NKSA

PU-ARBEIT

DATEN UND INFORMATIONEN

Magic2Brain

Autoren:
Roman Himmel G3E
Berke Ates G3E
Fabian Haller G3E

Betreungsperson:
Claude Gittelson

January 24, 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 Traiding 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 ungefair 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 gekonnt 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 Biblothek 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 im sich im Bereich der "Daten und Informationen" befinden. Wir entschieden uns etwas zu programmieren, weil das eine unserer Leidenschaften ist. Durch verschiedene Projekte konnten wir schon eine gewisse Menge an Erfahrung sammeln. Wir hatten die Idee eine App zu entwickeln um uns auf Neuland wagen zu können und damit wir unseren eigenen Horizont erweiteren konnten. Da 2 von 3 ein Gerät mit dem Betriebssystem Android besitzt und wir schon eine Android-Entwickler-Lizenz besassen, entschieden wir uns die App auf der basis von Android zu schreiben. Die App sollte sowohl funktional sein als auch von uns selbst verwendet werden. Wir sind zudem ambitionierte "Magic: The Gathering"-Spieler. Bei "Magic: The Gathering" (kurz: MTG) werden mehrmals im Jahr neue Kartensets veröffentlicht. Die darin enthaltenen Karten haben neue Namen, Bilder, Fähigkeiten und Attribute. Man muss diese aber erst kennenlernen, bevor man mit ihnen richtig spielen kann. Bisher war es relativ mühsam die Karten auswendig zu lernen, wir keine Software zum Lernen der Karten gefunden haben. Dies ist allerdings nötig, um ein schnelles und flüssiges Spielen zu ermöglichen. Wir liessen uns von Quizlet und Memorize inspirieren, welche Fremdwörter abfragen. Dadurch kamen wir auf die Idee eine App zu entwickeln, welche die aktuellen Karten dem Benutzer beibringt. Demnach kann man viel Zeit einsparen in der man die Karten nicht suchen muss. Folglich haben wir ein technisch anspruchvolles Projekt, welches wir auch privat nutzen können.

Inhaltsverzeichnis

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 Wanderrouten 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 Iterpretationsspielraum 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

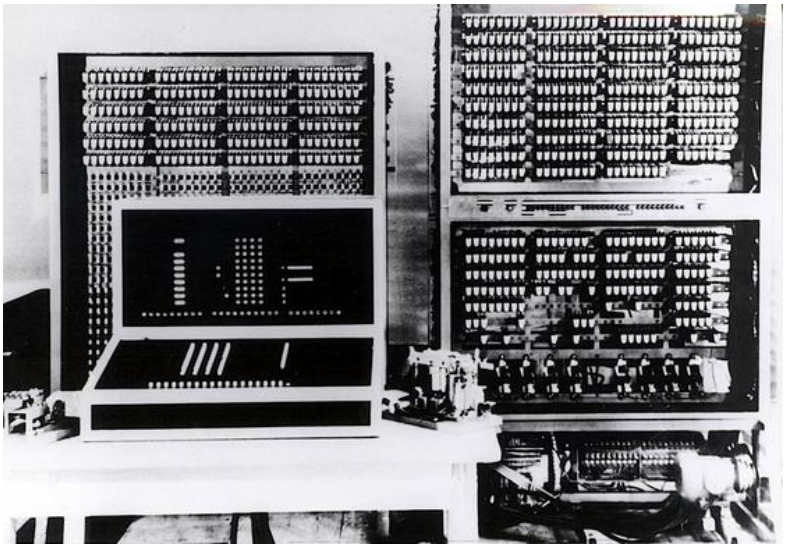


Figure 1: Zuse Z3 [3]

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 beizubringede 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 Platform 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 angeprochen, 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 Gedächniss 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"'

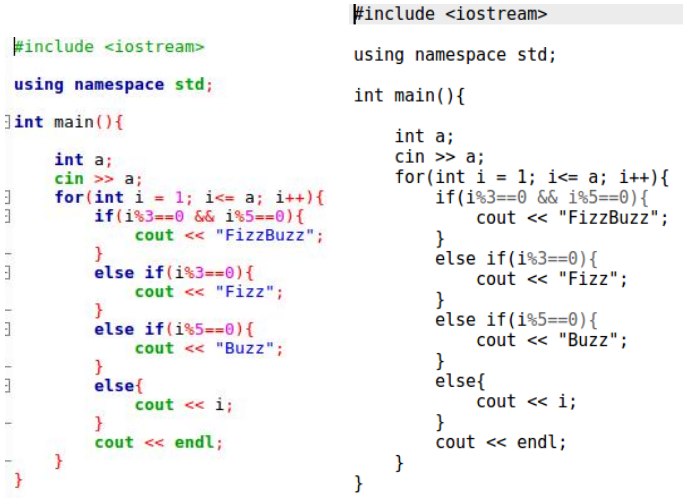
2 Theoretische Grundlagen

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 erklährt, 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 erklährt.

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 hingeschriben 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 gestallten 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 gennanten "'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 nähmlich 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 gennant. Offenbar ist der Debugger dazu in der Lage, die Programmierfehler zu finden. Er findet aber leider nur Syntaxfehler und keine, die den gewünschten Programmoutput betreffen. Man kann sich das so vorstellen: In Microsoft Word werden auch Rechtschreibefehler angezeigt, trotzdem kann man Sätze bilden die entweder keinen Sinn ergeben oder etwas anderes Aussagen, als gewünscht. Aber der Programmieraltag 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 komplieren versucht, es gibt aber auch solche, die das in Echtzeit machen, so wie Microsoft Word Rechtschreibefehler 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 beinhalted 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 anneinander 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 willkürich irgendeiner Schalterkombination zugeordnet sondern es gibt ein gewisses System dahinter, damit man die Zahlen nachher auch miteinander verrechnet werden können. Dieses System nenn man das binäre Zahlensystem. Es ist nicht wie unser dezimales Zahlensystem auf zehn ausgerichted sondern auf zwei. Um die Zahlen schreiben können, muss aber zuerst deffiniert 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 sieh die Zahl 0 also binär dargestellt so aus: 0000. Auch die Zahl 1 ist noch eifach darzustellen: 0001. Wenn aber die Zahl 2 geschrieben werden soll, wird es bisschen komplizierter. Wir können nämlich den ersten Schalter nicht auf 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 bischen genauer analyziert, 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:

1		0		1		0		0		1		0		1		
128	+	0	+	32	+	0	+	0	+	4	+	0	+	1	=	165

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 gennante Maschienenzahlen abspeichern kann. Das heist 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).

Addition

	0	0	1	1	0
+	0	0	0	1	1
	0	1	0	0	1

Subtraktion

	0	0	1	1	0
-	0	0	0	1	1
	0	0	0	1	1

Multiplikation

0	0	1	1	0	·	0	0	0	1	1
						0	1	1	0	0
						0	0	1	1	0
						1	0	0	1	0

	0	0	1	1	0	:	0	0	0	1	1	=	0 0 0 1 0
Division			1	1									
			0	0	0								

2.2.3 Von der Zahl zur Text

Mit dem Wissen, wie Zahlen binär gespeichert werden, kann man dann sehr eifach 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 Standart 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

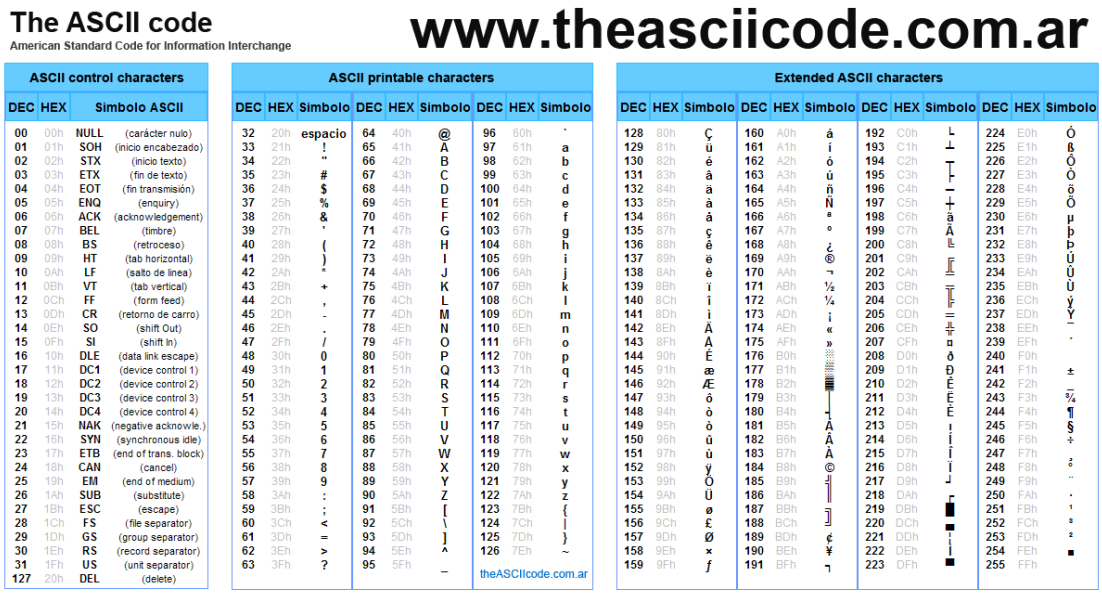


Figure 4: ASCII Tabelle [7]

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 heutezutage 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 dier 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 wir 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 zwei stelligen 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 ein 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 hexadezimales System entschieden. Es gibt auch hier nicht nur ein System, ein weiteres sehr wichtiges Farbsystem ist das "'CMYK"', welches vor allem für Drücker 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 ausgegange, das mit RGB Farben ausgestattet ist. Digitale Bilder können in zwei Gruppen unterteilt werden: Rastergrafiken und Vektorgrafiken. Zuerst wir ersteres angeschaut. Ein Computerbildschirm ist bekanntlicherweise aus einzelnen Pixeln aufgebaut. Jedes dieser Pixel funktioniert eigenstädnig und kann eine belibige 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. Wir aber aus dem letzt 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 geschriben werden kann, was das gleiche wie 3 Bytes ist. Um also ein Bild abzuspeichern, werden $1920 \cot 1080 \cdot 3 = 6220800$ Bytes ≈ 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 errechnetetn Bildes mit 6.5MB speichern würde, würde man für jede Sekunde 390MB verbrauchen. Dadurch wäre also ein Film mit einer durschnittlichen 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 solle 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 gezeichnet.

2.3 Grundlagen von Java

Jetzt da alles nötige erarbeitet wurde, um sich auch etwa vorstellen zu können, wie ein Computer Informationen speicheren kann, wird es einfach werden, die Datentypen von einer Programmiersprache wie Java zu verstehen. Im Abschnitt "'Grundlagen von Java"' wird die Programmiersprache schnell überflogen, so dass jemand, der bereits eine andere Programmiersprache kennt, Java schnell begreifen sollte und den Code im Anhang verstehen können sollte. Da jedoch keine prak-

tischen Aufgaben gestellt werden und auch nicht alles im Detail angeschaut wird, sind diese Grundlagen nicht dazu geignet, nach diesen selber entwickeln zu können. Für jene, die noch über keine Grundlagen über eine Programmiersprache verfügen, könnte dieser Einstieg zu schwierig sein und auch für jene, die über eine umfassende und vertiefte Erklährung wünschen, empfehlen wir eine andere Wissensquelle. Die Fachhochschule Südwestfahlen hat zum Beiepiel ein schönes Skript zur Einführung in Java geschriben **https://www4.fh-swf.de/media/java.pdf**. Wer lieber ein Buch zur Hand nimmt, dem können wir O'Reilly's "'Java von Kopf bis Fuss"' oder "'Java ist auch eine Insel"' empfehlen, die ein solides Grundwissen vermitteln.

2.3.1 Die Main-Methode

In Java liegt der Einstiegspunkt in das Programm immer in der Main-Methode. Diese Klasse wiederum liegt in einem Package und das Package im Projektordner. Fürs erste steht aber nur die Klasse im Fokus. Die Packages dienen hierbei einzig und allein der Struktur, wie Ordner auf dem Betriebssystem. Sobald der Compiler die Main-Methode endeckt (unten aufgeführt) erkennt er diese als Einstiegspunkt an. Im Normalfall wird auch die Klasse "'main"' gennant, um den Ort der Main Methode sofort ersichtlich zu machen. In eben dieser Main Methode können jetzt beliebige Befehle geschrieben werden. Im Code unten wird zum Beispiel eine Konsolenausgabe gemacht, der Wohl wichtigste Befehl. Innerhalb der Anführungs- und Schlusszeichen kann ein beliebiger Text geschrieben werden, der dann vom Programm in der Konsole ausgegeben wird.

```
1 public class main { //erstellt eine neue Klasse mit dem Namen main
2   public static void main(String[] args) { //Beginn der Main Methode
3     System.out.println("Hello World!"); //erzeugt eine
      Konsolenausgabe mit dem Text Hello World
4   } //Ende der Main Methode
5 }
```

Konsole:

```
Hello World!
```

2.3.2 Datentypen

Einfache Datentypen Im Kapitel 2.2 "'Vom Binärcode zum Bild"' wurde bereits ausführlich beschrieben, wie ein Computer die Daten abspeichert. Um einer Programmiersprache zu sagen, dass sie etwas abspeichern soll, muss man eine Variable definiert werden. Es gibt hierbei verschiedene Variabeln, die unterschiedlich gross sind und auch unterschiedliche Variablen abspeichern können.

```
1 public class main {
2   public static void main(String[] args) {
3     boolean b; //Speichert einen Wahrheitswert ab, also true oder
      false. Anders ausgedrueckt 0 oder 1
4     int i; //Speichert eine ganze Zahl ab
5     float f; //Speichert eine gleitkomma Zahl ab
6     char c; //Speichert einen einzelnen Buchstaben ab
7     String s; //Speichert einen Text ab
8   }
9 }
```

Der obige Codeabschnitt reserviert aber nur den Speicher, um etwas darin abspeichern zu können, muss zuerst noch ein Wert hinzugefügt werden

```
1 public class main {
2   public static void main(String[] args) {
3     boolean b = true;
4     int i = 14;
5     float f = 3.14159265;
6     char c = 'l';
7     String s = "Hello World";
8   }
9 }
```

Um also eine Information abspeichern zu können, braucht es zuerst einen Bezeichner, der angibt, was abgespeichert werden soll und dem Computer sagt, wieviel speicher er reservieren muss. Als zwetes wird diesem Speicher einen Namen gegeben, um diesen später wieder abrufen zu können. Es ist aber nicht erlaubt, zwei dieser Speicher mit demselben Namen zu versehen. als letztes kann optional noch ein Wert zugewiesen werden, der in diesen Speicher passt, der vom Bezeichner deklariert wurde.

Arrays Es gibt aber nicht nur diese einfachen Datentypen (auch primitiv gennant) sondern es gibt auch sogennante Arrays (Listen), die z.B. aus primitiven Datentypen besteht. Man kann also zum Beispiel ein Array aus Integers mit der Länge 5 generieren, das dann 5 Integers beinhaltet, auf welche man über das Array zugreifen kann. Jedoch muss bei den Standartarrays, auf welche hier eigegangen wird, immer schon am Anfang die Grösse festgelegt werden. Im Anschluss kann das Array dann gefüllt werden und die einzelnen Elemente können dann wieder abgerufen werden. Um ein Array zu deklarieren, muss zuerst der Datentyp, aus welchem das Array besteht, hingeschrieben werden und gleich im Anschluss zwei eckige Klammern. Dann muss, wie bei primitiven Datentypen, der Bezeichner hingeschrieben werden. Um die Grösse zu bestimmen muss dann hinter einem Gleichheitszeichen ein new und nocheinmal der Datentyp mit den eckigen Klammern hingeschrieben werden. Innerhalb der eckigen Klammern muss dann die Grösse des Arrays geschriben werden. Um dann auf einen beliebigen Element des Arrays zuzugreifen, schreibt man ganz normal den Variablennamen und dann innerhalb der eckigen Klammern die Position des Elementes. Die Positionszahl muss sich aber zwischen 0 und n-1 befinden, wobei n die Grösse des Arrays darstellt. Da üblicherweise auch 0 ein Element in der Informatik darstellt, bildet bereits n-1 das letzte Element. Nun ein Beispiel dazu:

```
1  public class main {
2    public static void main(String[] args) {
3      int[] array = new int[5];
4      array[0] = 0;
5      array[1] = 2;
6      array[2] = 4;
7      array[3] = 6;
8      array[4] = 5;
9      System.out.println("Das dritte Element des Arrays: " + array[3]);
10     System.out.println("Das nullte Element des Arrays: " + array[0]);
11   }
12 }
```

Konsole:

```
Das dritte Element des Arrays: 6
Das nullte Element des Arrays: 0
```

2.3.3 Rechnen mit Datentypen

Rechnen mit Boolean Da Boolean nur zwei Zustände kennen, nämlich true und false haltet sich auch die Menge der möglichen Operationen in Grenzen.

```
1  public class main {
2    public static void main(String[] args) {
3      boolean a = false;
4      boolean b = false;
5      boolean c = true;
6      boolean d = true;
7      //Nur wenn beide Werte bereits true beinhalten, wird true
       zurueckgegeben (AND)
8      boolean e = a && b;
9      System.out.println("false && false = " +e);
10     e = a && c;
11     System.out.println("false && true = " +e);
12     e = c && d;
13     System.out.println("true && true = " +e);
14     //Wenn einer der beiden Werte true besitzt, wird true
       zurueckgegeben (OR)
15     e = a || b;
16     System.out.println("false || false = " +e);
17     e = a || c;
18     System.out.println("false || true = " +e);
19     e = c || d;
20     System.out.println("true || true = " +e);
21     //Der aktuelle Wert wird invertiert
22     e = !a;
23     System.out.println("!a = " +e);
24     e = !c;
25     System.out.println("!c = " +e);
26     //Zwei Werte werden verglichen
27     e = a == b;
28     System.out.println("false == false = " +e);
29     e = a == c;
30     System.out.println("false == true = " +e);
31     e = c == d;
32     System.out.println("true == true = " +e);
33   }
34 }
```

Konsole:

```
false && false = false
false && true = false
true && true = true
false || false = false
false || true = true
true || true = true
!a = true
!c = false
false == false = true
false == true = false
true == true = true
```

Rechnen mit Zahlen Bei Zahlen gibt es zwar wesentlich mehr Operatoren, diese funktionieren aber genau gleich wie in der Mathematik. Sobald eine ganze Zahl (Datentyp int) dividiert wird und die Division nicht aufgeht, wird die Zahl auf die nächst kleinere ganze Zahl abgerundet.

```
1  public class main {
2    public static void main(String[] args) {
3      int a = 5;
4      int b = 3;
5      int c = a/b;
6      //Ganzzahlen generieren auch einen Ganzzahligen Output
7      System.out.println("5/3 = " +c);
8      int c = a%b;
9      //Der Modulooperator gibt den Rest aus
10     System.out.println("5%3 = " +c);
11     float d = 5;
12     float e = 3;
13     float f = d/e;
14     //Gleitkommazahlen berechnen auch den Wert hinter dem Komma.
15     System.out.println("5/3 = " +f);
16   }
17 }
```

Konsole:

```
5/3 = 1
5%3 = 2
5/3 = 1.6666666
```

Es gibt natürlich auch noch andere Operatoren als nur +, -, * und /. Im Programmieren auch sehr wichtig ist der Modulo Operator % . Dieser gibt immer den Rest einer Division zurück (siehe oben). Die anderen Datentypen besitzen natürlich ebenfalls Operatoren, auf diese wird hier aber nicht genauer eingegangen.

