

Strukturiertes aktives Lernen von Algorithmen mit interaktiven Visualisierungen

Dissertation

zur Erlangung des Grades
eines Doktors der Ingenieurwissenschaften
am Fachbereich Informatik
der Carl von Ossietzky Universität Oldenburg

von

Dipl.-Inf. Nils Faltin

Gutachter: Prof. Dr.-Ing. Peter Gorny
Prof. Dr. Günther Stiege
Prof. Dr. Wolfram Luther

Tag der Disputation: 31.5.2002

Danksagung

Mein herzlicher Dank gilt allen, die mich bei meiner Doktorarbeit unterstützt haben:

- Meinem Doktorvater Prof. Dr. Peter Gorny für den großen Freiraum bei der Gestaltung der Doktorarbeit, die guten Arbeitsmöglichkeiten in seiner Abteilung und dafür, dass er mir das Gefühl gab, an einem wertvollen Thema zu arbeiten.
- Prof. Dr. Günther Stiege und Prof. Dr. Wolfram Luther für das Interesse an meiner Arbeit und die Übernahme des Gutachtens.
- Ingvor, Paul und Günter Faltin für die emotionale und materielle Unterstützung.
- Der Heinz Neumüller Stiftung Oldenburg für das Stipendium.
- Den Diplomanden und studentischen Hilfskräften Karsten Block, Jan Schormann und Tobias Gross für die fachliche Diskussion und die Arbeit an den Lernprogrammen.
- Prof. Dr. Sonnenschein für die Möglichkeit, die Lernprogramme in seiner Lehrveranstaltung zu erproben. Den Lehrenden und Studierenden für ihre Anmerkungen auf den Fragebögen.
- Den Korrekturlesern Gernot Lorz, Ines Faltin, Marco Eichelberg und Guido Rößling, die Fehler und Schwachstellen beheben halfen.
- Und nicht zuletzt meiner Frau Ines und meinem Sohn Elias, dass sie bereit waren, so manchen Tag auf mich zu verzichten und mir immer den Rücken gestärkt haben.

Kurzfassung

Bei der Softwareentwicklung müssen immer wieder Grundprobleme, wie z. B. Sortieren, Suchen und das Finden kürzester Wege, gelöst werden. Dafür stellt die Informatik eine reiche Auswahl an Algorithmen und Datenstrukturen bereit.

Da Algorithmen oft schwer zu verstehen sind, ist es wichtig, sie für die Informatik-Ausbildung didaktisch gut aufzubereiten. Diese Arbeit stellt eine neue Methode für die Vermittlung von Algorithmen vor, die auf dem Konzept des entdeckenden Lernens und einer starken Modularisierung des Algorithmus basiert. Die Studierenden lernen einen Algorithmus mithilfe interaktiver visueller Simulationen, die in Lehrtexte eines Lernprogramms eingebettet sind. Es ist die Aufgabe der Studierenden, eine korrekte Schrittfolge für den Algorithmus zu finden.

Für Autoren von Lernprogrammen wurde eine Gestaltungsmethode ausgearbeitet, die ein Grundgerüst für die Gliederung der Programme und die Funktionsweise der darin enthaltenen interaktiven Simulationen von Algorithmenteilen beschreibt. Als Beispiel wird die Gestaltung eines Lernprogramms zu Heapsort vorgestellt. Weitere Hinweise zur software-ergonomischen und didaktischen Gestaltung solcher Simulationen werden als Gestaltungsregeln bereitgestellt. Die im Rahmen dieser Arbeit entwickelte Bibliothek SALABIM erleichtert die Implementierung der interaktiven Simulationen.

Zwei Lernprogramme zu den Themen Heapsort und Binomial Heap sind nach der Gestaltungsmethode entwickelt worden. Sie wurden im Grundstudium der Informatik eingesetzt und erprobt. Studierende und Lehrende bewerteten über Fragebögen die Lernprogramme und das dahinter stehende didaktische Konzept. Insgesamt kamen sie zu einer positiven Einschätzung. Einsichten in die Nutzung der webbasierten Lernprogramme lieferte eine Analyse der Logdateien des Webservers. So konnten das Navigationsverhalten, die Intensität der Nutzung und das Verhältnis von Online- zu Offline- Nutzern ermittelt werden.

Abstract

During software development basic problems like sorting, searching and finding a shortest path must be solved. Computer science provides a wealth of algorithms and data structures for such problems.

As algorithms can be difficult to understand it is important for computer science education to prepare their presentation well. This thesis presents a new method for the teaching of algorithms through courseware. It is based on the concept of exploratory learning and on strong modularization of the algorithm. Students learn an algorithm with the help of interactive visual simulations that are contained in courseware tutorial text. It is the student's task to find a correct sequence of steps for the algorithm.

For authors of courseware a design method has been developed. It describes the framework of the courseware and the functioning of the interactive simulations contained therein. As an example the design of a courseware on the heapsort algorithm is described. Additionally didactic and usability oriented design rules for the simulations are provided. The software library SALABIM developed in this project eases the implementation of the interactive simulations.

Two courseware on heapsort and binomial heap have been developed following the design method. They have been evaluated in a first year university computer science course. Students and teachers assessed the courseware and its didactic concept on questionnaires. Overall the assessment was positive. The use of the web based courseware was analyzed. So the user navigation behavior, intensity of use and proportion of online to offline users could be determined.

Inhaltsverzeichnis

Kurzfassung	3
Abstract	3
Abbildungsverzeichnis	9
1 Einleitung	11
1.1 Ansatz dieser Arbeit	11
1.2 Historische Entwicklung: Algorithmen in grafischen Lernmedien	12
1.2.1 Statische Bilder und mechanische Lehrmittel	12
1.2.2 Neue Möglichkeiten im Medium Film	13
1.2.3 Interaktive Visualisierung	14
1.3 Akzeptanz und Effektivität von Algorithmen-Animationen	15
1.3.1 Verbreitung und Akzeptanz	15
1.3.2 Untersuchungen zur Effektivität	16
1.4 Übersicht über die folgenden Kapitel	18
2 Lerngegenstand „Algorithmus“	19
2.1 Aspekte eines Algorithmus	19
2.1.1 Der Algorithmusbegriff	19
2.1.2 Korrektheit	20
2.1.3 Effizienz	20
2.1.4 Datenstruktur	20
2.1.5 Trace	20
2.2 Darstellung des Verfahrens	21
2.2.1 Pseudocode oder Programmcode	21
2.2.2 Modularität	21
2.2.3 Algorithmen und Entwurfsprinzipien	21
3 Verwandte Arbeiten zum aktivem Lernen von Algorithmen	23
3.1 Überblick	23
3.1.1 Lerner führen Algorithmus aus	23
3.1.2 Lerner entwickeln Algorithmen-Animation	24
3.1.3 Lerner programmieren Algorithmus	25
3.2 Visuelle Programmierung mit OPSIS	26
3.2.1 Übersicht	26
3.2.2 Notation der visuellen Programmiersprache	26
3.2.3 Programme sind nicht strukturierbar	26
3.2.4 Das Werkzeug OPSIS	27
3.2.5 Erprobung der Lehrmethode	28
3.2.6 Bewertung des Ansatzes	28
3.3 Studierende entwickeln und präsentieren Animationen	29
3.3.1 Lerntheoretischer Hintergrund	29
3.3.2 Erprobung der Lehrmethode	29
3.3.3 Das Visualisierungswerkzeug ALVIS	30
3.3.4 Bewertung der Arbeiten Hundhausens	31
4 Die Gestaltungsmethode „Strukturiertes Aktives Lernen von Algorithmen (SALA)“	33
4.1 Einführung	33
4.1.1 Das Potential multimedialer Lernprogramme	33
4.1.2 Die Methode SALA	33
4.1.3 Vom Lehrbuch zum Lernprogramm	34

4.1.4	Gestaltung nach SALA am Beispiel des Heapsort	34
4.2	Lernpsychologische Grundlagen	35
4.2.1	Entdeckendes versus rezeptives Lernen.....	35
4.2.2	Entdeckendes Lernen nach Bruner	36
4.2.3	Vertiefung in eine Aufgabe	37
4.2.4	Intrinsische Motivation	37
4.3	SALA: Gliederung des Lernprogramms in Sektionen	39
4.4	Funktionale Struktur eines Algorithmus	40
4.4.1	Methodische Grundlage: Algorithmen als modulare Software	40
4.4.2	Funktionale Struktur in SALA	42
4.4.3	Reihenfolge der Funktionen	43
4.4.4	Heapsort: Struktur, Reihenfolge und Schnittstellen.....	43
4.5	Erlernen einer Funktion.	44
4.5.1	Lernphasen beim entdeckenden Lernen	44
4.5.2	Modulare Darstellung der Funktion	45
4.5.3	Heapsort: Build-Heap Funktion	46
4.6	Interaktive Simulation einer Algorithmen-Funktion	49
4.6.1	Die interaktive Simulation	49
4.6.2	Heapsort: Aufgabe zur Funktion Build-Heap	49
4.7	Simulation und Animation mit SALABIM-Applets	49
5	Gestaltungsregeln für interaktive Visualisierungen von Algorithmen	53
5.1	Algorithmen-Animationen	53
5.1.1	Gestaltungsregeln für Algorithmen-Animationen.....	53
5.1.2	Beispiele für Algorithmen-Animationen	54
5.2	Algorithmen-Übungen	55
5.2.1	Gestaltungsregeln für Algorithmen-Übungen	55
5.2.2	Beispiele für Algorithmen-Übungen.....	55
5.3	Algorithmen-Simulationen	56
5.3.1	Gestaltungsregeln für Algorithmen-Simulationen	56
5.3.2	Beispiele für Algorithmen-Simulationen.....	57
6	Software-Bibliothek SALABIM	59
6.1	Einführung	59
6.2	Anforderungen an die Bibliothek	59
6.3	Beispielapplet Simple	60
6.4	Java-Swing einbinden	60
6.5	SALABIM als Framework	61
6.6	Kontrollfluss im Applet	61
6.6.1	Initialisierung.....	61
6.6.2	Interaktionsschleife.....	63
6.7	Klassen eines SALABIM-Applets	64
6.7.1	Klassendiagramm.....	64
6.7.2	Die Klassen	65
6.7.3	SimpleModel Programmcode	66
6.8	Beispielapplet „Blöcke stapeln“	67
6.9	Verwandte Arbeiten zu Undo/Redo	68
7	Studierende bewerten SALA-Lernprogramme und SALA	69
7.1	Einführung	69
7.1.1	Eingesetzte Lernprogramme	69
7.2	Lehrbetrieb	70
7.2.1	Lehrveranstaltung Datenstrukturen	70
7.3	Gestaltung der Evaluation	71
7.3.1	Wahl der Evaluationsmethode.....	71
7.3.2	Gestaltung der Erhebung	72

7.3.3	Vortest von Lernprogrammen und Fragebögen	73
7.4	Auswertung der Fragebögen	74
7.4.1	Abgegebene Fragebögen.....	74
7.4.2	Studienfach der Studierenden.....	74
7.4.3	Vorwissen über den Algorithmus.....	74
7.4.4	Übung im Umgang mit Webbrowsers.....	75
7.4.5	Technische Probleme mit dem Lernprogramm	75
7.4.6	Gesamteinschätzung des Lernprogramms.....	75
7.4.7	Punkte für die Aufgaben	76
7.4.8	Gesamteinschätzung von SALA.....	76
7.4.9	Einschätzung didaktischer Punkte.....	77
7.5	Auswertung der Anmerkungen auf den Fragebögen	78
7.6	Anmerkungen der Lehrenden auf den Fragebögen	79
7.7	Anmerkungen der Studierenden zu Heapsort	80
7.8	Anmerkungen der Studierenden zu Binomial Heap	83
8	Analyse der Nutzung zweier SALA-Lernprogramme	87
8.1	Einführung	87
8.2	Struktur der Lernprogramme	87
8.2.1	Blöcke bei Heapsort	87
8.2.2	Blöcke bei Binomial Heap.....	90
8.2.3	Standardpfad.....	90
8.2.4	Verzweigungspunkte für die freie Navigation	90
8.3	Ermittlung der Nutzungsdaten	91
8.3.1	Technische Basis der Lernprogramme.....	91
8.3.2	Auswertung von Logdaten.....	92
8.4	Visualisierung der Nutzungsdaten	93
8.5	Analyse der Nutzungsdaten	96
8.5.1	Das Navigationsverhalten.....	96
8.5.2	Die Intensität der Nutzung.....	97
8.5.3	Das Verhältnis von Online-Nutzung zu Offline-Nutzung.....	98
8.5.4	Zusammenfassung.....	98
9	Zusammenfassung und Ausblick	99
9.1	Zusammenfassung	99
9.2	Ausblick	99
9.2.1	Lernprogramme für weitere Algorithmen.....	99
9.2.2	Erweiterung von SALABIM.....	99
9.2.3	Wissensbasierte tutorielle Hilfe	100
9.2.4	Evaluation der Lernprogrammen und des didaktischen Konzepts	100
9.2.5	Aktives und soziales Lernen.....	100
9.3	Zukunftsaussichten für interaktive Lernmedien	100
10	Anhang: Unterlagen zur Evaluation	101
10.1	Aufgabenblatt zu Heapsort.	101
10.2	Aufgabenblatt zu Binomial Heap	102
10.3	Fragebogen zu Heapsort	103
10.4	Fragebogen zu Binomial Heap.....	104
10.5	Antwortdaten der Fragebögen	105
10.6	Anmerkungen der Nutzer auf den Fragebögen	108
11	Quellenverzeichnis	113
11.1	Quellen.....	113
Index		117
Lebenslauf		121

Abbildungsverzeichnis

1	Interaktive Simulation eines Algorithmus	12
2	Double-Ended-Queue	12
3	Heapsort-Vertauschungen	13
4	Sorting out Sorting	13
5	BALSA insertion sort	14
6	Animated Algorithms	14
7	Stepwise Refinement bei „Algorithms in Action“	23
8	Heapsort im Modus Üben bei „Algorithms in Action“	24
9	Endlicher Automat in JFLAP	24
10	ANIMAL - Interaktiver Editor für Animationen	25
11	Visuelles Programm für Suche im Binärbaum	26
12	OP SIS im Editiermodus	27
13	OP SIS im Tracemodus	27
14	Erstellen der Bubblesort-Animation in ALVIS	31
15	Präsentieren der Animation in ALVIS	31
16	Heapbaum mit markiertem Knoten	35
17	Gliederung des Lernprogramms zu Heapsort	40
18	Funktionale Struktur des Heapsort	43
19	Gliederung der Funktionenseite	45
20	Funktion Build-Heap im Lernprogramm zu Heapsort (oberer Teil der Seite)	47
21	Funktion Build-Heap im Lernprogramm zu Heapsort (unterer Teil der Seite)	48
22	Simulation des Build-Heap	49
23	Bedienelemente eines SALABIM-Applets	50
24	Ablaufsteuerung eines SALABIM-Applets	51
25	Lernen mit Animationen: Bubblesort	54
26	Lernen durch Übung: Verschmelzen-Funktion	56
27	Simulation zu Heapify-Locally aus Heapsort	57
28	Standardverfahren Heapify-Locally	57
29	Alternatives Verfahren Heapify-Locally	57
30	Simple - Beispielapplet zu SALABIM	60
32	Initialisierung des Applets in SALABIM	61
31	Interaktionsschleife im SALABIM-Applet	62
33	History-Liste der Call-Objekte	63
34	Klassen eines SALABIM-Applets	65
35	GUI eines SALABIM-Applets	65
36	NumberCall::perform Programmcode	66
37	SimpleModel::setNumberSelected Programmcode	66
38	NumberCall::equalsCall Programmcode	66
39	SimpleModel::finish Programmcode	67
40	Beispielapplet: Blöcke stapeln	67
41	Startseite Binomial Heap	69
42	Seitenstruktur des Lernprogramms zu Heapsort zur Zeit der Evaluation	88
43	Seitenstruktur des Lernprogramms zu „Binomial Heap“	89
44	Nutzung des Lernprogramms Heapsort	94
45	Nutzung des Lernprogramms zu „Binomial Heap“	95

1 Einleitung

1.1 Ansatz dieser Arbeit

Bei der Softwareentwicklung müssen immer wieder Grundprobleme, wie z.B. Sortieren, Suchen und das Finden kürzester Wege, gelöst werden. Dafür stellt die Informatik eine reiche Auswahl an Algorithmen und Datenstrukturen bereit. Sie werden typischerweise im ersten Jahr des Informatikstudiums in der Vorlesung „Algorithmen und Datenstrukturen“ behandelt. Auch in der Mathematik spielen Algorithmen eine wichtige Rolle. Einfache mathematische Algorithmen, wie das schriftliche Addieren, werden bereits in der Grundschule eingeführt.

Da Algorithmen oft schwer zu verstehen sind, ist es wichtig, dass sie für die Vermittlung in der Informatik-Ausbildung didaktisch gut aufbereitet werden. Diese Arbeit stellt eine neue Methode für die Vermittlung von Algorithmen vor, die auf dem Konzept des entdeckenden Lernens und einer starken Modularisierung des Algorithmus basiert. Die Studierenden lernen einen Algorithmus mithilfe interaktiver visueller Simulationen, die in Lehrtexte eines Lernprogramms eingebettet sind.

Durch die Arbeit werden für die Fachdidaktik der Informatik zwei wesentliche Innovationen erreicht:

- Die im Rahmen dieser Arbeit entwickelten Lernprogramme erlauben Lernern einen neuartigen, motivierenden und kreativen Zugang zum Lernen von Algorithmen.
- Die in dieser Arbeit vorgestellte Gestaltungsmethode mit der Sammlung von Gestaltungshinweisen und der Softwarebibliothek SALABIM erleichtern es zukünftigen Autoren didaktisch und softwareergonomisch ansprechende Lernprogramme zu Algorithmen zu erstellen.

Forschungsgebiet Algorithmen-Animation

Dozenten und Lehrbuchautoren verwenden seit langem grafische Darstellungen der Datenstruktur, um die Arbeitsweise eines Algorithmus zu erklären. Da ein Algorithmus während seiner Laufzeit die Daten verändert, werden sie zumeist in einer Folge statischer Bilder dargestellt.

Das Forschungsgebiet der Algorithmen-Animation beschäftigt sich mit der Erstellung und Bewertung von Trickfilmen, die die Veränderung der Daten darstellen. Dabei wurden bisher vor allem Softwarebibliotheken und Autorensysteme entwickelt, die die Erstellung von Algorithmen-Animationen erleichtern. Hingegen wurde wenig untersucht, welche Eigenschaften eine Animation aufweisen muss, damit der Algorithmus von Studierenden gut verstanden wird. Auch gehen die bisherigen Systeme zumeist von einer Präsentation des Algorithmus als Animation aus, bei der die Studierenden nur die Rolle eines Betrachters haben.

Entdeckendes Lernen in einer Mikrowelt

Diese Arbeit präsentiert eine neue Methode für das Lernen von Algorithmen, in der die Studierenden eine aktive Rolle einnehmen. Während bestimmte Teile des Algorithmus den Studierenden in einem hypermedialen tutoriellen Teil präsentiert werden, sollen sie andere Teile selbst entwickeln. Als Arbeitsmittel erhalten sie dazu eine interaktive grafische Simulation des Algorithmus mit der sie experimentieren können. Da die Lösung in der Simulation verborgen ist und quasi nur gefunden werden muss, spricht man von *entdeckendem Lernen*. Die Simulation bildet für die Studierenden eine eigene kleine Welt (sogenannte *Mikrowelt*) in der sie für eine gewisse Zeit handeln und Erfahrungen sammeln können.

In der Mikrowelt können im Allgemeinen von einer Situation aus verschiedene Schritte gegangen werden. Diese Vielfalt macht den Reiz einer Simulation aus, kann aber Studierende auch überfordern. Sollten Studierende an der Aufgabe scheitern, so können sie eine Musterlösung abrufen. Die Simulation zeigt dann eine Algorithmus-Funktion als Pseudocode und führt die Lösungsschritte automatisch aus. Anschließend können die Studierenden die Schritte der Musterlösung selbst ausführen. Dabei achtet die Simulation darauf, dass nur die Schritte der Musterlösung ausgeführt werden, so dass in jeder Situation nur ein Schritt möglich ist.

Phasen einer Simulation

Technisch gesehen arbeitet die Simulation wiederholt die folgenden Phasen ab:

1. **Visualisierung:** Die Simulation stellt die Datenstruktur des Algorithmus grafisch dar.
2. **Interaktion:** Der Studierende wählt mit dem Zeigegerät Elemente der Datenstruktur und eine Algorithmus-Funktion, die auf diesen Elementen gestartet werden sollen.
3. **Regelprüfung:** Die Simulation prüft, ob diese Algorithmus-Funktion jetzt ausgeführt werden darf.
4. **Ausführung:** Falls erlaubt wird die Funktion ausgeführt und die Datenstruktur entsprechend verändert.

Visualisierung und Interaktion bilden die Benutzungsoberfläche zur Regelprüfung und Ausführung (Abb. 1).

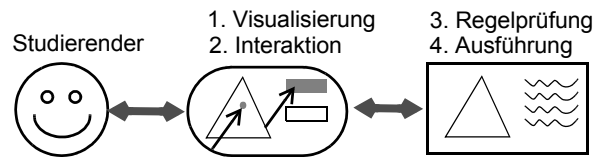


Abb. 1: Interaktive Simulation eines Algorithmus

Diese Phasen werden bisher von den Studierenden mit Papier und Stift bearbeitet, wenn sie als Hausaufgabe den Ablauf eines Algorithmus durchgehen. Demgegenüber übernimmt eine Simulation die Phasen Visualisierung und Ausführung, so dass sich der Studierende auf die eigentliche Arbeitsweise des Algorithmus konzentrieren kann. Die Phase Regelprüfung verhindert, dass der Studierende eine fehlerhafte Lösung weiter bearbeitet.

1.2 Historische Entwicklung: Algorithmen in grafischen Lernmedien

Im Folgenden sollen wichtige Phasen in der Entwicklung grafischer Darstellungen von Algorithmen aufgezeigt werden. Neue Techniken wie Lehrfilme oder interaktive Animationen brachten neben neuen Möglichkeiten auch neue Herausforderungen für einen didaktisch sinnvollen Einsatz als Lernmedium. Die Kenntnis dieser Zusammenhänge kann helfen, die neue Lehrmethode gegenüber den bestehenden Lehrmethoden und ihren Medien einzuordnen. Umfassende Übersichten über Algorithmen-Visualisierungen und Systeme zur Erstellung solcher Visualisierungen bieten der Sammelband [SDBP 1998] und die Dissertation von Rößling [Rößling 2001b].

1.2.1 Statische Bilder und mechanische Lehrmittel

Bereits in frühen Lehrbüchern zu Algorithmen wird die Datenstruktur grafisch dargestellt. Ein besonders schönes Beispiel findet sich in Abbildung 2. Hier wird das Bild eines Rangierbahnhofs als Metapher verwendet, um die Datenstruktur Double-Ended-Queue (deque) zu illustrieren. Sie ist eine Verallgemeinerung einer Warteschlange und eines Stapels. Elemente können jeweils am Anfang und Ende der Liste angefügt und entnommen werden. Die Darstellung lädt zum spielerischen Erkunden ein. Man kann als Betrachter mit dem Finger die Gleise entlangfahren und an den Weichen ver-

schiedene Wege wählen. Nimmt man kleine Gegenstände wie z.B. verschiedenfarbige Reißzwecken als Symbol für Elemente, so kann man damit die Double-Ended-Queue visualisieren und simulieren.

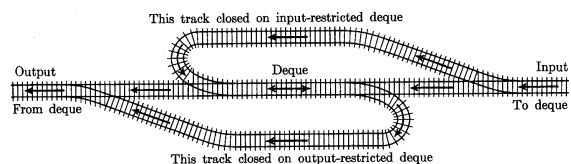


Abb. 2: Double-Ended-Queue.
[Knuth 1969], S. 236

In der Regel findet man aber eine nüchterne Darstellung der Datenstruktur vor. Oft wird in einer Bildfolge die Veränderung der Datenstruktur gezeigt. Ein typisches Beispiel ist Abbildung 3, die Momentaufnahmen aus dem Heapsort-Algorithmus zeigt. Es werden dort solange Schlüssel vertauscht, bis die Heapbedingung hergestellt ist. Zum besseren Verständnis habe ich im Bild die Bereiche markiert, die verändert werden oder verändert wurden.

Mechanische Lehrmittel

Will man als Leser selbst einen Ablauf durchspielen, so wird man in der Regel die Grafiken für mehrere Momentaufnahmen zeichnen müssen. Alter-

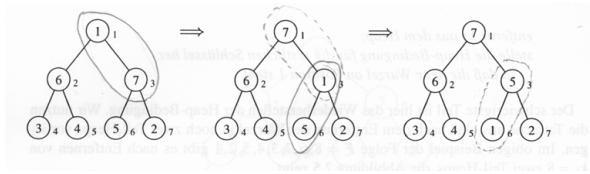


Abb. 3: Heapsort-Vertauschungen
[Ottmann, Widmayer 1996], S. 92
Hervorhebungen durch N.F.

nativ kann man ein *mechanisches Lehrmittel* herstellen, wie ich es für Heapsort getan habe. Für die Schlüssel schnitt ich kleine Quadrate aus Pappe aus, die auf dem festen Hintergrund eines binären Baumes mit leeren Knoten verschoben werden können.

1.2.2 Neue Möglichkeiten im Medium Film

Der Ablauf eines Algorithmus kann auch als Film dargestellt werden. Analog zur Trickfilm-Produktion spricht man dann von einer *Algorithmen-Animation*. Zum einen erlaubt dies, viel mehr Zwischenzustände der Datenstruktur zu zeigen, als es bei einer Bildfolge auf Papier möglich ist. Zum anderen kann der Übergang zwischen zwei Zuständen der Datenstruktur mit weichen Bewegungen dargestellt werden. Zum Beispiel kann man bei einem Sortieralgorithmus zwei als Balken dargestellte Schlüssel in einer weichen Bewegung die Plätze tauschen lassen. Auch ist es einfacher die Aufmerksamkeit des Betrachters auf die entscheidenden Stellen der Datenstruktur zu lenken, indem man z.B. Elemente kurz aufblinken lässt.

Meilenstein: „Sorting out Sorting“

Der Lehrfilm „Sorting out Sorting“ von 1981 gilt heute als Meilenstein in der Entwicklung des Gebietes Algorithmen-Animation ([Baecker 1981], [Baecker 1998]). In dem 30-minütigen vertonten Farbfilm werden 9 Sortieralgorithmen vorgestellt und miteinander verglichen. In Abb. 4 ist der „große Wettlauf“ der 9 Sortieralgorithmen zu sehen, der am Ende des Filmes gezeigt wird. Jeder Algorithmus hat dort ein ungeordnetes Array mit 2500 Elementen zu sortieren. Das Array ist als Streugraph dargestellt: die horizontale Achse entspricht den Indizes und die Höhe der Punkte entspricht dem unter dem Index gespeicherten Schlüsselwert. Ein ungeordnetes Array erscheint als Punktwolke, ein sortiertes Array als Linie von links unten nach rechts oben. Bei den langsamen $O(n^2)$ Algorithmen verändert sich die Punktwolke während des

Wettlaufes nur wenig. Bei den schnellen $O(n \log n)$ -Algorithmen bildet sich schnell die Linie. So kann der Betrachter den Unterschied in der Laufzeit gut wahrnehmen. Gut zu erkennen sind auch die Rechtecke, die sich bei Quicksort bilden. Sie entstehen zwischen Pivotelementen, die ihren Platz gefunden haben.

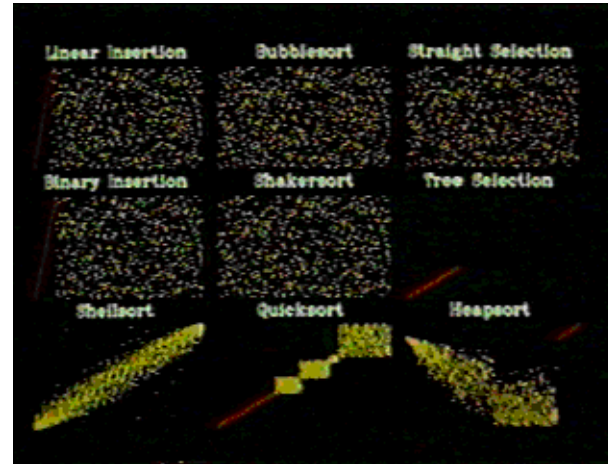


Abb. 4: Sorting out Sorting
[Baecker 1981]; entnommen aus [Price, Baecker, Small 1993]

Im Gegensatz zu vielen späteren interaktiven Animationen wurde dieser Lehrfilm didaktisch sorgfältig gestaltet. Die Animationen werden von einem Sprecher erklärt, wobei aktive Elemente hervorgehoben werden. Während der Erklärung eines Algorithmus werden die Schritte der Animation langsam abgespielt, damit die Betrachter den Algorithmus verstehen können. Danach wird das Tempo gesteigert, damit der Algorithmus schneller zum Ende kommt, und die Betrachter nicht durch immer gleiche Schritte gelangweilt werden.

Nachteil des Mediums Lehrfilm

Ein wesentlicher Nachteil des Mediums Lehrfilm (und teilweise auch der späteren interaktiven Animationen) ist die passive Rolle der Studierenden. Es ist üblich, dass Studierende als Hausaufgabe den Ablauf eines Algorithmus als Bildfolge zeichnen sollen. Auch können sie mit statischen Bildern die Funktionsweise eines Algorithmus mit Studienkollegen und Lehrenden besprechen. Da die Herstellung eines Lehrfilms aber in der Regel sehr aufwändig ist, können Studierende ihn nicht selbst produzieren. Das Medium Lehrfilm steht als Übungs- und Kommunikationsmedium nicht zur Verfügung.

1.2.3 Interaktive Visualisierung

BALSA

Mit dem Aufkommen leistungsstarker Computer mit grafischen Benutzungsoberflächen entstand ein neues Medium: interaktive Algorithmen-Animationen. Der Betrachter kann damit auf vielfältige Weise den Inhalt und die Gestaltung der Animation beeinflussen. Insbesondere kann er den Ablauf selbst steuern und z.B. in Einzelschritten die Animation weiterschalten. Als Meilenstein ist hier das System BALSA von Marc Brown anzusehen, das für den Apple Macintosh erstellt wurde ([Blumstengel 1998a], [Brown 1998]). Man kann als Betrachter selbst eine Animation zusammenstellen. Dazu:

1. Wählt man einen oder mehrere Algorithmen
2. Verknüpft man einen Algorithmus mit Sichten, die die Operationen des Algorithmus auf der Datenstruktur oder seinen Pseudocode zeigen. Man kann die Fenster der Sichten auf dem Bildschirm frei arrangieren, zoomen und scrollen.
3. Wählt man einen Satz Eingabedaten
4. Startet man die Animation. Man kann die Animation kontinuierlich laufen lassen, unterbrechen und fortsetzen. Oder man bewegt sich in einzelnen Schritten durch die Animation.

Abb. 5 zeigt BALSA bei der Animation des Insertion Sort. Die linke Sicht zeigt das zu sortierende Array, wobei der aktuell einzufügende Schlüssel als „Schatten“ dynamisch verschoben wird. In der rechten Sicht ist die gesamte bisherige Geschichte der benötigten Vergleiche und Vertauschungen visualisiert.

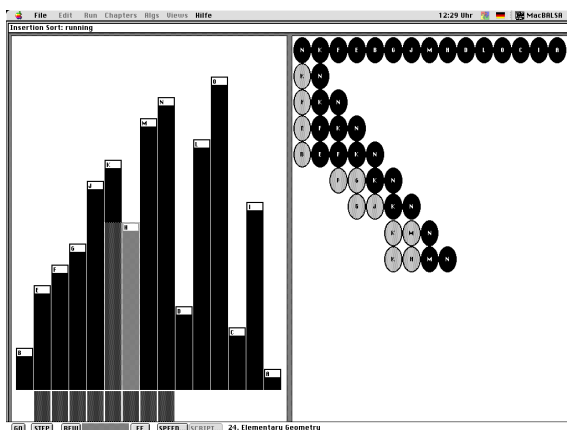


Abb. 5: BALSA insertion sort
Bildschirmabzug von [Brown 1998]

Es ist auch möglich, mehrere Algorithmen parallel ablaufen zu lassen. BALSA synchronisiert die Algorithmen dabei so, dass sie gleich viel „Rechenzeit“ pro Zeiteinheit der Animation erhalten.

Ein zentrales Konzept in BALSA sind die *Interessanten Ereignisse* („interesting events“). Der Autor einer Animation legt fest, welche Operationen eines Algorithmus „interessant“ sind und in der Animation gezeigt werden sollen. Die Sichten werden so programmiert, dass sie die interessanten Ereignisse grafisch darstellen, indem sie z.B. die veränderten Teile der Datenstruktur neu zeichnen.

Der Betrachter ordnet den interessanten Ereignissen jeweils eine „virtuelle Rechenzeit“ zu. Beim Ablauf des Algorithmus werden diese Rechenzeiten aufaddiert, so dass man am Ende die Gesamtlaufzeit ablesen kann.

Ebenso kann der Betrachter selbst entscheiden, auf welche interessanten Ereignisse er besonders achten möchte. Damit steuert er die Schrittweite der Animation, d.h. an welchen Punkten die Animation pausiert und auf den Befehl des Betrachters zum Weiterlaufen wartet.

Animated Algorithms

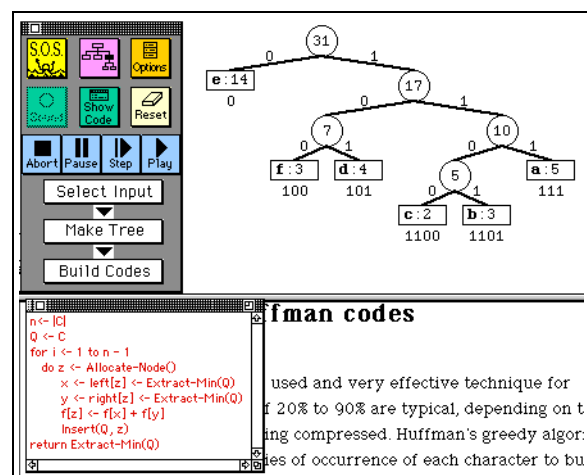


Abb. 6: Animated Algorithms
Bildschirmabzug von [Gloor, Dynes, Lee 1993]

BALSA ist ein reines Animationssystem, das keine erklärenden tutoriellen Texte und Grafiken bietet, wie man es von einem Lehrbuch gewohnt ist. Ein weiterer Meilenstein war daher die Entwicklung des Systems „Animated Algorithms“, das Animationen mit tutoriellen Bestandteilen verbindet [Gloor 1998]. Es enthält folgende, hypermedial verknüpfte Elemente:

- Digitale Videos, die die Bedienung des Systems erklären
- 24 Algorithmen-Animationen
- Einführende Texte und Grafiken zu den Animationen

- Das Lehrbuch „Introduction to Algorithms“ [Cormen, Leiserson, Rivest 1990] in Hypertextform

Abb. 6 zeigt einen Bildschirmabzug des Programms „Animated Algorithms“. Er enthält Fenster mit Steuerung der Ausführung, Datenstruktur, Pseudocode und tutoriellem Text.

Die Lernumgebung wurde mit HyperCard für Apple Macintosh Computer erstellt und wird als CD-ROM vertrieben [Gloor, Dynes, Lee 1993].

Lernplattform WWW

Mit dem World Wide Web (WWW) ist eine neue Plattform für interaktive Lernmedien entstanden. Bereits mit den Standardformaten des Web wie

HTML und GIF für Text und Bilder und Java-Applets für interaktive Animationen lassen sich umfassende und interessante Lernmedien entwickeln. Sie lassen sich auf einfache Weise über Webserver bereitstellen und sind plattformunabhängig in normalen Webbrowsern ausführbar.

Betrachtet man Lernmedien für das individuelle Lernen (im Gegensatz zu z.B. Teleteaching und kollaborativen Lernformen), so liegt der Vorteil der Lernplattform Internet hauptsächlich im Zugang zu dem riesigen Informationsraum des WWW. Die einzelnen Lernmedien aus dem WWW sind per Mausklick ohne Beschaffungsaufwand und Installation verfügbar.

1.3 Akzeptanz und Effektivität von Algorithmen-Animationen

Nach der exemplarischen Darstellung der historischen Entwicklung grafischer Lernmedien zu Algorithmen soll nun untersucht werden, in welchem Maß Algorithmen-Animationen in der Ausbildung eingesetzt werden und welche Faktoren den Lernerfolg beeinflussen.

1.3.1 Verbreitung und Akzeptanz

Algorithmen-Animationen sind leicht erhältlich

Es sind heute eine Vielzahl von Algorithmen-Animationen frei aus dem Internet erhältlich. In seiner Diplomarbeit ermittelte Frank Peters rund 200 interaktive Visualisierungen von Algorithmen, die über das Internet erhältlich sind [Peters 2000]. Redaktionell gepflegte Listen von Algorithmen-Animationen finden sich z.B. beim Deutschen Bildungsserver [DBS 2001] und bei einer von Guido Rößling geführten Datenbank [Rößling 2001a]. Eine Übersicht über weitere computergestützte Lernmedien zur Informatik stellen wir in der unten genannten OLLI-Sammlung bereit.

Studierende nutzen webbasierte Lernmedien

Im Projekt „Medienunterstütztes Studium der Informatik“ [Gorny, Faltin 2000] haben wir 18 webbasierte Lernmedien zu Themen der Informatik er-

stellt und in der Sammlung „Oldenburger Lernprogramme zur Informatik (OLLI)“ [Faltin 2001a] zusammengefasst. Sie können online und offline zu den Bedingungen der GNU General Public License kostenlos genutzt werden.

Eine Auswertung der Zugriffe im Juni/Juli 2001 ergab, dass an einem Tag durchschnittlich 137 Personen (genauer: IP-Adressen) auf den Webserver zugreifen, 1051 Seiten abrufen und 35 Dateien herunterladen. Etwa zwei Drittel der heruntergeladenen Dateien enthalten Archive der Lernprogramme für Benutzer, der Rest enthält Versionen für die Weiterentwicklung der Lernprogramme. Die Lernmedien sind in der Regel in Deutsch verfasst, so dass wir vor allem Zugriffe aus dem deutschsprachigen Bereich verzeichnen.

Bei den Lernmedien aus OLLI handelt es sich um spezielle Themen, die wohl vor allem für Informatikstudierende, kaum aber für typische Informatik-Anwender oder Privatpersonen relevant sind. Selbst bei einer konservativen Schätzung, die nur von den rund 20 Downloads der Nutzerversionen der Lernprogramme am Tag ausgeht und reine Online-Nutzer ignoriert, erhält man die beachtliche Zahl von etwa 7000 Personen im Jahr, die mit unseren webbasierten Lernmedien lernen. Daraus kann man schließen, dass das Interesse der Studierenden an webbasierten Lernmedien offensichtlich groß ist.

Wenige Lehrende nutzen Algorithmen-Animationen

Demgegenüber wird von Fachleuten aus dem Bereich Algorithmen-Animation immer wieder behauptet, dass Algorithmen-Animationen nur von wenigen Lehrenden in der Informatikausbildung eingesetzt werden ([Hundhausen 1999], S. 3). So äußert sich z.B. Jim Foley, einer der Pioniere der Computergrafik, in der Einleitung der Aufsatzsammlung „Software Visualization“:

„My only disappointment with the field is that Software Visualization has not yet had a major impact on the way we teach algorithms and programming or the ways in which we debug our programs and systems. While I continue to believe in the promise and potential of SV, it is at the same time the case that SV has not yet had the impact that many have predicted and hoped for.“ ([SDBP 1998], S. xii)

In der Einladung zum „Dagstuhl Seminar on Software Visualization 2001“ waren die Teilnehmer aufgefordert worden die Frage zu erörtern:

„Why is software visualization not widely used in education and software development?“ [SV 2001]

Auf dem Seminar war diese Frage das Thema einer Diskussionsveranstaltung. Die Teilnehmer bestätigten die Ansicht, dass Algorithmen-Animationen bisher nur wenig in der Lehre eingesetzt werden.

1.3.2 Untersuchungen zur Effektivität

Diese Zurückhaltung könnte darin begründet sein, dass die Lehrenden skeptisch sind, ob das Betrachten der derzeit verfügbaren Algorithmen-Animationen den Studierenden besser hilft, Algorithmen zu verstehen, als es die statischen Diagramme tun.

Einflußfaktoren auf den Lernerfolg

Es gibt eine ganze Reihe Faktoren, die den Lernerfolg beeinflussen können. Es ist daher wichtig zu ermitteln, welche Faktoren tatsächlich eine Rolle spielen und wie sie ausgeprägt sein sollen, um das Lernen von Algorithmen bestmöglich zu unterstützen. In den letzten Jahren sind eine Reihe formaler Experimente durchgeführt worden, die dies über vergleichende Experimente ermittelt haben. Dabei wurden Gruppen von Studierenden unterschiedlich unterrichtet und danach ihr neu erlangtes Wissen in einem schriftlichen Test bewertet.

Die Daten wurden statistisch analysiert. Zeigte sich ein deutlicher (d.h. signifikanter) Unterschied zwischen den Gruppen, so hatte der untersuchte Faktor einen Einfluss auf den Lernerfolg. Andernfalls konnte man schließen, dass der Faktor in dieser Lernkonstellation keinen wesentlichen Einfluss auf den Lernerfolg hat.

Christopher Hundhausen listet in einer Meta-Studie die Ergebnisse zehn solcher Experimente auf ([Hundhausen 1999], Kapitel 2.5). Wie mir Hundhausen mitteilte kommt eine neue von ihm mitverfasste, noch unveröffentlichte Studie von 24 Experimenten im Wesentlichen zu den gleichen Schlüssen bezogen auf die Wichtigkeit aktiven Lernens [Hundhausen, Douglas, Stasko].

Abweichend von der Aufteilung bei Hundhausen fasse ich die Faktoren hier in drei Bereichen zusammen (Tabellen 1, 2 und 3):

- **Präsentation des Algorithmus**

Der Algorithmus wird den Studierenden über Lernmedien präsentiert. Neben der Dynamik der Medien wird hier die grafische Repräsentation des Algorithmus und die Redundanz der Darstellung betrachtet. Erstaunlicherweise gab es bei 6 von 8 Telexperimenten keine signifikanten Unterschiede. Die Studierenden scheinen das Wesentliche des Algorithmus in all diesen Fällen ungefähr gleich gut aufgefasst zu haben.

- **Individuelle Unterschiede**

In zwei Telexperimenten wurde untersucht, ob Personen mit überdurchschnittlichem räumlichen Vorstellungsvermögen oder verbalen Fähigkeiten eine Algorithmen-Animation besser dekodieren und damit den Algorithmus besser verstehen können. Es zeigten sich aber keine signifikanten Unterschiede.

- **Beteiligung der Lerner**

In der Praxis werden Algorithmen-Animationen den Studierenden zumeist wie ein Film gezeigt, den sie passiv betrachten. Man kann Studierende aber viel stärker am Lernprozess beteiligen, indem man ihnen die Aufgabe gibt den Algorithmus zu programmieren, Eingabedaten zu erstellen, Schritte vorherzusagen oder selbst eine Animation zu erstellen. Die fünf hierzu durchgeführten Telexperimente ergaben durchgängig signifikante Ergebnisse. Je stärker die Studierenden am Lernprozess beteiligt waren, desto besser hatten sie den Algorithmus verstanden.

Tabelle 1: Einfluss der Präsentation auf den Lernerfolg

Faktor	Betrachtete Werte	Anzahl Experimente	
		signifikant	nicht signifikant
Lernmedium	nur Text, Text und Animation	–	1
Reihenfolge Lernmedien	zuerst Text, zuerst Animation	–	2
Größe Datensatz	9, 25 oder 41 Elemente	–	1
Darstellung der Daten	horizontale Balken, vertikale Balken, Punkte	–	1
Farbigkeit	farbig, schwarzweiß	1	–
Beschriftung der Daten	beschriftet, unbeschriftet	–	1
Benennung der Schritte	benannt, unbenannt	1	–

Tabelle 2: Einfluss individueller Unterschiede

Faktor	Betrachtete Werte	Anzahl Experimente	
		signifikant	nicht signifikant
Räumliche Fähigkeiten	Test zum Papierfalten	–	1
Sprachliche Fähigkeiten	Vokabeltest	–	1

Tabelle 3: Einfluss der Beteiligung der Lerner

Faktor	Betrachtete Werte	Anzahl Experimente	
		signifikant	nicht signifikant
Beteiligung der Lerner	1. Passiv Animation betrachten 2. Nach dem Betrachten der Animation den Algorithmus programmieren 3. Animation aktiv betrachten - eigene Eingabedaten konstruieren - Schritte vorhersagen 4. Animation erstellen	5	–

Aktives Lernen als Erfolgsfaktor

Die Skepsis vieler Lehrender gegenüber Algorithmen-Animationen wurde durch diese Experimente in dem Sinn bestätigt, dass die bewegte Darstellung allein noch kein besseres Verständnis bewirkt. Die wesentliche Erkenntnis ist aber, dass Algorithmen so gelehrt werden sollten, dass die

Studierenden sich aktiv mit dem Lehrstoff auseinandersetzen können. Dies ist ein starkes Argument dafür, neue Lehrformen zu entwickeln, die dem Ansatz dieser Arbeit entsprechend darauf abzielen, die Studierenden aktiv in den Lernprozess einzubeziehen.

1.4 Übersicht über die folgenden Kapitel

Es wird nun ein Überblick über die folgenden Kapitel gegeben. Für Entwickler von Lernprogrammen zu Algorithmen sind vor allem Kapitel 4 bis 6 interessant. Im Einvernehmen mit dem Betreuer der Arbeit wurden Teile der Kapitel 4 bis 6 auf Fachtagungen präsentiert und in den Tagungsbänden veröffentlicht.

2 Lerngegenstand „Algorithmus“

Der Lerngegenstand „Algorithmus“ wird aus fachdidaktischer Sicht betrachtet. Dazu werden Aussagen zum Algorithmusbegriff, zur Darstellung von Algorithmen und zur Bedeutung von Algorithmen aus bekannten Lehrbüchern und dem Informatik-Duden zusammengestellt.

3 Verwandte Arbeiten zum aktivem Lernen von Algorithmen

Es werden verwandte Lehrmethoden vorgestellt bei denen die Studierenden Algorithmen nicht nur als Animation betrachten sondern aktiv erlernen. Diese Lehrmethoden nutzen den Computer als interaktives grafisches Medium, um Algorithmen darzustellen, unterscheiden sich aber in der Art der Einbeziehung der Studierenden in den Lernprozess.

4 Die Gestaltungsmethode „Strukturiertes Aktives Lernen von Algorithmen (SALA)“

Im Rahmen dieser Arbeit wurde die Methode „Strukturiertes Aktives Lernen von Algorithmen (SALA)“ für die Gestaltung von Lernprogrammen zu Algorithmen entwickelt. Wie oben in der Einführung beschrieben, setzt sie auf ein entdeckendes Lernen über interaktive grafische Simulationen eines Algorithmus. Als Beispiel wird gezeigt, wie nach der Methode ein Lernprogramm zum Heapsort-Algorithmus entwickelt wurde.

5 Gestaltungsregeln für interaktive Visualisierungen von Algorithmen

Es werden drei Arten interaktiver Visualisierungen von Algorithmen unterschieden: Animation, Übung und Simulation. Für diese werden Beispiele genannt und aus didaktischer und software-ergonomischer Sicht Gestaltungsregeln angegeben.

6 Software-Bibliothek SALABIM

Die Java-Bibliothek SALABIM erleichtert die Erstellung interaktiver Visualisierungen von Algorithmen als Java-Applet. Der Algorithmus und seine Visualisierung muss nur einmal programmiert werden und steht dann in den Modi Simulation, Üben und Animation zur Verfügung. In den Modi Üben und Simulation haben die Studierenden die Aufgabe, die Schritte des Algorithmus selbst auszuführen. Die Bibliothek realisiert die Ablaufsteuerung der interaktiven Visualisierung, stellt Tipps zur Lösung der Aufgabe bereit und kann Benutzereingaben mit den Schritten des Algorithmus vergleichen.

7 Studierende bewerten SALA-Lernprogramme und SALA

Zwei nach der Methode SALA gestaltete Lernprogramme wurden in der Lehrveranstaltung „Datenstrukturen“ erprobt. Die Studierenden arbeiteten als Hausaufgabe mit den Lernprogrammen. Sie bewerteten auf einem Fragebogen die Lernprogramme und das dahinter stehende didaktische Konzept SALA.

8 Analyse der Nutzung zweier SALA-Lernprogramme

Die Nutzung der Lernprogramme wurde aufgezeichnet. Die Daten werden ausgewertet um das Navigationsverhalten, die Intensität der Nutzung und das Verhältnis von Online- zu Offline-Nutzung zu ermitteln.

2 Lerngegenstand „Algorithmus“

Dieses Kapitel betrachtet den Lerngegenstand „Algorithmus“ aus fachdidaktischer Sicht. Will man ein Lernmedium erstellen, so muss man sich über die Aspekte des Lerngegenstandes und die Möglichkeiten seiner Darstellung bewusst sein, da diese den Gestaltungsraum bilden. Daher werden nun Definitionen und Aussagen zum Algorithmusbegriff und zur Darstellung von Algorithmen aus Lehrbüchern zu Algorithmen und aus Fachlexika zusammengestellt. Das Kapitel dient auch zur Definition grundlegender Begriffe, die in den folgenden Kapiteln benutzt werden.

Bedeutung der Algorithmen

Algorithmen spielen in der Informatik eine zentrale Rolle. Für viele Probleme der Informatik sind Algorithmen entwickelt worden. Auch in anderen formalen Wissenschaften wie der Mathematik sind sie von Bedeutung. Die zentrale Bedeutung wird in Lehrbüchern, wie z.B. bei Sedgewick und Ottmann/Widmayer immer wieder angeführt:

„Algorithms are the 'stuff' of computer science: they are central objects of study in many, if not most areas of the field.“
([Sedgewick 1992], S. 4)

„Sie [Algorithmen] sind *das* zentrale Thema der Informatik. Die Entwicklung und Untersuchung von Algorithmen zur Lösung vielfältiger Probleme gehört zu den wichtigsten Aufgaben der Informatik.“ ([Ottmann, Widmayer 1996], S. 1, Hervorhebung im Original)

Kapitelinhalt

In der Informatik gibt es eine lange Tradition des Umgangs mit Algorithmen. Es herrscht weitgehende Einigkeit darüber, welche formalen Bedingungen eine Verfahrensbeschreibung erfüllen muss, um als Algorithmus zu gelten. Zur Beschreibung eines Algorithmus gehören in der Regel der Nachweis seiner Korrektheit und die Bestimmung des Ressourcenbedarfs. Um das Verständnis des Algorithmus zu erleichtern, werden zumeist die Eigenschaften der Datenstruktur besprochen und ein Ablauf als Beispiel angegeben. Auch die Notation der Verfahrensbeschreibung und deren Modularität beeinflussen die Verständlichkeit. Werden mehrere Algorithmen behandelt, so kann es hilfreich sein, gemeinsame Entwurfsprinzipien herauszuarbeiten.

2.1 Aspekte eines Algorithmus

2.1.1 Der Algorithmusbegriff

Definitionen des Algorithmusbegriffes finden sich z. B. in bekannten Lehrbüchern zu Algorithmen und im Informatik-Duden. Da keine dieser von mir untersuchten Quellen alle mir wesentlich erscheinenden Merkmale nennt, stelle ich im Folgenden die Merkmale aus verschiedenen Quellen zusammen.

Algorithmus: Verfahren zur Lösung eines Problems

Ein Algorithmus ist ein Verfahren, das sich als Programm für einen Computer implementieren lässt. Das Verfahren löst ein zuvor definiertes formales Problem. Zu einer Instanz des Problems (der Eingabe) berechnet der Algorithmus eine Lösung (die

Ausgabe). Das Verfahren muss dabei für alle Probleminstanzen nach endlich vielen Schritten ein Resultat liefern (*Terminierung*) [DudenInf 1988].

