FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Transfer des LEGO MINDSTORMS EV3-Programmierwissen in Java

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Master of Science

im Rahmen des Studiums

Informatikdidaktik

eingereicht von

Bettina Gruber
Matrikelnummer 1128537

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung
Betreuerin: Ass. Prof. Dipl.-Ing. Dr. Monika Di Angelo

Wien, 26.07.2014

(Unterschrift Verfasser/in)

(Unterschrift Betreuer/in)

Eidesstattliche Erklärung

Bettina Gruber, BSc
Rotenmühlgasse 34/10
1120 Wien

Hiermit erkläre ich, dass ich diese Arbeit selbstständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 26.07.2014

…….....

(Unterschrift)

Kurzfassung

LEGO MINDSTORMS bietet mit seiner grafischen Programmieroberfläche einen einfachen Einstieg in die Welt des Programmierens. Für komplexere Beispiele ist die grafische LEGO-Oberfläche jedoch nicht geeignet, daher wird der Umstieg auf eine textuelle Programmiersprache angestrebt, um auch aufwändigere Beispiele lösen zu können.

In dieser Arbeit wird untersucht, wie der Wissenstransfer von der grafischen Oberfläche in die textuelle Programmierung bestmöglich stattfinden kann. Auch die Geschichte der Programmierlehre und der derzeitige Einsatz didaktischer Vorgehensweisen in der Programmierlehre sind Teil dieser Arbeit. Es wird näher auf die Unterschiede zwischen Object-First und Object-Later eingegangen, wobei zusätzlich Guidelines für die objektorientierte Programmierlehre und dabei auftretende Problemfälle aufgezählt werden.

Relevante Bücher werden hinsichtlich ihrer didaktischen Konzepte und der Brauchbarkeit bezüglich des Wissenstransfers in die textuelle Programmiersprache Java bewertet und analysiert. Doch trotz aller Vielfalt an existierenden Java-Büchern sind keine Bücher erhältlich, die auf dem bereits vorhandenen Wissen aus der visuellen Programmierung von LEGO aufbauen. Es fehlt eine didaktische Unterstützung beim Übergang von der grafischen LEGO-Programmierumgebung zur textuellen Programmierung in Java. Das Ergebnis dieser Arbeit stellt ein didaktisches Konzept für ein Buch dar, welches das bereits vorhandene Wissen aus der LEGO MINDSTORMS Programmierung mit der grafischen Oberfläche nutzt, um EinsteigerInnen das Programmieren mit Java näher zu bringen.

Abstract

LEGO MINDSTORMS, with its graphical programming, represents a simple introduction to the world of programming for beginners. Some young people have already learned to program with the graphical interface of LEGO MINDSTORMS and know basic programming concepts.

For more complex tasks LEGO's graphical interface does not offer enough performance; so the next step for young people interested, is to use textual programming to solve such elaborate examples. In this paper it is investigated how the popular Java programming language is taught today.

Relevant books are assessed for their didactic concepts and analysed if they are suitable to transfer graphical programming knowledge to Java. A transition from the graphical programming environment to LEGO-textual programming in Java is missing in the existing literature. There is no didactic support provided to change from graphical programming enviroment to textual programming. In this thesis a didactic concept for a book is created, which utilizes the existing knowledge from programming with the LEGO MINDSTORMS graphical user interface to support beginners to learn Java programming.

Inhaltsverzeichnis

1. Einleitung

Derzeit gibt es viele verschiedene Wege und Tools, um in die Programmierung einzusteigen, einige davon erfüllen ihren Zweck gut, andere weniger. Eine Methode stellt die Arbeit mit den LEGO-MINDSTORMS-Robotern dar. Diese können sowohl mit der grafischen Programmierumgebung von LEGO als auch mit einer textuellen Programmiersprache programmiert werden. Viele Bücher und Arbeiten sind bereits erschienen, die die Programmierung der LEGO-Roboter behandeln (z.B. [WAGN05]), andere untersuchen deren Verwendung dieser im Informatikunterricht (z.B. [Dian11], [Lawh08]), oder sie handeln über das Lehren von objektorientierten Konzepten ([Löwe09]). Auch der Einsatz dieser Roboter in Bereichen außerhalb der Programmierung, wie z.B. in der Physik, wurde bereits von [Schm09] betrachtet.

1.1 Problemstellung

„Der Einsatz einer künstlichen, auf die Lernsituation ausgerichteten Programmiersprache wirft natürlich früher oder später die Frage auf, wie Schüler in einem weiterführenden Kurs den Übergang zu einer professionellen Programmiersprache und Programmierumgebung schaffen.“ [Reic05] Genau dieselbe Frage ergibt sich, wenn Jugendliche bereits viel Erfahrung und Wissen in der grafischen Programmierung der LEGO-Roboter gesammelt haben. Sie können dann durchaus schon größere Robotersysteme erschaffen, sind aber dennoch aufgrund der Grenzen der grafischen Oberfläche in ihrer Schaffensfreiheit beschränkt. Weitaus komplexere Roboter können sie mit einer textuellen Programmierung entwickeln, was den meisten Jugendlichen durch Internet und soziale Medien über Videos auch klar wird. Deshalb stellt sich dann die Frage, wie dieses erworbene Wissen aus einer grafischen Programmierumgebung bestmöglich in eine textuelle Programmiersprache transferiert werden kann. Die vorliegende Arbeit erstellt ein Konzept für ein Buch, das Jugendliche beim Wissenstransfer in eine textuelle Programmiersprache wie Java unterstützen soll.

1.2 Aufbau der Arbeit

In Kapitel 2 wird zu Beginn dieser Arbeit die Programmierlehre genauer unter die Lupe genommen. Es wird auf die geschichtliche Entwicklung, auf unterschiedliche Vorgehensweisen in der Programmierlehre, auf LEGO-Roboter in der Lehre und auf die grafische und objektorientierte Programmierung eingegangen. Auch die Frage, warum Java verwendet wird, wird erörtert.

Nach der Erörterung der Frage, wie die Programmierlehre in der Gegenwart und in der Vergangenheit aussah, folgt in Kapitel 3 der analytische Teil. Es werden auf dem Markt vorhandene Bücher zum Einstieg in die Java-Programmierung (Kapitel 3.1), zum Erlernen von Java mittels anderer Tools (Kapitel 3.2) oder zur Programmierung der LEGO-Roboter (Kapitel 3.3) hinsichtlich ihrer Konzepte und ihrer Methoden zum Erlernen der objektorientierten Sprache Java analysiert. Daraus wird hervorgehen, ob diese Bücher für einen Transfer des konzeptuellen Vorwissens aus der grafischen Oberfläche geeignet sind oder eher weniger.

Kapitel 4 beschäftigt sich mit dem Wissenstransfer der Jugendlichen und damit, welches Vorwissen aus der Programmierumgebung bereits vorhanden sein sollte, um dieses aus dem Konzept entstehende Buch lesen zu können.

Neben der Bücheranalyse bildet das didaktische Konzept in Kapitel 5 einen wichtigen Punkt in dieser Arbeit. Dabei wird sowohl über die anzustrebenden Lernziele und Kompetenzen der Jugendlichen, das Medium, die Zielgruppe, aber auch die verwendete Methode (Object-First oder prozedurale Programmierung vor OOP) und die Art der einzusetzenden Beispiele sowie über das Inhaltsverzeichnis des entstehenden Buchs geschrieben. Wie der Inhalt präsentiert werden soll, was die Schwierigkeiten für Programmiereinsteiger sind und wie diese bestmöglich aufgehoben werden, ist ebenfalls Teil dieses Abschnitts.

Zu guter Letzt beinhaltet die vorliegende Arbeit eine Zusammenfassung und einen Ausblick auf die Thematik für die nahe Zukunft.

2. Programmierlehre

Das geplante Buch soll die Objektorientierung und Java-Programmierung oberflächlich und nicht bis ins Detail erklären, denn es geht darum, dass die Lernenden einen leichten Einstieg und eine gute Umstiegsmöglichkeit vom grafischen Programmieren der LEGO-Roboter zur textuellen Java-Programmierung erhalten.

In den Schulen ist oft zu wenig Zeit für den Informatikunterricht, Programmieren wird, wenn überhaupt, nur sehr wenig gelehrt, weil primär Anwendungssoftware behandelt wird. Viele Kinder und Jugendliche interessieren sich aber dafür und wollen lernen, wie man Programme ohne die grafische Oberfläche von LEGO schreiben kann. Die vorhandene Literatur in diesem Bereich bietet nur geringe Möglichkeiten für Kinder und Jugendliche, sich auf diesem Gebiet selbst weiterzubilden, also den Sprung von der grafischen Programmieroberfläche zu einer textuellen zu schaffen. Auch Online-Tutorials sind dafür zum heutigen Zeitpunkt noch keine bekannt. Kurse für Programmiereinsteiger finden meist nur für Erwachsene statt, auch wenn es mittlerweile einige Vereine und Organisationen gibt, die entsprechende Kurse anbieten: so z.B. das Institut zur Förderung des IT-Nachwuchs[1], welches seit 2011 Roboterkurse mit der grafischen Programmieroberfläche von LEGO veranstaltet und das Programm auf Java-Einsteigerkurse speziell mit LEGO-Robotern erweitert hat.

„Robot-based curricula is used today across diverse age groups and with a broad variety of purposes." [Nour05] Auch wenn Roboter bereits vielfach von unterschiedlichen Altersgruppen und zu den unterschiedlichsten Zwecken, nicht nur für das Programmieren lernen, verwendet werden, so gibt es nur eine geringe Anzahl an Büchern, die es den Jugendlichen ermöglichen, mit den Robotern Java als erste textuelle Programmiersprache zu lernen.

„... eine sinnvolle Nutzung komplexer Systeme und Sprachen meist eine fundierte Grundbildung an einfacheren Systemen erfordert. Um also einmal große Informatiksysteme schaffen zu können, sollte der Schüler das Programmieren im Kleinen beherrschen." [Will10] Diese geforderte Grundbildung in einfachen Systemen kann mittels der grafischen Programmieroberfläche von LEGO MINDSTORMS EV3[2] oder der

1 IFIT: https://www.facebook.com/ifit.org
2 http://www.lego.com/en-us/MINDSTORMS/downloads/software/ddsoftwaredownload/

Vorgängerversion NXT 2.0 geschaffen werden. Wichtig ist beim Übergang zu Java, dass geeignete Beispiele gewählt werden, um die Motivation zu erhalten und nicht durch die Komplexität der Beispiele die AnfängerInnen zu ermüden. Genau das können die Roboter bewirken, wie schon in vielen anderen Arbeiten, z.B. in [Löwe09], zu diesem Thema erläutert wurde.

„We must have programming, in one form or another, or we will have cut off half the power of the medium – like reading without writing.“ [Solo93]
„The fundamental fact is that 'provided resources' never do all and exactly what you need. We must give ordinary folks true combinability and modifiability.“ [Solo93] zeigt, wie wichtig es ist, dass die Bevölkerung Programmieren in der einen oder anderen Form kann, damit man die vollen Möglichkeiten des technischen Mediums Computer ausnutzen kann. Da Computer heutzutage zum Alltag gehören, wie es früher nicht einmal ansatzweise absehbar war, sollte es heutzutage möglich sein, dass jeder programmieren lernen kann, der es gerne möchte. Die Notwendigkeit der Programmierlehre ist also weit über die Grenzen der grafischen Programmierung und des konzeptuellen Verständnisses hinaus gegeben.

Da für die jüngere Bevölkerung die Notwendigkeit des Programmierenlernens durchaus gegeben ist, wird im Folgenden gezeigt, warum es sinnvoll ist, mit den LEGO-MINDSTORMS-Robotern in Java einzusteigen.
„Es muss auch gezeigt werden können, dass Programmieren eine anspruchsvolle Arbeit darstellt.“ [Will10] Auf die anspruchsvollen Bereiche des Programmierens kommen die Jugendlichen mit der Zeit von selbst. Warum soll man Anfängern auch sagen, dass es einmal schwerer werden wird, solange es derzeit noch Spaß macht und einfach ist. Der Vergleich aus einem anderen Bereich soll zeigen, wie unnötig es ist, Anfängern von den Schwierigkeiten des Faches schon zu Beginn zu erzählen. Ein/Eine SchilehrerIn, der/die ihren neuen Schülern die ersten paar Stunden unterstützt und ihnen Schifahren beibringt, wird diesen auch nicht sagen, dass es schwerer wird, wenn man ein Rennen fahren will, dass es dann eisig werden kann und der Hang viel steiler sein kann. Der/die SchilehrerIn wird sie Schritt für Schritt an die Aufgabe des Schifahrens heranführen. Dass es einmal nicht mehr so leicht ist, wenn keine Unterstützung dabei ist, dass es schwieriger wird, wenn der Hang steiler wird, die Pistenbedingungen sich ändern usw., werden die Neulinge später eigenständig her-

ausfinden. Womöglich kommen nicht alle Schüler so weit und werden ihr Leben lang nur auf flachem Gelände gemütlich die Piste herunterfahren, dennoch werden sie doch immer Spaß daran haben und froh sein, dass sie es gelernt haben. Wäre ihnen von dem/der SchilehrerIn gleich zu Beginn gesagt worden, dass es schwer werden wird, so hätten viele vielleicht gleich damals damit aufgehört und gar nie Schifahren gelernt und wären nie so weit gekommen.

Demnach sollte auch der Einstieg ins Programmieren möglichst einfach gewählt werden, wie es bei den Robotern der Fall ist. Denn bei der Programmierung der LEGO-Roboter verhält es sich ähnlich wie im erklärten Schifahreranfänger-Beispiel.

2.1 Geschichtliche Entwicklung der Programmierlehre

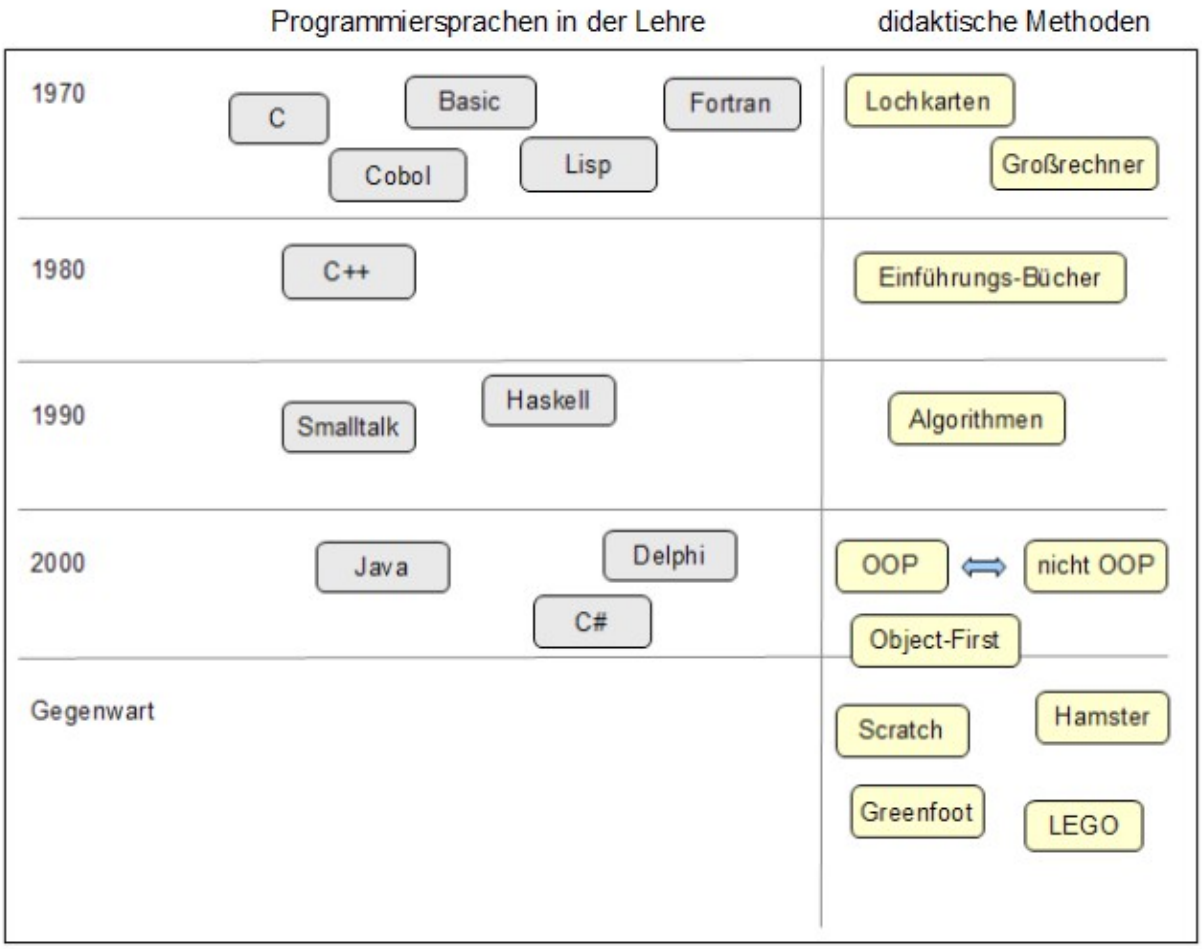


Abb. 1: Geschichte der Programmierlehre

Die Informatik ist eine vergleichsweise junge Wissenschaft und nahm ihren Anfang

vor nicht einmal 100 Jahren. Jahrzehnte später machten sich WissenschaftlerInnen über die Didaktik der Informatik bzw. die Programmierlehre Gedanken und forschten in diese Richtung. Erst vor einem Jahrzehnt kam es immer mehr dazu, dass mit der Objektorientierung begonnen wurde und die imperativen Konzepte in den Hintergrund rückten. Object-First [Jäge07] wird der Ansatz genannt, mit der objektorientierten Denkweise oder einer objektorientierten Sprache einzusteigen.

Vor ungefähr 40 Jahren wurde begonnen, Programmiersprachen in unterstützender Weise in anderen Wissenschaften einzusetzen. Damals waren es Sprachen wie C, Cobol, Basic, Fortran und Lisp, die verwendet wurden. Um 1970 herum wurde noch Assembler auf Großrechnern unterrichtet, auch Lochkarten kamen zum Einsatz.

Annähernd vor 30 Jahren kamen immer mehr Einführungsbücher über Informatik und Programmierung auf den Markt: damals noch zu den zuvor erwähnten Sprachen, doch auch C++ wurde zu dieser Zeit das erste Mal gezeigt.

Zu neuen, alternativen Programmiersprachen kam es vor 20 Jahren, wo an Universitäten vermehrt funktionale Programmiersprachen wie Haskell eingesetzt wurden. Auch Smalltalk, eine objektorientierte Programmiersprache, wurde zu dieser Zeit des Öfteren verwendet. Die Praxisrelevanz an OOP war zu dieser Zeit aber noch nicht gegeben, weshalb sich Smalltalk nicht durchsetzen konnte. Zunehmend wurden theoretische Konzepte wie Algorithmen und grundlegende Funktionalitäten des Computers gelehrt.

Java wurde vor rund 10 Jahren gang und gäbe, in Vorlesungen wurde jedoch zuvor primär imperativ unterrichtet, erst ein paar Jahre später kam der Object-First-Ansatz in Verwendung. Auch Softwareentwicklungsmodelle wie Scrum oder Extreme Programming entwickelten sich zu dieser Zeit. [Edel13]

Heute gibt es mittlerweile bereits wieder Gegner des Object-First-Weges, wie in Kapitel 5.5 zu lesen ist, und die unterschiedlichsten Tools ersetzen die Verwendung komplizierter Entwicklungsumgebungen.

2.2 Grafische Programmierung

Betrachtet man mechanische Geräte, so ist die Funktion dahinter leicht erkennbar. Anders sieht es da schon bei Anwendungen und Programmen aus, dort ist es oft viel schwieriger und nicht direkt zu erkennen, welche Funktion sie erfüllen. Die visuelle Programmierung soll eine intuitive und mühelose Softwareentwicklung ermöglichen und mit dieser Art der Programmierung die Verständlichkeit von Konzepten ungemein erleichtern.

„Staufer [87 S. 52 ff] nennt Experimente, die zeigen, daß Bilder schneller verarbeitet werden als Text und einen weitaus tieferen Eindruck hinterlassen als verbale Ausführungen.“ [Schif01] Bilder sind für AnfängerInnen gut geeignet, um schneller zu verstehen und – schneller als mit einer textuellen Programmiersprache – die gesuchten Funktionen zu finden. Außerdem kann sich bei einer visuellen Programmierung der/die ProgrammiererIn durch das Wegfallen der Syntax viel mehr auf die Semantik konzentrieren.

AnfängerInnen, die zuerst in der grafischen Oberfläche von LEGO programmiert haben und erst danach mit der textuellen Programmierung beginnen, haben auch einen Geschwindigkeitsvorteil gegenüber denjenigen, die direkt in die textuelle Programmierung einsteigen. Denn diese müssen sich nicht mit der Syntax und dem logischen Denken gleichzeitig beschäftigen. Da sie schon das logische Verständnis aufgebaut haben, verringert sich die Komplexität des Programmierenlernens, da nur noch die Syntax als Hindernis zu meistern ist.

Folgende weitere Vorteile einer grafischen Programmierung nennt Schiffer in seinem Buch: „Dadurch wären Zusammenhänge schneller erfaßbar, Konzepte besser erlernbar und Fehler eher vermeidbar, kurz gesagt, die Umsetzung einer Aufgabenstellung in ein Programm würde bedeutend erleichtert.“ [Schif01]

Viele Neulinge sind durch den Einstieg in eine textuelle Programmiersprache überfordert. Hätten sie zuvor in einer visuellen Programmierumgebung Erfahrungen gesammelt, so würde es wahrscheinlich deutlich mehr ProgrammiererInnen geben und somit weniger Interessierte, die überfordert sind. [Hund06] schreibt, dass in seinen Pro-

grammier-Einführungskursen eine durchschnittliche Ausfallrate von 43% gegeben ist, und stellt sich dann die Frage, warum so viele am Programmieren scheitern. Bisherige Forschungen konnten feststellen, dass es eine Vielzahl möglicher Faktoren für diese hohe Durchfallquote gibt. Die relevantesten Faktoren sind wohl: „(a) individual student differences, (b) a lack of a sense of community, (c) deficient pedagogical approaches, and (d) inadequate novice programming environments.“ [Hund06] Es konnte in [Hund06] auch gezeigt werden, dass nicht nur die ersten Hemmschwellen mit einer grafischen, also einer direkt manipulativen Programmierumgebung reduziert werden, auch der positive Transfer in eine textuelle Programmierung ist vorhanden. Jedoch können schon vermeintlich kleine Änderungen an der Software die Wirksamkeit des ganzen Systems stark beeinflussen.

2.3 Weshalb mit LEGO MINDSTORMS beginnen?

2.3.1 Motivation

Roboter üben schon seit jeher eine große Faszination auf Menschen aus. Der Gedanke daran, dass diese Roboter uns Arbeit abnehmen können, die für uns unangenehm, gefährlich oder einfach nur schwer ist, gefällt vielen. Dazu kommt, dass Roboter immer genau das tun, was ihnen programmiert wurde – und das immer gleich schnell.

Neben dieser Faszination für Roboter trägt auch Selbstbestimmung beim Lernen viel zur Motivation bei. Diese soll durch fantasievolle selbstgebaute Roboter gegeben sein. Laut [WAGN05] bietet LEGO mit den MINDSTORMS-Robotern eine Möglichkeit für den altersgerechten Einstieg in die Programmierung. Welche Altersgruppe gut geeignet ist für den Einstieg in die textuelle Programmierung, wird später in der Arbeit im Kapitel 5.3 dargelegt.

Neben Programmieren wird bei der Verwendung von LEGO MINDSTORMS noch weitaus mehr erlernt: Bauen, Designen, der ganze Prozess muss überlegt werden. Testen ist ebenfalls sehr wichtig – was sonst oft vergessen wird – hier jedoch auto-

matisch und gerne durchgeführt wird. Das Ziel ist ständig vor Augen, denn der Roboter muss zum Schluss eine bestimmte Aufgabe ausführen.

Es wurden bisher aber nicht nur positive Erfahrungen mit den LEGO-Robotern gemacht, denn beispielsweise [Fagi03] berichtet von negativen Ergebnissen aus seiner Studie, an der über 800 Studenten teilnahmen. Als Software zur Programmierung der Roboter wurde bei dieser Studie Ada/MINDSTORMS 2.0 verwendet, hierbei handelt es sich um eine textuelle Programmierumgebung. Er teilte seine Studenten in zwei Gruppen auf, die eine lernte Programmieren mit gewöhnlichen Beispielen ohne Roboter, und die andere hatte Roboter zur Verfügung. Es stellte sich heraus, dass bei einem gleichzeitigen Abschlusstest beide Gruppen gleich gut abschnitten hatten. Es konnten also keine besseren Ergebnisse bei der Robotergruppe festgestellt werden. Die Analyse stützte sich nur auf die Prüfungen – und nicht auf die einzelnen Aufgaben, die während der Kurse gemacht wurden. Es wurde nur ein einziger Jahrgang überprüft; außerdem ist zu bedenken, dass die „Roboterstudenten“ nicht zuhause damit üben konnten. Es herrschten also nicht gleiche Bedingungen für beide Studentengruppen vor.

Wenn es sich um Studenten handelt, die eine Fachrichtung der Informatik studieren, so sind Roboter eher hinderlich, weil sie eine zusätzliche Herausforderung in der Beschaffung darstellen und nicht ortsunabhängig geübt werden kann, da die Roboter meist nur im Universitätsgebäude vorhanden sind. Meiner Erfahrung nach ist bei diesen Studenten meist genügend Motivation vorhanden sich das Wissen aus diesem Bereich anzueignen, denn ansonsten hätten sie diese Studienrichtung nicht gewählt. Von daher sind Roboter zur Förderung der Motivation hier weniger von Belang als in anderen Bereichen.

Durch die haptische Komponente regen Roboter einen großen Teil des Gehirns zu vermehrter Denkleistung an, was auch im Artikel „Lernen macht glücklich“ verdeutlicht wird: „Lernt ein Proband seine Seh- und Motorikfähigkeit gleichzeitig zu aktivieren, dann arbeiten beim Denken Zweidrittel seines Gehirns.“ [@wiss]

Roboter agieren in der realen Welt, weshalb sie nicht nur mit eingeschränkten oder ausgewählten Benutzerinteraktionen funktionieren sollen; stattdessen müssen sie

alle möglichen Situationen, die auftreten könnten, betrachten. Auftretende Situationen können sehr oft für die Lernenden unvorhergesehen und überraschend sein. [Barn02] Dies bringt ProgrammieranfängerInnen dazu, gleich von Beginn an ein Programm für das Worst-Case-Szenario zu entwickeln.

2.3.2 LEGO MINDSTORMS EV3-G

Der EV3 ist die Weiterentwicklung des bekannten NXT-Robotersystems von LEGO. Der EV3 beinhaltet folgende Komponenten[3]:

- 1 mittlerer Motor
- 2 große Motoren
- 1 Berührungssensor
- 1 Infrarotsensor
- 1 Farbsensor
- Das Herzstück dieses Produkts ist der intelligente EV3-Stein mit leistungsstarkem ARM9-Prozessor, USB-Port für WiFi und Internetverbindung, Micro-SD-Kartenleser, beleuchteten Tasten und 4 Motoranschlüssen.

