

Multimediale Algorithmen und Datenstrukturen Assessments

Diplomarbeit an der Albert-Ludwigs-Universität Freiburg

Fakultät für Angewandte Wissenschaften

Lehrstuhl für Algorithmen und Datenstrukturen



Markus Krebs

Betreuer: Stephan Trahasch

Betreuender Professor: Prof. Dr. Thomas Ottmann

Markus Krebs

Brunnenstraße 4

79400 Kandern-Holzen

Erklärung

Name: Markus Krebs

Matrikelnummer: 9547500

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt, und alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder unveröffentlichten Schriften entnommen wurden, als solche kenntlich gemacht habe. Außerdem erkläre ich, dass ich die Diplomarbeit nicht, auch nicht auszugsweise, bereits für andere Prüfungen angefertigt habe.

Freiburg, den 13. Mai 2004

Danksagung

Hiermit bedanke ich mich bei den Menschen, die mich bei der Erstellung der Diplomarbeit unterstützt haben. Bei Herrn Prof. Dr. Ottmann bedanke ich mich für die Möglichkeit die Diplomarbeit an seinem Lehrstuhl anzufertigen. Bei Stephan Trahasch bedanke ich mich für die sehr nette und immer angenehme Betreuung. Des Weiteren bedanke ich mich bei Tobias Lauer und Martin Danielsson für Tipps und Anregungen.

Weiterer Dank gilt auch meinem Kommilitonen Andreas Horstmann, mit dem ich gute und schlechte Zeiten des Studiums stets erfolgreich meisterte.

Mein besonderer Dank gebührt meiner ganzen Familie, die mir dieses Studium ermöglicht hat.

Euch allen verdanke ich es, dass ich hier und heute diese Arbeit abliefern kann.

DANKE!

Abstract

Die Arbeit befasst sich mit der Entwicklung und Realisierung eines Assessment-Systems zur Generierung, Durchführung und Evaluation von Aufgaben im Bereich Algorithmen und Datenstrukturen. Es werden Anforderungen an das zu entwickelnde System, psychologische Faktoren, verschiedene Tutortypen, mögliche technologische Realisierungen und Kategorien von Assessment-Systemen angesprochen.

Die genauere Betrachtung einiger existierender Assessment-Systeme erlaubt einen Vergleich mit den Anforderungen des zu entwickelnden Systems.

Ferner werden wichtige Konzepte erörtert, die für die Erstellung des Systems nötig wurden. Einzelne bedeutende Details der JAVA-Implementierung werden herausgegriffen, um die interne Programmstruktur näher zu erläutern.

Die Stärken und Schwächen des entwickelten Systems werden ausgearbeitet und einige Erweiterungsmöglichkeiten aufgelistet.

Inhaltsverzeichnis

1.	EINLEITUNG	1
1.1.	TECHNOLOGISCHE REALISIERUNG	1
1.1.1.	Web-basierte Systeme	2
1.1.2.	Stand-Alone Applikationen	2
1.2.	KATEGORIEN VON ASSESSMENT-SYSTEMEN	3
1.3.	ANFORDERUNGEN AN DAS ZU ENTWICKELNDE SYSTEM	3
1.3.1.	Generierung von Aufgaben (Vorbereitungsphase)	4
1.3.2.	Durchführung von Aufgaben (Durchführungsphase)	4
1.3.3.	Evaluation von Aufgaben (Evaluationsphase)	4
1.3.4.	Die Anforderungen im Überblick	4
1.4.	GLIEDERUNG DER ARBEIT	5
2.	GRUNDLAGEN UND STAND DER TECHNIK	6
2.1.	KOGNITIV PSYCHOLOGISCHE GRUNDLAGEN	6
2.1.1.	ACT – Theorie	6
2.1.2.	Tutortypen	8
2.2.	AKTUELLE FORSCHUNG	9
2.2.1.	Geschichte	10
2.2.2.	Probleme bzgl. des Feedbacks	10
2.2.3.	Probleme der Evaluation	11
2.2.4.	Möglichkeiten durch die Benutzung von Computern	12
2.2.5.	Systeme von morgen	12
2.3.	EXISTIERENDE SYSTEME	14
2.3.1.	Authorware	14
2.3.2.	CUE Assessment-System	16
2.3.3.	Exorciser	17
2.3.4.	JHAVÉ	19
2.3.5.	Perception	21
2.3.6.	QuizPACK	25
2.3.7.	TRAKLA2	27
2.3.8.	TSA: Thinking Skill Assessments	29
2.3.9.	Tabellarische Übersicht	31
2.4.	JEDAS	32
3.	ENTWURF	33
3.1.	ZIEL DER ARBEIT	33
3.1.1.	Erstellung von Applikationspaketen	33
3.2.	AUFGABENKONZEPT	35
3.2.1.	„Normal-Mode“-Aufgaben	35
3.2.2.	„Fault-Mode“-Aufgaben	35
3.3.	EDITIERUNGSKONZEPTE	35
3.3.1.	Vertex-Format-based	35
3.3.2.	Function-based	38
3.3.3.	Entscheidung	39
3.3.4.	Die Konzepte im Vergleich	40
3.4.	AUTOMATISCHE GENERIERUNG VON AUFGABEN	40
3.4.1.	Aussagenlogisches Konzept	40
3.4.2.	Bewertungskonzept	41
3.4.3.	Überblick	42
3.4.4.	Entscheidung	42
3.4.5.	Verwendung des Bewertungskonzeptes zur automatischen Generierung	42
3.4.6.	Suche im Zustandsraum	42
3.4.7.	Komplettes Verfahren	44
3.4.8.	Test	44
3.4.9.	Fluss-Diagramm	45
3.4.10.	Diskussion	46
3.5.	AUFGABENABARBEITUNGSKONZEPT	46
3.5.1.	Normal-Mode	46

3.5.2.	<i>Fault-Mode</i>	47
3.5.3.	<i>Navigation</i>	47
3.6.	FEEDBACK-KONZEPT	47
3.6.1.	<i>Bereiche der Knowledge-Base</i>	48
3.6.2.	<i>Problem: Folgefehler</i>	48
3.6.3.	<i>Anfrage an die KB</i>	48
3.6.4.	<i>Skalierbarkeit</i>	48
3.6.5.	<i>Konkrete Verwendung</i>	49
3.7.	PLUG-IN-KONZEPT	49
3.7.1.	<i>Vorraussetzungen</i>	49
3.7.2.	<i>Komponenten</i>	49
3.7.3.	<i>Daten</i>	50
3.7.4.	<i>Konfigurationsdatei</i>	50
3.8.	META-DATEN	50
3.8.1.	<i>Vorbereitungsphase</i>	51
3.8.2.	<i>Durchführungsphase</i>	51
3.8.3.	<i>Evaluationsphase</i>	52
3.8.4.	<i>Sicherheit</i>	52
3.9.	TUTORIELLE EVALUATION	52
3.9.1.	<i>Das Prinzip</i>	52
3.9.2.	<i>Schlussfolgerung</i>	53
4.	IMPLEMENTIERUNG	54
4.1.	PROGRAMME & GUI'S	54
4.1.1.	<i>GenEditor</i>	54
4.1.2.	<i>AutoGenerator</i>	55
4.1.3.	<i>WorkEditor</i>	55
4.1.4.	<i>EvalEditor</i>	56
4.1.5.	<i>EvalPlayer</i>	56
4.1.6.	<i>Meta-Daten</i>	57
4.2.	AUFGABENERSTELLUNG MIT DEM GENEDITOR	58
4.2.1.	<i>Normal-Mode-Aufgaben</i>	58
4.2.2.	<i>Fault-Mode-Aufgaben</i>	58
4.3.	PLUG-IN-HANDLER	58
4.3.1.	<i>Konfigurations-Datei</i>	59
4.3.2.	<i>Tabellarische Übersicht</i>	60
4.3.3.	<i>Einzelne Klassen und zu implementierende Methoden im Detail</i>	61
4.3.4.	<i>Datenverzeichnis</i>	61
4.4.	OPERATIONS-AUFZEICHNUNGSPRINZIP	61
4.4.1.	<i>Beispiel</i>	62
4.5.	FEEDBACK: KNOWLEDGE-BASE	62
4.5.1.	<i>Regeln für die Formeln</i>	65
4.5.2.	<i>Ablauf der Auswertung einer Formel</i>	65
4.5.3.	<i>Bemerkung</i>	66
5.	LEISTUNGSBEWERTUNG	67
5.1.	EINHALTUNG DER GRUNDSÄTZE DER ACT-THEORIE	67
5.2.	ERFÜLLTE ANFORDERUNGEN	68
5.2.1.	<i>Allgemein</i>	68
5.2.2.	<i>Vorbereitungsphase</i>	68
5.2.3.	<i>Durchführungsphase</i>	68
5.2.4.	<i>Evaluationsphase</i>	68
6.	SCHLUSSFOLGERUNGEN UND AUSBLICK	69
6.1.	STÄRKEN UND SCHWÄCHEN VON MAUDA	70
6.2.	ERWEITERUNGSMÖGLICHKEITEN	71
6.2.1.	<i>Statistik/Report</i>	71
6.2.2.	<i>Bündeln von Plug-Ins</i>	71
6.2.3.	<i>Meta-Daten-Browser</i>	71
6.2.4.	<i>Web-Service / SOAP</i>	71
6.2.5.	<i>Betrugselimination</i>	71

6.2.6.	<i>Automatische Generierung von Fault-Mode-Aufgaben</i>	72
6.2.7.	<i>Entwerfen weiterer Plug-Ins</i>	72
6.2.8.	<i>Tests an Personen</i>	72
6.2.9.	<i>Verlagerung der Korrektheit in die Knowledge-Base</i>	72
7.	ZUSAMMENFASSUNG	74
8.	ANHANG.....	76
8.1.	LITERATURVERZEICHNIS	76
8.2.	LINKSAMMLUNG	76
8.2.1.	<i>Allgemein</i>	76
8.2.2.	<i>Projektmanagement</i>	77
8.2.3.	<i>Assessment-Systeme</i>	77

1. Einleitung

Assessment-Systeme sind Lernumgebungen und bieten dem Nutzer die Möglichkeit, das im Kurs erworbene Wissen durch die Bearbeitung interaktiver Aufgaben anzuwenden und zu vertiefen. Dadurch wird einerseits der Lernprozess selbst unterstützt und andererseits geben die Resultate dem Lerner Auskunft über seine Stärken und Schwächen. Ferner können Professoren oder Tutoren durch teilweise bis vollautomatische Analysen der bearbeiteten Aufgaben herausfinden, welche Teilbereiche eines Problems den Lernenden Schwierigkeiten bereiten. Somit kann die Qualität von Kursen verbessert werden.

Ein Assessment kann in die drei Phasen Vorbereitung, Durchführung und Evaluation eingeteilt werden [4].

In der Vorbereitungsphase werden Aufgaben generiert. Dies kann sowohl durch eine Person als auch automatisch geschehen. Die Aufgaben sollen dabei unterschiedliche Schwierigkeitsgrade aufweisen, was eine besondere Herausforderung für die automatische Generierung darstellt.

In der Durchführungsphase werden die Aufgaben durch den Lernenden bearbeitet. Wichtig hierbei ist, dass Möglichkeiten des Betrugs eliminiert [5] werden. Dies kann zum Beispiel durch parametrisierte Fragen [5] erreicht werden, indem diese für jeden Lernenden unterschiedlich instanziiert werden. Um den Lernprozess zu fördern, sollte ein Computertutor vorhanden sein, der zu falschen Aktionen ein passendes Feedback gibt. Da der Lösungsraum einer Aufgabe unter Umständen sehr groß sein [14] kann, kann meist nur ein Teil der Lösungen durch den Computertutor erkannt werden. Ein Mittel, dem entgegenzuwirken, wäre, den Lernenden bei Abweichungen zu unerkennbaren Lösungen wieder durch bestimmte Nachrichten auf einen erkennbaren Pfad zu bringen [1]. Dies kann auf verschiedene Art und Weisen geschehen und könnte in verschiedenen Tutortypen [5] repräsentiert werden, die dem Lernenden zur Auswahl stehen. Des Weiteren ist es hier auch wichtig, psychologische Faktoren zu berücksichtigen.

In der Evaluationsphase wird die durch den Lernenden bearbeitete Aufgabe evaluiert. Dies kann wie in der Vorbereitungsphase sowohl durch eine Person als auch automatisch durchgeführt werden. Eine tutorielle Evaluation wird meist dann angewendet, wenn eine Aufgabe ganz oder teilweise nicht automatisch evaluiert werden kann. Das wäre zum Beispiel bei Fragen der Fall, die als Antwort ganze Sätze verlangen. Durch Weiterentwicklung in der künstlichen Intelligenz wäre es aber durchaus denkbar, dass auch solche Antworten zukünftig automatisch evaluiert werden könnten [2].

Die Anwendungsbereiche von Assessment-Systemen sind vielfältig. Sie reichen von Intelligenztests, Übungen an Universitäten bis hin zu Vorauswahlen zur Aufnahme an Hochschulen. ES wäre sogar eine künftige Ersetzung der Abschlussprüfung von Schulen durch Assessment-Systeme denkbar [2].

1.1. Technologische Realisierung

Vom technologischen Standpunkt aus kann man zwei verschiedene Formen von Assessment-Systemen unterscheiden. Dies sind zum einen die web-basierten Systeme und zum anderen die Stand-Alone-Applikationen. Beide werden in den folgenden Abschnitten beschrieben.

1.1.1. Web-basierte Systeme

Web-basierte Assessment-Systeme [4] gibt es sehr viele. Typische Fragen sind dort Yes/No, Multiple-Choice/Single-Answer (MC/SA), Multiple-Choice/Multiple-Answer (MC/MA), Fill-In mit numerischen oder Wort-Antworten, Pointing (Antwort sind ein oder mehrere Bereiche eines Bildes) oder Graphing (Antwort ist eine einfache Zeichnung). Die Technologien der Durchführungsphase reichen dort von HTML, CGI, JavaScript, Plug-Ins (Shockwave) bis Java-Applets.

Reines HTML wurde in der ersten Generation von web-basierten Assessments verwendet. Es erlaubt allerdings nur Yes/No und MC/SA Fragen, da nur reine Links möglich sind und die Handhabung doch relativ starr ist.

Eine sehr ausgereifte Technologie ist die Kombination von HTML und CGI. Sie wird in vielen Universitäten und kommerziellen Einrichtungen verwendet. Durch sie lassen sich alle oben genannten Typen von Fragen realisieren, da nun Eingabefelder, Radio-Buttons, Auswahllisten und Pop-Up-Menüs verwendet und server-seitig durch Übermittlung ausgewertet werden können. Allerdings stößt man auch hier auf Grenzen, da Aktionen wie Drag&Drop oder die Auswahl von Textsegmenten nicht realisiert werden können. Dies kann man nun durch JavaScript erreichen, welches client-seitig ausgeführt wird.

Reine JavaScript-Benutzung ohne CGI wäre ebenfalls denkbar und wäre auch ausreichend um alle Typen von Fragen zu realisieren. Selbst die Evaluation und die Bildung von Gesamtergebnissen könnten durchgeführt werden. Das Problem ist dann allerdings, dass die Antworten der Fragen client-seitig vorhanden sein müssten, so dass der Lernende diese durch Anzeigen des Source-Codes herausfinden könnte. Des Weiteren gibt es bei reiner JavaScript-Benutzung keine Möglichkeit die Evaluationsergebnisse aufzuzeichnen und an den Server zurückzuschicken. Somit wäre diese Methode nur für Self-Assessments, also zur eigenen Übung nutzbar. In Verbindung mit CGI lassen sich aber sehr gute Assessments realisieren.

Schlussendlich gibt es noch die Nutzung von Java-Applets in der Durchführungsphase. Dadurch dass Java eine vollständige Programmiersprache ist, lässt sich alles realisieren, was man sich an Fragen vorstellen kann. Jetzt sind unter Anderem auch interaktive Animationen möglich, was mit CGI nicht zu implementieren wäre. Im Gegensatz zur CGI-Technologie ist die Erstellung der Benutzerschnittstelle jedoch kompliziert zu entwickeln.

1.1.2. Stand-Alone Applikationen

Eine Alternative zu web-basierten Systemen sind eigenständige, in höheren Programmiersprachen programmierte Systeme. Diese Applikationen könnten durch den Lernenden zur Übung von Aufgaben lokal benutzt werden. Durch Integration des Aufgabengenerators kann der Student dabei verschiedene neue Aufgaben bearbeiten.

Ein eventuell erforderlicher Datenaustausch könnte hier per Web-Service [WA6] stattfinden. Dabei werden nicht HTML-Seiten zu einem Web-Browser geschickt, sondern es werden direkt Daten ausgetauscht, die auf entfernten Rechnern Funktionen auslösen können, so genannte Remote-Procedure-Calls (RPC). Eine Benutzung von SOAP als Schnittstellenprotokoll hat hierbei viele Vorteile. Es unterstützt zum Beispiel die Kommunikation zwischen den konträren Plattformen Sun/Java und Microsoft/.NET. Dies wird unter Anderem durch XML als Übertragungsformat erreicht. Ferner werden Web-Services per SOAP schon vielfach angeboten. Selbst in größeren Anwendungen, wie zum Beispiel Microsoft-Produkten, soll dieser Standard zukünftig verwendet werden.

1.2. Kategorien von Assessment-Systemen

Assessment-Systeme lassen sich in vier Kategorien aufteilen, wobei diese sich auch geringfügig überschneiden können.

Die erste Kategorie bilden die Systeme, die den Work-Flow unterstützen. Dabei wird der normale Übungsbetrieb einer Universität nachgebildet, um die Bearbeitungszeit von Übungsaufgaben zu verkürzen. Ein Beispiel dafür wäre WebAssign [3]. Hier werden Studenten, Tutoren, Korrektoren und System-Administratoren in den Übungsbetrieb mit einbezogen. WebAssign beschleunigt dabei den Übungsbetrieb von normalerweise fünf auf maximal zwei Wochen.

Die nächste Kategorie stellen die klassischen Self-Assessments dar. Dies sind Systeme, die aus den normalen Fragetypen (vgl. oben: 1.1 web-basierte Systeme) wie Yes/No, MC/SA, MC/MA, usw. Aufgaben bilden und vom Studenten abfragen. Typische existierende Systeme, welche auch später noch erklärt werden, sind hierfür Perception, TSA (Thinking Skills Assessment) oder CUE.

Eine Erweiterung der klassischen Self-Assessments sind die komplexen Self-Assessments. Hier werden keine einfachen Fragen gestellt, sondern der Student muss vielmehr direkt auf Datenstrukturen arbeiten. Somit muss er eine Lösung zu einem Problem selbst aufbauen. Beispiele für diese Kategorie sind Exorciser, TRAKLA2 und auch das zu entwickelnde MAUDA-System.

Adaptive bzw. intelligente tutorielle Systeme [WAS14] bilden die letzte Kategorie. In diesen Systemen werden sämtliche Aktionen und Informationen, die während der Durchführung einer Aufgabe entstehen, in einer Datenbank gesammelt. Auf Grund dieser Informationen kann dann z.B. individuelles Feedback gegeben oder es können zugeschnittene Aufgaben erzeugt werden. Dadurch soll ein Lehrer simuliert werden, der auf die Bedürfnisse jedes einzelnen Studenten eingehen kann, um deren Schwächen gezielt zu beseitigen. Diese hochadaptiven Systeme verlangen allerdings einiges an künstlicher Intelligenz. Des Weiteren können nicht alle Aktionen des Studenten richtig interpretiert werden.

1.3. Anforderungen an das zu entwickelnde System

Assessment-Systeme, wie schon in der Einleitung beschrieben, setzen sich aus drei Phasen zusammen. In dieser Arbeit soll ein komplexes Self-Assessment-System zur Generierung, Durchführung und Auswertung von Aufgaben im Bereich Algorithmen und Datenstrukturen in der Programmiersprache JAVA entwickelt werden. Das System soll zunächst exemplarisch für den Bereich Fibonacci-Heaps implementiert werden. Es sollte aber trotzdem auf die Skalierbarkeit wie z.B. auf Binomial-Queues, AVL-Bäume, Bin-Packing-Probleme, etc. geachtet werden. Zur Visualisierung der Datenstrukturen soll JEDAS (Java-EDucational-Animation-System) [WA1] als Front-End benutzt werden.

Die zu entwerfenden GUIs sollen sowohl Toolbars als auch Menus zur Bedienung aufweisen. Hinzu kommt, dass eine einfache Navigation innerhalb einer Aufgabe existieren soll, die es erlaubt, direkt zu bestimmten durchgeführten Aktionen zu springen.

1.3.1. Generierung von Aufgaben (Vorbereitungsphase)

Das System sollte eine automatische Generierung von Aufgaben unterstützen. Dazu ist ein Aufgabengenerator zu entwickeln, der Aufgaben unterschiedlichen Schwierigkeitsgrades erzeugen kann. Die generierten Aufgaben sollen in einem Repository abgespeichert werden, aus dem der Student später die Aufgabe beziehen kann. Die Parameter einer Aufgabe wie z.B. Schwierigkeitsgrad, Erstellungszeitpunkt, etc. sollen hierbei durch Meta-Daten in den Aufgaben gespeichert werden.

Ferner soll es möglich sein, dass auch eine Person eine Aufgabe „von Hand“ erstellen kann. Hierzu wird ein Aufgabeneditor benötigt, in dem Aufgaben komfortabel erstellt werden können. Teile der Aufgabe sollten automatisch generierbar sein. Der automatische Aufgabengenerator sollte also in den Aufgabeneditor integriert werden. Durch eine Replay-Funktion soll sowohl das teilweise als auch das vollständige Abspielen der Aufgabe durchgeführt werden können.

1.3.2. Durchführung von Aufgaben (Durchführungsphase)

In dieser Phase kann der Student neue Aufgaben aus dem Repository oder vom Aufgabengenerator anfordern und bearbeiten.

Die Bearbeitung durch den Studenten soll direkt auf der Datenstruktur durchgeführt werden, so wie dies bei komplexen Self-Assessments gefordert wird. Es sollen deshalb keine schriftlichen Beschreibungen zur Lösung eines Problems abgegeben werden, sondern es muss direkt dynamisch eine Lösungs-Animation erstellt werden.

Des Weiteren sollen verschiedene Feedback-Schemata erstellt werden, die der Student später auswählen kann. Das Feedback sollte den Aktionen angemessen sein und nicht erklären wie die richtige Aktion aussieht, sondern sollte viel mehr anzeigen, warum die aktuelle Aktion falsch oder richtig ist. Nach mehrmaligem falschem Ausführen der benötigten Aktion soll dem Studenten die richtige Aktion vermittelt werden.

Die Aufgaben sollen sowohl in der Bearbeitungsphase als auch nach Abschluss der Bearbeitung gespeichert werden können.

1.3.3. Evaluation von Aufgaben (Evaluationsphase)

Die letzte Phase soll es erlauben, Aufgaben automatisch bewerten zu lassen.

Des Weiteren sollen Aufgaben auch durch einen Tutor evaluiert werden können. Hierzu soll das JEDAS-Annotations-System in Verbindung mit JEDAS-Aufnahmen benutzt werden. Ferner sollen durch ein zusätzliches Kommentar-Feld Bemerkungen zu Aktionen separat, also unabhängig von der JEDAS-Aufnahme, gespeichert werden können.

Damit der Student die tutoriell evaluierte Aufgabe betrachten kann, soll ein Player implementiert werden, der es erlaubt, sowohl die Kommentare als auch die Aufnahme zu betrachten. Durch ein Navigationssystem soll es möglich sein, die Aufnahme teilweise oder komplett in Bezug auf gemachte Aktionen abspielen zu können.

1.3.4. Die Anforderungen im Überblick

Allgemein:

- Entwicklung exemplarisch für Fibonacci-Heaps, optionale Skalierbarkeit mittels Plug-In
- JEDAS als Visualisierung (Front-End)
- GUI: Menus, Toolbars, Kontextmenüs (Rechts-Klick)

- Einfache Navigation innerhalb einer Aufgabe

Vorbereitungsphase:

- automatische Generierung von Aufgaben unterschiedlichen Schwierigkeitsgrades
- Ablegen in einem Repository
- Meta-Daten
- Erstellung durch Person
- Teile der Aufgabe durch Person automatisch generierbar
- Integration des Aufgabengenerators
- Replay-Funktion: Abspielen der Aufgabe

Durchführungsphase:

- Anfordern und Bearbeiten von Aufgaben
- Verschiedene Feedback-Schemata mit angepasstem Feedback
- Speicherung von Aufgaben die in Bearbeitung sind
- Speicherung von komplett bearbeiteten Aufgaben

Evaluationsphase:

- Automatische Bewertung
- Tutorielle Evaluation durch JEDAS-Aufnahmen und zusätzlichen, von der Animation unabhängigen Kommentaren zu Operationen
- Player für tutoriell evaluierte Aufgaben

1.4. Gliederung der Arbeit

In Kapitel 2 wird auf psychologische Grundlagen, verschiedene Tutortypen und auf die aktuelle Forschung im Bereich Assessment-Systeme eingegangen. Es wird ein spekulativer Blick in die Zukunft gegeben und einige derzeit existierende Systeme vorgestellt, die hinsichtlich der Anforderungen an das zu entwickelnde System verglichen werden. Es handelt sich dabei sowohl um web-basierte als auch lokal ausführbare, eigenständige Systeme. Das Kapitel schließt mit einer kurzen Erklärung zum Java EDucational Animation System (JEDAS) ab.

In Kapitel 3 wird auf verschiedene wichtige Konzepte eingegangen, die im Bezug auf das zu entwickelnde System (im folgenden MAUDA-System genannt) ausgearbeitet werden mussten.

Kapitel 4 beschäftigt sich mit wichtigen Details der konkreten Implementation. Dabei werden zunächst sämtliche GUIs vorgestellt, um einen Eindruck zu bekommen, wie die Benutzerschnittstellen aussehen. Die folgenden Abschnitte erfordern zum Teil gute JAVA-Kenntnisse und richten sich vor allem an Dritte, die das Projekt erweitern wollen.

In Kapitel 5 wird das System sowohl bezüglich Einhaltung psychologischer Grundsätze als auch mit den Anforderungen verglichen.

Kapitel 6 gibt ein Resümee zur Arbeit, beleuchtet die Stärken und Schwächen des Systems und gibt Ideen für einige Erweiterungsmöglichkeiten.

Kapitel 7 schließt die Arbeit mit einer Zusammenfassung ab.

2. Grundlagen und Stand der Technik

2.1. *Kognitiv psychologische Grundlagen*

Ein besonders erfolgreicher Ansatz bei der Entwicklung von Assessment-Systemen stellt die ACT-Theorie aus der Psychologie dar, die im Folgenden beschrieben wird.

2.1.1. ACT – Theorie

ACT steht für „Advanced Computer Tutoring“ und beschäftigt sich mit dem Lernmechanismus aus psychologisch kognitiver Sicht und wurde von J. R. Anderson [5] entwickelt. Die Theorie unterteilt den Lernprozess in drei Stufen [WA2]:

- Erwerb deklarativen Wissens
- Aufbau prozeduralen Wissens
- Vertiefung prozeduralen Wissens

Deklaratives Wissen bedeutet Wissen von Theoremen wie z.B. was ein Fibonacci-Heap ist und was er repräsentiert. Prozedurales Wissen hingegen ist Wissen über den Ablauf bestimmter Operationen wie z.B. Delete-Min auf einem Fibonacci-Heap.

2.1.1.1. *Wissenserwerb*

Im Folgenden werden diese drei Stufen des Wissenserwerbs genauer erklärt.

Stufe 1:

In der ersten Stufe geht es um den Erwerb deklarativen Wissens, welches bei Fibonacci-Heaps in der Vorlesung beigebracht werden muss. Dieses Wissen befindet sich beim Menschen im Langzeitgedächtnis, wobei über die Anwendung und den Bezug zu diesem Wissen noch keine Informationen vorliegen. Das Wissen wird hierbei in Segmente, so genannten Chunks, unterteilt, welche einzeln im Gedächtnis zugreifbar sind. Bei einem Fibonacci-Heap könnte man sich hier vorstellen, dass man Wissen darüber hat, dass Knoten markiert sein können, Kinder haben, etc.

Stufe 2:

Als nächste Stufe folgt die Anwendung des prozeduralen Wissens, also der Aufbau von diesem. Hierbei müssen Produktionen gebildet werden, die einfach gesprochen IF-THEN Konstrukten in Programmiersprachen entsprechen. Diese Produktionen werden in einem Produktionsgedächtnis abgelegt. Somit ist für die Anwendung von prozeduralem Wissen sowohl eine Abfrage aus dem Langzeitgedächtnis als auch aus dem Produktionsgedächtnis notwendig.

Beispiel bei Fibonacci-Heap:

IF Knoten verliert Kind und Knoten nicht markiert

THEN Markiere diesen Knoten

Die ACT-Theorie nimmt an, dass Produktionsregeln nur gelernt werden können, wenn man sich mit deklarativem Wissen im Kontext von Problemlösungsaktivitäten beschäftigt.

Stufe 3:

Dieses erworbene prozedurale Wissen muss in einer dritten Stufe vertieft werden. Dazu gibt es in der ACT-Theorie vier Mechanismen und zwar Komposition, Prozeduralisierung, Generalisierung und Diskrimination. Die Komposition beschreibt hierbei die Verkettung von Produktionen wie z.B.:

IF Operation = Delete k

THEN trenne k von seinem Vater und füge k in Wurzelliste ein

IF Knoten verliert Kind und Knoten nicht markiert

THEN Markiere diesen Knoten

➔

IF Operation = Delete k und Vater von k ist nicht markiert

THEN Markiere Vater von k, trenne k von seinem Vater und füge k in die Wurzelliste ein

Durch diese Komposition werden einzelne Produktionen zu größeren Produktionen zusammengesetzt, welche jetzt in einem kognitiven Schritt verarbeitet werden können.

Auf Prozeduralisierung, Generalisierung und Diskrimination soll an dieser Stelle nicht weiter eingegangen werden.

Es wird angenommen, dass man sich sowohl deklaratives als auch prozedurales Wissen nur gut aneignen kann, wenn man es anwendet.

2.1.1.2. 8 Prinzipien

Farrel und Reiser haben die ACT-Theorie (J. R. Anderson 1983) genauer untersucht und dabei acht Prinzipien [1] für das Entwerfen von Tutoren herausgearbeitet, welche im Folgenden erklärt werden:

Grundsatz 1: „Represent student competence as a production set“

Das heißt, ein Tutor sollte die Kompetenz des Lernenden in Produktionen repräsentieren und somit ein akkurates Modell der Zielfertigkeit besitzen.

