

NKSA

PU-ARBEIT

DATEN UND INFORMATIONEN

---

# Magic2Brain

---

*Autoren:*

Roman HIMMEL G3E

Berke ATES G3E

Fabian HALLER G3E

*Betreungsperson:*

Claude GITTELSON

January 21, 2017

# Abstract

Daten und Informationen müssen nicht nur verarbeitet sondern auch gespeichert werden können. Dies ist sowohl beim Computer also auch bei uns Menschen so. Unser Vorhaben ist, einem Menschen Daten beizubringen. Nicht irgendwelche Daten sondern Karten des berühmten Trading Card Game "Magic: The Gathering". Dies soll in Form einer Lern-App passieren. Diese App soll im Android-Store zur Verfügung stehen und heruntergeladen werden können. In der App sind alle Karten von "Magic: The Gathering" beinhaltet und können auf das Gerät lokal heruntergeladen werden. Sobald die Karten heruntergeladen wurden, können sie mit selbst definierbaren Einstellungen gelernt werden. Die geschieht ungefähr so: Es wird zum Beispiel der Name der Karte genannt und man soll dann die dazugehörige Stärke nennen können. Wenn man richtig geantwortet hat, kommt die Karte auf einen "Gewusst" Stapel, wenn man sie nicht gekannt hat auf einen "Nicht gewusst" Stapel. Alle Karten vom "Nicht gewusst" Stapel werden wiederholt und müssen ein weiteres Mal gelernt werden. So kann man gezielt diese Karten abfragen, die der Benutzer noch nicht kann. Die App stellt aber auch eine Bibliothek dar. Mit Filteroptionen kann man gezielt nach einer Karte suchen, die einen gerade interessiert. Es sind natürlich auch noch weitere Optionen verfügbar, die der Benutzung der App helfen.

## Vorwort

Das Thema unserer Projektarbeit musste sich im Bereich von "Daten und Informationen" liegen. Dazu wollten wir etwas programmieren, da wir interessierte Programmierer sind und das Programmieren uns grosse Freude bereitet. Durch verschiedene Projekte hatten wir schon eine gewisse Menge an Erfahrung gesammelt. Wir kamen auf die Idee, eine App zu entwickeln, da wir noch nie eine App programmiert haben und wir somit neues lernen konnten. Da 2 von 3 ein Gerät mit dem Betriebssystem Android besitzt und wir schon eine Android-Entwickler-Lizenz hatten, entschieden wir uns die App für Android zu entwickeln. Die App sollte funktional sein und wir sollten Verwendung für die App haben. Wir sind zudem ambitionierte "Magic: The Gathering"-Spieler. Bei "Magic: The Gathering" (kurz: MTG) werden mehrmals im Jahr ein neues Set Karten veröffentlicht. Diese haben neue Namen, Bilder, Fähigkeiten und Attribute. Somit muss man diese erst kennenlernen, bevor man mit ihnen richtig spielen kann. Bisher war es relativ mühsam die Karten auswendig zu lernen, wir keine Lernwebsites oder Apps gefunden haben, die einem das Lernen erleichtern. Dies ist allerdings nötig, um ein schnelles und flüssiges Spielen zu sichern. Wir liessen uns von Quizlet inspirieren, welches Fremdwörter abfragt. Dadurch kamen wir auf die Idee eine App zu entwickeln, welches die aktuellen Karten dem Benutzer beibringt. Demnach spart man Zeit, da man die Karten nicht suchen muss. Folglich haben wir ein technisch anspruchvolles Projekt, welches wir auch privat nutzen können.

# Inhaltsverzeichnis

<b>1</b>	<b>Einleitung</b>	<b>4</b>
<b>2</b>	<b>Theoretische Grundlagen</b>	<b>7</b>
2.1	Einführung . . . . .	7
2.1.1	Die IDE . . . . .	7
2.1.2	Der Compiler . . . . .	8
2.1.3	Der Debugger . . . . .	8
2.1.4	Die Programmiersprache . . . . .	9
2.2	Vom Binärcode zum Bild . . . . .	9
2.2.1	Vom Binärcode zur Zahl . . . . .	10
2.2.2	Grundoperatoren bei binären Zahlen . . . . .	11
2.2.3	Von der Zahl zur Text . . . . .	12
2.2.4	Von der Zahl zur Farbe . . . . .	13
2.2.5	Von der Farbe zum Bild . . . . .	14
2.3	Grundlagen von Java . . . . .	15
2.4	Android Studio . . . . .	15
<b>3</b>	<b>Methode</b>	<b>20</b>
3.1	Die Arbeit mit Phonegap . . . . .	20
3.2	Der Aufbau der App . . . . .	20
3.3	Der Wechsel zu ”‘Android Studio’” . . . . .	20
<b>4</b>	<b>Darstellung und Ergebnisse</b>	<b>22</b>
<b>5</b>	<b>Diskussion der Ergebnisse</b>	<b>23</b>
<b>6</b>	<b>Zusammenfassung</b>	<b>24</b>

# 1 Einleitung

Das Thema "Daten und Informationen" wird heutzutage immer wichtiger. Aber was beinhaltet Daten und Informationen alles? In der Steinzeit gab es die erste Form von Datenspeicherung in der Form von Höhlenmalereien. Bereits diese Bilder konnten "Erfahrungen mit Jagdwild, Jagdtechniken oder Wanderrouen von Tieren festhalten" [1]. Wenn man dann weiter in die Zukunft schaut, erkennt man etwas revolutionäres in der Ägyptischen Kultur. Jedem Ägyptischen Zeichen wurde eine Bedeutung gegeben. Somit hatte man nun eine (teilweise) willkürliche Zuordnung von einem Zeichen für ein Objekt. Man musste also nun nicht mehr mit Bildern Informationen weitergeben, welche dann auch sehr verschieden interpretiert werden konnten, sondern man hatte eine Schrift entwickelt, die viel weniger Interpretationsspielraum bot und dadurch auch viel präziser war. Genau diese Schrift hat uns dann, wenn auch mit einigen Modifikationen und Abstraktionen bis in die Neuzeit als effizienteste Datenspeicherung gedient. Natürlich gab es auch in dieser Zeit noch revolutionäre Erfindungen wie die Druckerpresse von Johannes Gutenberg. Aber der erste Schritt zur heutigen Datenspeicherung wurde erst vor 75 Jahren gemacht, als im Mai der "Zuse Z3" (Abbildung 1) fertiggestellt wurde, den man heute als den "ersten funktionstüchtigen Computer der Geschichte" bezeichnet. [2] Jedoch konnte dieser erste Computer, wie der Name