Für den EV3 gibt es eine neue Programmieroberfläche (EV3-G), worauf im Kapitel „Vorwissen“ noch genauer eingegangen wird. Es sei an dieser Stelle erwähnt, dass sich die Software in folgende Teilbereiche unterteilt: die Lobby (zu sehen in Abbildung 2) mit den Bauanleitungen, Einsteigervideos und Neuigkeiten; dann gibt es den Bereich für das Projekt, wo einerseits Dokumentation einfach und schön gestaltbar ist, andererseits aber auch die eigentliche Programmierung des Roboters stattfindet.

3 http://www.lego.com/en-us/MINDSTORMS/gettingstarted/whatsinthebox/

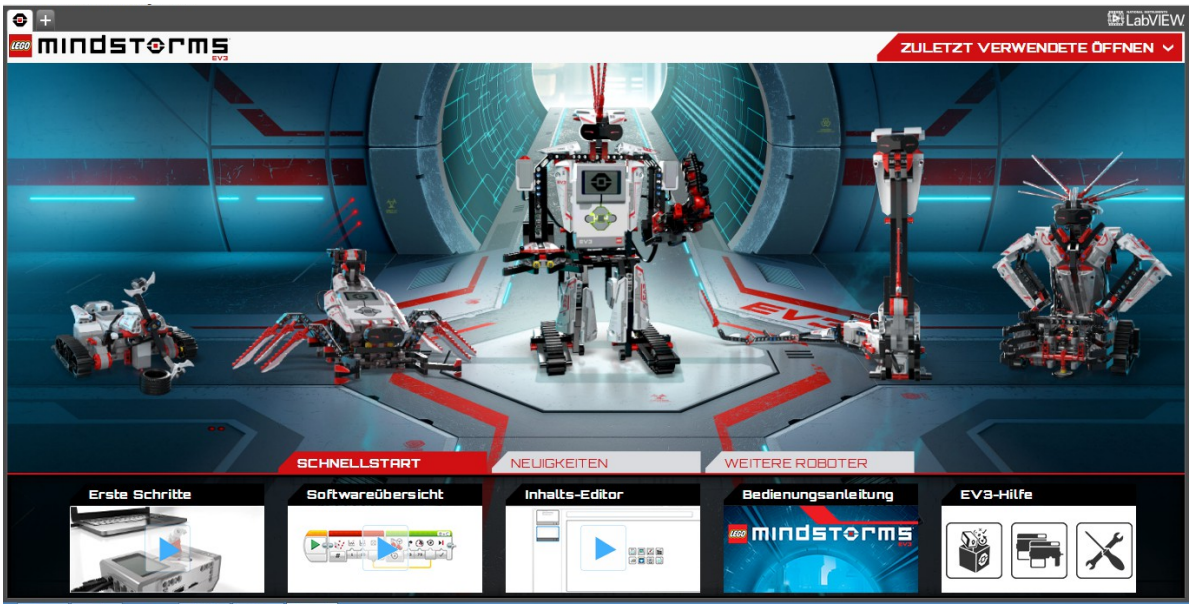


Abb. 2: EV3-G Programmstartseite

2.3.3 Unterschiede EV3 und NXT[4]

All jene Lernenden, die bereits Programmiererfahrung und Know-how in der grafischen Programmieroberfläche des NXT-Roboters gemacht haben, können genauso wie jene, die mit dem EV3-Roboter gearbeitet haben, damit beginnen, Java zu erlernen. Es sind jedoch ein paar Unterschiede sowohl an den Bauteilen als auch an der Programmierumgebung zu nennen: Das Herzstück des EV3 besitzt 4 Motoranschlüsse, das des NXT hingegen nur 3. Jedoch sind auch im EV3 nur 3 Servomotoren vorhanden. Anstatt eines Ultraschallsensors gibt es im EV3 einen Infrarotsensor. Der Farbsensor ist geblieben, er wurde nur verbessert und neu gestaltet.

In der Programmierumgebung ist bei EV3 die Möglichkeit zur Dokumentation enthalten. Es können Bauanleitung, Dokumentation, Programm und Video gesammelt gespeichert werden. Hingegen war bei NXT nur die Möglichkeit zur Programmierung gegeben. Diese neuen Features machen das Programm zwar praktischer, aber auch etwas unübersichtlicher, was jedoch keinen Unterschied hinsichtlich der Programmierung der Roboter in Java macht. Praktisch ist es lediglich, wenn die Lernenden alle notwendigen Unterlagen gesammelt haben – und nicht wie bei NXT die Bauanleitung und das Programm extra suchen müssen.

4 Bauanleitung NXT 2.0: http://www.northsideprep.org/ncphs/depts/cs/dyanek/ect/LEGONXT/assets/languages/german/print_all/content/9797_LME_UserGuide.pdf

2.3.4 Nachteil von LEGO MINDSTORMS

Der in dieser Arbeit vorgestellte Weg, das Programmieren zu erlernen, erfordert es, dass die Lernenden einen eigenen Roboter zuhause haben, um üben zu können – oder zumindest einen geliehenen, der ihnen zur Verfügung steht. Während also beim alleinigen Üben einer Programmiersprache auf anderem Wege und mit anderen Tools kein LEGO-MINDSTORMS-Roboter, sondern nur ein Computer notwendig wäre, brauchen die Lernenden hierfür einen Roboter zuhause.

Aufgrund der hohen Anschaffungskosten von mehreren hundert Euro wird es schwierig für die Jugendlichen, zu einem LEGO MINDSTORMS-Bausatz zu kommen und für viele stellt das ein zu großes Hindernis dar.

2.4 Vorgehensweisen

Das weit verbreitete und beliebte Hello-World-Programm wurde bereits 1999 als kein guter Einstieg in die objektorientierte Programmierung identifiziert, siehe [SPOL99]. Auch wenn mittlerweile die Lehrenden immer seltener diesen Programmiereinstieg wählen, so gibt es doch noch alteingesessene LehrerInnen, die diese Variante bevorzugen.

Schon 1999 zählt Sally Fincher in [FINC99] einige verschiedene Ansätze der Programmierlehre auf: den Syntax-freien Ansatz, den „Literacy“-Ansatz, den Problemlösungsansatz und den „Computation as Interaction“-Ansatz. All diese Wege, Programmieren zu unterrichten, können sowohl im Object-First, als auch im Object-Later (Kapitel 5.4.1 und 5.4.2) eingesetzt werden.

Syntax-freier Ansatz kommt häufig vor, ist aber oft nur der Einstieg, dem später das Erlernen einer Programmiersprache folgt. Hierbei geht es meist um die algorithmische Denkweise und das Computerverständnis im Allgemeinen. [FINC99]

Da jeder/jede SchülerIn in seinem/ihrem Leben bereits eine abstrakte Notation gelernt hat, nämlich das Alphabet, wird es ihm/ihr laut Literacy-Ansatz leichter fallen, das Programmieren auch auf diesem Weg zu lernen. Jedoch wurde das Alphabet in

sehr jungen Jahren gelernt, und das Gehirn und seine Struktur ändern sich sehr deutlich im Laufe der Jahre, weshalb dieser Ansatz in späteren Jahren ein Problem darstellen könnte. Im Gegenteil zum Syntax-freien Ansatz wird mit einer Programmiersprache begonnen, keine Notation oder Pseudocode werden zu Beginn gezeigt. [FINC99]

Beim Problemlösungsansatz werden Fähigkeiten beigebracht, die ein/eine ProgrammiererIn außer den Codierungsskills noch braucht. Die Fertigkeit, Probleme lösen zu können, lässt sich auf die unterschiedlichsten Domänen übertragen und bringt den Lernenden daher im Leben weitere Vorteile als die bloße Beherrschung einer Programmiersprache. [FINC99]

„Computation as Interaction“ nach [FINC99] beschreibt, dass alle Studenten, die heutzutage Programmieren lernen, in einer Welt mit dem Computer aufgewachsen sind. Der Computer wird als multi-threaded und GUI-gesteuertes Gerät wahrgenommen und durch den „single-threaded“ Problemlösungsweg werden Studenten und Computer nicht richtig gefordert und gefördert.

Wurde keine der bereits erwähnten Einführungen gewählt, so werden immer öfter Modelle benutzt, die einen spielerischen Zugang zur objektorientierten Programmierung repräsentieren. Diese Mikrowelten funktionieren auf unterschiedliche Weise, einige basieren auf grafischer Programmierung, andere, wie z.B. Java-Hamster[5], bieten grafische Anreize und ermöglichen den Lernenden so, die Auswirkungen ihres Programms direkt zu verfolgen und nachzuvollziehen. Auch Scratch[6] stellt eine dieser Mikrowelten dar. Die jeweiligen Vor- und Nachteile dieser unterschiedlichen, eher neuartigeren Programmier-AnfängerInnen-Tools sind in anderen Arbeiten, z.B. [Will10], genügend und ausführlich erläutert.

2.5 Textuelle Programmierung

Warum überhaupt textuelle Programmierung erlernen? [Carl09] zeigt, dass Studen-

5 [Bole08]

6 http://scratch.mit.edu/

ten bessere Ergebnisse liefern, wenn sie Programmieren mit einer grafischen Programmieroberfläche lernen. Durch den bereits erfolgten Einsteig mittels der grafischen LEGO-Oberfläche wurden die Vorteile von grafischer Programmierung bereits genutzt. Die Software von LEGO bietet viele Möglichkeiten, Konzepte der Programmierung kennenzulernen und anzuwenden.

Irgendwann sind aber entweder die Möglichkeiten der grafischen Programmieroberfläche ausgereizt, und gewisse komplizierte Roboter sind damit nicht mehr zu programmieren, oder die interessierten Kinder und Jugendlichen wollen etwas Neues lernen, was auch außerhalb der Roboterwelt zum Programmieren verwendet werden kann. Außerdem bieten die meisten grafischen Programmiersysteme nicht komplette Entwicklerfreiheit, meistens sind diese entweder einfach zu verstehen, dann aber nur beschränkt nutzbar, oder sie sind weitläufiger verwendbar, dann aber oft schwieriger zu verstehen als eine textuelle Programmiersprache, die der natürlichen Sprache möglichst nahe kommt.

Wollen die AnfängerInnen etwas Neues lernen, so wäre es auch eine Option, mit diversen anderen Tools, die zum Erlernen von Programmiersprachen erfunden wurden, weiterzumachen. Dann wäre es aber schade um das bereits vorhandene spezifische Vorwissen, welches dann nicht genützt werden würde. Deshalb soll es nicht Thema dieser Arbeit sein, die unterschiedlichen Tools zu untersuchen, sondern es soll auf diesem Wissen aufgebaut werden und der Einstieg in Java dadurch so einfach und schnell wie möglich erfolgen können. Lernende fühlen sich im neuen Umfeld viel wohler, wenn sie bereits Experten auf einem Teilgebiet sind [HUND06], was beim Erlernen von Java mit LEGO-Robotern der Fall ist, weil die Jugendlichen bereits viel Erfahrung mit den Robotern gesammelt haben.

2.6 Warum objektorientiert?

Die Frage stellt sich, warum als erste Programmiersprache eine objektorientierte gewählt werden soll, denn die Auswahl an Programmierparadigmen ist groß ([BOLE10], [@para]):

- Imperative Programmierung: Zur imperativen Programmierung zählen viele

Sprachen, unter anderem auch Modula-2. Die imperative Programmierung geht Befehl für Befehl der Reihe nach durch und verwendet für die Ablaufplanung Kontrollkonstrukte.

- Prozedurale Programmierung: Die prozedurale Programmierung stellt eine Erweiterung der imperativen Programmierung dar und bietet weitere Kontrollkonstrukte, die hauptsächlich der Übersichtlichkeit dienen. So können hier Funktionen erstellt werden und dadurch, zusätzlich zu den in der imperativen Programmierung vorhandenen Kontrollkonstrukten, Wiederholungen im Code vermieden werden. Zeitlich betrachtet war diese Art des Programmierens bereits Jahre vor dem objektorientierten Programmieren vorhanden. Beispiele für solche Programmiersprachen sind Fortran, Cobol und Pascal.

- Funktionale Programmierung: Zu den funktionalen Programmiersprachen zählen Lisp, Miranda und Haskell. Solche Programmiersprachen verwenden nur Funktionen, keine anderen Kontrollkonstrukte. Diese Programmierrichtung kommt z.B. bei der Hardware-Simulation und beim Compiler-Bau zum Einsatz.

- Logische Programmierung: Dieses Programmierparadigma wird auch oft als prädikative Programmierung bezeichnet und unterscheidet sich von der imperativen Programmierung dahingehend, dass keine Reihenfolge für Anweisungen vorgegeben wird, sondern vielmehr Regeln programmiert werden, die dann mittels Logik gelöst werden. Prolog zählt z.B. zu den logischen Programmiersprachen.

- Objektorientierte Programmierung: Bei der objektorientierten Programmierung handelt es sich um Paradigma, welches dem natürlichen Denken der Menschen am ehesten entspricht. Durch die Abstraktion der Objekte aus der realen Welt in objektorientierte Konzepte, wie z.B. Vererbung, Interfaces oder Klassen, wird versucht, Daten möglichst gebündelt und zusammengehörend zu speichern.

Weshalb soll die Objektorientierung gelernt werden? Ein Grund dafür ist, dass es die

gängigste Form der Programmierung in der Praxis und Gegenwart ist.
Doch auch weitere Vorteile (nach [Bole10]) stellen gute Gründe für OOP dar:

- Modellierung und Repräsentation kommen der menschlichen Denkweise sehr nahe.
- Daten und die dazugehörigen Methoden werden in Klassen zusammengefasst, was besonders der Übersichtlichkeit und Verständlichkeit dient und somit eine geringere Fehleranzahl verursacht.
- Wiederverwendbarkeit: „Teile von Programmen lassen sich einfach und flexibel in anderen Programmen wiederverwenden.“ [Bole10]
- Erweiterbarkeit: „Programme lassen sich einfach erweitern und an geänderte Anforderungen anpassen.“ [Bole10]
- Durch ein einheitliches Modell im gesamten Entwicklungsprozess ist in jeder Phase eine bessere Wartbarkeit gegeben.

Objektorientierung ist sicher nicht die Heilung aller in der Softwareentwicklung auftretenden Probleme, denn auch sie hat ihre Grenzen:

- In einigen Fällen kann es zu einer komplizierteren Problembeschreibung kommen, da Aktionen in der Objektorientierung zwangsläufig an Objekte gebunden sind. Bei anderen Sprachen sind Aktionen eigenständig und können unabhängig von Objekten eingesetzt werden.
- Dynamisches Laufzeitverhalten und Kontrollfluss werden vor dem/der EntwicklerIn abgekapselt – was der besseren Wartbarkeit dient – jedoch sind sie somit nicht von ihm/ihr beeinflussbar.
- Relationale Datenbanken können schlecht an objektorientierte Sprachen angeschlossen werden. Für dieses Problem existiert sogar die eigene Bezeichnung „Object-relational impedance mismatch“, welche beschreibt, wie diese Differenz zwischen Objektdarstellung und relationalen Daten zu lösen versucht wird. Noch existiert jedoch keine Lösung, die keinerlei Einschränkung für eine der beiden Seiten bedeutet.
- Die Laufzeiteffizienz für Algorithmen ist bei prozeduralen Programmiersprachen deutlich besser als bei objektorientierten Sprachen, dafür ist die Leistungsaufnahme stark erhöht, was bei mobilen Applikationen einen enormen Vorteil mit sich bringt. [@oop]

Es stellt sich demnach die Frage, welche objektorientierte Programmiersprache erlernt werden soll. Werden dazu viele verschiedene SoftwareentwicklerInnen befragt, so entstehen viele unterschiedliche Antworten, welcher Programmiersprache der Vorzug zu geben sei. Die Entscheidung für eine Sprache ist immer geprägt durch die persönlichen Vorlieben und durch das Ziel, das mit der Programmierung versucht wird zu erreichen. Da für das Konzept dieser Arbeit Java als Programmiersprache verwendet wird, wird in Kapitel 2.6 näher darauf eingegangen, warum Java sinnvoll ist.

Darüber, ob in einer objektorientierten Sprache direkt mit den objektorientierten Konzepten begonnen werden sollte, oder ob zuvor bereits prozedurale und imperative Grundkonzepte bekannt sein sollten, sind sich WissenschaftlerInnen auf der ganzen Welt nicht einig. Diese Thematik wird in Kapitel 5.5 behandelt.

2.7 Java

Die Lernenden sollen das Wissen, welches sie bereits aus der grafischen Programmierung der LEGO-MINDSTORMS-Roboter erworben haben, mitnehmen und so leichter und schneller eine textuelle Programmiersprache erlernen. Aber welche textuelle Programmiersprache ist geeignet, um auf dem Wissen aufzubauen und damit auch die Roboter programmieren zu können?

LeJOS NXJ ist eine winzige Java Virtual Maschine und bietet laut offizieller Website[7] folgende Möglichkeiten:

- „Object oriented language (Java)
- Preemptive threads (tasks)
- Arrays, including multi-dimensional
- Recursion
- Synchronization
- Exceptions

7 http://www.lejos.org/

- Java types including float, long, and String
- Most of the java.lang, java.util and java.io classes
- A Well-documented Robotics API“

LeJOS stellt ein sehr komfortables Plugin für eclipse zur Verfügung, ermöglicht die Verwendung der gängigsten Java-Bibliotheken und kann auch durch seine Dokumentation überzeugen. Für leJOS spricht auch, dass bei der Verwendung dieser JVM eine professionelle Entwicklungsumgebung verwendet wird und nicht, wie z.B. im Java-Hamster-Buch, eine eigens für AnfängerInnen erstellte Entwicklungsumgebung.

Neben leJOS existieren noch weitere Varianten, den beliebten MINDSTORMS-Roboter mit einer textuellen Programmiersprache zu programmieren. Eine davon ist NXC[8] (Not eXactly C), welche eine High-Level-Sprache, ähnlich der von C, zur Roboter-Programmierung zur Verfügung stellt. NXC wurde auf dem NBC Compiler, welcher eine einfache Sprache zum Programmieren der LEGO-Roboter zur Verfügung stellt, aufgesetzt.

Auf der offiziellen Website von NXC steht: „If you are just getting started with programming, then graphical environments such as the MINDSTORMS NXT software may be better choices for you. If, however, you're a programmer and you prefer typing a few lines to drag and drop icon programming, then either NBC or NXC may be perfect for you.“ [@bric] Die Zielgruppe dieser Arbeit hat bereits mit der grafischen Oberfläche gearbeitet, zählt sich aber auch noch nicht zu den Programmierern, die in diesem Zitat erwähnt werden. Dennoch wäre es mit geeigneter Unterstützung für diese Altersgruppe möglich, die Roboter auch mit NXC zu programmieren.

8 http://bricxcc.sourceforge.net/nbc/

Jul 2014	Jul 2013	Change	Programming Language	Ratings	Change
1	1		C	17.145%	-0.48%
2	2		Java	15.688%	-0.22%
3	3		Objective-C	10.294%	+0.05%
4	4		C++	5.520%	-3.23%
5	7	↑	(Visual) Basic	4.341%	+0.01%
6	6		C#	4.051%	-2.16%
7	5	↓	PHP	2.916%	-4.27%
8	8		Python	2.656%	-1.38%
9	10	↑	JavaScript	1.806%	-0.04%
10	12	↑	Transact-SQL	1.759%	+0.19%
11	9	↓	Perl	1.627%	-0.52%
12	13	↑	Visual Basic .NET	1.495%	+0.24%
13	37	⇈	F#	1.093%	+0.86%
14	11	↓	Ruby	1.072%	-0.51%
15	45	⇈	ActionScript	1.067%	+0.86%
16	-	⇈	Swift	1.054%	+1.05%
17	17		Delphi/Object Pascal	1.031%	+0.34%
18	15	↓	Lisp	0.829%	-0.04%
19	18	↓	MATLAB	0.781%	+0.10%
20	20		Assembly	0.777%	+0.20%

Abb. 3: TIOBE Index[9] Juli 2014

Die TIOBE Programming Community veröffentlicht monatlich eine Index der populärsten Programmiersprachen. Die Reihung basiert auf der Anzahl geschulter Entwickler weltweit, auf den Kursen und den Angaben von Verkäufern. Die bekanntesten Suchmaschinen, wie Google, Bing, Yahoo!, Wikipedia, Amazon, YouTube und Baidu werden zur Berechnung herangezogen. TIOBE erwähnt besonders, dass der in Abbildung 3 gezeigte Index nicht die besten Programmiersprachen oder die Sprachen mit den meisten Zeilen an Code zeigt. Laut dem Index von TIOBE im Juli 2014 steht C in der Beliebtheit sogar vor Java, doch da in Kapitel 2.5 festgelegt wurde, dass die Lernenden eine objektorientierte Sprache erlernen sollen, fällt C als nicht objektorientierte Sprache aus der Auswahl heraus. Außerdem gibt es auf die Frage, welche

9 TIOBE Index Juli 2014 http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html

Programmiersprache die erste sein soll, die erlernt wird, wohl keine richtige oder falsche Antwort gibt, da dies immer situationsabhängig beantwortet werden muss.

Im Folgenden wird genauer auf leJOS und Java eingegangen. Java ist heutzutage auf 850 Millionen Computern installiert und wird für diverse Programme, Spiele und Geschäftsanwendungen verwendet. Java ist nicht nur eine Programmiersprache, sondern auch eine schnelle und zuverlässige Laufzeitumgebung. Die Zahl der Websites und Anwendungen, die Java verwenden, nimmt täglich zu, und es ist kostenlos herunterzuladen für jedermann. [@java] Die weite Verbreitung von Java stellt einen Grund dar, warum die Jugendlichen von der grafischen Oberfläche auf die Programmiersprache Java umsteigen sollten.

„... Java ist aus der Werkzeugtruhe eines Programmierers aktuell kaum weg zu denken.“ [@jax] Mit Java werden viel öfter seriöse Projekte abgewickelt, als es z.B. bei PHP oder Ruby der Fall ist. Seriöse Projekte bieten aber selten die Möglichkeit, etwas Tolles oder Lustiges herzuzeigen oder zu erzählen. Dementsprechend werden nicht viele – für Kinder spannende – Geschichten über Java Anwendungen erzählt. Somit bieten die Roboter nicht nur die Möglichkeit, die Grundlagen von Java zu lernen und erzeugen Motivation sich weiter mit dieser Programmiersprache zu beschäftigen, sie liefern auch jene Geschichten, die gewöhnliche Java-Projekte womöglich nicht bieten können.

Ein anderer Grund für diese Programmiersprache ist die Einfachheit von Java im Vergleich zu den Programmiersprachen C oder C++. Zeiger, Speicherplatzreservierung und dergleichen fallen weg und erleichtern den Einstieg ungemein, denn Java wurde auch aus dem Grund der Einfachheit entwickelt. Dazu kommt noch, dass es kostenlos ist und eine umfangreiche Bibliothek zur Verfügung stellt. Natürlich sind auch in der Programmiersprache Java kompliziertere Konzepte enthalten, diese müssen jedoch nicht gleich zu Beginn erlernt werden, denn sie sind für den Einstieg und die Programmierung der LEGO-MINDSTORMS-Roboter nicht zwangsläufig notwendig.

3. Bücheranalyse

In diesem Kapitel werden ausgewählte Bücher zum Lernen von Java genau untersucht, inwieweit sie geeignet sind, der Zielgruppe – mit entsprechendem Vorwissen – Java näherzubringen. Unter den ausgewählten Büchern befinden sich welche, die mit LEGO MINDSTORMS arbeiten, und solche, die reine Java-AnfängerInnen-Bücher sind. Demnach werden die Bücher in folgende Teilkategorien unterteilt: Java Einsteiger Bücher, Bücher mit speziellen didaktischen Java-Umgebungen und Bücher über LEGO MINDSTORMS.

Aus der Vielzahl vorhandener Bücher wurde eine Vorauswahl getroffen, und es werden im Anschluss acht Bücher vorgestellt, welche zumindest teilweise das Themengebiet „Java lernen mit LEGO MINDSTORMS“ abdecken. In der Analyse werden folgende Kriterien untersucht:

Aufbau und Gliederung:

- Aufbau der Gliederung
- Transparenz der Gliederung
- Umfang der Abschnitte
- Beziehung der Abschnitte zueinander
- Verhältnis von Beispielen und Theorie
- Zusammenfassung am Ende des Kapitels

Zielgruppe:

- Vorwissen und Erfahrung der Lesenden
- Angesprochene Altersgruppe

Methodisches Konzept:

- Verwendete Sprache
- Spielerische Auseinandersetzung
- Ziele transparent
- Präsentation des Inhalts
- Wurde Object-First gewählt oder andere Ansätze?

Lehrziele:

- Behandelte Themen

3.1 Java Einsteiger Bücher

Aus der Vielzahl der Java-Einsteiger-Bücher auf dem Markt wurden einige deutschsprachige Werke ausgewählt und auf ihre Tauglichkeit zum Erlernen von Java für Jugendliche geprüft. Natürlich gibt es noch viel mehr Bücher als hier vorgestellt, jedoch wurden solche nicht untersucht, welche zu wissenschaftlich sind, zu detailliert oder für Java-ProgrammieranfängerInnen, die bereits eine andere Sprache beherrschen, geschrieben wurden. Darüber hinaus wurden nur Bücher analysiert, die dazu dienen, die Programmiersprache Java zu erlernen, und nicht solche, die ein Nachschlagewerk darstellen. Bei den untersuchten Büchern wurde beim Erstellen Wert auf die didaktische Vorgehensweise gelegt.

3.1.1 Buch 1: „Java will nur spielen“

Das erste analysierte Buch ist „Java will nur spielen: Programmieren lernen mit Spaß und Kreativität“ [Pani08] von Sven Eric Panitz aus dem Jahr 2008. Es handelt sich um ein Einsteigerbuch, bei dem der Autor empfiehlt, nebenbei noch ein Lehr- und Nachschlagewerk bereitzuhalten.

a) Aufbau und Gliederung

Aufbau der Gliederung: Mit seinen 248 Seiten ist dieses Buch eine nette Einführung in die Java-Programmierung für Studenten, welche bereits die ein oder andere Programmiersprache verwendet haben und auch mathematisch vor keiner Herausforderung stehen, wenn sie dieses Buch lesen.

Transparenz der Gliederung: Leider ist aufgrund der Gliederung nicht gleich ersichtlich, worum es in den jeweiligen Kapiteln geht.

Umfang der Abschnitte: Die Abschnitte sind kurz und übersichtlich, somit auch für die Zielgruppe dieser Arbeit geeignet.

Beziehung der Abschnitte zueinander: Alle Bereiche bauen aufeinander auf und die Abfolge der Kapitel ergibt Sinn.

Verhältnis von Beispielen, Einführung und Theorie: Gut ist, dass sich die Kapitel nicht an den theoretischen Konzepten der Objektorientierten Programmierung orientieren, sondern anhand der kleinen Spiele die Java Programmierung beigebracht wird und die jeweiligen Programmierthemen in den Hintergrund rücken (siehe Abbildung 4). So werden Neulinge nicht durch Begrifflichkeiten verwirrt, die sie zu Beginn des Buches sowieso noch nicht zuordnen können.