„The term algorithm is used in computer science to describe a problem-solving method suitable for implementation as a computer program“ ([Sedgewick 1992], S. 4)

„Informally, an algorithm is any well-defined computational procedure that takes some value, or set of values, as input and produces some value, or set of values, as output. An algorithm is thus a sequence of computational steps that transform the input into the output. We can also view an algorithm as a tool for solving a well-specified computa-

tional problem. The statement of the problem specifies in general terms the desired input/output relationship. The algorithm describes a specific computational procedure for achieving that input/output relationship.“ ([Cormen, Leiserson, Rivest 1990], S. 1)

Anforderungen an das Verfahren

Die Beschreibung des Verfahrens muss folgenden Anforderungen genügen:

- **Statische Finitheit.**

Die Beschreibung des Verfahrens hat endliche Länge. ([DudenInf 1988], S. 27)

- **Definit (engl. Definiteness).**

Bei jeder Anweisung ist eindeutig klar, was zu tun ist. ([Knuth 1969], S. 5)

- **Ausführbar (engl. Effectiveness).**

Jede Anweisung ist ausführbar und ihre Ausführung dauert endlich lange. ([Knuth 1969], S. 6)

Die weitaus meisten Algorithmen der Informatik sind deterministisch und determiniert:

- **Deterministisch.**

Zu jedem Zeitpunkt gibt es nur eine Fortsetzungsmöglichkeit. ([DudenInf 1988], S. 27)

- **Determiniert.**

Wird der Algorithmus mehrfach mit gleicher Eingabe gestartet, liefert er jedesmal die gleiche Ausgabe. ([DudenInf 1988], S. 27)

2.1.2 Korrektheit

Ein Algorithmus ist korrekt, wenn er für jede Problem Instanz in endlicher Zeit ein korrektes Ergebnis liefert. Meist wird in Lehrbüchern im Begleittext auf die Stellen des Algorithmus eingegangen, deren Korrektheit nicht unmittelbar einleuchtet und teils formal, teils umgangssprachlich argumentiert. Teilweise werden im Pseudocode zum Nachweis der Korrektheit Schleifeninvarianten und Schleifenvarianten angegeben.

„Die wichtigste formale Eigenschaft eines Algorithmus ist zweifellos dessen Korrektheit. [...] Wo aber die Korrektheit eines Algorithmus nicht unmittelbar offensichtlich ist, geben wir durch Kommentare, Angaben von Schleifen- und Prozedurinvarianten und anderen Hinweisen im Text ausreichende Hilfen, auf

die der Leser selbst formalisierte Korrektheitsbeweise gründen kann.“ ([Ottmann, Widmayer 1996], S. 2)

2.1.3 Effizienz

Unter der Effizienz eines Algorithmus versteht man seinen Verbrauch an Ressourcen wie Rechenzeit und Speicherplatz in Abhängigkeit von der Größe der Eingabe. Die Rechenzeit ist meist am schwierigsten zu berechnen. In der Regel wird im Begleittext ein Beweis skizziert, der sich auf Rekursionsgleichungen und Summenformeln stützt. Im Pseudocode findet man meist keine Kommentare zur Effizienz.

„Die weitaus wichtigsten Maße für die Effizienz sind der zur Ausführung des Algorithmus benötigte Speicherplatz und die benötigte Rechenzeit.“ ([Ottmann, Widmayer 1996], S. 2)

2.1.4 Datenstruktur

Viele Algorithmen arbeiten auf komplexen Datenstrukturen. Zum Verständnis des Algorithmus gehört daher auch das Verständnis der Datenstruktur.

„Die meisten Algorithmen erfordern jeweils geeignete Methoden zur Strukturierung der von den Algorithmen manipulierten Daten. Algorithmen und Datenstrukturen gehören also zusammen.“ ([Ottmann, Widmayer 1996], S.1)

„Most algorithms of interest involve complicated methods of organizing the data involved in the computation. Objects created in this way are called data structures, and they also are central objects of study in computer science. Thus algorithms and data structures go hand in hand [...] and thus need to be studied in order to understand the algorithms.“ ([Sedgewick 1992], S. 4, Hervorhebung im Original)

2.1.5 Trace

Startet man einen Algorithmus auf konkreten Eingabedaten, so führt er eine Folge konkreter Schritte aus, die hier als *Trace* bezeichnet wird. Zu einem Schritt gehört dabei auch die Veränderung der Daten. Bei der großen Mehrzahl der Algorithmenbeschreibungen wird ein Beispiel des Ablaufes als

Trace angegeben, indem zu wichtigen Zeitpunkten des Ablaufes die Daten des Algorithmus dargestellt werden (siehe z.B. Abb. 3, S. 13).

2.2 Darstellung des Verfahrens

2.2.1 Pseudocode oder Programmcode

Ein Algorithmus kann auf verschiedene Weisen dargestellt und implementiert werden, z.B. in einer Programmiersprache, als Pseudocode oder als Hardwareimplementierung. In Lehrbüchern wird zumeist Pseudocode verwendet, da dies den Autoren die Möglichkeit gibt programmiersprachliche Notation mit natürlichsprachlichen Anweisungen zu mischen. ([Cormen, Leiserson, Rivest 1990], S. 2)

2.2.2 Modularität

Während für Programme im allgemeinen gefordert wird, dass sie modular und fehlerrobust erstellt werden sollen (siehe z.B. [Meyer 1997]), scheint unter den von mir betrachteten Lehrbuchautoren zu Algorithmen ein stillschweigender Konsens zu bestehen, diese Anforderung bei der Darstellung von Pseudocode zu ignorieren. So werden z.B. Hilfsfunktionen nur eingeführt, wenn eine Funktion mehrfach verwendet wird. Dabei ist den Lehrbuchautoren wohl bewusst, dass eine modulare Darstellung leichter zu verstehen ist, denn der Algorithmus wird im Begleittext oft entlang des logischen Aufbaus der Kontrollstruktur erklärt. Bei den betrachteten Lehrbüchern findet sich nur bei [Cormen, Leiserson, Rivest 1990], S. 2) eine Aussage dazu:

„Another difference between pseudocode and real code is that pseudocode is not typically concerned with issues of software engineering. Issues of data abstraction, modularity, and error handling are often ignored in order to convey the essence of the algorithm more concisely“

In dieser Arbeit wird ein modularer Ansatz verfolgt, bei dem der Pseudocode auf mehrere Funktionen aufgeteilt wird. Dies soll einerseits das Verständnis erleichtern, ermöglicht andererseits aber auch erst das entdeckende Lernen über Simulationen.

Wiederverwendung eines Algorithmus

Ein modularer Ansatz erleichtert auch die Wiederverwendung eines Algorithmus oder seiner Bestandteile in einem neuen Kontext. Gelegentlich kann man einen bestehenden Algorithmus so abwandeln, dass er ein neues Problem löst. Dazu ist es hilfreich, wenn der ursprüngliche Algorithmus möglichst modular dargestellt war. Er muss dann nur an wenigen Stellen geändert werden.

Beispielsweise kann man den Heapbaum des Heapsort-Algorithmus verwenden, um eine Prioritäten-Warteschlange zu realisieren (siehe dazu das Kapitel zum Lernprogramm Heapsort und ([Ottmann, Widmayer 1996], S. 378). Ein weiteres Beispiel ist die Abwandlung des Quicksort-Algorithmus, um einen N-Order-Median zu bestimmen.

Algorithmen werden auch als Ganzes (unverändert) als Teil eines neuen Algorithmus genutzt. So benötigt man für die kürzeste-Wege-Suche nach Dijkstra eine Hilfsdatenstruktur, die die als Nächstes zu untersuchenden Knoten des Graphen speichert. Es bietet sich an, hierfür eine Prioritätenwarteschlange zu nutzen ([Ottmann, Widmayer 1996], S. 379-382).

2.2.3 Algorithmen und Entwurfsprinzipien

Ein Algorithmus wird in der Regel ausgehend von einem Entwurfsprinzip entwickelt. Beispiele für Entwurfsprinzipien sind „Divide and Conquer“, Inkrementeller Entwurf und Backtracking. Lehrbücher, die eine Sammlung von Algorithmen beschreiben, benennen und erklären zumeist das Entwurfsprinzip im Begleittext zum Algorithmus (z.B. [Ottmann, Widmayer 1996], [Sedgewick 1992], größter Teil von [Cormen, Leiserson, Rivest 1990]). Es kann aber auch umgekehrt ein Entwurfsprinzip zum Thema werden, während eine Reihe von Algorithmen danach aus dem Blickwinkel dieses Entwurfsprinzips beschrieben werden (insbesondere bei [Ottmann 1998], aber auch in Teilen von [Cormen, Leiserson, Rivest 1990]). An ameri-

kanischen Universitäten werden die Vorlesungen zu Algorithmen oft nach den Entwurfsprinzipien gegliedert:

„CIS 315, 'Algorithms', typifies the third year undergraduate course taught within the computer science departments of American universities. In the course, students explore efficient algorithmic problem-solving techniques, including

divide-and-conquer, dynamic programming, and greedy approaches.“
([Hundhausen 1999], S. 47)

Studierenden sollten mehrere Entwurfprinzipien erlernen, da diese nicht nur das Verständnis von Algorithmen erleichtern, sondern auch als Strukturierungstechniken beim Entwurf eigener Programme einsetzbar sind.

3 Verwandte Arbeiten zum aktivem Lernen von Algorithmen

3.1 Überblick

Die Untersuchungen zur Effektivität von Algorithmen-Animationen ergaben, dass die bewegte Darstellung allein noch kein besseres Verständnis des Algorithmus bewirkt. Algorithmen sollten vielmehr so gelehrt werden, dass die Lerner sich aktiv mit dem Lehrstoff auseinandersetzen können (vgl. Kap. 1.3, S. 15). Daher werden nun neue Lehrmethoden und Systeme betrachtet und bewertet, die, dem Ansatz dieser Arbeit entsprechend darauf abzielen, die Studierenden aktiv in den Lernprozess einzubeziehen. In den Lehrmethoden haben die Lerner die Aufgabe, den Algorithmus auszuführen, eine Animation zum Algorithmus zu entwickeln oder den Algorithmus zu programmieren. Einige weitere Systeme zum aktiven Lernen von Algorithmen werden in den Übersichten von Hundhausen und Rößling aufgeführt ([Hundhausen, Douglas, Stasko], [Rößling 2001b]). Es handelt sich dabei zumeist um Systeme, bei denen die Lerner den Algorithmus selbst ausführen.

3.1.1 Lerner führen Algorithmus aus

Die erste Klasse von Lehrmethoden lässt die Lerner den Algorithmus selbst ausführen. Der Algorithmus wird zunächst auf klassische Weise in der Vorlesung oder über ein Lehrbuch gelehrt. Dann arbeiten die Lerner mit einer Lernumgebung, in der sie den Algorithmus auf Beispieldaten selbst ausführen können. Sie müssen sich dabei genau an die Schrittfolge des Standardverfahrens halten. Die unten genannten Methoden unterscheiden sich darin, ob der Lerner Basisoperationen oder Operationen höherer Ordnung angeben muss.

Lernumgebung „Algorithms in Action“

Die Lernumgebung „Algorithms in Action“ enthält eine Reihe von Elementen, die für ein eigenständiges Lernen wichtig sind und in vielen konventionellen Algorithmen-Animationen fehlen ([Stern, Sondergaard, Naish 1999], [Stern 2001]). Sie bietet ausführliche Erklärungen zum Algorithmus als Ganzes und zu den einzelnen Teilen des Pseudocodes. Der Pseudocode des Algorithmus wird als hierarchische Struktur dargestellt, bei der die Lerner den gewünschten Detaillierungsgrad einstellen können.

Dieses Konzept des „stepwise refinement“ entspricht in etwa der in SALA verwendeten funktionalen Struktur. Die Bestandteile werden aber im Allgemeinen nicht benannt und als Funktionen deklariert, sondern über ein kleines Ordnersymbol neben der Pseudocodezeile kenntlich gemacht. Durch Klicken auf das Symbol kann dieser Teil des Pseudocodes auf- und zugeklappt werden. In Abb. 7 ist der Teil „Restore Heap Order“ und die darin enthaltene Downheap-Funktion aufgeklappt worden. Lässt man die Animation ablaufen, so entspricht ein Schritt in der Animation einer sichtbaren Zeile im Pseudocode. Ist ein Teil des Pseudocodes komplett eingeklappt, so wird dieser Teil also in einem Schritt ausgeführt (in der Abbildung z.B. der Aufbau des Heap). Vorteilhaft ist auch die flexible Ablaufsteuerung, mit der man in Einzelschritten vor und zurück durch die Animation gehen kann.

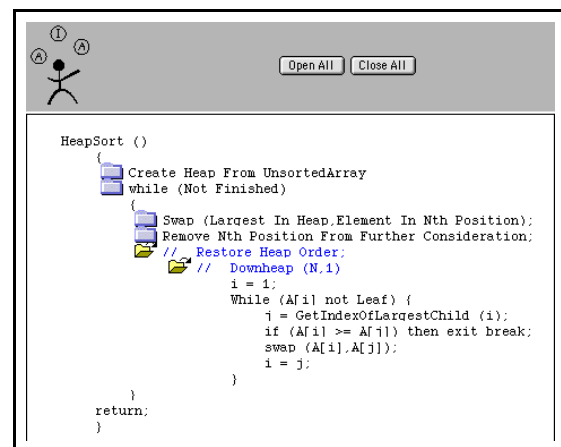


Abb. 7: Stepwise Refinement bei „Algorithms in Action“
Bildschirmabzug von [Stern 2001]

Die Lernumgebung bietet einen Animationsmodus („Normal Mode“) und einen Übungsmodus („Self Test Mode“). Schaltet man in den Übungsmodus, so wird man an bestimmten Stellen des Algorithmus aufgefordert, Parameter für die aktuelle Operation anzugeben. Dazu wählt man Objekte in der grafischen Darstellung der Daten mit der Maus aus. Wurden die richtigen Objekte gewählt, so läuft der Algorithmus weiter, andernfalls hat man einen weiteren Versuch. Nach drei Fehlversuchen wechselt die Lernumgebung in den Animationsmodus zurück. In Abb. 8 ist Heapsort im Modus Üben dargestellt. Die Lernumgebung verlangt, dass der Kindknoten mit dem größeren Schlüssel ausgewählt wird. In der gesamten Übung zu He-

apport werden nur die Basisoperation „größerer Kindknoten finden“ und „Zwei Schlüssel vertauschen“ abgefragt. Es wird also nur für einen kleinen Teil des Pseudocodes eine Interaktion angeboten, während der größere Teil automatisch abläuft. Ein weiterer Schwachpunkt ist, dass die interaktiven Basisoperationen sehr oft wiederholt werden. Als Lerner fragt man sich, warum man immer wieder die gleichen Basisoperationen ausführen muss. Dies lässt sich nicht abschalten, da die Lernumgebung im Übungsmodus die gewählte Detaillierung des Pseudocodes ignoriert.

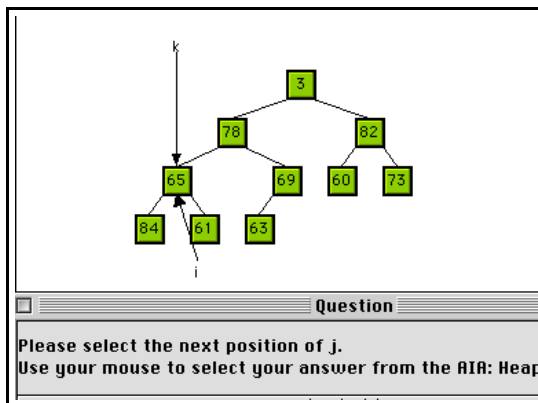


Abb. 8: Heapsort im Modus Üben bei „Algorithms in Action“
Bildschirmabzug von [Stern 2001]

Algorithms in Action bietet aus didaktischer Sicht vieles, was andere Lernumgebungen vermissen lassen. Innovativ ist besonders der Übungsmodus, der aber noch ausgebaut und verbessert werden sollte.

JFLAP: formale Sprachen ineinander umwandeln

Susan Rodger entwickelte die Lernumgebung JFLAP für eine universitäre Lehrveranstaltung zur Automatentheorie [Hung, Rodger 2000]. JFLAP erlaubt es, Automaten interaktiv zu erstellen und zu simulieren (Abb. 9), sowie Automaten, Grammatiken und reguläre Ausdrücke ineinander umzuwandeln. Die Repräsentationen der regulären Sprachen können in JFLAP auf drei Arten ineinander überführt werden. Bei der reinen Animation lässt sich der Nutzer schrittweise die Entstehung der neuen Repräsentation vorführen. Im Übungsmodus führt der Nutzer selbst einen Konstruktionsschritt aus und lässt diesen von JFLAP mit dem entsprechenden Schritt des Standardverfahrens vergleichen. JFLAP analysiert die Teillösung und gibt Hinweise auf die festgestellte Abweichung von der Standardlösung. Reicht der Hinweis nicht aus, so kann der Nutzer den Schritt von JFLAP ausführen lassen. Im Modus der Konstruktion er-

stellt der Nutzer ohne Hilfe die komplette Repräsentation und lässt diese dann von JFLAP mit der Musterlösung vergleichen.

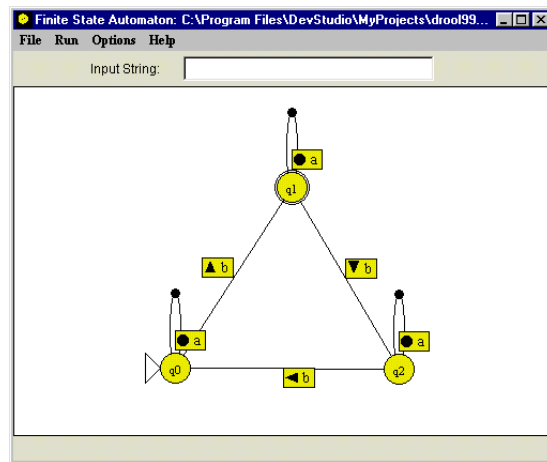


Abb. 9: Endlicher Automat in JFLAP
[Rodger 2001]

JFLAP ist ein leistungsfähiges und relativ ausge-reiftes Werkzeug, das mit dem Übungsmodus ein aktives Lernen ermöglicht. Leider wird der jeweils verwendete Konversionsalgorithmus nicht angezeigt, so dass man ohne externe Hilfsmittel (z.B. Lehrbuch) schwer den nächsten richtigen Schritt findet. Es scheint auch kein erklärendes Lernmaterial in die Lernumgebung integriert zu sein. JFLAP stellt den Algorithmus nicht strukturiert dar, so dass man nicht in größeren Schritten, die höheren Operationen entsprechen, durch die Animation oder Übung gehen kann. Da die Konversionsalgorithmen in allen Schritten ausgeführt werden müssen, ist das Werkzeug nur für kleine Beispiele geeignet.

3.1.2 Lerner entwickeln Algorithmen-Animation

Die Lerner können einen Algorithmus aktiv erlernen, indem sie eine Animation zum Algorithmus entwickeln. Der Algorithmus wird den Lernern zunächst konventionell über eine Vorlesung oder ein Lehrbuch vermittelt. Anschließend erhalten sie Hilfsmittel und Werkzeuge, mit denen sie eine Animation zum Algorithmus erstellen.

Stasko: Studierende entwickeln Animationen mit SAMBA

John Stasko setzte diese Lehrmethode in der universitären Veranstaltung „Design and Analysis of Algorithms“ ein [Stasko 1996]. Die Studierenden entwickelten als Hausaufgabe mit dem Visualisierungssystem SAMBA Animationen zu zwei Algorithmen. Anschließend äußerten sie sich auf einem

Fragebogen zu ihren Erfahrungen. Stasko zieht eine positive Bilanz zum Einsatz der Lehrmethode. Die Studierenden hatten die Algorithmen sehr gut verstanden und waren sehr motiviert bei der Arbeit.

„It also was clearly evident, however, that the animations helped the students to truly learn and understand the two algorithms under study. Feedback on the student surveys indicated this as students noted how they felt that they understood an algorithm well, but then discovered how their early conceptions were incorrect. Likewise, although an informal measure, student performance on final exam questions regarding the two algorithms was nearly perfect.“ ([Stasko 1996], S. 11)

„The animation assignments provided a forum that encouraged diligence, study and creativity in the students, and did this in an engaging way. Is this not the fundamental objective of education?“ (ebd., S. 12)

Stasko führt dies darauf zurück, dass die Studierenden gerne programmieren und dass man am besten lernt, wenn man lehrt.

„Fundamentally, building an algorithm animation forces a student to become the teacher, if only for a brief time. Instructors certainly know that the best way to learn a concept is to have to teach it.“ (ebd., S. 12)

Hundhausen: fachliche Kommunikationsfähigkeit entwickeln

Christopher Hundhausen hat Staskos Lehrmethode empirisch untersucht und unter soziokulturellen Gesichtspunkten weiterentwickelt. Seine Arbeiten werden unten ausführlich besprochen (vgl. Kap. 3.3, S. 29). Hundhausen bestätigt den Wert dieser Lehrmethode, kritisiert aber, dass es für Studierende recht aufwändig ist, Animationen mit Visualisierungssystemen wie SAMBA zu erstellen.

Rößling: Interaktiver Editor für Animationen

Ein Weg um den Aufwand für die Studierenden zu senken, ist der Einsatz direkt-manipulativer Animationseditoren, die ähnlich einem Zeichenprogramm bedient werden. Als ein aktueller Vertreter soll hier der Animationseditor ANIMAL vorgestellt werden.

ANIMAL wurde von Guido Rößling an der Universität Siegen entwickelt [Rößling 2001b]. Das Werkzeug läuft plattformunabhängig unter Java. Es ist kostenlos verfügbar und auch ohne Netzanbindung lauffähig. Neben der Veränderung der Daten kann der Pseudocode angezeigt werden, wobei die aktive Zeile hervorgehoben wird. Die Schritte der Animation können zeitlich genau gesteuert werden. Animationen können über API-Aufrufe, eine Skriptsprache oder einen interaktiven Editor erzeugt werden. Mit dem Editor können Objekte auf die Zeichenfläche platziert und verschoben werden (Abb. 10). Eigenschaften der Objekte, wie ihr dynamisches Verhalten, werden über Dialogfelder eingestellt.

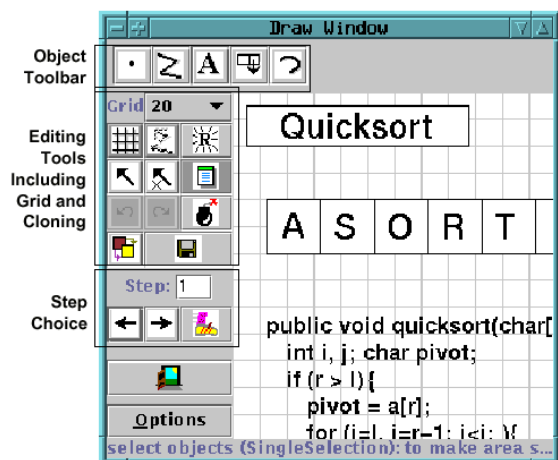


Abb. 10: ANIMAL - Interaktiver Editor für Animationen [Rößling 2001b], Fig. 1.

Dank der einfachen Bedienbarkeit konnten auch studentische Tutoren Animationen mit ANIMAL entwickeln, so dass nun rund 50 Animationen zur Verfügung stehen. Sie wurden in Vorlesungen an der Universität Siegen verwendet, um Studierenden die Algorithmen zu erklären. Diese empfanden die Animationen als motivierend und als hilfreich für das Verständnis der Algorithmen.

3.1.3 Lerner programmieren Algorithmus

Bei der dritten Kategorie des aktiven Lernens von Algorithmen erhalten die Lerner die Aufgabe, den Algorithmus selbst zu programmieren. Der Pseudocode des Algorithmus wird zunächst auf herkömmliche Weise in einer Vorlesung oder über ein Lehrbuch gelernt. Die Lerner arbeiten mit einer Entwicklungsumgebung, in der die Datenstruktur und grundlegende Basisoperationen des Algorithmus bereits implementiert sind. Als wichtiger Vertreter dieser Lehrmethode wird im folgenden Ab-

schnitt der Ansatz von Amir Michail besprochen, der Studierende den Algorithmus in einer visuellen Programmiersprache entwickeln lässt.

3.2 Visuelle Programmierung mit OPSIS

3.2.1 Übersicht

Amir Michail stellt in [Michail 1996] eine Lehrmethode für Algorithmen vor, die Elemente aus den Bereichen Programmierung und Algorithmen-Animation mit Beweistechniken kombiniert. Die Studierenden erstellen in einer für diesen Zweck geschaffenen visuellen Programmiersprache Algorithmen für binäre Suchbäume. Der erstellte Programmgraph ist zugleich eine Art visueller Beweis für die Korrektheit des Programms. Anschließend können sie das visuelle Programm auf Beispieldaten ausführen und erhalten einen visuellen Trace des Programms. Der Trace zeigt in einer Folge statischer Bilder, die ausgeführten Operationen und die Zustände der Datenstruktur.

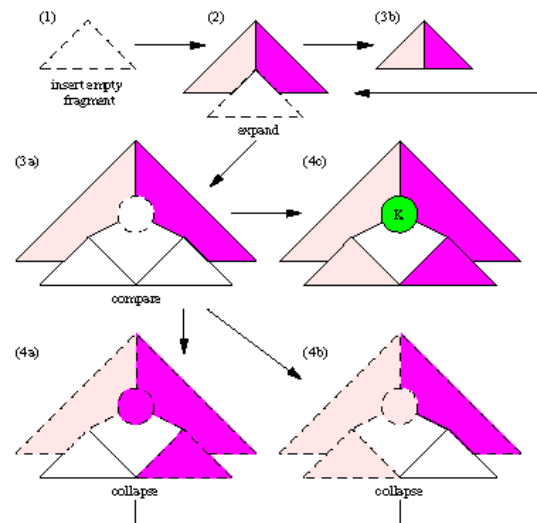


Abb. 11: Visuelles Programm für Suche im Binärbaum
[Michail 1996], S. 3, Fig. 1

3.2.2 Notation der visuellen Programmiersprache

Abb. 11 zeigt ein visuelles Programm für die Suche nach einem Knoten mit Schlüssel K in einem binären Suchbaum. Ein Knoten des Programmgraphen steht für eine Menge von konkreten Binärbäumen, die bestimmte Bedingungen erfüllen. Das grafische Symbol des Knotens beschreibt den Aufbau des binären Baumes aus Fragmenten. Die Farbe der Fragmente bezeichnet die Bedingung, die für die Knoten des Fragmentes gilt. So steht z.B. Pink für Knoten kleiner K und Magenta für Knoten größer K. Unter dem Symbol ist die Operation angegeben, die in diesem Zustand ausgeführt wird. Parameter der Operation sind dabei die gestrichelt umrahmten Fragmente. Nach Ausführung der Operation kann die Datenstruktur verschiedene Zustände haben. Daher gibt es zu einem Zustand des Programmgraphen ein oder mehrere Folgezustände. Schleifen entstehen dadurch, dass der Folgezustand ein bereits bestehender Zustand ist (in Abb. 11 ist dies von (4a) nach (2) der Fall).

3.2.3 Programme sind nicht strukturierbar

Während kleine Programme, wie das in Abb. 11, gut zu überblicken sind, werden größere Programme schnell unübersichtlich. Dieses Phänomen zeigt sich bereits beim dritten Programm ([Michail 1996], S. 4, Fig. 3). Michails visuelle Programmiersprache enthält keine Mechanismen, um bereits definierte Programme als Hilfsfunktionen aufzurufen. Hilfsfunktionen würden es aber erlauben, größere Programme in überschaubare Teile zu zerlegen, die getrennt verstanden und bearbeitet werden können. Michail begründet seine Ablehnung einer solchen Strukturierung mit dem erhöhten Lernaufwand der Studierenden. Dass Hilfsfunktionen eigentlich benötigt werden, sieht man aber bereits im zweiten Programm ([Michail 1996], S. 3, Fig. 2). Michail durchbricht dort die von ihm geschaffene Notation, indem er die zuvor definierte Suchen-Funktion als Hilfsfunktion aufruft.

3.2.4 Das Werkzeug OPSIS

Für die visuelle Programmiersprache entwickelte Michail den grafischen Editor und Interpreter OPSIS. Abb. 12 zeigt OPSIS im Editiermodus. In der Mitte ist ein Zustand des Programmgraphen dargestellt. Der Nutzer selektiert Fragmente und wählt eine Operation aus. Daraufhin erzeugt OPSIS automatisch die Folgezustände. Dies wiederholt der Nutzer, bis alle benötigten Zustände erzeugt sind und das visuelle Programm damit erstellt ist.

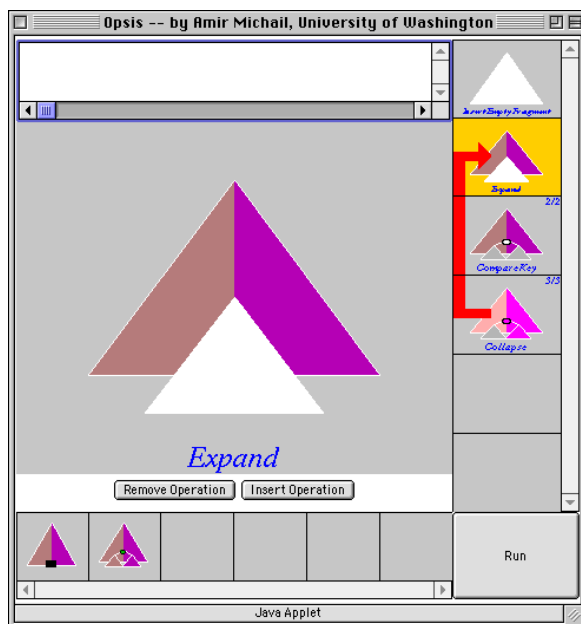


Abb. 12: OPSIS im Editiermodus
Bildschirmabzug von [Michail 1998]

Programmgraph wird nicht dargestellt

Der Programmgraph wird allerdings nicht als Grafik dargestellt. OPSIS stellt nur einen kleinen Ausschnitt aus einer Liste der Zustände dar (Abb. 12, rechts neben dem aktuellen Zustand). Als Nutzer hat man daher nie den Überblick, den z.B. Abb. 11 bietet. Auch ersetzt der Nachfolgezustand überganglos den bisher aktuellen Zustand auf dem Bildschirm. Beides erschwert nach meiner Erfahrung mit OPSIS die Orientierung im Programmgraphen. Entsprechend äußert sich auch eine Nutzerin:

“When a change was made and I progressed to the next state it was almost impossible to find where I had previously been working.” ([Michail 1996], S. 12)

Mächtige Operationen

Michail stellt bewusst recht mächtige Operationen zur Verfügung, die es den Studierenden ermöglichen, sich auf den Kern des Algorithmus zu konzentrieren. Sie sollen davon entlastet werden, sich Gedanken um Implementierungsdetails, wie z.B. der Verwaltung von Zeigern, zu machen.

„Generally speaking, one should provide operations that are high-level but that do not trivialize the algorithm being taught. Moreover, we provide only operations that maintain the key ordering property of the binary search tree; otherwise, we feel that the operation is too low-level and error prone.“ ([Michail 1996], S. 6)

Ausführen des Programms

Nachdem das Programm erstellt ist, können es die Studierenden auf Beispieldaten ausführen. Abb. 13 zeigt einen von OPSIS erzeugten visuellen Trace zum Programm aus Abb. 11.

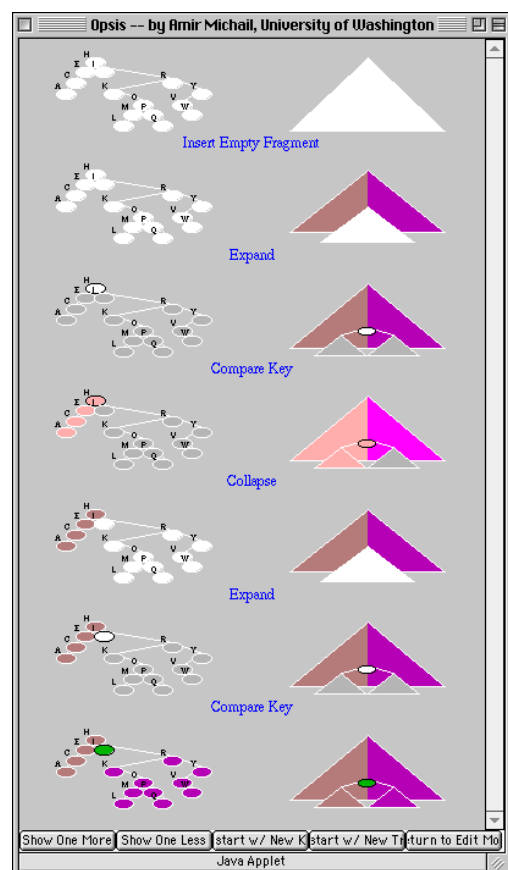


Abb. 13: OPSIS im Tracemodus
Bildschirmabzug von [Michail 1998]

3.2.5 Erprobung der Lehrmethode

Michail erprobte die Lehrmethode in den Übungen einer Vorlesung zu Algorithmen und Datenstrukturen. Die Hörer studierten Informatik als Nebenfach („non-majors“) und waren im dritten Studienjahr. Sie hatten die Aufgabe, sich eigenständig zwei Algorithmen auf balancierten binären Suchbäumen aus Lehrbüchern zu erarbeiten und anschließend zu programmieren. Mindestens einer der Algorithmen musste mit OPSIS implementiert werden. Für den verbleibenden Algorithmus konnte eine textuelle Programmiersprache verwendet werden.

Aus der Erprobung ergaben sich u.a. folgende Erkenntnisse:

- Es gelang den Studierenden, sich in das für sie neue Paradigma der visuellen Programmierung einzudenken und die Algorithmen mit OPSIS zu programmieren.
- Entgegen der Erwartung der Autoren zeigte sich weder beim Programmieraufwand noch beim Verständnis des Algorithmus ein eindeutiger Unterschied zwischen Gruppen die einen Algorithmus visuell programmiert hatten und denen, die ihn textuell implementierten. Die Autoren erklären dies mit der zu kleinen Stichprobe, da nur acht Gruppen teilgenommen hatten und die Werte sehr stark von Gruppe zu Gruppe schwankten.
- Die Studierenden gaben an, dass die Visualisierungen der binären Bäume und der Schleifeninvarianten ihnen geholfen hatten, die Algorithmen besser zu verstehen.

3.2.6 Bewertung des Ansatzes

Abstraktion und Strukturierung

Abstraktion und Strukturierung sind meines Erachtens sehr wesentliche Arbeitsmittel der Informatik. Vergleicht man OPSIS mit typischen Algorithmen-Animationen, so fällt an OPSIS der hohe Grad an Abstraktion auf. OPSIS arbeitet auf abstrakten Zuständen der Datenstruktur, die über Bedingungen auf Fragmenten definiert sind. Dadurch lassen sich die wesentlichen Fälle der Belegung der Datenstruktur kompakt darstellen.

Neben der Datenstruktur wird auch die Lösung des algorithmischen Problems auf der Ebene eines allgemeingültigen Programms angegangen statt auf der Ebene von Schrittfolgen (Trace) für mehrere Probleminstanzen (Eingabedaten). Hilfreich für

die Studierenden sind die leistungsstarken Operationen, die OPSIS zum Aufbau der Lösung bereitstellt.

Als großer Nachteil erscheint mir das Fehlen von Hilfsfunktionen in OPSIS. Das Abstraktionsniveau ist dadurch „festgefroren“. Es ist den Studierenden nicht möglich, ihr Programm zu strukturieren und Programmteile für verwandte Probleme wiederzuverwenden.

Die Abstraktion der Datenstruktur und der Programmschritte ermöglichen es, mit einem visuellen Programm gleichzeitig eine Art visuellen Beweises der Korrektheit des Programmes zu erstellen. Meines Wissens hat Michail damit Neuland betreten. Im Bereich der Algorithmen-Animation kenne ich keine derartige Verknüpfung von Beweistechniken mit visuellen Darstellungen der Datenstruktur. Ich sehe hierin eine kreative Leistung Michails und einen Fortschritt für die Ausdrucksmöglichkeiten in der Darstellung von Algorithmen für Lehrzwecke. Es ist damit möglich, in Vorlesungen einen Beweis für die Korrektheit eines Algorithmus sehr anschaulich auszudrücken. Der Dozent kann z.B. den Programmgraphen wie eine Karte durchgehen und die Beweisschritte mündlich kommentieren.

Visualisierung und Interaktion

Die Visualisierung der abstrakten Zustände der Datenstruktur ist ansprechend und aussagekräftig. OPSIS setzt moderne Interaktionstechniken ein. So wird die Fragmentfarbe in OPSIS textuell erklärt, sobald man mit dem Mauszeiger ein Fragment berührt (Tooltip). Operationen werden einfach aus einer Liste ausgewählt und ihre Parameter in der Datenstruktur markiert.

Zu bemängeln ist, dass der Zustandsgraph in OPSIS nicht visualisiert und der Folgezustand übergangslos eingeblendet wird.

Lernunterstützung und Aktivität

Die von Michail präsentierte Lehrmethode und das Werkzeug OPSIS bieten kaum Unterstützung für ein eigenständiges Lernen, wie es z.B. im Fernstudium üblich ist. Mit dem Werkzeug sind keine Lehrtexte zum Lerninhalt oder den Aufgaben verknüpft. OPSIS berücksichtigt nicht, dass gerade eine Aufgabe bearbeitet wird. Es gibt keine kontextabhängigen Tipps zur Lösung der Aufgabe, keine Hinweise wie nahe man einer Lösung gekommen ist und keine Musterlösung. Erschwerend kommt hinzu, dass den meisten Studierenden

das Paradigma der visuellen Programmierung und die hier genutzte Notation noch unbekannt sein dürften.

Positiv an der Lehrmethode ist, dass die Studierenden sehr aktiv lernen, indem sie selbst einen Algorithmus programmieren. Dies ist wesentlich anspruchsvoller, als z.B. einen gegebenen Algorithmus von Hand abzuarbeiten oder die Schritte eines Algorithmus für eine konkrete Problem Instanz zu finden. OPSIS wurde im Rahmen einer klassischen Präsenzveranstaltung erfolgreich erprobt. Für ein eigenständiges Lernen, z.B. im

Rahmen eines Fernstudiums, sollte es aber um Lehrtexte und kontextabhängige Hilfen zur Lösung der Aufgaben erweitert werden.

Fazit

Michails Lehrmethode und das Werkzeug OPSIS sind ein innovativer und vielversprechender Ansatz, Algorithmen visuell und aktiv zu lernen. Um den Studierenden ein eigenständiges Lernen zu ermöglichen, müssten die genannten Schwächen beseitigt werden. Leider scheinen die Arbeiten an dem Projekt aber seit einigen Jahren zu ruhen.

3.3 Studierende entwickeln und präsentieren Animationen

3.3.1 Lerntheoretischer Hintergrund

In der Wirtschaft und an den Hochschulen werden viele Entwicklungsaufgaben im Team bearbeitet. Die entwickelten technischen Systeme werden den Fachkollegen präsentiert und mit diesen diskutiert. Deshalb sollte ein Informatiker nicht nur Fachwissen, sondern auch die Kompetenz besitzen, über technische Inhalte zu kommunizieren. Neuere pädagogische Richtungen greifen diese Anforderung auf und betonen das Lernen in sozialen Zusammenhängen sowie die Anwendung des erlernten Wissens zur Lösung von Problemen.

Christopher Hundhausen ordnet seine Arbeiten in die Lerntheorie des *Soziokulturellen Konstruktivismus* ein ([Hundhausen 1999], Kap. 3). Das Lehrziel ist dort, dem Studierenden Fachwissen und soziale Fähigkeiten zu vermitteln, die es ihm ermöglichen, ein vollwertiges Mitglied einer professionellen Arbeitsgruppe zu werden. Als eine entscheidende soziale Kompetenz für die Mitarbeit in einer Arbeitsgruppe zu Algorithmen sieht Hundhausen die Fähigkeit, einen Algorithmus visuell aufzubereiten und mit Fachkollegen über Ablauf, Korrektheit und Effizienz des Algorithmus zu diskutieren.

3.3.2 Erprobung der Lehrmethode

Hundhausen entwickelte Aufgaben zur Vorlesung „Algorithms“ an der Universität von Oregon (USA), die es den Studierenden ermöglichen sollten, diese Fähigkeit auszubilden ([Hundhausen 1999], Kap 4). Die Studierenden sollten selbst eine Algorithmen-Animation entwickeln, diese anschließend vor dem Dozenten und den Mitstudie-

renden präsentieren und Fragen zum Algorithmus mit ihnen diskutieren. Das Verhalten und die Einschätzung der Studierenden erfasste Hundhausen mit ethnographischen Techniken wie teilnehmender Beobachtung, Videoaufzeichnung und Fragebogen. Im ersten Durchlauf der Veranstaltung erstellten die Studierenden die Animationen mit dem Visualisierungssystem SAMBA. Es zeigte sich aber, dass dabei sehr viel Implementierungsarbeit anfiel, die von der Beschäftigung mit den Kernfragen des Algorithmus ablenkte. Daher wurde im folgenden Semester die Aufgabenstellung verändert.

Beliebige Eingabe oder Satz Beispieldaten

Die erste Änderung war, dass die Animation nur für einen vorgegebenen festen Satz Eingabedaten funktionieren musste statt für beliebige Eingaben. Dadurch wurde es wesentlich einfacher, die Daten des Algorithmus zu visualisieren, da keine allgemeinen Layoutverfahren eingesetzt werden mussten.

Es ist unstrittig, dass ein Algorithmus alle Problem Instanzen lösen muss. Es gibt aber unterschiedliche Meinungen dazu, ob auch eine für Lernzwecke eingesetzte Animation beliebige Eingaben verarbeiten muss (sog. „*Input Generality*“, beliebige Eingabe). Mit der beliebigen Eingabe soll dem Lerner ermöglicht werden, frei mit Problem Instanzen zu experimentieren und das Verhalten des Algorithmus zu beobachten. Ebenso wie Hundhausen bin ich hingegen der Meinung, dass nicht alle Problem Instanzen visualisiert werden müssen. Es gibt zumeist nur eine kleine Zahl typischer Abläufe, in denen sich die verschiedenen Fälle der Problemlösung und Unterschiede im Bearbeitungsaufwand

zeigen. Da es für den Betrachter im Allgemeinen schwer ist, für jeden wichtigen Fall ein Exemplar Eingabedaten zu finden und es unnötig ist, zwei Abläufe zum gleichen Fall zu betrachten, reicht es, dem Betrachter eine Sammlung der interessanten Fälle zu zeigen.

Konzeptphase mit Storyboards

Die zweite Änderung der Aufgabenstellung zielte darauf ab, die didaktischen und konzeptionellen Aspekte der Algorithmen-Animation stärker zu betonen. Dazu wurde vor die Erstellung der SAMBA-gestützten Animation eine Konzeptphase eingeschoben. In ihr erstellten die Studierenden ein Storyboard der Animation, präsentierten es vor dem Dozenten und ihren Mitstudierenden und diskutierten Fragen zum Algorithmus mit ihnen. Dadurch konnten sie ihre Ideen vorstellen und Anregungen für die Animation erhalten.

Storyboards wurden bereits in den 1930er-Jahren in der Trickfilmbranche verwendet. Ein Storyboard zeigt in einer Folge von statischen Bildern, den sogenannten „Key Frames“, die wesentlichen Ereignisse eines noch zu erstellenden Trickfilms. Die Bilder können einfache Skizzen sein und müssen nicht der grafischen Qualität des Endprodukts entsprechen. Ein Vortragender zeigt die Bilder und erklärt, was auf den Bildern zu sehen ist und was zwischen den Bildern passiert. Dadurch können sich die Betrachter eine Vorstellung vom geplanten Trickfilm machen und Ihre Meinung und Ideen zur Gestaltung des Trickfilms beisteuern. Da Algorithmen-Animationen einige Gemeinsamkeiten mit Trickfilmen haben, kann man ein Storyboard für die Planung einer Animation einsetzen.

Während der Präsentation zeigten die Studierenden entweder eine vorbereitete Folge statischer Bilder oder veränderten ein Ausgangsbild entsprechend dem Fortschreiten des Algorithmus. Dazu markierten Sie z.B. mit einem Stift Elemente der Darstellung oder bewegten Objekte vor einem statischen Hintergrund. Die Bewegung von Objekten in der Animation wurde oft durch Handbewegungen angedeutet. Die Bilder waren zumeist mit einem Grafikprogramm, teils aber auch von Hand gezeichnet worden. Die meisten Studierenden verwendeten Overheadfolien, andere verwendeten Papier und Pappe.

Es gab eine lebhaftere Kommunikation während der Präsentation. Die Zuhörer stellten Verständnisfragen und der Dozent gab Anregungen, wie die Ge-

staltung der Animation verbessert werden könnte. Die diskutierten Fragen ließen sich in die folgenden Kategorien einordnen:

- Welche Aspekte eines Algorithmus sollen illustriert werden?
- Wie sollen diese Aspekte dargestellt werden?
- Was sind angemessene Beispiele für Eingabedaten?
- Wie sollen mehrere Sichten auf den Algorithmus koordiniert werden?
- Wie kann ein bestimmter Aspekt des Storyboards in der Animation implementiert werden?
- Was ist ein angemessener Umfang für das Projekt?

Vergleich von Konzept- und Implementierungsphase

Wie erhofft wurden bei der Präsentation des Storyboards vor allem konzeptionelle und didaktische Fragen zur Algorithmen-Animation besprochen. Dagegen kam es bei der Präsentation der SAMBA-gestützten Animation kaum zu einem Gedankenaustausch, sondern die Studierenden erklärten den Algorithmus und berichteten von ihren Erfahrungen mit der Implementierung. Somit trug die Konzeptphase wesentlich stärker zum Lehrziel der Kommunikationsfähigkeit bei. Hinzu kommt noch, dass für die Konzeptphase wesentlich weniger Arbeitszeit benötigt wurde. Die Studierenden brauchten für die Erstellung des Storyboards durchschnittlich 6,3 Stunden, während für die SAMBA-gestützte Animation 33,2 Stunden anfielen.

3.3.3 Das Visualisierungswerkzeug ALVIS

Erstellen der Animation

Storyboards hatten sich in der Erprobung als ein Kommunikationsmedium bewährt, da sich die Studierenden bei deren Erstellung und Präsentation vor allem mit den Kernkonzepten eines Algorithmus auseinandersetzten. Dagegen war die Programmierung der Animation mit SAMBA recht mühevoll und zeitaufwändig. Ausgehend von diesen Erfahrungen entwickelte Hundhausen ein Visualisierungswerkzeug, das auf der Metapher der Storyboards basiert ([Hundhausen 1999], Kap. 7). Es soll ein ähnlich intuitives Erstellen und Präsentieren einer Animation ermöglichen, wie es bei herkömmlichen Storyboards der Fall ist.

Das Werkzeug ALVIS ist ein Editor und Interpreter für Programme in der Sprache SALSA. Abb. 14 zeigt einen Bildschirmabzug des ALVIS-Werkzeugs. Das linke Teilfenster zeigt das SALSA-Programm als eine Folge von Textzeilen. Die im Programmablauf erzeugten grafischen Objekte werden zunächst im rechten unteren Teilfenster als Symbol dargestellt. Ein Problem bei der Arbeit mit SAMBA war, dass Variablen und Datentypen mit Hilfe geometrischer Objekte und kartesischer Objektkoordinaten dargestellt werden mussten. Bei ALVIS kann man ein neu erzeugtes Objekt über intuitivere relative Bezüge zu bestehenden Objekten (z.B. „left-of“) platzieren. Alternativ kann man es per Drag-and-Drop aus dem rechten unteren Teilfenster auf die Leinwand im rechten oberen Teilfenster ziehen. ALVIS setzt dann automatisch die kartesischen Koordinaten ein. Auch Bewegungen der Objekte können direkt-manipulativ über Mausebewegungen spezifiziert werden.

Die bei anderen Visualisierungswerkzeugen übliche Trennung von internen Variablen und den sie repräsentierenden grafischen Objekten ist in SALSA weitgehend aufgehoben. Zum Beispiel können Schleifenvariablen durch grafische Objekte realisiert werden. Ihr Wert ist dann durch ihre relative Lage zu anderen Objekten auf der Leinwand gegeben. Der Variable kann durch relative geometrische Operationen ein neuer Wert zugewiesen werden. Um die Schleifenbedingung zu testen, wird ihre Lage mit der anderer Objekte verglichen.

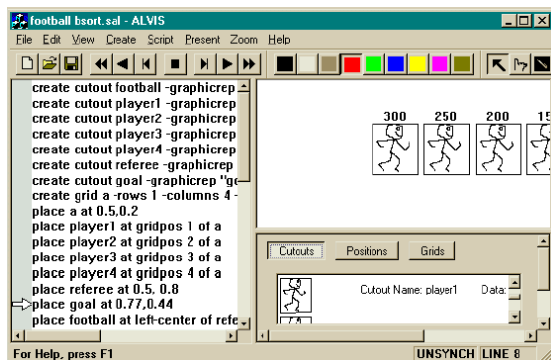


Abb. 14: Erstellen der Bubblesort-Animation in ALVIS [Hundhausen 1999], S. 103, Fig. 41

Abb. 14 zeigt das Erstellen einer Animation des Bubblesort Algorithmus in ALVIS. Die Visualisierung basiert auf der Metapher des Spieles American Football. Der Ball dient in der Animation als Schleifenvariable. Der Schiedsrichter wirft den Ball zum am weitesten links stehenden Spieler. Dieser läuft solange Richtung Tor (indem er den Platz mit seinem rechten Nachbarn tauscht), bis er auf einen stärkeren Spieler trifft. Er gibt dann den Ball an diesen Spieler ab, der nun die Rolle des aktiven Spielers übernimmt und Richtung Tor läuft. Er-

reicht ein Spieler die bereits sortierte Teilfolge, so ist er an seiner endgültigen Position angelangt und gibt den Ball an den Schiedsrichter ab. Der Durchlauf beginnt nun von neuem.

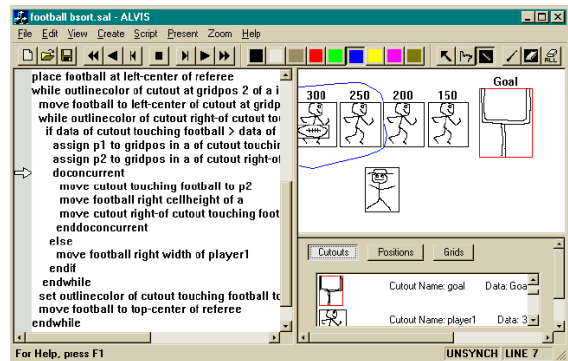


Abb. 15: Präsentieren der Animation in ALVIS [Hundhausen 1999], S. 114, Fig. 54

Präsentieren der Animation

ALVIS erlaubt es, die Animation sehr flexibel zu präsentieren. Man kann in Einzelschritten vor und zurück durch die Animation schalten. Der Vortragende kann mit einem vergrößerten Mauszeiger auf Objekte zeigen und mit einem Stiftwerkzeug Anmerkungen anbringen (Abb. 15). Im Gegensatz zu den üblichen Visualisierungswerkzeugen kennt ALVIS keine strikte Trennung von Editiermodus und Präsentationsmodus. Es ist jederzeit möglich, das SALSA-Programm während der Präsentation zu ändern. Die Animation wird dann sofort angepasst.

3.3.4 Bewertung der Arbeiten Hundhausens

Die große Stärke der Lehrmethode liegt meines Erachtens in der Ausbildung der Kommunikationsfähigkeit. Viele Studierende empfinden es als motivierend, selbst eine Algorithmen-Animation zu erstellen. Dadurch, dass sie für kurze Zeit die Rolle eines Dozenten übernehmen und in der fachlichen Diskussion bestehen können, stärken sie ihr Selbstbewusstsein und ihr Vertrauen in ihre fachlichen Fähigkeiten. Da sich die Studierenden sehr intensiv mit dem zu präsentierenden Algorithmus auseinandersetzen, werden sie ihn in der Regel gut verstehen. Durch die Diskussionen in der Arbeitsgruppe und bei der Präsentation des Storyboards können die Studierenden verbleibende Verständnislücken schließen.

Als abschließende größere Aufgabe zu einer herkömmlichen Lehrveranstaltung erscheint mir der Arbeitsaufwand von rund 36 Stunden als vertret-

bar. Wegen des Aufwandes kann ein Studierender aber nur einen der in der Vorlesung behandelten Algorithmen für seine Präsentation aufbereiten. Für das Erlernen der verbleibenden Algorithmen der Vorlesung leistet die Lehrmethode daher keinen Beitrag. Im Gegensatz dazu lassen sich mit der Methode SALA dieser Arbeit auch eine größere Zahl an Algorithmen erlernen, da der Arbeitsaufwand kaum höher als beim Durcharbeiten eines Lehrbuches ist.

Ein wertvoller Beitrag zur Fachdidaktik erscheint mir die sorgfältige Beobachtung der Studierenden in der Konzeptionsphase und der Implementierungsphase sowie der Vergleich der beiden Phasen. Das technisch einfachere Kommunikationsmittel Storyboard führte, verglichen mit der SAMBA-gestützten Animation, bei geringerem Arbeitsaufwand zu einer stärker an den Konzepten des Algorithmus orientierten Diskussion.

Die meisten Evaluationen der Wirksamkeit von Algorithmen-Animationen versuchten, den Einfluss bestimmter Faktoren quantitativ zu bestimm-

men (siehe z.B. "Untersuchungen zur Effektivität" S. 16ff). Demgegenüber vermitteln die von Hundhausen benutzten qualitativen ethnographischen Techniken einen tieferen Einblick in die psychosozialen Vorgänge einer Arbeitsgruppe und geben konkretere Hinweise für die Gestaltung von Arbeitswerkzeugen wie ALVIS. Hundhausen hat sein Vorgehen sehr sorgfältig dokumentiert, so dass zukünftige Forscher hier Anregungen für ihre Arbeit entnehmen können.