Grundsatz 2: „Communicate the goal structure underlying the problem solving“

Es wurde in der ACT-Theorie festgestellt, dass beim Problemlösen das Unterteilen in Ziele und Unterziele stark beteiligt ist und die Zielstruktur, um ein Problem zu lösen, nicht adäquat dem Studenten mitgeteilt wurde. Die Schlussfolgerung hieraus war, dass das herausstellen und vermitteln solcher Ziele durch Instruktionen erfolgen soll.

Grundsatz 3: „Provide instruction in the problem-solving context“

Dieser Grundsatz basiert auf der Forschung, dass Lernen kontext-spezifisch ist und somit Instruktionen während dem Lernen gegeben werden sollen. Leider konnte nicht ermittelt werden, wann diese Instruktionen gegeben werden sollen. Vor dem Problem oder mittendrin?

Grundsatz 4: „Promote an abstract understanding of the problem-solving knowledge“

Man hat herausgefunden, dass Studenten oft zu spezifisches Wissen erlernen und nicht richtig generalisieren können. Auf Produktionsregeln bezogen heißt das, dass die erlernten Regeln nicht ausreichend allgemein waren. Leider kann man nicht genau angeben, wie diese Regel erreicht werden kann. Praktisch gesehen macht es Sinn, die Abstraktionen in den Hilfe- oder Fehlermitteilungen zu platzieren.

Grundsatz 5: „Minimizing working memory load“

Man hat entdeckt, dass bei einer hohen Auslastung des Arbeitsgedächtnisses das Lernen der Zielinformation zu kurz kommt und somit eine Interferenz mit dem Lernen entsteht.

Grundsatz 6: „Provide immediate feedback on errors“

Dies ist ein umstrittener Punkt in der ACT-Theorie, da die Theorie verlangt, dass neue Produktionen aus Erfahrungen beim Problemlösen erstellt werden. Deshalb, je länger gewartet wird, bis ein Fehler korrigiert wird, desto länger ist die Spanne, bis im Problemlösen eine Produktion erzeugt und integriert wird. Die aktuelle ACT-Theorie erklärt, dass aus Problemlöseproduktionen gelernt wird. Somit untersucht der Lernende die entsprechende Lösung und bildet Produktionen davon. Infolgedessen macht es keinen Unterschied, ob alle kritischen Schritte zusammen erscheinen oder nicht. Das bedeutet, dass sie nur in der endgültigen Lösung repräsentiert sind. Somit ist dieser Grundsatz in der neueren ACT-R Theorie (J. R. Anderson 1993) nicht mehr gerechtfertigt.

Grundsatz 7: „Adjust the grain size of instruction with learning“

Man sollte die Instruktionen der Benutzerschnittstelle auf eine geeignete Größe bringen, indem man die Ziele nicht zu fein und nicht zu grob unterteilt. Dieser Grundsatz basiert auf der „Komposition“ in der ACT, da dort die Chunks aneinandergelinkt zu neuen Chunks komponiert werden und die Chunk-Größe deshalb abgestimmt werden muss.

Grundsatz 8: “Facilitate successive approximations to the target skill”

Der Lernende kann oft nicht alle Schritte ausführen, somit sollte der Tutor diese Lücken füllen. Je mehr der Lernende mit dem Tutor zusammenarbeitet, desto mehr unternimmt er selbst, um zur Lösung zu kommen und der Tutor rückt mehr und mehr in den Hintergrund. In der Praxis hat sich diese Annäherung oft bewährt.

2.1.2. Tutortypen

Eine große Frage bei der Erstellung von intelligenten Computer-Tutoren ist, wann sie ein Feedback geben sollen. Albert T. Corbett und John R. Anderson untersuchten dabei vier verschiedene Typen [5], die aus der ACT-Theorie folgen:

- Immediate Feedback & Error Correction
- Error-Flagging
- Feedback on demand
- No-Tutor

Die vier Typen werden im nachfolgenden detailliert beschrieben.

2.1.2.1. Immediate Feedback & Error Correction

Bei diesem Typ wird sofort ein Feedback gegeben. Enthält die aktuelle Aktion einen Fehler, wird eine Nachricht angegeben, die den Fehler erklärt aber nicht verrät, wie die korrekte Aktion wäre. Vielmehr wird dem Benutzer vermittelt, warum die Aktion falsch war. Außerdem wird der Fehler sofort automatisch rückgängig gemacht. Durch diese Art

wird der Lernende immer auf einem für den Tutor jederzeit erkennbaren Pfad gehalten, wodurch immer eine erfolgreiche Folgerung der Art des Fehlers durch den Tutor möglich ist.

Wird dieselbe Aktion mehrmals falsch durchgeführt, führt der Tutor automatisch die richtige Aktion aus und erklärt warum diese richtig ist.

Dieser Tutor-Typ hat sich bei kognitiven Programmieraufgaben und bei mathematischen Problemen 15 Jahre lang bewährt und war im Vergleich auch die effizienteste Lernmethode.

2.1.2.2. *Error-Flagging*

Hier wird wie bei Immediate Feedback & Error Correction immer sofort ein Feedback gegeben. Der Unterschied ist jedoch, dass Fehler nicht korrigiert werden und der Student somit im Arbeitsfluss nicht unterbrochen wird. Wird ein Fehler begangen, so muss der Student selbst den Fehler korrigieren.

Im Unterschied zu Immediate Feedback & Error Correction ist es hier möglich, dass eine im Kontext richtige Lösung von dem Studenten erarbeitet werden kann, die aber vom Tutor nicht erkannt wird. Der Tutor muss solche Lösungen mit Testfällen auf Richtigkeit testen und kann unter Umständen feststellen, dass eine Lösung korrekt ist, obwohl er manche Aktionen als falsch ausgewiesen hat.

2.1.2.3. *Feedback on demand*

Im Demand-Feedback bietet der Tutor nur dann Hilfe an, wenn der Student diese anfordert. Die Ratschläge sind die gleichen wie bei den beiden vorangehenden Tutortypen. Fordert der Student den Tutor auf, die Lösung auf Richtigkeit zu überprüfen, testet der Tutor die Aufgabe und akzeptiert jede funktionierende Lösung. Findet der Tutor einen Fehler, berichtet er den ersten gefundenen Fehler, den er entdeckt, und gibt Anweisungen zu diesem.

Durch dieses schrittweise Feedback ist es dem Studenten möglich, eine korrekte Lösung zu finden.

2.1.2.4. *No-Tutor*

Diese Art des Tutors gibt dem Studenten keine Anweisungen über die Korrektheit von ausgeführten Aktionen. Der Student kann lediglich anfordern, dass der Tutor einen Test durchführt, ob die für den Studenten fertig erscheinende Arbeit korrekt ist. Der Tutor gibt dabei nur „korrekt“ oder „inkorrekt“ aus, ohne Angabe wo sich der Fehler befindet.

Die Studenten werden dadurch ermutigt, solange an einer Aufgabe zu arbeiten, bis sie korrekt ist. Aufgrund der Tatsache, dass keine Hilfestellung möglich ist, kann hierbei auch eine falsche Aufgabe abgegeben werden. In diesem Fall präsentiert der Tutor eine richtige Lösung.

Somit ähnelt diese Art des Tutors stark der gebräuchlichen Art der „Hausaufgaben“, wie sie weit verbreitet sind.

Dieser Tutortyp ermöglicht somit bei empirischen Untersuchungen, die Effizienz der vorangehenden drei Typen zu messen, indem er als Vergleichsgruppe dient.

2.2. *Aktuelle Forschung*

Bei den normalen Hausaufgaben wird eine Aufgabe vom Tutor generiert, dann vom Studenten schriftlich bearbeitet und später vom Tutor ausgewertet. Dies wird unter

Anderem auch generate-solve-grade pattern [14] genannt. Dieser relativ lange Prozess verhindert ein sofortiges Feedback zu den erreichten Leistungen des Studenten. Durch die Verwendung eines automatischen Tutors wird die „solve-grade“ Phase durch eine Interaktion zwischen Student und Tutor ersetzt. Die Verwendung eines automatischen Tutors ist aber nur in mathematisch strukturierten Bereichen möglich, bei denen Lösungen und Aktionen algorithmisch überprüft werden können [14]. Somit ist der Bereich „Algorithmen und Datenstrukturen“ (Lehrstuhl von Prof. Ottmann) gut für automatisches Feedback geeignet.

2.2.1. Geschichte

Historisch gesehen, kann man die Entwicklung von Assessment-Systemen in zwei Bereiche unterteilen [14]:

- Verwendete Techniken zur Abfrage des Wissens
- Verwendeter Wissensbereich

Die verwendeten Techniken zur Abfrage des Wissens haben ihren Ursprung im Jahre 1920 als Sidney Pressey, ein pädagogischer Psychologie Professor an der Ohio State University, ein mechanisches Gerät für automatisches Testen und Auswerten konstruiert hat [WA7]. Dieses Gerät ähnelte einem Schreibmaschinenfahrwerk mit einem Fenster auf dem vier Fragen angezeigt wurden. Es gab vier Knöpfe, mit denen man die Auswahl treffen konnte, die dann mechanisch vermerkt wurde. Nach der Auswahl wurde die nächste Frage angezeigt. Am Schluss konnte die Wertung am Gerät abgelesen werden.

Durch technologischen Fortschritt in der Computertechnologie in den 50er und 60er Jahren [14] wurde eine bedeutende Veränderung und Verbesserung erreicht. Es gab jetzt Multiple-Choice-Fragen, die nach dem Alles-oder-Nichts- und dem Richtig-oder-Falsch-Prinzip arbeiteten. Die Voraussetzungen für solcherlei Verfahren war aber immer, dass Testen gleichbedeutend mit Lernen ist.

Man erkannte dann, dass mehr Interaktion mit dem Lernenden notwendig war als lediglich starres Richtig/Falsch Feedback. Diese Erkenntnis bzw. dieser Rückschlag führte dazu, dass noch vergleichsweise wenig Forschung im Bereich „Step-by-Step“ Feedback unternommen wurde.

Die Wissensbereiche, die für eine automatische Generierung in Frage kommen, sind auch sehr begrenzt. Schließt man die Technik des automatischen Generierens von Fragen aus einer großen Wissensbasis aus, kann das Generieren einer Vielzahl verschiedener Aufgaben nur in einem stark strukturierten, formalisierten, mathematisch beschreibbaren Wissensbereich [14] funktionieren. Auch diese Voraussetzungen sind für den Bereich „Algorithmen und Datenstrukturen“ (Lehrstuhl von Prof. Ottmann) erfüllt.

2.2.2. Probleme bzgl. des Feedbacks

Bis jetzt wurde nur diskutiert, dass ein Step-by-Step Feedback nötig wäre, aber die Frage ist, wie kann man sinnvolle Kommentare zu den gemachten Aktionen anzeigen, so dass sie zur Lösung eines Problems beitragen? Ein menschlicher Tutor kann auch zu teilweise richtigen Lösungen in völlig freien Sätzen erklären, worin der Fehler liegt. Ein automatischer Tutor hingegen kann nur zu einem eingeschränkten Bereich von Aktionen, der ihm vorher vermittelt wurde, Feedback geben. Da der Lösungsraum eines Problems (Gesamtheit aller möglichen Antworten) vom trivialen bis zum unerkennbaren reicht, müssen Strategien gefunden werden, die dieses Problem lösen [14].

Ein Lösungsansatz, wie er in Exorciser¹ verwendet wird, ist der Formular-basierte Ansatz [14]. Hierbei wird nicht direkt auf der Datenstruktur gearbeitet sondern auf einer Repräsentation dieser. Bei kontext-freien Grammatiken sind dies dann $n \times n$ Matrizen, die unterhalb der Diagonalen gefüllt werden müssen. Bei dieser Vorgehensweise können verschiedene Eintragungen die richtige Lösung repräsentieren. Ob eine Lösung korrekt ist, wird anhand der Beziehungen der Einträge erkannt, die auch schon während der Eingabe überwacht werden müssen.

Ein weiterer Lösungsansatz ist ein Feedback zu benutzen, dass Aktionen, die von der Musterlösung abweichen, mit entsprechendem Feedback sofort rückgängig macht² [5]. Dadurch wird gewährleistet, dass sich der Student immer auf einem für den automatischen Tutor erkennbaren Pfad aufhält. Dabei werden jedoch keine anderen Lösungswege akzeptiert.

Der vorherige Ansatz kann aber auch etwas gelockert werden, indem man dem automatischen Tutor mehrere Lösungsansätze zu verstehen gibt und das Feedback darauf fokussiert, den Studenten wieder auf den richtigen Weg zurück zu bringen.

Die in dem zu entwickelnden MAUDA-System verwendbare Strategie beruht auf dem letzten Lösungsansatz. Der Student kann sich auf nicht mit der Musterlösung identischen Pfaden bewegen (vgl. Abschnitt 3.6: Feedback-Konzept) und erhält trotzdem noch passendes für den automatischen Tutor erkennbares Feedback.

2.2.3. Probleme der Evaluation

Web-basierte Wissensaufgaben sind eine der ältesten Arten der Fernunterrichtung [13]. Diese sind mittlerweile auch sehr ausgereift. In diesen werden dem Studenten aus einem Pool von statischen Fragen mehrere ausgewählt, die er dann bearbeiten soll. Leider ist das Erstellen von Fragen immer noch sehr aufwendig. In früheren Systemen war das Erstellen von Fragen viel einfacher z.B. mit Stift und Papier, Microsoft Word, etc. als mit dem System. Aber auch bei den jetzigen Systemen ist der Aufwand noch sehr hoch, da das Erstellen der Inhalte und Antworten aufwendiger als die Eingabe selbst ist.

Eine Sammlung von statischen Fragen erzeugt des Weiteren zwei wohlbekannte Probleme [13]:

- Werden in einer Aufgabe für die ganze Unterrichts-Klasse dieselben Fragen gestellt, wird ein Betrug durch die Studenten geradezu heraufbeschwört. Betrug muss sich dabei aber nicht nur innerhalb einer Klasse abspielen sondern auch zwischen verschiedenen Klassen, da die Antworten ausgetauscht werden können.
- Bei Aufgaben, die der Student allein zuhause bearbeitet (Self-Assessments), wo der Betrug eine kleine Rolle spielt, beklagten sich Studenten jedoch des Öfteren über zu wenige Fragen, die gestellt wurden. Dadurch findet oft keine ausreichende Generalisierung zu einem Themenbereich statt.

Um diesen Problemen entgegenzuwirken, entwickelte man parametrisierte Fragen [13]. Hier werden nur Muster von Fragen erstellt, die dann später bei der Bearbeitung durch den Studenten mit Parametern instanziiert werden. Somit können aus jeder Frage viele bis unendlich viele verschiedene Fragen erstellt werden. Durch diese Methode wurde in

¹ Ein bereits existierendes System, dass später in Kapitel 2 noch betrachtet wird.

² Tutortyp: Immediate Feedback and Error Correction

verschiedenen Systemen festgestellt, dass das Problem des BetrÜgens nahezu eliminiert wurde.

Ein weiteres Problem bei der automatischen Evaluation ist, herauszufinden, was überhaupt evaluiert werden soll. Bei dem Web-basierten System Quiz-PACK³ [WAS9] ist es z.B. möglich, eine Frage mehrmals zu bearbeiten. Hier hat man den Weg gewöhlt, dass in die Evaluation nur die erste Antwort mit einfließt.

2.2.4. Möglichkeiten durch die Benutzung von Computern

Die Einsatzmöglichkeiten durch die Benutzung von Computern gehen weit über das Nachbilden des normalen Evaluationsprozesses hinaus [6]. Anfangs hatte man sich darauf beschränkt, dass man die normalen Übungsaufgaben eins zu eins im Computer nachgebildet hat mit der Idee, die Professoren und Tutoren zu entlasten. Einfache Fragetypen wie Multiple-Choice oder Matching-Fragen lassen sich gut automatisch evaluieren, nur offene Antworten, wie z.B. ganze Sätze als Antwort, mussten noch tutoriell bewertet werden.

Durch den Einsatz von Computern eröffnet sich allerdings eine Vielzahl neuer Möglichkeiten, die ohne PC praktisch nicht erreicht werden können oder zumindest nicht in einem zeitlichen Rahmen, der in guter Relation zum Aufwand steht.

Durch die Verwendung von Datenbanken, die zu jedem Studenten individuelle Informationen enthalten, können maßgeschneiderte Aufgaben generiert und individuelles Feedback gegeben werden. Es ist nun möglich detaillierte Informationen über die Stärken und Schwächen von einzelnen Benutzern zu gewinnen. Des Weiteren sind komplexe empirische Berechnungen durchführbar wie zum Beispiel Item-Schwierigkeiten, Durchschnittsberechnungen, Standardabweichungen und Vergleichsanalysen von einzelnen Personen oder Gruppen. Dies wird unter Anderem bei dem System Perception⁴ von Questionmark [WAS8] angeboten.

Ferner kann in Aufgaben Audio und Video verwendet werden, die Fragen der Art „Welches Lied hören Sie gerade?“ oder „Wie heißt der gezeigte Film?“ erst möglich machen. Durch Animationen und direktes Arbeiten auf Datenstrukturen wird die praktische Ausführung der gelernten Theorie vermittelt.

2.2.5. Systeme von morgen

Im Folgenden wird ein Szenario für die Zukunft beschrieben, so wie es Randy Elliot Bennett [2] ausgearbeitet hat.

Bennett's Spekulationen basieren auf dem zukünftigen technologischen Fortschritt. Dieser wird sich mit größter Sicherheit auf die Gesellschaft auswirken und somit auch auf den Bildungsprozess mittels Assessment-Systemen.

Die Änderung von Systemen wird sich dabei in drei Stufen unterteilen lassen:

2.2.5.1. Stufe 1: Einfachere Infrastruktur für elektronisches Testen

In der ersten Phase werden sich die herkömmlichen Assessment-Systeme in der Produktivität, also in der Leistung und Effizienz verbessern. Die Systeme werden

³ Ein bereits existierendes System, das in Kapitel 2 noch betrachtet wird

⁴ Ein existierendes Assessment-System, das in Kapitel 2 noch betrachtet wird.

vielfältiger anwendbar sein. Auch die Ansprüche seitens der Auftraggeber werden sich erhöhen.

Diese Stufe wurde durch aktuelle Systeme bereits erreicht, wohingegen mit der nächsten erst teilweise begonnen wurde.

2.2.5.2. Stufe 2: Qualitative Veränderungen

Durch technologischen Fortschritt, neuen psychologischen Ergebnissen und der vermehrten Forschung in der kognitiven Psychologie werden qualitativ bessere und kosteneffektivere Systeme entstehen, was eine signifikante Veränderung zur Folge hat.

Die Art der Fragen und der Antwortformate wird sich dramatisch verändern.

So werden Fragen multimedial gestellt und dadurch kognitive Fähigkeiten des Studenten genutzt, mit denen er schon von Fernsehen und Radio vertraut ist.

Die Antworten können aufgrund neuer Ergebnisse in der kognitiven Psychologie automatisch und ohne menschlichen Eingriff mit unterschiedlichen Schwierigkeitsgraden „on the fly“ erstellt werden und nicht wie bisher statisch. Sie müssen im Bezug auf bestimmte Randbedingungen im Aufsatzformat wiedergegeben werden. Außerdem können Antworten durch Computer kontrollierte Mikrofone und Kameras aufgezeichnet werden.

Durch die Möglichkeit, qualitativ hochwertige Multimedia-Daten zu übermitteln, steigt die Nutzungseffizienz von Audio und Video stark an. Dadurch sind auch umfassendere Messungen von Fähigkeiten möglich, die automatisch bewertet werden können. Nicht automatisch evaluierbare Antworten werden durch Zusammenwirken von Mensch und Computer erledigt. Des Weiteren wird sich die Software eigenständig trainieren, um zukünftig Entscheidungen bezüglich der Bewertung selbstständig treffen zu können.

2.2.5.3. Stufe 3: Überdenken von Zielen und Mechanismen

Durch ansteigende Kosten bei traditionellen Universitäten und der einfachen Bedienung von elektronischen Netzwerken wird dem Fernstudium immer mehr Kraft verliehen. Endgültige Abschlussnoten an Schulen werden nicht mehr durch einzelne Tests berechnet sondern vielmehr aus einer Serie von verschiedenen Messungen. Ein erster Schritt hierzu zeichnet sich bereits an der Universität Freiburg durch das neue Creditpoint-System ab. Durch intelligente Tutoren, Mikrowelten und Simulationen werden Umgebungen für den Studenten geschaffen, die man zur Übermittlung von Instruktionen und zur Zertifizierung der Kompetenz des Studenten in speziellen Gebieten nutzen kann.

Es wird spezielle Tools geben, die sowohl auf individuelles, isoliertes Lernen ausgerichtet sind als auch auf gemeinschaftliches Arbeiten.

Einfache Multimedia-Aufgaben werden in Virtual-Reality Simulationen übergehen, die komplexe Umgebungen wie Wissenschaftslabore oder Feldversuche wiedergeben. So wird Studenten die Chance gegeben, unter Bedingungen zu lernen, wie sie ihnen bei einer praktischen Ausführung begegnen würden.

Antworten zu Fragen können nun auch akustisch durchgeführt werden, da die Sprache vom Computer richtig verstanden wird. Durch zusätzliche Informationen, wie zum Beispiel aus Lippenbewegungen, können Mehrdeutigkeiten reduziert werden.

Die großen Änderungen in den Informations- und Kommunikationstechnologien beeinflussen ohne Zweifel, welche Fertigkeiten bewertet, gelernt und veranlagt sind.

In einer solchen wissensreichen Umgebung werden wir einfachen, günstigen und durchweg sofortigen Zugriff auf Informationen haben.

Elektronische Netzwerke werden physikalische Labore verdrängen und auf lange Sicht gesehen sogar Schulen, wie wir sie kennen.

2.3. Existierende Systeme

Auf dem Markt gibt es sehr viele Assessment-Systeme, wobei es sich bei den meisten um web-basierte oder in Java implementierte Systeme handelt, um die Plattformunabhängigkeit zu erreichen. Im Folgenden werden acht Systeme genauer erklärt und analysiert.

Zunächst wird zu jedem System definiert, wofür und von wem es wie entwickelt wurde. Danach wird auf die grobe Funktionalität des Systems eingegangen, wie z.B. verwendete Konzepte, mögliche Schwierigkeitsgrade oder interne Strukturen. Die nächsten Abschnitte befassen sich dann mit der Anzeige und der Bearbeitung, wobei auch auf das Feedback-Verfahren des Systems eingegangen wird. Abgeschlossen wird jeweils mit einem Fazit, in dem die Stärken und Schwächen des Systems erläutert werden und ein Vergleich mit den Anforderungen des zu entwickelnden MAUDA-Systems gegeben wird.

Der Abschnitt „existierende Systeme“ wird abgeschlossen mit einer tabellarischen Übersicht, in der die Anforderungen an das MAUDA-System noch einmal mit den existierenden Systemen gegenübergestellt werden.

2.3.1. Authorware

Authorware [WAS1, WAS2] wurde für die Erstellung interaktiver und multimedialer Lerninhalte von der Firma Macromedia entwickelt. In Authorware lassen sich Grafiken, Sounds, Animationen, sowie Text- und Video-Dateien integrieren. Die Lerninhalte lassen sich sowohl im Web als auch auf CD und DVD präsentieren. Die Präsentation über das Web benötigt dabei auf der Client-Seite mindestens das Authorware-Web-Player-Plug-In [WAS2], was vergleichbar mit dem von der gleichen Firma entwickelten Flash-Plug-In ist. Nach der Installation des Plug-Ins ist die Präsentation praktisch plattform- und browserunabhängig.

2.3.1.1. Konzept zur Erstellung von Lerninhalten

Die Schnittstelle zur Erstellung von Lerninhalten unterteilt sich in die zwei Fenster Design und Präsentation. Im Design-Fenster wird entworfen, was im Präsentations-Fenster passieren soll (siehe Abbildung 1).

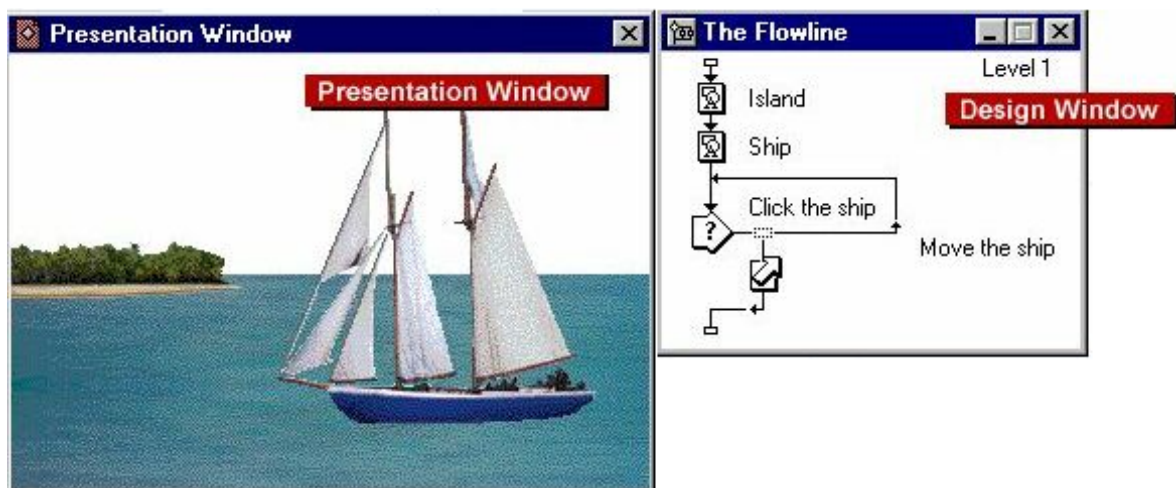


Abbildung 1: Authorware: Design Window + Presentation Window

Der Ablauf einer Präsentation wird durch ein editierbares Flussdiagramm bestimmt, in das man per Drag & Drop Icons mit bestimmten Funktionalitäten einfügen kann. Um eine grobe Vorstellung zu bekommen, um welche Icons es sich dabei handelt, werden im Folgenden einige exemplarisch erklärt:

- Icons zum Einfügen von Grafiken, Texten
- „Motion-Icon“ zum Erstellen von Bewegungsabläufen (z.B. ein eingefügtes Bild von A nach B in Geschwindigkeit C bewegen)
- „Decision Icon“ um Verzweigungsstrukturen aufzubauen, die aber nicht von Benutzereingaben abhängen (sequentiell, zufällig, berechnet in Abhängigkeit eines bestimmten Parameters)
- „Interaction Icon“ um vom Studenten eine Wahl anzufordern.
 - o Multiple Choice und sonstige Fragetypen
 - o Buttons, HotSpot, Tries limit, Time limit, Text-Eingabe, Drücken einer Taste, Pull-Down-Menu, uvm.
- „Calculation Icon“ zum Erstellen und Ausführen von selbst erstellten Skripts, wie z.B. „Wenn der Prozentsatz der korrekten Antworten über 85% liegt, dann zeige eine bestimmte Nachricht, ansonsten zeige eine andere“
- „Map Icon“ zum Gruppieren von bereits erstellten Fluss-Diagrammen, um diese in anderen Fluss-Diagrammen als ein einzelnes Icon zu verwenden
- „Digital Movie Icon“ (und „Video Icon“) zum Abspielen und Navigieren in Filmsequenzen
- „Sound Icon“ zum Hinzufügen von Sounds zur Präsentation
- „Wait Icon“, um eine bestimmte Zeit zu warten, bevor die nächste Aktion abgearbeitet wird

Somit unterstützt das System die Funktionalitäten eines klassischen Self-Assessments. Teilweise werden auch Grundzüge von komplexen Self-Assessments unterstützt, da auch das direkte Arbeiten auf einer Datenstruktur mit viel Aufwand eingeschränkt realisierbar wäre.

2.3.1.2. Fazit

Das System ist gut geeignet um relativ statische Lerninhalte zu Erstellen. Theoretisch wäre es möglich Aufgaben im Bereich Algorithmen und Datenstrukturen mit diesem System zu erstellen, allerdings auf einem Abstraktionsniveau, das in keinem Verhältnis zum Aufwand steht. Die Algorithmen und deren Animationen müssten komplett von Hand realisiert werden, d.h. ein Insert in einen Fibnacci-Heap müsste durch die Motion-Icon-Funktion komplett selbst definiert werden.

Selbiges trifft auch für das Feedback zu.

Eine automatische Generierung von Aufgaben ist dabei nicht möglich, sondern es ist lediglich eine zufällige Auswahl (Decision-Icon) aus statischen Aufgaben durchführbar. Eventuell könnte man zur automatischen Generierung die Skripting-Technik des „Calculation Icons“ verwenden, was aber dazu führen würde, dass sehr viel Programmier-Aufwand von Nöten wäre.

Da die Möglichkeit eine Aufgabe zu speichern nicht vorhanden ist, lässt sich keine tutorielle Evaluation realisieren. Wohingegen eine automatische Evaluation, begrenzt auf Anzahl richtiger Antworten, durchaus im Bereich des Möglichen wäre.

Die Möglichkeit Audio-Wiedergaben zu verwenden, um die Aufgabenstellung oder das Feedback sprachlich zu erklären, wäre eine sinnvolle zusätzliche Anforderung an das MAUDA-System, da dadurch das Lesen von Texten entfallen würde.

2.3.2. CUE Assessment-System

CUE Assessment System [WAS5] entstand aus CALM1 und CALM2, welche durch das CALM-Team (Computer Aided Learning In Mathematics) [WAS4] entwickelt wurden. Den Ursprung hat CALM an der Herriot-Watt-Universität in Edinburgh im Jahre 1985. CALM entwickelte Software für eine Vielzahl von Lehrfächern, inklusive Mathematik-Kursen. Durch Forschung in Bildung und Technik entstand das neueste Assessment-System CUE, welches in vielen Projekten in Großbritannien (UK) verwendet wird.

Zu den Stärken dieses klassischen Self-Assessment-Systems gehören:

- Mächtiger, leicht bedienbarer Editor, um Fragen zu erstellen und zu modifizieren
- ein PC und Web-Interface, um Fragen zu prüfen und Tests zu übermitteln
- ein Editor zum Erstellen von Tests, um verschiedene Fragen in verschiedenen Variationen zusammenzustellen
- Eine Vielzahl von verschiedenen Fragetypen (werden unten näher erklärt)
- Parametrisierte Fragen mit zufälliger Instanzierung
- Step-by-Step Design, um den Studenten zu Lösungen hinzubewegen
- Automatische Bewertung
- Flexibles anzeigen von Hinweisen und Ratschlägen

Das System unterstützt folgende Fragetypen:

- Multiple-Choice auf Texte und Bilder
- Single-Choice auf Texte und Bilder
- Fragen zu einer Audiowiedergabe, wie z.B.: „Welches Lied hören Sie gerade?“
- Fragen, die als Antwort mathematische Ausdrücke erfordern, wie z.B. die textuelle Eingabe des Satzes von Pythagoras als „ $\sqrt{a^2 + b^2}$ “. Dies ist die mächtigste Funktion von CUE, da hier auch Abweichungen wie „sqr“ anstatt „sqrt“ erkannt werden.
- HotSpots in Bildern, d.h. Fragen der Art „Klicken Sie den höchsten Berg dieser Landkarte an!“
- Matching-Fragen
- Bar-Graph: Ziehen eines Balkens mit der Maus

In den Antwortfeldern sind auch offene Antworten möglich, d.h. ganze Sätze. Diese werden dann mit Schlüsselwörtern verglichen um die Richtigkeit zu prüfen.

