



Frankfurt University of Applied Sciences
- FACHBEREICH 2 INFORMATIK - MOBILE ANWENDUNGEN -

Dokumentation Human Machine Interface: Android Projekt: Grundlagen, Elementare Bausteine und Spiele

Projekt Dokumentation zur Erlangung des Moduls 23 Human Machine Interfaces (HMI)

vorgelegt von

Marcel Klamm

E-Mail: Marcel.Klamm@googlemail.com

5. Fachsemester

Matrikelnummer: 1055627

Frankfurt University of Applied Sciences, Frankfurt a.M. WiSe 2015/2016

Referent:

Prof. Dr. Matthias Deegener

Inhaltsverzeichnis

Abbildungsverzeichnis	iv
Tabellenverzeichnis	vii
1 Einleitung	1
2 Einführung	3
3 Das Human Machine Interface	4
4 Android - eine offene und mobile Plattform	6
4.1 Systemarchitektur	7
4.2 Linux-Kernel	7
4.3 Libraries	8
4.4 Android Runtime	8
4.5 Applications	8
4.6 Application Framework	8
5 Grundlagen	10
5.1 Android-Projekte	10
5.2 Benutzeroberfläche	12
5.3 Texte	12
5.4 Oberflächenbeschreibungen	13
5.5 Activities	14
6 Android-App: HMI	17
6.1 Main Activity	17
6.2 Voice	18
6.3 Widgets	19
6.4 Menüs	19
6.4.1 Kontextmenüs	20
6.5 Layouts	20
6.5.1 Absolute Layout	21

6.5.2	Frame Layout	22
6.5.3	Relative Layout	22
6.5.4	Linear Layout	23
6.5.5	Grid Layout	23
6.5.6	Table Layout	24
6.5.7	List View	24
6.5.8	Swipe View	25
6.5.9	Action Bar tabs	26
6.5.10	Navigation Drawer	26
6.6	Optische Täuschungen	27
6.7	Dialoge	28
6.8	Sensoren	28
6.9	Eingabe	30
6.9.1	Button	30
6.9.2	Textfelder	31
6.9.3	Checkbox	32
6.9.4	Radio Button	33
6.10	Games	34
6.10.1	Spiel mit Zeit	34
6.10.2	Spiel mit Punkten	35
6.10.3	Spiel mit Analoger Steuerung	36
6.10.4	Spiel mit Touch Steuerung	36
6.10.5	Spiel mit Beschleunigungssensor	37
6.10.6	Kugel ins Loch	38
6.10.7	Ich packe meinen Koffer	38
7	Fazit	41
8	Literaturverzeichnis	42
Literaturverzeichnis		I

Abbildungsverzeichnis

1.1	Statistik der Java-Quelltexte; Programm: LocMetrics, locmetrics.com	2
1.2	Statistik der Ressourcen-Quelltexte; Programm: LocMetrics, locmetrics.com . .	2
4.1	Schematische Aufbau der Android-Plattform; https://de.wikipedia.org/wiki/Android_(Betriebssystem-Architecture.svg	7
4.2	Schematische Aufbau der Android-Plattform; http://www.giga.de/apps/android/specials/android-berechtigungen-app-rechte-im-detail-erklaert/	9
5.1	Zeigt einen Ausschnitt des Datei-Explorers von Android Studio; http://code.tutsplus.com/tutorials/sdk-working-with-android-studio--mobile-20203	10
5.2	Beispielanwendung ohne Unterstützung für unterschiedliche Dichten, wie sie auf niedrig, mittel und High-Density-Bildschirme angezeigt wird; http://developer.android.com/guide	11
5.3	Beispielanwendung mit guter Unterstützung für unterschiedliche Dichten (Die App ist Bildschirm Dichte unabhängig), gezeigt auf niedrig, mittel und einer hohe Dichte.; http://developer.android.com/guide/practices/screens_support.html	12
5.4	Lebenszyklus von Activity; http://developer.android.com/guide/components/activities.html	15
6.1	Hauptbildschirm der app; activity_main.xml	17
6.2	VoiceActivity mit den Funktionen gesprochenes zu erkennen, die Zeit anzusagen und geschriebenes aus einer EditText-Box in einer beliebigen installierten Sprache zu Sprechen (Text-to-Speech); voice_layout.xml	18
6.3	VoiceActivity mit den Funktionen gesprochenes zu erkennen, die Zeit anzusagen und geschriebenes aus einer EditText-Box in einer beliebigen installierten Sprache zu Sprechen (Text-to-Speech); voice_layout.xml	19
6.4	MenuActivity mit Menüs und Kontextmenüs; menu_layout.xml	20
6.5	LayoutActivity mit Buttons zur Navigation zu unterschiedlichen Layout Beispielen; layout_layout.xml	21
6.6	AbsoluteActivity Oberfläche; absolute_layout.xml	21
6.7	FrameActivity Oberfläche; frame_layout.xml	22
6.8	RelativeLayout Oberfläche; relative_layout.xml	22
6.9	LinearLayout Oberfläche; linear_layout.xml	23
6.10	GridActivity Oberfläche; grid_layout.xml	23

6.11	TableActivity Oberfläche; table_layout.xml	24
6.12	ListView Oberfläche mit 2 Listen, die obere beinhaltet nur Text, die untere besitzt darüber hinaus Grafiken; list_layout.xml	25
6.13	SwipeviewActivity Oberfläche in Android 5.0 Material Design; tab_layout.xml .	25
6.14	ActionBarTabsActivity Oberfläche in Android 5.0 Material Design; actionBar_layout.xml	26
6.15	NavigationDrawerActivity Oberfläche in Android 5.0 Material Design; navigationdrawer_layout_layout.xml	27
6.16	OptischeTaeuschungActivity; optische_taeuschung_layout.xml	27
6.17	DialogActivity, im linken Teil sieht man die Toast-Nachricht mit dem Inhalt aus dem EditText Feld. Rechts daneben sieht an einen AlertDialog, auf den der Nutzer reagieren muss um zur normalen App-Ansicht zurückzukehren. Im vorletzten Bild ist ein DatePickerDialog gezeigt, der den Nutzer zum Auswählen eines Datum bittet. Das letzte Bild zeigt ein TimePickerDialog der den Anwender um eine Uhrzeit Eingabe bittet; dialog_layout.xml	28
6.18	SensorActivity, menü zur Auswahl einiger Sensoren. Am unteren Rand werden alle verfügbaren Sensoren samt Daten angezeigt; sensor_layout.xml	29
6.19	HelligkeitActivity, zeigt über den Smartphone Sensor die Helligkeit der Umgebung an; helligkeit_layout.xml	29
6.20	BeschleunigungActivity, zeigt auf dreidimensionalen Achsen: x, y, z die Beschleunigung an. Beim Hochhalten bzw liegend fällt die Erdanziehungskraft auf den Achsen auf; beschleunigung_layout.xml	30
6.21	InputActivity, übersicht der Eingabe-Methoden und verzweigung per Buttons; input_layout.xml	30
6.22	ButtonActivity, übersicht der verschiedenen Button Gestaltungsmöglichkeiten. Der untere Rotate Button dreht sich wie ein Drehregler beim anklicken; button_layout.xml	31
6.23	EdittextActivity, übersicht der Eingabe-Methoden für verschiedene Textfelder; edittext_layout.xml	32
6.24	CheckboxActivity, übersicht von verschiedenen Kontrollfeldern; checkbox_layout.xml	33
6.25	RadiobuttonActivity, gruppierungen von verschiedenen Radio-Buttons zur Auswahl; radiobutton_layout.xml	34
6.26	GameActivity, menü zum starten der Spiele; game_layout.xml	34
6.27	GameTimeActivity, spiel bei dem es um das Zeitgefühl geht; game_time_layout.xml	35
6.28	GameDotsActivity, klicke auf das gesuchte Objekt, bevor die Zeit abläuft . . .	36
6.29	MainActivity, steuer die Android Figur mit Hilfe der Buttons	36
6.30	MainActivity, steuer das HMI Logo mit Hilfe deiner Finger auf dem Touchscreen	37
6.31	MainActivity, mit dem Beschleunigungssensor des Gerätes wird die Android Figur gesteuert	37