Die in der Methode SALA zentralen Konzepte Strukturierung und Invarianten werden in Hundhausens Arbeit leider überhaupt nicht angesprochen. So gibt es z.B. in seiner Sprache SALSa keine Möglichkeit, Unterprogramme zu definieren und aufzurufen.

Das von Hundhausen geschaffene ALVIS-Werkzeug hebt sich in vielen Punkten innovativ von bestehenden Visualisierungssystemen ab. Es ist ein positives Beispiel für ein Werkzeug, das ausgehend von den Bedürfnissen der Benutzer für eine Aufgabe entwickelt wurde.

4 Die Gestaltungsmethode „Strukturiertes Aktives Lernen von Algorithmen (SALA)“

4.1 Einführung

4.1.1 Das Potential multimedialer Lernprogramme

Lernmedien für das eigenständige Lernen stehen als Mittler zwischen dem Lerner und dem Lerngegenstand. Sie sollten von ihren Autoren so gestaltet werden, dass sie dem Lerner einen möglichst ungehinderten, eigenständigen und angemessenen Zugang zum Lerngegenstand ermöglichen. Das Lernmedium „multimediale interaktive Lernprogramme“ hat hierbei für das Lerngebiet Algorithmen ein besonders großes Potential. Unter den mir bekannten Lernprogrammen, die (auch) Algorithmen behandeln, schöpfen selbst die besten dieses Potential noch nicht aus. Zu nennen wären hier z. B. „Animated Algorithms“ ([Gloor 1997], [Gloor, Dynes, Lee 1993]), „Viacobi“ ([Janser 1998a], [Janser 1998b]) und „ORWelt“ ([Blumstengel 1998a], [Blumstengel 1998b]).

Diese Lernprogramme nutzen im erklärenden Teil Texte, Bilder und Algorithmen-Animationen. Aufgaben sind, falls vorhanden, meist auf Multiple-Choice-Fragen beschränkt. Es stellt sich daher die Frage, wie Aufgaben für das Medium Lernprogramm zum Themengebiet Algorithmen gestaltet werden können.

4.1.2 Die Methode SALA

In diesem Kapitel stelle ich die von mir entwickelte didaktische Methode „Strukturiertes Aktives Lernen von Algorithmen (SALA)“ vor. Sie beschreibt eine Möglichkeit, Lernprogramme zum Themenbereich Algorithmen zu gestalten. Die Methode zielt vor allem auf die zu erstellende Struktur des Lernprogramms und die darin enthaltenen interaktiven Aufgaben und weniger auf den Weg dorthin. Demgegenüber strukturieren allgemeine Gestaltungsmethoden den Entwicklungsprozess indem Sie Arbeitsschritte, Sichten und Notationen für die Analyse der Anforderungen und die Entwicklung des Produkts definieren und vorgeben. Beispiele für allgemeine didaktische Gestaltungsmethoden sind die in unserer Abteilung entwickelten Methoden „Didaktisches Design telematik-ge-

stützter Lernsoftware“ [Gorny 1998] und „Didaktisches Interaktions- und Informationsdesign“ [Donker 2001].

Die Methode SALA sieht vor, den Algorithmus modular in Funktionen zu gliedern, die jeweils für sich beschrieben werden. Der Pseudocode der Algorithmen-Funktionen wird dem Lerner nicht sofort präsentiert, sondern soll von ihm in einem Prozess des entdeckenden Lernens selbst entwickelt werden. Die Methode gibt Hinweise für eine Gliederung des Lernprogramms in Sektionen und Funktionen und die Gestaltung von interaktiven Simulationen für das entdeckende Lernen.

Angestrebtes Lehrziel

Die Methode SALA ist darauf ausgelegt, ein tiefes Verständnis für die Funktionen eines Algorithmus zu vermitteln. Der Lerner soll sowohl die Datenstruktur als auch die Schritte der Funktionen verstehen. Für die Datenstruktur soll er lernen, welche Typen von Objekten es gibt, wie sie miteinander verzeigert sind und welche (mathematischen) Bedingungen zwischen den Objekten gelten. Es soll ihm klar werden, aus welchen Schritten eine Funktion besteht, warum die Schritte erlaubt sind und zum Ziel führen.

Das Verständnis des Algorithmus umfasst damit auf jeden Fall die Fähigkeit, den Algorithmus von Hand auszuführen. Das didaktische Ziel von SALA geht aber darüber hinaus. Der Lerner soll befähigt werden, den Algorithmus um neue Funktionen zu erweitern. Daher wird er schon beim Erlernen der Funktionen in die Rolle eines Algorithmus-Entwicklers gestellt und soll kreativ und problemlösend tätig werden.

Dieses tiefe Verständnis, ist eine gute Voraussetzung, um den Algorithmus zu programmieren oder mathematisch zu analysieren. Man könnte natürlich auch versuchen, einen Algorithmus einfach Zeile für Zeile aus einer Pseudocode-Beschreibung in ein Programm zu übertragen, ohne ihn verstanden zu haben. Damit würde aber eine wichtige Kontrolle auf Korrektheit entfallen und es ist fraglich, ob man die Möglichkeiten der Programmiersprache ausschöpfen könnte. In den meisten Fällen wird auch die mathematische Analyse nur

gelingen, wenn man das Zusammenspiel von Schritten einer Funktion und den Objekten der Datenstruktur verstanden hat.

Nach SALA entwickelte Lernprogramme

Karsten Block entwickelte in seiner Diplomarbeit nach der Methode SALA ein Lernprogramm zur Datenstruktur „Binomial Heap“ und den für die Datenstruktur verwendeten Algorithmen (vgl. Kap. 7.1.1). Ebenfalls in einer Diplomarbeit entwickelte Jan Schormann einige interaktive JAVA-Applets zum Voronoi-Algorithmus [Schormann 1999]. Ein von mir nach SALA entwickeltes Lernprogramm zu Heapsort dient in diesem Kapitel als Beispiel für die Anwendung der Methode (vgl. Kap. 4.1.4 und Kap. 7.1.1).

4.1.3 Vom Lehrbuch zum Lernprogramm

Mit einem nach SALA entwickelten Lernprogramm soll es möglich sein, selbständig zu lernen. Damit ist gemeint, dass keine zusätzlichen Texte oder helfende Personen nötig sind, um den Algorithmus zu erlernen. Natürlich kann und soll ein solches Lernprogramm auch begleitend zu herkömmlichen Lehrformen, wie Vorlesung und Übung einsetzbar sein. Es liegt daher nahe, herkömmliche Lehrbücher zu Algorithmen (z. B. [Ottmann, Widmayer 1996], [Cormen, Leiserson, Rivest 1990]) zu betrachten und zu überlegen, wie dortige Präsentationsformen und Lernmethoden in das Medium Lernprogramm übertragen werden können.

Präsentationsformen

Texte und Formeln sollten als Hypertext dargestellt werden. Bilder aus dem Buch können im Lernprogramm wieder als Bilder gezeigt werden. Sollen aber die Bilder im Buch einen dynamischen Vorgang veranschaulichen, wie z. B. Veränderungen der Werte oder der Verzeigerung der Datenstruktur, so wird man im Lernprogramm Animationen zeigen. Autoren von Lernsoftware, die einen Einblick in das Gebiet der Algorithmen-Animation gewinnen wollen, finden hierzu im Buch „Software Visualization“ [SDBP 1998] eine Fülle von Hinweisen und Berichte über einschlägige Projekte.

Aufgaben im Lernmedium

Der erklärende Teil eines Lehrbuches lässt sich damit gut darstellen. Aber wie steht es mit den Aufgaben? Aufgaben fordern den Lerner heraus, sich aktiv mit dem Stoff zu beschäftigen. Sie decken

Wissenslücken auf, festigen das gelernte Wissen und geben einen Rahmen, um abstraktes Wissen auf konkrete Aufgaben oder Situationen anzuwenden. In der didaktischen Literatur wird immer wieder darauf hingewiesen, dass Lerner aktiv am Stoff arbeiten sollen, da sie ihn auf diese Weise besser verstehen und behalten können.

Aufgaben aus Lehrbüchern soll der Lerner typischerweise mit Papier und Stift bearbeiten. Vereinzelt gibt es auch Programmieraufgaben, die dann aber meist zeitaufwändig sind und Lerner ohne Programmiererfahrung überfordern. Beim eigenständigen Lernen gibt es niemanden, der weiterhelfen kann, wenn bei der Programmierung ein Problem auftaucht.

Dem Autor von Lernprogrammen bietet das Medium Computer viele neue Ausdrucksmöglichkeiten: Farbgrafik, Animation, Ton und insbesondere Interaktivität mit dynamischer Hilfestellung und Fehlermeldungen. Es sind mir nur wenige Lernprogramme zu Algorithmen bekannt, die überhaupt interaktive Aufgaben beinhalten. Die Interaktion beschränkt sich dabei auf die Auswahl von Lösungsalternativen bei Multiple-Choice-Tests und auf die Vorhersage des nächsten Schrittes eines Algorithmus (z. B. durch Anklicken eines Objekts mit der Maus). Demgegenüber enthalten nach SALA gestaltete Lernprogramme Aufgaben mit interaktiven Simulationen von Algorithmusfunktionen. Sie ermöglichen einen forschenden und kreativen Zugang zu den Funktionen und Datenstrukturen.

4.1.4 Gestaltung nach SALA am Beispiel des Heapsort

Im Folgenden zeige ich am Beispiel des von mir entwickelten Lernprogramms zum Heapsort-Algorithmus die Gestaltung eines Lernprogramms nach SALA. Heapsort ([Floyd 1964], [Williams 1964]) ist hierfür gut geeignet, da der Algorithmus vielen bekannt ist und sein Pseudocode eine angemessene Komplexität hat. Ich werde den Algorithmus nur skizzieren. Eine detaillierte Beschreibung ist in den meisten Lehrbüchern zu finden.

Heapsort verwendet eine spezielle Datenstruktur, den Heap, nach dem der Algorithmus benannt ist. Der Heap ist ein Binärbaum, bei dem alle Ebenen bis auf die letzte vollständig gefüllt sind. Die unterste belegte Ebene ist von links nach rechts gefüllt, aber nicht notwendigerweise vollständig. In einem Heap muss der Schlüssel an jedem Knoten größer oder gleich den Schlüsseln der Kindknoten sein. Das ist die sog. Heapeigenschaft oder auch

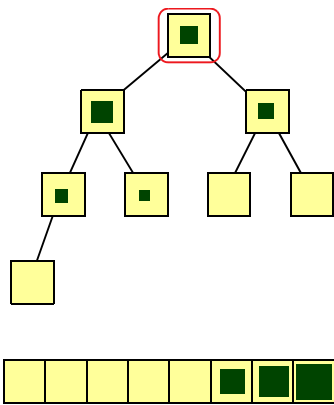


Abb. 16: Heapbaum mit markiertem Knoten

Heapordnung. Heap und Ergebnisliste können effizient in einem einzigen Array gespeichert werden. Abb. 16 zeigt einen Heapbaum bei dem die Schlüsselwerte durch die Größe des inneren Quadrats dargestellt sind. Ein Schlüssel, der die Heapeigenschaft verletzt, ist markiert. Unter dem Baum ist die Ergebnisliste zu sehen. Drei Schlüssel wurden bereits aus dem Heapbaum in die Ergebnisliste verschoben.

4.2 Lernpsychologische Grundlagen

Innovativ an der Gestaltungsmethode SALA ist neben der Modularisierung des Algorithmus insbesondere die Einbeziehung des entdeckenden Lernens. Daher werden nun das „entdeckende Lernen“ und verwandte lernpsychologische Begriffe eingeführt. Aus den Erkenntnissen der lernpsychologischen Forschung lassen sich einige Gestaltungsregeln für Lernmedien ableiten. Sie werden hier auf Elemente der Methode SALA bezogen. Dabei werden Details der Methode angesprochen, die dem Leser möglicherweise erst nach der Lektüre der darauffolgenden Abschnitte verständlich sein werden. Da sich die Darstellung lernpsychologischer Grundlagen und der Methode SALA zyklisch aufeinander beziehen, waren Vorwärtsreferenzen nicht zu vermeiden.

4.2.1 Entdeckendes versus rezeptives Lernen

In den üblichen Lehrbüchern werden Algorithmen dem Lerner als fertig aufbereitetes Wissen präsentiert. Der Lerner hat nur die Aufgabe, das Wissen in sich aufzunehmen. Man nennt diese Lernform *rezeptives Lernen*:

„Bei rezeptivem Lernen [...] wird dem Schüler der vollständige Inhalt von dem, was gelernt werden soll, in seiner fertigen Form übermittelt. Die Lernaufgabe verlangt von ihm keinerlei selbständige Entdeckung. Von ihm wird nur gefordert, daß er sich den Stoff, der ihm gegeben wird [...] so einprägt oder einverleibt, daß er zu einem späteren

Zeitpunkt zur Verfügung steht oder reproduziert werden kann.“ (Ausubel et al., 1980/81, S. 47, zitiert nach [Edelmann 1996], S. 210)

Nach SALA gestaltete Lernprogramme enthalten auch fertig aufbereitetes Wissen (sog. tutorielle Teile). Die Methode zielt aber vor allem darauf, die Lerner den eigentlichen Pseudocode des Algorithmus selbst entwickeln zu lassen. Dazu werden vom Autor Informationen zur Struktur des Algorithmus und zu den Invarianten der Datenstruktur in Simulationen der Algorithmusfunktionen integriert. Die Informationen sollen vom Lerner durch ein experimentelles Arbeiten mit den Simulationen entdeckt werden. SALA folgt damit dem Prinzip des entdeckenden Lernens:

„Das wesentlichste Merkmal des entdeckenden Lernens ist [...] die Tatsache, daß der Hauptinhalt dessen, was gelernt werden soll, nicht gegeben ist, sondern vom Schüler entdeckt werden muß [...]“ (Ausubel et al., 1980/81, S. 47, zitiert nach [Edelmann 1996], S. 211).

Die Fähigkeit zum rezeptiven Lernen entwickelt sich im Kind erst im Laufe der Grundschulzeit. Junge Kinder lernen vor allem über direkte Interaktion mit den Dingen ihrer Umwelt und nur wenig über verbal kommuniziertes Wissen. Die Fähigkeit zum entdeckenden Lernen bleibt auch im Erwachsenenalter erhalten. So werden Probleme des Alltags oft über entdeckendes Lernen gelöst, indem z.B. die Bedienung eines neuen technischen Gerätes über Ausprobieren der Funktionalität erlernt wird ([Edelmann 1996], S. 213).

4.2.2 Entdeckendes Lernen nach Bruner

Ein wichtiger Vertreter des entdeckenden Lernens ist der Lernpsychologe Jerome Bruner. Seiner Ansicht nach soll der Schulunterricht nicht nur Sachwissen vermitteln, sondern das selbständige Denken herausbilden, so dass die Schüler in der Lebenspraxis auch neue Probleme bewältigen können.

„Das Ziel, das wir uns als Lehrer stellen, ist, dem Schüler nach besten Kräften ein fundiertes Verständnis des Gegenstands zu vermitteln und ihn so gut wir können zu einem selbständigen und spontanen Denker zu machen, daß er am Ende der Schulzeit allein weiterkommen wird.“ (Bruner, 1973, S. 16, zitiert nach [Edelmann 1996] S. 214).

Das entdeckende Lernen trägt dazu auf folgende Weise bei (vgl. [Edelmann 1996], Kap. 5.4.6):

Transferförderung

Der Lernstoff soll an exemplarischen Beispielen so dargestellt werden, dass die fundamentalen Begriffe und Regeln des Fachgebietes erlernt werden. Der Lerner kann diese später in neuen Problemsituationen anwenden. Dabei möchte Bruner induktive Denkvorgänge anregen, in denen vom konkreten Einzelfall auf die generelle Regel geschlossen wird.

Bei SALA wird erwartet, dass der Lerner aus der Lösung einzelner Probleminstanzen (Schrittfolgen bei einer Reihe von Eingabedaten) auf den allgemeingültigen Pseudocode der Algorithmenfunktion schließen kann. Dadurch, dass der Algorithmus modular dargestellt wird, soll der Lerner die Prinzipien der Modularisierung verinnerlichen und bei späteren Entwicklungsvorhaben anwenden.

Problemlösefähigkeit

Neben dem konkreten Lernstoff soll der Lerner auch Techniken des Problemlösens erlernen. Deshalb erhält er die Aufgabe, in einer vorbereiteten Lernumgebung ein Problem zu lösen.

„Meiner Meinung nach kann man nur durch Üben des Problemlösens [...] die heuristische Methode der Entdeckung lernen; je mehr man geübt ist, um so eher wird man das Gelernte zu einem Problemlösungs- oder Fragestil verallgemeinern können, der sich auf jede oder

fast jede angetroffene Aufgabenart anwenden läßt.“ (Bruner, 1973, S. 26, zitiert nach [Edelmann 1996], S. 216)

In ihrem Berufsleben werden viele der Informatikstudierenden neue Systeme erschaffen. Daher üben Sie in einem SALA-Lernprogramm an kleinen Aufgaben die Konstruktion von Algorithmenteilen (Funktionsbeschreibungen). Als Problemlösestil dienen die Entwurfsprinzipien modularer Softwareentwicklung.

Intuitives Denken

Durch entdeckendes Lernen wird das intuitive Denken gefördert. Es ist laut Bruner ein wesentlicher Bestandteil kreativer Problemlösungsprozesse. Edelmann umschreibt das intuitive Denken folgendermaßen:

„Das intuitive Denken scheint eher bildhaft und konkret, geht von einzelnen Erfahrungen aus, zielt auf Erfassung des Problems in seiner Gesamtheit, ermöglicht neuartige Lösungen und ist relativ einfallsartig und ggf. sprunghaft.“ ([Edelmann 1996], S. 216)

Die Simulationen der Algorithmenfunktionen in SALA bieten einige der angesprochenen Merkmale und unterstützen dadurch das intuitive Denken bei der Problemlösung. Der Lerner interagiert mit einer bildhaften Darstellung der Daten. Dabei bewegt er sich zunächst auf der konkreten Ebene einzelner Schrittfolgen und nicht auf der abstrakten Ebene des Pseudocodes. Der Abstraktionsschritt von der einzelnen Funktion zum gesamten Algorithmus wird durch die Grafik der funktionalen Struktur unterstützt.

Förderung der intrinsischen Motivation

Das entdeckende Lernen appelliert an die Neugier des Lerners und an den Wunsch, sich als kompetent zu erweisen. In diesem Sinne fördert es die intrinsische Motivation, d. h. ein Lernen aus Interesse an der Sache und nicht aus äußerem Druck.

Ich habe viele Personen mit nach SALA gestalteten Lernprogrammen arbeiten lassen, sie bei der Arbeit beobachtet und sie anschließend nach ihrer Meinung zu dieser Art des Lernens befragt. Dabei hob die überwiegende Mehrzahl hervor, dass das Lösen der Aufgaben mit den Simulationen sie motivierte und ihnen Spaß machte. Es gab aber auch Stimmen, die den höheren Arbeitsaufwand gegenüber rein rezeptivem Lernen kritisierten.

4.2.3 Vertiefung in eine Aufgabe

Der amerikanische Psychologe Mihaly Csikszentmihalyi erforscht seit vielen Jahren den Zustand des *Flow*, bei dem ein Mensch in eine Aufgabe oder Tätigkeit völlig vertieft ist und diese als höchst erfüllend erlebt [Csikszentmihalyi 2001]. Diese Erfahrung ist prinzipiell in allen Lebensbereichen, auch beim Arbeiten und Lernen, möglich.

Es ist sicher nicht möglich, eine Lernumgebung zu Algorithmen zu entwickeln, die jeder Person zu allen Zeiten ein motivierendes und erfüllendes Lernerlebnis verschafft. Die Lernumgebung sollte aber so gestaltet sein, dass sie möglichst viele Voraussetzungen für ein solches Lernerlebnis schafft. Daher werden nun die Kennzeichen und Bedingungen einer Flow-Erfahrung betrachtet.

„Aus unseren Studien geht hervor, daß die Phänomenologie der Freude acht Hauptkomponenten umfaßt. Wenn Menschen darüber nachdenken, wie sie sich fühlen, wenn eine Erfahrung höchst positiv ist, nennen sie gewöhnlich eine, oft auch alle anderen.

Erstens, die Erfahrung findet gewöhnlich statt, wenn wir auf eine Aufgabe stoßen, der wir uns gewachsen fühlen.

Zweitens müssen wir fähig sein, uns auf das zu konzentrieren, was wir tun.

Drittens und viertens, die Konzentration ist gewöhnlich möglich, weil die angefangene Aufgabe deutliche Ziele umfaßt und unmittelbare Rückmeldung liefert.

Fünftens, man handelt mit einer tiefen, aber mühelosen Hingabe, welche die Sorgen und Frustrationen des Alltagslebens aus dem Bewußtsein verdrängt.

Sechstens, erfreuliche Erfahrungen machen es möglich, ein Gefühl der Kontrolle über Tätigkeiten zu erleben.

Siebtens, die Sorgen um das Selbst verschwinden, doch paradoxerweise taucht das Selbstgefühl nach der flow-Erfahrung gestärkt wieder auf.

Und schließlich ist das Gefühl für Zeitabläufe verändert; Stunden vergehen in Minuten, Minuten können sich vermeintlich zu Stunden ausdehnen.

Die Kombination dieser Bestandteile ruft ein tiefes Gefühl von Freude hervor, welches so lohnend ist, daß man bereit ist, viel Energie dafür aufzuwenden, um es immer wieder zu erleben.“ ([Csikszentmihalyi 2001], S. 74)

Zu den Zielen und Regeln schreibt er:

„Doch die bei weitem größte Anzahl optimaler Erfahrungen erfolgt den Angaben nach bei Aktivitäten, die zielgerichtet und durch Regeln gebunden sind - Aktivitäten, für die man psychische Energie einsetzen muß und die ohne entsprechende Fähigkeiten nicht ausgeführt werden können“ (ebd., S. 75)

Dabei ist der Schwierigkeitsgrad der Aufgabe für eine Flow-Erfahrung sehr wichtig:

„Bei allen Aktivitäten, über die die Teilnehmer unserer Studien berichteten, tritt Freude an einem ganz bestimmten Punkt auf: Wenn die Handlungsmöglichkeiten von einer Person als für ihre Fähigkeiten angemessen eingestuft wird.“ (ebd., S. 79)

Nach SALA gestaltete Lernprogramme bieten durch Berücksichtigung der Punkte eins (angemessene Aufgabe), drei und vier (Ziele und Rückmeldung) gute Voraussetzungen für eine Flow-Erfahrung. In SALA sind interaktive Aufgaben integriert, die nach den Erfahrungen aus der Erprobung der Lernprogramme für die Mehrzahl der Lerner eine angemessene Herausforderung darstellten. Der Schwierigkeitsgrad der Aufgaben kann über den Bearbeitungsmodus (Simulation, Übung, Animation) eingestellt werden. Außerdem bietet die in den Simulationen der SALA-Lernprogramme realisierte formale Welt der Datenstrukturen und Algorithmen klare Ziele und Regeln.

4.2.4 Intrinsische Motivation

Kommt der Antrieb zu Lernen aus dem Interesse an der Sache selbst und aus der Freude an der Beschäftigung mit dem Thema, so spricht man von *intrinsischer Motivation* des Lernens. Lerntheoretiker wie Piaget und Bruner schreiben ihr positive Wirkungen auf das Lernen zu. Die Lerner verbringen mehr Zeit mit dem Lerngegenstand und sind eher bereit das Gelernte in der Zukunft anzuwenden. Auch kann das Lernerlebnis tiefere Spuren hinterlassen und grundsätzliche Fähigkeiten wie „Lernen wie man lernt“ ausbilden helfen. Natürlich ist die Freude am Lernen auch für sich allein

genommen schon ein erstrebenswertes Ziel. Daher sollten Lernmedien so gestaltet werden, dass sie die intrinsische Motivation des Lerners anregen.

Thomas Malone hat erforscht, welche Faktoren eines computergestützten Lernmediums die intrinsische Motivation beeinflussen [Malone 1981]. Als Untersuchungsgegenstand wählte er Computerspiele, da sie bekanntermaßen Kinder sehr stark zum Spielen motivieren. Die untersuchten Spiele standen den Schülern einer amerikanischen Schule bereits einige Zeit zur Verfügung. Ein Teil der Spiele diente Lernzwecken (z.B. Bruchrechnen), der größere Teil diente der Unterhaltung. In einer ersten Studie bewerteten die Kinder 25 Computerspiele auf einer Skala von „mag ich nicht“ bis „mag ich sehr“. In zwei folgenden Studien veränderte Malone systematisch Teile eines Computerspiels und erprobte die verschiedenen Versionen mit den Kindern. Er beobachtete, wie lange sie mit der jeweiligen Version spielten und notierte die Aussage der Kinder, wie sehr sie sie mochten.

Malone identifizierte in der Literatur und in seinen Studien drei wesentliche Einflussfaktoren auf die intrinsische Lernmotivation: Herausforderung, Fantasie und Neugier.

Herausforderung

Eine Umgebung ist herausfordernd, wenn Sie ein Ziel setzt, dessen Erreichen unsicher ist. Dazu soll der Schwierigkeitsgrad an die Fähigkeiten des Lerners anpassbar sein. Die Umgebung sollte Ziele auf mehreren Ebenen anbieten, z.B. ein Problem zu lösen und dies in möglichst kurzer Zeit tun. Die zur Lösung nötigen Informationen können bewusst im Spiel versteckt werden, um die Herausforderung zu steigern. Wird die Aufgabe bei jedem Durchgang leicht verändert, so bleibt das Spiel länger interessant, da es schwieriger zu bewältigen ist.

Fantasie

Mit Fantasie bezeichnet Malone die Einbettung der Spielaufgabe in eine Rahmenhandlung oder eine bekannte Situation. Zum Beispiel werden Punkte auf dem Zahlenstrahl durch Ballons dargestellt, die mit einem Wurfpeil getroffen werden sollen, indem ein Bruch eingegeben wird. In einem weiteren Beispiel regiert der Spieler ein simuliertes Königreich. Eine *intrinsische Fantasie* ist mit dem Inhalt der Spielaufgabe verknüpft, während eine *extrinsische Fantasie* nur äußerlich damit verknüpft ist und z.B. nur den Punktestand visualisiert. Intrinsische Fantasien sind interessanter als extrinsi-

sche und auch lernförderlicher, da sie die Behaltensleistung verbessern. Das von Malone als Fantasie bezeichnete Konzept entspricht dem Begriff der Metapher in der Software-Ergonomie.

Christopher Hundhausen wandte das Prinzip der Motivation durch eine Rahmenhandlung in seiner Lehrmethode zu Algorithmen an. Er forderte die Studierenden auf, eine Algorithmen-Animation vor dem Hintergrund einer Geschichte zu entwickeln. Algorithmen-Animationen, die auf einer Geschichte basierten motivierten die Zuhörer dazu, sich an der Diskussion des Entwurfs zu beteiligen:

„Finally, as discussed above, a minority of students' animations depicted algorithms against the backdrop of a story. As it turned out, storyboard presentation participants took great pleasure in refining and further developing the internal logic and structure of these stories.“
([Hundhausen 1999], S. 57)

Neugier

Allgemein streben Menschen danach, ein inneres Modell der sie umgebenden Welt zu entwickeln, das vollständig und konsistent ist und mit wenigen Beschreibungsregeln auskommt. Die Neugier des Spielers kann geweckt werden, indem ihm eine unbekannte Welt präsentiert wird, so dass er ein inneres Modell der Spielewelt aufbauen muss. Diese Welt sollte neu sein und Überraschungen bereithalten, ihre Komplexität sollte den Lerner zwar herausfordern, aber nicht überfordern. Audiovisuelle Effekte können die Neugier anregen, sollten aber möglichst mit der Aufgabe verknüpft sein und z.B. Rückmeldung über Ereignisse in der Spielewelt geben. Reine Verzierungen, wie Vorspannmusik ermüden auf Dauer die Spieler. Zwar soll die Spielewelt komplex sein, die Rückmeldungen des Systems sollten dem Spieler aber stets Informationen über die Welt geben, die ihm helfen, sein inneres Modell zu vervollständigen. Dies gilt besonders für den Lernkontext.

Unberücksichtigte Faktoren

Malone untersuchte nur Computerspiele für eine Person. Bei Spielen, bei denen mehrere Personen kooperieren oder im Wettkampf miteinander stehen, kann die soziale Interaktion stark motivierend sein. Ein weiterer unberücksichtigter Faktor ist die Freiheit der Aufgabenwahl, die sich typischerweise positiv auf die Motivation auswirkt.

Bezug zu SALA

Die beiden bisher nach SALA realisierten Lernprogramme appellieren über das Element des entdeckenden Lernens vor allem an die Neugier des Lerner. Die dabei eingesetzten Aufgaben mit interaktiven Simulationen wirken aber auch herausfordernd. Die nach dem Lehrteil angebotenen

Programmieraufgaben fordern den Lerner heraus, da er in seinen persönlichen Fähigkeiten angesprochen ist. Hingegen ist der Lernstoff nicht in eine Rahmenhandlung eingebunden, so dass der Lerner nicht über die Fantasie motiviert wird. Bei zukünftigen Lernprogrammen sollte der Lehrstoff in eine Rahmenhandlung eingebettet werden.

4.3 SALA: Gliederung des Lernprogramms in Sektionen

In nach SALA gestalteten Lernprogrammen soll der Lernstoff in Sektionen gegliedert werden. Eine Sektion soll es dem Lerner ermöglichen, sich auf einen Teil des Stoffes zu konzentrieren.

Standardpfad durch das Lernprogramm

Da einem Anfänger der Überblick über den Stoff fehlt, kann er schwerlich eine sinnvolle Reihenfolge für die Bearbeitung der Sektionen bestimmen. Daher sollte der Autor des Lernprogramms einen Vorschlag für eine Reihenfolge ausarbeiten. Dazu werden unten einige Kriterien genannt. Der Lerner soll aber nicht stark gelenkt werden, wie dies in den 1960er Jahren bei den Lernprogrammen der „programmierten Instruktion“ der Fall war. Vielmehr soll er vom Standardpfad abweichen dürfen. Zur freien Navigation dienen einerseits das Inhaltsverzeichnis, andererseits Hyperlinks zwischen Sektionen, die inhaltliche Zusammenhänge aufzeigen. Somit eignet sich das Lernprogramm neben dem ersten Erlernen eines Algorithmus auch zum „Nachschlagen“ einzelner Teilthemen und zum (erneuten) Durcharbeiten auf neuen Wegen.

Sektionen des Lernprogramms

Ein Lernprogramm nach SALA sollte in folgende Sektionen aufgeteilt werden:

- **Das zu lösende Problem und seine praktische Bedeutung**

Ein Algorithmus kann erst verstanden werden, wenn das zu lösende Problem verstanden ist. Anwendungsfälle aus der Praxis veranschaulichen zum einen die Problemstellung, motivieren zum anderen auch dazu, sich mit dem Algorithmus zu beschäftigen.

- **Vergleich verschiedener Algorithmen, die das Problem lösen**

Algorithmen, die das gleiche Problem lösen unterscheiden sich teilweise stark in ihrem Zeit- und Speicherbedarf. Diese können z.B. von der Probleminstanz und vom verwendeten Hintergrundspeicher für Zwischenergebnisse abhängen. In der Praxis stehen Algorithmen oft in Softwarebibliotheken zur Verfügung (z.B. LEDA). Für die Auswahl eines Algorithmus müssen Softwareentwickler über solide Vergleichskriterien verfügen.

- **Abstrakte Sicht auf die Datenstruktur**

Die Datenstruktur gibt den Aufbau der Objekte und die Beziehungen an, die zwischen Ihnen gelten. Die Operationen (Funktionen) eines Algorithmus lassen sich oft leichter beschreiben und verstehen, wenn man auf eine abstrakte Sicht der Datenstruktur aufsetzt. So lassen sich die Operationen des Heapsort gut auf dem binären Heapbaum und der Ergebnisliste beschreiben. Die Abbildung von Heapbaum und Ergebnisliste auf ein einziges Array wird dann in der späteren Sektion „Implementierung der Datenstruktur“ beschrieben. Dies entspricht einer strukturierten top-down-Vorgehensweise, die Implementierungsdetails erst zuletzt behandelt. Diese Reihenfolge ist aber nicht zwingend, da die anderen Bestandteile der Methode SALA davon nicht abhängen. Ein Autor kann sich also auch dafür entscheiden zuerst die Implementierung der Datenstruktur und dann die abstrakte Sicht zu behandeln, wie dies auch in Lehrbüchern üblich ist.

- **Funktionale Struktur und Funktionen des Algorithmus**

Die funktionale Struktur besagt, aus welchen Funktionen der Algorithmus besteht und welche Funktionen einander aufrufen. Nach einem Überblick über die Funktionen und

ihre Schnittstellen wird jede Funktion in einem eigenen Abschnitt behandelt. Dabei wird für jede Funktion eine Aufgabe mit interaktiver Simulation angeboten.

- **Implementierung der Datenstruktur**

Wie oben beschrieben wird, wird hier die Abbildung von abstrakten Datenstrukturen auf Datentypen einer Programmiersprache behandelt. So kann z.B. ein Mehrwegebaum durch einen binären Baum realisiert werden.

- **Aufgaben zur Anwendung.**

Die Aufgaben zur Anwendung sollen das Übertragen des Pseudocodes in eine Programmiersprache behandeln. Sie können, z.B. wie im Lernprogramm zu Binomial Heap aus einer Art Multiple-Choice-Aufgabe bestehen, in der die Lerner Programmcode den Funktionen des Algorithmus zuordnen. Eine andere Möglichkeit ist, wie im Lernprogramm zu Heapsort, klassische Programmieraufgaben zu stellen und anschließend eine Musterlösung anzubieten.

Beispiel Heapsort: Sektionen des Lernprogramms

Die Gliederung des Lernprogramms zu Heapsort ist in Abb. 17 angegeben. Sie folgt der oben angegebenen Empfehlung. Da das Lernprogramm web-basiert ist, enthält es einige zusätzliche Sektionen: Über die Kurzbeschreibung des Lernprogramms kann sich ein Leser über den abgedeckten Lehr-

Lernkonzept
Sortierverfahren
Datenstruktur des Heapsort
Funktionen des Heapsort
Übersicht
Bedienung der Applets
Heapsort
Build-Heap
Heapify
Heapify-Locally
Sort
Move-Max
Zusammenfassung
Implementierung der Datenstruktur
Aufgaben zur Anwendung des Wissens
Technische Voraussetzung
Archive zum Herunterladen
Über dieses Lernprogramm
Kurzbeschreibung auf der Startseite
Copyright und Nutzungsrechte

Abb. 17: Gliederung des Lernprogramms zu Heapsort

stoff und Besonderheiten des Lernprogramms informieren. Das Lernprogramm kann seitenweise von einem Webserver abgerufen oder als Archiv heruntergeladen und dann lokal installiert werden. Die Systemvoraussetzungen informieren über die benötigte Java-Unterstützung des Webbrowsers.

4.4 Funktionale Struktur eines Algorithmus

4.4.1 Methodische Grundlage: Algorithmen als modulare Software

In der Disziplin des Software-Engineering wurden Prinzipien herausgearbeitet, wie Software modularisiert werden soll, damit die entstehenden großen Softwaresysteme verständlich, handhabbar und wartbar sind. Meines Erachtens lassen sich viele dieser Prinzipien auf die Aufbereitung von Algorithmen für das Lernen übertragen, denn in beiden Fällen muss sich ein Mensch in ein komplexes technisches System eindenken.

Ich gehe im Folgenden von Prinzipien aus, die Bertrand Meyer, ein führender Vertreter des objektorientierten Software-Engineering formuliert hat. Er

sieht den strukturierten Aufbau eines Softwaresystems aus Modulen als grundlegende Bedingung dafür, dass das System leicht erweitert werden kann und dass die entwickelten Module wieder in neuen Anwendungskontexten verwendet werden können ([Meyer 1997], S. 39). Daraus kann man folgern, dass auch Algorithmen modular dargestellt werden sollten, da eine Datenstruktur mit den dazugehörigen Algorithmen in verschiedenen Anwendungskontexten einsetzbar sein soll. Einen weiteren Grund sehe ich darin, dass ein Algorithmus leichter verstanden werden kann, wenn er in für sich verständliche Teilmodule zerlegt wird. Die modulare Zerlegung ist für die hier präsentierte

Lernmethode besonders wichtig, da die beim entdeckenden Lernen zu lösenden Aufgaben dadurch handhabbarer werden.

Meyer definiert den Begriff der Modularität zunächst informell:

„A software construction method is modular, then, if it helps designers produce software systems made of autonomous elements connected by a coherent, simple structure“ ([Meyer 1997], S. 39)

Er konkretisiert diese Definition, indem er Prinzipien aufstellt, die ein modulares System zu erfüllen hat ([Meyer 1997], Kap. 3)¹. Eine Reihe der Prinzipien lässt sich auf das Erlernen von Algorithmen anwenden. Insbesondere übertrage ich sie auf die Zerlegung eines Algorithmus in Teilfunktionen, d.h. die funktionale Strukturierung.

Modular Composability

„A method satisfies Modular Composability if it favors the production of software elements which may then be freely combined with each other to produce new systems, possibly in an environment quite different from the one in which they were initially developed.“ (ebd., S. 42)

Auf Algorithmen angewandt heißt dies, dass die Datenstruktur und der Algorithmus so allgemein formuliert sein sollen, dass sie leicht an einen spezifischen Anwendungskontext angepasst werden können. Zum Beispiel sollte bei einem Sortieralgorithmus der Typ der Schlüssel generisch sein.

Modular Understandability

„A method favors Modular Understandability if it helps produce software in which a human reader can understand each module without having to know the others, or, at worst, by having to examine only a few of the others.“ (ebd., S. 43)

Da ein Mensch nur begrenzt viele Informationen in seinem Arbeitsgedächtnis speichern und verarbeiten kann, ist es wichtig, dass ein Modul aus sich heraus verständlich ist.

1. Meyer verwendet andere Begriffe. Er unterteilt seine Prinzipien in „Criteria“, „Rules“ und „Principles“.

Wird ein Algorithmus in viele Funktionen zerlegt, so wird die Realisierung der einzelnen Funktionen kürzer und damit leichter zu verstehen, es wird aber auch nötig die Schnittstelle der Hilfsfunktionen zu kennen. Daher sollte die Bedeutung und die Aufgabe einer Hilfsfunktion intuitiv verständlich sein. Möglichst sollte die Hilfsfunktion als „Begriff“ (im Sinne der Lernpsychologie) im Langzeitgedächtnis bereitstehen ([Edelmann 1996], Kap 5).

Explicit Interfaces

„Whenever two modules A and B communicate, this must be obvious from the text of A or B or both.“ (ebd., S. 50)

Für Algorithmen bedeutet dies, dass in der Beschreibung einer Algorithmenfunktion explizit (separat vom Pseudocode der Funktion) aufgeführt wird, welche Funktionen als Hilfsfunktionen aufgerufen werden.

Information Hiding

„The designer of every module must select a subset of the module’s properties as the official information about the module, to be made available to authors of client modules.“ (ebd., S. 51)

Hier wird die wichtige Unterscheidung zwischen der Schnittstelle einer Funktion und ihrer Realisierung angesprochen. Wird die Funktion als Hilfsfunktion verwendet, so ist nur die Kenntnis der Schnittstelle nötig. Sie benennt die Aufgabe der Funktion, die formalen Parameter und, falls zutreffend, die Vorbedingung unter der sie ausgeführt werden darf. Für das Erlernen eines Algorithmus ist natürlich auch die Kenntnis der Realisierung wichtig. Sie gibt an, welche Schritte und Aufrufe von Hilfsfunktionen die Funktion ausführt, um ihre Aufgabe zu erfüllen.

Linguistic Modular Units

„Modules must correspond to syntactic units in the language used.“ (ebd., S. 53)

In SALA entspricht jedes Modul einer Funktion, die auch in einem eigenen Abschnitt dargestellt wird. In gängigen Lehrbüchern werden zwar die logisch zusammengehörigen Teile des Pseudocodes im Begleittext separat erklärt, sie werden aber nicht zu einer Funktion zusammengefasst.

Self-Documentation

„The designer of a module should strive to make all information about the module part of the module itself.“ (ebd., S. 54)

Die Präsentation eines Algorithmus soll sowohl den Pseudocode als auch einen erklärenden Begleittext zum Pseudocode enthalten. Sie sollen eng aufeinander abgestimmt und leicht zugänglich sein. Dies ist bei SALA und bei gängigen Lehrbüchern erfüllt, nicht aber bei den üblichen Algorithmen-Animationen, die zumeist keinen erklärenden Begleittext enthalten.

4.4.2 Funktionale Struktur in SALA

Aus den oben genannten Gründen erscheint es mir wichtig, einen Algorithmus im Lernprogramm modular darzustellen. Dazu wird er vom Autor in mehrere kleine Teilfunktionen zerlegt. In der Sektion „Funktionen des Algorithmus“ eines Lernprogramms werden die funktionale Struktur und die einzelnen Funktionen des Algorithmus behandelt. Die funktionale Struktur eines Algorithmus gibt an, in welche Funktionen der Algorithmus aufgeteilt ist und welche Funktionen einander aufrufen. Im ersten Abschnitt der Sektion soll ein Überblick über die funktionale Struktur und die Funktionen des Algorithmus gegeben werden. Dann wird jede Funktion in einem eigenen Abschnitt des Lernprogramms erklärt.

Übersichten über die Funktionen

In einer frühen Fassung des Lernprogramms zu Heapsort gab es nur eine vor den Funktionsseiten platzierte Übersichtsseite zur funktionalen Struktur und zu den Funktionen des Algorithmus. Sie sollte beim ersten Durcharbeiten des Lernprogramms einen Überblick über den Inhalt der folgenden Seiten mit einzelnen Funktionen bieten. Zusätzlich sollte sie ein späteres Nachschlagen der Schnittstelle der Funktionen und ein Verzweigen zu einzelnen Funktionsseiten ermöglichen. Viele Lerner ließen sich aber dazu verleiten, bereits beim ersten Durcharbeiten von dort eine Funktionsseite anzusteuern. Wenn sie danach zu den Folgeseiten weiterblättern, so erreichten sie nicht mehr die davorliegenden Funktionsseiten. Ein weiteres Problem war, dass die Übersichtsseite die gesamte Schnittstelle der Funktionen aufführte. Einige Lerner beklagten zurecht, dass sie diese Information nicht verstehen konnten. Daher wurde die Übersichtsseite gekürzt, indem die Vorbedingung der Funktionen und die Hyperlinks zu den Funktionsseiten entfernt wurden. Die alte, ausführliche

Übersichtsseite wurde als Zusammenfassungsseite hinter die letzte Funktionsseite eingefügt. Sie enthält eine Grafik zur funktionalen Struktur, sowie eine Tabelle zur Schnittstelle der Funktionen (Name, Kurzbeschreibung, Parameter, Vorbedingung).

Nutzung der funktionalen Struktur

Lehrbücher zu Algorithmen

Lehrbücher stellen gewöhnlich einen Algorithmus auf strukturierte Weise dar. Zum Beispiel wird der Heapsort-Algorithmus im Lehrbuch ([Cormen, Leiserson, Rivest 1990], Kap. 7) in sechs Funktionen aufgeteilt. Sedgewick verwendet in seinem Lehrbuch ([Sedgewick 1992], Kap. 11) nur zwei Funktionen, erklärt aber die beiden Phasen des Algorithmus (Aufbau des Heap und Sortieren) im Begleittext. Der sparsame Einsatz von Funktionen in manchen Lehrbüchern mag an dem erhöhten Aufwand einer starken Strukturierung liegen. Das Aufteilen eines Algorithmus in mehrere Funktionen erfordert zusätzliche Funktionsdeklarationen und Funktionsaufrufe im Programmtext. Dies verlängert das Programm und kann die Ausführung verlangsamen, falls der Compiler die aufgerufenen Funktionen nicht in den Maschinencode einfügt (Inlining). Es können auch Möglichkeiten zur Beschleunigung des Programms verlorengehen.

Originalveröffentlichung des Heapsort

Ein Beispiel für eine schwach strukturierte Darstellung ist die Originalveröffentlichung des Heapsort ([Floyd 1964]). Sie verwendet nur zwei heapsort-spezifische Funktionen (neben einer Tausch-Funktion) und verlängert die Aufbauphase des Heaps in den ersten Durchlauf der Sortierphase hinein. Es scheint, dass der Bedarf an einer kurzen Beschreibung in der Zeitschrift und schneller Ausführung auf einem Computer die Gründe für einen schwach strukturierten Stil waren. Während dieser Kompromiss zwischen Platzbedarf und Verständlichkeit für eine wissenschaftliche Veröffentlichung akzeptabel ist, ist es sicher nicht angemessen um Studierenden Algorithmen zu lehren.

Algorithmen-Animationen

Leider verwenden die meisten Autoren von Algorithmen-Animationen einen schwach strukturierten Stil und erstellen nur eine Animation, die die grundlegenden Schritte des Algorithmus zeigt. Zum Beispiel zeigen die meisten Animationen des Quicksort nur die Vergleiche und die Vertauschungen der Elemente, aber nicht die Hauptfunktionen Quicksort und Partition. Es ist besser für jede Algorithmenfunktion eine eigene Animation

zu erstellen und jeden Aufruf einer Hilfsfunktion als ein interessantes Ereignis zu visualisieren (siehe Einleitung, BALSAs). Einen Schritt in diese Richtung geht die Arbeit von Stern et al. ([Stern, Sondergaard, Naish 1999]). Ähnlich wie bei einem interaktiven Debugger kann der Benutzer dort den Detaillierungsgrad der Animation einstellen, indem er Zweige des hierarchisch dargestellten Programmcodes ein- und ausblendet (vgl. Kap. 3.1.1, S. 23).

Strukturierung durch den Autor

Oft wird der Autor eines Lernprogramms den Algorithmus selbst in Funktionen gliedern müssen, da gängige Lehrbücher dies nur unzureichend leisten. Dazu fasst er wiederholt elementare Schritte des Algorithmus und Funktionsaufrufe zu einer Funktion zusammen. Ein triviales Beispiel ist eine Swap-Funktion, die zwei Werte in einem Array vertauscht. Eine Funktion kann dann wieder in anderen Funktionen verwendet werden. Der Algorithmus selbst wird als eine (oberste) Funktion modelliert.

4.4.3 Reihenfolge der Funktionen

Theoretisch sollte es keine Rolle spielen in welcher Reihenfolge die Funktionen im Lernprogramm behandelt werden, da jede Funktion so dargestellt wird, dass sie aus sich heraus verständlich ist. Tatsächlich beeinflusst aber das bereits gelernte Wissen das Verhalten der Lerner beim Lösen der Aufgaben. In einer frühen Version des Lernprogramms zu Heapsort wurden die Funktionen in einer bottom-up-Reihenfolge behandelt, d.h. Hilfsfunktionen vor den sie verwendenden „höheren“ Funktionen. Zunächst lernten die Nutzer die grundlegenden Funktionen und führten jeweils bei der Aufgabe zur Funktion die Schritte der Funktionen selbst aus. Beim Erlernen der „höheren“ Funktionen trat dann das Problem auf, dass die Lerner die Schrittfolgen der Hilfsfunktion verwendeten, statt die Hilfsfunktion als Ganzes aufzurufen.

Dieser sogenannte *Übungseffekt* ist auch aus der kognitionspsychologischen Forschung bekannt. Im Experiment von Luchins sollten Probanden eine Reihe von Aufgaben lösen. Sie neigten dazu Lö-

sungsmuster, die sich bei den ersten Aufgaben bewährt hatten, unbesehen weiter zu verwenden, auch wenn es für die späteren Aufgaben bessere Lösungsmuster gab (wiedergegeben in [Edelmann 1996], S. 347-349).

„Einstellung oder Gewöhnung verursachen eine Automatisierung der Denkvorgänge, ein blindes Vorgehen gegenüber Aufgaben; man geht an die Aufgabe nicht mit der ihr angepaßten Überlegung heran, sondern verbleibt automatenhaft bei dem eingeübten Denkmuster“ (Luchins, zitiert nach [Edelmann 1996], S. 349)

Als Konsequenz habe ich das Lernprogramm verändert und die Präsentationsreihenfolge der Funktionen umgekehrt. Dadurch ist es nicht mehr möglich, vorher gelernte Lösungsmuster erneut anzuwenden. Zusätzlich habe ich in der Aufgabe zu einer Funktion nur noch die benötigten Hilfsfunktionen angeboten und nicht mehr alle zuvor behandelten. Der Übungseffekt trat dann nicht mehr auf. Daher empfehle ich die Funktionen in einer top-down-Reihenfolge zu präsentieren, d.h. Hilfsfunktionen erst nach den „höheren“ Funktionen, die sie verwenden, zu behandeln.

4.4.4 Heapsort: Struktur, Reihenfolge und Schnittstellen

Funktionale Struktur des Heapsort

Für die Aufteilung des Heapsort in Funktionen wählte ich die Darstellung in ([Cormen, Leiserson, Rivest 1990], Kap. 7) als Ausgangspunkt, da sie meiner Vorstellung einer starken Strukturierung in mehrere kleine Funktionen am ehesten entsprach. Andere von mir betrachtete Darstellungen des Heapsort ([Sedgewick 1992], [Floyd 1964], [Ottmann, Widmayer 1996]) verwenden im Programmtext nur eine schwache Strukturierung in wenige Funktionen. Um nach dem Aufbau des Heap die Sortierphase konzeptuell herauszuheben, führte ich zwei neue Funktionen ein (Sort und Move-Max). Im Gegenzug verzichtete ich auf einige Zugriffsfunktionen auf das Array (Parent, Left, Right), da entsprechende Zugriffe im Applet durch direkte Interaktion auf dem Heapbaum realisiert sind.

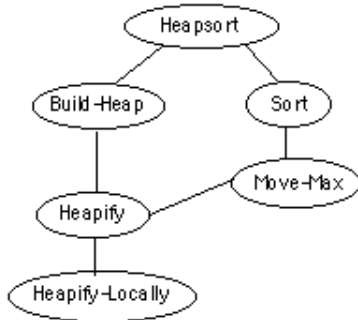


Abb. 18: Funktionale Struktur des Heapsort

Abb. 18 zeigt die dabei entstandene funktionale Struktur des Heapsort. Eine Funktion wird durch eine Ellipse dargestellt. Eine Linie zwischen zwei

Funktionen besagt, dass die weiter oben stehende Funktion die weiter unten stehende Funktion als Hilfsfunktion aufruft. Die Funktion Heapsort wird von keiner Funktion des Algorithmus aufgerufen. Nur sie wird von außen aufgerufen, und dient somit als *Einstiegsfunktion* in den Algorithmus.

Schnittstelle und Reihenfolge der Funktionen

Die Funktionen des Heapsort-Algorithmus haben die in Tabelle 4 angegebene Schnittstelle. Sie werden im Lernprogramm in der Reihenfolge behandelt, in der sie auch in der Tabelle aufgeführt sind. Die Reihenfolge wurde aus den möglichen top-down-Reihenfolgen ausgewählt, da sie dem chronologischen Ablauf des Algorithmus am besten entspricht.

Tabelle 4: Funktionen des Heapsort

Funktion (Parameter)	Kurzbeschreibung	Vorbedingung
Heapsort ()	Sortiert die Eingabedaten. Dies ist die Einstiegsfunktion für den Heapsort-Algorithmus.	Heapbaum: vollständig und voll gefüllt. Ergebnisliste: leer.
Build-Heap ()	Stellt die Heapordnung für den gesamten Heapbaum her.	Heapbaum: vollständig und voll gefüllt.
Heapify (Knoten k) p	Stellt die Heapordnung für den Baum mit Wurzel k her.	Sohnbäume unter k : heapgeordnet.
Heapify-Locally (Knoten k) p	Stellt die Heapordnung für den Knoten k her. Rückgabe: neue Position des k -Schlüssels oder Null wenn nicht bewegt.	- (keine)
Sort ()	Verschiebt alle Schlüssel des Heapbaums in die Ergebnisliste.	Heapbaum: vollständig, voll gefüllt und heapgeordnet. Ergebnisliste: leer.
Move-Max ()	Verschiebt den maximalen Schlüssel des Heapbaums in die Ergebnisliste.	Heapbaum: vollständig und heapgeordnet. Ergebnisliste: aufsteigend geordnet.

Die Funktionen des Algorithmus greifen über die folgenden Basisfunktionen auf das zu sortierende Array zu (Tabelle 5):

Tabelle 5: Funktionen zum Datenzugriff

Funktion (Parameter)	Kurzbeschreibung	Vorbedingung
Compare (Knoten a,b)	Vergleicht die Schlüssel der Knoten a und b . Im Applet werden die Schlüssel sichtbar und zwischen den Knoten erscheint ein Relationszeichen.	- (keine)
Swap (Knoten a,b)	Vertauscht die Schlüssel der Knoten a und b .	- (keine)
Move (Knoten a,b)	Bewegt den Schlüssel von Knoten a nach Knoten b .	- (keine)

4.5 Erlernen einer Funktion

Jede Funktion des Algorithmus wird in einem eigenen Abschnitt (z. B. einer Webseite) des Lernprogramms behandelt. Die Gliederung der Darstellung der Funktion orientiert sich am Lernprinzip des entdeckenden Lernens und an den Prinzipien einer modularen Darstellung des Algorithmus.