2.3.2.1. Anzeige und Bearbeitung

Eine Aufgabe besteht aus einer Sammlung von Fragen. Diese werden während der Bearbeitung sequentiell nacheinander angezeigt. Es ist aber auch möglich, dass mehrere Fragen gleichzeitig erscheinen.

Die Navigation erfolgt durch Vor- und Zurück-Buttons am oberen Bildschirmrand, wobei durch ein Pull-Down-Menu auch direkt zu einem bestimmten Fragenblatt gesprungen

werden kann. Verlässt man die gesamte Aufgabe durch „Exit“, erscheint ein Auswertungsfenster, in dem die Dauer der Bearbeitung, die Anzahl nicht bearbeiteter Aufgaben und die erreichte Punktzahl dargestellt wird.

2.3.2.2. Feedback

Wird eine Frage beantwortet erscheint ein Fenster, in dem erläutert wird, warum die Antwort richtig oder falsch ist. Dies kann durch Texte, Bilder oder sogar Animationen illustriert werden. Somit ist das Feedback doch relativ starr und ähnelt dem Feedback-Typ „Immediate Feedback and Error Correction“. Im Kontext betrachtet ist aber sowieso kein weiteres Feedback zu den möglichen Fragetypen denkbar.

2.3.2.3. Fazit

Ein praktisch fast ausgereiftes Programm auf dem Gebiet der web-basierten Assessments. Dies wurde auch schon durch mehrere Preise für das System bestätigt.

Negativ zu bewerten ist, dass sowohl Aufgaben in Bearbeitung als auch komplett bearbeitete nicht gespeichert werden können. Des Weiteren lassen sich Aufgaben auch nicht tutoriell bewerten.

Interessant ist das Konzept der Bilder und Animationen in Feedback-Nachrichten, welches eine sinnvolle zusätzliche Anforderung für das zu entwickelnde MAUDA-System darstellen würde.

2.3.3. Exorciser

Exorciser [14] ist ein System für die Einführungsveranstaltung „Computer-Theorie“ an der ETH Zürich, bei dem Studenten ihr Wissen über Grundkonzepte aus diesem Bereich testen können. Es besteht aus einem Grundsystem, das in Java implementiert ist, aus verschiedenen Plug-Ins, die das Hauptprogramm einsetzen kann, und einem Generator für Aufgaben.

Die Plug-Ins sind unterteilt in verschiedene Klassen. Im Moment besitzt das System Plug-Ins für reguläre Sprachen, Markov Algorithmen und kontextfreie Sprachen und Grammatiken. Jedes Plug-In hat seinen eigenen Aufgabengenerator, seine eigene graphische Schnittstelle für den Studenten (GUI) und einen eigenen Aufgabenbewerter.

Das Hauptprogramm von Exorciser verfügt über einen Aufzeichnungsmechanismus, der den Verlauf der Aktionen des Studenten komplett aufzeichnet. Bei der Speicherung der Aufgabe wird der gesamte Pfad bis zur Lösung aufgezeichnet. Somit ist es möglich, nach zu vollziehen, wie der Student eine Aufgabe gelöst hat.

Es gibt drei verschiedene Schwierigkeitsgrade für die Anforderung von Aufgaben:

- Beginners
- Advanced
- Experts

Diese können nach Angabe des Aufgabentyps ausgewählt werden, wobei dann eine Aufgabe des entsprechenden Schwierigkeitsgrades und -typs geladen wird. Des Weiteren ist es möglich, eine angeforderte Aufgabe selbst zu modifizieren.

2.3.3.1. Anzeige und Bearbeitung

Das Programm startet mit einer Übersicht der möglichen zu bearbeitenden Aufgabentypen. Nach Auswahl eines Typs erscheint eine Anzeige, die komplett Plug-In-spezifisch ist. Das bedeutet, dass der ganze Fensterinhalt durch das Plug-In spezifiziert wird.

Die Editierung einer Aufgabe wird in Exorciser über Kontextmenüs und Drag & Drop durchgeführt. So ist es unter Anderem möglich, wenn man einen Automaten bearbeitet, Zustände per Kontextmenü zu erstellen, Verbindungen zwischen Zuständen per Drag & Drop zu ziehen und die Übergangsbedingungen per automatisch aufklappendem Kontextmenü festzulegen.

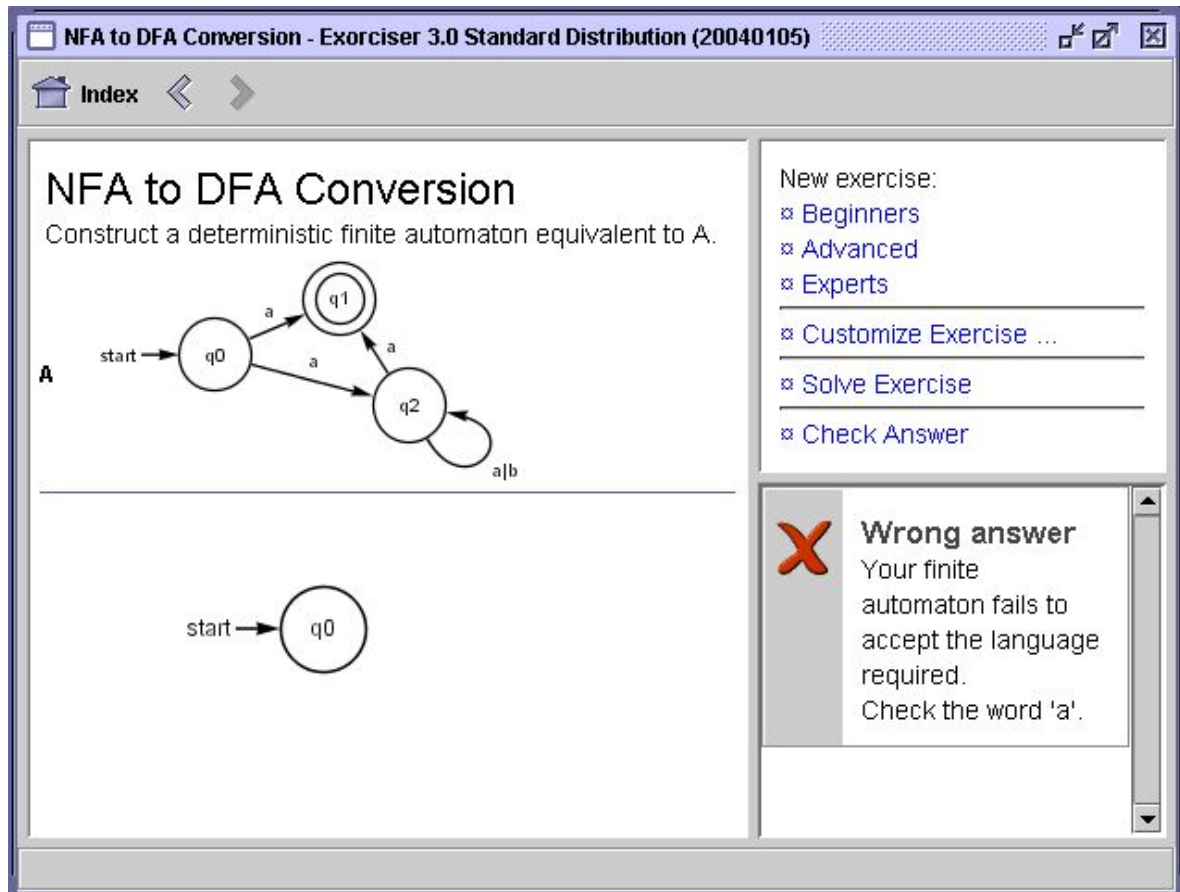


Abbildung 2: Exorciser NFA to DFA Conversion

2.3.3.2. Feedback

Ein Feedback kann über Links wie „Check Answer“ erreicht werden. Daraufhin wird überprüft, ob die bisher gemachten Aktionen korrekt sind, und falls nicht, wird ein kurzer Hinweis gegeben. Durch „Exhaustive check“ bei Markov-Algorithmen ist es möglich, den Algorithmus für eine begrenzte Menge von Eingaben auf Richtigkeit zu testen.

Des Weiteren ist es möglich, die Aufgabe durch „Solve Exercise“ komplett lösen zu lassen.

2.3.3.3. Fazit

Komplexes Self-Assessment-System, das durch seine Funktionalität und Plug-In-Anpassbarkeit besticht. Leider gibt es keine Auswahl für verschiedene Feedback-Typen. Weiterhin werden fertige Aufgaben auch nirgends automatisch abgelegt, um sie später noch einmal zu betrachten. Dies muss der Benutzer selbst erledigen. Daraus folgt, dass

auch keine generelle tutorielle Bewertung einer Aufgabe möglich ist. Das Laden von Aufgaben ist nicht immer möglich, da dies von dem Plug-In abhängt.

Allgemein gesagt, liefert das System keine endgültigen Evaluationsergebnisse in Form von erreichten Punkten oder Ähnlichem.

Dennoch könnten für das zu entwickelnde MAUDA-System viele Konzepte übernommen werden, wie z.B. das Plug-In-Konzept, das Aufzeichnungsprinzip oder die Modifizierung von Aufgaben durch den Studenten.

2.3.4. JHAVÉ

JHAVÉ [11, WAS7] ist die Abkürzung für “Java-hosted Algorithm Visualization Environment” und wurde von der Lawrence University entwickelt.

JHAVÉ ist ein in Java implementiertes Client-Server-System, das sich mit der Übertragung und Visualisierung von Algorithmen via Internet beschäftigt. Das System soll das Verständnis für Algorithmen bei Studenten fördern, indem es folgendes erlaubt:

- Animierte Darstellung der Algorithmen
- Zur Animation passende Texte darstellen
- Eingabemöglichkeiten für den Studenten
- Aktive Teilnahme an der Animation durch so genannte „Stop-and-Think“-Fragen

Somit gehört dieses System in die Kategorie der klassischen Self-Assessments, da die „Stop-and-Think“-Fragen vom Typ Multiple-Choice, Matching, usw. sein können. Es hat auch Grundzüge von komplexen Self-Assessments, da eine Animation abgespielt werden kann.

Man darf JHAVÉ aber nicht mit einem reinen Algorithmen-Visualisierungs-Programm verwechseln, da das System zur Visualisierung spezifische Algorithm-Visualization-Engines (AV-Engine) benötigt, die ihrerseits in beliebigen Programmiersprachen verfasst sein können. Die Kommunikation zwischen Client und Server findet komplett über Skript-Files statt, in denen Visualisierungs- und Eingabe-Kommandos abgelegt werden. Die Visualisierungskommandos müssen von der AV-Engine auf der Client-Seite entsprechend umgesetzt werden, während die Eingabe-Kommandos auf der Client-Seite einfache Fenster mit Eingabefeldern öffnen, deren Einträge zurück zum Server geschickt werden.

Somit ist es möglich, zunächst eine Auswahl an verfügbaren Aufgabenthemen bzw. Algorithmen darzustellen (Eingabe-Kommando), aus denen der Student eines wählt und anschließend per Visualisierungskommandos eine initiale, graphische, zu dem Thema passende Struktur aufgebaut wird. Infolgedessen ist es möglich, zum „Prim’s minimal spannende Bäume“-Algorithmus anzuzeigen, wie der Algorithmus von Kruskal auf denselben Daten arbeiten würde, indem eine Anfrage an den Server mit den aktuellen Daten geschickt wird und dieser die entsprechenden Visualisierungs-Kommandos zurückliefert.

Der Vorteil dieser Methode ist, dass die komplette Visualisierung in der AV-Engine erledigt wird und somit eine große Flexibilität bzgl. der Visualisierung gewährleistet wird.

2.3.4.1. Anzeige und Bearbeitung

Die Anzeige einer Aufgabe hängt wie im vorherigen Abschnitt beschrieben komplett von der AV-Engine ab. Wie solche Visualisierungen aussehen können, wird in den Abbildungen 3 und 4 einmal für das 0/1-Rucksack-Problem und andererseits für den Knuth-Morris-Pratt Zeichenketten-Such-Algorithmus illustriert.

Hierbei gibt es mehrere Anzeigebereiche für:

- Beschreibung zu dem Algorithmus
- Steuerfeld zur Ansteuerung der Animation (Geschwindigkeit, Zurück, Vorwärts, etc.)
- Visualisierung der Animation
- Fragefenster

Die eigentliche Aufgabe besteht darin, die ablaufende Animation inklusive den kontext-spezifischen Informationen genau zu verfolgen, denn an einer definierten Stelle stoppt die Animation und der Student muss eine Frage (Multiple Choice, Numerische Antwort, etc) zur aktuellen Situation beantworten. Danach läuft die Animation weiter bis zum nächsten Stopp.

The screenshot displays the JHAVE AV-Engine interface for the 0/1 Knapsack Problem. The top control panel includes buttons for 'Start', 'Step', 'Suspend', 'Beginning', 'End', 'Slower', 'Faster', 'Quiz Mode', 'Rewind', and 'QUIT'. The main display area shows a table of items and their weights and values, with a status bar indicating 'With objects 1 through 3 completed' and 'Testing object 4 with weight 2 and value 8'. A sidebar on the right contains a text box explaining the problem and a multiple-choice question window.

	0	1	2	3	4	5	6	7	8	9	10
1	0	0	0	0	0	0	5	5	5	5	5
2	0	0	0	0	10	10	10	10	10	10	15
3	0	0	0	3	10	10	10	13	13	13	15
4	0	0	8	8	10	11					
5											
6											

With objects 1 through 3 completed

Testing object 4 with weight 2 and value 8

The 0/1 knapsack problem can be defined in terms of a thief who enters the place she will rob with a single knapsack to carry away her spoils. This knapsack has a specified limit on the weight it can support without tearing. We will designate this weight capacity as **cap**. After cracking open the safe, the thief finds that it contains **n** items, each with a specific weight and value. We will designate the weight and value of the *i*th item as **wt_i** and **value_i**, respectively. For

What values are compared for entry at (4,6)

☐ #1 This is a trick question. Two values don't need to be compared for this entry.

☐ #2 11 and (11+12)

☐ #3 11 and (12+10)

☐ #4 10 and (10+8)

☐ #5 14 and (11+9)

No Answer Submitted Submit Response

Abbildung 3: JHAVE: AV-Engine: Beispiel 1: 0/1 Rucksack-Problem

2.3.4.2. Feedback

Ein Feedback zu Fragen wird nur indirekt gegeben, indem einerseits bei falsch beantworteten Fragen zunächst angezeigt wird, dass die Antwort falsch ist und andererseits, indem der richtige nächste Schritt animiert wird. Da dabei auch eine kontext-spezifische Information angezeigt wird, könnte man diese als Feedback betrachten.

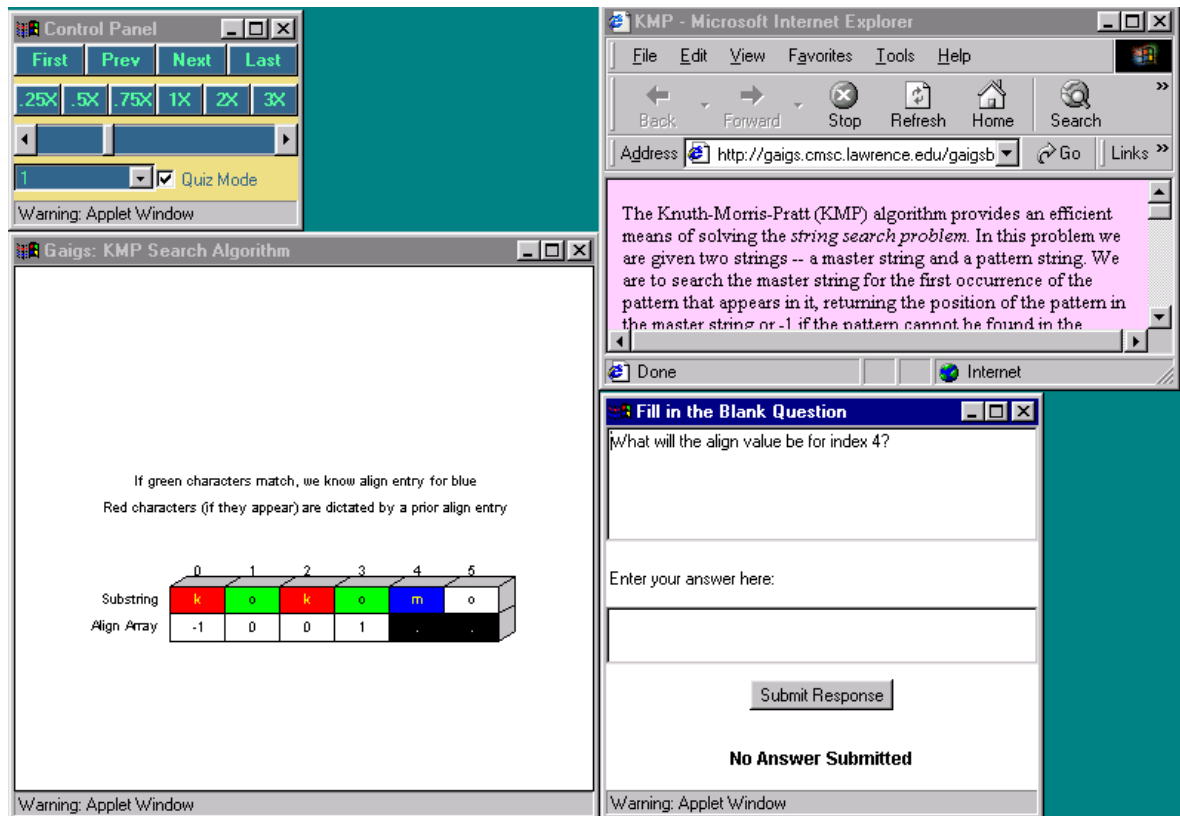


Abbildung 4: JHAVÉ: AV-Engine: Beispiel 2: KMP – Zeichenketten-Such-Algorithmus

2.3.4.3. Fazit

Eine vollständig automatische Generierung von Aufgaben mit „Stopp-and-Think“-Fragen ist derzeit nicht möglich, da der Fragegenerator den zugrunde liegenden Algorithmus komplett kennen müsste und dazu wäre ein hohes Maß an künstlicher Intelligenz nötig. Nichts desto trotz wird in diesem Bereich noch Forschung betrieben.

Sich in Bearbeitung befindende Aufgaben lassen sich nicht speichern. Eine automatische Evaluation der Fragen ist insofern möglich, als dass die Anzahl der richtig beantworteten Fragen gezählt werden kann.

Das Skript-Datei-Übertragungs-Konzept des JHAVÉ-Systems erlaubt eine hohe Flexibilität bzgl. der Anwendungsgebiete des Systems. Doch die statische Definierung der kontext-spezifischen Informationen zur aktuellen Animationssituation wirft auch Probleme auf. So lässt sich z.B. schreiben „der rote Knoten des Baumes wird sich um einen Level nach oben bewegen“, aber ein Satz wie „der rote Knoten des Baumes mit Schlüssel 4 wird sich um einen Level nach oben bewegen“ ist nicht möglich.

2.3.5. Perception

Perception [WAS8] wurde von der Firma Questionmark entwickelt und ist ein klassisches Self-Assessment-System um Tests, Quizze und Umfragen auf Intranets, dem Internet oder mit einem Windows PC zu erstellen und zu verteilen. Bei der Erstellung von Assessments sind keine Programmierkenntnisse notwendig.

2.3.5.1. Web-Version

Die Web-Version setzt sich aus drei Modulen zusammen:

- Ein Modul zur Erstellung von Prüfungen, Umfragen oder Tests. Hiermit lassen sich Fragen erstellen, ändern und zu Gruppen zusammenstellen.

- Einem Server-Modul mit dem sich völlig interaktive Befragungssysteme entwickeln lassen
- Einem System zur Erstellung von Berichten aus den Ergebnissen der Fragebögen

Diese Version enthält die typischen Fragenarten bzw. Antworttypen von web-basierten Systemen wie:

- Aufsatz (Essay)
- Lückentext
- Hotspot (Bild mit definierbarem Richtigkeitsrechteck)
- Matrix (Mehrfach-Matching-Frage)
- Multiple-Choice
- Mehrfachantwort
- Numerisch (mit Angabe eines Akzeptanz-Bereichs)
- Zuordnung von mehreren Möglichkeiten zu mehreren Optionsmöglichkeiten
- Text (mit Angabe mehrerer richtiger Wörter bzw. Sätze)

Das System ähnelt somit dem CUE-Assessment-System. Die Fragen werden hierbei in Gruppen auf einer Web-Seite angezeigt und müssen auch komplett abgearbeitet werden, bevor man die Antworten per „Submit“-Button einreichen kann.

Danach wird die gleiche Seite noch einmal dargestellt, aber diesmal mit einem Feedback und einer Anzeige der korrekten Antworten.

Per „continue“-Button gelangt man jetzt zum nächsten Fragenblock. Gibt es keinen Fragenblock mehr, wird das erreichte Gesamtergebnis in Prozent angezeigt, indem die Anzahl der Fragen mit der Anzahl der richtigen Antworten relativiert wird.

2.3.5.2. Windows-Version

Die Windows-Version besteht ebenfalls aus drei Modulen:

- Ein Autorenmodul, das mit dem oben genannten Erstellungsmodul gleichzusetzen ist.
- Ein unter Windows lauffähiges Modul zur Präsentation der erstellten Fragebögen
- Ebenfalls wie in der Web-Version ein Modul zur Erstellung von Berichten, das hier jetzt allerdings für Windows implementiert wurde.

Das Erstellen von Fragen wird mittels „Question Manager“ (Windows-Programm) durchgeführt. Hierzu muss zunächst ein neues Fragen-Thema angelegt werden, in welchem dann erstellte Fragen abgelegt werden können. Somit wird bereits bei der Fragenerstellung eine Gruppierung durchgeführt. Für das Erstellen von Fragen kann man einen einfach zu bedienenden Assistenten nutzen.

Im Session-Manager werden dann die erstellten Fragen oder ganze Fragethemen zu Tests zusammengefasst. Die Präsentation der Fragen lässt sich dabei auf zufällig einstellen. Des Weiteren gibt es eine „Bestanden/Durchgefallen Option“, in der man bestimmen kann, ab welchem Prozentsatz ein Test als bestanden gilt. Das Erstellen wird abgerundet durch diverse Angaben von Mitteilungen, die erscheinen sollen, wie z.B. Mitteilungen vor und nach Durchführung des Tests.

Der Test lässt sich nun für CD, Netzwerk oder für einen eigenständigen PC publizieren. Im CD-Modus wird der komplette Inhalt einer CD erstellt, die dann alle Programme und Daten enthält, um den Test darzustellen.

Die Durchführung eines Tests ist gleichzustellen mit der bei der Web-Version, nur dass hier ein spezieller Viewer (Windows-Programm) verwendet wird.

2.3.5.3. *Report-Modul*

Beide Versionen besitzen ein Report-Modul, das eine Vielzahl an Auswertungen zu den gemachten Antworten erlaubt. Folgende Report-Bereiche gibt es:

- „Assessment Overview Report“: Überblick der Ergebnisse einer oder mehrerer Assessments.
- „Gap Report“: Vergleichen zweier Sets von Ergebnissen und Unterschiede anzeigen. Hierbei lassen sich zur Bestimmung der zu vergleichenden Sets jeweils separate Filter-regeln definieren.
- „Item Analysis Report“: Berechnete Analysen im Bezug auf die klassische Test-Theorie (Zur Benutzung für Professionelle Test-Analytiker), z.B. Durchschnitts- und Korrelationsanalysen.
- „Score List Report“: Ergebnisse eines einzelnen Assessments anzeigen
- „Transcript Report“: Liste von Ergebnissen eines einzelnen Teilnehmers anzeigen
- „Coaching Report“: Detaillierte Ergebnisse eines Teilnehmers eines Assessments
- „Grade Book Report“: Tabelle der Teilnehmer und Assessments anzeigen mit den erreichten Punktezahlen.
- „Question Statistics Report“: Analyse einer Frage aus Test-Perspektive, inklusive Statistiken
- „Survey Report“: Analysen von Fragen aus Umfrage-Perspektive inklusive Häufigkeitsgraphen der Antworten
- „Export for Excel“: Konvertieren einer Liste von Ergebnissen für ein Assessment ins Microsoft Office Excel-Format

[Home](#) > [Reporter](#) > [Item Analysis Report](#)

Abbildung 5: Perception: Report (Web)



Abbildung 6: Perception: Report (Win)

2.3.5.4. Feedback

Zu jedem Fragetyp bzw. Antworttyp lässt sich ein speziell zugeschnittenes Feedback angeben. So lassen sich z.B. beim numerischen Antworttyp Feedback-Nachrichten für folgende drei Situationen angeben:

- Richtige Antwort
- Falsche Antwort
- Innerhalb der akzeptierten Abweichung liegende Antwort

Durch Angabe von Bewertungspunkten zu jeder Antwortsituation, die bei einer Frage auftauchen kann (wie z.B. die obigen drei Typen), lässt sich die Schwierigkeit der Beantwortung einer Frage positiv wie negativ gewichten. Somit lässt sich ein Schwierigkeitsgrad einstellen, der dann bei der End-Bewertung mit einfließt.

2.3.5.5. Fazit

Das System hebt sich hervor durch seine umfassenden Report-Funktionen, die nahezu nichts offen lassen. Die Sicherheit wird durch Angaben von Passwörtern zu Tests gewährleistet. Bis auf die Fragetypen „Aufsatz“ und „Erklärung“ wird alles automatisch evaluiert. Um diese beiden Typen zu evaluieren ist eine Person nötig.

Eine komplett automatische Generierung von Tests aus dem Fragenpool existiert nicht. Dafür bietet das System aber Funktionen an, um die Fragen von der Reihenfolge her zufällig anzeigen zu lassen. Des Weiteren gibt es so genannte Sprungblöcke um in Abhängigkeit von zuvor gemachten Antworten bestimmte Fragenblöcke abzufragen.

Für das zu entwickelnde MAUDA-System wäre ein Report-System, wie es hier erstellt wurde eine sinnvolle Ergänzung. So könnte man z.B. Statistiken über häufig gemachte Fehler bei Fibonacci-Heaps erstellen lassen, die dann die Stärken und Schwächen der Studenten in Balkendiagrammen darstellen.

2.3.6. QuizPACK

QuizPACK [WAS9] ist ein von TALER-Lab (Teaching And LEarning Research Lab) [WAS13] entwickeltes, web-basiertes Assessment-System, dessen Bezeichnung für "Quizzes for Parameterized Assessment of C Knowledge" steht. Es ermöglicht das Erstellen und Vermitteln von dynamischen Aufgaben unter Verwendung von parametrisierten Fragen. Hierbei wird der grobe Inhalt der Aufgabe spezifiziert, wie z.B. ein C-Programmstück mit Parametern. Der Lernende muss nun eine automatisch generierte Instanz des Programmes verstehen und Fragen dazu beantworten. Ob die Antwort richtig ist, wird überprüft, indem das Programm kompiliert und gestartet wird und die Ausgabe mit der Antwort des Studenten verglichen wird. Die Ergebnisse werden dann Server-seitig in einer benutzerspezifischen Datenbank gespeichert. Somit ist es möglich anzuzeigen, wie oft ein bestimmter Benutzer eine bestimmte Aufgabe besucht hat.

Es existieren mehrere verschiedene Typen von Aufgaben:

2.3.6.1. Aufgabentypen

Dissections:

Hier werden C-Programme vorgegeben. Zu den meisten Code-Zeilen gibt es anklickbare Felder, bei deren Anwahl eine Beschreibung bzw. Annotation angezeigt wird. Diese Beschreibung enthält eventuell auch Informationen über Beziehungen zu anderen Code-Zeilen, wie z.B.: „Erhöht die oben eingeführte Variable“.

[INFSCI 0015](#) > **Example 4.7**

Click on a green bullet to see the annotation

```
☐ /* Example: Simple Accumulator
☐   Author: Peter Brusilovsky
☐   Add numbers until 0 is entered. Prints the sum.
☐   */
☐
☐ #include <stdio.h>
☐ void main () {
☒     int sum = 0, nextnumber;
☐
☒     printf("Number: ");
☒     scanf("%d", &nextnumber); /* read first number */
☐
☒     while (nextnumber != 0) {
☒         sum += nextnumber;
☒         printf("Number: ");
☒         scanf("%d", &nextnumber);
☒     }
☒     printf ("Sum = %d\n", sum);
☐ }
```

This line reads an integer from the user and places it into nextnumber. We need to pre-read the first number in a sequence since a meaningful value of nextnumber is required **before** the checking the loop condition

Abbildung 7: QuizPACK: Dissection

Quizzes:

In diesem Bereich werden auch C-Programme vorgegeben, aber der Student muss nun entweder beantworten was die Ausgabe des Programmes ist oder welchen Wert eine Variable nach Ausführung des Programmes hat.

Applets:

Durch die Verwendung von JAVA-Applets werden Aufgabenstellungen wie „Wie viele Iterationen hat eine Schleife?“, „Was ist die Ausgabe?“ oder „Gibt es keine Ausgabe?“ visualisiert. Richtig zum Tragen kommen Applets aber erst bei Aufgabenstellungen zu Mengen.

Bei Mengenaufgaben wird ein Mengenausdruck, z.B.: „ $(A \cap B) \cup C$ “, vorgegeben. Im unteren Bereich werden 3 sich schneidende Kreise angezeigt, die jeweils die Mengen A, B und C repräsentieren. Nun muss der Benutzer die Segmente auswählen, die den Mengenausdruck repräsentieren.