6.32 MainActivity, mit dem Beschleunigungssensor des Gerätes wird die Android Figur gesteuert	38
6.33 MainActivity, aus dem laufenden Spiel kann man entnehmen, dass es die erste Runde ist und Spieler 1 an der Reihe ist. Nach einen Klick auf den Button beginnt die "Voice Recognition", also die Spracherkennung	39
6.34 MainActivity, "Voice Recognition", die Spracherkennung fragt den Nutzer, was im Koffer bisher alles drinne steckt	39
6.35 MainActivity, anschließend ist der nächste Spieler am Zug	40

Tabellenverzeichnis

Listingverzeichnis

5.1	Das Listing zeigt wie auf eine Referenz namens nachricht ein Text im Quellcode gesetzt wird	12
5.2	Das Listing zeigt einen Ausschnitt der strings.xml Ressource	12
5.3	Das Listing zeigt ein Beispiel zur Oberflächenbeschreibung einer Layout Datei .	13
6.1	Das Listing zeigt wie mit einen Intent eine neue Activity gestartet wird. Ein Intent erwartet als ersten Parameter die Start Activity und als zweiten Parameter die Ziel Activity	18
6.2	Das Listing zeigt einen Ausschnitt wie Benutzereingaben von Buttons registriert werden	31

Kapitel 1

Einleitung

In dieser Dokumentation soll anhand einer Android Applikation die Kommunikation und Interaktion zwischen Mensch und Maschine, auch “*Human Machine Interface*” genannt, aufgezeigt werden. Insbesondere werden die Problematiken der Kommunikation zwischen Mensch und Maschine verdeutlicht und auf die sich daraus ergebenen Konsequenzen eingegangen. Zum Beispiel wird die Frage aufgeworfen, welche Folgen sich aus einer nicht funktionierenden Kommunikation ergeben, wie diese Probleme mittels bestimmten Methoden und Techniken in Android vermieden werden können und letztlich sogar zu einer Optimierung beitragen

Für das Projekt wurde ein YouTube¹ Video erstellt, welches die Inhalte und Codeabschnitte erläutert sowie die App zeigt. Zusätzlich wurde das Projekt auf GitHub² veröffentlicht. Das Projekt umfasst insgesamt 9072 Code-Zeilen verteilt auf 135 Quellcode Dateien. Für die Java-Quelltexte fallen darunter 5385 Zeilen Code auf 60 Dateien an und für die Ressourcen sind es 3687 Zeilen Code auf 75 Dateien verteilt, siehe Abbildung 1.1 und 1.2. Programmiert wurde mit Googles Android Studio³ in der aktuellen stabilen Version 1.5 mit dem Software Development Kit in Version 23.1.0. Während der Entwicklung wurde die App auf einem Virtuellen Google Nexus 5 mit API 23 Android 6.0 sowie einem Samsung Galaxy Note 3 API 21 Android 5.0 getestet. Die Dokumentation wurde mit LATEX erstellt, welches ich mit diesem Projekt zusätzlich erlernen wollte.

¹<https://youtu.be/tJVBKmsb4ZM>

²<https://github.com/fujay/MyApplication>

³<http://developer.android.com/tools/studio/index.html>

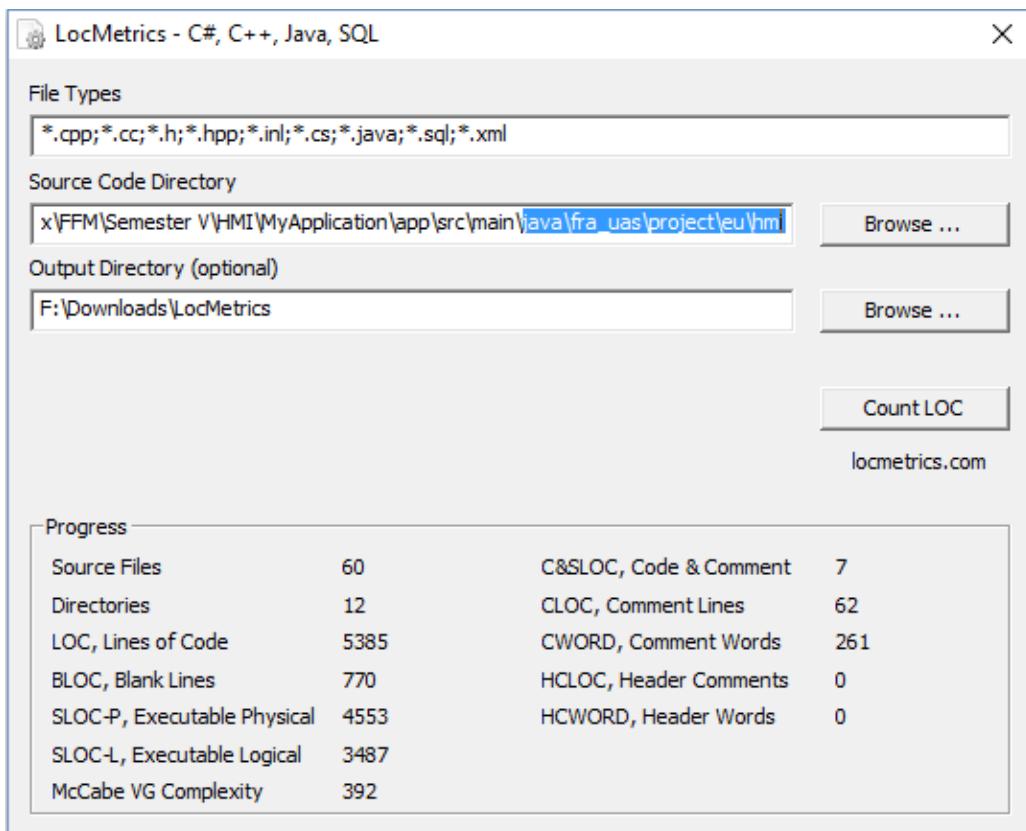


Abbildung 1.1: Statistik der Java-Quelltexte; Programm: LocMetrics, locmetrics.com

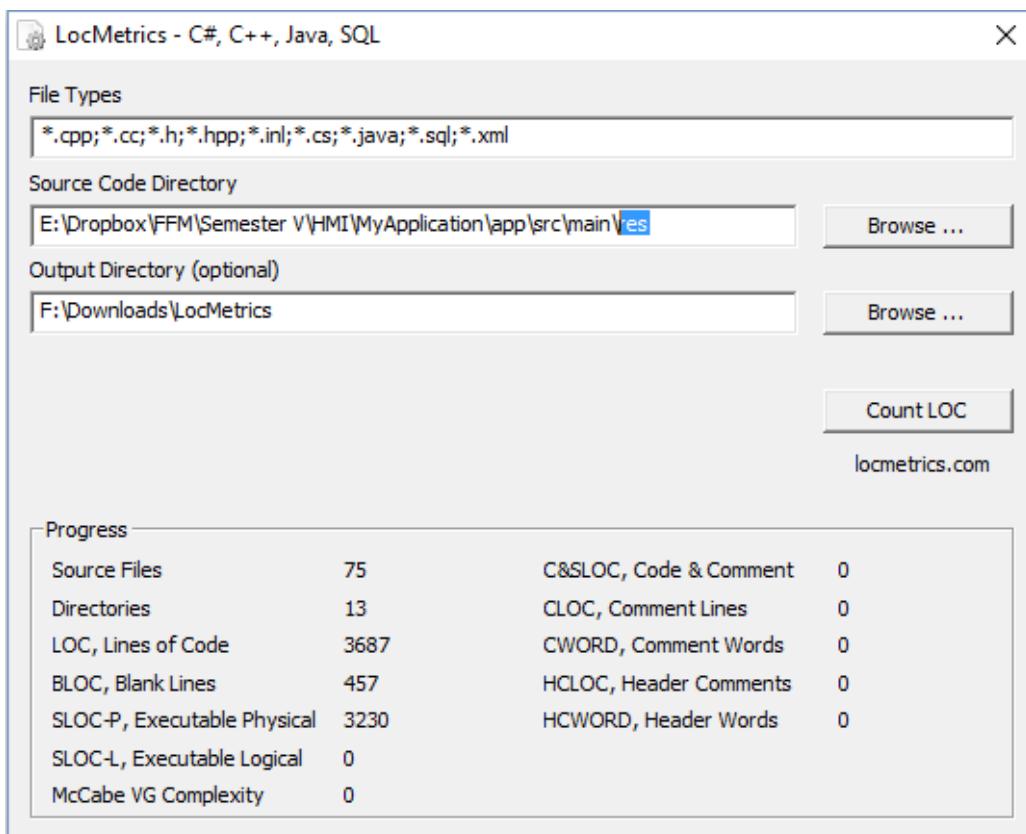


Abbildung 1.2: Statistik der Ressourcen-Quelltexte; Programm: LocMetrics, locmetrics.com

Kapitel 2

Einführung

Das Android System boomt und das liegt nicht zuletzt daran, dass Google es den Entwicklern leichtmacht: Programmiert wird mit der weitverbreiteten Sprache Java, Projektstruktur und Ressourcen werden in menschlich lesbaren XML-Formaten geschrieben, es stehen kostenlose und leistungsfähige Entwicklungswerkzeuge zur Verfügung, die Zugangshürde ist somit niedrig. Dank der mobilen und offenen Plattform, wird sich das System noch weiterverbreiten und auf weitere Technische mobile bzw. Embedded Geräte finden lassen.