2.3.4 Die if-Abfrage

Bei einer if-Abfrage können Bedingungen geprüft werden, die entweder true oder false sind. Ist der Wert true, wird der Inhalt der if-Abrage ausgeführt, ansonnsten nicht.

```
1  public class main {
2    public static void main(String[] args) {
3      boolean IMTrue = true;
4      boolean IMFalse = false;
5      if(IMTrue){ //Wert wird hier ueberprueft
6        /*Ab hier bis zur geschlossenen geschweiften Klammer wird der
       Code nur ausgefuehrt, falls die Bedingung wahr ist.*/
7        System.out.println("IMTrue-Abfrage positiv");
8      }
9      if(IMFalse){ //ein neuer Wert wird ueberprueft
10       System.out.println("IMFalse-Abfrage positiv");
11     }
12   }
13 }
```

```
IMTrue-Abfrage positiv
```

Auch innerhalb der Bedingungsklammern der if-Abrage können Berechnungen durchgeführt werden. Ausserdem kann es auch vorkoommen, dass irgendein Befehl ausgeführt werden soll, wenn die Abfrage korrekt war und ein anderer, wenn die Abfrage falsch war. Für das gibt es dann die else-Abfrage. Wenn man dann sogar mehrere Abfragen von derselben Variable machen will, gibt es die else if-Abfrage..

```
1  public class main {
2    public static void main(String[] args) {
3      boolean IMFalse = false;
4      if(IMFalse){
5        System.out.println("IMFalse besitzt den Wert true");
6      }
7      else{
```

```
8         System.out.println("IMFalse besitzt den Wert false");
9       }
10      int a = 2;
11      if(a==1){
12        System.out.println("a besitzt den Wert 1");
13      }
14      else if(a==2){
15        System.out.println("a besitzt den Wert 2");
16      }
17      else{
18        System.out.println("a besitzt weder den Wert 1 noch den Wert 2"
        );
19      }
20    }
21  }
```

```
IMFalse besitzt den Wert false
a besitzt den Wert 2
```

Übrigens lassen sich auch Variablen in einer if-Abfrage definieren, jedoch kann man ausserhalb der if-Abfrage nicht mehr auf diese zugreifen.

2.3.5 Schleifen

Eine Schleife ist eine Konstruktion wie die if-Abfrage. Die Aufgabe der Schleife liegt aber nicht in der Abfrage eines Wertes sondern in der mehrmaligen Ausführung deren Inhaltes.

Die For-Schleife Die For-Schleife lässt sich aus 3 Komponenten aufbauen: Im ersten wird eine Variable deklariert und einem Wert zugewiesen. Im zweiten Teil muss eine Bedingung wie in einer if-Abfrage geschrieben werden. Falls die Bedingung wahr ist, wird die Schleife ein weiteres Mahl ausfegührt, ansonsten wird sie abgebrochen. Im letzten Teil muss die Variable irgendwie verändert werden, damit diese irgendwann abgebrochen wird. Wenn sie nicht abgebrochen werden würde, würde sie endlos andauern und das Programm würde dann stecken bleiben. Deswegen ist es sehr wichtig, dass sie immer irgendwann abgebrochen wird.

```
1 public class main {
2   public static void main(String[] args) {
```

```
3       for(int i = 0; i < 10; i++){ //Der ++Operator erhoeht die
        Variable i um 1
4         System.out.println("i besitzt im Moment den Wert " +i);
5       }
6     }
7   }
```

```
i besitzt im Moment den Wert 0
i besitzt im Moment den Wert 1
i besitzt im Moment den Wert 2
i besitzt im Moment den Wert 3
i besitzt im Moment den Wert 4
i besitzt im Moment den Wert 5
i besitzt im Moment den Wert 6
i besitzt im Moment den Wert 7
i besitzt im Moment den Wert 8
i besitzt im Moment den Wert 9
```

Die obige For-Schleife zeigt den normalen Aufbau: einer Variablen wird der Wert 0 zugewiesen und die Schleife wird ausgeführt bis ein gewisser Wert (hier im Beispiel 10) erreicht wurde. Dadurch ist schon im voraus klar, wie oft der Code ausgeführt wird.

Die While-Schleife Eine While-Schleife ist eigentlich das selbe wie eine For-Schleife, der enzige Unterschied liegt darin, dass die erste und die letzte Komponente der For-Schleife fehlen. Dadruch wird nur noch überprüft, ob ein Wert noch immer wahr ist. Die Aufgabe liegt jetzt beim Programmierer, wann er die Schleife abbrechen will bzw. wann die Abfrage falsch werden wird.

```
1   public class main {
2     public static void main(String[] args) {
3       boolean b = true;
4       int a = 8;
5       while(b){
6         a--; //Reduzuert den Wert a um 1
7         if(a<0){
8           /*Sobald der Wert von a kleiner als 0 wurde, wird b auf false
        gesetzt und somit die while-Schleife abgeborchen.*/
9           b = false;
10        }
```

```
11        System.out.println("Der Wert von a betraegt im Moment " +a);
12      }
13    }
14 }
```

```
Der Wert von a betraegt im Moment 7
Der Wert von a betraegt im Moment 6
Der Wert von a betraegt im Moment 5
Der Wert von a betraegt im Moment 4
Der Wert von a betraegt im Moment 3
Der Wert von a betraegt im Moment 2
Der Wert von a betraegt im Moment 1
Der Wert von a betraegt im Moment 0
Der Wert von a betraegt im Moment -1
```

Es gibt auch noch zwei andere wichtige Befehle in der Schleife: break und continue. Sobald ein continue ausgeführt wird, springt man wieder zum Anfang und die Schleife wird wieder von neuem ausgeführt, wobei auch die Bedingung neu überprüft wird und die Variable verändert wird. Break dagegen bricht die Schleife komplett ab.

```
1  public class main {
2    public static void main(String[] args) {
3      for(int i = 0; i < 1000; i++){
4        if(i%3 == 0){
5          /*Falls i ein Vielfaches von drei ist, wird diese Zahl
       uebersprungen*/
6          continue;
7        }
8        else if(i==10){
9          /*Falls i = 10 ist wird die Schleife abgebrochen*/
10         break;
11       }
12       else if(i%2 == 0){
13         /*Falls i eine gerade Zahl ist, wird diese Ausgegeben und
       dann wird die Schleife wieder neu gestartet*/
14         System.out.println("CONTINUE " +i);
15         continue;
16       }
17       /*Ansonsten wird die Zahl ausgegeben*/
18       System.out.println("Keine der Bedingungen erfuellt: " +i);
19     }
```

```
20    }
21 }
```

```
Keine der Bedingungen erfuellt: 1
CONTINUE 2
CONTINUE 4
Keine der Bedingungen erfuellt: 5
Keine der Bedingungen erfuellt: 7
CONTINUE 8
```

2.3.6 Methoden

Mit der Einführung der Methoden wird erstmals die Main-Methode verlassen. Methoden sind eine der besten Möglichkeiten, ein Projekt gut zu strukturieren. Eine Methode übernimmt im Normalfall ein kleines Sub-Problem des Programmes. Eine Methode ekönnte man also als ein speparates Programm ansschauen, das irgendetwas für das Hauptprogramm berechnet. Deshalb hat jede Methode mehrere Inputwerte (Variablen) und genau einen Outputwert (auch eine Variable). Man könnte also zum Beispiel eine Methode schreiben, die 2 Zahlen zusammenzählen kann. Jede Methode braucht unter anderem einen Namen, um diese von einem anderen Ort her benutzen zu können. Die Struktur zur deklaration einer Methode sieht also wiefolgt aus: Als erstes der Bezeichner static (darauf wird später noch genauer eingegangen), als zweites der Datentyp des Outputwertes, als drittes der Name der Methode und dann in Klammern die verschiedenen Inputwerte, die durch ein Komma voneinander getrennt sind und für jeder dieser Inputwerte braucht es eine Variablendeklaration (also Datentyp und Name). Die Inputvariablen können dann innerhalb der Methode benutzt werden und beliebig miteinander verrechnet werden. Als letztes wird dann der Outputwert mit return zurückgegeben. Der Datentyp der zurückgebenden Variable muss logischer Weise mit dem am Anfang deklarierten der Methode übereinstimmen. Hier ein Beispiel, in dem eine Methode zwei Werte zusammenzählt:

```
1 public class main {
2   public static void main(String[] args) {
3     int a = 2;
4     int b = 5;
5     int c;
6     c = addition(a, b);
```

```
7       System.out.println("The calculated result: " +c);
8     }
9     static int addition(int f, int d){
10      int e = f+d;
11      return e;
12    }
13  }
```

```
The calculated result: 7
```

Falls kein Wert von der Methode zurückgegeben werden sollte, kann anstelle eines Datentyps auch void geschrieben werden. Falls keine Input-Daten gebraucht werden, darf die Klammer einfach leer gelassen werden, sie muss aber trotzdem noch existent sein. Von einer Funktion kann auch eine weitere Funktion aufgerufen werden oder sie kann sich sogar selbst aufrufen (das zweite wird auch Rekursive gennant).

```
1  public class main {
2    public static void main(String[] args) {
3      start();
4    }
5    static void start(){
6      System.out.println("Thats the starting Method");
7      rek(5);
8    }
9    static void rek(int i) {
10     if(i > 0){
11       rek(i-1);
12     }
13     System.out.println("Thats the Rek Method and i has the value " +i
       );
14   }
15 }
```

```
Thats the starting Method
Thats the Rek Method and i has the value 0
Thats the Rek Method and i has the value 1
Thats the Rek Method and i has the value 2
Thats the Rek Method and i has the value 3
Thats the Rek Method and i has the value 4
Thats the Rek Method and i has the value 5
```

2.3.7 Klassen

Die grösste Struktur in Java wir Klasse gennant. Es ist auch der Grund, weshalb Java "'Objekt orientiert"' heisst. Bisher hat sich der ganze obere Abschnitt auf die Main-Klasse bezogen, in diesem Teil wird diese verlassen. Mit dem Prinzip des Objekt orientieren Programmierens öffnen sich einem eine riesen Menge neuer Möglichkeiten. Eine Klasse könnte man ganz einfach als eigener Datentyp anschauen, der das kann, was man selbst deffiniert hat. Gleich wie die Main-Klasse, die immer in der Main-Methode beginnt, hat auch ein Objekt dasselbe. Diese Main-Methode einer neuen Klasse wird aber Konstruktor gennant. Der Konstruktor muss immer denselben Namen tragen wie der der Klasse lautet. Wenn man also ein neues Objekt erzeugt (wir so gennant, wenn man eine Klasse aufruft), wird zu aller erst dieser Konstruktor aufgerufen und dessen Inhalt ausgeführt. Im vorherigen Kapitel wurden die Methoden durchgenommen. Eine Methode muss nicht zwingendermasen vom Konstruktor aufgerufen werden, um ausgeführt zu werden. Sie kann, wenn man eben ein solches Objekt erzeugt hat auch von dem Erzeugungsort aufgerufen werden. Eine Klasse wird in eine separate Datei geschrieben. In der Main-Klasse wird dann ein Objekt dieser Klasse generiert. Das Erzeugen eines Objektes passiert fast genau gleich wie bei einem Array. Anstatt des Datentypes mit den eckigen Klammern muss aber der Klassenname aufgeführt werden und am Ende müssen noch die Argumente des Konstruktors der Klasse in runden Klammern eingeüllt werden. Hier ein Beispiel, in welchem ein Objekt der Klasse test erzeigt wird, welche einen Parameter des Datentyps Integer verlangt.

```
1 public class main {
2   public static void main(String[] args) {
3     Test t1 = new T est(5);
4   }
5 }
```

Die Klasse test könnte so aussehen:

Test.java

```
1 public class Test {
2   /* Eine globale Variable wird erstellt, auf welche alle Methoden
      zugreifen koennen*/
3   int ga;
4   public Test(int a) {
```

```
5     /*Der dem neu generierten Objekt mitgegebenen Parameter wird vom
      Konstruktor auf eine globale Variable kopiert, damit auch von den
      Methoden aus darauf zugegriffen werden kann*/
6     ga = a;
7   }
8
9   public int square(int n){
10    //In dieser Methode wird die n-te Potenz der Zahl ga ausgegeben
11    int result = 1;
12    for(int i = 0; i < n; i++){
13      result = result*ga;
14    }
15    return result;
16  }
17 }
```

Und die Main-Klasse etwa so:

main.java

```
1  public class main {
2    public static void main(String[] args) {
3      /*Zwei Objekte werden generiert mit den Parametern 5 und 3*/
4      Test t1 = new Test(5);
5      Test t2 = new Test(3);
6      /*Es wird von der in der Klasse beinhaltenden Methode square
       gebrauch gemacht*/
7      System.out.println("5^4 = " + t1.square(4));
8      System.out.println("5^9 = " + t1.square(9));
9      System.out.println("3^0 = " + t2.square(0));
10     System.out.println("3^5 = " + t2.square(5));
11   }
12 }
```

Konsole

```
5^4 = 625
5^9 = 1953125
3^0 = 1
3^5 = 243
```

2.4 Android Studio

Android Studio ist eine für das Andoid 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 propietäre 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-Empulator (Erleichtert das testen der App auf einem Computer).

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

- **Linker, bzw. Zusammentragen von zusätzlichen Paketen und androidspezifischen Dateien, die für das Kompillieren 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 geschriebenen Code und Assets ein lauffähiges Programm zu erstellen. Er liest jede Datei durch und bereitet sie auf die nachfolgenden Verar-

beitungsprozesse vor, indem er unter anderem Kommentare entfernt, nichtgenutzte Funktionen und Variabeln entfernt, Makros ersetzt und durch die `include`-Kontrollzequenz 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 Kompilliervorgang eine geringere Laufzeit und Speicheraulastung 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 sogt 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 additionsoperation 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 stürtzt 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 etabiliert: armeabi-v7, Intel x86 und AMD64. Die App wird also in mehrere Sprachen übersetzt bevor sie ausgeleifert wird.

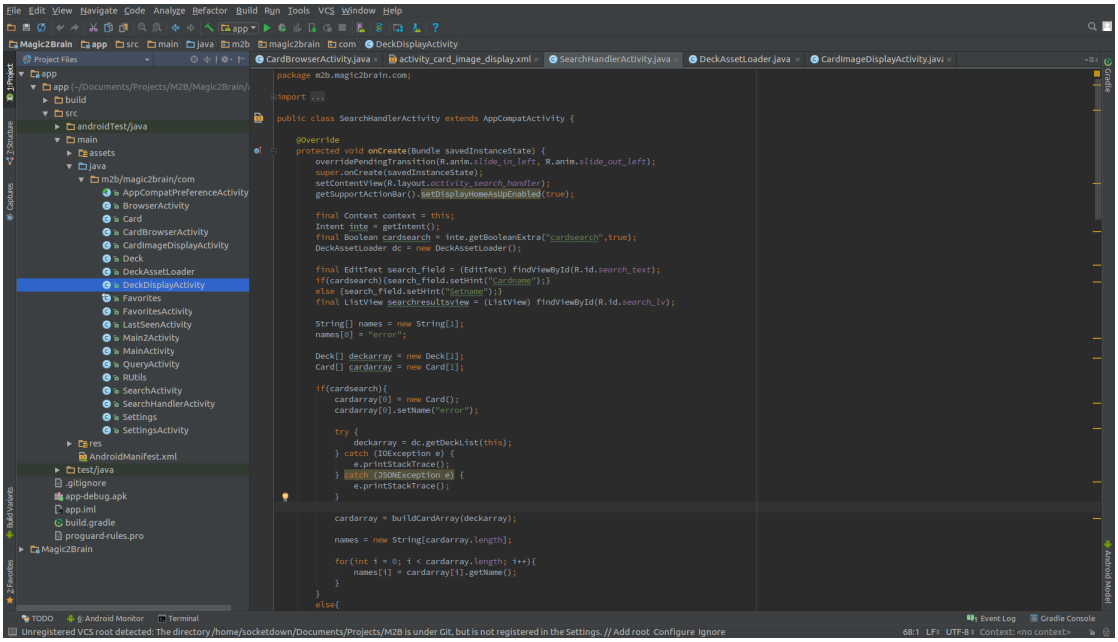


Figure 5: Android Studio Benutzeroberfläche [**?**]

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 registirert 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 funktionnieren. 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, sonder 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.

2.4.1 Programmieren mit dem Android SDK

Das Android SDK ist eines der wichtigsten Elemente in Android Studio. Man muss zwar nicht zwingendermassen Android Studio herunterladen, um dieses SDK zu nutzen, da es als Open Source Produkt frei verfügbar ist. Das Eigentliche Entwickeln basiert auf bereits vorgegebenen Klassen, welche man über das Klassenattribut `extends` mit seinem eigenen Code erweitert. Selbstverständlch gehen dabei die herkömmlichen Progrmmiermöglichkeiten von Java nicht verloren, da es einem immernoch möglich ist eigene Klassen und Bibliotheken zu erstellen. Eine Besonderheit von Android ist die enge Verbindung zwischen den XML-Dateien, welche Layouts, Strings und Vektorgrafiken der einzelnen Viewports enthalten. Diese Verbindung wird einzig und allein von Gradle aufrecht erhalten. Eine aus statischen Variabeln bestehende indexierte Liste wird somit bei jedem Gradle-Build resp. bei jeder Veränderung in der Ordnerstruktur von neuem erstellt.

2.4.2 Die Bedeutung von XML Dateien für Android

Das Auslagern von bestimmten Informationen in XML-Dateien ist für den erfahrenen Programmierer eine sehr schnelle, platzsparende aber auch nützliche Angelegenheit. Somit kann er sich einige Zeilen Code sparen um ein Element einer Activity hinzuzufügen. Als Beispiel dafür wird das Erstellen eines Texteingabefeldes in einer Activity benutzt.

Dieses Resultat kann entweder mit XML bewerkställigt werden, aber auch mit Java selber. Im grunde ist einem diese Entscheidung selber überlassen, da das Wichtigste beim Programmieren persönliche prefärenzen sind. Dies funktionniert nach dem Schema "Solange sich der Programmierer wohl fühlt, leistet er gute Arbeit".