Zusammenfassung am Ende des Kapitels: Die großen Kapitel haben ein eigenes Unterkapitel, welches immer „In diesem Kapitel eingeführte Javaeigenschaften“ heißt, zu sehen in Abbildung 4 (Kapitel 2.5). Darin wird mit wenigen Worten nochmals erklärt, welche Java-Konzepte zuvor eingeführt wurden.

b) Zielgruppe

Vorwissen und Erfahrung der Lesenden: Dieses Buch setzt voraus, dass gewisse Konstrukte aus der textuellen Programmierung bereits bekannt sind. Dazu zählen if/else, Schleife, primitive Datentypen, Variablen und deren Zuweisung.

Da keine Entwicklungssoftware verwendet wird, wäre es für die Zielgruppe der 12- bis 16-Jährigen ein großes Hindernis, sich selbst eine geeignete Entwicklungsumgebung auszuwählen, richtig zu installieren und anschließend auch entsprechend zu verwenden.

Angesprochene Altersgruppe: Der Titel verspricht ein Buch, welches auch für jüngere Leser geeignet sein könnte. Es stellt sich jedoch heraus, dass es eher an Studenten gerichtet ist, die zwar noch nicht mittels Java programmieren können, die aber bereits die ein oder andere textuelle Programmiersprache verwendet haben.

Abb. 4: Inhaltsverzeichnis von Buch 1

c) Methodische Konzepte

Verwendete Sprache: Leider werden Begriffe verwendet, bevor sie eingeführt werden, und durch Zusammenhänge lassen sich die Bedeutungen dieser Wörter nicht erkennen. Wie z.B. „Hauptmethode“ im Kapitel 2.1.3, obwohl Methoden erst einige Seiten später erklärt werden.

Spielerische Auseinandersetzung: Durch die Programmierung eines Spiels ist ein spielbezogener Zusammenhang durchaus gegeben, jedoch ist der Text eintönig und wenig spielerisch gestaltet.

Ziele transparent: Positiv fällt auf, dass ein roter Faden von Beginn an bis zum Schluss durch dieses Buch führt. Anhand von Spielen wird die notwendige Theorie erklärt, die benötigt wird, damit die LeserInnen Spiele programmieren können. So ist während des ganzen Buches zu erkennen, was das Ziel ist und wozu man die jeweiligen Konzepte verwenden kann. Die Frage „Wozu braucht man das jetzt?“ entsteht hier bei den Lesenden mit Sicherheit nicht.

Präsentation des Inhalts: Alle Beispiele behandeln die Programmierung von Spielen. Zu Beginn werden Bildpunkte im zweidimensionalen Raum gezeichnet und gespeichert, dann wird dieses Grundgerüst um geometrische Figuren, Spielfelder, Interaktionen erweitert, bis zum Schluss ein einfaches Spiel erstellt wird.

Ansatz: Viele kleine Beispiele und Tests erleichtern das Mitarbeiten für die Lesenden. Leider erfreuen sich geometrische Figuren als gewählte Beispiele bei Jugendlichen eher wenig an Beliebtheit. Die vielen mathematischen Beispiele sind für ein sehr junges Publikum ebenfalls ungünstig gewählt, weil sie dessen Motivation dämpfen könnten. Denn sofern die Lernenden nicht überdurchschnittlich an Mathematik interessiert sind, werden sie sich durch die Mathematik überfordert fühlen.

d) Lehrziele

Es wird keine Entwicklungsumgebung vorgestellt, denn es wird empfohlen, zuallererst Java von der Konsole aus zu starten, also keine Entwicklungssoftware zu verwenden, weil diese sonst viele technische Details vor dem/der EntwicklerIn versteckt und das Erlernen solch einer Umgebung – laut Autor – wiederrum ein Hindernis darstellt.

Die Grundlagen der objektorientierten Programmierung sind das Hauptthema dieses Buches: Klassen, Objekte, Vererbung, Überschreiben von Methoden, Standardmethoden und Interaktionen des Users. Aber auch SWING, grafische Oberflächenprogrammierung, Ereignisbehandlung, innere anonyme Klassen, Menüleisten, Dialogfenster, Layout-Manager, Ausnahmebehandlungen, Netzwerkkommunikation über RMI, Java ME, Algorithmen aus der KI, XML-Verarbeitung mit DOM, SAX und StAX und der Aufzählungstyp enum werden erklärt.

Gewisse Inhalte sind auf universitärem Niveau: Reflexivität, Symmetrie und Transivität von Objekten sind nur eine Auswahl an Konzepten, die vorkommen, aber für Java-EinsteigerInnen der untersuchten Zielgruppe absolut nicht notwendig sind. Viele der Kapitel sind für die Altersgruppe der 12- bis 16-Jährigen viel zu komplex und vor allem für den Einstieg in Java und die Programmierung der LEGO-MINDSTORMS-Roboter unnötig. So z.B. RMI und Algorithmen der Künstlichen Intelligenz, Menüleisten, Layout-Manager, Java ME, XML Verarbeitung und Enums.

Das Buch behandelt zwar die Programmierung kleiner Spiele, soll aber kein Werk sein, anhand dessen Spieleprogrammierung im Großen gelernt werden kann, so der Autor selbst. Die Spiele dienen mehr der Motivation und dem besseren Verständnis der Sinnhaftigkeit von gewissen objektorientierten Konstrukten.

Zu Beginn des Buches gibt es das Kapitel „Java in 3 Minuten“, welches die grundlegenden Konstrukte, die vorausgesetzt werden, noch einmal kurz und bündig in Java erklärt. Das Paradoxe daran ist, dass selbst der Autor erwähnt, dass dieses Kapitel wohl nicht in 3 Minuten durchführbar ist.

e) Fazit

Dieses Buch veranschaulicht gut, wie ein roter Faden durch ein ganzes Buch führen kann und die Programmier-Themen dabei in den Hintergrund rücken, ohne jedoch vergessen zu werden. Es stellt durch den Schreibstil und die Art der verwendeten Beispiele aber kein geeignetes Werk für die 12- bis 16-Jährigen dar.

3.1.2 Buch 2: „Java von Kopf bis Fuß“

Das Buch „Java von Kopf bis Fuß“ [Sier06] von Kathy Sierra und Bert Bates ist 2006 erschienen und stellt ein weiteres Werk zum Erlernen von Java dar. Dabei handelt es sich um kein gewöhnliches Programmierlernbuch, sondern um ein didaktisch sehr gut aufbereitetes, zum Schmunzeln anregendes Lernbuch für Java-AnfängerInnen.

Java von Kopf bis Fuß ist ein Buch der „Von Kopf bis Fuß“-Buchreihe, die von unter-

schiedlichen Autoren zu den unterschiedlichen technischen, genauer gesagt zu den Computer-, bzw. Software-Entwicklungsthemen, geschrieben wurden.

a) Aufbau und Gliederung

Aufbau der Gliederung: Die neuartig gemachten Zusammenfassungen von jedem Kapitel zu Beginn des Buches springen direkt ins Auge, da nicht nur ein gewöhnliches Inhaltsverzeichnis vorhanden ist. Jedes Kapitel wird am Anfang des Buches in einem Absatz mit einigen passenden kleinen Bildern bekannt gemacht (z.B. Kapitel 2 in Abbildung 5). So kann sich der/die Lesende gut vorstellen, was auf ihn/sie zukommt, ohne zuvor Ahnung von der Materie zu haben.

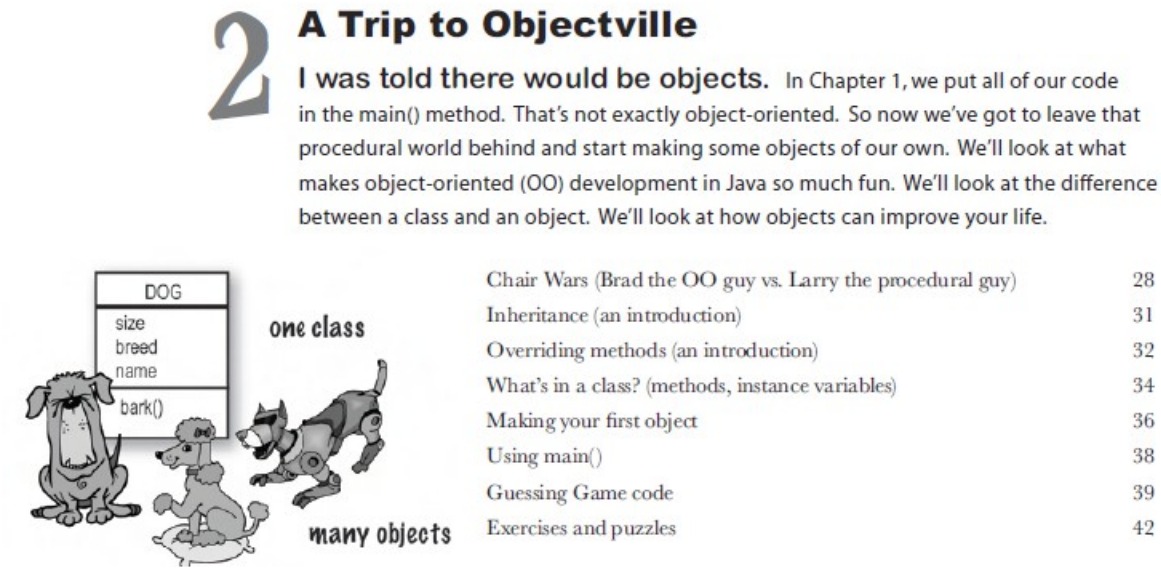


Abb. 5: Ausschnitt des Inhaltsverzeichnisses aus „Java von Kopf bis Fuß“

Transparenz der Gliederung: Die Gliederung ist durch die kurzen Erklärungen und die in Klammern geschriebenen Begriffe sehr verständlich, und die Ziele offenbaren sich dem/der LeserIn gleich zu Beginn.

Umfang der Abschnitte: Das komplette Buch wirkt mit seinen 690 Seiten sehr umfangreich, jedes Kapitel für sich ist jedoch kurzweilig zu lesen und durch die unterschiedlichen didaktischen Methoden, welche noch näher erklärt werden, sehr abwechslungsreich.

Beziehung der Abschnitte zueinander: Die Kapitel stehen zueinander in einer ausgewogenen Beziehung, sie bauen aufeinander auf und verweisen auch auf die entsprechenden Kapitel, wenn etwas an anderer Stelle ausführlicher erklärt wird.

Verhältnis von Beispielen, Einführung und Theorie: Es sind sehr viele verschiedene Beispiele vorhanden. Die Einführung ist lang genug und abwechslungsreich, die Theorie kommt auch nicht zu kurz und wird spannend in Beispielen eingebaut.

Zusammenfassung am Ende des Kapitels: Es sind keine Zusammenfassungen am Ende der Kapitel zu finden, stattdessen befinden sich diese am Beginn des Buches über jedes Thema.

b) Zielgruppe

Vorwissen und Erfahrung der Lesenden: Programmierung von Schleifen und if/else-Anweisungen sind den Autoren als vorausgesetzte Programmiererfahrung ausreichend egal, ob diese nun in einer textuellen oder grafischen Programmierumgebung gesammelt wurden, lediglich die Beherrschung dieser Konzepte zählt. Es sollte jedoch mehr als bloßes HTML-Wissen vorhanden sein. Das Vorwissen für dieses Buch ist also durchaus bei den mit dieser Arbeit angesprochenen Lernenden vorhanden.

Angesprochene Altersgruppe: Die Autoren beantworten gleich zu Beginn die Frage, für wen das Buch ihrer Meinung nach geeignet ist: nämlich für diejenigen, die bereits erste Programmiererfahrungen haben, für diejenigen die Java lernen wollen und für solche, die lieber an anregenden Konversationen auf Partys interessiert sind als an trockener, technischer Literatur.

c) Methodische Konzepte

Verwendete Sprache: Die benutzte Sprache ist amüsant und für jedermann leicht verständlich. Der Schreibstil ist für die jüngere Zielgruppe der über 10-Jährigen genauso gut geeignet wie für Erwachsene oder Studenten.

Spielerische Auseinandersetzung: Die bildliche Darstellung ist gut durchdacht und wirklich gelungen.

Ziele transparent: In jedem Kapitel sind die Lernziele stets bekannt und durch die zahlreichen Zusammenfassungen, sowohl am Buchanfang als auch zu Beginn jedes Kapitels, verständlich erklärt.

Präsentation des Inhalts: Dieses Buch hat mehr Seiten als das Buch 1, welches bis auf ein paar Bereiche dieselben Themengebiete abdeckt. Der Grund hierfür sind die unterschiedlichen Methoden, die in diesem Buch verwendet werden, um das Neuerlernte zu festigen.

Ansatz:
Trotz 690 Seiten stellt es kein Nachschlagewerk für Java dar. Die Autoren verwenden Bilder, die den Lesern dabei helfen sollen, sich den Inhalt leichter zu merken. Durch Quiz-Aufgaben und Herausforderungen wird versucht, die Lernenden mehr zum Denken anzuregen. Hinzu kommt die persönliche Ansprache – auf eine lustige Art und Weise – des/der Lernenden, welche eine emotionale Bindung herstellen soll. Wiederholungen kommen in unterschiedlicher Weise mit unterschiedlichen Methoden vor, um die verschiedenen Sinne anzusprechen. Die Autoren versuchen, durch Humor und Überraschungen die Gehirnzellen zum Nachdenken zu animieren. Eine individuelle Gestaltung mit persönlicher Ansprache wird verwendet, weil dem Gehirn dadurch vorgetäuscht wird, dass es sich in einer persönlichen Unterhaltung befindet, und dieses dem Inhalt so mehr Interesse zukommen lässt.

Dieses Lernbuch gehört sicherlich zu den moderneren didaktischen Unterlagen, die auf dem Markt vorhanden sind, denn es verwendet die unterschiedlichsten didaktischen Methoden, aber es ist zugleich auch gut gegliedert, und der rote Faden geht vom Beginn bis zum Schluss nie verloren.

Die didaktische Gestaltung des Werks wird durch die unterschiedlichen Lernarten unterstützt, einige Lernende mögen „step-by-step“ lernen, und andere möchten lieber zuerst das große Ganze erfassen. Darüber hinaus profitieren alle Lernenden von den

unterschiedlichen Darstellungsformen, welche darin vorkommen. Geschichten und Fragen, mit nicht immer einfachen Lösungen, stellen eine Herausforderung für diejenigen dar, die gerne knobeln, und lassen die Gedanken länger bei Java weilen, vielleicht auch dann noch, wenn das Buch schon zur Seite gelegt wurde. Die Autoren schreiben nicht über alle möglichen Java-Themen, sondern nur über die Dinge, die auch wirklich von einem/einer Java-AnfängerIn gebraucht werden.

Bis Kapitel 16 wird keine Entwicklungsumgebung eingesetzt, sondern es wird nur der Texteditor verwendet. Erst wenn die Lernenden die grundlegenden Themen verstanden haben, sollten diese – den Autoren zufolge – mit einer Entwicklungssoftware beginnen.

Dialoge werden ebenfalls als Methode benutzt, z.B. führen der Compiler und die JVM ein Gespräch darüber, wer sinnvoller ist und welche Aufgaben wer hat. Diese Art und Weise, Inhaltliches näherzubringen liest sich viel spannender, als würden einfach nur die Eigenschaften und Aufgaben der beiden aufgezählt werden. So lässt sich eine Beziehung zum Compiler und zur Java Virtuell Maschine aufbauen, und dies spricht viel mehr Bereiche im Gehirn an, als es bei reiner Aufzählung der Fall wäre.

Ein weiteres didaktisches Mittel, das zum Einsatz kommt, ist der spielerische Gedanke des Anordnens von Codeteilen. Dabei wird von Kühlschrankstickern gesprochen, diese sind bereits beschriftet, liegen durcheinander vor und gehören vom Leser geordnet – in der entsprechenden Reihenfolge – auf den linken Teil der Seite (den sogenannten Kühlschrank) sortiert, sodass das geforderte Ergebnis – nämlich die entsprechende Konsolenausgabe – dabei entsteht.

Dieselbe Methode, nur etwas anders dargestellt, wird im Buch Pool Puzzle genannt und bietet neben einer kurzen Erklärung der Aufgabenstellung zusätzlich die Programmausgabe (Ziel der Aufgabe), die Bonusfrage und den Antworten-Pool. Die Lernende suchen sich aus dem Pool der unterschiedlichen Codeteile den jeweils passenden – um zur gezeigten Ausgabe zu kommen – Code aus, notieren diesen an der geeigneten Stelle und streichen ihn aus dem Pool. Zu sehen in Abbildung 6.

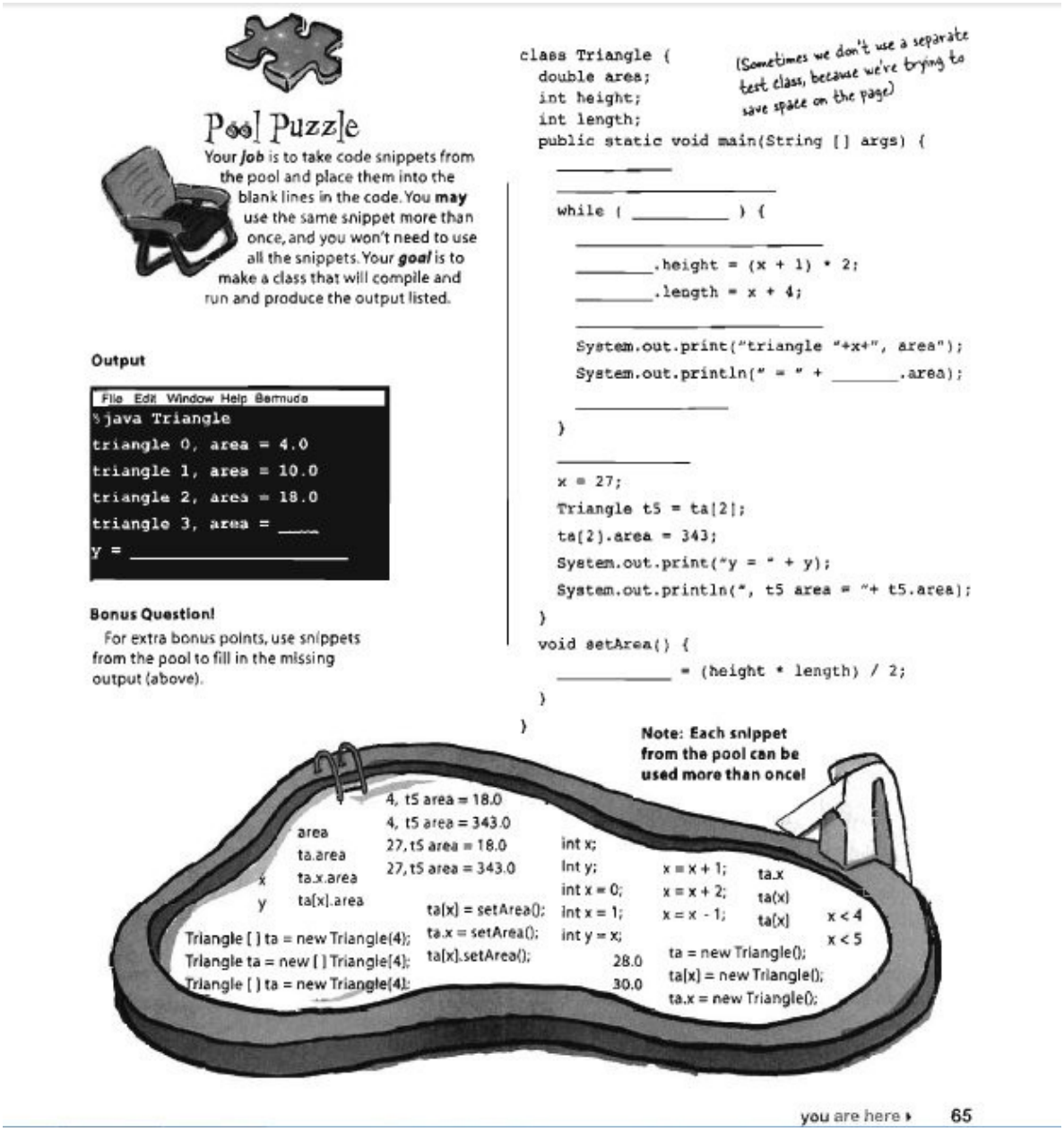


Abb. 6: Pool-Puzzle-Beispiel aus „Java von Kopf bis Fuß“

d) Lehrziele

Nach einem kurzen Überblick über prozedurales Grundwissen beginnt die Einführung in die Objektorientierung. Danach geht es weiter mit Vererbung und Objektbeziehungen, Polymorphismus, Java Library, Variablen – sowohl Primitive als auch Referenzen – Ausnahmebehandlungen, Event Handling und GUI, SWING, Datenstrukturen, Netzwerkkommunikation, Remote Deployment, Datenstrukturen, RMI und EJB; und zum Schluss werden noch die Top-10-Themen kurz erklärt, die kein eigenes Kapitel im Buch erhalten haben.

Neben den objektorientierten Programmierkonzepten werden auch gängige bzw. sinnvolle Vorgehensweisen aus der Programmierpraxis erläutert und gezeigt, wozu vieles an theoretischem Wissen verwendet werden kann und gebraucht wird.

e) Fazit

Java von Kopf bis Fuß ist sehr interessant und fesselnd für ein Fachbuch – und auch für Jugendliche besser geeignet als Buch 1. Es werden viele verschiedene didaktische Methoden verwendet und somit die unterschiedlichsten Lerntypen angesprochen. Viel geschieht auch, ohne in den Bildschirm schauen zu müssen. Das Konzept dieser Arbeit soll in dieselbe Richtung gehen und ebenfalls so interessant und anregend gestaltet werden.

3.1.3 Buch 3: „Programmieren lernen: Eine grundlegende Einführung in Java“

Das Buch „Programmieren lernen: Eine grundlegende Einführung mit Java“ [Pepp07] von Peter Peppert bezeichnet sich selbst als Einführungsbuch in die Programmierung und verwendet dafür die Sprache Java. Es werden Programmierkonzepte ausführlich erklärt, und der Umfang bleibt mit 485 Seiten dennoch überschaubar.

a) Aufbau und Gliederung

Aufbau der Gliederung: Das Inhaltsverzeichnis ist zum größten Teil wie bei einem normalen Fachbuch geschrieben, es werden die Fachbegriffe als Kapitelüberschriften verwendet. Jedoch sind manche Überschriften auch als Synonyme zu verstehen, denn diese Bezeichnungen sind teilweise nicht treffend gewählt, sie verwirren mehr, als dass sich der/die LeserIn darunter etwas vorstellen kann. Als Beispiel sei hier folgendes Kapitel genannt: „Des einen Vergangenheit ist des anderen Zukunft“. Dabei geht es um die Polymorphie. Der Bezug zum Titel ist im gesamten Kapitel zu vermissen. Dadurch kann der/die interessierte LeserIn verunsichert werden, wenn kein Zusammenhang zwischen der Überschrift und dem Inhalt hergestellt werden kann. Die wahrscheinlich ursprünglich geplante Eselsbrücke kann so nicht gebildet werden, und jene Mischung zwischen Fachbegriffen und „Synonymen“ lässt eine Einheitlich-

keit im Inhaltsverzeichnis vermissen.

Transparenz der Gliederung: Wie bereits unter Aufbau der Gliederung geschrieben wurde, ist die Gliederung für den/die LeserIn nicht durchsichtig.

Umfang der Abschnitte: Die Kapitel sind für die junge Zielgruppe dieser Arbeit zu lang, und es ist zu viel neuer Lernstoff auf einmal darin enthalten. Zu sehen in Kapitel 5, Kontrollstrukturen, darin werden Schleifen, break, continue, if/else und switch besprochen. Jedes dieser genannten Themen würde aber aufgrund der Länge auch für ein eigenes Kapitel genügend Inhalt bieten.

Beziehung der Abschnitte zueinander: Die Abschnitte befinden sich dahingehend in einer Beziehung zueinander, dass sie aufeinander aufbauen und zuerst die Grundstrukturen eingeführt werden, bevor anschließend die aufbauenden Themen vorkommen.

Verhältnis von Beispielen, Einführung und Theorie: Im Vergleich zu den anderen analysierten Büchern fällt in diesem Buch der Theorieanteil lange aus und übertönt die Anzahl der Beispiele.

Zusammenfassung am Ende des Kapitels: Es sind keine Zusammenfassungen über ein ganzes Kapitel – weder am Ende des Kapitels noch am Anfang des Buches – zu finden.

b) Zielgruppe

Vorwissen und Erfahrung der Lesenden: Die gewählten Beispiele wären ungeeignet für eine jüngere Zielgruppe, da zu komplizierte Formeln im Buch vorkommen (siehe S.76 Übung 5.3 oder Differenzieren auf S. 155). Meist reicht ein kleiner Teil aus, der nicht verstanden wird, damit die Lernenden gedanklich daran hängen bleiben und immer wieder versuchen, diesen Teil zu verstehen, sodass sie sich sich nicht mit dem neuen Inhalt beschäftigen können. Als eines der Beispiele, die zu kompliziert sind, sei an dieser Stelle auch noch die Berechnung des Binominialkoeffizienten auf Seite 90 erwähnt, welcher in den Mathematikstunden an Schulen erst in spä-

teren Jahrgängen durchgenommen wird.

Angesprochene Altersgruppe: Peter Peppert unterrichtet an der Technischen Universität Berlin, wo das Buch aus einer zweisemestrigen Vorlesung entstand, die vor allem für Wirtschaftsingenieure und Elektrotechniker als Einführung in die Informatik gedacht war. Bei der Zielgruppe handelt es sich demnach um Studenten, genauer um die Teilgruppe, die Informatik nicht als Hauptfach belegt haben und schon in anderen Sprachen programmiert hat.

c) Methodische Konzepte

Verwendete Sprache: Dieses Buch liest sich sehr wissenschaftlich und theoretisch; der Inhalt ist schwer zu merken, da viel Text, der meist auch ziemlich lang ist, verwendet wird.

Spielerische Auseinandersetzung: Jene Beispiele, die vorkommen, sind größtenteils stark durch Mathematik geprägt, wie Zinsrechnung, Berechnung der Polygonfläche, Gauß-Elimination, Pivotelement, Differenzieren, Integrieren und Extrapolation. Weitere Beispiele die vorkommen, sind die „Türme von Hanoi“ und die Fibonaccireihe – zur Einführung der Rekursion –, welche allgemein als altbewährte, aber nicht unbedingt interessante und geeignete Beispiele gelten.

Ziele transparent: Teilweise werden die Lernenden auch mit einem neuen Thema konfrontiert, die Notwendigkeit bzw. die Aufgabe des neuen Konstrukts wird jedoch nicht erklärt, wie es in dem Kapitel über Threads zu sehen ist. Hier wäre eine Erklärung, wozu diese überhaupt benötigt werden, wünschenswert.

Präsentation des Inhalts: Ohne großes Drumherum und ohne weitere Ausschweifungen wird neuer Inhalt auf direktem Wege beigebracht. Keine Grafiken, die eine andere Darstellung der Theorie ermöglichen, und auch keine farblichen Hervorhebungen sind in diesem Buch zu vermerken.

