesen sammelt er ein und transportiert ihn in seinen Bau. Dort legt er alle Körner wieder ab.



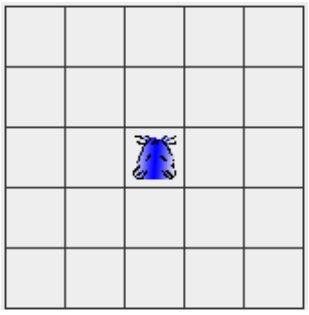
Beispielterritorium

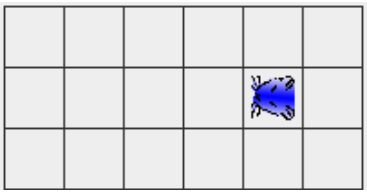
1. Speichere das Territorium in einem Ordner „Heimweg“ unter demselben Namen.
2. Schreibe ein Programm für das oben beschriebene Szenario. *Beachte, dass die Größe des Körnerhaufens, sowie der Abstand zwischen Hamster und Körnerhaufen, als auch der Abstand zwischen Körnerhaufen und Hamsterbau variieren können.*
3. Speichere das Programm im Ordner „Heimweg“ unter demselben Namen.
4. Teste dein Programm nun. Falls das Programm funktioniert, teste es außerdem mit dem Territorium „Heimweg_2“ (Tauschordner). Kopiere „Heimweg_2“ zunächst vom Tauschordner in den von dir erstellten Ordner „Heimweg“ auf deinem Homeverzeichnis.

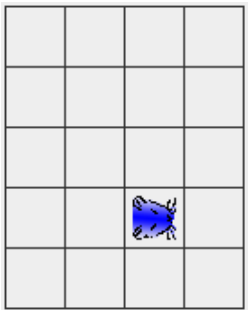
Musterbeispiel 2

Programmentwurf mit WHILE-Schleifen

Übungsblatt 2, Aufgabe 1:
Mögliche Territorien sind beispielsweise:







Unser Hamster soll, ausgehend von einem beliebigen Territorium, in eine beliebige Ecke des Territoriums laufen.

Lösung:

ohne Prozedur

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

mit Prozedur

```
void main ( )
{
   vorBisMauer ( );
   linksUm( );
   vorBisMauer ( );
}

// Hamster geht vor, solange vorne frei
 void vorBisMauer ( )
{
   while (vornFrei( ))
   {
         vor( );
   }
}
```

Vertiefungsaufgabe 1

Die DO..WHILE-Schleife

als Beispiel einer weiteren Wiederholungsanweisung

Neben der WHILE-Schleife gibt es auch noch die DO..WHILE-Schleife:

```
do
{
   Anweisung(en)
}
while (Schleifenbedingung)
```

Welchen Unterschied zur WHILE-Schleife erwartest du?

Führe nun auf dem Territorium  folgende Programme aus:

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

```
void main( )
{
   do
   {
        vor( );
   }
   while (vornFrei( ));
}
```

Hat sich deine Vermutung bestätigt? Wie unterscheiden sich beide Schleifen?

Übungsblatt 3

Geschachtelte WHILE-Schleifen

Aufgabe 1 (Gefräßiger Vierbeiner):

Unser Hamster ist aus seinem Winterschlaf erwacht. Völlig hungrig und abgemagert macht er sich auf den Weg, um seine Fettreserven wieder aufzufüllen. Er frisst und frisst – bis er vor der Mauer völlig übersättigt zusammenbricht und einschläft.

Musterlösung:

ohne Prozedur

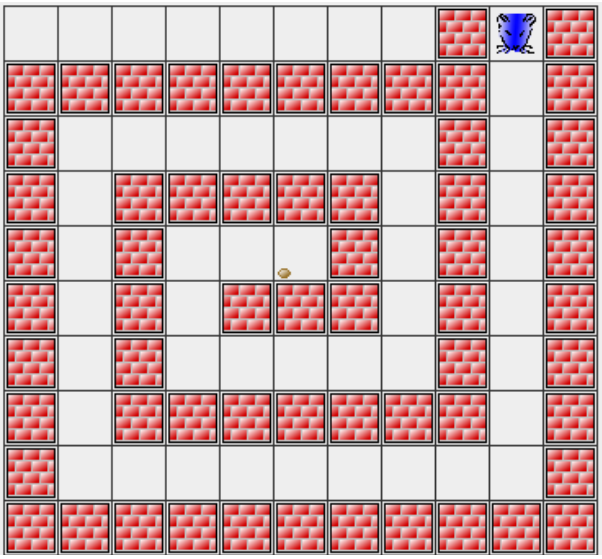
```
void main ( )
{
    while (vornFrei( ))
    {
        while (kornDa( ))
        {
            nimm( );
        }
        vor( );
    }
}
```

mit Prozedur

```
// Hauptprogramm
void main ( )
{
    while (vornFrei( ))
    {
        nimmAlle( );
        vor( );
    }
}

// Hamster nimmt alle Körner auf akt. Feld
void nimmAlle( )
{
    while (kornDa( ))
    {
        nimm( );
    }
}
```

Aufgabe 2 (Labyrinth) [aus [UHe], modifiziert]:

Unser Hamster macht einfach alles für Körner! Sogar durch ein unterirdisches Labyrinth will er laufen, nur um das eine Korn zu fressen. Kannst du ihn sicher durch das Labyrinth navigieren?



1. Speichere das Territorium in einem Ordner „Labyrinth“ unter demselben Namen.
2. Schreibe ein Programm für das oben beschriebene Szenario. *Verwende unbedingt WHILE-Schleifen und Prozeduren.*
3. Speichere das Programm im Ordner „Labyrinth“ unter demselben Namen.
4. Teste dein Programm nun. Falls das Programm funktioniert, teste es außerdem mit dem Territorium „Labyrinth_2“ (Tauschordner). Kopiere „Labyrinth_2“ zunächst vom Tauschordner in den von dir erstellten Ordner „Labyrinth“ auf deinem Homeverzeichnis.

Anmerkung:
Dein Programm sollte ca. 20 Zeilen lang sein.

Aufgabe 3 (Entwurf einer eigenen Aufgabe):

Überlege dir nun eine eigene Aufgabe, die nur unter Einsatz von geschachtelten Wiederholungsanweisungen (WHILE-Schleifen) lösbar ist.

Gehe dabei folgendermaßen vor:

- Konstruiere ein Territorium, und speichere es unter einem sinnvollen Namen.
- Überlege dir eine Aufgabe für den Hamster bzw. einfach nur den gewünschten Endzustand des Territoriums.
- Schreibe nun das Programm, welches den initialen Anfangszustand des Territoriums in den gewünschten Endzustand überführt.
- Speichere dein Programm unter demselben Namen wie das Territorium.

