HAW HAMBURG

Bachelorarbeit

Alexander Wingerath

Visualisierung von Lösungsversuchen auf einer Programmierübungsplattform mittels schrittweise durchlaufbaren Kontrollflussgraphen

Fakultät Technik und Informatik
Department Informatik

Faculty of Computer Science and Engineering
Department Computer Science

Alexander Wingerath

Visualisierung von Lösungsversuchen auf einer Programmierübungsplattform mittels schrittweise durchlaufbaren Kontrollflussgraphen

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung
im Studiengang *Bachelor of Science Angewandte Informatik*
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Axel Schmolitzky
Zweitgutachter: Prof. Dr. Klaus-Peter Kossakowski

Eingereicht am: 19. April 2023

Alexander Wingerath

Thema der Arbeit

Visualisierung von Lösungsversuchen auf einer Programmierübungsplattform mittels schrittweise durchlaufbaren Kontrollflussgraphen

Stichworte

Softwarevisualisierung, Informationsvisualisierung, OPPSEE, Kontrollflussgraph

Kurzzusammenfassung

Diese Bachelorthesis befasst sich mit der Entwicklung eines Systems für eine Programmierübungsplattform, das es ermöglichen soll, die auf der Plattform abgegebenen Lösungsversuche zu visualisieren. Dazu soll der Ablauf des Lösungsversuchs als Kontrollflussgraph dargestellt werden. Der Graph soll für vorgegebene Werte schrittweise durchlaufbar sein. Die Belegung der Variablen sowie die Inhalte von Sammlungen und 1- und 2-dimensionalen Arrays sollen in jedem Schritt angezeigt werden. Dafür werden die präattentive Wahrnehmung und die Gestalttheorie als Aspekte der menschlichen Wahrnehmung vorgestellt, die berücksichtigt werden sollen. Außerdem wird die Programmierübungsplattform und deren Struktur vorgestellt, nach der sich der Entwurf des Systems richtet. Im Anschluss wird untersucht, nach welchen Kriterien Informationsvisualisierungen gestaltet werden können, um dem Nutzer die präsentierten Daten angemessen zu übermitteln. Schließlich wird untersucht, wie der pädagogische Wert dieser Visualisierung, insbesondere für Programmieranfänger, erhöht werden kann. Auf Grundlage der präsentierten Erkenntnisse wird ein Entwurf vorgestellt, der diesen gerecht wird. Abschließend wird auf interessante Details der Implementierung eingegangen und das entwickelte System mit realen Daten getestet.

Alexander Wingerath

Title of Thesis

Visualization of solution attempts on a programming practice plattform by means of a stepable controlflowgraph

Keywords

Softwarevisualization, Informationvisualization, OPPSEE, Controlflowgraph

Abstract

This bachelor thesis deals with the development of a system for a programming practice platform, which should make it possible to visualize the solution attempts submitted to the platform. For this purpose, the solution attempt is to be represented as a control flow graph. The graph will be stepwise traversable for given values. The values of variables as well as the contents of collections and 1- and 2-dimensional arrays shall be displayed in each step. For this purpose, preattentive perception and Gestalt theory are introduced as aspects of human perception that shall be considered. In addition, the programming exercise platform and its structure are presented, according to which the design of the system is based. Next, the criteria by which information visualizations can be designed to appropriately convey the presented data to the user are examined. Afterwards, it is examined how the pedagogical value of this visualization can be increased, especially for novice programmers. Based on the presented findings, a design is presented that satisfies them. Finally, interesting details of the implementation are discussed and the developed system is tested with real data.

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

hier kann ein abstract als einführung rein

Um ein guter Programmierer zu werden, und viel Übung.

Die Hochschule für angewandte Wissenschaften Hamburg unterstützt ihre Studenten beim Lernen durch die Programmierübungsplattform OPPSEE (Online Programming Practice for Software Engineering Education). Auf dieser Plattform können sich Studenten aus einem Angebot vieler Programmieraufgaben unterschiedlichen Schwierigkeitsgrades eine aussuchen und versuchen diese in einer Web-IDE zu lösen. Wenn sie meinen, die Aufgabe gelöst zu haben, können sie die Korrektheit des Lösungsversuchs durch hinterlegte Tests prüfen lassen. Sie erhalten dann ein Feedback darüber, welche Tests erfolgreich absolviert wurden und welche fehlgeschlagen sind.

Diese Arbeit befasst sich mit dem Ziel, mehr Informationen über den abgegebenen Lösungsversuch und das Verhalten des darin enthaltenen Quelltextes präsentiert zu bekommen. Es soll ein Prototyp für ein System entwickelt werden, der es möglich macht, Lösungsversuche detailliert im Browser anzeigen lassen zu können. Dafür soll ein Kontrollflussgraph der abgegebenen Methode erstellt und angezeigt werden. Der Nutzer soll durch Bedienelemente schrittweise den Ablauf seines Lösungsversuchs für vorgegebene Parameter nachvollziehen können, indem er den Graphen durchläuft. Dabei sollen in jedem Schritt die Werte von Variablen und Inhalte aus Sammlungen dargestellt werden. (Anm.:Sammlungen meint hier, sowie in der kompletten folgenden Arbeit, die Collections Set, List, Map, sowie 1- und 2-dimensionale Arrays.)
Um gerade die Anfänger der Programmierung zu unterstützen, soll das Erstellen dieser Visualisierung keine Konfigurationen benötigen, wie zum Beispiel das Setzen von Breakpoints.

Das entwickelte System soll Anfängern der Programmierung eine Unterstützung darin bieten, Verständnis für Zusammenhänge der Programmierung aufzubauen und mögli-

cherweise Fehler in ihrem Lösungsansatz leicht zu identifizieren. Die dafür zu erfüllenden Ziele werden im Folgenden präsentiert.

Zielsetzung
Das entwickelte System soll folgende die im Folgenden aufgelisteten Merkmale haben:

- Es soll den Kontrollfluss eines untersuchten Lösungsversuchs visuell darstellen können
- Der Ablauf einer Ausführung des Versuchs soll schrittweise durchlaufbar sein
- Es soll erkennbar sein, welches der aktuelle Schritt ist
- In jedem Schritt soll die Belegung aller Variablen und Inhalte von Sammlungen, 1- und 2-dimensionalen Arrays angezeigt werden
- Die Bedeutung von Referenzen soll ersichtlich sein
- Ein Unterschied zwischen geordneten und ungeordneten Sammlungen soll ersichtlich sein
- Unterschiede zwischen den Sammlungen sollen deutlich werden
- Es soll keine weiteren Konfigurationen benötigen - insbesondere das Setzen von Breakpoints soll nicht benötigt werden

Abgrenzung
Das zu entwickelnde System soll nicht den kompletten Quellcode des Lösungsversuch darstellen, sondern nur die erste Methode, die darin enthalten ist. Methodenaufrufe innerhalb dieser Methode werden nicht im Detail, sondern nur als ein einziger Knoten des Graphen dargestellt. Weiter sollen Sammlungen nicht in ihrer konkreten Implementierung dargestellt werden, sondern nur als unterschiedliche Typen von Sammlungen. Eine ArrayList soll also aussehen wie eine LinkedList, aber anders als eine Set-Implementierung. Die dargestellten Variablen beschränken sich auf die 8 primitiven Datentypen in Jave (boolean, byte, short, char, int, float, double und long), auf deren Wrapper (Integer, Float, usw.) und auf Strings. Und das System soll nur ausführbaren Code verarbeiten. Fehlerhafter Code wird nicht untersucht.

Aufbau der Arbeit
Im anschließenden Kapitel wird auf den **Kontext** der Arbeit eingegangen. Darin werden Visualisierungen im Allgemeinen, Wahrnehmung der Nutzer und OPPSEE als Plattform, für die das System entwickelt wird, vorgestellt. Anschließend werden **Anforderungen**

herausgearbeitet, die es für das System zu erfüllen gilt. Dafür werden pädagogische Aspekte der Nutzung und solche, die ihren Ursprung in der Informationsvisualisierung haben, betrachtet. Außerdem werden Anforderungen an die Arbeitsweise des Programms benannt, die berücksichtigt werden müssen, die sich aus den anderen Anforderungen und Rahmenbedingungen ableiten. Es folgt ein **Entwurf** des Systems, in dem das angestrebte Design vorgestellt und anhand der Erkenntnisse und Anforderungen der vorherigen Kapitel begründet wird. Die **Umsetzung** geht auf interessante Aspekte der technischen Realisierung ein. Darauf folgt die **Validierung** anhand von aus Testdaten erzeugter Visualisierungen. Schließlich werden im Kapitel **Fazit und Ausblich** die Ergebnisse der Arbeit und Validierung betrachtet und mögliche Anwendungen und Erweiterungen des Systems vorgeschlagen.

2 Kontext

Dieses Kapitel liefert eine Einleitung in die Grundlagen von Visualisierung. Die Visualisierung, die für diese Arbeit von Bedeutung sind werden in ihren Grundzügen dargestellt. Die Informationsvisualisierung ermöglicht es den Nutzenden, Daten zu filtern und zu untersuchen. Die Softwarevisualisierung befasst sich mit der visuellen Darstellung von Software und teilt diese in drei Bereiche, die im Kapitel wiedergegeben werden. Anschließend beschäftigt sich das Kapitel mit dem Thema der menschlichen visuellen Wahrnehmung. Dies ist dahingehend relevant, als dass mittels zweier Theorien gezeigt wird, wie Daten angemessen und zielführend visualisiert werden können. Diese Theorien bearbeiten zum Einen das optische Hervorstechen von einzelnen Objekten, wenn sich diese aus einer Masse anderer Objekte hervorheben und zum Anderen die Zusammengehörigkeit von Objekten mittels vorher aufgestellter Regeln. Abschließend wird die Programmierübungsplattform OPSSEE der HAW vorgestellt. Auf dieser können Studierende der Informatik unterschiedliche Programmieraufgaben lösen und erhalten automatisiert Feedback, ob ihre Lösung den Anforderungen entspricht. Diese Informationen sind notwendig, da das in der Arbeit entwickelte System in die Plattform integriert werden soll, um die Lerneffekte der Studierenden zu verstärken.

2.1 Visualisierung

Visualisierung dient nicht nur in der Informatik dazu, Dinge besser zu verstehen und zu kommunizieren. Zum Beispiel werden in der Mathematik der Brüche als Tortendiagramme dargestellt, um den Lernenden in seiner Vorstellung von Brüchen zu unterstützen. Aber auch in der Informatik sind viele solcher Veranschaulichungen zu finden. Begriffe wie ”Tree”, ”Stack” oder ”Queue” sind keine zufällig gewählten Bezeichnungen für Datenstrukturen, sondern vermitteln eine Vorstellung davon, wie sie organisiert sind und funktionieren. Auch das Erstellen und Lesen von UML-Diagrammen ist ein bekannter

Vertreter von Visualisierung. Das Ziel von Visualisierungen ist es, für besseres Verständnis abstrakte Daten über eine visuelle Schnittstelle an den Nutzer zu leiten [10]. Dabei dient die Visualisierung als Erweiterung des Hirns. So kann sie beispielsweise als temporäre Speichererweiterung des menschlichen Verarbeitungsmechanismus betrachtet werden: [4] führt als Beispiel dafür das Multiplizieren zweier großer natürlicher Zahlen an. Nimmt man dafür Zettel und Stift zur Hilfe, dienen die aufgeschriebenen (/visualisierten) Zahlen als Gedächtnisstütze, während eine Reihe von kleineren Multiplikationsoperationen ausgeführt wird. Der Nutzen von Visualiserung liegt also im Allgemeinen darin, das Gehirn dabei zu unterstützen, Daten zu verstehen und zu vergleichen. Dieses geschieht durch eine unterstützte Mustererkennung, reduzierte Suchzeit für relevante Daten und das Nutzen von Wahrnehmungsmechanismen, um den Fokus zu stärken oder lenken.[4]

Durch die vielfältigen Einsatzgebiete haben sich viele unterschiedliche Kategorien von Visualisierungstechniken und -methoden entwickelt. Diese werden jedoch leider nicht klar definiert und verschiedene Autoren vertreten unterschiedliche Kategorisierungen [10]. Diese Arbeit macht sich Erkenntnisse der Informationsvisualisierung und Softwarevisualisierung zu Nutze und betrachtet die beiden wie folgt:

2.1.1 Informationsvisualisierung

Die Informationsvisualisierung ist der Datenvisualisierung sehr ähnlich. Wie einleitend beschrieben, werden diese mitunter gleichgesetzt oder überschneiden sich in ihren Themenbereichen je nach Autor. Ein vertretener Unterschied der beiden ist, dass Datenvisualisierung große Mengen von Daten unverarbeitet darstellt und Informationsvisualisierung diese Daten zunächst verarbeitet und filtert, um Zusammenhänge besser darzustellen. Daraus ergibt sich für die Datenvisualisierung eine gewisse Relevanz, die Erhebung der Daten zu optimieren, die für die Informationsvisualisierung keine große Rolle spielt. Informationsvisualisierung befassen sich dagegen mit kognitiven Aspekten der Nutzer.[13]

> ”The goal of information design must be to design displays so that visual queries are processed both rapidly and correctly for every important cognitive task the display is intended to support” -[23]

Dem Nutzer soll es möglich sein, die Daten nach eigenen Vorstellungen zu filtern und zu untersuchen, um Zusammenhänge zu bestätigen oder welche zu erkennen, die ihm vorher nicht bewusst waren.[4] Informationsvisualisierung bietet sich an,

- wenn dem Nutzer die Inhalte einer Datensammlung unbekannt sind
- der Nutzer nur ein eingeschränktes Verständnis davon hat, wie ein System organisiert ist
- der Nutzer eine leicht verständliche Darstellung der Daten bevorzugt
- oder der Nutzer Schwierigkeiten hat, konkrete Fragen zu formulieren, die er durch die Visualisierung beantworten möchte

hat der Nutzer dagegen sehr spezielle Fragen, ist ein Visualisierungssystem möglicherweise nicht die beste Wahl, da eine gezielte Suche nach Antworten häufig zielführender ist (vgl. [4])

Um diese Ziele zu erreichen, befasst sich die Informationsvisualisierung auch mit den Interaktionsmöglichkeiten, die die Visualisierung bieten sollte[10]. Interaktion mit der Visualisierung kann dazu dienen, die Limitierungen einer Representation zu überwinden und die Erkenntnis der Nutzer zu verstärken[4].[26]

Auf die kognitiven Aspekte der Nutzer wird im folgenden Kapitel 2.2 eingegangen. Interaktionsmöglichkeiten werden in Kapitel 3.1 analysiert und anschließend daraus Anforderungen an das zu entwickelnde System abgeleitet.