Ansatz: Der Autor bietet nur wenige Möglichkeiten das neu Gelesene gleich direkt zu benutzen und selbst zu programmieren. Die gezeigten Lösungsbeispiele können

zwar nachprogrammiert werden, dienen aber eigentlich mehr zum Lesen und Nachvollziehen des Codes als dem selbstständigen Lösen (siehe Abbildung 7). Die methodische Vielfalt bezüglich der Beispiele ist eher gering, auch Wiederholungen werden nicht als didaktisches Mittel eingesetzt.

Programm 6.1 Binomialfunktion

Für Lottospieler ist die Frage interessant, wie viele Möglichkeiten es gibt, aus n gegebenen Elementen k Elemente auszuwählen. Diese Anzahl wird durch die sog. Binomialfunktion „n über k“ ausgerechnet. Sie ist definiert als $\frac{n!}{(n-k)!\cdot k!}$, wobei mit $n!$ die sog. Fakultätsfunktion $1\cdot 2\cdot 3\cdot \cdots \cdot n$ bezeichnet wird. Für die Binomialfunktion gelten folgende Gesetze

$$\binom{n}{0} = 1, \quad \binom{n}{n} = 1, \quad \binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k} \quad \textit{für } n > k > 0$$

Das lässt sich unmittelbar in ein JAVA-Programm übertragen.

```
int binom ( int n, int k ) {
  // ASSERT n ≥ k
  if ( k == 0 | k == n ) {
    return 1;
  } else {
    return binom(n-1, k-1) + binom(n-1, k);
  }//if
}//binom
```

In diesem Programm benutzen wir erstmals ein Dokumentationsmittel, das uns noch viel nützen wird. In einer *Zusicherung* (engl.: *assertion*) setzen wir zusätzliche Einschränkungen für die Parameter fest, die für das Funktionieren der Methode notwendig sind. (Wir gehen im nächsten Kapitel genauer auf Assertions ein.)

Abb. 7: Übungsbeispiel aus „Programmieren lernen: Eine grundlegende Einführung in Java“

d) Lehrziele

Zuerst werden Objekte und Klassen vorgestellt, danach kommen Typen, Variablen und Methoden an die Reihe, noch bevor die verschiedenen Kontrollstrukturen erklärt werden. Suchen und Sortieren heißt ein Kapitel und widmet sich den diversen Sortieralgorithmen, wie Quicksort, Mergesort, Heapsort, Bucket Sort, Selection Sort und Insertion Sort. Diese Algorithmen sind für Studenten, nicht jedoch für 16-jährige oder jüngere ProgrammieranfängerInnen von Interesse.

Vererbung ist ein Bereich, aber auch weiterführende Konzepte der OOP werden durchgenommen. Dazu zählen abstrakte Klassen, Interfaces, Packages und Generi-

zität. Datenstrukturen wie Referenzen, Listen, Bäume und Graphen erhalten alle ihr eigenes Kapitel und werden mit 87 Seiten ziemlich genau vorgestellt. Den Bereichen Ausnahmebehandlung, Ein- und Ausgabe, Grafik in Java, GUI und Threads wird jeweils ein eigenes Kapitel gewidmet.

Das Buch soll nicht nur das Programmieren, sondern auch andere Grundlagen der Informatik vermitteln, weshalb Bäume, Pivotelement, Interpolation, Differenzieren und Integrieren ebenfalls Teile davon sind.

Gegen Ende des Buches schreibt der Autor einen Ausblick auf Java-Gebiete, welche jedoch zur weiterführenden Programmierung zählen.

e) Fazit

Dieses Buch ist durch die vorkommenden Vergleiche mit der imperativen Programmierung sowie durch andere Passagen für AnfängerInnen ungeeignet. Es werden zu schnell neue Themen eingeführt und diese durch eine zu geringe Anzahl an verschiedenen Methoden erklärt.

3.2 Bücher mit speziellen didaktischen Java-Umgebungen

Da die im vorigen Kapitel untersuchten Bücher nicht geeignet sind für die Zielgruppe dieser Arbeit, werden nun Bücher untersucht, die mit speziellen didaktischen Umgebungen versuchen, den LeserInnen das Programmieren mit Java näherzubringen.

Auch mit anderen Ansätzen als den LEGO-Robotern wird in diversen Büchern versucht, die Java-Programmierung einem Neuling beizubringen, z.B. mit dem Java-Hamster-Modell in „Objektorientierte Programmierung spielend gelernt mit dem Java-Hamster-Modell“ von Boles, oder mit Greenfoot in „Einführung in Java mit Greenfoot“ von Kölling, oder mittels BlueJ in „Java lernen mit BlueJ: Eine Einführung in die objektorientierte Programmierung“ von Barnes. Die genannten Bücher werden nun genauer betrachtet.

3.2.1 Buch 4: „Objektorientierte Programmierung spielend gelernt mit dem Java-Hamster-Modell“

Dr.-Ing. Dietrich Boles ist seit 1996 zuständig für die Programmierausbildung der Informatikstudenten an der Universität Oldenburg und hat gemeinsam mit Dr. Cornelia Boles das Buch „Objektorientierte Programmierung spielend gelernt mit dem Java-Hamster-Modell“ [Bole10] verfasst. Es ist der zweite Band der Java-Hamster-Bücher auf den ersten Band „Programmieren spielend gelernt mit Java-Hamster-Modell“ von 2004 aufbaut. Das erste Buch beschäftigt sich mit der imperativen Programmierung und es ist im gesamten Buch immer nur von einem Hamster zu einem Zeitpunkt die Rede. Der Autor empfiehlt den ProgrammieranfängerInnen, vor diesem objektorientiertem Buch auch den Band 1 durchzuarbeiten. Zu dieser Reihe gehört ebenso der dritte Band „Parallele Programmierung spielend gelernt mit dem Java-Hamster-Modell – Programmierung mit Java-Threads“, welcher 2008 erschienen ist.

a) Aufbau und Gliederung

Aufbau der Gliederung: Die Gliederung ist gut strukturiert, Themen sind aufeinander aufbauend und nach den theoretischen Konzepten, die in diesem Buch erlernt werden, erstellt.

Transparenz der Gliederung: Auch die Transparenz der Gliederung ist im gesamten Buch gegeben. Die Überschriften werden nach dem jeweiligen Programmierkonzept benannt, und es ist leicht, ein bestimmtes Thema zu finden.

Umfang der Abschnitte: Für die Altersgruppe der Studenten, die eine längere Konzentrationsspanne besitzen, ist es sicherlich gut geeignet, um im Selbststudium die Inhalte zu erarbeiten. Für die jüngere Zielgruppe sind aber die einzelnen Kapitel zu lang.

Beziehung der Abschnitte zueinander: An unterschiedlichen Stellen wird auf andere Kapitel verwiesen, wenn es der Inhalt erfordert.

Verhältnis von Beispielen, Einführung und Theorie: Beispiele und Theorie wechseln sich ständig ab, was einen angenehmen und verständlichen Lesefluss ergibt. Beispiele, in denen eine Angabe vorhanden ist, und nicht gleich die gesamte codierte Lösung vorzufinden ist, sind aber eher selten.

Zusammenfassung am Ende des Kapitels: Es gibt keine Zusammenfassung am Ende der Kapitel, jedoch wird zu Beginn jedes Abschnitts erklärt, worum es darin gehen wird.

b) Zielgruppe

Vorwissen und Erfahrung der Lesenden: Folgendes Zitat zeigt, wie viele Begriffe teilweise in einem Satz vorkommen, die im Folgenden nicht erklärt werden: „Sie können sich den Hamster quasi als einen virtuellen Prozessor vorstellen, der im Gegensatz zu realen Computer-Prozessoren (zunächst) keine arithmetischen und logischen Operationen ausführen kann, sondern in der Lage ist, mit einem kleinen Grundvorrat an Befehlen ein Hamster-Territorium zu ‚erkunden'." [Bole10] Es kann nicht davon ausgegangen werden, dass die Altersgruppe der über 12-Jährigen weiß, was ein virtueller Prozessor ist, und dass arithmetische und logische Operationen allen LeserInnen bekannt sind. Genau solche Ausdrücke sind es, die nicht erklärt werden und für die Zielgruppe zu schwierig wären, weshalb dieses Buch eher für Studenten und ältere Schüler geeignet ist.

Angesprochene Altersgruppe: Geprägt durch die Tätigkeit des Autors an der Universität Oldenburg, ist dieses Buch vor allem für die Altersgruppe der Studenten geeignet. Den jüngeren Jahrgängen wäre der Schreibstil zu trocken und zwischen den Beispielen die Theorie-Abschnitte zu lange.

c) Methodische Konzepte

Verwendete Sprache: Das Buch erinnert sehr an Studienunterlagen einer Vorlesung, es erscheint eher trocken und verwendet außer den Hamsterbildern keine weiteren Grafiken. Für die jüngere Zielgruppe sind die Begrifflichkeiten, die verwendet

werden, zu kompliziert.

Folgender Abschnitt aus dem Buch verwendet einen Begriff, welcher zuvor nicht eingeführt wurde und auch im direkten Anschluss daran nicht erklärt wurde: „Ein Aufruf des neuen Befehls wird Prozeduraufruf genannt. Ein Prozeduraufruf entspricht syntaktisch dem Aufruf eines der vier Grundbefehle des Hamsters.“ Das Wort syntaktisch ist jener Begriff, der zwar verwendet, aber nicht erklärt wird. Diese Methode, Begriffe zu verwenden, ohne sie bereits eingeführt zu haben, bzw. sie erst im Anschluss zu erklären, verwirrt die jungen LeserInnen und trägt dazu bei, dass sie sich auf diesen Begriff konzentrieren und dem eigentlich wichtigen Inhalt nicht mehr folgen kann.

Spielerische Auseinandersetzung: Die spielerische Auseinandersetzung ist durch die Verwendung des Java-Hamster-Modells gegeben. Auch wenn die Herausforderungen, betrachtet man andere gleichartige Modelle, durchaus noch spielerischer gestaltet sein könnten.

Ziele transparent: Die Ziele sind durch die Überschriften der entsprechenden Abschnitte ersichtlich, weitere Erklärungen, z.B. anhand von Praxisbeispielen, gibt es aber nicht.

Präsentation des Inhalts: Viele Beispiele sind vorhanden, jedoch erst am Ende des jeweiligen Kapitels, an dem der/die LeserIn schon viel neue Theorie erarbeitet hat und noch keine Möglichkeit hatte, diese auch zu üben. Wenn die Lernenden den neu gelesenen Inhalt so lange im Gedächtnis behalten können, bevor sie ihn das erste Mal anwenden müssen, stellt dies keinen Nachteil dar. Viele werden sich damit jedoch schwer tun.

Ansatz: Die im Buch gezeigten Codeausschnitte sind gut geeignet, um sie zu lesen, jedoch weniger sinnvoll zum selbstständigen Programmieren und Problemlösen. Es gibt keine Aufgabenstellung, stattdessen befinden sich die Erklärung des Beispiels und die Lösung im selben Text (sieht Bild 6).

Kapitel eins erklärt das imperative Hamster-Modell, und in den folgenden Kapiteln

werden imperative Konzepte auf die OOP umgemünzt. Es wird also nicht mit der Objektorientierung, nach dem Ansatz Object-First begonnen, sondern – aufbauend auf den prozeduralen Konzepten – die Objektorientierung gelehrt. Diese Reihenfolge ist laut [Chen04] einfacher und kostet weniger Zeit, als wäre es umgekehrt.

Zum Vertiefen der Theorie wird die gängige „Text-Beispiel“-Methode verwendet, keine anderen didaktischen Maßnahmen werden eingesetzt.

d) Lehrziele

Beginnend mit einer kurzen Einführung in die imperativen Konzepte wie Variablen, Prozeduren, if-Anweisungen und Schleifen, geht es weiter über Rekursion bis hin zu den objektorientierten Konzepten wie Objekte und Klassen. Für jene, die bereits das Buch „Programmieren spielend gelernt mit dem Java-Hamster-Modell“ gelesen haben, stellt dieses Kapitel nur eine Wiederholung dar.

Anschließend werden Objekte und Klassen genauer erarbeitet sowie Arrays, Ein- und Ausgabe, Polymorphie, dynamisches Binden, abstrakte Klassen, Interfaces, Exceptions, Zugriffsrechte, Pakete und Generics eingeführt.

Was in der Java-Programmierung gang und gäbe ist, nämlich jede neue Klasse in einer eigenen Datei zu speichern, wird in diesem Buch zwar auch gezeigt, jedoch nicht von Anfang an gelebt. Gleich das erste Beispiel beinhaltet mehrere Klassen, wobei alle Klassen in einer Datei gespeichert werden.

e) Fazit

Das Einzige was dieses Buch abwechslungsreich und interessant macht, ist das Java-Hamster-Modell, denn leider werden keine sonstigen didaktischen Methoden verwendet, die Vielfältigkeit der Didaktik wird durchgehend vermisst. In Summe lässt sich also sagen, dass die Java-Hamster eine nette Methode zur Programmierlehre sind, dieses Buch aber noch besser umgesetzt werden hätte können.

Im folgenden Beispiel wird innerhalb der Klasse `SammelHamster` ein Klassenattribut `gemeinsamGesammelteKoerner` und ein Instanzattribut `alleineGesammelteKoerner` definiert:

```
class SammelHamster extends Hamster {

    int alleineGesammelteKoerner;              // Instanzattribut

    static int gemeinsamGesammelteKoerner = 0; // Klassenattribut

    SammelHamster(int r, int s, int b, int k) {
        this.init(r, s, b, k);
        this.alleineGesammelteKoerner = 0;
        // this.gemeinsamGesammelteKoerner = 0;
    }

    void sammle() {
        while (this.kornDa()) {
            this.nimm();
            this.alleineGesammelteKoerner =
                this.alleineGesammelteKoerner + 1;
            this.gemeinsamGesammelteKoerner =
                this.gemeinsamGesammelteKoerner + 1;
        }
    }

    void laufeUndSammle() {
        while (this.vornFrei()) {
            this.vor();
            this.sammle();
        }
    }
}

void main() {
    SammelHamster paul = new SammelHamster(0, 0,
            Hamster.OST, 0);
    paul.laufeUndSammle();

    SammelHamster willi = new SammelHamster(1, 0,
            Hamster.OST, 0);
    willi.laufeUndSammle();
```

[3] Aus implementierungstechnischen Gründen müssen Sie im Hamster-Simulator Klassen, die Klassenattribute definieren, und Hauptprogramme immer in separaten Dateien abspeichern (vergleiche Kapitel 5.9).

Abb. 8: Übungsbeispiel aus Buch 4

3.2.2 Buch 5: „Einführung in Java mit Greenfoot“

Greenfoot wurde eigens zur spielerischen Vermittlung von objektorientierter Programmierung entwickelt. Einer der Entwickler war Michael Kölling, der im Jahr 2010 das Buch „Einführung in Java mit Greenfoot“ [Köll10] geschrieben hat. Es handelt

sich dabei um ein 240-seitiges didaktisch gestaltetes Buch, das step by step die Grundlagen der Java-Programmierung erläutert.

a) Aufbau und Gliederung

Aufbau der Gliederung: Zu Beginn jedes Kapitels werden die Lernziele, sowohl die Themen, die darin behandelt werden, als auch die Konzepte, aufgezählt. Neben dem ansprechendem Design des Buches hat jeder Abschnitt seine eigene Farbe, wodurch sich leichter und auch schneller etwas finden lässt.

Alle Kapitel sind kurz gehalten und führen etappenweise die neuen Konzepte, über mehrere Kapitel hinweg, ein. Es wird nicht, wie weitläufig üblich, ein Thema als Ganzes und vollständig in einem Kapitel erläutert. Stattdessen wird damit begonnen, wenn es benötigt wird, und in einem der später folgenden Abschnitte weiter geführt. Durch diesen Spiralansatz werden ständig die Inhalte wiederholt, und der/die Lesende wird nicht von der neuen Substanz erdrückt. So sind z.B. die Schleifen in den unterschiedlichsten Kapiteln erklärt. Zuerst werden die while-Schleife in Kapitel 5 und die for-each-Schleife in Kapitel 6 erklärt, bevor anschließend im Kapitel 7 die for-Schleife vorgestellt wird. Dieses scheinbare Durcheinander der Inhalte stört beim Erarbeiten keineswegs, da die notwendige Theorie an richtiger Stelle zur Verfügung gestellt wird. Hinter dieser Art der Darstellung steht ein gut durchdachtes didaktisches Konzept, an dem die jahrelange Arbeit deutlich zu erkennen ist.

Transparenz der Gliederung: Das Inhaltsverzeichnis lässt nicht zwangsläufig auf die jeweiligen Inhalte schließen, sondern listet die vorkommenden Beispiele auf. Erst durch die Unterkapitel wird teilweise klar, worum es hierbei geht und was die Lernziele sind.

Umfang der Abschnitte: Die Kapitel sind kurz und sehr übersichtlich, die Inhalte eines Abschnittes sind leicht für die Zielgruppe dieser Arbeit zu erarbeiten und stellen keine zu große Herausforderung dar.

Beziehung der Abschnitte zueinander: Gewisse Kapitel bauen auf den vorherigen direkt auf, und um die neuen Aufgaben darin lösen zu können, müssen die LeserIn-

nen zuvor das vorhergehende Thema erarbeitet haben: z.B. muss für Kapitel drei das zweite Kapitel bereits ausgearbeitet worden sein.

Verhältnis von Beispielen, Einführung und Theorie: Das Verhältnis zwischen Beispielen und Theorie ist ausgewogen, denn trotz genügend vorhandener Beispielen, kommt auch die Theorie nicht zu kurz, und auch Wichtiges wird auf jeden Fall besprochen. Die Übungen selbst sind sehr kurz gehalten und so mit schnellem Erfolg zu erledigen.

Zusammenfassung am Ende des Kapitels: Nach jedem Kapitel gibt es einen Bereich für die Zusammenfassung der Konzepte, welcher die wichtigsten Grundkonzepte kurz und bündig erklärt. Auch an den Rändern des Buches in den Kapiteln sind immer wieder kleine Notizen, welche ein Konzept mit einem Satz oder ein paar einfachen Sätzen erklären.

b) Zielgruppe

Vorwissen und Erfahrung der Lesenden: Es ist kein Vorwissen bei den LeserInnen dieses Buches erforderlich.

Angesprochene Altersgruppe: Es ist sowohl für SchülerInnen als auch Studierende gleichermaßen geeignet, denn die langsame, schrittweise Heranführung an neue Themen ist so gestaltet, dass sogar AnfängerInnen verstehen, worum es geht, aber auch erfahrenere Studenten in späteren Kapiteln Neues erfahren können. Die farbliche Gestaltung lässt dieses Werk bunt erscheinen, wenngleich es nicht chaotisch und unstrukturiert ist. Auch der Schreibstil ist für jüngere LeserInnen durchaus passend.

c) Methodische Konzepte

Verwendete Sprache: Das Buch ist in einer sehr spielerischen Herangehensweise geschrieben und die Theorie wird mehr beiläufig erklärt, dabei aber nicht in den Vordergrund gerückt.

Spielerische Auseinandersetzung: Greenfoot bietet unterschiedliche Szenarien an: Krabben, Ameisen, Vögel, Planetensimulation oder Klaviersimulation. Diese können auch untereinander gemischt, verändert und erweitert werden. Somit sind genügend Beispiele vorhanden, dass alle ein Szenario finden sollten, welches sie anspricht.

„Du musst jedoch nicht unbedingt jede Aufgabe bearbeiten und lösen. Vielmehr kannst du selbst entscheiden, was dir Spaß macht, und gerne auch deine eigenen Szenarien gestalten.“ [Köll10] Es muss nicht zwangsläufig jedes Beispiel aus dem Buch von den Lernenden gelöst werden um dem Inhalt folgen zu können, es ist immer eine Auswahl vorhanden, die jeder/jede Lesende selbst treffen kann.

Ziele transparent: Durch die Zusammenfassungen, die kleinen Randnotizen und vor allem die Lernziele zu Beginn des jeweiligen Abschnittes sind die Ziele ständig erkennbar.

Präsentation des Inhalts: Der Inhalt wird farbig, abwechslungsreich und kurzweilig präsentiert. Durch die Übungen, welche im Laufe der einzelnen Kapitel vorgestellt werden, wird Learning by Doing gefördert.

Ansatz: Der Autor preist Greenfoot als skalierbar an, da es sowohl für AnfängerInnen interessant und verständlich, als auch für Programmierexperten verwendbar ist. Schnell kann etwas visuell dargestellt werden und der/die ProgrammieranfängerIn muss sich nur Gedanken um die Logik des Programms machen, weniger um die Darstellung, welche von Greenfoot übernommen wird.

Im Anhang D befinden sich alle im Buch vorkommenden grundlegenden Konstrukte, die in Java geschrieben wurden. Diese nochmalige Zusammenfassung liefert einen guten Überblick für jene, die bereits Programmiererfahrungen in anderen Sprachen haben oder die etwas nachlesen wollen.

Das Register ganz am Schluss liefert eine gute Möglichkeit zum Auffinden spezieller Themen. Da das Inhaltsverzeichnis sich eher an den Greenfoot-Inhalten als an den Programmierkonzepten orientiert, bietet das Register eine zusätzliche Suchmöglichkeit.

d) Lehrziele

Objekte und Klassen, Konstruktoren, Objektvariablen, Methoden mit Parametern und Rückgabewert, Klassendiagramme, if/else, Schleifen, Java-Klassenbibliotheken, Zufallszahlen, Kommentare, Abstraktion, Arrays, Collections, Listen und Casting: Das sind die Inhalte, die in diesem Buch erklärt werden.

Die Greenfoot API besteht aus 5 Klassen. Diese werden im Anhang B mit den jeweiligen Methoden und Aufgaben kurz vorgestellt. Auch die Installation von Greenfoot wird sehr einfach für Novizen erklärt und stellt ein eigenes Kapitel im Anhang dar.

Ergänzend werden Sound- und Bilddateiformate, Dateigrößen, Simulationen und Experimente eingeführt. Dem Autor ist es wichtig, mit diesen Inhalten den Spaß am Programmieren zu vermitteln.

e) Fazit

Keines der zuvor analysierten Bücher bietet eine so gute farbliche und auch didaktische Gestaltung wie dieses Buch. Dies ist ein Buch, das durchwegs für die Altersgruppe der über 12-Jährigen bis 16-Jährigen geeignet ist.

3.2.3 Buch 6: „Java lernen mit BlueJ: Eine Einführung in die objektorientierte Programmierung“

Bei BlueJ handelt es sich um eine interaktive, speziell für junge Leute entwickelte Lern-Entwicklungsumgebung, welche den Vorteil bietet, dass sie schnell einsetzbar ist und keine lange Einführung benötigt. In „Java lernen mit BlueJ: Eine Einführung in die objektorientierte Programmierung“ [Barn12] ist die Rede von 20 Minuten, nach denen man diese Umgebung kompetent benutzen kann. Ein weiterer Vorteil ist die Visualisierung von Klassen, Objekten und dergleichen.

BlueJ wurde speziell für ProgrammieranfängerInnen entwickelt. „Much functionality present in other environments, but not needed in first year courses, is not included in

BlueJ. This makes BlueJ less suitable for professional development, but represents a great win for introductory teaching.“ [Köll01]

a) Aufbau und Gliederung

Aufbau der Gliederung: „Java lernen mit BlueJ“ ist ein didaktisch gut aufbereitetes Buch, welches jedes Thema in kleine Teilgebiete zerlegt und somit das jeweils neu zu erlernende Stoffgebiet möglichst gering hält.

Transparenz der Gliederung: Die Gliederung ist übersichtlich und lässt vom Inhaltsverzeichnis sehr einfach auf den diesbezüglichen Inhalt schließen. Im Vorwort finden sich kurze Erklärungen der einzelnen Kapitel wieder.

Umfang der Abschnitte: Der Umfang der Unterkapitel ist für die Altersgruppe dieser Arbeit gut geeignet. Mit durchschnittlich rund vier Seiten samt Illustrationen und abgebildeten Beispielen ist das eine angenehme Länge für die jungen Lernenden.

Beziehung der Abschnitte zueinander: Die in sich abgeschlossenen Abschnitte bauen aufeinander auf, und es wird Bezug auf andere Kapitel genommen, sofern dies zur besseren Verständlichkeit beiträgt.

Verhältnis von Beispielen, Einführung und Theorie: Ein ausgewogenes Verhältnis zwischen den Beispielen und der Theorie ist in jedem Kapitel vorzufinden.

Zusammenfassung am Ende des Kapitels: Es lässt sich angenehm lesen, und am Ende jedes Kapitels findet sich eine kurze Zusammenfassung wieder. Der eingerahmte Bereich am Schluss jedes Themas (siehe Abbildung 9), welcher die neu eingeführten Begriffe zusammenfasst, dient sehr gut der Übersichtlichkeit und lässt den Lernenden schnell einen Überblick über den jeweiligen Inhalt gewinnen.

b) Zielgruppe

Vorwissen und Erfahrung der Lesenden: Es ist kein Vorwissen der Lernenden not-

wendig.

Angesprochene Altersgruppe: Sowohl jüngere Programmier-Interessierte im Alter von mindestens 14 Jahren als auch Studenten sind geeignet, um mit diesem Buch Java-Programmieren zu erlernen. Der Inhalt bietet sowohl für komplette Neulinge als auch für ProgrammiererInnen anderer Sprachen viel Interessantes und Lesenswertes.

c) Methodische Konzepte

Verwendete Sprache: Sprachlich ist dieses Buch für die jüngere Zielgruppe geeignet, da keine komplexen theoretischen Begriffe verwendet werden, welche mit hoher Wahrscheinlichkeit noch nicht bekannt sein dürften. Am Seitenrand werden an den passenden Stellen wichtige Notizen gemacht und farblich hervorgehoben, ohne jedoch störend zu wirken.

Spielerische Auseinandersetzung: Eine spielerische Auseinandersetzung ist gegeben, wenn auch nicht so stark, wie es z.B. in Buch 5 der Fall ist.

