



Fakultät Elektrotechnik Feinwerktechnik Informationstechnik

Studiengang: Media Engineering

Bachelorarbeit:

Realisierung einer plattformunabhängigen Browseranwendung zur Erlernung von Programmiersprachen inspiriert von „Karel The Robot“, mittels JavaScript und HTML5

Autor:

Armin V O I T

Sommersemester 2013

Ausgabedatum: 15.02.2013

Betreuer: Prof. Dr. (USA) Ralph Lano

Erklärung gemäß APO §19 (8)

Ich bestätige, dass ich die Abschlussarbeit mit dem Titel:

„Realisierung einer plattformunabhängigen Browseranwendung zur Erlernung von Programmiersprachen inspiriert von „Karel The Robot“, mittels JavaScript und HTML5“

selbstständig verfasst, noch nicht anderweitig für Prüfungszwecke vorgelegt, keine andere als die angegebenen Quellen oder Hilfsmittel benutzt, sowie wörtliche und sinngemäße Zitate als solche gekennzeichnet habe.

Bei dem vorliegenden Exemplar handelt es sich um eine unkorrigierte Fassung seitens des betreuenden Hochschullehrers.

Nürnberg, den 19. Juli 13

Unterschrift:

Autor:

Armin Voit

Keplerstraße 3

93047 Regensburg

Matrikelnummer:

20 87 399

Studiengang:

Media Engineering

Betreuer:

Prof. Dr. (USA) Ralph Lano

Prüfer:

Prof. Dr. (USA) Ralph Lano

Prof. Dr. Heinz Brünig



GEORG-SIMON-OHM
HOCHSCHULE NÜRNBERG

Georg-Simon-Ohm Hochschule Nürnberg

Fakultät Elektrotechnik Feinwerktechnik Informationstechnik

Wassertorstraße 10

90489 Nürnberg

Inhaltsverzeichnis

Erklärung gemäß APO §19 (8)	II
Abbildungsverzeichnis	VII
Abkürzungsverzeichnis	X
1. Einleitung	1
1.1 Hintergrund und Motivation.....	1
1.2 Ziel der Arbeit.....	4
1.3 Aufbau der Arbeit	4
2. Requirements	6
2.1 Bereits existierende Lernprogramme	6
2.1.1 Scratch	7
2.1.2 Codeacademy	8
2.1.3 CodeHS.org	9
2.2 Nicht-funktionale Requirements.....	9
2.3 Funktionale Requirements.....	12
2.3.1 „Muss“-Requirements.....	12
2.3.2 „Kann“-Requirements	14
2.4 Systemanforderungen	14
2.4.1 Client.....	14
2.4.2 Server.....	15
3. Implementierung	16
3.1 Entwicklungssprachen und Elemente.....	16
3.1.1 HTML5.....	16

3.1.2 Canvas	19
3.1.3 CSS3	20
3.1.4 JavaScript	23
3.1.5 jQuery	25
3.1.6 IndexedDB.....	26
3.2 Umsetzung der Spiellogik.....	27
3.2.1 Programmierung der Grundfunktionen	28
3.2.2 draw()-Funktion	30
3.2.3 Implementierung der Bedingungen	31
3.2.4 Positionierung der Spielemente.....	32
3.2.5 Wertübermittlung	34
3.2.6 Wertspeicherung mittels IndexedDB.....	40
3.3 Entwicklung einer GUI.....	44
3.3.1 Begriffserklärung	44
3.3.2 Softwareergonomie	45
3.3.3 Aufbau und Komponenten	47
3.4 Hauptschwierigkeiten und Lösungsstrategien bei der Entwicklung.....	51
3.4.1 Timing.....	52
3.4.2 Relative Darstellung	67
3.4.3 Datenbankverwaltung	69
3.4.4 Vermeidung von Fehleingaben	73
4. Evaluation	76
4.1 Validierung.....	76
4.1.1 HTML.....	76
4.1.2 JavaScript	77
4.1.3 CSS	78
4.2 Test Cases.....	79
4.3 Traceability Matrix	81
4.4 IsoMetrics Fragebogen.....	83

5. Zusammenfassung	87
5.1 Ausblick	87
5.2 Persönliches Fazit.....	88
5.3 Danksagung	89
Literaturverzeichnis	90
Anhang	95
IsoMetrics Fragebögen	96
CD	102

Abbildungsverzeichnis

Abbildung 1: Karel the robot (Original Version).	
http://www.cs.mtsu.edu/~untch/karel/fundamentals.html	
(Stand: 12.07.13)	2
Abbildung 2: Anzahl freier IT-Jobs.	
http://www.bitkom.org/files/documents/BTIKOM_PK_Arbeitsmarkt_30_10_2012.pdf (Stand: 05.07.13)	3
Abbildung 3: Scratch.	
http://scratch.mit.edu/projects/editor/?tip_bar=getStarted	
(Stand: 13.07.13)	7
Abbildung 4: Codeacademy	
http://www.codecademy.com/de/courses/getting-started-v2/0/1?curriculum_id=506324b3a7dff00020bf661 (Stand: 13.07.13)	8
Abbildung 5: CodeHS.org http://www.code.org/learn/codehs	
(Stand: 13.07.13)	9
Abbildung 6: Entwicklungsmodell nach Piaget. Armin Voit 2013.	
unveröffentlicht.	11
Abbildung 7: Marktanteile der führenden Browserfamilien an der Internetnutzung in Europa.	
http://de.statista.com/statistik/daten/studie/164995/umfrage/marktanteil-e-der-browser-bei-der-internetnutzung-in-europa-seit-2009/	
(Stand: 12.07.13)	18
Abbildung 8: Kartesisches Koordinatensystem eines 500x500 Canvas Element. Armin Voit 2013. unveröffentlicht.....	19
Abbildung 9: Replikat der ersten Webseite des Internets.	
http://www.w3.org/History/19921103-hypertext/hypertext/WWW/TheProject.html (Stand: 10.07.13)	21
Abbildung 10: Ausschnitt aus <body> Tag (l.) und DOM-Baum (r.). Armin Voit 2013. unveröffentlicht.	24
Abbildung 11: IndexedDB Unterstützung führender Browser.	
http://caniuse.com/#search=indexedDB (Stand. 12.07.13)	26

Abbildung 12: Web SQL Unterstützung führender Browser. http://caniuse.com/#search=websql (Stand: 12.07.13)	27
Abbildung 13: Funktionsaufruf der <i>frontIsClear()</i> Bedingung. Armin Voit 2013. unveröffentlicht.	32
Abbildung 14: Inhalt der Arrays für Karel und Beeper. Armin Voit 2013. unveröffentlicht.	33
Abbildung 15: Darstellung eines framesets mit zwei frames. http://de.selfhtml.org/html/frames/anzeige/frames.htm (Stand: 11.07.13)	35
Abbildung 16: URL mit angehängtem Query String. Armin Voit 2013. unveröffentlicht.	37
Abbildung 17: Auszug aus der indexed Database. Armin Voit 2013. unveröffentlicht.	42
Abbildung 18: Beispiel für ein "Modal". Armin Voit 2013. unveröffentlicht.	48
Abbildung 19: Abbildung eines Popovers. Armin Voit 2013. unveröffentlicht..	49
Abbildung 20: Aufbau des Spiels "Karel the robot". Armin Voit 2013. unveröffentlicht.	50
Abbildung 21: Timing-Problem ohne sleep().Armin Voit 2013. unveröffentlicht.	53
Abbildung 22: Beschäftigungstaktik in der Praxis. Armin Voit 2013. unveröffentlicht.	57
Abbildung 23: Karels Welt als Mikrovariante. Armin Voit 2013. unveröffentlicht.	65
Abbildung 24: Mittelpunktsuche mit Beepern. Armin Voit 2013. unveröffentlicht.	66
Abbildung 25: Darstellung mit großen Unterschied im Reihen-Spalten-Verhältnis. Armin Voit 2013. unveröffentlicht.	68
Abbildung 26: Dynamischer Inhalt des Popovers von Own functions. Armin Voit 2013. unveröffentlicht.	70
Abbildung 27: Problem beim Löschvorgang (1). Armin Voit 2013. unveröffentlicht.	71

Abbildung 28: Problem beim Löschgvgang (2). Armin Voit 2013. unveröffentlicht.	72
Abbildung 29: HTML Validierung vor und nach Optimierung. Armin Voit 2013. unveröffentlicht.	77
Abbildung 30: JavaScript Validierung vor und nach der Optimierung. Armin Voit 2013. unveröffentlicht.	78
Abbildung 31: CSS Validierung vor und nach der Optimierung. Armin Voit 2013. unveröffentlicht.	
Abbildung 32: Auswertung der Test Cases. Armin Voit 2013. unveröffentlicht.	81
Abbildung 33: Traceability Matrix. Armin Voit 2013. unveröffentlicht.	82
Abbildung 34: Auswertung der IsoMetrics Fragebögen. Armin Voit 2013. unveröffentlicht.	85
Abbildung 35: Auswertung der IsoMetrics Fragebögen gruppiert nach Gestaltungskriterien. Armin Voit 2013. unveröffentlicht.	86

Abkürzungsverzeichnis

API	Application Programming Interface
App	Applikation
CLI.....	Command Line Interface
CSS	Cascading Style Sheets
DIN.....	Deutsches Institut für Normung
DOM.....	Document Object Model
ECMA	European Computer Manufacturers Association
EN	Europäische Norm
GUI.....	Graphical User Interface
HTML.....	Hypertext Markup Language
IEEE	Istitute of Electrical and Electronics Engineers
indexedDB	Indexed Database API
ISO	International Organization for Standardization
JS	JavaScript
NoSQL	Not only SQL
PC	Personal Computer
PDF	Portable Document Format
URL.....	Uniform Resource Locator

1. Einleitung

„I think everybody in this country should learn how to program a computer because it teaches you how to think.“

— Steve Jobs, The Lost Interview

1.1 Hintergrund und Motivation

Aller Anfang ist schwer. Diese Redewendung hat wahrscheinlich schon jeder mindestens einmal in den Mund genommen, wenn er sich in einem neuen Metier versucht hat. Doch wieso wird stillschweigend vorausgesetzt, dass es nicht angenehm oder begeisternd, ja sogar spaßig sein kann, wenn man sich neues Wissen aneignen möchte?

Jeder Student, der in seinem Studium zum ersten Mal Kontakt mit Softwareprogrammierung hatte, wurde wahrscheinlich über das sogenannte „Hello world“-Programm herangeführt. Dabei wird ein Programm geschrieben, gleichgültig welche Sprache, welches die einfache Funktion hat, eine textbasierte Ausgabe auf dem Display zu erzeugen mit dem Wortlaut „Hello world“. Genauso wie es angefangen hat, verläuft es in der Regel auch im weiteren Lernprozess. Der Lernende versucht sich im Programmieren und erhält als Bestätigung seiner korrekten Arbeit jeweils kurze Textausgaben.

Richard Pattis, damaliger Professor an der Stanford Universität in Kalifornien hatte sich genau zu diesem Thema Gedanken gemacht. Er erkannte, dass der Einstieg in die Programmierung auf diese Weise sehr trocken und nicht wirklich interessant war. Aus diesem Grund fasste er 1981 den Entschluss, seine

Studenten anders an die Materie heranzuführen. Aus seiner Idee den Studenten in einer Art Spiel das Programmieren zu lernen, entwickelte sich Karel the robot.

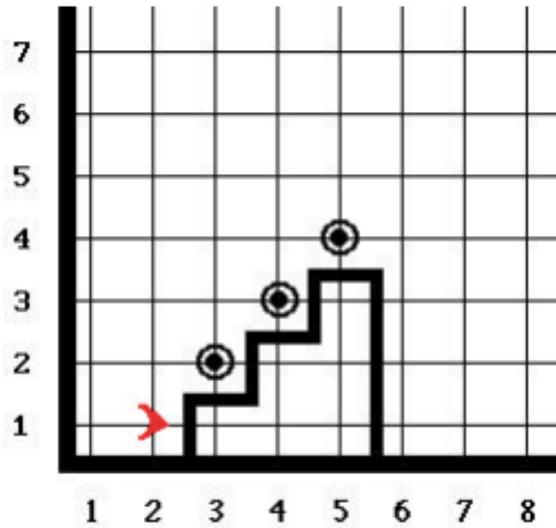


Abbildung 1: Karel the robot (Original Version)

Das Programm war eine sog. „Educational Programming Language“, sprich eine simple Programmiersprache mit der das Programmieren gelernt werden soll. Karel selbst wurde als roter Pfeil dargestellt. Karel wurde nach dem tschechischen Schriftsteller Karel Čapek (†1890 *1938) benannt. Dieser gilt als der Erfinder des Wortes Roboter.

Heute, knapp 30 Jahre später, ist die IT-Branche in gewaltigem Maße gewachsen. Und zwar so schnell, dass die Bevölkerung mit dem Bedarf an Fachkräften nicht mehr hinterherkommt. Im Jahr 2012 gab es in der deutschen Wirtschaft 43.000 offene Stellen in der IT-Branche. Davon lagen 75% in der Softwareentwicklung. Im Vergleich zum Vorjahr war dies ein Anstieg um 5000 Stellen. Das entspricht einem Zuwachs von 13%. Seit dem Jahr 2009 hat sich die Anzahl an nicht belegten Stellen in der IT-Branche mehr als verdoppelt.¹

¹ http://www.bitkom.org/files/documents/BTIKOM_PK_Arbeitsmarkt_30_10_2012.pdf (Stand: 12.07.13)

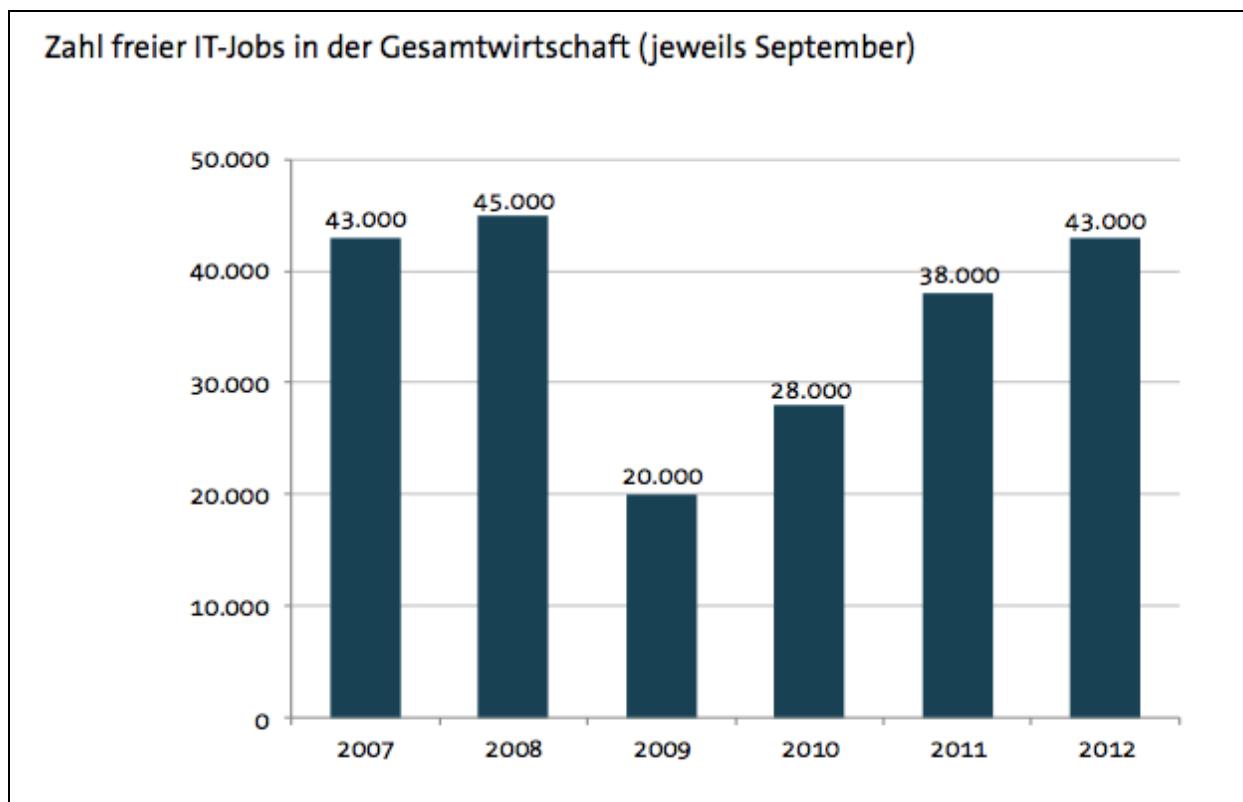


Abbildung 2: Anzahl freier IT-Jobs

Um diesem Trend entgegen zu wirken, haben sich mittlerweile viele Organisationen gegründet, die auch schon Kinder und Jugendliche für das Programmieren begeistern möchten. Ganz im Stil von Richard Pattis soll dabei der spielerische Aspekt beim Lernen im Vordergrund stehen. Einer der bekanntesten Vertreter dieses Genres ist die Webseite Code.org. Hinter der Seite steht eine gemeinnützige Organisation, die sich selbst die Aufgabe gesetzt hat, jedem Schüler die Chance zu ermöglichen, Programmieren zu lernen. Bis dato hat die Seite schon über 730.000 Unterschriften gesammelt, um ihr Ziel zu erreichen. Code.org bietet auf seiner Seite verschiedene Programme an, mit denen sich das Lernen leicht gestalten lassen soll. So z.B. Scratch; eine Mischung aus Paint und Programmieren. Der Spieler hat die Möglichkeit, seine eigene Figur zu erstellen und dieser daraufhin Fähigkeiten beizubringen. Ein weiteres Programm liefert Codeacademy. Hierbei kann der Spieler interaktiv programmieren lernen. Der Clou an der Sache ist, dass man sein Erlerntes in einer Art Highscore festhalten kann und sich so dann mit Anderen messen kann. Ein letztes Beispiel für ein Lernprogramm, das auf

Code.org angeboten wird, ist Karel the dog. Anspielungen auf Karel the robot sind gewollt, da es sich dabei um ein Remake des Klassikers aus den 80ern handelt. Eines von vielen; darunter Nicki der Roboter, Karel++, Guido van Robot und Karel J. Robot, um nur einige zu nennen.

Diese Bachelorarbeit ist ebenfalls als ein Remake, des von Richard Pattis geschaffenen Spiels Karel the robot anzusehen. Angetrieben von der Erinnerung an den eigenen zähen Lernprozess im Bereich der Programmierung, gepaart mit der Faszination dafür, dass Lernprogramme auch Spaß vermitteln können, entstand diese Version von Karel the robot erstmals in der Entwicklungsumgebung JavaScript.

1.2 Ziel der Arbeit

Das Ziel dieser Bachelorarbeit ist es, eine plattformunabhängige Webapplikation zur Erlernung von Programmiersprachen zu schaffen. Als Vorbild dafür diente das, von Richard E. Pattis geschriebene Spiel Karel the robot. Als grafische Orientierung wurde die Neuauflistung für Java, die von Eric Roberts im Jahr 2005 an der Stanford University verfasst wurde, herangezogen.

1.3 Aufbau der Arbeit

Nach dem einleitenden Kapitel, welches Auskünfte über Hintergrund und Motivation dieser Bachelorarbeit lieferte, folgen in Kapitel 2 die Anforderungen, die vor Beginn der Entwicklung an eine Neuauflistung des Spiels Karel the robot gestellt wurden.

Kapitel 3 befasst sich mit der gesamten Implementierung. Dabei wird zunächst auf die Entwicklungsumgebung und die benötigten Sprachen und

Elemente eingegangen. Im Anschluss wird die tatsächliche Umsetzung der Programmierung, sowie die Erstellung der grafischen Oberfläche beschrieben. Zum Schluss dieses Kapitels wird detailliert auf die Hauptschwierigkeiten im Entwicklungsprozess eingegangen.

Im 4. Kapitel wird der fertige Code getestet. Dabei wird das Programm sowohl in Hinsicht auf seine Syntax geprüft als auch auf seine Spielbarkeit. Abschließend wird im letzten Kapitel ein persönliches Fazit gezogen, sowie ein Ausblick auf eine mögliche Weiterentwicklung gegeben. Mit der Danksagung endet diese Arbeit.

2. Requirements

„The hardest single part of building a software system is deciding what to build.“²

Requirements, sprich Anforderungen, sind die Basis eines jeden Programms. Bevor auch nur die erste Zeile Code geschrieben wird, sollte man sich zunächst im Klaren darüber sein, was das Programm später einmal genau können soll. Dafür setzt man zu Beginn eines Softwareprojekts Requirements fest. Diese definieren meist in Textform, über welche Spezifikationen ein Softwareprogramm verfügen muss.

Im Folgenden wird zwischen funktionalen und nicht-funktionalen Requirements sowie Systemanforderungen unterschieden. Unter funktionalen Requirements versteht man die Funktionen, die dem Nutzer letztendlich im Programm zur Verfügung stehen, welche er auch bewusst wahrnimmt und die ihm eine Bedienung des Programms ermöglichen. Nicht-funktionale Requirements hingegen sind von allgemeiner Natur. Sie beschreiben Eigenschaften des Programms, wie z.B. eine einfache Bedienbarkeit. Systemanforderungen stellen die technischen Mindesterfüllungen von Seiten des Spielers dar.³

2.1 Bereits existierende Lernprogramme

Der Markt für Lernprogramme, die einen dabei unterstützen sollen programmieren zu lernen, ist mittlerweile sehr groß. Sogenannte Mini-

² Brooks, F.: *The Mythical Man-Month: Essays on Software Engineering*, Anniversary Edition. Boston: Addison-Wesley 1995

³ Dooley, J.: *Software Development and Professional Software*. New York: Apress 2011, S. 37

Sprachen, die in ihrer Komplexität sehr gering sind, versuchen dem Nutzer spielerisch logische Denkweise und Programmiersyntax beizubringen.

2.1.1 Scratch

Scratch ist ein Programm von der Lifelong-Kindergarten-Group am Media-Lab des MIT, bei dem der Spieler interaktive Geschichten und eigene Animationen programmieren kann. Die Zielgruppe liegt bei den 8- bis 16-Jährigen, die durch Scratch lernen sollen, systematisch und kreativ zu denken. Der Fokus bei dem Spiel liegt auf der Gemeinschaft. Alle erstellten Animationen werden der Öffentlichkeit zur Verfügung gestellt. So soll sich miteinander ausgetauscht werden und jeder jeden unterstützen. Gefördert wird Scratch unter anderem durch die Intel Foundation, Microsoft und Goolge.

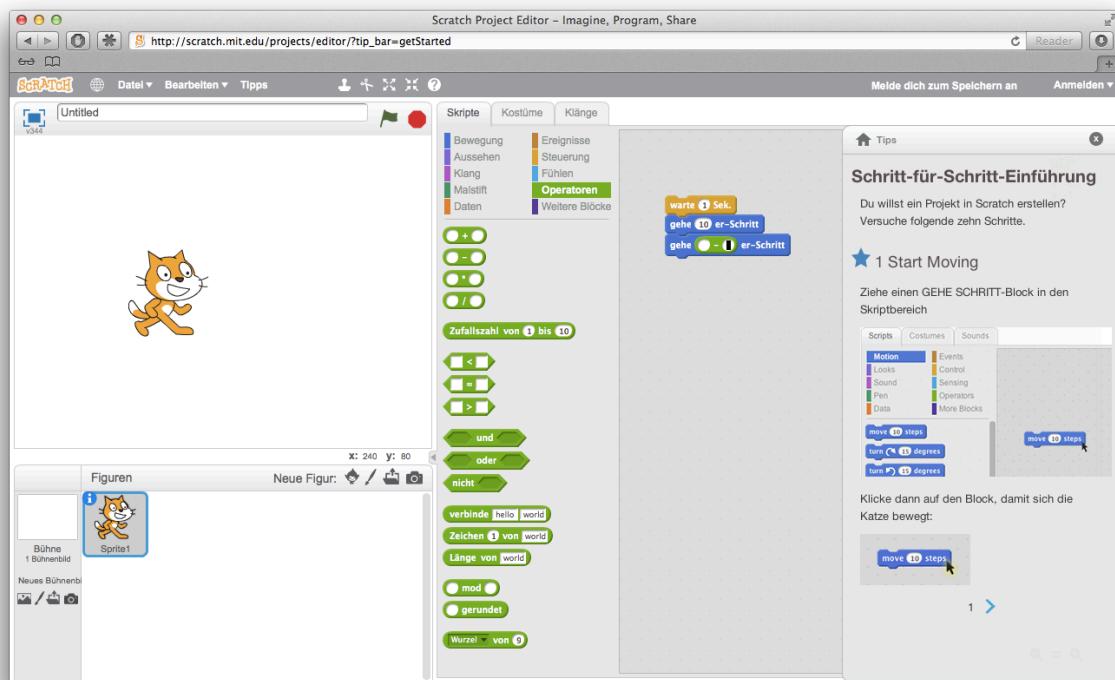


Abbildung 3: Scratch

2.1.2 Codeacademy

Codeacademy ist eine gemeinnützige Organisation, die eine Plattform anbietet, in der man interaktiv Programmieren lernen kann. Im Unterschied zu Scratch wird hierbei nicht eine generelle Programmiersyntax vermittelt, sondern konkrete Sprachen. Man kann wählen zwischen JavaScript, Ruby, Python u.v.m. Das Design ist sehr schlicht gehalten, da die Zielgruppe eher auf Jugendliche und Erwachsene ausgelegt ist. Knallige Farben werden bewusst vermieden. Dadurch entsteht ein seriöses Gesamtbild. Auch hier spielt das Gemeinschaftsgefühl eine große Rolle. Der Spieler kann sich registrieren und beim Lernen Auszeichnungen gewinnen, die er dann innerhalb der Gemeinschaft vorweisen kann. Im September 2011, einen Monat nach dem Release der Seite, hatten bereits 550.000 Menschen Codeacademy genutzt und mehr als sechs Millionen Übungen absolviert.⁴

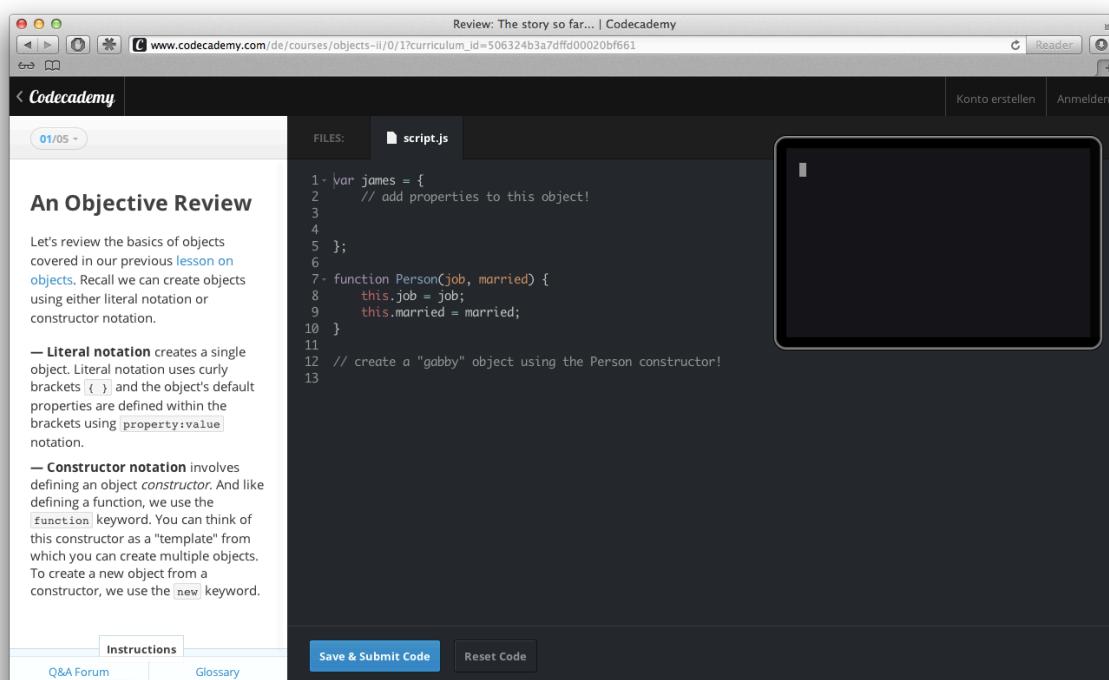


Abbildung 4: Codeacademy

⁴ <http://bits.blogs.nytimes.com/2011/09/14/codecademy-offers-free-coding-classes-for-aspiring-entrepreneurs/> (Stand: 01.07.13)

2.1.3 CodeHS.org

Ähnlich wie Scratch und Codeacademy ist auch CodeHS.org eine Plattform, die Menschen Programmieren beibringen möchte. Der große Unterschied ist, dass dieser Service kostenpflichtig ist. Der Spieler hat zwar die Möglichkeit auch kostenlos sich erste Programme anzusehen, möchte er jedoch alle Programme einsehen, Feedback zu seinem Programm erhalten oder sich in der Gemeinschaft austauschen, muss er monatlich einen Betrag von 25 USD bis 75 USD zahlen um einen Basis- bzw. Premiumzugang zu erhalten.

