FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

QRBASIC

Visualisierung einfacher Programmierkonzepte

MASTERARBEIT

zur Erlangung des akademischen Grades

Magister der Sozial- und Wirtschaftswissenschaften

im Rahmen des Studiums

Informatikmanagement

eingereicht von

Thomas Cap BSc MA
Matrikelnummer 0026356

an der Fakultät für Informatik
der Technischen Universität Wien

Betreuung: Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Peter Purgathofer

Wien, 5. November 2015

Thomas Cap	Peter Purgathofer

FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

QRBASIC

Visualizing Basic Programming Concepts

MASTER'S THESIS

submitted in partial fulfillment of the requirements for the degree of

Magister der Sozial- und Wirtschaftswissenschaften

in

Computer Science Management

by

Thomas Cap BSc MA
Registration Number 0026356

to the Faculty of Informatics
at the Vienna University of Technology

Advisor: Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Peter Purgathofer

Vienna, 5th November, 2015

Thomas Cap

Peter Purgathofer

Erklärung zur Verfassung der Arbeit

Thomas Cap BSc MA
Ruthnergasse 133, 1210 Wien

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

Wien, 5. November 2015

Thomas Cap

Danksagung

Es ist meine tief sitzende Überzeugung, dass ich besonders einem Menschen niemals oft und ausdrücklich genug danken werde können, egal wie viele Chancen sich mir dafür noch bieten mögen: meiner Mutter. Eine Kämpfernatur, die konsequent ihren eigenen Wert unterschätzt und herunterspielt, die es wirklich nie leicht im Leben hatte und trotz zahlreicher Enttäuschungen und Rückschläge immer Zeit und ein offenes Ohr für ihre Mitmenschen und vor allem ihre manchmal sehr anstrengenden Söhne hatte. Danke!

Bedanken möchte ich mich auch bei meinen Kollegen der flave GmbH. Die Doppelbelastung Vollzeitjob und Studienabschluss bekamen auch sie gelegentlich zu spüren. Danke Stefan, Gerhard & "Cheese"! Ein ganz besonders großes Dankeschön gilt aber vor allem Ing. "Super-Peter" Lampesberger, der einem absoluten Android- & Java-Neuling durch seine erste "App" geholfen hat.

Es mag etwas ungewöhnlich sein, aber ich möchte diese Arbeit auch den zahllosen "Namenlosen" widmen, die auch im Rahmen von wissenschaftlichen Arbeiten – bei denen man es mit Quellenangaben normalerweise sehr genau nimmt – meist ungenannt bleiben (müssen). Dazu zählen vor allem die zahlreichen und oft anonymen Entwickler & Designer freier und/oder kostenloser Software, Schriftarten und Vorlagen, aber auch Internetposter und Tutorialschreiber, die mir so manches Mal weitergeholfen haben, wenn mich ein Android-, PHP- oder LaTeX-Problem ratlos machte.

Schlussendlich möchte ich mich natürlich noch bei meinem Betreuer Peter Purgathofer bedanken. Welcher viel beschäftigte Universitätsprofessor gibt sich schon freiwillig mit einem Langzeitstudenten eines auslaufenden Studiums ab, der einfach eines Tages in seiner Sprechstunde auftaucht und als Berufstätiger auch noch jede Menge Sonderwünsche mitbringt? Er zum Beispiel. Ohne seine Betreuung und seine prägnanten Feedbacks wäre ich mit den Ergebnissen dieser Arbeit sicher nicht so zufrieden, wie es jetzt sein darf. Danke außerdem für viele anregende Gespräche und Diskussionen über zahlreiche Themen, auch wenn sie oft gar nichts mit dieser Arbeit zu tun hatten.

Acknowledgements

It is my deep-seated conviction that I will never be able to express my gratitude to one person in particular, no matter how many chances I have left to do so: my mother. A strong spirit who always underestimates and downplays her own value, who never had an easy life and despite numerous disappointments and setbacks always had time and an open ear for others – especially her ardous sons. Thank you!

I also wish to thank my colleagues of flave GmbH. The double burden of juggling both a full-time job and completing a course of study was occasionally felt by them as well. Thanks Stefan, Gerhard & "Cheese"! An especially big "thank you" goes out to Ing. "Super-Peter" Lampesberger who helped a complete Android & Java novice with his first steps with apps.

It might be a litte unusual but I also wish to dedicate this work to all those that normally (have to) remain unnamed – even in a scientific work where the correct handling of sources and quotes is essential. Among those I count the numerous and often anonymous developers & designers of free and-or non-commercial software, fonts and templates, but also other internet users and tutorial writers, that helped me along when an Android-, PHP- or LaTeX- problem had me stumped.

Last but not least I would like to express my gratitude to my adviser Peter Purgathofer. What kind of overworked university professor would willingly accept a long-term student of an expiring degree program who suddenly turns up during his office hours with a bag of special requests because of his job? His kind. Without his supervision and concise feedback I wouldn't be as happy with the result of this work as I'm now allowed to be. Further thanks for the many stimulating talks and discussions, even if they often didn't touch the topic of this work.

Kurzfassung

Eine der größeren Herausforderungen für viele, die sich zum ersten Mal mit Themen der Informatik beschäftigen, ist es oft, ein Verständnis für die Grundlagen des Programmierens zu entwickeln. Die vorliegende Arbeit möchte daher vor allem jenen, die mit Computernutzung und / oder Programmierkonzepten noch wenig bis gar nichts zu tun hatten, einen leichteren Einstieg in die Materie zu erlauben.

Verarbeitung und Präsentation erfolgen dabei zwar computergestützt, die Benutzer und Benutzerinnen selbst müssen aber niemals zu einer Maus oder Tastatur greifen, sondern arbeiten ausschließlich mit vorbedruckten Karten, die einfache (Programmier-)Statements enthalten. Diese werden entsprechend den "Spielregeln" aufgelegt und anschließend mit einem Mobilgerät auf Android-Basis fotografiert. Das so entstehende „Programm" wird ausgeführt und das Ergebnis auf dem verbauten oder einem externen Display dargestellt.

Durch Wegfall der wirtschaftlichen und/oder in manchen Fällen vielleicht sogar psychologischen Hürde „PC" wird erwartet, dass einem deutlich breiteren Zielpublikum die Grundlagen des Programmierens näher gebracht werden können. Erleichtert wird dies auch durch den teilweise sehr spielerischeren Ansatz (Stichwort: Gamification). Außerdem kann man das Lösen von einfachen Programmierbeispielen auf diesem Weg leicht zur Gruppenaufgabe machen – ein weiterer Aspekt, der bei Einzelarbeit am PC nicht leicht umzusetzen wäre.

Trotzdem versteht sich dieser Ansatz nicht als Ersatz für traditionellen Programmierunterricht. Er kann allerdings eine sinnvolle Ergänzung darstellen, um Einstiegshürden abzubauen. Besonders jüngere Zielgruppen und Informatik-„Quereinsteiger" können mit dieser Lösung vielleicht besser erreicht werden, als mit bisherigen Ansätzen, welche die Nutzer teilweise in das sprichwörtliche kalte Wasser stoßen.

Abstract

One of the most common and major challenges for many who deal with computer science topics for the first time is developing an understanding of the basics of programming. Therefore this thesis would like to offer an easier kind of approach to these matters, especially for those who have little to no experience with computer usage and programming concepts.

While processing and presentation are still computerized the users themselves never have to touch a mouse or keyboard. They work with pre-printed cards instead that include simple (programming-)statements. These are placed according to the "rules of the game" and then photographed using a mobile device based on Android. The resulting "program" is then interpreted and the result displayed on the built-in or an external display.

By eliminating the economic and sometimes also psychological hurdle posed by a personal computer it is expected that the basics of programming can be brought closer to a much broader audience. This is also facilitated by the more playful approach (Gamification). Additionally the solving of simple programming exercises can easily be made a group task – yet another benefit that is hard to achieve while working on individual computers.

Nevertheless this approach is by no means an attempt to replace traditional programming courses. It can however be a useful supplement to reduce barriers for beginners. Especially younger target groups or latecomers to all things computer sicence can perhaps be easier reached with this approach, without having to throw them into the proverbial deep end.

Inhaltsverzeichnis

Kurzfassung **xi**

Abstract **xiii**

Inhaltsverzeichnis **xv**

Abbildungsverzeichnis **xvi**

1 Einführung & Motivation **1**
1.1 Unterscheidung 2

2 Related Work - Bisherige Ansätze **5**
2.1 Programmieren als Spiel 5
2.2 Grafische Programmiersprachen 7
2.3 Erweiterung um Skriptsprachen 10
2.4 Der letzte Schritt 11
2.5 Fazit 13

3 QRBASIC – Konzept, Idee & Umsetzung **15**
3.1 Entstehungsgeschichte 15
3.2 Namensfindung & Logo 16
3.3 QR Codes 16
3.4 Kartendesign 19
3.5 Die Umsetzung 21
3.6 Präsentation der Ausgabe 25

4 QRBASIC – Version 1.0 **27**
4.1 Die Karten - der Befehlssatz 27
4.2 Konfiguration 32

5 Anwendungsbeispiele für den Unterricht **35**
5.1 Einzelaufgaben 35
5.2 Gruppenaufgaben 38
5.3 Weitere Arbeit mit QRBASIC 40

6 Evaluierung **45**
6.1 Versuchsaufbau . . . 45
6.2 Informelle Evaluierung . . . 47
6.3 Statistische Evaluierung . . . 49
6.4 Fazit Evaluierung . . . 50

7 Future Work **51**
7.1 Erweiterungen . . . 51
7.2 Portierungen . . . 52
7.3 Verbesserung & Optimierung . . . 53
7.4 To be continued 54

Literaturverzeichnis **55**

Anhang **61**
QRBASIC Demokarten . . . 62
Quellcode . . . 73

Abbildungsverzeichnis

2.1 Robot Turtles . . . 6
2.2 Scratch . . . 8
2.3 GameMaker & Game Maker: Studio . . . 11
2.4 Unity . . . 13

3.1 QRBASIC Logo . . . 16
3.2 QR Code Elemente: Fehlerkorrektur-Stufe & Matrix-Linie . . . 18
3.3 Der Text *QRBASIC* als QR Code in allen Fehlerkorrektur-Stufen . . . 18
3.4 Verschiedene Versionen der *QRBASIC*-Karten . . . 20
3.5 Beispiel des finalen Kartendesigns . . . 22
3.6 QRBASIC Beispiel-Ausgabe Prototyp v0.7 . . . 25
3.7 QRBASIC Beispiel-Ausgabe ohne Fehler . . . 26
3.8 QRBASIC Beispiel-Ausgabe mit Fehlern . . . 26

4.1 PRINT | AUSGABE . . . 27
4.2 SET | SETZE . . . 28
4.3 IF & END IF | WENN & ENDE WENN . . . 28

4.4 WHILE & END WHILE | WÄHREND & ENDE WÄHREND 29
4.5 FOR & END FOR | FÜR & ENDE FÜR 29
4.6 ADDITION, SUBSTRA(C|K)TION, MULTIPLI(C|K)ATION & DIVISION 30
4.7 PLAY SOUND | SPIELE MUSIK 31
4.8 Operators 31
4.9 VARIABLE x & CONSTANT | KONSTANTE 0 32

5.1 Beispiellösung: *Hello World!* 36
5.2 Beispiellösung: *Hello «you»!* 36
5.3 Beispiellösung: *Todays weather* 37
5.4 Beispiellösung: *Jukebox* 38
5.5 Beispiellösung: *Gruppenarbeit A* 41
5.6 Beispiellösung: *Gruppenarbeit B* 42
5.7 Beispiellösung: *Gruppenarbeit A & B* 43

6.1 Aufbau für den BEGINNER'S DAY 46
6.2 Mini-Poster "Generiere deine eigene QRBASIC VARIABLE | KONSTANTE 47
6.3 StudienanfängerInnen beim Ausprobieren von QRBASIC 48
6.4 Verhältnis korrekter zu nicht korrekter *QRBASIC*-Syntax 49
6.5 Verteilung der Anzahl der verwendeten Karten 50

7.1 Verschiedene Arten von 2D-Barcodes (Quelle: *Wikimedia Commons)* 52
1 Befehl AUSGABE 63
2 Befehl SETZE 63
3 Befehl WENN 63
4 Befehl ENDE WENN 63
5 Befehl WÄHREND 63
6 Befehl ENDE WÄHREND 63
7 Befehl FÜR 63
8 Befehl ENDE FÜR 63
9 Befehl ADDITION 65
10 Befehl SUBTRAKTION 65
11 Befehl MULTIPLIKATION 65
12 Befehl DIVISION 65
13 Befehl SPIELE MUSIK 65
14 Operator GRÖSSER 67
15 Operator GRÖSSER GLEICH 67
16 Operator KLEINER 67
17 Operator KLEINER GLEICH 67
18 Operator GLEICH 67
19 Operator UNGLEICH 67
20 Variable x 69
21 Variable y 69

22 Variable z 69
23 Konstante ‘Hallo Welt!’ 71
24 Konstante ‘Das Wetter ist’ 71
25 Konstante ‘gut’ 71
26 Konstante 0 71
27 Konstante 1 71
28 Konstante 2 71
29 Konstante 42 71
30 Konstante ‘01.mp3’ 71

KAPITEL 1

Einführung & Motivation

Betrachtet man die Bildungslandschaft im (nicht nur) deutschsprachigen Raum im Bereich IT, fällt einem schnell ein Paradoxon auf: Einerseits klagen Industrie und Wirtschaft immer wieder über den Fachkräftemangel im IT-Sektor und in den nächsten Jahren wird sich diese Situation laut einiger Studien sogar noch massiv verschärfen. [SVM10], [com11], [dW12] Andererseits stehen die entsprechenden Ausbildungsmöglichkeiten aber immer noch nicht jedem offen. Ein großer Teil des Problems ist sicherlich der Umstand, dass Qualität und/oder Umfang des Informatik-Unterrichts im sekundären Bildungsbereich extrem stark variieren. Wird anschließend eine IT-nahe akademische Ausbildung angestrebt, kann dies zum Stolperstein für viele StudienanfängerInnen werden, die durch die Vielzahl neuer Konzepte und Denkweisen vielleicht zu früh abgeschreckt werden. Zusätzlich ist zu bedenken, dass immer mehr berufliche Tätigkeiten – auch wenn das Berufsbild selbst eigentlich nichts mit der „klassischen“ IT zu tun hat – mit Computerunterstützung oder sogar direkt an einem PC erfolgen. Selbst der allgemeinen Bevölkerung scheint das Problem klar und offensichtlich zu sein. Laut einer in der Bundesrepublik Deutschland 2014 durchgeführten Umfrage wünschen sich mehr als drei Viertel (78%) der Befragten mehr und verpflichtenden Informatikunterricht schon während der Pflichtschulzeit. Nur gerade einmal 17% halten das derzeitige Angebot für ausreichend. [Red14] Bis sich die Situation also verbessert, scheint es sinnvoll, Mittel und Wege zu schaffen, die auch Studenten und Studentinnen, die mit Computernutzung und / oder Programmierkonzepten noch wenig bis gar nichts zu tun hatten, einen leichteren Einstieg in die Materie erlauben.

Für Kinder und Jugendliche gibt es bereits einige interessante Ansätze, um erste Grundlagen des Programmierens zu vermitteln, die in Folge auch näher betrachtet werden sollen. Für (junge) Erwachsene ist das Angebot hingegen schon deutlich reduzierter. Die meisten Ansätze setzen aber zu einem gewissen Grad sowieso eine nicht ganz unbedeutende Tatsache voraus: Erfahrung bei der Bedienung eines Computers. Zwar mag der Prozentsatz der Haushalte, die einen eigenen PC besitzen, hoch wie nie zuvor sein, das Fehlen eines Heimcomputers darf aber nicht zum Ausschlusskriterium werden. Außerdem

gibt es immer noch genug Menschen, die erst im zweiten (oder dritten) Bildungsweg zur Informatik kommen, oder aber aus wirtschaftlich oder anderwertig benachteiligten Situationen stammen, für die ein eigener PC einfach nicht möglich ist. Auch diesen soll und muss die Möglichkeit einer fundierten Informatikausbildung geboten werden. Diesen Umständen wird an österreichischen Universitäten schon seit vielen Jahren insofern Rechnung getragen, als man entsprechend ausgerüstete Laborplätze in entsprechender Anzahl anbietet. Aber warum vergisst man gleichzeitig auf diese doch scheinbar bekannte Problematik, wenn man Leute an die Grundlagen des Programmieresn heranführen will?

1.1 Unterscheidung

Im Folgenden möchte ich Abstufungen von Abstraktion verwenden, um zu unterscheiden, wie stark der Lernende mit dem reinen, nackten Programmcode in Berührung kommt. Klassisches Programmieren, also das Tippen von Codezeilen in einer entsprechenden Programmierumgebung, wäre dementsprechend ein Beispiel für *keine Abstraktion*, stellt aber logischerweise gleichzeitig die höchste Einstiegshürde für einen Anfänger dar. Um diese zu senken, gibt es Bestrebungen in zahlreiche Richtungen, die sich aber grob in zwei Kategorien unterteilen lassen:

1.1.1 Teilweise Abstraktion

Bei einer *teilweisen Abstraktion* wird zwar immer noch mit und an einem PC gearbeitet, doch es kommt mehr oder weniger stark spezialisierte Software zum Einsatz, die den Einstieg insofern erleichtert, als die gewünschten Lernziele in einem einfacheren / anderen Kontext dargestellt oder sogar „versteckt“ werden. Beispiele könnten (grafische) Elemente sein, die – vielleicht sogar in einem Spielkontext – einfach nur an eine bestimmte Stelle gezogen werden müssen, oder eine (stark) vereinfachte „Pseudo-Programmiersprache“. Die Bezeichnung Pseudo-Programmiersprache ist dabei nicht abwertend zu verstehen, sie soll nur verdeutlichen, dass die entsprechenden Programmierbefehle noch stärker als „normale“ Programmiersprachen von ihrer Umgebung abhängig sind. Die meisten Vertreter sind in ihre jeweilige Entwicklungsumgebung vollständig integriert und Code wird nach Anleitung bzw. relativ starren Schemen erstellt. So soll ausgeschlossen werden, dass Programmierfehler gemacht werden können. Oft wird hierbei sogar auf eine Schriftsprache verzichtet und es kommen stattdessen zum Beispiel Grafikblöcke zum Einsatz. Dadurch sind auch entsprechende Schreib- und Lesekenntnisse der Kinder nicht zwingende Voraussetzung. Ein klarer Nachteil: In entsprechendem Pseudocode geschriebene (oder „gebaute“) Programme können oft nicht kompiliert und damit (leicht) weitergegeben werden, sondern sind üblicherweise nur innerhalb des Programms lauffähig, in dem sie geschrieben wurden.

Mögliche Vorteile:

- Reduzierte Einstiegshürde

Mögliche Nachteile:

- Umstieg auf eine „richtige“ Programmiersprache kann erschwert sein
- PC-Grundkenntnisse (Bedienung!) werden vorausgesetzt.

1.1.2 Totale Abstraktion

Zusätzlich gibt es aber auch Ansätze, die komplett ohne die Nutzung eines PCs auskommen. In solchen Fällen möchte ich von einer *totalen Abstraktion* sprechen.

Mögliche Vorteile:

- Praktisch keine Einstiegshürde
- Keine bzw. geringe (Hardware)-Voraussetzungen

Mögliche Nachteile:

- Umstieg auf die PC-Nutzung doppelt erschwert, weil möglicherweise Bedienungskenntnisse fehlen und auch auch keine direkte Anbindung an eine vorhandene Programmiersprache möglich ist.

Auf den folgenden Seiten möchte ich einige bisherige Ansätze und Umsetzungen auch unter diesem Gesichtspunkt vorstellen und ihre Stärken und Schwächen analysieren.

KAPITEL 2

Related Work - Bisherige Ansätze

Im Folgenden sollen bereits existierende Konzepte und Ideen vorgestellt werden, mit denen Programmier-Anfänger unterschiedlichster Altersstufen erste Erfahrungen mit den Grundlagen des Progammierens sammeln können. In diesem Zusammenhang spricht man gelegentlich von sogenannten *Erziehungsorientierten Programmiersprachen.*

2.1 Programmieren als Spiel

Besonders bei Kindern stößt man bei jeglicher Art der Wissensvermittlung schnell an eine scheinbar unnachgiebige Grenze: ihre Aufmerksamkeitsspanne. Sie variiert je nach Alter des Kindes zwischen drei und fünf Minuten und steigert sich bis ins junge Erwachsenenalter auf ungefähr 20 Minuten am Stück. Ein Erwachsener kann sich danach aber sozusagen entscheiden, weiterhin aufmerksam zu sein, und setzt diesen „Timer“ damit quasi zurück. [CDCD09] Beim Spielen ist diese Aufmerksamkeitsspanne von Kindern nicht nur höher, sie lässt sich durch häufige, kurze Spielpausen sogar steigern. [PHJ95] Entsprechend wählen die erfolgreichsten Ideen zur Vermittlung von Programmiergrundlagen an Kinder einen eher spielerischen Ansatz.

2.1.1 Robot Turtles

Robot Turtles ist ein vom IT-Experten Dan Shapiro entwickeltes Brettspiel für Kinder im Alter von drei bis acht Jahren. Selbst seit Jahrzehnten in der IT-Branche tätig, wollte der Vater von vierjährigen Zwillingen seinen Kindern so früh wie möglich Grundkenntnisse des Programmierens vermitteln. Sein Ansatz kommt dabei ganz ohne Computer oder sonstige Elektronik aus, tatsächlich müssen mitspielende Kinder noch nicht einmal lesen können. Das Spielkonzept ist relativ simpel: Jedes Kind „programmiert“ eine von vier Roboterschildkröten, um durch ein zuvor aufgebautes Labyrinth zu navigieren und ein Juwel der eigenen Farbe zu erreichen. Dazu stehen Anfängern zuerst nur Bewegungskarten (Drehung links, Drehung rechts, Bewegung vorwärts) zur Verfügung. In jeder Spielrunde

wird dann eine Karte gespielt und die entsprechende Bewegung ausgeführt. Haben die Spieler dieses Konzept verstanden, werden nach und nach weitere Aktionsmöglichkeiten hinzugefügt. Mit der speziellen Funktionskarte „Laser“ kann zum Beispiel eine Wand aus Eis geschmolzen und ein Hindernis damit passiert werden. Bemerkt ein Spieler einen Fehler, kann er zudem eine „Bug“-Karte nutzen, um seinen letzten Zug rückgängig zu machen (erste Parallelen zu Debugging-Konzepten). Schließlich wird man dazu übergehen, die Kinder mehrere – oder vielleicht sogar alle – nötigen Lösungsschritte in einer Reihe und auf einmal auflegen zu lassen. Nur wenn der dadurch entstehende „Programmablauf“ korrekt ist, wird die eigene Robotschildkröte ihr Ziel erreichen, sonst muss eben noch einmal „debuggt“ werden. [LLC14] [Sha13b] Interessant ist vielleicht noch zu erwähnen, dass Shapiro mit seiner Spielidee bei verschiedenen Brettspiel-Herstellern und -Designern abblitzte, da man dort der Meinung war, dass sich das Produkt niemals verkaufen würde, da „Programmieren lernen zu kompliziert klingt“. Shapiro versuchte daher Mitte 2013 sein Glück auf der CrowdFunding-Plattform Kickstarter. Statt der benötigten 25.000 US$ investierten fast 14.000 Unterstützer aber sogar über 630.000 US$ in seine Idee. Damit avancierte *Robot Turtles* zum bis dato erfolgreichsten Brettspiel auf Kickstarter. [Sha13a] Shapiro gründete so beflügelt ein neues Unternehmen (Robot Turtles LLC) und auch ein Vertrieb war plötzlich schnell gefunden.