Kapitel 3

Das Human Machine Interface

Die Benutzerschnittstelle von Android erfolgt dort, wo ein Dialog zwischen Mensch und Maschine erfolgt, der eine Interaktion bewirkt. Eine solche Schnittstelle bestimmt die Art und Weise wie Mensch und Maschine unmittelbar miteinander kommunizieren, wie der Mensch seine Anweisungen an die Maschine übermittelt und in welcher Form diese Anweisung ausgeführt und die Ergebnisse ausgegeben werden.

In den Anfangsjahren der Computertechnik waren Benutzereingaben textorientiert. Der Benutzer musste spezielle Kommandos eingeben die dann vom Betriebssystem interpretiert wurden. Die Schnittstelle nannte sich *Character User Interface* (CUI).

Solche Eingaben waren sehr aufwendig, fehleranfällig und nicht intuitiv. Es entwickelte sich daraus die visualisierte Schnittstelle, bei der die Eingaben manuell über Maus und Tastatur oder Touch- und Multitouchscreens ausgeführt werden. Eine solch grafische Benutzerschnittstelle wurde *Graphical User Interface* (GUI) genannt.

Eine Weiterentwicklung, die *Voice User Interface* (VUI) arbeitet mit Spracherkennung und Sprachsteuerung sowie einer Sprachausgabe.

Der GUI und VUI kombiniert mit Gestenerkennung entwickelten sich zum *Natural User Interface* (NUI). Geboten wird eine Zweifingerbewegung für die Ausführung von Befehlen auf einem Touchscreen.

Eine Weiterentwicklung dreht sich um Barrierefreiheit für Eingeschränkte Benutzer, *Bain Computer Interface* (BCI). Eine solche Benutzerschnittstelle soll Sprach-, Gesten- und auch Gedankensteuerung besitzen. Seit Version 4.0 besitzt das Betriebssystem einen hohen Grad an Barrierefreiheit¹. Dazu gehören eine Sprachausgabe (Screenreader) für Bildschirminhalte, Bildschirmvergrößerung sowie Unterstützung für Braille Zeilen² und vieles Mehr. Eine Übersicht über Bedienungshilfen für Android gibt folgendes Dokument³.

Android bietet eine Vielzahl an Benutzerschnittstellen zur Eingabe und Kommunikation mit dem System. Dazu bietet es Unterstützung für unterschiedliche Bildschirmgrößen, beispielweise die einer Smartwatch bis hin zu großen Tablets und Bedienungshilfen für eingeschränkte

¹<http://www.marlem-software.de/marlemblog/tag/android/>

²<https://support.google.com/accessibility/android/answer/3535226?hl=de>

³https://support.google.com/accessibility/android/answer/6006564?hl=de&ref_topic=6007234

Nutzer. Dank diesen Schnittstellen ist es möglich, viele Benutzer und Geräte zusammenzuführen.

Kapitel 4

Android - eine offene und mobile Plattform

Android ist sowohl ein Betriebssystem als auch eine Software-Plattform, welche 2007 von Google veröffentlicht wurde. Die Interessen und Neugier der Entwickler und Hersteller an einer offenen Plattform für Embedded Systems war sehr groß. Android läuft mittlerweile auf einer Vielzahl verschiedener mobile Gerätetypen wie Smartphones, Auto-Infotainment, Mediaplayer, Netbooks, Tablet-Computer und Smartwatches. Dank zahlloser Apps für Android lässt sich die Funktionsvielfalt der Geräte zusätzlich erweitern und an die eigenen Bedürfnisse anpassen. Dank der engen Integration von Googles Diensten, den zahlreichen Sensoren und der Offenen Plattform sowie der leichte Einstieg in die Programmierung dank Java als Programmiersprache und Android Studio als Entwicklungsumgebung, machten Android so populär. Entwickelt wird Android von der Open Handset Alliance, ein Konsortium von Netzbetreiber, Softwareentwickler, Halbleiter- und Mobiltelefonhersteller welche mittlerweile 84 Unternehmen umfasst und von Google gegründet wurde. Die aus der Pressemitteilung vom 5.November 2007¹ formulierten Ziele waren unter anderem:

- Eine deutliche Verbesserung der Benutzerfreundlichkeit und des Benutzererlebnisses von mobilen Geräten und Diensten
- Die kostengünstigere Entwicklung und Verteilung innovativer Produkte
- Eine schnellere Markteinführung

Dies ließ sich am besten durch eine auf Offenheit und Flexibilität gegründete Zusammenarbeit erreichen. Deshalb wurden praktisch alle Teile des Android-Softwaresystems als Open Source veröffentlicht. Als Smartphone-Betriebssystem besaß Android im zweiten Quartal 2014 einen weltweiten Marktanteil von 84,6 Prozent². Wenn man bedenkt, dass Android im dritten Quartal

¹http://www.openhandsetalliance.com/press_110507.html

²<http://www.golem.de/news/mobile-betriebssysteme-android-laeuft-auf-fast-85-prozent-aller-smartphones.html>

2010 einen weltweiten Marktanteil von 25,5 Prozent³ betrug, ist das eine beachtliche Steigerung. Führt man den Gedankensprung weiter, lässt sich erahnen, dass die Android Plattform auch weiter auf anderen Plattformen ihren Platz finden wird.

4.1 Systemarchitektur

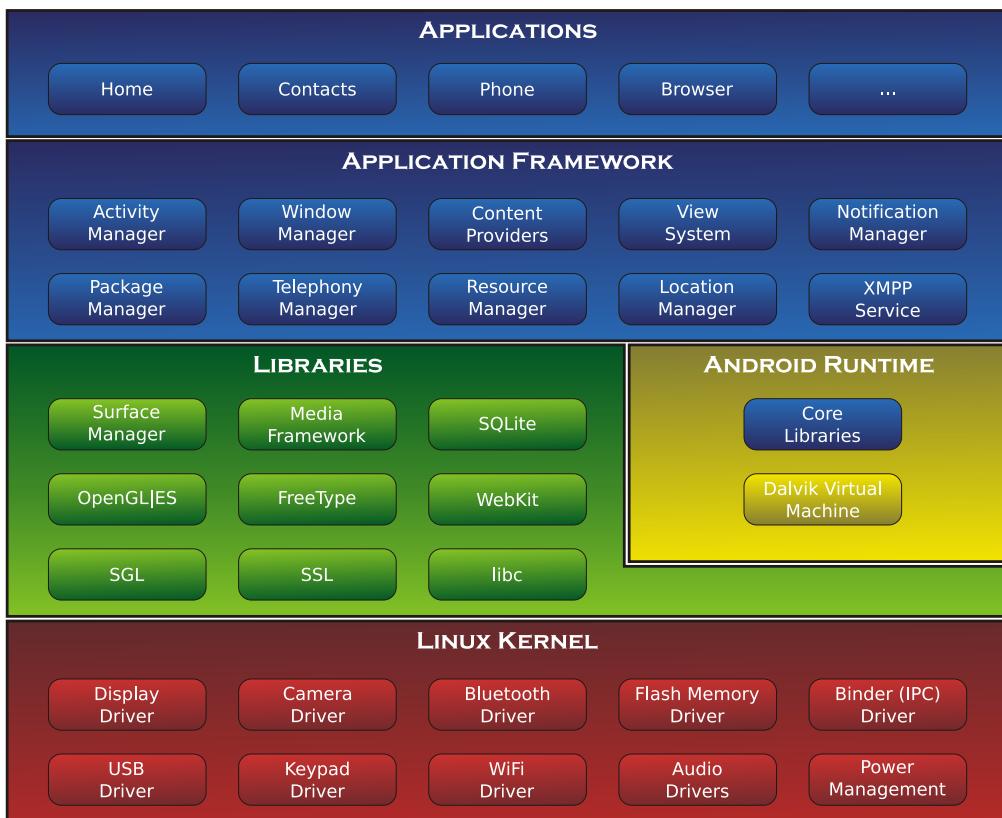


Abbildung 4.1: Schematische Aufbau der Android-Plattform;
[https://de.wikipedia.org/wiki/Android_\(Betriebssystem\)#/media/File:Android-System-Architecture.svg](https://de.wikipedia.org/wiki/Android_(Betriebssystem)#/media/File:Android-System-Architecture.svg)

4.2 Linux-Kernel

Android nutzt als Basis den Linux-Kernel, dass Systemdienste wie Sicherheit, Speicher- und Prozessverwaltung, Treibermodell und Netzwerkstapel zur Verfügung stellt, die in Abbildung 1 rot zu sehen sind.