4.5.1 Lernphasen beim entdeckenden Lernen

Die Funktion wird so präsentiert, dass der Lerner sie entdeckend erlernen kann. Dabei wird nur ein Teil des Wissens über erklärende Komponenten vermittelt (vgl. Kap. 4.2.2, S. 36). Den verbleibenden Teil soll der Lerner über eigene Erfahrungen mit der Lernumgebung gewinnen. Er erhält eine Aufgabe, die er mit einer interaktiven Simulation des Algorithmus bearbeitet. Anschließend kann er seine Lösung mit einer Musterlösung vergleichen. Das Lernen findet also in drei Phasen statt:

1) Wissensvermittlung und Problemstellung

Zur Orientierung wird angegeben, wo die Funktion in der funktionalen Struktur steht. Die Aufgabe der neuen Funktion wird informell und formal beschrieben. Dazu werden Parameter, Vorbedingung und Nachbedingung der Funktion benannt. Die von der Funktion genutzten Hilfsfunktionen werden mit Name, Zweck und Vorbedingung angegeben.

2) Entdeckungen in der Probierphase

In der Probierphase soll der Lerner den Pseudocode der Funktion selbst herausfinden. Der Pseudocode löst die Aufgabe der Funktion für den allgemeinen Fall (allgemeine Eingabedaten). Um den Lerner zur allgemeinen Lösung hinzuführen, erhält er die Aufgabe Schrittfolgen (spezielle Lösungen) für eine Reihe von Eingabedaten zu finden. Der Lerner kann nun experimentieren um eine korrekte Abfolge zu finden in der die Hilfsfunktionen auf die Datenobjekte angewendet werden. Für diese Aufgabe wird eine interaktive Simulation der Algorithmenfunktion verwendet.

3) Vergleich mit der Musterlösung

In der letzten Phase wird die Musterlösung als Pseudocode angegeben und erklärt. Der Lerner kann sich für jede der Eingabedaten eine Animati-

on der Lösung anzeigen lassen. Zusätzlich steht ihm jetzt die Simulation im Modus des Standardverfahrens zur Verfügung, bei dem nur die Schritte der Musterlösung ausführbar sind.

Sollte es sich um eine sehr einfache Funktion handeln, kann der Autor direkt die Musterlösung erklären und auf Probierphase, Simulation und Animation verzichten.

4.5.2 Modulare Darstellung der Funktion

Die Darstellung einer Algorithmenfunktion im Lernprogramm orientiert sich an den oben dargelegten Prinzipien der modularen Softwareentwicklung (vgl. Kap. 4.4.1, S. 40). Sie gliedert sich in die Einordnung, die Schnittstelle und die Realisierung der Funktion (Abb. 19).

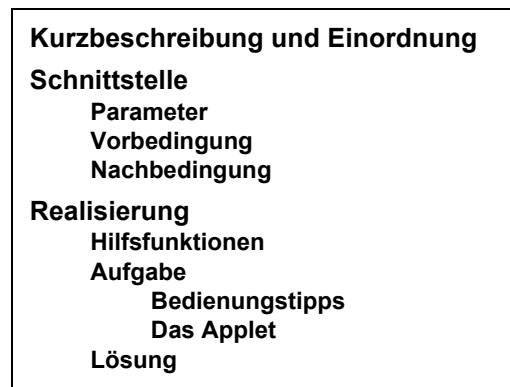


Abb. 19: Gliederung der Funktionenseite

Im Einzelnen enthält die Funktionenseite folgende Elemente:

- **Kurzbeschreibung und Einordnung**

Die Aufgabe der Funktion wird mit einem kurzen Text informell beschrieben. Es handelt sich um den gleichen Text, der in der Übersicht der Funktionen und bei der Beschreibung der Hilfsfunktionen verwendet wird.

Eine Grafik zeigt die funktionale Struktur des Algorithmus, wobei die aktuelle Funktion hervorgehoben ist. Dadurch kann sich der Lerner über das Verhältnis der Funktion zu den anderen Funktionen des Algorithmus informieren. Die Grafik dient auch als Bestätigung, dass der Lerner die von ihm gewünschte Funktionsseite angesteuert hat.

- **Schnittstelle**

Die Schnittstelle beschreibt, was ein Nutzer der Funktion wissen muss. Sie umfasst die Parameter, Vorbedingung und die Nachbedingung. Entsprechend dem Modularitätsprinzip „Information Hiding“ werden hier nur soviel Informationen zur Funktion wie für die Nutzung nötig weitergegeben.

- **Parameter**

Parameter geben an, auf welchem Teil der Datenstruktur die Funktion arbeiten soll. Explizite Parameter werden beim Funktionsaufruf übergeben und steuern damit die Funktion direkt. Unabhängig von diesen kennt die Funktion schon bestimmte Teile der Datenstruktur und begrenzt ihre Arbeit auf diese Teile. Diese werden implizite Parameter genannt, da sie nicht beim Aufruf übergeben werden, aber doch das Verhalten der Funktion beeinflussen.

- **Vorbedingung**

Die Vorbedingung beschreibt, wann eine Funktion angewendet werden kann. Sie ist als Bedingung formuliert, die die Daten erfüllen müssen.

Später werden die Lerner als Aufgabe einen Satz Eingabedaten erhalten, den sie verarbeiten müssen. Die Eingabedaten sind so ausgewählt, dass sie die Vorbedingung erfüllen. Zur Veranschaulichung werden diese Eingabedaten hier grafisch dargestellt, da sie Beispiele für Daten sind, die die Vorbedingung erfüllen.

- **Nachbedingung**

Die Nachbedingung beschreibt, was die Funktion bewirkt. Sie formuliert welche logischen Bedingungen auf den Daten gelten, nachdem die Funktion aufgerufen wurde. Auch hierzu wird eine Grafik der Daten als Beispiel angegeben. Bei der Aufgabe haben die Lerner das Ziel, diesen Zustand zu erreichen.

- **Realisierung**

Die Realisierung behandelt, wie die Funktion intern arbeitet. Sie umfasst die Hilfsfunktionen sowie die Aufgabe und Musterlösung zum Pseudocode der Funktion. Dabei wird die Realisierung auf einem relativ abstrakten Niveau beschrieben, da die Implementierung der Datenstruktur in einer eigenen Sektion behandelt wird. Wo es für das Verständnis

nötig ist, werden aber auch Hinweise zur Implementierung der Schritte in einer Programmiersprache gegeben.

- **Hilfsfunktionen**

Die aktuelle Funktion ruft andere Funktionen als Hilfsfunktionen auf, um ihre Aufgabe zu erledigen. Es ist daher wichtig, die Schnittstellen der Hilfsfunktionen zu kennen. Deshalb wird in einer Tabelle für jede Hilfsfunktion Name, Parameter, Kurzbeschreibung und Vorbedingung angegeben. Über Hyperlinks kann man zu den Abschnitten der Hilfsfunktionen gelangen, auf denen sie detaillierter beschrieben werden.

- **Aufgabe**

Mit der Aufgabe beginnt die Proberphase des entdeckenden Lernens. Nun ist der Lerner aufgefordert aktiv zu werden. Er soll herausfinden, welche Schritte die Funktion ausführen muss.

- **Bedienungstipps**

Es bietet sich an, an dieser Stelle Tipps zur Bedienung der interaktiven Aufgabe zu geben. Sie sollten insbesondere die Bedienelemente erklären, die speziell für die aktuelle Aufgabe benötigt werden. Daneben kann man allgemeine Bedienhinweise einstreuen, um die Lerner darüber zu informieren. Stehen die allgemeinen Hinweise nur auf einer Hilfeseite, so werden sie von vielen Benutzern nicht gelesen oder nicht erinnert.

- **Die interaktive Simulation**

An dieser Stelle erscheint die interaktive Simulation (vgl. Kap. 4.6, S. 47). Sie zeigt eine grafische Darstellung der Eingabedaten. Nun soll der Lerner Hilfsfunktionen aufrufen, indem er Datenobjekte als Parameter anklickt und Funktionen startet, indem er auf die Buttons mit den Funktionsnamen drückt. Er soll die Datenstruktur in den Zustand der Nachbedingung bringen. In einem webbasierten Lernprogramm kann sie z.B. als Java-Applet realisiert werden.

- **Lösung**

An dieser Stelle beginnt die letzte Phase des entdeckenden Lernens. Der Lerner kann jetzt die Musterlösung (den Pseudocode der Funktion) einsehen. Wurde die interaktive Aufgabe mit der SALABIM-Bibliothek erstellt, so muss der Lerner nur das Applet in den Modus Animation umschalten. Der Pseudocode der Funktion wird im Applet angezeigt und

der Lerner kann eine Animation der Funktion betrachten, wobei im Applet zu jedem Schritt ein kurzer Erklärungstext erscheint. Weitere Erklärungen zur Lösung sind unter dem Applet als statischer Text angegeben.

4.5.3 Heapsort: Build-Heap Funktion

Die folgenden Bildschirmabzüge zeigen die Seite zu Build-Heap aus dem Lernprogramm zu Heapsort (Abb. 20, S. 48 und Abb. 21, S. 49). Die Build-Heap-Funktion des Heapsort hat die Aufgabe, die Heapordnung im gesamten Heap herzustellen.

4.6 Interaktive Simulation einer Algorithmen-Funktion

4.6.1 Die interaktive Simulation

In der interaktiven Simulation werden die Daten grafisch dargestellt. Der Lerner startet Funktionen, indem er Datenobjekte als Parameter mit der Maus auswählt und dann mit Funktionsnamen beschriftete Schaltflächen drückt. Er muss dabei im Modus der Simulation nicht die Schritte des Standardverfahrens einhalten, sondern kann experimentieren und Fehler machen. Die Simulation führt die Funktionsaufrufe aus und gibt visuelle und textuelle Rückmeldungen über Fortschritt und Fehler. Wird die Simulation im Modus der Übung betrieben, so sind nur die Schritte der Musterlösung zugelassen.

4.6.2 Heapsort: Aufgabe zur Funktion Build-Heap

In Abb. 22 ist die Ausgangslage der Simulation zur Funktion Build-Heap zu sehen. Das Ziel der Funktion (und damit die Aufgabe für den Lerner) ist es, die Heapordnung für alle Knoten herzustellen. Aufgrund der Definition der Heapordnung sind die Blätter des Baumes immer heapgeordnet. Als Hilfsfunktion steht Heapify zur Verfügung. Heapify darf aber an einem Knoten nur gestartet werden, wenn die Teilbäume unterhalb des gewählten Knotens komplett heapgeordnet sind. Knoten, bei denen die Heapordnung gilt, werden grün gefüllt dargestellt. Dadurch kann der Lerner erkennen, wie nah er dem Ziel schon gekommen ist, und welche Knoten noch bearbeitet werden müssen. Klickt der Lerner auf den Knoten A oder B und dann auf

Heapify, so wird der Teilbaum an diesem Knoten heapgeordnet, da dort die Vorbedingung der Funktion Heapify erfüllt ist. Der Knoten wird dann ausgefüllt dargestellt. Würde der Lerner zu diesem Zeitpunkt einen der anderen Knoten auswählen und versuchen Heapify zu starten, so würde die Fehlermeldung „Falsch! Die Kindbäume müssen heapgeordnet sein!“ ausgegeben.

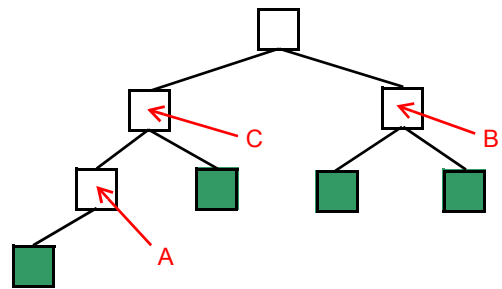


Abb. 22: Simulation des Build-Heap

Im Modus der Übung lässt die Simulation nur die Schritte der Musterlösung zu. In der obigen Situation kann dann nur Knoten A gewählt werden, da das Standardverfahren die Knoten von unten nach oben und innerhalb einer Ebene von rechts nach links durchgeht. Im Gegensatz zur reinen Simulation kann der Lerner nun als Hilfestellung auch Tipps zum nächsten Schritt abrufen.

Sind alle Knoten heapgeordnet, so muss der Lerner dies erkennen und die Schaltfläche „Haken“ anklicken (Abb. 21). Er erhält dann die Meldung, dass die Aufgabe erfolgreich gelöst wurde.

Die Funktion Build-Heap

```

graph TD
    Heapsort --> Build-Heap
    Heapsort --> Sort
    Build-Heap --> Heapify
    Heapify --> Heapify-Locally
    Heapify --> Move-Max
    Move-Max --> Sort
    
```

- [Kurzbeschreibung](#)
- [Schnittstelle](#)
 - [Parameter](#)
 - [Vorbereitung](#)
 - [Nachbereitung](#)
- [Realisierung](#)
 - [Hilfsfunktionen](#)
 - [Aufgabe](#)
 - ◻ [Bedienungstipps](#)
 - ◻ [Das Applet](#)
 - [Lösung](#)

Kurzbeschreibung

Stellt die Heapordnung für den gesamten Heapbaum her.

Schnittstelle

Parameter

Explizite Parameter: Keine.
Implizite Parameter: Heapbaum

Vorbereitung

Heapbaum: vollständig und voll gefüllt.

Knoten	Bedeutung
■ (black)	Verletzt möglicherweise die Heapbedingung
■ (green)	Erfüllt die Heapbedingung.

Nachbereitung

Heapbaum: vollständig gefüllt und heapgeordnet.

Knoten	Bedeutung
■ (green)	Erfüllt die Heapbedingung.

Abb. 20: Funktion Build-Heap im Lernprogramm zu Heapsort
(oberer Teil der Seite)

Realisierung

Hilfsfunktionen

Funktion (Parameter)	Kurzbeschreibung	Vorbedingung
Heapify (Knoten k)	Stellt die Heapordnung für den Baum mit Wurzel k her.	Sohnbäume unter k: heapgeordnet

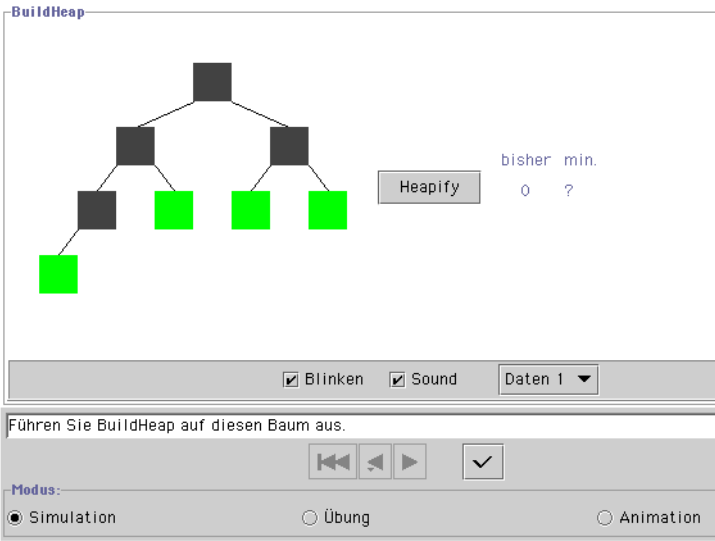
Aufgabe

Stellen Sie im ganzen Heapbaum die Heapordnung her. Die Blattknoten sind per Definition heapgeordnet und daher grün markiert. Die anderen Knoten wurden vom Algorithmus noch nicht untersucht und sind daher eventuell nicht heapgeordnet (grau). Sobald ein Knoten von Ihnen heapgeordnet wird, wird er grün markiert.

Bedienungstipps

- Es kann einige Minuten dauern, bis das Applet erscheint. [Wenn dann immer noch kein Applet erscheint.](#)
- [Bedienung der Applets](#)

Das Applet



Führen Sie BuildHeap auf diesen Baum aus.

Modus:

Simulation Übung Animation

Lösung

Die **Blätter sind heapgeordnet**, da sie keine Sohnknoten haben, die gegen die [Heapbedingung](#) verstoßen könnten.

Man berechnet die **Position des letzten inneren Knotens**, indem man vom letzten Knoten des Heapbaums zum Vaterknoten geht. Ein Knoten ist dann ein innerer Knoten, wenn er kein Blattknoten ist.

Erlaubte Reihenfolgen: Man kann Heapify an einem Knoten nur aufrufen, wenn die beiden Bäume unter dem Knoten schon heapgeordnet sind. Es gibt mehrere Reihenfolgen von Heapify-Aufrufen, die diese Bedingung einhalten. Der Modus Simulation erlaubt alle gültigen Reihenfolgen. Z.B. können sie bereits im ersten Schritt zwischen zwei Knoten wählen: den Knoten ganz links in der 3. Ebene von oben oder den rechten Knoten in der 2. Ebene.

Standardverfahren notwendig: Die Funktionen eines Algorithmus müssen die Vorbedingung ihrer Hilfsfunktionen beachten. Im Allgemeinen gibt es mehrere erlaubte Reihenfolgen der Aufrufe der Hilfsfunktionen. Damit ein Algorithmus als Programm ausgeführt werden kann, muss eine Reihenfolge fest gewählt werden. Diese Reihenfolge heißt hier *Standardverfahren*. Man wählt eine Reihenfolge, die effizient (d. h. mit wenigen Schritten auskommt) und leicht zu programmieren ist.

Das Standardverfahren beginnt beim letzten inneren Knoten und geht die Ebenen von unten nach oben durch. Innerhalb einer Ebene geht es von rechts nach links. Dadurch ist sichergestellt, dass die Sohnbäume eines Knotens schon heapgeordnet sind, wenn Heapify aufgerufen wird. Aufgrund der [Implementierung](#) der Datenstruktur lässt sich dies als eine einfache Wiederholungsanweisung (mit Integer-Decrement) programmieren. Sie können über den Modus "Animation" das Standardverfahren studieren.

Lernkonzept: In einem typischen Lehrbuch erfahren Sie nur das Standardverfahren. In diesem Lernprogramm sollen Sie auch die Hintergründe erfahren und selbst herausfinden, wie es zu dem Standardverfahren kam. Deshalb sollen Sie sich Gedanken über Auswahl und Reihenfolge der Schritte einer Funktion machen. (Weiteres zum [Lernkonzept](#)).

Modi des Applet: Bei diesem Applet können Sie besonders gut die Unterschiede zwischen den Modi Simulation, Üben und Animation erkennen. Probieren Sie die verschiedenen Modi doch einmal aus! Versuchen Sie jetzt die Aufgabe im **Modus "Üben"** zu lösen. Das Applet lässt Sie die Schritte dann nur in der Reihenfolge des Standardverfahrens ausführen. (Weiteres zur [Bedienung der Applets](#)).

Abb. 21: Funktion Build-Heap im Lernprogramm zu Heapsort (unterer Teil der Seite)

4.7 Simulation und Animation mit SALABIM-Applets

Die Applets zu den Funktionen des Lernprogramms Heapsort wurden mit der SALABIM-Bibliothek realisiert (vgl. Kap. 6, S. 59). Sie bieten dadurch dem Lerner eine einheitliche Benutzungsoberfläche und eine Reihe fortgeschrittener Möglichkeiten mit dem Algorithmus zu interagieren. In Abb. 23, S. 51 sind die Bedienelemente eines Applets am Beispiel der Heapify-Locally-Funktion beschrieben. Abb. 24, S. 51 zeigt die je nach Modus verfügbaren Bedienelemente zur Steuerung des Ablaufs der Schritte der Funktion.

So kann der Lerner ein Applet in den Modi *Simulation*, *Übung* oder *Animation* betreiben. Die Modi *Simulation* und *Übung* wurden oben bereits beschrieben. Im Modus der *Animation* wird der Pseudocode der Funktion angezeigt und die gerade aktive Zeile hervorgehoben. Der Lerner kann in Einzelschritten vor und zurück durch die Animation schalten oder sie als kontinuierlichen Film betrachten. Dabei werden die ausgeführten Schritte textuell erklärt und die Veränderung der Daten gegebenenfalls mit Bewegungen visualisiert.

In allen Modi steht ein mehrstufiges *Undo/Redo* zur Verfügung. Die ausgeführten Schritte werden in einer Pull-down-Liste mit Namen verzeichnet, so dass die Lerner direkt einen der vorherigen Zustände ansteuern können.

Neben der Schaltfläche einer Funktion wird mitgezählt, wie oft die Funktion bisher aufgerufen wurde. Am Ende der Aufgabe wird daneben eingeblendet, wieviel Aufrufe die Standardlösung benötigt hätte. So kann der Lerner die Effizienz seiner Lösung mit der des Standardverfahrens vergleichen.

Im Allgemeinen stehen zu einer Aufgabe mehrere Sätze Eingabedaten zur Verfügung, von denen der Lerner einen auswählt. Weiterhin kann er bestimmen, ob Fehlermeldungen und Erfolgsmeldungen vom Applet mit Tönen begleitet werden und ob die Veränderung der Daten durch Bewegungen visualisiert wird.

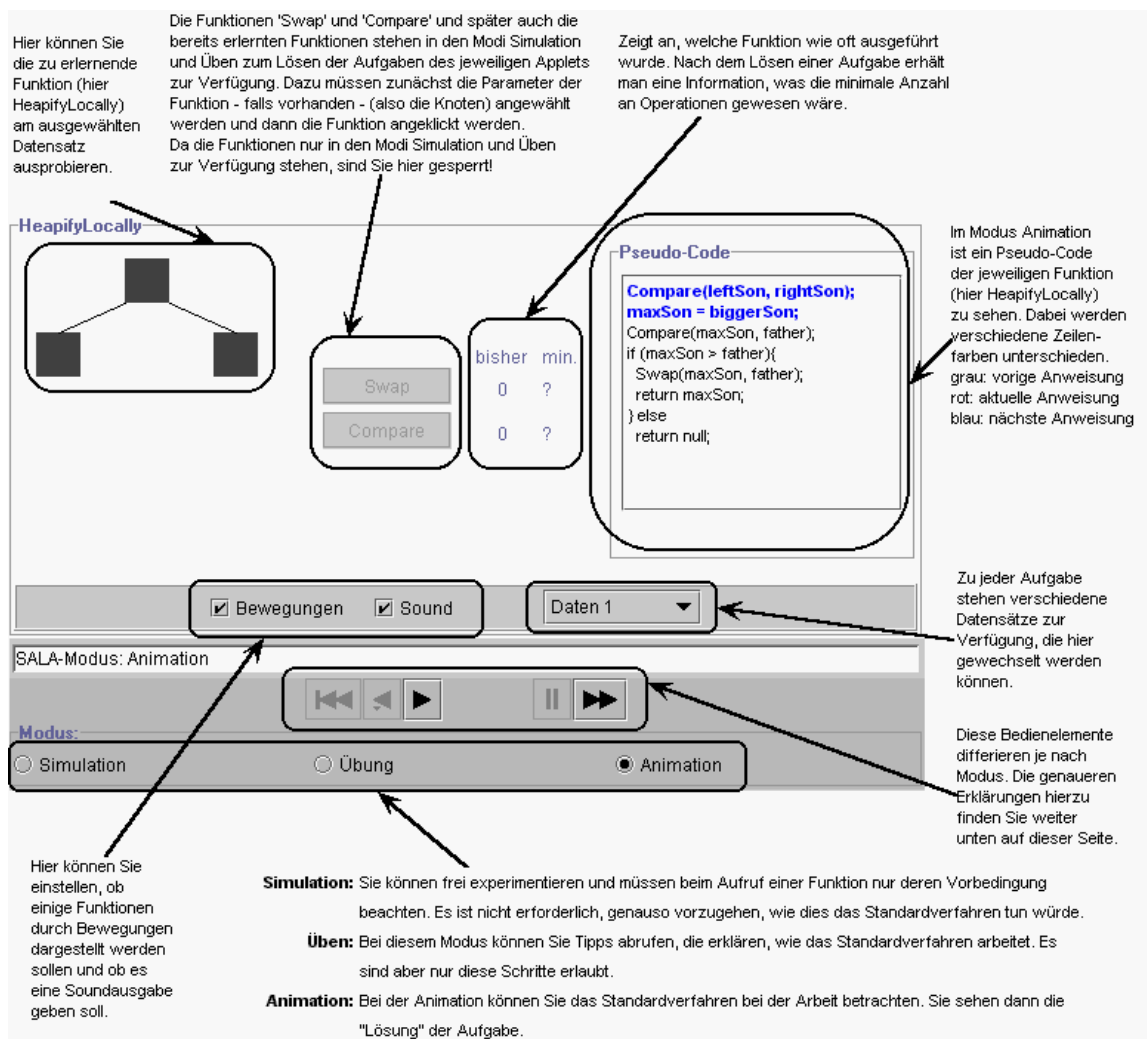


Abb. 23: Bedienelemente eines SALABIM-Applets

Für alle Modi

Bei Wechsel des Modus oder des Datensatzes beginnt man neu mit der Aufgabe.
In allen drei Modi stehen Ihnen die folgenden Bedienelemente zur Verfügung:

- Alle bisher ausgeführten Schritte werden rückgängig gemacht.
- Klickt man kurz, wird der zuletzt ausgeführte Schritt rückgängig gemacht. Hält man die Maustaste gedrückt, öffnet sich ein Pop-up-Menü, das sämtliche bisher ausgeführten Schritte enthält. Wählt man hieraus einen aus, werden alle Schritte bis zu diesem rückgängig gemacht.

Der Modus Simulation

- Der zuletzt rückgängig gemachte Schritt wird wieder ausgeführt.
- Wenn eine Aufgabe gelöst wurde, muss der Abschluss durch Klicken auf dieses Symbol bestätigt werden.

Der Modus Üben

- Man erhält einen Hinweis, welche Funktion vom Algorithmus als nächstes ausgeführt werden muss.
- Wenn eine Aufgabe gelöst wurde, muss der Abschluss durch Klicken auf dieses Symbol bestätigt werden.

Der Modus Animation

- Der nächste Schritt der Animation wird ausgeführt.
- Unterbricht die Animation.
- Startet die Animation und führt sie komplett aus.

Abb. 24: Ablaufsteuerung eines SALABIM-Applets

5 Gestaltungsregeln für interaktive Visualisierungen von Algorithmen

In der Methode SALA wird nicht ausgesagt, wie die Datenstrukturen visualisiert werden sollen. Im Folgenden stelle ich Gestaltungsregeln vor, die Autoren helfen sollen, lernförderliche interaktive Visualisierungen zu entwickeln. Die Gestaltungsregeln lassen sich gut bei der Entwicklung von Lernprogrammen nach SALA anwenden, sind aber nicht auf einen Einsatz in diesem Kontext beschränkt.

In diesen Regeln fasse ich meine persönlichen Einschätzungen zusammen, die ich in den letzten Jahren meiner Arbeit mit interaktiven Visualisierungen gewonnen habe. Wann immer ich eine Visualisierung betrachtete und mit ihr interagierte, reflektierte ich über ihre Gestaltung und über Punkte, die mir verbesserungswürdig schienen. In der Literatur zur Software-Ergonomie, Didaktik,

Wahrnehmungspsychologie und Algorithmen-Animation fand ich immer wieder hilfreiche Hinweise. Bei der Gestaltung des Lernprogramms zu Heapsort habe ich versucht, die Gestaltungsregeln soweit wie möglich zu berücksichtigen.

Ich unterscheide im Folgenden drei Arten von interaktiven Visualisierungen. Sie werden in der Reihenfolge der steigenden inneren Beteiligung des Lerners am Lernprozess besprochen. *Algorithmen-Animationen* werden vom Lerner betrachtet und analysiert. *Algorithmen-Übungen* fordern vom Lerner, zuvor gesehene Schritte eines Algorithmus selbst auszuführen. *Algorithmen-Simulationen* erlauben es dem Lerner, sich durch Experimentieren an die Arbeitsweise eines Algorithmus heranzutasten.

5.1 Algorithmen-Animationen

Bei Animationen ist der Ablauf des Algorithmus fest vorgegeben. Vom Lerner wird im Wesentlichen erwartet, dass er die Schritte des Algorithmus in einer Visualisierung betrachtet. Es wird nicht erwartet, dass er Schritte voraussagt oder selbst Teile des Algorithmus entwickelt. Von den drei Arten der interaktiven Visualisierung ist Animation die mit Abstand verbreitetste. Einen Überblick über Forschungsarbeiten zur Algorithmen-Animation geben [SDBP 1998] und die Einleitung dieser Arbeit (vgl. Kap. 1.2, S. 12).

5.1.1 Gestaltungsregeln für Algorithmen-Animationen

Aus didaktischer und software-ergonomischer Sicht möchte ich nun einige Gestaltungsregeln für Animationen angeben. Peter Gloor hat eine ähnliche Liste mit Schwerpunkt auf Software-ergono-

mischen Kriterien entwickelt [Gloor 1997]. Die Animation sollte möglichst viele der Regeln erfüllen.

- **Funktionale Struktur**

Der Algorithmus wird in Funktionen aufgeteilt. Jede nichttriviale Funktion wird in einer eigenen Animation erklärt. Bereits gelernte Funktionen können in einer Animation als Bausteine (Funktionsaufrufe) verwendet werden und werden nur noch als ein Schritt angezeigt (vgl. Kap. 4.4, S. 40).

- **Lehrtext**

Mit der Animation ist ein Lehrtext verbunden, der in der Art eines Lehrbuchs mit Text und Bildern den Algorithmus erklärt. Alternativ kann dies auch durch einen vorherigen Vortrag vermittelt werden.

- **Anzeige des Pseudocodes**

Zusätzlich zu den Datenstrukturen wird der Pseudocode des Algorithmus angezeigt. Die aktuelle Befehlszeile wird hervorgehoben. So kann der Lerner den Pseudocode und die Aktionen auf den Datenstrukturen miteinander in Beziehung setzen.

- **Erklärung der Schritte**

Die Schritte des Algorithmus werden textuell oder per Sprachausgabe kommentiert, damit der Lerner versteht, was der Algorithmus gerade tut.

- **Interessante Eingabedaten**

Viele Algorithmen zeigen bei bestimmten Eingabedaten ein besonderes Verhalten. Der Sortieralgorithmus Quicksort zeigt z. B. bei vorsortierten Eingabedaten ein anderes Laufzeitverhalten als bei zufällig gemischten Eingabedaten. Dem Lerner werden die verschiedenen Sätze von Eingabedaten zur Auswahl gestellt.

- **Kleine Datenmenge**

Die Animation arbeitet auf einer kleinen Zahl von Datenobjekten. Die Anzahl ist gerade so gewählt, dass das Wesentliche am Algorithmus noch zu erkennen ist. Große Datenmengen erschweren die Wahrnehmung und das Verständnis, da das menschliche Arbeitsgedächtnis nur begrenzt aufnahmefähig ist ([Anderson 1996], S 169-177).

- **Wenige Schritte**

Analog werden nur so viele Wiederholungen von Schleifendurchläufen gezeigt, wie für Wahrnehmung und Verständnis nötig sind. Andernfalls wird der Lerner schnell gelangweilt und seine Aufmerksamkeitskapazität unnötig verbraucht.

- **Flexible Ablaufsteuerung**

Die Animation des Algorithmus kann ähnlich wie bei einem Videorecorder gesteuert werden: Einzelschritte vor und zurück, Abspielen, Pause, Stopp, Sprung an Anfang und Ende. Aus didaktischer Sicht ist es besonders wichtig in Einzelschritten vor und zurück schalten zu können (vgl. [Hundhausen 1999], Kap. 7.1.3). Gerade Anfänger, die den Algorithmus neu lernen, sind mit einem durchlaufenden Film meist überfordert. Da normale Programme nicht „rückwärts“ laufen können, stellt dies besondere Anforderungen an die Software-Architektur der interaktiven Visualisierung.

- **Interessante Ereignisse**

Ein Schritt umfasst dabei ein „interessantes Ereignis“ des Algorithmus. Das kann z. B. eine Aktion des Algorithmus oder das Erreichen wichtiger Zwischenergebnisse sein. Im Gegensatz zu diesem Prinzip machen manche SW-Visualisierungs-Systeme grafische

Operationen zu Schritten. Das Konzept der „interesting events“ wurde von Marc Brown entwickelt [Brown 1987].

- **Fokussierung der Aufmerksamkeit**

Der Blick des Lerners wird auf die Teile der Darstellung gelenkt, die sich als Nächstes verändern werden. Dazu kann der Autor diese Objekte z.B. blinken oder ihre Größe ändern lassen. So wird verhindert, dass der Betrachter wichtige Schritte nur als diffuse Bewegung wahrnimmt.

- **Weiche Übergänge**

Veränderungen auf den Datenstrukturen werden möglichst mit weichen, fließenden Bewegungen gezeigt. Werden z.B. zwei Elemente eines Array vertauscht, so können sie sich jeweils auf einem Halbkreisbogen bewegen und so „Plätze tauschen“. Dies erleichtert die Wahrnehmung des Vorgangs. Damit beim Lerner nicht der Eindruck entsteht, in einem Computer würden sich die Daten „fließend bewegen“, sollte im Begleittext darauf hingewiesen werden, wie die Veränderung der Daten implementiert wird. Vertauschung wird z. B. über „plötzliche“ Kopiervorgänge im Dreieckstausch realisiert. Es war eines der wesentlichen Ziele bei der Entwicklung des SW-Visualisierungssystems Tango, Animationen mit weichen Übergängen zu ermöglichen [Stasko 1999].

5.1.2 Beispiele für Algorithmen-Animationen

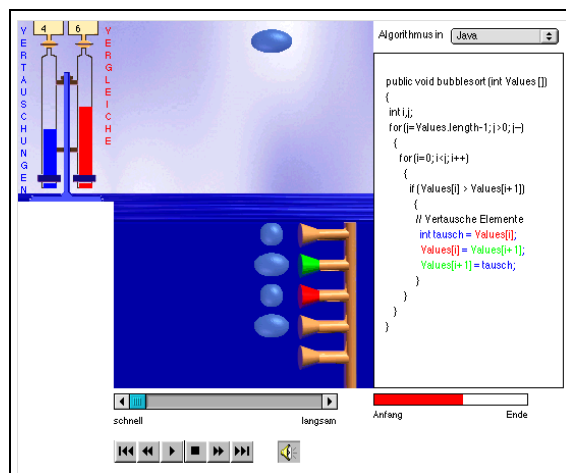


Abb. 25: Lernen mit Animationen: Bubblesort
Bildschirmabzug von [BDGKW 1998]

Die Autoren der Animation des Bubblesort-Algorithmus (Abb. 25) wollten eine ästhetische Animation entwickeln, welche die Metapher

„Luftblasen“ (Bubbles) aufgreift [BDGKW 1998]. Sie haben dabei die folgenden Prinzipien umgesetzt:

- Weiche Übergänge
- Flexible Ablaufsteuerung
- Anzeige des Pseudocodes

- Kleine Datenmenge
- Lehrtext

Ein weiteres Beispiel ist die Animation des Heapsort von Stern et al.. Bis auf die Fokussierung verwirklicht sie alle geforderten Prinzipien (vgl. Kap. 3.1.1, S. 23).

5.2 Algorithmen-Übungen

Nachdem ein Algorithmus durch einen Lehrtext und das Betrachten einer Animation gelernt worden ist, sollte das Wissen durch Übung überprüft und vertieft werden. Bei einer *Übung* soll der Lerner mehrfach den nächsten Schritt des Algorithmus voraussagen (vgl. Kap. 4.5.1, S. 45). Es gibt dabei nur einen „richtigen“ nächsten Schritt, und zwar den, den der Standardalgorithmus als Nächstes ausführen würde. Dazu interagiert der Lerner mit einer Visualisierung der Datenstruktur über eine grafische Benutzungsoberfläche. Es gibt nur wenige Projekte, die interaktive Visualisierungen entwickelt haben, die diese Lernform unterstützen (so z.B. [Block 1999a] und die beiden in Kap. 3.1.1 genannten Projekte).

5.2.1 Gestaltungsregeln für Algorithmen-Übungen

Die Gestaltungsregeln für Animationen, die ich oben angegeben habe, gelten bis auf den Punkt „Anzeige des Pseudocode“ auch für das Lernen mit Übungen. Zusätzlich sollten noch die folgenden Regeln beachtet werden:

- **Kein Pseudocode**

Da der Lerner den nächsten Schritt eigenständig voraussagen soll, wird kein Pseudocode angezeigt. Insbesondere der Charakter der Überprüfung des Wissens würde sonst unterlaufen.

- **GUI nutzen**

Die Möglichkeiten einer grafischen Benutzungsoberfläche (GUI) werden, soweit möglich und sinnvoll, für die Interaktion genutzt. Insbesondere kann der Lerner Datenobjekte in der Grafik der Visualisierung direkt auswählen (z.B. mit der Maus anklicken). Ein Schritt eines Algorithmus besteht oft im Aufruf einer Funktion. Die Funktionen können z.B. durch Schaltflächen (Buttons) dargestellt werden. Der Lerner ruft eine Funktion auf, in-

dem er erst die Objekte (aktuelle Parameter) in der Grafik und dann die Schaltfläche der Funktion anklickt (vgl. Kap. 4.6, S. 47).

- **Bedienung vereinfachen**

Die Bedienung der Aufgabe ist möglichst einfach gestaltet. Es ist einfach erkennbar, welche Funktionen angeboten sind und welche Objekte anwählbar sind. Der Lerner muss also Objekte und Funktionen nur auswählen. Bedienungsfehler sollten als solche gemeldet werden. Die Bedienung sollte beschrieben sein (z.B. als Online-Hilfe oder im Begleittext).

- **Lösung anspruchsvoll**

Es sind möglichst alle Objekte der Visualisierung anwählbar. Dadurch wird es dem Lerner erschwert, den nächsten Schritt zu erraten.

- **Fehlermeldung und Tipp**

Wählt der Lerner eine falsche Funktion oder falsche Parameter, so kann es sinnvoll sein zu begründen, warum dieser Funktionsaufruf jetzt keinen Sinn macht. Meist wird es reichen zu melden, dass der Algorithmus einen anderen Schritt ausführen würde. Es sollte aber ein Tipp abrufbar sein, der einen Hinweis gibt, was zu tun ist. Falls dieser Hinweis nicht ausreicht, sollte es möglich sein, den nächsten Schritt automatisch ausführen zu lassen.

5.2.2 Beispiele für Algorithmen-Übungen

Ein Beispiel für *Lernen durch Üben* ist das Java-Applet aus der Lernsoftware zum Binomial Heap Algorithmus (Abb. 26) [Block 1999b]. In ihm sollen zwei Listen von Binomialbäumen (Heap 1 und Heap 2) zu einer Ergebnisliste verschmolzen werden. Dies wird ähnlich dem schriftlichen Addieren

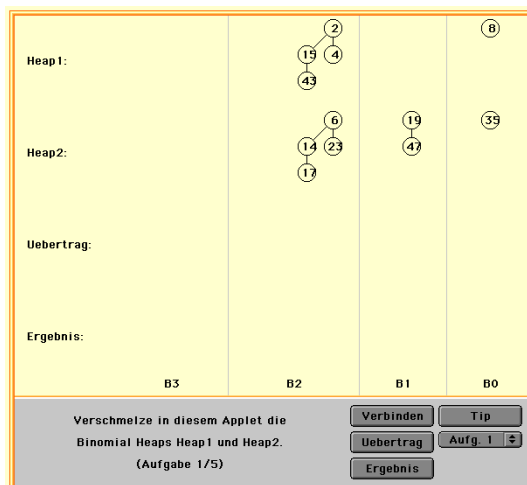


Abb. 26: Lernen durch Übung:
Verschmelzen-Funktion
Bildschirmabzug von [Block 1999b]

durchgeführt. Der Lerner muss genau die Schritte des Standardalgorithmus einhalten, kann aber über die Schaltfläche „Tipp“ eine Hilfestellung abrufen. Die oben genannten Kriterien sind im Wesentlichen erfüllt.

Weitere Beispiele finden sich in Kap. 3.1.1, S. 23.

5.3 Algorithmen-Simulationen

Algorithmen-Simulationen erlauben ein freies Experimentieren mit Funktionen und Datenobjekten. Der Lerner kann nun andere Schritte als die des Standardverfahrens ausführen, um zum Ziel zu gelangen. Im Abschnitt „Interaktive Simulation einer Algorithmen-Funktion“, Seite 47, wurde eine Form der Algorithmen-Simulation beschrieben.

5.3.1 Gestaltungsregeln für Algorithmen-Simulationen

Die Benutzungsoberfläche der Algorithmen-Simulation ähnelt der Oberfläche der Algorithmen-Übung. Zusätzlich zu den dort genannten Prinzipien sollten die folgenden Gestaltungsregeln beachtet werden:

- **Inhaltbezogene Fehlermeldungen**

Versucht der Lerner eine Funktion aufzurufen, deren Vorbedingung nicht erfüllt ist, so muss die interaktive Visualisierung dies mit

einer Fehlermeldung abweisen. Die Fehlermeldung sollte den Grund in Bezug auf die Vorbedingung angeben.

- **Fortschrittsanzeiger**

Der Lerner kann erkennen, wie nah er dem Ziel gekommen ist. Die Visualisierung kann dazu z.B. anzeigen, welcher Anteil der Daten die Nachbedingung bereits erfüllt.

- **Undo**

Es kann durchaus sein, dass der Lerner Schritte tätigt, die zwar die Vorbedingungen der Hilfsfunktionen nicht verletzen, aber nicht zum Ziel führen. Über einen Undo-Mechanismus kann der Lerner einzelne Schritte zurücknehmen. Durch die Java-Bibliothek SALABIM wird ein mehrstufiges Undo/Redo unterstützt (vgl. Kap. 4.6, S. 47).

5.3.2 Beispiele für Algorithmen-Simulationen

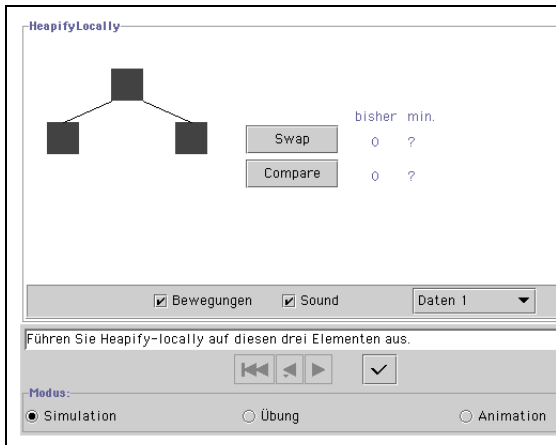


Abb. 27: Simulation zu Heapify-Locally aus Heapsort
Bildschirmabzug von [Faltin 2001b]

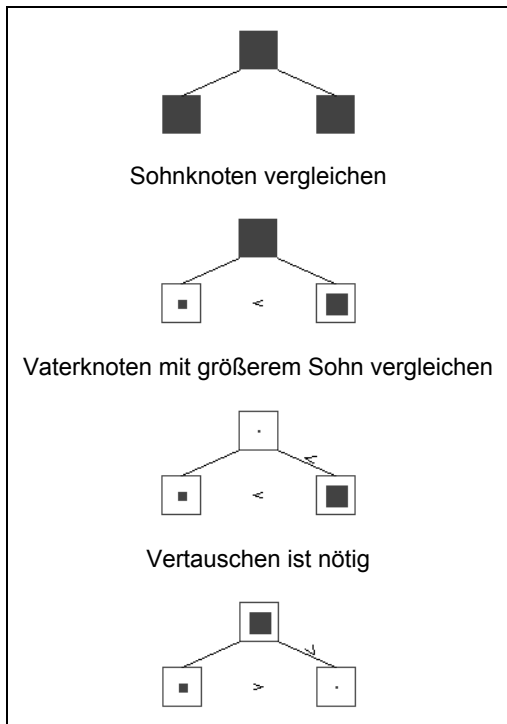


Abb. 28: Standardverfahren Heapify-Locally
Arrangiert nach [Faltin 2001b]

Als Beispiel zeigt Abbildung 27 die Simulation der Heapify-Locally-Funktion aus dem Lernprogramm zu Heapsort [Faltin 2001b]. Die Aufgabe ist, die Heapeigenschaft im dargestellten Heapbaum herzustellen. Der Schlüssel im Vaterknoten muss am Ende größer oder gleich den Schlüsseln in den Sohnknoten sein. Bei Beginn der Aufgabe ist der Inhalt der Knoten noch unbekannt, daher werden diese dunkelgrau dargestellt. Vergleicht man mit „Compare“ zwei Knoten, so werden die Schlüssel sichtbar und ein Relationszeichen (<,=,>)

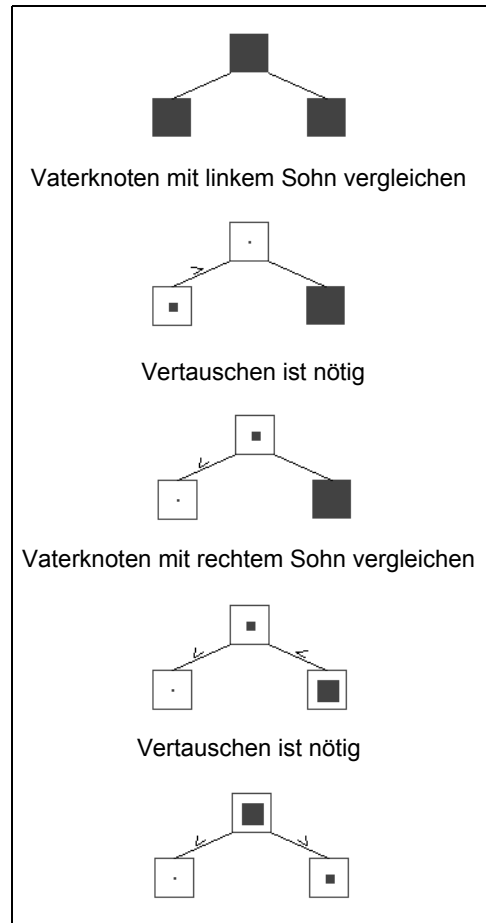


Abb. 29: Alternatives Verfahren Heapify-Locally
Arrangiert nach [Faltin 2001b]

zwischen den Knoten angezeigt. Über die Schaltfläche „Swap“ können die Schlüssel zweier Knoten vertauscht werden.

Abb. 28 zeigt die Schritte des Standardverfahrens. Zunächst werden die beiden Sohnknoten miteinander verglichen. Der größere Sohnknoten wird mit dem Vaterknoten verglichen. Wenn nötig werden die Schlüssel von Vaterknoten und Sohnknoten vertauscht.

Die Simulation erlaubt es, beliebige Schrittfolgen auszuprobieren. Dadurch kann sich der Nutzer in einem Prozess des entdeckenden Lernens an die optimale Lösung herantasten (vgl. Kap. 4.5, S. 45).

Abb. 29 zeigt eine andere mögliche Schrittfolge, um das Ziel zu erreichen. Der Vaterknoten wird mit dem linken Sohn verglichen. Da der Sohn den größeren Schlüssel enthält, wird er nach oben getauscht. Ebenso wird der Vaterknoten mit dem rechten Sohnknoten verglichen und wieder der Schlüssel des Sohnknotens nach oben getauscht. Diese Schrittfolge ist etwas aufwändiger, führt aber auch zur gewünschten Heapordnung.

Die Simulation zu Heapify-Locally erfüllt die Gestaltungsregeln:

- Die Hilfsfunktionen „Compare“ und „Swap“ haben keine Vorbedingung, so dass hier keine Fehlermeldungen ausgegeben werden müssen.
- Der Fortschritt wird erkennbar, da immer mehr Schlüssel und Relationszeichen zu sehen sind.
- Ein mehrstufiges Undo/Redo ist verfügbar.

Im Lernprogramm zu Heapsort kann für jede der sechs Funktionen eine Aufgabe im Modus der Simulation bearbeitet werden (vgl. Kap. 4.6, S. 47).

Weitere Beispiele für Algorithmen-Simulationen finden sich im Lernprogramm zu Binomial Heap von Karsten Block (vgl. Kap. 7.1.1, S. 69). Meines Wissens sind außerhalb unserer Arbeitsgruppe keine Algorithmen-Simulationen entwickelt worden.

6 Software-Bibliothek SALABIM

6.1 Einführung

In einem nach SALA gestalteten Lernprogramm wird eine Algorithmenfunktion mithilfe interaktiver Visualisierungen erlernt. Zunächst arbeitet der Lerner mit einer Simulation der Funktion, kann dann die Musterlösung in einer Animation betrachten und diese schließlich mit einer Übung nachvollziehen (vgl. Kap. 4.5, S. 45). Für den Autor eines Lernprogramms ergibt sich damit das Problem, drei Varianten der Visualisierung erstellen zu müssen. Um diesen Aufwand zu minimieren wurde die Software-Bibliothek SALABIM („SALA-Bibliothek zur Interaktiven Manipulation“) entwickelt. Der Autor programmiert eine allgemeine Visualisierung der Algorithmenfunktion, von der automatisch die Varianten Simulation, Übung und Animation abgeleitet werden. Weitere Anforderungen an die Bibliothek ergaben sich aus den in Kap. 5 genannten Gestaltungsregeln: flexible Ablaufsteuerung, Lernunterstützung und weiche Übergänge. Um die Lernprogramme im weltweiten Internet anbieten zu können, ist die Bibliothek webbasiert und multilingual angelegt.

SALABIM wurde von mir entworfen und von Jan Schormann im Rahmen einer HiWi-Tätigkeit programmiert. Die Bibliothek wurde erfolgreich bei der Entwicklung des Lernprogramms zu Heapsort eingesetzt, so dass alle Funktionen des Algorithmus mit einer interaktiven Aufgabe in den Modi Simulation, Übung und Animation bearbeitet werden können (vgl. die Beschreibung zum Lernprogramm Heapsort in Kap. 4).

SALABIM steht unter der Lizenz „GNU General Public Library License“ kostenlos zur Verfügung und ist zusammen mit dem Lernprogramm zu Heapsort herunterladbar [Faltin 2001b]. Eine detaillierte Dokumentation der Schnittstelle und der Klassen der SALABIM-Bibliothek ist zusammen mit dem Java-Quellcode in der Distribution enthalten.

6.2 Anforderungen an die Bibliothek

Im Einzelnen ergaben sich damit die folgenden Anforderungen an die Bibliothek SALABIM:

Gemeinsamer Rahmen für alle Modi

Der Autor braucht nur eine allgemeine Visualisierung der Algorithmenfunktion bereitzustellen. Die Bibliothek ermöglicht es dann dem Lerner, die Visualisierung in den Modi Simulation, Übung und Animation zu betreiben (vgl. Kap. 4.5, S. 45). Dazu verwaltet sie interne Datenstrukturen und stellt eine grafische Benutzerschnittstelle zur Verfügung. Der Autor kann für jede Instanz der Visualisierung den Modus festlegen oder es dem Lerner überlassen, den Modus zu wählen.

Flexible Ablaufsteuerung

Der Lerner kann den Ablauf der Visualisierung flexibel steuern. In den Modi Simulation und Übung unterstützt die Bibliothek ein mehrstufiges Undo/

Redo (vgl. Kap. 5.2 und 5.3). Im Modus Animation kann man in Einzelschritten vor und zurück durch die Animation gehen oder diese als einen kontinuierlichen Film ablaufen lassen (vgl. Kap. 5.1). Ein Schritt der Visualisierung entspricht dabei einem Funktionsaufruf.

Lernunterstützung

Die Visualisierung versorgt den Lerner mit Informationen zum ablaufenden Algorithmus (vgl. Kap. 5). In den Modi Simulation und Übung kann der Lerner prüfen lassen, ob die Aufgabe schon gelöst ist. Eine Statistik zeigt die bisher benötigte Anzahl der Aufrufe der Hilfsfunktionen. Im Modus Üben ist als Hilfe ein Tipp zum nächsten Schritt abrufbar. Bei Animationen wird der Pseudocode dargestellt und die Schritte des Algorithmus werden textuell kommentiert.

Weiche Übergänge

Die Bibliothek ermöglicht es der Visualisierung, grafische Objekte kontinuierlich auf der Zeichenfläche zu bewegen. Es können mehrere Objekte gleichzeitig auf frei gewählten Bahnen bewegt werden. Damit lässt sich ein weicher Übergang zwischen zwei Zuständen der Daten darstellen (vgl. Kap. 5.1).

Webbasiert

Die Bibliothek ist Java-basiert (Java-Swing) und erlaubt es Applets zu erstellen, die in herkömmlichen Webbrowsern laufen.

Multilingual

Natürlichsprachliche Texte der Visualisierung (wie z.B. Fehlermeldungen) werden vom Autor in nach Sprachen getrennten Ressourcen-Dateien abgelegt. Die Sprache des Applets wird über ein Applet-Tag gesteuert. Dadurch wird es erleichtert, die Visualisierung in andere Sprachen zu übersetzen.