Abbildung 2.1: Robot Turtles

Einschätzung:

- Einstiegshürde: keine
- Abstraktion: total

2.1.2 Spielen in der Gruppe

Auch Nikos Michalakis hat es sich auf seiner Webseite *DrTechniko* zum Ziel gesetzt, jungen Menschen Informatik und wissenschaftliches Denken näher zu bringen. [Mic14a] Mit „How to train your robot“ [Mic12] hat er ein Konzept entwickelt, mit dem er schon vielen

fünf- bis siebenjährigen Kindern erfolgreich Grundlagen des Programmierens vermitteln konnte – und das ebenfalls ganz ohne Computer. Jedes Kind bekommt dabei seinen eigenen Roboter – einen älteren oder erwachsenen Mitspieler –, der die Instruktionen des Kindes ausführen muss. Diese sind anfangs fix vorgegeben: linkes oder rechtes Bein vor oder zurück; Drehung nach links oder rechts; Gegenstand aufheben oder fallen lassen. Ziel ist es, den Roboter durch einen zuvor aufgebauten, einfachen Hindernisparkour zu navigieren und einen Ball aufzuheben und zurückzubringen. Jedes Kind schreibt dazu die (symbolischen) Befehle auf einen Zettel und übergibt diesen an seinen Roboter, der die Instruktionen dann entsprechend ausführt bzw. verweigert, falls sie unmöglich sind. Ist diese erste Hürde genommen, dürfen die Kinder auch eigene Instruktionen erfinden, ihren Robotern beibringen und von diesen ausführen lassen.

Michalakis war positiv überrascht, wie schnell die Kinder die Konzepte erfassten und in weniger als einer halben Stunde sogar noch komplexere Prinzipien von selbst erkannten und umsetzten. Schon eine Fünfjährige erkannte zum Beispiel die Vorteile einer Parametrisierung, da sie es für sinnvoller fand, eine Zahl vor eine Bewegung zu setzen, als den Befehl mehrfach untereinander zu schreiben. In weiterer Folge vereinfachte er die Idee darum noch weiter, um herauszufinden, ab welchem Alter selbst kleinste Kinder Grundkonzepte der Automatisierung verstehen. „How To Train Your Robot To Jump“ [Mic14b] beschränkt sich daher auf ein einziges Element: einen Pfeil. Legt das Kind den Pfeil so hin, dass er nach oben zeigt, springt der Roboter. Zeigt er hingegen nach unten, setzt oder legt sich der Roboter hin. Erste Versuche mit seinem eigenen zweijährigen Sohn schlugen fehl, aber ein erneuter Versuch nur ein halbes Jahr später war bereits von Erfolg gekrönt. Innerhalb von wenigen Minuten legte sein Sohn mehrere Pfeile hintereinander, ein Seitwärtssprung (Pfeil nach links oder rechts) wurde eingeführt und sogar eine Parametrisierung mit einfachen Zahlen war kein Problem.

Einschätzung:

- Einstiegshürde: keine
- Abstraktion: total

2.2 Grafische Programmiersprachen

Eine einheitliche Definition für grafische Programmiersprachen zu finden, fällt schwer, im Rahmen dieser Arbeit reicht allerdings die (Teil)definition der deutschen *Wikipedia*-Seite: „Als Grafische Programmiersprache (englisch visual programming language, VPL) bezeichnet man eine Programmiersprache, in der ein Programm, Algorithmus oder Systemverhalten durch grafische Elemente und deren Anordnung definiert wird.“ [Wik15] Durch die grafischen Elemente ist die Bedienung oft intuitiver und zudem können auch (noch) nicht bzw. nur teilweise alphabetisierte Kinder diese nutzen.

2.2.1 Scratch

Scratch ist ein Projekt der Lifelong-Kindergarten-Group des MIT Media Labs, welches im Juli 2009 gestartet wurde. [Lab15b] Ihr Selbstverständnis beschreibt die Lifelong-Kindergarten-Group wie folgt „We develop new technologies that, in the spirit of the blocks and fingerpaint of kindergarten, expand the range of what people can design, create, and learn.“ [Lab15a] Frei Übersetzt: „Wir entwickeln neue Technologien, die – im Geiste der Blöcke und Fingerfarben der Kindergartenzeit – erweitern, was Menschen designen, erschaffen und lernen können.“ Mit *Scratch* wurde eine Programmiersprache geschaffen, die diesem Auftrag gerecht werden soll. Es wurde zwar in erster Linie für Kinder zwischen acht und 16 Jahren entwickelt [Lab15c], eignet sich aber auch für ältere Programmiereinsteiger und – mit elterlicher oder vergleichbarer Unterstützung – sogar für noch jüngere Anfänger.

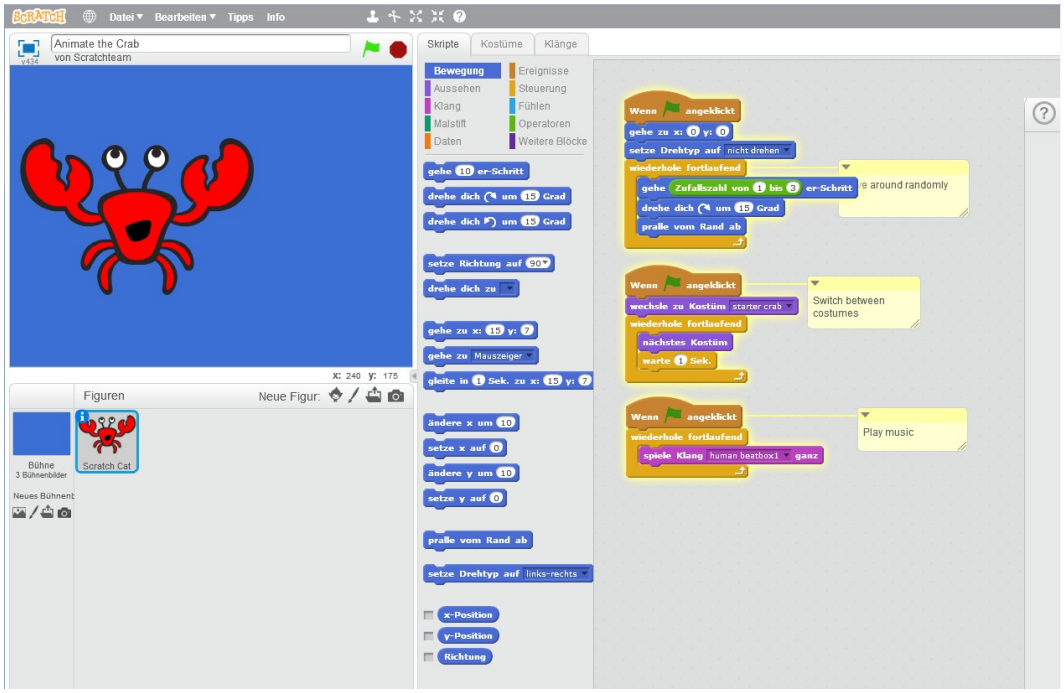


Abbildung 2.2: Scratch

Programmiert wird durch die Anordnung und Kombination vom (Skript-)Blöcken mittels Drag&Drop im online verfügbaren ScratchEditor [Lab15e]. Die neueste Version, *Scratch 2.0*, kann zudem auch offline betrieben werden. [Lab15d]. Insgesamt wird stark auf den grafischen Ansatz gesetzt und vor allem Anfänger konzentrieren sich meist nur auf die Manipulation einzelner grafischer Figuren (Sprites), um einfache Animationen, (interaktive) Geschichten und/oder erste Spiele zu gestalten.

Zu diesem Zweck bietet *Scratch* neun Arten von Blöcken: „Bewegung“ manipuliert die Sprites, bewegt und dreht sie, oder lässt sie am Rand des Bildschirms abprallen.

„Aussehen“ verändert Farben und Größen und blendet Texte ein, mittels „Klang“ können Geräusche und andere Soundeffekte abgespielt werden und der „Malstift“ zieht und/oder manipuliert Farblinien. Mit steigender Erfahrung werden sich „Scratcher“, wie man sich selbst nennt, aber auch an etwas fortgeschrittenere Blöcke wagen: Im „Daten“-Bereich findet man Blöcke, um globale oder nur auf ein Objekt (z.B. ein Sprite) bezogene Variablen und Listen zu definieren. „Ereignisse“ erlaubt, auf Tasteneingaben und (System-)Nachrichten zu reagieren. Unter „Steuerung“ findet man Möglichkeiten zur bedingten Programmierung, unter anderem von Schleifen. Mittels „Fühlen“ hat man Zugriff auf globale Variablen (z.B. Mausposition) und kann unter anderem auch (Text)eingaben abfragen. „Operatoren“ bietet schlussendlich einige grundlegende mathematische Funktionen an und erlaubt auch einfache String-Operationen. In vielen Fällen verfügen die entsprechenden Blöcke über ein oder mehrere Zahlen- und/oder DropDown-Felder, in denen die gewünschte Operation konkretisiert wird – z.B. bewege dich x Pixel, setze die *Lautstärke* auf y oder multipliziere z mit *42*.

Einschätzung:

- Einstiegshürde: gering
- Abstraktion: (beinahe) total

2.2.2 SNAP!

SNAP!, bis Version 4.0 als *BYOB / Build Your Own Blocks* bekannt, ist eine erweiterte Neuimplementierung von Scratch. [oCaB14] SNAP! zielt nicht mehr ausschließlich auf Programmieranfänger ab und ergänzt einige komplexere Konzepte, wodurch auch strukturierte Programmierung möglich wird. Zu den wichtigsten Ergänzungen gehört die Möglichkeit, neue Blöcke zu definieren, sowie die Unterstützung von First-Class-Funktionen, First-Class-Listen und First-Class-Sprites (in BYOB/SNAP! sind – anders als in *Scratch* – auch Sprites First-Class-Objekte [BH11]). [DSW14]

Weitere, teilweise nicht mehr aktive Alternativen, die auf Scratch basieren oder davon (auch nach eigener Aussage) inspiriert wurden: *DesignBlocks* [Lab10], *Pocket Code* [Tea14] und *Panther* [u.a11].

Einschätzung:

- Einstiegshürde: gering
- Abstraktion: semi-total

2.2.3 LEGO Mindstorms

Zumindest eine kurze Erwähnung hat sich auch das *Mindstorms*-Bausystem des dänischen Spielzeugherstellers LEGO verdient. [Gro14c] Zwar konzentriert sich dieses auf Einsatzgebiete in den Bereichen Robotik und Sensorik, die dafür nötigen Programmierkenntnisse

werden aber ebenfalls auf eine sehr einsteigerfreundliche Art vermittelt. Nach der Erstveröffentlichung von LEGO Mindstorms im Jahr 1998 und einer Hardwarerevision 2006 (Mindstorms NXT) ist LEGO Mindstorms EV3 seit 2013 die derzeit aktuelle Version des Roboterbaukastens. [Val13] [Bac13] Herzstück des Systems ist ein programmierbarer Legostein, der unterschiedliche Elektromotoren und Sensoren ansteuern bzw. auslesen kann. Programmiert wird mittels der kostenlos erhältlichen und symbolbasierten LEGO Mindstorms EV3-Software. [Gro14a]. Ähnlich wie bei vielen bereits vorgestellten Konzepten erfolgt die Programmierung durch Drag&Drop-Anordnung von Block-Elementen, die teilweise noch mit zusätzlichen Parametern versehen werden können. Mindstorms unterscheidet dabei zwischen „Aktions"- (z.B. Motorumdrehung), „Programmablauf"- (z.B. Wiederholung), „Sensor"- (z.B. Infrarot), „Datenoperations"- (z.B. Vergleich) und „Erweiterungs"-Blöcken (z.B. Datenübertragung). [Gro14b].

Einschätzung:

- Einstiegshürde: gering
- Abstraktion: (beinahe) total

2.3 Erweiterung um Skriptsprachen

Früher oder später wird trotzdem der Zeitpunkt kommen, an dem bestimmte Ideen sich mit den vorhandenen Blöcken nicht mehr umsetzen lassen und/oder der entsprechende (Nach)bau in Blockform unnötig kompliziert und aufwendig wird. Hier können Umgebungen punkten, die neben Nutzung ihrer eigenen, meist proprietären grafischen „Sprache" auch Erweiterungen in anderen, vielleicht sogar „normalen" Programmiersprachen erlauben.

2.3.1 GameMaker & GameMaker: Studio

Gute Beispiele für diesen Ansatz sind *GameMaker* bzw. dessen Nachfolger *GameMaker: Studio*. *GameMaker* wurde ursprünglich von Mark Overmars [Gam15], dem ehemaligen Leiter des Center for Geometry, Imaging, and Virtual Environments an der Universität Utrecht (Niederlande) entwickelt. [oSU15] Inzwischen hat Overmars die Entwicklung aber an das Softwarehaus YoYo Games übertragen und konzentriert sich auf seine neue Firma Tingly Games. [Gam15] Er hält zwar noch Anteile an YoYo Games, ist in die Entwicklung von *Game Maker* aber nicht mehr aktiv involviert. [Blo14]

GameMaker dient – wie der Name schon verrät – vorwiegend der Programmierung von Computer- und Videospielen. Wie schon bei den bisher vorgestellten Ansätzen will man Anfängern den Einstieg erleichtern, indem das Erlernen einer komplexen Programmiersprache keine Voraussetzung ist. Stattdessen erzeugt man „Räume", in denen man mittels Drag&Drop Objekte platziert und unterschiedliche Verhaltensmuster zuweist. „Raum" ist hier allerdings eher abstrakt zu verstehen, da es sich dabei sowohl um eine

einzelne Szene, als auch eine ganze Spielwelt handeln kann. [Moo14]. Der Fokus liegt außerdem eindeutig auf 2D-Spielen, die Unterstützung von 3D-Elementen ist bestenfalls rudimentär. [For09]

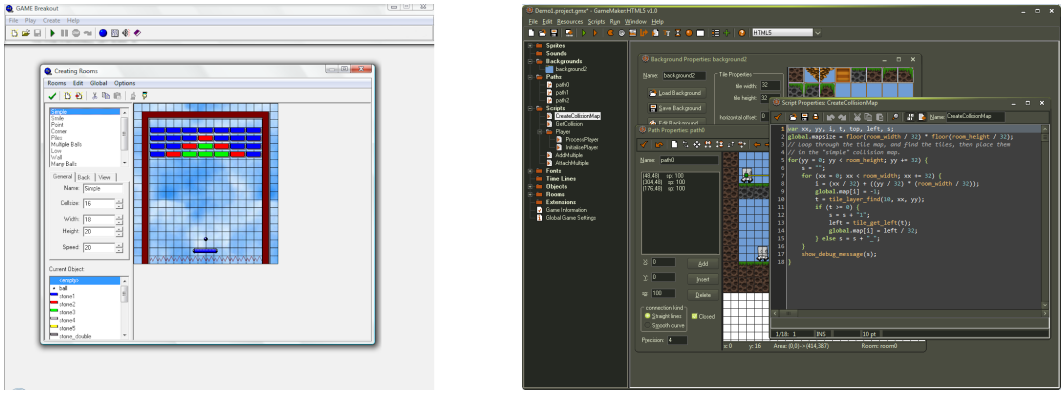


Abbildung 2.3: GameMaker & Game Maker: Studio

Schon in der Ursprungsversion von *GameMaker* konnte ergänzendes Verhalten und Logik mittels der Scripting-Sprache GML (Game Maker Language) implementiert werden. Ähnlich wie Java wurde der GML-Anteil dabei interpretiert [For09] und erst „Just-in-time“ kompiliert, also erst zur Laufzeit in Maschinencode übersetzt. Ein Nachteil war damit natürlich die teils spürbar schlechtere Performance im Vergleich zur „traditionellen“ Ahead-of-time-Kompilierung, bei der das komplette Programm vorab kompiliert wird. Seit *GameMaker: Studio* basiert darum das komplette Programm auf GML und jede Aktion / jeder Block entspricht einem GML-Skript. [**?**] Der GML-Code wird außerdem beim Export des fertigen Programms durch den GameMaker Language Compiler (GMLC) in C++ Code umgewandelt und anschließend kompiliert. *GameMaker: Studio* ist allerdings auch nicht mehr ganz so einfach zu bedienen wie das Original. Das ist allerdings ein Preis, den die Entwickler zugunsten mehr Funktionalität und Kontrolle zahlen mussten. [Blo14]

Einschätzung:

- Einstiegshürde: mittel
- Abstraktion: mittel

2.4 Der letzte Schritt

Die größtmögliche Flexibilität erlauben aber nach wie vor die „traditionellen“ Hochsprachen, wahlweise integriert in vollwertige Entwicklungsumgebungen.

2.4.1 Einsteigerfreundliche Programmiersprachen

Viele Programmiersprachen haben im Laufe der Jahre von sich behauptet, besonders einsteigerfreundlich zu sein. Schon das berühmte *BASIC* aus dem Jahre 1964 verankerte diesen Anspruch sogar in seinem Namen, steht das Akronym *BASIC* doch für „Beginner's All-purpose Symbolic Instruction Code", frei übersetzt „symbolische Allzweck-Programmiersprache für Anfänger". Über die Frage, welche Programmiersprache/n sich besonders für Anfänger eignet/eignen, kann man natürlich trotzdem trefflich streiten. Tatsächlich findet man für eigentlich jede gängige Sprache Bücher und Kurse, die – oft sehr nachvollziehbar – darlegen, warum gerade „ihre" Sprache eine gute Einstiegsmöglichkeit in die Welt des Programmierens sein kann. Natürlich gibt es trotzdem einige Sprachen, die sich besser eignen als andere. Das Online-Magazin *Lifehacker* stellte zum Beispiel im Jänner 2014 seinen Lesern die Frage, welche Programmiersprache die Beste für Anfänger sei (eng. „What's the Best Programming Language for First-Time Learners?"). [Hen14b] Eine Umfrage mit 18.665 Teilnehmern [Hen14a] nominierte Python,C/C++, JavaScript, Java und Ruby als die Top 5-Empfehlungen – allerdings mit der relativ einhelligen Einschränkung, dass C/C++ im Gegensatz zu den anderen Nominierungen zwar nicht unbedingt „einfach" ist, aber trotzdem eine gute Basis für Einsteiger darstellen kann, weil es „sauberes" Programmieren trainiert. Eine Erhebung von Juli 2014 [Guo14] ergab, dass von den 39 besten amerikanischen Bildungseinrichtungen im Bereich der Informatik [New14] die meisten entweder Java oder Python in den Anfängerkursen unterrichten. In weiterer Literatur werden neben diesen beiden aber auch Ruby und JavaScript gerne zitiert. [Wei14], [Bri13], [Mor14], [Ead14]

Einschätzung:

- Einstiegshürde: mittel bis hoch
- Abstraktion: mittel bis keine

2.4.2 Unity

Als Beispiel für eine vollständige Laufzeit- und Entwicklungsumgebung soll, schon wegen ihrer Beliebtheit und Verbreitung – 45 Prozent Marktanteil, vier Millionen registrierte Entwickler –, noch die Spiele-Engine *Unity* Erwähnung finden. [Tec15], [Mag12], [Awa14] Dass auch hier ein Produkt der Spiele-Industrie federführend ist, sollte eigentlich nicht überraschen. In keiner anderen Industrie verschmelzen schließlich mehr (und nicht nur Software-)Disziplinen und entsprechend ausgereift und umfangreich müssen erfolgreiche Entwicklungsumgebungen sein.

Wie viele Entwicklungsumgebungen ist *Unity* sehr grafisch aufgebaut. Einfache Objekte (Lichtquellen, grafische Primitive etc.) werden direkt im Editor erzeugt und platziert, komplexere Elemente (Modelle, Texturen usw.) nach ihrer Erstellung in externen Programmen importiert. Für Grafik, Sound und Animation gibt es eine Vielzahl von eingebauten Mechanismen, die durch selbst geschriebenen Code ergänzt werden. Dabei wird die

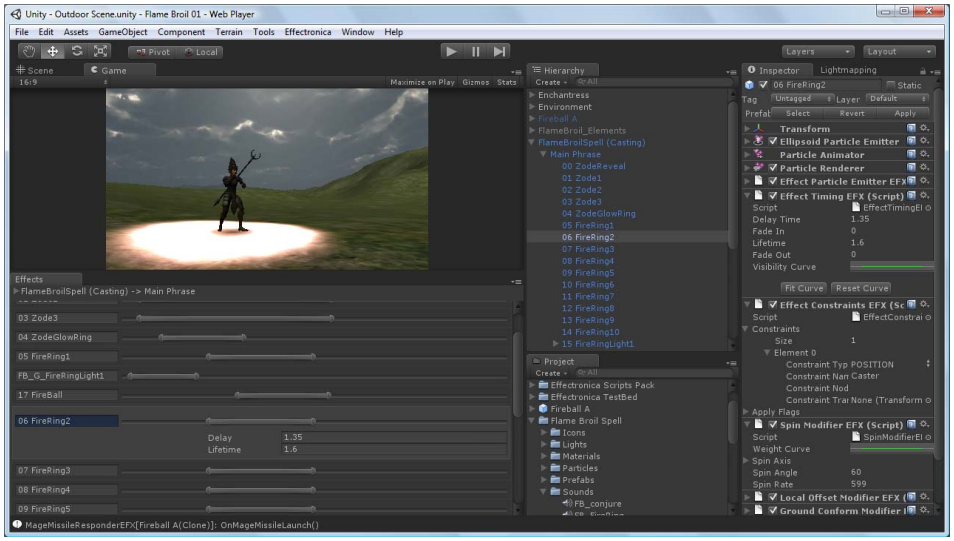


Abbildung 2.4: Unity

Skriptsprache UnityScript (ähnlich JavaScript [Wik14]) sowie C# und Boo (vergleichbar Python [dO05]) unterstützt.

Einschätzung:

- Einstiegshürde: mittel bis hoch
- Abstraktion: mittel bis keine

2.5 Fazit

Wie eingangs schon vorweggenommen, ist das Angebot für Kinder und Jugendliche also überraschend umfangreich und lässt Gutes für kommende Generationen erhoffen. Gleichzeitig verlieren diese frühen Anfänge aber viel von ihrem Nutzen, wenn diese oft nur zarten Flämmchen des Verständnisses nicht genährt werden, denn zu oft folgen darauf Jahre der Inaktivität in diesem Bereich, wodurch das bereits Erlernte oft schnell wieder vergessen wird.

KAPITEL 3

QRBASIC – Konzept, Idee & Umsetzung

3.1 Entstehungsgeschichte

Die Idee für *QRBASIC* entstand sehr spontan, in einem Gespräch über mögliche Themen für eine Diplomarbeit. Der Autor erzählte von seiner beruflichen Tätigkeit mit NFC-Chips, QR Codes und Plastikkarten sowie dem persönlichen Wunsch, dass die angestrebte Arbeit auch Anfängern und Laien von Nutzen sein sollte. Letzterer Gedankengang lenkte das Gespräch in Richtung der häufigsten Probleme von (Studien)anfängern in der Informatik und erinnerte den Betreuer dieser Arbeit, Peter Purgathofer, an einen gerade aktuellen Artikel von Judy Robertson ("Rethinking How to Teach Programming to Newcomers" [Rob14]) in dem sie erläutert, wie sie versucht, den unterschiedlichsten Vorbildungsniveaus ihrer Studentinnen und Studenten im Bereich des Programmierens Rechnung zu tragen. Robertsons Ansatz drehte sich um den Einsatz von App Inventor, einer spezialisierten, Scratch-ähnlichen (vergleiche Abschnitt 2.2.1 auf Seite 8) Programmierumgebung für Android-Applikationen. Diese Lösung war allerdings schon sehr spezifisch und vor allem computergebunden – und derartige Ansätze gibt es wie schon erläutert viele (siehe Kapitel 2). Purgathofer dachte aber eher an einen Ansatz, bei dem die Programmierenden kooperativ und kollaborativ agieren können. Für diese Arbeit wollte man also einen Schritt weiter gehen. Oder doch eher zurück? Warum nicht OHNE Computer programmieren? Die grundlegenden Konzepte – was sind Schleifen, Abfragen, Variablen, Konstanten etc. pp. – lassen sich ja schließlich auch anders, zum Beispiel in Worten beschreiben. Mit dieser Erkenntnis ging es dann eigentlich Schlag auf Schlag: Die bereits vorhandene Erfahrung mit QR Codes war plötzlich ein klarer Vorteil und die Arbeit mit Plastikkarten fixierte auch das Medium recht schnell. Letzteres trug auch dem bereits erwähnten Wunsch nach Zusammenarbeit Rechnung, oder wie es Purgathofer ausdrückte: "Jede/r hat eine Karte und gemeinsam haben wir ein Programm.". Er steuerte übrigens auch die Grundidee für die Evaluierung bei: Für eine sinnvolle Bewertung des

Ergebnisses war es wichtig, vorwiegend TesterInnen zu finden, die eben noch wenig bis keine Erfahrung mit Programmieren haben. Er schlug daher eine Präsentation bei der KinderuniTechnik und/oder am BEGINNERS DAY der TU Wien vor. Aus terminlichen Gründen sollte es schlußendlich der BEGINNERS DAY werden. Denn auch wenn die groben Eckpunkte schnell fixiert waren, sollten Design, Details der "Sprache" sowie die technische Umsetzung doch noch einige Zeit in Anspruch nehmen.