2.1.2 Softwarevisualisierung

> "The goal of software visualization is not to produce neat computer images, but computer images which evoke mental images for comprehending software better" - [2]

Während einige Autoren die wissenschaftliche Visualiserung, die Informationsvisualisierung und die Softwarevisualisierung als voneinander abgegrenzte Kategorien betrachten [10], wird die Softwarevisualisierung oft auch als Teilmenge der Informationsvisualisierung gesehen [2, S.3]. Visualisierung von Software lässt sich in drei Bereiche gliedern (vgl. [2]):

Struktur bezieht sich auf die statisch analysierbaren Teile der Software. Also diejenigen Bereiche, die visualisiert werden können, ohne die Software auszuführen. Bekannte Vertreter dieser Kategorie sind UML Klassen- oder Komponentendiagramme oder Kontrollflussgraphen.

Verhalten befasst sich mit dem Ablauf der Software mit konkreten Werten. Der Ablauf kann hierbei betrachtet werden als eine Folge von einzelnen Zuständen, die sowohl die aktuell ausgeführten Anweisungen als auch die Zustände von Variablen darstellen. Ein bekannter Vertreter ist z.B. das UML Use-Case-Diagramm, das einen bestimmten Ablauf des Programms darstellt.

Evolution betrachtet den Entwicklungsprozess des Softwaresystems. Hierbei werden zum Beispiel Änderungen am Quelltext oder die Entwicklung des Grades der Testabdeckung dargestellt. Bekannte Vertreter sind Versionskontrollsysteme, in denen geänderte Codezeilen dargestellt werden.

Diese Arbeit befasst sich sowohl mit der Struktur der Lösungsversuche, als auch mit deren Verhalten. Die Struktur wird durch den darzustellenden Kontrollflussgraphen dargestellt. Das Verhalten dadurch, dass der Graph für konkrete Werte durchlaufen wird und die Belegungen aller relevanter Variablen dabei ersichtlich sein soll. Für die dynamische Visualisierung (das Verhalten) werden idealerweise Daten- und Code-Visualisierung miteinander verknüpft, so dass der Nutzer sehen kann, wie die Ausführung einer bestimmten Anweisung den Speicher modifiziert [2, S.79]. Dieses soll erreicht werden, indem der als nächstes ausgeführte Schritt im Kontrollflussgraphen hervorgehoben wird, während die Belegung der genutzten Variablen angezeigt wird.

2.2 Wahrnehmung

Visualisierungen benötigen eine angemessene Darstellung der zu zeigenden Daten. Es soll sichergestellt sein, dass die Nutzer die Daten auch so interpretieren, wie vom Ersteller der Visualisierung vorgesehen. Dafür ist es nötig, sich näher mit der Wahrnehmung des Nutzers zu befassen. Über die Vermeidung von Mehrdeutigkeiten hinaus, lässt sich durch Nutzen der Eigenschaften der menschlichen Wahrnehmung die Aufmerksamkeit des Nutzers gezielt auf bestimmte Bereiche der Visualisierung lenken [21, S.99]. Weiter unterstützt eine geschickt gewählte visuelle Kodierung den Nutzer darin, die für ihn interessantesten Dinge unmittelbar zu erfassen, ohne das Bild ausführlicher untersuchen zu müssen [4][23]. Die Fähigkeit, zu verstehen welche optischen Abfragen am leichtesten ausgeführt werden, ist daher eine kritische Fähigkeit von Designern [23, S.21].

Die zwei wesentlichen psychologischen Theorien, die erläutern wie die Eigenschaften der menschlichen Wahrnehmung genutzt werden können, um Merkmale oder Muster effektiv

darzustellen, werden im Folgenden präsentiert. Die präattentive Wahrnehmung ist eine schnelle vorbewusste Wahrnehmung von Dingen, ”die einem ins Auge springen”. Und auf höherer kognitiver Ebene die Gestalttheorie, um das erfasste Bild in strukturierte Objekte zu ordnen.[21, S.97]

2.2.1 präattentive Wahrnehmung

Die präattentive Wahrnehmung (preattentive perception) beschreibt die Eigenschaft des menschlichen visuellen Systems, ein bestimmtes unter vielen Objekten wahrzunehmen, bevor die eigentliche visuelle Verarbeitung des Bilds beginnt, wenn sich dieses Objekt nur stark genug von den anderen abhebt.

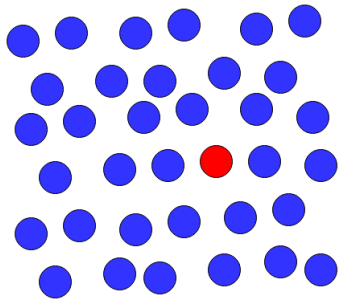


Abbildung 2.1: präattentiv wahrgenommener roter Kreis

Abb.2.1 zeigt einen roten Kreis, der sich von der Menge der übrigen blauen Kreise abhebt. Er wird präattentiv wahrgenommen. Anne Treisman untersuchte diese Eigenschaft durch Experimente, in denen die Probanden so schnell wie möglich erkennen sollten, ob ein einzigartiges Objekt unter vielen anderen, zueinander gleichen Objekten vorhanden ist. Die Anzahl der ablenkenden Objekte (Distraktoren) wurde bei weiteren Durchläufen stetig erhöht und die Auswirkung auf die Reaktionszeit untersucht. Änderte sich die Reaktionszeit bei steigender Anzahl von Distraktoren nicht, handelte es sich bei der Eigenschaft, in der sich das Zielobjekt von den anderen unterschied, um eine präattentive Eigenschaft.[20]

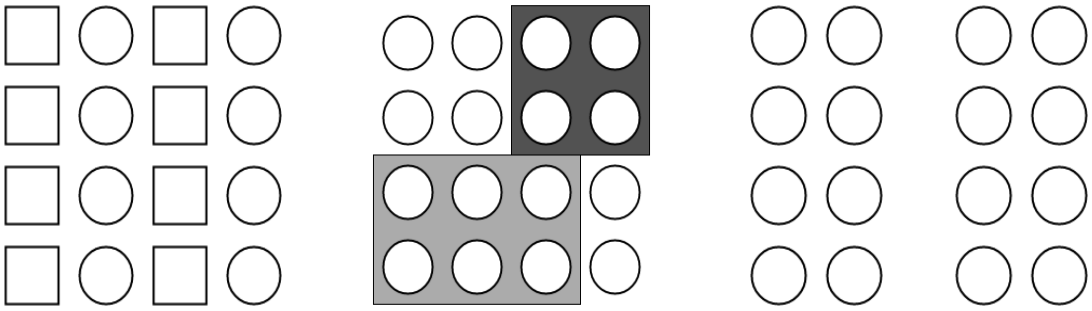
Distraktoren können sich auch untereinander unterscheiden, jedoch funktioniert die präattentive Wahrnehmung umso besser, je größer die Ähnlichkeit der Distraktoren zueinander ist.[21, S.106f] Die stärksten präattentiven Merkmale neben der in Abb.2.1 dargestellten Farbe sind Bewegung, Ausrichtung und Größe [25]. Der Bewegung kommt hierbei eine besondere Bedeutung zu, da sie im peripheren Sichtbereich besser wahrzunehmen

ist als alle anderen Eigenschaften ([2, S.19],[23, S.36],[21, S.124]). Eine Kodierung durch Bewegungen stört die anderen Kodierungen durch Farbe oder Form nicht [1].

Zuletzt sei erwähnt, dass präattentive Wahrnehmung nicht für jede Eigenschaft bidirektional ist. Im Falle von Farbe ist es möglich das Zielobjekt mit einer herausstechenden Farbe zu versehen, um bemerkt zu werden, oder alle Distraktoren mit dieser Farbe zu zeichnen, damit das Zielobjekt heraussticht. Dies ist nicht für alle Eigenschaften gegeben. Ein unterstrichendes Wort auf einer Textseite ist leichter zu bemerken als ein nicht-unterstrichendes Wort in einem Text aus unterstrichenden Wörtern [22, S.160f]

2.2.2 Gestalttheorie

Die ersten Untersuchungen, um Mustererkennung zu verstehen, wurden anfang des letzten Jahrhunderts von Max Westheimer, Wolfgang Köhler und Kurt Koffka unternommen [22, S.185]. Sie stellten Regeln auf, die zeigen welche Eigenschaften eine Rolle spielen, damit einzelne Elemente in der Wahrnehmung zu Feldern oder Strukturen organisiert werden. Abb.2.2 zeigt drei dieser Regeln, die zeigen wie durch sie ein Zusammengehören der einzelnen Objekte suggeriert wird. Bis heute ist die Zahl der Gestaltgesetze nach



(a) gleiche Form (b) gemeinsame Region (c) Nähe

Abbildung 2.2: Mustererkennung

einigen Zählungen auf bis zu 114 angestiegen, von denen viele sich auf visuelle Muster beziehen. Allerdings überschneiden sich diese Gesetze häufig, weswegen Designer und Künstler nur ein paar Ausgewählte davon berücksichtigen, um ihre Arbeiten aufzuwerten [5].

Die für diese Arbeit relevanten Gestaltgesetze sind:

Ähnlichkeit: Durch das gleiche Aussehen von Elementen nimmt der Betrachter an, die Elemente gehören physisch oder konzeptuell zueinander [5].

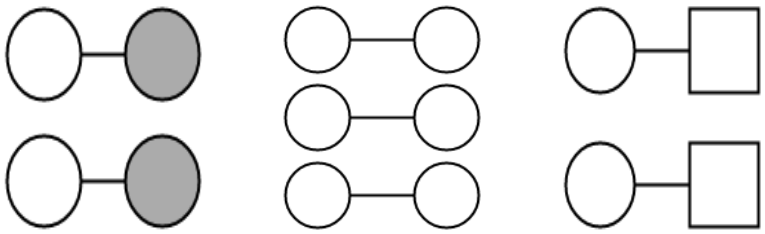
Nähe: Elemente, die näher aneinander stehen, werden als Gruppe wahrgenommen (siehe Abb.2.2c).

gemeinsame Region: Hierdurch wird eine Abgrenzung der Arbeitsbereiche und Trennen der zu visualisierenden Sammlungen voneinander möglich gemacht (siehe Abb.2.2b).

Verbindung: Verbindung stellt eine stärkere Verbindung dar als gemeinsame Farbe, Form oder Nähe zueinander dar [16]. Abb.2.3 zeigt auf, dass die Elemente eher als Verknüpfungen miteinander wahrgenommen werden als in der jeweils anderen Verbindung durch Farbe, Nähe oder Form.

Figur/Grund: Hiermit ist der Kontrast zwischen Vorder- und Hintergrund gemeint und die Auswirkung von Änderungen desselben. Ein Beispiel für interaktive Medien ist das Ziehen des Mauszeigers auf einen Link. Hierdurch ändert sich meist der Zeiger und die Art des Linktextes.[5].

Schwingungen: Bewegungen können genutzt werden, um Elemente, die synchron in die gleiche Richtung schwingen in visuelle Gruppen zu ordnen [21, S.123].



(a) Farbe (b) Nähe (c) Form

Abbildung 2.3: Verknüpfungen

2.3 OPPSEE

Das für diese Arbeit zu entwickelnde System hat zum Ziel, die Programmierübungsplattform ”Online Programming Practice for Software Engineering Education” (OPPSEE) im Department Informatik an der Hochschule für Angewandte Wissenschaften Hamburg (HAW-Hamburg) zu erweitern. Diese Plattform soll im folgenden kurz vorgestellt werden, um die Arbeit und Designentscheidungen des Systems besser nachvollziehen zu können.

OPPSEE ist eine Plattform, die durch Lernende komplett im Browser nutzbar ist. Die Plattform bietet Studenten die Möglichkeit, Programmieraufgaben unterschiedlicher Schwierigkeit zu lösen, automatisiert ein Feedback über die Korrektheit ihrer Lösung zu erhalten und dadurch ihre Programmierfähigkeiten zu trainieren.

Der Nutzer meldet sich an, wählt eine Aufgabe, an der er sich probieren möchte und kann direkt in einer Web-IDE (integrated development environment) loslegen, die die meisten Vorzüge anderer IDEs bietet, wie zum Beispiel Code-Completion oder Syntax-Highlighting. Die Hürde einzusteigen ist niedrig. Plattformbezogene Vorkenntnisse werden keine benötigt. Diese geringen Voraussetzungen werden in Kapitel3.2 das Nutzerbild und somit die Anforderungen an das zu entwickelnde System beeinflussen.