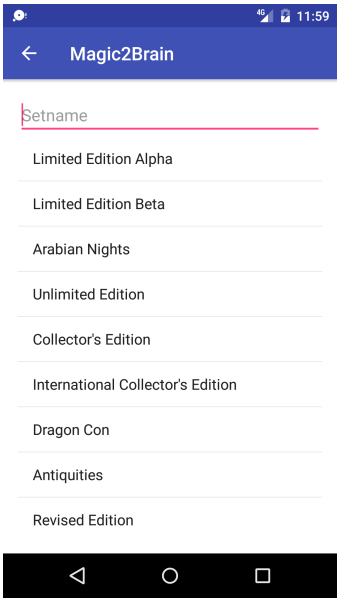


Figure 6: Search Activity [**?**]

XML

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/
    android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/activity_search_handler"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context="m2b.magic2brain.com.SearchHandlerActivity">

    <EditText
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:inputType="textPersonName"
        android:ems="10"
        android:id="@+id/search_text"
        android:layout_alignParentTop="true"
```

```
20          android:layout_alignParentStart="true"
21          android:layout_alignParentEnd="true"
22          android:contentDescription="Name" />
23 </RelativeLayout>
```

Das Element "EditText" ist hier das Texteingabefeld. Im wird mit `android:id=` eine eindeutige ID zugewiesen, mit der später vom Code aus auf das Element zugegriffen werden kann. In Java wird das XML folgendermassen eingebunden und auf das Element zugegriffen:

```
1 //Wichtige imports werden ausgelassen
2 import m2b.magic2brain.com.magic2brain.R;
3
4 //Das Element wird mit der Funktion findViewByID(int ID)
5 //in der indexierten liste gesucht und gefunden
6
7 EditText mTextField = (EditText) findViewByID(R.id.search_text);
8
9 //nun kann mit der Variabel mTextField wie gewohnt gearbeitet werden
```

Java

Auch bei einer puren Umsetzung in Java ist es nicht ganz möglich dem XML zu entrinnen. Um ein Objekt an einen View hinzuzufügen, muss zu allererst ein Layout verfügbar gemacht werden. Folglich sieht das XML deutlich kürzer aus, muss aber vorhanden sein. Es muss ausserdem darauf geachtet werden, dem Layout eine ID zu geben, sonst ist es nicht möglich vom Code aus auf das Layout zuzugreifen.

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <RelativeLayout xmlns:android="http://schemas.android.com/apk/res/
      android"
3     xmlns:tools="http://schemas.android.com/tools"
4     android:id="@+id/activity_search_handler"
5     android:layout_width="match_parent"
6     android:layout_height="match_parent"
7     android:paddingBottom="@dimen/activity_vertical_margin"
8     android:paddingLeft="@dimen/activity_horizontal_margin"
9     android:paddingRight="@dimen/activity_horizontal_margin"
10    android:paddingTop="@dimen/activity_vertical_margin"
11    tools:context="m2b.magic2brain.com.SearchHandlerActivity">
12 </RelativeLayout>
```

Wie bereits erwähnt besteht dieses XML nur noch aus dem rohen Layout block. `RelativeLayout` ist eine besondere Art von Layout, die sich hauptsächlich für Apps eignet, die später auf Endgeräten mit verschiedenen Bildschirmgössen vertrieben wird. Das einbinden in den Code geschieht wie gewohnt über `findViewByID(R.id.activity_search_handler);`

```
1  //Imports werden hier ausgelassen
2
3  //Das Layout muss f r die nachfolgenden Schritte gefunden werden
4  RelativeLayout lyt = (RelativeLayout) findViewById(R.id.
       activity_search_handler);
5
6  //Ein neues, unzugeordnetes Textfeld wird erstellt
7  EditText mTextField = new EditText(this);
8
9  //Nun wird das Textfeld zum Layout hinzugef gt
10 lyt.addView(mTextField);
```

Man sollte XML nur dann benutzen, wenn man wenig an seinem bisher entworfenen XML verändert. Würde man dynamisch Elemente wie z.B. Knöpfe oder Bilder hinzufügen, empfiehlt es sich dieses Problem programmiertechnisch anzugehen.

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 der Ergebnisse

Beim Start der App sieht man das Logo der App als Splash-Screen (Abbildung 7). Nach dem Laden wird man von der Home-Seite begrüsst. Streift man mit dem Finger von links nach rechts oder klickt auf den Menu-Sandwich, öffnet sich ein Side-Menü. Das Side-Menü hat oben wieder das Logo und 2 Texte darunter (”‘Magic 2 Brain”’ und ”‘Your MTG learning companion”’). Darunter befinden sich 7 Buttons mit den Texten: ”‘Home”’, ”‘Search”’, ”‘Set Browser”’, ”‘Quick Learn”’, ”‘Favorites”’, ”‘Recently Learned”’ und ”‘Share”’ (Abbildung 7).

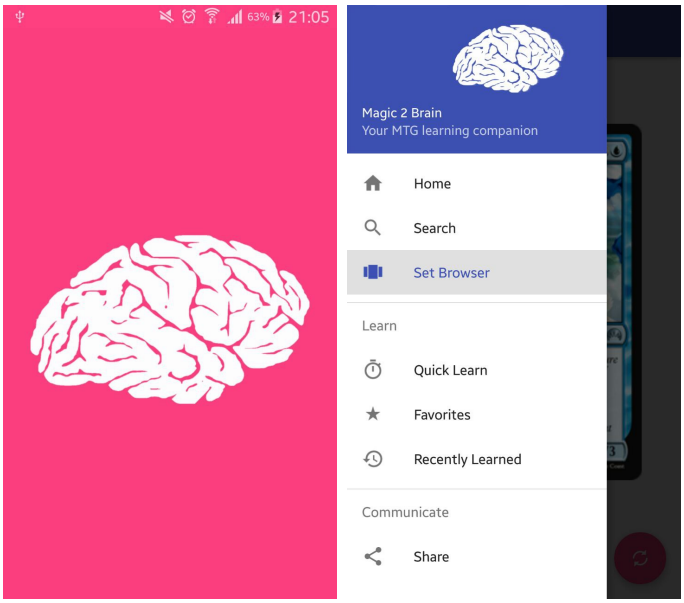


Figure 7: Links: Der Splash-Screen, Rechts: Das Menü

4.1 Home

Der Home-Screen ist simpel aufgebaut. Zuoberst steht ”‘Magic2Brain”’ und darunter ”‘Random Card”’. Unter diesem Text ist eine zufällige Karte abgebildet. Mit einem Klick auf die Karte lässt die sich vergrössern. Ganz unten rechts ist ein runder Knopf. Dieser ersetzt bei einem Klick die Karte mit einer anderen zufälligen Karte. (Abbildung 8).

Figure 8: Der Home-Screen

4.2 Search

Klickt man auf "'Search"' kann man auswählen, ob man nach Karten oder Sets suchen möchte. Wählt man einen aus, kommt man zu einer Ansicht mit einem Textfeld und einer Liste darunter. Tippt man etwas in das Textfeld ein, ändert sich die Liste und zeigt nur die Ergebnisse der Suche an.

4.3 Set Browser

Der Set-Browser zeigt alle Sets in einer Liste an (Abbildung 9). Klick man auf ein Set, kommt man zu einer weiteren Ansicht mit einer Liste. Diese Liste beinhaltet alle Karten vom ausgewählten Set. Klickt man auf eine Karte erhält man einen genaueren Blick auf die Karte. Bild, Text, Manakosten und der Name der Karte werden angezeigt (Abbildung 9). Zudem hat man einen Knopf mit einem Herzen drauf. Klickt man diesen, wird die Karte zu den Favoriten hinzugefügt.

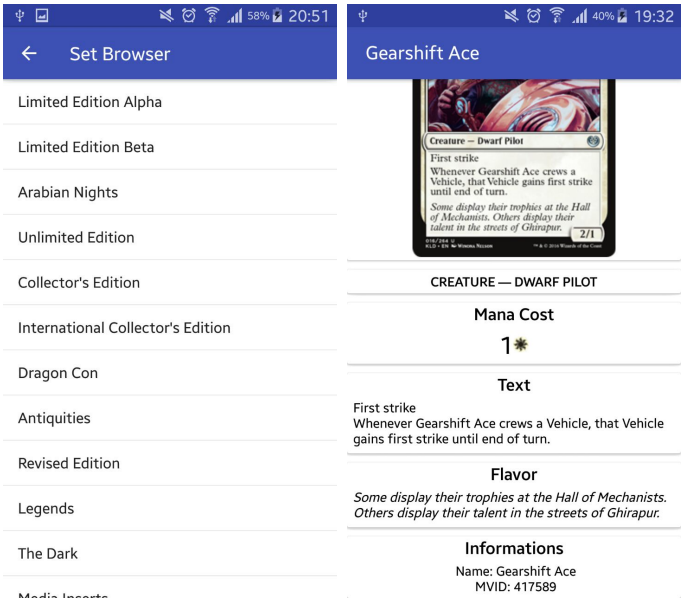


Figure 9: Links: Der Set-Browser, Rechts: Die Karten-Ansicht

4.4 Quick Learn

"'Quick Learn"' startet eine Abfrage mit dem aktuellsten Set. In diesem Fall ist es Kaladesh. (Abbildung 10).

4.5 Favorites

Klickt man auf Favoriten, werden alle Favoriten in einer Liste angezeigt.

4.6 Recently Learned

"'Recently Learned"' zeigt die letzten 10 gelernten Sets in einer Liste an.

4.7 Share

Klickt man auf diesen Knopf, öffnet sich ein Menü. Bei diesem Menü kann man auswählen über welche App man den Link zu dieser App versenden möchte.

Figure 10: Die Abfrage eines Sets

5 Diskussion der Ergebnisse

5.1 Diskussion der Teilaspekte

Unsere Teilaspekte umfassten vier Punkte:

1. **Der Benutzer muss Intuitiv mit einer anschaulichen Benutzeroberfläche zurechtkommen.**

   Dieser Punkt wurde recht gut erfüllt, da die Struktur sehr logisch aufgebaut ist und auch anderen ähnlichen Applikationen wie z.B. Quizlet, das für das Lernen von Vokabeln dient, gleicht. Auch haben mehrere Testpersonen die App als einfach und intuitiv zu handhaben bezeichnet. Auch grafisch und im Aufbau entspricht die App dem heutigen Standart.

2. **Der Benutzer kann sich selber aussuchen, welche Karten er lernen will und welche nicht.**

   Auch dieser Punkt wurde zweifelsohne erfüllt. In einem Menu kann der Benutzer das Set, das er lernen will, zuerst in einer Suchleiste eingeben und

dann über einen einfachen Knopfdruck lernen.

3. **Dem Benutzer stehen eine Android-OS App und eine Webseite zum Lernen bereit.**

   Dieser Punkt wurde leider nur teilweise erfüllt: Zwar steht dem Benutzer die versprochene Android-App zur Verfügung aber leider kann er auf der Webseite die Karten nicht lernen. Dieser Fakt ist aus dem Wechesel des Frameworks herzuleiten. Wenn wir das Projekt mit dem anfangs gewählten Phonegap abgeschlossen hätten, hätten wir automatisch zur Applikation auch eine funktionierende Webseite erhalten, da Phonegap beides erzeugen kann. Wegen dem Wechsel auf Android Studio wurde uns dieser Punkt leider verwehrt. Wegen mangelnder Zeit konnten wir nicht auch noch eine Webseite entwickeln. Das heisst aber auf keinen Fall, dass die Webseite nicht existiert. Die Webseite hat nun die einfache Funktion einer Homepage, auf der man die Applikation herunterladen kann. Sie ist auf **www.magic2brain.com** im Internet auffindbar.

4. **Das Backend, bzw. das Programm wird weitgehend autonom (wartungsfrei) arbeiten. Dabei ist die Problematik zu beachten, dass ständig neue Spielkarten veröffentlicht werden, welche ins System eingetragen werden müssen.**

   Auch dieser Punkt musste aufgrund mangelnder Zeit vernachlässigt werden, da die Priorität auf der Funktionsfähigkeit der Applikation lag. Das Problem mit den ständig neu erscheinenden Spielkarten wird jetzt ganz einfach durch regelmässige Updates der Applikation gelöst, damit der Benutzer trotzdem immer die neusten Karten lernen kann. Es ist auch gut möglich, dass wir nach der Projektabgabe diesen Punkt noch beheben werden.

5.2 Diskussion des Endergebisses

Nur Dank unserer bereits reichlich gesammelten Erfahrung war es möglich, eine solche Applikation umzusetzen, da wir uns genau nicht mehr stundenlang mit den Grundlagen auseinandersetzen mussten. Trotzdem haben wir uns auf Neu-

land gewagt und unseren Horizont erweitern können im Bereich der Entwicklung einer Android-Applikation. Wir haben es tatsächlich geschafft, eine ansehliche Applikation zu erstellen, welche zumindest im grafischen Bereich und in der Benutzerfreundlichkeit dem heutigen Standart entspricht. In der Optionen, über die der Beutzer verfügt, mussten wir aber sparen. Unser Entwicklungsteam bestand nur aus 3 Personen und eine Applikation mit reichhaltiger Funktionalität war deshalb schlicht und einfach nicht möglich. Dazu war die Zeit, die uns für die Projektarbeit zur Verfügung stand zu knapp bemessen, da wir zum einen auch den Schulalltag bewältigen mussten und zum anderen auch diese Arbeit hier schreiben mussten. Wenn man aber bedenkt, dass die meisten erfolgreichen Applikationen von einem professionellen Vollzeit Entwicklerteam geschrieben wurden, ist unser Produkt einer Projektarbeit definitv würdig. Deshalb sehen wir unser Projekt als gelungen an.

5.3 Diskussion des methodischen Vorgehens

Was man jedoch als grosser Kritikpunkt einwerfen muss und was man für ein nächstes Projekt besser machen muss ist die Errörterung eines geeigneten Frameworks. Während mehreren Wochen haben wir an einem Projekt mit Phonegap gearbeitet und viel Zeit und Arbeit darin investiert um dann zu merken, dass Phonegap für unsere Art von Projekt überhaupt nicht geeignet ist. In diesem Fall haben wir im Voraus viel zu wenig Nachforschung betrieben, um das geignete Framework zu finden, dies soll uns ein Lehre sein, es beim nächsten Mal besser zu machen. Die Arbeit an sich verlief sonst aber reibunglos mitunter dank der Organisationssoftware GitHub und der guten Aufteilung der Arbeit. GitHub hat uns hierbei nicht nur bei der Appikation sondern aucnh bei der schriftlichen Arbeit sehr geholfen. Dank der Include-Fuktion von LaTeX war es auch möglich, die Projektarbeit in einzelne Teile aufzuteilen wodurch ein paralleles Arbeiten an den Teilen problemlos funktionierte und trotzdem jeder von uns immer die aktuelste Version besass. Des Weiteren hat uns auch unsere Erfahrung von Projektarbeiten vom Lehrgang Infcom geholfen, dieses richtig zu organisieren. Auch während des Infcom-Unterrichts mussten wir mehrmals eine technische Arbeit organisieren und konnten dadurch aus der bereits gesammelten Erfahrung schöpfen.

6 Zusammenfassung

Unsere Projektidee entstrand mit dem hinblick auf neue und bestehende Turnierspieler der Kartenspiels Magic: the Gathering (*kurz: MTG*). Jeden Monat kommt ein neues Kartenpaket heraus. Dieses wird Set genannt. Die karten in jedem Set haben andere Eigenschaften in der Angriffsstärke, der Verteidigung oder haben sogar besondere oder einzigartige Eigenschaften. Um nicht ständig diese Daten von den Karten zu lesen wollten wir eine App machen, die genau diese Daten deren Nutzern beibringt. Ein Spiel hat nämlich auch eine zeitliche Begrenzung, die in einem Unentschieden endet. Daher ist es für beide Spieler von vorteil wenn sie schnell spielen.

Um dies zu bewerkstelligen haben wir zahlreiche APIs *(**A**pplication **P**programming **I**nterface)* verglichen. Zu beachten gab es 3 grundlegende Komponenten: Bilder der Karte, Text mit Eigenschafen der Karte und Eine Möglichkeit den Fortschritt zu speichern. Die Bilder haben wir direkt von dem Kartenbrowser von Wizards of the Coast genommen. Als API für die Karteninformationen hat MTGJSON gesiegt, wobei dies keine API ist, sondern eine Downloadquelle für Karten im JSON Format. JSON ist eine schnelle und einfach zu begreifende Alternative zu XML. Somit kann die App auch offline verwendet werden. Bilder, welche einmal heruntergeladen wurden, werden bis zum deinstallieren der App oder bis zum manuellen löschen auf dem Gerät des Benutzers gespeichert. Informationen über den Fortschritt werden in den *SharedPreferences* von Android gespeichert. Dies kann man sich wie ein Karteisystem vorstellen. Jede App erhält ihr eigenes Fach und kann dort Informationen ablegen.

Ursprünglich wollten wir unsere App in HTML, CSS und JavaScript schreiben, welche wir dann mit Cordova zu einer Android und IOS App kompilliert hätten. Jedoch wurde uns durch die Natur von JavaScipt verwehrt einige essentielle Funktionen zu implementieren. Dies lag vorwiegend daran, dass JavaScript Asynchron arbeitet und man dies nicht mehr wie damals ausschalten kann. Asynchron beschreibt die Arbeitsweise von einer Programmiersprache. In diesem fall werden mehrere Funktionen gleichzeitig ausgeführt, was ein gut durch dachtes und stabiles Programm erfordert. Ausserdem müssen in einigen Fällen *"Promises"* zurückgegeben werden. Das sind verweise auf einen Wert, der in der Zukunft existieren wird, aber noch nicht bekannt ist wann es so weit ist. Als Beispiel

nehme man eine ladende Webseite. Niemand weiss wann sie fertig geladen ist, da das von der Verbindungsgeschwindikeit abhängt, aber der Fakt, dass sie einmal geladen sein wird ist unbestritten. Promises kann man auch nicht ohne weiters abwarten bzw. *auflösen*, weil das in sehr vielen Fällen kritische auswirkungen auf die Geschwindigkeit des Programms hat. Synchrone sprachen hingegen arbeiten den Programmcode wie eine Art Stapel ab. Zeile für Zeile liest der Compiler den Code durch und wandelt ihn in Maschienensprache um. Das lässt dem Programmierer wenig Spielraum für Fehler übrig. Weil die Zeit schon relativ fortgeschritten und die App gröstenteil fertig war entschieden wir uns nur schewren Herzens dazu unser projekt aufzugeben und die App in Java zu schreiben. In Java schritten wir jedoch viel schneller voran als in JS.