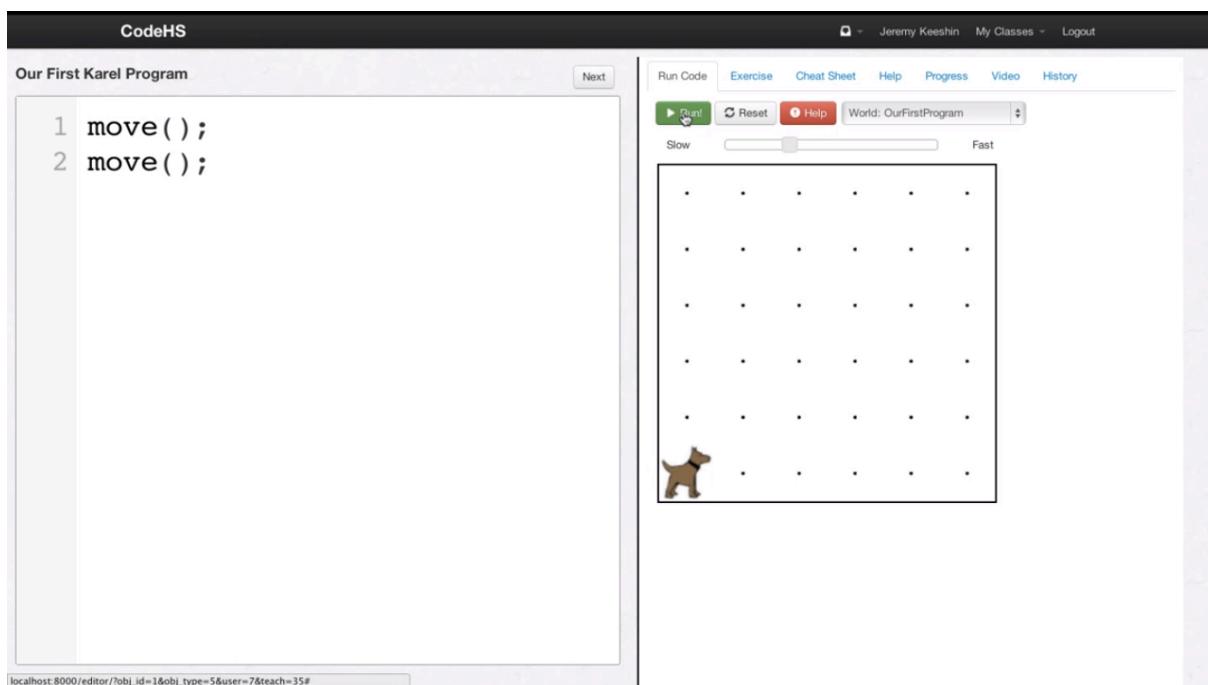


Abbildung 5: CodeHS.org

2.2 Nicht-funktionale Requirements

1. Zusammengehörige Elemente sollen als solche gekennzeichnet werden. Die Webapplikation soll über verschiedene Schaltflächen verfügen. Diese Schaltflächen beinhalten Buttons mit diversen Funktionen. Ähnliche Schaltflächen wie z.B. „Bearbeiten“ oder

„Löschen“ sollen gruppiert werden, um eine Bedienung logischer und intuitiver zu gestalten. Ebenfalls sollen Schaltflächen, die der gleichen Übergruppe angehören, wie beispielsweise *frontIsClear* und *leftIsClear* der übergeordneten Gruppe *Conditions*, gemeinsam im Spiel an gleicher Stelle erscheinen.

2. Unnötige Informationen sollen vermieden werden. Ein Spieler der etwas Neues lernt, darf nicht mit neuem Wissen überschüttet werden. Dies würde dem Lernen kontraproduktiv gegenüber stehen. Die einzelnen Spielemente sollen so kurz wie möglich und so detailliert wie nötig in einem vom eigentlichen Spiel separierten Bereich erklärt werden. Eine adäquate Menge an Information ist dem Spieler dabei zumutbar.
3. Ebenso wichtig ist eine intuitive Bedienung, da es sich bei Karel the robot um ein Lernprogramm handelt und es somit ohne große Vorkenntnisse benutzt werden soll. Ausgehend vom lateinischen Ursprungswort *intueri*, welches übersetzt soviel heißt wie betrachten, erwägen, aber auch ahnendes erfassen⁵, soll das Spiel Karel the robot schon durch bloße Betrachtung und ohne große Überlegungen schlüssig und verständlich sein. Aus diesem Grund sollen Bedienelemente und Design des Spiels auf ein Minimum reduziert werden.
4. Eckpfeiler dieser Arbeit sind die Anforderungen, dass das Spiel sowohl simpel gestaltet als auch intuitiv bedienbar ist.
Simpel sollte es sein, da es in erster Linie an Kinder gerichtet ist, die in ihrer kognitiven Entwicklung noch am Anfang stehen. Gemäß Jean Piaget⁶ unterteilt sich die kognitive Entwicklung von Kindern vor deren

⁵ Drosdowski, G.: Etymologie. Herkunftswörterbuch der deutschen Sprache; Die Geschichte der deutschen Wörter und der Fremdwörter von ihrem Ursprung bis zur Gegenwart. 7. Bd., Mannheim: Dudenverlag 1997, S. 310

⁶ Schweizer Entwicklungspsychologe und Epistemologe (Erkenntnistheoretiker)
*09.08.1896 - †16.09.1980

Pubertät in vier Stufen. Die ersten zwei Lebensjahre befindet sich ein Kind auf der *sensomotorischen Stufe*. Wie der Name vermuten lässt, entdeckt das Kind in dieser Zeit Zusammenhänge zwischen sensomotorischen Aspekten. Im Alter von zwei bis sieben Jahren durchschreitet ein Kind die *präoperationale Phase*. In diesem Lebensabschnitt beginnt das Kind Symbole zu gebrauchen, um intern Objekte zu repräsentieren, insbesondere durch Sprache. Stufe drei trägt den Namen *konkret-operationale Phase* und dauert bis zum elften Lebensjahr an. Auf dieser Stufe entwickelt sich das rationale Denken sowie die Logik des Kindes entscheidend. Demnach ist es am besten, dass sich Kinder in diesem Zeitabschnitt intensiv mit Übungen und Spielen, die das logische Denken fordern und fördern, wie z.B. Karel the robot, beschäftigen. Die vierte und letzte Phase lautet *formal-operationale Phase*. In dieser letzten Entwicklungsstufe des Kindes, die ab dem elften bis zum 15. Lebensjahr einsetzt, bildet sich die Fähigkeit abstrakt und hypothetisch denken zu können.

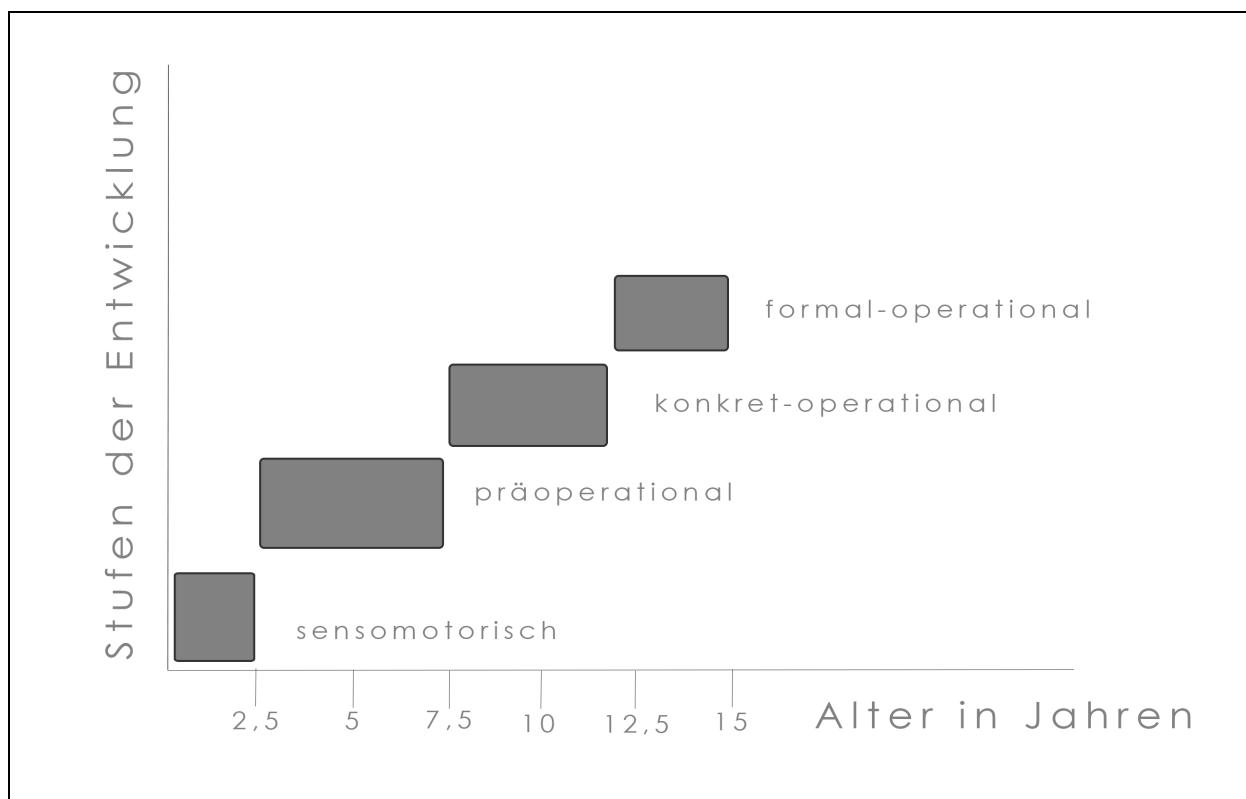


Abbildung 6: Entwicklungsmodell nach Piaget

All diese Phasen bauen aufeinander auf, wobei die Altersangaben nur Richtwerte sind und je nach Entwicklungstempo des Kindes unterschiedlich sein können.⁷

5. Schaltflächen mit Schlüsselfunktionen sollen durch Piktogramme zusätzlich verdeutlicht werden. So soll beispielsweise der Start-Button, welcher Karel den selbst geschriebenen Code ausführen lässt, mit einem Piktogramm in Form eines Play-Symbols (ein nach rechts gerichtetes Dreieck) erweitert werden.

2.3 Funktionale Requirements

Die funktionalen Requirements werden aufgeteilt in Muss- und Kann-Requirements.

2.3.1 „Muss“-Requirements

1. Der Spieler soll die Möglichkeit bekommen, auf der Startseite eine kurze Erklärung zur Seite zu erhalten. Um einem neuen Besucher der Webseite schnell den Sinn und Zweck der Seite zu vermitteln, soll er die Option haben mit geringem Aufwand eine kurze Zusammenfassung über das Spiel zu erhalten. Diese Information soll mit Hilfe eines schnell ersichtlichen Buttons auf der Startseite realisiert werden. Durch Drücken des Buttons soll der restliche Bildschirm ausgegraunt werden und lediglich die gewählte Information im Fokus stehen.
2. Der Spieler soll die Möglichkeit haben sich die komplette Steuerung im Detail erklären zu lassen. Dazu zählen alle Steuerelemente, die nötig sind

⁷ Piaget, J.; Inhelder, B.: Die Psychologie des Kindes. 9. Aufl., München: Deutscher Taschenbuch Verlag 1986, S. 153

um eine Funktion zu schreiben, diese zu speichern, zu bearbeiten und zu löschen. Des Weiteren zählen auch die Elemente dazu, die für die Ausführung von Code nötig sind, sowie Schaltflächen die der Navigation dienen.

3. Da der Spieler im Regelfall über keine Programmiererfahrung verfügt, soll er sich vor Spielbeginn ein Tutorial ansehen können, indem er schrittweise in die grundlegende Programmierung eingeführt wird. Dies soll entweder durch ein Video oder eine zum Durchklicken gestaltete Demo realisiert werden.
4. Der Spieler soll mehrere Levels auswählen können. Diese Levels sollen zudem in verschiedene Schwierigkeitsstufen unterteilt sein. Die Unterteilung erfolgt in Anfänger, Fortgeschrittene und Profis. Die einzelnen Stufen sollen ca. 4-6 Levels enthalten.
5. Die einzelnen Levels sind mit einer Anleitung zu versehen. Jeder Level erhält eine individuelle Anleitung, die dem Spieler zu Beginn sagt, welches Ziel er erreichen soll. Beginnt der Spieler, verschwindet die Anleitung. Falls der Spieler dies wünscht, soll sie jedoch im weiteren Spielverlauf zu jeder Zeit aufrufbar sein.
6. Um den Spieler beim Lernen zu unterstützen sollen Benutzereingaben, die dazu führen würden, dass Karel in einer Endlosschleife feststeckt, vermieden werden. Der Spieler soll in diesem Fall mit einem *alert*-Fenster oder ähnlichem auf seinen Fehler hingewiesen werden.
7. Der Spieler muss die Möglichkeit haben, eigene Funktionen speichern zu können. Die Speicherung soll dabei persistent und auf Clientseite erfolgen.

8. Vom Nutzer gespeicherte Funktionen sollen nach erfolgreicher Speicherung noch editier- und auch wieder lösbar sein.
9. Hauptaugenmerk bei der Konzipierung des Programms und somit auch ausschlaggebend für die Wahl der Entwicklungssprachen ist, die Zielsetzung, Karel the robot als plattformunabhängige Browseranwendung zu realisieren.

2.3.2 „Kann“-Requirements

10. Ein Leveleditor soll in das Spiel integriert werden. Der Spieler soll die Möglichkeit bekommen eigene Levels zu entwerfen.
11. Der Spieler soll die Möglichkeit haben, die Spielgeschwindigkeit, sprich die Bewegungsgeschwindigkeit von Karel, selbstständig zu regulieren. Realisiert werden soll dies durch einen Schieberegler.

2.4 Systemanforderungen

Um die Webapplikation fehlerfrei benutzen zu können, sollte der Spieler gewisse Systemvoraussetzungen erfüllen. Dazu wird zwischen clientseitigen und serverseitigen Voraussetzungen unterschieden.

2.4.1 Client

1. Der Spieler benötigt eine Internetverbindung. Da Karel the robot als Webapplikation gedacht ist, ist es nur online spielbar und bedarf einer ständigen Internetverbindung.

2. Der Spieler sollte einen Browser der neuesten Generation benutzen, der in der Lage ist HTML5 und CSS3 zu verarbeiten, mindestens aber einen Browser der folgenden Versionen:

Browser	IE	Firefox	Chrome	Safari	Opera
Version	10.0	10.0	23.0	3.1	15.0

3. Der Spieler sollte Cookies akzeptiert und die Option „Lösung von Cookies nach jeder Sitzung“ deaktiviert haben, da sonst eigene Funktionen nicht persistent im Browser gespeichert bleiben.

2.4.2 Server

Die gesamte Logik des Spiels läuft auf dem Browser des Nutzers, also clientseitig. Funktionen werden ebenfalls lokal beim Client abgespeichert. Aus diesem Grund sind Anforderungen an einen Server nicht notwendig.

3. Implementierung

Schwerpunkt dieser Bachelorarbeit ist es, sich auf praktische Weise mit der Thematik des Erlernens von Programmiersprachen auseinander zu setzen. Diesbezüglich wurde ein browserkompatibles Programm konzipiert und realisiert.

3.1 Entwicklungssprachen und Elemente

Alle Programme, die auf Computern oder anderen Endgeräten laufen, haben eines gemeinsam. Sie werden alle in einer Sprache geschrieben, die vom jeweiligen Endgerät interpretiert werden muss. Folgend werden die Sprachen vorgestellt, die für diese Arbeit verwendet wurden.

3.1.1 HTML5

Das Akronym HTML und steht für Hypertext-Markup-Language. Ein Hypertext ist ein elektronisches Dokument welches netzartig strukturiert ist und meist mit Querverweisen auf andere Dokumente verweist. Eine Markup-Language, zu deutsch Auszeichnungssprache, ist textbasiert und dient der Strukturierung von Inhalten wie Texten, Bildern und Hyperlinks⁸ in Dokumenten. HTML-Dokumente bilden die Basis des World Wide Web, die wiederum vom Browser interpretiert und dargestellt werden.

Da es Ende der 80er nicht möglich war, Informationen geordnet und in einheitlicher Form auf digitalem Weg zu übertragen, nahm sich der britische Informatiker Tim Burners-Lee dessen an und entwickelte 1989 die erste Version

⁸Verknüpfung, Verweis auf ein elektr. Dokument.

von HTML.⁹ Bis zur ersten Veröffentlichung vergingen noch drei weitere Jahre bis dann im November 1992 HTML offiziell vorgestellt wurde. Durch die rasante Entwicklung des Internets und die immer weiter steigenden Anforderungen an die dafür zuständige Auszeichnungssprache, wurden ständig Erweiterungen eingeführt. Um bei den fortwährenden Ergänzungen zu gewährleisten, dass HTML, aber auch das World Wide Web global einheitlich bleibt, setzte sich seit 1994 das W3C¹⁰ für deren Weiterentwicklung und Standardisierung ein.¹¹ Im November 1995 publizierte Tim Burners-Lee HTML in Version 2.0.¹² Wie schnell sich die HTML Spezifikationen weiterentwickelten, wurde ersichtlich als im Jahr 1995 die definierten Standards für Version 3.0 schon vor der Veröffentlichung wieder veraltet waren und diese somit nie erschien. Zwei Jahre später im Jahr 1997 wurden gleich zwei Versionen nämlich HTML 3.2 und HTML 4.0 auf den Markt gebracht. Erstmals wurden nun Inhalt und Layout durch sogenannte Cascading Stylesheets¹³ voneinander getrennt. Dieser Zustand ist auch heute noch so. Weitere zwei Jahre später stand HTML 4.01 bereit. Durch das rasante Erscheinen immer neuerer Versionen, hatte man die Vermutung, dass HTML nicht zukunftsweisend war und entschied sich somit für eine Abkehr vom bestehenden HTML hin zur Verwendung von XML¹⁴ und dem daraus resultierenden XHTML. Innerhalb der Jahre 2000 und 2006 erschienen so XHTML in Version 1.0, 1.1 und 2.0.¹⁵

Da jedoch immer noch ein großer Teil des WWW aus HTML Dokumenten bestand gründete sich 2004 eine Arbeitsgruppe namens WHATWG¹⁶, bestehend aus Mitgliedern der Browserentwickler Mozilla Foundation, Opera Software ASA und Apple Inc., um die HTML-Spezifikationen weiter

⁹ Böringer, J.; Bühler, P.; Schlaich, P.: Kompendium der Mediengestaltung. Produktion und Technik für Digital- und Printmedien. Berlin, Heidelberg: Springer Verlag 2011, S. 738

¹⁰ World Wide Web Consortium.

¹¹ Albert, K.; Stiller, M.: Smart Mobil Apps. Berlin, Heidelberg: Springer Verlag 2012, S. 149

¹² Burners-Lee, T.: Hypertext Markup Language – 2.0. MIT/W3C 1995, S. 1ff.

¹³ Beschreibungssprache zur Trennung von Inhalt und Darstellung.

¹⁴ Extensible Markup Language.

¹⁵ Lubbers, P.; Albers, B.; Salim, F.: Pro HTML5 Programming: Powerful APIs for Richer Internet Application Development. New York: Springer Verlag 2010, S. 1

¹⁶ Web Hypertext Application Technology Working Group.

voranzutreiben.¹⁷ Gemeinsam entwickelten sie einen Entwurf für HTML5, welchen sie 2009 präsentierten. In dieser Zeit entstand ebenfalls der Begriff des Web 2.0. Damit sollte ein völlig neues Internet beschrieben werden, welches sich nun nicht mehr rein statisch verhält, sondern zahlreiche Möglichkeiten der Interaktion für den Nutzer bereitstellt. Obwohl die HTML5-Spezifikationen bis dato noch nicht abgeschlossen sind, hat sich HTML5 schon zum Standard im Internet avanciert und kann auch von den führenden Browsern größtenteils interpretiert werden. Dazu zählen Googles Chrome, Firefox von Mozilla, Apples Safari und das Urgestein unter den Browsern der Internet Explorer von Microsoft.

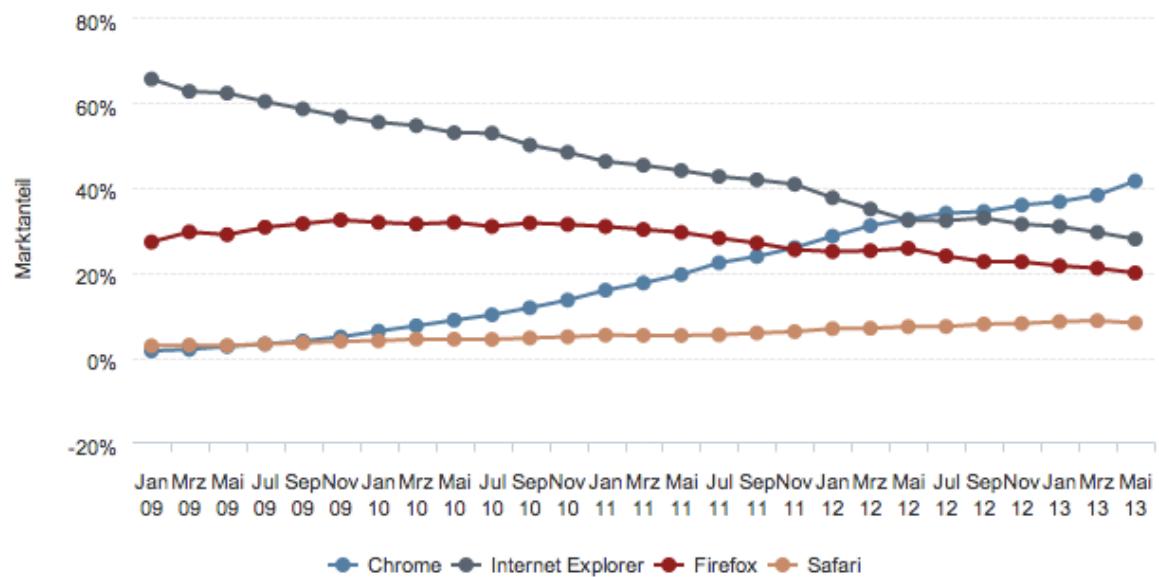


Abbildung 7: Marktanteile der führenden Browserfamilien an der Internetnutzung in Europa

Wie Abbildung 1 zu entnehmen ist, hat sich Google Chrome innerhalb von vier Jahren zum weitverbreitetsten Browser etabliert. Da Chrome eine Art Pionierrolle bei der Unterstützung von HTML5 übernahm, war dieser lange Zeit der Browser mit der besten HTML5 Unterstützung.¹⁸ Ein Grund für den Erfolg von Chrome ist wohl auch auf die frühe Integration der HTML5-Spezifikationen zurückzuführen.

¹⁷ Albert, K.; Stiller, M.: a. a. O., S. 149

¹⁸ <http://html5test.com/results/desktop.html> (Stand: 08.04.13)

3.1.2 Canvas

Um in Zeiten vor HTML5 z.B. Grafiken, Graphen oder einfache Animationen zu erstellen, war es den Entwicklern nicht möglich dies mit Hilfe des HTML-Dokuments zu definieren. Sie mussten sich hierfür mittels Flash behelfen und dies in das HTML-Dokument integrieren. Da moderne Browser, auch bedingt durch leistungsstärkere Rechner, mittlerweile aber durchaus in der Lage waren, selbst Animationen und Ähnliches zu erzeugen, setzte das W3C Canvas als Spezifikation für HTML5 fest.¹⁹ Canvas kann als Zeichenfläche innerhalb des HTML-Dokuments verstanden werden, auf welche der Entwickler wie auf ein Blatt Papier beliebig zeichnen kann. Die Zeichenfläche ist einem kartesischen Koordinatensystem nachempfunden, wobei der Nullpunkt des Koordinatensystems sich in der linken oberen Ecke der Canvas Elements, sprich der Zeichenfläche befindet.

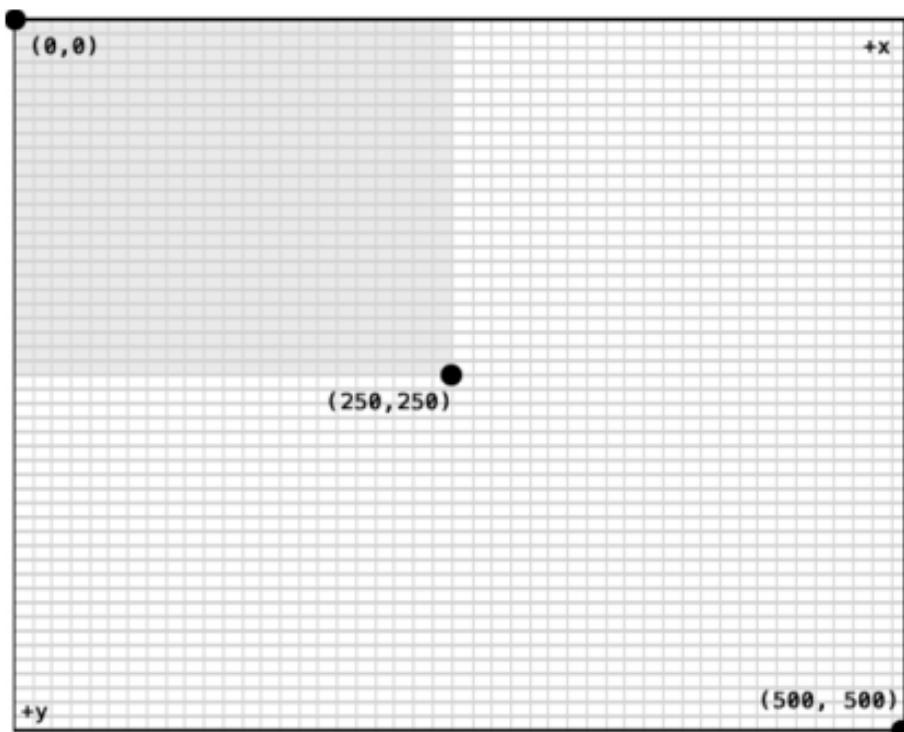


Abbildung 8: Kartesisches Koordinatensystem eines 500x500 Canvas Element

¹⁹ Graham, W.: Beginning Facebook Game Apps Development. New York: Springer Verlag 2012, S. 50

Anders als in der Mathematik geht die positive y-Achse nicht nach oben, sondern bedingt durch die Position des Ursprungs nach unten. Wie in eben dargestellter Abbildung zu sehen, liegt hier ein Canvas Element vor, welches eine Größe von 500x500 Pixeln aufweist. Darauf aufgezeichnet befindet sich ein semitransparentes Rechteck, welches sich vom Ursprung (0, 0) bis zum Mittelpunkt (250, 250) des Canvas Elements erstreckt. Die Rasterung des Canvas Elements ist hier nur zur Veranschaulichung eingefügt. Normalerweise ist das Element komplett transparent. Wie aus dem Koordinatensystem ersichtlich ist Canvas nur in der Lage zweidimensionale Graphiken abzubilden. Um dreidimensionale Elemente zu erstellen, kann man sich an WebGL²⁰ bedienen, welches ebenfalls in den gängigen Browsern integriert ist.

3.1.3 CSS3

Wie zuvor in Kapitel 3.1.1 *HTML5* kurz angesprochen, gibt es sogenannte Cascading Stylesheets, durch die es ermöglicht wird, Inhalt und Layout einer Homepage voneinander zu trennen. Doch wie kam es zur Entwicklung Cascading Stylesheets oder kurz CSS und worin liegt deren Vorteil?

Zu Beginn des Internets dachte noch niemand an eine „schöne“ Darstellung einer Webseite. Da es anfangs auch nur zu wissenschaftlichen Zwecken und dem reinen Datenaustausch verwendet wurde, war dies für die Nutzer auch nicht von Belang. Webseiten bestanden aus reinem Text. An die Integration von Bildern oder geschweige denn Videos war noch nicht zu denken. Deshalb fiel die Darstellung auch sehr schlicht aus. Im folgenden Bild wird die erste Webseite gezeigt, die im Internet online ging. Heute existiert diese Seite leider nicht mehr, jedoch hat das W3C ein Replikat hochgeladen, um einen Eindruck über die Anfänge des Internets zu vermitteln.²¹

²⁰ Web Graphics Library

²¹ <http://www.w3.org/History/19921103-hypertext/hypertext/WWW/TheProject.html> (Stand: 08.04.13)

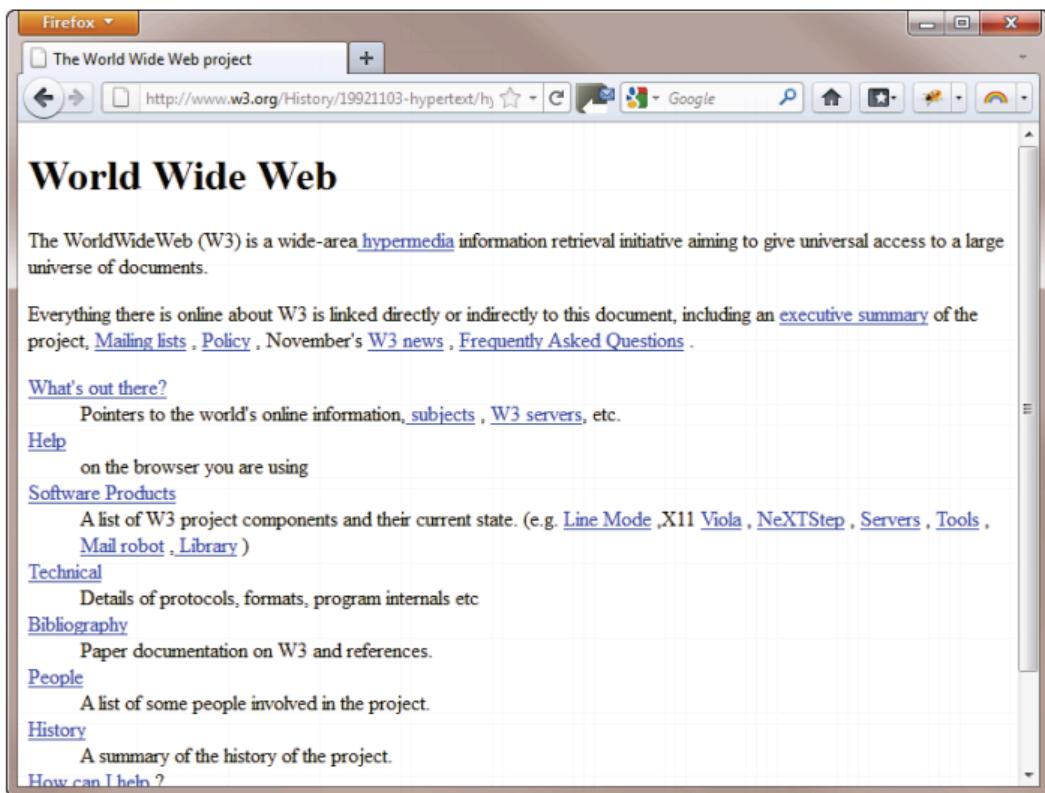


Abbildung 9: Replikat der ersten Webseite des Internets

Diese schlichte Darstellung langte den Nutzern jedoch schnell nicht mehr und man versuchte die beschränkten Möglichkeiten zu zweckentfremden. So wurden beispielsweise Überschriften, deren Schriftbild fetteter war als normaler Text, dazu verwendet, um Text fett zu schreiben. Tabellen wurden auf die gesamte Webseite angewendet, um verschiedene Bereiche optisch voneinander zu trennen. Die Zitierfunktion in HTML, welche den Text leicht einrückt, wurde öfter dazu benutzt, beliebige Elemente einzurücken, als tatsächlich etwas zu zitieren. Durch eine solche Missachtung von HTML-Strukturen fühlte sich das W3C berufen eine tiefgreifende Änderung herbeizuführen, woraufhin dieses 1996 die ersten Cascading StyleSheets vorstellte. CSS ermöglichte nun eine Vielzahl an Veränderungen, die der Entwickler vornehmen kann. Beispielsweise die Veränderung von Schriftgröße und -art, die Positionierung von Elementen oder das Verhalten von Elementen bei sog. Events.²² Zumindest theoretisch.