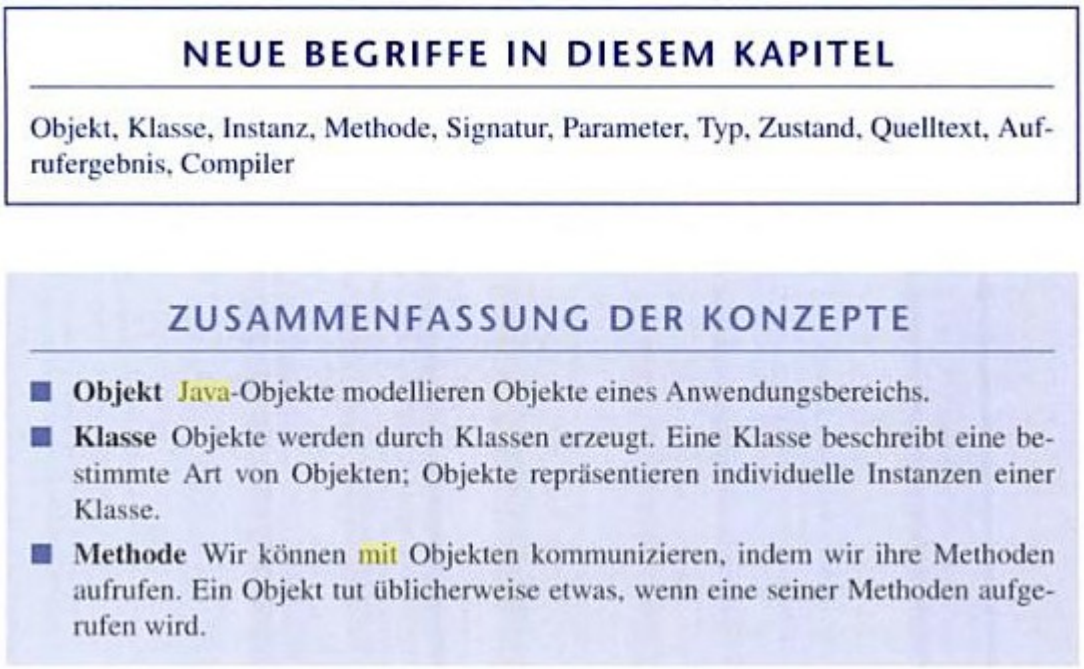
Ziele transparent: Die Ziele sind transparent, da sie sowohl am Anfang der Kapitel extra genannt werden, als auch zu Beginn des Buches erklärt werden.

Präsentation des Inhalts: Begriffe werden eingeführt und immer erklärt, wenn sie das erste Mal verwendet werden. Bilder werden an entsprechend passenden Stellen verwendet und zeigen, wie gewünschte Aufgaben in der Entwicklungsumgebung auszusehen haben oder wie das Ergebnis gewünscht ist. Unterschiedliche didaktische Methoden würden das Buch noch schön abrunden.

Ansatz: Alle nötigen Schritte, um zu einer Lösung zu kommen, werden sehr ausführlich erklärt; auch wie man dorthin gelangt, wird detailliert geschildert, egal um welches Thema es sich gerade handelt. Beispiele zum selbstständigen Üben sind im direkten Anschluss an jedes neu kennengelernte Konzept zu finden, nicht erst am Ende eines langen Kapitels.

Object-First ist die Devise dieses Buches, und im Gegensatz zu vielen anderen Lernunterlagen, die nur allzu oft davon erzählen und dann letztendlich doch wieder mit prozeduralen Kontrollstrukturen beginnen, wurde dieser Ansatz hier hundertprozentig durchgezogen.

Der Beispielcode ist sehr gut lesbar und verwendet immer wieder Pseudocode, der zu diesem Zeitpunkt nicht wichtige Abschnitte darstellt und nicht mit unnötigen Programmzeilen verwirrt.



NEUE BEGRIFFE IN DIESEM KAPITEL

Objekt, Klasse, Instanz, Methode, Signatur, Parameter, Typ, Zustand, Quelltext, Aufrufergebnis, Compiler

ZUSAMMENFASSUNG DER KONZEPTE

- **Objekt** Java-Objekte modellieren Objekte eines Anwendungsbereichs.
- **Klasse** Objekte werden durch Klassen erzeugt. Eine Klasse beschreibt eine bestimmte Art von Objekten; Objekte repräsentieren individuelle Instanzen einer Klasse.
- **Methode** Wir können mit Objekten kommunizieren, indem wir ihre Methoden aufrufen. Ein Objekt tut üblicherweise etwas, wenn eine seiner Methoden aufgerufen wird.

Abb. 9: Zusammenfassung eines Kapitels in Buch 5

d) Lehrziele

Der Autor stellt fest, dass man nicht von Lernunterlagen fordern kann, sich auf die wichtigsten Konzepte zu konzentrieren, aber gleichzeitig Vollständigkeit verlangen möchte. Demnach kommen in diesem Buch auch nicht alle möglichen Konzepte der Java-Programmierung vor, sondern es finden nur die wichtigsten Grundlagen darin Platz.

Objekte und Klassen werden ausführlich erklärt, und es wird darauf Wert gelegt, dass die Konzepte wirklich verständlich dargestellt werden. Folgende Inhalte kommen vor: Objekte und Klassen, Klassendefinition und Objektinteraktion, Objektsamm-

lungen, Nutzen von Bibliotheksklassen, Klassenentwurf, vom Modultest bis zum Debugger, Entwurf, Vererbung, Abstraktionstechniken, Grafische Benutzeroberflächen und Fehlerbehandlung.

e) Fazit

Dieses BlueJ-Werk bietet mehr Inhalt als Buch 5 („Einführung in Java mit Greenfoot“), ist im Vergleich dazu jedoch weniger spielerisch gestaltet als dieses.

Besonders gelungen an dem Buch sind die kurzen Zusammenfassungen am Ende jedes Kapitels, ähnlich wie es auch bei Greenfoot der Fall ist, jedoch fallen diese hier – dem Inhalt entsprechend – etwas länger aus.

Bis auf wenige Bereiche, wie etwa die grafische Benutzeroberfläche, die Abstraktionstechniken und die Objektsammlungen, sollten alle Inhalte im Konzept des zu erstellenden Buches vorkommen. Der Schreibstil ist für die Altersgruppe dieser Arbeit geeignet, die Länge der Texte sollte jedoch noch etwas kürzer ausfallen.

Die optische Gestaltung ist sehr gut gelungen und lässt ein spannendes Buch erwarten, welches es durchaus auch darstellt.

3.3 Bücher über LEGO MINDSTORMS

Neben reinen Java-Einsteiger-Büchern und solchen, die Java-Programmieren mit anderen Umgebungen lehren, existieren unzählige Bücher über LEGO-MINDSTORMS-Roboter und deren Programmierung. Genannt seien an dieser Stelle jene Werke, die sich mit Java und LEGO-Robotern beschäftigen. Jene Bücher, die andere API's zur Programmierung der Roboter behandeln, wie z.B. NXC, oder auch solche, die nur die grafische Oberfläche betrachten, werden nicht analysiert. Ziel soll die Grundlagenkenntnis von Java sein und nicht das Erlernen der grafischen Oberfläche, da diese bereits als Voraussetzung betrachtet wird.

3.3.1 Buch 7: „Programmierung mit LEGO MINDSTORMS NXT“

„Das Buch ‚Programmierung mit LEGO MINDSTORMS NXT' [Bern10] bietet einen Einstieg in die Programmierung anhand des Beispiels von LEGO MINDSTORMS NXT-Robotern und richtet sich an SchülerInnen der Oberstufe und an Studierende in den ersten Semestern. Ausgehend von wichtigen Grundkenntnissen der Informatik wie Rechneraufbau und Programmentwurf wird schrittweise in die Programmierung komplexer Softwaresysteme eingeführt.“ [Bern10] Durch diese Aussage der Autoren grenzen sie sehr gut den Inhalt und das Zielpublikum dieses Buches ein.

In diesem Buch von Karsten Berns und Daniel Schmidt werden Robotersysteme, Entwurfsmethodiken und Algorithmen ebenso vorgestellt wie Grundlagen der Programmierung – sowohl die Entwicklung in der grafischen Oberfläche von LEGO als auch mittels Java.

a) Aufbau und Gliederung

Aufbau der Gliederung: Das Glossar am Ende des Buches ist eine praktische Unterstützung und dient als Hilfe und Erklärung für Begriffe und Definitionen, die dem/der LeserIn nicht bekannt sind. Es geht von der Einleitung über die Robotik über die Grundlagen der Informatik bis zur grafischen und textuellen Programmierung der LEGO-Roboter. Systematisch wird ein Thema nach dem anderen erarbeitet.

Transparenz der Gliederung: Die Gliederung ist transparent, das Inhaltsverzeichnis benennt genau die Konzepte und Themen, die in den Kapiteln nähergebracht werden.

Umfang der Abschnitte: Die Kapitel sind sehr übersichtlich und auch im Umfang eher kurz und somit auch gut geeignet für die junge Altersgruppe der 12- bis 16-Jährigen. Jedoch wird stellenweise zu wenig genau erklärt, um dieses Buch auch tatsächlich der Altersgruppe dieser Arbeit empfehlen zu können.
Beziehung der Abschnitte zueinander: Wie bereits zuvor geschrieben, sind die Kapitel aufeinander aufbauend, die Beziehung zueinander ist dahingehend gegeben, weil der/die Lernende, um eine spätere Aufgabe erfüllen zu können, frühere Kapitel

gelesen und erarbeitet haben muss.

Verhältnis von Beispielen, Einführung und Theorie: Es ist eine ausgewogene Relation zwischen Beispielen und Theorie vor. Genügend Beispiele lockern die Theorie auf.

Zusammenfassung am Ende des Kapitels: Es gibt keine Zusammenfassung am Ende der Kapitel. Auch an anderen Stellen des Buches sind keine Zusammenfassungen vorzufinden.

b) Zielgruppe

Vorwissen und Erfahrung der Lesenden: Es ist nicht unbedingt Programmiervorwissen für die Lernenden erforderlich, um den Inhalt wirklich zu verstehen; ohne parallel woanders nachlesen zu müssen, wären grundlegende Programmierkenntnisse in einer textuellen Programmiersprache oder – noch besser – in einer objektorientierten Programmiersprache von Vorteil, da die Konzepte theoretisch meist nur oberflächlich erklärt werden.

Angesprochene Altersgruppe: Laut Umschlag eignet sich dieses Buch sowohl für Studenten der ersten Semester als auch für Schüler der Oberstufe. Jüngere Leser werden sich aufgrund der doch sehr theoretischen Schreibweise und der vielen Formeln schwer damit tun.

c) Methodische Konzepte

Verwendete Sprache: Die verwendete Sprache ist durchaus auch für die jüngere Zielgruppe geeignet. Sie ist zwar trocken und sachlich, jedoch nicht zu kompliziert.

Spielerische Auseinandersetzung: Die Auseinandersetzung mit den Inhalten ist weniger spielerisch, dafür aber zielführend und sachlich.

Ziele transparent: Die zu erfüllenden Ziele sind im gesamten Buch gut zu erkennen.

Nie braucht sich der/die LeserIn die Frage zu stellen, wozu das jeweilige Thema erklärt wird.

Präsentation des Inhalts: Grafisch ist das Buch wenig aufregend gestaltet, es ist farblos, und auch sprachlich ist es eher langweilig zu lesen.

Ansatz: Das Buch ist hervorragend zur Vorbereitung auf den Informatikunterricht und zur Durchführung von Informatik- und Robotik-AGs geeignet. Ein Einstieg in ein Studium oder anderweitige Beschäftigung mit dem Programmieren sollen erleichtert werden.

Die Lösung von Beispielen, die ohne Programmierung bewältigt werden können, aber eine Berechnung notwendig machen, wird direkt im Anhang an die Angabe – um 180 Grad rotiert – gedruckt, so dass man das Buch auf den Kopf stellen muss, um die Lösung lesen zu können (siehe Abbildung 10).

Aufgabe 7: Ultraschallmessung

Ein Ultraschallsensor mit einem Öffnungswinkel von $\pm 30°$ erkennt ein Hindernis in einer Entfernung von $s = 2.55m$. Wie lange war das Schallsignal bei einer Lufttemperatur von $30°C$ unterwegs? Wie lang ist der Kreisbogen des Schallkegels in dieser Entfernung?

Zunächst gilt für die Schallgeschwindigkeit (Gleichung 4.5):

$$v_{luft_{30°C}} = 331.4 \cdot \sqrt{(30 + 273)/273} = 349.1m/s$$

Durch Umformung der Gleichung 4.4 nach der Laufzeit erhält man:

$$\Delta t = 2 \cdot s / v_{luft_{30°C}} = 2 \cdot 2.55/349.1 = 0.0146s$$

Der Gesamtumfang eines Kreises mit einem Radius von $r = 255cm$ beträgt

$$U = 2 \cdot r \cdot \pi = 16.0m$$

Die Länge des gesuchten Kreisbogens (Öffnungswinkel 60°) entspricht einem Sechstel des gesamten Kreisumfangs (360°). Demnach ist der Kreisbogen $U/6 = 2.67m$ lang, auf dem sich das Hindernis befinden kann.

Abb. 10: auf den Kopf gestellte Lösungen in Buch 7

Die Beispiele im NXT-G-Bereich sind größere Projekte, jedoch immer in Teilaufgaben zerlegt. Jede Teilaufgabe kann auch für sich alleine erledigt werden; auch wenn dies von den Autoren ursprünglich nicht vorgesehen ist, so ist es doch möglich. Das erste Java-Beispiel ist ein Hello-World-Beispiel womit sich trotz des spannenden Anwendungsgebietes der Roboter dieses Werk in die doch eher klassische Schiene einordnet.

Die Lösungen zu den Beispielen aus dem Buch können von einem Server heruntergeladen werden, dies stellt eine gängige und gute Vorgehensweise dar, denn somit wird das Buch nicht unnötig dick und unübersichtlich. Gut ist auch, dass nicht nur der reine bloße codierte Teil zur Verfügung gestellt wird, sondern auch Tipps zur Lösungsfindung im Anhang vorhanden sind.

d) Lehrziele

Lehrinhalte sind die Grundlagen der Robotik, Roboterprogrammierung und Informatikgrundlagen, das LEGO MINDSTORMS NXT-G System im Allgemeinen und auch Java-Programme für den NXT mit leJOS.

Im Bereich der Grundlagen der Informatik werden folgende Abschnitte behandelt: Programmentwicklung, Modellierung und Abstraktion, Klassen, Methoden, Schleifen, Programmlogik und Compiler sowie Betriebssysteme und Elektronik. Für detaillierte-re Informationen wird vom Autor das Buch „Maximum LEGO – Building Robots with Java Brains“, welches im nächsten Kapitel (Buch 8) analysiert wird, empfohlen.

Das dritte Kapitel, welches die Elemente der Programmierung beinhaltet, behandelt die Konzepte nur theoretisch ohne Beispiele. Der Zusammenhang zwischen Theorie-Kapiteln und späteren Beispielen in anderen Kapiteln muss selbst hergestellt werden, denn im Buch wird diese Verknüpfung nicht erstellt.

e) Fazit

Dieses Buch ist gut für InformatiklehrerInnen oder TrainerInnen geeignet, da es einen guten Überblick über alle Bereiche der LEGO-Roboter bietet und auch viele kleinere Einsteigerbeispiele vorstellt. Jedoch ist es für die Altersgruppe der 12- bis 16-Jährigen nicht lückenlos zu empfehlen, da viele Bereiche nicht soweit erklärt werden, dass sie ohne weitere Unterlagen oder Vorwissen von diesen verstanden werden.

3.3.2 Buch 8: „Maximum LEGO NXT Building Robots with JavaTM Brains Third Edition“

Dieses in englischer Sprache verfasste Buch von [Bagn13] handelt von einer Einführung in die Welt der Robotik, um den LeserInnen zu ermöglichen, eigene originelle Roboterkreationen zu entwerfen. Dazu gehört mitunter auch die Programmierung der Roboter, weshalb dieses Buch hier näher – hinsichtlich der Brauchbarkeit zum Erlernen einer objektorientierten Programmiersprache wie Java – untersucht wird.

a) Aufbau und Gliederung

Aufbau der Gliederung: Die Gliederung ist nach den verschiedenen Roboterbauarten aufgeteilt und beinhaltet nur Hauptkapitel, keine Unterkapitel.

Transparenz der Gliederung: Es ist schon beim Betrachten des Inhaltsverzeichnisses klar, worum es in den einzelnen Kapiteln geht.

Umfang der Abschnitte: Durch die in manchen Kapiteln vorhandenen Bauanleitungen sind diese Kapitel sehr lang. Auch die Theorie-Abschnitte sind eher lang und ergeben so oft recht umfangreiche Passagen.

Beziehung der Abschnitte zueinander: Will der/die Lesende einen Roboter bauen, der mehrere Aufgaben erfüllen soll, so ist zwangsläufig das Wissen mehrerer Abschnitte notwendig. Um die Programmierung der Roboter zu erlernen, gibt es nur zwei Kapitel. Diese beiden Java-Kapitel stehen zueinander in Beziehung, weil sie aufbauend sind. Bei den Roboterkapiteln ist dies aber weniger der Fall, aber auch nicht unbedingt erforderlich.

Verhältnis von Beispielen, Einführung und Theorie: Dieses Buch ist eher Beispiel-orientiert, trotzdem ist genügend Theorie vorhanden.

Zusammenfassung am Ende des Kapitels: Es gibt keine Zusammenfassung am Schluss eines Kapitels.

b) Zielgruppe

Vorwissen und Erfahrung der Lesenden: Wenn der/die LeserIn nicht vertraut ist mit der LEGO-Technologie oder der Java-Programmierung, so wird er/sie – laut Autor – soweit geführt, um trotzdem erfolgreich Roboter bauen zu können. Jedoch stellt sich heraus, dass zumindest eine andere textuelle Programmiersprache, besser noch eine andere objektorientierte Programmiersprache, beherrscht werden sollte.

Angesprochene Altersgruppe: Zum Programmierenlernen ist dieses Buch nicht für reine AnfängerInnen und die entsprechenden Altersgruppe geeignet, da es eine viel größere Selbstständigkeit voraussetzt, als es bei dieser Altersgruppe der Fall ist.

c) Methodische Konzepte

Verwendete Sprache: Das Buch ist in englischer Sprache verfasst, was es der Zielgruppe erschwert, dem Inhalt zu folgen. Nach einer Übersetzung sollte dies aber kein Problem mehr darstellen, da keine komplizierten Sätze oder Ausdrücke verwendet werden.

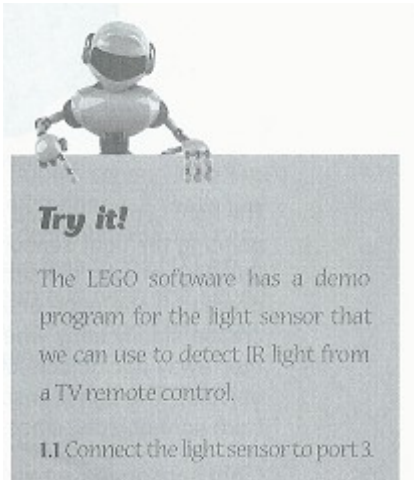


Abb. 11: Try it! Roboter aus Buch 8

Spielerische Auseinandersetzung: Durch die Roboter alleine ist schon eine spielerische Auseinandersetzung gegeben, zusätzlich leitet ein humanoider Roboter mit „Fun Facts“ und „Try it“-Grafik durch das Buch (siehe Abbildung 11)

Ziele transparent: Die angestrebten Ziele werden zu Beginn jedes Kapitels als Aufzählungspunkte übersichtlich und kurz genannt.

Präsentation des Inhalts: Die grafische Gestaltung des Buches ist schlicht, einfach, farblos und auch eher langweilig. In den grau hinterlegten Textbereichen wird auf Notizen und hilfreiche Tipps hingewiesen, was das Auge auf die darin enthaltenen wichtigen Angaben lenkt.

Der gesamte Code für alle im Buch vorhandenen Projekte wird nicht nur darin abgebildet, sondern auch auf der Website[10] zur Verfügung gestellt.

Ansatz: Sehr prägnant wird die Objektorientierte Programmierung eingeführt, nur zwei Kapitel behandeln die Java-Programmierung. Eines davon ist für die absoluten Java-AnfängerInnen geeignet und heißt „Java for primates“, und das andere „One More Cup of Java“ beinhaltet weiterführende Java-Konzepte. In Summe ist der Programmieranteil aber zu gering um einem/einer absoluten ProgrammieranfängerInnen Java beizubringen, ohne jegliche weitere Literatur als Unterstützung.

Das erste Projekt ist ein Hello-World-Programm, wo „Hello World“ am Display des Roboters ausgegeben wird. Das erste Beispiel nach dem Hello-World-Programm ist ein „Fishing Buddy“, dafür werden die Bauanleitung, weitere Informationen und auch der vollständige Code zur Verfügung gestellt. Am Ende dieses Kapitels und auch anderer Kapitel gibt es Zusatzaufgaben, die probiert werden sollten. Gleich die erste Aufgabe handelt von einer kleinen zusätzlichen Routine, die durch zufällige Intervalle den Köder hin- und herbewegen soll. Dies ist für ProgrammieranfängerInnen zu diesem Zeitpunkt ohne weitere Hilfe, die im Buch bis dahin noch nicht gegeben wurde, nicht möglich, denn erst im darauf folgenden Kapitel „Java for primates“ werden Grundlagen der Java-Programmierung durchgenommen.

d) Lehrziele

Hauptaugenmerk liegt nicht auf der Programmierung, vielmehr geht es darum, tolle Roboter bauen und konstruieren zu können. Die Java-Programmierung wird durch-

10 www.variantpress.com

aus auch erklärt, aber bei Weitem nicht so ausführlich wie die Robotik- und Bauabschnitte. Wie bereits beschrieben, sind zwei Kapitel über Java vorhanden: Das erste Kapitel „Java for Primates“ beinhaltet OOP im Allgemeinen, Core Java Language Syntax, Klassen, Interfaces, Methoden, primitive Datentypen, Operatoren und Flusskontrolle. Im zweiten Kapitel „One More Cup of Java“ geht es um Java.lang und java.util, das Event Model, Rekursion, Enumerations und Generics. Dabei handelt es sich aber vielmehr um eine Auflistung der einzelnen Klassen und Methoden mit kurzer Erklärung, die an eine API erinnert, als um eine ausführliche Erläuterung der Konzepte. Diese ist auch vorhanden, kommt aber zu kurz für unerfahrene AnfängerInnen.

Design Patterns zum Bauen der Roboter und spezielle Regeln aus der LEGO-Welt werden neben allgemeinen Konstruktionszielen ebenfalls erörtert.

Jedes Kapitel behandelt einen Roboter bzw. einen Anwendungsfall, dazu gibt es jeweils eine allgemeine Einführung, die technische Erklärung, die dazu passende Bauanleitung, den Programmierteil und weiterführende Tipps, Ideen oder Anwendungsfälle.

Ein Kapitel beschäftigt sich ausschließlich mit den Bauteilen des LEGO NXT-Baukastens.

e) Fazit

Hier wurde ein anderer Stil gewählt als in den bisher analysierten Büchern. In diesem Buch werden eher die technischen Grundlagen und Konzepte erklärt, um es der/dem Lernenden zu ermöglichen, weitere Eigenkreationen zu bauen. Das Programmieren selbst steht nicht im Vordergrund und wird auch in den beiden Kapiteln nicht so ausführlich erklärt, wie es für AnfängerInnen notwendig wäre. Es ist daher nicht für die Altersgruppe geeignet, um sich selbst und ohne weitere Unterlagen die Java-Programmierung beizubringen. Damit dieses Buch für die Zielgruppe geeignet wäre, müsste mehr Grundwissen als aus der grafischen LEGO-Programmierumgebung vorhanden sein.

3.4 Fazit

Die auf den Seiten 27 und 28 aufgezählten Bereiche werden im Folgenden hinsichtlich der Altersgruppe der 12- bis 16-Jährigen bewertet:

- \+ (gut für die entsprechende Zielgruppe umgesetzt)
- ~ (für die Zielgruppe dieser Arbeit nicht geeignet)
- – (eher spärlich bzw. schlecht umgesetzt für die Zielgruppe dieser Arbeit).

In Tabelle 1 werden diese Bewertungen übersichtlich dargestellt.

	1	2	3	4	5	6	7	8
Aufbau und Gliederung								
Aufbau der Gliederung	~	+	~	+	+	+	+	~
Transparenz der Gliederung	-	+	-	+	~	+	+	+
Umfang der Abschnitte	+	~	-	-	~	+	~	-
Beziehung der Abschnitte zueinander	+	+	~	+	~	+	~	~
Verhältnis von Beispielen, Einführung und Theorie	~	+	-	~	+	+	+	~
Zusammenfassung am Ende des Kapitels	+	~	-	~	+	+	-	-
Zielgruppe								
Vorwissen und Erfahrung der Lesenden	-	+	-	~	+	+	~	-
Angesprochene Altersgruppe	~	+	-	~	+	+	-	-
Methodisches Konzept								
Verwendete Sprache	~	+	-	-	+	+	~	-
Spielerische Auseinandersetzung	~	+	-	~	+	~	~	~
Ziele transparent	+	+	~	+	+	+	+	+
Präsentation des Inhalts	~	+	~	~	+	~	~	~
Ansatz	-	+	-	+	+	+	~	-
Lehrziele								
Behandelte Themen	~	~	-	+	+	~	~	-
Verfügbarkeit in Deutsch	+	+	+	+	+	+	+	-

Tab. 1 : Überblick über alle 8 Bücher

Jedes der acht analysierten Bücher ist auf die eine oder andere Art und Weise gut geeignet um Java zu erlernen, jedoch nicht für die Altersgruppe der 12- bis 16-Jährigen. Die Bücher 1, 2 und 3 sind viel zu detailliert. Es kann somit nicht sichergestellt werden, dass die Lernenden dieses Buch in absehbarer Zeit und möglichst selbstständig durcharbeiten können. Viele Bereiche darin sind für den ersten Einstieg in die Java-Programmierung nicht notwendig und können von den jungen Interessierten auch noch zu einem späteren Zeitpunkt erlernt werden.

Buch 8 beinhaltet viele Themen, die nicht die Java-Programmierung, sondern andere

Bereiche betrachten.
Die Bücher mit speziellen didaktischen Java-Umgebungen (Buch 4, 5 und 6) beinhalten die richtige Menge an Lehrinhalt und behandeln auch die wichtigsten Themen.

Trotz der großen Auswahl an reinen Java-Büchern fällt die Anzahl an Büchern, die Java und LEGO behandeln, sehr gering aus, und es existiert kein Werk, welches diese beiden Bereiche in einer für ProgrammieranfängerInnen tauglichen didaktischen Aufbereitung behandelt und auf dem Wissen der grafischen LEGO-Oberfläche aufbaut. Für ein entsprechendes Buch wird unter Berücksichtigung der analysierten Bücher in dieser Arbeit ein Konzept erstellt.

Buch 1 lässt die Lernenden Java zu Beginn von der Konsole aus starten, was unnötig ist und das Erlernen zu Beginn nur zusätzlich verkompliziert.

Buch 7 und 8, also beide Bücher, die LEGO MINDSTORMS behandeln, sind nur in Schwarz-Weiß und beginnen mit einem Hello-World-Programm – und dies, obwohl die Roboter doch so viele verschiedene spannendere Beispiele als das altbekannte Hello-World-Beispiel bieten, weshalb sie eine Wellenlinie und ein Minus erhalten haben.

Gute Ideen werden im Greenfoot-Buch und im BlueJ-Buch umgesetzt, beide haben eine optisch interessante Gestaltung und jeweils durch die Zusammenfassungen der Konzepte eine gute didaktische Möglichkeit genutzt.

Zusammenfassend betrachtet erfüllen die Bücher 2, 5, 6 und 7 am ehesten die gestellten Anforderungen, da sie die besten Bewertungen in allen Bereichen erhalten haben. Die Bücher 3 und 8 sind für die Zielgruppe nach den betrachteten Gesichtspunkten am wenigsten geeignet.

4. Wissenstransfer

Wenn Wissen aus einer Umgebung, wie es bei der grafischen Programmierumgebung der Fall ist, in eine andere Umgebung wie Java und eclipse übertragen werden soll, stellt sich die Frage, inwieweit dieser Wissenstransfer bei der entsprechenden Altersgruppe erfolgen kann.