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.xm1math.net/texmaker/** [online 5.1.2017]

[6] Wikipedia, the free encyclopedia. **https://de.wikipedia.org/wiki/UTF-8** [online 6.1.2017]

[7] BIld einer ASCII-Tabelle **http://www.theasciicode.com.ar/** [online 6.1.2017]

[8] **http://www.hbksaar.de/uploads/media/Dateiformate_Bilddateien.pdf** [online 7.1.2017]

7 Anhang

7.1 MainActivity.java

```
1  package m2b.magic2brain.com;
2
3  import android.content.Intent;
4  import android.content.SharedPreferences;
5  import android.graphics.Color;
6  import android.os.Bundle;
7  import android.preference.PreferenceManager;
8  import android.support.design.widget.FloatingActionButton;
9  import android.support.design.widget.NavigationView;
10 import android.support.v4.view.GravityCompat;
11 import android.support.v4.widget.DrawerLayout;
12 import android.support.v7.app.ActionBarDrawerToggle;
13 import android.support.v7.app.AppCompatActivity;
14 import android.support.v7.widget.Toolbar;
15 import android.view.Gravity;
16 import android.view.MenuItem;
17 import android.view.View;
18 import android.view.Window;
19 import android.view.WindowManager;
20 import android.widget.ImageView;
21 import android.widget.RelativeLayout;
22 import android.widget.TextView;
23 import com.google.gson.Gson;
24 import com.google.gson.reflect.TypeToken;
25 import com.squareup.picasso.Picasso;
26 import java.lang.reflect.Type;
27 import java.util.ArrayList;
28 import java.util.Arrays;
29 import m2b.magic2brain.com.magic2brain.R;
30
31 /*
32 This is where our app starts. It has a menu, shows a random card and
      loads some needed data.
33 */
34
35 public class MainActivity extends AppCompatActivity implements
      NavigationView.OnNavigationItemSelectedListener {
```

```
36
37      public static Card[] cardarray; // We will store all cards in
        this array. This is important for the SearchActivity. It makes the
         whole process alot faster.
38      private ImageView imgv; // We will show the random card with this
         ImageView
39
40      protected void onCreate(Bundle savedInstanceState) { // This is
        the very first function the app will execute at the start of the
        app.
41          if (!checkCardArray()) { // We try to load our card-list.
42              saveCardArray(); // If we don't find the card-list, we
        have to generate it and save it.
43          }
44          hideStatusBar(); // Does what is says. It hides the Statusbar
45          setTheme(R.style.AppTheme_NoActionBar);  // We want to hide
        the ActionBar too
46          super.onCreate(savedInstanceState); // This does some intern
        stuff. We don't need to worry about that. It's just needed.
47          setContentView(R.layout.activity_main); // This adds an View
        to our Activity. We defined at "/res/layout/activity_main.xml" how
         our activity should look like.
48          Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar); //
        With this we get the Toolbar of the View.
49          setSupportActionBar(toolbar); // We add the Toolbar as a
        SupportActionBar. If we click something on the Toolbar it will
        call onNavigationItemSelected();
50
51          FloatingActionButton fab = (FloatingActionButton)
        findViewById(R.id.fab); // We get the "new random card"-button
52          fab.setOnClickListener(new View.OnClickListener() {
53              public void onClick(View view) {
54                  setListener(setRandomCard()); // If we click the
        button it should generate a new random card.
55              }
56          });
57          // The following lines generate the drawer (Sidemenu)
58          DrawerLayout drawer = (DrawerLayout) findViewById(R.id.
        drawer_layout);
59          ActionBarDrawerToggle toggle = new ActionBarDrawerToggle(
60                  this, drawer, toolbar, R.string.
```

```
    navigation_drawer_open, R.string.navigation_drawer_close);
61          drawer.setDrawerListener(toggle);
62          toggle.syncState();
63          NavigationView navigationView = (NavigationView) findViewById
    (R.id.nav_view);
64          navigationView.setNavigationItemSelectedListener(this);
65
66          Favorites.init(); // Initate Favorites
67
68          // The following code loads the Favorites from the memory
69          SharedPreferences appSharedPrefs = PreferenceManager.
    getDefaultSharedPreferences(this);
70          Gson gson = new Gson();
71          String json = appSharedPrefs.getString("favobj", null);
72          Type type = new TypeToken<ArrayList<Card>>() {
73          }.getType();
74          ArrayList<Card> favs = gson.fromJson(json, type);
75          Favorites.init();
76          if (favs != null) {
77              Favorites.favorites_mvid = favs;
78          }
79          buildNewsFeed(); // This builds the frontpage
80      }
81
82      private void hideStatusBar() { // This method simply removes the
    Statusbar
83          requestWindowFeature(Window.FEATURE_NO_TITLE); // remove the
    title
84          getWindow().setFlags(WindowManager.LayoutParams.
    FLAG_FULLSCREEN, WindowManager.LayoutParams.FLAG_FULLSCREEN); //
    set it to fullscreen
85      }
86
87      protected void onPause() {
88          super.onPause();
89          // If the user leaves our app in any way, we save all his
    favorites.
90          SharedPreferences appSharedPrefs = PreferenceManager.
    getDefaultSharedPreferences(this);
91          SharedPreferences.Editor prefsEditor = appSharedPrefs.edit();
92          Gson gson = new Gson();
```

```
93          String json = gson.toJson(Favorites.favorites_mvid);
94          prefsEditor.putString("favobj", json);
95          prefsEditor.commit();
96      }
97
98      public void onBackPressed() { // Closes the Drawer if we press
    the Back-Button on the phone
99          DrawerLayout drawer = (DrawerLayout) findViewById(R.id.
    drawer_layout);
100         if (drawer.isDrawerOpen(GravityCompat.START)) {
101             drawer.closeDrawer(GravityCompat.START);
102         } else {
103             super.onBackPressed();
104         }
105     }
106
107     public boolean onNavigationItemSelected(MenuItem item) {
108         // Handle navigation view item clicks here.
109         int id = item.getItemId();
110         if (id == R.id.nav_search) {
111             // Starts the SearchActivitiy if we click on Search
112             Intent intent = new Intent(this, SearchActivity.class);
113             startActivity(intent);
114
115         } else if (id == R.id.nav_favorite) {
116             // Starts the FavoritesActivity if we click ond Favorites
117             Intent intent = new Intent(this, FavoritesActivity.class)
    ;
118             startActivity(intent);
119
120         } else if (id == R.id.nav_browser) {
121             // Starts the BrowserActivity if we click on the
    Cardbrowser
122             Intent intent = new Intent(this, BrowserActivity.class);
123             startActivity(intent);
124
125         } else if (id == R.id.nav_quick_learn) {
126             // Starts the QueryActivity and passes the newest set (
    Kaladesh), if we click Quick learn
127             Deck d = new Deck();
128             Card[] c = DeckAssetLoader.getDeck("KLD.json",this);
```

```
129             d.setSet(c);
130             d.setName("Kaladesh");
131             d.setCode("KLD");
132             Intent i = new Intent(this, QueryActivity.class);
133             i.putExtra("Set", d);
134             startActivity(i);
135 
136         } else if (id == R.id.nav_history) {
137             // Starts the LastSeenActivity if we click recently
    learned
138             Intent intent = new Intent(this, LastSeenActivity.class);
139             startActivity(intent);
140 
141         } else if (id == R.id.nav_share) {
142             // Starts an Activity, which opens all possibilities to
    send the link to our app.
143             Intent sendIntent = new Intent();
144             sendIntent.setAction(Intent.ACTION_SEND);
145             sendIntent.putExtra(Intent.EXTRA_TEXT, getString(R.string
    .play_store_link));
146             sendIntent.setType("text/plain");
147             startActivity(Intent.createChooser(sendIntent,
    getResources().getText(R.string.send_to)));
148         }
149 
150         DrawerLayout drawer = (DrawerLayout) findViewById(R.id.
    drawer_layout); // We get the drawer
151         drawer.closeDrawer(GravityCompat.START); // Th drawer should
    start closed
152         return true;
153     }
154 
155     public void buildNewsFeed() { // This Function generates a text
    and a Imageview on the screen
156         int scrWidth = getWindowManager().getDefaultDisplay().
    getWidth(); // Get the width of the screen
157         int scrHeight = getWindowManager().getDefaultDisplay().
    getHeight(); // Get the Height of the screen
158         RelativeLayout lyt = (RelativeLayout) findViewById(R.id.
    main_absolute); // Get the View of the XML
159         RelativeLayout.LayoutParams params; // The parameters we want
```

```
    to add our TextView and ImageView
160
161         TextView score = new TextView(this); // Create new Textview
162         score.setTextColor(Color.BLACK);
163         score.setGravity(Gravity.CENTER);
164         score.setText("Random Card");
165         score.setTextSize(26);
166         params = new RelativeLayout.LayoutParams(scrWidth /*Width*/,
    (int) (0.1 * scrHeight)/*Height*/);
167         params.leftMargin = 0; // X-Position
168         params.topMargin = (int) (0.1 * scrHeight); // Y-Position
169         lyt.addView(score, params); // add it to the View
170
171         imgv = new ImageView(this); // Create new Imageview
172         params = new RelativeLayout.LayoutParams(scrWidth /*Width*/,
    (int) (0.6 * scrHeight))/*Height*/;
173         params.leftMargin = 0; // X-Position
174         params.topMargin = (int) (0.2 * scrHeight); // Y-Position
175         lyt.addView(imgv, params); // add it to the View
176         setListener(setRandomCard()); // Opens a random card
177     }
178
179     public void setListener(int MultiID2) { // This function sets an
    listener to the image, so if we click it, it shows a big image of
    the card.
180         final int MultiID = MultiID2;
181         imgv.setOnClickListener(new View.OnClickListener() {
182             public void onClick(View view) {
183                 Intent intent = new Intent(MainActivity.this,
    CardImageDisplayActivity.class);
184                 intent.putExtra("pic", MultiID);
185                 startActivity(intent);
186             }
187         });
188     }
189
190     public int setRandomCard() { // This gets a random card from the
    card-array and loads the image into the ImageView
191         int MultiID = 1;
192         Card c = cardarray[(int) (Math.random() * cardarray.length)];
193         if (c != null) {
```

```
194             MultiID = c.getMultiverseid();
195         }
196         Picasso.with(this)
197                 .load(getString(R.string.image_link_1) + MultiID +
    getString(R.string.image_link_2)) // This tries to load an image
    from a link.
198                 .placeholder(R.drawable.loading_image) // We want to
    show a image while its loading. We load our image from the "/res/
    drawable" folder
199                 .error(R.drawable.image_not_found)  // If it fails to
     load image we show an error-image.
200                 .into(imgv);  // This places the image into our
    ImageView.
201         return MultiID;
202     }
203
204     public boolean checkCardArray() { // This method checks if there
    is a saved version of our card array. If that's the case it tries
    to load it into the card-array.
205         SharedPreferences sharedPrefs = PreferenceManager.
    getDefaultSharedPreferences(this);
206         Gson gson = new Gson();
207         String json = sharedPrefs.getString("cardArray", null);
208         Type type = new TypeToken<Card[]>() {
209         }.getType();
210         Card[] aL = gson.fromJson(json, type);
211         if (aL == null) {return false;}
212         cardarray = aL;
213         return true;
214     }
215
216     public void saveCardArray() { // This function saves the card-
    array into the memory.
217         Card[] c = buildCardArray();
218         SharedPreferences sharedPrefs = PreferenceManager.
    getDefaultSharedPreferences(this);
219         SharedPreferences.Editor editor = sharedPrefs.edit();
220         Gson gson = new Gson();
221         String json = gson.toJson(c);
222         editor.putString("cardArray", json);
223         editor.commit();
```

```
224         cardarray = c;
225     }
226 
227     private Card[] buildCardArray() { // This method generates the
    card-array by opening all sets and loading the cards from it into
    out array.
228         ArrayList<Card> list = new ArrayList<>();
229         Deck[] deckarray = DeckAssetLoader.getDeckList(this);
230         for (int i = 0; i < deckarray.length; i++) {
231             //load current deck and append it to list
232             Card[] c = DeckAssetLoader.getDeck(deckarray[i].getCode()
    + ".json", this);
233             if (c[0].getName().equals("error")) {
234                 System.err.println("error ocurred at " + deckarray[i
    ].getCode() + ", please update your database");
235             } else {
236                 list.addAll(Arrays.asList(c));
237             }
238         }
239         return list.toArray(new Card[list.size()]);
240     }
241 }
```

7.2 SearchActivity.java

```
1  package m2b.magic2brain.com;
2
3  import android.content.Context;
4  import android.content.Intent;
5  import android.os.Bundle;
6  import android.support.v7.app.AppCompatActivity;
7  import android.view.MenuItem;
8  import android.view.View;
9  import android.view.View.OnClickListener;
10 import android.widget.Button;
11 import m2b.magic2brain.com.magic2brain.R;
12
13 /*
14 This Activity asks the user if he wants to search for cards or set
       and opens a SearchHandlerActivity if the answer.
15 */
16
17 public class SearchActivity extends AppCompatActivity {
18
19     protected void onCreate(Bundle savedInstanceState) {
20         overridePendingTransition(R.anim.slide_in_left, R.anim.
       slide_out_left); // This adds an fancy slide animation, when this
       activity starts.
21         super.onCreate(savedInstanceState); // This does some intern
       stuff. We don't need to worry about that. It's just needed.
22         setContentView(R.layout.activity_search); // This adds an
       View to our Activity. We defined at "/res/layout/activity_search.
       xml" how our activity should look like.
23         getSupportActionBar().setDisplayHomeAsUpEnabled(true); //
       With this line we add an "back"-Button to the Toolbar. If we press
        it it calls onOptionsItemSelected();
24         final Context context = this; // This doesn't actually do
       anything, but it's needed if we want to refer to "this" from an
       inner class.
25         setTitle(getString(R.string.SearchActivity_title)); // We set
        a title to our View. We defined the title in "/res/values/strings
       .xml". We just load it from there.
26         Button cards = (Button) findViewById(R.id.c_search_cards); //
        We get the cards-button
```

```
27          Button decks = (Button) findViewById(R.id.c_search_decks); // We get the sets-button
28
29          cards.setOnClickListener(new OnClickListener() {
30              public void onClick(View view) {
31                  // If the user presses the cards-button, we start SearchHandlerActivity and pass to it, that the user pressed "cards"
32                  Intent intent = new Intent(context, SearchHandlerActivity.class);
33                  intent.putExtra("cardsearch", true);
34                  startActivity(intent);
35              }
36          });
37
38          decks.setOnClickListener(new OnClickListener() {
39              public void onClick(View view) {
40                  // If the user presses the sets-button, we start SearchHandlerActivity and pass to it, that the user pressed "sets"
41                  Intent intent = new Intent(context, SearchHandlerActivity.class);
42                  intent.putExtra("cardsearch", false);
43                  startActivity(intent);
44              }
45          });
46      }
47
48      public boolean onOptionsItemSelected(MenuItem item) {
49          switch (item.getItemId()) {
50              // Respond to the action bar's Up/Home button
51              case android.R.id.home:
52                  onBackPressed();
53          }
54          return true;
55      }
56
57      public void onBackPressed() {
58          finish(); // This closes our Activity
59          overridePendingTransition(R.anim.slide_in_right, R.anim.slide_out_right); // We want to close it with an fancy animation.
60      }
```

```
61 }
```

7.3 SearchHandlerActivity.java

```
1  package m2b.magic2brain.com;
2
3  import android.content.Context;
4  import android.content.Intent;
5  import android.os.Bundle;
6  import android.support.v7.app.AppCompatActivity;
7  import android.text.Editable;
8  import android.text.TextWatcher;
9  import android.view.MenuItem;
10 import android.view.View;
11 import android.widget.AdapterView;
12 import android.widget.ArrayAdapter;
13 import android.widget.EditText;
14 import android.widget.ListView;
15 import m2b.magic2brain.com.magic2brain.R;
16
17 /*
18 This Activity is the one that actually searches for cards/sets.
19 */
20
21 public class SearchHandlerActivity extends AppCompatActivity {
22     Card[] cardarray; // This is the array, where all cards are
    stored
23
24     protected void onCreate(Bundle savedInstanceState) {
25         overridePendingTransition(R.anim.slide_in_left, R.anim.
    slide_out_left); // This adds an fancy slide animation, when this
    activity starts.
26         super.onCreate(savedInstanceState);  // This does some intern
     stuff. We don't need to worry about that. It's just needed.
27         setContentView(R.layout.activity_search_handler); // This
    adds an View to our Activity. We defined at "/res/layout/
    activity_search_handler.xml" how our activity should look like.
28         getSupportActionBar().setDisplayHomeAsUpEnabled(true); //
    With this line we add an "back"-Button to the Toolbar. If we press
     it it calls onOptionsItemSelected();
29
30         final Context context = this; // This doesn't actually do
    anything, but it's needed if we want to refer to "this" from an
```

```
inner class.
31          Intent inte = getIntent();  // We want to access any data
    that is passed.
32          final Boolean cardsearch = inte.getBooleanExtra("cardsearch",
     true); // We look if the user wants to search for a card or set
33
34          final EditText search_field = (EditText) findViewById(R.id.
    search_text); // We get the search-field
35          if (cardsearch) {
36              search_field.setHint(R.string.SearchHandler_hint_1); //
    if he wants to search a card, we set the hint to "Cardname"
37          } else {
38              search_field.setHint(R.string.SearchHandler_hint_2); //
    if he wants to search a set, we set the hint to "Setname"
39          }
40          final ListView searchresultsview = (ListView) findViewById(R.
    id.search_lv); // We get the ListView so we can show the results
41
42          String[] names = new String[1]; // We only need the names so
    we create a names-array
43          names[0] = getString(R.string.error); // This is to ensure
    that the array isn't empty
44          Deck[] deckarray = DeckAssetLoader.getDeckList(this); // We
    get all decks
45
46
47          if (cardsearch) {
48              cardarray = MainActivity.cardarray; // We load our card-
    array
49              names = new String[cardarray.length];
50
51              for (int i = 0; i < cardarray.length; i++) {
52                  names[i] = cardarray[i].getName(); // And add all
    names to the string-array
53              }
54          } else {
55              names = new String[deckarray.length]; // The same like
    cards but with sets
56
57              for (int i = 0; i < deckarray.length; i++) {
58                  names[i] = deckarray[i].getName();  // The same like
```

```
    cards but with sets
59              }
60          }
61
62          //finalizing variables for further use
63          final Deck[] cdeckarray = deckarray;
64          final Card[] ccardarray = cardarray;
65
66
67          final ArrayAdapter adapter = new ArrayAdapter(context,
    android.R.layout.simple_list_item_1, names); // We turn the string
    -array into a list
68          search_field.addTextChangedListener(new TextWatcher() { //
    and add a listener, so it updates when we write something
69              public void beforeTextChanged(CharSequence charSequence,
    int i, int i1, int i2) {}
70              public void onTextChanged(CharSequence charSequence, int
    i, int i1, int i2) {}
71              public void afterTextChanged(Editable editable) {
72                  adapter.getFilter().filter(editable.toString().
    toLowerCase());
73              }
74          });
75
76          searchresultsview.setAdapter(adapter); // We show the list
77          searchresultsview.setOnItemClickListener(new AdapterView.
    OnItemClickListener() {
78              public void onItemClick(AdapterView<?> parent, View view,
     int position, long id) {
79                  if (cardsearch) {
80                      // If it's a card-search, we open
    CardBrowserActivity with the clicked card.
81                      String cardname = adapter.getItem(position).
    toString();
82                      Intent intent = new Intent(context,
    CardBrowserActivity.class);
83                      intent.putExtra("currentCard", searchforcard(
    cardname, ccardarray));
84                      startActivity(intent);
85                  } else {
86                      // If it's a set-search, we open
```

```
    DeckDisplayActivity with the clicked Set
87                  String deckname = adapter.getItem(position).
    toString();
88                  Intent intent = new Intent(context,
    DeckDisplayActivity.class);
89                  intent.putExtra("code", searchfordeck(deckname,
    cdeckarray));
90                  intent.putExtra("name", deckname);
91                  startActivity(intent);
92              }
93          }
94      });
95
96  }
97
98  private Card searchforcard(String name, Card[] cardarray) { //
    This function searchs for a card and returns it
99      Card rc = new Card();
100
101     for (int i = 0; i < cardarray.length; i++) {
102         if (cardarray[i].getName().equals(name)) {
103             rc = cardarray[i];
104             break;
105         }
106     }
107     return rc;
108 }
109
110 private String searchfordeck(String name, Deck[] darray) { //
    This function searchs for a set and returns it
111     String code = getString(R.string.error);
112
113     for (int i = 0; i < darray.length; i++) {
114         String dname = darray[i].getName();
115
116         if (dname.equals(name)) {
117             code = darray[i].getCode();
118             break;
119         }
120     }
121
```

```
122         return code;
123     }
124 
125     public boolean onOptionsItemSelected(MenuItem item) {
126         int id = item.getItemId();
127         switch (id) {
128             // Respond to the action bar's Up/Home button
129             case android.R.id.home:
130                 onBackPressed();
131         }
132         return true;
133     }
134 
135     public void onBackPressed() {
136         finish(); // This closes our Activity
137         overridePendingTransition(R.anim.slide_in_right, R.anim.
    slide_out_right); // We want to close it with an fancy animation.
138     }
139 }
```