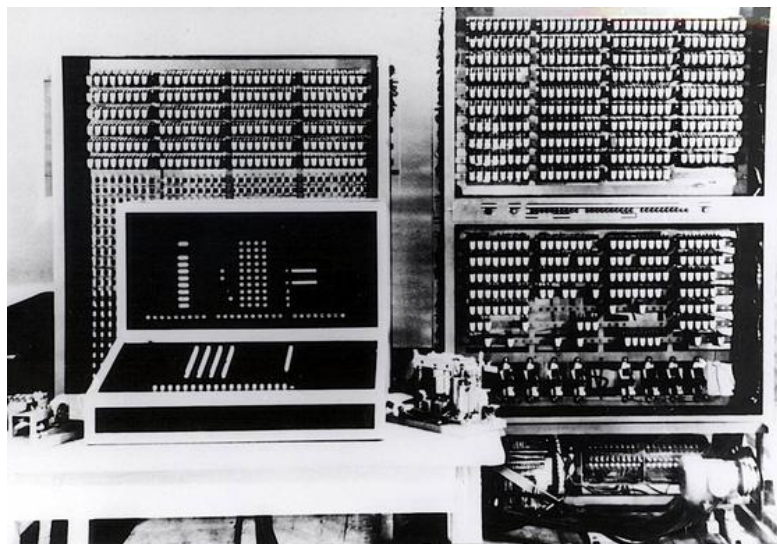


Figure 1: Zuse Z3 [3]

bereits verrät (von eng. to compute = rechnen), die Daten nicht hauptsächlich speichern sondern verarbeiten. Und genau diese Datenverarbeitung ist es, die uns in den letzten Jahren so stark geprägt hat. Viele einfache Dinge, für die früher ein Mensch praktizieren musste, wird heute von einem Computer übernommen, der das ganze dann meistens auch noch viel schneller kann. Und genau diese Datenverarbeitung in Verbindung mit der Benutzerinteraktion steht bei unser Projektarbeit im Mittelpunkt. Das Ziel unseres Projekt ist es, dem Benutzer etwas beizubringen und das auf einem möglichst ertragreichem Weg. Das beizubringende wird auf Karten des Spiels "Magic: The Gathering" (Abbildung 2) beschränkt. Neben der

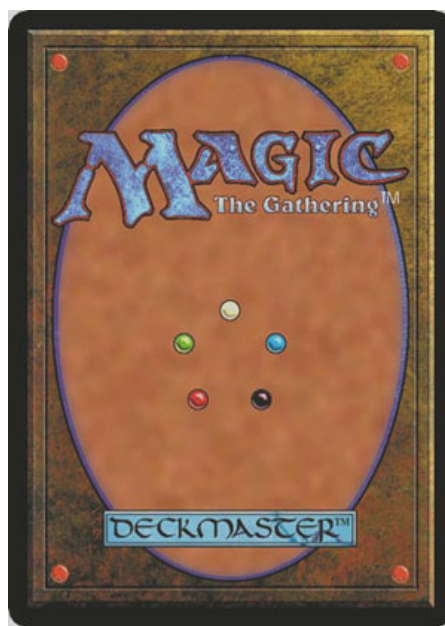


Figure 2: Magickarte [4]

Funktion mussten wir uns auch noch für eine Plattform entscheiden, auf welcher unsere App verfügbar sein sollte. Da man jeder Zeit die Karten lernen können sollte, haben wir uns für eine mobile Android-App entschieden. Der Benutzer kann über ein Menu entscheiden, welche Karten er lernen will. Dann kann er über eine Lernfunktion die Karten lernen. Wie oben schon angesprochen, steht die Datenverarbeitung im Mittelpunkt. Deshalb speichern wir ab, ob der Benutzer die Karte gekonnt hat oder nicht. Falls ja, sollte die Karte in nächster Zukunft nicht noch einmal erscheinen, falls nein, sollte sie in naher Zukunft wieder erscheinen. Insofern kann man den maximalen Lernerfolg erreichen, da gezielt die

Karten abgefragt werden, die noch nicht richtig sitzen. Ausserdem werden Daten auf zwei verschiedene Arten gespeichert: Einmal auf der App selbst und dann sollten diese auch noch ins Gedächtnis des Benutzers gelangen. Natürlich sollte die App nicht nur funktionieren sondern auch noch ansprechend Aussehen und Benutzerfreundlich sein. Aber genau das ist es, was unsere App ausmachen soll: "Magic2Brain"

## 2 Theoretische Grundlagen

In diesem Teil werden alle theoretischen Informationen gegeben, die man für das Verständniss des ganzen Projektes braucht. Nach dieser Einführung sollte man in der Lage sein, den Quellcode im Anhang zu verstehen. Da es jedoch sehr viel Übung erfordert, einen Quellcode zu lesen und zu verstehen, ist es naheliegend, dass der Quellcode nicht verstanden wird. In der Einführung wird grob angeschaut, wie man überhaupt vom Quellcode zur App kommt, im zweiten Teil wird erklärt, wie Daten jeglicher Art als Zahlen in einem Computer abgespeichert werden, im dritten Teil gibt es eine Einführung in die Programmiersprache Java und im letzten Teil werden dann noch ein paar spezifische Informationen in bezug auf Java gegeben, die für die Erstellung einer App gebraucht werden und die Software "Android Studio" wird erklärt.