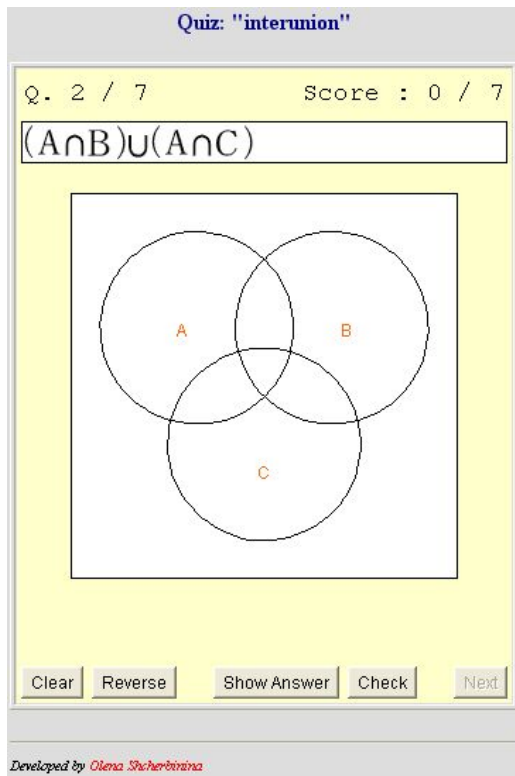


Abbildung 8: QuizPACK: Mengen-Applet

Weiterhin werden Applets für Fragestellungen zu mathematischen Ausdrücken verwendet. Hier muss man zu einem Ausdruck der Form „ $A = (-2 / -1.0) - 3$ “ angeben, in welcher Reihenfolge dieser Ausdruck abgearbeitet wird („-2“ lesen, „-1.0“ lesen, „/“ ausführen, „(-, bearbeiten, „-“, und dann „=“) und welche Zwischenergebnisse jeweils entstehen. Die aktuellen Werte der Variablen werden in eine Tabelle, die am unteren Rand des Darstellungsbereichs angezeigt wird, eingetragen. Die Ausdrücke, die man bearbeiten will, kann man selbst frei wählen, wobei für die in den Ausdrücken enthaltenen Variablen die Werte aus der Tabelle verwendet werden. Somit entsteht eine sehr große Menge an unterschiedlichen Aufgaben. Die Lernerfahrung wird hierbei durch Balken bzgl. der einzelnen Operatoren ausgedrückt. Diese erhöhen sich durch richtige Anwendung der Operatoren.

2.3.6.2. Anzeige und Bearbeitung

Eine Aufgabe wird ausgewählt, indem man in einem bestimmten Kapitel der dazugehörigen Vorlesung per Link eine Aufgabe auswählt oder in einer sehr großen Link-Sammlung eine entsprechende Aufgabe selektiert. Somit resultiert der Schwierigkeitsgrad einer Aufgabe aus dem entsprechenden Kapitel.

Die meisten Aufgaben basieren auf dem Prinzip, dass es ein Eingabefeld für Antworten gibt und einen dazugehörigen „Submit-Button“.

2.3.6.3. Feedback

Das Feedback ist meistens „on demand“ (vgl. Abschnitt 2.1.2: Tutortypen) und oft nur sehr spärlich.

2.3.6.4. Fazit

Das System besticht durch seinen sehr großen Aufgabenpool und die Vielzahl unterschiedlicher, aber meist kleiner Aufgaben. Das Feedback begrenzt sich meist nur auf Richtig oder Falsch und könnte somit etwas detaillierter sein. Aufgaben, die in Bearbeitung sind, können nicht gespeichert werden, was aber bei den meisten Aufgabentypen sowieso keinen Sinn macht, da fast alle sehr klein gehalten sind. Eine tutorielle Bewertung ist nicht möglich, dafür wird aber der Übungsgrad bestimmter Teilaktionen in Balkenform visualisiert und auch in einer Datenbank abgespeichert. Positiv anzumerken ist, dass man zum Beispiel bei den Aufgaben zur Evaluierung von mathematischen Ausdrücken selbst festlegen kann, welchen mathematischen Ausdruck einer Menge von Ausdrücken man als nächstes bearbeiten will. Dadurch kann man gezielt Ausdrücke auswählen, die Operatoren wie „!=“, „=“, „+=“, „-=“, usw. enthalten, in denen man noch nicht sehr viel Übung hat. Dieses Prinzip entspricht dem Anpassungs-Prinzip von Aufgaben bei Exorciser, bei dem die Aufgaben auch durch den Studenten verändert werden können.

Das Konzept der benutzerspezifischen Datenbank, in der individuelle Informationen über z.B. den Übungsgrad bestimmter Aufgaben gespeichert werden, ist vorbildlich und wäre eine sinnvolle zusätzliche Anforderung für das MAUDA-System.

Eine Kategorisierung dieses Systems ist nur sehr schwer möglich, da es Grundzüge von klassischen Self-Assessments, komplexen Self-Assessments und sogar Intelligenter Tutorieller Systeme hat. Es wird jedoch keine dieser drei Kategorien komplett abgedeckt. So sind die Quizzes klassischen Self-Assessments zuzuordnen, wohingegen die Applets in Richtung komplexer Self-Assessments zeigen. Das Benutzen einer personalisierten Datenbank ist ein typisches Merkmal von Intelligenten Tutoriellen Systemen.

2.3.7. TRAKLA2

TRAKLA2 [WAS10] ist eine Lernumgebung im Bereich Algorithmen und Datenstrukturen und wird momentan von der „Helsinki University of Technology“ (Laboratory of Information Processing Science) entwickelt. Dieses komplexe Self-Assessment-System entstand auf der Basis des WWW-TRAKLA-Systems.

Aufgaben können automatisch in den Bereichen, Suchalgorithmen, Baum-Traversion (pre-order, in-order, etc), Sortieralgorithmen, Suchbäume, Vorrangwarteschlangen (normale Heaps), Hash-Verfahren und Graphen-Algorithmen erstellt werden. Die Aufgaben werden dabei über das Internet durch Verwendung von Java-Applets bearbeitet.

2.3.7.1. Anzeige und Bearbeitung

Die Aufgabenstellung wird am oberen Bildschirmrand dargestellt, während darunter das Applet zur Bearbeitung der Aufgabe positioniert ist.

Wird z.B. eine Aufgabe zu Heaps bearbeitet, erscheint hier sowohl die Array-Darstellung des Heaps als auch der Heap als Baum. Dieser Heap-Baum ist aber immer vollständig, d.h. er enthält immer genau sieben innere Knoten und acht Blätter (siehe Abbildung 9).

Die Bedienung erfolgt einerseits über Drag & Drop, was bedeutet, dass z.B. zwei Schlüssel vertauscht werden können oder ein Schlüssel aus der Array-Darstellung in den Baum gezogen werden kann. Andererseits über Spezial-Buttons, aber nur dann, wenn es Aktionen gibt, die sich nicht mittels Drag & Drop realisieren lassen, wie z.B. Rotationen im Baum.

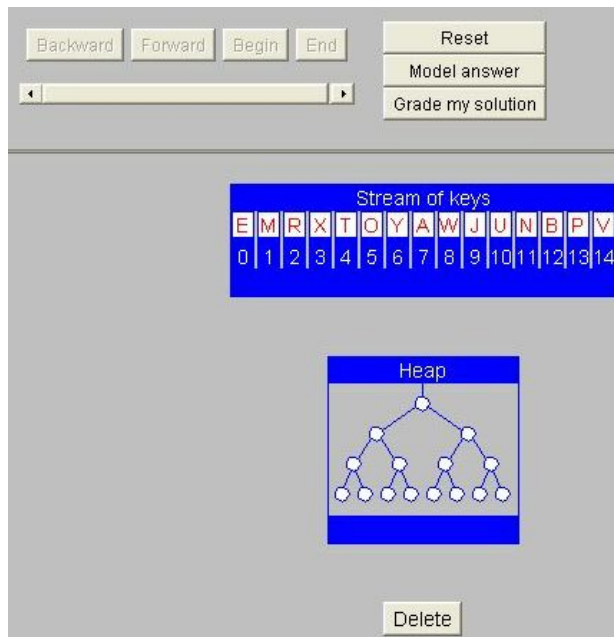


Abbildung 9: TRAKLA2: Heap-Applet

Die Navigation zwischen den durchgeführten Aktionen wird durch Anfang-, Zurück-, Vorwärts-, Ende-Buttons und durch einen Schieberegler erreicht. Über einen Reset-Button lässt sich eine neue Aufgabe, individuell zugeschnitten auf den Studenten, automatisch generieren.

Über einen Button „Model Answer“ lässt sich die Komplett-Lösung in einem eigenständigen Fenster anzeigen, in der auch entsprechend navigiert werden kann.

Mittels „Grade my solution“ lässt sich anzeigen, wie viele Einzelschritte aller notwendigen Schritte richtig durchgeführt wurden. Nun kann der Benutzer entscheiden, ob er seine Antwort zur Datenbank übermitteln will.

Führt man nach letzteren beiden genannten Aktionen („Model Answer“ und „Grade my solution“) noch Aktionen aus, kann man eine Aufgabe nicht mehr an die Datenbank übermitteln lassen. Dies bedeutet, dass eine Aufgabe komplett eigenständig vom Anfang bis zum Schluss gelöst werden muss.

Einige Informationen, wie z.B. Fehler, werden aufgezeichnet um daraus statistische Analysen zu erstellen. Diese Aufzeichnungen erlauben es, das System zu verbessern, individuell zugeschnittene Aufgaben zu erzeugen und den Lernprozess des Studenten besser zu verstehen.

2.3.7.2. Fazit

Das System gibt keinerlei Feedback zu durchgeführten Aktionen. Des Weiteren lässt sich eine Aufgabe, die in Bearbeitung ist, auch nicht speichern. Eine tutorielle Evaluation ist nicht möglich. Die generierten Aufgaben haben eine sehr starre Struktur, d.h., dass z.B. eine Aufgabe zum Bereich „Binäre Suche“ immer genau ein Array der Größe 29 hat und ein Heap immer das gleiche vollständig gefüllte Baum-Skelett besitzt. Die Auswertung einer Aufgabe erlaubt keinerlei Toleranzen zur Musterlösung. Bei „Binärer Suche“ kann es ja sein, dass es in einem folgenden Schritt keine exakte Mitte gibt. In diesem Fall wird nur das rechte Element (der beiden mittleren) akzeptiert. Die Aufgabenbeschreibung geht auf solche Fälle übrigens auch nicht ein.

Ferner werden die einzelnen Aktionen (z.B. bei Heap) auch nicht animiert sondern sofort in der Folgekonfiguration dargestellt.

Da dieses Projekt allerdings noch in der Entwicklung steckt, könnten in späteren Versionen diese Defizite beseitigt sein.

2.3.8. TSA: Thinking Skill Assessments

TSA [WAS11, WAS12] wurde an UCLES (University of Cambridge Local Examinations Syndicate) entwickelt. UCLES beschäftigt sich mit der Forschung und Entwicklung von Systemen zur Messung der wissenschaftlichen Begabung. Sie erarbeiteten dabei verschiedene Arten von gedanklicher Geschicklichkeit:

- kritisches Denken
- Problemlösen
- Kommunikationsfähigkeit
- Verständnis von Beweisen
- Numerische und räumliche Abläufe

Diese Tests sollen unter Anderem für Zulassungsentscheidungen zu bestimmten Universitäten verwendet werden.

TSA ist, wie QuizPACK und CUE, ein web-basiertes Assessment-System, das allerdings nur Multiple-Choice-Fragen mit genau fünf Antwortmöglichkeiten erlaubt. Somit gehört dieses System in die Kategorie klassischer Self-Assessments.

2.3.8.1. Anzeige und Bearbeitung

Beim Aufrufen einer Aufgabe erscheint zunächst eine Anweisungsbeschreibung, in der unter Anderem angesprochen wird, wie viele Fragen folgen, in welcher Zeit diese zu bearbeiten sind, wie die Bedienung funktioniert und was man zur Bearbeitung benutzen darf. Über einen Start-Link lässt sich dann mit der Aufgabe beginnen.

Jetzt wird (im Darstellungsbereich) links oben der Abarbeitungsstatus der verschiedenen Fragen dargestellt, während rechts die Fragestellung mit den möglichen Antworten präsentiert wird. Die möglichen Antworten können sowohl Texte als auch Tabellen, Bilder oder Diagramme enthalten. Über „Previous“- und „Next“-Links kann man zwischen den Fragen navigieren. Des Weiteren lassen sich die einzelnen Fragen auch direkt über den Abarbeitungsstatus-Bereich (siehe Abbildung 10: links oben) anspringen. Fragen können somit übersprungen werden und eventuell später bearbeitet werden. Die zu Beginn der Bearbeitung angezeigte Anweisungsbeschreibung lässt sich jeder Zeit über einen Link „Front Page“ anzeigen.

Bearbeitete Fragen, ob richtig oder falsch, werden im Abarbeitungsstatus-Bereich mittels einer durchgestrichenen Fragenidentifikationsnummer gekennzeichnet.

Da es sich hier um Denksport-Aufgaben handelt, kann kein bestimmter Schwierigkeitsgrad im Voraus gewählt werden. Vielmehr werden bewusst schwere und leichte Fragen vermischt. Dabei zählt aber die richtige Beantwortung einer schweren Frage nicht mehr als die einer leichten, d.h., alle Fragen werden gleich bepunktet.

2.3.8.2. Feedback

Zu jeder Antwort wird sofort in einem separaten Browser-Fenster ein Feedback gegeben, mit detaillierter Beschreibung, was richtig oder falsch gemacht wurde. In diesem Fenster wird auch die bis jetzt erreichte Punktzahl angezeigt. Die Punktzahl berechnet sich hier aus der Anzahl sofort richtig gemachter Antworten. Somit erhält man für eine Frage, die man einmal falsch beantwortet hat, keine Punkte mehr.

Menu


[Front Page](#)

1	2	3	4	5
6	7	8	9	10

[Finish Test](#)

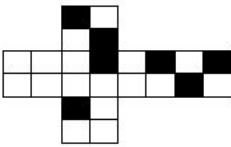
Question 6

The picture below shows a cube that I have made

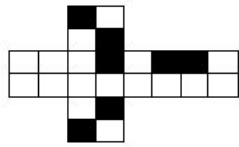


Which one of the shapes below, if cut out and folded, could make a cube the same as mine?

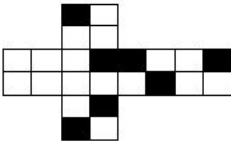
☐ A



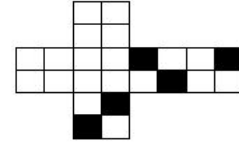
☐ B



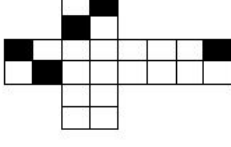
☐ C



☐ D



☐ E



[Previous](#)

[Next](#)

Abbildung 10: TSA

2.3.8.3. Fazit

Was die Fragetypen betrifft ist das System sehr starr, da nur genau ein Fragetyp unterstützt wird. Dennoch hat es aber gegenüber anderen web-basierten Systemen, wie QuizPACK oder CUE, einen Status über die Abarbeitung bestimmter Aufgaben, über dessen Anzeige auch einfach zwischen verschiedenen Fragen hin- & her- gesprungen werden kann. Da hier aber die Abarbeitungszeit begrenzt ist, ist dies sowieso erforderlich, damit der Student nicht viel Zeit durch die Navigation verliert.

Das Konzept des Abarbeitungsstatus könnte auch für das MAUDA-System übernommen werden.

2.3.9. Tabellarische Übersicht

	Authorware	CUE	Exorciser	JHAVÉ	Perception	QuizPACK	TRAKLA2	TSA	MAUDA
Kategorie (SA = Self-Assessment, ITS = Intelligentes tutorielles System)	Mischung aus Klass. SA und komplexem SA	Klass. SA	Komplexes SA	Mischung aus klass. SA und komplexem	Klass. SA	Klassisches - Komplexes SA und zum Teil ITS	Komplexes SA	Klass. SA	Komplexes SA
Umgebung	Web + Browser- Plug-In	Web	Java	Java + andere Sprachen	Web + Windows	Web	Web + Java	Web	Java
Automatische Generierung	Nein	nein	ja	Nein	Nein, nur zufällige Abfolge möglich	nein	Ja, allerdings nur starre Aufgaben	nein	Ja
Verschiedene Schwierigkeitsgrade bei autom. Gen.			Ja				Nein		Ja
Feedback	Indirekt	Ja, IF&EC	Ja, Demand- Feedback	Indirekt	Ja	Ähnlich zu Demand- Feedback	Nein	Ja, IF&EC	Ja, 4 Typen
Speicherung von Aufgaben die in Bearbeitung sind	Nein	Nein	Nein	Nein	Nein	Nein	Nein	Nein	Ja
Speicherung von komplett bearbeiteten Aufgaben	Nein	Nein	Ja	Nein, aber Ergebnisse → Datenbank	Nein, aber Ergebnisse → Datenbank	Nein, aber Ergebnisse → Datenbank	Nein, aber Ergebnisse → Datenbank	Nein, aber Ergebnisse → Datenbank	Ja
Automatische Evaluation	Ja	Ja	Ja	Ja	Ja	Ja	Ja	Ja	Ja
Tutorielle Evaluation	Nein	Nein	Nein	Nein	Ja	Nein	Nein	Nein	Ja
Statistik-/Report- Funktion	?	Ja	Ja	?	Ja	Ja	Ja	Ja	Nein
Interaktive komplexe Datenstruktur + Animation	Bedingt möglich	Nein	Ja, aber keine Animation	Nein, nur passive Animation	Nein	Nein	Ja, aber keine Animation	Nein	Ja

2.4. JEDAS

JEDAS [WA1] (Java EDucational Animation System) ist ein von der Universität Freiburg entwickeltes System, um anspruchsvolle 2-dimensionale Key-Frame-Animationen und Simulationen in Java-Anwendungen und Java-Applets durch zu führen.

Es lassen sich verschiedene graphische Objekte wie z.B. Kreise, Rechtecke, Polygone, Texte, etc.⁵ erstellen, welche sich dann durch Angabe von Pfaden sowohl in der Position als auch in der Farbe und Größe verändern lassen. Die Übergänge werden dabei flüssig durchgeführt, d.h., dass eine Änderung der Farben von z.B. Schwarz nach Weiß langsam durch Grautöne von statten geht.

Die Pfade von Positionsänderungen⁶ können durch verschiedene mögliche Angaben auch eine Kurve beschreiben. Durch Ausführung mehrerer aufeinander folgender Pfade lässt sich somit jeder beliebiger Pfad durch eine Szene realisieren.

Durch Sammelobjekte⁷ können Objekte an andere angehängt werden. Bewegt man nun das Sammelobjekt, bewegen sich auch alle angehängten Objekte. Somit ist es möglich z.B. einen kompletten Baum mit nur einer Pfadangabe zu bewegen.

Durch dieses Konzept lassen sich sämtliche Algorithmen und Datenstrukturen darstellen und insbesondere die Übergänge animieren.

Ein Annotationssystem gestattet es, Annotationen in Form von Texten, Kreisen, Rechtecken usw. von einer Person an ablaufende Animationen anzubringen. Die Annotationen können hierbei sowohl frei als auch an bestimmte Objekte gebunden sein. Durch ein Aufnahmemodul lassen sich diese annotierten Animationen in einem speziellen binären JEDAS-Animations-Format abspeichern.

Ein integrierter Player erlaubt es, diese Animationen abzuspielen. Dank einem Schieberegler kann man zu jedem beliebigen Zeitpunkt innerhalb der Animation springen.

In dem zu entwickelnden MAUDA-System soll JEDAS zur Darstellung und Animation der Algorithmen und Datenstrukturen verwendet werden.

⁵ ArcObj, OvalObj, PlineObj, Rect3DObj, RectObj, TextObj

⁶ Transition (siehe Trans-Objekt)

⁷ CompObj

3. Entwurf

3.1. Ziel der Arbeit

Um die Anforderungen an das System zu erreichen, mussten diverse Konzepte ausgearbeitet werden.

Das wohl wichtigste Konzept der Arbeit, das sich auch durch das gesamte Projekt erstreckt, ist das Erarbeiten eines Abstraktionsniveaus für die Editierung. D.h., wie sollte ein Student eine Aufgabe letztendlich bearbeiten.

Zur Erarbeitung eines solchen Konzepts ist die ACT-Theorie eine kleine Stütze. Aus Grundsatz 1 der ACT-Theorie (siehe Abschnitt 2.1.1: ACT-Theorie) folgt, dass man die Zielfertigkeit des Studenten in Produktionen fassen soll. Das bedeutet, dass der Lernende verschiedene Instruktionen nacheinander ausführen sollte. Nach Grundsatz 2 solle man Aktionen in Teilaktionen unterteilen und Grundsatz 7 besagt, dass man die Unterteilung auf eine angepasste Größe bringen soll. Grundsatz 5 schließlich besagt, dass man eine hohe Auslastung des Arbeitsgedächtnisses vermeiden soll.

Bezogen auf dieses Konzept heißt das, wenn man die Unterteilung der Editierung zu klein macht, kann es eventuell passieren, dass sehr viele Informationen zur Durchführung der Unterziele im Arbeitsgedächtnis abgelegt werden müssten.

Auf der Basis des Editierungskonzeptes muss nun überlegt werden, wie im Einzelnen eine Aufgabe aussehen soll. Dazu wird ein so genanntes Aufgabenkonzept benötigt.

Nach der Erarbeitung des Editierungs- und Aufgabenkonzeptes muss nun ein Konzept erstellt werden, welches sich mit der Generierung von Aufgaben eines bestimmten Schwierigkeitsgrades beschäftigt. Es ist zu überlegen, welche Methodik der Informatik oder Logik angewandt werden kann, um dies zu erreichen.

Für die Bearbeitung einer Aufgabe durch den Studenten ist es wichtig, zum Kontext passendes Feedback zu gemachten Aktionen zu bekommen (siehe ACT-Theorie Grundsatz 3). Laut dem umstrittenen Grundsatz 6 der ACT-Theorie sollte auf Fehler immer ein sofortiges Feedback gegeben werden. Dies wird später im Feedback-Konzept genauer erklärt.

3.1.1. Erstellung von Applikationspaketen

Unter Berücksichtigung oben genannter Konzepte sind mehrere verschiedene Programme zu erstellen.

3.1.1.1. *Generation-Editor*

Es ist ein Editor zur Erstellung von Aufgaben zu implementieren. Der Editor soll es Personen ermöglichen, beliebige Aufgaben zu generieren, zu editieren und zu speichern. Die gespeicherten Dateien sollen mit Meta-Daten versehen werden, so dass nach dem Speichern noch nachvollziehbar ist, wann und von wem die Aufgabe mit welchem Schwierigkeitsgrad erstellt wurde. Durch Integration der Module für die automatische Generierung von Aufgaben sollen sich solche auch durch einen Knopfdruck einfach im Editor erstellen lassen.

Des Weiteren soll ein einfaches Wechseln zwischen verschiedenen Plug-Ins möglich sein.

3.1.1.2. *Auto-Generator*

Hierbei handelt es sich um ein Programm, das nach Übergabe diverser Parameter völlig selbstständig Aufgaben erzeugt und abspeichert. Das Programm soll nach dem Starten keinerlei Benutzereingaben mehr erfordern. Es ist darauf zu achten, dass dieses Programm auch die Meta-Daten der erzeugten Aufgabe entsprechend füllt.

3.1.1.3. *Work-Editor*

Zweck dieses Programms ist es, den Studenten Aufgaben zur Bearbeitung zu präsentieren. Das Programm soll nicht nur alle vier Feedback-Typen (siehe Abschnitt 2.1.2: Tutortypen) unterstützen, sondern auch den Wechsel zwischen diesen während der Bearbeitung. Es soll für den Studenten möglich sein, Aufgaben jederzeit zu speichern, um später an ihnen weiterarbeiten zu können. Schließt der Student eine Aufgabe ab, soll diese automatisch gespeichert werden. In Abhängigkeit des gewählten Tutortyps soll die Aufgabe automatisch evaluiert und auf einer Skala von 0 bis 100 bepunktet werden.

3.1.1.4. *Evaluation-Editor*

Aufgaben, die durch den Studenten fertig bearbeitet aber noch nicht automatisch evaluiert wurden, können im Evaluation-Editor tutoriell bewertet werden. Dazu soll es möglich sein, Aufgaben ab zu spielen und nebenbei Annotationen zu der Aufnahme hinzu zu fügen. Zusätzlich sollen, unabhängig von der JEDAS-Animation, Kommentare zu einzelnen Aktionen angegeben werden können. Die Abspeicherung einer solchen Aufgabe soll aus zwei Komponenten bestehen. Zum einen die binäre JEDAS-Animations-Datei und zum anderen eine Datei, die die Meta-Daten, Kommentare und Steuerungsinformationen für die Animations-Datei enthält.

3.1.1.5. *Evaluation-Player*

Durch diesen Player sollen sich die tutoriell evaluierten Aufgaben des Evaluation-Editors für den Studenten abspielen lassen.

3.1.1.6. *Grundkonzepte*

Alle Programme sollen Plug-In-fähig sein, so dass verschiedene Aufgabentypen (wie z.B. Fibonacci-Heaps, Binomial-Queues, etc.) generiert, bearbeitet und evaluiert werden können. Sämtliche Abspeicherungsformate sollen dabei auf XML basieren. Durch Angabe zahlreicher Meta-Daten soll der genaue Status einer Aufgabe ermittelt werden können.

Für alle Programme (bis auf Auto-Generator) müssen GUIs entwickelt werden, die dem gängigen Standard von Anwendungen entsprechen. D.h., es sollen sowohl Menüs als auch Toolbars mit gebräuchlichen Icons [WA3] erstellt werden. Das Jedas-Animations-Fenster soll in die Programme integriert werden und über Zoom- und Scroll-Funktionen verfügen. Mit Hilfe von zahlreichen Navigationsmöglichkeiten soll ein einfaches Navigieren in den Aufgaben erreichbar sein. Mit Hilfe einer Replay-Funktion soll man die in Bearbeitung stehende Aufgabe von jeder beliebigen Position abspielen und auch wieder stoppen können. Mittels Undo- und Redo-Buttons sollen Aktionen rückgängig oder wiederhergestellt werden können.

Durch die Verwendung von JEDAS sollen Aktionen auf der Datenstruktur animiert dargestellt werden können. Die vorhandene Fibonacci-Heap Animationsklasse [10] soll erweitert und entsprechend verändert werden, so dass sie sich in das System integrieren lässt.

3.2. Aufgabenkonzept

Prinzipiell soll es zwei Arten von Aufgaben geben. Zum Einen soll es Aufgaben geben, bei denen der Lernende selbstständig Aktionen durchführen soll. Zum Anderen soll ihm eine fehlerhafte Animation vorgelegt werden, in der der Student den Fehler suchen soll. Im Folgenden wird der erste Typ als „Normal-Mode“ und der zweite als „Fault-Mode“ bezeichnet.

3.2.1. „Normal-Mode“-Aufgaben

Bei diesem Typ soll dem Studenten ein leerer oder vorgegebener Fibonacci-Heap vorgelegt werden, auf dem er mehrere Operationen korrekt durchzuführen hat. Die Operationen sind hierbei die Grundoperationen von Fibonacci-Heaps [WA5]:

- Insert x
- Delete x
- Delete-min
- Decrease-key x y

3.2.2. „Fault-Mode“-Aufgaben

Bei diesem Aufgabentyp soll dem Studenten ebenfalls ein leerer oder vorgegebener Fibonacci-Heap präsentiert werden, auf dem verschiedene vordefinierte Grundoperationen von Fibonacci-Heaps ausgeführt werden, in denen sich ein oder mehrere Fehler befinden. Aus diesen Fehlern gilt es den ersten zu finden.

Wie nun konkret eine Operation, wie z.B. Delete x, zur Editierung in Teilziele unterteilt wird, wird im folgenden Abschnitt beschrieben.

3.3. Editierungskonzepte

Wie schon in der ACT-Theorie im 7. Grundsatz angesprochen (siehe Abschnitt 2.1.1: ACT-Theorie) ist es wichtig, die Instruktionen der Benutzerschnittstelle auf eine geeignete Größe zu bringen. Dazu wurden zwei grundsätzlich in Frage kommende Konzepte bezüglich Fibonacci-Heaps mit ihren Problemen erörtert. Am Schluss werden diese beiden Konzepte anhand einer Gegenüberstellung ihrer Vor- und Nachteile miteinander verglichen.

3.3.1. Vertex-Format-based

Bei einem Fibonacci-Heap werden die Knoten intern in einem bestimmten Format gespeichert. Das Knotenformat eines Fibonacci-Heaps [WA5] sieht folgendermaßen aus:

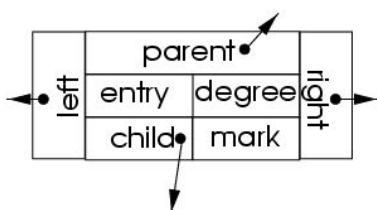


Abbildung 11: Editierungskonzept: Fibonacci-Heap: Knotenformat

Somit könnte z.B. eine Insert-Operation in einzelne Knotenformatsänderungen aufgeteilt werden. Die graphische Bedienung wäre folgendermaßen denkbar:

Man selektiert im Fibonacci-Heap einen bestimmten Knoten, woraufhin das Format dieses Knotens angezeigt wird. Nun lässt sich per Rechts-Klick auf einen Eintrag des Knotenformats ein Kontextmenü öffnen, in dem die möglichen Änderungen angezeigt werden. Für left, parent, right und child wäre das der Schlüssel (entry) eines anderen Knotens, während für mark „true“ oder „false“ ausgewählt werden könnte. Entry und degree könnte man per Text-Eingabe auf einen bestimmten Wert setzen. Knoten, die in der Wurzelliste des Fibonacci-Heaps stehen, hätten in parent einen „null“-Eintrag, und bei Knoten ohne Kinder wäre der child-Eintrag „null“.

Somit ergäbe sich für jede auszuführende Operation eine Kette von Knotenformatsänderungen.

Diese Art der Unterteilung in Unterziele wäre also sehr fein und würde ein hohes Maß an Interaktion benötigen.

Beispiel:

Insert y zwischen 2 Wurzelknoten x und z mit der Annahme, dass der Default-Wert von parent, child, left und right „null“ ist, und mark auf „false“ steht.

- y.entry = y
- y.left = x.entry
- y.right = z.entry
- x.right = y.entry
- z.left = y.entry

Diese Art der Editierung wirft allerdings einige Probleme auf:

3.3.1.1. Interaktion

Wie man im Beispiel sehen kann, benötigt ein einfaches Insert bereits fünf Interaktionen mit dem Benutzer. Man erkennt sofort, dass bei einem komplexen Heap und einer „Delete-min“-Operation die Anzahl der durchzuführenden Aktionen schnell explodiert. Im Bezug auf die ACT-Theorie wären die Chunks zu klein gewählt, und das Arbeitsgedächtnis wäre viel zu stark mit dem Vermerken von bereits durchgeführten Aktionen ausgelastet.

3.3.1.2. Visualisierung

Wird z.B. ein Knoten Kind eines anderen Knoten, dann muss eine Änderung des parent-Zeigers nicht unbedingt den child-Zeiger des betreffenden Knotens ändern. Dies müsste von dem Benutzer selbst erledigt werden. Wird eine solche gegenseitige Änderung nicht ausgeführt, erhält man enorme Fehler in der Struktur, die gar nicht mehr visualisiert werden können.