3.2 Namensfindung & Logo

Zumindest der Name war schnell gefunden. *QRBASIC* ist natürlich eine Anspielung auf die 1964 von John G. Kemeny und Thomas E. Kurtz am Darthmouth Collegt entwickelte imperative Programmiersprache *BASIC* [TEK03]. Syntax, Struktur usw. könnten zwar unterschiedlicher nicht sein, aber neben seiner historischen Bedeutung als eine der ersten Sprachen für Programmieranfänger war es vor allem das Akronym *BASIC* selbst, das einfach zu gut zu den Zielen der neuen „Sprache" passte. *BASIC* steht schließlich für **B**eginner's **A**ll-purpose **S**ymbolic **I**nstruction **C**ode – eine Allzweck-Programmiersprache für Anfänger also, genau was auch *QRBASIC* sein will. Sogar einen Verweis auf Symbole (QR-Codes) kann man hineininterpretieren – obwohl das Original natürlich etwas ganz Anderes damit meinte.

Ein (nicht zuletzt aus Zeitgründen) einfaches Logo soll außerdem für einen gewissen Wiedererkennungswert sorgen. In erneuter Anspielung auf die Anfänge der Programmierung wurde ein Design gewählt, das auch auf die niedrig auflösenden Bildschirme vergangener Tage gepasst hätte. Zusätzlich zu diesem Pixel-Design wurde das *Q* von *QRBASIC* noch so verändert, dass es an die speziellen Markierungen von QR Codes erinnert, welche die Orientierung vorgeben.

Abbildung 3.1: QRBASIC Logo

3.3 QR Codes

Ein QR Code ist ein zweidimensionaler Code, der 1994 von der japanischen Firma Denso Wave entwickelt wurde. Anders als eindimensionale Codes (zum Beispiel Strichcodes) werden Daten bei zweidimensionalen Codes nicht nur linear / nur in eine Richtung kodiert, sondern als Fläche über zwei Achsen (oder eben Dimensionen) – zum Beispiel horizontal und vertikal. Dadurch erhöht sich die mögliche Informationsdichte natürlich enorm. [fE15b], [fE15a], [Inc13b]

3.3.1 Geschichte des QR Codes

In den 1960ern erlebte Japan einen enormen wirtschaftlichen Aufschwung und die Veränderung zu einer modernen Konsumgesellschaft. Unzählige neue Supermärkte mit einer nie zuvor dagewesenen Auswahl an Artikeln gehörten schon bald zum Stadtbild. Anfangs musste noch jedes Produkt an den Registrierkassen händisch eingetippt werden, was viel Zeit kostete und auf Dauer auch zu Gelenks- und Nervenleiden beim Verkaufspersonal führte. Das wegen vergleichbarer Probleme in Amerika entwickelte und schnell etablierte Strichcode- oder Barcode-Systeme setzte sich darum auch in Japan schnell durch, löste aber bei Weitem nicht alle Probleme. Einerseits konnte ein Strichcode maximal 20 alphanumerische Zeichen enthalten, andererseits & gleichzeitig fehlte eine Unterstützung für die japanischen Schriften Kani, Hiragana und Katakana komplett. Das japanische Unternehmen Denso Wave, Hersteller von Strichcode-Lesern, beauftragte ein anfangs nur zweiköpfiges Team mit der Entwicklung eines neuen, zweidimensionalen Codes, der diese Probleme lösen sollte.

Neben dem Wunsch nach einem höheren Informationsgehalt, war den Entwicklern vor allem die Geschwindigkeit der Erkennung (Ist da überhaupt ein Code?) und des Auslesens enorm wichtig. Ihre Lösung war schlussendlich ein quadratisches Format mit speziellen Positionsmarkierungen. Warum diese Form und genau diese Positionsmarkierungen? Nach Analyse unzähliger Drucksorten, derer Layouts & Schriftbilder, sowie des allgemeinen Verhältnisses von hellen zu dunklen Flächen stellten sie schlussendlich fest, dass quadratische Formen sowie ein Verhältnis von 1:1:3:1:1 von dunklen Flächen zu hellen Flächen am seltensten war. Entsprechend ergab sich die quadratische Form der Positionsmarkierungen und ihre Einfärbung - jeweils ein Anteil Schwarz und Weiß links und rechts und drei Teile Schwarz in der Mitte. So konnte sichergestellt werden, dass bei beinahe fast allen Anwendungsszenarien eine optimale Erkennung erfolgte.

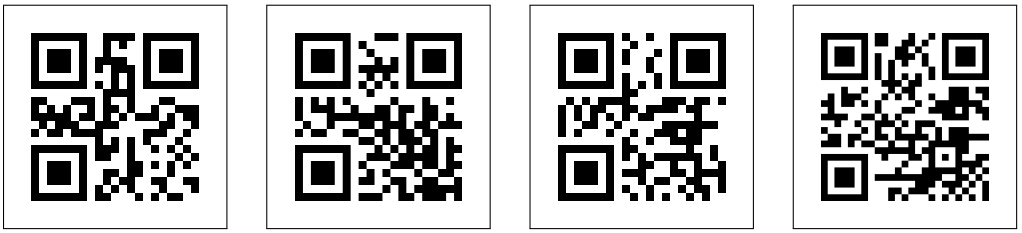
1994 stellte Denso Wave den fertigen QR Code-Standard vor, der über 7.000 Zahlen, 4.000 alphanumerische Zeichen oder 1.800 japanische Schrifzeichen (Kanji & Kana) enthalten und im Schnitt mehr als zehnmal schneller als andere Formate gelesen werden konnte. Die Autoindustrie ware eine der Ersten, welche das neue Format aufgriff und zur Effizienzsteigerung nutzte. Durch den weltweit wachsenden Wunsch und Bedarf nach kompakter und trotzdem detailierter Information – zum Beispiel bezüglich der Herkunft von Lebensmitteln oder einzelner Komponenten – wurde der QR Code schnell zu einem Industriestandard. Ein Umstand, der dabei sicherlich eine wesentliche Rolle spielte, war die fast schon selbstlose und uneigennützige Entscheidung von Denso, die Entwicklung nicht zu kommerzialisieren. Die Spezifikationen wurden öffentlich zugänglich gemacht und jeder durfte den neuen 2D-Code nutzen. Das Unternehmen hält zwar nach wie vor die Patentrechte, hat aber mehrfach erklärt, dass man diese nicht durchsetzen wird und ruhig so viele Leute wie möglich den Code benutzen sollen. [Inc15], [Inc13a]

3.3.2 Technische Merkmale

Ein QR Code besteht aus quadratischen Symbolelementen. In drei seiner Ecken finden sich spezielle Muster, die einerseits den Lesebereich begrenzen und andererseits die Orientierung vorgeben (durch das Fehlen des Musters in der vierten Ecke). Eine Linie abwechselnder Bits zwischen den Eckpunkten definert zudem die Größe der einzelnen Bildelemente.

Zusätzlich kennt der QR Code-Standard vier Fehlerkorrektur-Stufen. Diese erlauben je nach Level eine Rekonstruktion von 7% (Level L), 15% (Level M), 25% (Level Q) oder 30% (Level H) beschädigter und/oder verdeckter Daten. Dazu wird die Fehlerkorrektur der Reed-Solomon-Codierung verwendet. Erkennbar ist die jeweils verwendete Fehlerkorrektur-Stufe anhand des Zustands von zwei Symbolelementen nahe der linken unteren Ecke – siehe Abbildung 3.2 und Abbildung 3.3. Durch höhere Fehlerkorrektur-Stufen wächst aber natürlich auch der Informationsgehalt und bei gleichbleibender Fläche müssen die einzelnen Symbolelemente entsprechend kleiner werden. Es musste daher abgewägt werden, wie hoch die Korrektur-Stufe ausfallen darf, ohne die spätere Erkennung der Karten negativ zu beeinflussen. Nach diversen Tests wurde schließlich Level M als Standard-Stufe gewählt, da bei diesem auch etwas größere Inhalte nicht sofort dazu führen, dass der resultierende QR Code zu detailiert und für die Kamera-Applikation damit schlechter leserlich wird.

Abbildung 3.2: QR Code Elemente: Fehlerkorrektur-Stufe & Matrix-Linie



(a) Level L (b) Level M (c) Level Q (d) Level H

Abbildung 3.3: Der Text *QRBASIC* als QR Code in allen Fehlerkorrektur-Stufen

3.4 Kartendesign

Die grundsätzliche Entscheidung für die Darstellung auf Karten beruhte auf folgenden Überlegungen: Die Produktion der benötigten Elemente sollte für praktisch jedermann möglich und auch in größerer Stückzahl einfach, schnell und günstig sein. Dadurch ergab sich praktisch von selbst, dass ein Medium genutzt werden sollte, das gedruckt werden kann. Die offensichtliche – und in vielen Fällen auch sicher praktikable – Lösung auf normales Papier zu drucken, bringt allerdings den Nachteil der Kurzlebigkeit mit sich. Die Ausdrucke zu laminieren kann helfen, verursacht aber (minimale) Zusatzkosten sowie einen zeitlichen Mehraufwand. Letzterer ist aber ohnehin immer gegeben, da die Elemente schon aus Platzgründen nicht zu groß sein sollten. Jede gedruckte Seite würde daher vermutlich mehrere Elemente enthalten, die aber wiederum erst zusammengestellt und anschließend ausgeschnitten werden müssen.

Die Suche nach Alternativen fiel im vorliegenden Fall allerdings nicht schwer, da der Autor durch seine berufliche Tätigkeit mit Plastikkarten zu tun hat und diese in semi-professioneller Qualität günstig und einfach drucken kann. Das allgemein bekannte „Scheckkartenformat“ (genauer: ISO/IEC 7810, ID-1) von 85,60 x 53,98 mm bietet ausreichend Platz für die nötigen Elemente, ist praktisch jedem vertraut und punktet durch Langlebigkeit und günstige Bezugsmöglichkeiten auch für nicht-kommerzielle Nutzer. Zudem lassen sich problemlos 6-8 dieser Karten auf einer gedruckten A4-Seite unterbringen, wodurch das Format sich auch verträglich verhält, falls man keinen Kartendrucker zu Hand hat.

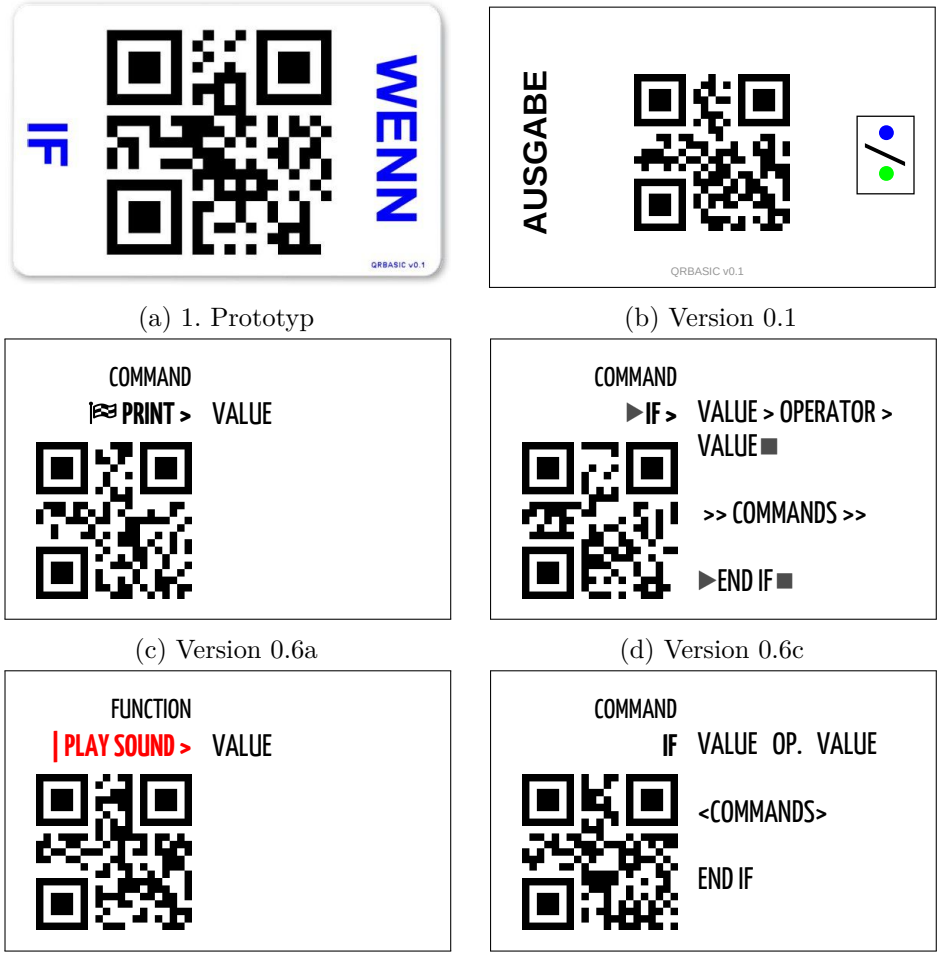
Das Design der Karten selbst selbst durchlief mehrere Iterationen, teilweise bedingt durch vorab gemachte Überlegungen, teilweise durch Erfahrungen während der Implementierung und der ersten Praxistests. Zu Beginn war es zum Beispiel geplant, die Karten im Hochformat aufzulegen. Dadurch wäre der Abstand zwischen den einzelnen Karten einer Reihe von Befehlen enger gewesen und gewissermaßen der Lesefluss optimiert worden. In der Praxis zeigten sich allerdings zwei Probleme:

1. Das Hochformat macht es beim Design der Karten nötig, die einzelnen Elemente schmal zu halten. Schriftgrößen mussten reduziert werden, damit nicht zu viele Zeilenumbrüche entstanden und es war nicht sinnvoll möglich, den Platz links oder rechts vom abgedruckten QR-Code zu nutzen.

2. Bedingt durch die Syntax von *QRBASIC* sind die einzelnen Befehlszeilen eher kurz – im Durchschnitt vier Karten oder weniger. Dadurch ergibt es sich, dass ein aufgelegtes Programm fast immer höher als breit ist – ein Umstand, der durch das Hochformat der Karte noch verschärft worden wäre. Nicht nur der menschliche Seh-Apparat, sondern auch die meisten Arbeitsflächen sind aber auf ein eher längliches Querformat ausgelegt, das Hochformat führte daher schnell zu Platzproblemen. Zudem hätte man die aufnehmende Kamera ebenfalls für Hochformat-Aufnahmen einrichten müssen. Ein Mehraufwand, weil einerseits das komplette Programm,

andererseits auch die Aufhängung (Stativschrauben befinden sich traditionell nur auf der Längsseite eines Kamera-Korpus) hätte angepasst werden müssen.

Wichtig war zudem, dass die Karten im Idealfall selbsterklärend sein sollen. Wäre es nötig vor der ersten Nutzung eine genaue Beschreibung zu lesen oder eine Einschulung zu erhalten, dann hätte es eines der grundlegenden Ziele dieser Arbeit verfehlt. Eine der größten Herausforderungen war daher klarzumachen, welche Karten „zusammenpassen“. Genauer gesagt: Welche Karten in einem (gültigen) „Programm“ als nächstes folgen können/dürfen – nach einer *AUSGABE* muss zum Beispiel immer der auszugebende Wert in Form einer *VARIABLE* oder einer *KONSTANTE* folgen.



(a) 1. Prototyp

(b) Version 0.1

(c) Version 0.6a

(d) Version 0.6c

(e) Version 0.8

(f) Version 0.9

Abbildung 3.4: Verschiedene Versionen der *QRBASIC*-Karten

Entsprechend viele Iterationen durchlief auch die Beschriftung der Karten und viele Ansätze mussten später wieder verworfen werden. Zur besseren Differenzierung unter-

schiedlicher „Arten“ von Karten wurde zum Beispiel eine farbliche Kodierung überlegt. Die Idee, diese zu nutzen, um anzuzeigen, was als nächstes folgen kann/darf, musste aus Rücksichtnahme auf Farbblindheit und andere Sehbeeinträchtigungen allerdings wieder verworfen werden. Ein Versuch, stattdessen Piktogramme zu verwenden (zum Beispiel einen kleinen Drucker für den Befehl *AUSGABE*), scheiterte wiederrum daran, dass nicht annähernd genügend „sprechende“ Symbole gefunden werden konnten, bei denen man hoffen durfte, dass die in vielerlei Hinsicht diverse Zielgruppe diese auch intuitiv versteht. Die schlussendlich gewählte Methode einer rein textlichen Beschreibung mag daher eher nüchtern wirken, ist aber vielleicht gerade durch ihre Schlichtheit ansprechend und intuitiv. Eine kleine optische Spielerei und Verschönerung wurde dennoch umgesetzt. Es ist durchaus möglich, die einzelnen Symbolelemente eines QR Codes leicht zu modifizieren, ohne die Erkennungsrate allzu negativ zu beeinflussen. Entsprechend werden alle QR Codes vor dem Druck in ein binäres Format (1 für einen Punkt, 0 wenn kein Punkt) umgewandelt. Anschließend wird eine von den Dimensionen her idente Grafik generiert, doch wird statt eines Quadrats ein Kreis gezeichnet, der denselben Durchmesser wie die Breite des ursprüngliuchen Quadrats hat. Das Ergebnis ist um einiges runder und – so die Meinung aller befragten Testuser (siehe Kapitel 6) – auch durchaus sehr ansprechend.

Dem einzig möglichen Nachteil, dass die Texte durch Festlegung auf eine Sprache für unterschiedliche Zielgruppen (nicht-deutsche Muttersprache, verschiedene Altersgruppen, etc.) potenziell weniger geeignet sind, wurde natürlich Rechnung getragen. Die Karten selbst liegen bereits in deutscher und englischer Sprache vor und mit nur einer einzigen Konfigurationseinstellung (siehe Abschnitt 4.2.1, Seite 32) kann die Sprache der kompletten Ausgabe inklusive aller Fehlermeldungen entsprechend umgestellt werden. Die Karten der verschiedenen Sprachversionen sind selbstverständlich untereinander austauschbar und können auch problemlos gemischt werden. Wie alle Elemente von *QRBASIC* liegt aber auch dieser Teil komplett „offen“. Die Basiskarten werden von einem frei zugänglichen PHP-Skript als beliebig bearbeitbare PDF-Dateien generiert und auch für die Erzeugung von Variablen und Konstanten gibt es entsprechende Skripts. Jedem steht dadurch die Möglichkeit offen, die Karten vor oder nach Generierung seinen oder ihren individuellen Wünschen anzupassen.

3.5 Die Umsetzung

Nach Abwegung aller Voraussetzungen, Wünsche und der vorhandenen Möglichkeiten wurde schlussendlich entschieden, das Projekt in zwei getrennte Komponenten / Applikationen aufzuspalten und diese auch getrennt zu programmieren:

1. Eine in Java geschriebene App für das Kamera-Smartphone, welches alle Aufnahmen verarbeitet und die darin enthaltenen QR Codes erkennt und verortet.

2. Das Ergebnis dieser Operationen wird anschließend an eine Reihe serverseitiger PHP-Skripts übermittelt, welche die QR Codes ordnen und den dabei entstehenden Programm-Code soweit möglich validieren. Bei erfolgreicher Validierung wird der

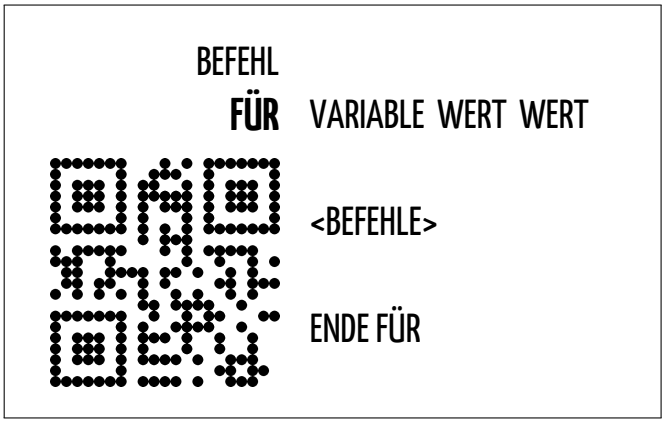


Abbildung 3.5: Beispiel des finalen Kartendesigns

QRBASIC Code in ausführbaren PHP-Code übersetzt, der anschließend ausgeführt wird. Zusätzlich wurden noch einige zusätzliche Skripts programmiert, welche einerseits die Standard-Karten für Befehle, Operatoren usw. erzeugen, andererseits auch individuell erzeugte Karten für zum Beispiel Variablen und Konstanten erlauben.

3.5.1 Die Android-Applikation

Die Aufgaben der Android-Applikation wurde bewusst minimal gehalten. Einerseits sollte die einzige Interaktionsschnittstelle zwischen Benutzer und tatsächlichem Programm so einfach und minimal wie möglich ausfallen. Andererseits ist der Autor in Java nicht besonders bewandert und musste sich erst mit Hilfe eines Android-Programmierbeispiels [Gee15] für QRCode-Erkennung zahlreicher Tipps eines erfahreneren Programmierers (siehe Danksagung) an das Endergebnis herantasten.

Der App obliegt die wichtige Aufgabe, die aufgelegten Karten zu fotografieren und anschließend alle enthaltenen QR Codes zu erkennen und zu decodieren. Entsprechend verfügt die App (nicht mehr) über ein eigenes Interface oder Menü. Bei Programmstart aktiviert sich sofort die Kamera und bei erfolgreicher Aufnahme wird das Bild wie erwähnt verarbeitet und das Ergebnis an den serverseitigen PHP-Code für die weitere Be- und Abarbeitung weitergeleitet.

Für die Bilderfassung wird die vom SDK gebotene Kamera-Schnittstelle von Android verwendet. Die Erkennung der im Bild enthaltenen QR Codes übernimmt die für 1D- und 2D-Code-Erkennung häufig verwendete quelloffene und unter Apache License v2.0 [Fou04] lizensierbare Bildverarbeitungs-Bibliothek *ZXing* [Tea15b], genauer gesagt die von dieser zur Verfügung gestellte Klasse *QRCodeMultiReader* [Tea15a]. Mit dem nach erfolgter Erkennung vorliegenden Ergebnis wird anschließend eine statisch hinterlegte URL innerhalb eines eigenen WebViews aufgerufen. Nach Betätigung der “Zurück”-Taste

am Gerät, Neustart oder Rückkehr Fokus-Verlust startet die Ausführung von vorne und es wird wieder die Kameraschnittstelle aufgerufen.