Arbeitsblatt 4

Programmierfehler

	Syntaxfehler	**Semantikfehler**
Beispiele		
Beschreibung		
Erkennung durch Compiler		

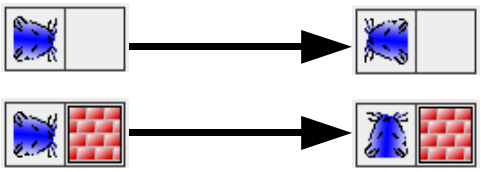
Arbeitsblatt 5

Die IF-Anweisung

als Beispiel für eine Auswahlanweisung

Beispiel 1:

vorher nachher



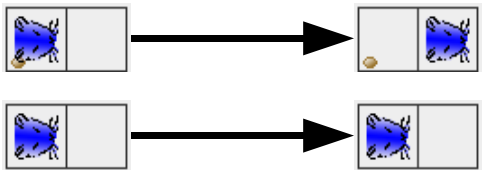
umgangssprachlich, Variante 1

Programmfragment, Variante 1

umgangssprachlich, Variante 2

Programmfragment, Variante 2

Beispiel 2:



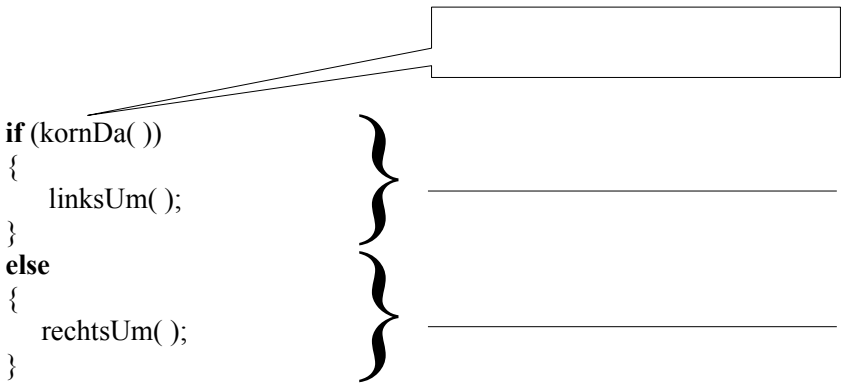
umgangssprachlich, Variante 1

Programmfragment, Variante 1

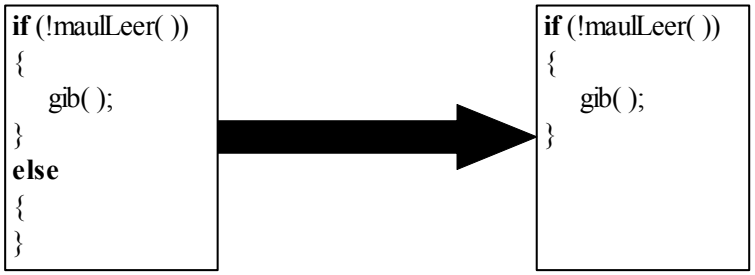
umgangssprachlich, Variante 2

Programmfragment, Variante 2

Begriffe:



Funktionsweise:
Die IF-Anweisung ist eine *Auswahlanweisung*.
Trifft die *Auswahlbedingung* zu, so werden die Anweisungen des *IF-Zweigs* ausgeführt. Andernfalls werden die Anweisungen des *ELSE-Zweigs* ausgeführt.
Einen leeren ELSE-Zweig kann man auch auslassen, z.B.



Den IF-Zweig kann man jedoch niemals auslassen, da in ihm die Auswahlbedingung steht. Es ist jedoch möglich, im IF-Zweig einen leeren Block anzugeben (siehe Beispiel 2).

Beachte:

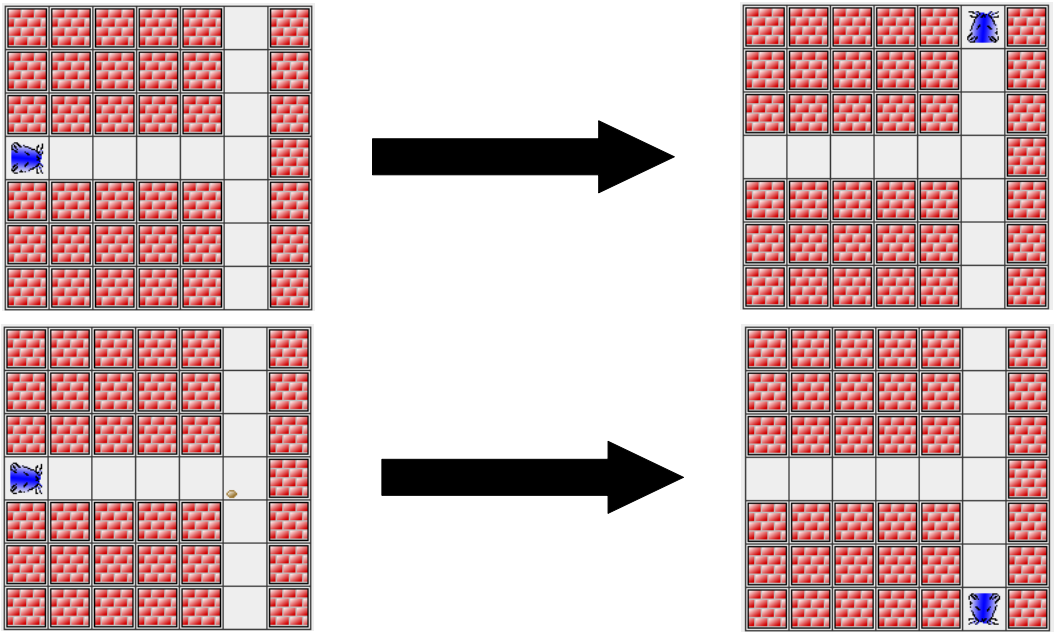
- Die Zeilen mit IF und ELSE werden nicht mit Semikola abgeschlossen.
- Sind IF- und ELSE-Zweig vorhanden, wird immer entweder der IF-Zweig oder der ELSE-Zweig ausgeführt.
- Ist nur der IF-Zweig vorhanden, wird dieser entweder ausgeführt, oder er wird nicht ausgeführt.

Vorteil oder Notwendigkeit?

Musterbeispiel 3

Programmentwurf mit IF-Anweisungen

Unser Hamster erkundet die Landschaft. Um zu verhindern, dass er sich verirrt, benutzt er Körner als Wegmarken. Nun befindet er sich auf dem Rückweg. Trifft er auf eine Wegkreuzung an der kein Korn liegt, so geht er nach links, andernfalls nimmt er das Korn und geht nach rechts.



Musterlösung:

```
// Hauptprogramm
void main( )
{
    vorBisMauer( );
    if (kornDa( ))
    {
        nimm( );
        rechtsUm( );
    }
    else
    {
        linksUm( );
    }
    vorBisMauer( );
}
```

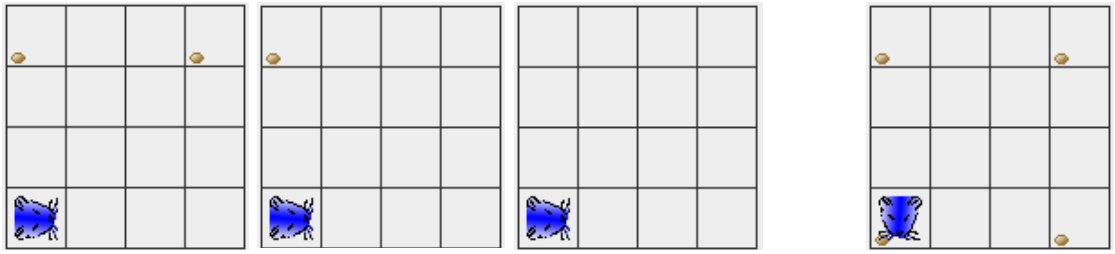
```
// Hamster geht vor bis zur
// nächsten Mauer
void vorBisMauer( )
{
    while (vornFrei( ))
    {
        vor( );
    }
}

// Hamster dreht sich 90° nach rechts
void rechtsUm( )
{
    linksUm( );
    linksUm( );
    linksUm( );
}
```

Übungsblatt 4
Programmentwurf mit IF–Anweisungen

Aufgabe 1 (Feng–Shui–Hamster)**:**

Unser Hamster hat sich einen neuen (4×4 großen) Bau gebaut. Jetzt will er diesen harmonisch einrichten, sodass sich in jeder Ecke des Baus genau ein Korn befindet. Allerdings befinden sich in einigen Ecken bereits Körner – in anderen nicht. Kannst du unserem Hamster helfen?