Somit können Zwischenschritte dem Benutzer nicht korrekt visualisiert werden, wodurch er keinerlei Feedback darüber erhält, wie sich diese auf die Datenstruktur auswirken.

3.3.1.3. Abarbeitungsreihenfolge

Die Reihenfolge der Abarbeitung ist nicht fest, d.h., bei obigem Beispiel ist es egal, ob man zuerst x.right = y.entry oder z.left = y.entry setzt. Bei einer späteren Evaluation müsste man somit Änderungen in Gruppen zusammenfassen und zwar so, dass die

Reihenfolge innerhalb einer Gruppe von Knotenformats-Änderungen beliebig ist. Erst wenn eine Gruppe komplett abgearbeitet ist, darf mit der nächsten Gruppe begonnen werden.

3.3.1.4. Skalierbarkeit

Dieses Verfahren ist sehr Problembereich-spezifisch und ließe sich nur auf Baumstrukturen gut skalieren. Aufgaben wie z.B. Bin-Packing würden ein komplett anderes Design der Programmstruktur benötigen.

3.3.1.5. Mehrdeutigkeiten bezüglich der Ausführung

Viele Operationen sind zwar im Algorithmus deterministisch aber nicht in der Ausführung. Dadurch entstehen korrekte Mehrdeutigkeiten, die später bei der Evaluierung viele Probleme aufwerfen würden:

Delete-min:

Im Algorithmus wird das consolidate bei dem aktuellen Minimums-Zeiger gestartet, was bedeutet, dass die ersten beiden Kandidaten für eine link-Operation der Minimums-Zeiger selbst und sein rechter Nachbar sind. Theoretisch kann das consolidate aber an jeder beliebigen Stelle beginnen, da nur gefordert wird, dass Bäume gleichen Ranges verlinkt werden. Man könnte vielleicht auf die Idee kommen und sagen, dass man an dem am weitesten links befindlichen Knoten beginnt, aber dann wäre die Einfügestelle abhängig von der Visualisierung, denn intern gibt es keinen „linkesten“ Knoten, da die Wurzel-Knoten in einer zirkulären Liste verkettet sind.

Insert:

Bei einem Insert ist nicht festgelegt, an welcher Stelle der neue Knoten eingefügt werden soll. Im Algorithmus hingegen wird er rechts neben dem aktuellen Minimums-Zeiger eingefügt.

Delete:

Hier stellt sich die Frage an welcher Stelle die Kinder in der Wurzelliste eingefügt werden sollen. Dieses Problem ergibt sich aber nur, wenn ein innerer Knoten entfernt wird, denn ein Entfernen aus der Wurzelliste hat zur Folge, dass die Kinder zwischen den 2 Nachbarn des Entfernten Knotens eingefügt werden. Im Algorithmus wird, beim Entfernen eines inneren Knotens, wie vorher auch schon, rechts neben dem aktuellen Minimums-Zeiger eingefügt.

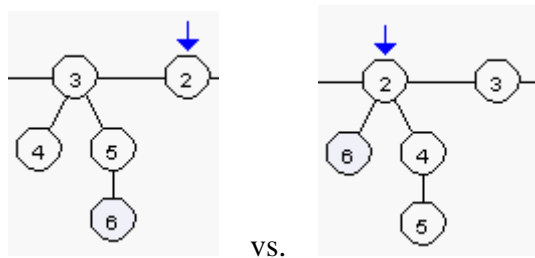
Decrease-key:

Wird ein Knoten unter den Wert seines Vater-Knotens herabgesetzt, ergibt sich dasselbe Problem wie bei einem Delete auf einen inneren Knoten.

Das größte Problem beim Nicht-Determinismus ist das spätere Erstellen von Aufgaben mit einem bestimmten Schwierigkeitsgrad, da sich durch ihn der Schwierigkeitsgrad beliebig verändern kann.

Beispiel:

Hat man mehrere Inserts und macht danach ein Delete-min, und es ist dem Lernenden überlassen, wo er die Knoten einfügt und an welcher Stelle er mit dem consolidate anfängt, dann wären folgende zwei Resultate korrekt:



Wie man sehen kann, variieren die Fibonacci-Heaps sehr stark. Wäre in der Aufgabenstellung nun eine Delete-min gefordert, wäre das im linken Fibonacci-Heap viel einfacher als im rechten. Gleiches gilt auch für z.B.: Delete 5.

Dieses einfache Beispiel verdeutlicht, dass wenn man Mehrdeutigkeiten zulässt, es keine Möglichkeit gibt, Aufgaben eines bestimmten Schwierigkeitsgrades zu erzeugen.

Es ist natürlich möglich dem Benutzer vorher mitzuteilen, dass sich sämtliche Aktionen immer auf den rechten Nachbarn des aktuellen Minimums-Zeigers auswirken. Doch dann stellt sich die Frage, was der Benutzer machen soll, wenn er vergisst den Minimums-Zeiger nach einer Operation, die ihn verändert (z.B. Delete-min), zu aktualisieren. Theoretisch wäre es ja dann möglich, dass der Minimums-Zeiger auf einem inneren Knoten steht.

Somit müssten ihm auch eine Vielzahl von Spezialfällen erklärt werden.

3.3.2. Function-based

Bei diesem Ansatz werden Operationen in funktionale Teiloperationen zerlegt. Im Bezug auf den weiter oben besprochenen Vertex-format-based-Ansatz heißt das, dass mehrere Knotenformats-Änderungen zu Gruppen zusammengefasst werden, die dann ausgewählt werden können.

Die Bedienung im GUI könnte man sich dann so vorstellen:

Man selektiert im Fibonacci-Heap einen Knoten per Rechts-Klick, woraufhin ein Kontextmenü aufklappt, in welchem die verschiedenen funktionalen Teiloperationen, die zu dem Knoten passen, aufgelistet werden. Man wählt eine davon aus, die dann eine Serie von Aktionen auslöst. Da nicht alle Operationen im Kontext zu einem Knoten stehen müssen, ließe sich ohne sich auf einem Knoten zu befinden ein Kontextmenü öffnen, welches die Auswahl entsprechender Operationen ermöglicht.

3.3.2.1. Variante 1

Die 1. Variante dieses Ansatzes enthält folgende funktionale Teiloperationen:

Funktion	Auswirkung
setkey x y	Ändert den Schlüsselwert des Knotens x nach y
cut x	Trennt einen Knoten x von seinem Vater ab und fügt ihn mit samt seinen Kindern in der Wurzelliste ein
mark x	Markiert den Knoten x
unmark x	Demarkiert den Knoten x
newfheap x	Erzeugt einen neuen temporären Fibonacci-Heap mit Schlüssel x
meld	Verschmilzt den temporären mit dem Haupt-Fibonacci-Heap, indem die Wurzellisten verkettet werden
consolidate	Führt ein consolidate an dem Fibonacci-Heap durch
updatemin x	Setzt x als neues Minimum
remove x	Entfernt einen Knoten x aus der Wurzelliste

Dieser Ansatz entstand aus der Idee heraus, dass alle Teiloperationen eindeutig im Bezug auf ihre Auswirkung definiert sein sollen, um Mehrdeutigkeiten zu beseitigen.

Das consolidate wurde nicht weiter unterteilt, da eine Unterteilung automatisch zu einem Nicht-Determinismus in den Teiloperationen führen würde (siehe folgende Variante 2).

Bei dieser Variante tritt allerdings das Problem auf, dass gerade das Delete-min zu grob unterteilt ist, so dass der Benutzer nicht wissen muss wie ein consolidate funktioniert.

Des Weiteren würde eine Operation wie newheap von dem System verlangen, gleichzeitig auf mehreren Fibonacci-Heaps zu arbeiten, was in Bezug auf den Lerngewinn keinen Unterschied macht.

Aus diesen Gründen musste ein Kompromiss gefunden werden, der in Variante 2 definiert wird.

3.3.2.2. Variante 2

Funktion	Auswirkung
setkey x y	Ändert den Schlüsselwert des Knotens x nach y
cut x	Trennt einen Knoten x von seinem Vater ab und fügt ihn mit samt seinen Kindern in der Wurzelliste ein
mark x	Markiert den Knoten x
unmark x	Demarkiert den Knoten x
newheapmeld x	Erzeugt einen neuen Fibonacci-Heap mit Schlüssel x und verschmilzt ihn sofort mit dem aktuellen
link x y	Verlinkt zwei Knoten x und y miteinander indem y mit samt Unterbaum an x angehängt wird. y wird also ein neues Kind des Vaters von x.
updatemin x	Setzt x als neues Minimum
remove x	Entfernt einen Knoten x aus der Wurzelliste

Die Operationen „newheap x“ und „meld“ wurden in einer Operation zusammengefasst. Später im Programm wird die Funktion durch „Create new FibHeap with node ... + meld“ beschrieben, um zu erreichen, dass der Lernende versteht, was hier wirklich passiert.

Das Aufsplitten des consolidate's in link-Operationen produziert leider einen Nicht-Determinismus, da jetzt mit dem Link an einer beliebigen Stelle in der Wurzelliste begonnen werden kann. Deshalb sollte im Feedback zuvor vermittelt werden, dass mit dem consolidate immer bei dem aktuellen Minimums-Zeiger-Knoten begonnen werden muss.

3.3.2.3. Interaktion, Visualisierung, Abarbeitungsreihenfolge, Skalierbarkeit

Die Menge der Interaktionen bewegt sich bei beiden Varianten in einem vertretbaren Rahmen, da hier nicht zu viel und nicht zu wenig gefordert wird. Jeder Interaktions-Schritt ist ohne Probleme visualisierbar, und in der Abarbeitungsreihenfolge gibt es bezüglich der korrekten Ausführung keine Alternativen. Da sich nicht nur für Bäume funktionale Unterteilungen von Grundoperationen finden lassen, erweist sich dieser Ansatz als sehr gut skalierbar. So könnte man sich beim Bin-Packing das Ausprobieren durch die verschiedenen Bins als einzelne Schritte definieren.

3.3.3. Entscheidung

Die Entscheidung fiel letztendlich auf die zweite Variante des Function-based-Ansatzes, da sich bei dieser Variante die Unterteilung der Arbeitsschritte in einem vernünftigen Maß bewegt, die geforderten Interaktionen nicht zu viel sind, die Abarbeitungsreihenfolge komplett festgelegt ist und es nur eine Mehrdeutigkeit gibt, die aber im Hinblick auf den Lerngewinn vertretbar ist.

3.3.4. Die Konzepte im Vergleich

	Vertex-format-based	Function-based	
		Variante 1	Variante 2
Unterteilung der Arbeitsschritte	fein	grob	mittel
Interaktion	Sehr viel	wenig	mittel
Visualisierung	Problematisch, bis nicht möglich	Keine Probleme	Keine Probleme
Abarbeitungsreihenfolge	Innerhalb Gruppen, zufällig, sonst klar definiert	festgelegt	festgelegt
Mehrdeutigkeiten	Sehr viele	keine	Nur bei Delete-min (consolidate)
Skalierbarkeit	Nur gut für Bäume skalierbar	gut	gut

3.4. Automatische Generierung von Aufgaben

Es soll ein Konzept zur automatischen Aufgabengenerierung erstellt werden, welches es erlaubt, Aufgaben völlig selbstständig und ohne Benutzerinteraktionen zu erstellen. Das Konzept soll es ermöglichen, Aufgaben unterschiedlichen Schwierigkeitsgrades zu erzeugen, wobei diverse Parameter einstellbar sein sollen, die den Schwierigkeitsgrad beschreiben. Des Weiteren soll darauf geachtet werden, dass das Konzept gut auf andere Aufgabentypen skalierbar ist.

Im Folgenden werden mehrere Umsetzungsideen zur Lösung des Problems erörtert.

3.4.1. Aussagenlogisches Konzept

Dieser erste Ansatz basiert auf der Idee, verschiedene Teilsituationen während der Bearbeitung in aussagenlogischen Sätzen zu beschreiben und jeder einen Schwierigkeitsgrad zwischen 1 (leicht) und 7 (schwer) zu zu weisen.

Drei Beispiele:

Blatt b wird durch Delete entfernt und Vater v sei nicht markiert

- Vater-Knoten v wird markiert
- Blatt b wird entfernt
- Rest des Baums bleibt unverändert
- ➔ Schwierigkeitsgrad 2

Blatt b wird durch Decrease-key erniedrigt und Vater v sei nicht markiert, und der neue Wert von b sei kleiner als der Wert des Vaters

- Vater-Knoten v wird markiert
- Blatt b wird entfernt und mit neuem Wert in Wurzelliste eingetragen
- ➔ Schwierigkeitsgrad 3

Innerer markierter Knoten v verliert Sohn s durch delete

- Sohn s wird entfernt

- Cascading Cuts
- die Kinder werden jeweils in die Wurzelliste übertragen
- der oberste nicht markierte Knoten wird markiert
- Schwierigkeitsgrad 7

Somit wäre es möglich, jeder Grundoperation (Insert, Delete, etc.) in Abhängigkeit des aktuellen Baumes einen Schwierigkeitsgrad zu zu weisen, der sich durch das arithmetische Mittel der in ihm enthaltenen Teilsituationen ergibt. Insbesondere bei cascading-cuts können sehr viele Teilsituationen beteiligt sein.

Leider entstehen bei dieser Methode auch Probleme. Erstens müsste zu jeder Situation ein aussagenlogischer Satz erstellt werden, was äußerst mühsam sein kann, da es eine Vielzahl solcher Situationen gibt. Zweitens muss darauf geachtet werden, dass die Sätze disjunkt sind. Und drittens ist diese Methode sehr Problembereich-spezifisch und und dadurch relativ schlecht skalierbar.

3.4.2. Bewertungskonzept

Unmittelbar aus dem aussagenlogischen Konzept entstanden Konzepte der Bewertung von Grundoperationen, ohne diese in sehr feine Teile aufzuschlüsseln.

3.4.2.1. *Konzept 1: Verwendung der Potentialfunktion*

Durch die Verwendung der Potentialfunktion⁸ $\phi_Q = r_Q + 2 * m_Q$ [WA5] ist es möglich, Grundoperationen direkt zu bewerten, so dass keine Unterteilung erforderlich ist. Hierzu müsste der Absolutbetrag der Änderung der Potentialfunktion zweier Fibonacci-Heaps als Bewertung herangezogen werden. Ein kleiner Wert würde eine leichte Operation identifizieren und ein großer Wert eine schwierige.

Durch dieses Konzept ist allerdings nur eine sehr grobe Unterteilung in den Schwierigkeitsgraden möglich, da die Änderungen der Werte der Potentialfunktion meist nur sehr klein sind. Ferner lassen sich nur wenige Parameter verändern, wie z.B. die Gewichtung von markierten inneren Knoten oder die der Wurzellistenknoten.

Da sich zu jedem Algorithmus solch eine Funktion finden ließe, ist die Skalierbarkeit auf andere Probleme gewährleistet.

Die Verwendung anderer möglicher Funktionen wurde nicht untersucht.

3.4.2.2. *Konzept 2: Zählen der Anzahl Schritte einer Grundoperation*

Grundoperationen wie Insert werden bekanntlich durch das Editierungskonzept (siehe Abschnitt 3.3: Editierungskonzepte) in einzelne Teilschritte aufgeteilt. Die Idee dieses Ansatzes ist nun, zur Bewertung einer Grundoperation die Anzahl dieser Schritte zu nehmen.

Auch hier ist die Unterteilung der Schwierigkeitsgrade nur sehr gering. Weiterhin werden sämtliche Teiloperationen vom Schwierigkeitsgrad her gleich eingestuft.

Das Konzept lässt sich aber ohne Probleme und sehr einfach auf andere Bereiche ausdehnen.

⁸ wobei r_Q die Anzahl der Knoten in der Wurzelliste und m_Q die Anzahl der markierten Knoten die sich nicht in der Wurzelliste befinden bezeichnen

3.4.2.3. *Konzept 3: Bewertung der Einzelschritte*

Dieses Konzept ist eine Erweiterung des vorherigen und zwar insofern, als jeder Teilschritt selbst ein vordefiniertes Gewicht hat. Somit ergibt sich die Bewertung einer Grundoperation aus der Summe der Gewichte der Teiloperationen.

Dieses Konzept ermöglicht eine relativ feine Unterteilung der Schwierigkeitsgrade, da dieser sich letztendlich aus der Spanne zwischen leichtester und schwerster Teiloperation ergibt. Es lassen sich viele Parameter einstellen, wodurch eine hohe Anpassungsfähigkeit im Bezug auf die Problemstellung erreicht wird. Auch dieses Konzept lässt sich einfach auf andere Problembereiche skalieren.

3.4.3. Überblick

	Aussagenlogisches Konzept	Bewertungskonzept		
		Potentialfunktion	Einzelschritte zählen	Bewertung von Einzelschritten
Schwierigkeitsgrad	sehr fein	grob	grob	fein
Parameter	sehr viel	wenig	keine	mittel
Skalierbarkeit	Sehr aufwendig	gut	sehr gut	sehr gut

3.4.4. Entscheidung

Das aussagenlogische Konzept erlaubt im Hinblick auf das Einstellen von Parametern sicher die größte Freiheit, lässt sich aber von allen vorgestellten Konzepten am schlechtesten auf andere Gebiete skalieren.

Das zweite Bewertungskonzept (Einzelschritte zählen) fällt auf Grund seiner niedrigen Anpassungsfähigkeit und nicht akzeptabler Unterteilung von Schwierigkeitsgraden sofort außer Betracht. Die Potentialfunktion war ein potentieller Kandidat, aber die Entscheidung fiel letztendlich doch auf das dritte Bewertungskonzept (Bewertung von Einzelschritten), da dieses im Vergleich mit den verlangten Anforderungen die meisten Übereinstimmungen aufweist. Ferner lässt es sich gut auf andere Algorithmen und Datenstrukturen ausdehnen.

3.4.5. Verwendung des Bewertungskonzeptes zur automatischen Generierung

Im vorherigen Abschnitt wurde erklärt, wie einzelne Grundoperationen bewertet werden können. Im Folgenden soll nun beschrieben werden, wie dies zur automatischen Generierung von Aufgaben am Beispiel von Fibonacci-Heap's angewendet werden kann.

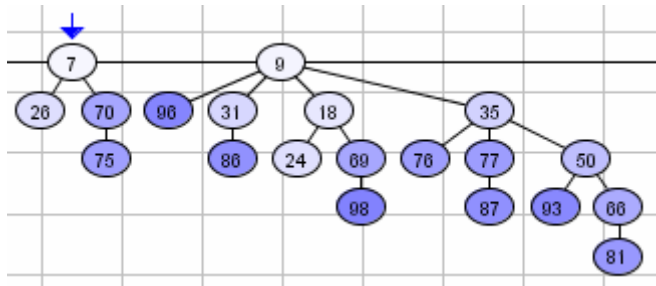
Es wird ein Konzept vorgestellt, das der tiefenbeschränkten Breitensuche in der Logik entspricht.

3.4.6. Suche im Zustandsraum

Da jetzt Bewertungen der Grundoperationen möglich sind, gilt es eine Abfolge von Grundoperationen eines speziellen Schwierigkeitsgrades zu finden. Dazu ist es notwendig Zustände im aufgespannten Zustandsraum zu durchsuchen. Im nächsten Abschnitt wird erörtert, ob das Absuchen des gesamten Zustandsraumes überhaupt möglich ist. Im letzten Abschnitt wird schließlich diskutiert, inwiefern dieses Konzept Aufgaben unterschiedlichen Schwierigkeitsgrades erzeugt.

Zunächst muss dazu der Verzweigungsgrad eines Fibonacci-Heap-Problems berechnet werden.

Betrachten wir folgendes Beispiel, welches aus einer Abfolge von 21 Inserts gefolgt von 1 Delete-min, entstanden ist.



Die einzelnen Schlüssel spannen nun 21 Intervalle auf, die für verschiedene Insert's in Frage kommen. Für Decrease-key und Delete kommen die 20 Knoten in Frage. Delete-min ist nur auf einen Knoten ausführbar.

Somit ergeben sich für einen Baum mit n Knoten folgende Komplexitäten der einzelnen Grundoperation:

- $O(n)$ Insert's
- $O(n^2)$ Decrease-key's, da jeder der n Knoten auf einen Wert innerhalb der $n+1$ Intervalle gesetzt werden kann
- $O(n)$ Delete's
- $O(1)$ Delete-min

➔ Verzweigungsgrad: $O(n^2)$

Im worst-case kann eine automatisch zu generierende Aufgabe aus k Decrease-key Operationen bestehen. In solch einem Fall wäre der Berechnungsaufwand $O(n^{2k})$. Für eine konkrete Aufgabe, die initial aus 20 Knoten besteht und auf die 7 Decrease-key Operationen ausgeführt werden sollen, ergeben sich dann $1,8 * 10^{18}$ verschiedene zu bewertende Operationen. Deshalb kann nicht der komplette Zustandsraum abgesucht werden.

Darum wurde die tiefenbeschränkte Suche verwendet.

Dazu musste im Feldversuch berechnet werden, wie tief man in einem Schritt gehen kann, damit der Rechner in einem vernünftigen Zeitrahmen die Berechnungen abschließen kann.

Ausgangsbasis war ein Fibonacci-Heap mit $n = 20$ Knoten.

$k = 1 \rightarrow 400$ Operationen

$k = 2 \rightarrow 160000$ Operationen

$k = 3 \rightarrow 64\,000\,000$ Operationen

$k = 4 \rightarrow 2,56 * 10^{10}$

$k = 5 \rightarrow 1,02 * 10^{13}$

$k = 4$ liegt auf einem Pentium II 333MHz Rechner gerade noch im Rahmen (ca. 10 Sekunden), wohingegen eine Permutationstiefe von 5 schon über 1 Minute dauert.

Somit werden immer nur 4-er Gruppen der durchzuführenden Operationen analysiert. Das heißt, es werden sämtliche Permutationen der einzelnen 4-er Gruppen durchgeführt und eine davon, die dem entsprechenden Schwierigkeitsgrad entspricht, wird ausgewählt. Diese 4-er Gruppe wird dann komplett auf der Datenstruktur ausgeführt, und das Resultat wird als Ausgangsbasis für die nächste 4-er Gruppe verwendet.

3.4.7. Komplettes Verfahren

Das komplette Verfahren zur automatischen Generierung von Aufgaben unterteilt sich dann letzten Endes in folgende drei Schritte:

3.4.7.1. Schritt 1: Init

Erzeugen einer initialen Baumstruktur durch mehrere zufällige Grundoperationen. Bei Fibonacci-Heaps könnten das zum Beispiel 20 zufällige Inserts gefolgt von einem Delete-min sein.

3.4.7.2. Schritt 2: Scramble

Durcheinanderbringen der Baumstruktur, so dass z.B. bei Fibonacci-Heaps auch markierte Knoten entstehen können. Die Grundoperationen, die dazu verwendet werden sollen, lassen sich durch eine Person spezifizieren.

Hier wird Suche im Zustandsraum verwendet, um aus den vorgegebenen Grundoperationen Operationen für das Durcheinanderbringen zu finden, die einen dem betreffendem Schwierigkeitsgrad angemessenen Baum erzeugen.

Dieser entstandene Baum ist nun die Ausgangsbasis für die Aufgabenstellung, die in dem letzten folgenden Schritt erzeugt wird.

3.4.7.3. Schritt 3: Auto-Gen

Hier wird aus einer Abfolge von Grundoperationen mit leeren Parametern eine mit Parametern gefüllte Abfolge erzeugt, die später die Aufgabe für den Studenten darstellt.

3.4.8. Test

Um zu zeigen, dass das Konzept auch funktioniert, wurden für die 3 Schwierigkeitsgrade „EASY“, „MEDIUM“ und „HARD“ Tests durchgeführt. Die Voraussetzung war ein Baum, der durch 21 Insert's und einem anschließenden Delete-min entstanden ist. Auf ihm wurden entsprechend dem Schwierigkeitsgrad die Scramble- und AutoGen-Operationen ausgeführt.

3.4.8.1. Test-Parameter

Init: 21 Insert's + 1 Delete-min

Scramble: 8 Insert's, 7 Delete's, 1 Delete-min

AutoGen: 2 Insert's, 2 Decrease-key, 2 Delete, 1 Delete-min

Bewertungen: cut=3, link=1, mark=4, newfheapmeld=2, remove=1, setkey=1, unmark=4, updatemin=1

3.4.8.2. Testergebnisse

Die folgenden Testergebnisse enthalten die Summe der Bewertungen der AutoGen-Operationen. Zu jedem Schwierigkeitsgrad wurden fünf Messungen durchgeführt.

	Messungen					Bewertungs-Spanne
	1.	2.	3.	4.	5.	min - max
EASY	10	20	15	14	14	10 - 20
MEDIUM	27	26	33	24	30	24 - 33
HARD	54	67	57	47	49	47 - 67

Man sieht sofort, dass sich die Ergebnisse signifikant unterscheiden (und nicht einmal überschneiden) und somit bestätigen, dass das Verfahren trotz der Tiefenbeschränkung auf 4 den gewünschten Effekt erreicht.

3.4.9. Fluss-Diagramm

Folgendes Fluss-Diagramm zeigt die einzelnen Schritte der automatischen Generierung.

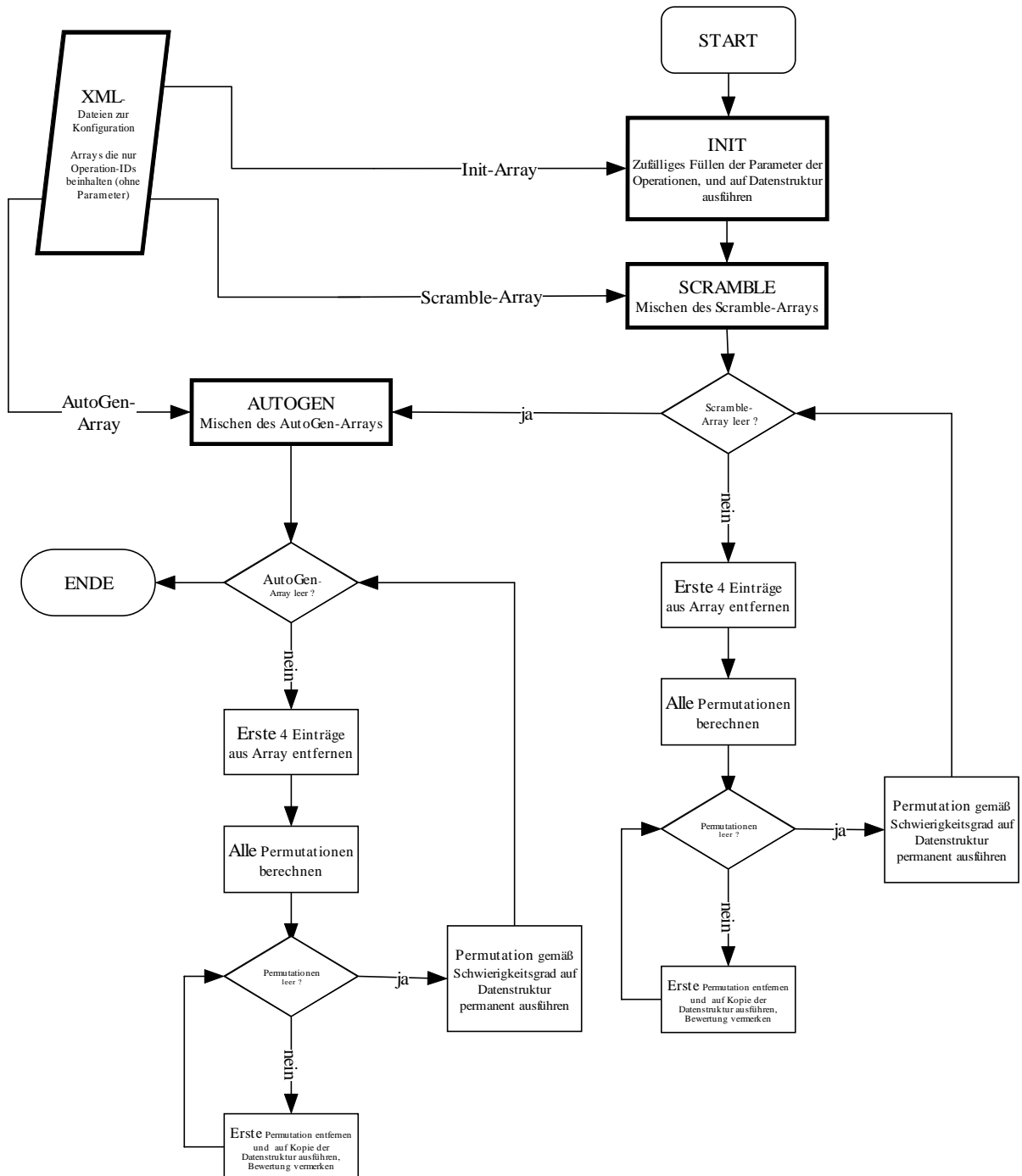


Abbildung 12: Automatische Generierung: Flussdiagramm

3.4.10. Diskussion

Eine wichtige Frage, die im Kontext zu diesem Konzept steht, ist: „Führt diese Methode wirklich zu Aufgaben eines bestimmten Schwierigkeitsgrades?“. Dies lässt sich ohne empirische Untersuchungen an Testpersonen nur schwer feststellen. Fakt ist aber, dass bei Aufgaben zu dem Bereich Algorithmen und Datenstrukturen fast nur prozedurales und nur wenig deklaratives Wissen abgefragt werden kann. Da diese Methode der Aufgabengenerierung bei höheren Schwierigkeitsgraden vom Umfang her mehr prozedurales Wissen zu den Grundoperationen überprüft, sollten auch die Aufgaben schwieriger sein.

3.4.10.1. Erzeugen des initialen Baumes

Der Ausgangsbaum für eine Aufgabenstellung wird wie oben besprochen durch Init- und Scramble-Operationen erzeugt. Um einem dem Schwierigkeitsgrad angemessenen Baum zu generieren, werden die Scramble-Operationen anhand der Suche im Zustandsraum ermittelt. Somit wird angenommen, dass schwierige Operationen auch einen komplexeren Baum erzeugen. Ob das tatsächlich so ist, sei dahingestellt. Fakt ist aber, dass unter Anderem bei einem niedrigen Schwierigkeitsgrad fast keine markierten Knoten auftauchen, während es bei einem hohen relativ viele gibt.

3.4.10.2. Beispiel an Hand einer Delete-min-Operation

Bei einem leichten Schwierigkeitsgrad sorgt der Algorithmus dafür, dass Grundoperationen von der Reihenfolge und den Parametern so erstellt werden, dass möglichst wenige Teiloperationen durchgeführt werden müssen. Das heißt, dass auch das Delete-min entsprechend positioniert wird. Dies wiederum führt dazu, dass meist nur wenige bis gar keine link-Teiloperationen nötig sind.