Die aktuelle Version der SALABIM-Bibliothek (November 2001) erfüllt bis auf die Statistik der Funktionsaufrufe und die Darstellung des Pseudocodes die soeben genannten Anforderungen. Statistik und Pseudocode wurden bereits innerhalb des Lernprogrammes zu Heapsort realisiert und sollen in zukünftige Versionen von SALABIM integriert werden.

6.3 Beispielapplet Simple

Gemeinsam mit SALABIM wurde das Applet Simple entwickelt (Abb. 30).

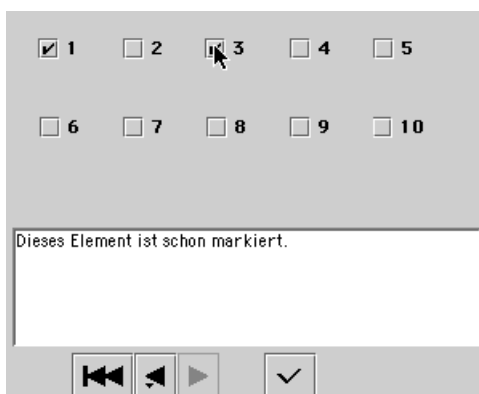


Abb. 30: Simple - Beispielapplet zu SALABIM

Es soll zukünftigen Autoren als Beispiel dienen, wie SALABIM genutzt werden kann, um eine interaktive Visualisierung zu entwickeln. Es ist be-

wusst sehr einfach gehalten, um den Autoren die Einarbeitung zu erleichtern und nutzt nur einige wesentliche Funktionen von SALABIM.

Im Kapitel 4.7 wurde die Benutzungsoberfläche der SALABIM-Applets beschrieben. Umfangreichere Beispiele für die Verwendung von SALABIM sind das unten beschriebene Applet Puzzle (Kap. 6.8, S. 67) und die Applets aus dem Lernprogramm zu Heapsort (Kap. 4.5.3, S. 47).

Das Applet Simple enthält zehn Kontrollkästchen (Checkboxes), die von 1 bis 10 durchnummeriert sind. Die Aufgabe des Lernalters ist, alle zehn Kästchen anzukreuzen. Dies darf in einer beliebigen Reihenfolge geschehen, nur darf ein bereits angekreuztes Kästchen nicht wieder deselektiert werden. Das im Applet realisierte Standardverfahren selektiert nacheinander die Kästchen 1 bis 10.

6.4 Java-Swing einbinden

Die Bibliothek SALABIM basiert auf der Java-Version 1.1 mit der Erweiterung Java-Swing für plattformunabhängige grafische Benutzungsschnittstellen. In der aktuellen Sprachversion Java 2 ist Swing bereits integriert.

Leider unterstützen die meisten der heute gängigen Browser Java-Swing nicht direkt, so dass Swing als Bibliothek über die Datei swingall.jar (2,4 MB) eingebunden werden muss. In der aktuellen Version des Lernprogramms zu Heapsort wird swingall.jar zusammen mit der Datei des Lernpro-

gramms (heapsort.jar, 290 KB) und der SALABIM-Bibliothek (salabim.jar, 26 KB) in einem Verzeichnis des Webservers bereitgehalten. Dadurch muss der Nutzer an seinem Computersystem die Java-Konfiguration nicht verändern, die große swing-all.jar-Datei wird aber bei jeder Sitzung neu übertragen. Bei einem schnellen Internetanschluss, wie er an Universitäten üblich ist, ist dies unproblematisch. Bei einem Modemanschluss empfiehlt es sich aber, das Lernprogramm zusammen mit der Swing-Bibliothek einmalig herunterzuladen und dann lokal ohne Internetzugang zu nutzen.

Es ist auch möglich, die Swing-Bibliothek lokal auf einem Computersystem zu speichern und dann über den CLASSPATH dauerhaft einzubinden. Dann können alle Swing-basierten Applets schnell über das Internet geladen werden. Leider gibt es aber in HTML/Java keine Möglichkeit automatisch zwischen lokalen und entfernten Swing-Bibliotheken umzuschalten.

6.5 SALABIM als Framework

SALABIM ist keine geschlossene Bibliothek, sondern ein Framework von Java-Klassen. Sie bieten einen Rahmen, in den der Autor seine Klassen über Vererbung einbinden kann (vergleiche z.B. [Meyer 1997], S. 72 zum Begriff des Framework). Die Klassen des Autors realisieren und visualisieren den Algorithmus in enger Zusammenarbeit mit den SALABIM-Klassen.

Das Zusammenwirken dieser Teile wird in den folgenden Abschnitten zu Kontrollfluss und Klassenstruktur eines SALABIM-Applets beschrieben. Weitere Informationen finden sich in der Online-Dokumentation der Bibliothek.

6.6 Kontrollfluss im Applet

Abbildung 32 zeigt den Kontrollfluss im Applet während der Initialisierungsphase und Abbildung 31 während der Interaktionsschleife. Die von der Bibliothek geleisteten Arbeitsschritte sind durch einen grauen Hintergrund hervorgehoben. Sie bestehen vor allem in der Realisierung der flexiblen Ablaufsteuerung und in der Verwaltung der Meldungen zur Lernunterstützung. Die Bibliothek unterstützt die Autoren auch bei der Realisierung weicher Übergänge und multilingualer Meldungen, ohne dass dies in den Abbildungen gesondert hervorgehoben ist.

6.6.1 Initialisierung

Das Applet wird initialisiert (Abb. 32), wenn es startet oder wenn der Lerner einen neuen Datensatz oder neuen Modus wählt. Der gewählte Datensatz mit den Eingabedaten des Algorithmus wird in den internen Daten des Algorithmus gespeichert. Anschließend werden diese Anfangsdaten visualisiert und Buttons für den Aufruf der Hilfsfunktionen bereitgestellt. In den Modi Ani-

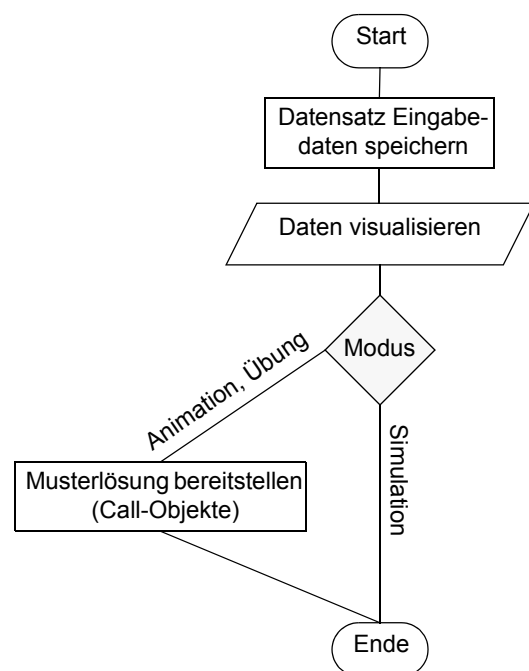


Abb. 32: Initialisierung des Applets in SALABIM

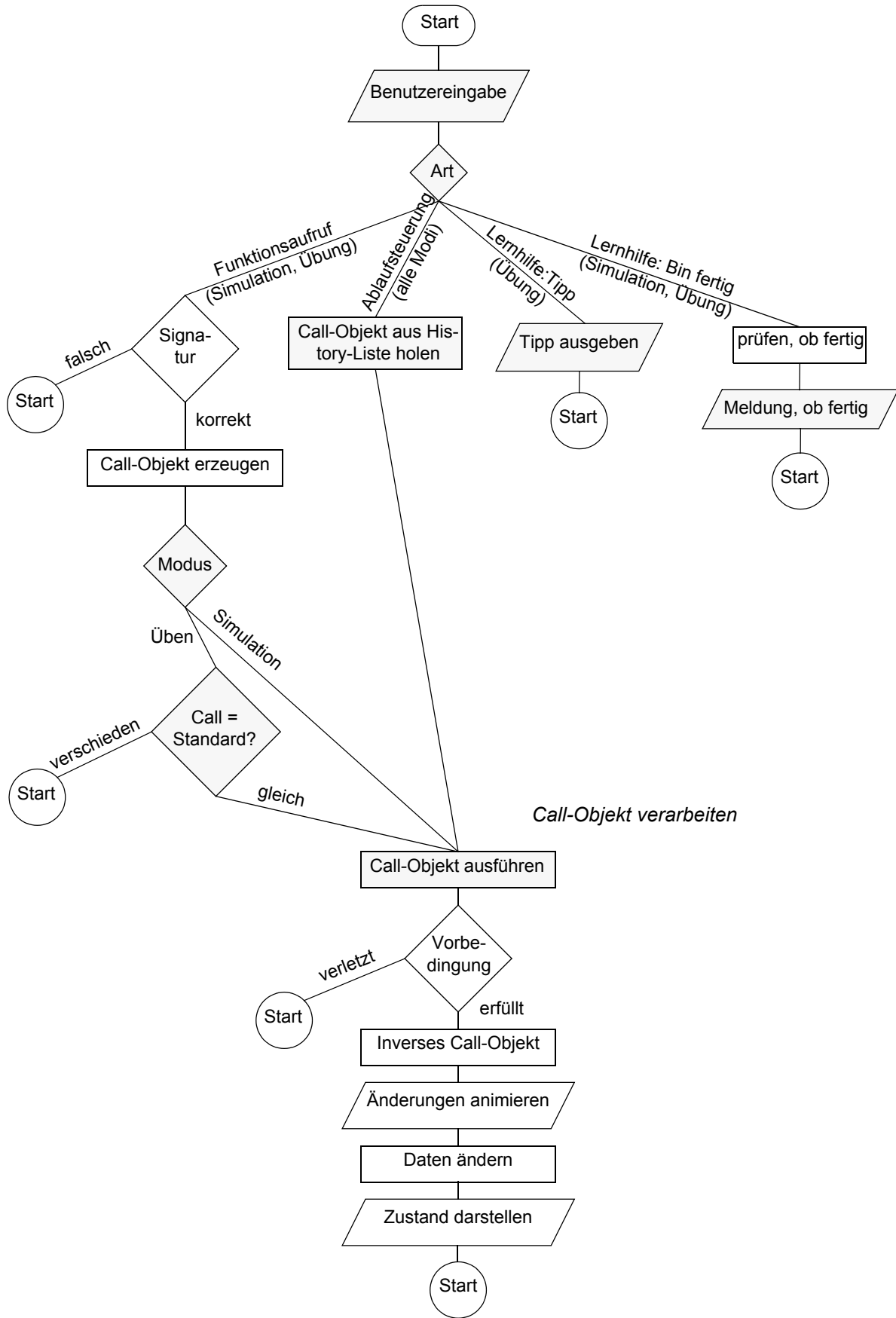


Abb. 31: Interaktionsschleife im SALABIM-Applet

mation und Übung benötigt das Applet in der späteren Interaktionsschleife Informationen über die Schritte der Musterlösung. Daher wird nun eine Referenzimplementierung der Algorithmusfunktion gestartet und die dort anfallenden Schritte in einer Liste gespeichert. Die Schritte werden noch nicht visualisiert und auch die interne Datenstruktur zur Visualisierung bleibt noch unverändert. Jeder Schritt entspricht einem späteren Aufruf einer Hilfsfunktion und wird mit den benötigten Parametern in einem sogenannten Call-Objekt gespeichert. Für jede Hilfsfunktion muss der Autor vorab eine entsprechende Call-Klasse anlegen.

6.6.2 Interaktionsschleife

Nach der Initialisierung startet die Interaktionsschleife. Das Applet wartet auf eine Eingabe des Lernalers und verzweigt je nach Art der Eingabe. Bei Änderung des Datensatzes oder des Modus wird das Applet neu initialisiert, da damit eine neue Aufgabe beginnt. Mit den anderen Eingabearten bearbeitet der Lerner die bestehende Aufgabe, indem er Funktionen aufruft, den Ablauf steuert und Lernhilfen abrufen (Abb. 31).

Funktionsaufruf

In den Modi Simulation und Übung kann der Lerner selbst Hilfsfunktionen aufrufen (vgl. Kap. 4.7, S. 50). Er wählt dazu zunächst die Parameter aus, indem er Datenobjekte in der Visualisierung anklickt. Anschließend klickt er auf den Button einer Hilfsfunktion. Das Applet prüft, ob Typ und Anzahl der Parameter (d.h. die Signatur) zur Schnittstelle der Hilfsfunktion passen. Ist die Signatur falsch, so gibt das Applet eine Fehlermeldung aus und beginnt die Interaktionsschleife von neuem. Andernfalls erzeugt es ein Call-Objekt von der zur Hilfsfunktion gehörenden Call-Klasse und trägt dort die aktuellen Parameter ein. Im Modus Üben muss sich der Lerner genau an die Schritte des Standardverfahrens (d.h. der Musterlösung) halten. Die entsprechenden Call-Objekte der Musterlösung wurden in der Initialisierungsphase bereitgestellt. Das Call-Objekt des Lernalers wird mit dem Call-Objekt der Musterlösung verglichen. Sind sie verschieden, so hat er versucht einen falschen Schritt auszuführen und dies wird mit einer Fehlermeldung abgewiesen. Andernfalls wird der Schritt weiter bearbeitet. Im Modus der Simulation wird der Schritt des Lernalers nicht mit einer Musterlösung verglichen.

Call-Objekt verarbeiten

Der gewünschte Schritt wird nun weiter geprüft und bearbeitet. Dazu ruft das Applet am Call-Objekt die Methode „perform“ auf. Da das Call-Objekt eine Instanz der zur Hilfsfunktion gehörenden Call-Klasse ist, ist diese Methode genau auf die Erfordernisse der Hilfsfunktion abgestimmt. Zunächst wird geprüft, ob die Vorbedingung der Hilfsfunktion erfüllt ist, d.h. ob die Hilfsfunktion jetzt mit diesen Parametern aufgerufen werden darf. Ist die Vorbedingung verletzt, wird der Schritt mit einer Fehlermeldung abgewiesen. Intern wird dies durch ein sogenanntes FailCall-Objekt signalisiert. Ist die Vorbedingung erfüllt, so kann der Schritt ausgeführt werden. Um ein Undo zu ermöglichen, wird ein inverses Call-Objekt erzeugt. Es enthält alle Daten, um den Schritt rückgängig zu machen und besitzt eine perform-Methode, die die Wirkung des Schrittes aufhebt. Das inverse Call-Objekt wird später als Rückgabewert zurückgegeben und in der History-Liste verwaltet. Bei Vorwärtsschritten wird die Veränderung der Daten möglichst in einer weichen Bewegung animiert. Bei Rückwärtsschritten (Undo) sollte der Übergang nicht animiert werden, um den vorhergehenden Zustand schneller anzeigen zu können. Schließlich werden die internen Daten des Algorithmus dem Schritt entsprechend verändert und der neue Zustand der Daten dargestellt. Das Applet kehrt dann wieder zum Anfang der Interaktionsschleife zurück.

Ablaufsteuerung

Die Bibliothek ermöglicht es dem Lerner, den Ablauf der Visualisierung flexibel zu steuern. In den Modi Simulation und Übung kann der Lerner bereits getätigte Schritte zurücknehmen (Undo) und zurückgenommene Schritte wieder ausführen (Redo). Dazu werden die inversen Call-Objekte der bereits ausgeführten Schritte und die Call-Objekte zurückgenommener Schritte in einer History-Liste gespeichert.

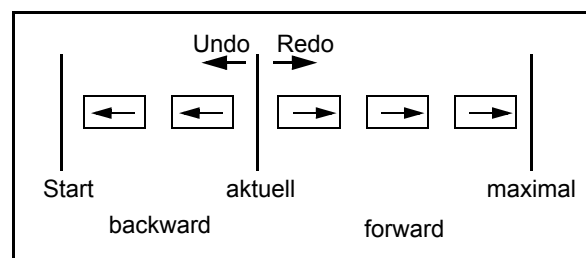


Abb. 33: History-Liste der Call-Objekte

Abbildung 33 zeigt eine History-Liste in der fünf Call-Objekte gespeichert sind. Der Lerner hat fünf Schritte ausgeführt und anschließend über Undo

drei Schritte zurückgenommen, so dass das System jetzt im Zustand „aktuell“ ist. Über Undo kann der Lerner Richtung Startzustand gehen und über Redo zurückgenommene Schritte wieder ausführen (bis zum Zustand „maximal“).

Falls Vorwärts- oder Rückwärtsschritte in der Liste verfügbar sind, werden die Buttons „Vor“ und „Zurück“ angezeigt (vgl. Kap. 4.7, S. 50). Klickt der Lerner auf einen der Buttons wird das dazugehörige Call-Objekt aus der Liste entnommen und verarbeitet (siehe oben „Call-Objekt verarbeiten“). Dabei wird ein inverses Call-Objekt erzeugt, mit dem dieser Schritt umgekehrt werden kann. Es wird in die History-Liste eingefügt. Ruft der Lerner eine Funktion auf, so werden für Redo gespeicherte Call-Objekte gelöscht, wenn der nächste gespeicherte Schritt nicht dem eingegebenen Schritt entspricht.

Die History-Liste wird im Modus Animation genutzt, um die Musterlösung darzustellen. Ein Einzelschritt „Vorwärts“ in der Animation ist als eine Art „Redo“, ein Schritt „Rückwärts“ als eine Art „Undo“ realisiert. Die History-Liste wird in der Initialisierung mit Call-Objekten der Musterlösung gefüllt. Anschließend wird das erste Call-Objekt der History-Liste zum aktuellen Objekt gemacht, das bereitsteht, beim Klick auf „Vorwärts“ ausgeführt zu werden.

Der Lerner kann den Algorithmus auch als kontinuierlichen Film ablaufen lassen. Der Scheduler erzeugt dann einen neuen Java-Thread, der nacheinander alle Call-Objekte der forward-Liste ausführt.

Zwischen zwei Schritten fügt der Scheduler eine Pause ein, so dass der Lerner das Gesehene verarbeiten kann. Der ursprüngliche Thread überwacht weiterhin die Benutzereingaben, so dass der Lerner den Film stoppen und pausieren lassen kann.

Die Ablaufsteuerung wird fast komplett von der Bibliothek realisiert. Der Autor muss nur Call-Klassen bereitstellen und inverse Call-Objekte berechnen.

Lernhilfe: Tipp

Der Tipp steht nur im Modus Übung zur Verfügung. Er ist ein textueller Hinweis zum nächsten Schritt der Musterlösung. Die Hinweistexte werden während der Initialisierungsphase zusammengestellt und in den Call-Objekten gespeichert. Klickt der Lerner auf den Button Tipp „?“, so wird der Hinweistext des nächsten auszuführenden Call-Objekts aus der History-Liste ausgelesen und als Meldungstext ausgegeben.

Lernhilfe: Bin fertig

In den Modi Simulation und Üben soll der Lerner selbst erkennen, wenn er die Algorithmenfunktion fertig ausgeführt hat. Wenn er meint, dass er das Ziel erreicht hat, so klickt er auf den Button „Fertig“ (Haken). In der Methode „finish“ wird geprüft, ob die Aufgabe tatsächlich erledigt ist und eine textuelle Meldung zusammengestellt. Sie wird durch die Bibliothek ausgegeben.

6.7 Klassen eines SALABIM-Applets

6.7.1 Klassendiagramm

Im Kapitel 4.7 wurde die Benutzungsoberfläche der SALABIM-Applets beschrieben. Dieser Abschnitt behandelt die Realisierung von SALABIM als objektorientierte Software-Bibliothek.

Notation des Klassendiagramms

Abbildung 34 zeigt die Klassen eines SALABIM-Applets am Beispiel des Applets Simple. Das Diagramm folgt der Notation von Gamma et al. [GHJV 1994], da diese gut geeignet ist, implementierungsnahe Klassenstrukturen zu dokumentieren. Klassen sind als Rechtecke gezeichnet und mit dem Namen der Klasse beschriftet. Vererbung wird durch

eine Linie mit Dreieck dargestellt. Die höher liegende Klasse ist dann die Oberklasse, die darunterliegende die Unterklasse. Relationen zwischen Klassen sind durch eine Linie mit Pfeil dargestellt. Gehören zu einem Element der Ausgangsklasse mehrere Elemente der Zielklasse, so ist an der Pfeilspitze ein dicker Punkt gezeichnet.

Die Klassen der SALABIM-Bibliothek sind grau gefüllt mit dünnem Rand. Sie sind zum Teil Unterklassen von Swing-Klassen (weiß gefüllt, dünner Rand). Die spezifischen Klassen des Simple-Applets (SimpleModel, NumberCall) sind weiß gefüllt und mit dickem Rand dargestellt. Im Diagramm sind die Klassen in drei Ebenen angeordnet: Swing oben, SALABIM in der Mitte und Simple unten.

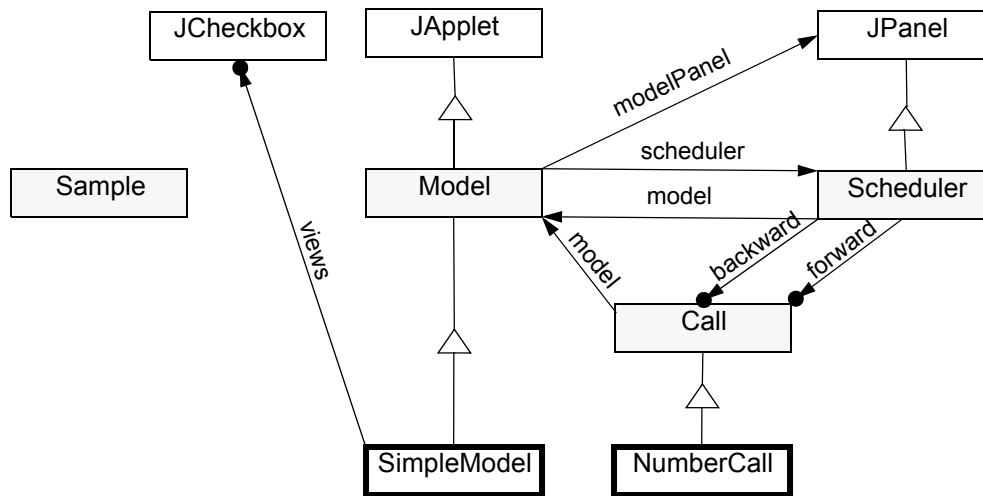


Abb. 34: Klassen eines SALABIM-Applets

6.7.2 Die Klassen

GUI des Applets

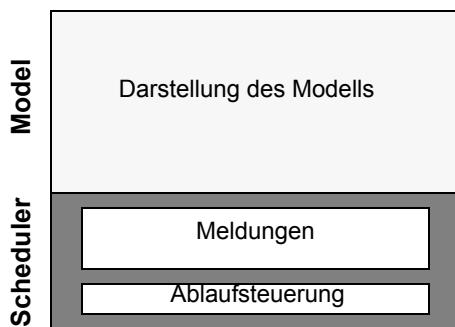


Abb. 35: GUI eines SALABIM-Applets

Die Klasse Model und ihre Unterklassen verwalten den algorithmenspezifischen Teil eines SALABIM-Applets, der in der Bildschirmdarstellung im oberen größeren Bereich dargestellt wird (Abb. 35). Der Meldungsbereich und die Buttons der Ablaufsteuerung, die bei allen Algorithmen gleich sind, werden von der Klasse Scheduler übernommen und im unteren, grau unterlegten Bereich angezeigt.

Model und SimpleModel

Model ist eine Unterklasse von JApplet und wird daher vom Webbrowser aufgerufen, um das Applet zu starten und anzuzeigen. Der Webbrowser ruft dazu die Methode `Model::init` auf, die jeweils

ein `JPanel` für das Modell und den Scheduler anlegt und dem Java-Layoutmanager übergibt. Anschließend verzweigt sie zur Methode `initModel` der konkreten Unterklasse (hier: `SimpleModel`). Dort werden die internen Datenstrukturen des Modells initialisiert, die Interaktionselemente der Benutzungsoberfläche angelegt und die Musterlösung erzeugt und an den Scheduler übermittelt. Beim Beispiel `SimpleModel` werden zehn nummerierte Checkboxes angelegt und als Musterlösung zehn `NumberCalls` erzeugt, die jeweils eine Checkbox selektieren. Die spezifische Unterklasse von Model ist auch dafür zuständig, zu prüfen, ob der Lerner tatsächlich die Aufgabe gelöst hat. In `SimpleModel::finish` wird geprüft, ob tatsächlich alle zehn Checkboxes selektiert sind.

Die Methode `Model::getResource` dient dazu textuelle Meldungen des Applets entsprechend der eingestellten Sprache (z.B. Deutsch, Englisch) zusammenzustellen.

Call und NumberCall

Für jeden Typ von Funktionsaufruf muss der Autor eine Unterklasse von Call anlegen. Sie speichert einen Verweis auf das Modell, einen Hilfetext, der die Ausführung beschreibt, einen Text für die Undo-Liste und die Parameter des Funktionsaufrufs. Beim Applet `Simple` speichert die Klasse `NumberCall` die Nummer des gewählten Kontrollkästchens, und ob es selektiert oder deselektiert wird.

Scheduler

Der Scheduler verwaltet die History-Liste mit den Funktionsaufrufen (*Call*). Intern wird sie in zwei Teillisten repräsentiert (Abb. 33). Bereits ausgeführte Calls werden in der Liste *backward*, noch auszuführende Calls in der Liste *forward* gespeichert. Über die Methode *Scheduler::addCall* fügt das Modell bei der Initialisierung mehrfach einen Call der Musterlösung hinzu, der dann am Anfang der *forward*-Liste eingefügt wird. Versucht der Lerner, eine Funktion auszuführen, so übernimmt das Modell die Signaturprüfung, erzeugt das Call-Objekt und übergibt es dem Scheduler über *Scheduler::doCall*. Im Modus Übung vergleicht der Scheduler den gewünschten Aufruf mit der Musterlösung über *Call::equalsCall*. Er führt einen Aufruf über *Call::perform* aus. Die *perform*-Methode führt den Aufruf aus und erzeugt ein inverses Call-Objekt, das in der *backward*-Liste des Schedulers gespeichert wird.

Interagiert der Nutzer mit einem GUI-Element der Ablaufsteuerung (z.B. Button „Vorwärts“), so ruft das Java-Laufzeitsystem die Methode *Scheduler::actionPerformed* auf, die je nach GUI-Element weiterverzweigt und spezifische Methoden des Schedulers aufruft. In zukünftigen Versionen der Bibliothek soll jedes GUI-Element durch ein Objekt realisiert werden, das seinen Zustand (unsichtbar, deaktiv, wählbar) entsprechend dem SALA-Modus und dem Zustand der History-Liste einstellt und Mausclick-Ereignisse entgegennimmt und weiterleitet.

Sample

Soll ein Applet mehrere Sätze von Eingabedaten für den Algorithmus bereithalten, so legt der Autor eine Unterklasse von *Sample* an. Die Sätze werden vom Modell verwaltet und in einer Pull-Down-Liste angeboten. Beim Applet *Simple* kommt dies nicht zum Einsatz.

6.7.3 SimpleModel Programmcode

Zur weiteren Illustration, wie ein Applet mit SALABIM erstellt werden kann, werden in den folgenden Abbildungen Auszüge aus den wesentlichen Methoden *perform* (Abb. 36), *setNumberSelected* (Abb. 37), *equalsCall* (Abb. 38) und *finish* (Abb. 39) des Applets *Simple* gezeigt. Die Auszüge stammen aus dem von Jan Schormann erstellten Programmcode.

```
public Call perform(boolean inverse){
    NumberCall inverseCall;
    boolean previousState, newState;

    // 1. *** Check the rules:
    // When stepping forward, mark as selected only.
    // When stepping backward, mark as deselected only.
    if (selects == inverse)
        return new
            FailCall(model.getResource("W
                arnFail"));

    // 2. *** Generate inverse data:
    inverseCall = new NumberCall(model,
        number, !selects, getHint(),
        getDescription());

    // 3. *** Display changes (possibly as an animation):
    // Isn't done in this simple model.

    // 4. *** Change the state of the model:
    // 5. *** Display the new state:
    model.setNumberSelected(number,
        selects);

    return inverseCall;
}
```

Abb. 36: *NumberCall::perform* Programmcode

```
public void setNumberSelected(int
    number, boolean selected){
    // 4. *** Change the state of the model:
    data[number] = selected;

    // 5. *** Display the new state:
    if (views[number].isSelected() !=
        selected){
        views[number].removeItemListener(t
            his);
        views[number].setSelected(selected
            );
        views[number].addItemListener(this
            );
    }
```

Abb. 37: *SimpleModel::setNumberSelected* Programmcode

```
public boolean equalsCall(Call other){
    // is it of the same class etc.?
    if (!super.equalsCall(other))
        return false;

    NumberCall call = (NumberCall) other;

    // has it the same data?
    return number == call.number &&
        selects == call.selects;
}
```

Abb. 38: *NumberCall::equalsCall* Programmcode

```

public String finish(){
    int i;

    for (i=1; i<=10; i++){
        if (data[i] == false)
            return
                getResource("WarnNotFinished");
    }

    return getResource("DescFinish");
}

```

Abb. 39: SimpleModel::finish Programmcode

Event-Handling der GUI-Komponenten

Werden GUI-Komponenten verwendet, die bei einer Interaktion sofort ihren Zustand ändern, so müssen diese gesondert eingebunden werden. Bei

der Implementierung gab es Schwierigkeiten mit dem Verhalten der Checkboxes (Swing-Klasse JCheckbox). Klickt ein Nutzer auf eine Checkbox, so verändert dies sofort ihren Zustand und löst erst dann die dazugehörige Methode *SimpleModel::itemStateChanged* aus. Wurde eine Checkbox z.B. „verbotenerweise“ deselektiert, so muss ihr Zustand in der Methode *itemStateChanged* wieder zurückgesetzt werden. Entsprechend wird bei *SimpleModel::setNumberSelected* der Zustand nur verändert, falls die Checkbox dies nicht schon selbst getan hat. Um den Zustand zu ändern, muss kurzzeitig der ItemListener deaktiviert werden, da die Checkbox sonst *SimpleModel::itemStateChanged* von ihrem Zustandswechsel informieren und damit eine Endlosschleife auslösen würde.

6.8 Beispielapplet „Blöcke stapeln“

Abbildung 40 zeigt das zweite Beispiel-Applet „Puzzle“ zur SALABIM-Bibliothek.

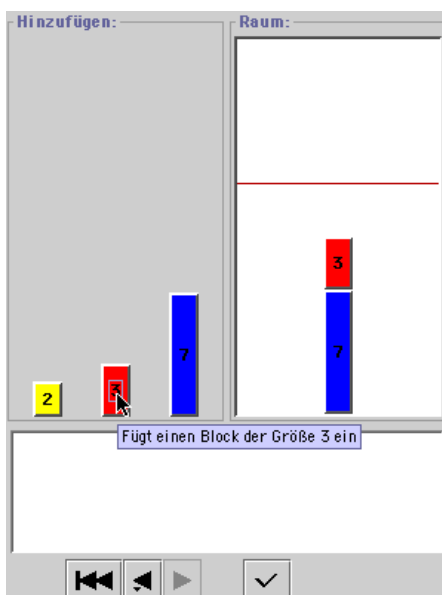


Abb. 40: Beispielapplet: Blöcke stapeln

Der Nutzer soll aus Blöcken einen Turm aufstapeln, der eine genau vorgegebene Höhe hat. Es stehen beliebig viele Blöcke in den Größen 2, 3 und 7 zur Verfügung. Durch Klicken auf einen der Musterblöcke im linken Bereich wird ein Block dieser Größe auf den Stapel gelegt. Dies wird in einer

weichen Bewegung dargestellt, indem der neue Block vom Musterblock zur Spitze des Stapels bewegt wird. Der Blockstapel soll genau die Höhe 13 erreichen, die durch die rote Linie angegeben ist.

Es ist möglich, einen Stapel zu bilden, der über die rote Linie reicht, dies wird aber vom Applet nicht als Lösung akzeptiert. Das Applet berechnet über Backtracking eine Musterlösung, die man sich im Modus Animation ansehen kann. Es ist nicht ganz einfach die Aufgabe zu lösen, da man in der Regel einige Male mit verschiedenen Blöcken probieren muss. Dabei zeigt sich der Wert der Undo-Funktion, da man damit einen oder mehrere Schritte zurücknehmen kann.

Der Programmcode dieses Beispiel-Applets ist umfangreicher als der des Applets Simple. Das Applet verwendet weiche Übergänge zwischen Zuständen der Datenstruktur (mithilfe von *Model::animate*). Das Modell besteht nun aus mehreren Klassen (*Block*, *BlockStack* und *StackLayout*), da insbesondere die Visualisierung aufwändiger ist. In diesem Applet gibt es zu den regulären Calls (*AddBlock*) eine eigene Inverse-Klasse *RemoveBlock*. Beide sind Unterklassen einer gemeinsamen Klasse (*BlockCall*). Das Applet zeigt damit, wie der Programmcode größerer Applets in mehrere Klassen gegliedert werden kann.

6.9 Verwandte Arbeiten zu Undo/Redo

In der Literatur zur Software-Ergonomie gibt es einen breiten Konsens, dass interaktive Systeme ein mehrstufiges Undo/Redo anbieten sollen (siehe z.B. [Preim 1999] Kap. 4.1.11 und [Herczeg 1994] Kap 13). Ein Undo/Redo-Mechanismus erlaubt es den Nutzern, verschiedene Operationen auszuprobieren, ohne die bisherige Teillösung einer Arbeitsaufgabe zu gefährden. Sie können damit die Leistungsmöglichkeiten eines Systems erkunden und sich an die Lösung einer Aufgabe heranarbeiten.

Meyer beschreibt in [Meyer 1997] Kapitel 21 die Implementierung einer Bibliothek (Framework) für ein mehrstufiges Undo/Redo. Eine Benutzerinteraktion für eine Operation wird in einem Command-Objekt gespeichert. Dieses enthält alle Daten, um die Operation ausführen und rückgängig machen zu können. Gamma et al. beschreiben einen ähnlichen Ansatz wie Meyer, speichern aber die Undo/Redo-Daten getrennt vom Command-Objekt in einem Memento-Objekt ([GHJV 1994], S. 233-242). Demgegenüber enthalten die Call-Objekte von SALABIM nur die Daten für Undo oder Redo. Erst beim Ausführen eines Call-Objektes werden die Daten für das inverse Call-Objekt berechnet. Geht der Benutzer mehrfach mit Undo/Redo durch die Operationen der History-Liste, so verursachen Meyers und Gammas Ansatz verglichen mit SALABIM einen geringeren Rechenaufwand, brauchen aber mehr Speicherplatz für die Command-Objekte.

Während SALABIM die Modi Simulation, Übung und Animation unterstützt, bieten die Ansätze von Meyer und Gamma nur eine freie Interaktion in der Art des SALABIM-Modus Simulation an. Es ist aber vermutlich möglich, ihre Ansätze so zu erweitern, dass auch die anderen Modi unterstützt werden.

Einige Systeme zur Erstellung von Algorithmen-Animationen unterstützen ein Vorwärts- und Rückwärtsgehen durch die Animation in Einzelschritten. In den mir bekannten Publikationen wurde aber nicht auf die softwaretechnische Realisierung dieses Mechanismus eingegangen. Keines der mir bekannten Systeme unterstützt ein freies Experimentieren mit dem Algorithmus im Sinne des SALABIM-Modus Simulation. Ein weiteres Problem vieler Systeme ist die Schrittweite der Animation. Sie orientiert sich nicht wie bei SALABIM an wichtigen Ereignissen im Ablauf des Algorithmus, sondern an einzelnen grafischen Operationen der Visualisierung (vgl. Kap. 5.1, S. 53).

Verglichen mit bestehenden Ansätzen ist die SALABIM-Bibliothek innovativ, da sie alle drei Modi (Simulation, Übung, Animation) unterstützt und sich die Schrittweite an wichtigen Ereignissen des Algorithmus (Funktionsaufrufen) orientiert.

7 Studierende bewerten SALA-Lernprogramme und SALA

7.1 Einführung

Im Sommersemester 2000 erprobte ich zwei nach SALA entwickelte Lernprogramme in einer Lehrveranstaltung des Grundstudiums. Bis dahin waren noch keine der nach SALA entwickelten Programme in regulären Lehrveranstaltungen eingesetzt worden. Die Studierenden sollten auf Fragebögen bewerten, wie hilfreich die Lernprogramme und das dahinter stehende didaktische Konzept für das Erlernen der Algorithmen waren. Um die Voraussetzungen der Studierenden einschätzen zu können, wurden das Studienfach, das Vorwissen zum Algorithmus und die Sicherheit im Umgang mit Webbrowsern abgefragt. Die Studierenden konnten die Lernprogramme im Rechnerpool der Informatik oder auf ihren privaten Rechnern nutzen. Daher wurde mit abgefragt, ob es technische Probleme mit den Lernprogrammen gab.

Nach der Erprobung wurden Daten des Webserver über die Nutzung der Lernprogramme ausgewertet. Dadurch wurde ermittelt, wie die Studierenden durch die Lernprogramme navigiert sind, wie intensiv sie die Programme genutzt haben und wie das Verhältnis von Online- zu Offline-Nutzung war. Die Nutzung der Lernprogramme wird im Kapitel 8 analysiert.

Dieses Kapitel stellt die eingesetzten Lernprogramme und die Lehrveranstaltung kurz vor. Anschließend werden die Gestaltung der Evaluation thematisiert und die Wertungen und Freitextantworten auf den Fragebögen analysiert.

7.1.1 Eingesetzte Lernprogramme

In der Lehrveranstaltung wurden die Lernprogramme „Binomial Heap“ und „Heapsort“ erprobt. Die Lernprogramme werden nun kurz vorgestellt.

„Binomial Heap“

Karsten Block entwickelte im Rahmen seiner Diplomarbeit [Block 1999a] nach dem Konzept SALA das Lernprogramm „Binomial Heap“ [Block 1999b] (Abb. 41). Ein „Binomial Heap“ ist eine Datenstruktur für die Verwaltung von Vorrangwarteschlangen (engl. Priority Queue). In die so reali-

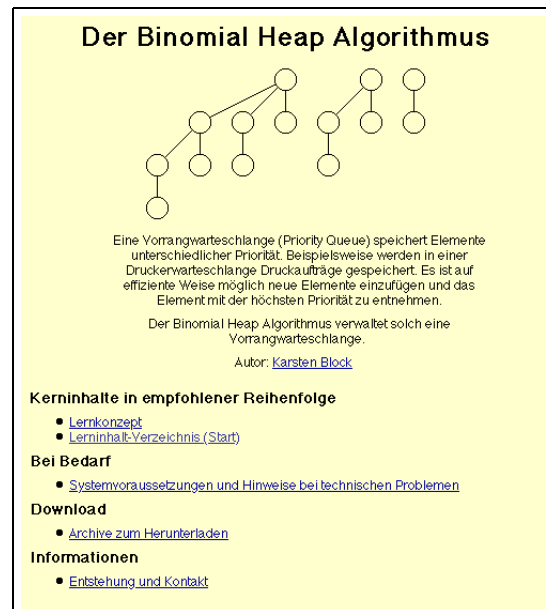


Abb. 41: Startseite Binomial Heap
Bildschirmabzug von [Block 1999b]

sierte Vorrangwarteschlange können effizient neue Elemente eingefügt und das kleinste Element entnommen werden. Die Datenstruktur und die dazugehörigen Algorithmen zur Manipulation der Daten sind nicht einfach zu verstehen. Daher bot es sich an, diese in einem interaktiven Lernprogramm zu beschreiben. Informationen zur Seitenstruktur des Lernprogramms „Binomial Heap“ finden sich in Kap. 8.2.2, S. 90.

Block evaluierte das Programm mit fünf Informatikstudenten. Dazu beobachtete er sie bei der Arbeit mit dem Programm, prüfte das erlangte Wissen mit einem schriftlichen Test und interviewte sie zu ihren Einschätzungen und ihrem Arbeitsverhalten. Die Studenten kamen insgesamt gut mit dem Programm zurecht und äußerten sich positiv zum dahinter stehenden didaktischen Konzept SALA.

„Binomial Heap“ war das erste vollständig nach SALA gestaltete Lernprogramm. Meines Erachtens werden in dem Programm die Kernpunkte des Konzepts sehr gut umgesetzt und es ist insgesamt von sehr guter Qualität. Es war aber noch nicht in der Ausbildung der Informatikstudierenden in Algorithmen eingesetzt worden.

Heapsort

Im Frühjahr 2000 entwickelte ich mit Tobias Gross das Lernprogramm „Heapsort“ nach den Prinzipien von SALA (vgl. Kap. 4). Wir erprobten auch dieses Programm in der Lehrveranstaltung, um Vergleichswerte zum Lernprogramm „Binomial Heap“ zu gewinnen. Das Lernprogramm zu Heapsort war zu diesem Zeitpunkt aber insofern proto-

typisch, als dass die tutoriellen Texte sehr knapp gehalten waren und das Lernprogramm hauptsächlich aus interaktiven Aufgaben bestand. Nach der Evaluation wurde das Lernprogramm weiterentwickelt. Die neue Version ist in Kapitel 4 beschrieben. Informationen zur Seitenstruktur finden sich in Kap. 8.2.1, S. 87.

7.2 Lehrbetrieb

Aus praktischen Gründen bot es sich an, die Lernprogramme in einer Lehrveranstaltung des Oldenburger Informatikstudiums einzusetzen und zu erproben. Die Abläufe im hiesigen Lehrbetrieb waren mir gut bekannt und es war für mich einfach, persönliche Kontakte zu den Lehrenden und Studierenden zu pflegen. Dies wurde mir dadurch erleichtert, dass ich als Tutor für die Lehrveranstaltung tätig war, in der ich die Lernprogramme erprobte.

7.2.1 Lehrveranstaltung Datenstrukturen

Heapsort und Binomial Heap wurden in der Lehrveranstaltung „Datenstrukturen“ behandelt. Sie gliedert sich in die Teile *Vorlesung* und *Übung*.

Die *Vorlesung* fand zweimal in der Woche für je 90 Minuten in einem Hörsaal statt. Sie sollte vor allem Sachwissen vermitteln. Der Dozent trug den Lernstoff mit Folien vor, die er erklärte. Die Studierenden hörten dem Vortrag zu und betrachteten die Folien. Sie konnten auch Fragen stellen, wovon vereinzelt Gebrauch gemacht wurde. Es bestand keine Anwesenheitspflicht. Für die Vorlesung wurde ein Skript angeboten, das die Studierenden aus dem Internet laden konnten. Für die einzelnen Vorlesungsteile wurde vom Dozenten Begleitliteratur angegeben. Für die in der Evaluation betrachteten Algorithmen war dies das Lehrbuch von Ottmann und Widmayer [Ottmann, Widmayer 1996]. Der Dozent erklärte die beiden Algorithmen in der gleichen Weise wie das Lehrbuch.

Die *Übung* soll die Studierenden befähigen den Stoff anzuwenden und Aufgaben zu lösen. Dabei werden unterschiedlich schwierige Aufgaben bearbeitet: Vom Anwenden einer Formel, über das Abarbeiten eines Algorithmus und Programmieraufgaben bis hin zu Beweisen. Ein guter Teil der Aufgaben erfordert kreatives logisches Denkvermögen und lässt sich nicht mechanisch mit dem in der Vorlesung kennen gelernten Methoden lösen. Einmal in der Woche erhielten die Studierenden ein Aufgabenblatt, das in Zweiergruppen binnen einer Woche zu lösen war. Jeweils rund 10 Zweiergruppen wurden von einem der fünf Tutoren betreut. Der Tutor korrigierte die schriftlich verfassten Lösungen. Die Studierenden konnten so erkennen, wie gut sie die Teilgebiete verstanden hatten und aus den Fehlern lernen. In einer 90-minütigen Übungsbesprechung wurden wöchentlich Lösungen zu den Aufgaben des Aufgabenblattes besprochen. Der Tutor konnte dazu Studierende aufrufen oder die Lösung selbst vorrechnen.

Die Lehrveranstaltungen „Programmierung“ und „Datenstrukturen“ bilden zusammen den Teil „Informatik A“, der als mündliche Prüfung im Vordiplom abgelegt werden muss. Als Voraussetzung für die Zulassung muss in einer der beiden Lehrveranstaltungen ein Schein erworben werden (Diplomprüfungsordnung 3, 1998). Für den Scheinerwerb muss ein Studierender regelmäßig und aktiv an den Übungen teilnehmen, mindestens die Hälfte der Punkte aus den Aufgaben erreichen und eine schriftliche Klausur bestehen.

7.3 Gestaltung der Evaluation

Die Evaluation sollte klären, wie sich nach SALA gestaltete Lernprogramme und das dahinter stehende didaktische Konzept SALA in einem praktischen Lernkontext bewähren. Dieser Abschnitt beschreibt die Gestaltung der Evaluation. Zunächst wurde eine Evaluationsmethode ausgewählt und dann unter Berücksichtigung der Evaluationsziele, der Lernprogrammeigenschaften und des Lehrbetriebs Aufgabenblatt und Fragebogen für die Evaluation gestaltet. Um die Qualität zu sichern, wurden schließlich das Lernprogramm zu Heapsort und der Fragebogen in einem Vortest erprobt.

7.3.1 Wahl der Evaluationsmethode

Die Hauptzielgruppe für die Lernprogramme „Binomial Heap“ und „Heapsort“ sind Studierende der Informatik. Es bot sich hier die Gelegenheit, die Lernprogramme in dem Umfeld einzusetzen, in dem der Lernstoff auf herkömmliche Weise vermittelt wird. Es ist anzunehmen, dass dies ein typisches Einsatzfeld für Lernprogramme werden wird: als Lernmaterial für das Eigenstudium, das ähnlich dem Lehrbuch die Lehrformen Vorlesung und Übung ergänzt. Ein solches Einsatzszenario war auch das Ziel der Entwicklung von Lernprogrammen im Projekt „Medienunterstütztes Studium der Informatik“ [Gorny, Faltin 2000].

Die Studierenden können ihren Lernprozess (methodisch gesehen) frei gestalten. Sie können die Vorlesung besuchen, Lehrbücher lesen, im Internet recherchieren und Lerngruppen bilden. Studierende, die den Schein in „Programmierung“ erworben haben, sind auch nicht gezwungen, an den Übungen zu „Datenstrukturen“ teilzunehmen. Es steht ihnen frei, wie viel Zeit und Energie sie den einzelnen Aktivitäten zuwenden. Meines Wissens gibt es erst sehr wenig interaktives Lernmaterial (z.B. Lernprogramme) zu Algorithmen, das über reine Animationen hinausgeht (dies wird durch [Peters 2000] bestätigt). Die Studierenden, mit denen ich gesprochen habe, kamen das erste Mal durch die Lernprogramme der Evaluation mit interaktivem Lernmaterial zu Datenstrukturen in Kontakt. Neben diesen Faktoren aus der persönlichen Lernplanung wirken noch Faktoren wie Vorwissen, Motivation, Leistungsbereitschaft und persönliche Lebensumstände auf den Lernprozess und den Lernerfolg ein.

Um zu bestimmen, wie hilfreich die Lernprogramme und das dahinter stehende didaktische Konzept SALA für das Lernen sind, ist es denkbar, zwei Indikatoren heranzuziehen:

1) Testleistung. Die Studierenden arbeiten das Lernprogramm durch. Anschließend werden ihnen Aufgaben gestellt, die sie lösen sollen. Die Lösungen werden korrigiert und bepunktet.

2) Meinungsbild. Die Studierenden arbeiten das Lernprogramm durch. Anschließend beurteilen sie die einzelnen Punkte des didaktischen Konzepts.

Testleistung

Um einen Zusammenhang zwischen einem Punkt des didaktischen Konzepts und der Testleistung nachzuweisen, müssen mehrere Paare von Versionen eines Lernprogramms entwickelt werden, bei denen sich die Paare in einem Punkt des Konzepts unterscheiden. Um Unterschiede in der Leistungsfähigkeit der verschiedenen Studierenden auszugleichen, wird jede Version mit einer Gruppe von Studierenden erprobt. Die anderen Faktoren des Lernkontextes müssen konstant gehalten werden. Dies ist nur in einer Laborsituation möglich. Dort wird genau gesteuert, welche Lernmaterialien die Studierenden bekommen und wie lange sie lernen können. Zum Beispiel haben [Hansen, Narayanan, Schrimpscher 2000] ein hypermediales Lernprogramm zu Algorithmen in dieser Weise evaluiert.

Es dürfte aber schwer sein, Studierende für solch einen „Versuch“ zu gewinnen. Auch ist es aufwändig mehrere Versionen eines Lernprogramms herzustellen und die Lernbedingung im Labor vorzubereiten und zu kontrollieren. Ein Vorteil einer solchen Evaluation wäre allerdings, dass das Ergebnis in dem Sinne objektiver wird, da „Lernerfolg“ durch „Testleistung“ recht genau angenähert werden kann und die Ergebnisse auch nicht von der Fähigkeit zum didaktischen Denken und zur Selbsteinschätzung der Studierenden abhängig ist.

Meinungsbild

Orientiert man sich am Indikator Meinungsbild, so können die Studierenden in ihrer gewohnten Umgebung arbeiten. Dadurch wird das Lernprogramm in dem Umfeld erprobt und bewertet, in dem es letztlich auch eingesetzt werden soll. Die Studierenden werden eher bereit sein, bei der Evaluation mitzumachen, da sie nicht in eine künstliche Laborsituation hineingezwängt werden. Die Aussagekraft der erhaltenen Antworten hängt aber davon ab, ob die Studierenden fähig sind, didaktisch orientierte Fragen zu verstehen und über Ihre Erfahrungen mit dem Lernprogramm zu re-

flektieren. Es ist nicht nötig, mehrere Versionen des Lernprogramms zu entwickeln oder Einflussfaktoren des Lernkontextes konstant zu halten.

Die Lehrveranstaltung „Datenstrukturen“ wurde von rund 100 Studierenden besucht. Es war mir daher vom Aufwand her nicht möglich Einzelbeobachtungen durchzuführen, wohingegen der Aufwand der Erhebung eines Meinungsbildes gut zu bewältigen war. Entscheidend für die Wahl der Evaluationsmethode „Meinungsbild“ war aber auch, dass nur so eine Erprobung im authentischen Lernkontext möglich war. Für diese Alternative sprach auch, dass zur Evaluation von SALA in der Diplomarbeit Block [Block 1999a] bereits die Evaluationsmethoden „Beobachtung bei der Arbeit mit dem Lernprogramm“, „Testaufgaben lösen“, und „Interview“ eingesetzt worden waren. Jede Evaluationsmethode gewährt eine andere Art von Einblick in den Lernprozess, so dass ich auch aus diesem Grund eine neue Evaluationsmethode einsetzen wollte.

7.3.2 Gestaltung der Erhebung

Die Studierenden arbeiten Lehrbücher und Skript während ihrer frei verfügbaren Zeit durch. Ebenso bearbeiten sie in dieser Zeit die Hausaufgaben der Übung. Es bot sich daher an, die Evaluation auch in Form einer „Hausaufgabe“ durchzuführen. In Absprache mit dem Dozenten und dem verantwortlichen Assistenten arbeitete ich zwei Aufgabenblätter aus. In einer Woche wurde das Lernprogramm „Heapsort“ bearbeitet, in der darauf folgenden Woche das Lernprogramm „Binomial Heap“. Die Aufgabenblätter sind im Anhang abgebildet (Seite 101 und Seite 102).

Die Erhebung bestand für die Studierenden aus den folgenden Teilen:

1. Lernprogramm durcharbeiten
2. Didaktisches Konzept bewerten (Fragebogen ausfüllen)
3. Klassische Aufgaben bearbeiten

Lernprogramm durcharbeiten

Die Studierenden konnten das Lernprogramm im studentischen Rechnerpool der Informatik (ARBI-Unix-Cluster) laufen lassen oder auf ihrem privaten PC installieren und verwenden. Das Lernprogramm baut technisch auf gängigen WWW-Formaten auf: HTML und GIF für den tutoriellen Teil und Java-Applets für die interaktiven Aufgaben. Für das Lernprogramm „Heapsort“ war es für die Installation auf dem privaten PC nötig, eine Java-

Bibliothek zu installieren. Die Lernprogramme konnten seitenweise oder als Ganzes über das Internet heruntergeladen werden. Im Rechnerpool waren die Programme vorinstalliert und konnten direkt gestartet werden. Beim Lernprogramm „Heapsort“ können die Applets in die Modi Simulation, Üben und Animation geschaltet werden. Dies wurde technisch möglich, da „Heapsort“ auf der Bibliothek SALABIM aufsetzt. Die Studierenden sollten dadurch auch die verschiedenen Modi ausprobieren können, um beurteilen zu können, welcher Modus ihren Lernvorlieben entspricht. Die Studierenden sollten aber zunächst versuchen, die Aufgaben im Modus Simulation zu bearbeiten, um die Funktionen des Algorithmus „nachentdeckend“ zu lernen. Da das Lernprogramm „Binomial Heap“ unter anderen technischen Voraussetzungen entstanden ist, ist es nicht möglich, den Modus eines Applets zu ändern. Der Autor des Lernprogramms „Binomial Heap“ hat für jede Aufgabe einen Modus festgelegt.