7.4 BrowserActivity.java

```
1  package m2b.magic2brain.com;
2
3  import android.content.Context;
4  import android.content.Intent;
5  import android.os.Bundle;
6  import android.support.v7.app.AppCompatActivity;
7  import android.support.v7.widget.Toolbar;
8  import android.view.MenuItem;
9  import android.view.View;
10 import android.widget.AdapterView;
11 import android.widget.AdapterView.OnItemClickListener;
12 import android.widget.ArrayAdapter;
13 import android.widget.ListView;
14 import m2b.magic2brain.com.magic2brain.R;
15
16 /*
17 This Activity should show a List with all sets and open
       DeckDisplayActivity with the selected set on a click.
18 */
19
20 public class BrowserActivity extends AppCompatActivity {
21
22     protected void onCreate(Bundle savedInstanceState) {
23         overridePendingTransition(R.anim.slide_in_left, R.anim.
       slide_out_left); // This adds an fancy slide animation, when this
       activity starts.
24         super.onCreate(savedInstanceState); // This does some intern
       stuff. We don't need to worry about that. It's just needed.
25         setContentView(R.layout.activity_browser); // This adds an
       View to our Activity. We defined at "/res/layout/activity_browser.
       xml" how our activity should look like.
26         Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar); //
       With this we get the Toolbar of the View.
27         setSupportActionBar(toolbar); // We add the Toolbar as a
       SupportActionBar. This is needed for the next line.
28         getSupportActionBar().setDisplayHomeAsUpEnabled(true); //
       With this line we add an "back"-Button to the Toolbar. If we press
        it it calls onOptionsItemSelected();
29         setTitle(getString(R.string.BrowserActivity_title)); // We
```

```
set a title to our View. We defined the title in "/res/values/
strings.xml". We just load it from there.
30      final Context currentContext = this; // This doesn't actually
 do anything, but it's needed if we want to refer to "this" from
an inner class.
31
32      Deck d[] = DeckAssetLoader.getDeckList(this); // We load all
Sets with our own function into an Deck-Array.
33      String[] it = new String[d.length]; // We just need the names
 of the sets. So we create an String-Array for the names.
34      for (int i = 0; i < d.length; i++) {
35          it[i] = d[i].getName();  // This for-loop adds the names
from every Deck from the Deck-Array to the String-Array.
36      }
37
38      final String[] listItems = it; // We need to finalize the
String-Array for the ArrayAdapter.
39      final Deck[] finalD = d; // We need to finalize the Deck-
Array, because we want to access it from an inner class.
40      final ArrayAdapter adapter = new ArrayAdapter(this, android.R
.layout.simple_list_item_1, listItems); // This is just a helper-
class. It transforms our String-Array into an clickable List.
41      ListView lv = (ListView) findViewById(R.id.deckbrowser_list);
 // We want to add our list to the ListView. So we get the
ListView from the View
42      lv.setAdapter(adapter); // We add our list into it.
43
44
45      lv.setOnItemClickListener(new OnItemClickListener() { // This
 performs an action when we click on an item from the list.
46          public void onItemClick(AdapterView<?> parent, View view,
 int position, long id) {
47              // The following code starts DeckDisplayActivity and
passes the Deck-Code and the Deck-Name of the item we clicked.
48              Intent intent = new Intent(currentContext,
DeckDisplayActivity.class);
49              intent.putExtra("code", finalD[position].getCode());
50              intent.putExtra("name", finalD[position].getName());
51              startActivity(intent);
52          }
53      });
```

```
54      }
55
56      public boolean onOptionsItemSelected(MenuItem item) {
57          switch (item.getItemId()) {
58              // Respond to the action bar's Up/Home button
59              case android.R.id.home:
60                  onBackPressed();
61          }
62          return true;
63      }
64
65      public void onBackPressed() {
66          finish(); // This closes our Activity
67          overridePendingTransition(R.anim.slide_in_right, R.anim.
    slide_out_right); // We want to close it with an fancy animation.
68      }
69
70  }
```

7.5 DeckDisplayActivity.java

```
1  package m2b.magic2brain.com;
2
3  import android.content.Context;
4  import android.content.DialogInterface;
5  import android.content.Intent;
6  import android.os.Bundle;
7  import android.support.design.widget.FloatingActionButton;
8  import android.support.v7.app.AlertDialog;
9  import android.support.v7.app.AppCompatActivity;
10 import android.view.MenuItem;
11 import android.view.View;
12 import android.widget.AdapterView;
13 import android.widget.AdapterView.OnItemClickListener;
14 import android.widget.ArrayAdapter;
15 import android.widget.ListView;
16 import java.util.ArrayList;
17 import java.util.Arrays;
18 import java.util.List;
19 import m2b.magic2brain.com.magic2brain.R;
20
21 /*
22 This class shows all cards of a deck in a list
23  */
24 public class DeckDisplayActivity extends AppCompatActivity {
25
26     protected void onCreate(Bundle savedInstanceState) {
27         overridePendingTransition(R.anim.slide_in_left , R.anim.
    slide_out_left); // This adds an fancy slide animation, when this
    activity starts.
28         super.onCreate(savedInstanceState); // This does some intern
    stuff. We don't need to worry about that. It's just needed.
29         setContentView(R.layout.activity_deck_display); // This adds
    an View to our Activity. We defined at "/res/layout/
    activity_deck_display.xml" how our activity should look like.
30         getSupportActionBar().setDisplayHomeAsUpEnabled(true); //
    With this line we add an "back"-Button to the Toolbar. If we press
     it it calls onOptionsItemSelected();
31
32         final Context currentContext = this; // This doesn't actually
```

```
 do anything, but it's needed if we want to refer to "this" from
an inner class.
33      Intent intent = getIntent(); // We want to access any data
that is passed.
34      final String deckcode = intent.getStringExtra("code"); // we
get the deck-code
35      final String name = intent.getStringExtra("name"); // we get
the deck-name
36
37      setTitle(name); // We set the title of the activity to the
name of the deck
38      Card c[] = DeckAssetLoader.getDeck(deckcode + ".json", this);
 // This gets all the cards from the deck
39      final Card[] cCopy = c; // and adds it to an Array
40
41      FloatingActionButton fam = (FloatingActionButton)
findViewById(R.id.fab_setlearn); // This gets the "Learn"-Button
42      fam.setOnClickListener(new View.OnClickListener() { // and
adds a listener to it
43          public void onClick(View view) {
44              // The following code starts the QueryActivity and
passes the deck
45              Intent intent = new Intent(currentContext,
QueryActivity.class);
46              Deck d = new Deck();
47              List<Card> clist = Arrays.asList(cCopy);
48              d.setCode(deckcode);
49              d.setName(name);
50              d.setSet(new ArrayList<Card>(clist));
51              intent.putExtra("Set", d);
52              startActivity(intent);
53          }
54      });
55
56      ListView lv = (ListView) findViewById(R.id.deckdisplay); //
We get the Listview
57
58      if (c[0] == null) { // If there are no cards in the deck, we
show an error message
59          AlertDialog.Builder dlgAlert = new AlertDialog.Builder(
this);
```

```
60              dlgAlert.setMessage("Sadly, this Deck was not Found");
61              dlgAlert.setTitle("Error");
62              dlgAlert.setPositiveButton("I understand, bill me your
    server costs",
63                      new DialogInterface.OnClickListener() {
64                          public void onClick(DialogInterface dialog,
    int which) {
65                              //dismiss the dialog
66                              onBackPressed();
67                          }
68                      });
69              dlgAlert.setCancelable(true);
70              dlgAlert.create().show();
71
72              lv.setVisibility(View.GONE);
73              fam.setVisibility(View.GONE);
74          } else { // If there are some cards in the deck, we generate
    a card-list like we did in various other Activities
75              final String[] listItems = RUtils.getListified(c);
76              final ArrayAdapter adapter = new ArrayAdapter(this,
    android.R.layout.simple_list_item_1, listItems);
77              lv.setAdapter(adapter);
78
79              final Card[] finalC = c;
80
81              lv.setOnItemClickListener(new OnItemClickListener() {
82                  public void onItemClick(AdapterView<?> parent, View
    view, int position, long id) {
83                      Intent intent = new Intent(currentContext,
    CardBrowserActivity.class);
84                      intent.putExtra("currentCard", finalC[position]);
85                      startActivity(intent);
86                  }
87              });
88          }
89      }
90
91      public boolean onOptionsItemSelected(MenuItem item) {
92          int id = item.getItemId();
93          switch (id) {
94              // Respond to the action bar's Up/Home button
```

```
95              case android.R.id.home:
96              onBackPressed();
97          }
98          return true;
99      }
100
101     public void onBackPressed() {
102         finish(); // This closes our Activity
103         overridePendingTransition(R.anim.slide_in_right , R.anim.
    slide_out_right); // We want to close it with an fancy animation.
104     }
105 }
```

7.6 CardBrowserActivity.java

```
1  package m2b.magic2brain.com;
2
3  import android.content.Context;
4  import android.content.Intent;
5  import android.graphics.Color;
6  import android.os.Bundle;
7  import android.support.design.widget.FloatingActionButton;
8  import android.support.design.widget.Snackbar;
9  import android.support.v7.app.AppCompatActivity;
10 import android.support.v7.widget.Toolbar;
11 import android.util.TypedValue;
12 import android.view.Gravity;
13 import android.view.View;
14 import android.widget.ImageView;
15 import android.widget.LinearLayout;
16 import android.widget.TextView;
17 import com.squareup.picasso.Picasso;
18 import java.util.ArrayList;
19 import java.util.Arrays;
20 import m2b.magic2brain.com.magic2brain.R;
21
22 /*
23 This activity should show one card with all informations
24 */
25 public class CardBrowserActivity extends AppCompatActivity {
26
27     ImageView cImage; // We will store our ImageView in a global
    variable, because we want to acces it from various methods.
28
29     protected void onCreate(final Bundle savedInstanceState) {
30         overridePendingTransition(R.anim.slide_in_left, R.anim.
    slide_out_left); // This adds an fancy slide animation, when this
    activity starts.
31         final Context context = this; // This doesn't actually do
    anything, but it's needed if we want to refer to "this" from an
    inner class.
32         super.onCreate(savedInstanceState); // This does some intern
    stuff. We don't need to worry about that. It's just needed.
33
```

```
34          Intent mIntent = getIntent(); // We want to access any data
    that is passed.
35          final Card card = (Card) mIntent.getSerializableExtra("
    currentCard"); // We load the card-object from the Intent
36
37          setContentView(R.layout.activity_card_browser);  // This adds
     an View to our Activity. We defined at "/res/layout/
    activity_card_browser.xml" how our activity should look like.
38          Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar); //
    With this we get the Toolbar of the View.
39          toolbar.setTitle(card.getName()); // We set a title to our
    View. The title should be the name of the card.
40          setSupportActionBar(toolbar); // We add the Toolbar as a
    SupportActionBar.
41
42          cImage = (ImageView) findViewById(R.id.cbaImage); // We load
    our ImageView from the View
43          cImage.setPadding(0, 10, 0, 10); // We add a padding to it.
44
45          // The next couple lines builds a GUI with the informations
    from the card. We access all Views from our View.
46          TextView tv = (TextView) findViewById(R.id.cbaInfo);
47          String info = "";
48          info += " Name: " + card.getName();
49          info += "\n MVID: " + card.getMultiverseid();
50          tv.setText(info);
51          tv.setTextColor(Color.BLACK);
52          tv.setPadding(0, 10, 0, 10);
53
54          TextView text = (TextView) findViewById(R.id.cbaText);
55          text.setText(card.getText());
56          text.setTextColor(Color.BLACK);
57          text.setPadding(20, 10, 10, 20);
58
59          TextView flavor = (TextView) findViewById(R.id.cbaFlavor);
60          flavor.setText(card.getFlavor());
61          flavor.setTextColor(Color.BLACK);
62          flavor.setPadding(20, 10, 10, 20);
63
64          TextView type = (TextView) findViewById(R.id.cbaType);
65          type.setText(card.getType());
```

```
66          type.setTextColor(Color.BLACK);
67          type.setPadding(0, 10, 0, 10);
68
69          LinearLayout ll = (LinearLayout) findViewById(R.id.
    cba_mcost_layout);
70          setManaCost(card.getManaCost(), ll);
71          ll.setGravity(Gravity.CENTER);
72          ll.setPadding(0, 10, 0, 10);
73
74          final int mvid = card.getMultiverseid(); // This gets the
    MultiverseID from the card
75          showPic(mvid); // and loads the picture of the card.
76
77          cImage.setOnClickListener(new View.OnClickListener() {  //
    This performs an action if we click the image
78              public void onClick(View view) {
79                  // The following code starts the
    CardImageDisplayActivity and passes the MultiverseID of the Card.
80                  Intent intent = new Intent(context,
    CardImageDisplayActivity.class);
81                  intent.putExtra("pic", mvid);
82                  startActivity(intent);
83              }
84          });
85
86          final FloatingActionButton fab = (FloatingActionButton)
    findViewById(R.id.fab); // This gets our "add to favorites"-button
     from the view
87          // We check if its already added to favorites. If so we mark
    it as already added.
88          if (!checkCard(card.getName())) {
89              fab.setImageResource(R.drawable.ic_favorite_border);
90          } else {
91              fab.setImageResource(R.drawable.ic_favorite);
92          }
93
94          fab.setOnClickListener(new View.OnClickListener() { // This
    performs an action if we click the "add to favorites"-Button
95              // The following code toggles the Button, so it shows
    that the card was added or not, gives a notification and add the
    card to the favorites-list (which will be saved).
```

```
96              public void onClick(View view) {
97                  if (checkCard(card.getName())) {
98                      removeCard(card.getName());
99                      fab.setImageResource(R.drawable.
    ic_favorite_border);
100                     Snackbar.make(view, R.string.remove_fav, Snackbar
    .LENGTH_LONG)
101                             .setAction("Action", null).show();
102                 } else {
103                     Favorites.favorites_mvid.add(card);
104                     fab.setImageResource(R.drawable.ic_favorite);
105                     Snackbar.make(view, R.string.add_fav, Snackbar.
    LENGTH_LONG)
106                             .setAction("Action", null).show();
107                 }
108             }
109         });
110     }
111
112     private boolean checkCard(String name) { // This function checks
    if the card was added to the favorites
113         for (Card c : Favorites.favorites_mvid) {
114             if (c.getName().contains(name)) {
115                 return true;
116             }
117         }
118         return false;
119     }
120
121     private void removeCard(String name) { // This function removes
    the card from the favorites-list
122         ArrayList<Card> cards = new ArrayList<>();
123         for (Card c : Favorites.favorites_mvid) {
124             if (c.getName().contains(name)) {
125                 cards.add(c);
126             }
127         }
128         for (Card c : cards) {
129             Favorites.favorites_mvid.remove(c);
130         }
131     }
```

```
132
133     private void setManaCost(String manatext, LinearLayout layout) {
    // This function adds the mana-cost to the View by loading the
    images.
134         manatext = manatext.replaceAll("\\{", "");
135         manatext = manatext.replaceAll("\\}", "");
136         String[] items = manatext.split("");
137         items = Arrays.copyOfRange(items, 1, items.length);
138 
139         for (int i = 0; i < items.length; i++) {
140             if (RUtils.isInteger(items[i])) {
141                 TextView tv = new TextView(this);
142                 tv.setText(items[i]);
143                 tv.setTextSize(TypedValue.COMPLEX_UNIT_PX, 80);
144                 tv.setTextColor(Color.BLACK);
145                 layout.addView(tv);
146             } else {
147                 ImageView imgv = new ImageView(this);
148                 int resid = 0;
149                 if (items[i].equals("B")) {
150                     resid = R.drawable.b;
151                 } else if (items[i].equals("C")) {
152                     resid = R.drawable.c;
153                 } else if (items[i].equals("G")) {
154                     resid = R.drawable.g;
155                 } else if (items[i].equals("R")) {
156                     resid = R.drawable.r;
157                 } else if (items[i].equals("U")) {
158                     resid = R.drawable.u;
159                 } else if (items[i].equals("W")) {
160                     resid = R.drawable.w;
161                 } else {
162                     resid = R.drawable.ic_action_cancel;
163                 }
164                 imgv.setBackgroundResource(resid);
165                 layout.addView(imgv);
166                 android.view.ViewGroup.LayoutParams layoutParams =
    imgv.getLayoutParams();
167                 layoutParams.width = 60;
168                 layoutParams.height = 60;
169                 imgv.setLayoutParams(layoutParams);
```

```
170             }
171         }
172     }
173
174     private void showPic(int MultiID) { // This method loads an
    cardimage into cImage with a given MultiverseID.
175         Picasso.with(this)
176                 .load(getString(R.string.image_link_1) + MultiID +
    getString(R.string.image_link_2)) // This tries to load an image
    from a link.
177                 .placeholder(R.drawable.loading_image) // We want to
    show a image while its loading. We load our image from the "/res/
    drawable" folder
178                 .error(R.drawable.image_not_found) // If it fails to
    load image we show an error-image.
179                 .into(cImage); // This places the image into our
    ImageView.
180     }
181
182     public void onBackPressed() {
183         finish(); // This closes our Activity
184         overridePendingTransition(R.anim.slide_in_right, R.anim.
    slide_out_right); // We want to close it with an fancy animation.
185     }
186 }
```