Beispiele für Territorien im Initialzustand: *Territorium im Endzustand:*



Vergesse nicht, dem Hamster vor Programmablauf mindestens vier Körner ins Maul zu legen.

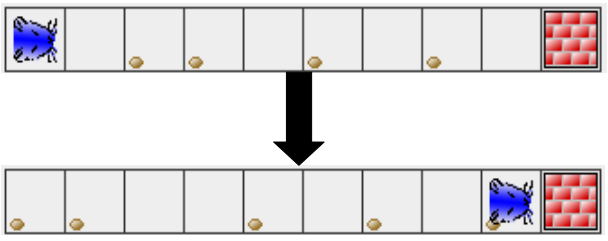
Speichere Territorium und Programm in einem Ordner „FengShui“ unter demselben Namen.

Verwende unbedingt die IF-Anweisung und Prozeduren sowie ggf. WHILE-Schleifen.

Aufgabe 2 (Inverter)**:**

Unser Hamster hat dieses Mal die ehrenvolle Aufgabe, die vor ihm liegende Körnerreihe zu invertieren, d.h. wenn er auf einem Korn steht, soll er es aufnehmen, wenn er keins vorfindet, soll er ein Korn ablegen.

Beispiel:



Vergesse nicht dem Hamster vor Programmablauf Körner ins Maul zu legen (z.B. 10 Stück) – das machst du im Simulator mit dem sechsten Button von links.

Speichere Territorium und Programm in einem Ordner „Inverter“ unter demselben Namen.

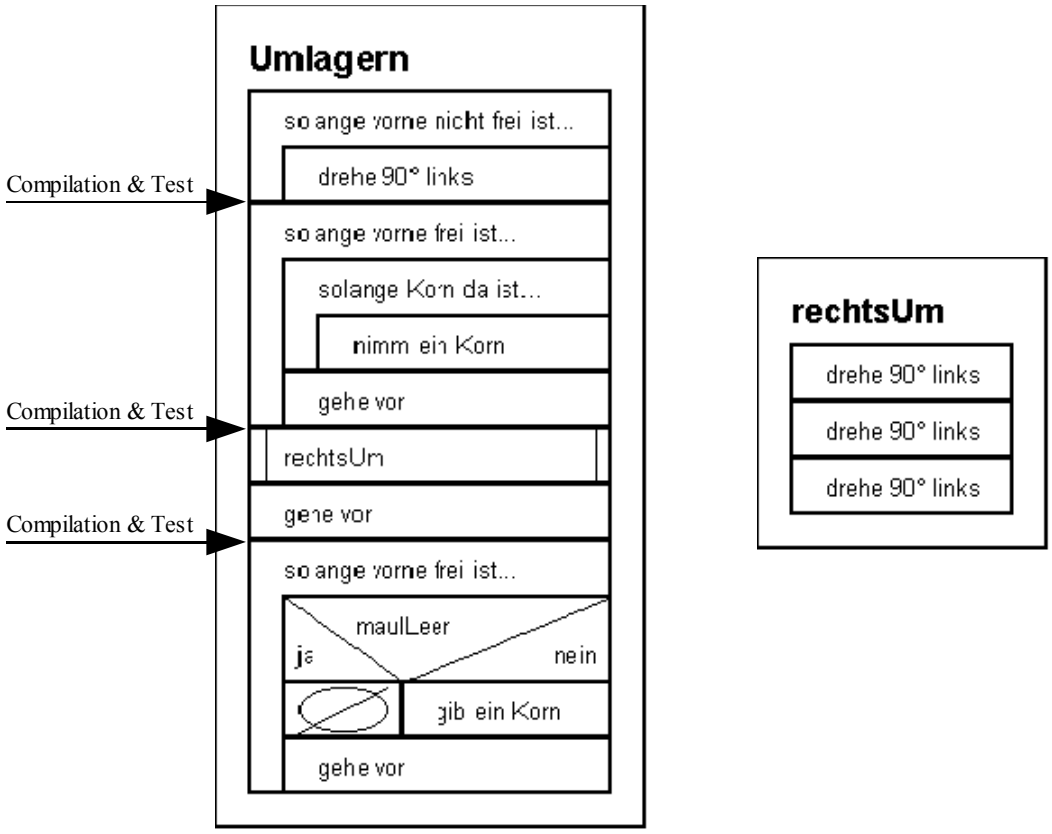
Verwende unbedingt die IF-Anweisung und WHILE–Schleifen sowie ggf. Prozeduren.

Übungsblatt 5

Programmentwurf mit Struktogrammen

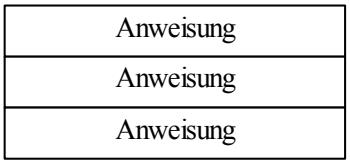
Aufgabe 1 (Umsetzung eines Struktogramms in ein Programm):

Setze folgendes Struktogramm in ein Programm um. Compiliere und teste dein Programm jeweils an den angegebenen Stellen.

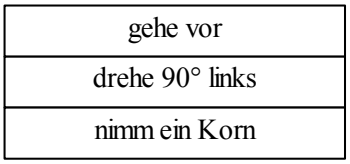


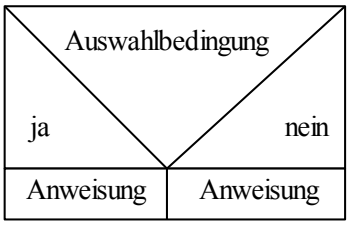
1. Speichere das Programm in einem Ordner „Umlagern“ unter demselben Namen.
2. Kopiere das Territorium „Umlagern“ (Tauschordner) in den erstellten Ordner „Umlagern“ auf deinem Homeverzeichnis.
3. Öffne nun das Territorium „Umlagern“, und teste dein Programm.

Arbeitsblatt 6
Struktogramme
zur alternativen Formulierung von Algorithmen

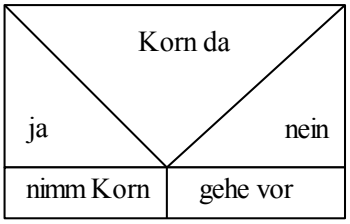


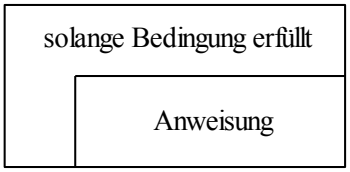
Befehlssequenz, z.B:



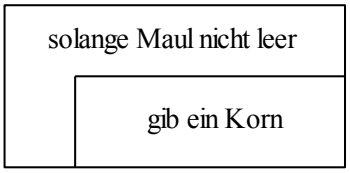


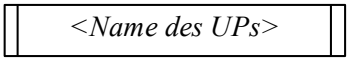
Auswahlstruktur
(IF...ELSE), z.B:



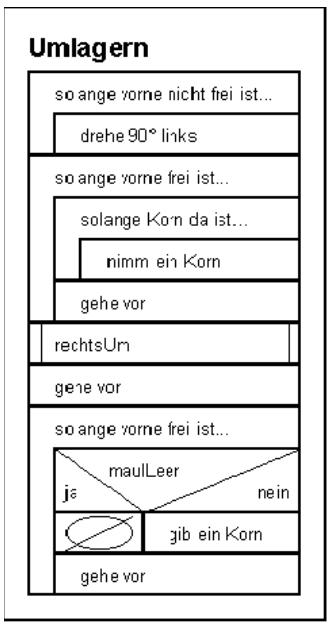


Wiederholungsanweisung
(WHILE), z.B:

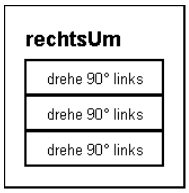


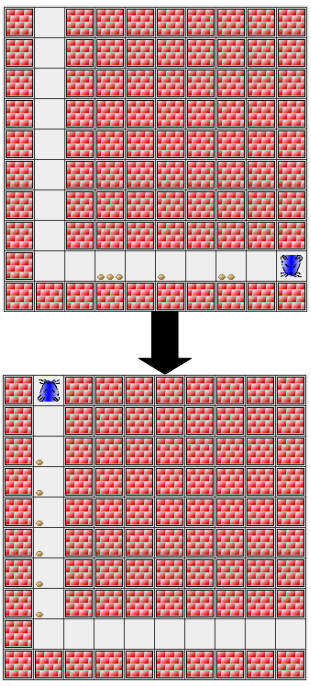


Unterprogrammaufruf
(Prozeduren), z.B:



Beispielstruktogramm





AB 6: Struktogramme ITG, 8

Sinn & Zweck von Struktogrammen:

Programmcode für das Beispielstruktogramm:

```
void main ( )
{
   while (!vornFrei( ))
   {
        linksUm( );
   }
   while (vornFrei( ))
   {
        while (kornDa( ))
        {
           nimm( );
        }
        vor( );
   }
   rechtsUm( );
   vor( );
```

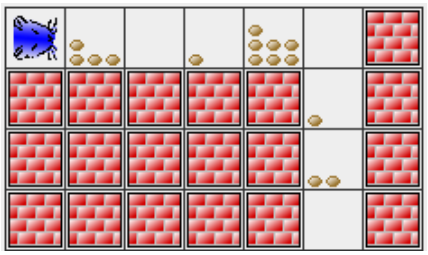
```
   while (vornFrei( ))
   {
      if (maulLeer( ))
      {
      }
      else
      {
         gib( );
      }
      vor( );
   }
}

// Hamster dreht sich 90° nach rechts
void rechtsUm( )
{
   linksUm( );
   linksUm( );
   linksUm( );
}
```

Vertiefungsaufgabe 2
Programmentwurf mit Struktogrammen

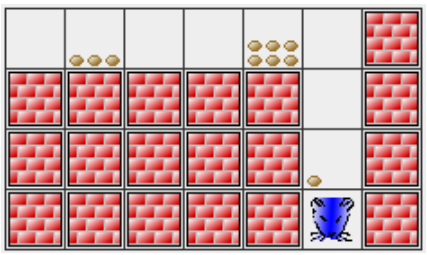
Aufgabe (Fastenzeit):

In letzter Zeit hat es unser Hamster etwas mit dem Körnerfressen übertrieben. Jetzt ist er übergewichtig und will schnellstens die überschüssigen Pfunde wieder los werden. Dazu nimmt er sich vor, von jedem Körnerhaufen nur noch ein Korn zu nehmen, und den Rest liegen zu lassen.

Entwerfe einen Algorithmus, welcher das Territorium



in



überführt.

Beachte, dass
- das „**L**“ des Territoriums beliebig groß sein kann,
- es beliebig viele Körnerhaufen geben kann,
- die Körnerhaufen beliebig groß sein können,
- die Abstände zwischen den Körnerhaufen beliebig variieren können,
- das „**L**“ des Territoriums immer mit einer Mauer „abgeschlossen“ wird.

Vorgehensweise:

1. Formuliere deinen Algorithmus in Form eines Struktogramms. (Überführe das Struktogramm jedoch noch nicht in ein Programm.)

2. Suche dir nun einen Partner mit dem du dein Struktogramm austauschst.

3. Schreibe nun ein Programm für das Struktogramm deines Partners. Speichere es in einem Ordner „Fastenzeit“ unter demselben Namen.

4. Konstruiere nun das oben angegebene Territorium, und speichere es ebenfalls im Ordner „Fastenzeit“ unter demselben Namen.

5. Setzt euch nun zusammen, und überprüft gemeinsam die jeweiligen Struktogramme und Programme.

Übungsblatt 6
Vermischte Aufgaben

Aufgabe 1 (Fastenzeit):
(Diese Aufgabe darf nur von denjenigen Schülern bearbeitet werden, die nicht die Vertiefungsaufgabe 2 gelöst bzw. bearbeitet haben.)

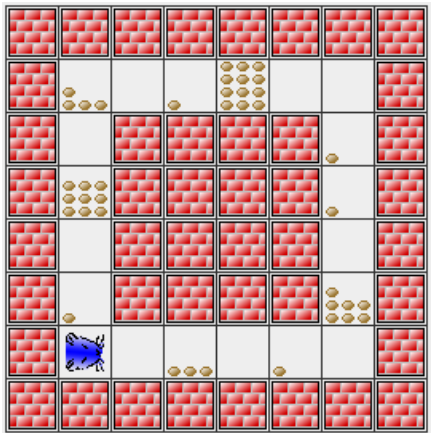
Betrachte das Szenario von Vertiefungsaufgabe 2 (Fastenzeit).

1. Speichere das Territorium in einem Ordner „Fastenzeit“ unter demselben Namen.
2. Schreibe ein Programm für das in der Aufgabenstellung beschriebene Szenario, und speichere es im Ordner „Fastenzeit“ unter demselben Namen.
3. Teste dein Programm nun. Falls das Programm funktioniert, teste es außerdem mit dem Territorium „Fastenzeit_2“ (Tauschordner). Kopiere „Fastenzeit_2“ zunächst vom Tauschordner in den von dir erstellten Ordner „Fastenzeit“ auf deinem Homeverzeichnis.

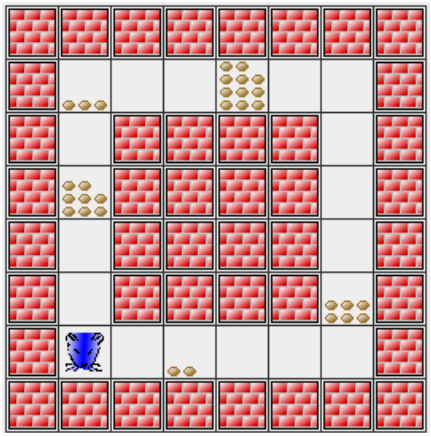
Aufgabe 2 (Rundgang):

Unser Hamster will einmal „ums Quadrat laufen“. Da er aber nicht zu schwer tragen möchte, entschließt er sich, von jedem Körnerhaufen nur ein Korn zu nehmen.

Schreibe ein Programm, welches das Territorium



in



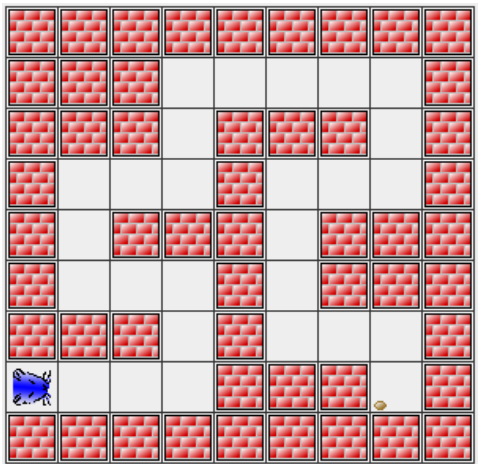
überführt.