### 2.1 Einführung

#### 2.1.1 Die IDE

Die IDE (eng. integrated development environment) oder auf Deutsch die Entwicklungsumgebung ist der Ort, an dem die meisten Programmierer arbeiten. Sie bietet alle wichtigen Werkzeuge, die man zum entwickeln von Software benötigt (Editor mit Color Highlighting, Compiler, Debugger, Dateibrowser etc.). Auf einige Begriffe wird später noch genauer eingegangen. In unserem Falle heisst die Entwicklungsumgebung übrigens Android Studio. In der Entwicklungsumgebung findet die ganze Entwicklung einer Software statt. Der wichtigste Bereich davon ist der Editor. Dort wird der ganze Quellcode hingeschrieben und dank Color Highlighting werden die wichtigen Komponenten (Kontrollstrukturen, Variablen, Kommentare etc. siehe Abschnitt 2.3 Grundlagen von Java) mit Farbe hervorgehoben (Abbildung 3). Das ist sehr wichtig, da man ansonsten schnell den Überblick verloren hat. Natürlich gibt es nicht nur eine IDE sondern ganz viele. Welche man davon benutzt ist jedem selbst überlassen. Es gibt auch Entwickler, die es bevorzugen, ohne eine IDE zu arbeiten. Zwar kann man dann alles selbst so gestalten wie es einem passt, es macht aber alles viel komplizierter ist besonders für neu beginnende Programmierer nicht empfehlenswert.



```

#include <iostream>
using namespace std;

int main(){
    int a;
    cin >> a;
    for(int i = 1; i <= a; i++){
        if(i%3==0 && i%5==0){
            cout << "FizzBuzz";
        }
        else if(i%3==0){
            cout << "Fizz";
        }
        else if(i%5==0){
            cout << "Buzz";
        }
        else{
            cout << i;
        }
        cout << endl;
    }
}

#include <iostream>
using namespace std;

int main(){
    int a;
    cin >> a;
    for(int i = 1; i <= a; i++){
        if(i%3==0 && i%5==0){
            cout << "FizzBuzz";
        }
        else if(i%3==0){
            cout << "Fizz";
        }
        else if(i%5==0){
            cout << "Buzz";
        }
        else{
            cout << i;
        }
        cout << endl;
    }
}

```

Figure 3: Beispiel mit und ohne Farbhervorhebung [5]

### 2.1.2 Der Compiler

Leider versteht der Computer nichts von dem, was wir in den Quellcode schreiben, alles was er versteht besteht aus Nullen und Einsen (mehr darüber im Abschnitt 2.2). Deshalb muss der für uns verständliche Quellcode in Maschinencode übersetzt werden. Dies geschieht mit dem so genannten ”Compiler” (eng. to compile = zusammentragen). Meistens ist dieser bereits in der IDE enthalten und auf Knopfdruck abrufbar.

### 2.1.3 Der Debugger

Der Debugger ist sehr eng mit dem Compiler verbunden. Meistens schafft man es nämlich nicht, auf Anhieb einen Fehlerlosen Code zu schreiben. Es passiert extrem schnell, dass irgendwo ein ”,” oder eine Kontrollstruktur falsch geschrieben wurde (mehr dazu im Abschnitt 2.3). Deshalb ist es sehr wichtig, dass man den Fehler findet. Wenn man jetzt aber Quellcode von 500 Zeilen geschrieben hat, wäre es doch sehr mühsam, wenn man nur wüsste, dass man einen Fehler hat. Genau dafür ist der Debugger. Ist aus dem englischen Wort ”Bug” entstanden, was so viel wie Käfer heisst, im Programmieren aber als Synonym zu Fehler verwendet wird. Dementprechend könnte man also Debugger als ”Entfehlerer” oder verdeutscht als ”Fehlersuchprogramm” bezeichnen. Meistens wird er aber ein-

fach Debugger genannt. Offenbar ist der Debugger dazu in der Lage, die Programmierfehler zu finden. Er findet aber leider nur Syntaxfehler und keine, die den gewünschten Programoutput betreffen. Man kann sich das so vorstellen: In Microsoft Word werden auch Rechtschreibfehler angezeigt, trotzdem kann man Sätze bilden die entweder keinen Sinn ergeben oder etwas anderes Aussagen, als gewünscht. Aber der Programmieralltag ohne Debugger wäre fast unvorstellbar, da man die meisten Fehler nicht so einfach sieht wie ein falsch geschriebenes Wort. Es gibt verschiedene Arten von Debugger. Die meisten zeigen einem die Fehler erst an, wenn man den Quellcode zu kompilieren versucht, es gibt aber auch solche, die das in Echtzeit machen, so wie Microsoft Word Rechtschreibfehler anzeigt.

#### **2.1.4 Die Programmiersprache**

Eine Programmiersprache kann man sich am einfachsten wie eine richtige Sprache vorstellen. Sie bildet den Grundbaustein des Programmierens. Bevor man dem Computer etwas beibringen kann, muss man eine solche lernen. Jede Programmiersprache hat seine eigene Syntax, trotzdem sind sie meistens ähnlich aufgebaut. Sobald man also eine Programmiersprache gelernt hat, fällt es einem einfach, eine nächste zu lernen. Man muss sich das so vorstellen: Nachdem man gelernt hat, wie Windows XP funktioniert, hat man nicht mehr so grosse Schwierigkeiten, zu lernen wie Windows 7 oder Windows 8 funktioniert. Meistens haben die Programmiersprachen verschiedene Anwendungsbereiche: Wird z. B. JavaScript und PHP meist nur in Webanwendungen verwendet, wird C++ wegen seiner Geschwindigkeit meist in Systemanwendungen gebraucht oder Java für Geräte wie Drucker oder eben Androidapplikationen, ausserdem ist das berühmte Computerspiel "Minecraft" in Java geschrieben.

## **2.2 Vom Binärcode zum Bild**

In diesem Teil geht es darum zu verstehen, wie ein Computer komplexe Informationen wie Bilder speichern kann. Dazu geht man zuerst von Binärcode (z.B. 01110110101001) aus und arbeitet sich dann hoch. Zwar ist dieses Thema nicht unbedingt notwendig, um unser Projekt nachvollziehen zu können, aber es hilft dabei, sich die Datenspeicherung besser vorstellen zu können, was auf jeden Fall sinnvoll ist, da ja die Datenspeicherung eines unserer Kernthemen ist. Ausserdem

kann man dann auch die Datentypen in Java besser verstehen (Abschnitt 2.3).