[Yuan11] erläutert, dass die Ähnlichkeit von Systemen den Wissenstransfer ungemein erleichtert. Somit wird die Verwendung derselben Roboter und der grundlegend ähnlichen Programmierkonzepte eine Erleichterung für die Jugendlichen darstellen.

Alle ProgrammiererInnen müssen im Laufe ihrer Karriere verschiedene Programmiersprachen erlernen und dabei auf dem Wissen, das sie aus anderen Sprachen gesammelt haben, aufbauen [Wied96]. Dieser erste Umstieg von einer grafischen Oberfläche auf eine textuell objektorientierte Sprache bereitet die Jugendlichen darauf vor, was sie früher oder später ohnehin erwarten wird. Denn es ist unwahrscheinlich, dass ein/eine ProgrammiererIn oder auch ein Mensch, der an Programmiersprachen interessiert ist, sein ganzes Leben lang nur eine Sprache benötigt.

Beim Transfer von Programmierwissen spricht Wiedenbeck in [WIED96] davon, dass es positiven und negativen Transfer gibt. Negativ ist der Transfer des Programmierwissens nur dann, wenn durch die Vorkenntnis eines anderen Programmierparadigmas die Denkweise an eine neue Sprache nicht angepasst werden kann. Denn das Lösen neuer Probleme wird durch die Erfahrungen aus den bisherigen Programmiersprachen stark beeinflusst. Jeder andere Transfer ist positiv.

4.1 Vorwissen

In [@Wahl06] wurde gezeigt, dass der wichtigste Faktor beim Erlernen neuer Dinge das Vorwissen ist, und nicht – wie oft oberflächlich betrachtet – die Motivation oder die Begabung. Wobei die Begabung im Vergleich zur Motivation noch eine größere Rolle einnimmt, aber diese ist weniger wesentlich als die Rolle des Vorwissens.

Bei Vorwissen handelt es sich um Fähigkeiten und Kenntnisse, die vor dem Erzeugen neuen Wissens vorhanden sind. Man spricht vom Matthäus-Effekt, wegen jener Passage aus der Bibel: „Denn wer da hat, dem wird gegeben, dass er die Fülle habe; wer aber nicht hat, dem wird auch das genommen, was er hat.“ [@Wahl06] Dieser Effekt beschreibt die Rolle des Vorwissens im Lernprozess. Wenn der/die Lernende bereits einiges an Wissen erlernt hat, wird er/sie sich leichter tun beim Erlernen neuer Dinge.

Meist wird Vorwissen implizit genutzt, jedoch gibt es auch die Möglichkeit, Vorwissen explizit zu aktivieren. Hierbei wird zwischen einer offenen und einer fokussierten Aktivierungsstrategie unterschieden. Wird Vorwissen aktiviert, so werden Verbindungen zwischen den einzelnen Bereichen im Gehirn erzeugt und Wissensinseln vermieden.

Bei der offenen Aktivierungsstrategie wird Vorwissen aus allen Bereichen wiederbelebt, wohingegen bei der fokussierten Strategie nur das vorhandene Wissen aus einem speziellen Bereich angekurbelt wird. In dieser Arbeit soll nur das themenspezifische Vorwissen genutzt werden, und deshalb wird genauer auf die fokussierte Vorwissensaktivierung eingegangen. Es werden verschiedene Strategien, das fokussierte Wissen zu aktivieren, unterschieden. Dazu zählen laut [MAND05]:

- die kognitive Vorstrukturierung: Durch Strukturierung des alten und neuen Themengebietes für die Lernenden – wie es z.B. bei Advanced Organizer der Fall ist – neue Verbindungen herstellen. Dies ist auch gut geeignet, wenn das vorhandene Lernmaterial nur wenig ist und/oder unübersichtlich ist und die Lernenden eher wenig Vorwissen besitzen.
- das Fragestellen: Die Beantwortung von Fragen erfordert immer eine kognitive Aktivierung von bereits vorhandenem Wissen. Mit Hilfe von Fragestämmen können gute Lernerfolge erzielt werden.
- Beispiele und Falldarstellungen: Konkrete Beispiele stellen Vorwissen in Zusammenhang. Es wird unterscheiden zwischen Lernende mit viel Vorwissen, die eine Tiefenstruktur erkennen, und solchen mit wenig Vorwissen, die nur die oberflächlichen Merkmale sehen.
- Analogien: Hierbei wird die relationale Struktur des Wissens auf ein anderes Wissensgebiet übertragen. Analogien sind sorgfältig auszuwählen, damit kei-

ne falschen Zusammenhänge induziert und womöglich auch noch so gespeichert werden. Maßnahmen zum Erkennen von Strukturen sind bei den Lernenden notwendig, um Analogien sinnvoll nutzen zu können.

Bevor man sich Gedanken um die Wissensaktivierung machen kann, muss nicht nur festgestellt werden, welches Vorwissen vorhanden ist, sondern es sind auch die folgenden Punkte nach [Mand05] zu beachten:

- Inhalt des Wissens: „Soll im Rahmen von Lehr-Lern-Prozessen vorhandenes Wissen gezielt aktiviert werden, gilt es zunächst, sorgfältig zu überlegen, welches Wissen tatsächlich vorausgesetzt und damit auch aktiviert werden kann.“ [MAND05] Das Vorwissen, welches die zukünftigen ProgrammiererInnen dieser Arbeit besitzen, wird in 4.2.1 wiedergegeben. Zu unterscheiden ist, ob das Wissen prozedural, deklarativ oder konditional ist. Themenspezifisch muss zwischen oberflächlichem Wissen oder Konzeptwissen unterschieden werden.

- Bewusstsein des Wissens und Explizierbarkeit: „Die Bewusstheitsdimension bezieht sich auf die Frage, inwiefern das Vorwissen explizit bzw. explizierbar ist oder in impliziter Form vorliegt. Explizites Wissen ist verbalisierbar, kann also bewusst aktiviert werden. Deklaratives Wissen ist in der Regel explizit. Implizites Wissen (vgl. z.B. Neuweg, 2001) ist im Allgemeinen nicht oder nur schwer verbalisierbar und wird automatisch und bewusst aktiviert. Prozedurales Wissen ist zu einem großen Teil implizit. Ein Beispiel für implizites Wissen ist muttersprachliches Wissen, das in der Regel sehr gut angewendet, jedoch kaum expliziert werden kann (vgl. Spitzer, 2002).“ [Mand05]

- Repräsentation: Wissen kann in unterschiedlicher Weise im Gehirn gespeichert sein. So kann es z.B. in assoziativen Netzwerken dargestellt sein, wie dies bei deklarativem Wissen der Fall ist. Die Schule stellt einen Realitätsbereich dieser Wissensstrukturen dar. Prozedurales Wissen hingegen wird als konkrete Handlungsanleitung, in Form einer Wenn-dann-Regel, realisiert.

- Wissenschaftlichkeit: Dabei geht es darum, dass das Wissen nicht rein subjektiven Theorien zugrunde liegt, sondern auch der aktuellen Wissenschaft

entspricht.

- Umfang: Je größer der Wissensbestand ist, umso effektiver kann der Lernende neue Informationen aufnehmen, wie es bereits weiter oben geschildert wurde. Es bestehen beim Lernen viele Möglichkeiten der Anknüpfung an bereits vorhandenes Wissen.

- Handlungsrelevanz: Hier stellt sich die Frage, inwieweit das Wissen anwendbar ist und welche Operationen es erlaubt, was wiederum stark durch die zuvor genannten Bereiche beeinflusst wird.

4.1.1 Programmierumgebung

Die Lernenden haben bereits einiges an Wissen und Erfahrung mit dem LEGO-MINDSTORMS-Roboter gesammelt. Folgende Bereiche aus der grafischen Programmieroberfläche sollen den Jugendlichen bereits bekannt sein:

4.1.1.1 Blöcke

Jeder einzelne Befehl, den der Roboter ausführen soll, wird mittels eines Blocks vorgegeben. Blöcke sind jene Bereiche, die in der grafischen Programmieroberfläche auf dem unteren Seitenbereich vorzufinden sind.

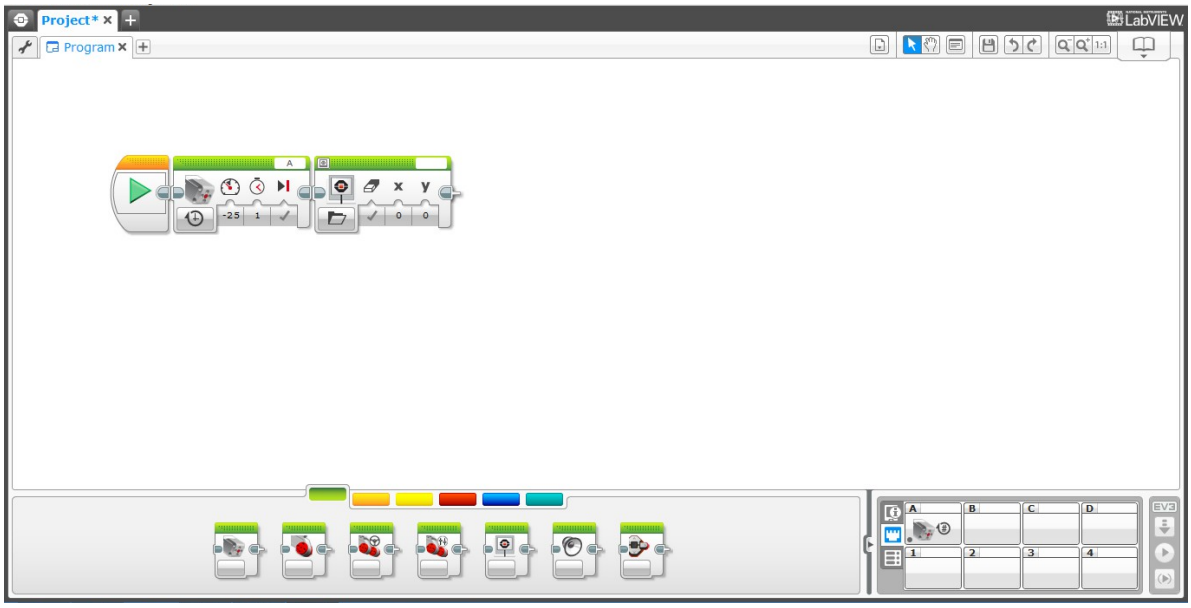


Abb. 12: EV3-G Programmfläche mit Blöcken

Abb. 13: Datenleitung zwischen zwei Blöcken

Blöcke sind nichts Geringeres als Methoden und haben genauso Ein- und Ausgabeparameter wie diese. Zu sehen in Abbildung 12, die orange Verbindung zwischen dem gelben und dem roten Block stellt den Ausgabewert des einen Blocks und den Eingabeparameter des anderen dar. In der grafischen Oberfläche werden solche Verbindungen Datenleitung genannt und müssen immer auf beiden Enden denselben Typ aufweisen. Versucht man einen Ausgabeparameter vom Texttyp mit einem Eingabeparameter vom Typ Logik zu verbinden, so wird die Datenleitung keine Farbe bekommen, sondern wird nur grau gestreift dargestellt.

Blöcke werden, beginnend beim Startblock, von links nach rechts aneinandergereiht und ergeben so den Programmablauf.

In EV3-G gibt es viele verschiedene Arten von Blöcken. Die grobe Einteilung wird kurz vorgestellt:

Aktionsblöcke

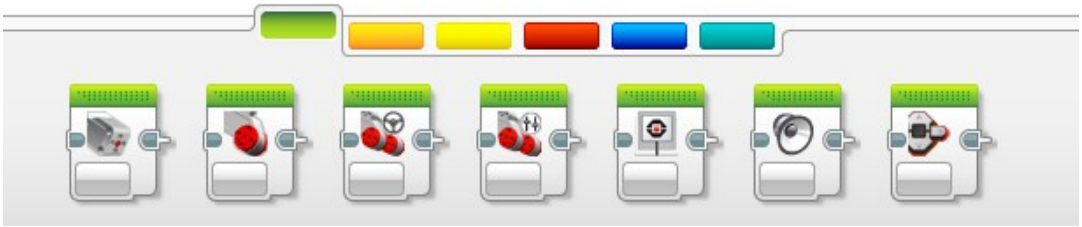


Abb. 14: alle Aktionsblöcke

Aktionsblöcke sind alle grünen Blöcke, die sowohl für die Motorbewegung als auch für die Klang- und Displayausgabe zuständig sind. Die Lernenden sollen alle in Abbildung 15 zu sehenden Aktionen bereits benutzt haben.

Ablauf-Regelung

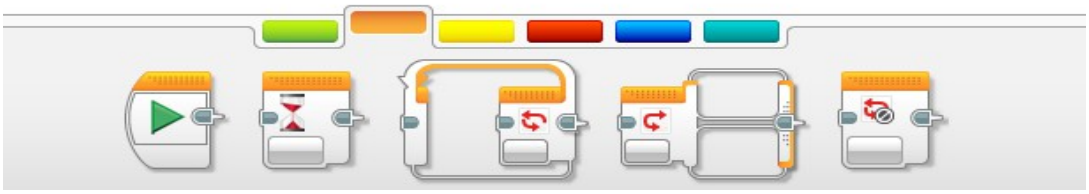


Abb. 15: alle Ablauf-Regelungsblöcke

Die in Abbildung 15 gezeigten Blöcke sind von links nach rechts: Start, Warten, Schleife, Verzweigung (wird Schalter genannt) und der Schleifen-Interrupt.

Warten: Warteblöcke dienen dazu, auf ein bestimmtes Ereignis zu warten.

Schleife: Schleifen in LEGO MINDSTORMS können mit unterschiedlichen Abbruchbedingungen programmiert werden. Dabei wird unterschieden zwischen unendlichen Schleifen, zeitlich begrenzten Schleifen, ereignisbedingte Abbruchbedingungen, zählergesteuerten Schleifen, Bluetooth-gesteuerten Abbruchbedingungen und Logisignalen als Abbruchbedingung. Die verschiedenen Abbruchbedingungen einer Schleife in EV3-G werden in Abbildung 16 dargestellt. Genauer soll in dieser Arbeit nicht darauf eingegangen werden, da es nur wichtig ist zu wissen, welche Schleifen die Jugendlichen bereits verwendet haben. Wichtig wären da einerseits die zählergesteuerten Schleifen, denn diese sind am einfachsten zu programmieren in Java und bieten daher einen leichten Einstieg. Andererseits sind auch die sensorgesteuerten Schleifen wichtig und sollten von den Interessierten bereits verstanden und verwendet worden sein.

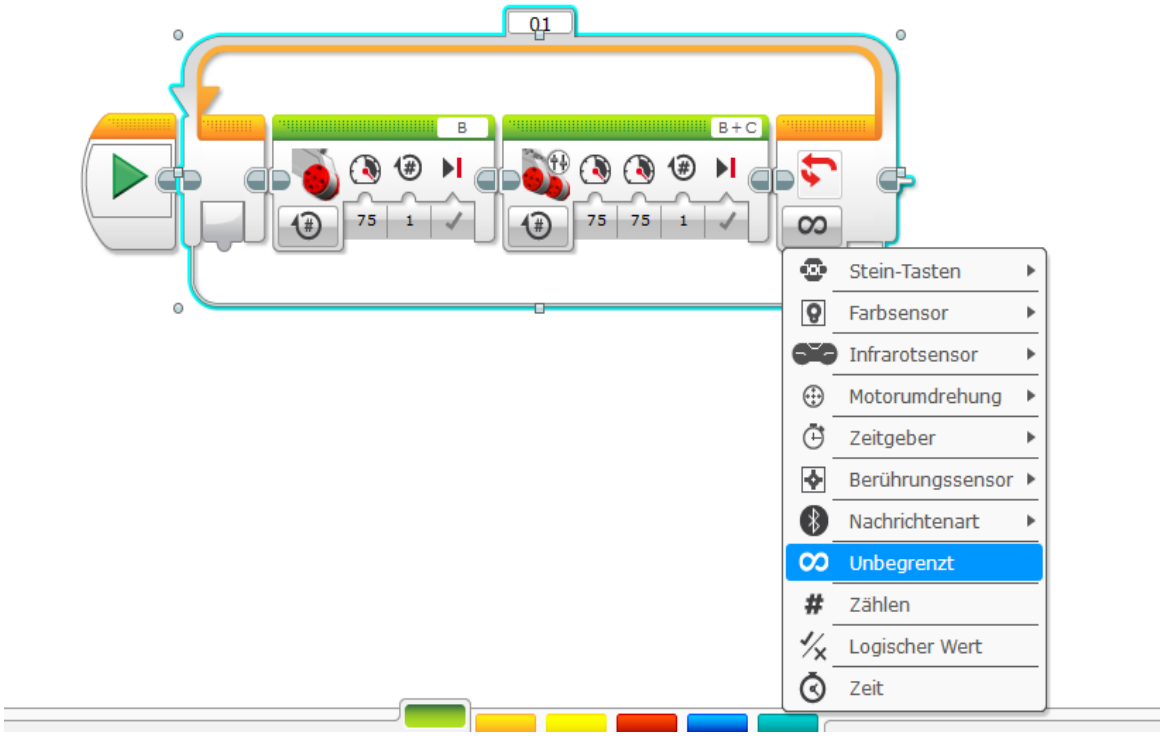


Abb. 16: Abbruchbedingung von Schleifen

Schalter: In Abbildung 15 ist ein Schalter als vierter Block von links zu sehen. Hierbei handelt es sich um Verzweigungen, die auch als switch-Anweisung eingesetzt werden können. Der Schalter kann dazu verwendet werden, um aufgrund eines Wertes verschiedene Befehle auszuführen. Dieser Wert kann von einem Sensor kommen, es kann sich aber auch um eine gewöhnliche Zahl, einen Text oder logischen Wert handeln.

Schleifen-Interrupt: Dieser Block muss nicht zwangsläufig von der Zielgruppe verstanden werden, denn darauf wird in der textuellen Programmierung des zu erstellenden Buches nicht aufgebaut.

Daten-Operation

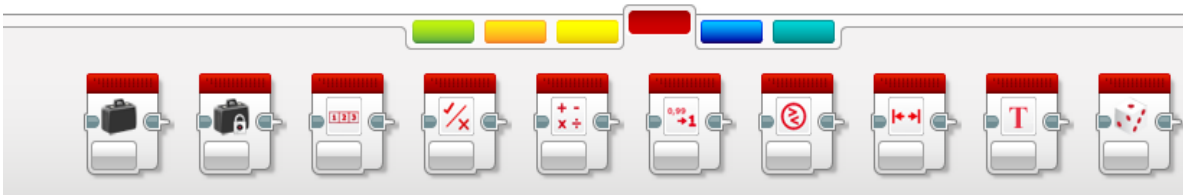


Abb. 17: alle Daten-Operations-Blöcke

Möglichkeiten werden hier viele geboten, genauso wie in anderen Programmiersprachen kann man Variablen, Konstanten, Arrays, logische Verknüpfungen und mathe-

matische Operationen benutzen. Es gibt auch rote Blöcke für das Vergleichen zweier Werte, für das Überprüfen, ob sich ein Wert in einem gewissen Wertebereich befindet, und das Verknüpfen von maximal 3 Texten zu einem Text usw.

Variablen: Es kann einer der folgenden Typen für Variablen ausgewählt werden: Text, Numerischer Wert, Logischer Wert, Numerisches Array, Logisches Array (siehe Abbildung 18). Außerdem können Variablennamen vergeben werden, und es ist zu unterscheiden, ob der Wert der Variable geschrieben bzw. gelesen wird.

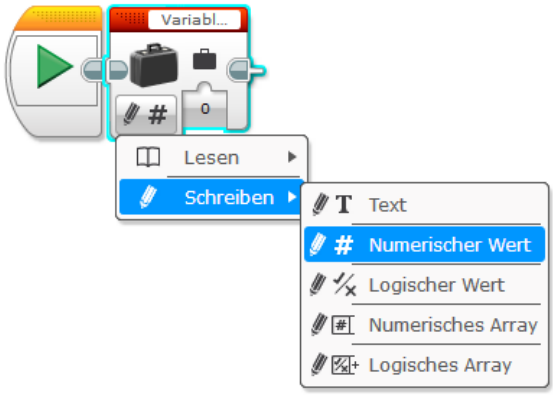


Abb. 18: Variable in EV3-G

Vergleichen: Der Block für das Vergleichen von Werten sei an dieser Stelle noch gesondert erwähnt, da für die richtige Verwendung dieses Blocks die Jugendlichen bereits verstanden haben müssen, worum es bei Datentypen geht (Abbildung 19).

Abb. 19: Block zum Vergleichen

Erweiterter Modus

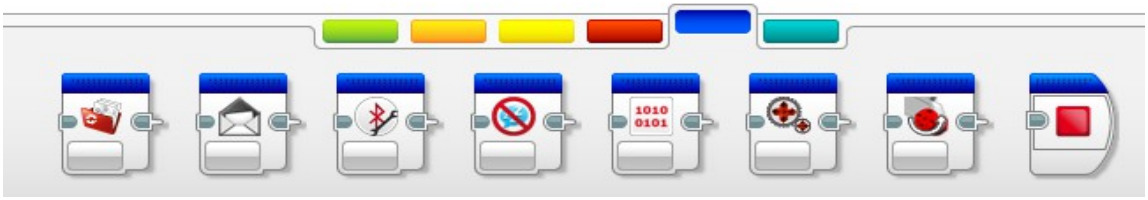


Abb. 20: alle Blöcke aus dem erweiterten Modus

Aus dem Repertoire des erweiterten Modus ist nur der letzte Block aus Abbildung 20 für das weitere Vorgehen wichtig, nämlich der Block zur Beendigung des Programms.

Eigene Blöcke

Abb. 21: Bereich für eigene Blöcke

Es können auch neue Blöcke, sogenannte „eigene Blöcke“, erstellt werden, worin dann mehrere Befehle zusammengefasst werden. Jeder eigene Block bekommt einen selbst gewählten Namen und ein Icon und kann dann an jeder Stelle im Programm verwendet werden. Somit haben Jugendliche, die diesen Block verwenden, bereits ihre eigene erste Methode geschrieben und sollten verstanden haben, wozu diese Blöcke gut sind; nämlich für die Übersichtlichkeit des Programms, für die bessere Lesbarkeit und für die zusammengefasste Darstellung.

In Abbildung 21 ist bereits ein selbst erstellter Block mit einem Icon zu sehen, worauf sich ein roter Kopf im Seitenprofil befindet.

Somit wurden die Möglichkeiten der grafischen Programmierumgebung vorgestellt, und es ist klar, welche Bereiche davon als notwendiges Vorwissen für die Zielgruppe betrachtet werden.

5. Didaktisches Konzept

Nachdem in Kapitel 3 analysiert wurde, welche Bücher bereits auf dem Markt vorhanden sind, und festgehalten wurde, dass kein Buch existiert, das geeignet ist, das bereits vorhandene Wissen aus der grafischen Programmierumgebung von LEGO MINDSTORMS zu nutzen, um den Einstieg in die Java-Programmierung zu erleichtern, wird in den kommenden Kapiteln das didaktische Konzept für solch ein Buch entwickelt.

5.1 Lernziel und Kompetenzen

Grundlegende Programmierkonzepte sollen von der Zielgruppe in einer textuellen Programmiersprache umgesetzt werden können, so dass der/die Lernende mit dem LEGO-MINDSTORMS-Roboter auch komplexere Aufgabenstellungen lösen kann als solche, die mit der grafischen Oberfläche möglich sind. Es ist im Zuge dessen für die Lernenden auch notwendig, das objektorientierte Konzept von Klassen und Objekten zu verstehen und anwenden zu können. Jedoch soll im Buch nicht allzu tief in die Materie eingegangen werden, denn auch wenn Vererbung für manche Aufgaben sicher praktisch ist, so reicht es aus, wenn die Lernenden wissen, wie eine Klasse von einer anderen erben kann. Es ist jedoch nicht notwendig, dass die AnfängerInnen hierbei z.B. abstrakte Klassen verstehen.

Neben dem Kennenlernen der Programmiersprache Java soll auch das logische und strukturierte Denken gefördert werden.

Ziel dieser Arbeit ist es nicht ,auf das Arbeiten im Team, welches durchaus sehr wichtig ist und bereits in anderen Arbeiten wie z.B. [WAGN05] erläutert wurde, näher einzugehen. Denn aufgrund der nicht institutionell vorgegebenen Strukturen steht im Vorhinein nicht fest, ob die Lernenden alleine oder mit KameradInnen arbeiten, und es bleibt ihnen je nach Belieben auch selbst überlassen.

Vorausgesetzt wird, dass alle Lernenden, die sich mit diesem Buch Java-Kenntnisse aneignen wollen, bereits zuvor mit einer der beiden grafischen Oberflächen NXT-G

oder EV3-G von LEGO gearbeitet haben und auch schon einige Roboter gebaut haben. Die Lernenden verfügen somit bereits über das Wissen der grundlegenden Programmierkonzepte, wie Verzweigungen, Schleifen, Variablen, Parallelität und iterative Programmabläufe. Darauf aufbauend soll das Wissen nun in die textuelle Programmiersprache Java übergeleitet werden.

5.2 Das Medium

Viele unterschiedlichen Lernmaterialien sind für das Ziel dieser Arbeit denkbar, jedoch soll hier nicht jedes Medium ausführlich erörtert und ausgearbeitet werden, denn dies würde den Rahmen dieser Arbeit sprengen. Es wird, trotz der großen Vielfalt an möglichen Lernunterlagen, die Form des Buches gewählt. Andere vorstellbare Lernmedien werden ebenfalls erläutert.

5.2.1 Buch

Lernen ist ein so hochgradig individueller Prozess, sodass es Sinn macht, ein Buch zur Verfügung zu stellen, mithilfe dessen sich jeder/jede Lernende selbst Wissen aneignen und vertiefen kann. „Aus allen diesen Überlegungen resultiert eine zentrale Botschaft: Lernen ist ein hochgradig einzigartiger Prozess. Kollektive Lernphasen sind hierfür ungeeignet. Was man benötigt, das sind innovative Formen des Lehrens und Lernens.“ [@Wahl06]

Bücher können zu jeder Tageszeit an jedem Ort gelesen werden, egal ob abends neben dem Computer oder mittags im Garten. Außerdem können die Lernenden die Länge der Lernintervalle selbst bestimmen.

Ein Buch ist für jede/jeden Lernenden einfach neben dem Bildschirm zu legen, und es kann darin gelesen werden, während gleichzeitig am Computer die Entwicklungsumgebung geöffnet ist und das Programm geschrieben wird. Ein Schließen oder Minimieren der Entwicklungsumgebung, um auf das Lernmaterial zuzugreifen, ist nicht notwendig. Dies ermöglicht ein leichteres Vergleichen der Teile aus dem Buch mit den eigenen Programmabschnitten, denn nicht jeder/jede hat einen so großen Bild-

schirm, dass ohne weiteres zwei Fenster parallel geöffnet sein können, oder verfügt sogar über einen zweiten Bildschirm.

Der Code aus dem Buch kann nicht in das Programm kopiert werden, was den großen Vorteil bietet, dass die Lernenden sich die neue Syntax schneller merken werden, wenn sie alles selbst schreiben müssen.