3.5.2 Der PHP-Code

Der serverseitige PHP-Code nimmt das Ergebnis der QR Code-Erkennung an und arbeitet es – nach vorheriger lokaler Speicherung zu Debug-Zwecken – in folgenden Schritten ab:

1. **Verortung:** Die erste Herausforderung auf Serverseite ist die korrekte Anordnung der erkannten Karten. Die verwendete Bibliothek gibt zwar für jeden erkannten QR Code auch die Koordinaten der linken oberen Ecke zurück, sonst aber auch nichts. Um die Karten daher in der Reihenfolge abzuarbeiten, wie sie auch ein Mensch vermutlich lesen würde, war darum einige Feinarbeit nötig. Schließlich kann man nicht davon ausgehen, dass die Karten absolut exakt neben- und übereinander aufgelegt werden. In jeder Zeile wird die eine oder andere Karte etwas höher oder niedriger als ihre Nachbarn liegen und vielleicht auch noch etwas schief. Die nächste Zeile fängt dafür vielleicht etwas weiter rechts oder links als ihre Vorgänger an. Daher war eine reine Sortierung der Koordinaten nicht zielführend. Stattdessen wurde zuerst versuchsweise ein Toleranzwert für die y-Koordinaten aller Karten ermittelt. Dieser ist abhängig von der gewählten Auflösung der Kamera, der Größe der verwendeten Karten und der Größe der verwendeten QR Codes. Es mag daher möglich sein, den entsprechenden Idealwert rechnerisch zu ermitteln, in vorliegendem Fall wurde er allerdings einfach durch zahlreiche Versuche näherungsweise ermittelt. Die (zuerst noch gerundeten) y-Koordinaten werden sodann mit diesem Toleranzwert verglichen und entsprechend werden sie einer Zeile zugeordnet. Anschließend müssen die Karten nur mehr nach der x-Koordinate sortiert werden, um die Verortung abzuschließen.

2. **Darstellung & Validierung:** Wurde die Reihenfolge der Karten fixiert, beginnt die erste Ausgabe: Alle erkannten Karten werden in vereinfachter Form dargestellt. Dabei wird allerdings nur der erkannte QR Code als Referenz hinzugezogen. Sprache und Design der Karte selbst sind also zweitrangig, die Bennenung der Karten erfolgt gemäß der vorhandenen Übersetzungdateien. Zeitgleich erfolgt aber auch die Hauptvalidierung. Alle Kartenabfolgen werden auf Gültigkeit entsprechend der *QRBAISC*-Spielregeln überprüft und eventuelle Verstöße optisch hervorgehoben. Zusätzlich werden auch einige prinzipiell nicht erlaubte Operationen soweit möglich erkannt und unterbunden. Sämtliche Fehlermeldungen werden ebenfalls in der serverseitig gewählten Sprache ausgegeben.

3. **Übersetzung in ausführbaren Code:** War die Validierung soweit erolgreich und wurden keine offensichtlichen Fehler erkannt, beginnt die Übersetzung in realen Code. Manche Befehle/Karten können dabei fast 1:1 übernommen werden, andere stehen für komplexe Konstrukte und Funktionen die gegebenenfalls einige zusätzliche Schritte nötig machen. Wo nötig und sinnvoll werden zusätzlich Sicher-

heitsmechanismen eingebaut, die ungewolltes Programmverhalten soweit möglich unterbinden – zum Beispiel Divisionen durch 0.

4. **Ausführung:** Tritt auch bei der Übersetzung kein Fehler auf, wird der resultierende Code ausgeführt und das Ergebnis ausgegeben.

Auf die genauere programmiertechnische Umsetzung wollen wir an dieser Stelle nicht eingehen. Der kommentierte und offen lesbare Quellcode - PHP ist schließlich eine Skriptsprache – kann aber jederzeit über einen im Anhang dieser Arbeit zu findenen Link heruntergeladen werden und der Autor steht auch gerne für Fragen zur Verfügung.

Sonstige Skripts

Zusätzlich wurden noch vier unterstützende Skripts geschrieben, deren einzige Aufgabe die Erzeugung der für *QRBASIC* nötigen Karten ist.:

1. **base_cards_de.php & base_cards_en.php** erzeugen ein 30 Karten umfassendes "Starter-Deck" für *QRBASIC* in deutscher oder englischer Sprache. Enthalten sind dabei die 13 Befehls- und sechs Operator-Karten; jeweils eine Karte für die Variablen *x*, *y* und *z* und schußendlich die acht Konstanten *"Hallo Welt!"*,*"Das Wetter ist"*,*"gut"*,*"0"*,*"1"*,*"2"*,*"42"* und *"01.mp3"*.

2. **create_const.php & create_var.php** Mit diesen beiden Skripts können zusätzliche und/oder eigene Variablen und Konstanten erzeugt werden. Bei Variablennamen müssen allerdings einige Konventionen erfüllt sein, die auch teilweise von der verwendeten (für den Benutzer normalerweise nicht ersichtlichen) Programmiersprache abhängen. Im Fall von *QRBASIC* ist dies *PHP* und hier müssen Variablen-Namen den gleichen Konventionen wie andere Bezeichner folgen: "Ein gültiger Variablen-Name beginnt mit einem Buchstaben oder einem Unterstrich ("_"), gefolgt von einer beliebigen Anzahl von Buchstaben, Zahlen oder Unterstrichen." [Gro15a] Diese Anforderung wird bei Generierung neuer VARIABLE-Karten erfüllt, indem der Variablenname über den regulären Ausdruck (regular expression) [a-zA-Z_\x7f-\xff][a-zA-Z0-9_\x7f-\xff]* auf seine Gültigkeit geprüft wird.

Auch der kommentierte Quell-Code dieser Skripts kann natürlich unter dem im Anhang zu findenden Link heruntergeladen werden.

3.5.3 Die Hardware

Eine der wohl einfachsten Aufgaben war die Beschaffung der nötigen Hardware, da diese bereits vorhanden war. Das Samsung Galaxy S4 Zoom mag zwar nicht mehr das neueste Modell sein, ist aber eines der wenigen Smartphones am Markt, das über einen "echten" optischen Zoom (10-fach) verfügt. Gepaart mit einer 16 Megapixel-Kamera samt dediziertem Auslöseknopf und dem Betriebssystem Android waren alle nötigen Voraussetzungen erfüllt. Durch softwareseitige Umbelegung einiger Hardwaretasten (u.a.

der Auslösetaste) konnte außerdem erreicht werden, dass der komplette Ablauf mit nur vier Tasten einer Bluetooh-Tastatur gesteuert werden konnte.

3.6 Präsentation der Ausgabe

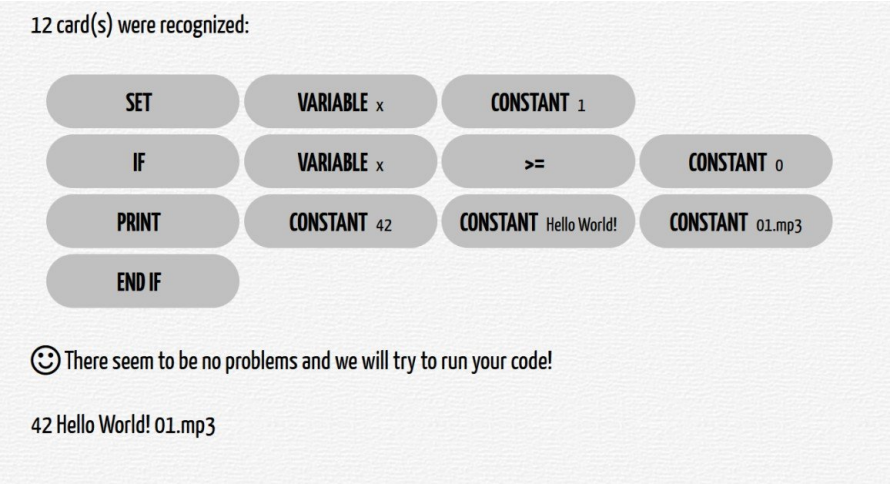


Abbildung 3.6: QRBASIC Beispiel-Ausgabe Prototyp v0.7

Wie auch die Karten selbst durchlief auch die Ausgabe mehrere Design-Iterationen. Schlussendlich schien es am sinnvollsten und effizientesten, die komplette Ausgabe und alle wichtigen Anzeigen auf eine Bildschirm-Seite zu vereinen. Dazu wurde der Bildschirm in drei Teile aufgeteilt:

1. Die linke Hälfte des Bildschirms wird für die Anzeige der erkannten Karten und ihrer Positionen genutzt. Um den Grundgedanken des Designs zu bewahren, werden dazu Miniaturkarten entsprechend der eingestellten Sprache erzeugt. Dadurch ist eine schnelle optische Kontrolle möglich, ob alle Karten erkannt und auch bezüglich Reihenfolge und Zeilennummer korrekt verortet wurden. Die Generierung jeder Zeile erfolgt dabei parallel mit der Validierung und erkannte Fehler werden direkt unter der problematischen Zeile dargestellt, die auch farblich hervorgehoben wird. Globale Fehler, die teilweise erst am Ende des Programms offensichtlich werden – zum Beispiel ungeschlossene Schleifen usw. – werden unterhalb der letzten Programmzeile ausgegeben. Zur besseren Lesbarkeit erfolgt zudem eine automatische Einrückung bei WHILE- und FOR-Schleifen sowie IF-Abfragen, was dem Benutzer – anfangs vielleicht ganz unbewusst – auch gleich diese “best pratice” nahe bringt.
2. Die rechte Bildschirmhälfte wurde vertikal geteilt. Der obere Teil dient dabei der Darstellung der Ausgabe. Dazu wird ein stilisierter Computerbildschrim dargestellt, der mit seiner Quarter VGA-Auflösung von 320x240 Pixel und einer klassischen Terminal-Schriftart ein wenig Retro-Charm verbreiten soll. Die Bildschrimgröße ist

dabei statisch und es wird während der Ausgabe automatisch nach unten gescrollt, damit stets nur die letzten ausgegebenen Zeilen dargestellt werden.

3. Das rechte untere Viertel des Bildschirms wird schlussendlich als Variablen-Register verwendet, das alle definierten Variablen und deren Inhalt mit Ende der Programmausführung beinhaltet.Nicht jedes Programm muss schließlich eine sichtbare Ausgabe beinhalten und zudem ist es zu Debug-Zwecken und zur Verständnissteigerung der Programmabläufe ein klarer Vorteil.

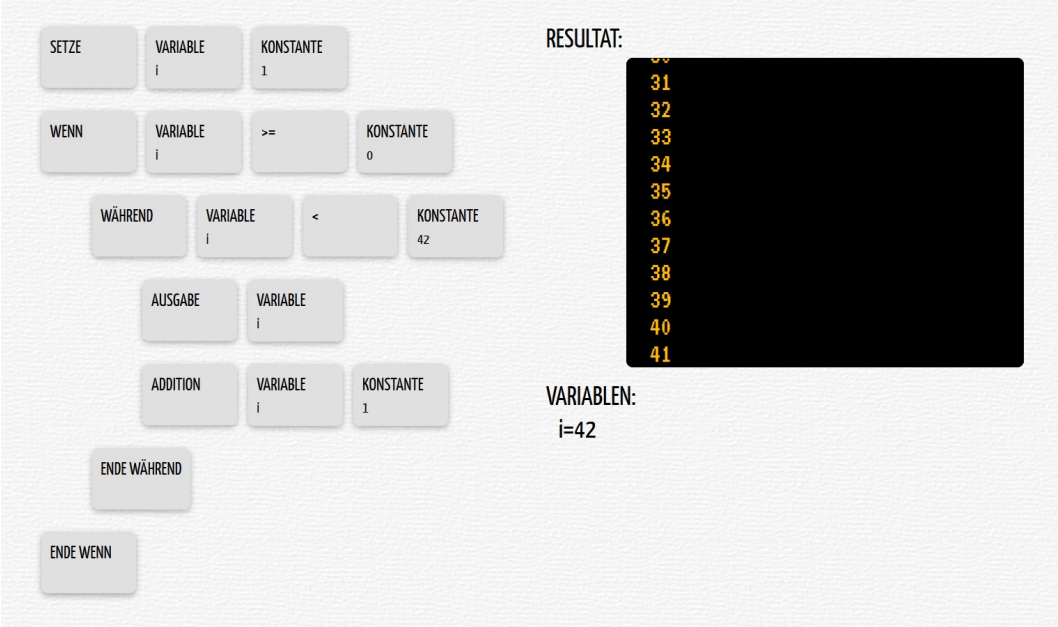


Abbildung 3.7: QRBASIC Beispiel-Ausgabe ohne Fehler

Abbildung 3.8: QRBASIC Beispiel-Ausgabe mit Fehlern

KAPITEL 4

QRBASIC – Version 1.0

4.1 Die Karten - der Befehlssatz

Der vollständige "Befehls-"Satz von *QRBASIC* umfasst 21 verschiedene Arten von Karten, aufgeteilt auf drei Kategorien: 13 verschiedene **Command- | Befehl-**, sechs Arten von **Operator-**, sowie zwei Arten von **Value- | Wert**-Karten. Jede dieser Karten muss unterschiedlich validiert und – falls kein Problem gefunden wurde – in ausführbaren Code übersetzt werden, teilweise auch noch abhängig vom gewählten Regelsatz. Allen Befehls-Karten (mit Ausnahme von PRINT) ist zumindest gemein, dass sie eine fixe Länge / aus einer bestimmten Anzahl von Karten bestehen (müssen). Entsprechend wird auf jeden Fall im Validierungs-Schritt immer überprüt, ob eine Zeile aus zu vielen oder zu wenigen Karten besteht.

4.1.1 Command- | Befehl-Karten

PRINT | AUSGABE

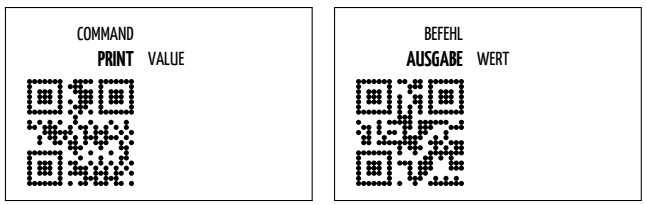


Abbildung 4.1: PRINT | AUSGABE

Die Command-Karte PRINT gibt die Inhalte/Werte aller darauf folgenden Value-Karten (also Variablen und/oder Konstanten) aus. Es muss mindestens eine Value-Karte folgen. Wird mehr als eine Value-Karte angelegt, werden alle Einzelinhalte hintereinander, getrennt durch automatisch ergänzte Leerzeichen, ausgegeben.

Im Validierungsschritt wird bei PRINT geprüft, ob die folgenden Karten (mindestens eine) alle Value-Karten sind. Falls der Regelsatz *strict* gewählt wurde und es sich um Variablen handelt, muss zudem kontrolliert werden, ob diese auch definiert wurden. Andere Arten von Karten werden nicht akzeptiert.

SET | SETZE

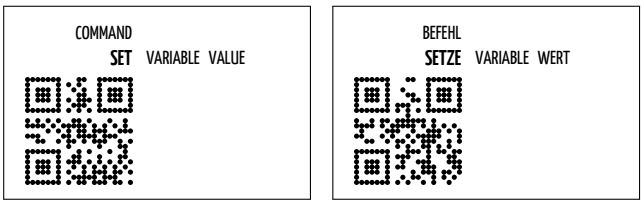


Abbildung 4.2: SET | SETZE

SET-Karten erwarten in weiterer Folge die zu setzende Variable und anschließend den zu setzenden Inhalt/Wert in Form einer Value-Karte.

Handelt es sich bei dem zu setzenden Inhalt/Wert um eine Variable, muss natürlich wieder auf Existenz geprüft werden, wenn der Regelsatz *strict* aktiviert wurde. Die zu setzende Variable selbst muss zu diesem Zeitpunkt natürlich noch nicht definiert sein. Unmittelbar nach erfolgter Validierung – und damit noch vor Ausführung des eigentlichen Codes – muss ihre Definition natürlich vermerkt werden, damit folgende, diese Variable nutzende Befehle und Funktionen, korrekt validiert werden können.

IF & END IF | WENN & ENDE WENN

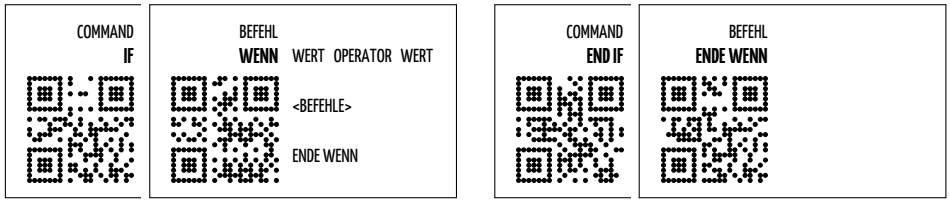


Abbildung 4.3: IF & END IF | WENN & ENDE WENN

Eine IF-Abfrage besteht aus der Command-Karte IF, gefolgt von einem Wert (Value), einem (Vergleichs)Operator und einem Vergleichswert (Value). Dannach kann eine beliebige Anzahl anderer Command-Karten folgen, die allerdings eben nur ausgeführt werden, wenn der Vergleich “wahr” ist. Um zu signalisieren, dass diese bedingungsabhängigen Befehle vorbei sind, muss irgendwann die für sich allein stehende Karte END IF folgen.

Auch hier müssen in der Validierung alle Variablen (falls *strict*) auf ihre Gültigkeit geprüft werden und als dritte Karte darf natürlich nur eine Operator-Karte akzeptiert werden. Um quasi "global" validieren zu können, muss außerdem genau mitverfolgt werden, wie viele IF-Karten eine entsprechende Bedingung begonnen bzw. wie viele END IF-Karten eben diese beendet haben. Eine Diskrepanz in der Anzahl bzw. zum Beispiel die Präsenz einer END IF-Karte ohne vorherige IF-Karte muss natürlich im Rahmen der Validierung abgefangen werden.

WHILE & END WHILE | WÄHREND & ENDE WÄHREND

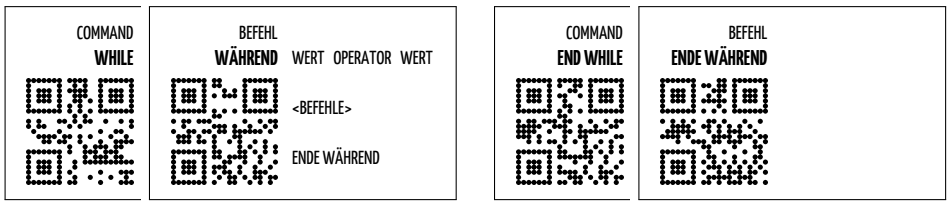


Abbildung 4.4: WHILE & END WHILE | WÄHREND & ENDE WÄHREND

Die Syntax einer WHILE-Schleife ähnelt jener der IF-Abfrage. Nach der Command-Karte WHILE folgt ein Wert (Value), ein (Vergleichs)Operator und ein Vergleichswert (Value). Dannach kann ebenfalls eine beliebige Anzahl anderer Command-Karten folgen, die solange immer wieder ausgeführt werden, bis der Vergleich nicht mehr zutrifft. Auch hier muss irgendwann eine allein stehende Karte END WHILE folgen, um vom restlichen Programm zu unterscheiden.

Die anschließende Validierung unterscheidet sich hingegen praktisch gar nicht: Alle Variablen werden (ggf. *strict*) auf ihre Gültigkeit geprüft und es wird global mitverfolgt, ob Anzahl und Reihenfolge aller WHILE- und END WHILE-Karten einer gewissen Grundlogik folgen.

FOR & END FOR | FÜR & ENDE FÜR

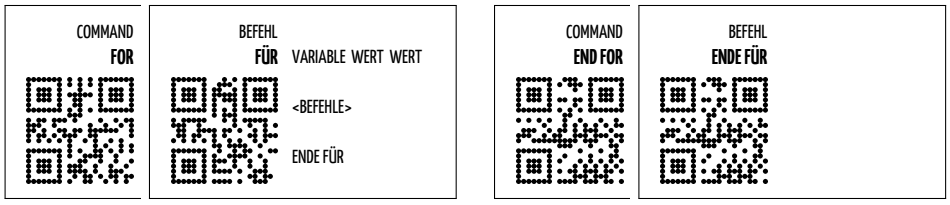


Abbildung 4.5: FOR & END FOR | FÜR & ENDE FÜR

Eine FOR-Schleife ist die zweite Art von Schleife in *QRBASIC*. Sie durchläuft für eine bestimmte Variable einen Bereich, der durch zwei Werte (Value) definiert wird.

Anders als andere Befehle, die eine Variable verwenden, ist es hier allerdings nicht nötig zu kontrollieren, ob diese bereits im Vorfeld definiert wurde, egal welche Regeln zur Anwendung kommen. In einer FOR-Schleife wird der anfängliche Wert der zu durchlaufenden Variable durch den gegebenen Bereich definiert, es ist also nur nötig, die Existenz der entsprechenden Variable zu vermerken, falls dies noch nicht geschehen ist.

ADDITION, SUBSTRA(C|K)TION, MULTIPLI(C|K)ATION & DIVISION

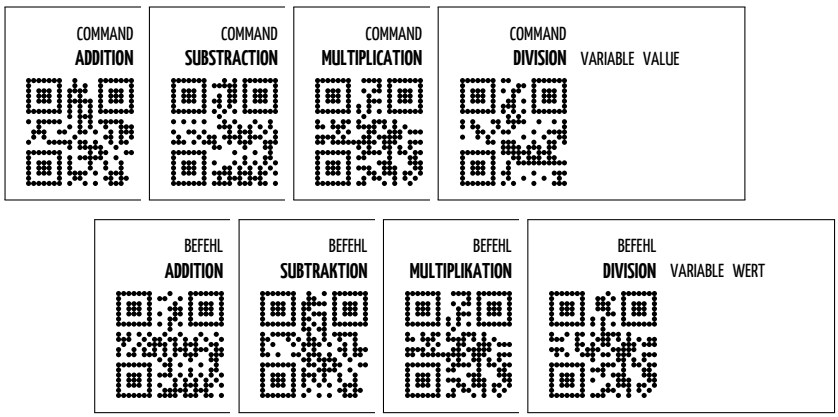


Abbildung 4.6: ADDITION, SUBSTRA(C|K)TION, MULTIPLI(C|K)ATION & DIVISION

Die mathematischen Operationen ADDITION, SUBSTRACTION, MULTIPLICATION & DIVISION folgen alle dem gleichen Schema: zuerst eine Variable und dann ein Wert (Value) um den sie erhöht oder reduziert, mit dem sie multipliziert oder durch den sie dividiert werden soll.

Sämtliche Variablen müssen (sofern die "strengen" Regeln gelten) natürlich bereits definiert sein. Ein Sonderfall ist zusätzlich bei der Karte DIVISION zu beachten. Teilen durch Null ist bekanntlich nicht definiert und daher prinzipiell verboten. Durch die zeilenweise Validierung der Befehle kann allerdings im Rahmen der Validierung nicht ausgeschlossen werden, dass – falls es sich beim Teiler um eine Variable handelt – diese zur Laufzeit irgendwann den Wert 0 annimmt. Daher wird es in diesem Sonderfall vorkommen, dass die Validierung kein Problem findet und mit der Ausführung des Codes begonnen wird. Bei der Code-Erzeugung selbst wird dann allerdings sehr wohl überprüft, ob eine Division durch Null versucht wird und mit einer entsprechenden Fehlermeldung und einem Programm-Abbruch Rechnung getragen.

PLAY SOUND | SPIELE MUSIK

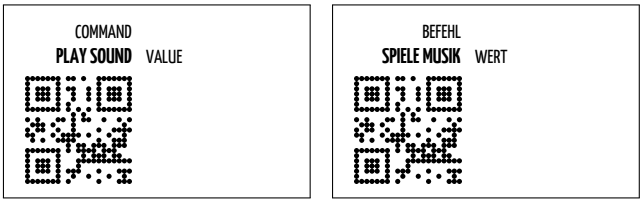


Abbildung 4.7: PLAY SOUND | SPIELE MUSIK

PLAY SOUND ist ebenfalls ein Befehl | Command, gleichzeitig aber ein Beispiel dafür, wie in *QRBASIC* Funktionen umgesetzt werden können. Dabei ist der Begriff "Funktion" allerdings sehr locker zu verstehen. Im Zusammenhang mit QRBASIC geht es einfach darum, dass damit Aktionen, Abbläufe, Aktivitäten etc. gesetzt werden können, die mit den absichtlich einfach gehaltenen Command-Karten nur schwer oder gar nicht realisiert werden könnten. PLAY SOUND ist eine solche Funktion, die aus dem Unterverzeichnis *media* (relativ zum Ausführungsort von qr.php) eine beliebige .mp3 Datei über einen dann aktivierten HTML5-Audioplayer wiedergibt.