Hat der Nutzer seinen Lösungsversuch verfasst, kann er durch die Plattform testen lassen, ob sie die Aufgabenstellung erfüllt. Dafür wird die Lösung über ein API-Gateway an die Grader Pipeline gesendet. Die Grader Pipeline verwaltet untschiedliche Arten von Grading-Units, um zu prüfen, ob der Lösungsversuch den Anforderungen der Aufgabe genügt.[7] Ein Testbericht wird erzeugt und im Frontend als HTML-Dokument angezeigt.[7]

Das zu entwickelnde System soll als eine Grading-Unit konzipiert werden, die, wie die oben beschriebenen Grader, den Lösungsversuch als Quellcode entgegennimmt, verarbeitet und ein HTML-Dokument zurückliefert, das diesen Lösungsversuch angemessen visualisiert.

3 Anforderungsanalyse

Um das Tool zu nutzen, ist es nicht nur relevant, menschliche Wahrnehmung zu untersuchen, sondern auch die menschliche Interaktion mit Visualisierungstools näher zu betrachten. In diesem Kapitel sollen die Anforderungen an das zu entwickelnde Tool herausgearbeitet werden. Dafür werden zunächst Eigenschaften beleuchtet, die die Nutzung einer Visualisierung begünstigen. Danach werden pädagogische Hintergründe beleuchtet und mögliche Ansätze aufgezeigt, Visualisierungen pädagogisch wertvoller zu gestalten. Aufbauend auf diesen beiden Unterkapiteln und den vorherigen Kapitel werden in 3.3 schließlich Anforderungen an das zu entwickelnde System formuliert.

3.1 Interaktion mit Visualisierung

Wie bereits in Kap. 2.1.1 beschrieben wurde, spielt die Interaktion mit einer Visualisierung in der Informationsvisualisierung eine große Rolle. Es gibt zahlreiche Techniken, diese Interaktionen zu ermöglichen. Daher scheint es sinnvoller, nicht die speziellen Techniken, sondern das verfolgte Ziel zu betrachten.[26] Basierend auf diesem Ansatz werden in [26] folgende Ziele identifiziert:

select Ein bestimmtes oder mehrere Elemente auswählen, um sie zu markieren und nicht aus den Augen zu verlieren, wenn sich die Darstellung ändert.

explore Etwas anderes anzeigen lassen. Die häufigste explore-Methode, ist das Zoomen oder Navigieren in einem Graphen.

reconfigure Die Elemente anders anordnen, um die Darstellung den eigenen Vorlieben oder auch der eigenen Vorstellung von Strukturen der Daten anzupassen

encode Eine andere Darstellung der Elemente wählen.

abstract/elaborate Mehr oder weniger Details anzeigen. Dies kann zum Beispiel durch Öffnen oder Expandieren von Fenstern geschehen. Aber auch simple Tool-Tips, die mehr Informationen zeigen, wenn auf das Element gezeigt wird, fallen in diese Kategorie.

filter Elemente ausblenden. Dafür können Elemente gezielt ausgeblendet werden oder bestimmte Grenzen definiert werden, nach denen Elemente gefiltert werden.(vgl.[21, S.412])

connect Verknüpfungen zwischen Elementen sichtbar machen. Zum Beispiel können verknüpfte Elemente hervorgehoben werden, wenn mit der Maus auf ein Element gezeigt wird (vgl. [6]).

Diese Ziele dienen in erster Linie dazu, Möglichkeiten aufzuzeigen, eine Visualisierung durch Interaktivität anzureichern. Nur weil es möglich ist, ist es aber nicht auch sinnvoll, alle davon zu implementieren. Das Nutzerinterface sollte so gestaltet werden, dass es eindeutig ist und nur relevante Funktionen bietet[21, S.402]. Die Möglichkeiten sollten leicht erkennbar sein und keinen zusätzlichen kognitiven Aufwand benötigen.[12] Alle Möglichkeiten sollten für den Nutzer leicht zu erinnern sein. Zum Beispiel Mausbewegungen, um zu zoomen und sich im Graphen zu bewegen[21, S.411].

Um mehrdeutige Interpretationen des Interfaces zu vermeiden, hat es sich bewährt, unterstützende Informationen (zum Beispiel in Form eines Hilfe-Buttons) zur Verfügung zu stellen. Darin sollte erklärt sein wie man das System nutzt, wie welche Daten dargestellt werden und welche Bedeutung Symbole, Felder und Farben haben.[21, S.412f]

Graphen

Wie die übrige Visualisierung sollte auch ein Graph leicht lesbar sein. Dafür sollten sich im Graphen möglichst wenige Kanten überschneiden und wenig Knicke aufweisen[18]. Experimente dazu haben ergeben, dass bei simplen Graphen diese beiden Eigenschaften zwar keinen Einfluss auf die Fehlerrate der Interpretation von Graphen hatten, wohl aber auf die benötigte Zeit, den Graphen zu verstehen[18]. Weitere erstrebenswerte Eigenschaften zur schnelleren Erfassung des Graphen sind eine symmetrische Struktur[18] und möglichst kurze[21, S.324] und durchgängige Kanten (also ein 180 Grad Winkel zwischen zwei Kanten)[24]. Weiter ist es wichtig, die Label der Knoten so zu setzen, dass

sie den Graphen nicht überdecken und selbst lesbar bleiben[21, S.334]. Und den Graphen manipulier- und navigierbar zu machen[12].

3.2 pädagogischer Standpunkt

Um einen pädagogischen Nutzen aus der Visualisierung von Software zu ziehen, reicht es nicht, eine schöne Visualisierung zu erstellen. Wichtiger ist es, die Nutzer zu motivieren, sich mit der Visualisierung zu beschäftigen.[14]

> "what learners do, not what they see, may have the greatest impact on learning" - [14]

Eine Untersuchung von mehreren Algorithmus-Visualisierungen und deren pädagogischen Wert kam zu dem Ergebnis, dass von 21 Visualisierungen nur 13 einen signifikanten Einfluss auf die Lernergebnis hatten. Nach einer Sortierung nach interaktiven und nicht-interaktiven Visualisierungen, zeigte sich, dass 3 von 9 nicht-interaktive und 10 von 12 interaktive Visualisierungen diesen positiven Einfluss hatten. Daraus lässt sich schließen, dass Visualierungen von Algorithmen viel eher zur Entwicklung von Verständnis beitragen, wenn sie interaktiv gestaltet werden.[8] Interaktivität ist aber nicht das einzige Kriterium, das zu einer gelungenen Visualisierung beiträgt. Im Folgenden werden einige Ansätze vorgestellt, Visualisierungen pädagogisch wertvoller zu gestalten:

Kontrolle der Ausführung der Visualisierung Der Nutzer soll selbst entscheiden, wie schnell der Algorithmus durchlaufen wird und welche Punkte er näher analysieren möchte[14]. Diese kontrollierte Ausführung erhöht den pädagogischen Wert der Visualisierung signifikant[19]. Auch die Möglichkeit, die Darstellung von vorn zu starten, unterstützt das Verstehen, wohingegen die Möglichkeit, einzelne Schritte zurückzugehen, keinen Effekt zu haben scheint[19].

Bezug zum Code Obwohl das Anbieten von Pseudocode zum ausgeführten Algorithmus eine weit verbreitete Praxis darstellt[14], belegt [19], dass dadurch lediglich die mit der Visualisierung verbrachte Zeit erhöht wird, ohne einen nennenswerten Einfluss auf das Verständnis zu haben.

Fragen zum Algorithmus Damit die Nutzer besser reflektieren wie der dargestellte Algorithmus funktioniert, wird es als nützlich angesehen, Fragen zu dessen Wirkweise

zu stellen. Nutzer beschäftigen sich so stärker damit und stellen eigene Überlegungen an. [14]

Nutzerwissen berücksichtigen Programmieranfänger können schnell überfordert sein, wenn die Darstellung zu komplex wird. Im Gegensatz dazu wollen erfahrenere Nutzer eher Details der Ausführung betrachten.[14] Den Entwurf der Visualisierung an das Nutzerwissen anzupassen ist ein wichtiger Punkt beim Erstellen von Visualisierungen.[11]

Möglichkeit eigene Werte einzugeben Um den Nutzer zum Untersuchen des Algorithmus zu verleiten, scheint ein möglicher Ansatz zu sein, ihn eigene Werte ausprobieren zu lassen, um den Algorithmus auszuführen[14]. Diese naheliegende Vermutung ließ sich durch Experimente nicht bestätigen [19].

Musterlösung anbieten Einige Nutzer scheinen eher davon zu profitieren, einen vorgegebenen Lösungsweg nachzuvollziehen, als selbst die Lösung zu erarbeiten[11]. [19] stellt fest, dass es einen größeren Lerneffekt bietet, Musterlösungen anzubieten, als die Nutzer eigene Werte ausprobieren zu lassen.

Algorithmus selbst programmieren Nutzer einen Zielalgorithmus programmieren zu lassen, ist eine weitere Möglichkeit, die Nutzer dazu zu bringen, sich tiefer mit der Visualisierung zu befassen[14]. Der Nutzen ist stark vom Kenntnisstand der Nutzer abhängig. Anfänger könnten dadurch abgeschreckt werden, sich überhaupt damit zu befassen.

Die genannten Punkte stellen eine wertvolle Orientierung dar. Für die Entscheidung, welche davon für das zu entwickelnde System zu berücksichtigen sind, ist es nötig, die Nutzer und das Ziel des zu entwickelnden Systems zu untersuchen.[11]

3.3 Anforderungen

Aus den Erkenntnissen der vorherigen Kapitel werden im Folgenden die Anforderungen abgeleitet, die das System erfüllen soll. Die Zielgruppe der Visualisierung stellen Programmieranfänger dar, die wenig Erfahrung im Programmieren haben und ihre Fähigkeiten damit verbessern möchten. Die Visualisierung soll dabei unterstützen, Abläufe zu verstehen und Hintergrundwissen aufzubauen. Es soll nicht dazu dienen, sehr spezielle Fragen

zu beantworten. Anfänger sind leicht abgelenkt durch unnötige Schnörkeleien, Animationen oder zu tiefe Details.[11] Sie benötigen daher eine einfache Nutzeroberfläche mit eindeutigen Steuerungsmechanismen. Die Nutzung per Konsole oder eigene Einstellungen (wie das Setzen von Breakpoints) werden vermieden. Weiter sollte sich die Struktur der Visualisierung bei mehrfacher Nutzung nicht ändern, damit Nutzer sich schneller darin einfinden.[11].

Daraus ergeben sich folgende Anforderungen an die Nutzbarkeit:

- Der eigene Lösungsversuch wird als Kontrollflussgraph dargestellt und ist schrittweise ausführbar
- Simples Starten ohne weitere Einstellungsmöglichkeiten
- Die Visualisierung ist mit wenigen Knöpfen steuerbar
- Klickbare Elemente sind leicht erkennbar
- Gleichbleibende Struktur und Farbcode mit Anleitung zur Nutzung

Die Visualisierung soll dazu dienen, Informationen an den Nutzer zu übermitteln. Sie soll Eigenschaften und Ablauf des untersuchten Lösungsversuchs darstellen. Dabei sollen dem Nutzer möglichst diejenigen Informationen präsentiert werden, die für seinen Kenntnisstand am wertvollsten scheinen.

Folgende Anforderungen an die Darstellung werden festgelegt:

- Der Bezug von ausgeführter Codezeile und dargestellter Variablenbelegung ist ersichtlich
- Die Schritte eines For-Schleifen-Kopfes werden als einzelne Schritte dargestellt
- Ein Unterschied zwischen geordneten und ungeordneten Sammlungen wird deutlich - die konkrete Implementierung einer Sammlung bleibt verborgen
- Sammlungen werden auf unterschiedliche Weise dargestellt, die Darstellung einzelner Sammlung lässt sich ausblenden
- Ein Unterschied zwischen primitiven Datentypen und Referenzdatentypen wird dargestellt
- Referenzen auf die selben Daten sind erkennbar

- Der Graph ist leicht lesbar, manipulierbar und navigierbar

Das System wird als Grader-Unit für die OPPSEE-Plattform konzipiert (siehe 2.3). Außerdem muss es in der Lage sein, Visualisierungen zu Quellcode zu erstellen, ohne dass der Nutzer Einstellungen vornimmt.

Die Folgenden Anforderungen werden an das System gestellt:

- Wird nur mit dem Quellcode des Lösungsversuchs aufgerufen
- Erstellt zu visualisierende Rohdaten automatisch ohne weitere Einstellungen
- Erstellt einen Kontrollflussgraphen zum Code des Lösungsversuchs
- Erstellt einen Durchlauf durch diesen Graphen für konkrete Werte
- Speichert die Belegung aller Variablen in jedem Schritt des Kontrollflussgraphen

4 Entwurf

Die besten Visualisierung übermitteln nicht nur Informationen, sondern sind auch schön anzuschauen. Eine schöne Visualisierung lädt den Nutzer dazu ein, sich weiter damit zu befassen und die übermittelten Informationen tiefer zu untersuchen.[21, S.417] Eine unästhetische Visualisierung veranlasst ihn im schlimmsten Fall dazu, die Visualisierung wieder zu schließen und keine Informationen zu erhalten.