²² Ereignisse wie z.B. ein Mausklicks

In der Praxis sorgte CSS bei vielen Entwicklern lediglich für frustrierte Gemüter, da die damals führenden Browser Netscape und Internet Explorer (IE) kaum oder überhaupt keiner CSS-Unterstützung aufwiesen. Microsoft erkannte allerdings das Potential und die Notwendigkeit der Unterstützung von CSS und betrieb starke Anstrengungen in der Verbesserung der CSS-Unterstützung, um die Marktführung zu erlangen. Diese Zeit um die Vorherrschaft im Browsermarkt wird auch als Browerkrieg bezeichnet.²³ Das W3C hielt an CSS fest und veröffentlichte zwei Jahre später CSS2. Es dauerte weitere drei Jahre bis Microsoft im Jahr 2001 den IE6 veröffentlichte, der nun endlich einen Großteil der CSS2.0-Spezifikationen erfüllte und somit Netscapes Niederlage im Browerkrieg besiegelte, da dieser nur über eine unzureichende Unterstützung verfügte. Seit dem Jahr 2000 befindet sich die aktuelle Version, CSS3 in Entwicklung, woran sich auch bis heute nichts geändert hat. Nach wie vor werden Spezifikationen vom W3C definiert und erweitert.

Die entscheidenden Vorteile bei der Verwendung von Stylesheets sind folgende: Zum einem bietet es dem Entwickler vielfältigere Möglichkeiten bei der Gestaltung einer Homepage. Des Weiteren ist es, durch Verändern der CSS-Datei, nun möglich die Optik mehrerer Seiten auf einmal zu verändern, sofern sich diese auf die gleiche CSS-Datei berufen. Ebenfalls ein, durch die Trennung von Inhalt und Darstellung, resultierender Vorteil ist die geringere Menge an Code innerhalb eines Dokuments. So befindet sich in der HTML Datei nur Code zur Beschreibung des Inhalts und in der CSS-Datei nur Code zu Beschreibung des Aussehens der Elemente einer Webseite. Dieser Zustand macht den Code allgemein leichter zu lesen und somit verständlicher.²⁴

²³ Buxmann, P.; Diefenbach, H.; Hess, T.: Die Softwareindustrie. Ökonomische Prinzipien, Strategien, Prinzipien. Berlin, Heidelberg: Springer Verlag 2008, S. 31

²⁴ Powers, D.: Beginning CSS3. New York: Apress 2012, S. 1ff.

3.1.4 JavaScript

Mittels HTML5 und CSS3 sind die Möglichkeiten für die Gestaltung des Layouts einer Webseite schier grenzenlos. Wären diese zwei Technologien die einzigen Elemente bei der Internetprogrammierung, würde sich jede Webseite statisch verhalten und Nutzereingaben wären nicht realisierbar. Für diese Problematik wurde 1995 von Netscape Communication Corp. die von Brendan Eich entwickelte Skriptsprache JavaScript erstmals der Öffentlichkeit vorgestellt. Anfangs lief JavaScript noch unter den Namen Mocha bzw. LiveScript. Dies wurde jedoch schnell zu JavaScript geändert.

JavaScript ermöglichte es nun den Inhalt einer Webseite dynamisch zu verändern und auf Benutzereingaben zu reagieren. Die schnell wachsende Popularität verdankte JavaScript der Einbindung in den damals sehr beliebten und marktführenden Browser Netscape Navigator. Aufgrund dessen raschen Erfolgs beschloss Microsoft ebenfalls eine eigene JavaScript-Variante für dessen Internet Explorer auf den Markt zu bringen. Aus Lizenzgründen nannte Microsoft seine Technologie JScript.

Um zu verhindern, dass sich JavaScript und JScript in unterschiedliche Richtungen entwickeln, wurde die Sprache von der internationalen Normungsorganisation von Informations- und Kommunikationssystemen und Unterhaltungselektronik ECMA²⁵ standardisiert und offiziell als ECMAScript benannt. JavaScript und JScript sind somit offiziell nicht als eigenständige Sprachen, sondern als Implementationen zu ECMAScript anzusehen. Im allgemeinen Sprachgebrauch hat sich allerdings die Bezeichnung JavaScript durchgesetzt.²⁶

Aktuell ist der ECMAScript-Standard in der 5. Version vorhanden. Obwohl mittlerweile alle Spezifikationen von Version 5 von den neusten

²⁵ European Computer Manufacturers Association.

²⁶ Koch, S.: JavaScript. Einführung, Programmierung und Referenz. Heidelberg: dpunkt.verlag 2011, S. 11f.

Browsersversionen der führenden Hersteller²⁷ interpretiert werden können, werden die Möglichkeiten von den Entwicklern nur vorsichtig genutzt, da nach wie vor noch viele veraltete Browsersversionen sowie Browser von anderen Herstellern im Umlauf sind, welche ECMAScript 5 nicht oder nur teilweise interpretieren können.²⁸

Das ECMAScript erlaubt nun eine Webseite dynamisch zu ändern, jedoch sind hier nur die grundlegenden Sprachelemente standardisiert. Um zu gewährleisten, dass alle Elemente einer Webseite auf unterschiedlichen Browser nicht unterschiedlich angesprochen werden, wurde zusätzlich das Document Object Model (DOM) vom W3C entwickelt. Das DOM gibt vor, wie jedes Element einer Seite angesprochen wird und wie sie in Relation zueinander stehen. Elemente können z.B. Bilder, Texte oder auch Attribute, wie Farben, Schriftart, Schriftstil usw. sein. Folgende Grafik soll dies veranschaulichen.

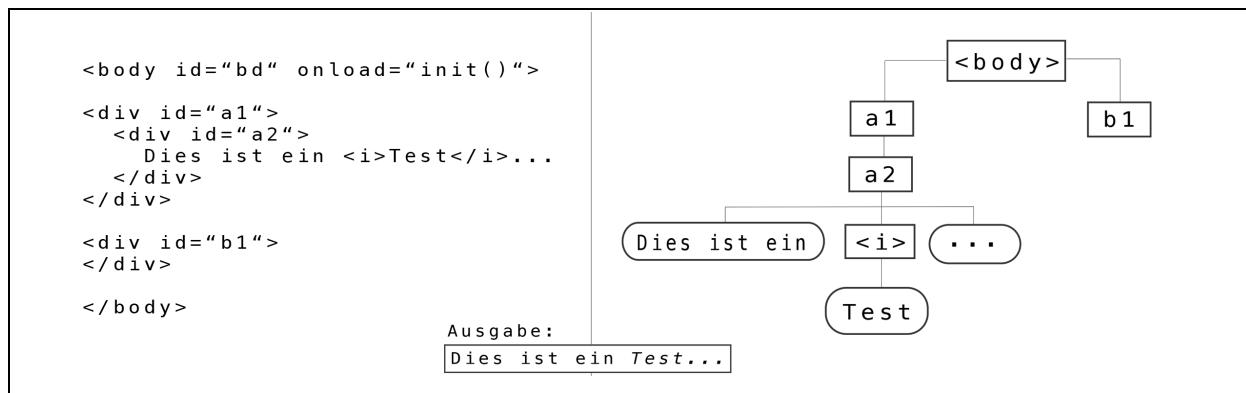


Abbildung 10: Ausschnitt aus <body> Tag (l.) und DOM-Baum (r.)

Links ist ein Ausschnitt aus dem <body>-Teil eines HTML Codes zu sehen und rechts der dazugehörige DOM-Baum²⁹. Anhand des DOM-Baums ist nun gut zu erkennen, wie die einzelnen Elemente zueinander in Verbindung stehen. Die einzelnen Positionen werden Knoten genannt. Die rechteckigen Kästchen werden als Elemente und die abgerundeten als Textknoten bezeichnet. Mit

²⁷ Google, Mozilla, Apple, Microsoft.

²⁸ <http://kangax.github.io/es5-compat-table/#> (Stand: 11.05.13)

²⁹ Objekthierarchie einer Webseite

dieser Anordnung ist es möglich, von einem Knoten zum anderen zu gelangen, diesen zu verändern, löschen oder neue Objekte an einer bestimmten Stelle hinzuzufügen.³⁰ ³¹

3.1.5 jQuery

jQuery ist eine freie und umfangreiche JavaScript-Bibliothek, die es ermöglicht HTML-Dokumente zu manipulieren. jQuery kann beispielsweise auf Attribute von einzelnen Elementen zugreifen und diese nach Belieben verändern. Des Weiteren ist jQuery sehr leichtgewichtig und in seiner Schreibweise deutlich kürzer als vergleichbarer JavaScript-Code. Folgender Beispielcode greift jeweils auf das Element „example“ über dessen ID zu:

JavaScript-Code:

```
document.getElementById(„example“)
```

jQuery-Code:

```
$( „#example“ )
```

Wie im obigen Beispiel gut zu sehen, fällt der jQuery-Code deutlich geringer aus. Diese komfortable Eigenschaft hat jQuery seit seiner ersten Veröffentlichung im Januar 2006 auf dem Bar Camp NYC durch dessen Erfinder John Resig stetig wachsen und an Popularität gewinnen lassen.³² ³³

³⁰ Koch, S.: JavaScript. Einführung, Programmierung und Referenz. Heidelberg: dpunkt.verlag 2011, S. 160f.

³¹ Franklin, J.: Beginning jQuery. New York: Apress 2013, S. 16.

³² <http://www.jquery.com> (Stand: 12.06.13)

³³ Franklin, J.: Beginning jQuery. A.a.O., S. 15.

3.1.6 IndexedDB

IndexedDB ist eine in HTML5 erschienene NoSQL-Datenbank³⁴ im Browser. Die vollständige Bezeichnung der Datenbank lautet: „Indexed Database API“. IndexedDB verfügt über die große Stärke, anders als Web Storage³⁵ neben Strings auch Zahlen, Objekte, Arrays und sogar BLOBs³⁶ persistent speichern zu können. Des Weiteren ist es möglich, einfach und gezielt gespeicherte Inhalte in der Datenbank anzusprechen und zu verändern, sowie über Teile der Datenbank zu iterieren. Bisher gibt es leider noch einige kleine Hürden bei der generellen Kompatibilität.

	IE	Firefox	Chrome	Safari	Opera	iOS Safari	Opera Mini	Android Browser	Blackberry Browser	Opera Mobile	Chrome for Android	Firefox for Android
23 versions back			4.0									
22 versions back			5.0									
21 versions back	2.0	6.0										
20 versions back	3.0	7.0										
19 versions back	3.5	8.0										
18 versions back	3.6	9.0										
17 versions back	4.0	moz 10.0										
16 versions back	5.0	moz 11.0	webkit									
15 versions back	6.0	moz 12.0	webkit									
14 versions back	7.0	moz 13.0	webkit									
13 versions back	8.0	moz 14.0	webkit									
12 versions back	9.0	moz 15.0	webkit									
11 versions back	10.0	moz 16.0	webkit									
10 versions back	11.0	moz 17.0	webkit	9.0								
9 versions back	12.0	moz 18.0	webkit	9.5-9.6								
8 versions back	13.0	moz 19.0	webkit	10.0-10.1								
7 versions back	14.0	moz 20.0	webkit	10.5								
6 versions back	15.0	moz 21.0	webkit	10.6			2.1		10.0			
5 versions back	5.5	16.0	webkit 3.1	11.0			2.2		11.0			
4 versions back	6.0	17.0	webkit 3.2	11.1	3.2		2.3		11.1			
3 versions back	7.0	18.0	24.0	4.0	11.5	4.0-4.1	3.0		11.5			
2 versions back	8.0	19.0	25.0	5.0	11.6	4.2-4.3	4.0		12.0			
Previous version	9.0	20.0	26.0	5.1	12.0	5.0-5.1	4.1	7.0	12.1			
Current	10.0	21.0	27.0	6.0	12.1	6.0-6.1	5.0-7.0	4.2	10.0 webkit	14.0	25.0	19.0
Near future	11.0	22.0	28.0		15.0							
Farther future	23.0	29.0										

Abbildung 11: IndexedDB Unterstützung führender Browser

Während ein Teil der führenden Browserhersteller IndexedDB unterstützt, ist dies bei Apples Safari, dessen iOS Browser sowie bei Android und Opera nicht

³⁴ Nicht-relationale Datenbank ohne starres Tabellenschemata.

³⁵ Technik für Webanwendungen zur Speicherung von Daten in einem Browser als String.

³⁶ Binary Large Objects. Große binäre Objekte, z.B. Audio- und Videodateien.

der Fall. Jedoch kann dies mit Hilfe eines Polyfills³⁷ und Web SQL³⁸ gelöst werden. Auf diesem Umweg wird die Funktionalität von IndexedDB mittels Web SQL hergestellt, da IndexedDB - inkompatible Browser mit Web SQL-Standard kompatibel sind, wie die folgende Abbildung aufzeigt.

	IE	Firefox	Chrome	Safari	Opera	iOS Safari	Opera Mini	Android Browser	Blackberry Browser	Opera Mobile	Chrome for Android	Firefox for Android
23 versions back			4.0									
22 versions back			5.0									
21 versions back	2.0		6.0									
20 versions back	3.0		7.0									
19 versions back	3.5		8.0									
18 versions back	3.6		9.0									
17 versions back	4.0		10.0									
16 versions back	5.0		11.0									
15 versions back	6.0		12.0									
14 versions back	7.0		13.0									
13 versions back	8.0		14.0									
12 versions back	9.0		15.0									
11 versions back	10.0		16.0									
10 versions back	11.0		17.0		9.0							
9 versions back	12.0		18.0		9.5-9.6							
8 versions back	13.0		19.0		10.0-10.1							
7 versions back	14.0		20.0		10.5							
6 versions back	15.0		21.0		10.6			2.1		10.0		
5 versions back	5.5	16.0	22.0	3.1	11.0			2.2		11.0		
4 versions back	6.0	17.0	23.0	3.2	11.1	3.2		2.3		11.1		
3 versions back	7.0	18.0	24.0	4.0	11.5	4.0-4.1		3.0		11.5		
2 versions back	8.0	19.0	25.0	5.0	11.6	4.2-4.3		4.0		12.0		
Previous version	9.0	20.0	26.0	5.1	12.0	5.0-5.1		4.1	7.0	12.1		
Current	10.0	21.0	27.0	6.0	12.1	6.0-6.1	5.0-7.0	4.2	10.0	14.0	25.0	19.0
Near future	11.0	22.0	28.0		15.0							
Farther future		23.0	29.0									

Abbildung 12: Web SQL Unterstützung führender Browser

3.2 Umsetzung der Spiellogik

Die Realisierung der Browseranwendung und somit die Programmierung erfolgte in mehreren Schritten. So wurden zu Beginn der Entwicklung lediglich Karel in seiner Welt und dessen Grundbewegungen implementiert. Über vorgefertigte Funktionen konnte Karel sich bereits in seiner Welt bewegen. Im weiteren Verlauf wurden Bedingungen implementiert, die Karel nun abfragen konnte und bei seinen Bewegungen berücksichtigte. Zum Schluss wurde es den Nutzern ermöglicht eigene Funktionen zu speichern und zu bearbeiten.

³⁷ Herunterladbarer Code um eine bestimmte Fähigkeit, die in einem Webbrower nicht integriert ist, nachträglich zu ermöglichen.

³⁸ Relationale Datenbank im Browser (Weiterentwicklung eingestellt).

3.2.1 Programmierung der Grundfunktionen

Da Karel primär ein Spiel für Kinder ohne Programmiererfahrung ist, ist es sehr simpel gehalten und bietet eine überschaubare Anzahl an Grundfunktionen. Neben den richtungsbestimmenden Funktionen `move()`³⁹ und `turnLeft()`⁴⁰ gibt es noch die Beeper-bezogenen Funktionen `pickBeeper()`⁴¹ und `putBeeper()`⁴². Um zu verstehen wie die Grundfunktionen aufgebaut sind, muss man sich vor Augen führen, dass Karels Welt einem Schachbrett-Muster gleicht. Karels Welt besteht aus einer Anzahl von Reihen und Spalten. An deren Schnittpunkten kann sich Karel entweder horizontal oder vertikal bewegen. Zu Beginn jeder Funktion muss erst herausgefunden werden an welcher Position Karel aktuell steht. Hierfür läuft ein zweidimensionales Array in Form einer Schleife über jeden Schnittpunkt der Spalten und Säulen des Spielfelds und somit über die möglichen Standpunkte Karels. Zuvor wurde Karels Ausgangsposition beim Spielstart in einem Array abgespeichert. Bei diesem Array handelt es sich ebenfalls um ein zweidimensionales Array, welches einen booleschen Wert über die An- bzw. Abwesenheit Karels zurückliefert.

Wird Karel nun nach Aufruf der `move()`-Funktion auf dem Feld geortet, wird daraufhin dessen Blickrichtung überprüft und ob sich direkt vor ihm in Laufrichtung eine Mauer befindet. Die Ermittlung der Blickrichtung ist von Belang, um zu bestimmen in welche Himmelsrichtung Karel sich fortbewegen soll. Ähnlich wie bei der Position von Karel, liefert auch hier ein boolescher Wert die Informationen über Karels Zustand. Die Speicherung der Zustände erfolgt hierbei in einem JSON Dokument. Befindet sich nun an der Position zu der sich Karel hinbewegen möchte eine Wand, so wird die Funktion an dieser Stelle abgebrochen. Ist dies aber nicht der Fall, wird das Positionsarray dementsprechend aktualisiert, überprüft ob ein Beeper an dieser Position

³⁹ Funktion, die Karel einen Schritt nach vorne machen lässt.

⁴⁰ Funktion, die Karel eine 90°-Linksdrehung vollziehen lässt.

⁴¹ Funktion, die Karel einen Beeper aufheben lässt.

⁴² Funktion, die Karel einen Beeper ablegen lässt.

liegt, sowie ob sich um Karel herum Wände befinden. Zusätzlich zu dem Array, das die Positionen der Beeper innehaltet, wird ein JSON-Dokument aktualisiert, welches zu jeder aktuellen Position Karels Auskunft darüber gibt, ob an selbiger Stelle ein Beeper liegt. Ebenso werden die unmittelbar umliegenden Wände in einem eigens dafür zuständigen JSON-Dokument eingetragen. Diese JSON-Dokumente sind für die Conditions, zu Deutsch Bedingungen, welche im nächsten Gliederungspunkt 3.2.3 Implementierung der Bedingungen näher erläutert werden, von Bedeutung. Abschließend wird nun die `draw()`-Funktion aufgerufen, um Karel an seiner neuen Position zu „zeichnen“. Diese Funktion wird gegen Ende des Kapitels erläutert.

Ähnlich wie bei der `move()`-Funktion wird bei der `turnLeft()`-Funktion zunächst Karel's Standpunkt über das Positionsarray abgerufen. Zuvor wurde die Blickrichtung erfasst und das JSON-Dokument auf die neue Blickrichtung nach Vollenden der Drehung aktualisiert. Da sich nach einer Drehung Karel in einer anderen Relation zu umliegenden Wänden befindet - so ist beispielsweise eine Wand die vor der Drehung vor ihm lag nach einer Linksdrehung nun zu seiner Rechten - wird das zuständige JSON-Dokument ebenfalls angepasst. Wie auch zuvor bei `move()` wird abschließend die `draw()` aufgerufen.

Im Gegensatz zu den beiden positionsverändernden Funktionen `move()` und `turnLeft()` wird bei `pickBeeper()` und `putBeeper()` Karels Umwelt verändert. Wird die Funktion `pickBeeper()` aufgerufen, wird zunächst, wie schon bei den beiden Funktionen zuvor, Karel's Position mittels Iteration ermittelt. Befindet sich nun ein Beeper an Karel's Position, wird dieser aufgenommen. Hierzu wird das Beeper-Array, sowie das dazugehörige JSON-Dokument, welches Auskunft darüber gibt, ob sich ein Beeper an der aktuellen Position und ob sich ein oder mehrere Beeper in Karel's Besitz befinden, aktualisiert. Zusätzlich wird die Variable `countBeeper` um den Wert eins hochgezählt. Diese Variable repräsentiert die Anzahl der Beeper über die Karel verfügt und ist für die Funktion `putBeeper()` von Bedeutung.

Als letzte Grundfunktion wäre `putBeeper()` zu nennen. Analog zu `pickBeeper()` wird anfangs eine Positionsbestimmung durchlaufen und überprüft ob sich einerseits schon ein Beeper an aktueller Stelle befindet und andererseits ob Karel noch über mindestens einen Beeper verfügt. Sollte bereits ein Beeper an Karels Aufenthaltpunkt vorhanden sein, bricht die Funktion ab. Andernfalls wird ein Beeper, sofern Karel über einen verfügt, abgelegt. Hierzu wird ebenfalls das Beeper-Array und das JSON-Dokument aktualisiert, wie zuvor auch bei `pickBeeper()`. Die beiden Funktionen `pickBeeper()` und `putBeeper()` rufen zusätzlich zur abschließenden `draw()`-Funktion noch die Funktion `setBeeperFalse()` bzw. `setBeeperTrue()` auf, welche ein zweites Beeper-Array aktualisieren, dessen genaue Funktionsweise im Punkt 3.4 *Hauptschwierigkeiten und Lösungsstrategien bei der Entwicklung* näher erklärt wird.

3.2.2 draw()-Funktion

Die `draw()`-Funktion ist von entscheidender Bedeutung, weshalb sie hier in einem eigenen Punkt erklärt wird. Um die Grundfunktionen überhaupt darstellen zu können, bedarf es einer eigenen Funktion, die diese Tätigkeit übernimmt. Diese Aufgabe wird der `draw()`-Funktion zugeschrieben. Sie ermöglicht es erst die einzelnen Arrays auszuwerten, um so Karels Welt grafisch abzubilden. Da das anfangs beschriebene Canvas für die grafische Darstellung zuständig ist, ist es leider nicht möglich einzelne Elemente der Welt, z.B. einen aufgenommenen Beeper, zu entfernen. Canvas verhält sich, gemäß der deutschen Übersetzung, wie eine Leinwand. Wenn einmal etwas gezeichnet wurde, kann es nicht mehr gelöscht sondern nur noch überzeichnet werden. Damit nicht irgendwann das komplette Spielfeld voller Karel Figuren gezeichnet ist, muss vor jeder Aktion die komplette Canvas Fläche gelöscht werden und wieder mit allen seinen Bestandteilen gezeichnet werden. Dazu zählt zunächst die Darstellung der Spalten und Zeilen, welche in den Variablen `columns` und `rows` gespeichert sind, sowie die

dazugehörige Nummerierung an den Rändern des Spielfelds. Ebenfalls zur Darstellung der Welt nötig sind die Markierungskreuze, die das Koordinatensystem der Welt repräsentieren. Nach dem Zeichnen der Wände und Markierungskreuze werden die Beeper und Karel gezeichnet. Hierfür werden die zuständigen Arrays abgefragt und ausgewertet. Karel selbst wird nicht gezeichnet, sondern als fertiges PNG-Bild geladen und eingefügt. Je nachdem in welche Richtung Karels Blick fällt, wird ein entsprechendes Bild geladen. Wird eine neue Aktion ausgeführt wird der komplette Funktionsapparat wiederholt.

3.2.3 Implementierung der Bedingungen

Damit Karel selbstständig Entscheidungen treffen kann, ist es nötig Bedingungen einzufügen, die er abfragen und auswerten kann. Diese Bedingungen werden mittels eigener Funktionen realisiert. Dabei kann grob zwischen drei Gruppen unterschieden werden. Die Richtungs-, die Beeper- und die Umgebungsbezogenen. Die richtungsbezogenen Bedingungen geben Auskunft darüber, in welche Himmelsrichtung Karels Blick zum Zeitpunkt des Funktionsaufrufs zeigt; die beeper-bezogenen darüber, ob sich an Karel Standpunkt ein Beeper befindet und ob sich mindestens ein Beeper in seinem Besitz befindet; die Umgebungsbezogenen darüber, ob sich direkt um Karel herum Wände befinden, die ihn am Weiterlaufen hindern.

Alle diese Funktionen sind in ihrem Aufbau identisch. Wird eine Funktion aufgerufen, liefert sie lediglich einen Rückgabewert der wiederum aus dem zuständigen JSON-Dokument gewonnen wird.

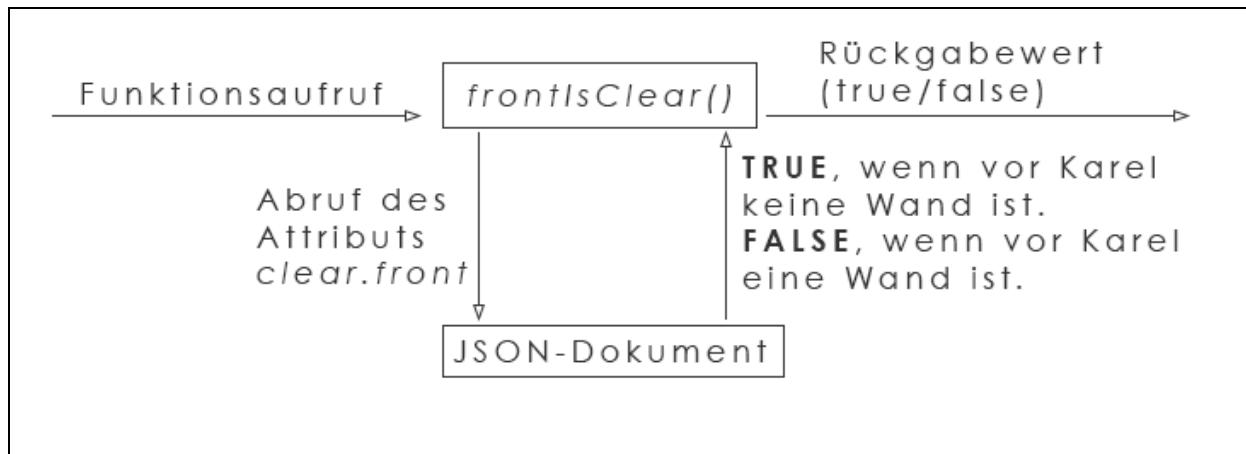


Abbildung 13: Funktionsaufruf der `frontIsClear()` Bedingung

Die Abfrage einer solchen Bedingung erfolgt entweder über eine einzelne Anfrage wie es bei `if` der Fall ist oder in Form einer Schleife, wie dies bei `for` oder `while` der Fall ist. Hierbei wird die Abfrage solange in einer Schleife wiederholt bis der Rückgabewert `FALSE` geliefert wird und die Schleife hierdurch abgebrochen wird.

3.2.4 Positionierung der Spielelemente

Wie schon im Punkt 2. Requirements erwähnt, war eine der Anforderungen eine plattformunabhängige Betrachtung zu ermöglichen. Dies bezieht sich nicht nur auf die reine Wahl des Browsers, sondern darüber hinaus auch auf das verwendete Endgerät. So verfügt ein Festrechner i.d.R. über ein größeres Display als ein Laptop; dieser wiederum über ein größeres als ein Tablet. Diese Faktoren mussten bei der Implementierung berücksichtigt werden.

Die Spielfläche selbst wird mittels Canvas „gezeichnet“. Die Canvas-Zeichenfläche bietet die Basis von Karel's Welt. Für das Leveledesign stand das Remake von „Karel the robot“ von Eric Roberts von der Stanford University aus dem Jahr 2005 Model.

Die Positionierung von Karel und den Beepern wurde mit Hilfe von Arrays realisiert. Diese zweidimensionalen Arrays entsprechen in ihrer Größe der Größe des Levels. Handelt es sich beispielsweise um ein 10x10 Spielfeld, so ist auch das Array für Karel und das der Beeper 10x10. Da sich die einzelnen Levels in ihrer Größe meist unterscheiden, ist es nötig, dass die Arrays für Karel und die Beeper dynamisch auf die Levelgröße reagieren können. Beide arbeiten ausschließlich mit booleschen Variablen⁴³, sprich die Arrays verfügen nur über Speicherinhalte, die entweder *true* oder *false* sind. Steht Karel beispielsweise am Schnittpunkt von der 3. Reihe und der 1. Spalte, so hat das Array an der Stelle [1,3] den Wert *true* gespeichert.