1. Speichere das Territorium in einem Ordner „Rundgang“ unter demselben Namen.
2. Schreibe ein Programm für das oben beschriebene Szenario, und speichere es im Ordner „Rundgang“ unter demselben Namen.
3. Teste dein Programm nun. Falls das Programm funktioniert, teste es außerdem mit dem Territorium „Rundgang_2“ (Tauschordner). Kopiere „Rundgang_2“ zunächst vom Tauschordner in den von dir erstellten Ordner „Rundgang“ auf deinem Homeverzeichnis.

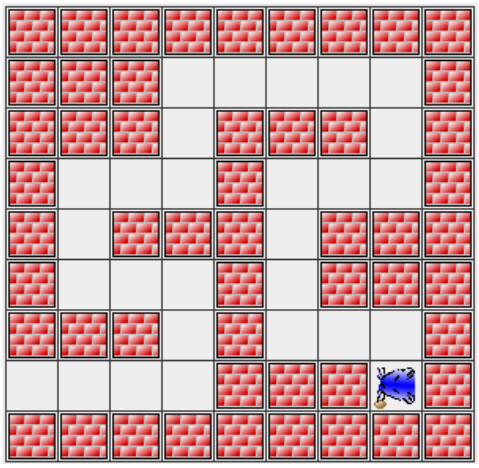
Aufgabe 3 (Maulwurf):

Unser Hamster hat einen unterirdischen Maulwurfgang entdeckt und will diesen unbedingt erkunden. Als Belohnung winkt ein Korn am Ende des Ganges. Kannst du ihm helfen, das Korn zu finden?

Schreibe ein Programm, welches das Territorium



in



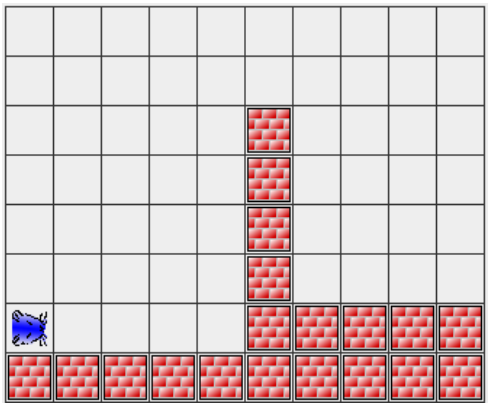
überführt.

1. Speichere das Territorium in einem Ordner „Maulwurf“ unter demselben Namen.
2. Schreibe ein Programm für das oben beschriebene Szenario, und speichere es im Ordner „Maulwurf“ unter demselben Namen.
3. Teste dein Programm nun. Falls das Programm funktioniert, teste es außerdem mit dem Territorium „Maulwurf_2“ (Tauschordner). Kopiere „Maulwurf_2“ zunächst vom Tauschordner in den von dir erstellten Ordner „Maulwurf“ auf deinem Homeverzeichnis.

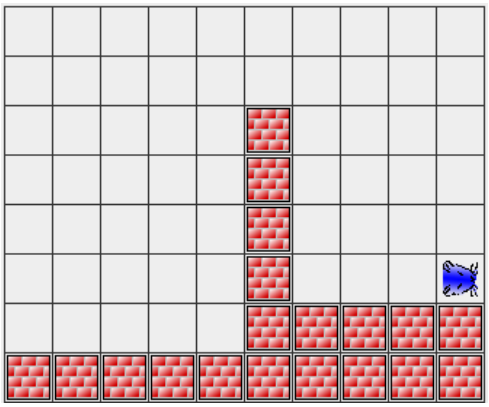
Aufgabe 4 (Hochsprung):

Zurück zu den olympischen Sommerspielen 2006 in Wiesloch. Unser Hamster will nun auch am Hochsprung–Wettbewerb teilnehmen.

Schreibe ein Programm, welches das Territorium



in



überführt.

1. Speichere das Territorium in einem Ordner „Hochsprung“ unter demselben Namen.
2. Schreibe ein Programm für das oben beschriebene Szenario. Verwende unbedingt die unten angegebene Prozedur *rechtsFrei( )* (als Schleifenbedingung). Speichere das Programm im Ordner „Hochsprung“ unter demselben Namen.
3. Teste dein Programm nun. Falls das Programm funktioniert, teste es außerdem mit dem Territorium „Hochsprung_2“ (Tauschordner). Kopiere „Hochsprung_2“ zunächst vom Tauschordner in den von dir erstellten Ordner „Hochsprung“ auf deinem Homeverzeichnis.

Prozedur *rechtsFrei( )*:

```
// prüft, ob Feld rechts vom Hamster frei ist
boolean rechtsFrei( )
{
    rechtsUm( );
    if (vornFrei( ))
    {
        linksUm( );
        return true;
    }
    else
    {
        linksUm( );
        return false;
    }
}
```

Glossar

Glossar ITG, 8

Weiterführende
Literatur & Links

E Musterlösungen zu den Arbeitsmaterialien

Auf den folgenden Seiten sind sämtliche (ursprünglich lückenbehaftete) Arbeitsmaterialien aufgeführt, welche von den Schülern im Unterricht ausgefüllt wurden. Bei den jeweiligen Lösungen handelt es sich jedoch nur um einen unverbindlichen Lösungsvorschlag meinerseits. Auch diese Musterlösungen befinden sich auf der Begleit–CD.

Rahmendaten

Schüler: *Paul Mustermann*

Klasse: *8 e*

Schule: *Ottheinrich-Gymnasium, Wiesloch*

Fach: *Informationstechnische Grundbildung (ITG)*

Zeitraum: *April 2006 – Mai 2006*

Lehrer: *Fruth*

Inhaltsverzeichnis

Arbeitsblatt 1

Probleme lösen mit dem Computer

Von der symbolischen Interaktion zu textuellen Befehlssequenzen

Bisher: Probleme lösen mit *Anwendersoftware* (MS Word, Paintbrush, HTML-Editor, ...) – meist durch symbolische Interaktion mit dem Rechner (z.B. Menüsteuerungen, Buttons, etc.).