5.2.2 E-Learning

E-Learning-Plattformen sind zurzeit eine gängige Variante der Lernunterstützung, sie bieten entweder mit oder ohne Login die Möglichkeit, Unterlagen herunterzuladen oder online zu betrachten. Neben der bloßen Verfügbarkeit von Materialien bieten sie immer öfter auch interaktive Lernmethoden an. So sind immer mehr und mehr Lernspiele, Kreuzworträtsel, Lückentexte usw. vorzufinden.

5.2.3 Kurs

In diversen Kursen wird mittlerweile österreichweit Kindern und Jugendlichen verschiedener Altersgruppen die Handhabung der LEGO-MINDSTORMS-Roboter nähergebracht. Diese Kurse bieten eine gute Möglichkeit für jene, die keinen eigenen Roboter zuhause haben, um sich damit zu beschäftigen. Kurse bieten den Vorteil, dass es eine Ansprechperson für die Lernenden gibt, dennoch wäre oft eine weiterführende Unterstützung für die AnfängerInnen häufig sinnvoll. Die Kombination aus einem Kurs und einem Buch wäre hier eine gute Alternative.

„Kooperative Lernformen fordern es, das eigene Vorwissen zu explizieren und neue Inhalte zu erläutern.“ [Mand05] Außerdem begünstigen sie das Erkennen von Fehlkonzepten und Wissenslücken. Innerhalb eines Kurses kann kooperatives Lernen stattfinden.

Abgesehen davon, dass Lernende bei einem Buch sich mit ihrem Lerntempo an keine anderen anpassen müssen, wie dies in einem Kurs der Fall ist, können sie beginnen, wann sie wollen, und sie können jederzeit die Räumlichkeiten verlassen, was bei einem Kurs nicht gegeben ist.

5.2.4 Video-Tutorials

Bis auf ein paar Ausnahmen bieten Video-Tutorials die gleichen Möglichkeiten wie Bücher. So ist es für die Lernenden möglich, das Videomaterial zu jeder Tageszeit in Anspruch zu nehmen, sofern ein Computer vorhanden ist. Durch das Vor- und Zurückspielen können beliebige Stellen wiederholt angesehen werden. Jedoch ist es bei Video-Tutorials nur möglich, diese gleichzeitig mit dem Programm zu betrachten, wenn der Bildschirm groß genug für zwei parallel geöffnete Fenster ist oder ein zweiter Bildschirm vorhanden ist. Im Gegensatz zum Buch kann hierbei nur der akustische und visuelle Lerntyp angesprochen werden.

5.3 Zielgruppe

Welche Jugendlichen sind es nun, die an dem Buch interessiert sind? An Roboter interessierte Jugendliche, welche bereits einiges mit LEGO-MINDSTORMS-Robotern und der grafischen Oberfläche gemacht haben und gerne weitere Aufgaben lösen bzw. Neues kennenlernen möchten, sind die definierte Zielgruppe für dieses Buch. Deren Anzahl ist zum derzeitigen Stand noch relativ gering; wenn jedoch immer mehr mit der grafischen Oberfläche gearbeitet haben, wird auch die Anzahl derer größer werden, die in eine „richtige“ Programmsprache hineinschnuppern wollen.
Mädchen und Jungen sollen mit dem Buch gleichermaßen angesprochen werden. Es wird deshalb auf Beispiele verzichtet, die sehr geschlechterspezifisch sind, wie z.B. Schussmaschinen oder niedliche Küken. Vielmehr sollen Beispiele enthalten sein, die auf den Spieldrang der Kinder zielen, ohne zu viel Wettbewerbscharakter zu beinhalten, z.B. das Befreien aus einem Labyrinth oder das Lösen eines Sudoku.

Welche Altersgruppe wird es sein, die sich mit diesem Buch beschäftigt? Zuvor wird das Alter betrachtet, ab welchem Jugendliche geeignet sind für die Verwendung der grafischen Oberfläche.

Kinder unter 8 Jahren sind meist noch zu jung, um selbstständig ohne dauerhafte Betreuung überhaupt einen Roboter nach Anleitung zu bauen und diesen dann zu programmieren. Dieser Altersgruppe fehlt größtenteils die notwendige Erfahrung im

Handling eines Computers. Die Programmierung eines Roboters überfordert sie dementsprechend, denn schon die gewöhnliche Benutzung eines Computers stellt für sie zumeist eine Herausforderung dar.

Kinder über 8 Jahren sind bereits mit dem Computer soweit vertraut, dass das alltägliche Handling damit kein Problem mehr darstellt. Sie sind im Bauen der Roboter gut genug, um alleine, egal ob anhand von Anleitungen oder selbstständig, tolle Ergebnisse zu erzielen. Auch die Konzentrationsfähigkeit ist ausreichend lange, um Projekte fertigstellen zu können. Ab einem ungefähren Alter von 8 Jahren können sie also die grafische Oberfläche sinnvoll benutzen, ohne dabei überfordert zu sein.

Erst in der formal-operationalen Phase nach Piaget (Kapitel 5.5.1) sind Kinder in der Lage, abstrahiert zu denken, und diese Phase beginnt im ungefähren Alter von 12 Jahren. Es gibt kein weitläufig bekanntes Alter, ab welchem es günstig ist, Kindern eine textuelle Programmiersprache beizubringen, bzw. ab welchem sie dazu in der Lage sind. Jedoch ist klar, dass, bevor sie nicht diese Phase erreicht haben, es noch keinen Sinn ergibt, ihnen Java beizubringen. Da das Ergebnis dieser Arbeit ein Konzept für ein Buch ist, ist es ausreichend, wenn man eine Altersgruppe von Jugendlichen über 12 Jahren festlegt. Es wird keine obere Altersgrenze definiert, da gerne auch ältere Interessierte das Buch lesen können. Jedoch wird das Ergebnis der Arbeit kein geeignetes Lernmaterial für die meisten Erwachsenen darstellen, da es bewusst in einem jugendlichen Stil erstellt werden wird.

5.3.1 Entwicklungspsychologie von Piaget

Jean Piaget (1896–1980) hat vier Stufen der geistigen Entwicklung festgelegt, die sich mit der geistig strukturierten Entwicklung von Kindern beschäftigen und sich an gewissen ungefähren Altersgrenzen orientieren. Folgende vier Faktoren spielen dabei eine entscheidende Rolle:

- „Reifung,
- aktive Erfahrung,
- soziale Interaktion,
- Streben nach Gleichgewicht.“ [@unid]

Im Folgenden werden die einzelnen Phasen der kindlichen Entwicklung nach Piaget zusammengefasst [UNID]:

Die **Sensomotorische Phase** von 0 bis 2 Jahren äußert sich dadurch, dass Kinder nur motorische Aktivität zeigen. In dieser Phase werden sechs Teilphasen unterschieden:

Im 1. Lebensmonat befinden sich die Neugeborenen in der Phase der angeborenen Reflexmechanismen: Hier kommt es zur wiedererkennenden und verallgemeinernden Assimilation, so saugt das Kleinkind z.B. nicht nur an der Brustwarze, sondern auch zwischen den Mahlzeiten an neuen Gegenständen oder ins Leere.

Das 1.-4. Lebensmonat ist die Zeit der primären Kreisreaktionen. Mit Kreisreaktionen sind hier folgende Reaktionen auf Aktivitäten gemeint, die sich für den Körper angenehm anfühlen und aufgrund dessen wiederholt ausgeführt werden, egal ob es sich dabei um eine Berührung, eine Betrachtung oder ein „In-den-Mund-stecken“ handelt. Generalisierte Assimilation nennt es Piaget, wenn diese Handlungen auf andere Objekte übertragen werden.

Die Zeit zwischen dem 4. und 8. Lebensmonat nennt Piaget die Periode der sekundären Kreisreaktionen. Nun merkt das Kind, dass seine eigenen Aktionen unterschiedliche Reaktionen hervorrufen, und kann mit Experimenten versuchen, das gewünschte Ziel zu erreichen.

Ungefähr vom 8.–12. Lebensmonat verhalten sich die Heranwachsenden intentional, d.h. sie verbessern bereits vorhandene Muster und wenden bekannte Effekte in neuen Bereichen an.

Die Zeit der tertiären Kreisreaktionen, wo gefestigte Verhaltensmuster intuitiv auf neue Situationen probiert werden, ist zwischen dem 12. und 18. Lebensmonat. Dabei werden neue Handlungsschemata erzeugt, und das Kind bemerkt, wie es auf die Umgebung einwirkt. „Das Kind untersucht verschiedene 'Spritztechniken' beim Baden. Es kann mit der eigenen Hand oder mit der Spielente auf das Wasser schlagen und das Wasser spritzt unterschiedlich weit...“ [@unid]

18.–24. Lebensmonat – Übergang zur voroperationalen Phase: Hierbei hat das Kind bereits ein inneres Bild vom Gegenstand gespeichert und kann Reaktionen voraussehen, ohne dass der Gegenstand physisch anwesend sein muss.

Der Beginn der **präoperationalen Phase,** in der sich Kinder von 2 bis 7 Jahren befinden, wird durch Anthropomorphismus (Vermenschlichung von Gegenständen) und menschliches Denken gekennzeichnet. Ab einem Alter von 4 Jahren nimmt die Zahl der logischen Irrtümer ab, der Egozentrismus ist jedoch noch immer deutlich vorhanden. Das Kind kann sich nicht in die Sichtweise anderer hineinversetzen. Ein egozentrisches Kind glaubt, dass alle anderen dasselbe sehen und denken wie es selbst.

„Dem Kind ist es zunehmend möglich, sich komplette Handlungen auf gedanklicher Ebene vorzustellen, wenn diese Handlungen bereits im ‚echten Leben' ausgeführt wurden.“ [UNID]

Neben dem Egozentrismus herrscht in dieser Phase auch die Zentrierung vor, die beschreibt, dass ein Kind nicht fähig ist, sich auf mehrere Merkmale gleichzeitig zu konzentrieren.
Danach kommt die **Phase der konkreten Operationen** von 7 bis 12 Jahren. Ungefähr im Grundschulalter löst sich die Denkweise von der Wahrnehmung und wird mehr und mehr zur konkreten Operation, das logische Denken setzt ein. Jedoch haben viele Kinder in dieser Phase Probleme, unrealistische Vorstellungen nachzuvollziehen.

Hypothetisch-deduktives Denken kennzeichnet den Übergang von der Phase der konkreten Operationen zur Phase der formalen Operationen, welche ca. ab einem Alter von 12 Jahren beginnt. Die Inklusionsbeziehung (sind zwei Aussagen wahr, ist auch die davon abgeleitete wahr) ist für Kinder in diesem Lebensabschnitt lösbar. Hierin begründet sich auch ein Mindestalter von 12 Jahren für die Altersgruppe der vorliegenden Arbeit.

Darauf folgt die **Phase der formalen Operationen** mit ca. 12 bis 15 Jahren. Mit dem formalen Denken tritt nach Piaget eine Sinnesumkehrung zwischen dem Wirklichen

und dem Möglichen ein. Das formale Denken ist grundsätzlich **hypothetisch-deduktiv**. Denkoperationen können auf dieser Stufe mit abstrakten, nicht mehr konkret vorstellbaren Inhalten durchgeführt werden. Dies entspricht der höchsten Form des logischen Denkens.

„Es sollte versucht werden, einem Kind Probleme in einem angemessenen Schwierigkeitsniveau entsprechend der jeweiligen Entwicklungsstufe zu präsentieren. Aus den Ansätzen von Jean Piaget ist ein sehr aktives Erziehungskonzept abzuleiten, welches von der Welt des Kindes (und nicht der Welt der Erwachsenen) ausgeht.“ [@unid] Demnach orientiert sich diese Arbeit mit ihren Beispielen an den Kindern in der Phase der formalen Operationen, da sie frühestens in diesem Alter dazu in der Lage sind, die Aufgabenstellungen zu bewältigen.

5.3.2 Hochbegabte

Nicht übermäßig begabte Jugendliche werden sich mit der grafischen Programmierung der Roboter zufriedengeben und nur wenige davon werden aus Interesse versuchen, hier weitere Schritte in Richtung Programmierung vorzunehmen. Jene, die aber daran interessiert sind, werden allerdings keine Unterlagen finden, welche auf ihr vorhandenes Wissen eingehen und sie auch nicht mit zu komplizierten Programmierkonstrukten oder Beispielen überfordert. Der geringe Prozentsatz an Lernenden, die gerne Java-Programmierung lernen wollen, kann als begabt bzw. hochbegabt bezeichnet werden. Es gibt diverse Modelle, um Hochbegabte zu erkennen, aber keine Kriterien um festzustellen was eine Hochbegabtenbildung ausmacht [BERG09]. Deshalb wird ein Konzept erstellt, wie es auch für normal begabte Jugendliche zutreffend wäre.

5.4 Methode

Um in die objektorientierte Programmierung einzusteigen, gibt es verschiedene didaktische Ansätze. Zurzeit werden hauptsächlich die Methoden Object-First und Object-Later in der Literatur besprochen.

Lawhead et.al [Lawh08] und Kölling [Köll01] sind Verfechter der Object-First Herangehensweise und nennen sowohl Guidelines als auch Vorteile dieser Methode.

Ehlert und Schulte stellen Object-First und Object-Later in [Ehle09] gegenüber und kommen zu dem Schluss, dass:
„Die Fragestellung braucht in Zukunft nicht mehr zu lauten, mit welchem Paradigma die Lehrerin bzw. der Lehrer in den Informatik-Anfängerunterricht einsteigt, da bezogen auf ein Schuljahr (mit ca. 100-120h) die Klassen fast das Gleiche lernen. Der Unterrichtende muss sich vielmehr Gedanken machen, warum bestimmte Themen aus welchen Gründen von den Schülern als schwer erlebt werden bzw. mangelhaft gelernt werden und warum bestimmte Themenübergänge Schwierigkeiten bereiten."

In [Coop03] werden sogar drei verschiedene Ansätze erwähnt, nämlich Imperativ-First, Functional-First und Object-First. Der funktionale Ansatz soll in dieser Arbeit nicht näher besprochen werden, da keine funktionale Programmiersprache dafür vorhanden ist, die LEGO-MINDSTORMS-Roboter zu programmieren. Außerdem stellt die funktionale Programmierung in der Wirtschaft eine Randgruppe dar und ist eher für die Wissenschaft und für Studenten als die in dieser Arbeit vorgestellte Altersgruppe relevant.
Imperativ-First kann genauso auch Object-Later genannt werden, denn in beiden werden zu erst Kontrollkonstrukte der imperativen Programmierung gelehrt, und erst anschließend geht man zu den Objekten und den anderen objektorientierten Konzepten über.

Die verwendete Methode ist jedoch nicht das, wovon der Lernerfolg am stärksten abhängt, wie in [Wied96] gezeigt wird. Dieser hängt viel stärker vom Lehrpersonal ab als von der verwendeten Methode. Da bei dem Konzept dieser Diplomarbeit kein/keine LehrerIn zur Verfügung steht, sind es andere Kriterien, die den Lernerfolg beeinflussen. Diese Kriterien werden in Guidelines von Kölling und Rosenberg weiter unten im Text angeführt.

Da prozedurale Programmiersprachen immer auch imperativ sind und der prozedurale Ansatz im Kapitel 5.4.2 erläutert wird, wird an dieser Stelle auf eine nähere Erklärung des imperativen Ansatzes verzichtet. Object-First wird im anschließenden Kapi-

tel unter die Lupe genommen.

5.4.1 Object-First

„One of the principal arguments for the ‚objects first' approach mentioned above is that the object-oriented paradigm offers a more natural modeling process for the problems we wish students to solve.“ [Barn02] Direkt mit der Objektorientierung Programmieren zu beginnen, erleichtert die Problemlösung, weil es sich um eine natürliche, dem alltäglichen Leben angepasste Denkweise handelt. Dennoch gibt es auch beim Object-First-Ansatz gewisse Grundsätze, an die sich das Lehrpersonal halten sollte, damit die Programmierlehre mit Erfolg gekrönt ist. Hierfür wurden Guidelines von Kölling und Rosenberg niedergeschrieben, die dabei unterstützen sollen.

Michael Kölling und John Rosenberg nennen folgende Guidelines in „Guidelines for Teaching Object Orienation with Java“ [Köll01]:

1. Object-First: Laut [BARN02] sollten die ersten Schritte eines/einer Programmieranfängers/Programmieranfängerin darin liegen, ein bestehendes Projekt zu öffnen und neue Objekte anzulegen.
2. Don't start with a blank screen: Es ist viel zu schwierig für AnfängerInnen auf einem leeren Bildschirm zu beginnen, daher sollten Lehrer zu Beginn Projekte vorgeben, worin die Lernenden nur den Code anpassen müssen.
3. Read Code: Leider wird viel zu oft vollkommen auf das Lesen von gut geschriebenem, schön strukturiertem Code verzichtet. Nicht nur, dass Lernende zuerst ihren eigenen Code schreiben, ohne zuvor auch nur einen anderen Code gesehen zu haben, sie kommen sehr oft gar nicht zum Lesen des Codes, auch dann nicht, wenn sie schon einige Zeit eigene Programme entwickelt haben.
4. Use „large“ projects: Laut den Autoren macht Objektorientierung nur in größeren Projekten mit mehr Klassen Sinn, und Klassen mit nur einer Methode stellen schlechte Beispiele dar.
5. Don't start with „main”: Da die main-Methode von Java nicht objektorientiert ist, sondern nur als Schnittpunkt zum Betriebssystem vorhanden ist und keine Objekte oder Klassen verwendet, sollte diese nicht als Einstieg in die Programmierung gewählt werden. Dies wäre, als würde man mit der Ausnahme

einer Regel beginnen.

6. Don't use “Hello World”: Beim bekannten „Hello World“-Beispiel wird eine Klasse geschrieben, wovon nie ein Objekt erzeugt wird, und eine Methode implementiert, die keine einzige objektorientierte Operation verwendet, weshalb dieses Einstiegsbeispiel nicht gewählt werden soll.
7. Show program structure: Wenn die interne Programmstruktur nicht sichtbar ist, so muss der/die Lehrende großen Wert darauf legen, diese zu visualisieren.
8. Be careful with the user interface: An dieser Stelle werden drei verschiedene Möglichkeiten erläutert. Erstens Text I/O, zweitens GUI oder drittens Applets. Applets und GUI-Interfaces sind kompliziert und erfordern viel Zeit und Lernende empfinden immer das als wichtig, womit sie am meisten Zeit verbringen, was in diesem Fall nicht zutrifft. Beim Text I/O ist der Output einfach und unproblematisch, der Input stellt jedoch eine gewisse Herausforderung dar.

Werden diese Guidelines befolgt, so ist es laut [Chen04] immer noch so, dass beim Object-First-Ansatz Schüler gezwungen werden, in einer höheren Abstraktionsebene zu denken, als in der, in der sie sich befinden.

Durch in der Lehre passierende Fehler können AnfängerInnen schnell Fehlvorstellungen festigen. [Dient07] „Außerdem müsste in empirischen Studien versucht werden zu klären, inwiefern die Schüler durch die hier vorgestellten Unterrichtsmethoden tatsächlich bessere Abstraktionsfähigkeiten erhalten oder ob sie nur bestimmte ;Kochrezepte' ausführen.“ [Diet07]

Laut [Coop03] müssen Lernende bei der Object-First-Vorgehensweise direkt in das Geschehen von Objekten, Klassen, Methoden und Kapselung einsteigen, und zur Komplexität dieser Konzepte kommt noch das aufzubauende Verständnis für Datentypen, Variablen und Werte hinzu. Als wäre dies nicht genug, werden dann auch noch das Event-gesteuerte Verhalten und die GUI miteinbezogen. Diese von [Coopp03] geschilderte Hürde des direkten Einstiegs in die objektorientierte Programmierung ist bei der Programmierung der LEGO-MINDSTORMS-Roboter mit Java nicht mehr so groß, wenn zuvor mit der grafischen Oberfläche schon viel gemacht wurde und grundlegende Konzepte verstanden wurden.

Alle Objekt-First-Bücher, die am Markt existieren, beinhalten nach [Chen04] mindestens eine der folgenden problematischen Vorgehensweisen:
„Traditional examples getting OO-dressed up.“ Hierbei werden gängige prozedurale Beispiele, wie etwa die häufig vorkommenden Konvertierungsbeispiele, in das Kostüm einer Klasse gestülpt.

„Using isolated OO examples.” Wenn dies der Fall ist, wird zwar mit funktionierenden Beispielen begonnen, aber anschließend, wenn z.B. die Vererbung an der Reihe ist, wird ein neues Beispiel, welches kürzer und einfacher ist, aber keinen Bezug zu den vorhandenen Beispielen darstellt, verwendet. Durch die Isolation der Beispiele fällt es den Lernenden schwerer, den Gedankengängen zu folgen.

„Using misleading ‚home-made' packages.“ In einigen Büchern werden die Standardbibliotheken gleich zu Beginn verwendet. Nach dem Motto, wenn die Verwendung bekannt ist, ist das Schreiben eigener Methoden einfacher – was aber nicht zutrifft. Denn die Benutzung kann nicht mit dem konzeptuellen Verständnis gleichgesetzt werden.

„OO is said, but not done.” Dieses Problem ist mehrschichtig, denn einerseits werden Beispiele wie CD-Samlungen oder Adressbücher verwendet, die gut geeignet sind, um die Datenkapselung zu zeigen, anderseits sind sie unvorteilhaft gewählt um die Vorteile der Objektorientierung hervorzuheben. Des Weiteren werden laut [Chen04] Event-gesteuerte Ereignisse eingesetzt, obwohl Events nicht objektorientiert behandelt werden. Drittens wird nicht gezeigt, dass durch Vererbung von GUI-Elementen aus der Library spezielle Bedürfnisse gedeckt werden können und nicht alles selbst programmiert werden muss.

„Distractions” Oft wird in Anfängerbüchern auch über Netzwerkkommunikation und Parallelität geschrieben, obwohl diese Konzepte nicht mehr zu den Grundlagen der Objektorientierung gehören.

„Weakened algorithm exposure.” Aufgrund des großen Zusatzmaterials, das den AnfängerInnen zur Verfügung gestellt wird und erlernt wird, wird älteres, traditionelles Know-how vernachlässigt oder sogar ausgelassen.

Hu schreibt 2004 in [Chen04] davon, dass es zwar gute Bücher über Objektorientierung gibt, aber diese eher für erfahrene ProgrammiererInnen zum Umstieg von einer anderen Sprache geeignet sind. Es hat 2004 aber die Bücher wie Java-Hamster oder Greenfoot und BlueJ noch nicht gegeben.

[Chen04] ist ein Gegner des Object-First-Ansatz, er bezeichnet es als kontraproduktiv und schädlich, gibt aber zu, dass durch die Nachfrage der Wirtschaft das Bedürfnis vorhanden ist, Objektverständnis früh zu erlernen.

Der Umstieg von OOP zu prozeduraler Programmierung oder umgekehrt ist in keinem Fall einfach. Es kann auch nicht davon ausgegangen werden, dass es in eine Richtung leichter fällt als in die andere, siehe [Chen04]. Soll also die Objektorientierte Programmierung als Ziel erlernt werden, so macht es auch Sinn, direkt damit zu beginnen.

5.4.2 Prozedurale Programmierung vor OOP

Neben dem Object-First-Ansatz gibt es auch andere Wege, in die Programmierung einzusteigen; einer davon ist der Einstieg mit prozeduralen Konzepten, um anschließend in die OOP umzusteigen. Auch dieser Weg soll hier kurz erörtert werden: „Structured programming is not simply about writing procedures or functions, but about step-wise refinement for functional decomposition and the disciplined programming practices, which is clearly not derived from the OO paradigm. A paradigm consists of a way of thinking and a way of doing things.“ [Chen04] Dieses Zitat zeigt bereits, dass jede Programmierart eine eigene Art zu denken erfordert und somit ein Umdenken stattfinden muss, wenn zuvor mit einer anderen Programmierart begonnen wurde, um anschließend auf die andere umzusteigen. Dementsprechend sollte zumindest eine gewisse Zeit in einer Programmierart entwickelt worden sein, um zumindest eine Denkweise zu festigen – und nicht durch den Umstieg eine zusätzliche Herausforderung bezüglich der Herangehensweise zu schaffen.

Betrachtet man den Einstieg in eine Programmiersprache mit rein der Sprache und einer gewöhnlichen Entwicklungsumgebung, so ergibt es durchaus Sinn, zuerst die

Ablaufsteuerungskonzepte wie Schleifen, if/else, Variablen, primitive Datentypen und dergleichen zu erarbeiten, damit anschließend mit der Objektorientierten Programmierung sinnvollere Beispiele gemacht werden können, ohne auf einen Schlag dann das Klassenkonzept und die Ablaufsteuerung erklären zu müssen, was sicherlich für Verwirrung unter den Lernenden sorgen würde.

Das konzipierte Buch verwendet zwar von Beginn an die objektorientierten Konzepte, meist werden diese aber vor den AnfängerInnen zuerst versteckt. Nach der Erklärung allgemeiner Punkte, die für die Installation und Verwendung von eclipse notwendig sind, wird gleich ein einfacher Fahrroboter programmiert und anschließend die Grundlagen wie Variablen, Schleifen oder if/else durchgenommen. Erst nachdem diese Bereiche bekannt sind, wird das zuvor versteckt gehaltene objektorientierte Konzept den LeserInnen immer mehr gezeigt.

5.5 Beispiele

In der Wirtschaft spielen das Lesen von Code und das Verstehen von bereits vorhandenen Programmen eine viel bedeutendere Rolle als das Erstellen neuer Softwareprogramme. Daher ist es auch wenig sinnvoll, wenn die AnfängerInnen immer nur auf einem leeren Bildschirm beginnen, um neue Programme zu schreiben [Köll04]. Demnach wird es für die ersten Beispiele auch Grundgerüste geben, die die Lernenden nur anpassen und verändern müssen, ähnlich wie es beim BlueJ-Buch der Fall ist.

Es ist davon auszugehen, dass der Großteil der Beispiele, welche in der Programmieroberfläche vorgestellt werden (Dinosaurier, Stapel-Roboter, Schlange, Humanoide usw.), bereits von den Lernenden gebaut und programmiert wurden. Sinnvoll ist es also, als Einstiegsbeispiel einen dieser Roboter zu verwenden, denn dadurch, dass er bereits bekannt ist, fühlen sich die Java-AnfängerInnen damit gleich zu Beginn sicherer.

„It is not easy to see how students can work on problems large enough to be truly interesting early in the course, while they have little experience with software development.“ [Köll04] Die Beispiele müssen groß genug gewählt werden, sollen aber nicht

zu groß ausfallen, damit auch ein Ziel absehbar ist.

Variante 1: Eine Idee wäre, ein großes Beispiel zu wählen und dieses im Laufe der unterschiedlichen Kapitel bis zum Schluss fertigzustellen – während immer zu jedem Kapitel kleinere Beispiele vorzufinden sind, die nichts mit dem großen Beispiel zu tun haben.

Variante 2: Eine andere Idee ist, viele kleinere Beispiele zu wählen, die alle unterschiedlicher Natur sind.