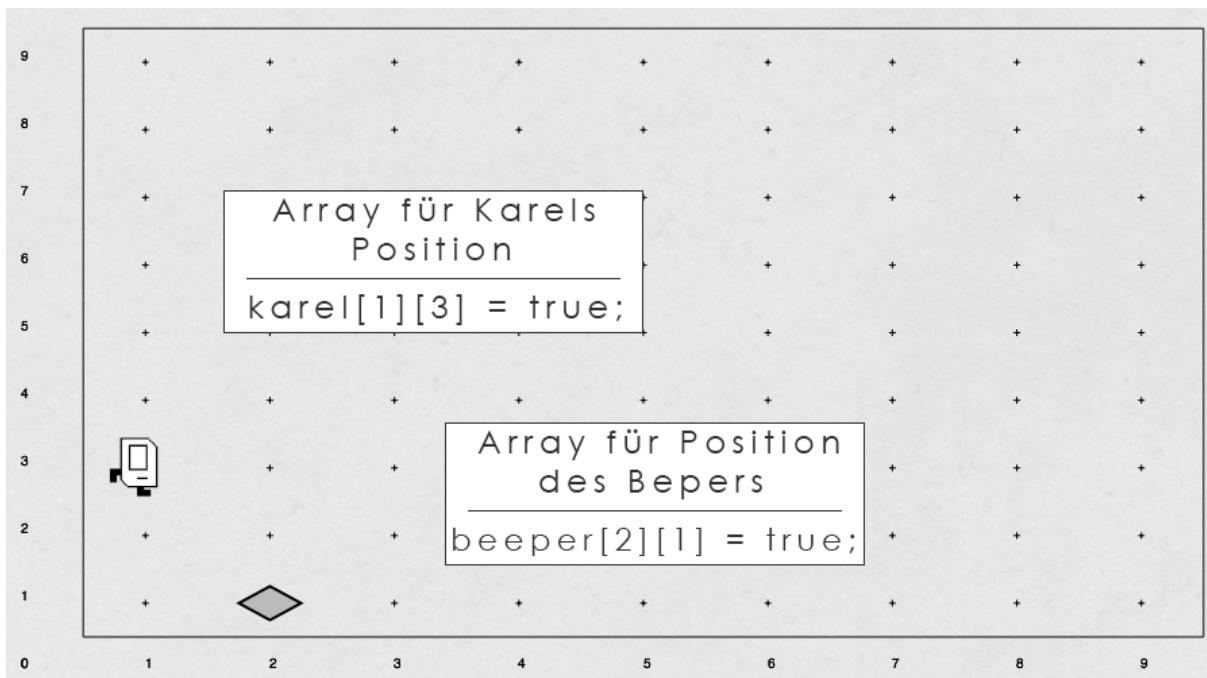


Abbildung 14: Inhalt der Arrays für Karel und Beeper.

Wie in der Abbildung gut zu erkennen, repräsentiert die erste Zahl im Array die x-Achse und die Zweite die y-Achse. Alle anderen Speicherplätze im Array sind demnach auf *false* gesetzt. Analog dazu verhält es sich für das Beeper-Array. Hier kann jedoch im Unterschied zum Positionsarray von Karel theoretisch jeder Speicherplatz zeitgleich auf *true* gesetzt sein, da mehrere

⁴³ Elemente aus der booleschen Algebra die nur zwei verschiedene Zustände annehmen können. Benannt nach George Bool.

Beeper auf dem Spielfeld erscheinen dürfen. Bei Karel besteht im Gegensatz zu den Beepern, die als Rauten mittels Canvas auf die Koordinatenkreuze gezeichnet werden, noch die Aufgabe vor bzw. nach jeder Bewegung die Blickrichtung von Karel zu eruieren. Dies liegt daran, dass Karel nicht gezeichnet wird, sondern als eines von vier möglichen PNG⁴⁴-Bildern geladen wird.

3.2.5 Wertübermittlung

Das Übertragen von Werten ist immer dann nötig, wenn wie beim Beispiel von karel-the-robot.com mehrere Einzelwebseiten miteinander verknüpft sind und diese Daten untereinander austauschen möchten. Die Wertübertragung ist ein essentielles Teilstück in der Gesamtheit der Browseranwendung Karel the robot. Da es sich bei Karel the robot um mehrere HTML-Dokumente handelt, ist die konkrete Aufgabe der Wertübertragung, die bei der Levelauswahl gewonnenen Werte zu verarbeiten und an ein weiteres HTML-Dokument zu schicken, welches die Spielelogik enthält.

Um Daten mittels JavaScript zwischen verschiedenen HTML-Dokumenten auszutauschen, gibt es diverse Methoden. Zu den bekanntesten und gebräuchlichsten zählen dabei die Übertragung mittels frames, via URL⁴⁵, über das name-Attribut des window-Objekts und über die relativ neuen Super-Cookies.⁴⁶

Wertübertragung mittels frames

Frames (dt.: Rahmen) werden in der HTML-Programmierung genutzt, um den Anzeigenbereich des Browsers in verschiedene und voneinander unabhängige Teilbereiche zu unterteilen. Die Definition aller frames

⁴⁴ Portable Network Graphics. Grafikformat für Pixelgrafiken.

⁴⁵ Uniform Resource Locator. Internetadresse, Webadresse.

⁴⁶ <http://aktuell.de.selfhtml.org/artikel/javascript/wertuebergabe/>
(Stand: 16.06.13)

bezeichnet man als *frameset*. Dies ermöglicht beispielsweise wie in der folgenden Abbildung zu sehen, eine Aufteilung des Browsers in zwei Segmente. Während das rechte Segment die eigentliche Webseite darstellt, durch die sich der Nutzer völlig frei navigieren kann, stellt der linke Teilbereich eine Navigationsleiste zur Verfügung, welche rein statisch ist und sich auch während der Navigation nicht verändert.⁴⁷

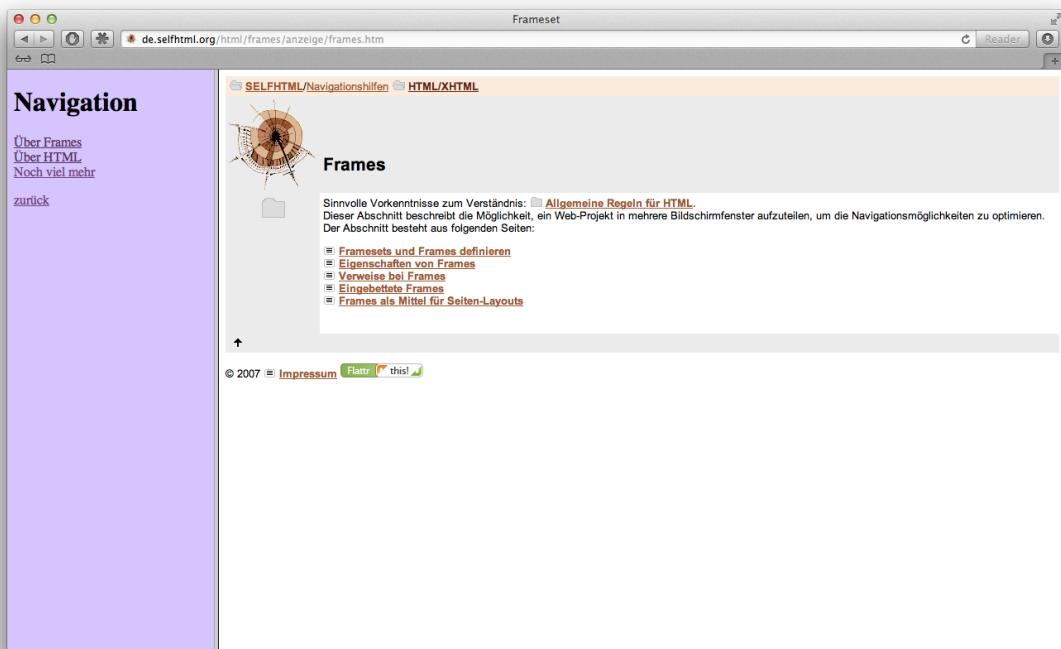


Abbildung 15: Darstellung eines framesets mit zwei frames

Um nun einen Wert zwischen zwei HTML-Seiten übertragen zu können, nutzen frames die Eigenschaft, dass sie Variablen im übergeordneten frameset-Dokument, in welchem alle frames definiert sind, speichern können. Konkret gesagt wird eine Variable, z.B. eine Benutzereingabe, gespeichert und die neue HTML-Seite geladen. Die neu geladene Seite kann sich nun auf das frameset-Dokument beziehen und die zuvor darin gespeicherte Variable abrufen und verarbeiten.⁴⁸

⁴⁷ <http://de.selfhtml.org/html/frames/definieren.htm> (Stand: 16.06.13)

⁴⁸ <http://aktuell.de.selfhtml.org/artikel/javascript/wertuebergabe/#frames> (Stand: 16.06.13)

Schon zu Zeiten von Netscape 2.0 und Internet Explorer 3.0 wurden frames verwendet. Tatsächlicher HTML-Standard wurden frames ein Jahr später 1997 durch HTML 4.0. In den HTML5-Standard wurden frames nicht mehr übernommen, da mit ihnen einige schwerwiegende Nachteile einhergehen. Nicht frame-fähige Browser waren und sind nach wie vor ein großes Problem. Da sich die Struktur bei frame-basierten Webseiten stark von denen ohne frames unterscheidet, ist es für einen nicht frame-fähigen Browser nicht möglich so eine Seite anzeigen zu können. Eine weitere Schwierigkeit besteht bei dem Verwenden von Lesezeichen. Auch wenn ein Browser frame-fähig ist, ist er oft nicht in der Lage, eine Seite, die sich in einem frame befindet, in einem Lesezeichen zu speichern. Zwei weitere Nachteile finden sich im Bereich Performance⁴⁹ und Usability⁵⁰. Frames ermöglichen es zwar mehrere Webseiten in einem Browserfenster unterzubringen, dies führt jedoch dazu dass auch mehr Daten geladen und zwischen Browser und Server ausgetauscht werden müssen. Ein letzter Nachteil ergibt sich aus der steigenden Popularität von Smartphones und Tablets. Da diese über einen weitaus kleineren Bildschirm verfügen als klassische PCs, wäre es für den Nutzer eine Zumutung, wenn der Browserinhalt auf dem kleineren Bildschirm nun mittels frames weiter unterteilt und somit verkleinert werden würde.⁵¹

Aufgrund dieser Nachteile und auch um zukunftsweisend dem HTML5-Standard zu entsprechen, wurde diese Variante nicht für diese Bachelorarbeit verwendet.

Wertübertragung mittels URL

Eine weitere Variante der Wertübertragung agiert mit Hilfe der URL. Dabei wird der Zieladresse, sprich der Adresse der zu ladenden Seite, ein sogenannter Query String angehängt. Ein Query String ist eine Zeichenkette, die zuvor abgespeicherte Variablen, in Form von Parameter/Wert-Paaren, enthält. Findet z.B. auf einer Seite „www.beispiel.de/senden.html“ eine

⁴⁹ Datenverarbeitungsgeschwindigkeit.

⁵⁰ Benutzbarkeit, Gebrauchstauglichkeit, Benutzerfreundlichkeit.

⁵¹ <http://de.selfhtml.org/html/frames/layouts.htm> (Stand: 17.06.13)

Benutzereingabe statt, bei der Vor- und Nachname sowie Alter eingegeben werden, so wird der Name kodiert am Ende der Adresszeile beginnend mit einem Fragezeichen nach dem Schema „?variable_1=wert_1&variable_2=wert_2&variable_3=wert_3“ eingetragen. Die neu geladene Seite „www.beispiel.de/empfangen.html“ ist nun in der Lage mittels der JavaScript-Eigenschaft *location.search* den Query String zu trennen und zu dekodieren.

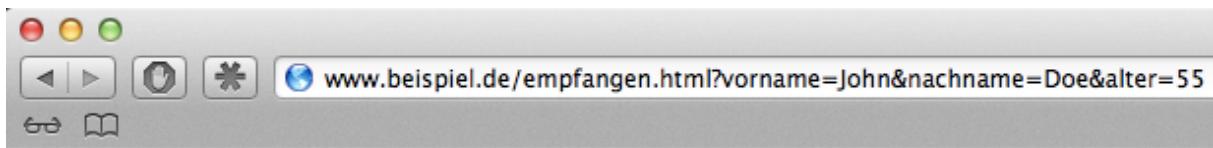


Abbildung 16: URL mit angehängtem Query String

Wie in der obigen Abbildung gut zu sehen, sind alle Variablen als zusammenhängende Zeichenkette nun in der Adresszeile, worin auch der Nachteil dieser Technologie liegt. Möchte man nicht nur einzelne Variablen, sondern ganze Objekte oder Arrays verschicken, bedarf es einer sehr aufwendigen Technik um die einzelnen Array-Elemente bzw. Objekteigenschaften in Parameter/Wert-Paare umzuwandeln. Aufgrund der Tatsache, dass ungeachtet dessen, bei jedem Aufruf einer neuen Seite, diese auch immer wieder neu vom Server angefordert werden muss, auch wenn sie schon im Cache vorliegt, wurde auch diese Variante nicht für diese Arbeit verwendet.⁵²

Wertübertragung über das name-Attribut des window-Objekts

Das Objekt *window* definiert die oberste Ebene für alles, was im Browserfenster angezeigt wird. Mit Hilfe des *window*-Objekts können Eigenschaften und Methoden des Browserfensters kontrolliert und verändert werden. Beispielsweise kann damit die Größe des Fensters definiert oder das Fenster an eine bestimmte Position auf dem Bildschirm gesetzt werden.

⁵² <http://aktuell.de.selfhtml.org/artikel/javascript/wertuebergabe/#url>
(Stand: 17.06.13)

Neben zahlreichen weiteren Eigenschaften und Methoden verfügt das window-Objekt auch das Attribut *name*. Darin wird der Name des Fensters gespeichert. Da der Name solange gespeichert bleibt wie das Fenster geöffnet ist, bietet es sich bei wechselndem Fensterinhalt an, in der *name*-Eigenschaft Variablen zu sichern und darüber auszutauschen. Um abzuspeichernde Variablen nun möglichst bequem und effektiv im *name*-Attribut zu platzieren und danach wieder auslesen zu können, kann man sich eines fertigen Skripts bedienen. Wie in dieser Arbeit der Fall, wurde für eine Wertübertragung eine externe JavaScript-Datei namens *storage.js* zur Unterstützung gewählt. Dieses JavaScript-Dokument wird in beiden Dokumenten, also dem sendenden und empfangenden HTML-Dokument, eingebunden und sorgt dafür, dass die zu speichernden Informationen in Parameter/Wert-Paaren innerhalb des *name*-Attributs gespeichert werden. Über die Funktion *getAll()* des *storage*-Dokuments kann nun die Eigenschaft *name* wieder ausgelesen werden und die gespeicherten Werte werden als Objekt zurückgegeben. Sensible Daten sollten auf diese Weise jedoch nicht übertragen werden, da die im *name*-Attribut gespeicherten Werte auch von anderen Seiten, die im gleichen Fenster geöffnet werden, ausgelesen werden könnten.⁵³

Wertübertragung mittels Super-Cookie

Eine letzte vorzustellende Möglichkeit der Wertübertragung sind sogenannte Super-Cookies. Cookies sind kleine Textdateien, die direkt im Browser gespeichert werden und diverse Informationen enthalten können. Bekannt sind Cookies z.B. beim Onlineshopping, wenn sie dafür sorgen, dass Artikel im Einkaufswagen auch nach Verlassen und erneutem Aufrufen einer Webseite immer noch vorhanden sind. Da Webseiten jedoch diese Masse an Informationen nicht clientseitig, also auf dem Heimrechner des Nutzers,

⁵³ <http://aktuell.de.selfhtml.org/artikel/javascript/wertuebergabe/#windowname> (Stand: 18.06.13)

speichern möchten, setzen diese auf Sessions⁵⁴. Hierbei wird von Seiten des Servers, also der jeweiligen Webseite, eine Session-ID in einem Cookie auf dem Client-PC gespeichert, welche dem Nutzer eine weltweit eindeutige Identität zuweist. Informationen über das Surfverhalten werden nun serverseitig gespeichert und der individuellen Session-ID zugeschrieben.

Da Cookies jedoch nicht temporär, sondern auch nach dem Ausschalten und Wiederhochfahren des Computers aktiv sind, werden so enorme Mengen über das Surfverhalten über einen langen Zeitraum gesammelt und an die Webseiten geschickt, welche einen Cookie platziert haben. Mit diesen Informationen ist es für Firmen möglich, gezielt Werbung auf dem Rechner des Nutzers zu schalten, da er für sie durch die Cookies zum gläsernen Menschen wird. Diese Tatsache möchten viele Nutzer nicht hinnehmen, was dazu führt, dass viele Internetnutzer Cookies erst gar nicht auf ihrem Rechner zulassen bzw. regelmäßig löschen.

Ein Super-Cookie ist nun im Grunde nichts anderes als eine erweiterte Form des Cookies. Die Aufgaben sind die gleichen, jedoch ist diese Cookie-Art sehr resistent. Während bei normalen Cookies das Löschen relativ einfach über den Browser gehandhabt werden kann, ist dies bei dieser speziellen Variante nicht mehr der Fall. Sie verstecken sich so gut in den Tiefen des Rechners, dass es schier unmöglich ist, einmal gesetzte Super-Cookies wieder zu löschen. Aufgrund der Eigenschaft, dass der Nutzer weder Einfluss darauf hat, ob er Super-Cookies zulassen möchte oder nicht, noch einen Überblick hat, wer welche Information von ihm bekommt, werden Super-Cookies sehr kritisch von Datenschützern und Experten angesehen.^{55 56}

⁵⁴ Sitzungen. Logische und zweitweise bestehende Verbindung zwischen Client und Server.

⁵⁵ <http://www.daserste.de/information/ratgeber-service/internet/sendung/wdr/2012/18082012-supercookies-102.html> (Stand: 18.06.13)

⁵⁶ <http://aktuell.de.selfhtml.org/artikel/javascript/wertuebergabe/#ausblick> (Stand: 18.06.13)

3.2.6 Wertspeicherung mittels IndexedDB

Eine Anforderung an das Spiel Karel the robot war die Option für den Spieler während der Laufzeit eigene Funktionen zu speichern, zu bearbeiten und bei Bedarf wieder zu löschen. Eine Möglichkeit dies dynamisch zu realisieren, ist das Speichern alterner Funktionen über das `window`-Objekt, welches wir bereits im vorherigen Punkt 3.2.5 Wertübermittlung in Verbindung mit dem `name`-Attribut kennengelernt haben.

```
window[test] = function() {alert(„Hello world!“)}
```

Im obigen Beispiel wird eine Funktion mit dem Namen „`test`“ erstellt und dem `window`-Objekt zugeschrieben. Der Funktion `test` wird eine anonyme Funktion zugewiesen, welche wiederum ein `alert`-Fenster⁵⁷ mit dem Inhalt „Hello world!“ öffnet. Diese Art der Funktionserstellung kann zu Laufzeiten geschehen. Ein großer Nachteil dieser Methode, welche es nicht ermöglichte ausschließlich diese Art der Speicherung für das Programm nutzen zu können, ist die transiente Speicherung. Zwar können die Funktionen auf diese Weise zu Laufzeiten gespeichert und gelöscht werden, jedoch kann der Browser nur solange über die vom Nutzer gespeicherten Funktionen verfügen, wie das Browserfenster nicht neu geladen oder geschlossen wird. Wird das Fenster neu geladen oder geschlossen, werden alle bisher abgespeicherten Funktionen aus dem `window`-Objekt gelöscht und sind nicht mehr für den Nutzer aufrufbar. Somit müsste der Spieler bei jedem Öffnen von `karel-the-robot.com` seine zuvor eingegebenen Funktionen erneut eingeben, was jedoch nicht im Sinne der Usability wäre.

Ziel war es nun eine persistente Speicherung zu implementieren. Hierfür wurde die zuvor beschriebene indexed Database verwendet. Die IndexedDB-Unterstützung über HTML verläuft transaktional. Gemäß der Definition ist eine

⁵⁷ Dialogfenster, welches erscheint und Text enthält.

Transaktion eine Menge von Arbeitsschritten mit einem konsistenten Ergebnis.⁵⁸ Das bedeutet, es ist nicht möglich, Befehle in der Datenbank außerhalb einer Transaktion auszuführen. In diesem Kapitel werden die Lese- und Schreibtransaktionen beschrieben.

Der Aufbau einer Datenbank in IndexedDB lässt sich wie folgt unterteilen. Eine Datenbank enthält *Object Stores*. *Object Stores* entsprechen den Datenbank-Tabellen und enthalten *Datensätze*. Ein *Datensatz* wiederum besteht aus einem sog. *Key* und einem dazugehörigen *Wert*. Der *Key* kann eine Zahl, ein String oder auch ein Array sein. Mit Hilfe des *Keys* wird der *Wert* identifiziert. Für den *Wert* selbst gelten keine Einschränkungen beim Dateityp.⁵⁹

Um nun mit IndexedDB arbeiten zu können, muss zunächst eine Datenbank angelegt werden. Nach Benennung der Datenbank wird darin ein *Object Store* angelegt. Als Argumente werden dem *Object Store* ein Name sowie ein Konfigurations-Objekt, welches dafür sorgt, dass die Datensätze mit einer fortlaufenden Nummer durchnummieriert werden, übergeben. Wurde die Datenbank erfolgreich angelegt, erfolgt ein sog. *Success-Event*, welches dem Browser mitteilt, dass der Vorgang fehlerfrei verlief und die Datenbank nun beschreibbar ist. Um nun einen gewünschten Datensatz, beispielsweise die fiktive Nutzereingabe aus dem Kapitel 3.2.5 *Wertübermittlung* einzufügen, müssen diese Daten, in Form eines JSON-Objekts, einer Variable *item* zugewiesen werden, mit welcher daraufhin eine Operation durchgeführt werden kann.⁶⁰

⁵⁸ Melzer, I.: Service-orientierte Architekturen mit Web Services. Konzepte - Standards - Praxis. 4.Aufl., Heidelberg: Spektrum 2010, S. 277.

⁵⁹ <http://www.peterkroener.de/indexed-db-die-neue-html5-datenbank-im-browser-teil-1-ein-kurzer-ueberblick/> (Stand: 19.06.13)

⁶⁰ ebd.

```

var item = {
    vorname: „John“,
    nachname: „Doe“,
    alter: 55
};

```

Da die Zahl 55 dem Datentyp number⁶¹ entspricht, sind im Gegensatz zu „John“ und „Doe“, die als Strings⁶² definiert sind, keine Anführungszeichen nötig. Das Anlegen eines neuen Datensatzes erfolgt in IndexedDB immer nach dem gleichen Schema. Zuerst wird ein Transaktions-Objekt erstellt. Dieses erhält man indem man die gewünschte Transaktion einem Object Store zuweist und eine Berechtigung (readwrite oder readonly) vergibt. Nun wird explizit der Object Store ausgewählt, in dem eine Operation durchgeführt werden soll, um nun selbige zu veranlassen. Nach erfolgter Speicheroperation erfolgt bei einer fehlerfreien Operation ein Success-Event. Nachfolgend ein Beispiel zum gespeicherten Objekt aus der Datenbank des Browsers Safari.⁶³

	key	inc	value
Datenbanken — Lokale Datei	1-0	1	{"name": "John", "nachname": "Doe", "alter": "55"}

Abbildung 17: Auszug aus der indexed Database

Die Visualisierung der Datenbank ist aufgeteilt in die Spalten key, inc und value. Key besteht aus einer „1“ und einer „0“ getrennt durch einen Bindestrich. Die erste Zahl ist eine interne Versionsnummer für die

⁶¹ Datentyp zur Darstellung von Zahlen im Bereich von 10^{-129} bis $999 \cdot 10^{125}$

⁶² Zeichenkette, die aus Ziffern, Buchstaben, Sonderzeichen oder Steuerzeichen besteht.

⁶³ <http://www.peterkroener.de/indexed-db-die-neue-html5-datenbank-im-browser-teil-1-ein-kurzer-ueberblick/>

Webapplikation. Wird die Struktur der Datenbank nachträglich noch verändert, so wird auch diese Zahl verändert. Die Zahl hinter dem Bindestrich beschreibt den eigentlichen key, sprich die Kennung des Datensatzes. Der nächste Datensatz würde dementsprechend den key „1-1“ haben, der Übernächste „1-2“ usw. *Inc* bezeichnet die Anzahl der Speichertransaktionen. Im obigen Fall wurde der erste Datensatz gespeichert, somit *inc* auf „1“ gesetzt. Nach jeder Speicherung wird dieser Wert inkrementell um eins erhöht. In der letzten Spalte, der *value*-Spalte ist das gesamte Objekt enthalten. Der Aufbau des Datensatzes wäre nun bei der Webapplikation Karel the robot der gleiche, jedoch würde sich der Wert in der *value*-Spalte etwas unterscheiden, was in einem konkreten Beispiel veranschaulicht werden soll.⁶⁴

Der Spieler schreibt eine Funktion, die Karel zwei Schritte hintereinander ausführen lässt. Diese nennt er “doubleMove”. In der indexed Database würde der *value*, sprich Wert, der gespeicherten Funktion wie folgt aussehen:

```
{"title": "doubleMove", "content": "move();move();"}
```

Title steht hierbei für den Funktionstitel und *content* für den Funktionsinhalt. In JavaScript Syntax formatiert würde die Funktion folgendes ergeben.

```
function doubleMove() {  
    move();  
    move();  
}
```

Um nun eine Funktion wieder zu löschen, kann dies entweder gezielt über den *key* erfolgen oder in iterativer Weise, bei der entweder ein Teil der Datenbank oder die komplette ausgelesen wird. Wichtig ist nur, dass die einzelnen Schritte der Transaktion eingehalten werden.

⁶⁴ ebd.

3.3 Entwicklung einer GUI

„Ohne einen Computer bedienen zu können, wird man in der neuen Informationsgesellschaft dastehen wie ein zufälliger Besucher.“⁶⁵

Die Fähigkeit einen Computer oder im Speziellen ein Programm bedienen zu können, liegt jedoch nicht allein in den Kompetenzen des Nutzers. Zwar setzt es die Bereitschaft beim Endnutzer voraus sich auf die Materie Computer und dessen Programme einzulassen, allerdings trägt der Entwickler einen entscheidenden Beitrag daran, ob der Nutzer das Programm auch verstehen kann.

3.3.1 Begriffserklärung

Das Kürzel GUI steht für **G**raphical **U**ser **I**nterface, dessen wörtliche Übersetzung grafische Benutzerschnittstelle bedeutet. Die Schnittstelle, sprich die Bedien- bzw. Benutzeroberfläche soll dem Verbraucher die Bedienung von Programmen oder kompletten Betriebssystemen erleichtern. Die grafische Benutzeroberfläche stellt geringere Anforderungen an das Lern- und Erinnerungsvermögen, sowie an den Bedienaufwand als dies bei befehlorientierten Benutzeroberflächen (CLI)⁶⁶ der Fall ist. Als Faustregel gilt: Je besser das User Interface, desto leichter gelingt der Einstieg in eine Anwendung selbst bei wenig Vorkenntnissen.⁶⁷

⁶⁵ John Naisbitt (*1930), amerik. Prognostiker

⁶⁶ Command-line interface. Nicht grafische Steuerung von Programmen mit Hilfe einer textgesteuerten Kommandozeile.

⁶⁷ <http://www.itwissen.info/definition/lexikon/graphical-user-interface-GUI-Grafische-Benutzeroberflaeche.html> (Stand: 20.06.13)

3.3.2 Softwareergonomie

Gerade in Zeiten, wo sich Technik immer rasanter entwickelt und immer komplexere Aufgaben bewältigen kann, ist es umso mehr von Bedeutung, dass sie trotz hoher Komplexität für den Menschen überschaubar und nachvollziehbar bleibt. Um dies zu gewährleisten, bedarf es einer für den Nutzer logisch aufgebauten Bedienoberfläche. Doch was bedeutet „logisch aufgebaut“? Gibt es eine Definition von logischen Bedienoberflächen und wann ist Software „gut“ und wann „schlecht“?

In den letzten 40 Jahren hat sich viel getan in der Softwareentwicklung. Als der Computer in seiner Anfangszeit nur einem Bruchteil der Menschheit zugänglich war, verlief die Mensch-Maschine-Interaktion noch rein textuell. Sollte ein Programm eine bestimmte Aktion ausführen, so schrieb der Benutzer den gewünschten Befehl in die sog. Kommandozeile (oft auch Befehlszeile, Terminal oder Konsole genannt), der die entsprechende Aktion startete. Mit der Weiterentwicklung von Computern ging auch die Verbreitung eben dieser einher. Da die klassische Kommandozeileneingabe für Menschen, die unerfahren mit der Bedienung von Computern waren, nur sehr schwer zu erlernen und wenig intuitiv war, begann man damit eine grafische Oberfläche als Alternative zu entwickeln. Damit eine GUI auch wirklich eine Erleichterung bei der Bedienung bringt, zeichneten sich nach und nach sieben empirische Gestaltungskriterien ab, die sich bei der Optimierung einer GUI als förderlich erwiesen.⁶⁸

Softwareergonomie befasst sich mit der Thematik Software anhand dieser sieben Kriterien leicht benutzbar und verständlich zu machen. In der Normenreihe DIN-EN-ISO 9241 „Ergonomie der Mensch-System-Interaktion“ Teil 110 „Grundsätze der Dialoggestaltung“ wurden diese sieben

⁶⁸ Jacko, A.; Sears, A.: The Human-Computer interaction handbook: Fundamentals, envolving technologies and emerging applications. 2.Aufl, New York: Taylor & Francis Group 2002

Gestaltungsgrundsätze als internationaler Standard festgelegt. Im Folgenden werden die Kriterien auf die Webapplikation Karel the robot angewendet.

1. Aufgabenangemessenheit:

Karel the robot ist ein Programm zur Erlernung der allgemeinen Programmiersyntax. Dabei wird berücksichtigt, dass der Spieler über wenig bis keine Kenntnisse im Bereich der Programmierung verfügt. Entsprechend können Nutzer je nach Erfahrungsstand die Spielelevels gemäß ihren Fähigkeiten auswählen. So wird gewährleistet, dass der Nutzer mit einer adäquaten Aufgabestellung rechnen kann.

2. Selbstbeschreibungsfähigkeit

Alle Schaltflächen in Karel the robot sind eindeutig beschriftet. Zudem sind wichtige Schaltflächen mit einem Piktogramm versehen, und dienen zusätzlich neben der Beschriftung zur Verdeutlichung. Gruppierte Schaltflächen haben zur Eigenbeschriftung noch eine gemeinsame Überschrift erhalten. In einer separaten Kategorie wird dem Nutzer die Möglichkeit gewährt, sich alle Steuerelemente detailliert beschreiben zu lassen.

3. Steuerbarkeit

Die Webapplikation ist zu jeder Zeit vom Nutzer frei steuerbar. Dies bedeutet, dass er zu jedem Zeitpunkt dorthin navigieren kann, wo er möchte. Des Weiteren ist der Spieler in der Lage Texteingaben beim Programmieren rückgängig zu machen oder zu verändern. Zudem kann der Nutzer das Spiel über einen Reset-Button wieder an den Anfang zurücksetzen, sowie über einen Home-Button zum Homescreen zurückkehren.