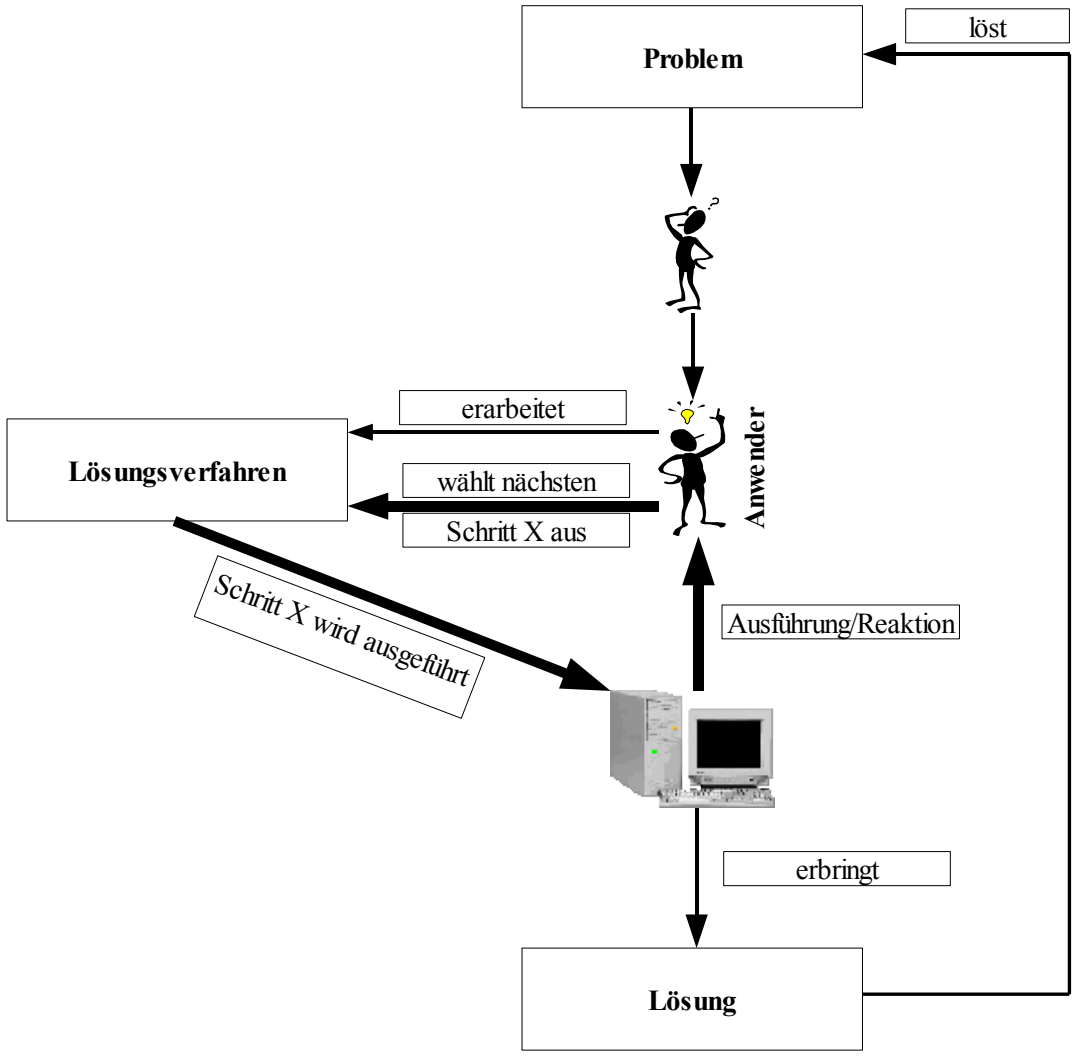


Abb. 1: Probleme lösen mit Anwendersoftware

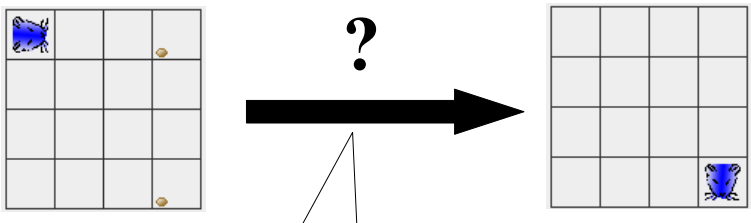
Bemerkung:
Bei der Problemlösung mit Anwendersoftware (Abb. 1) benutzt der *Anwender* eine Software, mit der er (meist symbolische) Befehle an den Computer gibt. Der Computer gibt ständig Rückmeldungen an den Anwender, worauf dieser mit dem nächsten Befehl fortfährt. Dieser Prozess wird solange wiederholt, bis die gewünschte Lösung erarbeitet wurde.

Musterlösung AB 2: Prozeduren ITG, 8

Arbeitsblatt 2

Prozeduren

Ein Konzept zur Definition neuer Befehle



bisher

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

Idee: *(Makro-)Befehle, wie bspw.* dreiVor *oder* rechtsUm *einführen*

Umsetzung: *mittels Prozeduren*

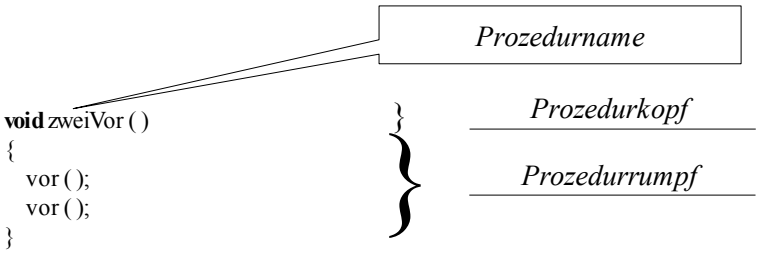
Befehl: *rechtsUm*

```
// Hamster dreht sich 90° rechts
void rechtsUm( )
{
   linksUm( );
   linksUm( );
   linksUm( );
}
```

Befehl: *dreiVor*

```
// Hamster geht drei Schritte vor
void dreiVor( )
{
   vor( );
   vor( );
   vor( );
}
```

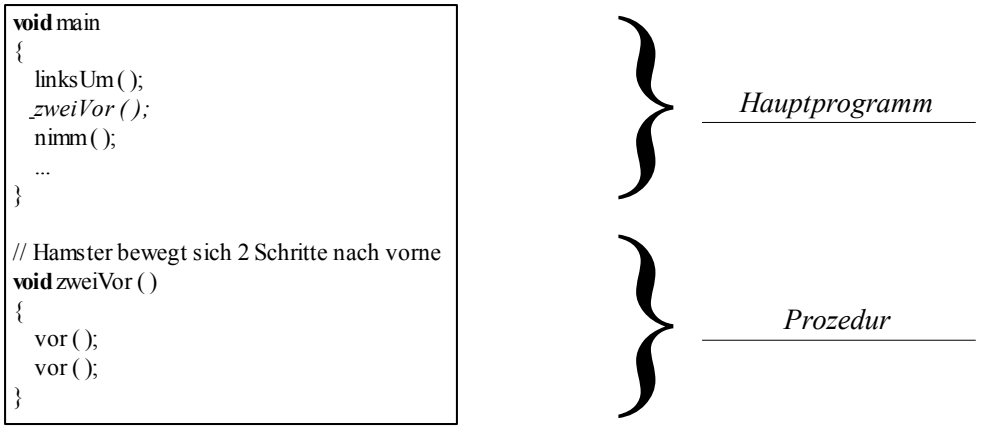
Begriffe:



Beachte:

- Der Prozedurkopf wird *nicht* mit einem Semikolon abgeschlossen.
- Der Prozedurname
  - darf keine Leerzeichen enthalten,
  - darf nicht mit einer Zahl beginnen,
  - sollte nur Buchstaben, Zahlen und „_“ (als Trennzeichen) enthalten,
  - sollte keine Umlaute (ä, ö, ü, ß) enthalten,
  - sollte aussagekräftig sein.
- Jeder Prozedur sollte ein kurzer *Kommentar* vorangestellt werden (siehe Prozedur *zweiVor* im nächsten Abschnitt), der Aufschluss über den Inhalt bzw. die Funktionsweise der Prozedur gibt.

Prozeduraufruf:



Vorteile von der Verwendung von Prozeduren:

kürzere und übersichtlichere Programme

Prozeduren kann man in folgenden Programmen wiederverwenden

Musterlösung AB 3: WHILE-Schleife ITG, 8

Arbeitsblatt 3

Die WHILE-Schleife

als Beispiel für eine Wiederholungsanweisung

Beispiel 1:

Unser Hamster soll alle Körner aufnehmen, die sich auf dem aktuellen Feld befinden. Schreibe eine Prozedur *nimmAlle( )*.

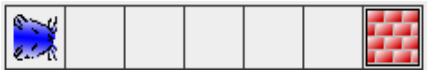
umgangssprachlich

Solange sich noch Körner auf dem aktuellem Feld befinden, nimm eins.