Zusätzlich zu bereits bekannten Validierungsschritten wird natürlich ebenfalls geprüft, ob die abzuspielende Datei überhaupt existiert.

4.1.2 Operator-Karten

>,>=,<,<=,=,≠

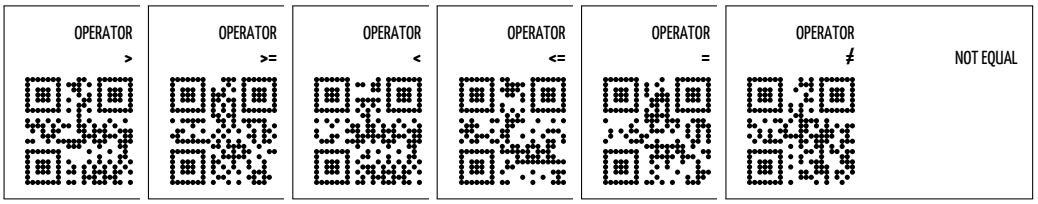


Abbildung 4.8: Operators

Die Vergleichsoperatoren werden nur innerhalb von Commands verwendet, müssen also nicht gesondert validiert werden und können auch bei der Codeerzeugung direkt übernommen werden.

4.1.3 Value- | Wert-Karten

VARIABLE & CONSTANT | KONSTANTE

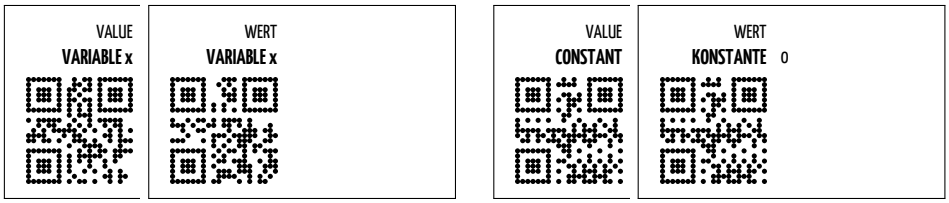


Abbildung 4.9: VARIABLE x & CONSTANT | KONSTANTE 0

Variablen und Konstanten werden ebenfalls nur im Rahmen von Befehlen verwendet. Die für sie notwendige Validierung und gegebenenfalls nötige Anpassungen des Codes ergeben sich dabei jeweils aus dem Zusammenhang und erfolgt daher im Rahmen der einzelnen Commands.

Einigen wird außerdem vielleicht schon aufgefallen sein, dass *QRBASIC* auf eine Typendefinition von Variablen verzichtet. Hier wird die praktische Eigenschaft von *PHP* genutzt, dass eine explizite Typendefinition nicht nötig ist. Der Typ einer Variablen ergibt sich automatisch aus dem Context, in dem diese verwendet wird. Diese Eigenschaft bezeichnet man auch als Type Juggling. [Gro15b]

4.2 Konfiguration

4.2.1 Server-Einstellungen

Zur leichteren Bedienbarkeit findet man im Hauptverzeichnis der serverseitigen PHP-Skripts, in der Datei *config.php*, die Möglichkeit, einige global wichtige Einstellungen zu verändern.
Einstellungen in *config.php*:

- Sprache:

```
$lang = 'en';
$lang = 'de';
```

  Entsprechend dieser Einstellung werden Kartennamen und Fehlermeldungen aus dem Sprachfile *language.php* geladen.

- Validierungs-Regeln:

```
$rules = 'relaxed';
$rules = 'strict';
```

Die Validierungs-Regeln legen fest, ob die bei der Validierung auf bestimmte Regeln "guten" Programmierens geprüft werden soll, oder eben nicht. Der wichtigste Unterschied ist hier auf jeden Fall die Definition von Variablen. Ist der Regelsatz ''strict" aktiv, müssen Variablen vor ihrer ersten Verwendung mittels des Befehls SETZEN | SET initialisiert werden. Wird darauf vergessen, bricht das Programm sofort ab. Der Regelsatz "relaxed" verzeiht diesen Umstand hingegen und jeder nicht bekannten Variable wir bei ihrer ersten Verwendung stattdessen automatisch der Wert 0 zugeordnet.

- Fehlerausgabe:

```
error_reporting(-1);
error_reporting(0);
```

error_reporting ist ein direkt von PHP implementierter Befehl, mit dem gesetzt werden kann, ob und welche PHP-Fehlermeldungen ausgegeben werden. Üblicherweise empfiehlt es sich im produktiven (Unterrichts-)Einsatz, die Fehlerberichterstattung von PHP komplett zu deaktivieren (0). Zur Fehlersuche und bei der Entwicklung kann hingegen eine Ausgabe aller Fehler und Warnungen durchaus hilfreich sein (-1).

- Maximale Ausführungszeit:

```
ini_set('max_execution_time', 10);
set_time_limit(10);
```

Ein (auch) von *QRBASIC* nicht gelöstes Problem ist die Erkennung/Verhinderung von Endlosschleifen. Um deren Auswirkungen zumindest zu beschränken und Probleme zu vermeiden, können der PHP-Befehl *set_time_limit* und die PHP-Laufzeitvariable *max_execution_time* genutzt werden. Wegen der eingeschränkten Möglichkeiten von *QRBASIC*, die (gewollt) rechenintensive (und trotzdem valide) Programme unwahrscheinlich machen, kann dieser Wert durchaus klein gewählt werden.
HINWEIS: Ob diese Werte von einem Skript überhaupt gesetzt/geändert werden können, hängt von den globalen Einstellungen der lokalen PHP-Installation ab. Im Zweifel müssen diese Einstellungen angepasst werden, sonst werden diese Einstellungen ignoriert.

4.2.2 Die Kamera-Applikation

Die Kamera-Applikation verfügt über kein Interface und entsprechend auch keine Einstellungsmöglichkeiten. Änderungen müssen daher direkt im Java-Quellcode und vor der Kompilierung vorgenommen werden. Der einzig offensichtliche Wert, der hier für zukünftige Einsätze angepasst werden muss, ist die Internet-Adresse, unter welcher die PHP-Skripts zu finden sind. Hierzu muss der String *output* am Ende der Hauptdatei QRBASIC.java entsprechend angepasst werden.

KAPITEL 5

Anwendungsbeispiele für den Unterricht

In diesem Kapitel soll anhand einiger praktische Beispieler der mögliche Einsatz von *QRBASIC* – zum Beispiel im Unterricht – erläutert werden. Neben einigen Einzelaufgaben finden sich auch Beispiele für Gruppenaufgaben, sowie ein Ansatz, der zwei anfangs unabhängig agierende Gruppen dazu motivieren soll, ihre Erkenntnisse untereinander auszutauschen.

In den folgenden Beispielen werden die Karten aus Platzgründen nur verkleinert dargestellt. Sämtliche für die Beispiellösungen dieser Aufgaben benötigte Karten finden sich im Anhang dieser Arbeit in voller Druckauflösung. Einzig für einige optionale bzw. alternative Lösungswege müssen gegebenenfalls noch zusätzliche Karten erzeugt und gedruckt werden. Dabei unterstützende Hilfs-Skripts werden ebenfals im Anhang verlinkt, die entsprechenden Karten in den Beispiellösungen mit einem * markiert. Für sämtliche Aufgaben gilt natürlich, dass nur die unter **Benötigte Karten** angeführten Karten zur Lösung verwendet werden sollten.

5.1 Einzelaufgaben

5.1.1 Hello World!

Man würde mit althergebrachten Programmiertraditionen brechen, wenn das erste *QRBASIC*-Beispiel nicht das klassische „Hello World!“ wäre. Aufgrund der Einsteigerfreundlichkeit von *QRBASIC* benötigt man dazu auch nur zwei Karten.

Beispieltext:
Nutze die beiden vorgegebenen Karten um “Hello World!” am Bildschirm darzustellen.

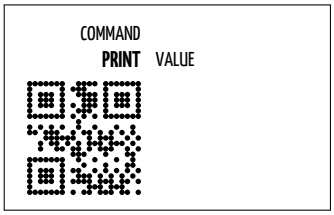
Benötigte Karten:

- 1x PRINT
- 1x CONSTANT „Hello World“

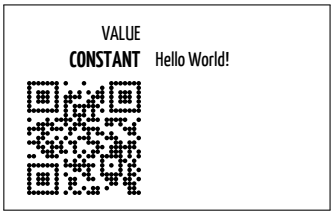
Lernziele:

- Verständnis des Befehls PRINT
- Verständnis von Konstanten

Beispiellösung:



(a) PRINT



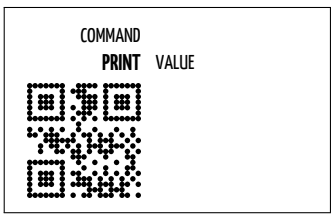
(b) CONSTANT „Hello World!“

Abbildung 5.1: Beispiellösung: *Hello World!*

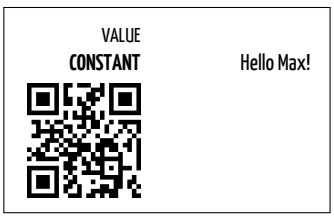
Alternative: Hello «you»!

Um diese Aufgabe etwas persönlicher zu gestalten – und solange es die Anzahl der Teilnehmer erlaubt – könnte man statt „Hello World“ natürlich auch persönliche Willkommensnachrichten für jeden Teilnehmer generieren, z.B. „Hello Max!“. Enstprechend personalisierte Konstanten-Karten können über den im PHP-Code inkludierten Karten-Generator schnell und einfach erzeugt und anschließend ausgedruckt werden.

Alternative Beispiellösung:



(a) PRINT



(b) CONSTANT „Hello Max!“*

Abbildung 5.2: Beispiellösung: *Hello «you»!*

5.1.2 Todays weather

Der nächste logische Schritt ist die Aufgabe dynamischer zu gestalten. Zu diesem Zweck führen wir einen neuen Kartentyp ein: Variablen. Ziel ist es, das aktuelle Wetter auszugeben, indem man zuerst der Variablen x eine Konstante – zum Beispiel “nice” – zuordnet.

Beispieltext:
Wirf einen Blick nach draußen. Wie ist das Wetter heute? Lege ein Programm, dass deine Antwort in Form einer Variablen auf dem Bildschirm darstellt.

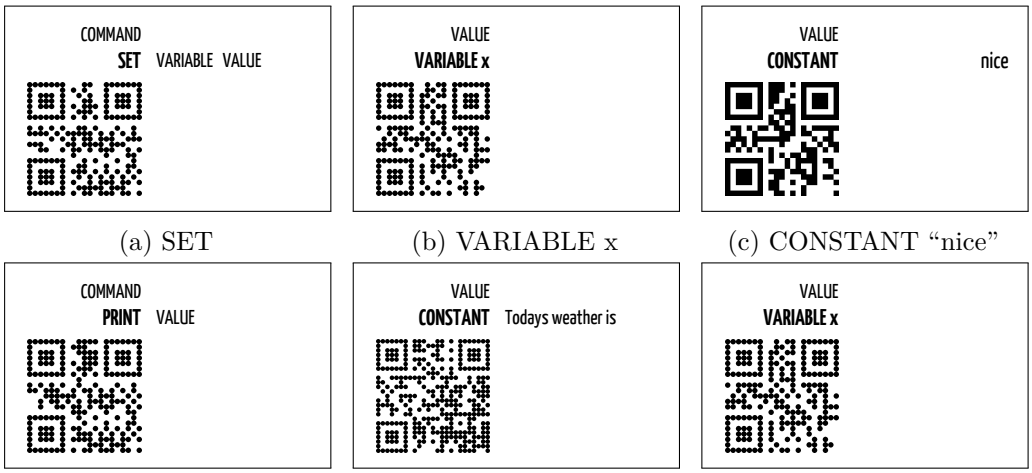
Benötigte Karten:

- 1x PRINT
- 1x SET
- 2x VARIABLE x
- 1x CONSTANT „Todays weather is“
- 1x CONSTANT „nice“
- 1x CONSTANT „cloudy“ *
- 1x CONSTANT „snowy“ *
- ... und weitere Wetterzustände *

Lernziel:

- Zuweisung von Variablen

Beispiellösung:



(a) SET (b) VARIABLE x (c) CONSTANT “nice”

(d) PRINT (e) CONSTANT ”Todays weather is” (f) VARIABLE x

Abbildung 5.3: Beispiellösung: *Todays weather*

5.1.3 Jukebox

Eine abschließende und ebenfalls sehr einfache (Einzel-)aufgabe soll dem Anwender ein Verständnis für Funktionen näher bringen. Der Begriff “Funktionen” ist dabei wie schon

in Kapitel 4 erwähnt recht großzügig zu verstehen. PLAY SOUND dient dazu, eine beliebige .mp3-Datei aus dem Unterverzeichnis *media* wiederzugeben. Aus urheberrechtlichen Gründen enthält *QRBASIC* keine entsprechenden Dateien, diese sind selbst zu organisieren und ins entsprechende Unterverzeichnis hochzuladen.

Beispieltext:
Wie wäre es mit etwas Musik? Spiele eine Lied aus der *QRBASIC*-Jukebox.

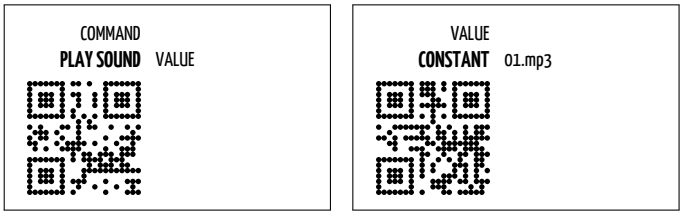
Benötigte Karten:

- 1x PLAY SOUND
- 1x CONSTANT "01.mp3"

Lernziel:

- Grundverständnis Funktionen

Beispiellösung:



(a) PLAY SOUND (b) CONSTANT „01.mp3“

Abbildung 5.4: Beispiellösung: *Jukebox*

5.2 Gruppenaufgaben

Bei etwas anspruchsvolleren Konzepten, aber auch größeren Nutzerzahlen, empfiehlt sich die Verwendung von Gruppenarbeiten. Ob man die Karten dabei der Gruppe als Ganzes überlässt, oder versucht, den Einzelnen zu fordern, indem jeder für "seine" Karte quasi verantwortlich ist, sollte von Fall zu Fall entschieden werden. Die beiden ersten Gruppenaufgaben sind auch ganz bewusst unterschiedlich schwer - das Beispiel für Gruppe A ist eindeutig leichter. Es empfiehlt sich daher, die Gruppen entsprechend aufzuteilen: absoulte Anfänger und/oder jene TeilnehmerInnen, die sich schon bei den Einzelaufgaben schwer getan haben, eher in Gruppe A; TeilnehmerInnen mit Vorwissen eher in Gruppe B.

HINWEIS: Die Beispiellösungen für diese Aufgaben wurden aus Platzgründen erst am Ende des Kapitels abgedruckt. Aus dem gleichen Grund wurden die Karten dabei auch nicht mehr einzeln beschriftet und auch teilweise übereinander gelegt dargestellt. Diese komprimierte Art der Auflegung soll und darf ganz ausdrücklich auch in der Praxis genutzt werden.

5.2.1 Gruppe A - Spiele Musik, wenn ...

Das erste Gruppenbeispiel stellt die Benutzer gleich vor zwei Probleme: Erstens soll ein Lied abgespielt werden, aber nur dann, wenn die Varable x (das "Guthaben") einen bestimmten Wert innehat. Das Verständnis und die korrekte Anwendung einer IF-Abfrage wird also nötig. Zweitens ist der gewünschte Wert nur einmal im verfügbaren Kartensatz vorhanden, kann also nicht gleichzeiitig zur Zuweisung und zur Abfrage verwendet werden. Durch Nutzung einer ADDITION kann das Ziel aber auch auf andere Weise erreicht werden.

Beispieltext:
Das zu erzeugende Programm soll das Lied "01.mp3" abspielen, aber nur dann, wenn die Variable x den Wert 2 hat.

Benötigte Karten:

- 1x ADDITION
- 1x END IF
- 1x IF
- 1x PLAY SOUND
- 1x SET
- 1x EQUAL =
- 3x VARIABLE x
- 2x CONSTANT 1
- 1x CONSTANT 2
- 1x CONSTANT "01.mp3"

Lernziele:

- Mathematische Operationen (ADDITION)
- IF-Abfragen

5.2.2 Gruppe B - Schleifen

Die zweite Gruppe hat es etwas schwerer. Sie soll sich zeitgleich mit dem Verständnis von Schleifen (konkret: einer WHILE-Schleife) und der mathematischen Operation MULTIPLICATION beschäftigen. Außerdem muß die Gruppe für die Lösung etwas mehr nachdenken, da im Beispieltext nichts vom Lösungsweg, sondern nur das gewünschte Ergebnis angegeben wird.

Beispieltext:
Gebt – nur unter Zuhilfenahme der vorhandenen Karten – die Zahlenfolge *2,4,8,16,32,64* aus.

Benötigte Karten:

- 1x END WHILE
- 1x MULTIPLICATION
- 1x PRINT
- 1x SET
- 1x WHILE
- 1x SMALLER <
- 4x VARIABLE y
- 1x CONSTANT 1
- 1x CONSTANT 2
- 1x CONSTANT 42

Lernziele:

- Mathematische Operationen (MULTIPLICATION)
- WHILE-Schleifen

5.2.3 Gemeinsame Gruppenaufgabe A & B - Potenzen

Ein abschließendes Beispiel führt beide Gruppen zusammen und bietet so die Möglichkeit, bereits gesammeltes Wissen auszutauschen. Aber auch neues muss gemeinsam gelernt werden, zum Beispiel die Möglichkeiten von FOR-Schleifen. Das gewünschte Programm ist nicht umfangreicher als die bisherigen, aber sicher komplexer, da man schon etwas nachdenken muss, um herauszufinden, wie man mit nur einer MULTIPLICATION potenziert und zum Beispiel, wann die Ausgabe der VARIABLE erfolgen muss, damit die Ausgabe den Erwartungen entspricht.

Beispieltext:
Gebt alle Potenzen von 2 zwischen 2^0 (=1) und 2^{42} (=4.398.046.511.104) aus!

Benötigte Karten:

- 1x ADDITION
- 1x END WHILE
- 2x SET
- 1x MULTIPLICATION
- 1x PRINT
- 1x WHILE
- 1x SMALLER <
- 3x VARIABLE x
- 3x VARIABLE y
- 1x CONSTANT 0
- 2x CONSTANT 1
- 1x CONSTANT 2
- 1x CONSTANT 42

Lernziele:

- Nutzung mehrerer Variablen
- FOR-Schleifen

5.3 Weitere Arbeit mit QRBASIC

Selbstverständlich können noch viele weitere Beispiele und Ideen mit *QRBASIC* umgesetzt werden. Mit fortschreitendem Verständnis und Wissen der NutzerInnen ist die Grenze, was Komplexität und Möglichkeiten betrifft aber natürlich trotzdem (schnell) erreicht. Ein Wechsel zu anderen, "vollwertigeren" Programmiersprachen ist daher nach einiger Zeit nötig und auch ausdrücklich vorgesehen. Gegen eine spätere Rückkehr zu *QRBASIC* – um zum Beispiel selbst eine neue (Befehls-)Karte zu ergänzen und die "Sprache" dadurch zu erweitern – spricht natürlich absolut nichts und würde auch den Autor dieser Arbeit sehr freuen...

Beispiellösungen der Gruppenarbeiten:

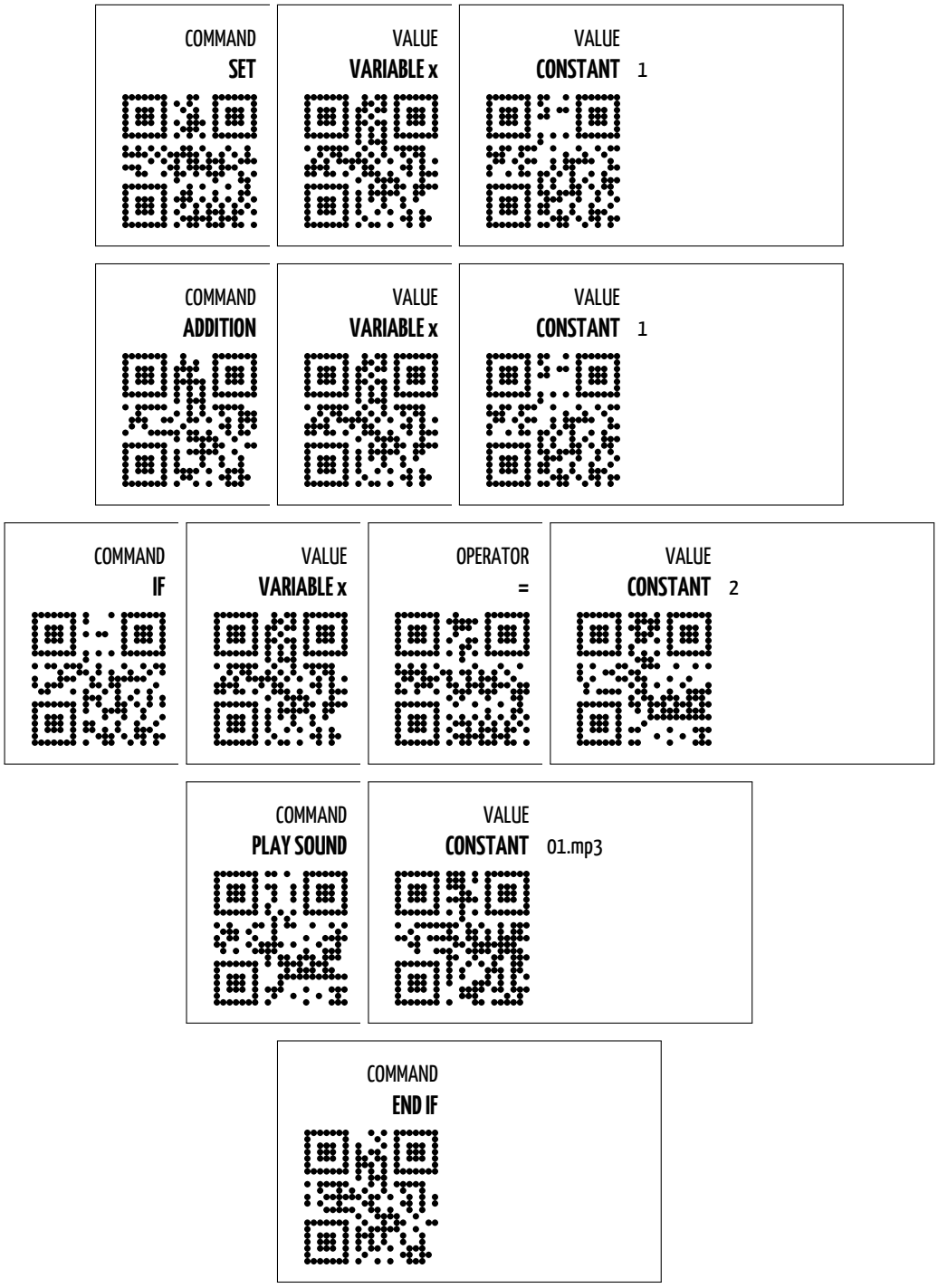


Abbildung 5.5: Beispiellösung: *Gruppenarbeit A*

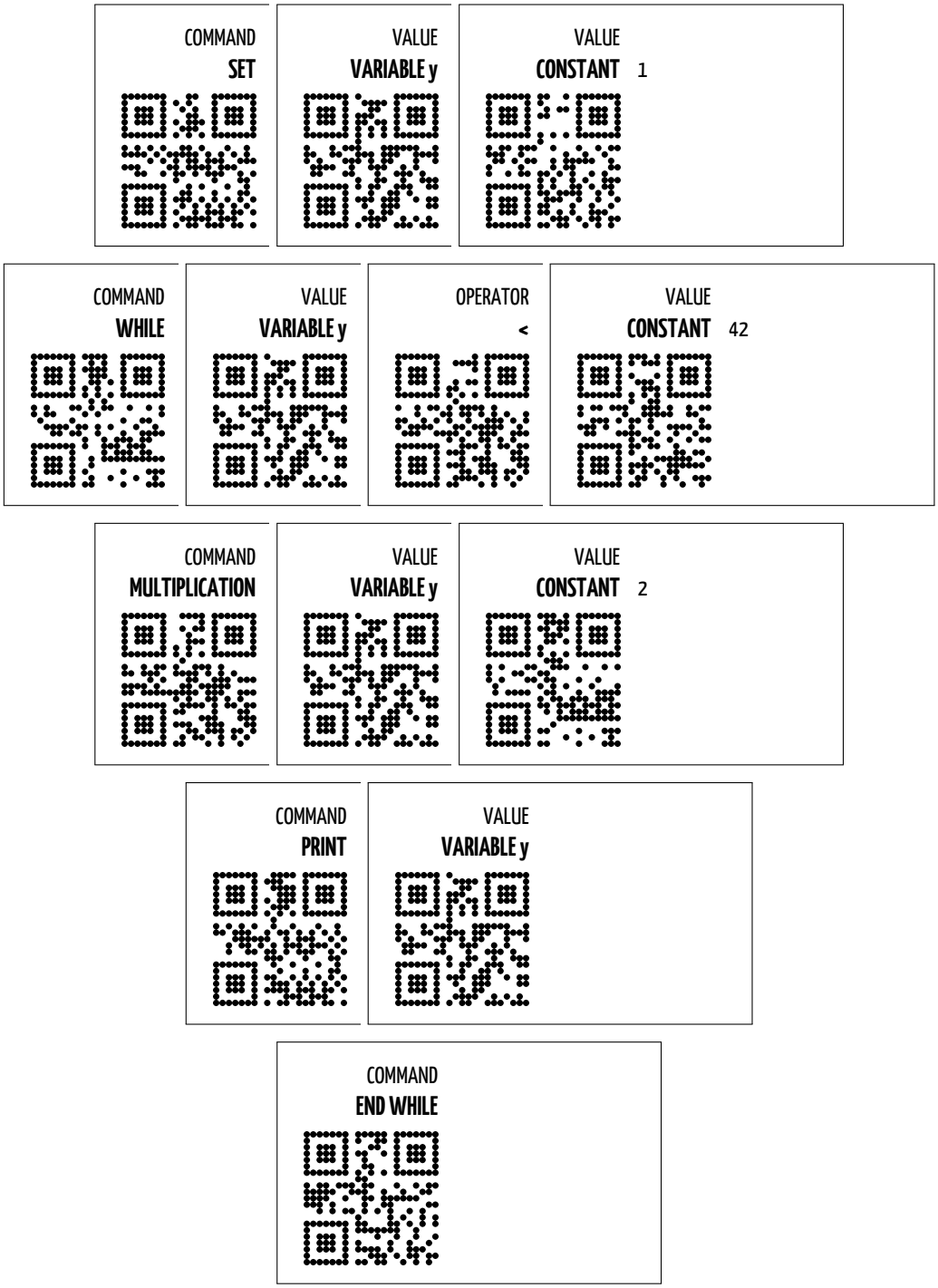


Abbildung 5.6: Beispiellösung: *Gruppenarbeit B*

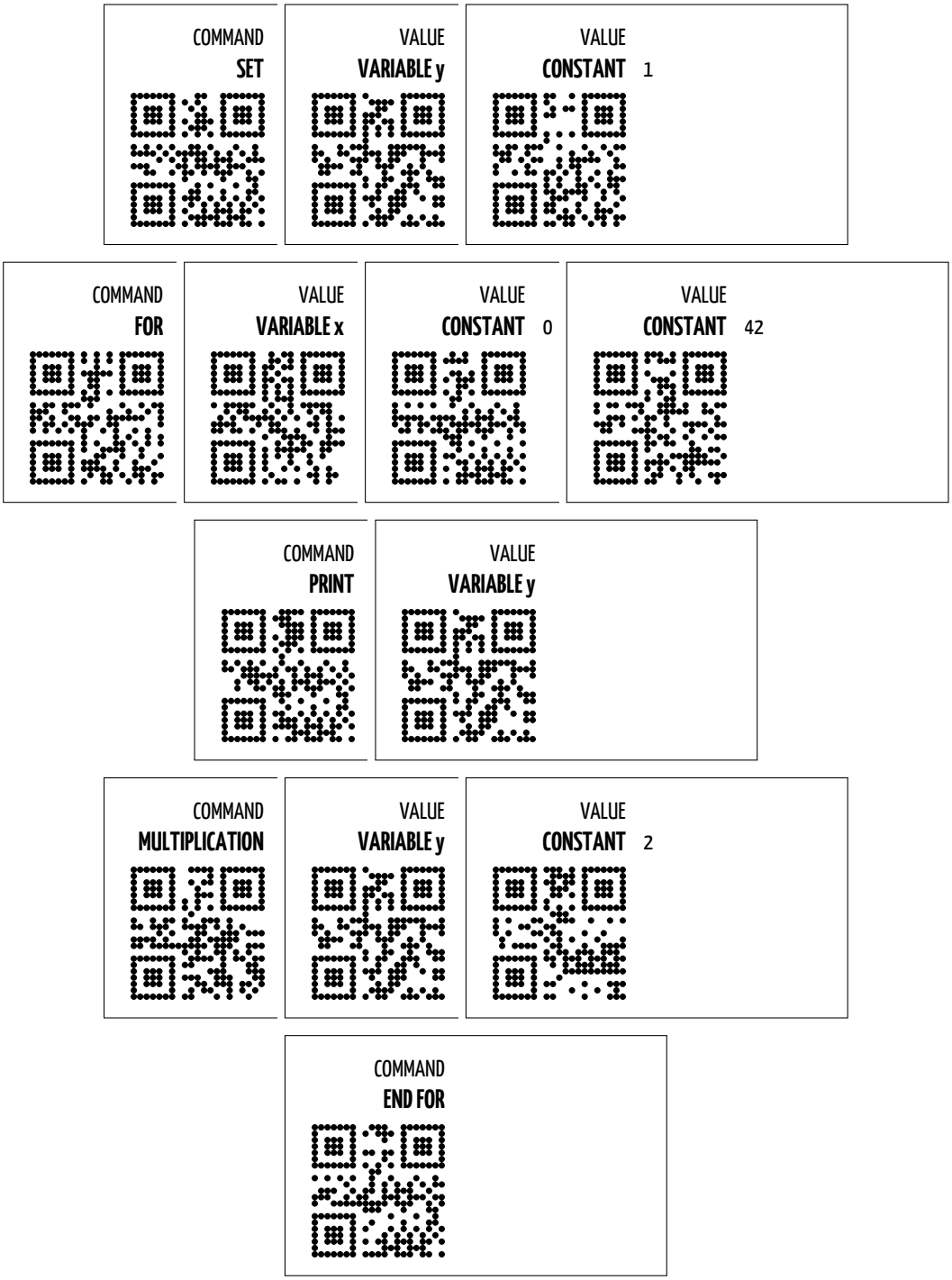


Abbildung 5.7: Beispiellösung: *Gruppenarbeit A & B*

KAPITEL 6

Evaluierung

QRBASIC soll intuitiv und einsteigerfreundlich sein, vor allem für erstmalige Nutzer. Welch bessere Evaluierung konnte es daher geben, als es einem möglichst breiten und differenzierten Publikum von (Studien)anfängerInnen zu präsentieren. Am BEGINNERS DAY der TU Wien für das Wintersemester 2015/2016, am 21. September 2015, wurde *QRBASIC* daher am Institut für Gestaltungs- und Wirkungsforschung, Arbeitsbereich HCI erstmals der Öffentlichkeit vorgestellt und vorgeführt.

6.1 Versuchsaufbau

Die verwendete Android-Kamera wurde auf einem Stativ befestigt und so zwischen zwei zusammengeschobene Tische gestellt, dass die Kamera auf eine zuvor platzierte Matte ausgerichtet und scharfgestellt war. Die Matte selbst ist/war für die Funktionalität von *QRBASIC* zwar nicht nötig, bot den Nutzern aber eine optische Hilfe, in welchem Bereich die Karten zu platzieren sind. Damit auch eventuelle Zuschauer mehr vom Ablauf sehen, wurde die Anzeige der Android-Kamera zudem auf einen Großbildfernseher gespiegelt, wodurch sowohl Kamerasicht als auch die anschließende Validierung und Ausführung für alle TeilnehmerInnen gut sichtbar war.

Schlußendlich wurde eine mittelgroße Anzahl unterschiedlichster QRBASIC-Karten produziert, um unterschiedlichste Programme realisieren zu können. Diese wurden in von einem 3D-Drucker produzierte Kartenhalterungen sortiert und so aufgestellt, dass sie für den Benutzer / die Benutzerin gut erreichbar waren.

Demonstrierte ein Teilnehmer oder eine Teilnehmerin entsprechendes Vorwisseen, wurde zudem auf zwei aufgehängte Mini-Poster verwiesen. Unter den dort abgedruckten Internetadressen (natürlich alternativ auch in Form eines QR Codes) konnten zusätzliche und/oder eigene Variablen und Konstanten definiert werden. Einfach am Smartphone

oder Handy aufgerufen, Karte generiert und dann einfach zu den anderen Karten gelegt – für die Kartenerkennung von *QRBASIC* überhaupt kein Problem.

Abbildung 6.1: Aufbau für den BEGINNER'S DAY

Über 150 StudienanfängerInnen ließen sich das Projekt an diesem Tag auf diese Weise präsentieren und sehr viele von ihnen versuchten sich auch selbst an einem Programm. Um die Intuivität von Konzept und Design zu prüfen, wurde die dazu nötige "Einschulung" ganz bewusst knapp gehalten. Ihnen wurde kurz die Idee und Motivation hinter *QRBASIC* erläutert und der generelle Ablauf anhand eines einfachen Beispiels – eine 'AUSGABE' in Kombination mit einem 'WERT' demonstriert. Dazu wurden ihnen nur zwei (Spiel-)Regeln mit auf den Weg gegeben:

1. Eine Befehls-Karte pro Zeile.

2. Folge den Anweisungen auf den Karten.

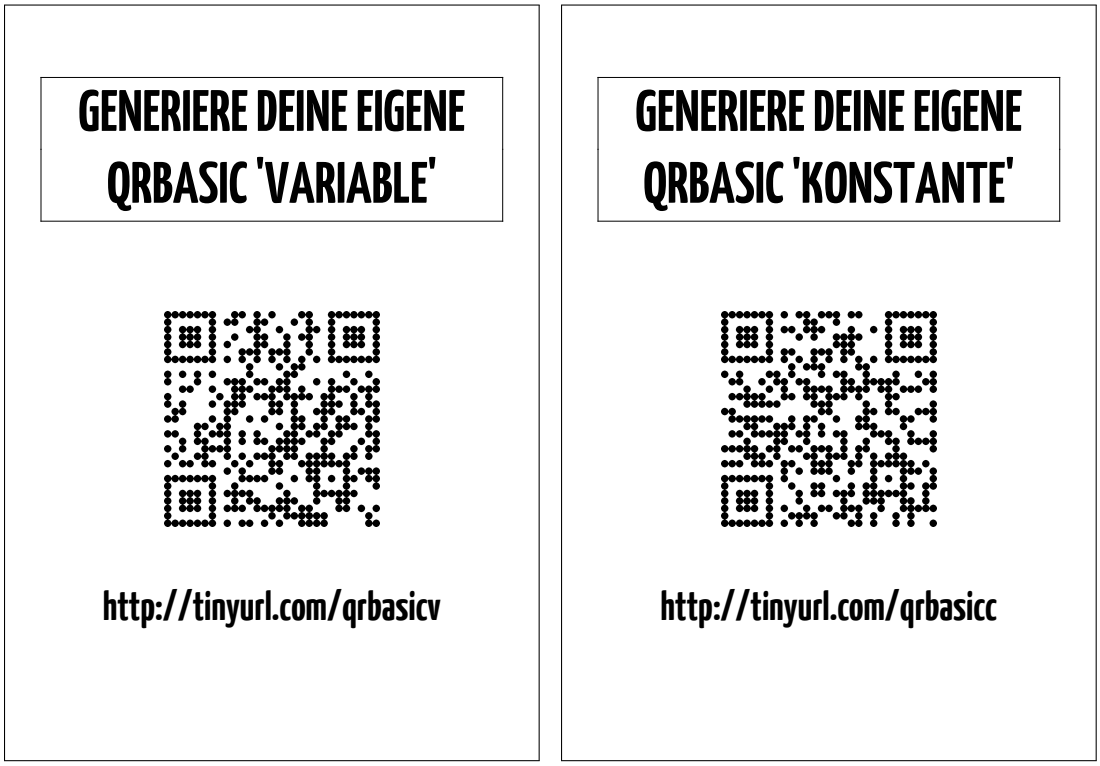


Abbildung 6.2: Mini-Poster “Generiere deine eigene QRBASIC VARIABLE | KONSTANTE

6.2 Informelle Evaluierung

Aufgrund der Gegebenheiten, vor allem der extrem knappen Zeitfenster, die pro Studentengruppe zur Verfügung standen, erfolgte die Evaluierung selbst größtenteils informell, durch Gespräche mit den StudienanfängerInnen, während sie selbst *QRBASIC* ausprobierten. Erfreulicherweise war das Feedback dabei ausschließlich positiv. Egal, ob kaum oder doch zumindest etwas Programmiervorwissen vorhanden war, die Studentinnen und Studenten waren voll des Lobes für den Ansatz und die meisten versuchten sich auch selbst an ein bis zwei “Programmen”, oft – und damit genau wie konzeptuell erhofft – auch gleich in Gruppen von zwei oder mehr Personen.

Erfreulicherweise traten auch keine nennenswerten Abstürze, Programmfehler oder vergleichbare Probleme auf. Einzig ein reiner Darstellungsfehler wurde im Zusammenhang mit dem “Spiele Musik”-Befehl entdeckt. Es wurde in der Darstellung verabsäumt, einen Zeilenumbruch nach dem eingeblendeten Mini-Player einzubauen, wodurch ein möglicherweise darauf folgender “Ausgabe”-Befehl außerhalb des sichtbaren Bereichs dargestellt wird.

Abbildung 6.3: StudienanfängerInnen beim Ausprobieren von QRBASIC

Verbesserunspotential liegt außerdem noch in der Fehlerbehandlung von Endlos-Schleifen, vor allem, wenn die Fehlerberichterstattung von PHP deaktiviert wird. Der gewählte Timeout von 45 Sekunden war eindeutig zu lang und in dieser Zeit wird nur ein weißer Bildschirm dargestellt. Der Timeout sollte deutlich kürzer sein und als neuer Standard-Wert wurden jetzt 10 Sekunden definiert. In Zukunft sollte zudem noch eine aussagekräftige Fehlermeldung bzw. bessere Fehlerbehandlung eingebaut werden. Diese noch zu verbessern, war im Zeitrahmen dieser Arbeit leider nicht mehr möglich. Die derzeitige Implementierung verlässt sich für das Erkennen von Endlosschleifen nämlich auf die (programmseitig gesetzte) maximale Skript-Ausführungszeit von PHP. Wird diese überschritten, löst dies einen Fatal Error aus, der die Skript-Ausführung sofort stoppt. Leider bedeutet dies aber auch, dass keine Möglichkeit existiert, diesen Fehler skriptseitig abzufangen und/oder zu behandeln. Die Erkennung (zu) langer Ausführungszeiten müsste daher komplett anders gelöst werden, was bis zur Abgabe dieser Arbeit aber nicht mehr zeitgerecht realisiert werden konnte.

6.3 Statistische Evaluierung

Obwohl – wie schon erwähnt – durch die Situation erschwert, wurden natürlich im Hintergrund auch Vorbereitung für eine minimale formelle Evaluierung getroffen. Zu diesem Zweck wurden sämtliche Interaktionen mit *QRBASIC* mitprotokolliert. Gespeichert wurde jeweils ein Zeitstempel sowie das vom QR-Reader erkannte Resultat und das Ergebnis der Validierung in Form des Zustandes "OK" oder "NOK" (not OK). Daraus ließen sich dann einige weitere Rückschlüsse ziehen:

QRBASIC wurde an diesem Tag insgesamt 121-mal ausgeführt. Dabei validierten mehr als zwei Drittel (81) der aufgelegten Programme auf Anhiebt korrekt. Dieser Wert ist mehr als zufriedenstellend, vor allem wenn man berücksichtigt, dass es am späteren Nachmittag leider zu einigen, nicht von den NutzerInnen verschuldeten Erkennungsproblemen durch die Kamera kam. Der dann herrschende Sonnenstand führte zu Spiegelungen auf den Karten, da der Versuchsaufbau wegen der Position des Großbildfernsehers nahe an einem Fenster erfolgen hatte müßen. In (geschätzt) 15-17 Fällen wurde daher von der Kamera-Applikation nicht alle Karten gleich beim ersten Versuch erkannt, wodurch ein eigentlich korrekt aufgelegtes Programm nicht korrekt validierte.

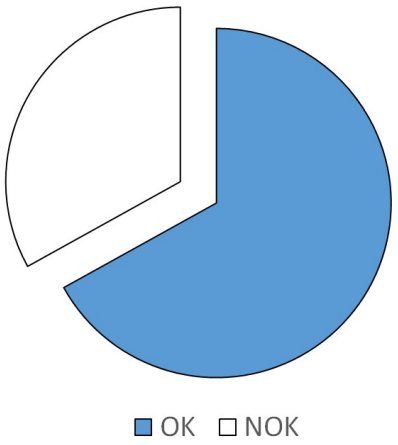


Abbildung 6.4: Verhältnis korrekter zu nicht korrekter *QRBASIC*-Syntax

Weiters konnten einige weitere Rückschlüsse über die Anzahl der dabei verwendeten Karten gezogen werden. Wie erwartet wurden bei den meisten Ausführungen (insgesamt 41) nur zwei Karten aufgelegt – die ersten "Gehvesuche". Erfreulich und fast schon überraschend sind aber die weiteren Häufungen. Die meisten TeilnehmerInnen zeigten sich nämlich experimentierfreudig und legten in weiterer Folge gleich deutlich komplexere/längere Programme auf. Auf dem zweiten Platz liegen dann nämlich gleich Programme mit sieben (25x) Karten und Platz Drei teilen sich, mit jeweils neun Aufrufen, Versuche mit sechs oder neun Karten. Insgesamt wurden sogar 51-mal (und damit bei über 40 Prozent) sieben Karten oder mehr verwendet.

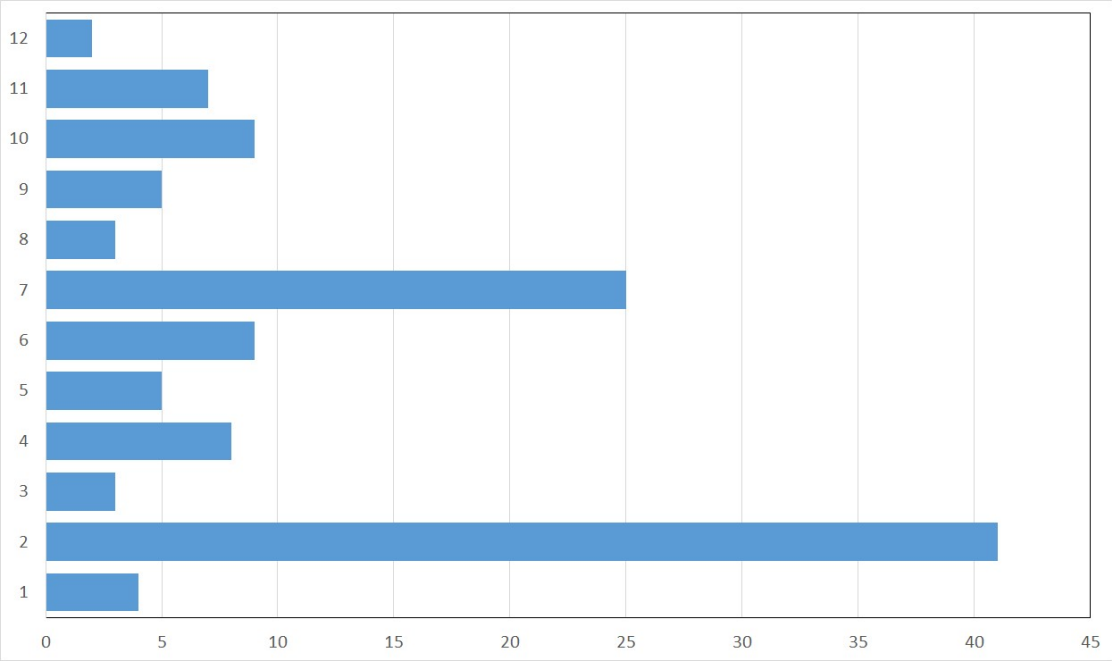


Abbildung 6.5: Verteilung der Anzahl der verwendeten Karten

6.4 Fazit Evaluierung

Auch mit den eingeschränkten Möglichkeiten einer formellen Evaluierung scheint es offensichtlich, dass die Ziele von *QRBASIC* erreicht wurden. Das Konzept ist einfach verständlich und intuitiv und was vielleicht am wichtigsten ist: Es hat den Probanden Spaß gemacht und von Berührungsängsten mit Programmieren war nichts (mehr) zu merken. Natürlich können erst weitere Einsätze und Versuche diesen Ersteindruck wirklich bestätigen und es gibt natürlich trotzdem noch viel Verbesserungspotential. Insgesamt kann man mit den bisherigen Ergebnissen aber durchaus sehr zufrieden sein.

KAPITEL 7

Future Work

Die vorliegende Version von *QRBASIC* mag zwar bereits viel mehr als ein reiner „proof of concept“ sein, steht aber trotzdem noch gerade erst am Anfang was die potenziellen Möglichkeiten betrifft und ist damit eine ideale Grundlage für zukünftige Projekte, Forschung und Erweiterungen. Der modulare Aufbau des Gesamtkonzepts und die Offenheit des Quellcodes (online und im Anhang dieser Arbeit zu finden) wurden ganz bewusst gewählt, um die Möglichkeit zu bieten, dass auch andere am Konzept *QRBASIC* weiterarbeiten. In diesem abschließenden und kurzen Kapitel sollen darum einige Ideen angerissen werden, die sich im Laufe der Entwicklung ergeben haben, aus Zeitgründen aber für die vorliegende Version nicht umgesetzt werden konnten bzw. dem ursprünglichen Forschungsziel sogar entgegengelaufen wären. Die einzelnen Ideen wurden vage drei Kategorien von zukünftigen Forschungsgebieten zugeordnet:

7.1 Erweiterungen

Die wohl offensichtlichste Möglichkeit *QRBASIC* zu erweitern findet man in der Sprache selbst. Der vorliegende Befehlssatz an Instruktionen erlaubt zwar bereits die Demonstration einiger interessanter Konzepte, reizt die theoretischen Möglichkeiten aber natürlich alles andere als aus. Schon eine einfache Erweiterung des Befehlsatzes, unter alleiniger Verwendung der bereits vorhandenen Konzepte und Konstrukte, kann bereits eine enorme Bereicherung darstellen.

Eine ganz konkrete – und ursprünglich auch bereits geplante, aber aus Zeitgründen dann doch wieder gestrichene – Wunsch-Erweiterung dieses Projekts wäre allerdings die Zusammenführung von QR-Code-Erkennung bzw. -Verarbeitung & Ausführung der erkannten Instruktionen auf einem Endgerät bzw. sogar in einer Applikation. Die einfachste Variante wäre vermutlich, einen entsprechenden Server auf dem Android-Endgeräte zu installieren, der die erkannten Code-Zeilen auch gleich lokal interpretieren kann. Geeignete

Umsetzungen sind bereits teilweise vorhanden (zum Beispiel: *Server for PHP* [And15]) und könnten eventuell für diesen Zweck adaptiert werden.

Weitere absolute Wunsch-Erweiterungen sind im Bereich der Hardware-Anbindung zu sehen. Wie mehrfach betont, hängen Einsteigerfreundlichkeit und spielerischer Ansatz oft zusammen. Je mehr Aktion und Reaktion der Lernende mit seinem Code anregen kann, desto besser. Schon eine Anbindung und Nutzbarmachung einfacher Lichtquellen, Aktuatoren und Sensoren würde dem potenziellen Lernerfolg von *QRBASIC* sicherlich extrem zuträglich sein.