Bei einem hohen Schwierigkeitsgrad ist genau das Gegenteil der Fall. Somit resultiert für das Delete-min, dass meist sehr viele link-Teiloperationen erforderlich werden. Des Weiteren werden auch bevorzugt link-Teiloperationen berücksichtigt, die eine Änderung der Markierung eines Knotens zur Folge haben.

Gleiches gilt natürlich auch für alle anderen Grundoperationen. Somit ergibt sich zumindest aus dem Gesichtspunkt der Menge von abgefragten Teiloperationen eine Erhöhung des Schwierigkeitsgrades.

3.5. Aufgabenabarbeitungskonzept

Wie im Abschnitt Aufgabenkonzept erklärt wurde, gibt es zwei Typen von Aufgaben. In diesem Abschnitt wird kurz abstrakt erklärt, wie die beiden Aufgabentypen von dem Studenten letztendlich bearbeitet werden. Vorausgesetzt wird, dass die Aufgaben schon fertig geladen zur Verfügung stehen.

3.5.1. Normal-Mode

Dem Studenten wird eine Übersicht präsentiert, in der aufgelistet wird, welche Grundoperationen er auf dem gegebenen Fibonacci-Heap durchzuführen hat. Um nun mit der ersten Grundoperation zu beginnen, muss er diese anfordern. Jetzt hat er die Möglichkeit, die einzelnen Teiloperationen entsprechend auszuführen. Für jede Teiloperation hat er dazu je nach Feedbacktyp drei Versuche. Benötigt er mehr, wird die richtige Teiloperation automatisch ausgeführt und er muss mit der nächsten beginnen. Nach jeder Operation kann ihm gemäß einem ausgewählten Feedback-Typ ein entsprechendes Feedback gegeben werden. Wenn der Student glaubt, er habe alle

Teiloperationen ausgeführt, kann er die nächste auszuführende Grundoperation anfordern. Dies setzt sich so lange fort, bis die Aufgabe komplett abgearbeitet ist.

3.5.2. Fault-Mode

Im Unterschied zu Normal-Mode muss hier der erste Fehler in der Aufgabe gefunden werden. Der Ablauf dazu sieht folgendermaßen aus.

Zunächst einmal wird dem Studenten die komplette Aufgabe abgespielt, damit es ihm möglich ist, sich einen Überblick zu verschaffen. Anschließend kann er zu beliebigen Stellen in der Aufgabe springen oder Teile der Aufgabe noch mal abspielen lassen. Glaubt er, eine Operation sei falsch, kann er dies über einen Button signalisieren. Im Falle, dass diese Operation richtig ist oder es nicht der erste Fehler in der Aufgabe ist, wird ihm das durch ein Feedback vermittelt. Sollte es sich dabei aber um die erste falsche Operation handeln, muss er den Typ des Fehlers genauer spezifizieren:

Typ 1:

Alle folgenden Teiloperationen sind falsch, was heißt, dass die zur entsprechenden Grundoperation erforderlichen Teiloperationen alle schon ausgeführt wurden.

Typ 2:

Zur Vervollständigung der aktuellen Grundoperation fehlen noch Teiloperationen.

Typ 3:

Die aktuelle Teiloperation unterscheidet sich von der korrekten Teiloperation.

Wird ein falscher Typ ausgewählt, wird dem Studenten ein entsprechendes Feedback gegeben und er muss sich für einen der restlichen entscheiden. Dies wiederholt sich solange bis der richtige Typ ausgewählt wurde.

Im Falle, dass Typ 1 die richtige Antwort ist, ist die Bearbeitung der Aufgabe abgeschlossen.

Bei den anderen beiden Typen wird anschließend verlangt, die richtige Teiloperation auszuführen. Dazu hat der Student drei Versuche. Die Aufgabe ist fertig bearbeitet, wenn entweder die richtige Operation spezifiziert wurde oder mehr als drei Versuche benötigt wurden.

3.5.3. Navigation

In beiden Aufgabentypen ist es möglich, zu jeder Stelle einer Aufgabe zu springen, beliebige Teile abzuspielen und durchgeführte Aktionen rückgängig zu machen oder zu wiederholen. Die Aufgaben können sowohl automatisch als auch tutoriell evaluiert werden. Im Falle einer automatischen Evaluation werden Punkte im Bereich 0 (schlecht) bis 100 (alles richtig) vergeben.

3.6. Feedback-Konzept

Für das Feedback musste ein Konzept ausgearbeitet werden, welches es erlaubt, zu gemachten Aktionen passende Informationen zu liefern.

Zur Lösung dieses Problems wurde das Prinzip einer Knowledge-Base aus der KI verwendet. Die Idee ist, bestimmte Situationen in Formeln zu fassen, denen parametrisierte Sätze zugewiesen werden.

Beispiel:

Wenn die aktuell ausgeführte Teiloperation die erste falsche Teiloperation innerhalb eines Delete-mins oder Delete (min) ist, es sich um eine link-Operation handelt und der erste Parameter der link-Operation größer als der zweite Parameter ist, dann gib „A link links always the bigger node to the smaller node, so that the bigger node is a child of the smaller node. Otherwise are the Heap-Constraints violated.“ als Feedback aus.

Solche Fälle werden in geeigneter Kodierung in der Knowledge-Base abgelegt.

3.6.1. Bereiche der Knowledge-Base

Die Knowledge-Base (KB) unterstützt folgende Fälle:

Zunächst ist ein Feedback zu korrekten Aktionen möglich, was unter Anderem benötigt wird, wenn z.B. bei einer Fault-Mode-Aufgabe behauptet wird, dass eine richtige Operation falsch sei.

Ferner gibt es Einträge zu ersten falschen Teiloperationen innerhalb von Grundoperationen. Dies ist wichtig, weil sich diese Formeln nicht auf Folgefehler anwenden lassen.

Zu guter Letzt ist es möglich, Feedback zu Grundoperationen zu geben, die noch nicht vollständig sind, d. h., dass noch ein oder mehrere Teiloperationen fehlen. Dieser Fall wurde notwendig weil sich die Formeln zu „ersten falschen Aktionen“ nicht auf fehlende anwenden lassen. Das ist z.B. dann der Fall, wenn ein Student ein updatemin am Ende einer Delete-min-Grundoperation vergisst, aber behauptet, er wäre fertig mit der Bearbeitung des Delete-min. In solch einem Fall gibt es noch keine falsche Operation und könnte somit auch nicht durch oberen Fall erkannt werden.

3.6.2. Problem: Folgefehler

Folgefehler innerhalb einer Grundoperation werden nicht unterstützt, weil sie sich nur sehr schwer in der Logik beschreiben lassen. Des Weiteren würde die Anzahl der benötigten Sätze explodieren da sämtliche Kombinationen zwischen erstem und zweitem Fehler kodiert werden müssten. Da die Knowledge-Base derzeit ca. 60 Einträge enthält, wären für Folgefehler bereits ungefähr $60 * 60 = 3600$ Einträge notwendig.

Wird allerdings eine neue Grundoperation begonnen, kann wieder Feedback für die erste falsche Teiloperation innerhalb dieser neuen Grundoperation gegeben werden.

Da bis auf Delete-min die meisten Operationen selbst bei höchstem Schwierigkeitsgrad relativ wenige Teiloperationen (ca. 3 bis 5) enthalten, geht an Feedback-Information nicht viel verloren.

3.6.3. Anfrage an die KB

Anfragen an die KB lassen sich momentan nur zur aktuellen Situation der Aufgabe stellen. Wird ein Feedback zu vorhergehenden oder nachfolgenden Situationen benötigt, müssen diese entsprechend vermerkt werden.

3.6.4. Skalierbarkeit

Leider ist dieser Ansatz nur relativ schlecht auf andere Bereiche skalierbar und zwar insofern, als für jedes andere Problem eine komplett neue KB aufgebaut werden muss. Bestimmte Teile haben solche verschiedenen KB's allerdings gemeinsam (siehe Kapitel 4

Implementierung: Abschnitt 4.5: Feedback: Knowledge-Base). Dennoch wurde der Ansatz verwendet, da er ein Höchstmass an passendem Feedback zu jeder Situation ermöglicht, was für ein Feedback äußerst wichtig ist.

3.6.5. Konkrete Verwendung

Verwendet wird die KB in den insgesamt fünf Tutortypen. Die ersten vier entstammen aus den in Kapitel 2 (Abschnitt 2.1.2: Tutortypen) vorgestellten Tutortypen. Lediglich für das Feedback bei Fault-Mode-Aufgaben musste ein eigener fünfter Typ erstellt werden. Auf diesen soll hier aber nicht weiter eingegangen werden, da der Ablauf im Aufgabenabarbeitungskonzept prinzipiell schon beschrieben wurde.

3.7. Plug-In-Konzept

Um das System auf andere Algorithmen und Datenstrukturen anwenden zu können, mussten Überlegungen angestellt werden, wie dies mittels Plug-Ins zu realisieren wäre. In diesem Abschnitt wird das Konzept abstrakt erklärt. Eine detaillierte Beschreibung bezüglich der Implementierung findet sich in Kapitel 4 (Abschnitt 4.3: Plug-In-Handler).

3.7.1. Voraussetzungen

Damit eine neue Problemstellung in das MAUDA-System integriert werden kann, muss diese einige wenige aber notwendige Eigenschaften erfüllen:

- Das Gesamtproblem muss in Schritten aus Grundoperationen beschrieben werden können
- Die Grundoperationen müssen sich in Teiloperationen unterteilen lassen können, welche später bei der Aufgabenstellung vom Studenten abgefragt werden.
- Sowohl Grundoperationen als auch Teiloperationen müssen folgendermaßen darstellbar sein:
 - o ID der Operation im Textformat
 - o Keine bis maximal 2 numerische Parameter⁹
- Das Problem muss in JEDAS in Schritten von Grund- und Teiloperationen darstellbar bzw. animierbar sein
- Die ganze Problemstellung muss durch die Maus kontrollierbar sein, d.h., andere Eingabemöglichkeiten, wie z.B. Eingabe von Zahlen, müssten zuerst durch die Maus ein Fenster öffnen, in welchem man die Angaben dann eingeben kann.

Diese wenigen Anforderungen müssten sich theoretisch auf jedes Problem aus dem Bereich Algorithmen und Datenstrukturen anwenden lassen.

3.7.2. Komponenten

Ein Plug-In besteht aus mehreren spezifischen, meist kleinen und leicht implementierbaren Komponenten:

⁹ Durch Nutzung von ID's lassen sich hier auch Aktionen im Kontext zur Operation kodieren. Somit lassen sich alle Operationen mit 2 Parametern integrieren. (Beispiel: AVL-Bäume: Teiloperation = rotate, 0 = Rechts-Rotation, 1 = Links-Rotation)

- JEDAS-Objekt, welches die Problemstellung visualisieren und animieren kann.
- Analyse-Modul, welches die Funktionalität anbietet um Aufgaben verschiedenen Schwierigkeitsgrades zu erzeugen.
- Interaktionskomponente, die es erlaubt, interaktiv mit der Datenstruktur zu kommunizieren. Diese Komponente sollte die auszuführenden Operationen generieren.
- Abbildungs-Modul, welches es gestattet sowohl Grundoperationen als auch Teiloperationen auf die betreffende Datenstruktur abzubilden bzw. auf ihr auszuführen. (Gegenstück zur Interaktionskomponente).
- Ein Formel-Auswertungs-Modul um spezifische Teile der Formeln der Knowledge-Base zu evaluieren.

3.7.3. Daten

Das Plug-In benötigt folgende Daten, die in einem selbst spezifizierbaren Datenverzeichnis abgelegt werden müssen:

- Bewertungsdatei für die Teiloperationen im XML-Format (optional).
- Knowledge-Base für das Feedback als Text-Datei.

Die Bewertungsdatei ist optional, da diese beim ersten Start des Aufgabenerstellungs-Editors (Generation-Editor) automatisch erstellt wird, falls sie nicht bereits existiert. Dabei wird jeder Teiloperation eine Bewertung von 1 zugewiesen. Die möglichen Teiloperationen werden dazu aus der Konfigurationsdatei, deren Zweck im Folgenden beschrieben wird, bezogen.

3.7.4. Konfigurationsdatei

Damit ein Plug-In im MAUDA-System verwendet werden kann, muss das Plug-In mittels Einträgen in einer Konfigurationsdatei aktiviert werden. Dort werden Verweise auf die Komponenten und Daten, die oben beschrieben wurden, angelegt.

Nach dieser Erweiterung ist das neue Plug-In automatisch in allen Teilen des MAUDA-Systems ansprechbar und benutzbar.

Das Plug-In wirkt sich dann automatisch auf die verschiedenen Teile der GUI's aus, die Plug-In-spezifische Eingaben erfordern, wie z.B. Einträge in Menus und Eingaben, die das Hinzufügen von bestimmten Operationen zu einer Liste erfordern.

3.8. Meta-Daten

Zur Verwaltung der generierten Dateien wurde ein einheitliches Meta-Daten-Konzept erstellt, um später die Möglichkeit zur Erstellung eines speziellen Meta-Daten-Datei-Browsers zu geben. Die Meta-Daten teilen sich in folgende Bereiche auf:

Allgemeines:

- Typ der Aufgabe (z.B. Fibonacci-Heap)
- Schwierigkeitsgrad
- Aufgabentyp (Normal-Mode, Fault-Mode)

Entwickler der Aufgabe

- Name
- Erstellungszeitpunkt
- Bewertung der Aufgabe (interne Verwendung)
- Kommentar

Bearbeiter der Aufgabe

- Name (+ Matrikelnummer)
- Zeitpunkt an dem die Bearbeitung der Aufgabe begonnen wurde
- Bearbeitungsstatus (unworked, in process, completed)
- Feedback-Typ
- Kommentar

Evaluator der Aufgabe

- Name
- Datum
- Bewertungsstatus (not evaluated, evaluated)
- Dateiname der zugehörigen Jedas-Animations-Datei
- Erreichte Punkte
- Kommentar

Im Folgenden wird für jede Phase eines Assessments erklärt wie die Meta-Daten gefüllt werden.

3.8.1. Vorbereitungsphase

Nach der Erstellung einer Aufgabe wird zunächst das Allgemein-Feld entsprechend automatisch komplett gefüllt, bevor die Person, die die Aufgabe erstellt hat, ihren Namen in das entsprechende Feld eintragen muss. Im Falle, dass die Aufgabe automatisch generiert wurde, wird „AutoGenerator“ eingetragen. Der Erstellungszeitpunkt wird automatisch auf das Systemdatum und –zeit gesetzt. Das Bewertungsfeld enthält zum einen die Bewertung der Init-Operationen¹⁰ und zum anderen die der Todo-Operationen¹¹. Diese Angaben werden nur für interne Zwecke verwendet. Durch das Kommentar-Feld ist es möglich, eine Information zu der Aufgabe anzugeben. Des Weiteren werden der Bearbeitungsstatus auf „unworked“, und der Bewertungsstatus auf „not evaluated“ gesetzt.

3.8.2. Durchführungsphase

Wird die Aufgabe durch den Studenten bearbeitet, werden sowohl Zeitpunkt als auch gewählter Feedback-Typ in den Meta-Daten festgehalten. Über ein Kommentar-Feld kann der Student eine Anmerkung zur Aufgabe einbringen, während er im Name-Feld seinen Namen und möglichst auch Matrikelnummer festhalten sollte. Wird eine unfertige Aufgabe gespeichert, wird im Status „in process“ vermerkt ansonsten wird das Feld auf „completed“ gesetzt.

¹⁰ Aufsummieren der Grundoperations-Bewertungen, durch die die Ausgangsbasis der Aufgabe erzeugt wird

¹¹ Aufsummieren der Grundoperations-Bewertungen, die später die Aufgabe an den Studenten darstellen

3.8.3. Evaluationsphase

In der Evaluierungsphase wird, wie oben auch verlangt, dass die Person, die die Aufgabe evaluiert, ihren Namen in das korrespondierende Feld einträgt. Im Falle einer automatischen Evaluation wird hier „AutoEvaluator“ vermerkt. Das Datum wird automatisch gesetzt und im Falle einer tutoriellen Evaluation können Punkte vergeben werden. Auch hier ist ein Kommentar möglich, der sich eventuell an den Studenten richten könnte. Wenn die Aufgabe durch einen Tutor evaluiert wurde, hat dieser eine entsprechende Jedas-Animation erstellt, deren Dateiname in das entsprechende Feld automatisch eingetragen wird. Zum Schluss wird der Bewertungsstatus auf „evaluated“ gesetzt.

3.8.4. Sicherheit

Jede Person sieht jedes Feld und somit sollten in z.B. Entwickler-Kommentaren keine Hinweise zur Lösung der Aufgabe vermerkt werden. Es ist aber anzumerken, dass jede Person immer nur ihren eigenen Bereich bearbeiten kann. D.h. ein Student der sich in der Bearbeitungsphase befindet kann keine Eintragungen im Evaluator-Bereich vornehmen und umgekehrt.

Anhand der Meta-Daten wird auch sichergestellt, dass in jeder Phase nur die entsprechenden Dateien geladen werden können.

3.9. Tutorielle Evaluation

Das System soll es ermöglichen, Aufgaben durch eine Person zu evaluieren. Dazu soll die Aufnahmekomponente von JEDAS in Verbindung mit dem Annotationssystem benutzt werden. Das entsprechende Modul soll dabei so einfach wie möglich zu bedienen sein und unter Anderem erlauben, dass man Annotationen zu einer bestimmten Aktion mehrmals durchführen kann. Das heißt, dass, wenn zu einer Aktion Annotationen gemacht wurden, diese aber verworfen werden sollen, ein Neuversuch möglich sein soll, ohne dass Annotationen zu vorherigen Aktionen verloren gehen.

Da eine JEDAS-Aufnahme während der Aufnahme nicht pausiert oder angehalten werden kann, müsste die ganze Aufgabe sofort beim ersten Versuch komplett ohne Fehler annotiert werden. Würde man einen Fehler machen, müsste man die ganze Aufnahme komplett neu starten und vorherige richtige Annotationen noch einmal vornehmen.

Um diesem entgegenzuwirken, wurde ein Konzept entwickelt, welches zusätzlich zur Aufnahme-Datei eine Steuerdatei benutzt, um mehrmalige Versuch zu ermöglichen.

3.9.1. Das Prinzip

Zu jeder Aktion werden Start- und Stopp-Zeiten vermerkt. Die Start-Zeit beinhaltet dabei den Zeitpunkt der JEDAS-Aufnahme, bei dem eine Teiloperation in JEDAS zur Ausführung kommt. Die Stopp-Zeit ergibt sich aus der Start-Zeit der nächsten ausgeführten Teiloperation. Wird nun eine Teiloperation mehrmals zur Ausführung gebracht, werden neue Start- und Stopp-Zeiten zugeordnet. Diese Zeiten werden nun in der Steuerdatei vermerkt.

Bei der Wiedergabe einer JEDAS-Aufnahme werden jetzt nur noch die Segmente jeder Aktion abgespielt, welche sich durch die Start- und Stopp-Zeiten ergeben und zwar in der Reihenfolge, wie sie vom Studenten durchgeführt wurden. Somit werden die Fehlversuche nicht abgespielt, obwohl sie in der Aufnahme vorhanden sind.

3.9.2. Schlussfolgerung

Das Prinzip erfüllt somit die Anforderungen. Annotationen zu einzelnen Aktionen können mehrmals und in beliebiger Reihenfolge versucht werden. Es ist allerdings darauf zu achten, dass man jede Operation auch mindestens einmal zur Ausführung gebracht hat, damit die Zeiten gefüllt werden. Ansonsten würde zu den entsprechenden Aktionen nichts wiedergegeben. Des Weiteren muss beachtet werden, dass wenn man ein und dieselbe Annotation über mehrere Teiloperationen hinweg darstellen will, diese auch in einem zusammenhängenden Versuch durchgeführt werden müssen.

4. Implementierung

4.1. Programme & GUI's

In den folgenden Abschnitten werden die erstellten Programme und ihre graphischen Benutzerschnittstellen vorgestellt.

4.1.1. GenEditor

Der GenEditor dient zum Erstellen von Aufgaben durch eine Person. Über einen Button lassen sich aber auch Aufgaben automatisch erstellen und anschließend nachbearbeiten. Des Weiteren können hier auch die Parameter für die völlig automatische Generierung mittels des AutoGenerators (siehe 4.1.2) eingestellt werden.

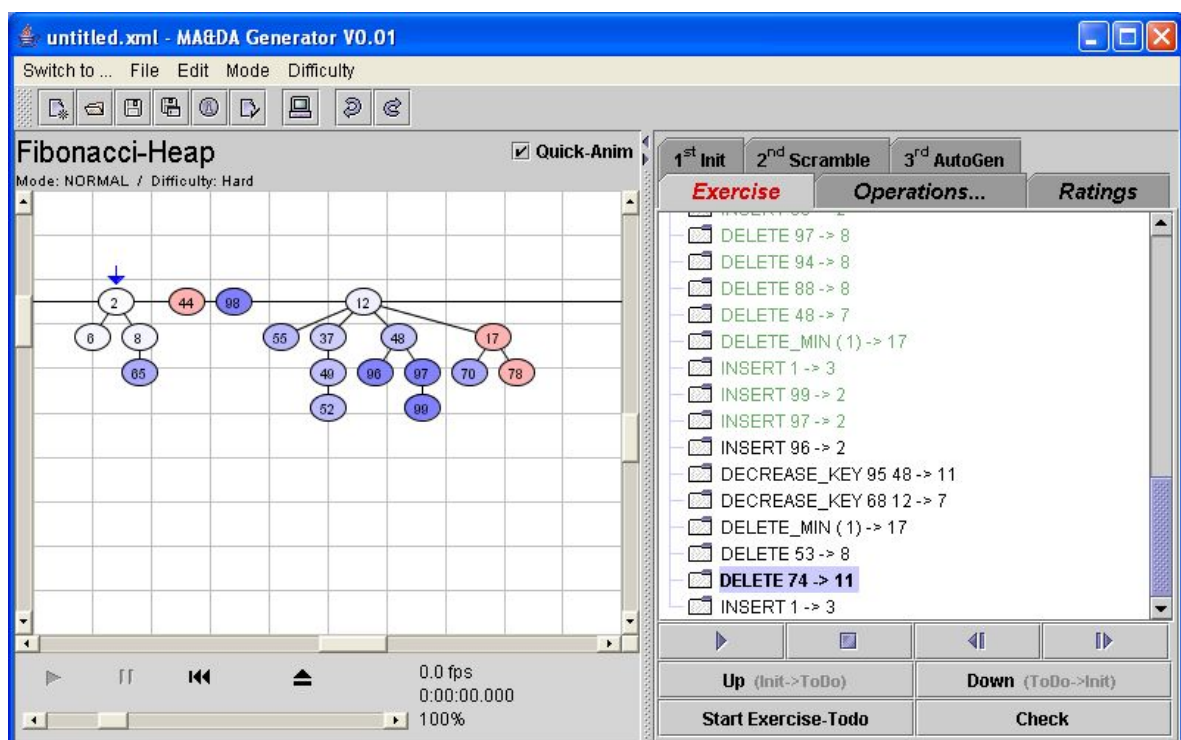


Abbildung 13: GUI: GenEditor

Im linken Bereich wird der aktuelle Zustand der Datenstruktur dargestellt, während sich im rechten Bereich mehrere Reiter befinden, die im Folgenden erklärt werden. Die Reiter Ratings, 1st Init, 2nd Scramble und 3rd AutoGen enthalten dabei Parameter, die vor allem für die automatische Generierung von Aufgaben relevant sind (siehe Abschnitt 3.4: Automatische Generierung von Aufgaben). Sie lassen sich aber auch separat im GenEditor ausführen.

4.1.1.1. Exercise

Hier werden zum einen die Operationen dargestellt aus welchen der Fibonacci-Heap entstanden ist (grün) und zum anderen welche Operationen später von dem Studenten korrekt durchgeführt werden sollen (schwarz). Durch Fettdruck wird die aktuelle Bearbeitungsstelle gekennzeichnet.

4.1.1.2. Operations

Enthält eine Auflistung momentan durchführbarer, logisch unterscheidbarer Operationen sowohl sortiert nach Bewertungen als auch nach ID's.

4.1.1.3. Ratings

Darstellung der Teiloperationen und zugewiesener Bewertungen. Des Weiteren lassen sich diese Bewertungen auch modifizieren.

4.1.1.4. 1st Init

Darstellung und Editierung der Operation-ID's zur initialen Generierung einer gefüllten Datenstruktur.

4.1.1.5. 2nd Scramble

Darstellung und Editierung der Operation-ID's zum Durcheinanderbringen einer durch Init erstellten Datenstruktur.

4.1.1.6. 3rd AutoGen

Angabe der Operation-ID's die später in der Aufgabenstellung vorkommen sollen.

4.1.2. AutoGenerator

AutoGenerator ist ein Kommando-Zeilen-Programm (besitzt also kein GUI) das nach Übergabe des Typs, Schwierigkeitsgrades und Aufgabentyps völlig selbstständig eine Aufgabe erstellt und speichert.

4.1.3. WorkEditor

In diesem Programm lassen sich Aufgaben durch den Studenten bearbeiten.

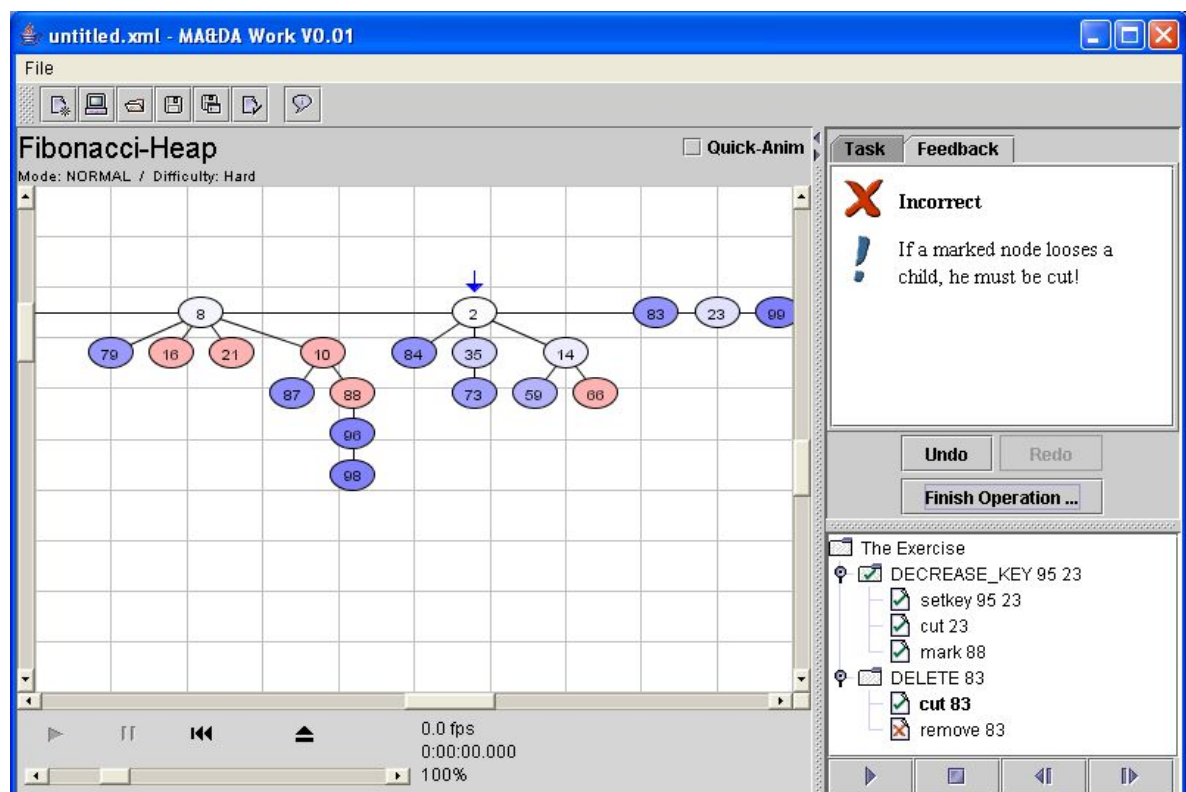


Abbildung 14: GUI: WorkEditor

Im Task-Bereich oben rechts wird die Aufgabenstellung erklärt, während im Feedback-Bereich ein Feedback zur aktuell durchgeführten Operation angezeigt wird. Unten rechts wird angezeigt, welche Operationen bisher durchgeführt wurden.

Über einen Button lassen sich auch Aufgaben direkt vom AutoGenerator anfordern. Dazu müssen Angaben zur zu generierenden Aufgabe gemacht werden:

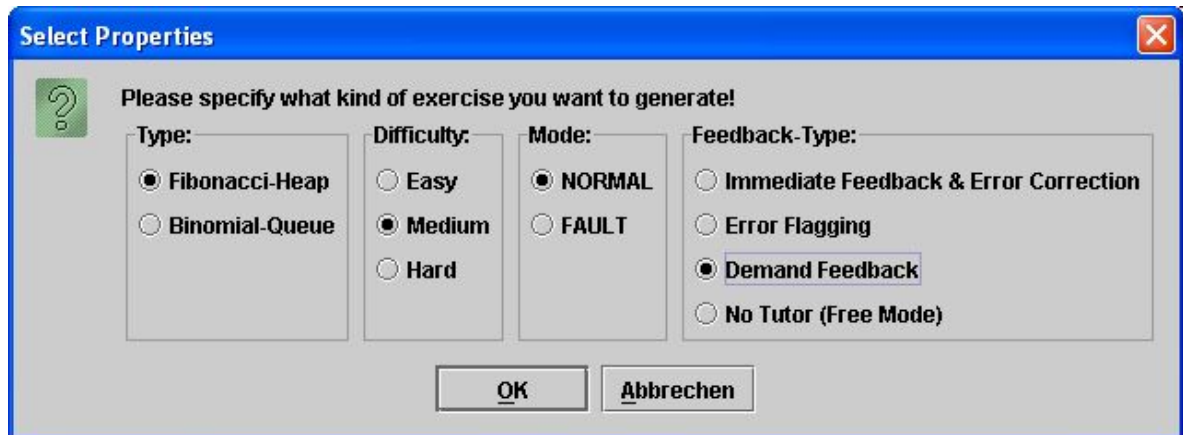


Abbildung 15: GUI: AutoGenerator-Aufgabenanforderungs-Auswahl

Das Type-Feld variiert dabei je nach verfügbaren Plug-Ins.

4.1.4. EvalEditor

Im Evaluation-Editor kann ein Tutor eine Aufgabe bewerten. Dabei kann er zur Jeda-Animation Annotationen erstellen und unabhängig von der Aufnahme Kommentare zu jeder Operation angeben.