4. Erwartungskonformität

Der Nutzer erwartet bei Karel the robot ein Lernprogramm, welches ihn zunächst mit leichten Aufgaben an die Materie Programmieren

heranführt und bei Bedarf zusätzliche schwerere Aufgaben liefert. Dies wird durch die verschiedenen Schwierigkeitsstufen gewährleistet.

5. Fehlertoleranz

Das Spiel führt den geschriebenen Code nicht nur eins zu eins aus, sondern wirkt „Fehleingaben“ vom Nutzer, die zu Endlosschleifen führen, entgegen. Hierbei wird ein Hinweisfenster angezeigt, welches den Fehler kurz beschreibt. Beim Schreiben von Text hat der Spieler ebenfalls die Möglichkeit schon Geschriebenes wieder zu entfernen oder zu bearbeiten. Gleiches gilt für gespeicherte Funktionen.

6. Individualisierbarkeit

Da Karel the robot ein sehr kleines Programm ist, hält sich die Individualisierbarkeit natürlich in Grenzen. Nichtsdestotrotz hat jeder Spieler die Möglichkeit, eigene Funktionen zu schreiben und in seinem Browser zu speichern. Diese Funktionen existieren nur für den jeweiligen Nutzer, was wiederum bedeutet, dass jeder Spieler eine individuell gestaltete Applikation benutzt.

7. Lernförderlichkeit

Die Webapplikation Karel the robot unterstützt den Nutzer beim Erlernen des Programmierens durch detaillierte Informationen zu den Steuerelementen sowie mit Beispielcode. Dem Spieler ist dabei keine Vorgabe zur Reihenfolge der Informationsbeschaffung gesetzt. Des Weiteren sind die Levels so konzipiert, dass gerade Gelerntes aufeinander aufbaut und so gefestigt wird.

3.3.3 Aufbau und Komponenten

Sowohl im Aufbau als auch bei der Auswahl und Anzahl der Komponenten besticht das Spiel durch seine Einfachheit. Karel the robot besteht aus nur

zwei verschiedenen Seitendarstellungen. Beim Besuch der Seite karel-the-robot.com erblickt der Betrachter eine kurze Animation, die die Seite präsentiert. Kurz darauf gelangt er automatisch auf die Startseite. Hier stehen dem Besucher alle Möglichkeiten gebündelt in einer Menüleiste zur Verfügung. In dieser Menüleiste werden nun verschiedene Varianten zur Darstellung gebündelt. Das am meisten auf der Startseite verwendete Element ist ein sogenanntes „Modal“. Dabei wird nach Betätigung einer Schaltfläche ein event ausgelöst. Ein sog. event-listener fängt dieses Ereignis nun ab und öffnet ein Hinweisfenster, welches die angeforderte Information enthält. Zusätzlich wird der Rest des Bildschirms ausgegraunt, um den Fokus komplett auf das Hinweisfenster zu richten.

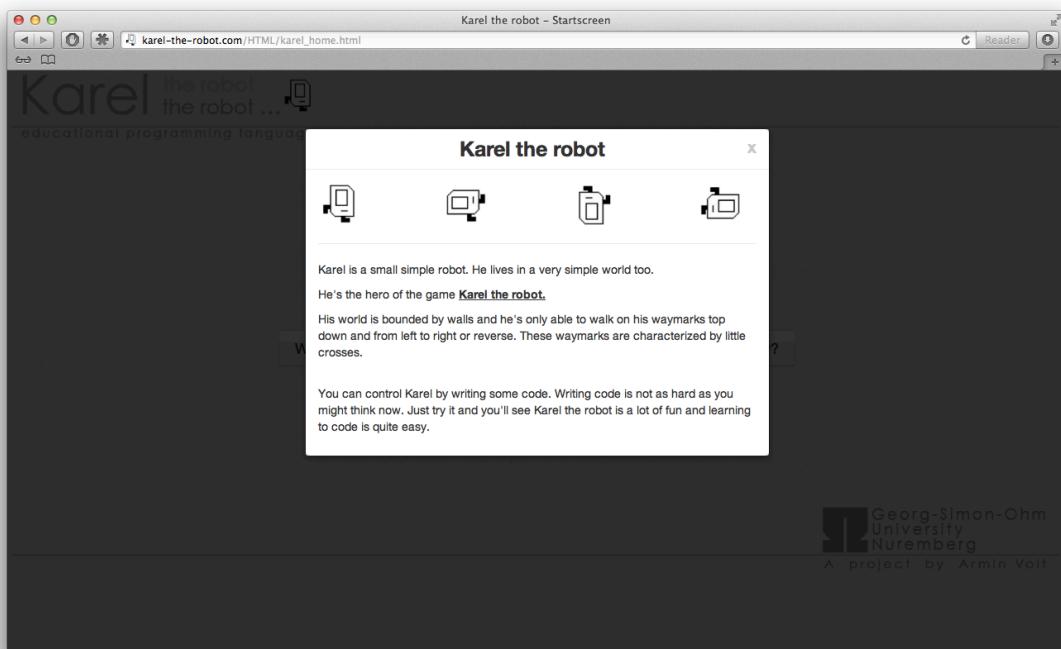


Abbildung 18: Beispiel für ein "Modal"

Für Spieler, die sich die Frage stellen, warum sie Programmieren lernen sollten, bietet eine Zitate-Slide-Show Abhilfe. Diese Slide-Show ist im Modal von „Who is Karel?“ enthalten und liefert Zitate von diversen einflussreichen Personen aus dem Bereich der Informatik und Naturwissenschaften. Der Clou

an der Sache ist, dass die Zitate dabei selbstständig durchgewechselt werden und der Spieler nichts weiter tun muss.

Eine detaillierte Beschreibung der Steuerung befindet sich im *HowTo*-Bereich. Dort sind die kompletten Spielelemente gelistet und vom eigentlichen Spiel übernommen, jedoch ohne deren Funktionsumfang. Stattdessen wird bei einem Klick auf einen beliebigen Button durch ein „Popover“ die dazugehörige Funktionsbeschreibung angezeigt. Ein Popover ist ebenfalls wie ein *Modal* ein Hinweisfenster, allerdings mit kleinen Unterschieden. Wird ein Popover aufgerufen so erscheint es, im Gegensatz zum *Modal*, welches sich bildmittig positioniert, direkt an einem zugewiesenen Element; beispielsweise an einem Button. Um die Zugehörigkeit noch zu verdeutlichen hat das Hinweisfenster noch eine Art Spitze, die in Richtung des Objekts zeigen kann. Das Ausgrauen des restlichen Bildschirms wird ebenfalls nicht verwendet.

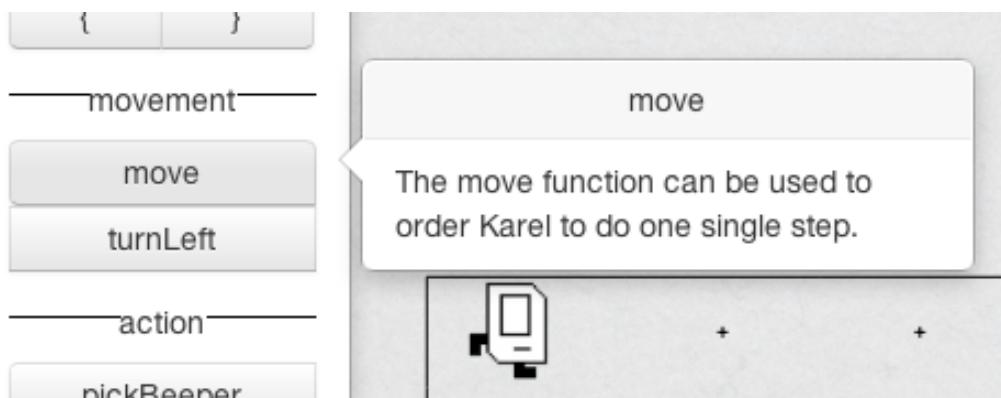


Abbildung 19: Abbildung eines Popovers

Die Levelauswahl bringt nun ein neues Element mit sich. Das Collpasselement kann z.B. einem Button zugewiesen und bei dessen Betätigung ausgelöst werden. Nach Aktivierung fährt dieses unter dem Button hervor und verschiebt den nachfolgenden Inhalt um die eigene Länge. Die eigentliche Levelauswahl ist in drei Bereiche aufgeteilt. Das linke Drittel enthält die anzuwählenden Levels. Eine Levelvorschau in Form eines Screenshots, sowie die Einstellung der Schwierigkeitsstufe werden mittig angezeigt. Auf der rechten Seite liegt der farblich hervorgehobene *Play*-Button. Zu Beginn ist

noch kein Level ausgewählt, weshalb die Levelvorschau ein unscharfes Bild eines Levels präsentiert. Auf diesem unscharfen Screenshot liegt ein gut sichtbares Fragezeichen. Dieses soll dem Spieler verdeutlichen, dass er sich noch für einen Level entscheiden muss, bevor er das Spiel antreten kann. Betätigt er dennoch den Play-Button, ohne einen Level zuvor ausgewählt zu haben, erscheint ein Popover, welches ihn auffordert, dies zu tun. Nach getätigter Levelauswahl wird der unscharfe Screenshot mit dem Fragezeichen gegen eine Vorschau vom gewählten Level ausgetauscht.

Die nachfolgende Grafik soll den Aufbau des eigentlichen Spielfelds veranschaulichen.

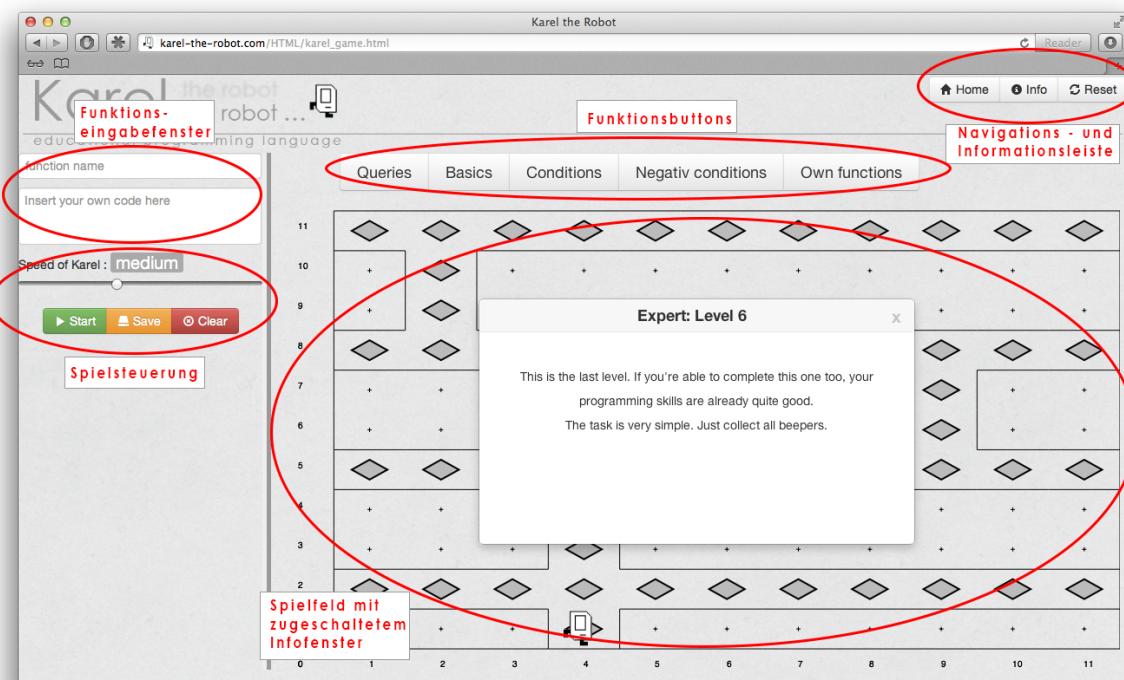


Abbildung 20: Aufbau des Spiels "Karel the robot"

Auf der linken Browserhälfte finden sich die Steuerelemente mit denen der Nutzer Karel starten lassen sowie eigene Funktionen speichern kann, die er zuvor im darüber liegenden Funktionseingabefenster eingegeben hat. Ebenfalls ist hier das Regulierungselement zur Bestimmung der Geschwindigkeit positioniert.

In der rechten oberen Ecke befindet sich die Navigations- und Informationsleiste. Mit dieser hat der Spieler die Möglichkeit das begonnene Spiel wieder zurückzusetzen, sowie komplett zur Startseite zurückzukehren. Des Weiteren kann er sich mittels des *Info*-Buttons ein Hinweisfenster zuschalten, welches ihm die Aufgabe des Levels schildert. Dieses Hinweisfenster erscheint auf dem Spielfeld, sprich auf Karel's Welt. Das Spielfeld nimmt aufgrund seiner zentralen Bedeutung den meisten Platz im Browser ein.

Oberhalb von Karel's Welt finden sich schließlich die Funktionsbuttons, mit denen der Spieler eine einfache Eingabe in das Funktionsfenster tätigen kann. Diese Funktionsbuttons sind nach Bedeutung sortiert und gruppiert, was die Übersichtlichkeit fördert.

3.4 Hauptschwierigkeiten und Lösungsstrategien bei der Entwicklung

„If builders built buildings the way programmers wrote programs, then the first woodpecker that came along would destroy civilization.“⁶⁹

Dieser doch zu tiefst ironische Satz, weißt trotz oder gerade wegen seiner nicht ganz ernst gemeinten Aussage auf einen entscheidenden Punkt bei der Softwareentwicklung hin. Fehlerfreie Software ist nach Meinung von Jerry Weinberg so gut wie unmöglich. Wahrscheinlich gibt es wenige Software-Nutzer, die bisher von sog. Bugs⁷⁰ verschont blieben. Umso wichtiger ist es, Softwarefehler schon vor der Veröffentlichung zu lokalisieren und zu korrigieren.

⁶⁹ Gerald M. Weinberg (*1933), amerik. Informatiker und Autor

⁷⁰ Programm-, Softwarefehler.

Im folgenden Kapitel soll auf Bugs aber auch auf generelle Schwierigkeiten bei der Umsetzung des Programms eingegangen und die speziellen Lösungsstrategien beschrieben werden.

3.4.1 Timing

Unter dem Begriff Timing wird hier die Fähigkeit Karel's beschrieben, sich sukzessiv auf dem Spielfeld zu bewegen und in festgelegter Geschwindigkeit seine Aufgaben zu verrichten. Diese recht einfach klingende Fähigkeit, stellte während der Programmierung jedoch ein nicht unerhebliches Problem dar. Zunächst soll geklärt werden wie es überhaupt zu diesem Problem kam.

Karel soll sich, wie schon seine Vorgänger in den Sprachen Pascal und Java, Schritt für Schritt durch seine Welt bewegen können. Soll Karel beispielsweise zwei Schritte hintereinander tätigen, so führt er zuerst die Funktion `move()` aus, wartet daraufhin einen Moment und führt anschließend seinen zweiten Schritt mittels eines erneuten Aufrufs von `move()` aus. In der Theorie scheint dieser Ablauf keineswegs kompliziert zu sein. Praktisch gesehen entstand genau an dieser Stelle ein entscheidendes Problem, welches die Realisierung der Webapplikation gefährdete.

Um den eben erwähnten Pausenmoment zwischen den beiden `move()`-Funktionen zu ermöglichen, muss der Rechner kurz mit dem Auslesen des Codes innehalten, um im Anschluss daran mit kurzer Verzögerung das Auslesen wieder fortzuführen. So würde er eine kurze Pause erreichen und Karel würde eine Bewegung nach der anderen praktizieren. Bei den meisten Programmiersprachen gibt es für diesen gewünschten Effekt eine Funktion namens `sleep()`. Wie der Name schon sagt, pausiert der Rechner eine gewünschte Zeit lang und setzt anschließend das Auslesen fort. In JavaScript hingegen gibt es keine `sleep()`-Funktion. Hier wird es dem Programmierer nicht

erlaubt, den Interpreter⁷¹ in einen Pausenmodus zu versetzen. Dies hat im konkreten Beispiel von gerade zur Folge, dass Karel nicht wie gewünscht einen Schritt macht, pausiert und einen weiteren Schritt macht. Stattdessen würde Karel von seiner Ausgangsposition mit einem Satz zwei Felder nach vorne springen ohne dazwischen eine Pause zu machen. Das Ergebnis wäre natürlich mit und ohne Pause das Gleiche, jedoch ist der sprichwörtliche Weg das Ziel.

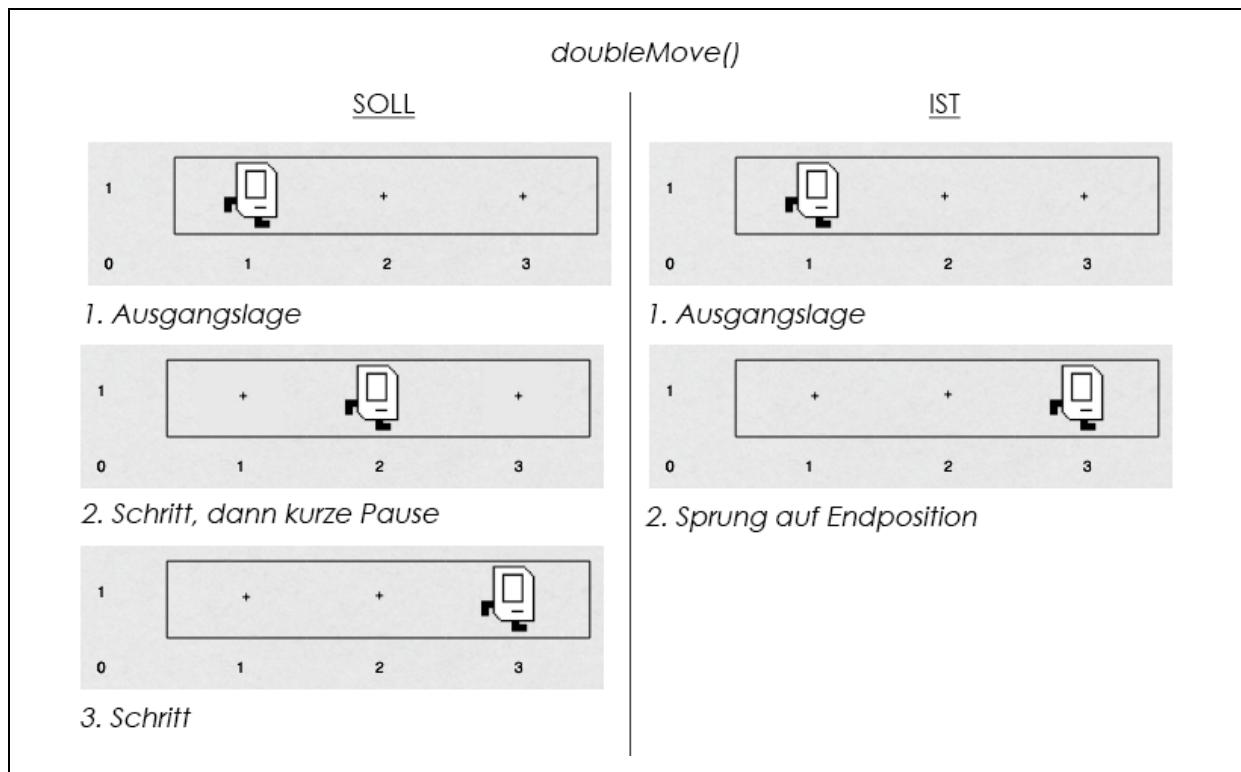


Abbildung 21: Timing-Problem ohne sleep()

Um das Fehlen der `sleep()`-Funktion zu kompensieren und das Programm zu einem Ende zu führen, wurden mehrere Ansätze ausprobiert.

Beschäftigungstaktik

Der erste Lösungsweg dieses Problem zu beheben, war der Versuch eine eigene `sleep()`-Funktion zu schreiben. Leider ist es nicht möglich, die in JavaScript schon integrierten Funktionen um eine eigene Funktion zu erweitern, welche das Programm beim Abarbeiten des Quellcodes dazu

⁷¹ Programm, das den Quellcode direkt ausliest, analysiert und ausführt.

bringt eine Pause einzulegen. Als mögliche Problemlösung wurde versucht durch das genaue Gegenteil der `sleep()`-Funktion ans Ziel zu gelangen. Statt nun wie beschrieben das Programm mitten in seinem Ablauf mit Hilfe eines `sleep()`-Befehls zum Stillstand zu bringen, wurde versucht eine Pause durch eine plötzlich eintretende Auslastung des Rechners zu erzwingen.

Hierfür wurde eine eigene Funktion namens `sleep()` geschrieben. Diese sorgt jedoch entgegen ihrem Name dafür, dass der Arbeitsspeicher auslastet wird und in einer Schleife gefangen ist. Die Funktion läuft konkret wie folgt ab. Die `sleep()`-Funktion bekommt als Argument den Parameter `milliSeconds` mitgeliefert. Dieser Parameter gibt an wie lang die Pause zwischen einzelnen Funktionen sein soll. Zu Beginn überprüft `sleep()` die aktuelle Uhrzeit mittels der Standardfunktion `getTime()`. Diese liefert die Anzahl der Millisekunden, die seit dem 01.01.1970 um 00:00 Uhr⁷² verstrichen sind zurück. Nachdem `sleep()` die aktuelle Zeit in Millisekunden ermittelt hat, addiert sie die im Argument enthaltene Zahl und setzt diese in Verbindung mit der aktuellen Zeit in eine `while`-Schleife. In der `while`-Schleife wird nun geprüft, ob die aktuelle Zeit kleiner ist als die, um den Parameter `milliSeconds` aufaddierte Zahl.

```
function sleep(milliSeconds) {  
    var startTime = new Date().getTime();  
    while( new Date().getTime() < startTime + milliseconds);  
}
```

Beträgt der Wert des Parameters `milliSeconds` z.B. „500“, so liefert die `while`-Schleife 500 Millisekunden `true`, sprich der mathematische Ausdruck ist für 500 Millisekunden wahr und das Programm bleibt in der Schleife gefangen. Da die Schleife leer ist und keine weitere Aktion eingeleitet wird, prüft das Programm nach jedem zurückgelieferten `true` den Ausdruck innerhalb der Schleife erneut. Dies führt dazu, dass der Arbeitsspeicher des Rechners völlig

⁷² Unixzeit. Beginn der Unix Zeitzählung. Heute das weitverbreitetste interne Datum von Computern und im Internet

ausgelastet wird. Zwar wird hier nur eine einfache `while`-Bedingung geprüft, zudem ohne Inhalt, aber da es nach jedem Prüfen unmittelbar zum erneuten Prüfen kommt, reicht diese quantitative Arbeit aus, den Rechner völlig zu beanspruchen. Sobald im obigen Beispiel mehr als 500 Millisekunden verstrichen sind, ist der Ausdruck in der Schleife nicht mehr korrekt, da das aktuelle Datum in Millisekunden nun größer ist als das anfängliche mit der aufaddierten Zahl. Somit ist der Ausdruck falsch, die Schleife liefert ein `false` zurück und wird abgebrochen. Erst jetzt geht der Interpreter im Code weiter und führt den nächsten Teil aus.

Bei dieser Herangehensweise kommen allerdings zwei Probleme auf. Das erste ergibt sich durch die Eingabe einer eigenen Funktion durch den Spieler. Die Funktion wird im Browser gespeichert und dann mittels der `eval()`-Funktion, welche es ermöglicht zu Laufzeit des Programms JavaScript-Code auszuwerten, ausgeführt. Hätte der Spieler nun seine Funktion `doubleMove()` abgespeichert, so müsste danach automatisch zwischen beiden `move()`-Funktionen ein `sleep()` gesetzt werden, um eine Pause zu ermöglichen. Bei einfachen Befehlen wie Aneinanderreihungen von `move()`-Funktionen wäre dies mit einer sog. String-Manipulation⁷³ relativ einfach machbar. Hierbei wird die gespeicherte Funktion in ihre Einzelfunktionen zerlegt und nach jedem Schritt ein `sleep()` eingefügt. Am Beispiel von `doubleMove()` zeigt sich, wie die Funktion nach erfolgter String-Manipulation aussieht:

```
function doubleMove() {  
    move();  
    sleep(500);  
    move();  
}
```

⁷³ Verändern einer Zeichenkette durch Löschen, Hinzufügen oder Vertauschen einzelner Zeichen.

Enthält die vom Spieler gespeicherte Funktion jedoch eine Schleifenvariante, so kommt es zu einem weiteren Problem, wenn nun die Schleife mittels String-Manipulation verändert und mit einem *sleep()* versehen wird. Als Referenz soll wieder die Funktion *doubleMove()* dienen. Diese kann man auch mit einer *for*-Schleife ausdrücken:

```
for (i=0; i<2; i++)
    move();
```

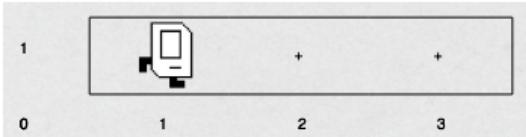
Die obige *for*-Schleife führt zweimal *move()* aus und entspricht demnach *doubleMove()*. Das Prinzip der String-Manipulation ist das Gleiche wie zuvor. Nach jedem Teilschritt wird ein *sleep()* eingefügt. Da hier nur eine Aktion enthalten ist, die dann zweimal hintereinander ausgeführt wird, wird auch nur ein *sleep()* angehängt, welches dementsprechend auch zweimal ausgeführt wird. Die modifizierte Variante würde folgendermaßen aussehen:

```
for (i=0; i<2; i++) {
    move();
    sleep(500);
}
```

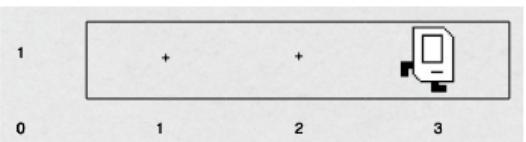
Theoretisch sollte die Schleife nun Karel einen Schritt machen lassen, darauf hin eine halbe Sekunde eine Pause einlegen und im Anschluss ein zweites Mal *move()* ausführen, gefolgt von einer weiteren Pause. In der Praxis läuft dies leider etwas anders ab.



1. Schritt:
for-Schleife wird ein erstes Mal durchlaufen.



2. Schritt:
500 Millisekunden später, iteriert das Programm ein zweites Mal durch die Schleife.



3. Schritt:
1 Sekunde nach Beginn der for-Schleife springt Karel von seiner Startposition zwei Positionen ohne Zwischenstop weiter.

Abbildung 22: Beschäftigungstaktik in der Praxis

Wie in der oberen Abbildung zu sehen, wird `sleep()` zwar ausgelesen, allerdings führt es nicht zum gewünschten Ergebnis mit einem Zwischenstopp auf Position [3,1]. Dies liegt daran, dass der Interpreter nach dem Auslesen des ersten `move()` sofort in die `sleep()`-Funktion springt ohne eine Übergabe an die `draw()`-Funktion einzuleiten. Beim zweiten Schleifendurchgang verhält es sich genau so. Auch hier übergibt er das `move()` nicht an die `draw()`-Funktion, sondern springt wieder direkt in `sleep()`. Eine Übermittlung des Status quo zwischen den Schleifendurchgängen ist dem Programm nicht möglich. Erst wenn die Schleife beendet ist, wird `move()` zweimal an `draw()` übergeben. Das hat zur Folge, dass Karel von Position [1,1] nach 1 Sekunde, dies entspricht einem zweimaligen Ausführen der `sleep()`-Funktion, auf Position [3,1] springt.

Rekursiver Lösungsansatz

Eine weitere Strategie zur Lösung des Timing-Problems war der Versuch eines rekursiven Lösungsansatzes. Unter einer rekursiven Funktion versteht man eine Funktion, die in der Lage ist, sich selbst wieder aufzurufen. Ein Beispiel für eine

solche Anwendung ist, das Ausrechnen der mathematischen Fakultät. Möchte man beispielsweise die Fakultät von 3 ausrechnen, so kann dies leicht mit einer rekursiven Funktion realisiert werden.

```
function fakultaet (zahl) {  
    if (zahl == 0)  
        return 1;  
    else  
        return (zahl * fakultaet (zahl-1))  
}
```

Beim Aufruf der Funktion *fakultaet(3)* liefert diese den Rückgabewert „*3 * fakultaet(2)*“. Daraufhin ruft sich die Funktion erneut selbst auf, was nun zu dem Rückgabewert „*3 * 2 * fakultaet(1)*“ liefert. Ein wiederholter selbständiger Aufruf gibt letztlich der Rückgabewert „*3 * 2 * 1 * fakultaet(0)*“. Nach einem letzten Aufruf liefert die Funktion *fakultaet(3)* schließlich das Endergebnis aus „*3 * 2 * 1 * 1*“ also „*6*“.