Fragebogen ausfüllen

Anschließend sollte jede Arbeitsgruppe einen Fragebogen ausfüllen. Die Studierenden sind zwar angehalten, jedes Aufgabenblatt in Gruppenarbeit zu lösen, doch wechseln sich in vielen Gruppen die beiden Mitglieder in der Bearbeitung eines Aufgabenblatts ab. Damit in diesem Fall niemand gedrängt wird, den Fragebogen auszufüllen, wenn er das Lernprogramm nicht durchgearbeitet hat, sollte jede Arbeitsgruppe nur einen Fragebogen ausfüllen.

Die Fragebögen sind im Anhang dargestellt (Seite 103 und Seite 104). Sie bestehen aus den Teilen:

- Allgemeine Fragen zu den Voraussetzungen
- Fragen zum didaktischen Konzept
- Fragen zum Lernprogramm
- Platz für freie Anmerkungen
- Erreichte Punktzahlen in den Aufgaben

Über die *allgemeinen Fragen* sollte festgestellt werden, ob es Defizite bei den Voraussetzungen für die Arbeit mit dem Lernprogramm gab. Es ist möglich, dass Informatik-Studierende gegenüber Studierenden aus anderen Studienfächern besseres Vorwissen haben oder stärker motiviert sind. Vom Studienverlauf haben sie aber höchstens ein Semester „Vorsprung“ an Wissen. Das Vorwissen bezüglich des Algorithmus kann sich auf die Wahrnehmung des Lernprogramms und des didaktischen Konzepts auswirken. Da das Lernprogramm innerhalb eines Webbrowsers abläuft, ist es vorteilhaft, wenn der Studierende den Um-

gang mit einem Webbrowser sicher beherrscht. Sonst können ihn Probleme mit der Bedienung des Browsers von der Lernaufgabe ablenken. Besonders bei Studierenden, die das Lernprogramm auf ihrem privaten PC installierten, war wichtig zu wissen, ob die Installation problemlos geklappt hatte. Technische Probleme können den Studierenden verärgern, so dass die nachfolgenden Urteile schlechter ausfallen. Sie können den Einsatz des Lernprogramms auch ganz verhindern. Sie deuten auf jeden Fall an, inwieweit die Zielgruppe mit der Installationsprozedur zurechtkommt.

Bei den Fragen zum *didaktischen Konzept* sollte der Studierende einzelne Punkte des didaktischen Konzepts der Lernprogramme bewerten. Dabei sollten sie sowohl ihre soeben gemachten Erfahrungen mit dem Lernprogramm als auch ihre Lernvorlieben berücksichtigen. Sie konnten dabei Bewertungen von 1 (sehr hilfreich) bis 6 (kontraproduktiv) vergeben. Der Fragebogen zu „Binomial Heap“ enthält den zusätzlichen Punkt „Anschließend wird die Funktion erklärt“, da nur in diesem Lernprogramm nach der Probierphase auch eine Erklärung des Standardverfahrens geboten wird. Bei „Heapsort“ muss das Standardverfahren aus dem Applet im Modus Animation erschlossen werden. Da die Applets bei „Binomial Heap“ nicht zwischen verschiedenen Modi umgeschaltet werden können, werden bei den Punkten für jeden Applet-Typ Beispiele genannt.

Die Frage nach dem *Gesamturteil zum Lernprogramm* sollte Gelegenheit geben, die Qualität des Lernprogramms getrennt von der Qualität des didaktischen Konzepts zu bewerten.

Der „*Platz für Ihre Anmerkungen*“ sollte es den Studierenden ermöglichen, auf Dinge hinzuweisen, die ihnen wichtig sind, die aber durch die vorgegebenen Fragen nicht abgedeckt sind.

Die Studierenden hatten anschließend zwei „klassische“ Aufgaben zu bearbeiten. Die darin *erreichten Punkte* wurden von den Tutoren auf dem Fragebogen eingetragen. Die Aufgaben wurden von mir so gestellt, dass sie Wissen aus dem Lernprogramm voraussetzen. Dadurch sollte ein gewisser Druck aufgebaut werden, das Lernprogramm auch wirklich durchzuarbeiten. Es ist für den Lehrbetrieb auch nötig Aufgaben zu stellen, für die die Studierenden „arbeiten“ müssen und die (auf Papier) abgegeben, korrigiert und bepunktet werden können. Die Aufgaben wurden so gestaltet, dass man in ihnen das Wissen aus dem Lernprogramm anwendet, aber auch Transferfähigkeit und Problemlösefähigkeit sind gefragt. Da die Einflussfaktoren auf den Lernprozess weder kontrolliert noch

protokolliert wurden, schätzte ich die Aussagekraft dieser Punktzahlen als Indikator für den Lernerfolg nicht als hoch ein.

7.3.3 Vortest von Lernprogrammen und Fragebögen

Ein erster Prototyp des Lernprogramms „Heapsort“, der im Wesentlichen nur die Applets enthielt, war einige Wochen vor dem Einsatz in der Übung fertig geworden. Ich testete diesen Prototyp informell mit einigen Personen aus meinem Umfeld und mit dem Dozenten der Veranstaltung „Datenstrukturen“. Ich beobachtete die Testpersonen bei der Arbeit mit dem Lernprogramm und sprach danach mit ihnen über ihre Einschätzung. Die Testpersonen äußerten sich positiv zum didaktischen Konzept und merkten an, dass die Arbeit mit dem Lernprogramm motiviert. Die Testpersonen ignorierten anfangs fast immer das Abstraktionsprinzip und versuchten Funktionen Schritt für Schritt von Hand auszuführen, obwohl die „höhere“ Funktion in einer Aufgabe „per Knopfdruck“ zur Verfügung stand. Der Unterschied zwischen den Modi war für die Testpersonen schwer zu verstehen.

Kurz vor dem Einsatz in der Übung testete ich das Lernprogramm und den Fragebogen mit einem Diplom-Informatiker und einem Informatikstudenten höheren Semesters. Auch sie hatten Probleme mit der Abstraktion und der Unterscheidung der Modi. Eine Person „verließ“ sich im Programm und übersprang dadurch mehrere Seiten. Die andere Person wünschte sich eine weitere Visualisierung als Ergänzung, in der die Abbildung des Array auf den Baum und die Ergebnisliste und gleichzeitig die Arbeit des Algorithmus veranschaulicht wird. Sie kamen insgesamt gut mit dem Programm zurecht und äußerten sich positiv zum didaktischen Konzept. Sie hatten keine Probleme, den Fragebogen zu verstehen und auszufüllen. Allerdings hatte ich sie auf den Sinn des Fragebogens mündlich hingewiesen.

Als Konsequenz aus dem Vortest bauten wir in das Lernprogramm eine Beschreibung der Modi als Tooltip ein und brachten einen Warnhinweis auf der Seite „Funktionsübersicht“ an, in welcher Reihenfolge man die Seiten des Lernprogramms aufrufen sollte.

Das Lernprogramm zu „Binomial Heap“ war schon vor längerer Zeit mit Benutzern getestet und danach überarbeitet worden, so dass ich hier auf einen Vortest verzichtet habe.

7.4 Auswertung der Fragebögen

Im Folgenden werden die Daten der abgegebenen Fragebögen zu „Heapsort“ (Seite 103) und „Binomial Heap“ (Seite 104) ausgewertet. Die Originaldaten sind im Anhang aufgeführt (vgl. Kap. 10.5, S. 105).

7.4.1 Abgegebene Fragebögen

Tab. 6: Abgegebene Fragebögen

Person	Heapsort	Bin. Heap	Gesamt
Lehrende	4	3	7
Studierende	43	38	81
Gesamt	47	41	88

Tabelle 6 zeigt die Anzahl der abgegebenen Fragebögen. Sie entsprechen etwa den Nutzerzahlen der Lernprogramme (vgl. Tabelle 15, S. 98), so dass man davon ausgehen kann, dass die Studierenden und Lehrenden zunächst die Programme durchgearbeitet und dann den Fragebogen ausgefüllt haben. Zu „Binomial Heap“ wurden etwas weniger Fragebögen als zu „Heapsort“ abgegeben. Möglicherweise hatten einige Übungsgruppen schon genug Punkte für die Bescheinigung der erfolgreichen Teilnahme erreicht und bearbeiteten deshalb das Aufgabenblatt zu „Binomial Heap“ nicht mehr. Von den Lehrenden (ein Dozent und fünf Tutoren) hat etwas mehr als die Hälfte Fragebögen abgegeben. Die im Folgenden behandelten Fragen aus den Fragebögen wurden fast immer vollständig ausgefüllt. Daher wird dort nicht mehr auf die Anzahl der abgegebenen Fragebögen eingegangen.

7.4.2 Studienfach der Studierenden

Tabelle 7 zeigt das Studienfach der studentischen Nutzer. Fast alle diese Fragebögen stammen von Informatik-Studierenden, nur vier Fragebögen stammen von Nebenfach-Studierenden. Bei drei der abgegebenen Fragebögen wurde als Fach In-

Tab. 7: Studienfach der Studierenden

Studienfach	Heapsort	Bin. Heap	Gesamt
Informatik	41	34	75
Biologie	1	1	2
Lehramt BBS	1	1	2
Gesamt	43	36	79

formatik und ein weiteres Studienfach genannt. Sie wurden unter Informatik eingeordnet, da in diesen Übungsgruppen das Vorwissen eines Informatikstudierenden vorhanden war.

7.4.3 Vorwissen über den Algorithmus

Tab. 8: Vorwissen

Person	Heapsort	Bin. Heap	Gesamt
Lehrende	2,67	2,50	2,60
Studierende	1,90	1,64	1,78
Gesamt	1,95	1,68	1,83

Die Teilnehmer sollten angeben, wie sie ihr Vorwissen zum Algorithmus einschätzen (Tabelle 8). Für die statistische Auswertung wurde jeder Antwortmöglichkeit ein Zahlenwert zugeordnet („keines“=1, „Grundidee“=2, „detailliert“=3).

Beide Algorithmen waren zuvor in der Vorlesung behandelt worden. Wie mir Informatiklehrer auf einer Fachtagung berichteten wird Heapsort manchmal im Gymnasium behandelt. Dagegen wird Binomial Heap dort nicht behandelt. Heapsort ist auch nicht so umfangreich wie Binomial Heap und dadurch einfacher zu verstehen. Es zeigt sich in der Selbsteinschätzung der Studierenden, dass sie ihr Vorwissen bei Heapsort als sicherer einstufen. Dieser Unterschied zeigt sich etwas schwächer auch bei den Lehrenden. Lehrende

schätzen ihr Vorwissen bei beiden Algorithmen als wesentlich sicherer ein, als dies die Studierenden tun.

7.4.4 Übung im Umgang mit Webbrowsern

Tab. 9: Übung mit Webbrowsern

Person	Heapsort	Bin. Heap	Gesamt
Lehrende	3,00	3,00	3,00
Studierende	2,62	2,57	2,59
Gesamt	2,64	2,59	2,62

Beide Lernprogramme sind webbasiert und werden daher über einen Webbrowser gelesen und bedient. Wenn ein Nutzer ungeübt im Umgang mit Webbrowsern ist, so ist zu befürchten, dass er sich auf die Bedienung des Webbrowsers konzentrieren muss und von der Lernaufgabe abgelenkt ist. Tabelle 9 zeigt die Selbsteinschätzung der Teilnehmer. Den Antwortmöglichkeiten wurden wieder Zahlenwerte zugeordnet („ungeübt“=1, „mittel“=2, „sicher“=3). Die Lehrenden stufte sich erwartungsgemäß durchgehend als „sicher“ im Umgang mit Webbrowsern ein. Aber auch die Studierenden sind im Umgang mit Webbrowsern geübt, so dass von dieser Seite keine Bedienprobleme zu erwarten sind. Die Sicherheit im Umgang mit Webbrowsern hing nicht vom verwendeten Lernprogramm ab, wie an den fast identischen Werten abzulesen ist.

7.4.5 Technische Probleme mit dem Lernprogramm

Tab. 10: Technische Probleme

Person	Heapsort	Bin. Heap	Gesamt
Lehrende	0,25	0,33	0,29
Studierende	0,60	0,18	0,41
Gesamt	0,57	0,20	0,40

Die Lernprogramme enthalten Java-Applets für die interaktiven Aufgaben. Erfahrungsgemäß gibt es mit Java-Applets immer wieder technische Probleme, die sich darin äußern, dass ein Applet nicht unter jedem Webbrowser und jedem Betriebssystem läuft. In diesem Fall kam erschwerend hinzu,

dass das Lernprogramm zu Heapsort eine lokal installierte Java-Swing-Bibliothek benötigte. Beide Lernprogramme waren von uns im Unix-Rechnerpool der Informatik erfolgreich getestet worden. Auch die Swing-Bibliothek war dort bereits installiert. Die Studierenden können diesen Rechnerpool jederzeit nutzen.

Auf dem Fragebogen wurden die Nutzer gefragt, ob sie technische Probleme mit dem Lernprogramm hatten. Wenn dies der Fall war, sollten sie die Probleme beschreiben. Für die statistische Auswertung wurde die Nennung eines Problems mit „1“ und der problemfreie Fall mit „0“ bewertet. Tabelle 10 zeigt die Mittelwerte zu den technischen Problemen. Lehrende und Studierende hatten erheblich unter technischen Problemen zu leiden. „Heapsort“ verursachte bei mehr als der Hälfte der Nutzer technische Probleme, während dies bei „Binomial Heap“ immer noch rund jeden fünften Nutzer betraf. Dies ist sehr bedauerlich und war von mir nicht erwartet worden. Vermutlich haben viele Nutzer die Arbeit mit „Heapsort“ deshalb bei der Seite mit dem ersten Applet abgebrochen (vgl. Kap. 8.5.2, S. 97). Als Konsequenz habe ich das Lernprogramm so verändert, dass Swing nun automatisch eingebunden wird (vgl. Kap. 6.4, S. 60). Die Probleme mit „Binomial Heap“ sind mir nicht erklärlich, da das Programm auf einer sehr frühen Java-Version aufsetzt und auf allen bisher getesteten Plattformen sofort problemlos lief. Leider helfen die wenigen Problembeschreibungen zu „Binomial Heap“ (vgl. Seite 83) hier nicht weiter. Es werden dort fast nur allgemeine Darstellungsprobleme des Webbrowsers angesprochen.

7.4.6 Gesamteinschätzung des Lernprogramms

Tab. 11: Gesamteinschätzung Lernprogramm

Person	Heapsort	Bin. Heap	Gesamt
Lehrende	1,88	2,00	1,93
Studierende	2,71	2,05	2,39
Gesamt	2,64	2,05	2,35

Die Nutzer wurden gebeten zu bewerten, wie hilfreich die Lernprogramme für das Erlernen des Stoffes waren. Die Skala der möglichen Antworten reichte von „sehr hilfreich“=1 bis „kontraproduktiv“=6. Die Mittelwerte der Gesamturteile sind in Tabelle 11 angegeben. Das Lernprogramm „Binomial Heap“ erhielt eine bessere Einstufung als das Lernprogramm „Heapsort“. Dieser Unterschied ist

bei den Studierenden stärker als bei den Lehrenden ausgeprägt. Er findet sich auch in den textuellen Anmerkungen wieder (vgl. Seite 84). Gemittelt über beide Programme waren die Lehrenden in ihrem Urteil etwas positiver als die Studierenden. Gemittelt über beide Nutzergruppen und beide Programme ist die Einschätzung recht positiv und liegt zwischen „hilfreich“ und „brauchbar“. In persönlichen Gesprächen haben mir Lehrende und Studierende immer wieder bestätigt, dass sie die beiden Lernprogramme für eine wertvolle Ergänzung zum Präsenzstudium halten. Auch die in den Lernprogrammen verwirklichten Konzepte, wie das entdeckende Lernen wurden immer wieder wertgeschätzt. Die technischen Probleme und die Schwächen der Lernprogramme wurden aber ebenso offen angesprochen.

7.4.7 Punkte für die Aufgaben

Wenn eine Übungsgruppe die schriftlichen Aufgaben bearbeitet hatte, so vermerkte der Tutor oder die Tutorin die erreichte Punktzahl auf dem Fragebogen. Da nur etwas mehr als die Hälfte der Gruppen die Aufgaben bearbeitet hat, werden die Punktzahlen hier nicht weiter ausgewertet.

7.4.8 Gesamteinschätzung von SALA

Tab. 12: Gesamteinschätzung von SALA

	Heapsort	Bin. Heap	Differenz
Didaktisches Konzept	2,47	2,07	-0,40
Lernprogramm	2,64	2,05	-0,59
Differenz	0,17	-0,02	

Tabelle 12 zeigt die Mittelwerte der Gesamteinschätzungen des didaktischen Konzepts SALA und des Lernprogramms auf den Fragebögen zu „Heapsort“ und „Binomial Heap“.

Während zu erwarten war, dass zwei verschiedene Lernprogramme unterschiedlich eingeschätzt werden, so hätte man für das didaktische Konzept bei beiden Fragebögen ähnliche Werte erwartet, da hinter beiden Lernprogrammen das didaktische Konzept SALA steht. Erstaunlicherweise differieren die Werte immerhin um 0,40, wobei auf den Fragebögen zu „Binomial Heap“ wieder die positiveren Werte vergeben wurden. Offensichtlich haben die Nutzer die negativere Bewertung des Lernprogramms „Heapsort“ auf die Bewertung des didaktischen Konzepts übertragen. Aus psychologischer Sicht ist das verständlich, da die Nutzer ja gerade intensiv mit dem Lernprogramm gearbeitet hatten und noch unter dem Eindruck ihres Lernerlebnisses standen. Auch die Mehrzahl der schriftlichen Anmerkungen der Nutzer widmet sich konkreten Eigenschaften der Lernprogramme und weniger den dahinter stehenden didaktischen Prinzipien. Einzelne Gespräche mit Teilnehmern meiner Übungsgruppe und den Tutoren deuteten ebenfalls in diese Richtung. Daher berücksichtige ich bei der Analyse der folgenden didaktisch orientierten Fragen, dass die Nutzer stets auch das Lernprogramm mit bewertet haben.

Vergleicht man nun für die Fragebögen zu „Binomial Heap“ und „Heapsort“ die Bewertung von SALA und Lernprogramm, so zeigt sich wieder, dass die Nutzer in der Regel nicht klar zwischen der Qualität des Lernprogramms und der Qualität des dahinter stehenden didaktischen Konzepts trennten, da die Werte sehr ähnlich sind. Bei „Binomial Heap“ sind die Werte sogar fast identisch. Dies mag daran liegen, dass die Nutzer das Konzept als gut umgesetzt sahen und daher keinen Unterschied in der Bewertung von Konzept und Lernprogramm machten. Demgegenüber wird auf den Fragebögen zu „Heapsort“ das Lernprogramm mit 0,17 schlechter bewertet als das Konzept SALA. Einige Nutzer schrieben, dass sie zwar ein gutes Konzept hinter dem Lernprogramm „Heapsort“ sehen, dieses aber im Lernprogramm nicht zufriedenstellend umgesetzt ist. Der Unterschied ist aber nicht groß, so dass man auch hier davon ausgehen muss, dass im Regelfall nicht zwischen Konzept und Lernprogramm unterschieden wurde, sondern die Fragen hauptsächlich im Hinblick auf das Lernprogramm beantwortet wurden.

7.4.9 Einschätzung didaktischer Punkte

Tab. 13: Einschätzung didaktischer Punkte

	Heapsort	Bin. Heap	Differenz
Grafische Benutzungsoberfläche	1,91	1,78	0,13
Aufgaben integriert	2,24	2,07	0,17
Viele Funktionen	2,24	1,93	0,32
Datenstruktur grafisch	2,38	2,04	0,34
Lernen durch Experimentieren	2,61	2,26	0,35
Modus Übung	2,46	2,09	0,37
Modus Animation	2,82	2,26	0,55
Modus Simulation	2,78	1,94	0,85
Anschließend Erklärung		2,12	

Tabelle 13 zeigt die Mittelwerte der Einschätzung didaktischer Punkte durch die Nutzer, getrennt nach den beiden Lernprogrammen. Der Wortlaut der Fragen zu diesen Punkten ist in den Fragebögen enthalten (Seite 103 und Seite 104).

„Heapsort“ hat bei allen Punkten schlechtere Werte bekommen als „Binomial Heap“ (Unterschied 0,13 bis 0,85). Dies ist verständlich, da vermutlich das schlechtere Gesamturteil zum Lernprogramm auf die Einschätzung der einzelnen Fragen abgefärbt hat. Auch streuen die Werte bei „Heapsort“ (1,91 bis 2,82) stärker als bei „Binomial Heap“ (1,78 bis 2,26). Dies kann daran liegen, dass einzelne Punkte die Nutzer bei „Heapsort“ besonders störten, während „Binomial Heap“ in den verschiedenen Punkten gleichmäßigere Qualität hatte.

In der Tabelle sind die Punkte aufsteigend geordnet nach der Differenz zwischen „Heapsort“ und „Binomial Heap“. Die zuerst aufgeführten Punkte zeigen nur eine geringe Differenz, was sich so deuten lässt, dass sie hauptsächlich mit Blick auf das didaktische Konzept beantwortet wurden, wäh-

rend die weiter unten stehenden Punkte hauptsächlich mit Blick auf das Lernprogramm beantwortet wurden. Da die Nutzer wenige technische und inhaltliche Probleme bei der Arbeit mit „Binomial Heap“ hatten, gehe ich davon aus, dass die Nutzer bei der Beantwortung der Fragen dieses Fragebogens stärker das didaktische Konzept im Blick hatten. Daher beziehe ich mich im Folgenden vor allem auf die Werte bei „Binomial Heap“.

Mit einer Differenz von nur 0,13 und 0,17 wurden die Punkte zur grafischen Benutzungsoberfläche und zur Integration von Aufgaben mit Blick auf das Konzept bewertet und auch recht positiv eingeschätzt (Wertung 1,78 und 2,07).

Auf die Bewertung der Fragen zur Aufteilung in Funktionen, zur grafischen Darstellung der Datenstruktur und dem Lernen über Experimentieren hatten sowohl das Konzept als auch das Lernprogramm Einfluss, denn die Differenz zwischen den Bewertungen liegt zwischen 0,32 und 0,35. Auch diese Punkte des Konzepts wurden recht positiv eingeschätzt (Wertungen von 1,93 bis 2,26).

Die Bewertung der SALA-Modi (Übung, Animation, Simulation) wurde hingegen stark vom Lernprogramm beeinflusst. Die Differenz zwischen „Heapsort“ und „Binomial Heap“ liegt hier zwischen 0,37 und 0,85. Betrachtet man wieder die Wertung bei „Binomial Heap“ als Indikator für die Wichtigkeit der einzelnen SALA-Punkte, so sieht man, dass die Modi recht ähnlich bewertet wurden (1,94 bis 2,26). Innerhalb dieses Rahmens ist eine leichte Abstufung zu sehen. Die Wertschätzung steigt von Animation über Übung hin zur Simulation. Hingegen schnitten bei „Heapsort“ Simulation und Übung sehr schlecht ab. Möglicherweise überforderte dort die Simulation die Nutzer. Bei der reinen Animation vermissten die Nutzer möglicherweise die Möglichkeit zur Interaktion mit dem Algorithmus. In diesem Fall schnitt die Übung am besten ab, da der Schwierigkeitsgrad angemessen war und die Nutzer aktiv lernen konnten.

Zusammenfassend kann man vermuten, dass bei einem ausgewogen gestalteten Lernprogramm die Wertschätzung der Nutzer von der Animation über die Übung zur Simulation hin steigt.

Der Punkt „zusätzliche Erklärungen“ war nur bei dem Fragebogen zu „Binomial Heap“ abgefragt worden, da nur in diesem Lernprogramm ausführliche Erklärungstexte integriert waren.

7.5 Auswertung der Anmerkungen auf den Fragebögen

Auf den Fragebögen war ein Feld für freie Anmerkungen der Nutzer vorgesehen. Ein weiteres Feld stand für die Beschreibung technischer Probleme zur Verfügung. Die meisten Nutzer haben davon Gebrauch gemacht und Anmerkungen notiert, so dass 60 der 88 Fragebögen Anmerkungen enthalten. Die Transkription der Anmerkungen aus den Fragebögen zu „Heapsort“ findet sich im Anhang (Kap. 10.6).

Da die Anmerkungen für Außenstehende oft schwer verständlich sind, habe ich sie im Folgenden mit eigenen Worten umschrieben und dabei versucht die Intention des Originals zu ermitteln und auszudrücken. Die Aussagen habe ich in folgende Kategorien eingeordnet:

- *Technik* (Technische Probleme mit Applets, Webseiten und dem Webbrowser)
- *Präsentation* (Layout der Webseiten und Applets, Bedienung der Applets)
- *Didaktik* (Verständlichkeit des Lernmaterials, Lernerlebnis)

Da es in der Kategorie Didaktik sehr viele Anmerkungen gab, habe ich dort Unterkategorien definiert. Gleichartige Aussagen habe ich zusammengefasst und die Anzahl der Aussagen in Klammern notiert.

Studierende und Lehrende hatten ähnliche Anmerkungen zu Technik und Präsentation, so dass in diesen Kategorien die Anmerkungen beider Gruppen zusammen besprochen werden. Dagegen haben Lehrende in der Regel ein besseres Fachwissen zum Lehrstoff und mehr Erfahrung mit didaktischen Fragen. Daher werden in der Kategorie Didaktik die Anmerkungen von Lehrenden und Studierenden getrennt besprochen.

Die Anmerkungen der Nutzer geben z.B. wieder, wo sich die Nutzer an den Programmen geirrt haben und welche Veränderungen sie sich wünschen. Darunter sind auch wertvolle Anregungen für eine Weiterentwicklung der Lernprogramme und der Methode SALA. Diese Anregungen wurden zum Teil schon in der neuen Version des Lernprogramms zu Heapsort und in der aktuellen Darstellung von SALA berücksichtigt.

Wo es mir sinnvoll erscheint, nehme ich kurz Stellung zu den Anmerkungen der Nutzer. Globale Einschätzungen der Lernprogramme und der Didaktik sehe ich als persönliche Meinungsäußerung der Nutzer und kommentiere sie daher nicht.

7.6 Anmerkungen der Lehrenden auf den Fragebögen

Heapsort – Didaktik

Anmerkungen der Nutzer	Stellungnahmen zur Anmerkung
<ul style="list-style-type: none"> • Unterschied zwischen den Modi Simulation und Übung ist nicht klar. (2x) 	<p>Der Unterschied zeigt sich, wenn mehrere Wege zur Lösung führen, wie bei Build-Heap und Heapsort-Locally (vgl. Kap. 4.5). Dann lässt das Applet im Modus Übung nur den einen Weg des Standardverfahrens zu, während im Modus Simulation alle Lösungswege erlaubt sind. In beiden Modi werden die Schritte aber auf die gleiche Weise ausgeführt, so dass die Benutzungsoberfläche gleich ist. Der gewählte Modus ist über die Radio-buttons und die Helligkeit des Hintergrunds der Ablaufsteuerung abzulesen.</p>
<ul style="list-style-type: none"> • Die Aufgaben prüfen nicht den gesamten Stoff ab. • Lernprogramm behandelt nur die Funktionalität des Algorithmus, nicht die Komplexität. 	<p>Effizienz und Korrektheit des Algorithmus werden weniger intensiv behandelt als der Ablauf. Die Modularisierung und die Vorbedingungen der Hilfsfunktionen sind aber bereits ein Beitrag zur Analyse der Effizienz und zum Beweis der Korrektheit.</p>
<ul style="list-style-type: none"> • Für verschiedene Lernertypen sollte es verschiedenes Lernmaterial geben. 	
<ul style="list-style-type: none"> • Die bottom-up-Reihenfolge der Funktionen ist hilfreich um die Verfahren zu verstehen, besonders bei dem komplizierteren Binomial Heap. Sie hindert aber die Studierenden daran, den Algorithmus selbst (in klassischer top-down-Weise) zu entwickeln. 	<p>Ich denke nicht, dass die Reihenfolge, in der die Funktionen behandelt werden, einen wesentlichen Einfluss darauf hat, ob die Lerner den Algorithmus selbst entwickeln können. Ich denke, dass es gerade die große Stärke von SALA-Lernprogrammen ist, dass sie die Lerner in die Rolle eines Algorithmenentwicklers stellen. Die Reihenfolge der Funktionen im Lernprogramm wurde aber aus anderen Gründen inzwischen zu einer top-down-Reihenfolge verändert (vgl. Kap. 4.4).</p>

Binomial Heap – Didaktik

Nur ein Lehrender schrieb auf den Fragebogen zu „Binomial Heap“ eine Anmerkung zur Didaktik. Er verwies dabei auf seine Anmerkungen zu „Heapsort“ zu den Punkten „bottom-up-Reihenfolge“ und „nur Funktionalität, nicht Komplexität“.

7.7 Anmerkungen der Studierenden zu Heapsort

Anmerkungen zu: Heapsort –Technik

Anmerkungen der Nutzer	Stellungnahmen zur Anmerkung
<ul style="list-style-type: none"> • Installation der Swing-Klassen war schwierig oder misslang. (8x) • Applets liefen erst nach einigen Versuchen. (2x) • Applets funktionierten mit einem bestimmten Browser oder Betriebssystem nicht. (16x) 	Da Swing nun automatisch eingebunden wird, sollten diese Probleme nicht mehr so massiv auftreten (vgl. Kap. 6.4, S. 60).
<ul style="list-style-type: none"> • Die Applets laden langsam oder laufen langsam. (2x) 	Auf modernen Rechnern mit schneller Internetanbindung laden und starten die Applets in weniger als einer Sekunde. Hingegen benötigte z.B. ein älterer Apple Macintosh etwa eine Minute um ein Applet zu laden und anzuzeigen. Auf allen getesteten Plattformen liefen die Applets ausreichend schnell.
<ul style="list-style-type: none"> • Offline-Version wäre wünschenswert. 	Eine Offline-Version war vorhanden. Sie war über die Download-Seite erhältlich.

Heapsort - Präsentation und Bedienung

<ul style="list-style-type: none"> • Animation läuft zu schnell. (3x) 	Dieses Problem trat besonders bei den Applets zu Sort und Heapsort auf. Es wurde in der neuen Version des Lernprogramms behoben.
<ul style="list-style-type: none"> • Es ist unklar, wie man einen Schlüssel bei Move-Max verschiebt. 	Die Beschriftung der Schaltfläche wurde verändert und zusätzliche Beschreibungen im Text hinzugefügt.
<ul style="list-style-type: none"> • Die grafische Präsentation des Lernprogramms sollte aufgepeppt werden. (2x) 	
<ul style="list-style-type: none"> • Die Quadrate der Schlüssel sind, sobald sie kleiner werden, schwer zu erkennen. 	

Heapsort – Didaktik – Details des Lernprogramms

<ul style="list-style-type: none"> • Es ist unklar, warum bei Move-Max das größte Element in den rechtesten Knoten der Ergebnisliste gehört. (2x) 	Zusätzliche Beschreibung und Verweis auf Zusatzinformationen wurde hinzugefügt.
<ul style="list-style-type: none"> • Der Begriff „impliziter Parameter“ ist nicht gut genug erklärt. 	Wird nun ausführlicher erklärt.

Heapsort – Didaktik – Applets allgemein

<ul style="list-style-type: none"> • Auch indirekt aufgerufene Hilfsfunktionen sollten in der Statistik mitgezählt werden. 	Die bisherige Statistik gibt die Benutzerinteraktion wieder. Es könnte daher die Benutzer verwirren, wenn die indirekt aufgerufenen Hilfsfunktionen auch mitgezählt werden. Es könnte aber sinnvoll sein, zusätzlich auch die Basisoperationen „Vergleich“ und „Vertauschen“ zu zählen.
<ul style="list-style-type: none"> • Es sollten nur die benötigten Hilfsfunktionen angeboten werden. 	Wird jetzt so gemacht (vgl. Kap. 4.4.3, S. 43).
<ul style="list-style-type: none"> • Die Schlüssel in den Knoten sollten sichtbar sein. 	Die Schlüssel sind absichtlich verborgen. Bei Heapsort-Local werden die Schlüssel erst nach einem Vergleich der Knoten sichtbar, da der Algorithmus erst dann die Schlüsselwerte kennt. Bei den anderen Funktionen hängt der Ablauf nicht mehr (direkt) von den Schlüsselwerten ab. Dort sind dann andere Informationen, wie z.B. die Heap-Ordnung wichtig.
<ul style="list-style-type: none"> • Unterschied zwischen den Modi Simulation und Übung ist nicht klar oder ist zu gering. (4x) 	(Siehe Stellungnahme auf Seite 79)
<ul style="list-style-type: none"> • Die Hilfetexte im Modus Üben sind manchmal irreführend. 	Die Hilfetexte wurden überarbeitet.
<ul style="list-style-type: none"> • Bei den Funktionen, wie z.B. MoveMax oder Sort sind Unterschiede zum Teil schwer ersichtlich. 	Tatsächlich ist die Visualisierung der Daten in den genannten Fällen gleich, auch wenn unterschiedliche Aufgaben auszuführen sind. Möglicherweise lässt sich das Problem beheben, wenn man wie bei OPSIS nur abstrakte Zustände der Daten anzeigt (vgl. Kap. 3.2, S. 26).
<ul style="list-style-type: none"> • Ablauf einer Funktion sollte als Tabelle der Schritte angezeigt werden. 	Eine Liste der ausgeführten Schritte erscheint, wenn man längere Zeit auf den „Undo“-Button drückt. Zusätzlich wird nun bei jedem Applet im Modus Animation der Pseudocode angezeigt.
<ul style="list-style-type: none"> • Der komplette Sortierablauf kann doch innerhalb eines einzigen Applets ablaufen. Man könnte so auf seine eigenen Teilschritte aufbauen. 	Nein, das Gegenteil ist der Fall. Man hätte viel mehr Schritte auszuführen, wenn man auf die funktionale Abstraktion verzichten und nur mit Basisfunktionen arbeiten würde (vgl. Kap. 4.4.2, S. 42).

Heapsort – Didaktik – Zulässigkeit von Schritten in den Applets

<ul style="list-style-type: none"> • Manchmal erkennt das Applet eine korrekte Schrittfolge nicht als Lösung an. (2x) 	Es gab hier Fehler in den Applets, die inzwischen behoben wurden.
<ul style="list-style-type: none"> • Simulation sollte nur Algorithmen-konforme Schritte zulassen. (2x) 	Dafür gibt es den Modus Übung (vgl. Kap. 4.5).

Heapsort – Didaktik – Zulässigkeit von Schritten in den Applets (Forts.)

<ul style="list-style-type: none"> • Simulation sollte auch unzulässige Schritte ausführen. 	Das widerspricht einem zentralen Prinzip von SALA. Man darf eine Hilfsfunktion nur aufrufen, wenn ihre Vorbedingung erfüllt ist. Dadurch soll sich der Lerner die Vorbedingungen besser bewusst machen. Er wird auch davor bewahrt, eine falsche Lösungen weiterzuverfolgen (vgl. Kap. 4.5, S. 45).
<ul style="list-style-type: none"> • Im Modus Simulation konnten Schritte nicht ausgeführt werden. 	Vermutlich waren die Vorbedingungen der Hilfsfunktionen nicht erfüllt (vgl. Kap. 4.5).
<ul style="list-style-type: none"> • Im Modus Übung sollten auch grundlegende Hilfsfunktionen verwendet werden dürfen. 	Dies widerspricht dem Prinzip der Abstraktion/Modularisierung (vgl. Kap. 4.5).

Heapsort – Didaktik – Ausführlicherer tutorieller Teil

<ul style="list-style-type: none"> • Die Funktionsweise des Lernprogramms sollte detaillierter erklärt werden. 	Wird nun detaillierter erklärt.
<ul style="list-style-type: none"> • Vertiefungen sind zu spärlich und nicht hilfreich. • Mehr erklärender Text ist nötig. • Das Lernprogramm ist aus sich heraus nicht verständlich. Man braucht noch die Vorlesung. • An manchen Stellen ist die Beschreibung im Lernprogramm unverständlich. (2x) 	Die neue Version hat nun mehr erklärenden Text und Bilder.
<ul style="list-style-type: none"> • Zu den Aufgaben sollte es eine Musterlösung geben. 	Ist nun integriert durch die Pseudocode-Anzeige im Modus Animation und einen Lehrtext zur Musterlösung am Ende jeder Funktionsseite.

Heapsort – Didaktik – Gesamtbewertungen

<ul style="list-style-type: none"> • Macht den Algorithmus sehr anschaulich. 	
<ul style="list-style-type: none"> • Motiviert sehr, weil kleinere Unsicherheiten schnell überprüft werden können, zum Beispiel Ablaufreihenfolgen durch den Modus Übung. 	
<ul style="list-style-type: none"> • Das Niveau ist zu niedrig, so dass man zum Durcharbeiten zu lange braucht. 	
<ul style="list-style-type: none"> • Das Konzept hinter dem Lernprogramm ist gut, aber das Lernprogramm hat Schwächen. (5x) 	

7.8 Anmerkungen der Studierenden zu Binomial Heap

Anmerkungen zu: Binomial Heap –Technik

Anmerkungen der Nutzer	Stellungnahmen zur Anmerkung
<ul style="list-style-type: none"> • Technische Probleme mit der Anzeige einzelner Webseiten. (4x) • Mauszeiger verschwindet teilweise in den Applets. 	Dies scheinen Einzelfälle zu sein, deren Ursache nicht im Lernprogramm, sondern im Computersystem des Nutzers liegt.
<ul style="list-style-type: none"> • Einbinden von swingall.jar war problematisch, hat aber dann funktioniert. 	Die Datei swingall.jar wird für dieses Lernprogramm nicht benötigt.
<ul style="list-style-type: none"> • Der Internetanschluss unseres Studentenwohnheims fiel immer wieder aus. 	Dann empfiehlt sich die lokale Installation der Offline-Version.
<ul style="list-style-type: none"> • Das Programm ist besser als das zu Heapsort: Es gab keine Fehlermeldungen. Die Ladezeiten waren in Ordnung. • Man konnte endlich mal sofort von zu Hause arbeiten, ohne jegliche Anpassung. 	

Binomial Heap – Präsentation und Bedienung

<ul style="list-style-type: none"> • Bedienung teilweise etwas umständlich. 	
<ul style="list-style-type: none"> • Bedien- und Anleitungstexte sind im Applet größenmäßig nicht gut angepasst. 	
<ul style="list-style-type: none"> • Grafische Benutzungsoberfläche könnte ansprechender gestaltet werden bezüglich Hintergrundfarbe, Farben allgemein (2x) 	
<ul style="list-style-type: none"> • Animationen laufen zu schnell. 	

Binomial Heap – Didaktik – Einzelne Applets

<ul style="list-style-type: none"> • Das Applet „Verschmelzen“ sollte eine Animation und Undo/Redo bieten. 	Das sehe ich auch so. Bei der Entwicklung des Lernprogramms stand SALABIM noch nicht zur Verfügung. Die Diplomarbeit, in der das Programm entstanden ist, ist schon abgeschlossen, eine Überarbeitung des Lernprogramms ist momentan nicht geplant.
<ul style="list-style-type: none"> • Applet „Code erkennen“ sollte statt XXXXX aussagekräftige Bezeichner enthalten 	Nein, das ist absichtlich so. Es ist gerade der Sinn der Aufgabe, dass der Nutzer den Namen der Funktion ermitteln soll. Daher wird der Bezeichner erst angezeigt, wenn der Nutzer den richtigen Namen gewählt hat.
<ul style="list-style-type: none"> • Die Code-Erkennung ist eine sehr gute Idee. 	

Binomial Heap – Didaktik – Einzelne Applets

<ul style="list-style-type: none"> • Das „Binomial Heap“ Applet hilft die gewonnenen Kenntnisse zu überprüfen und zu festigen. • Applet „Binomial Heap“ sollte alle Funktionen zur Verfügung stellen. Die „speziellen“ Applets bringen mehr. 	Das Applet Binomial Heap veranschaulicht die Entwicklung der Daten nach Ausführen der Operationen. Die anderen Applets dienen dem Erlernen des Algorithmus.
<ul style="list-style-type: none"> • Das „Binomial Heap“ Applet sollte es erlauben beliebige Schlüssel zu entfernen. (2x) 	Ja, das wäre wünschenswert.
<ul style="list-style-type: none"> • Das „Binomial Heap“ Applet sollte Zwischenergebnisse anzeigen. 	Nein. Dafür sind die anderen Applets da.
<ul style="list-style-type: none"> • Trennen der Bäume vor dem Löschen (=Baum entfernen) irritiert etwas. 	Die Beschriftung der Buttons sollte verändert werden.

Binomial Heap – Didaktik – Allgemeines zu den Applets

<ul style="list-style-type: none"> • Es sollten nur Buttons da sein, die auch gebraucht werden. (2x) 	Es werden absichtlich überflüssige Buttons angeboten, um die Lösung zu erschweren, denn die Aufgaben sollen einen angemessenen Schwierigkeitsgrad haben.
<ul style="list-style-type: none"> • Applets sollten sagen, ob es einen besseren Lösungsweg gibt. 	Ja.
<ul style="list-style-type: none"> • Die integrierten Aufgaben sind zu leicht, man kann sie fast ohne Hintergrundwissen lösen. 	
<ul style="list-style-type: none"> • Direkte Erklärung neben den durchgeführten Schritten wäre noch sinnvoller. 	Ja.

Binomial Heap – Didaktik – Gesamteinschätzungen

<ul style="list-style-type: none"> • Sehr hilfreich! Eine sehr gute Methode neue Dinge zu lernen. 	
<ul style="list-style-type: none"> • Besser als Heapsort-SALA, sowohl von der grafischen Oberfläche als auch vom Verständnis her (und, da es komplett in Deutsch ist). 	
<ul style="list-style-type: none"> • Das Programm sieht solider und professioneller aus als das zu Heapsort. Die einzelnen Schaltflächen/Funktionen werden gut erklärt. Es gibt im allgemeinen mehr zu lesen. 	
<ul style="list-style-type: none"> • Besser gelungen als das Programm zu Heapsort, wegen ausführlicher Erklärungen. 	

Binomial Heap – Didaktik – Sonstiges

<ul style="list-style-type: none"> • Kapitel 5 sollte interaktive Bestandteile bekommen. 	Wäre schön, ist aber nicht zwingend.
<ul style="list-style-type: none"> • Realistisches Beispiel zur Anwendung als Applet hinzufügen. 	Man könnte weitere Anwendungen mit Lehrtext und Bildern zeigen (z.B. Nutzung des Binomial Heap als Prioritätenwarteschlangen bei der Suche kürzester Wege in Graphen nach Dijkstra). Applets zur Anwendung würden den Rahmen des Lernprogramms aber sprengen.
<ul style="list-style-type: none"> • Aufgaben teilweise zu einfach. 	
<ul style="list-style-type: none"> • Wenn man Deutsch nicht gut versteht, versteht man auch das Lernprogramm nicht so gut. 	Heapsort gibt es inzwischen auch auf Englisch. Bei Binomial Heap sind erst die Applets ins Englische übersetzt. Eine Übersetzung des Lehrtextes wäre wünschenswert, ist aber leider recht zeitaufwändig.
<ul style="list-style-type: none"> • Funktionen/Abschnitte etwas detaillierter/praxisorientierter darstellen. 	

Binomial Heap – Sonstiges

<ul style="list-style-type: none"> • Die Evaluationsfragen zum didaktischen Konzept sind einseitig gestellt. Es ist doch klar, dass diese Punkte gut ankommen. 	(vgl. Kap. 7.4.9, S. 77)
<ul style="list-style-type: none"> • Hinweise auf andere Lernprogramme aufnehmen. 	Das Programm wurde inzwischen ins OLLI-Archiv eingefügt. Dort finden sich auch Verweise auf weitere Lernprogramme.

8 Analyse der Nutzung zweier SALA-Lernprogramme

8.1 Einführung

In Kapitel 7 wurde die Erprobung zweier nach SALA gestalteter Lernprogramme beschrieben, bei der Studierende die Lernprogramme und das dahinter stehende didaktische Konzept auf Fragebögen bewerteten. Die Nutzung der Lernprogramme wurde während der Erprobung von einem Webserver protokolliert. In diesem Kapitel werden die Nutzungsdaten ausgewertet. Dadurch wird ermittelt, wie die Studierenden durch die Lernprogramme navigiert sind, wie intensiv sie die Programme genutzt haben und wie das Verhältnis von Online-

zu Offline-Nutzung war. Dazu wird zunächst die Struktur der Lernprogramme vorgestellt, da sie die Navigationsmöglichkeiten der Nutzer vorgibt. Es wird beschrieben, wie die Daten aufbereitet und in Diagrammen dargestellt wurden. Anschließend werden die Diagramme der Nutzungsdaten interpretiert, um das Navigationsverhalten der Nutzer festzustellen. Die Diagramme liefern auch die Informationen, um die Intensität der Nutzung und das Verhältnis von Online- zu Offline-Nutzung zu analysieren.

8.2 Struktur der Lernprogramme

Die Lernprogramme „Heapsort“ und „Binomial Heap“ wurden nach SALA gestaltet und sind daher ähnlich einem Lehrbuch strukturiert. Der eigentliche Lerninhalt ist hierarchisch gegliedert. Von einer Seite können alle hierarchisch darunter liegenden Seiten angesteuert werden. Die Programme bestehen jeweils aus rund 20 HTML-Seiten.

Mit *Seitenstruktur* bezeichne ich hier die Struktur, mit der ein Autor ein Lernprogramm in Seiten gliedert und diese mit Hyperlinks verbunden hat. Abbildung 42 zeigt die Seitenstruktur des Lernprogramms „Heapsort“ zum Zeitpunkt der Evaluation (zur jetzigen Fassung vgl. Kap. 4). Abbildung 43 zeigt die (auch heute noch gültige) Seitenstruktur des Lernprogramms zu „Binomial Heap“.

Notation zur Seitenstruktur

Die Diagramme verwenden grafische Symbole, um die Seiten, Blöcke und Hyperlinks eines Lernprogrammes darzustellen. Dabei werden die *Seiten* des Lernprogramms als Ellipsen dargestellt. Die Seiten lassen sich nach logischen Gesichtspunkten zu *Blöcken* gruppieren. Blöcke, die aus mehreren Seiten bestehen, sind in der Abbildung durch graue ausgefüllte Rechtecke dargestellt. Seiten, die nicht in einem solchen Rechteck liegen, bilden jeweils einen eigenen Block.

Gibt es die Möglichkeit, von einer Seite über einen Hyperlink zu einer zweiten Seite zu gelangen, so sind ihre Ellipsen durch einen Pfeil verbunden. Gelangt man von einer Seite zu allen Seiten eines Blocks, so sind Seite und Block ebenso mit einem Pfeil verbunden. Seiten und Hyperlinks auf dem sogenannten Standardpfad sind grün hervorgehoben (vgl. Kap. 8.2.3, S. 90).

Im folgenden werden die einzelnen Blöcke der Lernprogramme und ihre Aufgabe genauer beschrieben.

8.2.1 Blöcke bei Heapsort

Das Lernprogramm „Heapsort“ bestand zum Zeitpunkt der Evaluation aus folgenden Blöcken:

- Die Seite „*Inhalt*“ ist die Startseite des Lernprogramms und enthält das Inhaltsverzeichnis. Von ihr aus kann man die meisten Blöcke erreichen.
- Das Konzept des entdeckenden Lernens über Aufgaben, nach dem das Lernprogramm gestaltet ist, wird auf der Seite „*Lernkonzept*“ beschrieben.
- Der eigentliche Lerninhalt ist in fünf Kapiteln enthalten. Dabei besteht nur das Kapitel „*Funktionen*“ aus mehreren Seiten. Die Seite „*Funktionenübersicht*“ benennt die einzelnen Funktionen tabellarisch und in einer grafischen Übersicht. Von dort gelangt man zum Block „*Lerninhalt - Funktionen*“ mit den Seiten für jede der 6 Funkti-

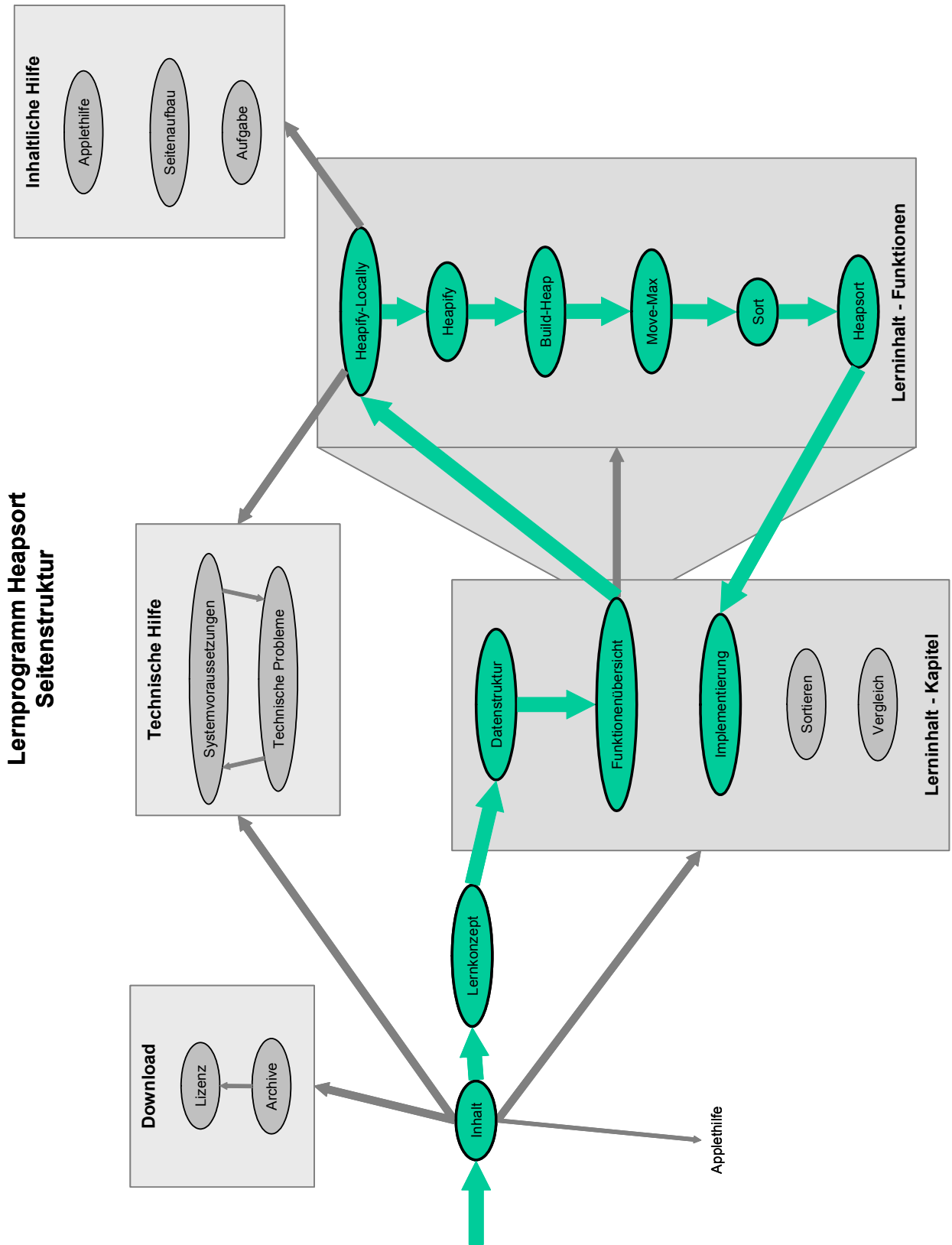


Abb. 42: Seitenstruktur des Lernprogramms zu Heapsort zur Zeit der Evaluation

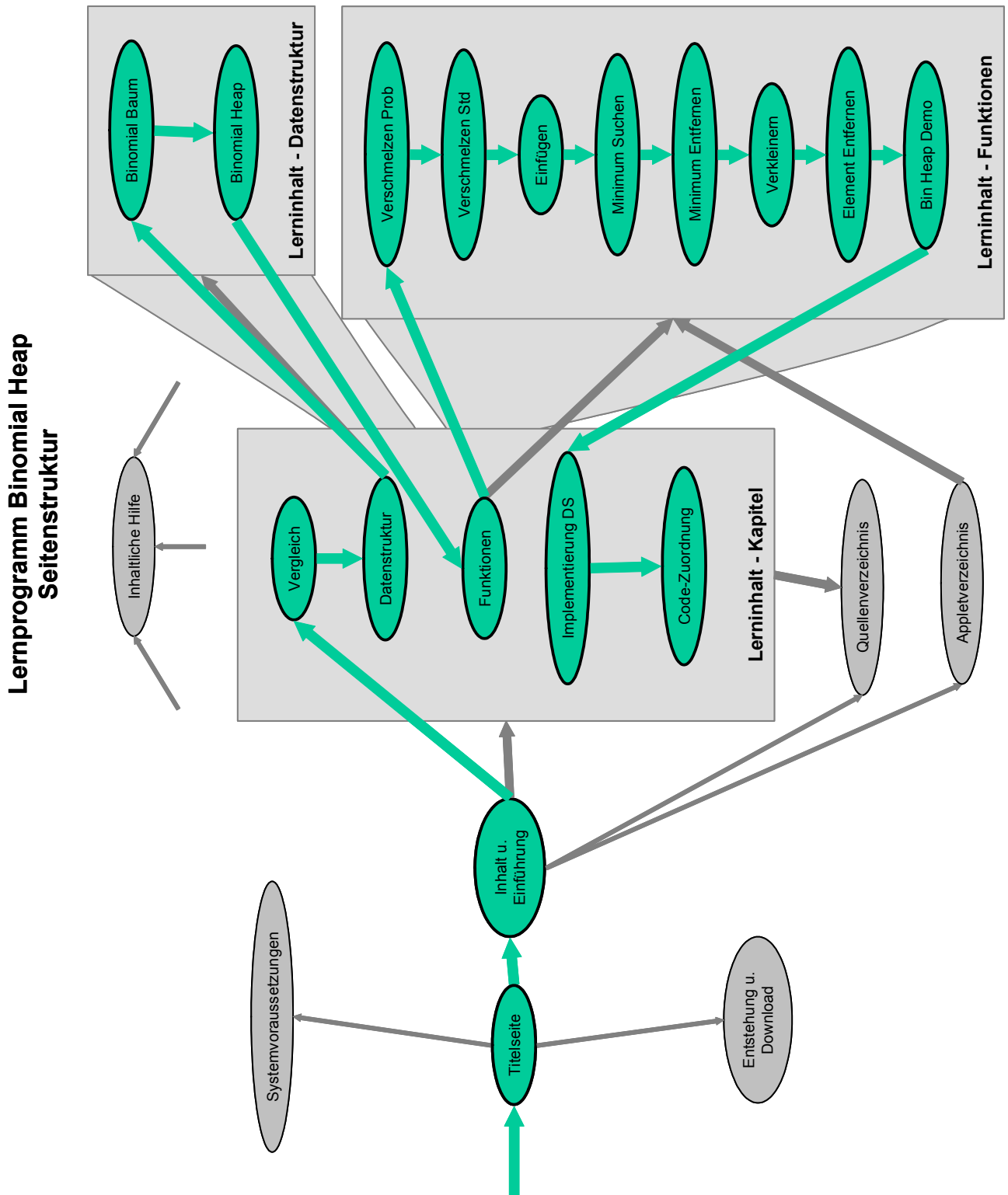


Abb. 43: Seitenstruktur des Lernprogramms zu „Binomial Heap“

onen des Algorithmus. Daher ist im Diagramm der Block durch ein grau gefülltes Trapez mit der Seite „Funktionenübersicht“ verbunden.