In diesem Kapitel soll das entworfene Design vorgestellt und anhand der Erkenntnisse vorheriger Kapitel begründet werden. Dafür wird zunächst die Aufteilung des Bildes vorgestellt. Anschließend werden die Darstellungen der Variablen und des Graphen präsentiert. Das letzte Unterkapitel beleuchtet die Daten, die für die vorgestellte Darstellung benötigt werden, um damit das folgende Kapitel der Umsetzung einzuleiten.

4.1 visuelles Design

[21, S.417] empfiehlt für ansprechende Visualisierungen den Fokus des Betrachters auf denjenigen Bereich zu lenken, der am wichtigsten für ihn ist. Untersuchungen der spontanen Aufmerksamkeit eines Nutzers haben ergeben, dass sich diese nicht gleichmäßig über den Bildschirm verteilt. Teilt man den Sichtbereich in vier Quadranten, erhält der Quadrant links oben mit 40% die größte Aufmerksamkeit, gefolgt vom Quadranten links unten mit 25%. Der Quadrant rechts unten erhält nur 15%.(vgl.[2, S.20])

Eine gute Visualisierung braucht eine klare Struktur mit wohldefinierten Funktionsbereichen. Die spontane Aufmerksamkeit der Nutzer gibt einen guten Rahmen vor, wie die Visualisierung zu entwerfen ist. Darzustellen sind der Kontrollflussgraph, die Belegung von Variablen, die Inhalte von Sammlungen und Bedienelemente. Die Bedienelemente benötigen die geringste Aufmerksamkeit des Nutzers. Sie sind zwar wesentlich für die Nutzung, sie ändern sich aber nicht und übermitteln keine eigenen Information. Sie erscheinen rechts unten in der Visualisierung.

Die übrigen drei Elemente werden in zwei thematische Gruppen geteilt: Den statischen Kontrollflussgraphen und die Darstellung aller sich ändernder Elemente. Da die dynamische Änderung der Variablen die höherwertige Information darstellt, werden die Darstellung der Variablen und der Sammlungen auf der linken Seite dargestellt und der Graph auf der rechten. Die Steuerleiste unter dem Kontrollflussgraphen zu positionieren, der durch sie durchlaufen wird, ist ein weiterer Punkt, der für diese Anordnung spricht. Die drei Bereiche werden optisch durch Trennlinien und leicht unterschiedliche Hintergrundfarben voneinander getrennt, um dem Nutzer die verschiedenen Funktionen zu signalisieren (siehe 2.2.2).

Abbildung 4.1: Bedienelemente zum Navigieren durch den Kontrollflussgraphen

Die Bedienelemente werden gemäß 3.3 möglichst simpel dargestellt. Sie bestehen aus vier Knöpfen zur Navigation und einem Hilfe-Knopf mit kleinem Abstand zu den anderen, der eine Anleitung der Visualisierung öffnet (vgl. Abb.4.1). Die Bezeichnung der Knöpfe ist denen von gängigen Mediensystemen ähnlich, um eine intuitive Nutzung zu gewährleisten. Trotzdem ist die Funktion zusätzlich in der Anleitungsseite erklärt. Wenn der Mauszeiger über einem der Knöpfe positioniert wird, ändert sich seine Darstellung und der entsprechende Knopf erhält einen farbigen Rand, um zu signalisieren, dass es sich um klickbare Elemente handelt (siehe 2.2.2).

4.1.1 Variablen und Sammlungen

Die Variablen und Sammlungen werden auf der linken Seite der Visualisierung dargestellt. Sie werden untereinander ohne feste Grenzen platziert, da der Platz, der für die Darstellung von beiden genutzt wird, variiert. So lässt sich der verfügbare Platz besser nutzen. Sammlungen werden in der Darstellung zwei mal berücksichtigt. Zum einen gibt es eine detaillierte Darstellung der Inhalte der Sammlung und zum anderen wird die Variable dargestellt, die auf die Sammlung verweist. Es ist also möglich, dass Variablen und keine Sammlungen in einer Visualisierung auftreten. Der umgekehrte Fall ist nicht möglich. Um eine eiheitliche Visualisierung zu erzeugen, in der die Nutzer Bereiche wiedererkennen, werden also die Variablen oben platziert und die Sammlungen darunter (siehe 3.3).

Variablen

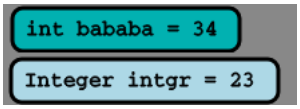


Abbildung 4.2: Darstellung von primitivem und Referenz-Typ

Der Unterschied zwischen primitiven Datentypen und Referenztypen wird durch eine farbliche Unterscheidung der beiden hervorgehoben. Die Form ist bei beiden identisch. Nach 2.2.2 sieht der Nutzer so, dass es zwei unterschiedliche Arten von Variablen gibt, aber trotzdem beides Variablen sind. Die Bezeichnungen in den visualisierten Variablen bestehen aus Variablentyp, Variablenname und dem Wert der Variable. Abb.4.2 zeigt die unterschiedliche Darstellung von primitiven Typen und Referenztypen.

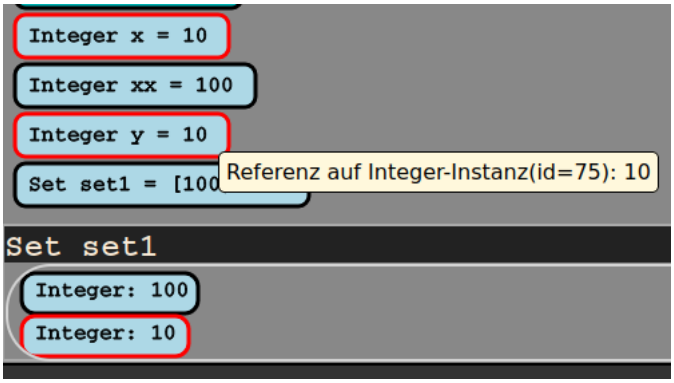


Abbildung 4.3: Highlighten von gleichen Referenzen

Zieht der Nutzer die Maus über ein Element, erscheint ein Tooltip, in dem steht, ob es sich um einen primitiven Typen oder eine Referenz handelt. Bei Referenzen wird zusätzlich die id aus dem Java-Debugger angegeben. Außerdem wird das Element selbst, sowie alle weiteren Elemente, die das gleiche Objekt referenzieren optisch hervorgehoben. Die Hervorhebung geschieht durch einen roten Rand. Dadurch stechen sie aus den anderen Elementen hervor und können präattentiv wahrgenommen werden, auch wenn viele Elemente gleichzeitig angezeigt werden. Siehe Abb.4.3.

Sammlungen

Die Sammlungen mit deren Inhalten werden immer unter den Variablen angezeigt. Sie alle bieten die Möglichkeit, sie für eine bessere Übersicht zu minimieren. Der Knopf dazu

befindet sich in ihrer Kopfzeile rechts oben und wird durch ein Minus-Zeichen dargestellt. Außerdem haben einige der Sammlungen einen zweiten Button, der ihre Darstellung in eine alternative Form ändert(siehe 3.3). Bei beiden dieser Buttons ändert sich die Form des Mauszeigers, wenn er sich darüber befindet. Außerdem haben sie die gleiche Farbe wie die Knöpfe zum Durchlaufen des Graphen.

Arrays werden maximal 2-dimensional berücksichtigt. Bei 2-dimensionalen Arrays werden Elemente des äußeren Arrays als Referenzen auf 1-dimensionale Arrays dargestellt. Die Elemente der inneren Arrays werden mit etwas Abstand jeweils in der gleichen Zeile wie das dazugehörige äußere Array positioniert. Bei Arrays lässt sich der Inhalt der Elemente der inneren Arrays anzeigen oder die Position. Abb.4.4 zeigt die zwei unterschiedlichen Ansichten. Auch der Unterschied zwischen den Referenzen auf innere Arrays und den primitiven int-Daten ist durch die Farbe zu sehen.

Abbildung 4.4: zwei Darstellungen von 2-dimensionalen int-Arrays

Listen Elemente aus Listen werden dargestellt wie die Variablen. Ihr Name setzt sich zusammen aus der Position des Elements in der Liste in eckigen Klammern, dem Datentyp und dem Wert. Zum Beispiel ”[0] Integer: 500”. Die Variablen stehen untereinander und es ist keine alternative Ansicht möglich.

Sets Wie die anderen Sammlungen bietet die Darstellung von Sets die Möglichkeit diese auszublenden. Weiter werden zwei Ansichten angeboten, die sich per Knopf umstellen lassen. In einer davon werden die Elemente wie in den Listen dargestellt, nur ohne die Position. In der anderen Ansicht werden die Werte der Elemente als aneinanderliegende Kreise dargestellt, um zu signalisieren, dass keine Ordnung vorherrscht (siehe 4.5). Ein weiterer Hinweis darauf, dass in der Sammlung keine Ordnung herrscht, ist durch die abgerundeten Kanten gegeben, die der Rahmen um die Elemente hat. Arrays und Listen haben diese nicht (vgl.Abb.4.4).

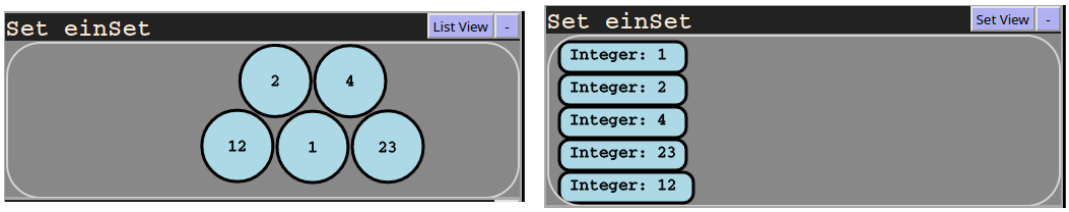


Abbildung 4.5: zwei Darstellungen von Integer-Sets

Maps Aus zeitlichen Gründen wurde eine Darstellung der Maps im prototypischen Entwurf des Systems nicht umgesetzt. Die Elemente der Maps sollen als unsortierte Kreise dargestellt werden wie in den Sets. Um die Zusammengehörigkeit von Keys und Values untereinander darzustellen, werden sie jeweils in einer Gruppe positioniert. Keys stehen auf der linken Seite und Values stehen rechts. Die Verknüpfung von Keys zu Values soll durch eine Verbindungslinie dargestellt werden (siehe 2.2.2). Die alternative Ansicht soll wie in den Sets die enthaltenen Elemente untereinander positionieren. Auch hier stehen die Keys wieder links, die Values rechts und eine Verbindungslinie zwischen den beiden stellt die dazugehörigen Beziehungen dar.

4.1.2 Kontrollflussgraph

Die Formen der Elemente des Kontrollflussgraphen sind durch die DIN66001 geregelt[2, S.40]. Beschriftet werden die Elemente durch den ihnen entsprechenden Quelltext-Teil der visualisierten Methode. Damit diese Beschriftungen keine anderen Teile verdecken und den Graphen schwerer lesbar machen, werden sie direkt in den Knoten dargestellt (vgl. 3.1). Der nächste ausgeführte Schritt des Graphen wird farblich hervorgehoben, indem er einen grünen Rand erhält.

Die Schleifenköpfe von for-Schleifen werden für besseres Verständnis als drei einzelne Elemente dargestellt: Initialisierung der Laufvariablen, Abbruchbedingung und Inkrementierung der Laufvariablen. Die Abbruchbedingung wird als Entscheidungsknoten als Raute dargestellt. Die Knoten hängen gemäß Kap.2.2.2 zusammen, um den Kopf als ein Ganzes darzustellen.

Die Kanten zwischen den Knoten werden als Linien dargestellt. Entscheidungsknoten (if, for-Abbruchbedingung und while) haben zwei Ausgangslinien, von denen diejenige, die bei einer Auswertung zu 'true' zutrifft, grün eingefärbt ist. Auf eine Beschriftung der Kante mit 'true' wurde bewusst zu Gunsten der Lesbarkeit verzichtet.

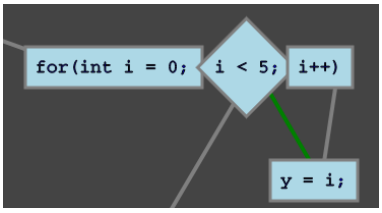


Abbildung 4.6: Darstellung einer for-Schleife

Abb.4.6 zeigt einen solchen for-Schleifenkopf. Die unterschiedlichen Pfade in den Schleifenrumpf und aus der Schleife heraus sind erkennbar. Der Nutzer kann sehen, dass der Schleifenrumpf nach dem Prüfen der Abbruchbedingung betreten wird. Weiter sieht der Nutzer, dass nach dem Durchlaufen des Schleifenrumpfes zunächst die Inkrementierung ausgeführt wird.

Der Graph ist dadurch navigierbar, dass ein Nutzer auf die Zeichenfläche des Graphen klickt, die Maustaste gedrückt hält und die Maus zieht. Gezoomt wird mittels Mausrad. Eine Nutzung die durch gängige interaktive Karten geläufig ist und dem Nutzer so leicht im Gedächtnis bleibt. Knoten lassen sich durch Anklicken und Ziehen bei gedrückter Maustaste in ihrer Anordnung verschieben. Diese Mechaniken sind auf der angebotenen Hilfeseite erklärt. Knoten richten sich neu aus, wenn ein Knoten gezogen wird. Dies soll dazu dienen, dass man den Graphen durch einen einzigen Knoten auseinanderziehen oder stauchen kann und nicht jeden Knoten einzeln positionieren muss. Ein vom Nutzer gezogener und gesetzter Knoten behält seine Position bei, auch wenn Nachbarknoten vom Nutzer neu positioniert werden. Durch einen Mausklick auf den Knoten löst man diesen und er positioniert sich wieder von allein. Der Unterschied zwischen festgesetzten und frei beweglichen Knoten wird durch einen Schatten unter dem jeweiligen Knoten dargestellt. Dieses soll ein intuitives Verständnis der Nutzer zu begünstigen, da Knoten mit Schatten zu 'schweben' scheinen.