Um nun eine rekursive Funktion in Karel the robot zu integrieren, bedarf es ein wenig mehr Aufwand. Ausgehend vom Beispiel *doubleMove()* muss die vom Nutzer gespeicherte Funktion zunächst mit Hilfe der zuvor beschriebenen String-Manipulation in ihre einzelnen Bestandteile zerlegt werden. Bei der Funktion *doubleMove()* wären die beiden *move()*-Befehle die einzelnen Bestandteile. Diese Bestandteile werden nun in einem Array abgelegt. Konkret würde sich folgendes Array ergeben:

```
var befehl = new Array("move()", "move()");
```

Die Variable *befehl* erhält eine Referenz auf ein Array mit dem Befehlen der *doubleMove()*-Funktion. Als nächster Schritt wird das Array *befehl* an eine rekursive Funktion mit dem Namen *recFunc()* übergeben. Die rekursive Funktion führt zunächst mittels der *eval()*-Funktion das erste *move()* an erster

Stelle im Array aus. Anschließend wird überprüft, ob sich im Array noch mehr Befehle befinden. Um dies herauszufinden, wird eine Variable „a“ die zu Beginn auf „0“ gesetzt ist, mit der Länge des Arrays *befehl* verglichen. Solange „a“ kleiner ist als die Länge des Arrays kommt es zu einer Rekursion. Nach jedem Durchgang erhöht sich diese Variable um den Wert „1“. Wenn keine Befehle mehr im Array vorhanden sind, bricht die rekursive Funktion an dieser Stelle ab und die Variable wird wieder auf „0“ gesetzt. Wenn wie in diesem Fall noch weitere vorhanden sind, so kommt es zu einer Rekursion. Da die Rekursion jedoch nicht unmittelbar erfolgen soll, sondern nach einer kurzen Pause, behilft man sich hierbei mit der *setTimeout()*-Funktion. Diese Funktion ermöglicht es, eine andere Funktion nach einer festgelegten Zeit aufzurufen. Hierzu werden *setTimeout()* zwei Argumente übergeben; die aufzurufende Funktion und die Zeit, nach der diese aufgerufen werden soll. Nun wird der im Array an zweiter Stelle befindliche Befehl ausgeführt. Im konkreten Beispiel ist dies wiederum *move()*. Danach bricht die rekursive Funktion ab, da im Array keine weiteren Stellen belegt sind. In der Praxis würde die Funktion *recFunc()* folgendermaßen aussehen:

```
function recFunc() {
    eval(befehl[0]);
    if (a < befehl.length) {
        setTimeout( "recFunc", 500);
        a++;
    } else
        a = 0;
}
```

Auch hier liegt wieder die Situation vor, dass bei Verwendung von komplexeren Befehlen wie Schleifen Probleme auftreten. Speichert der Spieler eine Funktion mit einer *while*-Schleife ab, so kann diese nicht einfach als solche im Array abgespeichert werden. Andernfalls würde dies wieder den Effekt erzielen, dass Karel vom Anfangs- zum Endpunkt springt, da keine Pause

dazwischen wäre. Stattdessen muss die `while`-Schleife durch String-Manipulation in eine `if`-Schleife umgewandelt werden, um eine Pause dazwischen zu ermöglichen. Am Beispiel der folgenden `while`-Schleife soll dies veranschaulicht werden.

```
while (frontIsClear())
    move();
```

Die Schleife bewirkt, dass Karel sich solange schrittweise nach vorne bewegt (`move()`), wie keine Mauer vor ihm ist (`frontIsClear()`). Ersetzt man nun das „`while`“ durch ein „`if`“ und fügt es an 0. Stelle in das Array ein, so muss man zusätzlich auch noch die Funktion `recFunc()` modifizieren.

```
function recFunc() {
    eval(befehl[0]);
    if (frontIsClear() == false) {
        if (a < befehl.length) {
            setTimeout( "recFunc", 500);
            a++;
        } else
            a = 0;
    } else
        setTimeout( "recFunc", 500);
}
```

Die Zeilen mit roter Schrift stellen die zusätzlichen Zeilen Code dar, die nötig sind, damit die rekursive Funktion geht. Da in der ursprünglichen Version jede Stelle im Array nur einmal ausgelesen wird, ist es jetzt nötig, dass die 0. Stelle im Array solange wiederholt wird, bis eine Wand vor Karel erscheint. Erst dann kann die Funktion prüfen, ob weitere Stellen im Array belegt sind und diese auslesen.

Würde der Spieler nun mehrere Schleifen miteinander kombinieren, sprich eine verschachtelte Schleife erzeugen wollen, so muss die rekursive Funktion immer weiter angepasst werden. Je mehr Verschachtelungen, desto komplizierter wird es `recFunc()` an diese anzupassen. In der Praxis hat sich diese Methode als nur sehr schwer realisierbar erwiesen, weshalb sie auch nicht weiter verfolgt wurde.

Pseudofunktion

Nach Beschäftigungstaktik und rekursiver Funktion wurde eine neue Strategie zur Lösung des Timing-Problems mit Hilfe von Pseudofunktionen unternommen. Unter dem Begriff Pseudofunktion verbirgt sich eine Funktion ohne eigenen Inhalt. Eine Pseudofunktion hat lediglich die Aufgabe eine andere Funktion nach einer gewissen Zeitspanne aufzurufen. Dabei ist jede Pseudofunktion einer ganz bestimmten Funktion zugewiesen. Um eine Zugehörigkeit zu einer Funktion zu verdeutlichen, unterscheidet sich diese im Namen nur in einem Zeichen von der aufzurufenden Funktion. Folgendes Beispiel ist die Pseudofunktion, die dafür zuständig ist, die Grundfunktion `move()` zeitversetzt aufzurufen.

```
function _move() {  
    setTimeout("move()", 500);  
}
```

Auch hier wird sich wieder der Funktion `setTimeout()` bedient, um eine Verzögerung zu erreichen. Aber wie auch schon bei beiden Varianten zuvor, scheitert die Umsetzung der Pseudofunktion an den Schleifen. Der genaue Konflikt wird im nächsten Überpunkt detailliert beschrieben.

`setTimeout()`-Funktion

Die schon bekannte Funktion `setTimeout()` sollte sich als die Lösung für das Timing-Problem erweisen; jedoch nicht ohne Änderungen an der Spielelogik und der Funktion selbst vorzunehmen.

Wie schon gerade in der Funktion `_move()` zu sehen, bekam `setTimeout()` als zweites Argument immer einen konkreten Wert (hier: 500 Millisekunden) als Verzögerungszeit übergeben. Möchte man nun auf diese Weise zwei `move()` hintereinander auslösen, so benötigt man dementsprechend zwei `setTimeout()`-Funktionen. Das Problem ist nun, dass Karel's Bewegungsablauf nicht nach dem Muster „Schritt, Pause, Schritt“ verläuft, sondern er zuerst eine Pause macht und dann zwei Positionen nach vorne springt. Liest das Programm des ersten `setTimeout()`, wird festgelegt, dass `move()` in 500 Millisekunden ausgeführt wird. Unmittelbar danach, also noch vor dem ersten `move()` wird das zweite `setTimeout()` ausgeführt, welches definiert, dass das zweite `move()` nach einer halben Sekunde startet. Jedoch nicht, wie man annehmen könnte, eine halbe Sekunde nach dem Ausführen des ersten `move()`, sondern nach dem Lesen des zweiten `setTimeout()`. Da zwischen dem Lesen des ersten und zweiten `setTimeout()` nur wenige Millisekunden liegen, liegen auch nur wenige Millisekunden zwischen dem ersten und zweiten `move()`. Für den menschlichen Betrachter wirkt dies nun, als würde Karel eine Positionen überspringen. Um dieses Problem zu beseitigen, verwendet man statt eines statischen Wertes eine Variable die nach jeder Aktion erhöht wird.

```
var time = 500;  
  
setTimeout("move()", time);  
time += 500;  
setTimeout("move()", time);
```

Die Variable `time` liefert die Zeit, die `setTimeout()` wartet, bevor die darin enthaltene Funktion ausgeführt wird. Da nach dem ersten `move()` die Variable um „500“ erhöht wird, erfolgt von Seiten Karel's nach einer halben Sekunde der erste und nach einer weiteren halben Sekunde der zweite Schritt.

Die weitaus größere Herausforderung bestand darin, `setTimeout()` kompatibel zu `while`- und `for`-Schleifen zu machen.

```
while (frontIsClear())
    setTimeout("move()", time);
```

Obiges Beispiel soll Karel schrittweise durch seine Welt bewegen, solange sich keine Wand vor ihm befindet. Beim Ausführen der Funktion kommt es allerdings, anders als erhofft, entweder zu einem Absturz oder einem Einfrieren des Browsers. Bei genauerer Betrachtung wird klar wieso es dazu kommt. Zunächst prüft das Programm, ob die Bedingung `frontIsClear()` erfüllt ist. Diese gilt bekanntermaßen als erfüllt wenn vor Karel keine Wand ist. Nach erfolgreicher Prüfung startet `setTimeout()` die `move()`-Funktion mit einer halben Sekunde Verzögerung. Da sich Karel nicht unmittelbar bewegt, sondern zunächst pausiert und somit nach wie vor auf der Ausgangsposition steht, fällt die Prüfung des zweiten Schleifendurchlaufs ebenfalls positiv aus, weil sich die Bedingung `frontIsClear()` auf die aktuelle Position Karels bezieht. Ein zweites `move()` wird nun mit einer Zeitverzögerung angestoßen. Zu einer Bewegung Karels kommt es allerdings nicht mehr. Stattdessen prüft das Programm in der Zeit, in der Karel auf seiner Ausgangsposition steht, so oft hintereinander die Bedingung `frontIsClear()`, dass es sich letztlich selbst überlastet und zusammenbricht, was einen Browserabsturz nach sich zieht.

Das essentielle Problem liegt also darin, dass das Programm beim Überprüfen von Bedingungen, z.B. `frontIsClear()`, nicht die Stelle prüft, an der sich Karel nach Ablauf der Verzögerungszeit befinden würde, sondern die an der er tatsächlich zum Zeitpunkt der Überprüfung steht. Dies hat Fehlberechnungen bis hin zum Programmabsturz zur Folge.

Die Lösung für diesen Konflikt sah eine Entkopplung von Überprüfung und Zeichnung vor. Dies bedeutet, dass das Programm bei der Berechnung von

Karels Aktionen unabhängig von dessen aktuellem Standpunkt agiert. Mit folgendem Beispiel soll diese Vorgehensweise verdeutlicht werden:

Soll Karel nun mittels einer while-Schleife solange laufen, bis vor ihm eine Wand erscheint, so lautet der entsprechende Code dazu wie folgt:

```
while (frontIsClear())
    move();
```

Auf dieser Ebene wird nun kein `setTimeout()` mehr verwendet, damit Fehler und Systemabstürze vermieden werden. Die Kompetenzen von `move()` liegen jetzt ausschließlich in der Berechnung von Karels neuer Position. Nach der Berechnung jeder einzelnen Aktion werden die gewonnenen Daten mittels `setTimout()` an `draw()` weitergeleitet. Diese ist dann für die zeitverzögerte Zeichnung Karels zuständig. Die Übergabe an `draw()` gelingt dann folgendermaßen:

```
setTimeout(draw, time, x, y, "west");
```

Wie oben zu sehen, verfügt `draw()` nun über mehr Argumente (rot markiert) als bisher. Statt „`draw`“ könnte man auch die in Anführungszeichen stehende Schreibweise „`draw()`“ verwenden. Dies hat innerhalb der `setTimeout()`-Funktion keinerlei Auswirkungen, da die Funktion weiß, dass es sich beim ersten Argument um die auszuführende Funktion handelt. Die Variable `time` steht nach wie vor für die Zeit bis zur Ausführung von Karels Aktion. Die neuen Variablen „`x`“ und „`y`“ stehen für die Koordinaten, an denen Karel nach der Verzögerung gezeichnet werden soll. Das letzte Argument gibt an in welche Richtung Karel ausgerichtet ist. Die neuen Argumente haben folgende Ursache. Für den Spieler nicht sichtbar, wird im Hintergrund Karels kompletter Bewegungsablauf berechnet. Am Beispiel der zuvor erwähnten Schleife, die Karel solange laufen lässt bis eine Wand erscheint, erklärt sich dies wie folgt:

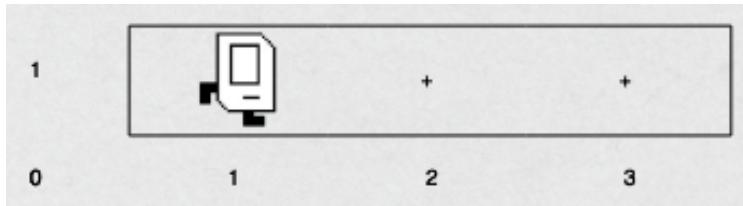


Abbildung 23: Karel's Welt als Mikrovariante

Zu Beginn steht Karel auf Position [1,1]. Das Array *karel*, welches die Position Karels innehaltet, wird während der Berechnung zunächst mit Position [2,1] und abschließend mit Position [3,1] überschrieben. Danach bricht die Schleife ab, weil Karel eine Wand erreicht hat. Dies passiert alles noch bevor Karel seinen ersten Schritt getan hat. Das bedeutet, dass Karel programmintern schon auf Position [3,1] steht, offiziell und für den Spieler sichtbar aber noch auf der Ausgangsposition. Da die *draw()*-Funktion nach jedem Schleifendurchlauf über die drei neuen Argumente die exakten Daten von Karels einzelnen Positionen zugewiesen bekommen hat, kann sie nun, unabhängig vom Array *karel*, den Bewegungsablauf nachvollziehen und rekonstruieren.

Ebenfalls notwendig für die Trennung von Berechnung und Darstellung ist ein weiteres Array für die Beeper. Zusätzlich zum bestehenden Array *beeper* wird nun ein Array namens *beeperToDraw* eingeführt. Dieses beinhaltet die Beeper, die auch für den Spieler sichtbar sind. Es wird erst im Verlauf von Karels Bewegungen und Aktionen umgeschrieben. Wohingegen *beeper* schon zur Zeit der Berechnung neu beschrieben wird. Zur Veranschaulichung soll ein Szenario dienen, bei welchem Karel mit Hilfe von Beepern den Mittelpunkt einer Reihe finden muss:

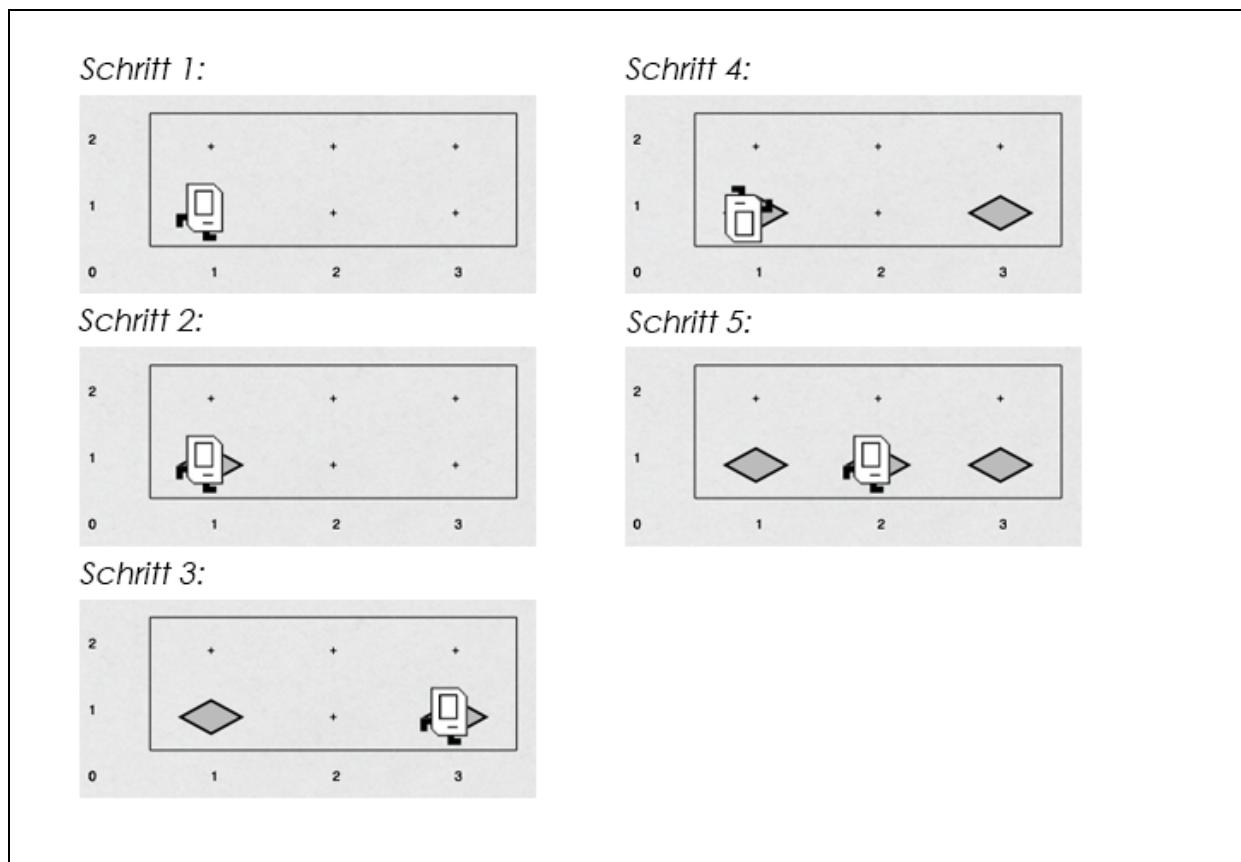


Abbildung 24: Mittelpunktsuche mit Beepern

Zunächst steht Karel auf Position [1,1]. In „Schritt 2“ legt er einen Beeper an seiner Startposition ab. Daraufhin läuft er in „Schritt 3“ bis zur anderen Seite und legt dort ebenfalls einen Beeper nieder. Da Karel nicht weiß wie groß der Level ist, tastet er sich anhand der Beeper von außen nach innen vor. In „Schritt 4“ läuft er soweit zurück bis er auf einem Beeper steht. Nun weiß Karel, dass er an dieser Stelle schon war und geht einen Schritt zurück und legt einen letzten Beeper in der Mitte der Reihe ab.

Im Array beeper waren die Beeper auf den drei Position gesetzt nachdem die Berechnung abgeschlossen war und bevor Karel eine Aktion ausgeführt hat. Nur so konnte Karel überprüfen, ob auf den Außenpositionen Beeper vorhanden sind. Wären die Positionen [1,1] und [3,1] nicht unmittelbar auf true gesetzt worden, wäre er weiter von Wand zu Wand gelaufen und wäre in einer Endlosschleife gefangen gewesen. Die Positionen in beeperToDraw() wurden erst in dem Moment in dem Karel auf der Position erscheint und einen

Beeper ablegt auf `true` gesetzt, um zu gewährleisten, dass die Beeper nicht zu früh erscheinen. Dies geschah mit Hilfe der `setTimeout()`-Funktion.

Durch diese strikte Entkopplung von Berechnung und Darstellung ist es nun möglich auch Schleifen problemlos zu verwenden und das Fehlen einer klassischen `sleep()`-Funktion, welches zu diesem Timing-Problem geführt hat, zu kompensieren.

3.4.2 Relative Darstellung

Karel the robot ist als Webapplikation konzipiert und somit für jegliche Endgeräte mit Internetzugang nutzbar. Dabei muss berücksichtigt werden, dass das Spiel nicht nur verschiedene Bildschirmgrößen, sondern auch diverse Formate bewerkstelligen muss. Eine Realisierung dieser Vorgaben war nur durch eine relative Darstellung möglich. Dies bedeutet, dass Größenangaben aller sichtbaren und nicht sichtbaren Elemente in Relation zur Bildschirmgröße gemacht werden. In der Praxis erweist sich eine solche relative Darstellung jedoch als nicht so leicht zu realisieren.

Das Spielfeld, welches den meisten Platz des Bildschirms einnimmt, wird mittels Canvas gezeichnet. Die sog. Leinwand auf der Canvas zeichnet entspricht dem Bereich in dem Karel's Welt abgebildet wird. Bei einem ähnlichen Reihen-Spalten-Verhältnis, d.h. die Anzahl der Reihen entspricht etwa der Anzahl der Spalten, erfüllt der Level so die optischen Kriterien, ansprechend und relativ symmetrisch zu sein. Bei einem starken Unterschied im Reihen-Spalten-Verhältnis, beispielsweise in einer Welt, die über zehn Spalten aber nur zwei Reihen verfügt, werden die zwei Reihen nun ebenfalls über die gesamte Darstellungsebene abgebildet. Dies hat zur Folge, das ein Schritt zur Seite 1/10 der Canvasbreite beträgt; ein Schritt in der Vertikalen jedoch die halbe Canvashöhe. Dieser sehr große Unterschied zwingt Karel bei Laufrouten in

beide Dimensionen sehr große Sprünge zu vollführen und lässt keinen ruhigen und gleichmäßigen Bewegungsablauf aufkommen.

Aus diesem Grund wurde eine Mindestanzahl an Reihen und Spalten für die Berechnung der Darstellung eingeführt, welche dafür sorgt, dass bei wenigen Reihen bzw. Spalten diese nicht über die gesamte Canvasleinwand gestreckt werden. Am Beispiel von Level 2 im fortgeschrittenen Schwierigkeitsgrad wird dies ersichtlich.

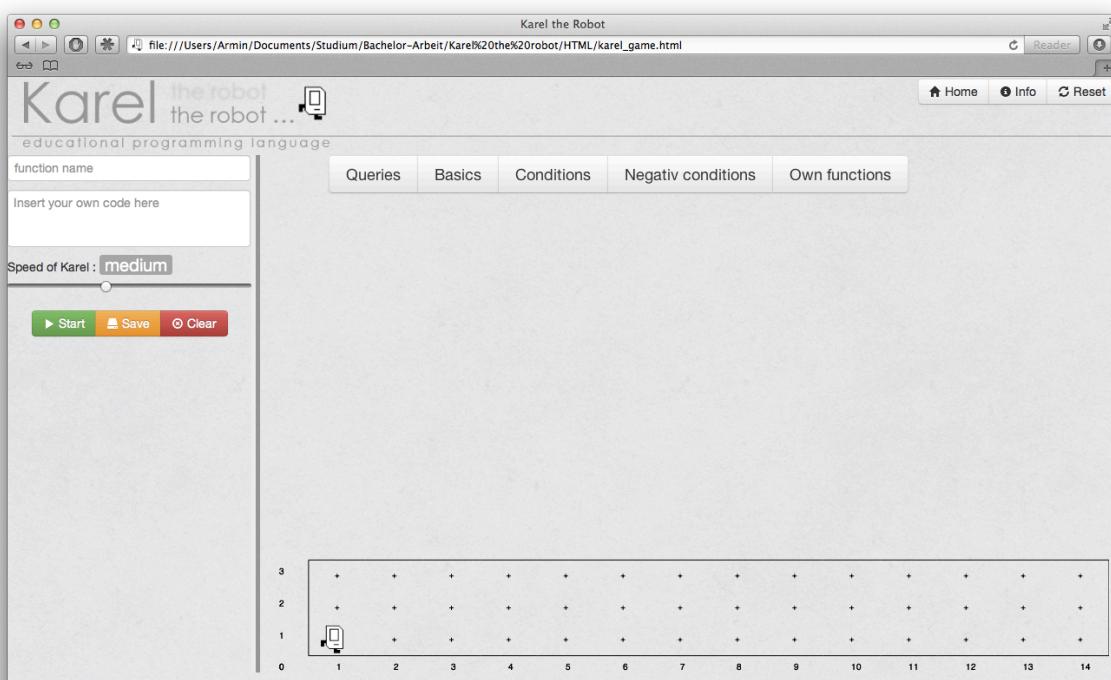


Abbildung 25: Darstellung mit großen Unterschied im Reihen-Spalten-Verhältnis

Hier stehen 14 Spalten lediglich drei Reihen gegenüber. Durch die Verwendung einer Mindestanzahl von Reihen, die hier nicht erreicht wird, wird vermieden, dass die drei Reihen bis hoch zu den Funktionsbuttons gestreckt werden und Karel in vertikaler Richtung große Sprünge absolvieren muss.

Nicht nur das Spielfeld sondern auch die Objekte, die sich darauf befinden, müssen relativ zur Bildschirmgröße dargestellt werden. Beeper und Karel müssen in einem 15x15 Spielfeld natürlich viel kleiner erscheinen als in einem

5x5 Spielfeld, da sie sonst gleich mehrere Positionen bedecken würden. Da das Spielfeld nicht nur allein durch die Bildschirmgröße, sondern auch durch die Mindestanzahl an Reihen und Spalten geregelt wird, gilt für Karel und die Beeper dementsprechend die gleiche Regel. Beispielsweise wird Karel in einer Welt mit 20 Reihen kleiner dargestellt als in einer mit zehn Reihen. Wohingegen seine Größe in einer Welt mit 3 Reihen nicht weiter zunimmt, da die Mindestanzahl von 10 Reihen bzw. Spalten unterschritten und auch das Spielfeld nicht gestreckt wird.

Zuletzt sind die Schaltflächen, sprich Buttons und die Texteingabefelder zu nennen. Hier hat eine relative Darstellung zwar den gewünschten Effekt die Elemente an den jeweiligen Bildschirm anzupassen, jedoch nicht ohne Haken. Wird nun ein Button, der auf dem PC Bildschirm gut lesbar ist, auf die Größe eines Smartphone runterskaliert, so bleibt zwar das Verhältnis zum Bildschirm gleich, allerdings leidet darunter die Lesbarkeit und somit die Bedienbarkeit. Letztlich wurde bei diesen Elementen auf eine skalierbare Größe verzichtet und ein fixes Maß eingeführt. Um nun jedoch auf unterschiedlich große Ausgabemedien reagieren zu können, wird zu Beginn, wenn der Spieler die Seite besucht, die Auflösung seines Endgerätes geprüft und in eine von drei Kategorien - groß, mittel, klein - eingeteilt. Je nachdem lädt das Programm nun eines von drei CSS-Dateien, die für die entsprechende Größe des Endgerätes eine optimierte Bildschirmdarstellung beinhaltet.

3.4.3 Datenbankverwaltung

Die Notwendigkeit einer Datenbank für die Webapplikation wurde in dieser Arbeit bereits geklärt. Die damit verbundenen Schwierigkeiten werden nun im Folgenden erläutert.

Die untergeordneten Buttons der Funktionsbuttons *Queries*, *Basics*, *Conditions* etc. sind so programmiert, dass sie bei Betätigung den ihnen zugeordneten

Wert in das Funktionseingabefenster übergeben. Soll eine eigene Funktion wieder gelöscht werden, so wird zuvor der `delete`-Button aktiviert und anschließend der entsprechende Button, der die Funktion repräsentiert, betätigt. Damit jedoch beim Anklicken der eigenen zu löschen Funktion deren Wert nicht wie gewohnt in das Funktionseingabefenster eingetragen wird, wird vor jedem Eintrag überprüft, ob der `delete`-Button aktiv ist. Ist dies der Fall, so wird kein Eintrag in das Textfeld getätigkt. Ähnlich verhält es sich mit dem `edit`-Button. Ist dieser aktiviert wird jedoch der Eintrag in das Funktionseingabefeld nicht verhindert, sondern die Funktion wird samt Funktionsname und all seinen Einzelfunktionen in das Textfeld geschrieben, um sie dort bearbeiten zu können. Nachdem die Funktion gelöscht oder bearbeitet wurde, geht die Datenbank ihren Inhalt durch, um zu prüfen, welche Datensätze nun vorhanden sind. Anhand dieser Datensätze wurde der Inhalt für das Popover, welches die eigenen Funktionen enthält, bestimmt.

Das Problem war nun, dass die Datenbank nach dem Löschen einer Funktion diese zwar löscht, jedoch nach wie vor an anderer Stelle gespeichert war. Dazu muss man verstehen woher das Popover von *Own functions* seinen Inhalt und somit die Buttons der selbst geschriebenen Funktionen bekommt.

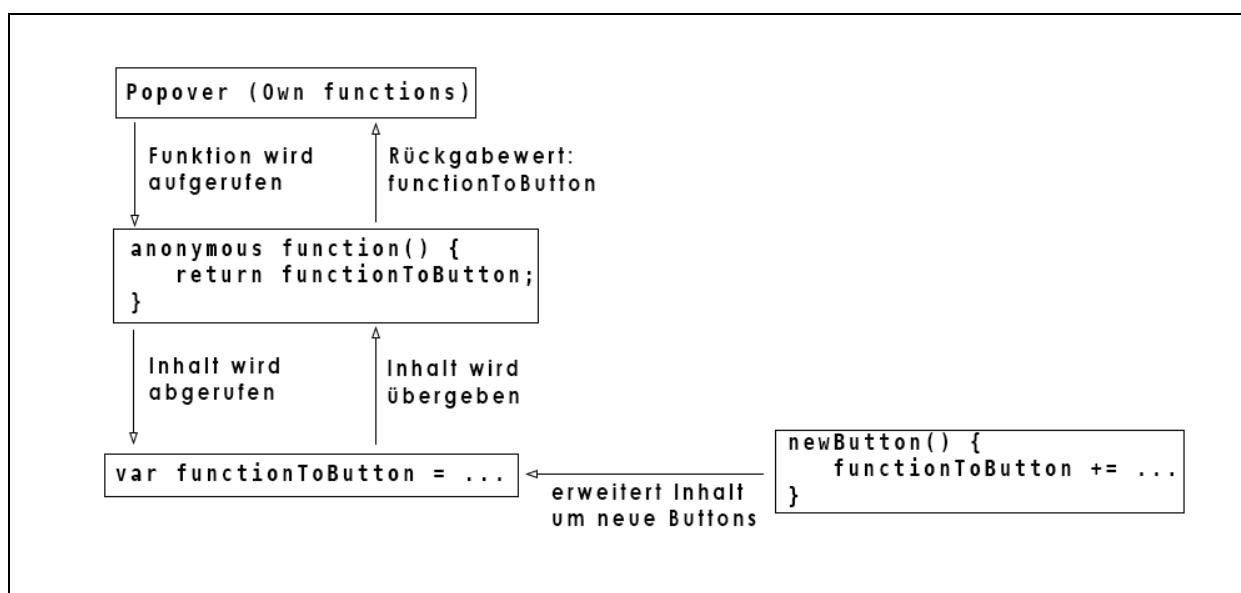


Abbildung 26: Dynamischer Inhalt des Popovers von *Own functions*

Das Popover selbst hat keinen Inhalt. Seinen Inhalt bekommt es dynamisch von einer anonymen Funktion, die wiederum die Variable `functionToButton` als Rückgabewert liefert. In dieser Variable ist der gesamte Inhalt des Popovers gespeichert. Über die Funktion `newButton` kann diese Variable für den Fall, dass eine neue Funktion vom Spieler gespeichert wird, erweitert werden.