### 2.2.1 Vom Binärcode zur Zahl

Die kleinste Speichereinheit eines Computers ist das Bit. Man sollte das Bit aber auf keinen Fall mit dem Byte verwechseln. Das Byte beinhaltet nämlich acht Bits und ist dadurch viel grösser als das Bit. Man kann sich das Bit als Schalter vorstellen: Entweder ist er an oder er ist aus. Es gibt genau diese 2 Zustände. Mit der 1 bezeichnet man den angeschalteten Zustand und mit 0 den ausgeschalteten. Wenn man jetzt zwei Schalter nimmt dann gibt es bereits 4 verschiedene Schalterzustände: 00, 01, 10 und 11. Wenn wir nun  $N$  Schalter aneinander reihen haben wir dementsprechend dann auch  $2^N$  verschiedene Schalterzustände. Wenn wir jetzt jeden dieser Schalterzustände einer Zahl zuordnen, kann man je nach Schalteranzahl beliebig grosse Zahlen abspeichern. Jedoch wurden Zahlen nicht willkürlich irgendeiner Schalterkombination zugeordnet sondern es gibt ein gewisses System dahinter, damit man die Zahlen nachher auch miteinander verrechnet werden können. Dieses System nennt man das binäre Zahlensystem. Es ist nicht wie unser dezimales Zahlensystem auf zehn ausgerichtet sondern auf zwei. Um die Zahlen schreiben können, muss aber zuerst definiert werden, wie viele Bits gross eine Zahl ist. Diese Definition wird hier der Einfachheit halber auf 4 Bits gesetzt. Wie auch in unserem Zahlensystem wird die Zahl 0 als 0 dargestellt. Weil aber 4 Bits zur Verfügung stehen und nicht nur eines, müssen wir auch den anderen 3 einen Zustand geben, also auch 0. Dann sieht die Zahl 0 also binär dargestellt so aus: 0000. Auch die Zahl 1 ist noch einfach darzustellen: 0001. Wenn aber die Zahl 2 geschrieben werden soll, wird es bisschen komplizierter. Wir können nämlich den ersten Schalter nicht auf 2 Stellen. Die Antwort ist aber eigentlich ganz einfach: Wie in unserem dezimalen Zahlensystem nach der Zahl 9 eine neue Stelle gebraucht wird, so wird es binär genau gleich nach der Zahl 1 gemacht. Deshalb wird die Zahl 2 also so geschrieben: 0010. In diesem System geht es weiter:

Dezimal	0	1	2	3	4	5	6	7	8	...
Binär	0000	0001	0010	0011	0100	0101	0110	0111	1000	...

Dieses System gut zu kennen kann sehr nützlich sein. Wenn man die Zahlen nämlich noch ein bisschen genauer analysiert, kann man feststellen, dass man die

hinterste Stelle der binären Zahl immer angibt, ob der Zahl eine 1 addiert werden muss, die zweit hinterste dasselbe mit 2, der dritt hintersten mit 4. Bei jeder weiteren Stelle nach vorne nimmt die dazuzuaddierende Zahl um den Faktor 2 zu. Wenn man dieses System erkannt hat, kann man auch grosse binäre Zahlen in einer nützlichen Frist in eine dezimale übersetzten und umgekehrt. Hier ein Beispiel mit der achtstelligen Binärzahl 10100101:

$$\begin{array}{cccccccc} 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 128 & + & 0 & + & 32 & + & 0 & + & 0 & + & 4 & + & 0 & + & 1 & = & 165 \end{array}$$

Das hier vorgestellte binäre Zahlensystem ist jedoch stark vereinfacht, da sich in diesem nur Zahlen  $\in \mathbb{N}$  darstellen lassen. Negative Zahlen und Gleitkommazahlen würden aber zu stark vom eigentlichen Thema abweichen. Wichtig ist nur, dass der Computer nur so genannte Maschinenzahlen abspeichern kann. Das heisst die Zahlen müssen endlich gross sein und deshalb können periodische und irrationale Zahlen nicht genau abgespeichert werden.

## 2.2.2 Grundoperatoren bei binären Zahlen

Wenn man erst mal verstanden hat, wie das binäre Zahlensystem funktioniert, dann sind die Grundoperationen sehr einfach. Sie lassen ganz einfach mit den schriftlichen Operationen berechnen. Als Beispiel werden 5 Bit grosse Zahlen genommen: Die Zahl 6 (00110) und die Zahl 3(00011).

$$\begin{array}{r} \text{Addition} \quad \begin{array}{r} 0 \ 0 \ 1 \ 1 \ 0 \\ + \ 0 \ 0 \ 0 \ 1 \ 1 \\ \hline 0 \ 1 \ 0 \ 0 \ 1 \end{array} \end{array}$$

$$\begin{array}{r} \text{Subtraktion} \quad \begin{array}{r} 0 \ 0 \ 1 \ 1 \ 0 \\ - \ 0 \ 0 \ 0 \ 1 \ 1 \\ \hline 0 \ 0 \ 0 \ 1 \ 1 \end{array} \end{array}$$

$$\begin{array}{r} \text{Multiplikation} \quad \begin{array}{r} 0 \ 0 \ 1 \ 1 \ 0 \cdot 0 \ 0 \ 0 \ 1 \ 1 \\ \hline 0 \ 1 \ 1 \ 0 \ 0 \\ 0 \ 0 \ 1 \ 1 \ 0 \\ \hline 1 \ 0 \ 0 \ 1 \ 0 \end{array} \end{array}$$

$$\begin{array}{r} 0 \ 0 \ 1 \ 1 \ 0 : 0 \ 0 \ 0 \ 1 \ 1 = 0 \ 0 \ 0 \ 1 \ 0 \\ \text{Division} \quad \underline{1 \ 1} \\ \quad \quad \quad 0 \ 0 \ 0 \end{array}$$

## 2.2.3 Von der Zahl zur Text

Mit dem Wissen, wie Zahlen binär gespeichert werden, kann man dann sehr einfach verstehen, wie Texte abgespeichert werden. Im Grundsatz ist es nur eine willkürliche Zuordnung von Zeichen zu einer Zahl. Es gibt verschiedene Zuordnungen, die wichtigsten hierbei sind ASCII (American Standard Code for Information Interchange) und UTF-8(Universal Character Set Transformation Format). ASCII hierbei speichert jegliche Zeichen in 7 Bits ab und kann aber nur 128 verschiedene Zeichen speichern. Zum Beispiel würde man das Wort Haus so kodieren: 72 97 117 115. Wenn man diese Zahlen auch noch in Bits übersetzt, hat man die exakte Kodierung, wie sie auch auf dem Computer abgespeichert würde: 1001000110000111101011110011(siehe Abbildung 4) Dadruch werden viele