³<http://www.heise.de/open/meldung/Smartphones-Android-ueberholt-Symbian-Apple-verliert-Marktanteile.html>

4.3 Libraries

Android enthält eine Reihe von C/C++-Bibliotheken, die von verschiedenen Komponenten der Plattform genutzt werden, siehe grüner Teil der Abbildung 1. Entwickler greifen indirekt über das Anwendungsframework auf sie zu. Zu diesen Bibliotheken gehören beispielsweise der *Surface Manager*, welcher Bildschirmzugriffe koordiniert und 2D- und 3D-Ausgaben von Anwendungen zu einem Gesamtbild zusammenfügt. *Media Bibliotheken* für die Aufnahme und Wiedergabe von zahlreichen Audio-, Video- und Grafikformaten. *SQLite* ist eine leichtgewichtige relationale Datenbank. Eine 3D-Grafikbibliothek bei der es sich um *OpenGL ES* handelt. *WebKit* ist eine Rendering Engine für Webinhalte.

4.4 Android Runtime

Android-Programme werden durch die virtuelle Maschine *Dalvik* ausgeführt, welche als Aufgabe hat, eine in Java programmierte App aus Bytecode in maschinenlesbaren Code zu übersetzen. Dieser Code wird anschließend vom Prozessor ausgeführt. Mit Android 5.0 wurde *Dalvik*⁴ durch den Ahead-of-time-Compiler⁵ *Android Runtime* (ART) ersetzt. Siehe dazu den gelben Teil der Abbildung 1. Der zweite Baustein der Android Runtime bilden die sogenannten Core Libraries. Sie enthalten die Android-Java-Klassenbibliotheken. Die Inhalte orientieren sich an der Funktionalität der Java-Standard-Edition (Pakete wie java.math, java.util uvm.).

4.5 Applications

Android ist nicht nur ein Betriebssystem. Zu Android gehören auch eine Reihe von Standardanwendungen, beispielweise Kontakte, Browser und Telefonfunktion.

4.6 Application Framework

Das *Application Framework* erlaubt es äußerst komfortable, ästhetische und leicht bedienbare mobile Anwendungen mit großem Funktionsumfang zu erstellen. Es bietet Zugriff auf die Gerätehardware, zum Beispiel der Kamera, dem Netzwerk oder Sensoren, aber auch das Lesen und Schreiben von Kontaktdataen oder Terminen ist möglich. Ein Rechtesystem steuert hierbei, was ein Programm tun darf, wie auf dem Abbild 2 zu entnehmen ist. Das Rechtesystem erlaubt es auch, dass Anwendungen ihre Funktionen veröffentlichen und anderen Programmen so verfügbar machen können. Programme können also auch Funktionen anderer Anwendungen anfordern. Es lassen sich auch Programmfunctionen ersetzen, so kann der Benutzer sehr leicht den Webbroweser, den E-Mail-Client oder anderes austauschen.

⁴Just-in-time-Compiler, führt Konvertierung in eigenen Bytecode-Format aus für die Java Virtual Machine

⁵Compiler, der Programmcode vor der Ausführung in native Maschinensprache übersetzt

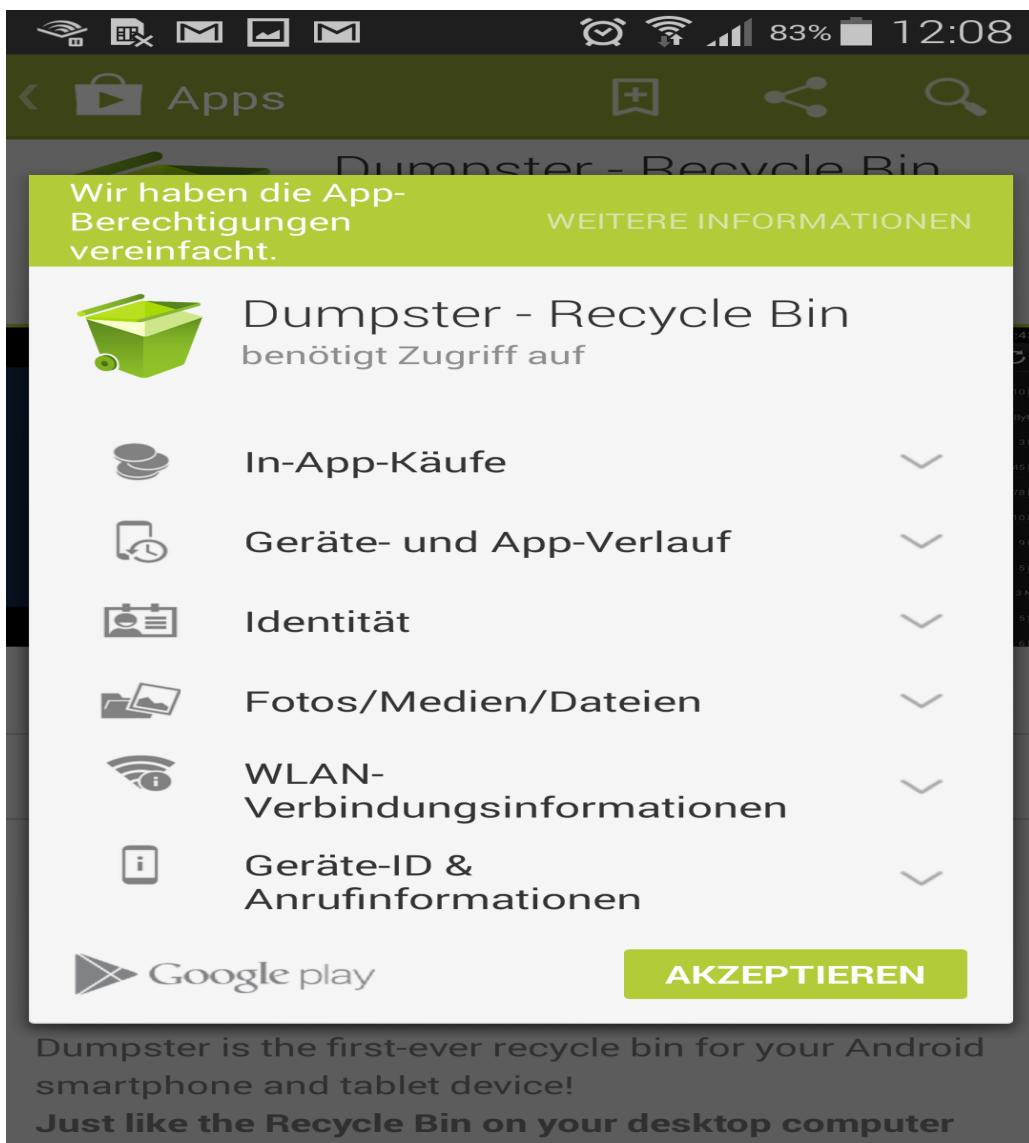


Abbildung 4.2: Schematische Aufbau der Android-Plattform;
<http://www.giga.de/apps/android/specials/android-berechtigungen-app-rechte-im-detail-erklaert/>

Kernbestandteile des *Application Frameworks* (blauer Teil im Abbild 1) sind unter anderem *Views*, welche die Basis für alle Benutzeroberflächen bildet. Dazu gehören auch zahlreiche Standardbedienelemente wie etwa Textfelder oder Schaltflächen. Solche *Views* können durch sogenannte Themes nahezu beliebig angepasst werden. Einen *Activity Manager*, der den Lebenszyklus einer Anwendung steuert. Ein *Content Provider* gestattet Anwendungen den Zugriff auf Daten anderer Programme. Auch das Bereitstellen der eigenen Anwendungsdaten ist auf diese Weise möglich. Der *Resource Manager* gewährt Zugriff auf lokalisierte Zeichenketten, Grafiken und Layoutdateien.