- Da das Lernprogramm Java-Swing-Applets enthält, muss eine Java-Bibliothek eingebunden werden. Der Block „*Technische Hilfe*“ beschreibt die Systemvoraussetzungen und gibt Hinweise bei technischen Problemen.
- Fragen zur Bedienung der Aufgaben (Applets), dem Aufbau der Aufgaben-Seiten und der Herangehensweise an die Aufgaben beantwortet der Block „*Inhaltliche Hilfe*“.
- Für das Herunterladen des Lernprogramms in einer Offline-Version gibt es den Block „*Download*“.

8.2.2 Blöcke bei Binomial Heap

Das Lernprogramm „Binomial Heap“ besteht aus folgenden Blöcken:

- Die Startseite „*Titelseite*“ enthält nur die Verzweigung zu den Seiten „Systemvoraussetzungen“, „Entstehung und Download“ und „Inhalt und Einführung“.
- Die Seite „*Inhalt und Einführung*“ enthält das Inhaltsverzeichnis. Von ihm aus sind alle Seiten außer den obigen direkt erreichbar. Die Seite enthält auch schon Lerninhalte zur Problemstellung sowie einen Überblick über das Lernprogramm und Informationen zum didaktischen Konzept.
- Der Lerninhalt besteht aus den Kapiteln „*Vergleich*“, „*Datenstruktur*“, „*Funktionen*“, „*Implementierung der Datenstruktur*“ und „*Code-Zuordnung*“. Die Kapitel „*Datenstruktur*“ und „*Funktionen*“ bestehen aus mehreren Seiten.
- Verweise im Lerninhalt auf Literatur und andere Quellen gehen ins „*Quellenverzeichnis*“.
- Vom „*Appletverzeichnis*“ kann man die Applets des Lernprogramms gezielt ansteuern.
- Die Seite „*Inhaltliche Hilfe*“ erklärt die Navigation und den Seitenaufbau.

8.2.3 Standardpfad

Die Lernprogramme „Heapsort“ und „Binomial Heap“ erlauben es, frei entlang der Hyperlinks durch das Programm zu navigieren. Der Nutzer kann sich selbst für eine Auswahl der Seiten und eine Reihenfolge des Lesens entscheiden. Anfänger sind damit aber oft überfordert. Daher ist es sinnvoll, wenn ein Autor hier einen Weg vorschlägt. Er wählt dazu einen Teil der Seiten aus, die der durchschnittliche Lerner lesen und durchar-

beiten soll und ordnet sie in einer Reihenfolge an, die das Lernen erleichtert. Diesen Weg durch das Lernprogramm bezeichne ich hier als *Standardpfad*.

Für die beiden Lernprogramme haben die Autoren jeweils einen Standardpfad angelegt und sich dabei an SALA orientiert (Kap. 4).

Standardpfad bei Heapsort

Im Lernprogramm „Heapsort“ führt der Standardpfad über Inhaltsverzeichnis und Lernkonzept zu den Kernkapiteln Datenstruktur, Funktionen und endet bei der Implementierung der Datenstruktur. Die Kapitel Sortieren und Vergleich sind nicht Teil des Standardpfades, sondern dienen der Vertiefung.

In der neuen Version des Lernprogramms „Heapsort“ wurde die Reihenfolge der Funktionsseiten umgekehrt, da einige Nutzer Probleme mit der Abstraktion von Hilfsfunktionen hatten (vgl. Kap. 4.4.3, S. 43).

Standardpfad bei „Binomial Heap“

Bei „Binomial Heap“ führt der Standardpfad von der Titelseite über das Inhaltsverzeichnis und die Einführung durch alle Kapitel des Lerninhaltes. Es gibt also keine Seiten, die explizit zur inhaltlichen Vertiefung dienen.

Navigationselemente auf der Webseite

Auf den Seiten des Standardpfades ist unten eine Navigationsleiste enthalten. Sie enthält jeweils einen Button, um zur nächsten Seite, vorhergehenden Seite oder zum Inhaltsverzeichnis zu gelangen (Abb. 20). Bei Binomial Heap ist die Navigationsleiste am oberen und unteren Seitenrand angebracht und enthält auch einen Button, mit dem man zur Hilfeseite gelangt.

8.2.4 Verzweigungspunkte für die freie Navigation

Einige Seiten der Lernprogramme sind bewusst als *Verzweigungspunkte* gestaltet, so dass man je nach Interesse und Informationsbedarf von ihnen aus mehrere Seiten erreichen kann.

Verzweigungspunkte bei Heapsort

Das Lernprogramm „Heapsort“ enthielt zum Zeitpunkt der Evaluation folgende Verzweigungspunkte:

- Die Seite „*Inhalt*“ ist die Startseite des Lernprogramms und enthält das Inhaltsverzeichnis. Von ihr aus kann man die Blöcke „Download“, „Technische Hilfe“, „Lernkonzept“ und die „Lerninhalt-Kapitel“ erreichen. Die Blöcke „Lerninhalt-Funktionen“ und „Inhaltliche Hilfe“ wurden hier nicht aufgenommen, da sie am Anfang noch nicht benötigt werden. Zu viele Einträge hätten das Inhaltsverzeichnis überfrachtet.
- Wie erwähnt erreicht man von der „*Funktionenübersicht*“ jede der Funktionsbeschreibungen. Dies ist für einen zweiten Lerndurchgang wichtig, da die Funktionen stark voneinander abhängen. Auch zum Nachschlagen einzelner Daten lässt sich solch eine Seite nutzen.
- Auf der Seite „*Heapify-Locally*“ wird die erste Funktion vorgestellt und die erste Aufgabe bearbeitet. Daher ist dort ein Abschnitt mit Verzweigungen zur technischen Hilfe und inhaltlichen

Hilfe eingebaut. Bei den anderen Funktionen wurde, um deren Seiten nicht aufzublähen, dieser Abschnitt nicht wiederholt, sondern nur per Hyperlink darauf verwiesen.

Verzweigungspunkte bei „Binomial Heap“

Das Lernprogramm „Binomial Heap“ hat ähnliche Verzweigungspunkte wie „Heapsort“. Die Seiten „Titelseite“ und „*Inhalt und Einführung*“ übernehmen zusammen die Rolle des Inhaltsverzeichnisses. Die Seite „*Funktionen*“ dient wie beim Lernprogramm „Heapsort“ zum Nachschlagen einzelner Funktionen. Bei „Heapsort“ gibt es von der Seite mit dem ersten Applet aus Verzweigungsmöglichkeiten zur technischen und inhaltlichen Hilfe. Bei „Binomial Heap“ gibt es dort aber nur den auch sonst verfügbaren Verweis auf die Hilfeseite zur Navigation.

8.3 Ermittlung der Nutzungsdaten

Die Analyse der Nutzung der Lernprogramme baut auf Grunddaten auf, die aus den Logdateien des Webservers gewonnen wurden. Zunächst wird zwischen Nutzern unterschieden, die die Lernprogramme heruntergeladen und offline ausgeführt haben und solchen, die online über das Internet mit den Programmen gearbeitet haben. Für die Online-Nutzer wurde für jede Seite und jeden Link der Programme ermittelt, wie viele Studierende diese genutzt haben. Es wird nun beschrieben, wie die Lernprogramme technisch beschaffen sind und wie aus den Logdateien auf die Nutzerzahlen geschlossen wurde.

8.3.1 Technische Basis der Lernprogramme

Webbasiert

Die Lernprogramme sind webbasiert, d.h. sie können mit normalen Webbrowsern aus dem Internet geladen und ausgeführt werden. Die einzelnen Bestandteile sind in den im WWW üblichen Dateiformaten gespeichert. Die Texte sind in HTML verfasst, die Grafiken im GIF-Format abgelegt und die interaktiven Teile als Java-Applets realisiert.

Heapsort benötigt Java-Swing

Beim Lernprogramm „Heapsort“ ist der Java-Teil mit der modernen Sprachversion Java-Swing realisiert. Da die zur Zeit der Evaluation gebräuchlichen Browser Java-Swing noch nicht direkt unterstützten musste die Swing-Bibliothek auf dem Rechner installiert sein (vgl. Kap. 6.4, S. 60). Die Bibliothek war im Rechnerpool der Informatik bereits installiert und in den Webbrowser eingebunden. Wollten die Studierenden das Lernprogramm auf einem anderen Rechner nutzen, so mussten sie dort die Bibliothek selbst installieren. Die Bibliothek konnte von unserem Webserver heruntergeladen werden. Sie war aber auch von anderer Stelle erhältlich, z.B. ist sie im Java-Entwicklungspaket JDK von Sun Microsystems enthalten. Das Lernprogramm „Binomial Heap“ benötigt keine separate Bibliothek.

Online- und Offline-Version

Von den Lernprogrammen gibt es eine Online-Version und eine Offline-Version. Bei der Online-Version werden die Seiten, sobald sie gelesen werden sollen, einzeln vom Webserver abgerufen. Für die Offline-Version wurden die Dateien in ein ZIP-Archiv gepackt. Das Archiv liegt zum Download bereit. Es kann von einem Datenträger oder direkt über das Internet auf einen Rechner kopiert und dort entpackt werden. Man kann dann auf dem Rechner mit dem Lernprogramm arbeiten, ohne dass die Seiten über das Internet angefordert wer-

den müssen. In diesem Fall ist für die Arbeit mit dem Programm keine Internetverbindung nötig und es sind auch keine Onlinegebühren zu zahlen. Auch stehen die Seiten in der Regel schneller zur Verfügung als in der Online-Version. Im Rechnerpool war der Zugriff auf die Online-Version eingestellt.

8.3.2 Auswertung von Logdaten

Die Logdatei wurde jeweils für die Zeit von der Ausgabe des Aufgabenblattes bis zum Tag der Abgabe der Lösung ausgewertet. Für das Lernprogramm „Heapsort“ war dies Donnerstag, der 29. Juni 2000 bis Donnerstag 6. Juli 2000. Für „Binomial Heap“ Donnerstag, 6. Juli 2000 bis Donnerstag 13. Juli.

Leider ermitteln die mir bekannten Auswerteprogramme nicht die Häufigkeit, mit der Links benutzt wurden. Daher ließ ich von einem studentischen Mitarbeiter ein Auswerteprogramm schreiben, das dies ermittelt.

Zahl der Nutzer ermitteln

Der Webserver protokolliert zwar die Seitenzugriffe, es lässt sich aber nicht mehr eindeutig feststellen, welche Person die Seiten angefordert hat. Lediglich die *IP-Adresse des Client-Rechners* wird übertragen und protokolliert. Man kann nun die Zahl der IP-Adressen als Maß für die Zahl der Nutzer einer Website nehmen. Dabei sind aber die folgenden zwei Störfaktoren zu berücksichtigen.

- Die Studierenden haben zumeist in einem Rechnerpool der Universität gearbeitet. Dort können nacheinander mehrere Studierende den gleichen Rechner nutzen. Dann werden weniger IP-Adressen aufgezeichnet, als es Nutzer gibt.
- Umgekehrt kann ein Studierender mehrmals mit dem Programm arbeiten und dabei verschiedene Rechner nutzen. Dann werden mehr IP-Adressen aufgezeichnet, als es Nutzer gibt. Beim Online-Dienst AOL werden sogar während einer Sitzung mehrere IP-Adressen verwendet.

Beide Störfaktoren treten auch auf, wenn sich Studierende über Modem oder ISDN ins Internet einwählen und sie eine dynamische IP-Adresse erhalten. Ich nehme im Folgenden an, dass sich diese Störfaktoren etwa ausgleichen, so dass man aus der Anzahl der IP-Adressen die Zahl der Nutzer schätzen kann.

Neben den Studierenden haben auch andere Nutzer auf die Seiten zugegriffen. Dies betrifft vor allem das Lernprogramm Binomial Heap, da es schon seit längerer Zeit vom Webserver abrufbar ist. In den zehn Tagen vor dem Einsatz des Lernprogramms in der Übung haben rund 3 Rechner pro Tag auf das Lernprogramm zugegriffen.

Es haben sich eine ganze Reihe Studierende die Programme heruntergeladen und dann offline mit ihnen gearbeitet, so dass dann keine Zugriffe im Logfile vermerkt sind.

Da die Programme im Beobachtungszeitraum von den Studierenden recht intensiv genutzt wurden kann man wohl die Zahl der studierenden Nutzer aus der Zahl der IP-Adressen abschätzen.

Linknutzer und Seitennutzer

In den Lernprogrammen führen oft mehrere Links auf eine Seite. Für jeden Link wurde ermittelt, wie viele Nutzer über diesen Link gegangen sind. Ebenso wurde die Anzahl der Nutzer jeder Seite ermittelt. Betrachtet man nun eine Seite und die Links die dorthin führen, so ist die Summe der Linknutzer in der Regel ungleich der Anzahl der Seitennutzer. Dafür gibt es mehrere Gründe:

- Wenn ein Nutzer über verschiedene Links eine Seite mehrmals besucht, wird der Nutzer bei jedem der Links einzeln gezählt, auf der Seite aber nur einmal. Die Nutzer werden auf einer Seite nur einmal gezählt, um die Nutzerzahlen nicht künstlich zu erhöhen.
- Manchmal nennt ein Browser beim Seitenzugriff nicht die Ausgangsseite, so dass nur der Seitenzugriff gezählt werden kann.
- Liegt die Ausgangsseite außerhalb des Lernprogramms, so wird nur der Seitenzugriff gezählt, nicht aber der Linkzugriff.

8.4 Visualisierung der Nutzungsdaten

Aus den Logdateien des Webservers wurde ermittelt, wie oft jede Seite und jeder Link der Lernprogramme genutzt wurde. Hier wird nun beschrieben, wie diese Daten grafisch und numerisch dargestellt werden. Sie sind im Diagramm 44 für das Lernprogramm „Heapsort“ und im Diagramm 45 für „Binomial Heap“ wiedergegeben.

Die grafische Repräsentation der Größen „Seitennutzer“ und „Linknutzer“ in einer Art Karte hat Vorteile gegenüber einer tabellarischen Auflistung, denn sie erleichtert es, interessanten Punkte zu finden und Werte schneller zu vergleichen.

Entstehung der Notation

Da mir kein Diagrammtyp bekannt war, mit dem sich diese Daten darstellen ließen, habe ich selbst eine Notation entwickelt. Angeregt wurde ich zu der Notation durch die historische Infografik Minards, in der er den Russlandfeldzug Napoleons darstellt (zu einer Übersicht über moderne Adaptionen dieser Grafik vgl. [Friendly 2001]). Die Stärke des französischen Heeres notiert Minard über die Breite des Striches. Analog verwende ich die Strichdicke, um die Nutzerzahl eines Links zu visualisieren.

Visualisierung als Graph

Die Notation ist in Abbildung 44 beschrieben. Es lag nahe, hypermediale Lernprogramme als Graphen zu visualisieren. Die Seiten werden als Knoten und die Links als gerichtete Kanten dargestellt. Die Lernprogramme lassen sich jeweils noch gut auf einer Seite darstellen, da sie aus rund 20 Seiten bestehen, die hauptsächlich hierarchisch verlinkt sind.

Nutzung der Seiten

Eine Seite des Lernprogramms wird durch eine Ellipse dargestellt. Sie ist mit dem Seitennamen beschriftet und hat daher eine feste Breite, um den Namen aufnehmen zu können. Die Höhe und damit die Fläche der Ellipse ist proportional zur Anzahl der Nutzer, die die Seite gelesen haben. Es ist

vorteilhaft, die Nutzerzahl über die Fläche zu kodieren, da Menschen die Größe zweier in beide Dimensionen ausgedehnter Objekte hauptsächlich über die Fläche vergleichen. Zusätzlich ist die Anzahl der Nutzer noch als Zahl rechts über der Ellipse notiert. Die Zahlen, die die Anzahl Nutzer notieren, sind grau oder grün (statt schwarz) gehalten, damit sie sich optisch nicht vordrängen.

Seiten, die ein Applet enthalten, sind mit einem Stern nach dem Namen markiert.

Beim Lernprogramm Binomial Heap konnte man bei Seiten mit Applet das Applet zusätzlich zur bereits geöffneten Seite in einem separaten Fenster öffnen. Die Anzahl der Nutzer, die das Applet zusätzlich separat öffneten, ist nach einem Pluszeichen notiert.

Nutzung der Links

Ein Link zwischen zwei Seiten wird durch einen Pfeil dargestellt. Seine Breite ist proportional zur Anzahl Nutzer, die über diesen Link gegangen sind. Die Anzahl steht neben dem Pfeil.

Um das Diagramm nicht zu überfrachten, wurden Links nur eingezeichnet, wenn sie mindestens von vier Nutzern benutzt wurden. Neben den im Abschnitt zu den Nutzungsdaten genannten Gründen (vgl. Kap. 8.3, S. 91) ist dies ein weiterer Grund, warum die Zahl der Linknutzer oft nicht mit der Zahl der Seitennutzer übereinstimmt.

Würde ein Link quer durch andere Elemente gehen, so wird für eine der Seiten ein Platzhalter eingezeichnet und der Pfeil des Links damit verbunden.

Standardpfad

Seiten und Links, die auf dem Standardpfad liegen, sind grün gefärbt, die sonstigen Seiten und Links grau. Es wurde ein ruhiges Grün gewählt, damit man die Größen der Objekte noch gut vergleichen kann, ohne durch den Farbunterschied zu stark abgelenkt zu werden.

Nutzung des Lernprogramms Heapsort
 durch Studierende der Veranstaltung „Datenstrukturen“ im Sommersemester 2000

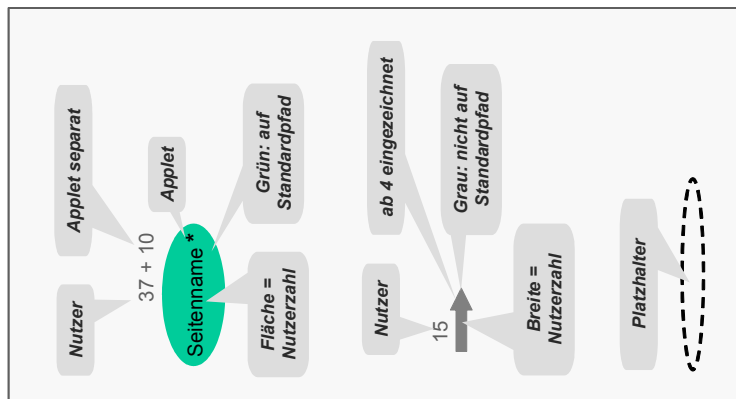
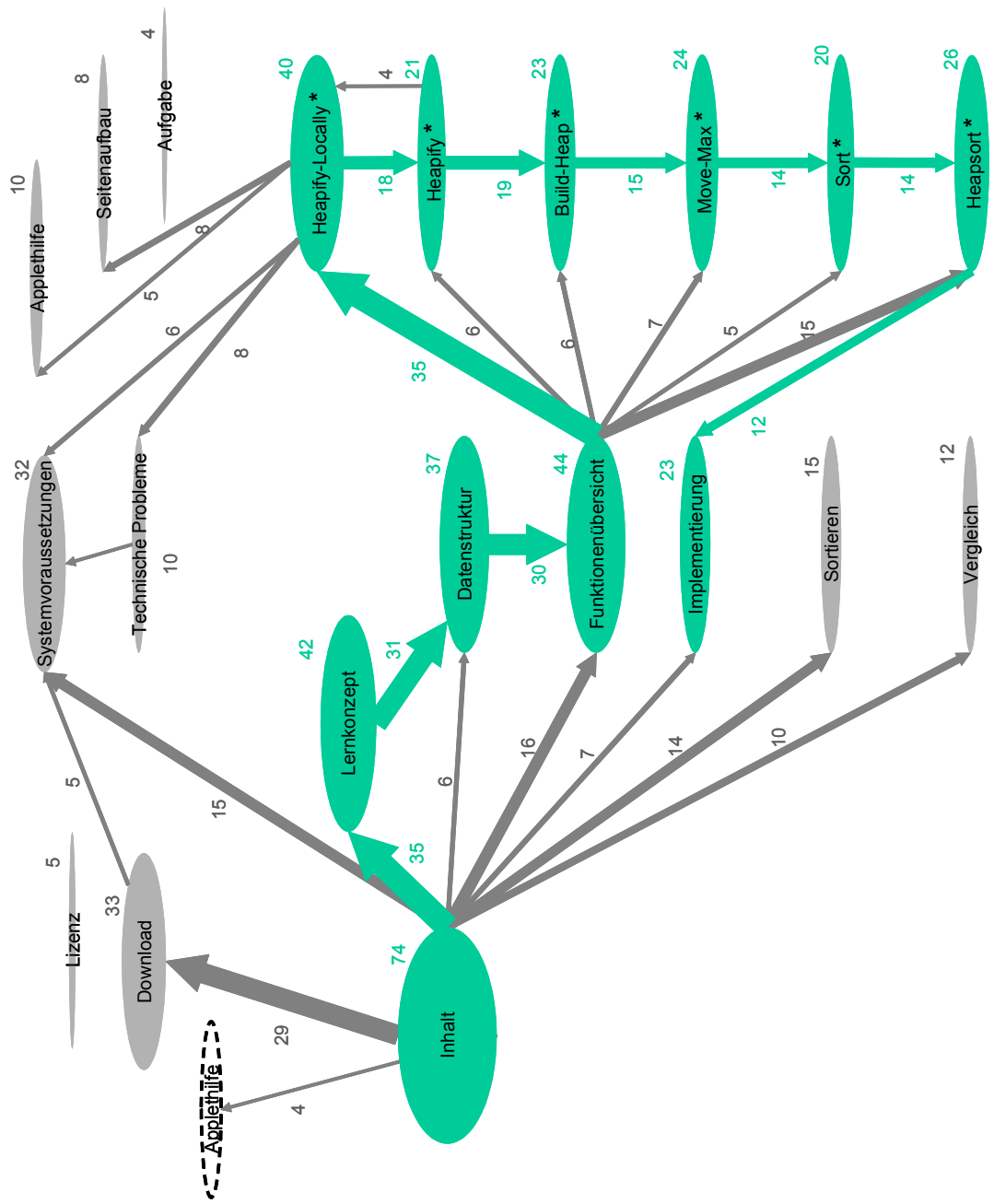


Abb. 44: Nutzung des Lernprogramms Heapsort

**Nutzung des Lernprogramms Binomial Heap
durch Studierende der Veranstaltung „Datenstrukturen“ im Sommersemester 2000**

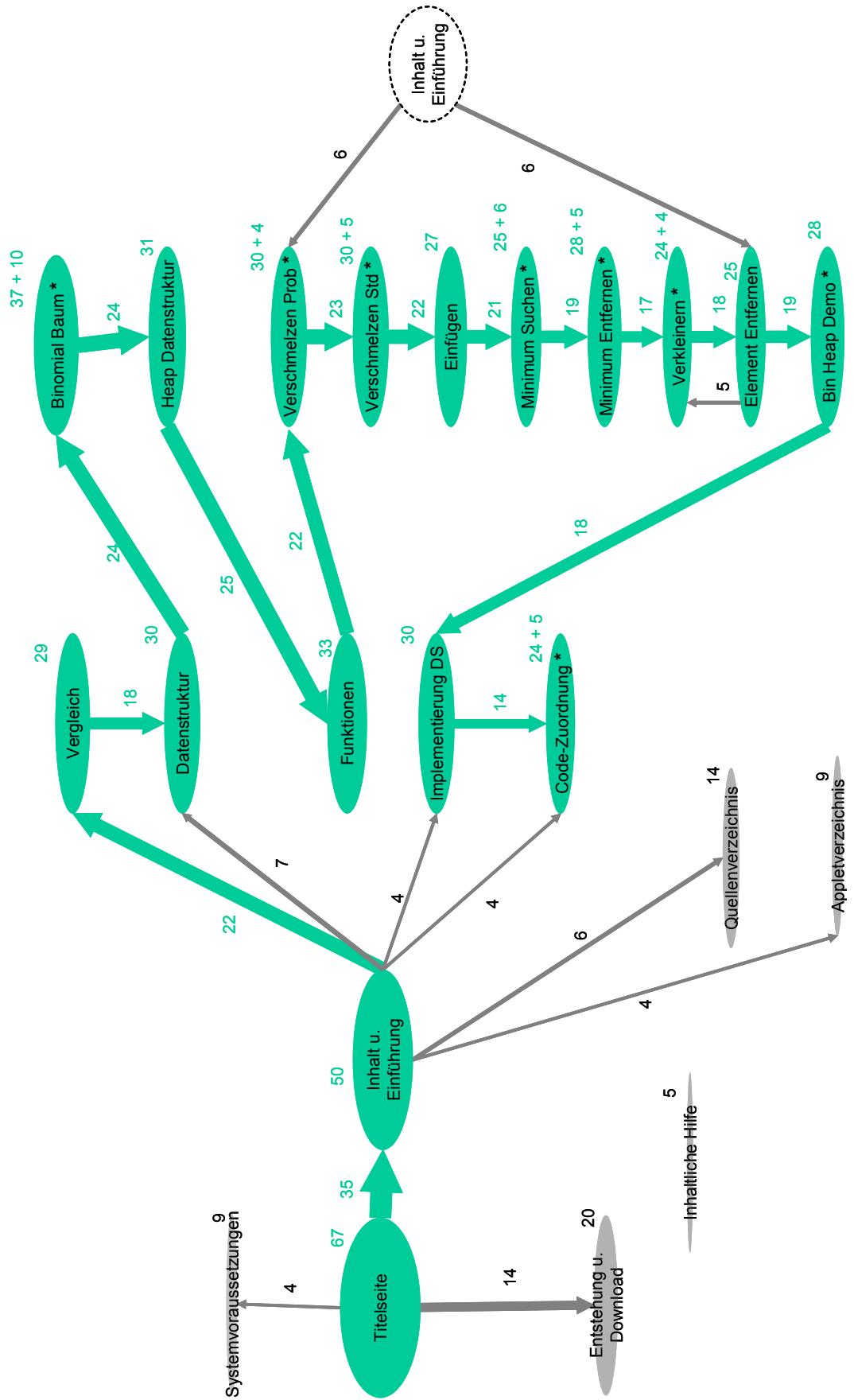


Abb. 45: Nutzung des Lernprogramms zu „Binomial Heap“

8.5 Analyse der Nutzungsdaten

Die in den Abbildungen 44 und 45 dargestellten Nutzungsdaten werden im Folgenden analysiert, um das Navigationsverhalten, die Intensität der Nutzung und das Verhältnis von Online-Nutzung zu Offline-Nutzung zu bewerten.

8.5.1 Das Navigationsverhalten

Nutzung des Standardpfades

Es soll nun untersucht werden, wie stark der Standardpfad von den Studierenden genutzt wurde. Die Seiten des Standardpfades sind in Vorwärts- und Rückwärtsrichtung durch Links miteinander verbunden.

Vergleicht man, wie oft von einer Seite aus der Vorwärtslink und wie oft die sonstigen Links der Seite genutzt wurden, so sieht man, dass immer der Vorwärtslink am meisten genutzt wurde. Auch im Vergleich aller Links eines Lernprogramms gehören die Vorwärtslinks zu den meistgenutzten. Während bei „Heapsort“ auch eine ganze Reihe anderer Links viel genutzt wurde, wurden bei Binomial Heap fast nur die Vorwärtslinks benutzt. Möglicherweise bewirkten bei Binomial Heap die längeren Texte auf den Seiten eine Nutzung ähnlich einem Lehrbuch, in dem man eine Lerneinheit typischerweise Seite für Seite liest. Die starke gedankliche Auseinandersetzung mit den Texten bindet möglicherweise die Nutzer an die Seite und verhindert so „Ausflüge“ vom Standardpfad.

Es ist leider nicht sicher zu sagen, wie oft die Rückwärtslinks genutzt wurden. Wenn ein Nutzer zunächst einen Vorwärtslink benutzt, und dann mit einem Rückwärtslink oder dem Back-Button auf eine bereits gelesene Seite zurückkehrt, so wird diese in der Regel aus dem Browsercache geladen und die Linknutzung wird daher nicht in den Logdateien und damit im Diagramm verzeichnet. In den Diagrammen ist jeweils nur einmal ein Rückwärtslink eingetragen, der auch nur wenig genutzt wurde (4 bzw. 5 mal).

Zusammenfassend lässt sich sagen, dass die Studierenden „Binomial Heap“ fast nur entlang des Standardpfades durchgearbeitet haben. „Heapsort“ wurde hauptsächlich über den Standardpfad bearbeitet, es wurden aber auch andere Links in nennenswertem Umfang benutzt.

Es soll nun anhand der Diagramme 44 und 45 untersucht werden, an welchen Stellen die Studierenden besonders viel Gebrauch von der freien Navigation machten und andere Links als die des Standardpfades nutzten.

Verzweigungspunkte für die freie Navigation bei Heapsort

Die folgenden Links wurden bei „Heapsort“ besonders häufig zur freien Navigation genutzt:

- **Inhalt->Download**

Von den 74 Nutzern sind knapp die Hälfte (29) von der Startseite (Inhalt) zur Downloadseite gegangen. Viele haben dort die Offline-Version heruntergeladen. Aber nur 5 Nutzer haben sich die Seite zu Copyright und Nutzungsbestimmungen durchgelesen. So haben sie nicht erfahren, dass das Programm unter der GNU General Public License verbreitet wird und sie als Nutzer viele Rechte haben. Man muss solche Hinweise wohl auf der Downloadseite selbst unterbringen.

- **Inhalt->Systemvoraussetzungen**

Gut die Hälfte der Nutzer hat sich über die Systemvoraussetzungen informiert. Von diesen kam aber nur rund die Hälfte von der Startseite (Inhalt). Der andere Teil hat sich erst informiert, als beim ersten Applet Probleme auftraten oder kam von der Downloadseite.

- **Inhalt->Funktionsübersicht**

Obwohl recht viele Zugriffe vom Inhalt zur Funktionsübersicht gingen, stieg die Anzahl der Nutzer der Seite „Funktionsübersicht“ kaum gegenüber den Seiten an, die vor und nach der Funktionsübersicht auf dem Standardpfad liegen. Vermutlich sind viele Nutzer von einer Funktionen-Seite über den Navigationsbutton zur Inhaltsseite, dann zur Funktionenübersicht und dann zum eigentlichen Ziel, einer Funktion gegangen. Die Funktionen-Seiten sollten also einen zusätzlichen Link zur Funktionenübersicht erhalten.

- **Funktionenübersicht->Heapsort**

Von der Funktionenübersicht gelangt man auf dem Standardpfad zur Funktion Heapify-Locally. Von den anderen Funktionen wurde „Heapsort“ besonders häufig direkt angesteuert. Die Seite enthält aber keine übergeordneten Informationen. Das Lernprogramm

ist nicht dafür ausgelegt, die Funktionen des Algorithmus in einer anderen Reihenfolge als der des Standardpfades zu erlernen. Vermutlich wollten einige Studierende trotzdem hier den Lernweg „abkürzen“ und schneller zum „Wesentlichen“ kommen.

- **Heapify-Locally -> Hilfeseiten**

Von Heapify-Locally aus wurde die technische und inhaltliche Hilfe sehr häufig aufgerufen. Der Grund dafür wird im Abschnitt „Intensität der Nutzung“ erörtert (vgl. Kap. 8.5.2, S. 97). Von den anderen Funktionsseiten gibt es keinen direkten Link zu den Hilfeseiten, sondern einen Link zum Hilfeabschnitt der Seite Heapify-Locally. Dieser Link wurde aber nur von Heapify aus genutzt, und das auch nur einige Male.

Verzweigungspunkte für die freie Navigation bei „Binomial Heap“

Wie bereits besprochen, haben die Studierenden bei „Binomial Heap“ kaum frei navigiert, sondern sind meist dem Standardpfad gefolgt. Zum Beispiel war die inhaltliche Hilfe von allen Seiten erreichbar, wurde aber fast nicht aufgerufen. Die freie Navigation wurde an folgenden Stellen genutzt:

- **Titelseite -> Download und Systemvoraussetzungen**

Download und Systemvoraussetzungen wurden von den Studierenden weniger häufig als bei „Heapsort“ genutzt. Es haben z.B. nur rund ein Siebtel der Nutzer (gegenüber der Hälfte bei „Heapsort“) die Systemvoraussetzungen gelesen. Vermutlich hatten viele Nutzer ihr System schon für „Heapsort“ eingestellt und erwarteten nun keine technischen Schwierigkeiten bei der Arbeit mit Binomial Heap. Da das Programm tatsächlich problemlos lief, gab es auch später keinen Grund, die Systemvoraussetzungen zu prüfen.

- **Inhalt -> Quellenverzeichnis und Appletverzeichnis**

Vom Inhaltsverzeichnis aus wurden das Quellenverzeichnis und Appletverzeichnis vereinzelt genutzt.

8.5.2 Die Intensität der Nutzung

Es soll nun untersucht werden, wie viele Studierende die einzelnen Seiten des Lerninhaltes nutzten, und ob darin ein Trend zu erkennen ist.

Heapsort

Die inhaltliche Arbeit mit dem Lernprogramm beginnt bei „Heapsort“ auf der Seite Datenstruktur (37 Nutzer). Die Zahl der Nutzer bleibt bis zur Seite Heapify-Locally (40 Nutzer) auf diesem Niveau.

Hier wird die erste Funktion behandelt und hier ist auch die erste Aufgabe in einem Applet zu lösen. Die Seite wirkte leider als Nadelöhr: Nur die Hälfte der Nutzer hatten die nächste Seite Heapify gelesen (21 Nutzer). Vermutlich hatten die anderen Nutzer technische Probleme mit dem Applet oder inhaltliche Probleme mit der Aufgabe und machten deshalb nicht weiter. Es gab von der Seite Heapify-Locally aus 14 Zugriffe auf technische Hilfeseiten und 13 Zugriffe auf inhaltliche Hilfeseiten (berechnet als Summe der Linknutzer). Einzelne Studierende haben von dem technischen Problem berichtet, dass sie die Applets auf ihrem privaten PC nicht zum Laufen brachten. Auch berichteten einzelne Studierende, dass sie sich hier mehr Lehrtext zur Aufgabenstellung oder eine textuelle Musterlösung der Aufgabe gewünscht hätten. Leider lässt sich der Grund des Abbrechens aus der Logfile-Analyse nicht sicher ermitteln. Hierzu wäre ein Gespräch mit den Studierenden direkt nach der Arbeit mit dem Lernprogramm erforderlich gewesen.

Die Studierenden, die die zweite Funktion bearbeitet haben, haben auch die restlichen Funktionen durchgearbeitet und die letzte Seite des Standardpfades (Implementierung) gelesen. Die Nutzerzahlen liegen bei diesen Seiten zwischen 20 und 23.

Binomial Heap

Bei „Binomial Heap“ beginnt der Lerninhalt bereits auf der Seite „Inhalt und Einführung“. Da auf der Seite auch das Inhaltsverzeichnis enthalten ist und sie daher als Verzweigungspunkt genutzt wurde, bestimme ich die Anzahl der Nutzer, die begonnen hat, den Lerninhalt durchzuarbeiten, aus der Anzahl Nutzer der Seite „Vergleich“. Sie wurde von 29 Nutzern besucht. Auf den folgenden Seiten des Lerninhaltes bewegen sich die Nutzerzahlen zwischen 37 und 24.

Bei „Heapsort“ ist rund die Hälfte der Nutzer des ersten Applets nicht zur Folgeseite gegangen. An der entsprechenden Stelle bei Binomial Heap (von „Binomial Baum“ zu „Heap Datenstruktur“) sank die Nutzerzahl nur leicht von 37 auf 31.

Zusammenfassung

Tabelle 14: Intensität der Nutzung

	Heapsort	Binomial Heap
Lerninhalt erster Teil	40,8 Nutzer	32,0 Nutzer
Lerninhalt zweiter Teil	22,8 Nutzer	27,9 Nutzer
Verhältnis zweiter Teil zu erster Teil	56%	87%

Es soll nun verglichen werden, wie intensiv die beiden Lernprogramme genutzt wurden. Bei beiden Lernprogrammen hatten einige Nutzer nach dem ersten Applet die Arbeit abgebrochen. Daher wird nun für die statistische Auswertung der Lerninhalt in zwei Teile geteilt. Der erste Teil umfasst alle Seiten bis zum ersten Applet, der zweite Teil die danach folgenden Seiten. Tabelle 14 zeigt den Mittelwert der Anzahl Seitennutzer für die beiden Teile der Lernprogramme. Die unterste Zeile der Tabelle zeigt das Verhältnis von Nutzern des zweiten Teils zu der Anzahl der Nutzer des ersten Teils. Diese Zahl kann man als Schätzwert für den Prozentsatz der Nutzer heranziehen, die die Arbeit nach dem ersten Applet bis zum Ende hin fortsetzten. Bei „Binomial Heap“ hat ein wesentlich höherer Anteil der Nutzer, die bis zum ersten Applet gekommen waren, die Arbeit mit dem Lernprogramm abgeschlossen.

Möglicherweise waren die folgenden Gründe ausschlaggebend. Studierende gaben an, dass es bei „Heapsort“, nicht aber bei Binomial Heap, technische Probleme mit den Applets gab. Weiterhin vermissten sie bei „Heapsort“ eine Musterlösung und ausführliche Lehrtexte, so wie sie bei „Binomial Heap“ vorhanden waren.

8.5.3 Das Verhältnis von Online-Nutzung zu Offline-Nutzung

Die Lernprogramme konnten online über das Internet genutzt werden. Zusätzlich war es möglich, sie als Archiv herunterzuladen, sie zu installieren und dann ohne Internetzugang offline mit ihnen zu arbeiten. Es ist durchaus denkbar, dass einige

Tabelle 15: Online zu Offline

	Heapsort Nutzer	Binomial Heap Nutzer
Online begonnen	40,8	32,0
Download	20	13

Studierende sowohl online mit einem Lernprogramm gearbeitet haben (z.B. in einem Rechneraum der Universität) als auch die Datei heruntergeladen haben (z.B. um das Programm zu Hause zu nutzen oder um es langfristig verfügbar zu haben). Bei einer Offline-Nutzung wurden keine Nutzungsdaten aufgezeichnet, so dass hier keine Auswertung des Navigationsverhaltens und der Nutzungsintensität möglich ist. Daher lassen sich die Offline-Nutzerzahlen am ehesten mit den Online-Nutzerzahlen für den ersten Teil des Lerninhalts vergleichen (vgl. Tab. 14). Sie werden in Tabelle 15 als „Online begonnen“ bezeichnet.

Es soll nun festgestellt werden, ob es für beide Formen (Online und Offline) einen Bedarf gibt. Die Download-Zahlen (Tabelle 15) wurden aus den Logdateien des Webservers ermittelt und finden sich nicht in den Abbildungen zur Nutzung der Lernprogramme. Bei beiden Lernprogrammen zeigt sich ein ähnliches Verhältnis von Offline-Nutzern zu Online-Nutzern von rund 2 zu 1. Es gibt offensichtlich für beide Nutzungsformen einen deutlichen Bedarf.

8.5.4 Zusammenfassung

Das Lernprogramm „Heapsort“ wurde hauptsächlich über den Standardpfad bearbeitet, es wurden aber auch andere Links in nennenswertem Umfang benutzt. Die Studierenden haben „Binomial Heap“ fast nur entlang des Standardpfades durchgearbeitet.

Während rund die Hälfte der Studierenden die Arbeit mit dem Heapsort-Lerninhalt abbrach, haben bei „Binomial Heap“ praktisch alle Studierenden bis zum Ende gearbeitet.

Es gab einen deutlichen Bedarf für Online- und Offline-Versionen der Lernprogramme. Die Zahl der Online-Nutzer zur Zahl der Downloads lag bei beiden Programmen etwa bei 2 zu 1.

9 Zusammenfassung und Ausblick

9.1 Zusammenfassung

Im Rahmen dieser Arbeit wurde eine neue Lehrmethode für Algorithmen entwickelt und erprobt. Die Lerner nehmen darin eine aktive Rolle ein, bei der sie mithilfe einer vorbereiteten Lernumgebung den Algorithmus selbst entdecken.

Einige verwandte Arbeiten zu aktivem Lernen von Algorithmen wurden vorgestellt und bewertet (vgl. Kap. 3). So lernen und vertiefen Studierende einen Algorithmus, indem sie den Algorithmus selbst ausführen, eine Algorithmen-Animation entwickeln oder ihn visuell programmieren. Der in dieser Arbeit behandelte Ansatz wurde von anderen Gruppen noch nicht untersucht.

Für Autoren von Lernprogrammen wurde eine Gestaltungsmethode ausgearbeitet, die ein Grundgerüst für die Gliederung der Programme und die Funktionsweise der darin enthaltenen interaktiven Simulationen von Algorithmenteilen beschreibt (vgl. Kap. 4). Als Beispiel wurde die Gestaltung eines Lernprogramms zu Heapsort vorgestellt. Weitere Hinweise zur software-ergonomischen und didaktischen Gestaltung solcher Simulationen wurden als Gestaltungsregeln bereitgestellt (vgl. Kap. 5). Die im Rahmen dieser Arbeit entwickelte Bibliothek SALABIM erleichtert die Implementierung der interaktiven Simulationen (vgl. Kap. 6).

Zwei Lernprogramme zu den Themen Heapsort und Binomial Heap sind nach der Gestaltungsmethode SALA entwickelt worden. Sie wurden in einer Lehrveranstaltung im Grundstudium der Informatik eingesetzt und erprobt (vgl. Kap. 7). Studierende und Lehrende bewerteten über Fragebögen die Lernprogramme und das dahinter stehende didaktische Konzept. Insgesamt kamen sie zu einer positiven Einschätzung. Sie schlugen auch Verbesserungen der Lernprogramme vor, von denen die Mehrzahl in einer neuen Version des Lernprogramms Heapsort berücksichtigt werden konnte. Weitere Einsichten in die Nutzung der webbasierten Lernprogramme lieferte eine Analyse der Logdateien des Webserver (vgl. Kap. 8). So konnten das Navigationsverhalten, die Intensität der Nutzung und das Verhältnis von Online- zu Offline- Nutzern ermittelt werden.

Durch die Arbeit werden für die Fachdidaktik der Informatik zwei wesentliche Innovationen erreicht:

- Die nach der Methode SALA entwickelten Lernprogramme erlauben Lernern einen neuartigen, motivierenden und kreativen Zugang zum Lernen von Algorithmen.
- Die Methode SALA, die Gestaltungshinweise und die Softwarebibliothek SALABIM erleichtern es zukünftigen Autoren didaktisch und softwareergonomisch ansprechende Lernprogramme zu Algorithmen zu erstellen.

9.2 Ausblick

9.2.1 Lernprogramme für weitere Algorithmen

Nach der Methode SALA wurden bisher zwei vollständige Lernprogramme zu Heapsort und Binomial Heap erstellt. Für den Voronoi-Algorithmus wurde ein Lernprogramm konzipiert und in Teilen realisiert. Entwürfe für weitere Lernprogramme existieren z.B. für Dijkstras Algorithmus der Suche kürzester Wege in einem Graphen. In diesen Fällen war es problemlos möglich, den Algorithmus in neue, kleinere Einheiten zu gliedern und für diese Einheiten interaktive Simulationen zu entwerfen. Da SALA sich an allgemeingültigen Prinzipien der Modularisierung orientiert, ist anzunehmen, dass

sich SALA zur Erstellung von Lernprogrammen für eine weite Klasse von Algorithmen anwenden lässt. Um dies zu bestätigen, sollten auch für Algorithmen aus weiteren Bereichen (z.B. Numerik, Speicherverwaltung, Hashing, Textsuche) Lernprogramme entwickelt werden. Es wäre auch vorteilhaft, für alle Algorithmen einer Vorlesung Lernprogramme nach SALA anbieten zu können.

9.2.2 Erweiterung von SALABIM

Die Bibliothek SALABIM unterstützt momentan ein flexibles Undo/Redo, das Arbeiten in den Modi Simulation, Übung und Animation und die weiche

Bewegung von Objekten auf der Zeichenfläche. Wenn weitere Lernprogramme nach SALA entwickelt werden, so kommen sicher Wünsche nach neuen Leistungsmerkmalen hinzu. Denkbar sind z.B. Standardvisualisierungen für Datenstrukturen wie Array oder Zeiger.

9.2.3 Wissensbasierte tutorielle Hilfe

Beim entdeckenden Lernen sollen die Lerner im Modus Simulation frei experimentieren, um eine korrekte Schrittfolge der Algorithmen-Funktion zu finden. Sie können jederzeit in die Modi Animation und Übung wechseln, in denen Lösungshinweise angeboten werden. Um auch im Modus Simulation Lösungshilfen geben zu können, könnten die Lernprogramme um eine wissensbasierte tutorielle Komponente erweitert werden, wie sie in Intelligenten Tutoriellen Systemen (ITS) verwendet wird. Sie besteht aus einem Experten-, Studenten-, Unterricht- und Kommunikationsmodul [Lusti 1992]. Während die erstgenannten Module neu entwickelt werden müssten, könnte für das Kommunikationsmodul die bestehende grafische Benutzerschnittstelle der Applets genutzt werden.

9.2.4 Evaluation der Lernprogrammen und des didaktischen Konzepts

Nach SALA entwickelte Lernprogramme wurden bereits mehrfach erprobt. So hat Karsten Block Studierende bei der Arbeit mit dem Lernprogramm „Binomial Heap“ beobachtet, ihr erlangtes Wissen getestet und sie zu ihrer Einschätzung interviewt. Heapsort und Binomial Heap wurden in der Lehr-

veranstaltung „Datenstrukturen“ erprobt. Es wäre wünschenswert, die Lernprogramme und SALA auch in vergleichenden Studien zu erproben. So könnten, wie in der Studie zu HalVis, einige Gruppen von Studierenden mit herkömmlichen Lehrformen wie Lehrbuch und Vorlesung und andere Gruppen mit SALA-Lernprogrammen unterrichtet werden [Hansen, Narayanan, Shrimpsheer 2000]. Mit Vor- und Nachtests könnte das erlangte Wissen ermittelt und anschließend die Effektivität der Lehrmethoden verglichen werden. Auch könnten verschiedene Versionen eines Lernprogramms verglichen werden, in denen gezielt Teile des Programms entfernt wurden. Auf diese Weise ließe sich der Beitrag der einzelnen Teile für den Lernerfolg abschätzen.

9.2.5 Aktives und soziales Lernen

In seiner jetzigen Form beschreibt SALA die Erstellung von Lernprogrammen für ein eigenständiges, entdeckendes Lernen von Algorithmen. Man könnte nun untersuchen, wie sich SALA mit anderen aktiven Formen des Lernens kombinieren lässt. Ähnlich wie bei Hundhausen könnten die Studierenden eine Reihe von Algorithmen-Simulationen entwickeln und präsentieren. Eine weitere Möglichkeit ist die Kombination mit einer Aufgabe, in der die Studierenden den Algorithmus programmieren. Dazu könnte z.B., wie bei OPSIS, eine visuelle Programmiersprache mit vordefinierten Algorithmenfunktionen eingesetzt werden. Für einen Einsatz in virtuellen Lerngruppen müssten die Lernprogramme um kollaborative Elemente wie Annotationen, Whiteboard und Chat erweitert werden.

9.3 Zukunftsaussichten für interaktive Lernmedien

In der Einleitung wurde ausgeführt, dass viele Lehrende sich scheuen, Algorithmen-Animationen in der Lehre einzusetzen. Dagegen nutzen bereits viele Studierende webbasiertes Lernmaterial zu Informatik-Vorlesungen. Ich gehe davon aus, dass Lehrende zukünftig verstärkt interaktives Lernmaterial entwickeln und in ihr Lehrangebot integrieren werden. Wie die neueren Forschungen zum aktiven Lernen von Algorithmen zeigen, bringt ein didaktisch sinnvoller Einsatz interaktiver Visualisierungen nicht nur eine höhere Motivation, sondern auch einen verbesserten Lernerfolg hervor. Besonders bei den neuen virtuellen Studiengängen erwarten die Studierenden eine in-

teraktive Aufbereitung des Lehrstoffes und geben sich nicht mit statischen elektronischen Skripten zufrieden. Jugendliche wachsen heute in einer durch visuelle und interaktive Medien geprägten Welt auf. Daher wird in den nächsten Jahren auch bei herkömmlichen Präsenzstudiengängen die Erwartung der Studierenden an eine multimediale und interaktive Aufbereitung des Lernstoffes zunehmen. Auch durch die anstehende Liberalisierung des universitären Bildungsmarktes wird der Druck zur Verbesserung der Lehre wachsen. Ich schätze daher die Zukunftsaussichten für interaktive Lernmedien als sehr positiv ein.

10 Anhang: Unterlagen zur Evaluation

Die folgenden Unterlagen dienen der in Kapitel 7 beschriebenen Evaluation.

10.1 Aufgabenblatt zu Heapsort

Aufgabe 29 (Lernprogramm und Fragebogen)**5 Punkte**

Arbeiten Sie das Lernprogramm *Heapsort-SALA* durch. Versuchen Sie die Aufgaben im Modus Simulation zu lösen. Wenn dies nicht gelingt, können Sie auf die Modi Üben und Animation zurückgreifen.

Im Unix-Cluster der ARBI starten Sie das Lernprogramm in einer Shell mit dem Kommando *heapsort*.

Sie können das Lernprogramm auch auf Ihrem privaten Computer laufen lassen. Näheres finden Sie unter http://www-cg-hci.informatik.uni-oldenburg.de/~da/heapsort_SALA/ (oder: <http://informatik-lernen.de/>).

Heapsort-SALA wurde in einem Forschungsprojekt an unserer Universität unter der Leitung von Dipl.-Inf. Nils Fal-tin (Abt. Prof. Gorny) entwickelt. Im Projekt wird erforscht, wie sich interaktive Lernmedien für ein aktives Lernen von Algorithmen einsetzen lassen.

Jede Arbeitsgruppe soll einen Fragebogen ausfüllen. Sollte Ihnen kein Fragebogen vorliegen, finden Sie ihn unter [...].

Sie bekommen die Punkte für Ihre Mitarbeit bei der Evaluation und nicht für die Art, wie Sie den Fragebogen ausfüllen. Der Fragebogen wird ausgewertet, um das didaktische Konzept hinter dem Lernprogramm zu bewerten.

Aufgabe 27 (Heapsort in Java)**5 Punkte**

Der Algorithmus wird im Lernprogramm in sechs Funktionen strukturiert. Programmieren Sie den Algorithmus in Java entsprechend dieser Struktur. Sie können weitere Funktionen hinzufügen.