# The ASCII code

American Standard Code for Information Interchange

## www.theasciicode.com.ar

ASCII control characters				ASCII printable characters				Extended ASCII characters							
DEC	HEX	Simbolo	ASCII	DEC	HEX	Simbolo	DEC	HEX	Simbolo	DEC	HEX	Simbolo	DEC	HEX	Simbolo
00	00h	NULL	(carácter nulo)	32	20h	espacio	64	40h	@	96	60h	.	128	80h	Ç
01	01h	SOH	(inicio encabezado)	33	21h	!	65	41h	A	97	61h	a	129	81h	ü
02	02h	STX	(inicio texto)	34	22h	"	66	42h	B	98	62h	b	130	82h	é
03	03h	ETX	(fin de texto)	35	23h	#	67	43h	C	99	63h	c	131	83h	â
04	04h	EOT	(fin transmisión)	36	24h	\$	68	44h	D	100	64h	d	132	84h	ä
05	05h	ENQ	(enquiry)	37	25h	%	69	45h	E	101	65h	e	133	85h	å
06	06h	ACK	(acknowledgement)	38	26h	&	70	46h	F	102	66h	f	134	86h	â
07	07h	BEL	(timbre)	39	27h	'	71	47h	G	103	67h	g	135	87h	ç
08	08h	BS	(retroceso)	40	28h	(	72	48h	H	104	68h	h	136	88h	ê
09	09h	HT	(tab horizontal)	41	29h	)	73	49h	I	105	69h	i	137	89h	ë
10	0Ah	LF	(salto de línea)	42	2Ah	*	74	4Ah	J	106	6Ah	j	138	8Ah	è
11	0Bh	VT	(tab vertical)	43	2Bh	+	75	4Bh	K	107	6Bh	k	139	8Bh	í
12	0Ch	FF	(form feed)	44	2Ch	,	76	4Ch	L	108	6Ch	l	140	8Ch	ì
13	0Dh	CR	(retorno de carro)	45	2Dh	-	77	4Dh	M	109	6Dh	m	141	8Dh	ï
14	0Eh	SO	(shift Out)	46	2Eh	=	78	4Eh	N	110	6Eh	n	142	8Eh	À
15	0Fh	SI	(shift in)	47	2Fh	/	79	4Fh	O	111	6Fh	o	143	8Fh	Á
16	10h	DLE	(data link escape)	48	30h	0	80	50h	P	112	70h	p	144	90h	Ê
17	11h	DC1	(device control 1)	49	31h	1	81	51h	Q	113	71h	q	145	91h	æ
18	12h	DC2	(device control 2)	50	32h	2	82	52h	R	114	72h	r	146	92h	Æ
19	13h	DC3	(device control 3)	51	33h	3	83	53h	S	115	73h	s	147	93h	ô
20	14h	DC4	(device control 4)	52	34h	4	84	54h	T	116	74h	t	148	94h	ö
21	15h	NAK	(negative acknowle.)	53	35h	5	85	55h	U	117	75h	u	149	95h	ò
22	16h	SYN	(synchronous idle)	54	36h	6	86	56h	V	118	76h	v	150	96h	ù
23	17h	ETB	(end of trans. block)	55	37h	7	87	57h	W	119	77h	w	151	97h	û
24	18h	CAN	(cancel)	56	38h	8	88	58h	X	120	78h	x	152	98h	ÿ
25	19h	EM	(end of medium)	57	39h	9	89	59h	Y	121	79h	y	153	99h	ÿ
26	1Ah	SUB	(substitute)	58	3Ah	:	90	5Ah	Z	122	7Ah	z	154	9Ah	Ü
27	1Bh	ESC	(escape)	59	3Bh	;	91	5Bh	[	123	7Bh	{	155	9Bh	ø
28	1Ch	FS	(file separator)	60	3Ch	<	92	5Ch	\	124	7Ch		156	9Ch	£
29	1Dh	GS	(group separator)	61	3Dh	=	93	5Dh	]	125	7Dh	}	157	9Dh	Ø
30	1Eh	RS	(record separator)	62	3Eh	>	94	5Eh	^	126	7Eh	~	158	9Eh	×
31	1Fh	US	(unit separator)	63	3Fh	?	95	5Fh	_				159	9Fh	f
127	20h	DEL	(delete)												

[www.theasciicode.com.ar](http://www.theasciicode.com.ar)

Figure 4: ASCII Tabelle [7]

sprachspezifischen Zeichen wie die deutschen Umlaute nicht abspeicherbar. Deshalb wird ASCII heutzutage auch meistens nicht mehr gebraucht. UTF-8 ist

hierbei viel nützlicher, aber auch komplizierter. Es kann jegliche Spezialzeichen jeder Sprache abspeichern. UTF speichert die einfachen ASCII Zeichen in 8 Bits ab, das erst hinzugekommene Bit speichert ab, ob es sich bei dem Zeichen um ein ASCII Zeichen handelt oder nicht. Falls ja, können die restlichen sieben Bits wie ein ASCII Zeichen interpretiert werden. Ansonsten folgen nicht nur 7 Bits sondern wieder abhängig von der nächsten Zeichenfolge bis zu 31 weitere Bits (im ganzen dann 4 Bytes). Die deutschen Umlaute brauchen zum Beispiel 2 Bytes Speicherplatz. UTF-8 ist heutzutage der meist verbreitete Zeichensatz im Internet (im November 2016 benutzen rund 88% aller Webseiten UTF-8). [6] Ein weiterer heutzutage sehr verbreiteter Zeichensatz ist Unicode, auf diesen wird hier aber nicht weiter eingegangen.

#### 2.2.4 Von der Zahl zur Farbe

Um Bilder abspeichern zu können, muss man zuerst einmal wissen, wie Farben abgespeichert werden. Die meist verbreitete Methode für das Abspeichern von Farben ist RGB (eng. Red Green Blue). Der Name spricht für sich, nacheinander wird die Menge der Farbe Rot, Grün und Blau in einer Skala von 0 bis 255 angegeben. Aber auch die Farben werden in einem unüblichen Zahlensystem dargestellt: im Hexadezimalen. Dieses basiert nicht wie das binäre auf 2 und das dezimale auf 10, sondern auf 16. Da in unserem Zahlensystem keine Ziffer für 10-15 existieren, hat man die ersten Buchstaben des Alphabetes genommen. 10 wäre also "A" und 15 wäre "F" hexadezimal geschrieben. **Achtung: Wegen diesem System ist die hexadezimale 10 nicht dasselbe wie die dezimale 10 sondern 16.** 255 wäre dementsprechend "FF" hexadezimal geschrieben und somit die grösste hexadezimale Zahl, die mit zwei Ziffern geschrieben werden kann. Das ist natürlich kein Zufall sondern extra so gewählt. Um eine hexadezimale Zahl zu kennzeichnen, benutzt man in der Regel den Präfix 0x. Ein RGB wert besteht also aus drei zweistelligen hexadezimalen Zahlen. Meistens wird bei RGB-Farben ein # als Präfix verwendet anstatt des hexadezimal üblichen 0x. Auch noch wichtig anzumerken ist, dass das RGB-System ein additiver Farbraum ist. Das heisst also wenn man die volle Farbstärke aller Farben Rot, Grün und Blau verwendet, erhält man weiss. Hier nun eine Tabelle mit den wichtigsten Farben und dem dazugehörigen hexadezimalen Wert.