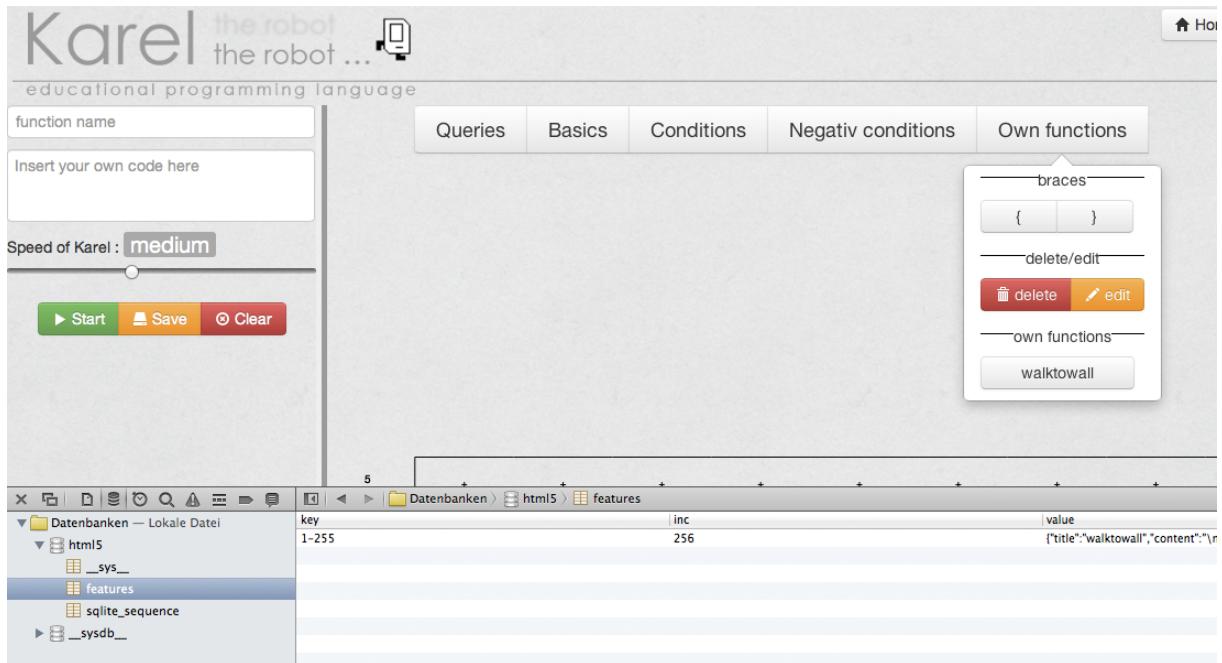


Abbildung 27: Problem beim Löscheingang (1)

Im obigen Beispiel sehen wir die eigene Funktion `walkToWall` im Popover des Buttons *Own functions* stehen. In der unteren Bildhälfte sieht man den Inhalt der angelegten Datenbank. Diese besagt, dass die Funktion mit dem key 255 (Version 1) abgespeichert wurde.

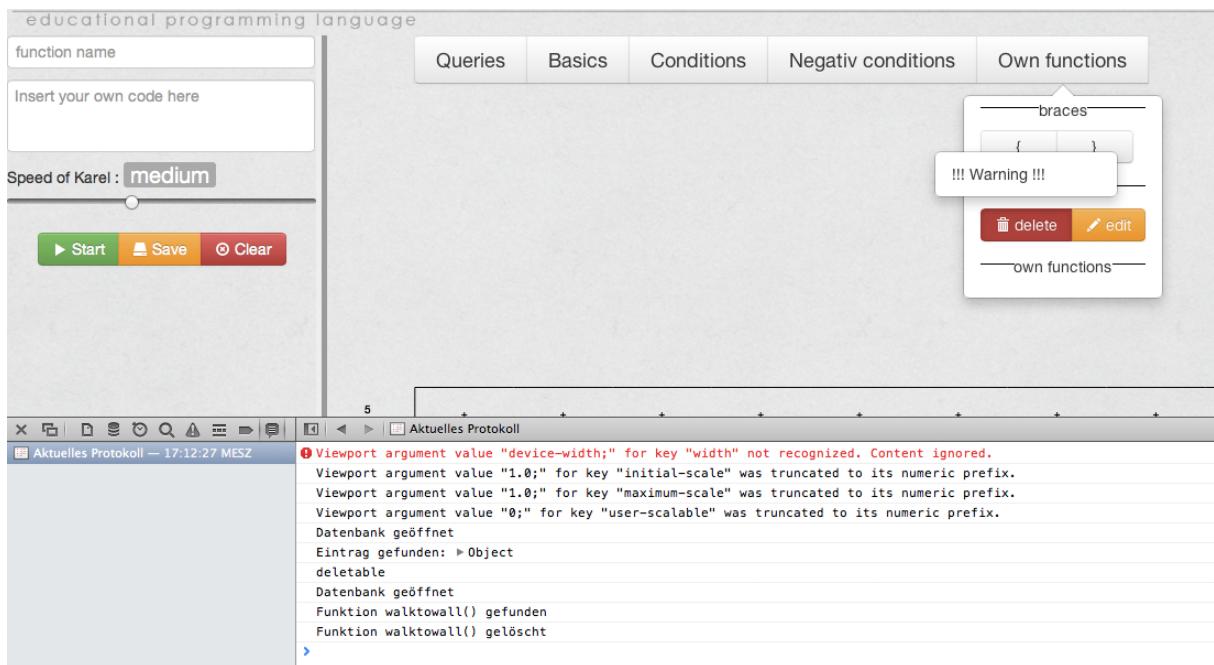


Abbildung 28: Problem beim Löschvorgang (2)

Nach Aktivierung des *delete*-Buttons und dem Erscheinen des obligatorischen Hinweisfensters, wurde die Funktion *walkToWall* gelöscht. In der Konsole in der unteren Bildhälfte ist zunächst „*deletable*“ zu lesen, was sich auf die Betätigung des *delete*-Buttons bezieht. Folgend ist zu erkennen, dass die Funktion in der Datenbank gefunden und gelöscht wurde, sowie der Button der Funktion verschwunden ist. Schloss der Spieler jedoch nun das Popover von *Own functions* und öffnete es erneut, so erschien die Funktion *walkToWall* wieder unter eigene Funktionen. Wurde die komplette Seite erneut geladen, so war die Funktion verschwunden.

Dieser bekannte und doch lange Zeit unergründliche Fehler lag an einem winzigen aber entscheidenden Detail.

Nachdem eine Funktion gelöscht wurde, wurde der Datensatz aus der Datenbank entfernt und der Inhalt der Variablen *functionToButton* um ein Attribut mittels String-Manipulation erweitert. Der Button der gelöschten Funktion wurde um die Eigenschaft „*display: none*“ ergänzt. Dies bewirkte, dass der Button daraufhin sofort aus dem Popover verschwand. Nach einem erneuten Öffnen des Popovers wurde diese Eigenschaft allerdings vom Programm ignoriert und entfernt. Dies hatte zur Folge, dass der Button wieder

sichtbar war. Warum dieses Attribut immer wieder automatisch entfernt wurde, konnte auch nach zahlreichen Überlegungen nicht in Erfahrung gebracht werden. Ließ man die Seite nun neu laden, so verschwand der Button `walkToWall`.

Die Antwort lag in der Variablen `functionToButton`. Diese wurde nach dem Speichern einer eigenen Funktion dynamisch über die Funktion `newButton()` um einen entsprechenden Button erweitert. Dies geschah allerdings nur temporär. Beim erneuten Laden der Seite wurde der Variablen wieder ihr ursprünglicher Inhalt zugeschrieben. Nachdem die Seite fertig geladen war, wurde automatisch die Funktion `newButton()` aufgerufen, welche die Datenbank auslas und die Variable um die darin gespeicherten Funktionen erweiterte. Da `walkToWall` im konkreten Fall aus der Datenbank gelöscht wurde, konnte `newButton()` diese auch nicht mehr auslesen und in die Variable einfügen. Wurde die Seite jedoch nicht neu geladen, so blieb der Button von `walkToWall` in der Variablen `functionToButton` erhalten, da er nicht explizit gelöscht wurde. Deshalb erschien der Button immer wieder, da sich das Popover immer auf die Variable bezieht.

Die Lösung war also eine zusätzliche Funktion zu schaffen, die nach dem Löschen einer Funktion die Beschreibung des Buttons aus der Variablen `functionToButton` entfernt.

3.4.4 Vermeidung von Fehleingaben

Eigene Funktionen schreiben funktioniert bei Karel the robot indem der Spieler die entsprechenden Buttons anklickt. Daraufhin erscheinen diese im Textfeld. Die Logik dahinter besagt, dass jeder Wert eines Button, der angeklickt wird, z.B. „`move();`“, am Ende des Textfeldes angereiht wird und die Gesamtfunktion so erweitert. Konkret wird das Textfeld mittels jQuery ausgelesen und um die angewählte Funktion erweitert. Dabei wird

berücksichtigt, ob es sich um *Basics*, *Queries*, *Conditions* etc. handelt. Je nachdem findet eine andere Einrückung oder ein zusätzlicher Zeilensprung statt. Diese Logik hat aber nur für den Fall funktioniert, solange der Spieler seine Funktion von vorne nach hinten durchschreibt.

```
if (beepersPresent())
    move();
```

Hätte er beispielsweise obige *if*-Abfrage um eine zusätzliche *Condition* im Laufe seiner Programmierung im Textfeld erweitern wollen, so wäre die logische Vorgehensweise die gewesen, an entsprechender Stelle mit der Maus hinzuklicken und zuerst den *Operator* auszuwählen und schließlich die *Condition*. Als Ergebnis wäre allerdings folgendes herausgekommen.

```
if (beepersPresent())
    move();
&& frontIsClear()
```

Dies lag an der starren Vorgabe des Programms alle neuen Werte ans Ende der Funktion zu schreiben. Die Lösung hierfür war, vor der automatischen Eingabe in das Textfeld zu überprüfen ob und wenn ja, wo der Spieler in das Textfeld geklickt hat. Wurde das Feld angeklickt, so wird der Wert nun an eben dieser Stelle eingefügt.

Eine weitere Unterstützung um Fehleingaben zu vermeiden wurde erreicht, indem das Klammern Setzen auf ein Minimum reduziert wird. Richtiges Klammern Setzen ist beim Programmieren eine häufige Fehlerquelle. Am Beispiel der obigen *if*-Abfrage ist zu erkennen, dass nach *beepersPresent* erst eine geöffnete dann zwei geschlossene Klammern folgen. Dies kann für einen Laien zunächst sehr verwirrend wirken. Um dies zu vermeiden, wird Klammersetzung zumindest bei den runden Klammern vom Programm übernommen. Auch wenn der Spieler eine *if*-Abfrage nachträglich um eine

weitere Condition erweitern möchte, wird die Klammerersetzung automatisch erledigt. Im Konkreten klickt der Spieler hinter die erste Condition und wählt einen Operator aus. Daraufhin wird die abschließende Klammer der if-Abfrage gelöscht und hinter der zweiten Condition wieder eingefügt, ohne dass der Spieler dadurch verwirrt wird und eine Klammer falsch setzt. Die Verwendung von Operatoren wird auch nur dann zugelassen, wenn der Spieler bereits mindestens eine Condition gesetzt hat. So wird vermieden, dass ein Programmieranfänger beispielsweise auf die Idee kommt und zwei move-Funktionen mit einem &&-Operator verknüpft.

Ein letzter aber sehr wichtiger Aspekt ist die Vermeidung von Endlosschleifen. Schleifen, die so geschrieben wurden, dass ihre Bedingung niemals *false* wird, führen beim Beispiel der Webapplikation Karel the robot dazu, dass sich der Browser aufhängt und nicht mehr bedienbar ist. Dies kann bis zum Systemabsturz führen. Da es wahrscheinlich ist, dass ein Programmierneuling sich dessen nicht bewusst ist, muss das System darauf reagieren können. Wie in den Requirements gefordert, findet vor jeder Ausführung von Code durch die eval()-Funktion eine genaue Prüfung der Quellcodes statt. Dabei wird der komplette übergebene String ausgelesen und mit im System gespeicherten Endlosschleifen verglichen. Wird beispielsweise folgender Code eingegeben,

```
while (frontIsClear())
    putBeep();
```

so erkennt das Programm die Endlosschleife und übergibt sie nicht an die eval()-Funktion, was sonst einen Browserabsturz zur Folge hätte. Stattdessen öffnet sich ein Hinweisfenster mit der Beschreibung, dass es beim Ausführen des Codes zu einer Endlosschleife kommen würde.

4. Evaluation

Nach der erfolgreichen Implementierung einer Software muss diese nun auf ihre Benutzbarkeit getestet werden; denn das intelligenteste Programm nützt nichts, wenn der Endnutzer nicht in der Lage ist es zu bedienen. Im Folgenden werden verschiedene Methoden aufgeführt, die dazu beitragen die Webapplikation Karel the robot objektiv zu testen und zu bewerten.

4.1 Validierung

Eine Validierung wird verwendet um den korrekten Aufbau von Code zu untersuchen. Dabei wird überprüft, ob der Code den Spezifikationen der jeweiligen Sprache entsprechend verfasst wurde. Kleinste Syntaxfehler können große Auswirkungen haben und sollen somit vermieden werden. Detaillierte Informationen zu den Fehlern und Warnungen sind der CD im Anhang zu entnehmen.

4.1.1 HTML

Die Validierung der HTML-Dokumente von Karel the robot wurde mit Hilfe des offiziellen Validierungsservice⁷⁴ des W3C vollzogen. Die Ergebnisse der Validierung und der daraus resultierenden Optimierung finden sich in folgender Tabelle.

⁷⁴ Der Service ist zu finden unter <http://validator.w3.org/>

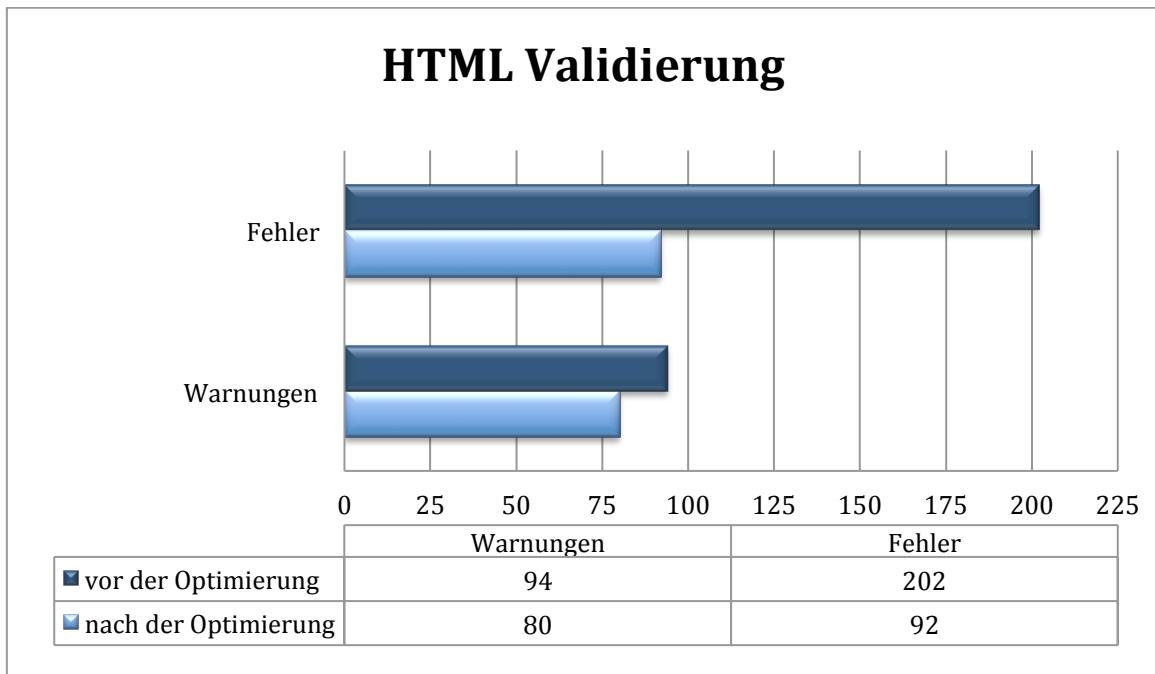


Abbildung 29: HTML Validierung vor und nach Optimierung

Wie oben zu erkennen, konnte die Fehleranzahl nach der Optimierung deutlich verringert werden. Ebenso ein Teil der Warnungen. Eine vollständige Fehlerbeseitigung ist jedoch sehr unrealistisch, da Browser beispielsweise mittels Frameworks in der Lage sind Eigenschaften darzustellen, die in der offiziellen HTML5 Syntax nicht spezifiziert sind.

4.1.2 JavaScript

Die Validierung der JavaScript Dokumente von der Applikation Karel the robot wurden mit Hilfe von JSHint⁷⁵ durchgeführt. Die Ergebnisse der Validierung und der daraus resultierenden Optimierung finden sich in folgender Tabelle.

⁷⁵ Das Tool ist zu finden unter <http://www.jshint.com/>

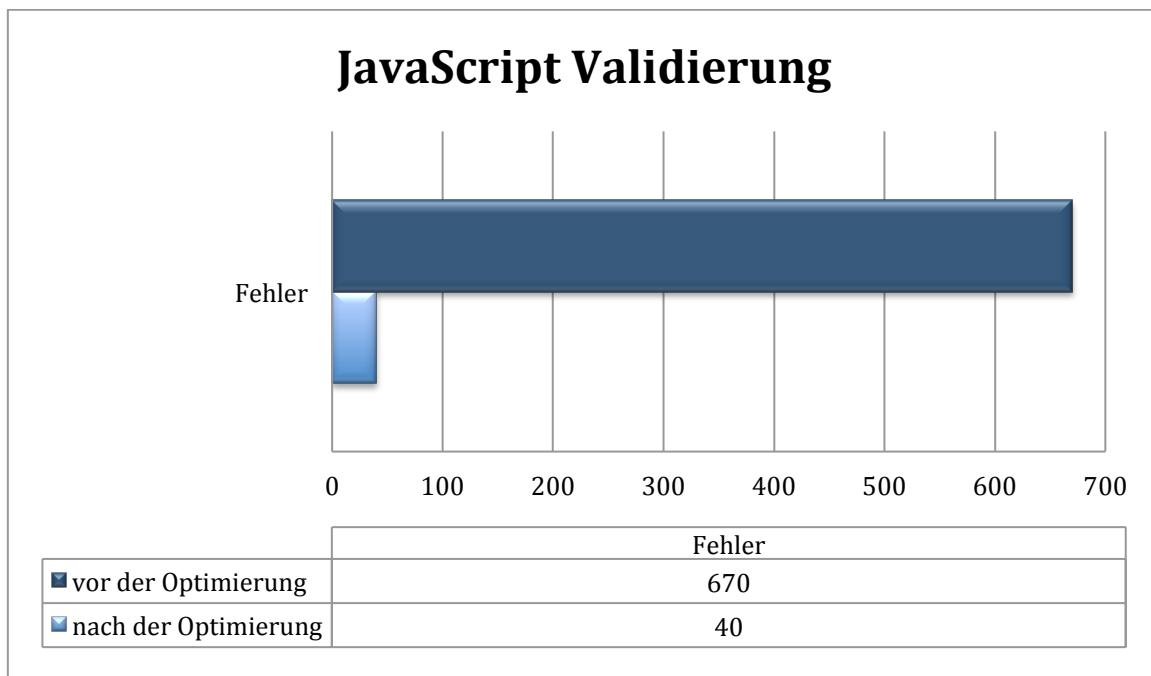


Abbildung 30: JavaScript Validierung vor und nach der Optimierung

Bei der oberen Grafik fällt sofort die enorme Menge an Syntax-Fehlern vor der Optimierung auf. Dennoch waren die aktuellen Browser in der Lage das Programm fehlerfrei wiederzugeben. Dies liegt daran, dass Browser in der Regel sehr großzügig auf Fehler reagieren. Nichtsdestotrotz sollte eine Optimierung stattfinden. Wie oben zu sehen, konnten die Fehler auf ein Bruchteil der anfänglichen reduziert werden und so die Wahrscheinlichkeit, dass es beim Browser zu Problemen bei der Interpretation kommt, drastisch reduziert werden.

4.1.3 CSS

Die Validierung der CSS-Dokumente wurde ebenfalls mit dem Validierungsservice von W3C vollzogen.

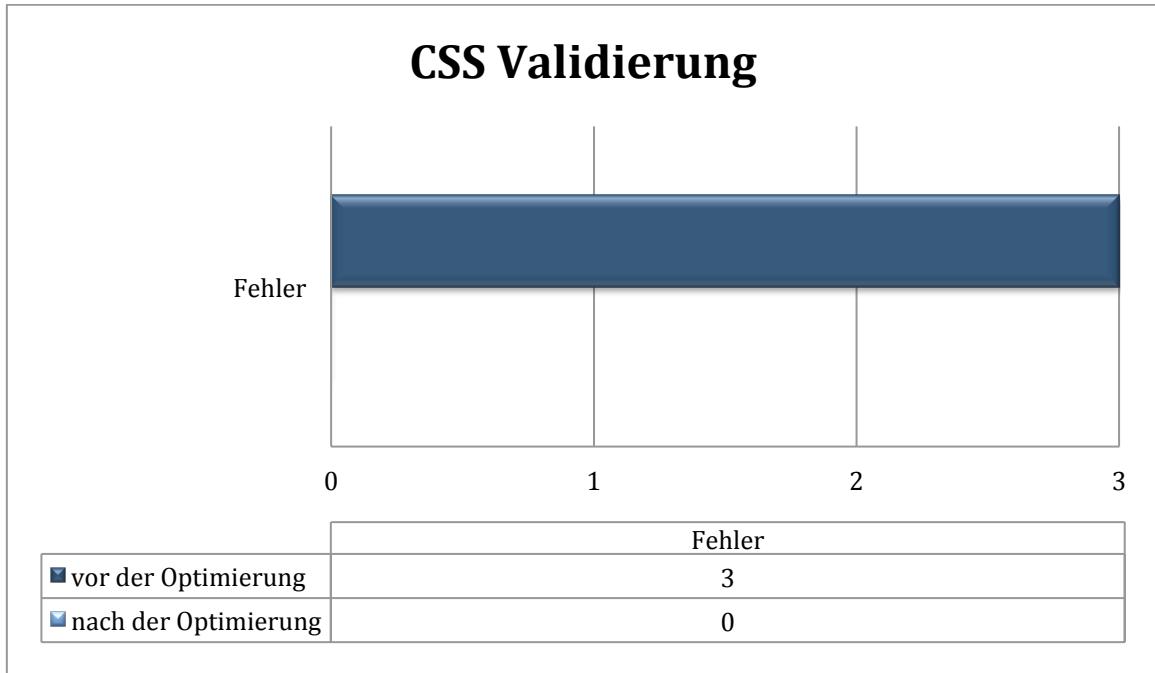


Abbildung 31: CSS Validierung vor und nach der Optimierung

Die Validierung der CSS Dateien wurde lediglich aus Gründen der Vollständigkeit durchgeführt. Hier liegen in der Regel keine Fehlerquellen.

4.2 Test Cases

Test Cases, zu Deutsch Testfälle, werden als funktionaler Softwaretest eingesetzt. Mit ihnen werden konkrete Spezifikationen der Software geprüft und so die erfüllten Requirements nachgewiesen. Um zu messen, ob ein Test Case erfolgreich abgearbeitet wurde, wird für jeden einzelnen ein erwarteter Wert, bzw. ein erwartetes Ergebnis definiert. Für die Applikation Karel the robot wurden zehn Test Cases erstellt. Die Test Cases sind durchnummieriert und mit dem Buchstabenkürzel „TC“ versehen. Die umrahmten Begriffe auf der rechten Seite der Tabelle weisen auf konkrete Buttons hin, die gedrückt werden müssen, um den Test Case erfolgreich abzuschließen.

Test Case	Erwartetes Ergebnis
TC1: Bringe in Erfahrung wer Karel the robot ist.	Who is Karel?
TC2: Finde eine detaillierte Beschreibung zur Steuerung	HowTo → Full description
TC3: Beginne das Tutorial.	HowTo → Coding Tutorial
TC4: Spiele einen Anfängerlevel.	Let's play → Beginner
TC5: Versuche während des Spielverlaufs Informationen über deine Aufgabe zu erhalten.	Info
TC6: Schreibe eine Funktion, deren Bedingung immer erfüllt ist und führe sie aus.	Endlosschleife wird nicht zugelassen, Hinweisfenster erscheint.
TC7: Speichere eine eigene Funktion.	Funktionsname, Funktionsinhalt eingeben. → Save
TC8: Editiere eine eigene Funktion.	Own functions → edit → Button eigener Funktion anklicken. → Save
TC9: Lösche eine eigene Funktion.	Own functions → delete → Button eigener Funktion anklicken.
TC10: Verändere die Spielgeschwindigkeit	Schieberegler betätigen.
TC11: Öffne Karel in verschiedenen Browsern.	Karel läuft in verschiedenen Browsern.

Die Test Cases wurden von fünf Probanden, die über wenig bis gar keine Programmiererfahrung verfügten, durchgeführt.

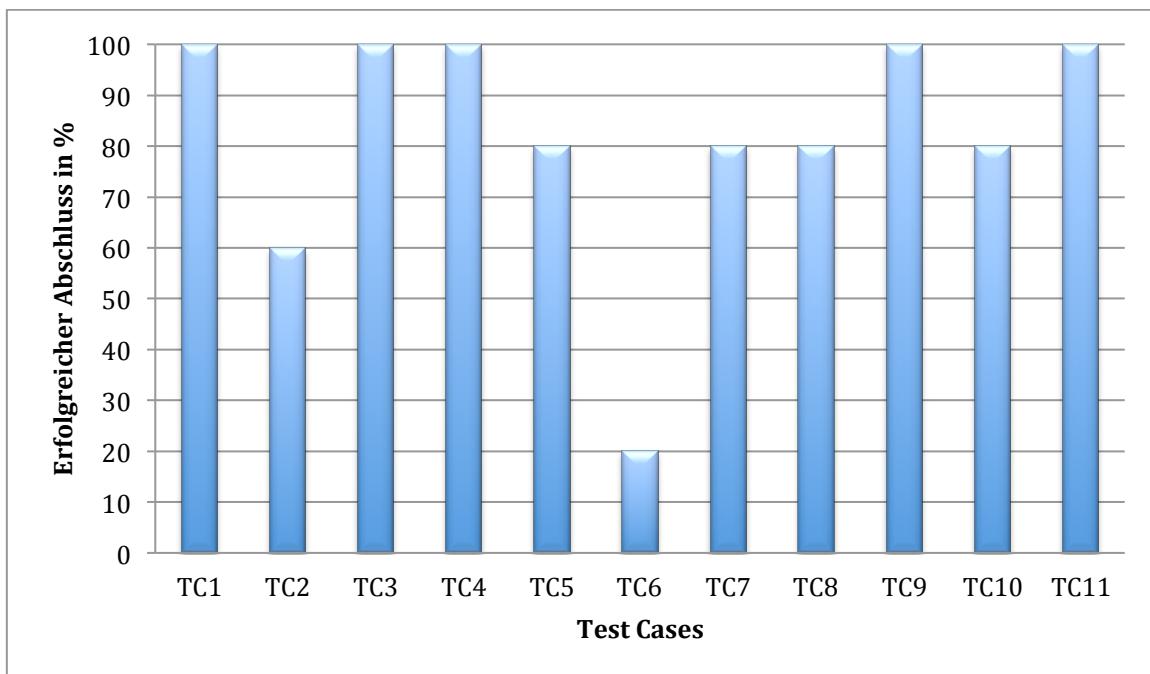


Abbildung 32: Auswertung der Test Cases

Die Auswertung der Test Cases lässt ein durchaus positives Ergebnis erkennen. Die Mehrheit der Test Cases wurde erfolgreich abgeschlossen. Lediglich zwei Ausnahmen waren zu verzeichnen. Bei Nummer 2 konnte eine Erfolgsquote von nur 60% erreicht werden, was als Konsequenz bedeutete, dass die Beschreibung neben dem Button *Full description* verständlicher gestaltet werden musste. Am schlechtesten wurde Test Case 6 gelöst. Lediglich einer Person gelang ein erfolgreicher Abschluss. Dies hatte seine Ursache darin, dass die Probanden nicht wussten, wie sie eine Endlosschleife erzeugen können. Für das Spielverständnis ist dieses Wissen jedoch nicht notwendig. Sollte ein Spieler im Verlauf des Spiels unwissentlich eine Endlosschleife erzeugen, wird er durch ein Hinweisfenster auf einen Fehler hingewiesen, weshalb es zu keinen Störungen kommt.

4.3 Traceability Matrix

Eine Traceability Matrix wird im Bereich des Softwaretestens eingesetzt. In x-Richtung verlaufen Test Cases und in y-Richtung Requirements. Mit Hilfe der

Matrix lässt sich schnell überprüfen welcher Test Case welches Requirement überprüft. So kann effektiv nachgewiesen werden, dass alle Requirements der Software überprüft wurden.

Wurde ein Requirement mit einem Test Case geprüft, so wurde die entsprechende Schnittstelle in der Tabelle eingefärbt. Die Spalte „REQs Tested“ beschreibt die Anzahl der Requirements, die insgesamt in einem Test Case getestet wurden. Die Zeile „Test Cases“ gibt an wie viele Test Cases ein Requirement überprüft haben. „REQ1“ bis „REQ11“ bezieht sich dabei auf die gleiche Nummerierung der Requirements wie in Punkt 2.3 *Funktionale Requirements*. Lediglich das Kürzel „REQ“ wurde vorangestellt. Die dunkelblauen Felder stehen für „Muss“- und die hellblauen für „Kann“-Anforderungen.

	REQs Tested	REQ1	REQ2	REQ3	REQ4	REQ5	REQ6	REQ7	REQ8	REQ9	REQ10	REQ11
Test Cases		1	1	1	1	3	1	3	2	1	0	1
TC1	1											
TC2	1											
TC3	2											
TC4	2											
TC5	1											
TC6	2											
TC7	1											
TC8	2											
TC9	1											
TC10	1											
TC11	1											

Abbildung 33: Traceability Matrix

Wie in der Abbildung zu erkennen, wurde jedes Requirement mindestens einmal getestet. Requirement 10, eine „Kann“-Anforderung, wurde nicht realisiert und ist aus diesem Grund durchgestrichen.