Aufgabe 28 (Die Funktion *ChangeKey*)**4 Punkte**

Der Heapbaum kann aber auch für eine Prioritäten-Warteschlange, die Sie noch in der Vorlesung kennenlernen, verwendet werden. Entwickeln und programmieren Sie die Funktion *ChangeKey(Knoten k, Schlüssel s)*, die für eine Prioritätenwarteschlange benötigt wird. Sie ändert den Schlüssel am Knoten *k* auf den Wert *s* und stellt danach die Heapordnung wieder her. Beschreiben Sie auch die Grundidee, nach der ihr *ChangeKey* arbeitet.

10.2 Aufgabenblatt zu Binomial Heap

Aufgabe 29 (Lernprogramm und Fragebogen)

5 Punkte

Arbeiten Sie das Lernprogramm zum „Binomial Heap“-Algorithmus durch. Sie finden das Lernprogramm unter der folgenden Adresse:

<http://www-cg-hci.informatik.uni-oldenburg.de/~da/block/BHeapA/Deutsch/Titelseite.html>

Jede Arbeitsgruppe soll einen Fragebogen ausfüllen. Sollte Ihnen kein Fragebogen vorliegen, finden Sie ihn unter [...].

Sie bekommen die Punkte für Ihre Mitarbeit bei der Evaluation und nicht für die Art, wie sie den Fragebogen ausfüllen. Das Lernprogramm wurde in einem Forschungsprojekt an unserer Universität unter der Leitung von Dipl.-Inf. Nils Faltin (Abt. Prof. Gorny) entwickelt. Im Projekt wird erforscht, wie sich interaktive Lernmedien für ein aktives Lernen von Algorithmen einsetzen lassen. Der Fragebogen wird ausgewertet, um das didaktische Konzept hinter dem Lernprogramm zu bewerten.

Aufgabe 30 (Datenstruktur und Funktionen)

5 Punkte

- a) Aus wie vielen Elementen besteht ein B4-Baum?
- b) In einem Binomial Heap sind 15 Elemente gespeichert. Aus welchen Binomialbäumen (B0, B1, B2 ...) besteht er?
- c) Fügen Sie in einen leeren Binomial Heap in der folgenden Reihenfolge Schlüssel ein: 7 1 5 9 3 7 2 4. Zeichnen sie den Binomial Heap nach jeder Einfügung.
- c) Entfernen sie das Element mit Schlüssel 5. Zeichnen Sie auch die wesentlichen „Zwischenergebnisse“ (Zustände der Datenstruktur).

Aufgabe 31 (Aufwandsabschätzung)

5 Punkte

- a) Wie viele Binomialbäume S hat ein Binomial Heap mit N Elementen maximal? Welche Höhe H hat der größte Binomialbaum darin maximal? Liefern Sie ein gute Abschätzung.
- b) Schätzen Sie für die sieben Funktionen des Algorithmus die Anzahl Schlüsselvergleiche im schlechtesten Fall in Abhängigkeit von S und H ab. Begründen Sie jeweils kurz ihre Abschätzung.

10.3 Fragebogen zu Heapsort

Fragebogen zum didaktischen Konzept hinter dem Lernprogramm Heapsort-SALA

Allgemeine Fragen

Ihr Studienfach: _____

Ihr Vorwissen über den „Binomial Heap“-Algorithmus: keines, Grundidee, detailliert.

Wie geübt sind Sie im Umgang mit Webbrowsern? ungeübt, mittel, sicher

Gab es technische Probleme mit dem Programm?

Wenn ja, welche: _____

Fragen zum didaktischen Konzept

Sie haben ein Lernprogramm kennengelernt, das nach dem didaktischen Konzept SALA gestaltet wurde. Wie schätzen Sie die einzelnen Punkte des didaktischen Konzepts nach dieser Erfahrung (und ihren Lernvorlieben) ein? Wie hilfreich sind sie für das Lernen des Algorithmus?

Abstufungen:

sehr hilfreich (1), hilfreich (2), brauchbar(3), fraglich(4), nutzlos(5), kontraproduktiv(6)

Punkt des Konzepts	Bewertung (1-6)
Das Lernprogramm bietet integrierte Aufgaben mit Simulation und Bewertung, mit denen man aktiv arbeiten kann.	
Die Aufgaben werden über eine grafische Benutzeroberfläche bedient.	
Die Datenstruktur wird grafisch dargestellt. Bildfolgen zeigen den zeitlichen Ablauf.	
Der Algorithmus wird in viele kleine Funktionen zerlegt.	
Die Arbeitsweise einer Funktion soll zunächst durch Experimentieren selbst herausgefunden werden	
Der Modus Simulation	
Der Modus Übung	
Der Modus Animation	
Gesamturteil zum didaktischen Konzept hinter dem Lernprogramm	

Fragen zum Lernprogramm

Ihr Gesamturteil zum Lernprogramm (1-6): ____

Platz für Ihre Anmerkungen

Vom Tutor auszufüllen (Teilnehmer bleiben anonym):
Aufgabe Programmierung ____ Punkte. Aufgabe ChangeKey: ____ Punkte.

10.4 Fragebogen zu Binomial Heap

Fragebogen zum didaktischen Konzept hinter dem Lernprogramm „Binomial Heap“

Allgemeine Fragen

Ihr Studienfach: _____

Ihr Vorwissen über den „Binomial Heap“-Algorithmus: keines, Grundidee, detailliert.

Wie geübt sind Sie im Umgang mit Webbrowsern? ungeübt, mittel, sicher

Gab es technische Probleme mit dem Programm?

Wenn ja, welche: _____

Fragen zum didaktischen Konzept

Sie haben ein Lernprogramm kennengelernt, das nach dem didaktischen Konzept SALA gestaltet wurde. Wie schätzen Sie die einzelnen Punkte des didaktischen Konzepts nach dieser Erfahrung (und ihren Lernvorlieben) ein? Wie hilfreich sind sie für das Lernen des Algorithmus?

Abstufungen:

sehr hilfreich (1), hilfreich (2), brauchbar(3), fraglich(4), nutzlos(5), kontraproduktiv(6)

Punkt des Konzepts	Bewertung (1-6)
Das Lernprogramm bietet integrierte Aufgaben mit Simulation und Bewertung, mit denen man aktiv arbeiten kann.	
Die Aufgaben werden über eine grafische Benutzeroberfläche bedient.	
Die Datenstruktur wird grafisch dargestellt. Bildfolgen zeigen den zeitlichen Ablauf.	
Der Algorithmus wird in viele kleine Funktionen zerlegt.	
Die Arbeitsweise einer Funktion soll zunächst durch Experimentieren selbst herausgefunden werden (Probierphase).	
- Anschließend wird die Funktion erklärt (Erklärungsphase)	
Applets zur Simulation (z. B. B3-Baum, Verschmelzen1, Minimum suchen)	
Applets zur Übung (Verschmelzen2, Minimum entfernen, Code erkennen)	
Applets mit Animation (Binomial Heap in Kap. 4.7)	
Gesamturteil zum didaktischen Konzept hinter dem Lernprogramm	

Fragen zum Lernprogramm

Ihr Gesamturteil zum Lernprogramm (1-6): ____

Platz für Ihre Anmerkungen

Vom Tutor auszufüllen (Teilnehmer bleiben anonym):
Aufgabe Datenstruktur ____ Punkte. Aufgabe Aufwand: ____ Punkte.

10.5 Antwortdaten der Fragebögen

Nr	Person	Lernprogramm	Studienfach	Vorwissen	Webbrowser	Probleme	Aufgaben	GUI	DS_grafisch	Viele_FN	Experimentieren	Erklärung	Simulation	Übung	Animation	Ges_SALA	Ges_LP	Aufgabe1	Aufgabe2
1	L	HS				0	2	3	2	3	1		2	2	2	2	2		
2	L	HS		2	3	0	2	1	3	2	3		2	4	2	3	2		
3	L	HS		3	3	0	2	1	1	1,5	1		1		3	1,5	1,5		
4	S	HS	I	2		1	4	3	3	4	3		4	4	3	4	4		
5	L	BH		2	3	0	2	1	2	2	3	2	3	2	2	3	2		
6	S	BH	B	1	3	0	1	2	1	2	1	2	1	1	1	2	2		
7	S	BH	I	3	2	1	3	2	2	3	4	3	3	3	5	3	3	5	5
8	S	BH	I	2	3	1	1	2	1	2	1	2	1	1	1	1	1	5	1
9	S	BH	I	2	1	1	1	1	1	2	3	2	1	1	1	1	1	5	2
10	S	BH	L	1	2	0	3	2	2	2	3	2	3	3	2	2	2	5	3
11	S	HS	I	1	2	1	2	1	3	4	6		5	2	2	2	3	-	-
12	S	HS	I	1	3	1	3	2	2	2	3		4	2	3	2	3	4	-
13	S	HS	I	2	1	0	2	1	1	2	3		3	2	2	2	2		
14	S	HS	I	2	1	0	2	1	2	1	3		1	2	1	2	2		
15	S	HS	I	3	3	0	1	2	2	2	2		1	2	1	2	2		
16	S	HS	I	2	3	1	3	3	2	4	4		3	3	3	2	4		
17	S	HS	I	2	3	1	3	2	2	2	3		2	3	2	3	3	5	1
18	S	BH	I	1	3	0	3	2	2	3	3	2	3	3	3	2	2	4,5	1
19	S	BH	I	3	3	0	2	2	2	1	1	2	2	3	3	2	2	5	-
20	S	BH		2	3	1	3	2	2	3	3	2	2	2	2	2	2	-	-
21	S	BH	I	2	1	0	2	2	2	3	3	2	1	1	1	2	2	-	-
22	S	BH	I	2	2	0	2	1	2	1	1	3	2	2	2	2	2	5	1
23	S	BH	I	2	3	0	2	2	2	2	3	2	2	2	2	2	2	5	-
24	S	BH	I	2	2	0	1	1	2	3	2,5	1	2	2	3	2	2	5	-
25	S	BH	I	1	3	0	3	2	2	2	3	3	2	2	3	4	3	5	1
26	S	BH	I	2	3	0	2	1	3	2	2	2	2	2	4	2	2	5	-
27	S	BH	I	1	3	0	2	2	1,7	3	2,7	2	1,5	1,5	1,5	2	2	5	-
28	S	HS	I	2	3	1	1	1	1	2	1		3	1	2	1	2	2	-
29	S	HS	I	2	3	0	2	2	2	1	1		4	4,5	4	2		0	-
30	S	HS	I	2		1	1	1	1	2	4		2	3	1	3	2	5	-
31	S	HS	I	2	3	1	3	3	3	3	3		3	3	3	3	3	-	-
32	S	HS	I	2	3	0	3	4	2	2	2		2	2	2	2	2	5	0

Nr	Person	Lernprogramm	Studienfach	Vorwissen	Webbrowser	Probleme	Aufgaben	GUI	DS_grafisch	Viele_FN	Experimentieren	Erklärung	Simulation	Übung	Animation	Ges_SALA	Ges_LP	Aufgabe1	Aufgabe2
33	S	HS	I	1	3	1	1	1	1	1	2		2	1	2,5	1,5	1,5	5	-
34	S	BH	I	1	3	0	3	2	3	2	3	3	1	2	2	3	3	-	-
35	S	BH	I	2	2	1	2	1	1	2	3	1	2	2	1	1	2	4	-
36	S	BH	I	1	3	0	2	2	2	2	2	2	2	2	2	2	2	4	1
37	S	BH	I	1	3	0	1	1	1	1	1,5	1	1	1	3	1	1	5	1
38	S	BH	I	1	3	1	1	1	1	1	1	1	1	1	1	1,3	1	4	2
39	S	BH	I	2	1	0	2	2	2	1	1	1	2	1	3	2	2	5	1
40	S	HS	I	1	3	0	3	2	2	2	1		3	2	3	2	2	-	-
41	S	HS	I	2	3	0	3	2	1	1	4		2	1	3	2	2	4	-
42	S	HS	I		3	1	3	2	2	3	4		4	4	4	5	5	4	-
43	S	HS	I	2	3	1	3	2	4	2	1		2	2	2			5	-
44	S	HS	L	2	2	1	2	2	3	2	3		4	2	3	2	2	-	-
45	S	HS	I	1	3	0	2	1	2	1	1		3	1	2	2	2	-	-
46	S	HS	I	2	3	1	2	3	3	2	2		2	2	3	2	2	4	-
47	S	HS	I	3	2	1	1	2	3	2	3		5	3	2	3	4		
48	S	HS	I	2	2	1	2	1	3	3	2		2	2	3	2	3		
49	S	HS	I	3	2	0	1	1	3	4	3		1	3	1	2	2	-	-
50	S	HS	I	1	2	1	2	2	1	3	4		2	1	2	3	2	4	-
51	S	HS	I	1	3	1	2	1	4	1	2		2	2	5	2	3	4	4
52	S	HS	B	3	3	1	2	1	2	1	3							0	0
53	S	HS	I	2	3	1	2	4	3	2	2		2	3	2	3	3	0	0
54	S	HS	I			0	3	2	2	3	3		4	3	4	3	3	0	0
55	S	HS	I	2	2	1	3	2	2	3	3		4	3	4	3	3	0	0
56	S	HS	I	2	2	0	2,5	1	3	2	3,5		2	2	5	3	2,5	0	3
57	S	HS	I	1	3	0	2	1	3	2	2		4	3	5	2	3,5	2	2
58	S	HS	I	3	3	1	1	2	3	1	1		1	2	3	2	1	4,5	0
59	S	HS	I			1	4	3	3	3	3		4	4	4	4	4	5	0
60	S	HS	I	2	3	1	3	4	4	3	4		4	4	4	4	4	5	0
61	S	HS	I	1	2	0	2	2	3	2	1		3	2	2	2	2	4	4
62	S	HS	I	2	2	1	2	2	2	2	1		3	2	3	2	3	4	4
63	S	HS	I	1	3	0	4	4	4	4	4		4	4	4	4	4	5	0
64	S	HS	I	2	2	0	2	1	4	3	2		2	3	5	3	3	2	2
65	S	HS	I	2	3	1	2	3	2	1	4		2	2	2	3	3	4	0
66	S	HS	I	3	3	1	2	1	2	3	3		2	3	4	2	3	5	0

Nr	Person	Lernprogramm	Studienfach	Vorwissen	Webbrowser	Probleme	Aufgaben	GUI	DS_grafisch	Viele_FN	Experimentieren	Erklärung	Simulation	Übung	Animation	Ges_SALA	Ges_LP	Aufgabe1	Aufgabe2
67	S	HS	I	2	3	0	2	1	2	2	3		2	2	2	2	2	5	0
68	S	HS	I	2	3	0	3	2	3	2	2		5	2	3	2	2	5	2
69	S	BH	I	2	3	0	1	1	2	2	1	2	2	2	2	2	2	5	1
70	S	BH	I	1	2	0	2	1	2	2	3	2	2	3	3	2,5	2,5	4,5	1
71	S	BH	I	3	3	0	6	2	2	1	2	4	2	5	2	3	3	4	2
72	S	BH	I	1	3	0	3	2	2	2	3	3	2	4	2	2	2	5	0
73	S	BH	I	3	3	0	2	2	1	2	4	3	2	2	2	2	2	4,5	1
74	S	BH	I	2	3	0	1	3	2	1	1	2	1	2	2	2	1,5	3,5	2
75	S	BH	I	1	2	1	2	2	2	2	2	2	2	2	3	2	2	5	4
76	S	BH	I	1	2	0	2	2	2	1	1	1	2	2	3	2	2	5	0
77	S	BH	I	1		0	3	3	3	2	3	3	2	2	2	3	3	3	0
78	S	BH	I	2	3	0	2	1	4		2	2				2	1,5	5	0
79	S	BH	I	1	3	0	2	1	2	3	2	1	2	2	2	2	2	5	1
80	S	BH	I	1	3	0	1	1	2	2	2	3	2	2	2	2	2	5	4
81	S	BH	I			0	3	3	3	3	3	3	2	2	2	2	2	3	0
82	S	BH	I			0	3	3	3	2	3	3	2	2	2	3	3	3	0
83	S	BH		2	3	0	1	2	4	2	2	2	2	2	2	2	2	5	1
84	S	BH	I	2	2	0	2	2	2	1	2	3	3	3	3	2	2,3	5	4
85	S	BH	I	1	3	0	2	2	3	1	1	1	1	1	3	1		5	4
86	L	BH				0	2	3	2	2	2	2	3	4	2	2	2		
89	L	HS		3	3	1	1	1	1	2	3		4	1	2	2	2		
90	L	BH		3	3	1	1	1	1	1	3	2	3	1	3	2	2		

10.6 Anmerkungen der Nutzer auf den Fragebögen

Nr	Anmerkungen
2	Anmerkung: Ein Lernmaterial für versch. Lernertypen -> problematisch. Aufgaben gelöst, heisst nicht alles verstanden. Unterschied Übung zu Simulation war nicht klar.
3	Anmerkung: Unterschied Simulation, Übung? Animation läuft sehr schnell.
4	Problem: Das Applet funktionierte nicht im Mac-Raum.
6	Anmerkung: Direkte Erklärung neben den durchgeführten Schritten wäre noch sinnvoller. Notiz bei Frage Übung: Applet Code erkennen -> Note 4 (statt sonst Note 1 bei Frage Übung).
7	Probleme: Anzeige von Kapitel 6 (Netscape Nav.).
8	Probleme: Hatte Probleme Kapitel 4.5-4.7 zu öffnen, hat aber nach mehrmaligen Versuchen geklappt. Anmerkung: in Kapitel 4.4 könnte das Verschmelzen vielleicht auch in einzelnen Schritten ersichtlich sein, damit meine ich, dass man durch Klicken die einzelnen Schritte verfolgen kann, oder zw. vorher und nachher schalten kann.
9	Probleme: Ein paar Probleme mit die Sprache (Wörter). [sic] Anmerkung: Sehr hilfreich! Eine sehr gute Methode neue Dinge zu lernen.
11	Probleme: Applets funktionierten nur auf den 500er Unix-Rechnern, weder zu Hause noch im Mac-Raum noch direkt im Internet. Anmerkungen: move_max: Wer baut eine Heapordnung von rechts nach links auf? Wieso sind nach Heapify-Locally alle Funktionen wie Swap etc. noch mit drin, wenn man sie gar nicht braucht? Dass man sie nicht mehr benutzen darf, war nicht klar und die Frustration hinterher groß, wenn die Aufgabe im Prinzip richtig ausgeführt war. Dieses „Mitschleifen“ der Funktionen ist absolut kontraproduktiv! Insgesamt vom Konzept her gut, aber doch ziemlich verwirrend!
12	Probleme: Ließ sich außerhalb der ARBI nicht ausführen. Anmerkungen: Ohne Vorlesung ist ein Verständnis des Algorithmus fraglich!
16	Probleme: Offline-SALA lief gar nicht. Online-SALA nur am 4.7. auf den 500er Alphas der ARBI, sonst auf keinem anderen Rechner der ARBI (Macs, Mips) (auf die Applets bezogen). Anmerkungen: Intoleranz der Applets gegenüber manueller Ausführung der Algorithmen hat negativen Lerneffekt. Die Sortierreihenfolge (links-rechts, rechts-links) sollte egal sein. Das Konzept an sich ist nicht schlecht, allerdings leider viele technische Probleme.
17	Probleme: Installation swingall.jar. Anmerkungen: Eine detailliertere Erläuterung der zugrundeliegenden Funktionsweise wäre wünschenswert;).
19	Anmerkungen: Aufgaben teilweise zu einfach. Bedienung teilweise etwas umständlich.
20	Probleme: Es war nicht immer alles verfügbar (-> Problem HRZ-Anbindung). Es wäre mal wieder wünschenswert, wenn die einzelnen Funktionen/Abschnitte etwas detaillierter/praxisorientierter dargestellt würden.
22	Bedien- und Anleitungstexte sind im Applet größtmäßig nicht gut angepasst.
26	Anmerkungen: Allgemein: Das Programm sieht solider und professioneller aus als der Vorgänger. Die einzelnen Schaltflächen/Funktionen werden gut erklärt. Es gibt i. allg. mehr zu lesen. Es gab keine Fehlermeldungen. Die Ladezeiten waren in Ordnung. Applet „Verkleinere Schlüssel“: Das Textfeld ist zu klein. Die Information, auf welchen Wert man den Schlüssel verringern soll, ist nicht immer zu lesen. -> Es muß „blind“ verkleinert werden, bis der Computer sich meldet. Applet „Binomial Heap“: Fast nutzlos, da nicht alle Funktionen zur Verfügung stehen. Die „speziellen“ Applets bringen mehr. Übung „Code erkennen“: Aussagekräftige Bezeichner wären schön gewesen! (X?)

Nr	Anmerkungen
28	<p>Problem: swingall.jar - Einbindung. Anmerkungen: Das Konzept ist sehr gut und hilfreich, die Ausführung hat Mängel und setzt das erkennbare Konzept zum Teil nicht richtig um. Die Fragen zum didaktischen Konzept sind meiner Meinung einseitig gestellt. Ist es nicht klar, dass grafische Abläufe/Benutzungsoberflächen gut ankommen? Fordern die Fragen nicht ein positives Ergebnis? Problem Heapsort - Bedienung/Ausführung: - Worin besteht der Unterschied von Simulation und Übung? (Wurde uns erst nach dem Ausprobieren klar!) - In „Simulation“ werden alle möglichen Aktionen zugelassen, auch welche die nicht Algorithmus-konform sind. (Wäre gut, wenn klar wäre, dass es falsch ist!) Move-Max: Wo steht *wie* man den Max Schlüssel in die Ergebnisliste schiebt!? Sort: Animation geht so schnell, man sieht gar nicht was passiert! Heapsort: Unter Simulation wird die falsche Anweisung gar nicht ausgeführt, eigentlich schade, denn dann würde man seinen Fehler selbst entdecken, oder?</p>
29	<p>Erläuterung zur Wertung „Modus Übung“: „Nur im Standardverfahren“. Anmerkungen: Die Didaktik ist gut, die Methodik ist fragwürdig. Beispiel in Heapify(Knoten k): Eine Lösung wird nicht erreicht, wenn H-Locally aufgerufen wird. Erreicht man eine richtige Sortierung mit Swaps und Compares wird diese als falsch bewertet. Drückt man auf H-Locally werden nicht Swaps und Compares gezählt, wofür brauche ich die dann? Beispiel in Heapify-Locally: Die Übungstexte sind irreführend.</p>
30	<p>Probleme: Laden der Applets dauert z.T. sehr lange. Anmerkungen: An manchen Stellen unklar, was gemeint ist.</p>
31	<p>Probleme: Die Swingclasses mussten von Hand eingebunden werden. Umständlich und 2.3 MB download. Anmerkungen: An der grafischen Präsentation sollte vielleicht noch gearbeitet werden. Die Applets sehen ja ganz gut aus, aber der Rest ...</p>
32	<p>Anmerkungen: Offline-Version wäre wünschenswert.</p>
33	<p>Probleme: Läuft nach anfänglich langer Ladezeit recht flott (P90). Anmerkungen: Die Schlüsselwörter könnten in 1-2 Sätzen kurz erläutert werden. Eventuelle Vernetzung zu anderen interaktiven Lernprogrammen.</p>
35	<p>Erläuterung zur Wertung „Experimentieren“: z. B. beim „Minimum Entfernen“ Applet einfach nur in richtiger Reihenfolge Schaltflächen anklicken. Probleme: „Abgestürzte“ Grafik („verrutschte“ Zeilen) über den Applets (nach Scrollen). Mauszeiger verschwindet teilweise in den Applets.</p>
37	<p>Anmerkungen: Die Code-Erkennung ist eine sehr gute Idee, das Binomial Heap Applet ist bestimmt geeignet, die gewonnenen Kenntnisse zu überprüfen und zu festigen, wirklich notwendig ist es jedoch kaum. Um die Effizienz noch zu verbessern, könnte man die Möglichkeit einbauen, um einen beliebigen (zufälligen) Schlüssel zu löschen, bzw. zu entfernen. Die Darstellung der Zwischenergebnisse erscheint hierbei sinnvoll. Insgesamt ist das Lernprogramm sehr effektiv.</p>
38	<p>Probleme: Einbinden von swingall.jar war problematisch, hat aber dann funktioniert. Anmerkungen: Es ist schade, dass es in Kapitel 5 keinen „ausprobieren & lernen“ Teil gibt.</p>
39	<p>Anmerkungen: Es wäre gut, wenn man beim Applet „Binomial Heap“ auch das Löschen usw. ausprobieren könnte. Besser als Heapsort-SALA, sowohl von der grafischen Oberfläche als auch vom Verständnis her (und, da es komplett in Deutsch ist). Trennen der Bäume vor dem Löschen (=Baum entfernen) irritiert etwas; auch, dass Buttons da sind, die nie gebraucht werden.</p>
41	<p>Anmerkungen: Der komplette Sortierablauf kann doch innerhalb eines einzigen Applets ablaufen. Man könnte so auf seine eigenen Teilschritte aufbauen.</p>
42	<p>Probleme: Installationsprobleme von swing.jar, keine Applets.</p>
43	<p>Probleme: Die Applets laufen bei privater Nutzung nicht.</p>
44	<p>Probleme: Lief nicht unter Netscape.</p>

Nr	Anmerkungen
46	Probleme: Applets ließen sich nicht laden, erst nach Schwierigkeiten. Anmerkungen: Man verliert leicht die Lust, wenn man so lange benötigt bis das Applet läuft.
47	Probleme: Änderung der Registry (Windows) klappte nicht, erst die Änderung der Autoexec.bat führte zum Erfolg. Anmerkungen: Systemanforderungen (2 MB herunterladen) zu hoch! Bei Heapify-Locally war es ja noch ganz nett, dass man den Inhalt nicht sah, für die restlichen Funktionen wäre es aber verständlicher gewesen, den Inhalt mit anzuzeigen. „Impliziter Parameter“ ist für nicht OOP-Kenner verwirrend!
48	Probleme: Applet-Anzeige unter Netscape. Anmerkungen: Im Simulationsmodus kann man leider nicht immer alle Einzelschritte ausführen, da die Darstellung nicht reagiert (z.B. Heapsort). Darüber hinaus sind die Vertiefungen eher spärlich und nicht sehr hilfreich.
49	Erläuterung zur Wertung „Simulation“: Bei (*) ist es fragwürdig, ob es wirklich hilfreich ist, dass die Lösungen, die in der optimalen Anzahl von Schritten durchgeführt werden, als richtig akzeptiert werden, obwohl sie dem Algorithmus nicht entsprechen, sondern nur intuitiv richtig sind. Außerdem scheint die Verwirklichung des Konzeptes nicht alle Punkte optimal zu realisieren.
50	Probleme: IE funktionierte so nicht, Problemlos mit NN. Anmerkungen: Den Algorithmus selbst herauszufinden ist zwar für Tüftler recht nett und ein Ansporn, aber eine Musterlösung zur Kontrolle wäre nicht schlecht. Schließlich ist das ja ein Lernprogramm und nicht „Was bin ich“.
51	Probleme: Classpath musste erst gesetzt werden. Anmerkungen: Stellenweise Unterforderung. Lernprozess langsamer als erwartet/notwendig.
52	Probleme: Es war nicht lauffähig trotz neuerer Version. Anmerkungen: Leider konnte ich mir das Programm nicht anschauen, weil die Applets auch nach sämtlichen Versuchen nicht liefen. (swingall gedownloaded, Applets eingeschaltet, Datei mehrmals gedownloaded, aber leider lief das Programm, aber der Platz vom Applet blieb leer).
53	Probleme: Anfangsprobleme mit den Applets.
55	Probleme: Applets liefen nur nach einigen Mühen.
56	Anmerkungen: Animation zu schnell. Die Quadrate (Größe) sind, sobald sie kleiner werden, schwer zu erkennen.
57	Anmerkungen: Z.T. keine Unterschiede zw. den Tasks erkennbar. Unterschiede zw. Übung und Simulation minimal. Ablauf nicht anschaulich genug (Werte? Schritte der Prozeduren??). Es stehen noch die Unterfunktionen zur Verfügung, werden allerdings bei 'Schritt für Schritt' Nachvollzug nicht korrekt umgesetzt bzw. quittiert!!
58	Erläuterung zur Wertung „Datenstruktur“: (manchmal zu schnell); zu „Simulation“: (Reihenfolgen!). Probleme: Ist ab und zu mal abgestürzt (an den Uni-Rechnern). Lief so eckig, ruckelig und langsam, dass es kaum zu benutzen war. Anmerkungen: Macht den Algorithmus sehr anschaulich. Motiviert sehr, weil kleinere Unsicherheiten schnell überprüft werden können, zum Beispiel Ablaufreihenfolgen durch den Modus Übung.
59	Probleme: Lief nur in der ARBI.
60	Probleme: Lief NUR in der ARBI.
62	Probleme: CLASSPATH setzen unter Windows: unter Systemvoraussetzungen steht SET CLASSPATH=[...]/swingall.jar. Eintrag, mit dem es bei mir funktioniert, ist aber SET CLASSPATH = ...; Anmerkungen: - Ziemlich langweilige grafische Oberfläche. - Genauere Ausführungen/Beschreibungen wären hilfreich.

Nr	Anmerkungen
64	Anmerkungen: - Bei den Funktionen, wie z.B. MoveMax oder Sort sind Unterschiede zum Teil schwer ersichtlich. - Die Unterschiede zwischen Simulation und Übung sind minimal. - Der Ablauf der einzelnen Funktionen könnte anhand von einer Tabelle, die die vorgenommenen Schritte dokumentiert, besser veranschaulicht werden.
65	Probleme: Applet ließ sich nicht unter Netscape starten.
66	Probleme: Ließ sich zu Hause überhaupt nicht starten.
68	Anmerkung: Die Modi Simulation und Übung sind zu ähnlich.
70	Anmerkung: Bei der Bewertung steht immer: „Gratulation, Sie haben die Aufgabe geschafft!“. Aber man weiß nicht, ob es einen besseren/schnelleren Weg gibt. Man konnte endlich mal sofort von zu Hause arbeiten, ohne jegliche Anpassung.
71	Erläuterung zur Wertung „Aufgaben“ (6): Die integrierten Aufgaben sind zu leicht, man kann sie fast ohne Hintergrundwissen lösen.
74	Anmerkung: Grafische Benutzeroberfläche könnte ansprechender gestaltet werden bzgl. Hintergrundfarbe, Farben allg.
75	Probleme: Grafikprobleme mit dem Internet Explorer.
76	Anmerkung: Es gibt schönere Hintergrundfarben. Realistisches Beispiel zur Anwendung im Anschluss als Applet realisiert, wäre nicht schlecht.
80	Anmerkung: Besser gelungen als das Programm zu Heapsort, wg. ausführlicher Erklärungen.
83	Erläuterung zur Wertung „DS grafisch“ (Note 4 statt sonst 1 bis 2). [NF: mit Sternchen markiert, aber kein Erläuterungstext].
84	Erläuterung zur Wertung „Gesamturteil LP“: [NF: Als Mittelwert aus Konzept-Bewertungen gebildet. $23/10 = 2,3$].
85	Erläuterung zur Wertung „DS grafisch“ (Note 3 statt sonst bei unkommentierten 1 bis 2); „Simulation/Minimum suchen“: Vater, Bruder, Sohn Buttons überflüssig?; „Animation“ (Note 3): zu schnell.
89	Erläuterung zur Wertung „Experimentieren“ und „Gesamturteil SALA“: Das Programm trägt „lediglich“ dazu bei, den Algorithmus in seiner Funktionalität zu verstehen. Es trägt nicht dazu bei, die Komplexität des Verfahrens zu erkennen bzw. die Funktion selbst (algorithmisch) zu entwickeln, da es quasi „bottom up“ ansetzt. Probleme: Installation von Swing-Klassen aus unerfindlichen Gründen zunächst fehlerhaft. Applets unter Netscape zu langsam.
90	Probleme und Anmerkung: jeweils Verweis auf den Fragebogen zu Heapsort (Nr 89).

11 Quellenverzeichnis

Bei Dokumenten aus dem World Wide Web ist hinter der URL das Datum des letzten Zugriffs angegeben.

11.1 Quellen

Anderson 1996

John R. Anderson: „*Kognitive Psychologie*“. 2. Auflage. 1996. Spektrum Akad. Verlag. Deutsche Übersetzung von "Cognitive Psychology and its Implications". 4th ed. Freeman.

BDGKW 1998

A. Barbu, M. Dromowicz, X. Gao, M. Köster und C. Wolf: Lernsoftware „*Sortieren mit Bubblesort und Quicksort*“. 1998. <http://OLLI.Informatik.Uni-Oldenburg.DE/fpsort/index.html> (Oktober 2001).

Baecker 1981

Ronald Baecker: „*Sorting out Sorting*“. With the assistance of Dave Sherman. 30 minute color sound film, Dynamic Graphics Project, University of Toronto, 1981. Excerpted and „reprinted“ in SIGGRAPH Video Review 7, 1983. Distributed by Morgan Kaufmann, Publishers.

Baecker 1998

Ronald Baecker: „*Sorting out Sorting: A Case Study of Software Visualization for Teaching Computer Science*“. Kap. 24 in [SDBP 1998].

Block 1999a

Karsten Block: „*Möglichkeiten der interaktiven Visualisierung des Binomial Heap Algorithmus*“. Diplomarbeit, Universität Oldenburg. 1999. <http://OLLI.Informatik.Uni-Oldenburg.DE/BHeapA/archive/archive.html> (Oktober 2001).

Block 1999b

Karsten Block: „*Lernsoftware zum Binomial Heap Algorithmus*“. 1999. <http://OLLI.Informatik.Uni-Oldenburg.DE/BHeapA/>(Oktober 2001).

Blumstengel 1998a

Astrid Blumstengel: „*Entwicklung hypermedialer Lernsysteme*“. 1998. Dissertation. Wiss. Verl. Berlin. Hypertext-Version: <http://dsor.uni-paderborn.de/de/forschung/publikationen/blumstengel-diss/> (Oktober 2001).

Blumstengel 1998b

Astrid Blumstengel: „*ORWelt*“. Lernsoftware zu Operations Research. Download-Datei für Windows. 1998. Univ. Paderborn. <http://dsor.uni-paderborn.de/de/forschung/wbs/orwelt/>(Oktober 2001).

Brown 1987

Marc Brown: „*Algorithm Animation*“. 1987. MIT Press, Cambridge.

Brown 1998

Marc H. Brown: „*MacBalsa*“. Software und Dokumentation. <http://www.research.digital.com/SRC/zeus/balsa/>. Heruntergeladen im August 1998.

Cormen, Leiserson, Rivest 1990

Thomas H. Cormen, Charles E. Leiserson und Ronald L. Rivest: „*Introduction to Algorithms*“. 1990. MIT Press, Cambridge, Massachusetts.

Csikszentmihalyi 2001

Mihaly Csikszentmihalyi: „*Flow - Das Geheimnis des Glücks*“. Neunte Auflage, 2001. Klett-Cotta.

DBS 2001

„*Ressourcen-Datenbank des Deutschen Bildungsservers*“. 2001. <http://www.bildungsserver.de/db/listen.html> (August 2001).

Donker 2001

Hilko Donker: „*Didaktisches Interaktions- und Informationsdesign*“. 2001 (im Druck). Dissertation.de -Verlag im Internet GmbH. <http://www.dissertation.de/>(Dezember 2001).

DudenInf 1988

„*Duden Informatik*“. Hermann Engesser (Hrsg.). Bearbeitet von Volker Clauss u. Andreas Schwill. 1988. Dudenverlag.

Edelmann 1996

Walter Edelmann: „*Lernpsychologie*“. 5. Auflage. 1996. Psychologie Verlags Union.

Faltin 2001a

Nils Faltin: „Oldenburger Sammlung von Lernprogrammen zur Informatik“. 2001. <http://OLLI.Informatik.Uni-Oldenburg.DE/> (August 2001).

Faltin 2001b

Nils Faltin: „Lernprogramm zum Heapsort-Algorithmus“. 2001. http://OLLI.Informatik.Uni-Oldenburg.DE/heapsort_SALA/deutsch/start.html (Oktober 2001).

Floyd 1964

Robert W. Floyd. *Algorithm 245 (treesort)*. Communications of the ACM, 7:701, 1964.

Friendly 2001

Michael Friendly: „Re-Visions of Minard“. <http://www.math.yorku.ca/SCS/Gallery/re-minard.html>. (November 2001).

GHJV 1994

Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides: „Design Patterns: Elements of Reusable Object-oriented Software“. 1994. Addison-Wesley.

Gloor, Dynes, Lee 1993

Peter Gloor, Scott Dynes und Irene Lee: „Animated Algorithms“. CD-ROM. 1993. MIT Press, Cambridge, MA.

Gloor 1997

Peter Gloor: „Elements of Hypermedia Design“. 1997. Birkhäuser, Boston, MA.

Gloor 1998

Peter Gloor: „Animated Algorithms“. Kap. 27 in [SDBP 1998].

Gorny 1998

Peter Gorny: „Didaktisches Design telematik-gestützter Lernsoftware“. In: B. Koerber und I.-R. Peters, Informatische Bildung in Deutschland, Berlin 1998, LOG IN Verlag. S.127-155. <http://www-cg-hci.informatik.uni-oldenburg.de/resources/DidDesign.pdf>

Gorny, Faltin 2000

Peter Gorny und Nils Faltin: „Das Projekt Medien-unterstütztes Studium der Informatik (MuSIK)“. März 2000. Abt. Computer Graphics und Software-Ergonomie, FB Informatik, C. v. Ossietzky Universität Oldenburg, D-26111 Oldenburg. <http://www-cg-hci.informatik.uni-oldenburg.de/~musik/> (4.3.2000).

Hansen, Narayanan, Schrimpscher 2000

Steven R. Hansen, N. Hari Narayanan und Dan Schrimpscher. 2000. „Helping learners visualize and comprehend algorithms“. Interactive Multimedia Electronic Journal of Computer-Enhanced Learning, 1(1). <http://imej.wfu.edu/articles/2000/1/02/index.asp> (Dezember 2001).

Herczeg 1994

Michael Herczeg: „Software-Ergonomie: Grundlagen der Mensch-Computer-Kommunikation“. 1994. Addison-Wesley.

Hundhausen 1999

Christopher Hundhausen: „Toward Effective Algorithm Visualization Artifacts. Designing for Participation and Communication in an Undergraduate Algorithms Course“. Dissertation. June 1999. CIS-TR-99-07. Dept. of Comp. and. Inf. Science, University of Oregon, Eugene, USA. <http://lilt.ics.hawaii.edu/%7Ehundhaus/dis/> (August 2001).

Hundhausen, Douglas, Stasko

Christopher Hundhausen, Sarah Douglas and John Stasko. (In press). „A meta-study of algorithm visualization effectiveness“. Journal of Visual Languages and Computing. To appear.

Hung, Rodger 2000

T. Hung and S. H. Rodger: „Increasing Visualization and Interaction in the Automata Theory Course“, Thirty-first SIGCSE Technical Symposium on Computer Science Education, p. 6-10, 2000.

Janser 1998a

Achim Janser: „ViACoBi - Ein interaktives Lehr-/Lernsystem für Algorithmen der Computergrafik und Bildverarbeitung“. CD-ROM für Windows. 1998. FB Informatik II, Univ. Duisburg. Info: <http://www.informatik.uni-duisburg.de/Info2/Janser/Janser.html>.

Janser 1998b

Achim Janser: „Entwurf, Implementierung und Evaluierung des interaktiven Lehr- und Lernsystems VIA-COBI für die Visualisierung von Algorithmen der Computergraphik und Bildverarbeitung“. 1998. Berlin: Logos-Verl.

Knuth 1969

Donald E. Knuth: „The Art of Computer Programming. Volume 1 / Fundamental Algorithms“. 2. print 1969. Addison-Wesley.

Lusti 1992

Markus Lusti: „Intelligente tutorielle Systeme“. 1992. Oldenburg.

Malone 1981

Thomas W. Malone: „*Toward a Theory of Intrinsically Motivating Instruction*“. *Cognitive Science* 4 (333-369) (1981).

Michail 1996

Amir Michail: „*Teaching Binary Tree Algorithms through Visual Programming*“. Qualifying Project Paper, University of Washington, 1996. <http://opsis.sourceforge.net/quals/tech.pdf> (September 2001).

Michail 1998

Amir Michail. *Applet OPSIS zur visuellen Programmierung*. 1998. <http://opsis.sourceforge.net/> (September 2001).

Meyer 1997

Bertrand Meyer: „*Object-Oriented Software Construction*“. 2nd ed. 1997. Prentice Hall.

Ottmann 1998

Thomas Ottmann (Hrsg.): „*Prinzipien des Algorithmenentwurfs*“. 1998. Spektrum Akad. Verlag, Heidelberg und Berlin.

Ottmann, Widmayer 1996

Thomas Ottmann und Peter Widmayer: „*Algorithmen und Datenstrukturen*“. 3. Auflage. 1996. Spektrum Akademischer Verlag, Heidelberg u.a..

Peters 2000

Frank Peters: „*Bestandsaufnahme von Algorithmenanimationen*“. 2000. <http://www-cg-hci.informatik.uni-oldenburg.de/~da/peters/Kalvin/Start.htm> (Oktober 2001).

Preim 1999

Bernhard Preim, „*Entwicklung interaktiver Systeme: Grundlagen, Fallbeispiele und innovative Anwendungsfelder*“. 1999. Springer.

Price, Baecker, Small 1993

B.A. Price, R.M. Baecker und I.S. Small: „*A Principled Taxonomy of Software Visualization*“. 1993. *Journal of Visual Languages and Computing* 4(3):211-266. <http://kmi.open.ac.uk/bp/papers.html> (10.2.2000).

Rodger 2001

Susan Rodger: „*Java Formal Languages & Automata Package (JFLAP)*“. <http://www.cs.duke.edu/~rodger/tools/jflap/index.html> (Oktober 2001).

Rößling 2001a

Guido Rößling: „*Datenbankgestütztes Verzeichnis von Algorithmen-Animationen*“. 2001. <http://www.animal.ahrgr.de/de/AnimList.html> (Dezember 2001).

Rößling 2001b

Guido Rößling: „*Design and Evaluation of a Dynamically Adaptable Algorithm Animation System*“. Vorabversion der Dissertation. Fachbereich Informatik der Universität Siegen.

Rößling, Schüler, Freisleben 2000

Guido Rößling, Markus Schüler und Bernd Freisleben: „*The ANIMAL Algorithm Animation Tool*“. pp. 37-40 in Proc. ACM 5th Annual Conference on Innovation and Technology in Computer Science Education (ITiCSE 2000), Helsinki, Finland. ACM Press, 2000.

Schormann 1999

Jan Schormann: „*Strukturiertes aktives Lernen des Voronoi-Algorithmus im Lernprogramm*“. 1999. Diplomarbeit. Fachbereich Informatik. C. v. Ossietzky Universität Oldenburg. <http://www-cg-hci.informatik.uni-oldenburg.de/~da/schormann/> (Dezember 2001).

SDBP 1998

John T. Stasko, John B. Domingue, Marc H. Brown und Blaine A. Price (Hrsg.): „*Software Visualization*“. 1998. MIT Press, Cambridge, Massachusetts.

Sedgewick 1992

Robert Sedgewick: „*Algorithms in C++*“. 1992. Addison-Wesley, New York u.a..

Stasko 1996

John T. Stasko, „*Using Student-Built Algorithm Animations as Learning Aids*“, Graphics, Visualization, and Usability Center, Georgia Institute of Technology, Atlanta, GA, Technical Report GIT-GVU-96-19, August 1996. <http://www.cc.gatech.edu/gvu/softviz/algoanim/algoanim.html> (Oktober 2001).

Stasko 1999

John Stasko: „*Smooth Continuous Animation for Portraying Algorithms and Processes*“. S. 103-118 in [SDBP 1998].

Stern, Sondergaard, Naish 1999

Linda Stern, Harald Sondergaard and Lee Naish: „*A Strategy for Managing Content Complexity in Algorithm Animation*“. pp 127-130 in: Bill Manaris (Ed.), Proceedings of the 4th Annual SIGCSE/SIGCUE Conference on Innovation and Technology in Computer Science Education - ITiCSE'99. June 1999. ACM Press, New York.

Stern 2001

Linda Stern: „*Algorithms in Action*“. <http://www.cs.mu.oz.au/aia/> (Oktober 2001).

SV 2001

Stephan Diehl, Peter Eades, John Stasko: „*Dagstuhl Seminar on Software Visualization 2001*“. 2001. <http://www.dagstuhl.de/DATA/Reports/01211/> (August 2001).

Williams 1964

J. W. J. Williams. *Algorithm 232 (heapsort)*. Communications of the ACM, 7:347-348, 1964.

Index

A

Abgegebene Fragebögen 74
 Ablaufsteuerung
 SALABIM 63
 Algorithmen-Animationen 53
 Algorithmen-Simulationen 53
 Algorithmen-Übungen 53
 Algorithms in Action 23
 ALVIS 30
 Animation
 Modus 49
 Animationseditor
 ANIMAL 25
 Animationsmodus
 Alg. in Action 23
 Anmerkungen
 Auswertung 78
 Lehrende 79
 Studierende 80
 API-Aufrufe
 Animal 25
 Aufgabe 46
 Aufgaben zur Anwendung 40
 Aufgabenblatt 72
 Automatentheorie 24

B

Bedienungstipps 46
 Beweistechniken 26
 Bibliothek SALABIM 59
 Binärbaum 34
 Binomial Heap
 Erprobung 69
 Lernprogramm 69
 Block
 Seitenstruktur 87
 Bruner 36
 Build-Heap-Funktion 46

C

Call 65
 Call-Klasse 63
 Call-Objekt 63
 Call-Objekt verarbeiten 63

Checkboxes 60
 CLASSPATH 61
 Client-Rechner 92
 Csikszentmihalyi 37

D

Datenstruktur
 abstrakte Sicht auf 39
 Implementierung 39
 Datenstrukturen
 Lehrveranstaltung 70
 Didaktik
 Fragebogenkategorie 78
 didaktische Punkte
 Einschätzung 77
 didaktisches Konzept
 Fragebogen 73

E

Editiermodus
 OPSIS 27
 Effizienz einer Lösung 50
 Einordnung 45
 Einstiegsfunktion 43
 entdeckendes Lernen 35
 Ergebnisliste 35
 Evaluation
 Gestaltung 71
 Evaluationsmethode 71
 Explicit Interfaces 41
 explizite Parameter 45
 extrinsische Fantasie 38

F

FailCall-Objekt 63
 Fantasie 38
 finish
 Methode 64
 Flow 37
 Fragebogen 72
 Framework 61, 68
 freie Navigation 39
 Funktion
 modulare Darstellung 45

Funktionale Struktur 39
 Heapsort 43
 Funktionale Struktur in SALA 42
 Funktionen
 Reihenfolge 43
 Funktionen des Algorithmus 39
 Funktionsaufruf
 SALABIM 63

G

Gesamteinschätzung des Lernprogramms 75
 Gesamteinschätzung von SALA 76
 Geschichte 38

H

Heap 34
 Heapeigenschaft 34
 Heapordnung 35
 Heapsort 34
 Build-Heap Funktion 46
 Erprobung 70
 Reihenfolge der Funktionen 44
 Schnittstelle 44
 heapsort.jar 61
 Herausforderung 38
 hierarchische Struktur 23
 Hilfsfunktionen 46
 History-Liste 63
 Hyperlink
 Seitenstruktur 87

I

implizite Parameter 45
 Information Hiding 41
 Initialisierung 61
 Initialisierungsphase 61
 Input Generality 29
 Intelligentes Tutorielles System 100
 Interaktionsschleife 61, 63
 intrinsische Fantasie 38
 intrinsische Motivation 36
 Intuitives Denken 36
 IP-Adresse 92
 ITS 100

J

Java-Swing 60, 91

JFLAP 24

K

Key Frames 30
 Kommunikationskompetenz 29
 Konstruktion
 JFLAP-Modus 24
 Kontrollfluss im Applet 61
 Kontrollkästchen 60
 Konzeptphase 30
 Kurzbeschreibung 45

L

Lehrbetrieb 70
 Lehrbuch zu Lernprogramm 34
 Lehrmethoden 23
 Lehrveranstaltung Datenstrukturen 70
 Lehrziel 33
 Lernhilfe
 Bin fertig 64
 Tipp 64
 Lernphasen beim entdeckenden Lernen 44
 Lernunterstützung 59
 Linguistic Modular Units 41
 Linknutzer 92
 Logdaten 92
 Lösung 46

M

Malone 38
 Meinungsbild 71
 Metapher 38
 Meyer, Bertrand 40
 Minard 93
 Model 65
 Modi 49
 Modular
 Composability 41
 Software 40
 understandability 41
 Multilingual 60
 Musterlösung 45
 Initialisierung 63

N

Nachbedingung 46
 Napoleon 93

- Navigationselemente
 - auf der Webseite 90
- Navigationsleiste 90
- Navigationsverhalten 96
- Neugier 38
- NumberCall 65
- Nutzung
 - Intensität 97
- Nutzung der Links 93
- Nutzung der Seiten 93
- Nutzung zweier SALA-Lernprogramme 87
- Nutzungsdaten 91
 - Analyse 96
 - Visualisierung 93

O

- Offline-Nutzung 98
- Offline-Version 91
- Online-Nutzung 98
- Online-Version 91
- Operationen
 - OP SIS 27
- OP SIS 26

P

- Parameter 45
- Präsentation
 - Fragebogenkategorie 78
- Präsentationsmodus 31
- Präsentieren einer Animation 31
- Probierphase 45
- Problemlösefähigkeit 36
- Problemstellung 45
- Programmgraph 26
- Pseudocode
 - Musterlösung 45
- Puzzle 67

R

- Rahmenhandlung 38
- Realisierung 46
- Redo 50, 68
 - SALABIM 63
- Referenzimplementierung 63
- rezeptives Lernen 35

S

- SALA 33
 - Erprobung 69
- SALABIM 59
 - Applets 49
- salabim.jar 61
- SALA-Lernprogramme
 - Erprobung 69
- SALSA 31
- SAMBA
 - bei Hundhausen 29
- Sample 66
- Scheduler 66
- Schlüsselwerte 35
- Schnittstelle 45
- Schwierigkeitsgrad 37
- Seitennutzer 92
- Seitenstruktur 87
- Sektion 39
- Self-Documentation 41
- Simple Beispielapplet 60
- SimpleModel 65
- Simulation 46
 - Modus 49
- Skriptsprache
 - ANIMAL 25
- Software-Engineering 40
- soziales Lernen 100
- Soziokultureller Konstruktivismus 29
- Standardpfad 39, 90
 - Notation 93
 - Nutzung 96
- Standardverfahren 50
- stepwise refinement 23
- Steuerung des Ablaufs 49
- Storyboard 30
- Struktur der Lernprogramme 87
- Studienfach 74
- Swing 60, 91
- swingall.jar 60

T

- Technik
 - Fragebogenkategorie 78
- Technische Probleme 75
- Testleistung 71
- Trace
 - OP SIS 27
- Transferförderung 36

Transkription 78
tutorielle Teile 35

U

Übung
 Modus 49
Übung zu Datenstrukturen 70
Übungseffekt 43
Übungsmodus
 Alg. in Action 23
 JFLAP 24
Undo 50, 68
 SALABIM 63

V

Vertiefung in eine Aufgabe 37
Verzweigungspunkte 90
 Binomial Heap 97

 Heapsort 96
Visualisierungswerkzeug
 ALVIS 30
Visuelle Programmierung 26
Vorbedingung 46
Vorlesung Datenstrukturen 70
Vortest 73
Vorwissen 74

W

Webbasiert 60
Webbrowser
 Übung im Umgang 75
Wissensbasierte tutorielle Hilfe 100
Wissensvermittlung 45

Z

ZIP-Archiv 91

Lebenslauf

Nils Günter Faltin,
geboren am 5.10.1967 in Haßfurt

1978 - 1987	Gymnasium Lauf, Abschluß: Abitur
01/1988 - 08/1989	Zivildienst, Diakonie Eltersdorf
09/1989 - 10/1989	Programmierer, Fa. Neugebauer
11/1989 - 12/1995	Studium der Informatik, F.-Alexander Universität, Erlangen. Abschluß: Diplom
02/1996 - 11/1997	Wiss. Mitarbeiter, Informatik VII, Univ. Erlangen. Forschungsprojekt zur parallelen Implementierung von Kommunikationsprotokollen.
12/1997 - 12/1999	Wiss. Mitarbeiter an der C. v. Ossietzky Universität Oldenburg, Abt. Prof. Gorny, im Projekt „Medienunterstütztes Studium der Informatik“. Entwicklung webbasierter Lernprogramme.
01/2000 - 09/2000	Wiss. Mitarbeiter in der Abteilung „Computer Graphics und Software-Ergonomie“ (Prof. Gorny). Mitarbeit in der Lehre.
10/2000 - 02/2001	Wiss. Mitarbeiter in der Abteilung „Prozessinformatik“ (Prof. Jensch). Mitarbeit in der Lehre.
01/2001-12/2001	Stipendiat der Heinz Neumüller Stiftung
seit 01/2002	Wiss. Mitarbeiter am Learning Lab Lower Saxony, Univ. Hannover (Prof. Wagner). Forschung zu internetbasierten Fernexperimenten für die Ingenieurausbildung.