7.7 CardImageDisplayActivity.java

```
1  package m2b.magic2brain.com;
2
3  import android.content.Intent;
4  import android.os.Bundle;
5  import android.os.Handler;
6  import android.support.v7.app.ActionBar;
7  import android.support.v7.app.AppCompatActivity;
8  import android.view.View;
9  import android.widget.ImageView;
10 import com.squareup.picasso.Picasso;
11 import m2b.magic2brain.com.magic2brain.R;
12
13 /*
14 This Activity should show a big image of a card
15  */
16
17 public class CardImageDisplayActivity extends AppCompatActivity {
18     private final Handler mHideHandler = new Handler();
19     private View mContentView;
20     private final Runnable mHidePart2Runnable = new Runnable() {
21         public void run() {
22             // Delayed removal of status and navigation bar
23             // Note that some of these constants are new as of API 16
    (Jelly Bean)
24             // and API 19 (KitKat). It is safe to use them, as they
   are inlined
25             // at compile-time and do nothing on earlier devices.
26             mContentView.setSystemUiVisibility(View.
   SYSTEM_UI_FLAG_LOW_PROFILE
27                     | View.SYSTEM_UI_FLAG_FULLSCREEN
28                     | View.SYSTEM_UI_FLAG_LAYOUT_STABLE
29                     | View.SYSTEM_UI_FLAG_IMMERSIVE_STICKY
30                     | View.SYSTEM_UI_FLAG_LAYOUT_HIDE_NAVIGATION
31                     | View.SYSTEM_UI_FLAG_HIDE_NAVIGATION);
32         }
33     };
34     private View mControlsView;
35     private final Runnable mShowPart2Runnable = new Runnable() {
36         @Override
```

```
37          public void run() {
38              // Delayed display of UI elements
39              ActionBar actionBar = getSupportActionBar();
40              if (actionBar != null) {
41                  actionBar.show();
42              }
43              mControlsView.setVisibility(View.VISIBLE);
44          }
45      };
46
47      protected void onCreate(Bundle savedInstanceState) {
48          super.onCreate(savedInstanceState);
49          setContentView(R.layout.activity_card_image_display);
50
51          mControlsView = findViewById(R.id.fullscreen_content_controls
    );
52          mContentView = findViewById(R.id.fullscreen_content);
53          Intent mIntent = getIntent();
54
55          // Set up the user interaction to manually show or hide the
    system UI.
56          mContentView.setOnClickListener(new View.OnClickListener() {
57              public void onClick(View view) {
58                  finish();
59              }
60          });
61
62          // Upon interacting with UI controls, delay any scheduled
    hide()
63          // operations to prevent the jarring behavior of controls
    going away
64          // while interacting with the UI.
65          hide();
66
67          int mvid = mIntent.getIntExtra("pic", 0);
68          showPic(mvid);
69      }
70
71      private void showPic(int MultiID) {
72          ImageView cImage = (ImageView) findViewById(R.id.cida_imgview
    );
```

```
73          Picasso.with(this)
74                  .load(getString(R.string.image_link_1) + MultiID +
    getString(R.string.image_link_2))
75                  .placeholder(R.drawable.loading_image)
76                  .error(R.drawable.image_not_found)
77                  .into(cImage);
78      }
79
80      private void hide() {
81          // Hide UI first
82          ActionBar actionBar = getSupportActionBar();
83          if (actionBar != null) {
84              actionBar.hide();
85          }
86          mControlsView.setVisibility(View.GONE);
87          // Schedule a runnable to remove the status and navigation
    bar after a delay
88          mHideHandler.removeCallbacks(mShowPart2Runnable);
89          mHidePart2Runnable.run();
90      }
91  }
```