Prozedur *nimmAlle*

```
// Hamster nimmt alle Körner auf
// aktuellem Feld
void nimmAlle( )
{
   while (kornDa( ))
   {
         nimm( );
   }
}
```

Beispiel 2:



Unser Hamster soll bis zur Mauer gehen, ohne gegen sie zu stoßen. Schreibe eine Prozedur *vorBisMauer( )*.

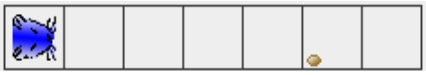
umgangssprachlich

Solange das Feld vor dem Hamster frei ist, gehe einen Schritt vor.

Prozedur *vorBisMauer*

```
// Hamster geht bis Feld vor, welches
// sich unmittelbar vor einer Mauer be-
// findet
void vorBisMauer( )
{
   while (vornFrei( ))
   {
         vor( );
   }
}
```

Beispiel 3:



Unser Hamster soll auf das Feld mit dem Korn laufen. Schreibe eine Prozedur *vorBisKorn( )*.

umgangssprachlich

Solange sich kein Korn auf dem aktuellem Feld befindet, gehe einen Schritt vor.

Prozedur *vorBisKorn*

```
// Hamster geht auf das nächste Feld,
// auf welchem sich mindestens ein Korn
// befindet.
void vorBisKorn( )
{
   while (!kornDa( ))
   {
         vor( );
   }
}
```

Beispiel 4:

Was geschieht, wenn auf dem Territorium

das Programm

```
void main ( )
{
   vor( );
   while (kornDa( ))
   {
         nimm( );
         vor( );
   }
}
```

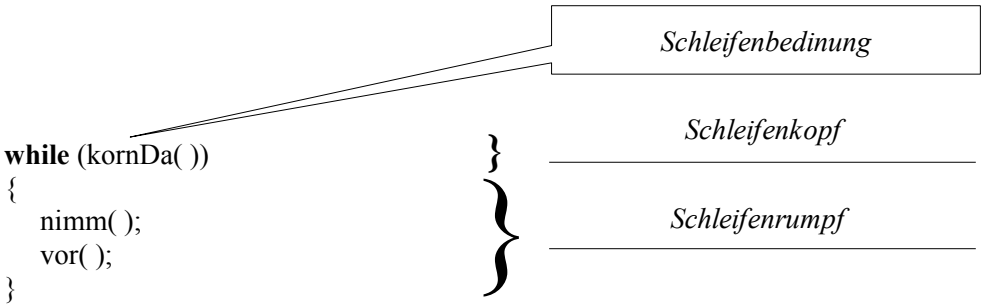
ausgeführt wird?

Antwort

Hamster geht einen Schritt vor, nimmt von den nächsten zwei Feldern jeweils ein Korn und verharrt auf dem zweiten Feld von rechts.

Musterlösung AB 3: WHILE-Schleife ITG, 8

Begriffe:



Schleifenbedingungen:

Schleifenbedingung	***Bedeutung***
kornDa( )	auf aktuellem Feld befindet sich mindestens ein Korn
!kornDa( )	auf aktuellem Feld befindet sich kein Korn
vornFrei( )	auf Feld vor Hamster befindet sich weder eine Mauer, noch das Ende des Territoriums
!vornFrei( )	auf Feld vor Hamster befindet sich eine Mauer oder das Ende des Territoriums
maulLeer( )	Im Maul des Hamsters befinden sich keine Körner.
!maulLeer( )	Im Maul des Hamsters befindet sich mindestens ein Korn.

Funktionsweise der WHILE-Schleife:

Die WHILE-Schleife ist eine sogenannte *Wiederholungsanweisung*.
Bei ihrer Ausführung wird im Schleifenkopf zunächst überprüft, ob die Schleifenbedingung erfüllt ist. Falls dies nicht der Fall ist, wird der auf die WHILE-Schleife folgende Block {...} übersprungen, und die Schleife ist unmittelbar beendet.

Falls die Bedingung erfüllt ist, wird der Schleifenrumpf *genau ein Mal* ausgeführt. Anschließend wird die Schleifenbedingung erneut ausgewertet. Falls sie immer noch erfüllt ist, wird der Schleifenrumpf ein weiteres Mal ausgeführt. Dieser Prozess wiederholt sich, *solange* die Schleifenbedingung erfüllt ist.

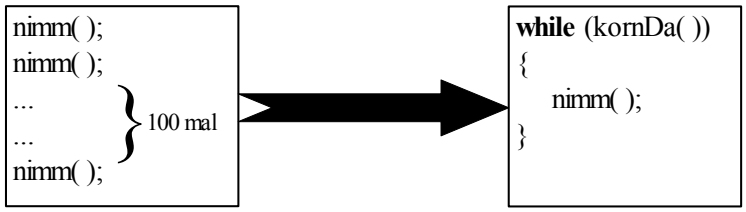
Beachte:

- Der Schleifenkopf wird nicht mit einem Semikolon abgeschlossen.
- Die Befehle im Schleifenrumpf rücken wir um 4 Leerzeichen bzw. einen Tabulator ein.

Einstieg in die Programmierung Seite 3 von 4

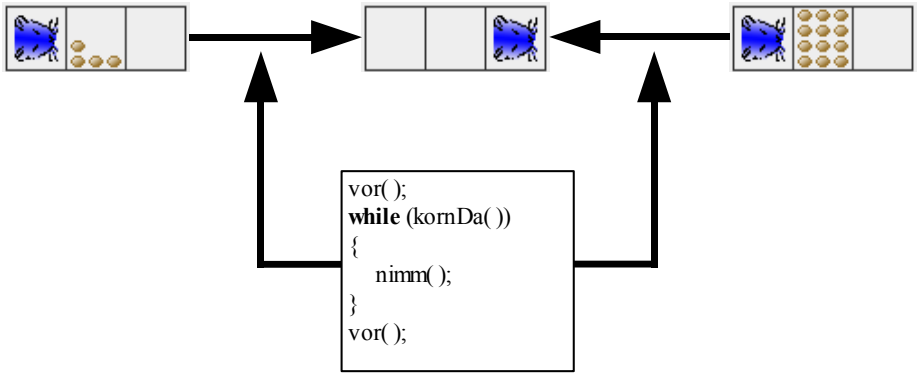
Vorteile & Notwendigkeit

1.



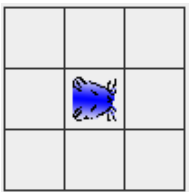
Vorteil, da man hier auch ohne WHILE–Schleife auskommen würde, weil die Anzahl der nimm( )*–Anweisungen im Voraus bekannt ist*

2.



Notwendigkeit, da die Anzahl der Körner variieren kann und man somit im Voraus nicht weiß, wie oft nimm( ) *ausgeführt werden soll*

Endlosschleife

Was geschieht, wenn auf dem Territorium  das Programm

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

ausgeführt wird?

Antwort

*Da das Feld vor dem Hamster immer frei ist, dreht sich dieser endlos im Kreis (→*Endlosschleife*).*

Arbeitsblatt 4
Programmierfehler

	Syntaxfehler	**Semantikfehler**
Beispiele	*vor( )* statt *vor( );* *linksUm;* statt *linksUm( );* *rechtsUm( );* *falls die Prozedur rechtsUm überhaupt nicht definiert ist*	 *vor( );* *vor( );* ***Fehler****, weil Hamster gegen Mauer stößt*
Beschreibung	*Verstoß gegen die* Syntax *(Rechtschreibe- und Grammatikregeln) der Programmiersprache*	*Alle Arten von Fehlern, die während der Programmabarbeitung auftreten*
Erkennung durch Compiler	*Syntaxfehler werden vom Compiler frühzeitig (bei der Übersetzung) erkannt, also vor Programmstart.*	*Semantikfehler werden vom Compiler* nicht *erkannt. Sie sind demnach schwerwiegender, da sie Programmabstürze verursachen können*

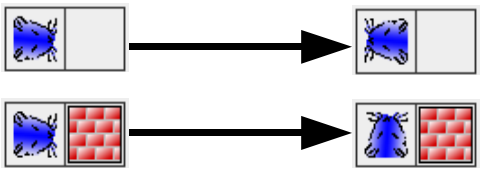
Arbeitsblatt 5

Die IF-Anweisung

als Beispiel für eine Auswahlanweisung

Beispiel 1:

vorher nachher



umgangssprachlich, Variante 1

Falls das Feld vor dem Hamster frei ist, drehe zwei Mal links, andernfalls ein Mal links.