Kapitel 5

Grundlagen

5.1 Android-Projekte

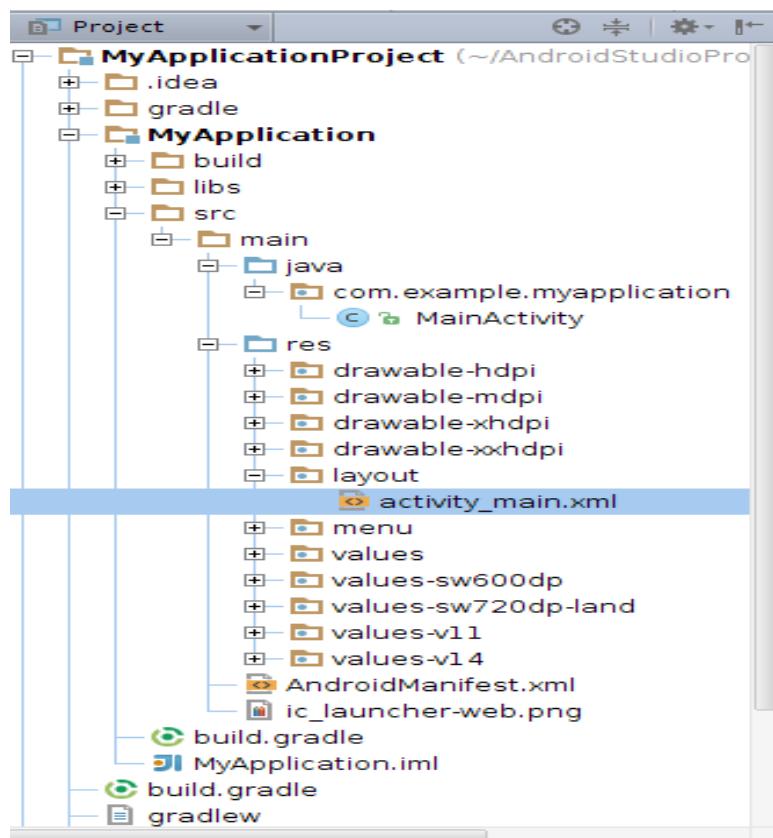


Abbildung 5.1: Zeigt einen Ausschnitt des Datei-Explorers von Android Studio; <http://code.tutsplus.com/tutorials/android-sdk-working-with-android-studio-mobile-20203>

Android-Apps bestehen aus einer ganzen Reihe von Artefakten, die zu einer baumartigen Struktur zusammengefasst werden. Dazu gehören unter anderem Quelltexte, Konfigurationsdateien, Testfälle, aber auch Grafiken, Sounds und Animations. Quelltexte werden in den Ordner src abgelegt. Der Ordner gen der für generated steht, beinhaltet die Klasse R, welche für Ressourcen steht. Sie hat große Bedeutung beim Zugriff auf unterschiedlichste Elemente einer

Android-App. Die Ressource wird aber durch den Erstellprozess oder auch Build-tool genannt, Gradle verwaltet und soll deshalb nicht vom Entwickler verändert werden. Der Inhalt ergibt sich aus Dateien des Verzeichnisses res welches für Ressourcen steht. Die Klasse R wird entsprechend angepasst und übersetzt durch Gradle¹ ², wenn Änderungen an den Unterordnern vorgenommen werden. Der Unterordner values enthält beispielsweise die Datei strings.xml. Sie nimmt Texte auf, die später im Quelltext mithilfe der Klasse R referenziert werden. Genauso wird für die Ordner drawable-[density³]dpi verfahren, wobei das density für eine der Größen⁴ steht. Dort werden Grafiken beispielsweise im png- oder xml-Format gespeichert. Die Unterschiedlichen drawable-Ordnern stehen für unterschiedliche Bildschirme zur Verfügung. Android läuft auf eine Vielzahl von Geräten mit unterschiedlichen Bildschirmgrößen und Auflösungen. Dadurch resultieren unterschiedliche Pixel Dichten, abhängig von beiden Faktoren. Je nach Pixel Dichte wird eine entsprechende Grafik geladen und skaliert. Eine Grafik wirkt auf einen 5.0 Zoll Smartphone mit einer Bildschirmauflösung von 1920*1080 z.B. sehr groß, auf einem Tablet mit 10.0 Zoll mit identischer Auflösung sehr klein, siehe als Vergleich Abbildung 5.1.

Android führt selbst auch eine Skalierung und Größenänderung durch um verschiedene Bild-

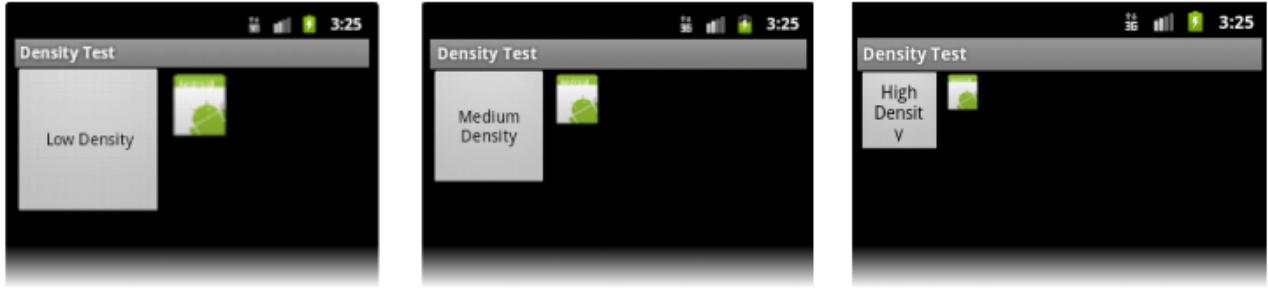


Abbildung 5.2: Beispielanwendung ohne Unterstützung für unterschiedliche Dichten, wie sie auf niedrig, mittel und High-Density-Bildschirme angezeigt wird; http://developer.android.com/guide/practices/screens_support.html

schirme zu unterstützen. Um ein gleiches Bild einer Anwendung auf unterschiedlichen Geräten zu ermöglichen, sollten Grafiken in mehreren Auflösungen in den Drawable-Ordnern vorliegen, siehe Abbildung 5.2 für eine gute Umsetzung.

AndroidManifest.xml ist die zentrale Beschreibungsdatei einer Anwendung. In dieser werden unter anderem die Bestandteile der App aufgeführt. Dazu zählen Informationen darüber, welche Activities (Bildschirmseiten) gestartet werden, welche Rechte eine App benötigt, welche Hardware sie erwartet und unter welchen Systemversionen sie lauffähig ist.

¹Auf Java basierendes Build-Management-Automatisierungs-Tool

²<https://gradle.org>

³http://developer.android.com/guide/practices/screens_support.html

⁴ldpi (low): 120dpi, mdpi (medium): 160dpi, hdpi (high): 240dpi, xhdpi (extra-high): 320dpi, xxhdpi (extra-extra-high): 480dpi, xxxhdpi (extra-extra-extra-high): 640dpi.

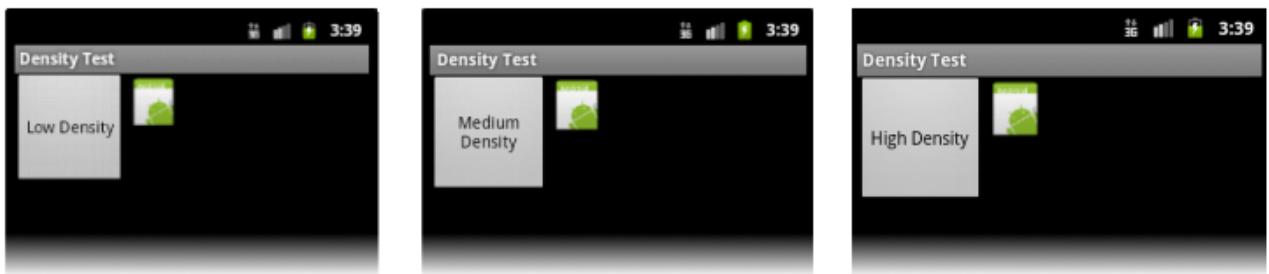


Abbildung 5.3: Beispielanwendung mit guter Unterstützung für unterschiedliche Dichten (Die App ist Bildschirm Dichte unabhängig), gezeigt auf niedrig, mittel und einer hohe Dichte.; http://developer.android.com/guide/practices/screens_support.html

5.2 Benutzeroberfläche

Die Benutzeroberfläche ist das Aushängeschild einer Anwendung. Gerade auf mobilen Geräten mit vergleichsweise kleinen Bildschirmen sollte jede Funktion leicht zugänglich und intuitiv erfassbar sein.