4.2 Backend

Um diese Visualisierungen der Daten zu ermöglichen, müssen die Daten beschafft werden. Diese Beschaffung soll serverseitig durchgeführt werden (vgl. Implementierung als Grader-Unit 2.3). Die Visualisierung des Ablaufs als Kontrollflussgraph und die Visualisierung der Daten in jedem Schritt stellen zwei unterschiedliche Arten von Softwarevisualisierung dar: Zum einen gibt es die Visualisierung von Struktur und zum anderen die

von Verhalten (siehe 2.1.2). Daher werden auch die beiden Arten, diese Daten zu erlangen getrennt voneinander umgesetzt. Die Daten für den Graphen können durch statische Analyse des Quellcodes erhalten werden. Die Daten für die Variablen und Sammlungen werden durch eine dynamische Analyse gewonnen.

Aus dem visuellen Design (Kap. 4.1) ergibt sich, dass die **Variablen** Felder für ihren Namen, ihren Datentypen und ihren Wert besitzen müssen. Bei Referenzdatentypen muss zusätzlich gespeichert werden auf welches Objekt sie referenzieren. Außerdem bekommen Variablen ein Feld für ihre Kategorie, in dem gespeichert wird, ob sie einen primitiven Datentypen, Referenztypen oder eine Sammlung beschreiben.

Für die Darstellung der **Sammlungen** wird ihr Name benötigt und der Inhalt der Sammlung. Letzterer wird in einer Liste von Zeichenketten-Arrays gespeichert. Durch diese Speicherung wird ermöglicht, dass die unterschiedlichen Sammlungen mit unterschiedlich vielen Informationen darin festgehalten werden können.

Ein **Beispiel**: Ein 1-dimensionales 3-stelliges Array aus int-Werten benötigt zum Speichern lediglich die 3 ganzzahligen Werte des Arrays. Diese können in einem einzigen String-Array der Größe 3 gehalten werden.
Ein 1-dimensionales 3-stelliges Integer-Array benötigt 2 String-Arrays der Größe 3 zum speichern: Das erste für die Werte und das zweite für die Referenzen, die ja auch dargestellt werden sollen.
Ein 2-dimensionales int-Array benötigt nicht nur mehrere String-Arrays für die Werte der inneren Arrays, sondern zusätzlich dazu auch die Werte und Referenzen der äußeren Arrays. Siehe hierzu auch Abb. 4.4 des vorherigen Unterkapitels. Weiter können inneres und äußeres Array unterschiedlich groß sein. Dies führt dazu, dass auch die benötigten String-Arrays unterschiedlicher Größe sind.

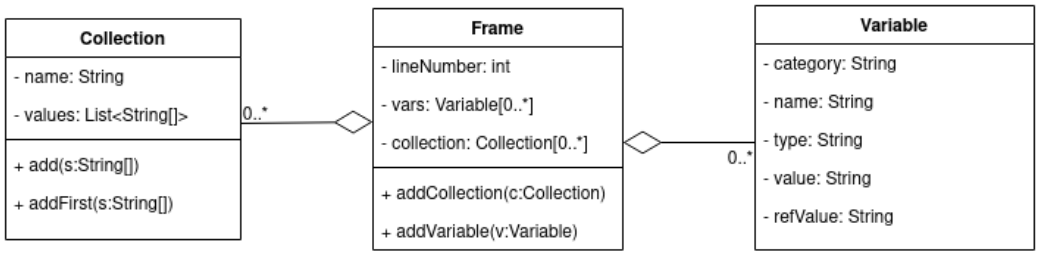


Abbildung 4.7: UML Klassendiagramm von Variablen und Sammlungen

Um die Variablen und Sammlungen in der Darstellung einem Knoten aus dem Graphen zuordnen zu können, werden sie in einem **Frame** gespeichert. Ein Frame beinhaltet alle

Variablen und Sammlungen, die sich vor Ausführung einer Codezeile im Speicher befinden. Die zugehörige Codezeile wird im Frame als lineNumber gespeichert (vgl.Abb. 4.7). Sie ermöglicht die Zuordnung zu einem Knoten des Graphen. Einer oder m ehrere Frames in geordneter Reihenfolge bilden den Programmablauf ab.

Um diese Daten zu erlangen, wird der zu visualisierende Code in der Java Virtual Machine(JVM) ausgeführt, darauf zugegriffen und die Werte werden ausgelesen (vgl.[2, S.82]). Dafür muss bekannt sein, in welchen Zeilen ausgelesen werden soll. Da die Nutzer selbst keine Einstellungen vornehmen müssen sollen, müssen diese Zeilen vor der Ausführung des Codes automatisch bestimmt werden. Diese Aufgabe übernimmt ein Parser, der gleichzeitig dazu dient, die statische Analyse durchzuführen, um die Daten für den Kontrollflussgraphen zu sammeln.

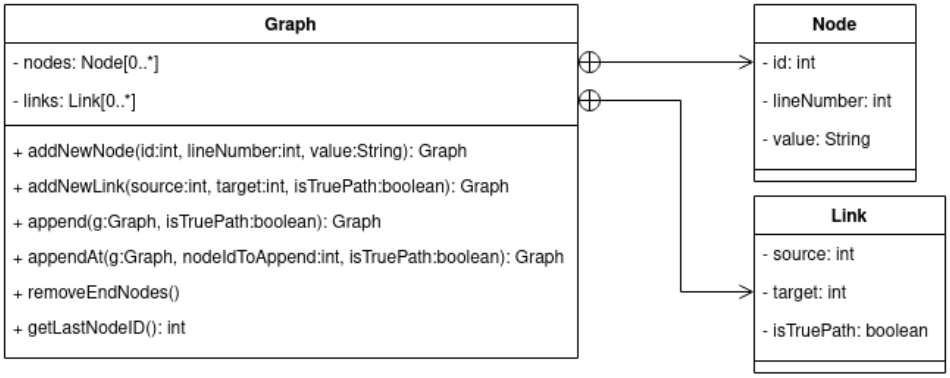


Abbildung 4.8: UML Klassendiagramm: Graph

Der vom Parser erstellte **Kontrollflussgraph** besteht aus Knoten und Kanten, die als innere Klassen des Graphen realisiert werden. Er bietet außerdem die Möglichkeit andere Graphen ans Ende anzuhängen oder an einer Zielposition einzufügen. Dadurch ist es möglich, einen Graphen aus mehreren Teilgraphen zu erstellen.

Ein **Knoten** des Graphen steht für eine Anweisung im untersuchten Lösungsversuch. Um die entsprechende Anweisung im Knoten anzuzeigen, muss diese als Zeichenkette im Knoten gespeichert werden. Außerdem wird die Zeile der Anweisung gespeichert, damit Frame und Knoten sich in der Visualisierung einander zuordnen lassen. Zuletzt benötigt ein Knoten noch eine eindeutige Kennzahl(ID), damit er Kanten zugeordnet werden kann.

Kanten speichern die ID der Knoten, die sie verbinden, als int-Werte 'source' und 'target'. Diese Bezeichnungen dienen lediglich der Verständlichkeit. Die Reihenfolge in der

zwei verbundene Knoten in einer Kante gespeichert werden, ändert an der Darstellung nichts. Die Eigenschaft, dass es sich um den Pfad eines zu 'true' ausgewerteten Entscheidungsknoten handelt, wird ebenfalls in einer Kante festgehalten. (Siehe Abb.4.8)

5 Umsetzung

In diesem Kapitel wird die Umsetzung des Entwurfs in einen lauffähigen Prototypen betrachtet. Die Reihenfolge der Unterkapitel richtet sich nach dem Ablauf des Systems. Zunächst wird der Parser betrachtet, der die Daten für den Kontrollflussgraphen und die Nutzung der zweiten Komponente erzeugt. Anschließend wird die dynamische Analyse betrachtet, die auf den in einer JVM laufenden Lösungsversuch zugreift und die benötigten Werte der Variablen ausliest. Zuletzt wird die Umsetzung der Darstellung dieser Daten in einem Browserfenster eingegangen.

Um das Ziel, den Kopf einer for-Schleife in 3 Komponenten (Initialisierung der Laufvariablen, Schleifenbedingung, In-/Dekrementation der Laufvariablen) zu teilen und auch so darzustellen, zu erreichen, muss die zu untersuchende Datei entsprechend angepasst werden, bevor die einzelnen Komponenten ihre jeweiligen Daten daraus gewinnen. Dazu wird die Datei zunächst mittels eines regulären Ausdrucks auf das Muster einer for-Schleife hin untersucht und nötige Zeilenumbrüche ergänzt. Anschließend wird die geänderte Datei gespeichert und kompiliert. Dieses ist notwendig, um beim Zugriff auf das in einer JVM laufende Programm die Schritte des for-Schleifenkopfes als drei einzelne Schritte betrachten zu können. Ohne diese Änderung könnte der komplette Schleifenkopf nur als Einzelschritt ausgewertet werden.

Die Breakpoints, an denen die Daten der JVM abgegriffen werden sollen, müssen entsprechend an diese Änderung der Datei angepasst werden. Daher ist es wichtig, dass die o.g. Änderung vollzogen wird, bevor der Parser die Datei scannt und die Breakpoints identifiziert, da ansonsten Quellcode des Parse-Objekts und kompilierte Java-Datei nicht übereinstimmen würden. Der Parser speichert auch den Namen der zu untersuchenden Methode, damit diese später gefunden werden kann.

5.1 Parser

Die Daten für den Kontrollflussgraphen werden erlangt, indem das Programm mit einem Parser untersucht wird. Ein Parser, der Java-Programme parsen kann, ist komplex. Daher wird ein Generator genuzt, um ihn zu erzeugen. Parsergeneratoren wie byacc und bison, die beide Fortführungen von yacc (yet another tool for language recognition) sind, und ANTLR (ANother Tool for Language Recognition) existieren seit mehreren Jahrzehnten und werden stetig weiterentwickelt und gepflegt. Die nutzende Gemeinde ist groß. Somit ist die Möglichkeit Hilfe bei Problemen zu erhalten gegeben und die Wahrscheinlichkeit, dass die Projekte eingestellt werden ist gering.
Ein Parsergenerator arbeitet nach einer ihm gegebenen Grammatik, in der die Regeln der Sprache festgehalten sind, für die er einen Parser erstellen soll. Für dieses System wird als Parsergenerator ANTLR gewählt, da hierfür eine große Sammlung von Grammatiken online verfügbar ist (siehe [3]), die mindestens im Falle der Java-Grammatik gut gepflegt wird.

ANTLR

Im Folgenden wird eine knappe Einführung zu ANTLR und zu Parsern im Allgemeinen gegeben, um die Erzeugung des Kontrollflussgraphen im nächsten Abschnitt besser nachvollziehen zu können. Für eine tiefere Auseinandersetzung mit dem Thema ANTLR und Parser sei an dieser Stelle [17] empfohlen.

Zum Parsen eines Textes oder Satzes gehen Parser in zwei Schritten vor. Diese sind mit dem Lesen eines Satzes eines Menschen vergleichbar. Der Mensch liest nicht Buchstaben für Buchstaben, sondern verbindet Buchstaben zu Wörtern. Anschließend verarbeitet er Wörter zu einem kompletten Satz. Das Gruppieren von Buchstaben zu Wörtern wird lexikalische Analyse genannt. Programme, die diese Aufgabe beim Parsen übernehmen, heißen daher 'Lexer'. Ein Lexer speichert diese Wörter als Token, die mindestens aus dem Tokentyp und der ursprünglichen Zeichenkette bestehen.[17]

Der zweite Teil des Vorgangs besteht darin, dass ein Parser den Strom der Token analysiert. Der Parser ignoriert dabei die ursprüngliche Folge von Buchstaben und verarbeitet nur die Typen der Token. Dies ist die syntaktische Analyse. Der Parser fasst mehrere Token zu einem Satz zusammen. Der Aufbau der Sätze ist dabei durch die Regeln der Grammatik definiert. Regeln bestehen aus Token, aus anderen Regeln oder beidem. Der

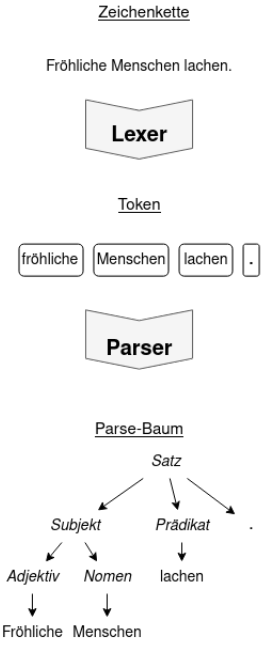


Abbildung 5.1: Lexer und Parser

deutsche Bespielsatz 'Fröhliche Menschen lachen.' besteht nach lexikalischer Analyse aus Adjektiv, Subjektiv, Prädikat und einem Punkt. Eine Regel, diesen zu lesen wäre, dass ein Satz aus der Regel für Subjekt, der Regel für Prädikat und dem Punkt-Token besteht. Die Regel für Subjekt bestünde aus Adjektiv-Regel und Nomen-Regel, die wiederum aus Token bestehen. Regeln sind meist nicht eindeutig, sondern bieten mehrere Möglichkeiten, damit verschiedene Sätze gelesen werden können. Die im Beispiel genannte Subjekt-Regel sollte als Alternative mindestens noch die Ableitung auf nur eine Nomen-Regel (ohne Adjektiv) anbieten.
Von ANTLR generierte Parser bilden die Token auf die ihnen bekannten Regeln ab und erstellen einen Syntaxbaum (auch Parsebaum)[17]. Die Blätter dieses Baums sind immer Token. Alle Blätter von links nach rechts aneinander gereiht ergeben wieder die ursprüngliche Zeichenfolge. Regeln sind Knoten in dem Baum. Siehe hierzu Abb. 5.1.