7.2 Portierungen

Am weitläufigsten und damit gleichzeitig auch am schwersten zu definieren oder einzuschränken – was auch gar nicht das Ziel dieser Aufzählungen sein soll – sind natürlich die Möglichkeiten QRBASIC oder auch nur Teile davon zu portieren. Denn alle drei Kernbereiche des Projekts – Konzept und Idee an sich, die Android-Applikation und der Interpreter – lassen sich natürlich nach Belieben auf andere Technologien, Sprachen und Systeme übertragen.

Statt QR-Codes könnte man zum Beispiel andere existierende 2D-Codes (DataMatrix, Aztec, mCode, Beetagg etc. pp.) verwenden oder sogar eine eigene Symbologie z.B. in der Form von Piktogrammen entwickeln. Auch ein Ansatz in Richtung 3D-Codes (zum Beispiel: *PM Code* [CC13]) könnte interessant sein. In diesen Fällen wird der Informationsgehalt noch einmal deutlich gesteigert, indem ein normaler 2D-QR-Code quasi um eine dritte Dimension erweitert wird. Im Fall von *PM Code* wird dies dadurch erreicht, dass die einzelnen Pixel/Punkte des QR-Codes zusätzlich noch farbcodiert werden.

Abbildung 7.1: Verschiedene Arten von 2D-Barcodes (Quelle: *Wikimedia Commons)*

Es kann natürlich auch ein kompletter Weggang von optischen Codes angestrebt werden, wodurch der Bedarf einer entsprechenden Erkennung natürlich komplett eliminiert wird.

Alternativen könnten hier unter anderem im Bereich der NFC- und RFID-Technologien gefunden werden. Solche Ansätze gehen allerdings mit ganz neuen Problemen bezüglich der Verortung einher.

Android als Hardware-Plattform für die Code-Erkennung ist natürlich ebenfalls alles andere als verbindlich. Portierung auf andere mobile Plattformen wie iOS, Windows Phone und Co. sind dabei ebenso denkbar wie eine Rückkehr zu traditionellen Desktop-Lösungen, solange mit ihnen entsprechende Bildverarbeitung möglich ist.

Die verwendeten Programmiersprachen sind natürlich ebenfalls nur als Empfehlungen zu verstehen. Es wurde bereits erläutert, warum die App, welche für die QR-Code-Erkennung zuständig ist in Java geschrieben wurde und warum die Weiterverarbeitung des Codes im PHP erfolgt. Sicherlich gibt es Ansätze und Ideen, die den Einsatz anderer Sprachen sinnvoller machen würden.

7.3 Verbesserung & Optimierung

Ein enorm wichtiger, aber leider oft vernachlässigter Bereich ist schlussendlich die Verbesserung und Optimierung des eigentlichen Grundsystems. Natürlich ist die Entwicklung komplett neuer Konzepte und Ansätze für viele oft interessanter als die Optimierung existierender – und möglicherweise auch noch fremder – Systeme, aber auch dies gehört zu den Aufgaben eines Forschers. Im Fall von *QRBASIC* besteht sicher noch Möglichkeit zur Verbesserung, wenn nicht sogar akuter Optimierungsbedarf. Es wäre auch die ganz große Ausnahme, dass ein komplett neues Programm nicht noch deutliches Verbesserungspotenzial bietet. Auch hier bieten die einzelnen Teilbereiche ganz verschiedene Möglichkeiten der Verbesserung:

Vor allem die angebotene Android-Applikation – nicht zuletzt wegen der mangelnden Java-Erfahrung des Autors – ist in vielen Bereichen sicher noch optimierbar. Wie aber schon in Kapitel 3 festgestellt, bietet vor allem die gleichzeitige Erkennung multipler QR-Codes, sowie deren Verortung, noch viel deutlicheres Verbesserungspotenzial. Die dafür verwendete, aber aus dritter Hand stammende 1D/2D Barcode-Bildverarbeitungs-Bibliothek *ZXing* war dabei zwar eine enorme Hilfe, hat aber natürlich ebenfalls noch Optimierungsbedarf. Die Weiterentwicklung von ZXing im Auge zu behalten und/oder selbst am Projekt mitzuarbeiten und entsprechende Verbesserungen wiederum in *QRBASIC* zu implementieren stellt damit ebenfalls eine Möglichkeit für weitere Tätigkeit im Sinne dieser Arbeit dar. Gleiches gilt natürlich für die Implementierung einer komplett anderen Bildverarbeitungs- und Verortungs-Lösung – egal ob bereits existierend oder neu geschaffen.

Auch der PHP-Code , der für die Umsetzung der erkannten Befehle zuständig ist, könnte sicher effizienter geschrieben sein. Hier ist allerdings festzuhalten, dass manches was ineffizient schein mag und vielleicht nicht der „best practice“ entspricht, möglicherweise

absichtlich so umgesetzt wurde. Da sich die Nutzer auf Wunsch auch den tatsächlich ausgeführten Code anzeigen lassen können – in der Hoffnung, dass ihnen der Umstieg auf komplexere, „richtige“ Programmiersprachen dadurch erleichter wird – wurde teilweise ganz bewusst auf Eleganz und Effizienz verzichtet, solange es der Lesbarkeit zuträglich war.

7.4 To be continued ...

Diese Aufzählung erhebt natürlich trotzdem nicht den geringsten Anspruch auf Vollständigkeit. Sie ist bestenfalls als Denkanstoß zu verstehen, der hoffentlich andere dazu motiviert, die hier begonnene Arbeit aufzugreifen und fortzusetzen.

Literaturverzeichnis

[And15] Tautvydas Andrikys. Server for PHP. https://play.google.com/store/apps/details?id=com.esminis.server.php, 2015. [Online; Zugriff 24. Mai 2015].

[Awa14] Develop Awards. 2014 Winners. http://www.developawards.com/winners, 2014. [Online; Zugriff 29. März 2015].

[Bac13] Daniel Bachfeld. Mindstorms EV3: Lego kündigt neuen Robotikbaukasten an. http://www.heise.de/hardware-hacks/meldung/Mindstorms-EV3-Lego-kuendigt-neuen-Robotikbaukasten-an-1778288.html, 2013. [Online; Zugriff 18. Jänner 2015].

[BH11] Jens Mönig Brian Harvey. BYOB Reference Manual 3.1. https://snap.berkeley.edu/BYOBManual.pdf, 2011. [Online; Zugriff 08. Februar 2015].

[Blo14] GameMaker Blog. Exclusive Interview With The Creator Of GameMaker. http://gamemakerblog.com/2014/06/15/exclusive-interview-with-the-creator-of-gamemaker/, 2014. [Online; Zugriff 28. März 2015].

[Bri13] J.R. Briggs. *Python for Kids: A Playful Introduction to Programming*. No Starch Press Series. No Starch Press, 2013.

[CC13] Ltd C.I.A Co. PM Code. http://pmcode.co-site.jp/pmcode/pmcode_jp.html, 2013. [Online; Zugriff 24. Mai 2015].

[CDCD09] D. Cornish, M.D.F.A.D.D. David Cornish, and D. Dukette. *The Essential 20: Twenty Components of an Excellent Health Care Team*. Rosedog Press, 2009.

[com11] Heinrich Vaske / computerwoche.de. Informatikermangel - die Zeitbombe tickt. http://www.computerwoche.de/a/informatikermangel-die-zeitbombe-tickt,2366032, 2011. [Online; Zugriff 15. Dezember 2014].

[dO05] Rodrigo B. de Oliveira. The boo Programming Language. http://boo.codehaus.org/BooManifesto.pdf, 2005. [Online; Zugriff 29. März 2015].

[DSW14] DACH-Scratch-Wiki. Build Your Own Blocks (Scratch Modifikation). http://wiki.scratch-dach.info/index.php?title=Build_Your_Own_Blocks_%28Scratch_Modifikation%29, 2014. [Online; Zugriff 08. Februar 2015].

[dW12] dapd / WAZ.de. Informatiker-Mangel lässt Sorge um Wettbewerbsfähigkeit steigen. http://www.derwesten.de/nachrichten/informatiker-mangel-laesst-sorge-um-wettbewerbsfaehigkeit-steigen-id6428714.html, 2012. [Online; Zugriff 15. Dezember 2014].

[Ead14] Lisa Eadicicco. The Best Programming Languages Every Beginner Should Learn. http://uk.businessinsider.com/best-programming-languages-2014-12, 2014. [Online; Zugriff 28. März 2015].

[fE15a] WIKIPEDIA Die freie Enzyklopädie. 2D-Code. https://de.wikipedia.org/w/index.php?title=2D-Code&oldid=138681247, 2015. [Online; Zugriff 11. Oktober 2015].

[fE15b] WIKIPEDIA Die freie Enzyklopädie. QR-Code. https://de.wikipedia.org/w/index.php?title=QR-Code&oldid=146101424, 2015. [Online; Zugriff 11. Oktober 2015].

[For09] J. Ford. *Getting Started with Game Maker*. Cengage Learning, 2009.

[Fou04] The Apache Software Foundation. Apache License, Version 2.0. http://www.apache.org/licenses/LICENSE-2.0.html, 2004. [Online; Zugriff 17. Oktober 2015].

[Gam15] Tingly Games. Team. http://company.tinglygames.com/about-us/team/, 2015. [Online; Zugriff 28. März 2015].

[Gee15] Chryssa Aliferi / Java Code Geeks. Android Barcode and Qr Scanner Example. http://examples.javacodegeeks.com/android/android-barcode-and-qr-scanner-example/, 2015. [Online; Zugriff 03. März 2015].

[Gro14a] The LEGO Group. Downloads - Mindstorms LEGO.com. http://www.lego.com/de-de/mindstorms/downloads/download-software, 2014. [Online; Zugriff 18. Jänner 2015].

[Gro14b] The LEGO Group. Lerne programmieren - Mindstorms LEGO.com. http://www.lego.com/de-de/mindstorms/learn-to-program, 2014. [Online; Zugriff 18. Jänner 2015].

[Gro14c] The LEGO Group. Mindstorms LEGO.com. http://www.lego.com/de-de/mindstorms, 2014. [Online; Zugriff 18. Jänner 2015].

[Gro15a] The PHP Group. PHP: Grundlegendes - Manual. http://php.net/manual/de/language.variables.basics.php, 2015. [Online; Zugriff 06. September 2015].

[Gro15b] The PHP Group. PHP: Type Juggling - Manual. http://php.net/manual/de/language.types.type-juggling.php, 2015. [Online; Zugriff 06. September 2015].

[Guo14] Philip Guo. Python is Now the Most Popular Introductory Teaching Language at Top U.S. Universities. http://cacm.acm.org/blogs/blog-cacm/176450-python-is-now-the-most-popular-introductory-teaching-language-at-top-us-universities/fulltext, 2014. [Online; Zugriff 16. August 2015].

[Hen14a] Alan Henry. Five Best Programming Languages for First-Time Learners. http://lifehacker.com/five-best-programming-languages-for-first-time-learners-1494256243, 2014. [Online; Zugriff 28. März 2015].

[Hen14b] Alan Henry. Whatś the Best Programming Language for First-Time Learners? http://lifehacker.com/whats-the-best-programming-language-for-first-time-lea-1493135749, 2014. [Online; Zugriff 28. März 2015].

[Inc13a] Denso Wave Inc. About the patent. https://de.wikipedia.org/w/index.php?title=2D-Code&oldid=138681247, 2013. [Online; Zugriff 11. Oktober 2015].

[Inc13b] Denso Wave Inc. What is a QR Code? http://www.qrcode.com/en/about/, 2013. [Online; Zugriff 11. Oktober 2015].

[Inc15] Denso Wave Inc. History of QR Code. http://www.qrcode.com/en/history/, 2015. [Online; Zugriff 11. Oktober 2015].

[Lab10] Lifelong Kindergarten Group / MIT Media Lab. DesignBlocks. http://www.designblocks.net, 2010. [Online; Zugriff 08. Februar 2015].

[Lab15a] Lifelong Kindergarten Group / MIT Media Lab. Lifelong Kindergarten. https://llk.media.mit.edu/, 2015. [Online; Zugriff 21. März 2015].

[Lab15b] MIT Media Lab. Über Scratch. http://scratch.mit.edu/about/, 2015. [Online; Zugriff 21. März 2015].

[Lab15c] MIT Media Lab. Für Eltern. http://scratch.mit.edu/parents/, 2015. [Online; Zugriff 21. März 2015].

[Lab15d] MIT Media Lab. Offline-Editor für Scratch 2. http://scratch.mit.edu/scratch2download/, 2015. [Online; Zugriff 21. März 2015].

[Lab15e] MIT Media Lab. Scratch Project Editor. http://scratch.mit.edu/projects/editor/, 2015. [Online; Zugriff 21. März 2015].

[LLC14] Robot Turtles LLC. Robot Turtles. http://www.robotturtles.com/, 2014. [Online; Zugriff 10. Jänner 2014].

[Mag12] Gamasutra / Game Developer Magazine. Mobile game developer survey leans heavily toward iOS, Unity. http://www.gamasutra.com/view/news/169846/Mobile_game_developer_survey_leans_heavily_toward_iOS_Unity.php, 2012. [Online; Zugriff 29. März 2015].

[Mic12] Nikos Michalakis. Teaching the Ḧow to train your robot c̈lass. http://drtechniko.com/2012/04/21/teaching-the-how-to-train-your-robot-class/, 2012. [Online; Zugriff 21. März 2015].

[Mic14a] Nikos Michalakis. DrTechniko‘s Children‘s Stories and Games. http://drtechniko.com/, 2014. [Online; Zugriff 21. März 2015].

[Mic14b] Nikos Michalakis. How To Train Your Robot To Jump. http://drtechniko.com/2014/02/04/how-to-train-your-robot-to-jump/, 2014. [Online; Zugriff 21. März 2015].

[Moo14] Bo Moore. No coding required: How new designers are using GameMaker to create indie smash hits. http://www.pcgamer.com/no-coding-required-how-new-designers-are-using-gamemaker-to-create-indie-smash-hits/, 2014. [Online; Zugriff 28. März 2015].

[Mor14] N. Morgan. *JavaScript for Kids: A Playful Introduction to Programming*. No Starch Press, 2014.

[New14] U.S. News. Best Computer Science Programs. http://grad-schools.usnews.rankingsandreviews.com/best-graduate-schools/top-science-schools/computer-science-rankings, 2014. [Online; Zugriff 16. August 2015].

[oCaB14] University of California at Berkeley. SNAP! (Build Your Own Blocks). http://snap.berkeley.edu/, 2014. [Online; Zugriff 08. Februar 2015].

[oSU15] Faculty of Science/Universiteit Utrecht. former colleagues - group Games and Virtual Worlds. http://www.cs.uu.nl/staff/old/IDX/mho.html, 2015. [Online; Zugriff 28. März 2015].

[PHJ95] A. D. Pellegrini, Patti Davis Huberty, and Ithel Jones. The Effects of Recess Timing on Children's Playground and Classroom Behaviors. *American Educational Research Journal*, 32(4):845–864, 1995.

[Red14] Lehrer-Online Redaktion. Umfrage: Informatik-Unterricht soll Standard werden. http://www.lehrer-online.de/1052194.php, 2014. [Online; Zugriff 15. Dezember 2014].

[Rob14] Judy Robertson. Rethinking how to teach programming to newcomers. *Commun. ACM*, 57(5):18–19, May 2014.

[Sha13a] Dan Shapiro. Robot Turtles - Kickstarter. https://www.kickstarter.com/projects/danshapiro/robot-turtles-the-board-game-for-little-programmer, 2013. [Online; Zugriff 10. Jänner 2015].

[Sha13b] Dan Shapiro. Robot Turtles Standard Rules. https://docs.google.com/a/danshapiro.com/document/d/1sUvO56g-quVt-PXdazk9qT_oqun16hTquMZX1tuWlz8/edit?pli=1, 2013. [Online; Zugriff 10. Jänner 2015].

[SVM10] swissinfo.ch Susan Vogel-Misicka. Switzerland faces a lack of IT professionals. http://www.swissinfo.ch/eng/switzerland-faces-a-lack-of-it-professionals/28833064, 2010. [Online; Zugriff 15. Dezember 2014].

[Tea14] The Catrobat Team. Pocket Code Website. https://pocketcode.org, 2014. [Online; Zugriff 08. Februar 2015].

[Tea15a] ZXing Team. ZXing Class QRCodeMultiReader. http://zxing.github.io/zxing/apidocs/com/google/zxing/multi/qrcode/QRCodeMultiReader.html, 2015. [Online; Zugriff 17. Oktober 2015].

[Tea15b] ZXing Team. ZXing ("Zebra Crossing") project home. https://github.com/zxing/zxing, 2015. [Online; Zugriff 17. Oktober 2015].

[Tec15] Unity Technologies. Unity - Fast Facts. https://unity3d.com/public-relations, 2015. [Online; Zugriff 29. März 2015].

[TEK03] Voices of CIS Thomas E. Kurtz. Thomas E. Kurtz. http://cis-alumni.org/TKurtz.html, 2003. [Online; Zugriff 16. August 2015].

[u.a11] SeptimusHeap [Synonym] u.a. Panther - based on Scratch. http://pantherprogramming.weebly.com, 2011. [Online; Zugriff 08. Februar 2015].

[Val13] Laurens Valk. EV3 and NXT: Difference and Compatibility. http://robotsquare.com/2013/07/16/ev3-nxt-compatibility, 2013. [Online; Zugriff 18. Jänner 2015].

[Wei14] E. Weinstein. *Ruby Wizardry: An Introduction to Programming for Kids*. No Starch Press, 2014.

[Wik14] Unify Community Wiki. UnityScript versus JavaScript. http://wiki.unity3d.com/index.php?title=UnityScript_versus_JavaScript&oldid=18272, 2014. [Online; Zugriff 29. März 2015].

[Wik15] Wikipedia. Grafische Programmiersprache — Wikipedia, die freie Enzyklopädie. http://de.wikipedia.org/w/index.php?title=Grafische_Programmiersprache&oldid=137340046, 2015. [Online; Zugriff 21. März 2015].

Anhang

QRBASIC Demokarten

Auf den folgenden Seiten findet sich ein Satz deutscher Demokarten für *QRBASIC*. Enthalten sind sämtliche Basisfunktionen sowie einige Beispielkonstanten und -variablen mit denen die in Kapitel 5 angeführten Unterrichts-Beispiele umgesetzt werden können.

Die gebundene Ausgabe dieser Arbeit enthält zudem einige zusätzliche Folien, in die enstprechend bedruckte PVC-Karten einsortiert wurden. Werden die Einzelseiten auf A4 ausgedruckt, sollten die einzelnen Karten dem ISO-Standard 7816 (85,60 mm x 53,98 mm) entsprechen. Sollen einzelne Karten zum Beispiel via Kartendrucker gedruckt werden, wird empfohlen, den inhaltlich identen Kartensatz zu verwenden, der mittels *QRBASIC*-Quellcode-Paket (wahlweise auch auf Englisch) erzeugt werden kann.