5.3 Texte

Neben Bildern und Symbolen als Gestaltungsmittel spielen auch Texte eine sehr wichtige Rolle. Sie dienen als Beschriftungen von Bedienelementen. Werden für erläuternde Texte genutzt, die durch Screenreader vorgelesen werden. Dazu noch für Hinweis- und Statusmeldungen. Da die Apps in der Programmiersprache Java geschrieben werden, kann man sie einfach im Quelltext ablegen. Das sieht folgendermaßen aus:

```
1 /* Java Text Beispiel */
2
3     nachricht.setText("Hallo Android");
```

Listing 5.1: Das Listing zeigt wie auf eine Referenz namens `nachricht` ein Text im Quellcode gesetzt wird

Das hätte allerdings eine ganze Reihe von Nachteilen. Da jede Klasse in einer eigenen Datei abgelegt wird, merkt man oft nicht, wenn man gleiche Texte mehrfach definiert. Dies vergrößert die App und kostet unnötig Speicher. Zudem wird es auf dieser Weise sehr schwer, mehrsprachige Anwendungen zu programmieren.

Unter Android werden Texte daher zentral in der Datei `strings.xml` abgelegt.

```
1 /* strings.xml Beispiel */
2
3 <?xml version="1.0" encoding="utf-8"?>
4 <resources>
5     <string name="app_name">HMI</string>
```

```
</resources>
```

Listing 5.2: Das Listing zeigt einen Ausschnitt der strings.xml Ressource

Das Attribut *name* des Elements `<string>` wird später im Quelltext als Bezeichner verwendet. Der Name muss also projektweit eindeutig sein. Auf den Text wird dann innerhalb der App mittels Ressource Klasse *R* zugegriffen. Für das Listing aus 5.2 würde der Zugriff folgend lauten: `nachricht.setText(R.string.app_name);` Neben dem Verzeichnis *values* kann es lokalisierte Ausprägungen⁵ geben. Zum Beispiel *values-en* für Englisch oder *values-fr* für Französisch. Die Datei *string.xml* in diesen Ordner enthält Texte in den korrespondierenden Sprachen⁶. Greift Android auf eine Zeichenkette zu, geht das System vom Speziellen zum Allgemeinen. Ist die Standardsprache beispielsweise Englisch, wird zuerst versucht, den Text in *values-en(strings.xml)* zu finden. Gelingt dies nicht, findet *values(strings.xml)* Verwendung.

5.4 Oberflächenbeschreibungen

Eine Android-App beschreibt ihre Benutzeroberflächen mittels XML-basierter Layoutdateien. Dieser werden zur Laufzeit der App zu Objektbäumen *aufgeblasen*. Alle Bedienelemente werden in einen Container gepackt.

Oberflächenbeschreibungen werden im Ordner *layout*, einem Unterverzeichnis von *res*, gespeichert. Die XML-Datei bildet die Hierarchie der Benutzeroberfläche ab.

```
/* mainlayout.xml Beispiel */

2 <?xml version="1.0" encoding="utf-8"?>
<LinearLayout>
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >

    10 <Button
        android:id="@+id/buttonOK"
        android:text="R.string.OK"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
14 </LinearLayout>
```

Listing 5.3: Das Listing zeigt ein Beispiel zur Oberflächenbeschreibung einer Layout Datei

Um eine Referenz auf die Schaltflächen ermitteln zu können, wird ein Name definiert, beispielweise *buttonOK* aus dem Listing 5.3. Dieser wird wieder als Konstante in der Klasse *R* aufgenommen. Die Angabe `android:id="@+if/` sorgt also dafür, dass ein Bedienelement ein Bezeichner zugewiesen wird, auf den mittels *R.id* zugegriffen werden kann.

Übersicht vorhandener Layouts:

⁵<http://developer.android.com/training/basics/supporting-devices/languages.html>

⁶<http://developer.android.com/guide/topics/resources/localization.html>

Name	Beschreibung
FrameLayout	Anzeigen einer einzigen View
LinearLayout	Horizontale oder vertikale Ausrichtung
TableLayout	Jedes Oberflächenelement wird in einem Feld einer Tabelle dargestellt
AbsoluteLayout	Oberflächenelemente werden absolut zum Bildschirmrand ausgerichtet
RelativeLayout	Elemente werden in relation zu sich, anderen Views oder Bildschirmrand ausgerichtet
ListView	Listendarstellung von Oberflächenelementen
Gallery	Stellt eine Liste von Bildern horizontal dar
GridView	Stellt eine Oberfläche in Gitterstruktur dar
TabHost	Layout für Verwendung von Tab- und Reiter-Strukturen
MapView	Stellt eine Google-Maps-Karte dar
WebView	Dienst zur Darstellung von Internetseiten innerhalb der App

5.5 Activities

Jeder Activity ist eine Benutzeroberfläche⁷, also ein Baum bestehend aus *Views*⁸ und *ViewGroups*⁹, zugeordnet. Activities bilden demnach die vom Anwender wahrgenommenen Bausteine einer App. Sie können sich gegenseitig aufrufen und somit innerhalb einer App eine Vorwärtsnavigation realisieren. Da das System Activities auf einem Stack, also einem Stapel, ablegt, muss man sich als Entwickler nicht darum kümmern, von wem eine Activity aufgerufen wurde. Drückt der Benutzer den *Zurück-Knopf*, wird automatisch die zuvor angezeigte Activity reaktiviert. Android fördert die saubere Strukturierung einer App durch Activities. Jeder Activity steht ein Fenster für ihre Bedienelemente zur Verfügung.

Jede Activity besitzt einen Lebenszyklus, siehe Abbildung 5.3 Die Methode *onCreate()* wird von praktisch jeder Activity überschrieben. Android ruft sie während der Initialisierungsphase einer Aktivität auf. Sie erledigt normalerweise folgende Aufgaben:

1. Aufrufen der gleichnamigen Elternmethode
2. Initialisieren von Instanzvariablen
3. Setzen der Benutzeroberfläche
4. Wiederherstellen eines gespeicherten Zustands

Das setzen der Benutzeroberfläche erfolgt durch die Anweisung *setContentView()*. Der übergebene Parameter, zum Beispiel R.layout.main, referenziert das entsprechende Layout. Wiederherstellen von gespeicherten Zuständen wird genutzt, wenn sich die Ausrichtung des Handy