Programmfragment, Variante 1

```
if (vornFrei())
{
   linksUm();
   linksUm();
}
else
{
   linksUm();
}
```

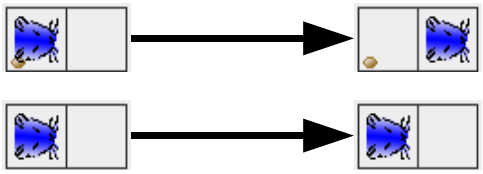
umgangssprachlich, Variante 2

Falls das Feld vor dem Hamster nicht frei ist, drehe ein Mal links, ansonst zwei Mal links.

Programmfragment, Variante 2

```
if (!vornFrei())
{
   linksUm();
}
else
{
   linksUm();
   linksUm();
}
```

Beispiel 2:



umgangssprachlich, Variante 1

Falls sich Korn auf dem aktuellen Feld befindet, gehe vor, andernfalls tue gar nichts.

Programmfragment, Variante 1

```
if (kornDa())
{
   vor();
}
else
{
}
```

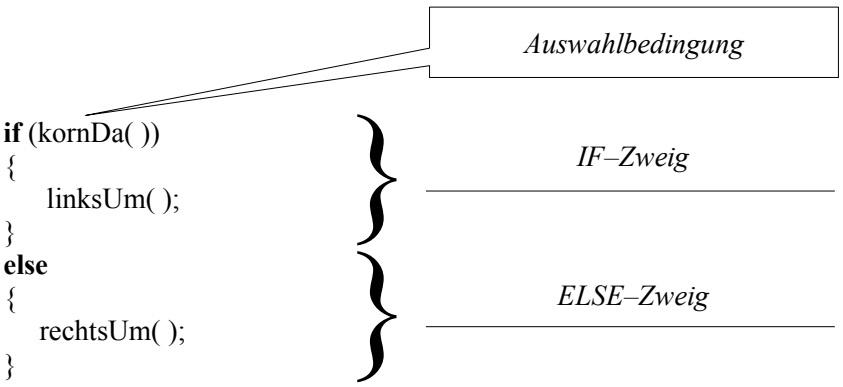
umgangssprachlich, Variante 2

Falls sich kein Korn auf dem aktuellen Feld befindet, tue gar nichts, andern–falls gehe vor.

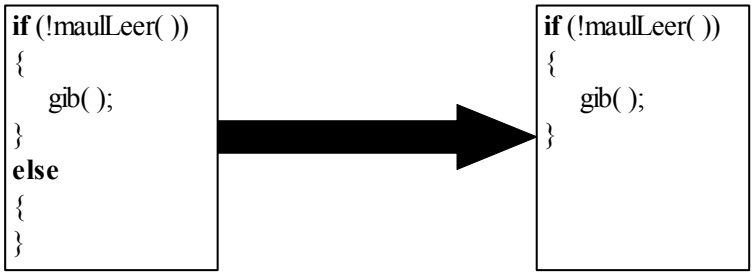
Programmfragment, Variante 2

```
if (!kornDa())
{
}
else
{
   vor();
}
```

Begriffe:



Funktionsweise:
Die IF-Anweisung ist eine *Auswahlanweisung*.
Trifft die *Auswahlbedingung* zu, so werden die Anweisungen des *IF-Zweigs* ausgeführt. Andernfalls werden die Anweisungen des *ELSE-Zweigs* ausgeführt.
Einen leeren ELSE-Zweig kann man auch auslassen, z.B.



Den IF-Zweig kann man jedoch niemals auslassen, da in ihm die Auswahlbedingung steht. Es ist jedoch möglich, im IF-Zweig einen leeren Block anzugeben (siehe Beispiel 2).

Beachte:

- Die Zeilen mit IF und ELSE werden nicht mit Semikola abgeschlossen.
- Sind IF- und ELSE-Zweig vorhanden, wird immer entweder der IF-Zweig oder der ELSE-Zweig ausgeführt.
- Ist nur der IF-Zweig vorhanden, wird dieser entweder ausgeführt, oder er wird nicht ausgeführt.

Vorteil oder Notwendigkeit?

Die Auswahlanweisung (hier IF–Anweisung) ist absolut notwendig, da man ohne sie keine Entscheidungen in einem Programm formulieren könnte. Die Probleme aus Beispiel 1 und 2 wären ohne eine Auswahlanweisung nicht lösbar!

Musterlösung AB 6: Struktogramme ITG, 8

Sinn & Zweck von Struktogrammen:

zur alternativen Formulierung von Algorithmen

zur übersichtlichen Darstellung von Algorithmen

sind sehr leicht in ein Programm überführbar

Dokumentation von Programmen

beschreibt zuerst die Struktur der Lösung, ohne dass man mit programmiersprach–lichen Details konfrontiert wird

Programmcode für das Beispielstruktogramm:

```
void main ( )
{
   while (!vornFrei( ))
   {
        linksUm( );
   }
   while (vornFrei( ))
   {
        while (kornDa( ))
        {
           nimm( );
        }
        vor( );
   }
   rechtsUm( );
   vor( );
```

```
   while (vornFrei( ))
   {
      if (maulLeer( ))
      {
      }
      else
      {
         gib( );
      }
      vor( );
   }
}

// Hamster dreht sich 90° nach rechts
void rechtsUm( )
{
   linksUm( );
   linksUm( );
   linksUm( );
}
```

Glossar

Algorithmus	*genau festgelegte Handlungsvorschrift zur Lösung eines Problems oder einer Klasse von Problemen*
Programm	*Folge von Befehlen, die in einer →Programmiersprache formuliert werden*
Programmiersprache	*genormte, standadisierte Sprache zur computernahen Formulierung von →Algorithmen*
Programmierer	*Person, die →Programm(e) schreibt*
Compiler	*übersetzt →Programm in →Binärcode und überprüft dabei die →Syntax der Programmzeilen*
Binärcode	*Darstellung eines →Programms, die der Prozessor unmittelbar „versteht“ (Nullen und Einsen)*
Maschinencode	*→ Binärcode*

Musterlösung Glossar ITG, 8

Endlosschleife	*Schleife, deren Schleifenbedingung immer zutrifft und somit endlos durchlaufen wird*
Struktogramm	*graphische Darstellung von Lösungswegen und –verfahren, alternative Möglichkeit zur Formulierung von →Algorithmen*
(Syntax)	*„Grammatik– und Rechtschreiberegeln“ einer →Programmiersprache*

Weiterführende
Literatur & Links

Dietrich Boles: Programmieren spielend gelernt. *Teubner–Verlag, 3. Auflage, 2006*

http://www.java–hamster–modell.de, Offizielle Java–Hamster–Website

http://www.u–helmich.de/inf, Online–Programmierkurs für den Java–Hamster