Um diese Parsebäume zu untersuchen, bieten ANTLR-generierte Parser zwei Möglichkeiten. Die erste Möglichkeit ist eine 'Listener-Klasse', die mit einer 'Walker-Klasse' zusammenarbeitet. Der Walker läuft den Parsebaum der Tiefe nach ab. Dabei löst er beim Passieren eines Knotens ein Ereignis im Listener aus. Der Listener kann also auf das Betreten oder das Verlassen eines Knoten reagieren. Abbildung 5.2 zeigt einen Parsebaum.

Der graue Pfeil ist der Pfad, den der Walker abläuft. Die Punkte auf dem Pfad sind die Stellen, an der der Listener reagieren kann.

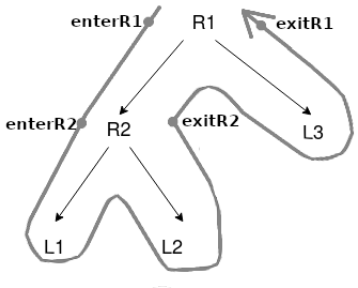


Abbildung 5.2: Parse-Tree Walker und Listener

Die zweite Möglichkeit ist eine 'Vistitor-Klasse'. Visitors brauchen keinen Walker und lassen den Nutzer des Parsers entscheiden, welche Knoten besucht werden sollen. Eine von ANTLR generierte Visitor-Klasse startet standardmäßig im Wurzelknoten und besucht alle Knoten.
Um die Daten für den Graphen zu erzeugen, wird eine Visitor-Klasse vom default-Visitor abgeleitet und die entsprechenden Methoden werden angepasst.

Graphen erzeugen

Die Daten für den Kontrollflussgraphen werden ähnlich wie in [27] aus einem Syntaxbaum erzeugt, indem eine visit-Methode sich, von der Wurzel ausgehend, rekursiv für die Kindknoten aufruft bis ein Blatt erreicht ist. Der Visitor erzeugt zu Beginn ein leeres Graph-Objekt, an das er das Ergebnis des nächsten Knotenbesuchs anhängt. Jeder Besuch eines Knoten gibt einen Teilgraphen zurück.

Der Quelltext des zu untersuchenden Lösungsversuchs befindet sich in ursprünglicher Reihenfolge in den Blättern des Baumes. Im zu erstellenden Kontrollflussgraphen hängen die Knoten von aufeinanderfolgenden Anweisungen des Quelltextes in Reihe zusammen. Um diese Teile des Graphen zu erstellen, können also einfach die Blätter des Baumes als Knoten aneinandergereiht werden. Der implementierte Visitor besucht dafür aber nicht erst die Blätter, sondern bricht seinen Teilpfad schon in der Regel für Anweisungen ab. Erreicht er einen Knoten, der der 'Anweisung-Regel' entspricht, liest er die Zeichen des dazugehörigen Quelltextes und speichert diese als 'value' des Knoten. Außerdem wird eine ID zugewiesen, die sich bei jedem Knoten erhöht. Die Zeilennummer des zum Knoten

gehörenden Quelltextes wird zum einen im Knoten-Objekt gespeichert, zum anderen auch in einer Menge von Breakpoints, die für die spätere Analyse benötigt werden. Der Knoten wird als Graph (mit nur einem Knoten) zurückgegeben und an den bisherigen Graphen angehängt.

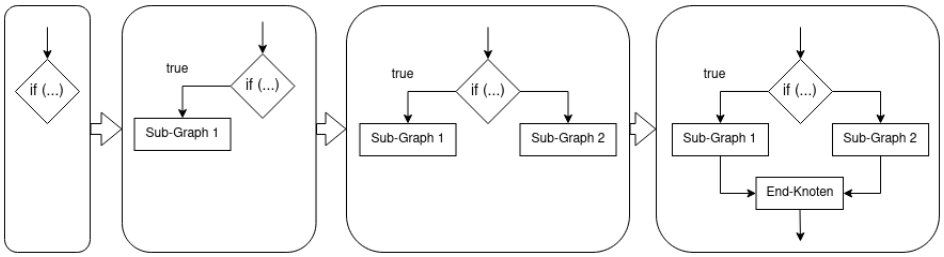


Abbildung 5.3: Entstehung von Kontrollflussgraph-Baustein: if-else

Der Visitor kann aber auch auf Knoten im Parsebaum stoßen, die einer Verzweigung im Kontrollflussgraphen entsprechen, also zum Beispiel einem if-else-Knoten. Dann wird auch ein neuer Knoten für den Kontrollflussgraphen erstellt und ein Breakpoint gespeichert. Weiter wird die ID dieses 'if-else-Graphen-Startknotens' temporär festgehalten und die visit-Methode für den if-Rumpf auferufen. Der von diesem Methodenaufruf erhaltene Graph wird an den bisherigen Graphen (bestehend aus einem if-Knoten) angehängt. Anschließend wird die visit-Methode für den else-Rumpf aufgerufen. Der von ihr zurückgegebene Graph wird mittels zu Beginn gemerkter Startknoten-ID auch an den if-Knoten gehängt. Dann wird ein Endknoten erstellt und an den Graphen angehängt. Dieser Knoten dient der Strukturierung des Graphen. Er wird später wieder entfernt werden und bekommt, um wiedererkannt zu werden, die Zeilennummer -1 zugewiesen. Zuletzt wird der letzte Knoten des else-Rumpfes noch mit dem End-Knoten verbunden. Der so erhaltene Graph hat eine Eingangskante und eine Ausgangskante und lässt sich wie ein Knoten in einen Graphen einfügen. (Vgl. Abb. 5.3)
Die Teilgraphen aus denen der vorgestellte Graph-Baustein besteht, können wiederum selbst aus ähnlichen Teilgraphen bestehen. So ist es möglich, geschachtelte Ausdrücke zu parsen, ohne die Knoten im Blick behalten zu müssen, die gemeinsam auf einen Folgeknoten zeigen.

Der Visitor beginnt die Erstellung des Graphen, im Klassenrumpf. An dieser Stelle wird ein Start-Knoten erzeugt, an den der restliche Baum angehängt wird. Dann wird die visit-Methode nur für die erste Methodendeklaration aufgerufen, um auch nur diese zu untersuchen und die anderen Methoden zu ignorieren. Ist die Methode besucht und der Kontrollflussgraph erstellt, wird ein Ende-Knoten ergänzt. Start- und Ende-Knoten die-

nen dazu, in der Darstellung an festen Positionen verankert zu werden, um dem Graphen immer eine ähnliche Form zu geben.

5.2 Debugger

Die nächste Komponente ist die Debugger-Komponente, die den zu untersuchenden Lösungsversuch in einer JVM ausführt und dann auf diesen zugreift, um die benötigten Daten auszulesen. Dafür wird das Java Debug Interface (JDI) verwendet. Das JDI ist Teil der Java Platform Debug Architecture (JPDA). Die JPDA besteht aus drei Ebenen[15]:

JVM TI Das Java Virutal Machine Tool Interface ist ein Interface, das von einer virtuellen Maschine implementiert wird. Es definiert die Dienste, die eine virtuelle Maschine für das Debugging anbieten muss. Dieses umfasst zum Beispiel das Setzen von Breakpoints oder das Benachrichtigen, wenn ein Breakpoint erreicht wurde. Weiter auch das Bereitstellen von Informationen wie zum Beispiel diese zu Variablen im aktuellen Frame.[15]

JDWP Das Java Debug Wire Protocol definiert die Kommunikation zwischen der virtuellen Maschine, die das JVM TI implementiert, und dem Prozess, der darauf zugreift.[15]

JDI Das Java Debug Interface ist eine Schnittstelle in Java, um mittels JDWP auf die virtuelle Maschine zuzugreifen, die das JVM TI implementiert.[15]

Um in einer JVM zu laufen, benötigt das dazugehörige Programm eine main-Methode. Bei den zu untersuchenden Lösungsversuchen handelt es sich um Lösungsversuche zu Aufgaben, in denen kleinere Hilfsmethoden entwickelt werden sollen. Diese sind selbst nicht eigenständig ausführbar, haben also keine eigene main-Methode.
Um diesem Problem gerecht zu werden, nutzt das System eine Klasse, die eine main-Methode hat, ein Objekt der zu untersuchenden Klasse erzeugt und dann die gewünschte Methode darauf aufruft. Die main-Methode dieser 'Wrapper-Klasse' wird in der JVM ausgeführt. Sie erhält zwei Argumente: den Klassennamen des Lösungsversuchs und den Methodennamen. Den Klassennamen benötigt sie, um zu wissen, von welcher Klasse sie ein Objekt erzeugt. Von diesem Objekt ist aber nur eine ungeordnete Menge aller darin enthaltenen Methoden verfügbar. Daher benötigt der Wrapper auch den Methodennamen als Argument. Im derzeitigen System wird die Methode ohne Parameter aufgerufen. Aufrufe mit variablen Argumenten könnten realisiert werden, indem man diese auch als

Argumente an den Wrapper übergibt und der Wrapper diese dann zum Starten der Methode nutzt. Eine weitere Alternative wäre es, nur bestimmte Variablen zuzulassen, die vom Ersteller der Aufgabe festgelegt werden (siehe 3.2 Musterlösung anbieten). Diese könnte man in einer Datei halten und vom Wrapper je nach Aufgabe auslesen lassen.

Mittels JDI wird nun die Wrapper-Klasse mit den Argumenten, die der Parser zuvor ermittelt hat, in einer JVM gestartet. Die ebenfalls vom Parser ermittelten Breakpoints werden gesetzt und die zu untersuchende Methode ausgeführt. Bei jedem Erreichen eines Breakpoints werden die Daten aller Variablen ausgelesen und gemeinsam mit der Zeilennummer in einem Frame gespeichert (Aufbau des Frames: siehe 4.7). Die Frames werden in Reihenfolge ihrer Erstellung in einer Liste gespeichert. Diese Liste stellt den Ablauf der kompletten Methode dar. Werden Zeilen wiederholt ausgeführt, gibt es auch mehrere Einträge mit gleicher Reihenfolge. Knoten im Kontrollflussgraphen können also mehrfach besucht werden.

5.3 Frontend

Das Frontend ist in HTML, CSS und JavaScript entwickelt worden. Die in Kap.4.1 vorgestellte Strukturierung wird durch die HTML-Seite mit CSS vorgegeben. Die sich ändernden Darstellungen von Graph und Variablen-Belegungen werden durch JavaScript ermöglicht. Dieser Teil des Systems ist immer gleich. Die Darstellung ändert sich nur durch die im Backend gewonnenen Daten. Ein als Grader-Unit konzipiertes System braucht also auch nur die gewonnenen Daten an das Frontend zu schicken. Die statischen Teile der Darstellung könnten durch andere Services bereitgestellt werden. Zum Verschicken werden die Daten von Graph und Frames der vorherigen Komponenten in das JSON-Format gewandelt.

Das Darstellen der Frames wird über die Funktionen der Knöpfe gesteuert. Beim Aufrufen der Seite wird die Methode des 'An-den-Anfang'-Knopfes ausgeführt. Es wird ein Schrittzähler auf 0 gesetzt. Der Schrittzähler wird durch das Bedienen der Knöpfe verändert. Dargestellt werden immer die Daten an derjenigen Stelle der Frame-Liste, die dem Schrittzähler entspricht. Mit der Zeilennummer, die in diesem Frame hinterlegt ist, wird eine Methode des Graphen aufgerufen, die den Knoten hervorhebt, der dieser Zeilennummer entspricht. Durch die Funktionen der Knöpfe werden der aktuelle Schritt und die Frame-Liste verwaltet und das Darstellen der Daten und Hervorheben des Graphenknotens angestoßen.

Das Darstellen der Frames geschieht dadurch, dass durch die Methode eines Knopfes eine Update-Methode mit dem aktuellen Frame aufgerufen wird. Für jede Variable wird ein Rechteck als SVG-Element in dem HTML-Dokument erzeugt. An dieses Element wird die ID, die der Debugger verwendet, um das Objekt zu referenzieren (siehe Tooltip in Abb.4.3), angehängt. Sie wird genutzt, um Referenzen auf gleiche Objekte zu finden und hervorzuheben, wenn der Nutzer die Maus über eines der Objekte zieht. Die Elemente, die in den Sammlungen dargestellt werden, erhalten dieses Attribut ebenfalls.

Wenn sich in der Collection-Sammlung des darzustellenden Frames Collections befinden, werden auch diese dargestellt. Dafür wird der Name der Collection unter den Namen der Variablen des Frames gesucht, um seine Kategorie zu bestimmen. Für die unterschiedlichen Sammlungen und deren Darstellung werden unterschiedliche Klassen verwendet, die abhängig von der Kategorie gewählt werden.