⁷<http://developer.android.com/guide/topics/ui/overview.html>

⁸<http://developer.android.com/reference/android/view/View.html>

⁹<http://developer.android.com/reference/android/view/ViewGroup.html>

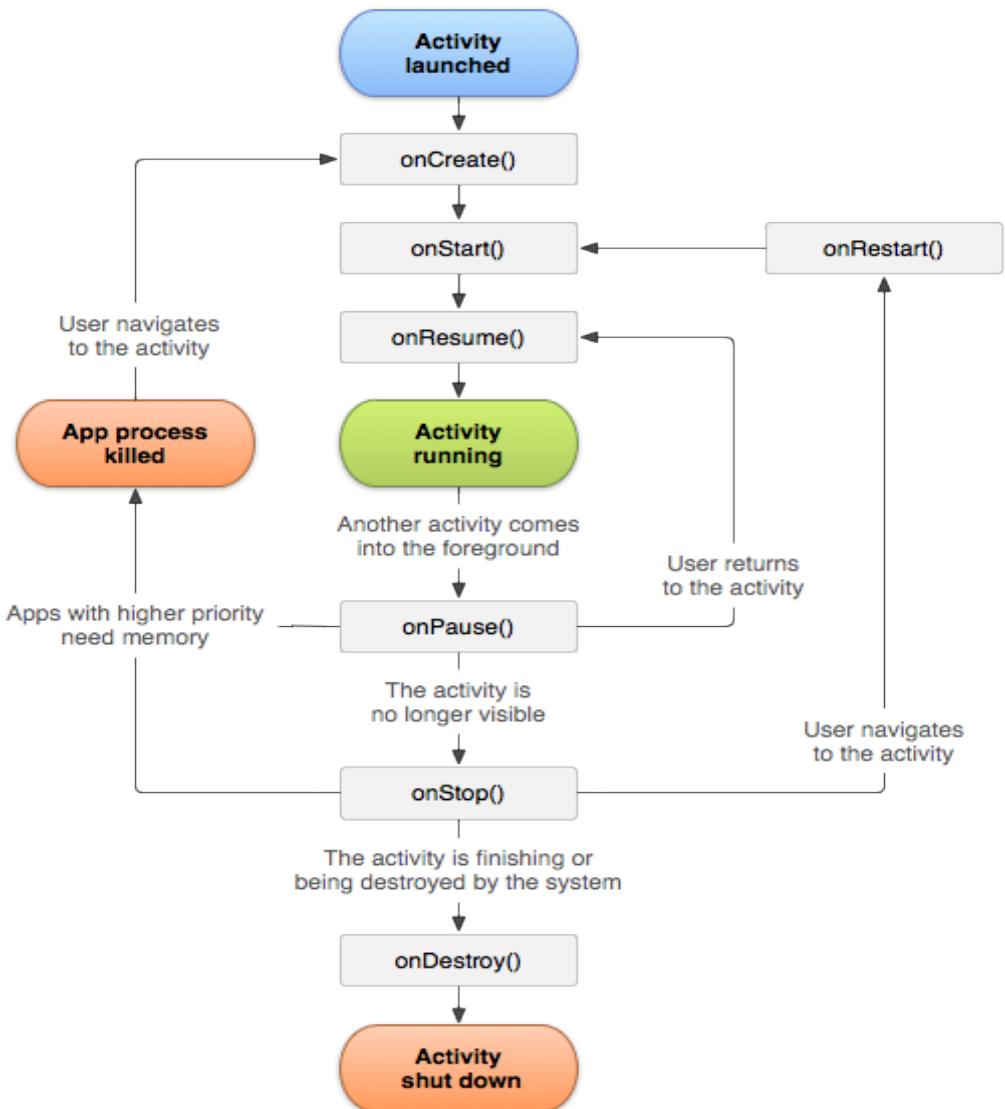


Abbildung 5.4: Lebenszyklus von Activity; <http://developer.android.com/guide/components/activities.html>

ändert, wird die `onCreate()` Methode erneut aufgerufen und die App neugestartet. Um nicht gespeicherte Daten zu verlieren, können diese als Zustände gespeichert werden und wiederhergestellt werden. Die `onRestart()` Methode wird aufgerufen, nachdem die Aktivität gestoppt wurde, gerade bevor sie wieder in Betrieb genommen wird. `onStart()` wird aufgerufen, kurz bevor die Aktivität für den Benutzer sichtbar wird. Die `onResume()` Anweisung wird aufgerufen, kurz bevor die Aktivität mit dem Benutzer beginnt zu interagieren. An diesem Punkt ist die Aktivität an der Oberseite des Aktivitätsstapel und bieten Benutzereingaben an. `onPause()` wird aufgerufen, wenn das System kurz vor dem Start eine andere Tätigkeit wieder aufnimmt. Die Methode wird verwendet um nicht gespeicherte Änderungen persistent zu speichern. `onStop()` Wird aufgerufen, wenn die Aktivität nicht mehr sichtbar für den Benutzer ist. Dies kann passieren, weil es zerstört wurde oder weil eine andere Aktivität (entweder eine bestehende oder eine neue) wieder aufgenommen wurde und die Activity bedeckt. Zum Beenden der App, wird die `onDestroy()` Methode aufgerufen, bevor die Aktivität zerstört wird. Dies ist der letzte

Aufruf, die die Aktivität erhält. Es kann passieren, weil entweder die Aktivität beenden ist (Finish() aufgerufen wurde), oder weil das System vorübergehend Platz zu sparen möchte.

Kapitel 6

Android-App: HMI

Im Rahmen der Veranstaltung entschied ich mich für ein Android Projekt, dass anhand von der Android-Plattform, *Human Machine Interface* Möglichkeiten aufzeigen soll. Diese sollten untersucht und unter kleinen Beispielen dargestellt werden.

6.1 Main Activity

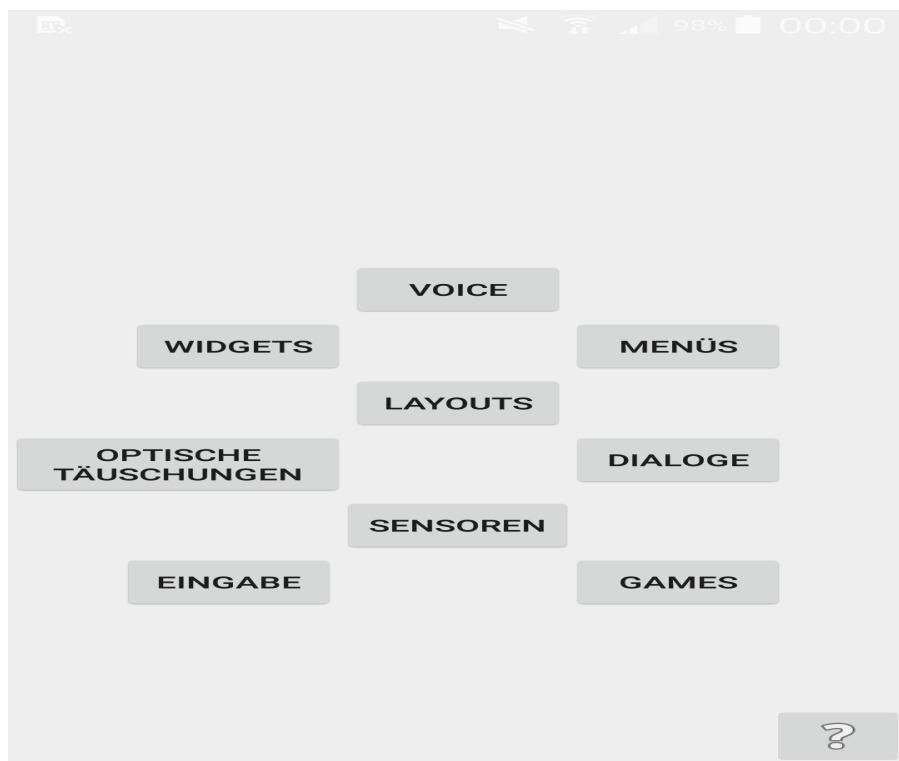


Abbildung 6.1: Hauptbildschirm der app; activity_main.xml

Das Hauptmenü der App präsentiert sich schlicht und aufgeräumt, indem es nur über Buttons verfügt, mit denen sich der Nutzer durch die verschiedenen Bereiche der App navigiert. Wenn der Nutzer auf eine Schaltfläche klickt, wird mittels Intent, zu deutsch etwa *Absicht*, eine neue Activity gestartet, die einen neuen Bildschirm aufbaut, siehe Listing 6.1.

```

1 /* Android neue Activity starten Beispiel */
2 bVoice = (Button) findViewById(R.id.buttonVoice);
3     bVoice.setOnClickListener(new View.OnClickListener() {
4         @Override
5             public void onClick(View v) {
6                 startActivity(new Intent(MainActivity.this, VoiceActivity.class));
7             }
8 });

```

Listing 6.1: Das Listing zeigt wie mit einen Intent eine neue Activity gestartet wird. Ein Intent erwartet als ersten Parameter die Start Activity und als zweiten Parameter die Ziel Activity

6.2 Voice

Schon seit Android 1.6 enthält die Plattform die Sprachsynthesekomponente *Pico*. Diese kann Texte, abhängig der installierten Sprachpakete, vorlesen. Seit Navigationssystemen weiß man, wie praktisch eine Sprachausgabe ist. Aber die Einsatzmöglichkeiten sind weitaus vielschichtiger. Es lassen sich auch E-Mails vorlesen oder auf Knopfdruck die aktuelle Uhrzeit ansagen, siehe dazu Abbildung 6.2.