7.8 QueryActivity.java

```
1  package m2b.magic2brain.com;
2
3  import android.content.Intent;
4  import android.content.SharedPreferences;
5  import android.graphics.Color;
6  import android.graphics.PorterDuff;
7  import android.os.Bundle;
8  import android.os.Handler;
9  import android.preference.PreferenceManager;
10 import android.support.v7.app.AppCompatActivity;
11 import android.support.v7.widget.Toolbar;
12 import android.view.Gravity;
13 import android.view.KeyEvent;
14 import android.view.Menu;
15 import android.view.MenuItem;
16 import android.view.View;
17 import android.view.WindowManager;
18 import android.view.animation.AccelerateInterpolator;
19 import android.view.animation.Animation;
20 import android.view.animation.AnimationUtils;
21 import android.view.animation.DecelerateInterpolator;
22 import android.widget.Button;
23 import android.widget.EditText;
24 import android.widget.ImageView;
25 import android.widget.RelativeLayout;
26 import android.widget.TextView;
27 import com.google.gson.Gson;
28 import com.google.gson.reflect.TypeToken;
29 import com.squareup.picasso.Picasso;
30 import java.lang.reflect.Type;
31 import java.util.ArrayList;
32 import java.util.Collections;
33 import java.util.Random;
34 import m2b.magic2brain.com.magic2brain.R;
35
36 public class QueryActivity extends AppCompatActivity {
37     private ImageView imgv; // The image of the card gets stored here
38     private ImageView imgCorr; // Is over the image. It's to indicate
      if the answer was correct or not
```

```
39      private Toolbar hiding; // This is a bar that hides a certain
    area
40      private ArrayList<Card> set; //This Arraylist holds all Cards
    that need to query. It won't be edited (after loading it)
41      private ArrayList<Card> wrongGuessed;
42      private int indexCard; // Actually not needed (because we remove
    cards from WrongGuessed) but may be useful in later edits
43      private boolean firstGuess; //This is to check if he guessed it
    at first try. If so we remove the card from the Arraylist. Else it
     stays there.
44      private String deckName; //Name of the deck. Only for saving/
    loading purpose
45      private TextView score; //this will show the user the current
    progress
46      private boolean queryLand = true; // should we really query lands
    ?
47      private boolean skipped = false; // this is to store if the user
    has skipped or not
48      private ArrayList<String> recentlyLearned; // this is to save the
     deckname if a new set is learned
49      private ArrayList<String> recentlyLearnedNames; // the name of
    the sets
50      private int Mode; // the query mode
51      private ArrayList<Button> choices; // We store the buttons here
52
53      protected void onCreate(Bundle savedInstanceState) {
54          // Standard stuff
55          overridePendingTransition(R.anim.slide_in_left , R.anim.
    slide_out_left); // This adds an fancy slide animation, when this
    activity starts.
56          super.onCreate(savedInstanceState); // This does some intern
    stuff. We don't need to worry about that. It's just needed.
57          setContentView(R.layout.activity_query); // This adds an View
     to our Activity. We defined at "/res/layout/activity_query.xml"
    how our activity should look like.
58          // Hide the status bar.
59          this.getWindow().setFlags(WindowManager.LayoutParams.
    FLAG_FULLSCREEN, WindowManager.LayoutParams.FLAG_FULLSCREEN);
60          // Get some informations
61          Intent i = getIntent();
62          Deck qur = (Deck) i.getSerializableExtra("Set");
```

```
63          deckName = qur.getName(); // Get the name of the Deck
64          String code = qur.getCode(); // Get the Deck-Code
65          if (deckName == null) {
66              deckName = "DEFAULT"; // If deckname isn't defined, we
    set it to "DEFAULT"
67          }
68          setTitle(deckName); // we set the title to the deckname
69          //Add set to recent learned
70          if (!loadRecent()) {
71              recentlyLearned = new ArrayList<>();
72              recentlyLearnedNames = new ArrayList<>();
73          }
74          if (!recentlyLearned.contains(code)) {
75              recentlyLearned.add(code);
76              recentlyLearnedNames.add(deckName);
77          }
78          if (recentlyLearned.size() == 10) {
79              recentlyLearned.remove(0);
80              recentlyLearnedNames.remove(0);
81          }
82          saveRecent();
83          //load set
84          set = qur.getSet();
85          if (!loadProgress()) { //First we try to load the progress.
    If this fails, we simply start over
86              wrongGuessed = (ArrayList) set.clone(); //Lets assume he
    guessed everything wrong and remove the card of this Array when he
     guesses it right
87              shuffleWrongs(); //Shuffle it a bit (better learn-effect)
88              indexCard = 0; // We start at card No. 1
89          }
90          //Set Mode
91          Mode = 1;
92          if (set.size() < 4) {
93              Mode = 0;
94          }
95          // Build UI
96          hiding = (Toolbar) findViewById(R.id.toolbar_query);
97          getSupportActionBar().setDisplayHomeAsUpEnabled(true); //
    With this line we add an "back"-Button to the Toolbar. If we press
     it it calls onOptionsItemSelected();
```

```
98          buildMenu();
99          //Start the query
100         showFirstPic();
101         if (deckName.contains("DEFAULT") || deckName.contains("
    Favorites")) {
102             restartAll(); // Because this decks are dynamic
103         }
104     }
105
106     protected void onPause() { // We save the progress when the user
    leaves
107         super.onPause();
108         saveProgress();
109     }
110
111     public boolean loadRecent() { // This loads all recently learned
    sets
112         SharedPreferences sharedPrefs = PreferenceManager.
    getDefaultSharedPreferences(this);
113         Gson gson = new Gson();
114         String json = sharedPrefs.getString("query_recent4", null);
115         String json2 = sharedPrefs.getString("query_recent_names",
    null);
116         Type type = new TypeToken<ArrayList<String>>() {
117         }.getType();
118         ArrayList<String> aL = gson.fromJson(json, type);
119         ArrayList<String> aL2 = gson.fromJson(json2, type);
120         if (aL == null) {
121             return false;
122         }
123         recentlyLearned = aL;
124         recentlyLearnedNames = aL2;
125         return true;
126     }
127
128     public void saveRecent() { // This saves all recently learned
    sets
129         SharedPreferences sharedPrefs = PreferenceManager.
    getDefaultSharedPreferences(this);
130         SharedPreferences.Editor editor = sharedPrefs.edit();
131         Gson gson = new Gson();
```

```
132         String json = gson.toJson(recentlyLearned);
133         String json2 = gson.toJson(recentlyLearnedNames);
134         editor.putString("query_recent4", json);
135         editor.putString("query_recent_names", json2);
136         editor.commit();
137     }
138 
139     public boolean loadProgress() { // This loads the progress of the
        current deck
140         SharedPreferences sharedPrefs = PreferenceManager.
    getDefaultSharedPreferences(this);
141         Gson gson = new Gson();
142         String json = sharedPrefs.getString("query_list_" + deckName,
         null);
143         Type type = new TypeToken<ArrayList<Card>>() {
144         }.getType();
145         ArrayList<Card> aL = gson.fromJson(json, type);
146         int loadedIndex = sharedPrefs.getInt("query_index_" +
    deckName, -1);
147         if (aL == null) {
148             return false;
149         }
150         wrongGuessed = aL;
151         indexCard = loadedIndex;
152         return true;
153     }
154 
155     public void saveProgress() { // This saves the progress of the
    current deck
156         SharedPreferences sharedPrefs = PreferenceManager.
    getDefaultSharedPreferences(this);
157         SharedPreferences.Editor editor = sharedPrefs.edit();
158         Gson gson = new Gson();
159         String json = gson.toJson(wrongGuessed);
160         editor.putString("query_list_" + deckName, json);
161         editor.putInt("query_index_" + deckName, indexCard);
162         editor.commit();
163     }
164 
165     public boolean onCreateOptionsMenu(Menu menu) {
166         // Inflate the menu; this adds items to the action bar if it
```

```
    is present.
167         getMenuInflater().inflate(R.menu.query, menu);
168         return true;
169     }
170 
171     public void shuffleWrongs() { // This shuffles the wrongGuessed-
    Arraylist
172         Collections.shuffle(wrongGuessed, new Random(System.nanoTime
    ()));
173     }
174 
175     public void showPic(int MultiID) { // This method loads an
    cardimage into imgv with a given MultiverseID.
176         Picasso.with(this)
177                 .load(getString(R.string.image_link_1) + MultiID +
    getString(R.string.image_link_2)) // This tries to load an image
    from a link.
178                 .placeholder(R.drawable.loading_image)  // We want to
     show a image while its loading. We load our image from the "/res/
    drawable" folder
179                 .error(R.drawable.image_not_found) // If it fails to
    load image we show an error-image.
180                 .into(imgv); // This places the image into our
    ImageView.
181     }
182 
183     public void showFirstPic() { // We need to call this, if we start
     with the first item
184         updateScore(); // This will update the shown score
185         showHiderInstant(); // This will instantly hide a part of the
     card
186         firstGuess = true; // The user didn't guess yet, so it's his
    first guess
187         showPic(wrongGuessed.get(indexCard).getMultiverseid()); // We
     show the current picture
188         hiding.bringToFront(); // We want the hider in the front
189         imgCorr.bringToFront(); // And imgCorr aswell, so the users
    sees these
190         if (Mode == 1) {
191             updateChoices(); // If the users uses mode one, we need
    to update the text of the buttons
```

```
192         }
193     }
194 
195     public void showNextPic() {
196         updateScore(); // This will update the shown score
197         showHider(); // We fade the hider in
198         final Handler handler = new Handler();
199         handler.postDelayed(new Runnable() {
200             public void run() { // We add a little delay of 800
    milliseconds, because of the animation
201                 firstGuess = true; // The user didn't guess yet, so
    it's his first guess
202                 showPic(wrongGuessed.get(indexCard).getMultiverseid()
    ); // We show the current picture
203                 hiding.bringToFront(); // We want the hider in the
    front
204                 imgCorr.bringToFront(); // And imgCorr aswell, so the
     users sees these
205                 skipped = false; // The user didn't skip the current
    card
206             }
207         }, 800);
208         if (Mode == 1) {
209             updateChoices(); // If the users uses mode one, we need
    to update the text of the buttons
210         }
211     }
212 
213     public void checkAnswer(String txt) { // This method checks the
    answer
214         if (txt.replaceAll("\\s+", "").equalsIgnoreCase(wrongGuessed.
    get(indexCard).getName().replaceAll("\\s+", ""))) {
215             if (firstGuess) {
216                 wrongGuessed.remove(indexCard);//if he guessed it, we
     remove it.
217             }
218             else {
219                 indexCard++; // else we continue with the next card
220             }
221             if (indexCard == wrongGuessed.size()) { //If this true he
    's through the set
```

```
222                 final Handler handler = new Handler();
223                 handler.postDelayed(new Runnable() { // Add a delay
    of 1 second
224                     public void run() {
225                         setDone();
226                     }
227                 }, 1000);
228             } else {
229                 if (!skipped) {
230                     imgCorr.setImageResource(R.drawable.
    correct_answer); // If the user didn't skip we show the "correct"-
    symbol
231                     showImgCorr();
232                 }
233                 final Handler handler = new Handler();
234                 handler.postDelayed(new Runnable() { // Add a delay
    of 800 milliseconds
235                     public void run() {
236                         if (!skipped) {
237                             hideImgCorr(); // We hide the symbol
    again
238                         }
239                         showNextPic(); // And show the next card
240                     }
241                 }, 800);
242             }
243         } else {
244             wrongAnswer(); // If the user didn't guess correctly, we
    call wrongAnswer()
245         }
246     }
247
248     public void skip() { // Skips the card with an animation
249         if (!skipped) {
250             skipped = true; // The user skipped we set skipped to
    true
251             wrongAnswer(); // The user didn't guess correctly
252             final Handler handler = new Handler();
253             handler.postDelayed(new Runnable() {
254                 public void run() {
255                     checkAnswer(wrongGuessed.get(indexCard).getName()
```

```
    ); // after a delay of 600 milliseconds we simulate a correct
    answer.
256                 }
257             }, 600);
258         }
259     }
260
261     public void wrongAnswer() {
262         firstGuess = false; // The user didn't guess it on the first
    time
263         imgCorr.setImageResource(R.drawable.wrong_answer); // We want
     to show the "Wrong"-image
264         showImgCorr(); // We show the image
265         hideHider(); // We show the correct answer
266         final Handler handler = new Handler();
267         handler.postDelayed(new Runnable() {
268             public void run() {
269                 hideImgCorr(); // We hide the image after an delay of
     1 second
270             }
271         }, 1000);
272     }
273
274     public void setDone() { // If the set is done, we show the
    endscreen and reshuffle the wrongGuessed-arraylist
275         buildEndScreen();
276         shuffleWrongs();
277     }
278
279     public void hideHider() { // Shows the name of the card fadingly
280         hiding.animate().alpha(0).setDuration(600).setInterpolator(
    new DecelerateInterpolator()).withEndAction(new Runnable() {
281             public void run() {
282                 hiding.animate().alpha(0).setDuration(1000).
    setInterpolator(new AccelerateInterpolator()).start();
283             }
284         }).start();
285     }
286
287     public void showHider() { // Hides the name of the card fadingly
288         hiding.animate().alpha(1).setDuration(600).setInterpolator(
```

```
    new DecelerateInterpolator()).withEndAction(new Runnable() {
289             public void run() {
290                 hiding.animate().alpha(1).setDuration(100).
    setInterpolator(new AccelerateInterpolator()).start();
291             }
292         }).start();
293     }
294 
295     public void showHiderInstant() { // Hides the name of the card
    instantly
296         hiding.animate().alpha(1).setDuration(10).setInterpolator(new
     DecelerateInterpolator()).withEndAction(new Runnable() {
297             public void run() {
298                 hiding.animate().alpha(1).setDuration(100).
    setInterpolator(new AccelerateInterpolator()).start();
299             }
300         }).start();
301     }
302 
303     public void hideImgCorr() {
304         Animation myFadeInAnimation = AnimationUtils.loadAnimation(
    this, R.anim.fadeout);
305         imgCorr.startAnimation(myFadeInAnimation); //Set animation to
     your ImageView
306     }
307 
308     public void showImgCorr() {
309         Animation myFadeInAnimation = AnimationUtils.loadAnimation(
    this, R.anim.fadein);
310         imgCorr.startAnimation(myFadeInAnimation); //Set animation to
     your ImageView
311     }
312 
313     public void buildMenu() { // This method simply build the UI
314         int scrWidth = getWindowManager().getDefaultDisplay().
    getWidth(); // Get the width of the screen
315         int scrHeight = getWindowManager().getDefaultDisplay().
    getHeight(); // Get the height of the screen
316         RelativeLayout lyt = (RelativeLayout) findViewById(R.id.
    query_absolute); // Get the View of the XML
317         RelativeLayout.LayoutParams params; // The parameters we want
```

```
    to add our views
318         lyt.removeAllViews(); //Clear the Board
319
320         imgv = new ImageView(this); // Create new Imageview
321         params = new RelativeLayout.LayoutParams(scrWidth /*Width*/,
    (int) (0.5 * scrHeight))/*Height*/;
322         params.leftMargin = 0; // X-Position
323         params.topMargin = (int) (0.02 * scrHeight); // Y-Position
324         lyt.addView(imgv, params); // add it to the View
325
326         imgCorr = new ImageView(this); // Create new Imageview
327         hideImgCorr(); // Hide this ImageView
328         params = new RelativeLayout.LayoutParams(scrWidth /*Width*/,
    (int) (0.5 * scrHeight))/*Height*/;
329         params.leftMargin = 0; // X-Position
330         params.topMargin = (int) (0.02 * scrHeight); // Y-Position
331         lyt.addView(imgCorr, params); // add it to the View
332
333         switch (Mode) {
334             case 0:
335                 buildMode0(); // If we have Mode 0, we continue
    building with Mode 0
336                 break;
337             case 1:
338                 buildMode1(); // else we build Mode 1
339         }
340     }
341
342     public void buildMode0() { // This method builds the UI for Mode
    0
343         int scrWidth = getWindowManager().getDefaultDisplay().
    getWidth(); // Get the width of the screen
344         int scrHeight = getWindowManager().getDefaultDisplay().
    getHeight();  // Get the height of the screen
345         RelativeLayout lyt = (RelativeLayout) findViewById(R.id.
    query_absolute); // Get the View of the XML
346         RelativeLayout.LayoutParams params; // The parameters we want
     to add our views
347
348         params = new RelativeLayout.LayoutParams((int) (0.55 *
    scrWidth) /*Width*/, (int) (0.03 * scrHeight)/*Height*/);
```

```
349         params.leftMargin = (scrWidth / 2 - (int) (0.55 * scrWidth) /
    2); // X-Position
350         params.topMargin = (int) (0.045 * scrHeight); // Y-Position
351         lyt.addView(hiding, params); // We add the hider
352 
353         // Add EditText like Imageview
354         final EditText inputtxt = new EditText(this);
355         inputtxt.setGravity(Gravity.CENTER);
356         inputtxt.setHint(R.string.query_mode0_hint);
357         params = new RelativeLayout.LayoutParams((int) (0.75 *
    scrWidth) /*Width*/, (int) (0.1 * scrHeight)/*Height*/);
358         params.leftMargin = (int) (0.125 * scrWidth);
359         params.topMargin = (int) (0.55 * scrHeight);
360         lyt.addView(inputtxt, params);
361         // Add a Listener to it (so the User can simply press ENTER
    on the keyboard)
362         inputtxt.setOnKeyListener(new View.OnKeyListener() {
363             public boolean onKey(View v, int keyCode, KeyEvent event)
    {
364                 if (event.getAction() == KeyEvent.ACTION_DOWN) {
365                     switch (keyCode) {
366                         case KeyEvent.KEYCODE_DPAD_CENTER:
367                         case KeyEvent.KEYCODE_ENTER:
368                             checkAnswer(inputtxt.getText().toString()
    );
369                             inputtxt.setText("");
370                             return true;
371                         default:
372                             break;
373                     }
374                 }
375                 return false;
376             }
377         });
378 
379         // add answer button
380         Button answer = new Button(this);
381         answer.setText("Answer");
382         answer.setTextColor(Color.WHITE);
383         answer.getBackground().setColorFilter(getResources().getColor
    (R.color.colorAccent), PorterDuff.Mode.MULTIPLY);
```

```
384         params = new RelativeLayout.LayoutParams((int) (0.25 *
    scrWidth) /*Width*/, (int) (0.1 * scrHeight)/*Height*/);
385         params.leftMargin = (scrWidth / 2) - (int) (0.25 * scrWidth);
386         params.topMargin = (int) (0.50 * scrHeight) + (int) (0.15 *
    scrHeight);
387         lyt.addView(answer, params);
388         answer.setOnClickListener(new View.OnClickListener() {
389             public void onClick(View view) {
390                 checkAnswer(inputtxt.getText().toString());
391                 inputtxt.setText("");
392             }
393         });
394 
395         // Add skip button
396         Button skip = new Button(this);
397         skip.setText("Skip");
398         skip.setTextColor(Color.WHITE);
399         skip.getBackground().setColorFilter(getResources().getColor(R
    .color.colorPrimary), PorterDuff.Mode.MULTIPLY);
400         params = new RelativeLayout.LayoutParams((int) (0.25 *
    scrWidth) /*Width*/, (int) (0.1 * scrHeight)/*Height*/);
401         params.leftMargin = (scrWidth / 2);
402         params.topMargin = (int) (0.50 * scrHeight) + (int) (0.15 *
    scrHeight);
403         lyt.addView(skip, params);
404         skip.setOnClickListener(new View.OnClickListener() {
405             public void onClick(View view) {
406                 skip();
407             }
408         });
409 
410         // show the score
411         score = new TextView(this); // Create new Textview
412         score.setGravity(Gravity.CENTER);
413         score.setTextSize(36);
414         params = new RelativeLayout.LayoutParams(scrWidth /*Width*/,
    (int) (0.1 * scrHeight)/*Height*/);
415         params.leftMargin = 0; // X-Position
416         params.topMargin = (int) (0.78 * scrHeight); // Y-Position
417         lyt.addView(score, params); // add it to the View
418     }
```

```
419
420     public void buildMode1() { // This method builds the UI for Mode
        1
421         int scrWidth = getWindowManager().getDefaultDisplay().
        getWidth(); // Get the width of the screen
422         int scrHeight = getWindowManager().getDefaultDisplay().
        getHeight(); // Get the height of the screen
423         RelativeLayout lyt = (RelativeLayout) findViewById(R.id.
        query_absolute); // Get the View of the XML
424         RelativeLayout.LayoutParams params; // The parameters we want
         to add our views
425
426         params = new RelativeLayout.LayoutParams((int) (0.55 *
        scrWidth)/*Width*/, (int) (0.16 * scrHeight)/*Height*/);
427         params.leftMargin = (scrWidth / 2 - (int) (0.55 * scrWidth) /
         2); // X-Position
428         params.topMargin = (int) (0.32 * scrHeight); // Y-Position
429         lyt.addView(hiding, params); // Add the hider
430
431         // Add all four buttons
432         Button choice0 = new Button(this);
433         choice0.setText("choice0");
434         choice0.setTextSize(8);
435         choice0.setTextColor(Color.WHITE);
436         choice0.getBackground().setColorFilter(getResources().
        getColor(R.color.colorAccent), PorterDuff.Mode.MULTIPLY);
437         params = new RelativeLayout.LayoutParams((int) (0.48 *
        scrWidth)/*Width*/, (int) (0.19 * scrHeight)/*Height*/);
438         params.leftMargin = (int) (0.02 * scrWidth);
439         params.topMargin = (int) (0.53 * scrHeight);
440         lyt.addView(choice0, params);
441         choice0.setOnClickListener(new View.OnClickListener() {
442             public void onClick(View view) {
443                 onClickChoice(0);
444             }
445         });
446
447         Button choice1 = new Button(this);
448         choice1.setText("choice1");
449         choice1.setTextSize(8);
450         choice1.setTextColor(Color.WHITE);
```

```
451         choice1.getBackground().setColorFilter(getResources().
    getColor(R.color.colorPrimary), PorterDuff.Mode.MULTIPLY);
452         params = new RelativeLayout.LayoutParams((int) (0.48 *
    scrWidth)/*Width*/, (int) (0.19 * scrHeight)/*Height*/);
453         params.leftMargin = (int) (0.50 * scrWidth);
454         params.topMargin = (int) (0.53 * scrHeight);
455         lyt.addView(choice1, params);
456         choice1.setOnClickListener(new View.OnClickListener() {
457             public void onClick(View view) {
458                 onClickChoice(1);
459             }
460         });
461
462         Button choice2 = new Button(this);
463         choice2.setText("choice2");
464         choice2.setTextSize(8);
465         choice2.setTextColor(Color.WHITE);
466         choice2.getBackground().setColorFilter(getResources().
    getColor(R.color.colorPrimary), PorterDuff.Mode.MULTIPLY);
467         params = new RelativeLayout.LayoutParams((int) (0.48 *
    scrWidth)/*Width*/, (int) (0.19 * scrHeight)/*Height*/);
468         params.leftMargin = (int) (0.02 * scrWidth);
469         params.topMargin = (int) (0.71 * scrHeight);
470         lyt.addView(choice2, params);
471         choice2.setOnClickListener(new View.OnClickListener() {
472             public void onClick(View view) {
473                 onClickChoice(2);
474             }
475         });
476
477         Button choice3 = new Button(this);
478         choice3.setText("choice3");
479         choice3.setTextSize(8);
480         choice3.setTextColor(Color.WHITE);
481         choice3.getBackground().setColorFilter(getResources().
    getColor(R.color.colorAccent), PorterDuff.Mode.MULTIPLY);
482         params = new RelativeLayout.LayoutParams((int) (0.48 *
    scrWidth)/*Width*/, (int) (0.19 * scrHeight)/*Height*/);
483         params.leftMargin = (int) (0.50 * scrWidth);
484         params.topMargin = (int) (0.71 * scrHeight);
485         lyt.addView(choice3, params);
```

```
486         choice3.setOnClickListener(new View.OnClickListener() {
487             public void onClick(View view) {
488                 onClickChoice(3);
489             }
490         });
491 
492         // Add the score
493         score = new TextView(this); // Create new Textview
494         score.setGravity(Gravity.CENTER);
495         score.setTextSize(0);
496         params = new RelativeLayout.LayoutParams(scrWidth /*Width*/,
    (int) (0.1 * scrHeight))/*Height*/;
497         params.leftMargin = 0; // X-Position
498         params.topMargin = (int) (0.78 * scrHeight); // Y-Position
499         lyt.addView(score, params); // add it to the View
500 
501         choices = new ArrayList<>(); // Fill the arraylist with the
    buttons
502         choices.add(choice0);
503         choices.add(choice1);
504         choices.add(choice2);
505         choices.add(choice3);
506     }
507 
508     public void onClickChoice(int nr) {
509         if (choices.get(nr).getText() == wrongGuessed.get(indexCard).
    getText()) {
510             if (firstGuess) {
511                 wrongGuessed.remove(indexCard);//if he guessed it, we
     remove it.
512             }
513             else {
514                 indexCard++; // else we continue with the next card
515             }
516             if (indexCard == wrongGuessed.size()) { //If this true he
    's through the set
517                 final Handler handler = new Handler();
518                 handler.postDelayed(new Runnable() { // 1 second
    delay
519                     public void run() {
520                         setDone(); // finish query
```

```
521                     }
522                 }, 1000);
523             } else {
524                 if (!skipped) {
525                     imgCorr.setImageResource(R.drawable.
    correct_answer); // set "correct"-image if the user guessed it
    right
526                     showImgCorr(); // show image
527                 }
528                 final Handler handler = new Handler();
529                 handler.postDelayed(new Runnable() { // 800
    milliseconds delay
530                     public void run() {
531                         if (!skipped) {
532                             hideImgCorr(); // hide image
533                         }
534                         showNextPic(); // continue query
535                     }
536                 }, 800);
537             }
538         } else {
539             wrongAnswer(); // If the user guessed it wrong, we call
    wrongAnswer()
540         }
541     }
542
543     public void updateChoices() { // This sets the text of one button
     to the correct answer and the other ones to texts of random cards
544         int rightOne = (int) (Math.random() * 4);
545         for (int i = 0; i < choices.size(); i++) {
546             if (i == rightOne) {
547                 choices.get(i).setText(wrongGuessed.get(indexCard).
    getText());
548             } else {
549                 choices.get(i).setText(set.get((int) (Math.random() *
     set.size())).getText());
550             }
551         }
552
553         for (int i = 0; i < choices.size() - 1; i++) {
554             if (choices.get(i).getText() == choices.get(i + 1).
```

```
    getText()) {
555                 choices.get(i + 1).setText(set.get((int) (Math.random
    () * set.size())).getText());
556             }
557         }
558     }
559
560     public void updateScore() { // This updates the score text
561         score.setText((set.size() - wrongGuessed.size()) + " / " +
    indexCard + " / " + (wrongGuessed.size() - indexCard));
562     }
563
564     public void buildEndScreen() { // This builds the end UI with the
     score and two buttons to restart.
565         int scrWidth = getWindowManager().getDefaultDisplay().
    getWidth(); // Get screen-width
566         int scrHeight = getWindowManager().getDefaultDisplay().
    getHeight(); // Get screen-height
567         RelativeLayout lyt = (RelativeLayout) findViewById(R.id.
    query_absolute); // Get the View of the XML
568         lyt.removeAllViews(); //Clear the board
569         RelativeLayout.LayoutParams params; // the parameters we need
     for our Views
570
571         // Show the amount of right guessed cards
572         TextView rights = new TextView(this);
573         rights.setText("Right: ");
574         rights.setTextSize(36);
575         rights.setTextColor(getResources().getColor(R.color.
    colorPrimary));
576         params = new RelativeLayout.LayoutParams(scrWidth /*Width*/,
    (int) (0.1 * scrHeight))/*Height*/;
577         params.leftMargin = (int) (0.1 * scrHeight); // X-Position
578         params.topMargin = (int) (0.05 * scrHeight); // Y-Position
579         lyt.addView(rights, params); // add it to the View
580
581         TextView rights2 = new TextView(this);
582         rights2.setText("" + (set.size() - wrongGuessed.size()));
583         rights2.setTextSize(36);
584         rights2.setTextColor(getResources().getColor(R.color.
    colorPrimary));
```

```
585         params = new RelativeLayout.LayoutParams(scrWidth /*Width*/,
    (int) (0.1 * scrHeight))/*Height*/;
586         params.leftMargin = (int) (0.4 * scrHeight); // X-Position
587         params.topMargin = (int) (0.05 * scrHeight); // Y-Position
588         lyt.addView(rights2, params); // add it to the View
589 
590         // Show the amount of wrong guessed cards
591         TextView wrongs = new TextView(this);
592         wrongs.setText("Wrong: ");
593         wrongs.setTextSize(36);
594         wrongs.setTextColor(getResources().getColor(R.color.
    colorAccent));
595         params = new RelativeLayout.LayoutParams(scrWidth /*Width*/,
    (int) (0.1 * scrHeight))/*Height*/;
596         params.leftMargin = (int) (0.1 * scrHeight); // X-Position
597         params.topMargin = (int) (0.05 * scrHeight) + (int) (0.1 *
    scrHeight); // Y-Position
598         lyt.addView(wrongs, params); // add it to the View
599 
600         TextView wrongs2 = new TextView(this);
601         wrongs2.setText("" + (wrongGuessed.size()));
602         wrongs2.setTextSize(36);
603         wrongs2.setTextColor(getResources().getColor(R.color.
    colorAccent));
604         params = new RelativeLayout.LayoutParams(scrWidth /*Width*/,
    (int) (0.1 * scrHeight))/*Height*/;
605         params.leftMargin = (int) (0.4 * scrHeight); // X-Position
606         params.topMargin = (int) (0.05 * scrHeight) + (int) (0.1 *
    scrHeight); // Y-Position
607         lyt.addView(wrongs2, params); // add it to the View
608 
609         // Show the amount of total cards
610         TextView total = new TextView(this);
611         total.setText("Total: ");
612         total.setTextSize(36);
613         total.setTextColor(Color.BLACK);
614         params = new RelativeLayout.LayoutParams(scrWidth /*Width*/,
    (int) (0.09 * scrHeight))/*Height*/;
615         params.leftMargin = (int) (0.1 * scrHeight); // X-Position
616         params.topMargin = (int) (0.05 * scrHeight) + (int) (0.1 *
    scrHeight) + (int) (0.1 * scrHeight); // Y-Position
```

```
617         lyt.addView(total, params); // add it to the View
618
619         TextView total2 = new TextView(this);
620         total2.setText("" + (set.size()));
621         total2.setTextSize(36);
622         total2.setTextColor(Color.BLACK);
623         params = new RelativeLayout.LayoutParams(scrWidth /*Width*/,
    (int) (0.09 * scrHeight))/*Height*/;
624         params.leftMargin = (int) (0.4 * scrHeight); // X-Position
625         params.topMargin = (int) (0.05 * scrHeight) + (int) (0.1 *
    scrHeight) + (int) (0.1 * scrHeight); // Y-Position
626         lyt.addView(total2, params); // add it to the View
627
628         // Show "repeat wrong guessed"-button if there are cards left
629         if (wrongGuessed.size() > 0) {
630             Button repWrong = new Button(this);
631             repWrong.setText("Repeat wrong guessed");
632             repWrong.setTextColor(Color.WHITE);
633             repWrong.getBackground().setColorFilter(getResources().
    getColor(R.color.colorAccent), PorterDuff.Mode.MULTIPLY);
634             params = new RelativeLayout.LayoutParams((int) (0.90 *
    scrWidth)/*Width*/, (int) (0.1 * scrHeight)/*Height*/);
635             params.leftMargin = (int) (0.05 * scrWidth);
636             params.topMargin = (int) (0.4 * scrHeight) + (int) (0.1 *
     scrHeight) + (int) (0.1 * scrHeight) + (int) (0.1 * scrHeight);
637             lyt.addView(repWrong, params);
638             repWrong.setOnClickListener(new View.OnClickListener() {
639                 public void onClick(View view) {
640                     buildMenu();
641                     indexCard = 0;
642                     showFirstPic();
643                 }
644             });
645         }
646
647         // Show "Repeat all"-button
648         Button repAll = new Button(this);
649         repAll.setText("Repeat all");
650         repAll.setTextColor(Color.WHITE);
651         repAll.getBackground().setColorFilter(getResources().getColor
    (R.color.colorPrimary), PorterDuff.Mode.MULTIPLY);
```

```
652         params = new RelativeLayout.LayoutParams((int) (0.90 *
    scrWidth)/*Width*/, (int) (0.1 * scrHeight)/*Height*/);
653         params.leftMargin = (int) (0.05 * scrWidth);
654         params.topMargin = (int) (0.5 * scrHeight) + (int) (0.1 *
    scrHeight) + (int) (0.1 * scrHeight) + (int) (0.1 * scrHeight);
655         lyt.addView(repAll, params);
656         repAll.setOnClickListener(new View.OnClickListener() {
657             public void onClick(View view) {
658                 restartAll();
659             }
660         });
661 
662     }
663 
664     public void restartAll() { // This restarts all. A fresh start
665         wrongGuessed = (ArrayList) set.clone();
666         if (!queryLand) {
667             removeLands();
668         }
669         shuffleWrongs();
670         buildMenu();
671         indexCard = 0;
672         showFirstPic();
673     }
674 
675     public void removeLands() { // This removes all Lands of
    wrongGuessed
676         ArrayList<Card> remove = new ArrayList<>();
677         for (Card c : wrongGuessed) {
678             if (c.getType().contains("Land")) {
679                 remove.add(c);
680             }
681         }
682         for (Card c : remove) {
683             wrongGuessed.remove(c);
684         }
685 
686     }
687 
688     public boolean onOptionsItemSelected(MenuItem item) {
689         int id = item.getItemId();
```

```
690         switch (id) {
691             case android.R.id.home:
692                 onBackPressed(); // If the user presses the "back"-
    button we close the activity
693                 break;
694             case R.id.restart_all:
695                 restartAll(); // restart the query if the user
    presses "restart"
696                 break;
697             case R.id.query_lands: // removes/adds lands
698                 queryLand = !queryLand;
699                 item.setChecked(queryLand);
700                 if (queryLand) {
701                     restartAll();
702                 } else {
703                     removeLands();
704                     updateScore();
705                 }
706                 break;
707             case R.id.query_revers: // Change modes
708                 if (set.size() < 4) {
709                     break;
710                 }
711                 if (Mode == 1) {
712                     Mode = 0;
713                 } else {
714                     Mode = 1;
715                 }
716                 restartAll();
717                 break;
718         }
719         return true;
720     }
721
722     public void onBackPressed() {
723         finish(); // This closes our Activity
724         overridePendingTransition(R.anim.slide_in_right, R.anim.
    slide_out_right); // We want to close it with an fancy animation.
725     }
726 }
```