Der Graph wird beim Aufrufen der Seite erstellt. Um die Knoten auszurichten wird die Force-Simulation von D3JS (Data Driven Documents JS) genutzt[9]. Dadurch wirken definierbare Kräfte auf die Knoten, durch die sie in der Zeichenfläche ausgerichtet werden. Alle Knoten stoßen sich hierbei voneinander ab. Durch Kanten verbundene Knoten ziehen sich zusätzlich an. Alle Knoten werden leicht in das Zentrum gezogen. Die Stärke der Kräfte kann festgelegt werden oder zusätzliche Kräfte ergänzt werden.[21][S.327] Die Position von Start- und End-Knoten sind festgelegt und immer gleich. Dadurch richtet sich der Graph bei jeder Nutzung ähnlich aus. Diese Umsetzung führt nicht immer zu optimalen Ergebnissen, bietet aber einen guten Start für die Darstellung [12].

6 Validierung

In diesem Kapitel soll das System, mit dessen Entwicklung sich diese Arbeit befasst hat, auf seine Brauchbarkeit hin überprüft werden. Zunächst wird dazu betrachtet, ob die in der Zielsetzung aus Kapitel 1 festgelegten Ziele erreicht wurden. Anschließend wird das entwickelte System genutzt, um anonymisierte, fehlerhafte Lösungsversuche von Klausuraufgaben zu visualisieren. Hierbei soll geprüft werden, ob durch die Visualisierung ein Fehler hätte bemerkt werden können. Screenshots dieser Visualisierungen befinden sich im Anhang.

Vergleich mit Zielsetzung:
Die in der Einleitung vorgestellten Ziele wurden nahezu alle im System umgesetzt. Es analysiert Quelltext ohne das Setzen von Breakpoints. Es liefert eine Darstellung des Kontrollflusses und lässt den Nutzer diesen für eine konkrete Ausführung des Codes schrittweise durchlaufen. Die Belegung aller beteiligten Variablen und Inhalte von Sammlungen werden für jeden Schritt angezeigt. Referenzen werden durch Hervorheben von Variablen, die auf gleiche Objekte referenzieren, deutlich gemacht und ein Unterschied zwischen geordneten und ungeordneten Sammlungen ist erkennbar. Die Unterschiede zwischen den Sammlungen wurden visuell berücksichtigt. Einzig die Darstellung von Maps wurde im System aus zeitlichen Gründen nicht so umgesetzt wie es im Entwurf vorgesehen war.

Testen der Anwendung
Für die folgenden Untersuchungen wurden anonymisierte Lösungsversuche zu Klausuraufgaben von Informatik-Studenten des ersten Semesters genutzt. Diese Lösungsversuche wurden nach fehlerhaften Versuchen gefiltert und deren Ablauf anschließend durch das entwickelte System visualisiert. Drei dieser Visualisierungen werden im Folgenden vorgestellt. Es sei angemerkt, dass es sich bei den drei gezeigten Versuchen um eine Auswahl handelt, die nicht die Gesamtheit der Fehlversuche widerspiegelt. Weiter soll nicht geprüft werden, ob die Visualisierung ausreicht, um zu einer Lösung zu führen. Viel mehr soll untersucht werden, ob durch die Visualisierung Denk- oder Programmierfehler auf-

gedeckt werden können, wodurch der Nutzer ein besseres Verständnis seines Versuchs erlangen kann.

Visualisierung 1

Aufgabenstellung: Implementiere die Methode extrahiereUnikate.
Die Methode bekommt eine Liste mit Woertern und liefert die Unikate darin, also die Woerter, die nur genau einmal in der Liste vorkommen.
Beispiel: [”a”, ”b”, ”c”, ”d”, ”c”, ”b”, ”a”] liefert [”d”]
Beispiel: [”a”, ”b”, ”b”, ”c”] liefert [”a”, ”c”] Die Reihenfolge in der Ergebnisliste muss nicht gleich der Reihenfolge der Unikate in der Parameterliste sein.

Quelltext des Lösungsversuchs

```
public List<String> extrahiereUnikate(List<String> woerter){
    List <String> erg = new ArrayList<String>();
    boolean x = true;
    for(String wort: woerter){
        if(woerter.size() > 1){
            for(int i = 0; i < woerter.size(); i++){
                woerter.remove(0);
                i--;
                if(woerter.contains(wort)){
                    x = false;
                }
            }
        }
        if(x){
            erg.add(wort);
        }
        x = true;
    }
    return erg;
}
```

Beobachtungen:

Beim Durchlaufen des Graphen in der Visualisierung ist zu erkennen, dass die meisten der Knoten kaum besucht werden. Der Großteil des Ablaufs findet in der inneren for-Schleife statt (in der Abb.A.1 rechts oben). Hier scheint das Programm 'festzustecken' und bei sich wechselndem Zähler i die ursprüngliche Liste Liste zu leeren, ohne etwas anderes zu tun.
Dies ist ein Verhalten, das der Ersteller des Lösungsversuchs wahrscheinlich nicht erreichen wollte. Es wird aber sehr anschaulich durch den Ablauf des Versuchs im Kontrollflussgraphen mit gleichzeitiger Darstellung der sich leerenden Liste und wechselndem i veranschaulicht.

Weiter lässt sich feststellen, dass die Darstellung der for-Schleifenköpfe eher hinderlich für die Nutzung ist. Diese scheinen zu groß und 'unhandlich', um sich die Graphenknoten so anzuordnen, wie man es möchte.

Visualisierung 2

Aufgabenstellung: Implementiere die Methode laengsterGemeinsamerString.
Die Methode liefert von zwei Strings den laengsten gemeinsamen Teilstring.
Wenn es mehrere laengste gemeinsame Teilstrings gibt, ist es egal, welcher von diesen geliefert wird. Gibt es keinen gemeinsamen Teilstring mit mindestens Laenge 1, wird der leere String geliefert.

Quelltext des Lösungsversuchs

```
public String laengsterGemeinsamerString(String s1,
 String s2){
    String ergebnis = "";
    for(int i = s2.length() ; i < s1.length();++i){
        if(s1.charAt(i) == s2.charAt(i)){
            ergebnis = ergebnis + s1.charAt(i);
        } else {
            ergebnis = ergebnis + "";
        }
```

```
        }
        return ergebnis;
    }
```

Beobachtungen:

Beim Durchlaufen der Knoten des Kontrollflussgraphen fällt direkt auf, dass die Knoten innerhalb der for-Schleife nicht besucht werden. Die Darstellung der Variablenbelegung zeigt zusätzlich auf, dass der Schleifenzähler i mit 5 initialisiert wird. Es ist davon auszugehen, dass dem Nutzer durch die Kombination dieser Informationen aufgefallen wäre, dass seine Schleife sich nicht so verhält wie er es vorgesehen hatte.

Auch in der Visualisierung dieses Lösungsversuchs fällt es auf, dass die Bedienbarkeit unter der besonderen Darstellung der for-Schleife leidet. Weiter fällt auf, dass das Fehlen der Pfeilspitzen das Verständnis erschwert. Insbesondere am if-Knoten, an dem zwei grüne Kanten anliegen, wird dieses deutlich.

Visualisierung 3

Aufgabenstellung: Implementiere die Methode enthaeltZeichenDoppelt.
Die Methode gibt an, ob in einer Zeichenkette mindestens ein Zeichen doppelt vorkommt.

Quelltext des Lösungsversuchs

```
public boolean enthaeltZeichenDoppelt(String s){
    boolean erg = false;
    for ( int i = 0; i <s.length(); i++) {
        char c = s.charAt (i);
        for ( int k = 0; k < s.length(); k++) {
            if ( c == s.charAt(k)) {
                erg = true ;
            } else  {
                erg = false;
```

```
                }
            }
        }
        return erg ;
    }
```

Beobachtungen:

Beim Durchlaufen des Kontrollflussgraphen fällt es auf, dass die beiden Schleifen sehr häufig durchlaufen werden. Außerdem ändert sich der Wert der Ergebnis-Variable bei jedem Durchlauf der inneren Schleife mehrfach. Bei der Frage, ob etwas doppelt vorkommt oder nicht, ist es auffällig, wenn die vermeintliche Antwort sich ständig ändert. Es ist davon auszugehen, dass dieses Verhalten nicht mit der vom Nutzer entwickelten Idee einer Lösung übereinstimmt und er daher durch die Veranschaulichung wertvolle Hinweise auf mögliche Änderungen erhalten hätte.

Auch bei dieser Visualisierung fällt auf, dass die Bedienbarkeit unter zu unhandlichen for-Schleifen leidet und dass bei geschachtelten Verzweigungen die fehlenden Pfeilspitzen der Kanten ein schnelles Erfassen des Ablaufs erschweren.

Zusammenfassung

Die betrachteten Visualisierungen schufen alle drei eine brauchbare Möglichkeit, ihren Nutzern bestimmte Fehler in ihren Lösungsversuchen aufzuzeigen. Es hat sich gezeigt, dass sowohl die Darstellung des Ablaufs im Kontrollflussgraphen als auch die der Variablenbelegung und Sammlungsinhalte wertvolle Hinweise auf die Wirkweise der Untersuchten Methoden liefern.

Die betrachteten Visualisierungen zeigten jedoch auch auf, dass das Design im Hinblick auf die Nutzbarkeit noch einiger Anpassungen bedarf. Die Darstellung der for-Schleifenköpfe erwies sich eher als hinderlich, da der Zusammenschluss aus 3 Knoten die Bedienung erschwert. Außerdem erschwert das Fehlen der Pfeilspitzen an den Kanten des Graphen das Verständnis. Dies gilt insbesondere dann, wenn geschachtelte Bedingungen betrachtet werden.

7 Fazit und Ausblick

Im Verlauf dieser Arbeit wurde ein System prototypisch entwickelt, das einzelne Methoden visuell darstellen kann. Dafür wurden zunächst Aspekte der menschlichen Wahrnehmung und Pädagogik herausgearbeitet und anschließend ein Entwurf entstellt, der vielen dieser Aspekte gerecht wird. Es folgte eine Umsetzung des Entwurfs mit prototypischer Entwicklung des Systems. Abschließend wurden Tests mit realen Daten durchgeführt, um Rückschlüsse auf die Brauchbarkeit des Systems zu ziehen.

Die Tests lassen vermuten, dass durch das entwickelte System einige Fehler in Lösungsversuchen aufgedeckt werden können und Nutzer dadurch ein besseres Verständnis von ihren entwickelten Lösungsversuchen und deren Verhalten erlangen können. Die in den Versuchen in Kap. 6 aufgezeigten Fehler hätten zwar alle auch mit Hilfe eines Debuggers entdeckt werden können, dieses Argument lässt aber außer acht, dass es sich bei der Zielgruppe um Programmieranfänger handelt, die oft noch keine Erfahrungen mit Debuggern gesammelt haben.

Die Nützlichkeit des entwickelten Systems lässt sich also durch mehrere Punkte begründen. Das Nutzen ist für Programmieranfänger ohne Schwierigkeiten möglich. Sie müssen sich nicht um Einstellungen kümmern und bedienen die Visualisierung mit vier simplen Knöpfen. Es ist sehr leicht verständlich, wodurch eine Einarbeitungszeit entfällt. Weiter haben die Tests gezeigt, dass die Visualisierung in einigen Fällen wertvolle Hinweise auf Fehler des Lösungsversuchs geben kann. Dafür waren sowohl der angezeigte Kontrollfluss als auch die Belegung von Variablen sehr hilfreich.

Einzig die detaillierte Darstellung der for-Schleifen scheint nicht gut gewählt, da sie die Benutzung erschwert. Der Vorteil der zusätzlichen Erkenntnis, die Programmieranfänger daraus ziehen können, nämlich zu sehen, wann der Zähler erhöht und die Bedingung für einen weiteren Durchlauf geprüft wird, scheint in keinem guten Verhältnis zu diesem Nachteil zu stehen. Selbst diejenigen Anfänger, für die diese Information beim ersten Mal

noch interessant ist, werden sich vermutlich bei weiteren Anwendung eher davon gestört fühlen.

Im Folgenden werden einige Anregungen präsentiert, welche Änderungen und Verbesserungen noch in dem System umgesetzt werden könnten.

Ausblick

Erweiterungen
Für eine bessere Nutzererfahrung und erleichterte Handhabung wären folgende Erweiterungen interessant:

- Nutzer können die detaillierte Darstellung des for-Schleifenkopfes ein- und ausschalten.

- Nutzer können Schleifen minimieren, um den Graphen übersichtlicher zu gestalten und schneller durch den Ablauf zu navigieren. Dieses ließe sich dadurch realisieren, dass die in der Umsetzung vorgestellten Hilfsknoten (z.B. end-if-Knoten) als lineNumber nicht -1 zugewiesen bekommen, sondern den negativen Wert der Zeilennummer des Knoten, der den Schleifenrumpf beginnt. Dadurch ließen sich die Knoten weiterhin leicht entfernen und zusätzlich alle Knoten identifizieren, die innerhalb eines Rumpfes liegen.

- Die in jedem Schritt geänderten Variablen werden hervorgehoben. Dadurch ist es schneller ersichtlich, auf welche Werte sich die Ausführung einer Anweisung gerade auswirkt. Da Farben schon ausgiebig in der Visualisierung instrumentalisiert werden, wäre eine Darstellung mittels Bewegungen interessant (vgl.Kap. 2.2). Der Knoten der ausgeführten Anweisung könnte zusammen mit den geänderten Werten wackeln, um eine Verknüpfung der beiden darzustellen, die auch bemerkt wird, wenn Graphenknoten und veränderter Wert weit auseinander liegen.