Variante 1 ist nicht umsetzbar, da es bei LEGO-Robotern im Gegensatz zu andern Modellen wie BlueJ, Greenfoot oder den Java-Hamstern so ist, dass nicht beliebig zwischen Beispielen gewechselt werden kann. D.h., wenn ein Roboter zu programmieren angefangen wurde, ist es sinnvoll diesen fertig zu programmieren, nicht dazwischen ein anderes Programm zu starten und später dieses erste Beispiel weiterzuführen. Das geht nur, wenn für beide Programme derselbe Roboter verwendet wird. Sobald ein Beispiel der beiden eine andere Roboterbauart erfordert, wird es kompliziert. Denn dann müsste der erste Roboter abgebaut werden, um den zweiten bauen zu können, und anschließend wieder den ersten zusammengebaut werden, um an dem vorherigen Beispiel weiterzuarbeiten zu können.

Bei Variante 2 zwei ist es wichtig, nicht zu viele unterschiedliche Roboter zu verwenden, da für jeden neuen Roboter umgebaut werden müsste; und da Lernende immer dem die größte Bedeutung schenken, was am meisten Zeit in Anspruch nimmt, sollte nicht das Bauen, sondern das Programmieren mehr Zeit benötigen. Das Buch wird also die zweite Variante einsetzen.

Zu erwähnen ist an dieser Stelle auch, dass keine Bauanleitungen Inhalt des Buches sein sollen, sondern es wird rein um die Programmierung der bereits gebauten Roboter in Java gehen. Zu den Bauanleitungen wird auf diverse andere Büchern oder Links verwiesen.

Die Lernenden werden soweit fit gemacht, dass sie nach dem Durcharbeiten des Buches selbstständig andere Roboter programmieren können und sie bei auftretenden

Problemen wissen, bei welchem Beispiel sie nachblättern müssen.

Für die Beispiele wird der Problemlösungsansatz benutzt: Neue Themen werden anhand eines Problems, das gelöst werden muss, eingeführt.

5.6 Schwierigkeiten für EinsteigerInnen

„Novices are ;very local and concrete in their comprehension of programs'.“ [Robi03] Dementsprechend darf nicht von ihnen verlangt werden, bereits Verknüpfungen zu gerade erst Erlerntem selbstständig zu erstellen.

Objekte aus der Problemstellung zu identifizieren ist keine einfache Aufgabe, geschweige denn für AnfängerInnen, und jene Objekte, die identifiziert werden, spiegeln oft nicht die Realität wider. Im Fall der Roboter entfällt diese Aufgabe, denn jedes Robotermodell stellt automatisch auch immer eine eigene Klasse dar.

Erfahrene ProgrammiererInnen verwenden sowohl prozedurale als auch objektorientierte Ansätze und entscheiden sich je nach Situation für die passende Vorgehensweise [Robi03]. AnfängerInnen sind auf diese Vorgehensweisen beschränkt, die sie bis jetzt kennengelernt haben – und dies sind meist nicht viele.

Folgende fünf Schwierigkeiten müssen nach [Robi03] von ProgrammieranfängerInnen gemeistert werden: „(1) general orientation, what programs are for and what can be done with them; (2) the notional machine, a model of the computer as it relates to executing programs; (3) notation, the syntax and semantics of a particular programming language; (4) structures, that is, schemas/plans as discussed above; (5) pragmatics, that is, the skill of planning, developing, testing, debugging and so on.“

5.7 Titel des Buchs

Der Titel eines Buches soll ansprechend sein und die gewünschte Aufmerksamkeit erzeugen. [@buch]

Die zuvor analysierten Bücher werden aufgezählt, um eine vielleicht vorhandene Tendenz zu erkennen und Redundanzen zu vermeiden: „Java will nur spielen“, „Java von Kopf bis Fuß“, „Programmieren lernen: Eine grundlegende Einführung in Java“, „Objektorientierte Programmierung spielend gelernt mit dem Java-Hamster-Modell“, „Java lernen mit BlueJ: Eine Einführung in die objektorientierte Programmierung“, „LEGO®-Roboter: Bauen und programmieren mit LEGO®MINDSTORMS® NXT 2.0“, „Maximum LEGO NXT“.

„Vom Programmieren mit Blöcken zum Programmieren wie Erwachsene – Der Umstieg von LEGO zu Java.“ Dieser Titel lässt sich mit keinem der analysierten Bücher verwechseln und bringt das Wesentliche zum Ausdruck.

5.8 Inhaltsverzeichnis des Buchs

Das Konzept des Buches, welches als Ziel dieser Arbeit entsteht, soll nicht mit zu vielen technischen Details verwirren und vom Wesentlichen (nämlich dem Erlernen der Programmiersprache) ablenken.

1. Was du bereits können solltest!

Kapitel eins soll beschreiben, was der/die Lernende bereits kennen soll, um zu vermeiden, dass sich an dem Buch Interessierte überschätzen und sie es dann womöglich wieder zur Seite legen, da sie den Inhalt nicht schaffen können, weil sie das entsprechende Vorwissen noch nicht besitzen.

2. Was ist Java?

In kindgerechter und AnfängerInnen tauglicher Sprache soll hier beschrieben werden, was in Kapitel „Warum Java?“ bereits erläutert wurde.

3. Warum mit Java programmieren?

Hier soll genauer darauf eingegangen werden, welche Vorteile es bringt, mit Java die LEGO-MINDSTORMS-Roboter zu programmieren und wo die Grenzen der grafischen Programmieroberfläche sind. Die meisten Lesenden werden diese Gründe zu

diesem Zeitpunkt bereits selbst erfahren haben, jedoch soll hier auch abgegrenzt werden, was mit der Java-Programmierung möglich ist.

4. Installation eclipse/leJOS/firmware

Die Installation einer neuen Entwicklungsumgebung stellt selbst für bereits erfahrene ProgrammiererInnen des Öfteren eine Herausforderung dar, da häufig Umgebungsvariablen zu setzen sind und es mit einem einfachen Klick zur Installation meist nicht getan ist. Hierbei soll dieses Kapitel die Jugendlichen unterstützen und Schritt für Schritt an das Ziel, nämlich die fertig installierte Entwicklungsumgebung, heranführen.

5. Einführung eclipse

Nach einer erfolgreichen Installation ist es notwendig die wichtigsten Basics von eclipse zu erklären, damit sich die AnfängerInnen nicht in dieser neuen, im Vergleich zur grafischen Oberfläche, doch viel größeren Entwicklungsumgebung verirren.

6. Erstes Programm am Roboter ausführen

Abschnitt 6 wird eher kurz gehalten und stellt die Ausführung des ersten Java-Programmes für den Roboter dar. Das fertige Programm wird zur Verfügung gestellt und dient nur dem Zweck, das Starten separat von sonstigen Inhalten zu betrachten.

7. Variablen

Ein Grundgerüst, welches, wie der restliche Code des Buches, auf einem Server zur Verfügung gestellt werden wird, soll es den Lernenden einfacher ermöglichen, das Prinzip und die Syntax von Variablen und Datentypen kennenzulernen. Dafür wird ein fertiges Programm mit Objekten und einer Klasse erstellt, und die Lernenden müssen nur Ergänzungen vornehmen: wodurch der Objektbegriff gleich zu Beginn eingeführt wird, aber noch nicht näher erörtert, sondern vorerst einfach verwendet wird.

8. Berechnen (+, -, /, *)

Für diverse Beispiele ist es notwendig, dass die Lernenden auch Rechnungen ausführen können. Dieses Kapitel wird die Rechenoperatoren einführen.

9. Schalter

Bereits aus der grafischen Oberfläche sind die Schalter bekannt, weshalb dieses Kapitel auch nicht if/else-Anweisung oder dergleichen genannt wird, sondern Schalter, sodass die Lernenden den Bezug schneller herstellen können.

10. Schleifen und Abbruchbedingungen

Schleifen stellen in jeder Programmiersprache ein wichtiges Programmkonstrukt dar und müssen ebenfalls in diesem Buch erklärt werden. Das Verständnis von Schleifen ist bereits aus der grafischen Oberfläche bekannt und muss nun nur noch in der richtigen syntaktischen Schreibweise erlernt werden.

11. Objekte/Klassen

Es wurden bereits Klassen verwendet, jedoch bis zu dieser Stelle im Buch ohne nähere Erklärung. In diesem Kapitel soll nun das Objekt- und Klassenkonzept erläutert werden. Dieser Bereich wird in folgende Unterkapitel unterteilt:

I. Attribute und Methoden

Für die OOP ist es wichtig, dass verstanden wird, wozu Attribute und Methoden verwendet werden und wie sie eingesetzt werden. Deshalb bekommen sie ein eigenes Kapitel.

II. Konstruktor

Für AnfängerInnen der OOP ist es oft schwer zu verstehen, wozu ein Konstruktor da ist und was er alles kann, weshalb es gut ist, diese spezielle Methode gesondert zu behandeln.

III. this

Zu einem guten Programmierstil gehören auch das Verständnis und die Verwendung von this. Um den AnfängerInnen dies gleich von Anfang an begreiflich zu machen, sollen sie diesen Operator so früh wie möglich kennenlernen.

IV. Wozu dieses „import“ ?

Im Zusammenhang mit Klassen wird immer etwas importiert, und auch bei den Roboterbeispielen ist es notwendig, die entsprechenden leJOS-Packages zu importieren.

12. Zugriffsrechte

Zugriffsrechte spielen bei der Kapselung von Daten eine bedeutende Rolle und werden in diesem Kapitel behandelt. Der Unterschied zwischen den jeweiligen Zugriffsspezifikatoren wird gleich anhand von Beispielen gezeigt.

13. Pakete

Um eine übersichtlichere Programmierung für die AnfängerInnen zu ermöglichen, wird in diesem Kapitel die Paketverwaltung in Java gezeigt.

14. Vererbung

Unbedingt notwendig ist es, gerade bei den Roboterbeispielen die Vererbung zu verstehen, diese kann immer wieder sinnvoll eingesetzt werden. Anhand eines früheren Beispiels werden die Lernenden die Vererbung einsetzen und kennenlernen. Hierbei ist vorerst nur wichtig, das Prinzip der Vererbung verstanden zu haben und Unterklassen mit neuen Funktionalitäten ausstatten zu können.

I. super

Konstruktoren von Unterklassen sind für AnfängerInnen meist nicht gleich direkt begreifbar, daher ist es wichtig, dass der super-Aufruf gesondert und detaillierter behandelt wird.

II. Überschreiben von Methoden

Auch das Überschreiben von Methoden der super-Klasse spielt eine wichtige Rolle und wird eigens behandelt, um die Lesenden nicht auf einmal mit der Thematik der Vererbung und des Überschreibens zu überfordern.

15. Exceptionhandling

Fehler treten immer wieder auf, und werden sie nicht behandelt, entstehen in diversen Softwareprojekten große Probleme. Roboter müssen immer auf ihre Umwelt reagieren, und dabei können durch unvorhersehbare Ereignisse schnell Exceptions auftreten, welche von ProgrammiererInnen abgefangen werden müssen. Das dafür nötige Know-how wird in diesem Kapitel erklärt.

16. Konstante

Konstanten sind etwas sehr grundlegend Wichtiges in jeder Programmierung, vor allem bei Projekten mit Robotern kommen sie öfters zum Einsatz und sollen somit an dieser Stelle eingeführt werden. Das konzeptuelle Verständnis der Konstanten wird womöglich den Interessierten aus der grafischen Oberfläche schon bekannt sein, sofern der jeweilige Block dafür bereits verwendet wurde.

17. Konvertieren

Schon in der grafischen Programmieroberfläche von LEGO gibt es einen Block, der für das Konvertieren von Textdaten in Zahlendaten und umgekehrt da ist. Im letzten Kapitel des Buches soll die Typkonvertierung in Java durchgenommen werden.

Aus didaktischen Gründen wird auf gewisse weiterführende Bereiche der Objektorientierten Programmierung in Java verzichtet. Dazu zählen unter anderem: Erstellen eigener Exceptions, innere Klassen, Generics, grafische Oberflächen-Programmierung, Threading und alle Arten der Netzwerk- oder Datenbankkommunikation.

Die inhaltliche Gliederung ist systematisch aufeinander aufbauend und soll Interessierte langsam und Schritt für Schritt an immer schwerere Aufgaben heranführen.

Die Installation der benötigten Software soll nicht zum Anhang werden, da es als nicht weniger wichtig zu betrachten ist. Denn erst, wenn dieser Vorgang erfolgreich abgeschlossen werden konnte, können die Beispiele aus den übrigen Kapiteln programmiert werden.

Mit diesem inhaltlichen Gerüst sind die Lernenden in der Lage, auch selbstständig neue Projekte mit Java zu programmieren und die vorgestellten Programme zu erweitern. Ebenfalls können sie so auch sehr schnell und einfach andere Java-Programme ohne LEGO-MINDSTORMS-Roboter entwickeln.

5.9 Präsentation des Inhalts

Der Inhalt wurde klar abgegrenzt, so stellt sich nun die Frage, welche Form/Präsentation dafür verwendet wird. Komplexe Inhalte können sich Lernende nur schwer

merken, wenn sie nur die Fakten erfahren. Viel einfacher merken sie sich witzige, spannende Geschichten, die den Inhalt verpackt wiedergeben. Dieser Vorteil soll durch die Verwendung des narrativen Ansatzes genützt werden.

Beim narrativen Ansatz handelt es sich um eine konstruktivistische Lerntheorie, welche der Meinung ist, dass Menschen Geschichten erzählen und verwenden, um Dingen einen Sinn zu geben. Gewisse Sachverhalte bekommen ihre Bedeutung und den Zusammenhang erst durch die Erzählungen. Auch die Aufmerksamkeit der jungen Zielgruppe kann länger erhalten bleiben, wenn es um Geschichten geht, von denen die Lesenden gerne den Ausgang erfahren möchten. Jedoch wäre eine lange Geschichte über alle Kapitel des Buches einerseits schwer zu erfinden und andererseits ein Risiko, wenn diese Geschichte einen Teil der Zielgruppe nicht ansprechen würde. Deshalb erachte ich es als sinnvoll, zu jedem geeigneten Thema eine eigene Erzählung zu schreiben, die auch unabhängig von den anderen gelesen werden kann. Der rote Faden soll sich trotzdem vom Beginn bis zum Schluss des Buches durchziehen.

5.9.1 Narrativer Ansatz

In der Lehre wird der narrative Ansatz gewählt, um Inhalten durch Geschichten einen Sinn zu verleihen. Entweder die Lernenden erfinden selbst Geschichten, dann wird die Narration als Prozess betrachtet, auch Story Telling genannt, oder es wird eine fertige Geschichte eingesetzt, dann wird Narration als Produkt verwendet. Beim Erstellen dieses Buches soll Narration als Prozess verwendet werden.

Eine Geschichte muss handelnde Figuren beinhalten und in einer Welt erzählt werden, die eindeutig abgegrenzt werden kann. Das macht die Gestaltung eines Themas mit der narrativen Methode sehr aufwendig und zeitintensiv. Doch wurde einmal eine Geschichte gefunden, die gut den Inhalt vermitteln kann, so ist diese viel effektiver hinsichtlich des zu erzielenden Lernerfolges.

Narration bietet viele Vorteile: So ist mitunter die Motivation zum Zuhören bzw. Lesen sehr groß, und das Erzählen von Geschichten wirkt zusammenführend, Emotionen werden beim Zuhören geweckt, ebenso regen sie zum Nachdenken an. Doch auch

Nachteile sind bekannt: So kann es z.B. zu Unsachlichkeiten bei der Verwendung von Geschichten kommen. [@lehr]

Die narrative Methode kann als „Anchored Instruction“-Ansatz oder als „Goal-Based Scenario"-Ansatz zum Einsatz kommen. Beim „Anchored Instruction“-Ansatz sind die Geschichten abenteuerlich, und beim „Goal-Based Scenario"-Ansatz handelt es sich eher um realitätsnahe Geschichten [@lehr]. Beim konzeptuierten Buch werden sowohl der „Anchored Instruction"- als auch der „Goal-Based Scenario"-Ansatz zum Einsatz kommen, denn meiner Meinung nach bietet es keinen Vorteil, sich hier auf einen der beiden Ansätze zu beschränken. Soweit es möglich ist, werden die Themen des Buches in Geschichten verpackt, denn diese Erzählungen sind es, die im Gedächtnis bleiben und auch nach Jahren noch nicht vergessen sind.

6. Zusammenfassung und Ausblick

Die Programmierlehre wird in Zukunft in der Allgemeinbildung eine immer größere Rolle einnehmen, und dementsprechend werden noch viele unterschiedliche didaktische Methoden auf den Markt kommen, die den ProgrammieranfängerInnen den Einstieg erleichtern sollen. LEGO-MINDSTORMS-Roboter stellen in der Gegenwart und wohl auch in der Zukunft eine sehr gut geeignete Vorgehensweise dafür dar.

Durch die große Verbreitung von Java wird diese Programmiersprache auch die nächsten Jahre und womöglich auch noch ein paar Jahrzehnte eine bekannte und lebende Programmiersprache bleiben, weshalb es also Sinn macht, dass die Informatik-Jugend diese Sprache und Denkweise kennenlernt. Wenn Java eines Tages weniger oder keine Bedeutung mehr zukommt, so wird für die LEGO-MINDSTORMS-Roboter in der jeweiligen aktuellen Programmiersprache eine Möglichkeit geschaffen werden, die zur Programmierung dieser Roboter verhilft.

Die Zukunft der Roboter und auch der dafür geeigneten Programmiersprache scheint also gesichert zu sein. Inwieweit die Roboter mehr und weniger für das Lernen einer

textuellen Programmiersprache genutzt werden, hängt wohl auch stark vom zur Verfügung stehenden Lernmaterial ab. Das in dieser Arbeit erstellte Konzept eines ProgrammieranfängerInnen-Buches soll als Grundlage für dessen Erstellung dienen. Dieses Buch soll später für Interessierte eine gute Lernunterlage darstellen.

7. Literaturverzeichnis

Paper

[Barn02] D. J. Barnes: Teaching Introductory Java through LEGO MINDSTORMS Models. In: ACM, SIGSCE'02, 2002.

[Carl09] M. C. Carlisle: Raptor: A Visual Programming environment for teaching object-oriented programming. In: ACM Journal of Computing Sciences in Colleges, Vol. 24 Iss. 4, 275-281, 2009.

[Chen04] C. Hu: Rethinking of Teaching Objects-First. In: Education and Information Technologies, Vol.9 Iss. 3, 209-218, 2004.

[Coop03] S. Cooper, W. Dann, R. Pausch: Teaching Objects-first In Introductory Computer Science. In: ACM, SIGCSE '03, 191-195, 2003.

[Diet07] I. Diethelm: Strictly models and objects first – Unterrichtskonzept für objektorientierte Modellierung. In: Didaktik der Informatik in Theorie und Praxis, Informatik und Schule INFOS 07, Bonn, pp. 45-56, 2007.

[Ehle09] A. Ehlert, C. Schulte: Unterschiede im Lernerfolg von Schülerinnen und Schülern in Abhängigkeit von der zeitlichen Reihenfolge der Themen (OOP-First bzw. OOP-Later). In: Forschungs- und Entwicklungsprojekte zur Didaktik und Methodik des Informatikunterrichts, Informatik und Schule INFOS 09, Berlin, pp. 121-132, 2009.

[Fagi03] B. Fagin, L. Merkle: Measuring the Effectiveness of Robots in Teaching Computer Science. In: ACM, SIGCSE'03, 2003.

[Finc06] Sally Fincher et.al: Some good ideas from the disciplinary commons. In: 7th Annual Conference of Higher Education Acadamy for Information and Computer Sciences, 29-32 August 2006, Dublin, Irland, 2006.

[FINC99] S. Fincher: What are we doing when we teach programming? In: 29th

ASEE/IEEE Frontiers in Education Conference, 10-13 November 1999, San Juan, Porto Rico, 1999.

[Hund06] C. D. Hundhausen, S. Farley, J. Lee Brown: Can Direct Manipulation Lower the Barriers to Programming and Promote Positive Transfer to Textual Programming? An Experimental Study. In: IEEE, Visual Languages and Human-Centric Computing, 2006.

[Köll01] M. Kölling, J. Rosenberg: Guidelines for Teaching Object Orientation with Java. The Proceedings of the 6th conference on Information Technology in Computer Science Education (ITiCSE 2001), Canterbury, 2001.

[Köll04] M. Kölling, D. J. Barnes: Enhancing Apprentice-Based Learning of Java. In: ACM, SIGCSE'04, 2004.

[Köll99] M. Kölling: The Problem of Teaching Object-Oriented Programming. In: Journal of Object-Oriented Programming 11(8), 8-15, 1999.

[Lawh08] Lawhead et.al: A Road Map for Teaching Introductory Programming Using LEGO MINDSTORMS Robots. In: ACM, SIGCSE08, 2008.

[Nour05] I. R. Nourbakhsh et. al: The Robotic Autonomy Mobile Robotics Course: Robot Design, Curriculum Design and Education Assessment. In: Autonomous Robots 18, 103-127, Springer Science + Business Media, Inc. Manufacturedin The Netherlands, 2005.

[Robi03] A. Robins, J. Rountree, N. Rountree: Learning and Teaching programming: A Review and Discussion. In: Computer Science Education 2003, Vol. 13, No. 2, 137-172, 2003.

[Solo93] E. Soloway: Should we teach Students to Program? In: Communication of the ACM 1993/Vol. 36, No. 10, 1993.

[Wied96] S. Wiedenbeck, J. Scholtz: Adaption of programming plans in transfer

between programming languages: A developmental approach. In: Empirical Studies of Programmers: Sixth Workshop, Ablex Publishing Corporation, 1996.

[Yuan11] Yuan Li, Kuo-Chung Chang: Exploring the Dimensions and Effects of Computer Software Similarities in Computer Skills Transfer. In: Journal of Organizational and End User Computing, Vol. 23, 48-66, 2011.

Websites

[@bric] Next Byte Codes, Not eXactly C, and SuperPro C (abgefragt am 14.02.2014)
http://bricxcc.sourceforge.net/nbc/

[@buch] Book Designs: Einen guten Buchtitel finden. (abgefragt am 15.01.2014)
http://www.buchveroeffentlichen.com/einen-guten-buchtitel-finden/

[@java] Java (abgefragt am 07.12.2013)
www.java.com

[@jax] jaxenter – Software & Support Media GmbH (abgefragt am 07.12.2013)
http://jaxenter.de/artikel/Warum-alle-Java-hassen-0

[@oop] Wikipedia: Objektorientierte Programmierung. (abgefragt am 07.07.2014)
http://de.wikipedia.org/wiki/Objektorientierte_Programmierung

[@unid] Das Entwicklungsstufenmodell nach Piaget. Universität Duisburg-Essen (abgefragt am 15.01.2014)
https://www.uni-due.de/edit/lp/kognitiv/piaget.htm

[@wiss] Wissenstransfer-Blog, M. Spitzer: Lernen macht glücklich. (abgefragt am 15.01.2014)

http://www.wissenstransfer-blog.de/training-e-learning/lernen-macht-glucklich.html

[@Wahl06] D. Wahl: Ergebnisse der Lehr-Lern-Psychologie. 2006. (abgefragt am 14.02.2014) http://www.dblernen.de/docs/Wahl_Ergebnisse-der-Lehr-Lern-Psychologie.pdf

[@lehr] lehrer-online: Unterrichten mit digitalen Medien (abgefragt am 02.03.2014) http://www.lehrer-online.de/narration-geschichte.php

[@para] Wikipedia: Programmierparadigma. (abgefragt am 07.05.2014) http://de.wikipedia.org/wiki/Programmierparadigma

Diplomarbeiten

[Berg09] K. Berger: Begabtenförderung mit LEGO MINDSTORMS. Diplomarbeit TU Wien, 2009.

[Dian11] M. Di Angelo: Robolab: LEGO MINDSTORMS im AHS-Informatikunterricht. Magisterarbeit TU Wien, 2011.

[Löwe09] B. Löwenstein: Vermittlung Objektorientierter Konzepte mittels LEGO MINDSTORMS. Magisterarbeit TU Wien, 2009.

[Schm09] Peter Schmidt: Einsatz von LEGO MINDSTORMS im Unterricht der HTL für Informationstechnologie. Diplomarbeit TU Wien, 2009.

[Will10] V. Willner: Wie gut eignen sich gängige Tools, um Schülern Java zu l ehren? Magisterarbeit TU Wien, 2010.

Bücher

[Bagn13] B. Bagnall: Maximum LEGO NXT Third Edition. Variant Press, 2012.

[Barn12] D. J. Barnes, M. Kölling: Java lernen mit BlueJ: Eine Einführung in die objektorientierte Programmierung. Pearson Deutschland GmbH, 2012.

[Bern10] K. Berns, Daniel Schmidt: Programmierung mit LEGO MINDSTORMS NXT. Springer-Verlag GmbH, 2010.

[Bole10] D. Boles: Objektorientierte Programmierung spielend gelernt mit dem Java-Hamster-Modell. Vieweg+Teubner |GWV Fachverlage GmbH, 2010.

[Bole08] D. Boles: Programmieren spielend gelernt mit dem Java-Hamster-Modell. Vieweg+Teubner |GWV Fachverlage GmbH, 2008.

[Köll10] M. Kölling: Einführung in Java mit Greenfoot. Pearson Deutschland GmbH, 2010.

[Mand05] H. Mandl: Handbuch Lernstrategien. Hogrefe Austria GmbH, 2005.

[Pani08] S. E. Panitz: Java will nur spielen: Programmieren lernen mit Spaß und Kreativität. Vieweg+Teubner | GWV Fachverlage GmbH, 2008.

[Pepp07] P. Peppert: Programmieren lernen: Eine grundlegende Einführung in Java. Springer-Verlag Berlin Heidelberg, 2007.

[Reic05] R. Reichert, J. Nievergelt, W. Hartmann: Programmieren mit Kara – ein spielerischer Zugang zur Informatik. Springer-Verlag GmbH, 2005.

[Schif01] S. Schiffer: Visuelle Programmierung. Grundlagen und Einsatzmöglichkeiten. Addison Wesley Verlag| Pearson Deutschland GmbH, 2001.

[Sier06] K.Sierra, B.Bates: Java von Kopf bis Fuß. O'Reilly Verlag GmbH &

Co.KG, 2006.

Sonstiges

[Edel13] S. Edelkamp: Praktische Informatik I – Der Imperative Kern; Didaktische Einführung. Tzi Technologiezentrum-Zentrum Informatik und Informationstechnologien, Universität Bremen, 2013.

[Jäge07] A. Jäger: Lernansatz: „Objects First”. Hauptseminar „Didaktik der Informatik" im WS 2007/08 an der Universität Innsbruck, Leitung: Prof. Dr. Peter Hubwieser, 2007.

[SPOL99] S. Spolwig: „Hello World" in OOP. In: LOG IN, 19 (1999) Heft 5, 1999.

[Wagn05] O. Wagner: LEGO-Roboter im Informatik Unterricht. 2. Schulpraktisches Seminar Treptow-Köpenick (Berlin), Schriftliche Prüfungsarbeit im Rahmen der zweiten Staatsprüfung für das Amt des Studienrates, 13.09.2005.

8. Abbildungsverzeichnis

9. Tabellenverzeichnis