Rot	Grün	Blau	Weiss
#FF0000	#00FF00	#0000FF	#FFFFFF
Schwarz	Gelb	Magenta	Cyan
#000000	#FFFF00	#FF00FF	#00FFFF

Übrigens ist es auch kein Zufall, dass die Farben hexadezimal dargestellt werden und nicht dezimal. Ein System, welches auf zehn basieren würde, hätte immer einen gewissen Datenverlust, da man um 10 darstellen zu können 4 Bits braucht. Mit 4 Bits lassen sich aber Zahlen bis 16 darstellen. Um also mit demselben Speicherbedarf die maximale Speicherausnutzung zu erreichen, hat man sich für ein hexadezimalen System entschieden. Es gibt auch hier nicht nur ein System, ein weiteres sehr wichtiges Farbsystem ist das "CMYK", welches vor allem für Drucker verwendet wird. Dieses System ist im Gegensatz zum RGB-Farbsystem ein subtraktives. Der Name steht für Cyan Magenta Yellow Key. Auch wie im RGB-Farbsystem werden mit Cyan, Magenta und Gelb die Menge der Farben dargestellt. Das CMYK-Farbsystem besitzt aber noch einen vierten Wert: Den Schwarzanteil (hier mit K für Key angegeben um nicht mit B für Black um eine Verwechslungsgefahr mit Blue zu vermeiden).

### 2.2.5 Von der Farbe zum Bild

Nun sollten alle Grundlagen vorhanden sein, um zu verstehen, wie ein Bild gespeichert werden kann. Hier wird zuerst von einem total unkomprimierten Bild ausgegangen, das mit RGB Farben ausgestattet ist. Digitale Bilder können in zwei Gruppen unterteilt werden: Rastergrafiken und Vektorgrafiken. Zuerst wird ersteres angeschaut. Ein Computerbildschirm ist bekanntlicherweise aus einzelnen Pixeln aufgebaut. Jedes dieser Pixel funktioniert eigenständig und kann eine beliebige Farbe anzeigen. Heutzutage ist das wohl am weitesten verbreitete Bildschirmformat das Full-HD mit 1920x1080 Pixeln. In einem völlig unkomprimierten Bild wird also für jedes Bild seine eigene Farbe gespeichert und schon hat man sein Bild abgespeichert. Wie aber aus dem letzten Kapitel über Farben bekannt ist, werden die Farben üblicherweise mit RGB abgespeichert. Wenn man nun bedenkt, wie diese Zahlen binär aufgebaut werden, kommt man für jede hexadezimale Stelle auf 4 Bits, die gebraucht werden. Weil eine normale RGB-Farbe aus 6 hexadez-

imalen Zahlen bestehen, heisst das wiederum, dass eine RGB-Farbe mit 24 Bits beschrieben werden kann, was das gleiche wie 3 Bytes ist. Um also ein Bild abzuspeichern, werden  $1920 \cdot 1080 \cdot 3 = 6220800$  Bytes  $\approx 6.5$  Megabytes. Das ist bereits eine sehr grosse Datenmenge. Für qualitativ hochwertige Bilder ist die Auflösung aber oftmals grösser und die Farbe genauer, wodurch der Bildspeicherplatzbedarf nochmals enorm wachsen würde. Wenn man jetzt sogar ein Video mit 60 FPS (eng. Frames Per Second = Bilder pro Sekunde) mit dieser Datengrösse des errechneten Bildes mit 6.5MB speichern würde, würde man für jede Sekunde 390MB verbrauchen. Dadurch wäre also ein Film mit einer durchschnittlichen Länge von 1.5h 2.106TB gross, also grösser als die meisten Speicher eines Computers. Die Realität sieht aber ganz anders aus: Um Speicherplatz zu optimieren werden erstens bei Bildern Komprimierungen angewendet, welche über komplexe Algorithmen die Bilder so abspeichern, dass diese viel weniger Speicherplatz einnehmen. Ausserdem speichern Videos nicht jedes Bild neu sondern nur dessen Veränderung. So kommt man dann auf eine etwa 1000 mal kleinere Speichergrösse. Bei Bildkompression unterscheidet man übrigens auch zwischen verlustbehafteten Kompressionen und verlustfreien. Bekannte verlustfreie Kompressionen sind zum Beispiel TIFF und PNG und bekannte verlustbehaftete JPEG und GIF [8]. Jedes Dateiformat hat aber seine eigenen Vorteile und Nachteile und je nach Zweck sollte man sich überlegen, welches davon am meisten Sinn macht. Wie Anfangs erwähnt gibt es aber auch noch die Vektorgrafik. Die Vektorgrafik speichert keine Informationen über einzelne Pixel sondern es speichert zum Beispiel eine Linie oder ein Kreis mit einer Funktion. Der Vorteil einer Vektorgrafik ist, dass man diese frei skalieren kann, ohne dass man eine Qualitätseinbusse hat. Meistens sind Firmenlogos und Ähnliches mit einer Vektorgrafik geschrieben.

## 2.3 Grundlagen von Java

## 2.4 Android Studio

Android Studio ist eine für das Android Open Source Project (kurz: AOSP) von Google entwickelte Entwicklungsumgebung auf der Basis von IntelliJ, dabei sollte beachtet werden, dass Android Studio im Gegensatz zu Android OS proprietären Code beinhaltet und somit weder Open Source ist, noch ohne Erlaubnis durch Dritte weiterverbreitet werden darf. IntelliJ ist eine bereits bestehende proprietäre



Entwicklungsumgebung für Java (bzw. JVM) und Android, des weiteren kann eine Lizenz für Firmen und Webentwickler erworben werden. Im Grunde wurde Android Studio mit der Hinsicht auf Applikationsentwicklung auf dem Android OS erstellt. Es beinhaltet von Haus aus notwendige, aber auch arbeitserleichternde Komponenten, wie einen Gradle-Compiler (Notwendig) oder einen Android-Emulator (Erleichtert das Testen der App auf einem Computer).