7.9 FavoritesActivity.java

```
1  package m2b.magic2brain.com;
2
3  import android.content.Context;
4  import android.content.Intent;
5  import android.os.Bundle;
6  import android.support.design.widget.FloatingActionButton;
7  import android.support.design.widget.Snackbar;
8  import android.support.v7.app.AppCompatActivity;
9  import android.view.MenuItem;
10 import android.view.View;
11 import android.widget.AdapterView;
12 import android.widget.ArrayAdapter;
13 import android.widget.ListView;
14 import java.util.ArrayList;
15 import m2b.magic2brain.com.magic2brain.R;
16
17 /*
18 This class shows all Favorites in a list
19 */
20
21 public class FavoritesActivity extends AppCompatActivity {
22
23     protected void onCreate(Bundle savedInstanceState) {
24         overridePendingTransition(R.anim.slide_in_left, R.anim.
    slide_out_left); // This adds an fancy slide animation, when this
    activity starts.
25         super.onCreate(savedInstanceState); // This does some intern
    stuff. We don't need to worry about that. It's just needed.
26         setTitle(getString(R.string.FavoritesActivity_title)); // We
    set a title to our View. We defined the title in "/res/values/
    strings.xml". We just load it from there.
27         setContentView(R.layout.activity_favorites); // This adds an
    View to our Activity. We defined at "/res/layout/
    activity_favorites.xml" how our activity should look like.
28         getSupportActionBar().setDisplayHomeAsUpEnabled(true); //
    With this line we add an "back"-Button to the Toolbar. If we press
     it it calls onOptionsItemSelected();
29         buildMenu(); // This simply builds the menu
30     }
```

```
31
32      protected void onResume() { // If we get back to this activity,
        we build the menu again, because its possible that the user
        removed a card.
33          super.onResume();
34          buildMenu();
35      }
36
37      private void buildMenu() {
38          final Context currentContext = this; // This doesn't actually
         do anything, but it's needed if we want to refer to "this" from
        an inner class.
39
40          final ArrayList<Card> alist_favs = Favorites.favorites_mvid;
        // This loads all the favorites
41          final Card[] cards = new Card[alist_favs.size()]; // We want
        to turn it into a card-array, so its finite
42          final String[] favs = new String[alist_favs.size()]; // For
        the list we need just the names
43          for (int i = 0; i < alist_favs.size(); i++) {
44              favs[i] = alist_favs.get(i).getName(); // We add all
        names into the string-array
45              cards[i] = alist_favs.get(i); // We add all cards into
        the card-array
46          }
47
48          final ArrayAdapter adapter = new ArrayAdapter(this, android.R
        .layout.simple_list_item_1, favs); // This is just a helper-class.
         It transforms our String-Array into an clickable List.
49          final ListView lv = (ListView) findViewById(R.id.favList); //
         We want to add our list to the ListView. So we get the ListView
        from the View
50          lv.setAdapter(adapter); // We add our list into it.
51
52          lv.setOnItemClickListener(new AdapterView.OnItemClickListener
        () { // This performs an action when we click on an item from the
        list.
53              public void onItemClick(AdapterView<?> parent, View view,
         int position, long id) {
54                  // The following code starts CardBrowserActivity and
        passes the card of the item we clicked.
```

```
55                  Intent intent = new Intent(currentContext,
    CardBrowserActivity.class);
56                  intent.putExtra("currentCard", cards[position]);
57                  startActivity(intent);
58              }
59          });
60
61
62          FloatingActionButton fam = (FloatingActionButton)
    findViewById(R.id.fab_addlearn); // This gets the "start"-button
    from the view
63          fam.setOnClickListener(new View.OnClickListener() { // when
    we press the button an action should be performed
64              public void onClick(View view) {
65                  // The following code checks the amount of the
    favorites. If it's zero, the user gets a notification which says,
    that there are no cards to learn.
66                  // If there are cards in favorites, it constructs a
    deck-object and fills it with all the cards. After that it starts
    the query with that deck.
67                  if (alist_favs.size() > 0) {
68                      Intent intent = new Intent(currentContext,
    QueryActivity.class);
69                      Deck d = new Deck();
70                      d.setCode("FAVS");
71                      d.setName("Favorites");
72                      d.setSet(alist_favs);
73                      intent.putExtra("Set", d);
74                      startActivity(intent);
75                  } else {
76                      Snackbar.make(view, R.string.No_Favs_to_learn,
    Snackbar.LENGTH_LONG).setAction("Action", null).show();
77                  }
78              }
79          });
80
81      }
82
83      public boolean onOptionsItemSelected(MenuItem item) {
84          switch (item.getItemId()) {
85              // Respond to the action bar's Up/Home button
```

```
86              case android.R.id.home:
87                  onBackPressed();
88          }
89          return true;
90      }
91
92      public void onBackPressed() {
93          finish();  // This closes our Activity
94          overridePendingTransition(R.anim.slide_in_right , R.anim.
    slide_out_right); // We want to close it with an fancy animation.
95      }
96  }
```

7.10 LastSeenActivity.java

```
1  package m2b.magic2brain.com;
2
3  import android.content.Context;
4  import android.content.Intent;
5  import android.content.SharedPreferences;
6  import android.os.Bundle;
7  import android.preference.PreferenceManager;
8  import android.support.v7.app.AppCompatActivity;
9  import android.view.MenuItem;
10 import android.view.View;
11 import android.widget.AdapterView;
12 import android.widget.AdapterView.OnItemClickListener;
13 import android.widget.ArrayAdapter;
14 import android.widget.ListView;
15 import com.google.gson.Gson;
16 import com.google.gson.reflect.TypeToken;
17 import java.lang.reflect.Type;
18 import java.util.ArrayList;
19 import m2b.magic2brain.com.magic2brain.R;
20
21 /*
22 This class should show all recently learned sets.
23  */
24 public class LastSeenActivity extends AppCompatActivity {
25     private ArrayList<String> recentlyLearned; // loadRecent() will
    load all deck-codes in this arraylist
26     private ArrayList<String> recentlyLearnedNames; // loadRecent()
    will load all deck-names in this arraylist
27     private String[] names; // this will contain all the names of the
     decks
28     private String[] reclearn; // this will contain all the deck-
    codes of the decks
29
30     protected void onCreate(Bundle savedInstanceState) {
31         overridePendingTransition(R.anim.slide_in_left, R.anim.
    slide_out_left); // This adds an fancy slide animation, when this
    activity starts.
32         super.onCreate(savedInstanceState); // This does some intern
    stuff. We don't need to worry about that. It's just needed.
```

```
33          setContentView(R.layout.activity_last_seen); // This adds an
    View to our Activity. We defined at "/res/layout/
    activity_last_seen.xml" how our activity should look like.
34          setTitle(getString(R.string.LastSeenActivity_title)); // We
    set a title to our View. We defined the title in "/res/values/
    strings.xml". We just load it from there.
35          getSupportActionBar().setDisplayHomeAsUpEnabled(true); //
    With this line we add an "back"-Button to the Toolbar. If we press
     it it calls onOptionsItemSelected();
36          final Context currentContext = this;  // This doesn't
    actually do anything, but it's needed if we want to refer to "this
    " from an inner class.
37
38          if (!loadRecent()) { // if it fails to load all the decks, we
     simply replace recentlyLearned with an empty arraylist
39              recentlyLearned = new ArrayList<>();
40          }
41          names = new String[recentlyLearned.size()]; // we initiate
    names with the size of recentlyLearned
42          reclearn = new String[recentlyLearned.size()]; // we initiate
     reclearn with the size of recentlyLearned
43          for (int i = 0; i < recentlyLearned.size(); i++) {
44              reclearn[i] = recentlyLearned.get(i); // we fill reclearn
     with all the deck-codes
45              names[i] = recentlyLearnedNames.get(i); // we fill names
    with all the names
46          }
47
48          final ArrayAdapter adapter = new ArrayAdapter(this, android.R
    .layout.simple_list_item_1, names); // This is just a helper-class
    . It transforms our String-Array into an clickable List.
49          ListView lv = (ListView) findViewById(R.id.mListView); // We
    want to add our list to the ListView. So we get the ListView from
    the View
50          lv.setAdapter(adapter); // We add our list into it.
51
52          lv.setOnItemClickListener(new OnItemClickListener() { // This
     performs an action when we click on an item from the list.
53              public void onItemClick(AdapterView<?> parent, View view,
     int position, long id) {
54                  // The following code starts DeckDisplayActivity and
```

```
    passes the Deck-Code and the Deck-Name of the item we clicked.
55                  String code = reclearn[position];
56                  String name = names[position];
57                  Intent intent = new Intent(currentContext,
    DeckDisplayActivity.class);
58                  intent.putExtra("code", code);
59                  intent.putExtra("name", name);
60                  startActivity(intent);
61              }
62          });
63
64      }
65
66      private boolean loadRecent() { // this function loads the
    recently learned deck from the memory. If it fails it returns
    false.
67          SharedPreferences sharedPrefs = PreferenceManager.
    getDefaultSharedPreferences(this);
68          Gson gson = new Gson();
69          String json = sharedPrefs.getString("query_recent4", null);
70          String json2 = sharedPrefs.getString("query_recent_names",
    null);
71          Type type = new TypeToken<ArrayList<String>>() {
72          }.getType();
73          ArrayList<String> aL = gson.fromJson(json, type);
74          ArrayList<String> aL2 = gson.fromJson(json2, type);
75          if (aL == null) {
76              return false;
77          }
78          recentlyLearned = aL;
79          recentlyLearnedNames = aL2;
80          return true;
81      }
82
83      public boolean onOptionsItemSelected(MenuItem item) {
84          switch (item.getItemId()) {
85              // Respond to the action bar's Up/Home button
86              case android.R.id.home:
87                  onBackPressed();
88          }
89          return true;
```

```
90      }
91
92      public void onBackPressed() {
93          finish(); // This closes our Activity
94          overridePendingTransition(R.anim.slide_in_right , R.anim.
        slide_out_right); // We want to close it with an fancy animation.
95      }
96  }
```

7.11 DeckAssetLoader.java

```
1  package m2b.magic2brain.com;
2
3  import android.content.Context;
4  import org.json.JSONArray;
5  import org.json.JSONObject;
6  import java.io.BufferedReader;
7  import java.io.InputStream;
8  import java.io.InputStreamReader;
9  import java.io.Reader;
10 import java.io.StringWriter;
11 import java.io.Writer;
12
13 /*
14 This is an helper-class. All functions all static so we don't need to
       create an object. This class basicly loads all informations of an
       deck or card from an JSON-file from our assets-folder.
15 There's alot of low-level actions going on there.
16 */
17 public class DeckAssetLoader {
18
19     public DeckAssetLoader() {}
20
21     public static Card[] getDeck(String deckname, Context context) {
22         Card[] c = null;
23         try {
24             InputStream is = context.getAssets().open(deckname);
25             Writer writer = new StringWriter();
26             char[] buffer = new char[1024];
27             Reader reader = new BufferedReader(new InputStreamReader(
    is, "UTF-8"));
28             int n;
29             while ((n = reader.read(buffer)) != -1) {
30                 writer.write(buffer, 0, n);
31             }
32             is.close();
33             String jsonString= writer.toString();
34             c = parseCardJSON(jsonString);
35         } catch (Exception e){}
36         return c;
```

```
37      }
38
39      public static Deck[] getDeckList(Context context) {
40          Deck[] darray = null;
41                  try {
42                      InputStream is = context.getAssets().open("
    SetList.json");
43                      Writer writer = new StringWriter();
44                      char[] buffer = new char[1024];
45                      try {
46                          Reader reader = new BufferedReader(new
    InputStreamReader(is, "UTF-8"));
47                          int n;
48                          while ((n = reader.read(buffer)) != -1) {
49                              writer.write(buffer, 0, n);
50                          }
51                      } finally {
52                          is.close();
53                      }
54                      String jsonString = writer.toString();
55                      darray = parseDeckJSON(jsonString);
56                  } catch (Exception e){}
57          return darray;
58      }
59
60      private static Deck[] parseDeckJSON(String json) {
61          Deck[] list = null;
62          try {
63              JSONArray file = new JSONArray(json);
64
65              list = new Deck[file.length()];
66
67              for (int i = 0; i < file.length(); i++) {
68                  JSONObject card = file.getJSONObject(i);
69                  Deck c = new Deck();
70
71                  String deck_name = card.getString("name");
72                  String deck_code = card.getString("code");
73                  String deck_releaseDate = card.getString("releaseDate
    ");
74
```

```
75                  c.setName(deck_name);
76                  c.setCode(deck_code);
77                  c.setReleaseDate(deck_releaseDate);
78
79                  list[i] = c;
80              }
81          } catch (Exception e){}
82          return list;
83      }
84
85      private static Card[] parseCardJSON(String json)  {
86          Card[] carray = null;
87          try {
88              JSONObject deck = new JSONObject(json);
89              JSONArray cards = deck.getJSONArray("cards");
90              carray = new Card[cards.length()];
91
92              for (int i = 0; i < cards.length(); i++) {
93                  JSONObject card = cards.getJSONObject(i);
94                  Card c = new Card();
95                  String mvid_as_string = card.getString("multiverseid"
    );
96                  String card_name = card.getString("name");
97
98                  String card_flavor = "";
99                  String card_text = "";
100                 String card_type = "";
101                 String card_cost = "";
102
103                 if (card.has("flavor")) {
104                     card_flavor = card.getString("flavor");
105                 }
106
107                 if (card.has("text")) {
108                     card_text = card.getString("text");
109                 }
110
111                 if (card.has("type")) {
112                     card_type = card.getString("type");
113                 }
114                 if (card.has("manaCost")) {
```

```
                    card_cost = card.getString("manaCost");
                }

                c.setName(card_name);
                c.setMultiverseid(Integer.parseInt(mvid_as_string));
                c.setText(card_text);
                c.setFlavor(card_flavor);
                c.setType(card_type);
                c.setManaCost(card_cost);

                carray[i] = c;
            }
        } catch (Exception e){}
        return carray;
    }
}
```

7.12 Card.java

```
1  package m2b.magic2brain.com;
2
3  import java.io.Serializable;
4
5  /*
6  This is a class to store Cards. It stores all crucial informations of
       a card and has the Getters and Setters for it. There are also
     three Constructors.
7  */
8  public class Card implements Serializable {
9      private int multiverseid;
10     private String name;
11     private String flavor;
12     private String text;
13     private String type;
14     private String manacost;
15
16     public Card(int multiverseid, String name) {
17         setMultiverseid(multiverseid);
18         setName(name);
19         // We fill the undefined variables with something to avoid
     NullPointer-Errors.
20         type = "UNKOWN";
21         flavor = "UNKOWN";
22         text = "UNKOWN";
23         manacost = "NA";
24     }
25
26     public Card(String name, String flavor, String text, String type,
       String manacost) {
27         this.type = type;
28         this.name = name;
29         this.flavor = flavor;
30         this.text = text;
31         this.manacost = manacost;
32     }
33
34     public Card() {}
35
```

```
36      public int getMultiverseid() {
37          return multiverseid;
38      }
39
40      public void setMultiverseid(int multiverseid) {
41          this.multiverseid = multiverseid;
42      }
43
44      public String getName() {
45          return this.name;
46      }
47
48      public void setName(String name) {
49          this.name = name;
50      }
51
52      public String getFlavor() {
53          return this.flavor;
54      }
55
56      public void setFlavor(String flavor) {
57          this.flavor = flavor;
58      }
59
60      public String getText() {
61          return this.text;
62      }
63
64      public void setText(String text) {
65          this.text = text;
66      }
67
68      public String getType() {
69          return type;
70      }
71
72      public void setType(String type) {
73          this.type = type;
74      }
75
76      public String getManaCost() {
```

```
77          return manacost;
78      }
79
80      public void setManaCost(String type) {
81          this.manacost = type;
82      }
83  }
```

7.13 Deck.java

```
1  package m2b.magic2brain.com;
2
3  import java.io.Serializable;
4  import java.util.ArrayList;
5  import java.util.Arrays;
6
7  /*
8  This class stores a deck with all its informations. It has the
       Getters and Setters for the informations. There are also two
       Constructors.
9   */
10 public class Deck implements Serializable /* We need to do this, so
       we can pass Decks with Intents */ {
11
12     private ArrayList<Card> set;
13     private String name;
14     private String code;
15     private String releaseDate;
16     private String icon;
17
18     public Deck() {
19         set = new ArrayList<>();
20     }
21
22     public Deck(String name, String code, String release_date, String
       iconUri) {
23         this.name = name;
24         this.code = code;
25         this.releaseDate = release_date;
26         this.icon = iconUri;
27         set = new ArrayList<>();
28     }
29
30     public String getIcon() {
31         return icon;
32     }
33
34     public void setIcon(String icon) {
35         this.icon = icon;
```

```
36      }
37
38      public String getName() {
39          return name;
40      }
41
42      public void setName(String name) {
43          this.name = name;
44      }
45
46      public String getCode() {
47          return code;
48      }
49
50      public void setCode(String code) {
51          this.code = code;
52      }
53
54      public String getReleaseDate() {
55          return releaseDate;
56      }
57
58      public void setReleaseDate(String releaseDate) {
59          this.releaseDate = releaseDate;
60      }
61
62      public int getSize() {
63          return set.size();
64      }
65
66      public ArrayList<Card> getSet() {
67          return set;
68      }
69
70      public void setSet(ArrayList<Card> al) {
71          set = al;
72      }
73
74      public void setSet(Card[] c){set = new ArrayList<>(Arrays.asList(
        c));}
75  }
```

7.14 Favorites.java

```
1  package m2b.magic2brain.com;
2
3  import java.util.ArrayList;
4
5  /*
6  This is very simple class. It's just here to temporary store all
       favorites and pass them to any other class.
7  */
8
9  public final class Favorites {
10     public static ArrayList<Card> favorites_mvid;
11     public static void init() {
12         favorites_mvid = new ArrayList<>();
13     }
14 }
```

7.15 RUtils.java

```
1  package m2b.magic2brain.com;
2
3  /*
4  This class is a helper-class. All functions are static so we don't
       need any objects.
5   */
6  public class RUtils {
7
8      public static String[] getListified(Card[] cards) { // This turns
       a Card-Array into a String-Array
9          String[] list = new String[cards.length];
10         for (int i = 0; i < cards.length; i++) {
11             list[i] = cards[i].getName();
12         }
13         return list;
14     }
15
16     public static boolean isInteger(String s) {
17         return isInteger(s, 10);
18     }
19
20     public static boolean isInteger(String s, int radix) {
21         if (s.isEmpty()) return false;
22         for (int i = 0; i < s.length(); i++) {
23             if (i == 0 && s.charAt(i) == '-') {
24                 if (s.length() == 1) return false;
25                 else continue;
26             }
27             if (Character.digit(s.charAt(i), radix) < 0) return false
       ;
28         }
29         return true;
30     }
31 }
```