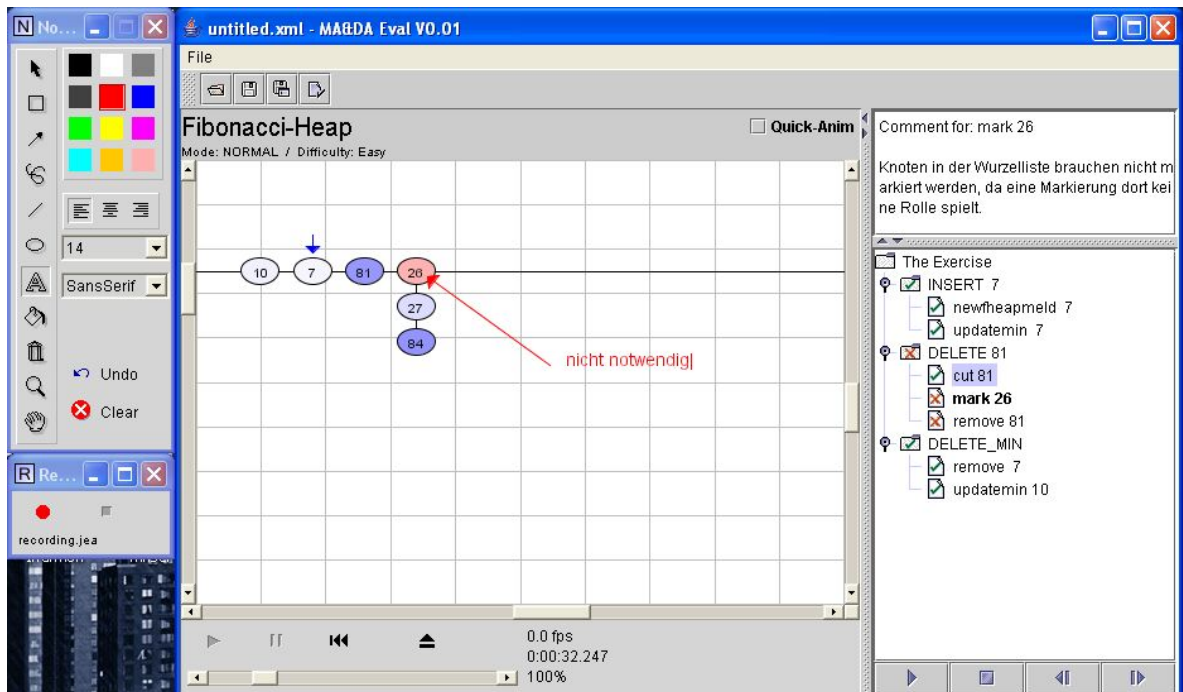


Abbildung 16: GUI: EvalEditor

4.1.5. EvalPlayer

Der Evaluation-Player letztendlich ist dazu da, tutoriell bewertete Aufgaben durch den Studenten betrachten zu können. Im Rahmen der Diplomarbeit konnte dieser nicht

endgültig getestet werden, da die vorhandene Fibonacci-Heap-Klasse [10] leider nicht JEDAS-Aufnahmen-Konform ist. Im Rahmen einer zusätzlichen, direkt anschließenden Arbeit soll das Fibonacci-Heap-Plug-In aber noch um die benötigte Funktionalität erweitert werden.

In Abbildung 17 wird gezeigt, wie der Player fertig aussehen soll.

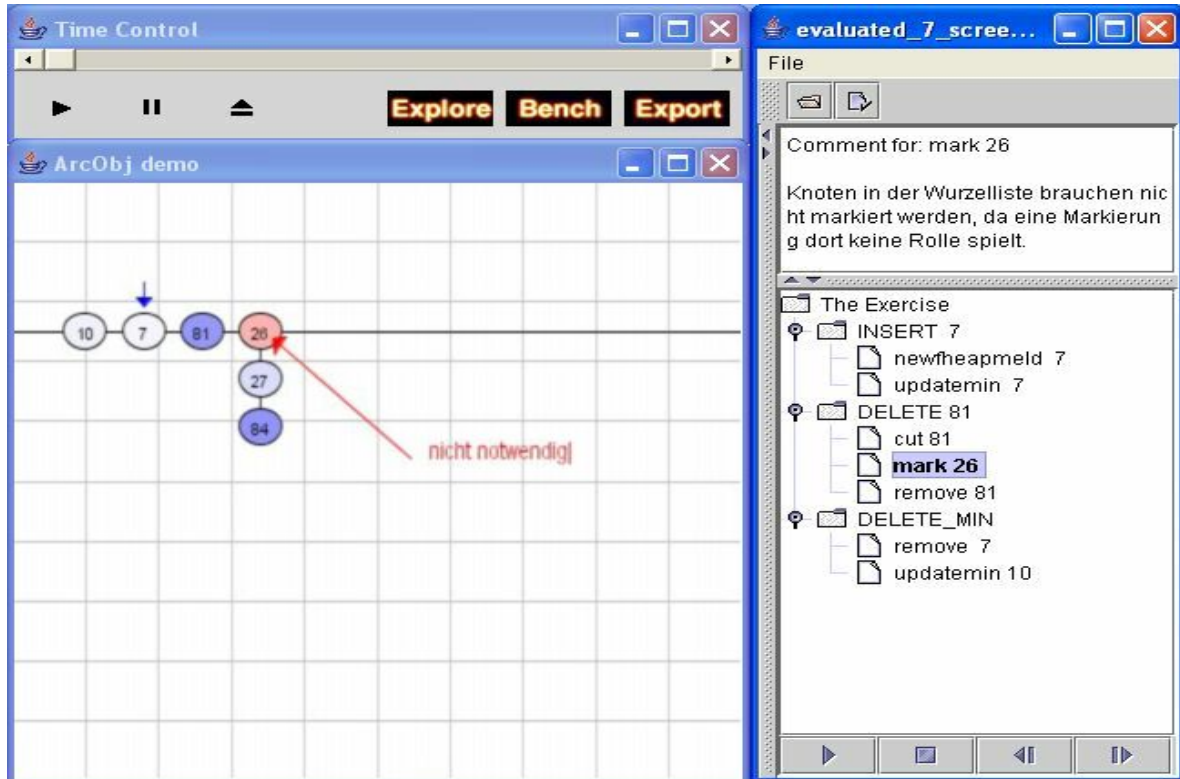


Abbildung 17: GUI: EvalPlayer

4.1.6. Meta-Daten

In Abbildung 18 wird die Umsetzung der Meta-Daten-Eingabe, wie sie in Kapitel 3 (Abschnitt 3.8: Meta-Daten) beschrieben wurde, gezeigt.



Abbildung 18: GUI: Meta-Daten-Eingabe

4.2. *Aufgabenerstellung mit dem GenEditor*

Prinzipiell gibt es, wie in Kapitel 3 (Abschnitt 3.2: Aufgabenkonzept) beschrieben wurde, zwei Arten von Aufgaben. An dieser Stelle soll kurz erklärt werden, wie diese jeweils mit dem GenEditor erstellt werden können.

4.2.1. **Normal-Mode-Aufgaben**

- Über das Menu „Mode“ „Normal“ auswählen
- Schwierigkeitsgrad der zu generierenden Aufgabe über das Menu „Difficulty“ auswählen
- Ausführen der Operationen, die den Aufgabenausgangspunkt erzeugen sollen
- Im Exercise-Reiter auf „Start Exercise-Todo“ klicken um zu signalisieren, dass ab hier die Aufgabenstellung beginnt
- Ausführen der Operationen, die die Aufgabenstellung bilden
- „Save as“-Button oder den Menüpunkt „File/Save as ...“ anwählen
- Datei angeben, in der die Aufgabe abgespeichert werden soll
- Meta-Daten eingeben

4.2.2. **Fault-Mode-Aufgaben**

Das Erzeugen von Fault-Mode-Aufgaben entspricht weitestgehend dem von Normal-Mode-Aufgaben. Es gibt lediglich einen Unterschied bei der Eingabe von Operationen, die die Aufgabenstellung bilden sollen. Dort soll nun ein Fehler eingebracht werden können.

Soll eine komplette Grundoperation ohne Fehler in der Aufgabenstellung erscheinen, wählt man diese wie bei Normal-Mode aus.

Sollen in eine Grundoperation ein oder mehrere Fehler eingebracht werden, muss man im Menu „Edit“ durch Anwahl von „Sub-Operation Edit-Mode“ in den Teiloperations-Editiermodus wechseln. Nun muss man zunächst die Grundoperation auswählen, in die man Fehler einbringen will. Diese Grundoperation wird jetzt nicht auf der Datenstruktur ausgeführt, sondern es wird lediglich ein so genanntes Template erzeugt. Anschließend lassen sich nun die Teiloperationen ausführen, die man mit dem Template verbinden will. Um die korrekten Teiloperationen dieses Templates in Erfahrung zu bringen, lassen diese sich mittels Rechts-Klick im Exercise-Reiter per Kontextmenüpunkt „Show Suboperations...“ anzeigen. Auf diese Weise ist ein direkter Vergleich mit ausgeführten und korrekten Teiloperationen möglich, und es lassen sich beliebige Fehler in eine Grundoperation einbringen.

Sollen nun wieder nur korrekte Grundoperationen folgen, muss man den Teiloperations-Editiermodus wieder deaktivieren.

4.3. *Plug-In-Handler*

In Kapitel 3 (Abschnitt 3.7: Plug-In-Konzept) wurde das Plug-In-Konzept abstrakt vorgestellt. An dieser Stelle soll dies auf Implementierungs-Ebene etwas vertieft werden.

Verantwortliche Java-Klasse: `mauda.plugin.PlugInHandler`.

Für jedes neue Plug-In sind folgende Java-Klassen zu erstellen, welche nochmals tabellarisch auf der nächsten Seite aufgeführt werden:

- JEDAS-Animations-Struktur, welche es erlaubt, die dem Plug-In zugrunde liegenden Algorithmus oder Datenstruktur (im Folgenden AD genannt) zu animieren. Dabei muss darauf geachtet werden, dass nur ein Item im CompPanel von JEDAS notwendig wird, in dem man mehrere Objekte in einem CompObj sammelt. Die Struktur muss `mauda.plugin.Copyable` implementieren was bedeutet, dass eine `deepCopy`-Methode implementiert werden muss, um die komplette Struktur kopieren zu können (wird für Undo/Redo benötigt)
- Eine Analyse-Klasse zur automatischen Generierung von Aufgaben, welche das `mauda.plugin.Analysable`-Interface implementieren muss. Es müssen Methoden für `init`, `scramble` und `autogen` implementiert werden.
- Eine Klasse die Operationen auf entsprechende Methoden des AD abbildet. Diese Klasse muss `mauda.plugin.OperationExecuter` per `extends` erweitern.
- Eine Klasse die Variablen aus der Feedback-Knowledge-Base interpretieren kann. Diese Klasse muss `mauda.plugin.KBFormulaEvaluator` per `extends` erweitern.
- Interactive-Klasse, welche es erlaubt, AD-kontext-spezifische Aktionen per Maus zu generieren. Diese Klasse muss das Interface `java.awt.event.MouseListener` implementieren.

Damit ein Plug-In im MAUDA-System aktiviert wird, müssen die Verweise auf diese Klassen in einer Konfigurations-Datei `mauda.plugin.plugin-properties` abgelegt werden.

4.3.1. Konfigurations-Datei

Dateiinhalt für Fibonacci-Heap- und noch nicht funktional fertigem Binomial-Queue-Plug-In:

```
AvailablePlugIns = FibHeap BinQueue

FibHeap = Fibonacci-Heap
FibHeapDS = mauda.plugin.fibheap.FibHeapExt
FibHeapAnalyzer = mauda.plugin.fibheap.FibHeapAnalyse
FibHeapPopupMenu = mauda.plugin.fibheap.FibHeapInteractive
FibHeapOperationExecuter = mauda.plugin.fibheap.FibHeapOperationExecuter
FibHeapKBFormulaEvaluator = mauda.plugin.fibheap.FibHeapKBFormulaEvaluator
FibHeapOperations = INSERT DELETE DELETE_MIN DECREASE_KEY
FibHeapSubOperations = SETKEY CUT MARK UNMARK NEWFHEAPMELD LINK UPDATEMIN REMOVE
FibHeapDataDirectory = data/fibheap/

BinQueue = Binomial-Queue
BinQueueDS = mauda.plugin.binqueue.BinQueue
BinQueueAnalyzer = mauda.plugin.binqueue.BinQueueAnalyse
BinQueuePopupMenu = mauda.plugin.fibheap.FibHeapInteractive
BinQueueOperationExecuter = mauda.plugin.binqueue.BinQueueOperationExecuter
BinQueueKBFormulaEvaluator = mauda.plugin.fibheap.FibHeapKBFormulaEvaluator
BinQueueOperations = INSERT DELETE DELETE_MIN DECREASE_KEY
BinQueueSubOperations = SETKEY CUT MARK UNMARK NEWFHEAPMELD LINK UPDATEMIN REMOVE
BinQueueDataDirectory = data/binqueue/
```

4.3.2. Tabellarische Übersicht

	Aufgabe	Kontext	Beispiel: Fibonacci-Heap-Plug-In
Animation-Structure	Animierung des AD	extends CompObj implements mauda.plugin.Copyable deepCopy → implementieren	mauda.plugin.fibheap.FibHeapExt
Analysable	Automatische Generierung von Aufgaben	implements mauda.plugin.Analysable getOperations → implementieren init → implementieren scramble → implementieren autoGen → implementieren fullAutomatic → implementieren	mauda.plugin.fibheap.FibHeapAnalyse
OperationExecuter	Ausführen von Operationen auf dem AD	extends mauda.plugin.OperationExecuter executeTemplate → Operation execute → Operation execute → SubOperation	mauda.plugin.fibheap.FibHeapOperationExecuter
KBFormulaEvaluator	Auswerten von Variablen der Feedback-Knowledge-Base	extends mauda.plugin.KBFormulaEvaluator getSpecialConstant → überschreiben	mauda.plugin.fibheap.FibHeapKBFormulaEvaluator
Interactive	Interaktion zwischen Benutzer und dem AD	implements java.awt.event.MouseListener mouseClicked → implementieren mouseEntered → implementieren mouseExited → implementieren mousePressed → implementieren mouseReleased → implementieren	mauda.plugin.fibheap.FibHeapInteractive

4.3.3. Einzelne Klassen und zu implementierende Methoden im Detail

Animation-Structure	
deepCopy	Kopieren der kompletten Animationsstruktur

Analysable-Interface	
getOperations	Rückgabe der möglichen, logisch verschiedenen Operationen
Init	Rückgabe von instanziierten Grundoperationen aus einer Übergabemenge
scramble	Rückgabe von instanziierten Grundoperationen eines bestimmten Schwierigkeitsgrades aus einer Übergabemenge
autoGen	Rückgabe von instanziierten Grundoperationen eines bestimmten Schwierigkeitsgrades aus einer Übergabemenge
fullAutomatic	Rückgabe von Init- & Todo-Grundoperationen die eine komplette Aufgabe bilden. (In der Regel eine Komposition von init, scramble und autoGen)

OperationExecuter-Klasse	
executeTemplate	Ausführen einer Grundoperation auf einer Kopie des AD, um die korrekten Teiloperationen dieser zu ermitteln
execute	Ausführen einer Grundoperation auf dem AD
execute	Ausführen einer Teiloperation auf dem AD

KBFormulaEvaluator	
getSpecialConstant	Muss überschrieben werden, und soll den Wert eines Teilausdrucks der Knowledge-Base zurückgeben.

Interactive	
mouseClicked	Mauskontrolle. Diese Klasse muss entsprechend dem derzeitigen InteractiveMode (exercise.getInteractiveMode()) verschiedene Eingabemöglichkeiten anbieten können: OPERATION: Auswahl von Grundoperationen SUBOPERATION: Auswahl von Teiloperationen BOTHOPERATION: Auswahl von Grund- und Teiloperationen NOPOPUP: keine Auswahlmöglichkeiten (nichts erlauben)
mouseEntered	
mouseExited	
mousePressed	
mouseReleased	

4.3.4. Datenverzeichnis

Für die Speicherung, von z.B. Konfigurationsinformationen für die automatische Generierung, ist die Angabe eines Datenverzeichnisses notwendig, in das diese Daten abgelegt werden.

4.4. Operationsaufzeichnungsprinzip

Zum Verständnis der Funktionsweise verschiedener Aspekte auf Implementierungsebene des MAUDA-Systems ist es notwendig zu wissen, wie letztendlich Grund- und Teiloperationen intern verwaltet werden. Da das Aufzeichnungsprinzip auch einer der Kernkomponenten des Systems darstellt, soll es hier kurz erklärt werden.

Verantwortliche Java-Klasse: `mauda.OperationRecorder`

Prinzipiell werden immer alle vom Benutzer durchgeführten Operationen (Grund- und Teiloperation) in einem sequentiellen Vector abgespeichert. Dabei enthält jede der schon ausgeführten Grundoperationen nach Ausführung auch die korrekten Teiloperationen. D.h.,

dass das Objekt zur Speicherung der Grundoperation einen Eintrag für dessen Teiloperationen enthält. Daraus ergeben sich folgende drei Situationen:

- Grundoperation wird ausgeführt
- Teiloperation wird ausgeführt
- Grundoperations-Template wird ausgeführt.

Das Grundoperations-Template ist ein Spezialfall einer Grundoperation. Wenn es zur Ausführung kommt, führt das dazu, dass es auf einer aktuellen Kopie der Datenstruktur ausgeführt wird und somit die korrekten Teiloperationen enthält, ohne diese permanent ausgeführt zu haben. Dadurch ist es möglich, später bei der Evaluation festzustellen, was die korrekte Teiloperation wäre, wenn eine falsche ausgeführt wurde.

4.4.1. Beispiel

Der Student befindet sich mitten in einer Aufgabe. Er fordert die nächste zu bearbeitende Grundoperation an und dies sei ein Delete-min. Danach führt er ein korrektes „remove 3“ und ein falsches „link 5 4“ aus. Die richtige link-Operation sei „link 4 5“. Dann sieht der Aufzeichnungsvektor folgendermaßen aus:

Eintragsnr.	Eintrag im Vektor	Aktion
....		
x	Delete-min (Template)	Anfordern der nächsten Grundoperation
x + 1	Remove 3	Vom Studenten ausgeführt
x + 2	Link 5 4	Vom Studenten ausgeführt
...		

Durch das Anfordern der nächsten Grundoperation wird diese automatisch auf der Datenstruktur ausgeführt, was dazu führt, dass in ihr die korrekten Teiloperationen eingetragen werden. Somit folgt:

Delete-min (Template) enthält „remove 3“, „link 4 5“ und zum Schluss „updatemin y“.

Somit ist es möglich durch Vergleich des Inhalts des Templates mit den folgenden ausgeführten Teiloperationen herauszufinden, dass der Fehler im „link 5 4“ liegt. Ein weiterer Vorteil ist, dass dieses Prinzip relativ flexibel ist. Denn korrekte Teiloperationen zu einer Grundoperation werden erst dann berechnet, wenn man sie erreicht, wodurch eine Unabhängigkeit zu vorigen fehlerhaft durchgeführten Grundoperationen erreicht wird.

4.5. Feedback: Knowledge-Base

In Kapitel 3 (Abschnitt 3.6: Feedback-Konzept) wurde das Feedback-Konzept bereits erklärt, aber nicht beschrieben, wie dies letztendlich umgesetzt wurde.

Wie dort beschrieben, werden Nachrichten an Formeln gebunden.

Verantwortliche Java-Klassen:

- mauda.feedback.FeedbackKB
- mauda.feedback.FeedbackKBObject
- mauda.plugin.KBFormulaEvaluator

Zunächst soll der Aufbau von Formeln erklärt werden.

Es gibt folgende Platzhalter in den Formeln, die durch bestimmte Texte ersetzt werden:

Variable	Modifizierer	Ersetzung
correct		TRUE, wenn die aktuell ausgeführte Teiloperation korrekt ist, sonst FALSE
firstincorrect		TRUE, wenn die aktuell ausgeführte Teiloperation die erste falsche innerhalb der aktuellen Grundoperation ist, sonst FALSE
missingsubop		TRUE, wenn eine Anfrage an die KB gemacht wird die sich auf fehlende Teiloperationen beziehen soll
op[nr]	id	Zurückgeben der Grundoperations-ID, z.B. „delete“
	param1	1. Parameter der Grundoperation zurückgeben
	param2	2. Parameter der Grundoperation zurückgeben
subop[nr]	id	Zurückgeben der Teiloperations-ID, z.B. „remove“
	param1	Zurückgeben des 1. Parameters der Teiloperation
	param2	Zurückgeben des 2. Parameters der Teiloperation
	count	Gibt zurück, die wievielte Teiloperation (als Zahl) dieses Typs diese innerhalb der zugehörigen Grundoperation ist. Somit kann z.B. überprüft werden ob die aktuelle link-Teiloperation die erste ausgeführte ist.
correctsubop[nr]	id	Analog zu subop[nr], nur das die Werte der an dieser Stelle korrekten Teiloperation zurückgegeben werden
	param1	
	param2	
current		Referenz auf aktuelle Grundoperation oder Teiloperation d.h., dass subop[current].id das ID der aktuell durchgeführten Teiloperation zurückgibt. Konstrukte wie subop[current-1].param1 sind auch möglich können aber nicht immer ersetzt werden. "current" wird nur innerhalb eckiger Klammern in op, subop und correctsubop interpretiert.

Meistens wird für „nr“ „current“ verwendet, da man ja meist die aktuellen Operationen abfragen will. Es kann aber manchmal auch sinnvoll sein, die erste ausgeführte Teiloperation mittels subop[0] anzusprechen.

Ausdrücke, die nicht ersetzt werden können, wie z.B. subop[current+1] unter der Bedingung, dass es keine folgende Teiloperation gibt, führen zur Unerfüllbarkeit der ganzen Formel.

Erlaubt sind nur AND und NOT, was meist aber auch ausreicht. Klammern sind nicht zulässig. Als Vergleichsoperatoren sind „=“, „!=“, „>“ und „<“ erlaubt.

Die oben erklärten Variablen werden von einer allgemeinen Klasse ersetzt. Dies reicht jedoch nicht aus, um sämtliche Begebenheiten auszudrücken. Deshalb müssen Spezialfälle Plug-In-spezifisch ersetzt werden.

Das Fibonacci-Heap-Plug-In unterstützt folgende zusätzlichen Modifizierer, die an die oben genannten durch den Verkettungs-Operator „.“ angehängt werden können.

Modifizierer	Erlaubt auf	Ersetzung
pre	Alles	Auf Vorgängerzustand zurückgreifen
parent	param1, param2	Schlüssel des Vaters zurückgeben
child	param1, param2	Schlüssel des Kindes zurückgeben
left	param1, param2	Schlüssel des linken Nachbarn zurückgeben
right	param1, param2	Schlüssel des rechten Nachbarn zurückgeben
ismarked	param1, param2	TRUE, wenn aktueller Knoten markiert, sonst FALSE
rank	param1, param2	Anzahl Kinder des aktuellen Knotens zurückgeben
isroot	param1, param2	TRUE, wenn Knoten in der Wurzelliste ist, sonst FALSE
min	Nichts	Gibt den kleinsten Schlüssel des Baumes zurück (nicht zu Verwechseln mit dem Knoten auf den der Minimums-Zeiger gerade zeigt)

Verkettungen der Modifizierer sind erlaubt, sofern sie einen Sinn ergeben.

Beispiele:

“subop[current].param1.child.pre.parent” ist erlaubt da param1 ein Kind hat. Dieses hat einen Vorgängerzustand und eventuell auch einen Vater.

“subop[current].param1.isroot.pre“ ist nicht erlaubt da isroot einen Boolean zurückliefert und somit kein Modifizierer mehr angewandt werden kann.

Ein Eintrag in der Knowledge-Base muss folgendermaßen aussehen:

```
FORMULA: Formel
[FORMULA: ...]
[...]
MESSAGE: parametrisierte anzuzeigende Feedback-Nachricht
[MESSAGE: ...]
[...]
```

Werden mehrere Formeln unmittelbar hintereinander angegeben, wird die Nachricht angezeigt, wenn bereits eine der Formeln wahr wird. Somit kann ein ODER zwischen den Formeln erreicht werden.

Mehrere Nachrichten bedeuten, dass diese später mittels Vor- und Zurück-Links angesprochen werden können. Somit ist es möglich, schrittweises Feedback zu geben.

Die Feedback-Sätze sind parametrisiert insofern, als bestimmte Zeichenfolgen ersetzt werden:

Parameter	Ersetzung
SUBOPERATIONID	ID der aktuellen Teiloperation
SUBOPERATION	Komplette Bezeichnung einer Teiloperation. Z.B.: link 4 5
SUBOPPARAM1	1. Parameter der aktuellen Teiloperation
SUBOPPARAM2	2. Parameter der aktuellen Teiloperation
CORRECTSUBOPERATIONID	Referenz auf korrekte Teiloperation. Gibt es keine korrekte Teiloperation weil zu allen korrekten zusätzliche Teiloperationen ausgeführt wurden, wird in allen Fällen „Next Operation“ zurückgegeben
CORRECTSUBOPERATION	
CORRECTSUBOPPARAM1	
CORRECTSUBOPPARAM2	
OPERATIONID	ID der aktuellen Grundoperation
OPERATION	Komplette Bezeichnung einer Grundoperation. Z.B.: Delete 5
OPPARAM1	1. Parameter der aktuellen Grundoperation
OPPARAM2	2. Parameter der aktuellen Grundoperation

Der Satz aus Kapitel 3 (Abschnitt 3.6: Feedback-Konzept) lässt sich dann folgendermaßen kodieren:

Zu kodierender Satz:

Wenn die aktuell ausgeführte Teiloperation die erste falsche Teiloperation innerhalb eines Delete-mins oder Delete (min) ist, es sich um eine link-Operation handelt, und der erste Parameter der link-Operation größer als der zweite Parameter ist, dann gib „A link links always the bigger node to the smaller node, so that the bigger node is a child of the smaller node. Otherwise are the Heap-Constraints violated.“ als Feedback aus.

Eintrag in der Knowledge-Base:

```
FORMULA: firstincorrect AND subop[current].id = link AND op[current].id =
delete_min AND subop[current].param1 > subop[current].param2

FORMULA: firstincorrect AND subop[current].id = link AND op[current].id = delete
AND correctsubop[0].id = remove AND subop[current].param1 > subop[current].param2

MESSAGE: A link links always the bigger node to the smaller node, so that the
bigger node is a child of the smaller node. Otherwise are the Heap-Constraints
violated.
```

Zu beachten ist, dass FORMULA und MESSAGE jeweils in einer Zeile stehen und direkt aufeinander folgen müssen. Das heißt, dass weder in Formeln noch in Nachrichten Zeilenumbrüche vorhanden sein dürfen. Leerzeilen zwischen FORMULA und MESSAGE sind ebenfalls nicht gestattet.

Die Blöcke an sich können jedoch beliebig getrennt werden; auch durch anderen Text. Dieser wird einfach nicht beachtet.

4.5.1. Regeln für die Formeln

Folgende Regeln müssen beim Erstellen von Formeln eingehalten werden, da diese sonst nicht geparkt werden können:

- Sämtliche Operatoren wie „=“, „!=“, „AND“, „NOT“ usw. müssen mit Leerzeichen umgeben sein
- Kettenausdrücke, die durch Modifizierer entstehen, dürfen keine Leerzeichen enthalten. So ist z.B: „subop[current]. id“ nicht erlaubt
- Klammern sind nicht erlaubt
- In eckigen Klammern dürfen keine aussagenlogischen Ausdrücke stehen, so dass dort nur Ausdrücke wie „current-1“ oder direkte Zahlenangaben erlaubt sind.

4.5.2. Ablauf der Auswertung einer Formel

An Hand eines kleinen Beispiels soll gezeigt werden, wie Formeln schrittweise ausgewertet werden und an welcher Stelle das Plug-In eingreift.

Beispielformel:

```
subop[current].id = remove AND subop[current].param1.pre.isroot
```

Randbedingungen:

Aktuelle Teiloperation sei ein „remove 5“ und 5 war vor Ausführung dieser Operation nicht in der Wurzelliste. Des Weiteren soll „remove 5“ die zehnte ausgeführte Operation sein.

4.5.2.1. Schritt 1:

Teilausdrücke werden so weit es geht ohne Benutzung des Plug-Ins ausgewertet. Wird das Plug-In benötigt, wird an den teilweise ausgewerteten Ausdruck eine Zahl in eckigen Klammern angehängt. Diese Zahl kennzeichnet die Position innerhalb des Operationsaufzeichners (OperationRecorder).

➔ remove = remove AND 5[10].pre.isroot

4.5.2.2. Schritt 2:

In Schritt 2 werden Plug-In-spezifische Teile ausgewertet, indem der in Schritt 1 ausgewertete Teilausdruck an die getSpecialConstant-Methode des Plug-Ins übergeben wird.

➔ remove = remove AND FALSE

4.5.2.3. Schritt 3:

Im dritten Schritt werden nun einzelne Teilausdrücke, wie „=“, „!=“, „>“, ausgewertet.

➔ TRUE AND FALSE

4.5.2.4. Schritt 4:

Im letzten Schritt wird nun der Wahrheitswert der Formel ermittelt.

→ FALSE = Formel ist nicht erfüllt

4.5.3. Bemerkung

Soll kein genaues Feedback angezeigt werden, sondern lediglich vermittelt werden, was die korrekte Operation wäre, kann man folgende drei Einträge in der KB verwenden:

FORMULA: correct

MESSAGE: That's correct

FORMULA: NOT correct AND NOT missingsubop

MESSAGE: CORRECTSUBOPERATION is the correct operation!

FORMULA: missingsubop

MESSAGE: There are missing suboperations!

5. Leistungsbewertung

Um die Leistung des Systems zu beurteilen, wird an dieser Stelle verglichen, ob die Grundsätze der ACT-Theorie aus Kapitel 2 (Abschnitt 2.1.1: ACT-Theorie) eingehalten worden sind und welche der gestellten Anforderungen an das System erfüllt wurden.

5.1. *Einhaltung der Grundsätze der ACT-Theorie*

Durch das Konzept der Unterteilung von Grundoperationen in Teiloperationen wird sowohl ein Modell der Zielfertigkeit erreicht, als auch der Problemlöseprozess in Unterziele aufgeteilt. Somit werden die ersten zwei Grundsätze erfüllt.

Dadurch, dass Instruktionen zur Aufgabenbearbeitung nicht nur im Gesamten sondern auch während der Bearbeitung in Form „nächster durchzuführender Grundoperation“ gegeben werden, ist Grundsatz 3 ebenfalls erfüllt.