BEFEHLE

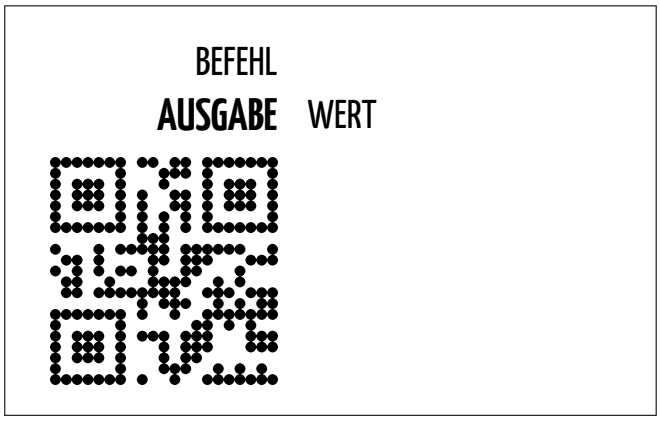


Abbildung 1: Befehl AUSGABE

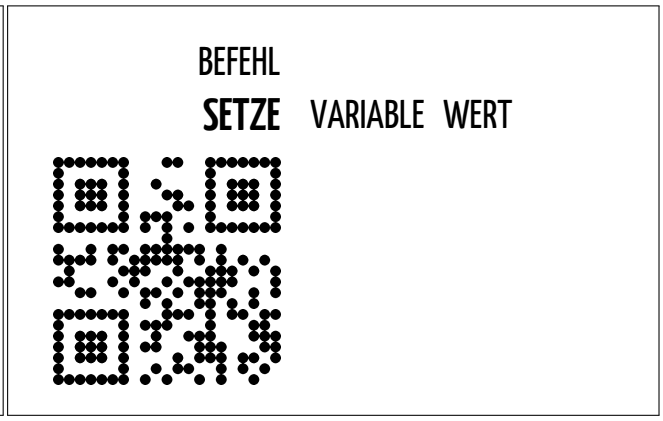


Abbildung 2: Befehl SETZE

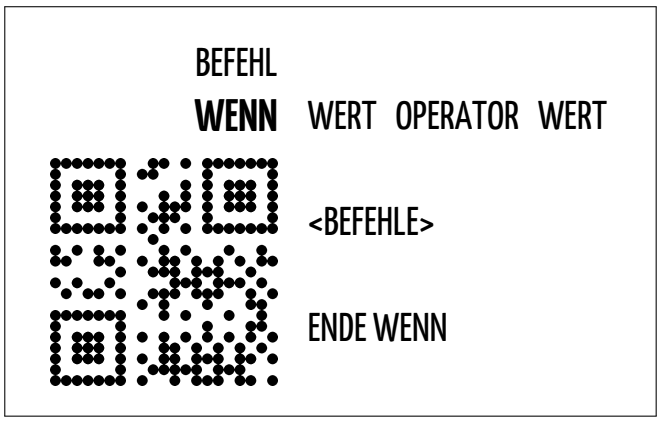


Abbildung 3: Befehl WENN

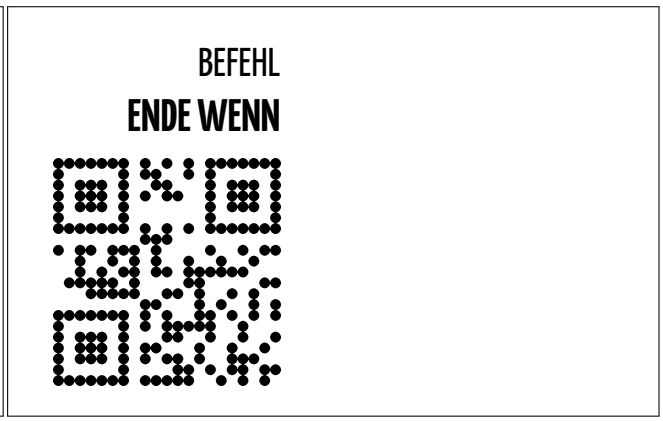


Abbildung 4: Befehl ENDE WENN

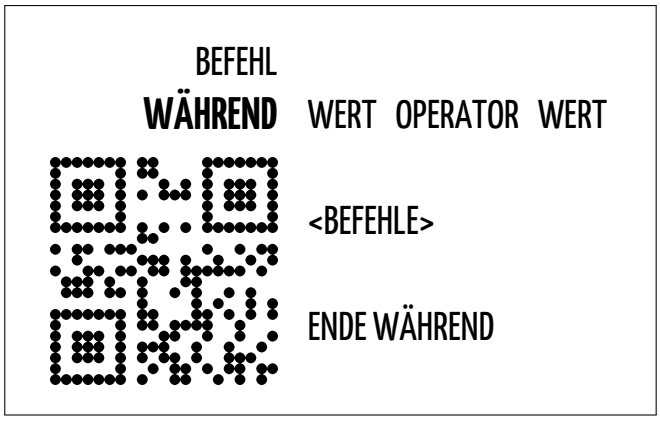


Abbildung 5: Befehl WÄHREND

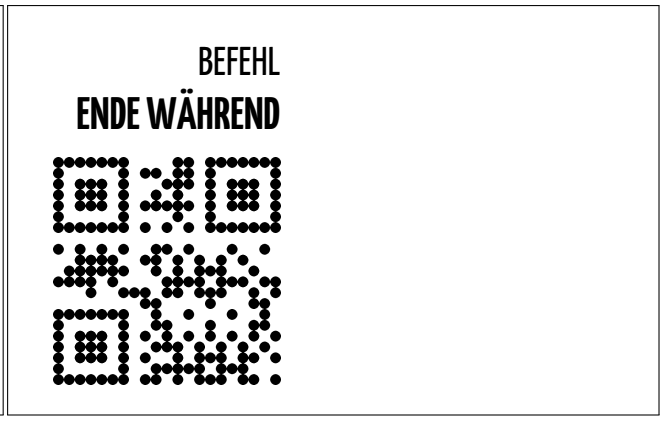


Abbildung 6: Befehl ENDE WÄHREND

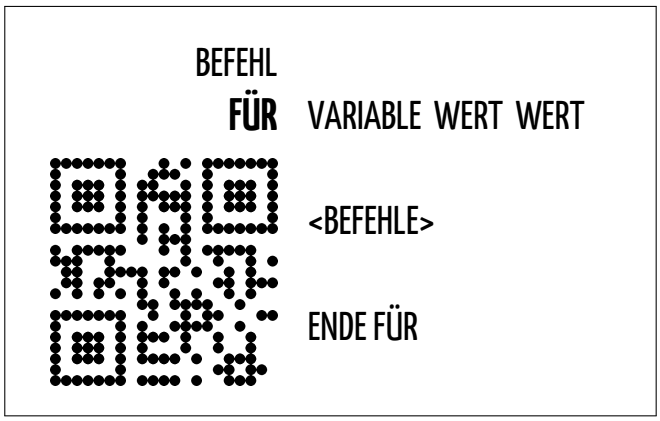


Abbildung 7: Befehl FÜR

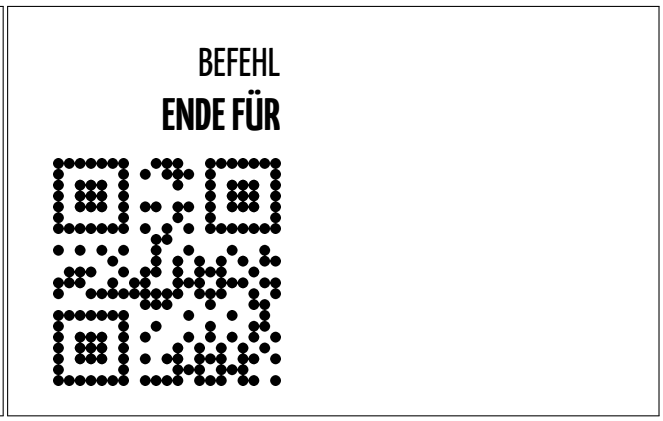


Abbildung 8: Befehl ENDE FÜR

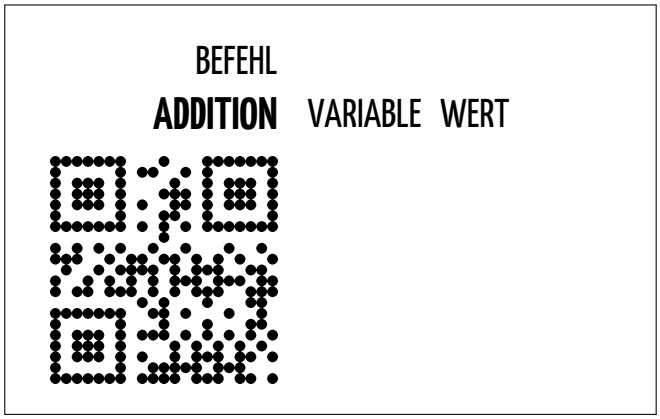


Abbildung 9: Befehl ADDITION

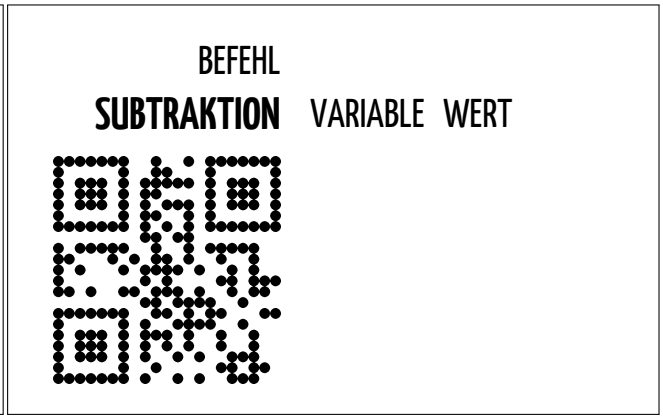


Abbildung 10: Befehl SUBTRAKTION

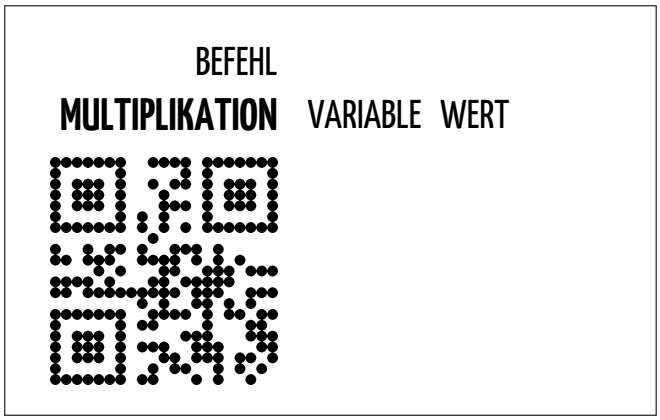


Abbildung 11: Befehl MULTIPLIKATION

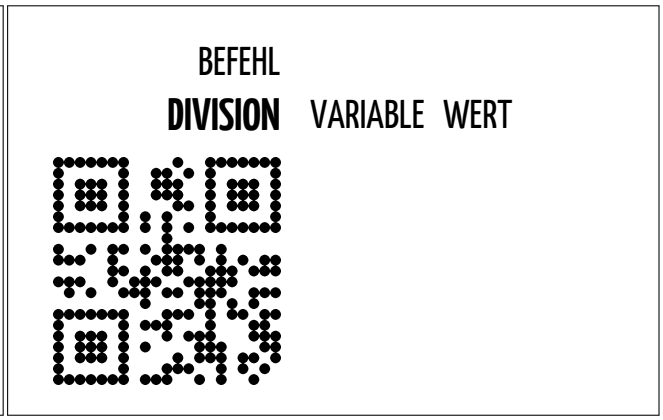


Abbildung 12: Befehl DIVISION

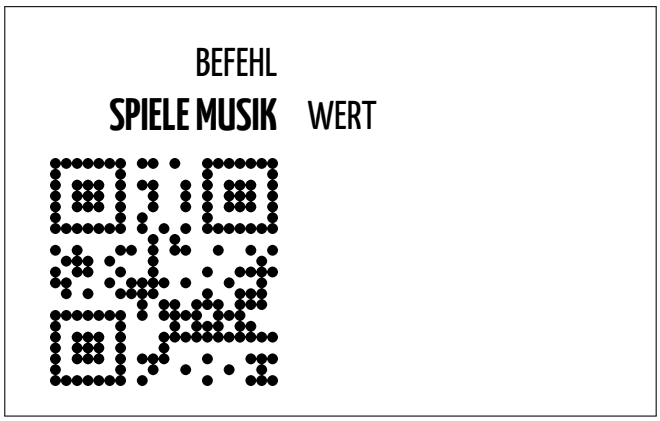


Abbildung 13: Befehl SPIELE MUSIK

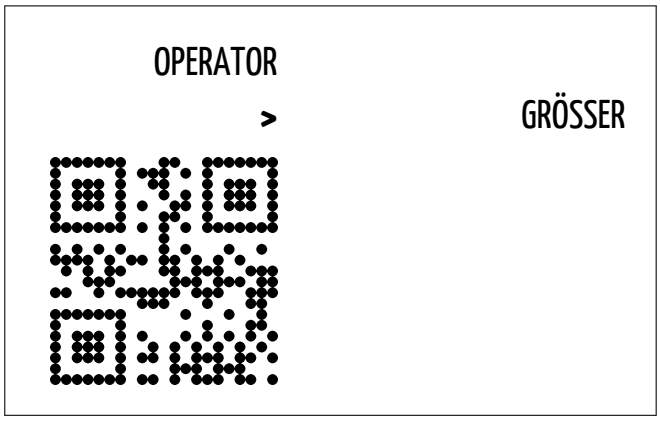


Abbildung 14: Operator GRÖSSER

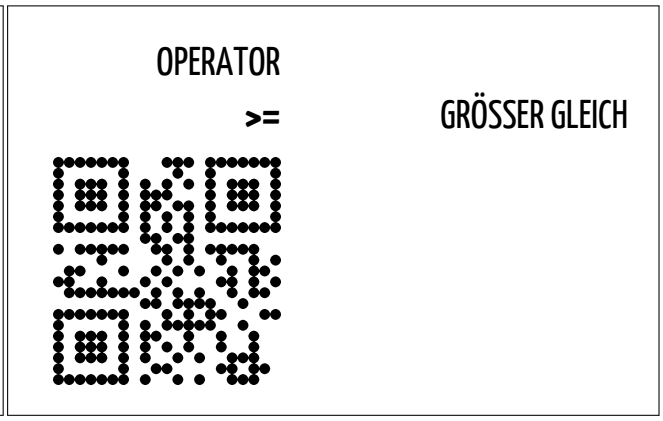


Abbildung 15: Operator GRÖSSER GLEICH

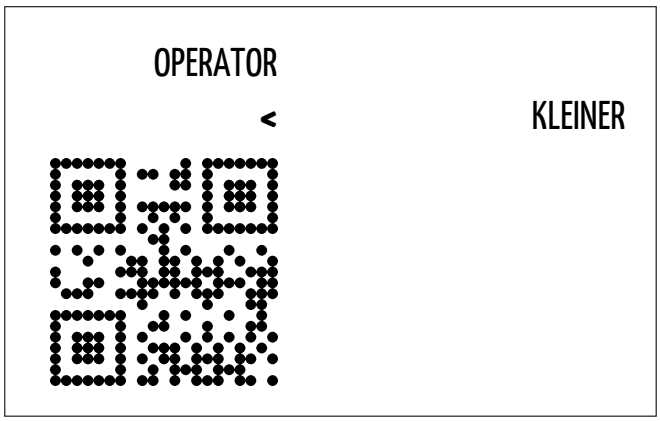


Abbildung 16: Operator KLEINER

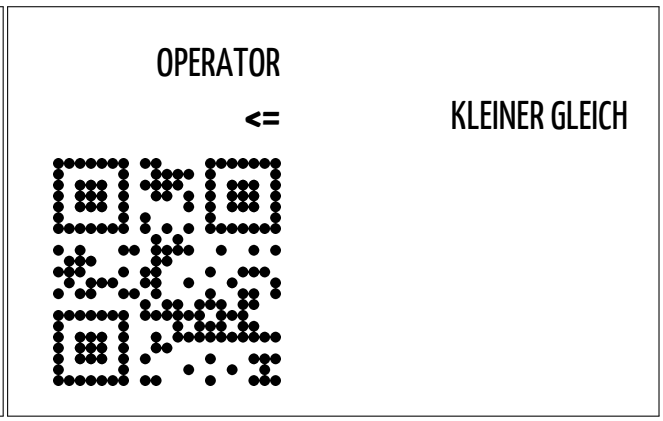


Abbildung 17: Operator KLEINER GLEICH

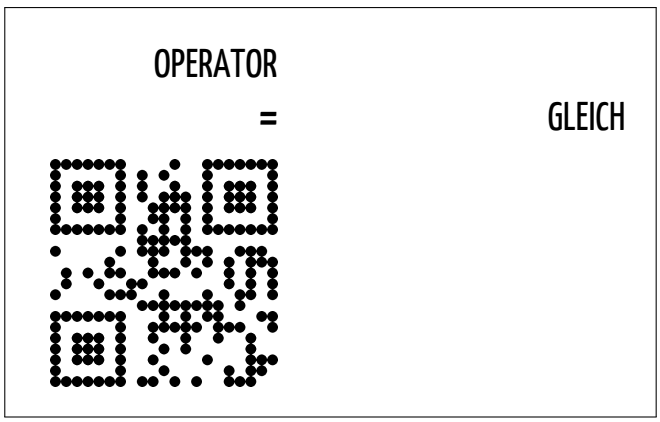


Abbildung 18: Operator GLEICH

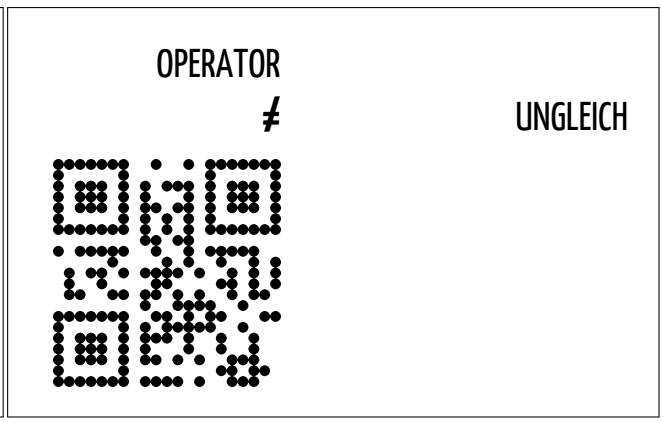


Abbildung 19: Operator UNGLEICH

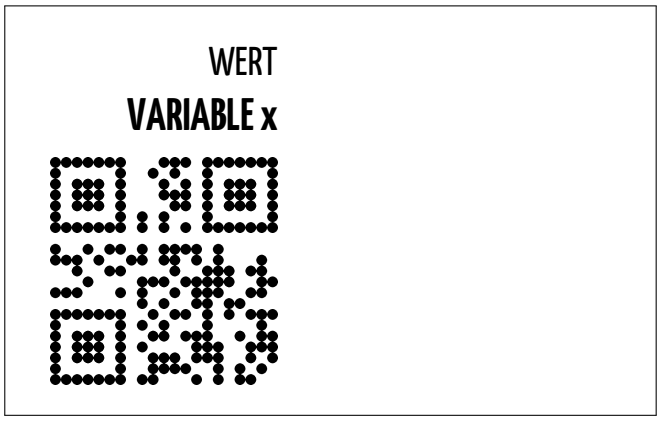


Abbildung 20: Variable x

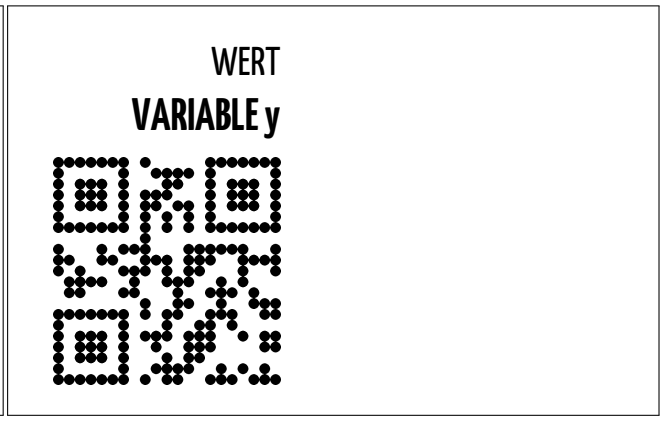


Abbildung 21: Variable y

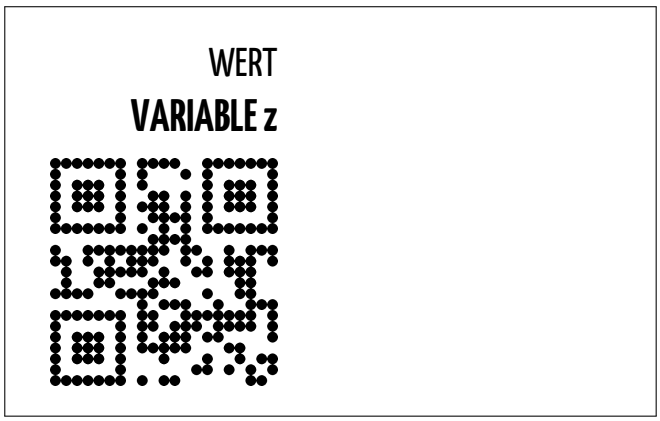


Abbildung 22: Variable z

Abbildung 23: Konstante ‘Hallo Welt!’

Abbildung 24: Konstante ‘Das Wetter ist’

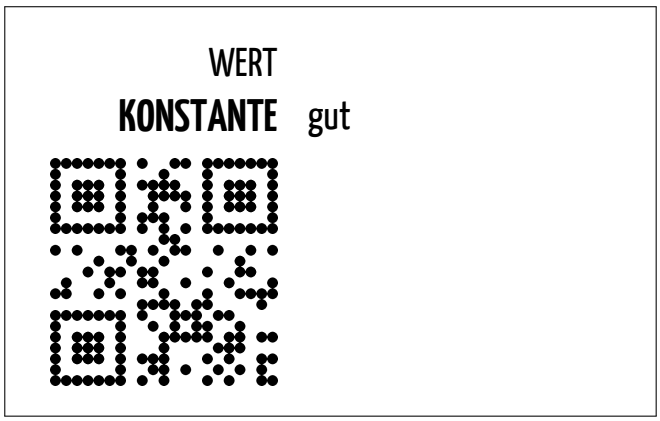


Abbildung 25: Konstante ‘gut’

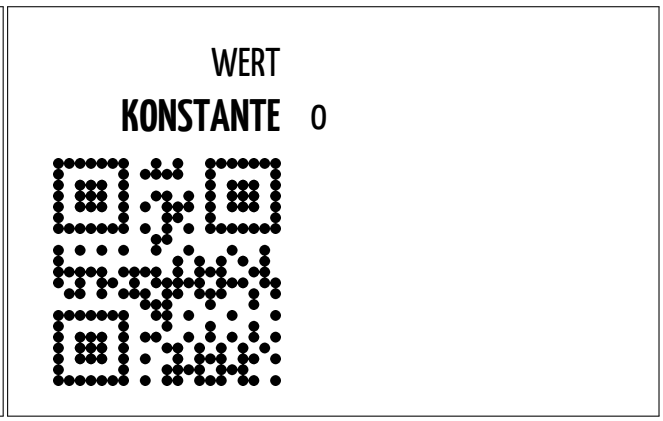


Abbildung 26: Konstante 0

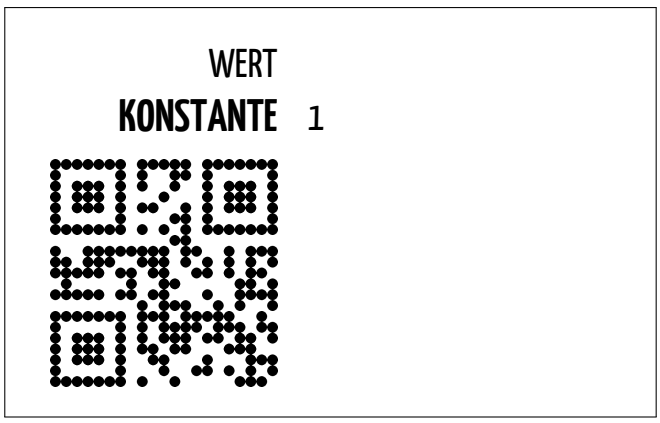


Abbildung 27: Konstante 1

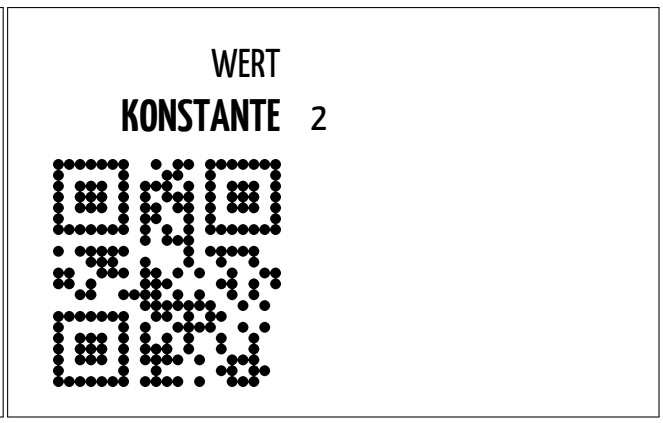


Abbildung 28: Konstante 2

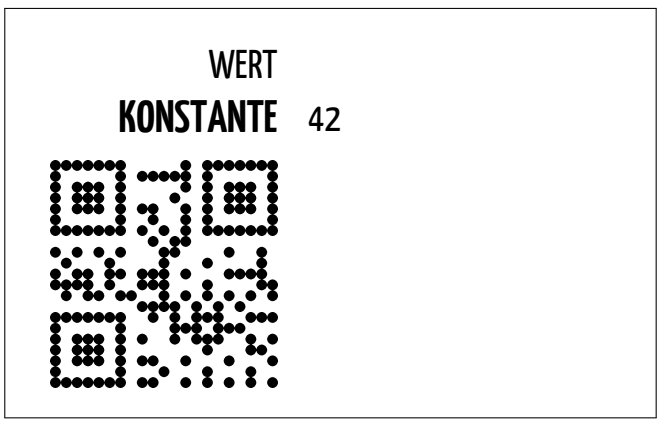


Abbildung 29: Konstante 42

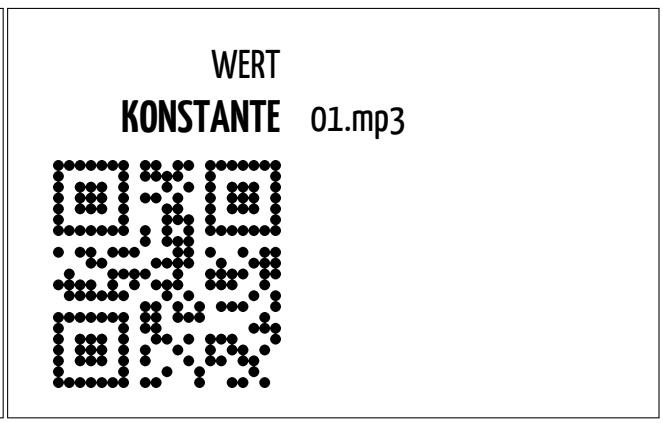


Abbildung 30: Konstante ‘01.mp3’

Quellcode

Aufgrund des Umfangs des Quellcodes der erstellten Programme sowie der von ihnen genutzen Funktionen und Bibliotheken wurde bewusst darauf verzichtet, den kompletten Quellcode hier abzudrucken. Ein Download des kompletten Pakets ist unter folgendem Link zu finden: http://bit.ly/1Nkj2GR

Nach dem Entpacken des Archivs stehen folgende Unterordner zur Verfügung:

- **androidcode**
  - Der vollständige Android-Quellcode.
- **apk**
  - Eine kompilierte Version der Kamera-App für Android.
    ACHTUNG: Die URL zu den zugehörigen PHP-Skripts ist in dieser Version statisch hinterlegt. Es kann nicht garantiert werden, dass die zum Zeitpunkt dieser Arbeit dafür aufgesetzte Internetpräsenz auf Dauer aktiv bleibt.
- **democards**
  - Einzelne Demokarten für QRBASIC im PDF-Format. Enthalten sind sämtliche Basisfunktionen sowie einige Beispielkonstanten und -variablen mit denen die in Kapitel 5 angeführten Unterrichts-Beispiele umgesetzt werden können.
- **phpcode**
  - Der komplette PHP-Quellcode. Aufruf der Datei *index.php* in einer Umgebung, die php-Skripts interpretieren kann, bietet Links zu Tools um zusätzliche Karten via Browser zu generieren und einen Basis-Kartensatz zu erzeugen.

Sollte dieser Link irgendwann in der Zukunft nicht mehr funktionieren, nehmen Sie bitte Kontakt mit dem Autor auf: thomas.cap@gmx.net.