Gradle ist ein Open Source Build-Tool, was im Grunde über den eigentlichen Compiler-Programmen eine oberste Schicht in der Compiler-Toolchain bildet. Daher verkörpert Gradle mehrere separate Aufgaben in sich und vereinfacht das Kompilieren von Android Apps enorm. Die Kernkomponenten von Gradle sind

- **Linker, bzw. Zusammentragen von zusätzlichen Paketen und androidspezifischen Dateien, die für das Kompilieren erforderlich sind.** Dabei werden Pakete resp. Bibliotheken von den Quellen runtergeladen, die man zu der Quellenliste hinzugefügt hat. Dies erleichtert das Hinzufügen von Bibliotheken wie z.B. einer Bibliothek zum Herunterladen und Anzeigen von Bildern, da man nur die Downloadadresse angeben muss und sich nicht mehr um das Einbinden der Bibliothek in sein Applikationsprojekt kümmern muss.
- **Debugger** Wie bereits beschrieben liest sich der Debugger noch vor dem Precompiler den Code durch und macht den Programmierer auf allfällige Fehler oder Verbesserungen aufmerksam.
- **Precompiler** Der Precompiler ist ein Kernbestandteil der Compiler-Toolchain. Eine Toolchain ist eine Ansammlung von Programmen, die einem helfen aus geschriebenem Code und Assets ein lauffähiges Programm zu erstellen. Er liest jede Datei durch und bereitet sie auf die nachfolgenden Verarbeitungsprozesse vor, indem er unter anderem Kommentare entfernt, nicht-genutzte Funktionen und Variablen entfernt, Makros ersetzt und durch die `include`-Kontrollsequenz eingebundene Dateien einfügt. Die heutigen Precompiler bleiben aber meist nicht nur bei ihren Kernaufgaben, sondern versuchen auch den Code so weit zu verbessern, dass er nach dem Kompilierungsvorgang eine geringere Laufzeit und Speicherauslastung aufweist. Dabei

ist aber zu beachten dass Compiler und Precompiler sehr eng miteinander zusammenarbeiten.

- **Compiler** Der Compiler ist wohl das bekannteste Programm aus einer klassischen Toolchain. Er sorgt dafür, dass die von dem Precompiler aufbereiteten Dateien in Maschinensprache übersetzt werden. Wie bereits in der Einleitung erwähnt rechnet der Computer Binär. Daher ist eine Datei nichts anderes als eine lange Zahl. Je nach Prozessortyp wird auch eine Zahl anders verarbeitet. Dieser Unterschied liegt in der Verarbeitung von Kontrollsequenzen. In diesem trivialen Beispiel soll verdeutlicht werden, wie zwei verschiedene Prozessoren ein und die selbe Zahl anders interpretieren: Der Befehl für eine Additionsoption wird hier durch 11100010 repräsentiert. Im Prozessortyp A löst dies wie gewollt eine Addition aus, hingegen im Prozessortyp B werden die nachfolgenden Bits in den Zwischenspeicher kopiert und das Programm stürzt ab. Prozessoren haben also auch ihre eigene Sprache, die sich durch ihre Herstellung ergibt. Diese Sprache wird *Befehlssatz* genannt, wohingegen der Prozessortyp *Architektur* genannt wird. Jede Prozessorarchitektur hat ihren eigenen Compiler. Im Fall von Android haben sich drei Architekturen etabliert: armeabi-v7, Intel x86 und AMD64. Die App wird also in mehrere Sprachen übersetzt bevor sie ausgeliefert wird.

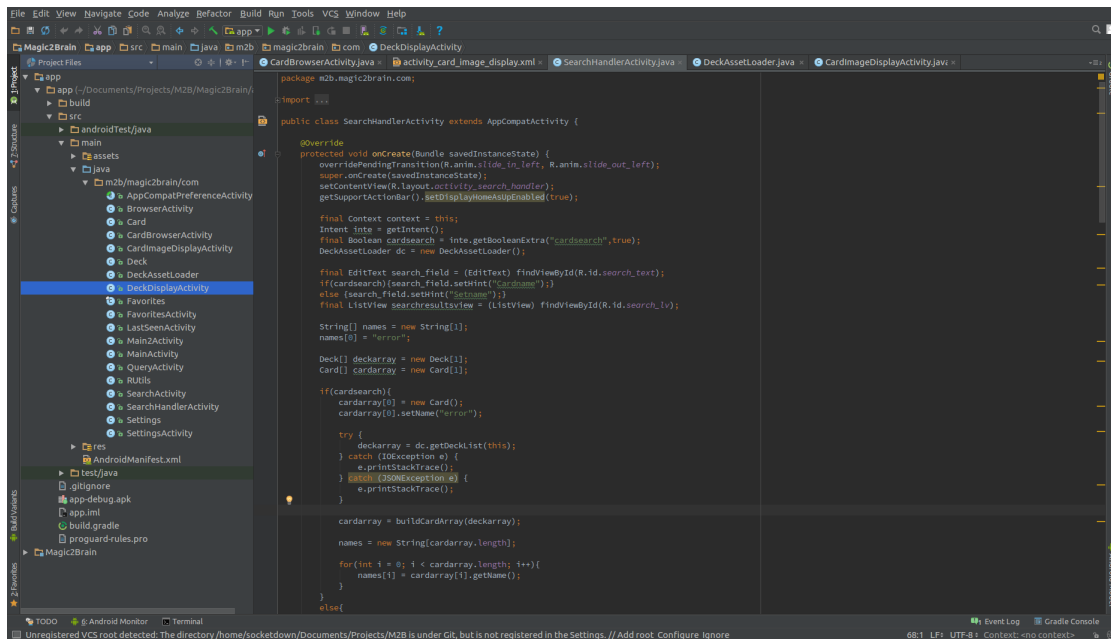


Figure 5: Android Studio Benutzeroberfläche [?]

Die Grafische Benutzeroberfläche ist das Wichtigste an einer IDE. Sie erleichtert dem Entwickler die Navigation durch seinen Code in Form von Farbmarkierungen (Rechts) und Dateibrowser (Links). Die Toolbar von Android Studio (Oben) ist mit wichtigen Shortcuts, wie z.B. Kompillierung starten, Emulator starten und Android SDK aktualisieren, gefüllt. Der Editor von Android Studio erlaubt eine automatische Vervollständigung mit Alt+Enter. Mitunter sind auch schon fertige Vorlagen für Activities in Android Studio verfügbar, die auf Knopfdruck in das Projekt kopiert werden und gleichzeitig für die Kompillierung registriert werden. Der Dateibrowser erlaubt es zwischen Quellcode-Dateien zu wechseln und diese zur Bearbeitung zu öffnen. Zusätzlich sind Funktionen zum importieren von Dateien oder Assets vorhanden. Selbstverständlich kann dies auch manuell geschehen, indem man in einem beliebigen Dateibrowser zu dem Assets resp. Res (kurz für *Ressources* - dt. *Ressourcen*) - Ordner navigiert, in dem man seine Dateien ablegen kann. Mit Assets sind hier nicht Immobilien oder Vermögen gemeint, sondern Dateien, die von der App benötigt werden um richtig zu funktionieren. Dazu gehören Bilder, Musikdateien und generell alles was kein Quellcode ist. Der IDE kann man auch Plugins hinzufügen. Plugins sind kleine Programmerweiterungen, welche dynamisch dazugeladen oder wieder entfernt wer-