Durch Platzierung von allgemeinen Feedback-Nachrichten wie „Ein unmarkierter Knoten, der ein Kind verliert, muss markiert werden!“ und keine Nachrichten der Art „Dieser Knoten muss markiert werden!“ folgt, dass die Generalisierung von Regeln laut Grundsatz 4 gefördert wird. Des Weiteren kann der Student auch mehrere unterschiedliche Aufgaben anfordern und muss somit sein Wissen durch Bearbeitung verschiedener Übungen generalisieren.

Laut Grundsatz 5 soll das Arbeitsgedächtnis möglichst wenig während der Bearbeitung von Aufgaben belastet werden. Dies wird zum einen durch das verwendete Editierungskonzept erreicht und zum anderen durch Visualisierung der bereits ausgeführten Operationen.

Da ein sofortiges Feedback durch „Immediate Feedback And Error Correction“ möglich ist, wurde auch Grundsatz 6 eingehalten.

Grundsatz 7 wurde schon in Kapitel 3 im Editierungskonzept (Abschnitt 3.3: Editierungskonzepte) angesprochen. Da dieser Grundsatz nicht klar definiert, wie genau ein Editierungskonzept aus zu sehen hat, lässt sich an dieser Stelle nur behaupten, dass der gewählte Ansatz „Function-based“ ein gutes ausbalanciertes Konzept darstellt.

Da nach 3-maligem falschem Ausführen einer Teiloperation automatisch die korrekte ausgeführt wird, und ein angepasstes Feedback bei Fehlern gegeben wird, werden Wissenslücken des Studenten durch den Tutor gefüllt. Allerdings erfolgt keine Annäherung des Tutors an die Zielfertigkeit. Das heißt, dass zu jeder falschen Teiloperation mit gleichem Kontext immer dasselbe Feedback angezeigt wird¹² und somit keine Anpassung des Feedbacks an den Studenten durchgeführt wird. Um das zu erreichen, müsste eine benutzerspezifische Datenbank angelegt werden, die individuelle Fehler bestimmter Studenten abspeichert, um passende individuelle Feedback-Nachrichten zu generieren.

¹² statisches Feedback

5.2. Erfüllte Anforderungen

In Kapitel 1 (Abschnitt 1.3: Anforderungen an das zu entwickelnde System) wurden die Anforderungen an das System vorgestellt. An dieser Stelle wird nun diskutiert, inwieweit diese im Projekt erfüllt wurden.

5.2.1. Allgemein

Sämtliche Anforderungen an die Darstellung wurden erfüllt. Ferner ist ein Plug-In-Konzept vollständig ausgearbeitet und implementiert worden, welches für alle Algorithmen und Datenstrukturen anwendbar sein sollte.

5.2.2. Vorbereitungsphase

In der Vorbereitungsphase wurde eine automatische Generierung von Aufgaben unterschiedlichen Schwierigkeitsgrades gefordert. Dies wird von dem entwickelten MAUDA-System nur teilweise erfüllt, da sich bisweilen nur Normal-Mode-Aufgaben (siehe Abschnitt 3.2: Aufgabenkonzept) und keine Fault-Mode-Aufgaben automatisch generieren lassen. Das Erstellen von Fault-Mode-Aufgaben durch eine Person, ist aber ohne Einschränkungen im GenEditor¹³ komfortabel durchführbar. Alle sonstigen Anforderungen an die Vorbereitungsphase wie Einbettung des AutoGenerators (automatische Generierung von Aufgaben), Replay der Aufgaben, etc. wurden erfüllt.

5.2.3. Durchführungsphase

In dieser Phase ist es möglich, Aufgaben aus dem Repository anzufordern oder komplett neue Aufgaben vom AutoGenerator generieren zu lassen, die dann bearbeitet werden können.

Es wurden die vier Feedback-Schemata (siehe Abschnit 2.1.2: Tutortypen) und ein Schema für Fault-Mode-Aufgaben entwickelt, die ein angepasstes Feedback erlauben. Aufgaben können jederzeit gespeichert werden, wobei komplett bearbeitete Aufgaben sofort automatisch im Repository abgelegt werden. Somit sind alle Anforderungen an diese Phase erfüllt worden.

5.2.4. Evaluationsphase

Eine automatische Evaluierung kann mit dem System durchgeführt werden, indem Punkte im Bereich von 0 bis 100 für beide Aufgabentypen (Normal-Mode und Fault-Mode) automatisch vergeben werden. Des Weiteren werden die gemachten Fehler aufgezeichnet und könnten in einem noch zu erstellendem Statistik-Programm ausgewertet werden. Die Funktionalität für eine tutorielle Evaluation ist vorhanden, wobei anzumerken ist, dass noch kein Plug-In existiert, mit dem dies komplett möglich wäre. Die vorhandene Fibonacci-Heap-JEDAS-Animations-Klasse [10] kann nicht verwendet werden, da diese bei einer JEDAS-Aufnahme zu Fehlern führt und somit anschließend nicht abgespielt werden kann.

¹³ Aufgabengenerierungsprogramm

6. Schlussfolgerungen und Ausblick

Um ein Assessment-System zu Erstellen, bedarf es der Ausarbeitung und Evaluation einiger wichtiger Aspekte.

So spielen psychologische Faktoren eine große Rolle, um den Lerneffekt des Studenten durch das System zu maximieren. Da psychologisch gesehen das Zusammenführen von Chunks, also Teilinformationen zu größeren Blöcken, gefördert werden soll, ist eine gute Auseinandersetzung mit dem Editierungskonzept unabdingbar, da dort die Chunk-Größe als Unterteilung von Grundoperationen vorgegeben wird. Auch die Erkenntnisse der ACT-Theorie sollten mit eingebracht werden, da diese den Lernprozess aus psychologisch kognitiver Sicht beschreiben, woraus sich Grundsätze für Assessment-Systeme und Computer-Tutoren ableiten lassen.

Ein wichtiger Punkt, um psychologische Aspekte einzuhalten, ist die Ausarbeitung eines Konzeptes, welches es erlaubt, zur aktuellen Situation ein angepasstes Feedback zu geben. Bei den meisten web-basierten Systemen ist das kein Problem, da sie sowieso nur Multiple-Choice-Fragen, Matching-Fragen, usw. anbieten, und somit zu jeder falschen Antwort statisch ein Feedback-Satz angegeben werden kann. Bei dem MAUDA-System sieht das allerdings etwas anders aus, da man hier keine statischen Sätze angeben kann. Stattdessen müssen die Randbedingungen der aktuellen Situation abstrakt beschrieben werden, um ein gutes und vor allem auch passendes Feedback zu geben. In dieser Arbeit wurde das Prinzip der Knowledge-Base mit Formeln und assoziierten parametrisierten Sätzen erarbeitet.

Des Weiteren ist es notwendig die Probleme und Lösungsansätze anderer zu recherchieren, um dieselben Fehler nicht noch einmal zu machen. Darunter fallen Probleme wie die Handhabung der eventuell riesigen Lösungsräume von Aufgaben, als auch die Eliminierung von Betrugmöglichkeiten, welche bei statischen Fragen eine sehr große Rolle spielt.

Ferner sollten verschiedene existierende Assessment-Systeme untersucht werden, um Erfahrungen und Verbesserungsvorschläge dieser zu erörtern und um aus jedem das Beste herauszupicken. Dabei stellt sich vielleicht auch heraus, ob nicht ein bereits existierendes System für den neuen Aufgabenbereich genutzt werden kann. Für das MAUDA-Projekt war dies schon von vornherein geklärt, da JEDAS zur Darstellung und Animation benutzt werden soll.

Auch die automatische Generierung von Aufgaben ist ein nicht zu unterschätzender Teil des Projektes. Das komplizierte an der Generierung ist nicht das Aufbauen einer Aufgabe, sondern vielmehr unterschiedliche Schwierigkeitsgrade zu erreichen. Hierzu muss man die Problemstellung genau analysieren und ein geeignetes Verfahren entwickeln. In diesem Projekt wurde tiefenbeschränkte Breitensuche mit Bewertung verwendet. Dieses Verfahren erzielt zumindest für den Bereich Algorithmen und Datenstrukturen gute Ergebnisse (siehe auch Abschnitt 3.4: Automatische Generierung von Aufgaben).

Ein letzter Schritt ist die automatische Evaluation von Aufgaben. Auch hierzu muss überlegt werden, was konkret in die Bewertung mit einfließt. Das Konzept, das im MAUDA-System verwendet wurde, arbeitet dabei relativ einfach, da alle gemachten Fehler des Studenten protokolliert werden und entsprechend zu negativen Punkten führen, die dann wiederum von der erreichbaren Gesamtpunktzahl abgezogen werden.

Für das Front-End ist es wichtig, sich an gängige Standards von GUI's zu halten. Das schließt unter anderem die Erstellung von Menüs und Toolbars ein. So sollte zum Beispiel die Position der Toolbar am oberen Bildschirmrand sein, während die gewählten Bilder der

Toolbar zu den assoziierten Aktionen passen sollten. Insbesondere sollten Aufgaben animiert dargestellt werden, da dies, aus psychologischer Sicht, essentiell den Lernprozess unterstützt. Außerdem sollte eine tutorielle Evaluation in Form von Annotationen zu den erstellten Animationen möglich sein.

6.1. Stärken und Schwächen von MAUDA

Eine der Stärken des Systems ist sicherlich die automatische Generierung von Aufgaben unterschiedlichen Schwierigkeitsgrades. Hier wird nicht wie bei TRAKLA2 eine starre Skelett-Struktur verwendet, sondern es können beliebige Strukturen entstehen. Das Verfahren liefert gute Ergebnisse und lässt sich auf andere Problemstellungen skalieren. Durch das Einstellen verschiedener Parameter lassen sich Schwerpunkte auf bestimmte Aspekte des Algorithmus setzen. So kann man beispielsweise den Schwerpunkt auf Teiloperationen setzen, welche im Falle von Fibonacci-Heaps Markierungen setzen.

Die komfortablen GUI's erlauben das Erstellen von Aufgaben jeglicher Art. Insbesondere bei Fault-Mode-Aufgaben lassen sich die korrekten Teiloperationen anzeigen, um das Erstellen solcher Aufgaben zu vereinfachen. Des Weiteren werden generierte Aufgaben vor dem Speichern auf Korrektheit überprüft, so dass zum Beispiel bei einer Fault-Mode-Aufgabe geprüft wird, ob überhaupt ein Fehler existiert. Die Baumartige Darstellung von Aufgaben in allen Programmen erlaubt einen kompletten Überblick über ausgeführte Operationen.

Das Plug-In-Konzept mit seinen wenigen Vorraussetzungen führt dazu, dass viele Problembereiche abgedeckt werden können.

Dank dem mächtigen Feedback-Konzept ist ein angepasstes Feedback zu nahezu jeder Situation möglich.

Trotzdem weist das System in gewissen Punkten noch Schwächen auf. So werden, z.B. bei der Bearbeitung von Aufgaben, nicht sämtliche Aktionen aufgezeichnet, so wie das bei Exorciser der Fall ist. Das heißt, dass z.B. das Drücken von Undo, Redo oder korrigierte Teiloperationen nicht vollständig mitprotokolliert werden. Dafür wird aber jeder gemachte Fehler so protokolliert, dass es möglich sein sollte, den Ablauf zumindest im Bezug auf alle durchgeführten Operationen zu rekonstruieren.

Des Weiteren wird auch keine Möglichkeit angeboten, dass der Student zu Übungszwecken eine angeforderte Aufgabe selbst modifizieren kann (vgl. Exorciser).

Was die Feedback Knowledge-Base betrifft, so hat es den Anschein, als wären Formeln leicht zu erstellen. Sind sie aber meist doch nicht, da sie oftmals auf andere Zustände auch zutreffen, die man nicht bedacht hat, wodurch ein unzutreffendes Feedback gegeben wird, das einen in die Irre führen kann. Außerdem wird kein individuelles Feedback unterstützt, so dass das Feedback nicht auf den jeweiligen bearbeitenden Studenten zugeschnitten wird.

Durch das vollständige Fehlen einer Report-Funktion können bislang keine Analysen über mehrere Personen getroffen werden.

Diese Schwächen bilden aber die Grundlage für zahlreiche verschiedene Erweiterungsmöglichkeiten des Systems. Im Folgenden sollen einige erläutert werden.

6.2. Erweiterungsmöglichkeiten

6.2.1. Statistik/Report

Report-Funktionen sind in Assessment-Systemen schon fast Pflicht geworden, da sich diese gerade durch die Verwendung von Computern stark automatisieren lassen. Sie erlauben einen Einblick in die häufigsten Fehler, die von Studenten gemacht wurden, woraufhin man diese Fehler in der Vorlesung noch einmal besprechen könnte. Des Weiteren könnte der Student feststellen, wie gut er gegenüber dem Klassendurchschnitt abschneidet.

In dem bis jetzt entwickelten System wurde ein Grundstein für eine Report-Funktion schon gelegt, dadurch, dass sämtliche gemachte Fehler protokolliert werden. Diese müssten von einem neuen Statistik-Programm ausgelesen und analysiert werden.

Eine gute Orientierungshilfe schafft hier das vorgestellte Assessment-System Perception von Questionmark, das durch seine vielseitigen verschiedenen Report-Funktionen glänzt.

6.2.2. Bündeln von Plug-Ins

Bisweilen müssen Plug-Ins in Bezug auf die automatische Generierung komplett neu geschrieben oder vom Fibonacci-Heap-Plug-In übernommen werden, so dass die tiefenbeschränkte Suche für jedes Plug-In neu implementiert werden muss. Die Idee ist nun, das Prinzip „Suche im Zustandsraum“ in allgemeinen Klassen abzulegen, welche dann durch verschiedene Plug-Ins genutzt werden können.

In Exorciser wird dies bereits praktiziert, in dem es drei Plug-In-Gruppen gibt, welche ihrerseits mehrere abgeleitete Plug-Ins besitzen.

6.2.3. Meta-Daten-Browser

Durch die Vereinheitlichung der Meta-Daten aller von dem Projekt erzeugten Aufgabendateien wäre es theoretisch möglich, alle Dateien in einem Verzeichnis abzulegen und mittels eines speziellen, zu entwickelnden Datei-Browsers, zu durchforsten oder nur entsprechende zur Zeit ladbare Dateien in diesem Datei-Browser anzuzeigen.

Dies war übrigens die Grundidee, die hinter den einheitlichen Meta-Daten steckte.

6.2.4. Web-Service / SOAP

Das System könnte durch die Möglichkeit eines Web-Services via SOAP-Protokoll erweitert werden. Somit könnten Studenten Aufgaben aus einem Repository übers Internet anfordern, bearbeiten und wieder einsenden. Nach einer eventuellen tutoriellen Evaluation wäre es dem Studenten möglich, diese wieder übers Web anzufordern und zu betrachten.

6.2.5. Betrugselimination

Dadurch, dass Aufgaben, die in Bearbeitung sind, vom Studenten gespeichert werden können, ist ein Betrug möglich, und zwar insofern, als er vor jeder Aktion die Aufgabe speichert. Macht er nun einen Fehler kann er die Aufgabe wieder laden und erneut die aktuelle Teiloperation durchführen. Diesen Vorgang kann er wiederholen, bis er jede Teiloperation richtig ausgeführt hat und somit die volle Punktzahl erhält.

Um diesem entgegenzuwirken, müsste ein Konzept ausgearbeitet werden, das dies verbietet.

6.2.6. Automatische Generierung von Fault-Mode-Aufgaben

Zu diesem Punkt wurden schon einige Überlegungen angestellt, doch leider entstanden bisweilen nur Konzepte, die relativ triviale Fehler erzeugten oder nicht immer anwendbar waren. So wurde diskutiert, Fehler auf folgende Arten zu erzeugen:

Erstens wurde erörtert, Teiloperationen aus einer Abfolge von korrekten Teiloperationen zu überspringen. Dies funktioniert allerdings nicht, da folgende Teiloperationen eventuell von vorherigen abhängig sind. Das einfachste Beispiel ist hierzu ein Insert eines neuen minimalsten Knotens. Würde hier nun das Einfügen an sich übersprungen, könnte das nachfolgende Aktualisieren des Minimums auch nicht durchgeführt werden.

Zweitens wurde überlegt, alle korrekten Teiloperationen ab einer bestimmten Stelle abzuschneiden. Leider erzeugt dies meist triviale Fehler.

Drittens: Verdrehen der zwei Parameter einer Teiloperation. Doch hier ergibt sich das Problem, dass es viele Teiloperationen gibt, die keine zwei Parameter besitzen und wenn doch, dann dürfen diese eventuell gar nicht verdreht werden, da sie den Sinn einer Teiloperation verändern oder nicht mehr ausführbar wären. So würde das Verdrehen der Parameter bei einer setkey-Teiloperation einer Decrease-key Grundoperation dazu führen, dass die Funktion gar nicht mehr ausführbar wäre, da nun versucht wird ein Knoten herabzusetzen, den es gar nicht gibt, und zwar auf einen Wert der schon existiert. Hingegen bei einer link-Teiloperation würde dieses Verfahren einen „schönen“ Fehler erzeugen.

Somit wurde festgestellt, dass das automatische Generieren einer guten Fault-Mode-Aufgabe einiges an spezieller Logik bedarf und wurde somit nicht weiter verfolgt.

6.2.7. Entwerfen weiterer Plug-Ins

Da das System relativ flexibel gegenüber neuen Plug-Ins aus dem Bereich Algorithmen und Datenstrukturen ist, wären Plug-Ins für Binomial-Queues, AVL-Bäume, Bin-Packing, etc. eine interessante Erweiterung, um dem Studenten auch solche Probleme anbieten zu können. Wie dies im Einzelnen durchzuführen ist, kann dem Abschnitt 3.7: Plug-In-Konzept und Abschnitt 4.3: Plug-In-Handler entnommen werden.

6.2.8. Tests an Personen

Interessant wäre sicher auch eine empirische Untersuchung bezüglich der Leistungen von Studenten, die dieses System benutzen. Dabei könnte man mehrere Gruppen von Personen bilden, die sich in der Benutzung der verschiedenen Tutortypen unterscheiden. Zusätzlich wäre eine Kontrollgruppe zu bilden, die das System gar nicht benutzt. Durch eine anschließende Befragung der Testteilnehmer könnte man das System sicher noch in bestimmten Dingen verbessern.

6.2.9. Verlagerung der Korrektheit in die Knowledge-Base

Die Bewertung von bestimmten Situationen würde es eigentlich erfordern, die Punktevergabe bei der Evaluation in die Knowledge-Base zu verlagern. Ein Beispiel wäre, wenn man bei einem consolidate der Grundoperation Delete-Min ganz links im Baum anfängt, anstatt bei dem aktuellen Zeiger auf das Minimum zu beginnen (siehe Abschnitt

3.3: Editierungskonzepte). Eine solche Aktion wirkt sich bisweilen gleich stark aus, als hätte man eine link-Teiloperation falsch ausgeführt. Theoretisch sollte sie aber nicht ganz so negativ bewertet werden.

7. Zusammenfassung

In dieser Diplomarbeit wurde ein Assessment-System zur Generierung, Durchführung und Evaluation von Aufgaben erstellt. Dabei wurde der gesamte Entwicklungsprozess von der Recherche psychologischer Grundlagen, bereits existierender Assessment-Systeme, benötigter Konzepte, bis zur Planung und Implementierung eines eigenständigen Systems beschrieben.

Zu Beginn wurde erklärt, wie der Lernprozess sich gemäß der ACT-Theorie in drei Stufen unterteilt. Zur Verdeutlichung wurden Beispiele anhand Fibonacci-Heap's gegeben. Aus ihr leiteten sich acht wichtige Grundsätze für Computertutoren ab.

Danach wurde erklärt, woraus sich letztendlich eine Aufgabe zusammensetzt und wie diese bearbeitet werden soll. Es entstanden Abarbeitungskonzepte für die beiden Typen Fault-Mode und Normal-Mode. Ferner wurde ein Editierungskonzept vorgestellt, welches beschreibt, wie Ziele in Teilziele zerlegt wurden.

Für die Vorbereitungsphase wurden Konzepte zur Bewertung von Aktionen ausgearbeitet. Aus der gewählten Methodik entstand ein Verfahren zur automatischen Generierung von Aufgaben. Dieses Verfahren entspricht im Wesentlichen einer tiefenbeschränkten Breitensuche, indem Übergänge (Teiloperationen) mit einem bestimmten Gewicht bewertet werden. Es wurde diskutiert, inwiefern dieses Verfahren zu Aufgaben unterschiedlichen Schwierigkeitsgrades führt.

Der zentrale Teil der Durchführungsphase bildet ein Konzept zur Generierung von Feedback-Nachrichten. Es wurde ein mächtiges Verfahren entwickelt und beschrieben, welches eine Wissensbasis verwendet. In ihr können bestimmte Situationen einer Aufgabe in Formeln kodiert und entsprechende parametrisierte Sätze zugewiesen werden. Durch vier verschiedene Tutortypen, die diese Wissensbasis benutzen, wird sowohl der Zeitpunkt als auch der Inhalt des Feedbacks zusätzlich gesteuert.

Gemäß der Evaluationsphase entstand sowohl ein Modul, welches Punkte in Abhängigkeit der gemachten Fehler automatisch vergibt, als auch ein Modul zur späteren tutoriellen Evaluation. Es wurde beschrieben, wie damit Aufgaben durch eine Person mittels JEDAS-Aufnahmen evaluiert werden können.

Um das System in verschiedenen Problembereichen anwenden zu können, wurde erörtert, wie das System Plug-In-fähig gemacht wurde. Des Weiteren wurden die verschiedenen GUI's kurz vorgestellt und einige wichtige Gesichtspunkte der Implementierung herausgegriffen, um eine spätere Weiterentwicklung durch Dritte zu fördern.

Am Schluss wurden die Stärken und Schwächen des Systems ausgearbeitet und einige wichtige Erweiterungsmöglichkeiten des Systems angesprochen.

Abbildungsverzeichnis

ABBILDUNG 1: AUTHORWARE: DESIGN WINDOW + PRESENTATION WINDOW	14
ABBILDUNG 2: EXORCISER NFA TO DFA CONVERSION	18
ABBILDUNG 3: JHAVÉ: AV-ENGINE: BEISPIEL 1: 0/1 RUCKSACK-PROBLEM	20
ABBILDUNG 4: JHAVÉ: AV-ENGINE: BEISPIEL 2: KMP – ZEICHENKETTEN-SUCH-ALGORITHMUS	21
ABBILDUNG 5: PERCEPTION: REPORT (WEB)	23
ABBILDUNG 6: PERCEPTION: REPORT (WIN)	24
ABBILDUNG 7: QUIZPACK: DISSECTION	25
ABBILDUNG 8: QUIZPACK: MENGEN-APPLET	26
ABBILDUNG 9: TRAKLA2: HEAP-APPLET	28
ABBILDUNG 10: TSA	30
ABBILDUNG 11: EDITIERUNGSKONZEPT: FIBONACCI-HEAP: KNOTENFORMAT	35
ABBILDUNG 12: AUTOMATISCHE GENERIERUNG: FLUSSDIAGRAMM	45
ABBILDUNG 13: GUI: GENEDITOR	54
ABBILDUNG 14: GUI: WORKEDITOR	55
ABBILDUNG 15: GUI: AUTOGENERATOR-AUFGABENANFORDERUNGS-AUSWAHL	56
ABBILDUNG 16: GUI: EVALEDITOR	56
ABBILDUNG 17: GUI: EVALPLAYER	57
ABBILDUNG 18: GUI: META-DATEN-EINGABE	57

8. Anhang

8.1. Literaturverzeichnis

- [1] Anderson, J. R., Corbett, A. T., Koedinger, K. R. & Pelletier, R. (1995). Cognitive Tutors: Lessons Learned. *Journal of the Learning Sciences*, 4(2), 167-207.
- [2] Bennett, R. E. (1998). Reinventing Assessment. A policy information perspective.. <http://www.ets.org/research/pic/bennett.html>
- [3] Brunsmann, J., Homrighausen, A., Six, H. & Voss, J. (1999). Assignments in a Virtual University The WebAssign-System. *Proceedings of the 19th World Conference on Open Learning and Distance Education (Vienna/Austria)*.
- [4] Brusilovsky, P. & Miller, P. (1999). Web-based Testing for Distance Education. In P. De Bra & J. Leggett (Ed.), *Proceedings of WebNet'99, World Conference of the WWW and Internet* (pp. 149-154). Honolulu.
- [5] Corbett, A. T. C. & Anderson, J. R. (2001). Locus of feedback control in computer-based tutoring: impact on learning rate, achievement and attitudes. *Proceedings of the SIGCHI conference on Human factors in computing systems* (pp. 245-252). Seattle, Washington, United States: ACM Press.
- [6] Hopper, M. (1998). Assessment in WWW-Based Learning Systems: Opportunities and Challenges. *Journal of Universal Computer Science*, 4(4), 330-348.
- [7] Imfeld, A. (2002). Lernkomponente zur Minimierung von Endlichen Automaten. Semesterarbeit (bei Professor: Nievergelt, J. / Assistenten: Tscherter, V. & Lamprecht, R.)
- [8] Jeger, S. P. (2000). Online-Testmethoden – Klassifikation, Implementierung und konzeptionelle Weiterentwicklung. Diplomarbeit. (Eingereicht bei: Prof. Dr. H. Schauer). Institut für Informatik der Universität Zürich)
- [9] Kandzia, P. & Trahasch, S. (2002). Tutored Assignments Going Online. *Proceedings of the 4th International Conference on New Educational Environments (ICNEE) Lugano, Schweiz*.
- [10] Lauer, T. (1999). Animation komplexer Datenstrukturen und der dazugehörigen Algorithmen am Beispiel der Fibonacci-Heaps. *Wissenschaftliche Arbeit im Rahmen der Staatsexamensprüfung für das Lehramt an Gymnasien im Fach Mathematik (betreut von Prof. Dr. Ottmann, T.)*
- [11] Naps, T. L., Norton, L. L. & Eagan, J. R. (2000). JHAVÉ -- An Environment to Actively Engage Students in Web-based Algorithm Visualizations. *Proceedings of the SIGCSE Session, ACM Meetings (Austin, Texas)*.
- [12] Naps, T. L., Rossling, G., Almstrum, V., Dann, W., Fleischer, R., Hundhausen, C., Korhonen, A., Malmi, L., McNally, M., Rodger, S. & Velazquez-Iturbide, J. A. (2002). Exploring the Role of Visualization and Engagement in Computer Science Education. *Report of the Working Group on "Improving the Educational Impact of Algorithm Visualization". Aarhus, Denmark*.
- [13] Pathak, S. & Brusilovsky, P. (2002). Assessing Student Programming Knowledge with Web-based Dynamic Parameterized Quizzes. In P. Barker & S. Rebelsky(Ed.), *Proceedings of ED-MEDIA'2002 - World Conference on Educational Multimedia, Hypermedia and Telecommunications* (pp. 1548-1553). Denver.
- [14] Tscherter, V., Lamprecht, R. & Nievergelt, J. (2002). Exorciser: Automatic Generation and Interactive Grading of Exercises in the Theory of Computation. *4th International Conference on New Educational Environments* (pp. 3.1.47-50). Lugano.

8.2. Linksammlung

8.2.1. Allgemein

- [WA1] Java EDucational Animation System
<http://ad.informatik.uni-freiburg.de/jedas/>

- [WA2] [Klaus Röder / Ramin Goettlich] / Kognitionswissenschaftliche Ansätze I (ACT-Theorie)
<http://wwwpaul.informatik.tu-muenchen.de/seminare/lehrsystemeSS98/Vortrag04/>
- [WA3] Technical Articles and Tips. Java look and feel Graphics Repository
<http://java.sun.com/developer/techDocs/hi/repository/>
- [WA4] Java™ 2 SDK, Standard Edition Documentation Version 1.4.2
<http://java.sun.com/j2se/1.4.2/docs/index.html>
- [WA5] [T. Ottmann, P. Widmayer] / Folien: Fibonacci-Heaps. Autor: Sven Schuierer. Institut für Informatik. Albert-Ludwigs-Universität Freiburg
<http://ad.informatik.uni-freiburg.de/bibliothek/books/ad-buch/k6/slides/FibonacciHeaps.pdf>
- [WA6] WebServices + SOAP
<http://www.torsten-horn.de/techdocs/soap.htm#WebServices>
- [WA7] [Sidney Pressey] Ohio state university. Erste dokumentierte Benutzung einer Test-Maschine
<http://www.coe.uh.edu/courses/cuin6373/idhistory/pressey.html>

8.2.2. Projektmanagement

- [WP1] Grundlagen des Projektmanagements. TU Berlin
<http://ig.cs.tu-berlin.de/ap/rl/2003-04/projektmanagement.pdf>
- [WP2] [Branahl, Jessenberger] Projektmanagement
<http://www.nt.hs-bremen.de/diglink/abschlussbericht/07.pdf>

8.2.3. Assessment-Systeme

- [WAS1] Authorware (Macromedia)
<http://www.e-teaching.org/technik/produkte/authorware>
- [WAS2] Authorware Web-Player Download
<http://www.macromedia.com/software/authorware/productinfo/webplayer/>
- [WAS3] Biomedical Admissions Test (BMAT)
<http://www.bmat.org.uk/>
- [WAS4] Computer Aided learning in mathematics (CALM). The CUE Assessment System
<http://www.calm.hw.ac.uk/cue.html#>
- [WAS5] [Jane Paterson] The CUE Assessment System. Department of Mathematics. Heriot-Watt University, Edinburgh EH14 4AS
<http://itsn.mathstore.ac.uk/articles/maths-caa-series/apr2002/index.shtml>
- [WAS6] EQL International Ltd. / Mit Authorware entwickelte Beispiele
<http://www.eql.co.uk/demonstrations.htm>
- [WAS7] JHAVÉ (Java-hosted Algorithm Visualization Environment)
http://www.uwosh.edu/faculty_staff/naps/paper.html
- [WAS8] Perception von Questionmark
<http://www.questionmark.com/deu/home.htm>
- [WAS9] QuizPACK (Quizzes for Parameterized Assessment of C Knowledge)
<http://www2.sis.pitt.edu/~taler/QuizPACK.html>
- [WAS10] TRAKLA2
<http://www.cs.hut.fi/Research/TRAKLA2/index.shtml>
- [WAS11] TSA (Thinking Skills Assessment)
<http://tsa.uct.ac.za/about.html>
- [WAS12] TSA (Thinking Skills Assessment). PDF, Introduction.... University of Cambridge. Local Examinations Syndicate.
<http://tsa.uct.ac.za/about.html>

[WAS13] TALER-Lab Homepage

<http://www2.sis.pitt.edu/~taler/index.html>

[WAS14] Klassifikation computerunterstützter Lehr- und Lernsysteme / Intelligente Tutorielle Systeme

<http://dsor.uni-paderborn.de/de/forschung/publikationen/blumstengel-diss/Klassifikation-computerunterstuetzter-Lehr--Lernsysteme.html>