Anwendungen
Als mögliche weitere Anwendung zusätzlich zu der Analyse der eigenen Lösungsversuche könnte es spannend sein, Aufgaben zu entwerfen, die bestimmte Algorithmen darstellen. Die Darstellung der Arrays zum Beispiel könnte dahingehend angepasst werden, dass Integer-Werte nicht in ihnen stehen, sondern ihre Höhe definieren. Ein mit dieser Änderung dargestellter Sortier-Algorithmus könnte gut veranschaulicht werden.

Verbesserungen

Mögliche Verbesserungen des Systems, die keine zusätzlichen Funktionen mit sich bringen könnten sein:

- Knoten von Verzweigungen in ihrer Größe limitieren
- Nicht alle aktuellen Werte in jedem Frame speichern, sondern nur die Änderungen darin festzuhalten. In der derzeitigen Umsetzung werden sehr viele redundante Daten gespeichert und von Backend an Frontend geschickt. Diese Datenmenge ließe sich gut reduzieren, wenn man genannte Verbesserung umsetzt

Literaturverzeichnis

[1] Bartram, Lyn ; Ware, Colin ; Calvert, Tom: Moticons: detection, distraction and task. In: *International Journal of Human-Computer Studies* 58 (2003), Nr. 5, S. 515–545

[2] Diehl, S.: *Software Visualization: Visualizing the Structure, Behaviour, and Evolution of Software*. Berlin, Heidelberg : Springer Berlin Heidelberg, 2007. – URL https://link.springer.com/book/10.1007/978-3-540-46505-8. – ISBN 978-3-540-46504-1

[3] Everett, Tom: *Antlr Project, grammars-v4*. 2023. – URL https://github.com/antlr/grammars-v4. – Zugriffsdatum: 2023-04-13

[4] Fekete, Jean-Daniel ; Van Wijk, Jarke J. ; Stasko, John T. ; North, Chris: The value of information visualization. In: *Information Visualization: Human-Centered Issues and Perspectives* (2008), S. 1–18

[5] Graham, Lisa: Gestalt theory in interactive media design. In: *Journal of Humanities & Social Sciences* 2 (2008), Nr. 1

[6] Heer, Jeffrey ; Boyd, Danah: Vizster: Visualizing online social networks. In: *IEEE Symposium on Information Visualization, 2005. INFOVIS 2005.* IEEE (Veranst.), 2005, S. 32–39

[7] Hinrichs, Torge ; Bureau, Henri ; Pilgrim, Jens von ; Schmolitzky, Axel: A Scaleable Online Programming Platform for Software Engineering Education. In: *Software Engineering (Satellite Events)*, 2021

[8] Hundhausen, Christopher D. ; Douglas, Sarah A. ; Stasko, John T.: A meta-study of algorithm visualization effectiveness. In: *Journal of Visual Languages & Computing* 13 (2002), Nr. 3, S. 259–290

[9] Kelleher, Curran: *d3-force*. 2016. – URL https://github.com/d3/d3-force. – Zugriffsdatum: 2023-04-18

[10] Khan, Muzammil ; Khan, Sarwar S.: Data and information visualization methods, and interactive mechanisms: A survey. In: *International Journal of Computer Applications* 34 (2011), Nr. 1, S. 1–14

[11] Khuri, Sami: A user-centered approach for designing algorithm visualizations. In: *Informatik/Informatique, Special Issue on Visualization of Software* 2 (2001), Nr. 2, S. 12–16

[12] Kienle, Holger M. ; Muller, Hausi A.: Requirements of software visualization tools: A literature survey. In: *2007 4th IEEE International Workshop on Visualizing Software for Understanding and Analysis* IEEE (Veranst.), 2007, S. 2–9

[13] Kim, Meen C. ; Zhu, Yongjun ; Chen, Chaomei: How are they different? A quantitative domain comparison of information visualization and data visualization (2000–2014). In: *Scientometrics* (2016), Nr. 1, S. 123–165

[14] Naps, Thomas L. ; Rössling, Guido ; Almstrum, Vicki ; Dann, Wanda ; Fleischer, Rudolf ; Hundhausen, Chris ; Korhonen, Ari ; Malmi, Lauri ; McNally, Myles ; Rodger, Susan u. a.: Exploring the role of visualization and engagement in computer science education. In: *Working group reports from ITiCSE on Innovation and technology in computer science education*. 2002, S. 131–152

[15] Oracle: *Java Platform Debugger Architecture Structure Overview*. 2023. – URL https://docs.oracle.com/en/java/javase/20/docs/specs/jpda/architecture.html. – Zugriffsdatum: 2023-04-18

[16] Palmer, Stephen ; Rock, Irvin: Rethinking perceptual organization: The role of uniform connectedness. In: *Psychonomic bulletin & review* 1 (1994), Nr. 1, S. 29–55

[17] Parr, Terence: *The definitive ANTLR 4 reference*. The Pragmatic Programmers, 2015. – ISBN 978-1-93435-699-9

[18] Purchase, Helen: Which aesthetic has the greatest effect on human understanding? In: *Graph Drawing* Bd. 97, 1997, S. 248–261

[19] Saraiya, Purvi ; Shaffer, Clifford A. ; McCrickard, D S. ; North, Chris: Effective features of algorithm visualizations. In: *Proceedings of the 35th SIGCSE technical symposium on Computer Science Education*, 2004, S. 382–386

[20] Treisman, Anne ; Gormican, Stephen: Feature analysis in early vision: evidence from search asymmetries. In: *Psychological review* 95 (1988), Nr. 1, S. 15

[21] Ward, M.O. ; Grinstein, G. ; Keim, D.: *Interactive Data Visualization: Foundations, Techniques, and Applications, Second Edition*. CRC Press, 2015 (360 Degree Business). – URL https://books.google.de/books?id=XHZ3CAAAQBAJ. – ISBN 978-1-4822-5737-3

[22] Ware, Colin: *Information Visualization: Perception for Design*. Fourth. Morgan Kaufmann, 2020. – URL https://www.elsevier.com/books/information-visualization/ware/978-0-12-812875-6. – ISBN 978-0-12-812875-6

[23] Ware, Colin: *Visual thinking for information design*. second. Morgan Kaufmann, 2021. – ISBN 978-0-12-823567-6

[24] Ware, Colin ; Purchase, Helen ; Colpoys, Linda ; McGill, Matthew: Cognitive measurements of graph aesthetics. In: *Information visualization* 1 (2002), Nr. 2, S. 103–110

[25] Wolfe, Jeremy M. ; Horowitz, Todd S.: Five factors that guide attention in visual search. In: *Nature Human Behaviour* 1 (2017), Nr. 3, S. 0058

[26] Yi, Ji S. ; Kang, Youn ah ; Stasko, John ; Jacko, Julie A.: Toward a deeper understanding of the role of interaction in information visualization. In: *IEEE transactions on visualization and computer graphics* 13 (2007), Nr. 6, S. 1224–1231

[27] Zambon, Eduardo ; Rensink, Arend: Recipes for Coffee: Compositional Construction of Jave Control Flow Graphs in GROOVE. In: *Principled Software Development* (2018), S. 305–323

A Anhang

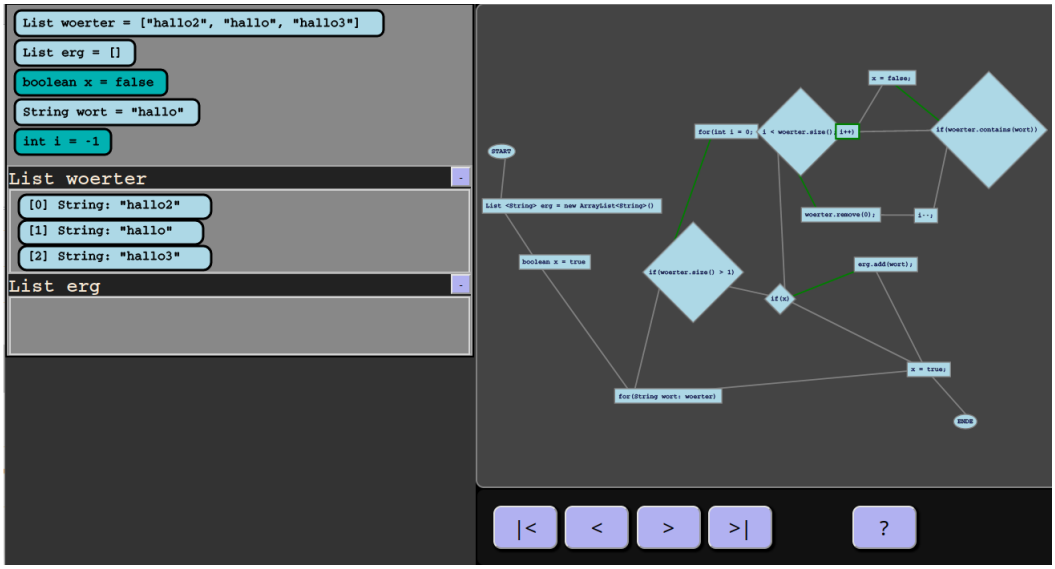


Abbildung A.1: Visualisierung 1

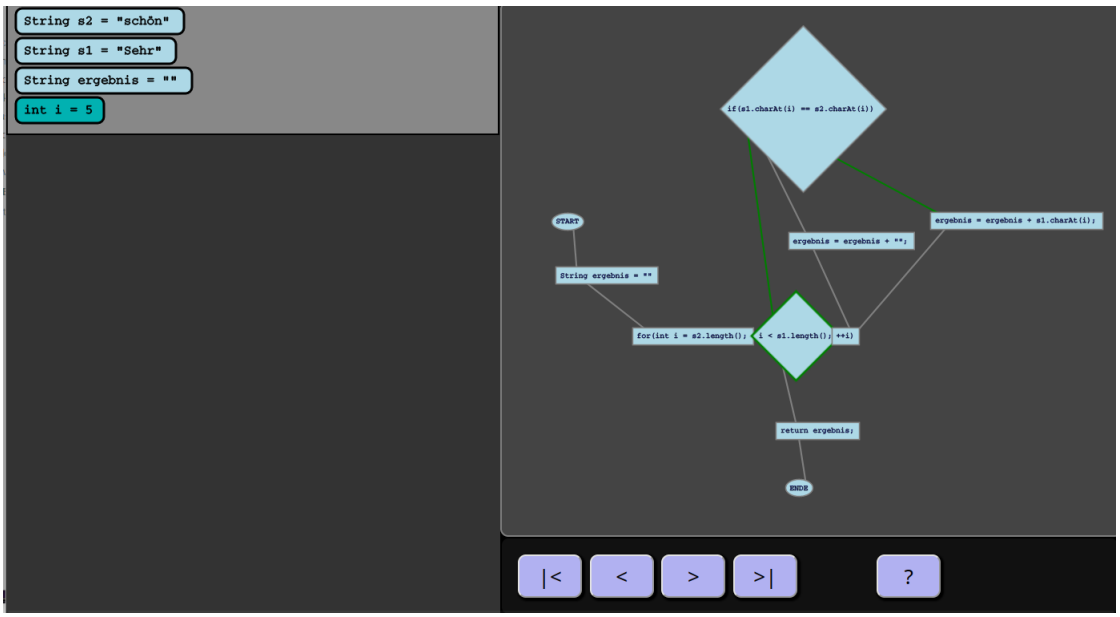


Abbildung A.2: Visualisierung 2

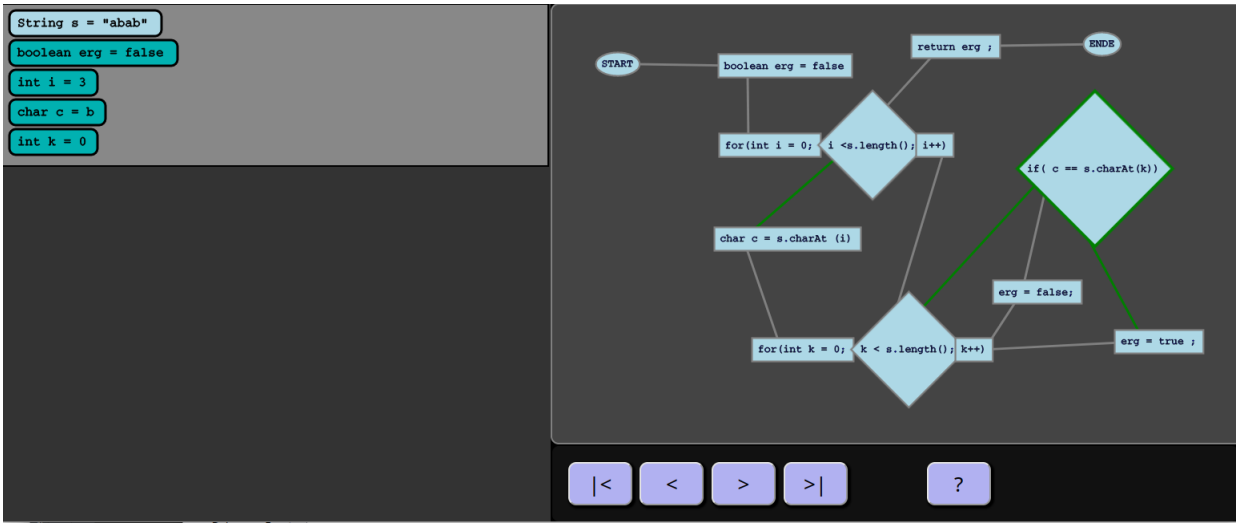


Abbildung A.3: Visualisierung 3

Erklärung zur selbstständigen Bearbeitung einer Abschlussarbeit

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich gemacht.

Ort	Datum	Unterschrift im Original