den können. Bei der Entwicklung von Magic2Brain ist das Plugin "Asset Studio" besonders nützlich gewesen, da man mit ihm Icons nicht nur in den Ressourcenordner kopieren konnte, sondern diese gleich in allen auf Android gängigen Auflösungen abspeichern konnte. Normalerweise müsste das von Hand gemacht werden, was bei unseren ca. 20 Icons sehr zeitaufwändig gewesen wäre.

## 3 Methode

Es gibt mehrere Möglichkeiten eine Android App zu programmieren. Man kann ein Framework (z.B. "LibGDX") nutzen um mit einer vertrauten Programmiersprache und IDE eine App zu entwickeln. Google selbst bietet eine eigene IDE namens "Android Studio". Wir entschieden uns anfangs mit Phonegap zu entwickeln.

### 3.1 Die Arbeit mit Phonegap

Phonegap ist mit dem Framework Cordova aufgebaut. Der Vorteil von Phonegap ist, dass es für das Design HTML nutzt. Wir sind vertraut mit HTML und können damit schnell relativ ansehnliche Benutzeroberflächen (kurz: GUI) erstellen. Als Programmiersprache nutzt Phonegap Javascript. Auch damit sind wir vertraut, da wir ein Semester lang damit gearbeitet haben. Wir konnten also direkt loslegen. Phonegap bietet zudem die Möglichkeit die App direkt auf dem Smartphone zu betrachten. Diese Funktion ist sehr praktisch, wenn man kurz was betrachten will.

### 3.2 Der Aufbau der App

Wir wollten die Arbeit am Aufbau der App gleichmässig aufteilen. Wenn mehrere Personen an einem Projekt arbeiten, muss man dafür sorgen, dass keine Konflikte entstehen. Konflikte können entstehen, wenn z.B. mehrere Personen die gleiche Datei bearbeiten. Um dies zu verhindern wollten wir jede Seite der App in eine eigene Datei auslagern. Es sollte ein Menu geben, von wo aus man auf jede Seite zugreifen konnte. Dieses Menu musste am Anfang von einer Person erstellt werden. Danach konnten wir die einzelnen Seiten auf die Personen aufteilen und gleichzeitig Arbeiten, ohne dass Konflikte entstehen. Jede Seite sollte eine eigene Funktion haben. Z.B. gab es eine Seite für die Abfragen, eine Seite mit Favoriten etc. Mit einem Zurück-Knopf kommt man von jeder Seite zum Menu zurück.

### 3.3 Der Wechsel zu "Android Studio"

Wir haben viel Zeit damit verbracht mit Phonegap eine schöne Benutzeroberfläche zu erstellen. Als wir dann dazu kamen die Funktionalität zu programmieren, merk-

ten wir was der Nachteil von Phonegap ist. Phonegap nutzt Javascript als Programmiersprache. Javascript ist asynchron. Das heisst, dass es nicht linear (Ein Befehl nach dem Anderen) ausgeführt wird, sondern die Befehle einer Funktion praktisch gleichzeitig ausführt. Dies ist super, wenn man Animationen erstellen will, aber wenn ein Befehl auf das Resultat von einem anderen Befehl aufbaut, gibt es viele Probleme. Man kann das Problem mit Workarounds lösen, aber dies ist einerseits möglicherweise stabil auf allen Geräten und andererseits ist es extrem aufwändig. Also fassten wir den Entschluss nochmals neu anzufangen, aber diesmal mit "Android Studio". "Android Studio" nutzt Java als Programmiersprache. Auch mit dieser Programmiersprache kennen wir uns bestens aus. Dies vereinfachte die Entwicklung der Funktionalität um einiges. "Android Studio" nutzt für die GUI XML. XML ist von der Syntax her praktisch gleich wie HTML. Also mussten wir nicht viel umlernen. Bei XML gab es nur andere Elemente, welche wir lernen mussten. Etwas gewöhnungsbedürftig war die Verknüpfung zwischen XML und Java. Wir mussten lernen, wie man mit Java auf Elemente vom XML zugreift und diese verändert. Hinzu kam noch, dass wir uns mit dem Lebenszyklus einer App auseinandersetzen mussten. Bei Phonegap wurde das automatisch erledigt. Nach dem Lernen dieser Kleinigkeiten ging die Entwicklung viel schneller voran als bei Phonegap. Somit hat sich der Umstieg definitiv gelohnt.

## 4 Darstellung und Ergebnisse

### DARSTELLUNG UND ERGEBNISSE

## **5 Diskussion der Ergebnisse**

DISKUSSION DER ERGEBNISSE



## **6 Zusammenfassung**

ZUSAMMENFASSUNG

## Quellenverzeichnis

- [1] Wikipedia, the free encyclopedia. <https://de.wikipedia.org/wiki/Höhlenmalerei>  
[Online, zugegriffen 4. Januar 2017]
- [2] Wikipedia, the free encyclopedia. <https://de.wikipedia.org/wiki/Computer>  
[Online, zugegriffen 4. Januar 2017]
- [3] <http://www.ingenieur.de/> Bild der Zuse Z3 [Online, zugegriffen 4. Januar 2017]
- [4] Magic the Gathering - Wizards of the Coast <http://magic.wizards.com/de>  
[Online, zugegriffen 4. Januar 2017]
- [5] Screenshots aus Codeblocks <http://www.codeblocks.org/> und aus texmaker <http://www.xmlmath.net/texmaker/> (online 5.1.2017)
- [6] Wikipedia, the free encyclopedia. <https://de.wikipedia.org/wiki/UTF-8>  
(online 6.1.2017)
- [7] Wikipedia, the free encyclopedia. <http://www.theasciicode.com.ar/> (online 6.1.2017)
- [8] [http://www.hbksaar.de/uploads/media/Dateiformate\\_Bilddateien.pdf](http://www.hbksaar.de/uploads/media/Dateiformate_Bilddateien.pdf)  
(online 7.1.2017)