4.4 IsoMetrics Fragebogen

Der IsoMetrics Fragebogen wurde an der Universität Osnabrück entwickelt und ist ein Instrument zur Evaluierung von Software auf Basis der ISO-Norm 9241 Teil 110. Der gesamte IsoMetrics Fragebogen umfasst sieben Subskalen - den sieben Gestaltungskriterien entsprechend - und 75 Fragen, sogenannte „items“. Durch die Auswertung eines solchen Fragebogens können wertvolle Informationen zur Bewertung von Software, sowie möglichen Fehlfunktionen und Schwachstellen gewonnen werden.⁷⁶

Da es sich bei Karel the robot nicht um ein klassisches Softwareprodukt handelt, wie z.B. Microsoft Word, sondern um eine Spieleapplikation, waren die Fragen nicht immer passend. Aus diesem Grund wurden die items teilweise etwas abgeändert, damit sie treffender formuliert sind, jedoch galten die Original items stets als Orientierung.

Nachfolgend werden die Fragen bzw. Aussagen aufgelistet, welche von den Probanden auf einer Skala von „trifft nicht zu“ (1) bis „trifft zu“ (5) angekreuzt werden mussten.

1. Im Spiel finde ich alle Informationen, die ich benötige.
2. Ich bin mit den Übungsaufgaben überfordert.
3. Das Programm zwingt mich unnötige Schritte durchzuführen.
4. Wenn nötig, kann ich Informationen zum jeweiligen Level abrufen.
5. Die Beschriftung der einzelnen Elemente finde ich leicht verständlich.
6. Das Programm liefert mir Informationen darüber, welche Aktionen gerade zulässig sind.
7. Ich kann zu jeder Zeit des Spiels dorthin navigieren, wo ich möchte.
8. Im Spiel ist ein einfaches Navigieren zwischen den Menüpunkten möglich.

⁷⁶ <http://www.cheval-lab.ch/was-ist-usability/usabilitymethoden/frageboegen/isometrics/> (Stand:12.07.13)

9. Die Bedienmöglichkeiten des Spiels fördern eine optimale Nutzung
10. Die Ausführung einer Funktion führt immer zu einem erwarteten Ergebnis.
11. Hinweise im Spiel erscheinen immer an der gleichen Stelle.
12. Bei Eingaben, die zu Fehlern führen würden, erhalte ich immer einen Hinweis.
13. Das Spiel gibt zu spät Rückmeldung zu Fehleingaben.
14. Beim Spielen kommt es zu Systemabstürzen.
15. Es dauerte lange, bis ich die Bedienung des Spiels erlernt habe.
16. Bei Bedarf bekomme ich Hilfe zur Unterstützung beim Lernen.

Hinweis:

Die meisten Fragen sind positiv formuliert. Das bedeutet, dass der Wert 1 das schlechteste und der Wert 5 das beste Ergebnis in der Umfrage darstellt. Einige der Aussagen sind allerdings negativ gestellt. Hier bedeutet „trifft zu“ (5), das für die Software schlechtmögliche Ergebnis. Damit nun in der Grafik einheitlich der Wert 5 das bestmögliche Ergebnis darstellt, werden die Ergebnisse von negativen Formulierungen mittels der Formel

Neuer_Wert = 6 – Bewertung umgedreht.

Im Folgenden werden Auswertung und Erkenntnisse vorgestellt, die der IsoMetrics Fragebogen ergeben hat. Geprüft wurden sechs der sieben Gestaltungskriterien mit jeweils zwei bis drei Fragen. Das Kriterium Individualisierbarkeit wurde nicht berücksichtigt, da dieses, bei der Webapplikation Karel the robot nur eine untergeordnete Rolle innehat.

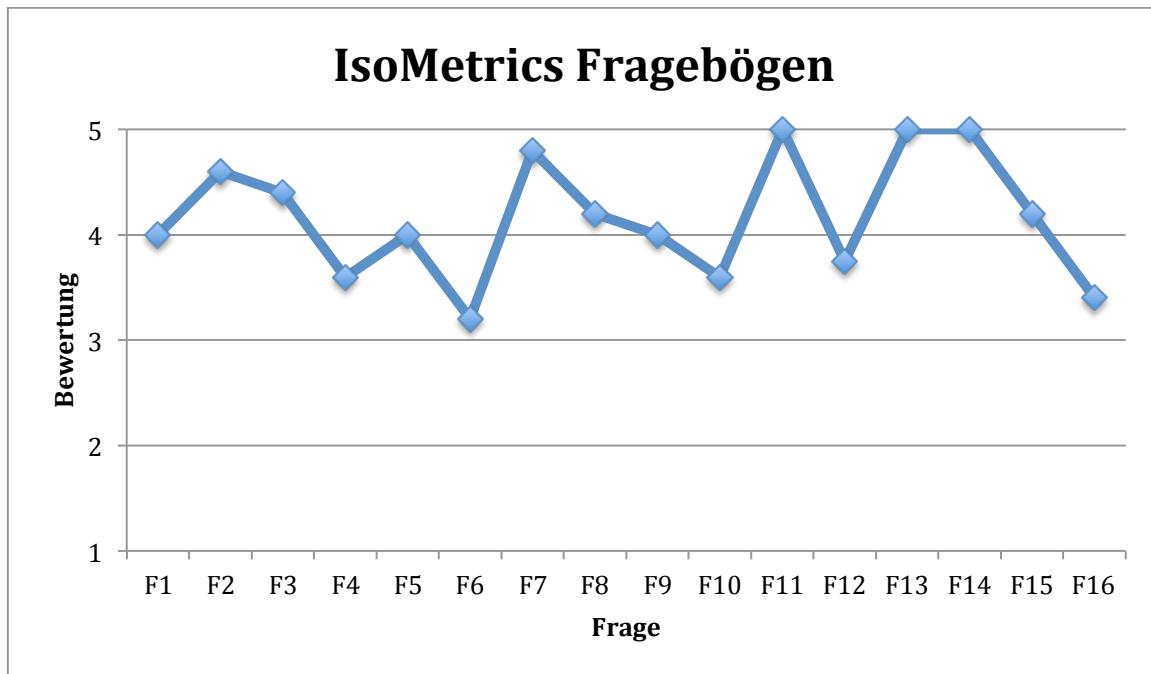


Abbildung 34: Auswertung der IsoMetrics Fragebögen

Wie oben zu erkennen, wurde bei fast allen Fragen ein Ergebnis im oberen Drittel ermittelt. Ausgehend davon, dass der Wert 5 das bestmögliche Ergebnis für die Software ist, lässt sich hiermit eine rundum positive Bilanz ziehen. Lediglich Frage F6 und F16 liegen im mittelmäßigen Bereich. Hier herrscht gegebenenfalls noch Nachholbedarf.

Zusammengefasst in die jeweiligen Gestaltungskriterien nach DIN-EN-ISO 9241 Teil 110 ergibt sich folgende Grafik.

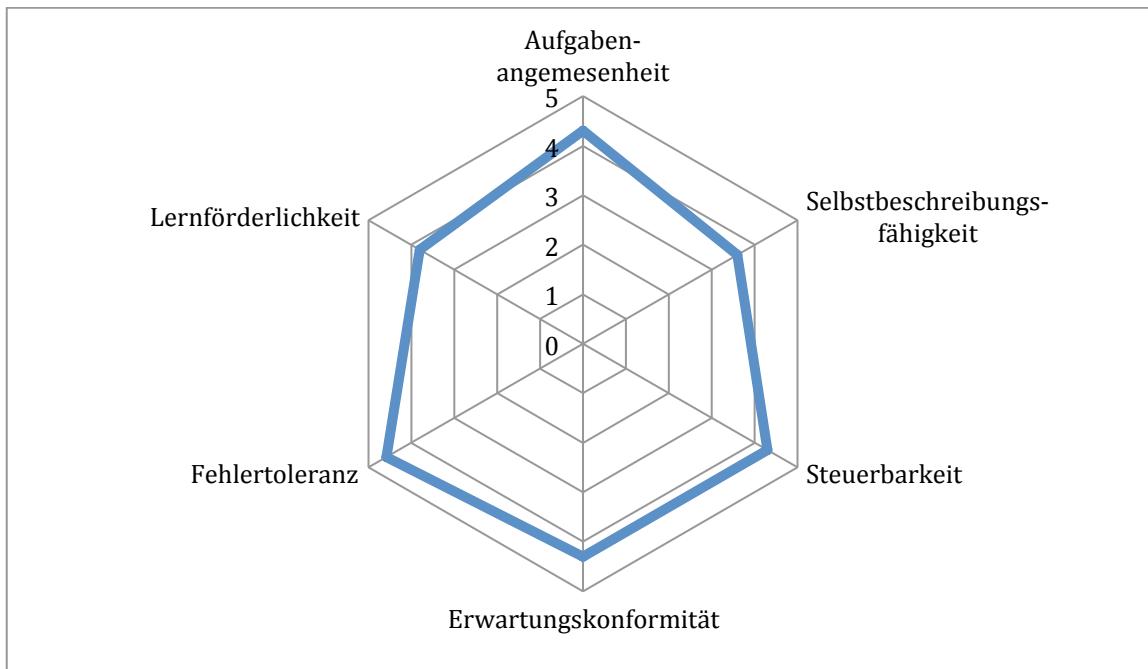


Abbildung 35: Auswertung der IsoMetrics Fragebögen gruppiert nach Gestaltungskriterien

Es ist deutlich zu erkennen, dass alle Gestaltungskriterien gut bis sehr gut umgesetzt wurden. Lediglich die „Selbstbeschreibungsfähigkeit“ bot noch Potential zur Verbesserung.

5. Zusammenfassung

Das Ergebnis dieser Bachelorarbeit ist eine vollständig implementierte und spielbare Version von dem Spieleklassiker Karel the robot. Alle „Muss“-Anforderungen wurden erfolgreich umgesetzt. Eine der zwei „Kann“-Anforderungen wurde ebenfalls realisiert. Der Grund warum nicht alle „Kann“-Anforderungen erfüllt werden konnten lag darin, dass die Entwicklung des Spiels mehr Zeit in Anspruch genommen hatte, als anfangs veranschlagt. So wurde der Puffer, der zu Beginn des Projekts, festgelegt wurde, um mögliche „Kann“-Anforderungen zu entwickeln, bereits von der Entwicklung der notwendigen Anforderungen aufgebraucht. Abschließend wird nun ein Ausblick auf eine mögliche Weiterentwicklung, sowie ein persönliches Fazit folgen.

5.1 Ausblick

Karel the robot steht als vollständige Webapplikation online unter folgender Adresse zur Verfügung.

<http://www.karel-the-robot.com/>

Alle notwendigen Elemente stehen dem Nutzer zur Verfügung. Nichtsdestotrotz gibt es noch viele Optionen, die das Spielerlebnis steigern könnten und eventuell auch der Lernförderlichkeit dienlich wären.

Ein großer Punkt, der leider nicht mehr in der vorgegebene Zeit realisiert werden konnte, ist die Einbindung eines Level Editors. Mit diesem sollte dem Spieler die Möglichkeit gegeben werden, selbstständig eigene Levels zu entwerfen und diese zu spielen. Natürlich muss es nicht bei einem Level Editor

bleiben. Eine interessante Überlegung ist sicherlich die Einbindung einer Community in das Lernprogramm Karel the robot. So könnte man beispielsweise eigene Level entwerfen, diese dann server-seitig speichern und für jeden zur Verfügung stellen.

Ebenfalls denkbar wäre auch eine Einführung von Bewertungskriterien beim Absolvieren der Levels. Bisher ist es dem Spieler nicht möglich zu erfahren, in welcher Qualität er den Level abgeschlossen hat. Hierzu könnte man die benötigte Zeit und die Anzahl der geschriebenen Zeilen Code zurate ziehen. Mittels derer könnte das Programm eine quantitative Bewertung erzeugen, die dem Spieler dann als Orientierung dient.

Die Möglichkeit einer Weiterentwicklung ist durchaus gegeben, da das gesamte Projekt ganz im Sinne des Open Source Gedankens jedem öffentlich zugänglich auf GitHub unter folgender Adresse zur Verfügung steht.

<https://github.com/arvo86/Karel-the-robot>

5.2 Persönliches Fazit

Karel the robot war nicht mein erstes Softwareprojekt im Laufe meiner Studienzeit. Zuvor konnte ich bereits Erfahrungen in der Entwicklung von Programmen in den zwei Projektarbeiten, die für den Studiengang Media Engineering vorgeschrieben sind, sammeln. Einmal in der Rolle des Mitglieds des Entwicklerteams und einmal als Projektleiter. Jedoch ein Projekt komplett alleine zu stemmen war auch für mich Neuland.

Hierbei wurde mir sehr deutlich wie komplex und teilweise schwer vorherseh- und steuerbar Softwareentwicklung sein kann. Schon zu Beginn zeigte sich, dass für die Einarbeitungszeit in die verschiedenen Themenbereiche und Entwicklungssprachen zu wenig Zeit veranschlagt wurde, weshalb die Entwicklung schon mit Verzug begann. Während der Entwicklungsphase

konnte diese Zeit aber wieder aufgeholt werden. Mitten in dieser Phase jedoch wurde das Projekt den gesamten Monat Mai unterbrochen und die Arbeit daran notgedrungen komplett eingestellt. Grund dafür war ein Umzug meinerseits, der mit seinen damit verbundenen Folgeaufgaben meine komplette Zeit forderte.

Nichtsdestotrotz habe ich dadurch die Erkenntnis gewonnen, dass ein Wiedereinstieg nach langer Pause ebenfalls unnötig viel Zeit kostet. Diesen Zeitverlust kann man also schon vorher verhindern, indem man es gar nicht erst zu langen Pausen kommen lässt und seine Aufgaben, der verfügbaren Zeit entsprechend, richtig priorisiert.

Da ich nun alle Aspekte der Softwareentwicklung am eigenen Leib durchlaufen habe, kann ich mir gut vorstellen, dies auch im Weiteren zu betreiben. Jedoch werde ich dann die Arbeit im Team bevorzugen.

5.3 Danksagung

Ein besonderer Dank gilt Prof. Dr. Ralph Lano, der mir das Thema vorgestellt und sich als Betreuer bereit erklärt hat. Auch stand er mir stets für Fragen zur Verfügung und fand auch außerhalb seiner Sprechzeiten kurzfristig Zeit für gemeinsame Projekttreffen. Weiterer Dank geht an Prof. Dr. Heinz Brünig der sich als Zweitkorrektor bereit erklärt hat. Ebenfalls bedanke ich mich bei Richard Pattis, der mir persönlich die Freigabe für die Webseite erteilt hat.

Besonderer Dank geht auch an meine Freundin Marlene, die mich stets motivierte und aufbaute, besonders wenn die Verzweiflung beim Programmieren durch unüberwindbar scheinende Probleme enorm war, sowie an meine Eltern Irene und Hermann, die mir das Studium durch finanzielle Unterstützungen erst ermöglicht haben. Ebenso wie mein Praxissemester in der bayerischen Landeshauptstadt München. Zuletzt bedanke ich mich bei meiner Schwester Sylvia, die meine Arbeit Korrektur gelesen und mich stets beraten hat.

Literaturverzeichnis

Albert, K.; Stiller, M.: Smart Mobil Apps. Berlin, Heidelberg: Springer Verlag 2012

Böringer, J.; Bühler, P.; Schlaich, P.: Kompendium der Mediengestaltung. Produktion und Technik für Digital- und Printmedien. Berlin, Heidelberg: Springer Verlag 2011

Bowers, M.; Synodinos, D.; Sumner, V.: Pro HTML5 and CSS Pattern Design. New York: Apress 2011

Brooks, D.: Guide to HTML5, JavaScript and PHP. For Scientists and Engineers. Heidelberg, New York: Springer 2011

Brooks, F.: The Mythical Man-Month: Essays on Software Engineering, Anniversary Edition. Boston: Addison-Wesley 1995

Broulik, B.: Pro jQuery Mobile. New York: Apress 2011

Burners-Lee, T.: Hypertext Markup Language – 2.0. MIT/W3C 1995

Buxmann, P.; Diefenbach, H.; Hess, T.: Die Softwareindustrie. Ökonomische Prinzipien, Strategien, Prinzipien. Berlin, Heidelberg: Springer Verlag 2008

Casario, M.; Elst, P; Brown, C.; Wormser, N.; Hanquez C.: HTML5 Solutions: Essential Techniques for HTML5 Developers. New York: Apress 2011

Dooley, J.: Software Development and Professional Software. New York: Apress 2011

Drosdowski, G.: Etymologie. Herkunftswörterbuch der deutschen Sprache; Die Geschichte der deutschen Wörter und der Fremdwörter von ihrem Ursprung bis zur Gegenwart. 7. Bd., Mannheim: Dudenverlag 1997

Franklin, J.: Beginning jQuery. New York: Apress 2013

Freeman, A.: Pro jQuery. New York: Apress 2012

Freeman, A.: The Definitive Guide to HTML5. New York: Apress 2011

Graham, W.: Beginning Facebook Game Apps Development. New York: Springer Verlag 2012

Hawkes, R.: Foundations HTML5 Canvas. New York: friendsof 2011

Jacko, A.; Sears, A.: The Human-Computer interaction handbook: Fundamentals, envolving technologies and emerging applications. 2.Aufl, New York: Taylor & Francis Group 2002

Jordan, L; Greyling, P.: Practical Android Projects. New York: Apress 2011

Keith, J.; Sambells, J.: DOM Scripting. Web Design with JavaScript and the Document Object Model. 2.Aufl., New York: Apress 2010

Koch, S.: JavaScript. Einführung, Programmierung und Referenz. Heidelberg: dpunkt.verlag 2011

Lubbers, P.; Albers, B.; Salim, F.: Pro HTML5 Programming: Powerful APIs for Richer Internet Application Development. New York: Springer Verlag 2010

MacDonald, M.: HTML5. The Missing Manual. Sebastopol: O'Reilly 2011

Melzer, I.: Service-orientierte Architekturen mit Web Services. Konzepte - Standards - Praxis. 4.Aufl., Heidelberg: Spektrum 2010

Piaget, J.; Inhelder, B.: Die Psychologie des Kindes. 9. Aufl., München: Deutscher Taschenbuch Verlag 1986

Powers, D.: Beginning CSS3. New York: Apress 2012

Preston, S.: Learn HTML5 and JavaScript for iOS. New York: Apress 2012

Richter M.; Flückiger M.: Usability Engineering Kompakt. Benutzbare Software gezielt entwickeln. 2.Aufl., Heidelberg: Spektrum 2010

Internetquellen

<http://aktuell.de.selfhtml.org/artikel/javascript/wertuebergabe/>
(Stand: 16.06.13)

<http://aktuell.de.selfhtml.org/artikel/javascript/wertuebergabe/#frames>
(Stand: 16.06.13)

<http://aktuell.de.selfhtml.org/artikel/javascript/wertuebergabe/#url>
(Stand: 17.06.13)

<http://aktuell.de.selfhtml.org/artikel/javascript/wertuebergabe/#windowname>
(Stand: 18.06.13)

<http://aktuell.de.selfhtml.org/artikel/javascript/wertuebergabe/#ausblick>
(Stand: 18.06.13)

<http://bits.blogs.nytimes.com/2011/09/14/codecademy-offers-free-coding-classes-for-aspiring-entrepreneurs/> (Stand: 01.07.13)

<http://de.selfhtml.org/html/frames/definieren.htm> (Stand: 16.06.13)

<http://de.selfhtml.org/html/frames/layouts.htm> (Stand: 17.06.13)

<http://de.wikipedia.org/wiki/Plattformunabh%E4ngigkeit> (Stand: 06.06.13)

<http://html5test.com/results/desktop.html> (Stand: 08.04.13)

<http://jigsaw.w3.org/css-validator/> (Stand: 15.07.13)

<http://kangax.github.io/es5-compat-table/#> (Stand: 11.05.13)

<http://validator.w3.org/> (Stand: 15.07.13)

http://www.bitkom.org/files/documents/BTIKOM_PK_Arbeitsmarkt_30_10_2012.pdf (Stand: 12.07.13)

<http://www.cheval-lab.ch/was-ist-usability/usabilitymethoden/frageboegen/isometrics/> (Stand: 12.07.13)

<http://www.daserste.de/information/ratgeber-service/internet/sendung/wdr/2012/18082012-supercookies-102.html> (Stand: 18.06.13)

<http://www.itwissen.info/definition/lexikon/graphical-user-interface-GUI-Grafische-Benutzeroberflaeche.html> (Stand: 20.06.13)

<http://www.jquery.com> (Stand: 12.06.13)

<http://www.jshint.com/> (Stand: 15.07.13)

<http://www.peterkroener.de/indexed-db-die-neue-html5-datenbank-im-browser-teil-1-ein-kurzer-ueberblick/> (Stand: 15.04.13)

<http://www.stanford.edu/class/cs106a/handouts/karel-the-robot-learns-java.pdf> (Stand: 15.07.13)

<http://www.w3.org/History/19921103-hypertext/hypertext/WWW/TheProject.html> (Stand: 08.04.13)

Anhang

Anhang A

IsoMetrics Fragebögen

Fragestellung (Proband 1)	stimmt nicht 1	stimmt wenig 2	stimmt mittelmäßig 3	stimmt ziemlich 4	stimmt sehr 5	Keine Angabe
Aufgabenangemessenheit						
1. Im Spiel finde ich alle Informationen, die ich benötige.				X		
2. Ich bin mit den Übungsaufgaben überfordert.		X				
3. Das Programm zwingt mich unnötige Schritte durchzuführen.		X				
Selbstbeschreibungsfähigkeit						
4. Wenn nötig, kann ich Informationen zum jeweiligen Level abrufen.			X			
5. Die Beschriftung der einzelnen Elemente finde ich leicht verständlich.				X		
6. Das Programm liefert mir Informationen darüber, welche Aktionen gerade zulässig sind.			X			
Steuerbarkeit						
7. Ich kann zu jeder Zeit des Spiels dorthin navigieren, wo ich möchte.					X	
8. Im Spiel ist ein einfaches Navigieren zwischen den Menüpunkten möglich.					X	
9. Die Bedienmöglichkeiten des Spiels fördern eine optimale Nutzung				X		
Erwartungskonformität						
10. Die Ausführung einer Funktion führt immer zu einem erwarteten Ergebnis.			X			
11. Hinweise im Spiel erscheinen immer an der gleichen Stelle.					X	
Fehlertoleranz						
12. Bei Eingaben, die zu Fehlern führen würden, erhalte ich immer einen Hinweis.				X		
13. Das Spiel gibt zu spät Rückmeldung zu Fehleingaben.	X					
14. Beim Spielen kommt es zu Systemabstürzen.	X					
Lernförderlichkeit						
15. Es dauerte lange, bis ich die Bedienung des Spiels erlernt habe.			X			
16. Bei Bedarf bekomme ich Hilfe zur Unterstützung beim Lernen.				X		

Fragestellung (Proband 2)	stimmt nicht 1	stimmt wenig 2	stimmt mittelmäßig 3	stimmt ziemlich 4	stimmt sehr 5	Keine Angabe
Aufgabenangemessenheit						
1. Im Spiel finde ich alle Informationen, die ich benötige.					X	
2. Ich bin mit den Übungsaufgaben überfordert.	X					
3. Das Programm zwingt mich unnötige Schritte durchzuführen.		X				
Selbstbeschreibungsfähigkeit						
4. Wenn nötig, kann ich Informationen zum jeweiligen Level abrufen.				X		
5. Die Beschriftung der einzelnen Elemente finde ich leicht verständlich.			X			
6. Das Programm liefert mir Informationen darüber, welche Aktionen gerade zulässig sind.		X				
Steuerbarkeit						
7. Ich kann zu jeder Zeit des Spiels dorthin navigieren, wo ich möchte.					X	
8. Im Spiel ist ein einfaches Navigieren zwischen den Menüpunkten möglich.				X		
9. Die Bedienmöglichkeiten des Spiels fördern eine optimale Nutzung				X		
Erwartungskonformität						
10. Die Ausführung einer Funktion führt immer zu einem erwarteten Ergebnis.				X		
11. Hinweise im Spiel erscheinen immer an der gleichen Stelle.					X	
Fehlertoleranz						
12. Bei Eingaben, die zu Fehlern führen würden, erhalte ich immer einen Hinweis.						X
13. Das Spiel gibt zu spät Rückmeldung zu Fehleingaben.	X					
14. Beim Spielen kommt es zu Systemabstürzen.	X					
Lernförderlichkeit						
15. Es dauerte lange, bis ich die Bedienung des Spiels erlernt habe.	X					
16. Bei Bedarf bekomme ich Hilfe zur Unterstützung beim Lernen.			X			

Fragestellung (Proband 3)	stimmt nicht 1	stimmt wenig 2	stimmt mittelmäßig 3	stimmt ziemlich 4	stimmt sehr 5	Keine Angabe
Aufgabenangemessenheit						
1. Im Spiel finde ich alle Informationen, die ich benötige.			X			
2. Ich bin mit den Übungsaufgaben überfordert.	X					
3. Das Programm zwingt mich unnötige Schritte durchzuführen.		X				
Selbstbeschreibungsfähigkeit						
4. Wenn nötig, kann ich Informationen zum jeweiligen Level abrufen.				X		
5. Die Beschriftung der einzelnen Elemente finde ich leicht verständlich.				X		
6. Das Programm liefert mir Informationen darüber, welche Aktionen gerade zulässig sind.				X		
Steuerbarkeit						
7. Ich kann zu jeder Zeit des Spiels dorthin navigieren, wo ich möchte.				X		
8. Im Spiel ist ein einfaches Navigieren zwischen den Menüpunkten möglich.				X		
9. Die Bedienmöglichkeiten des Spiels fördern eine optimale Nutzung				X		
Erwartungskonformität						
10. Die Ausführung einer Funktion führt immer zu einem erwarteten Ergebnis.					X	
11. Hinweise im Spiel erscheinen immer an der gleichen Stelle.					X	
Fehlertoleranz						
12. Bei Eingaben, die zu Fehlern führen würden, erhalte ich immer einen Hinweis.			X			
13. Das Spiel gibt zu spät Rückmeldung zu Fehleingaben.	X					
14. Beim Spielen kommt es zu Systemabstürzen.	X					
Lernförderlichkeit						
15. Es dauerte lange, bis ich die Bedienung des Spiels erlernt habe.		X				
16. Bei Bedarf bekomme ich Hilfe zur Unterstützung beim Lernen.			X			

Fragestellung (Proband 4)	stimmt nicht 1	stimmt wenig 2	stimmt mittelmäßig 3	stimmt ziemlich 4	stimmt sehr 5	Keine Angabe
Aufgabenangemessenheit						
1. Im Spiel finde ich alle Informationen, die ich benötige.				X		
2. Ich bin mit den Übungsaufgaben überfordert.		X				
3. Das Programm zwingt mich unnötige Schritte durchzuführen.	X					
Selbstbeschreibungsfähigkeit						
4. Wenn nötig, kann ich Informationen zum jeweiligen Level abrufen.				X		
5. Die Beschriftung der einzelnen Elemente finde ich leicht verständlich.				X		
6. Das Programm liefert mir Informationen darüber, welche Aktionen gerade zulässig sind.			X			
Steuerbarkeit						
7. Ich kann zu jeder Zeit des Spiels dorthin navigieren, wo ich möchte.					X	
8. Im Spiel ist ein einfaches Navigieren zwischen den Menüpunkten möglich.				X		
9. Die Bedienmöglichkeiten des Spiels fördern eine optimale Nutzung				X		
Erwartungskonformität						
10. Die Ausführung einer Funktion führt immer zu einem erwarteten Ergebnis.				X		
11. Hinweise im Spiel erscheinen immer an der gleichen Stelle.					X	
Fehlertoleranz						
12. Bei Eingaben, die zu Fehlern führen würden, erhalte ich immer einen Hinweis.			X			
13. Das Spiel gibt zu spät Rückmeldung zu Fehleingaben.	X					
14. Beim Spielen kommt es zu Systemabstürzen.	X					
Lernförderlichkeit						
15. Es dauerte lange, bis ich die Bedienung des Spiels erlernt habe.		X				
16. Bei Bedarf bekomme ich Hilfe zur Unterstützung beim Lernen.			X			

Fragestellung (Proband 5)	stimmt nicht 1	stimmt wenig 2	stimmt mittelmäßig 3	stimmt ziemlich 4	stimmt sehr 5	Keine Angabe
Aufgabenangemessenheit						
1. Im Spiel finde ich alle Informationen, die ich benötige.				X		
2. Ich bin mit den Übungsaufgaben überfordert.	X					
3. Das Programm zwingt mich unnötige Schritte durchzuführen.	X					
Selbstbeschreibungsfähigkeit						
4. Wenn nötig, kann ich Informationen zum jeweiligen Level abrufen.			X			
5. Die Beschriftung der einzelnen Elemente finde ich leicht verständlich.					X	
6. Das Programm liefert mir Informationen darüber, welche Aktionen gerade zulässig sind.				X		
Steuerbarkeit						
7. Ich kann zu jeder Zeit des Spiels dorthin navigieren, wo ich möchte.					X	
8. Im Spiel ist ein einfaches Navigieren zwischen den Menüpunkten möglich.				X		
9. Die Bedienmöglichkeiten des Spiels fördern eine optimale Nutzung						X
Erwartungskonformität						
10. Die Ausführung einer Funktion führt immer zu einem erwarteten Ergebnis.				X		
11. Hinweise im Spiel erscheinen immer an der gleichen Stelle.					X	
Fehlertoleranz						
12. Bei Eingaben, die zu Fehlern führen würden, erhalte ich immer einen Hinweis.					X	
13. Das Spiel gibt zu spät Rückmeldung zu Fehleingaben.	X					
14. Beim Spielen kommt es zu Systemabstürzen.	X					
Lernförderlichkeit						
15. Es dauerte lange, bis ich die Bedienung des Spiels erlernt habe.	X					
16. Bei Bedarf bekomme ich Hilfe zur Unterstützung beim Lernen.				X		

Anhang B

CD

Auf der CD befindet sich die Bachelorarbeit im *.docx und *.pdf Format. Alle hier aufgeführten Grafiken, sowie die der Webapplikation sind ebenfalls auf der CD enthalten. Des Weiteren ist der komplette Quellcode enthalten. Die genaue Struktur ist der README-Datei zu entnehmen.