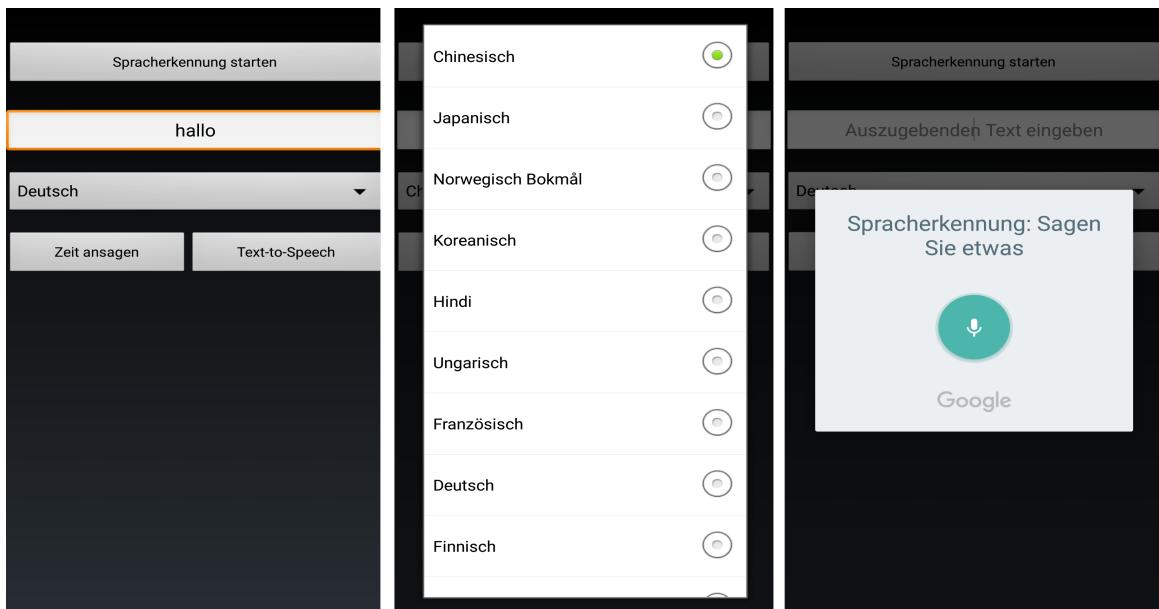


Abbildung 6.2: VoiceActivity mit den Funktionen gesprochenes zu erkennen, die Zeit anzusagen und geschriebenes aus einer EditText-Box in einer beliebigen installierten Sprache zu Sprechen (Text-to-Speech); voice_layout.xml

Die Plattform kann nicht nur Sprache synthetisieren, sondern auch erkennen. Eine App könnte also ausschließlich in gesprochener Sprache mit dem Anwender kommunizieren. Beispielseitweise gibt es das automatische Wählen eines Kontakts nach Nennung des Namens oder die Steuerung des Smartphones durch einfache mündliche Befehle.

6.3 Widgets

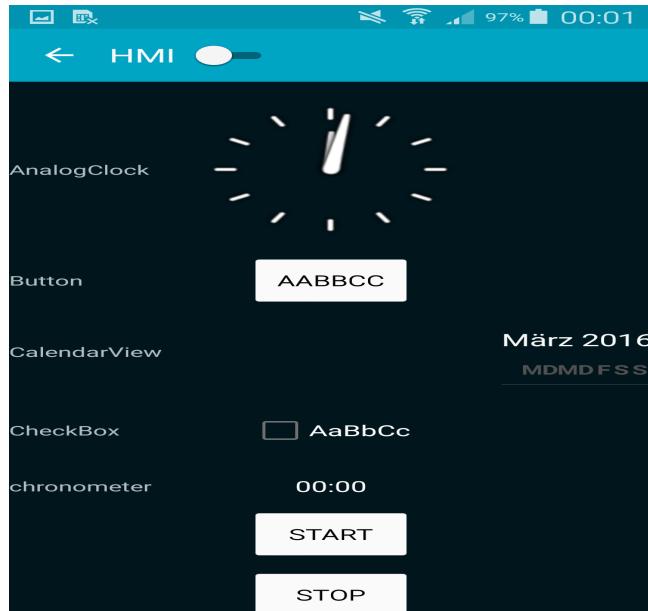


Abbildung 6.3: VoiceActivity mit den Funktionen gesprochenes zu erkennen, die Zeit anzusagen und geschriebenes aus einer EditText-Box in einer beliebigen installierten Sprache zu Sprechen (Text-to-Speech); voice_layout.xml

Als Entwickler kann man auf eine Reihe vorhandener Views zurückgreifen. Über ihre Attribute kann man sie teilweise in Form, Farbe und Größe verändern.

Übersicht über die gebräuchlichsten Views:

Name	Beschreibung
<TextView>	Einfache Textausgabe
<Button>	Schaltfläche
<ImageButton>	Grafische Schaltfläche
<EditText>	Formularelement zur Texteingabe
<CheckBox>	Ankreuzfeld
<RadioGroup> <RadioButton>	Auswahlschalter
<Spinner>	Auswahlliste
<DatePicker>	Auswahl von Datum
<TimePicker>	Auswahl von Uhrzeit
<AnalogClock>	Analoge Anzeige der Uhrzeit
<DigitalClock>	Digitale Anzeige der Uhrzeit

6.4 Menüs

Menüs präsentieren dem Benutzer Funktionen bzw. Aktionen, die er zu einem bestimmten Zeitpunkt ausführen kann. Klassische Desktop-Systeme kennen neben einer Menüleiste sogenannte



Abbildung 6.4: MenuActivity mit Menüs und Kontextmenüs; menu_layout.xml

Kontextmenüs, die nach dem Anklicken eines Objekts mit der rechten Maustaste geöffnet werden.

In Android gewährt eine virtuelle Taste in der System Bar Zugriff auf das Optionsmenü. Siehe Abbildung

6.4.1 Kontextmenüs

Android kennt das von Desktop-Systemen bekannte Konzept der Kontextmenüs. Anstelle der rechten Maustaste löst das lange Antippen (Tippen und Halten) eines Elements das Menü aus.

6.5 Layouts

Die Layout Klassen dienen zur Anordnung von Steuerelementen. Die verschiedenen Layouts können beliebig geschachtelt werden. Beim erzeugen der Benutzerschnittstelle wird zur Laufzeit aus der Layout XML Datei eine Java Klasse erstellt. Die Benutzeroberfläche ist die wichtigste visuelle Kommunikation mit den Nutzern. Jede Funktion sollte leicht zugänglich und intuitiv erfassbar sein.

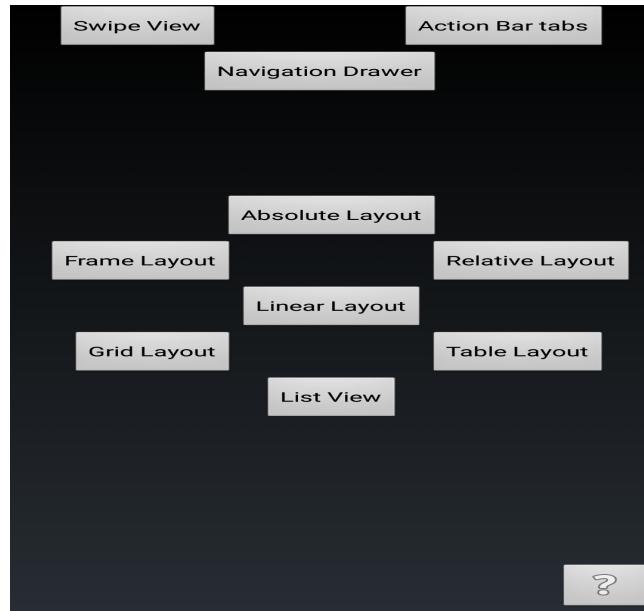


Abbildung 6.5: LayoutActivity mit Buttons zur Navigation zu unterschiedlichen Layout Beispielen; layout_layout.xml

6.5.1 Absolute Layout

Definiert genaue Standorte (x/y Koordinaten) für die Steuerelemente. Solche Layouts sind weniger flexibel und schwer zu warten und an verschiedene Bildschirmauflösungen und Größen anzupassen.



Abbildung 6.6: AbsoluteActivity Oberfläche; absolute_layout.xml

6.5.2 Frame Layout

Die eingebetteten Elemente werden übereinander angezeigt. Im Programmcode können dann verschiedene Steuerelemente abhängig von der Benutzereingabe angezeigt oder ausgeblendet werden.

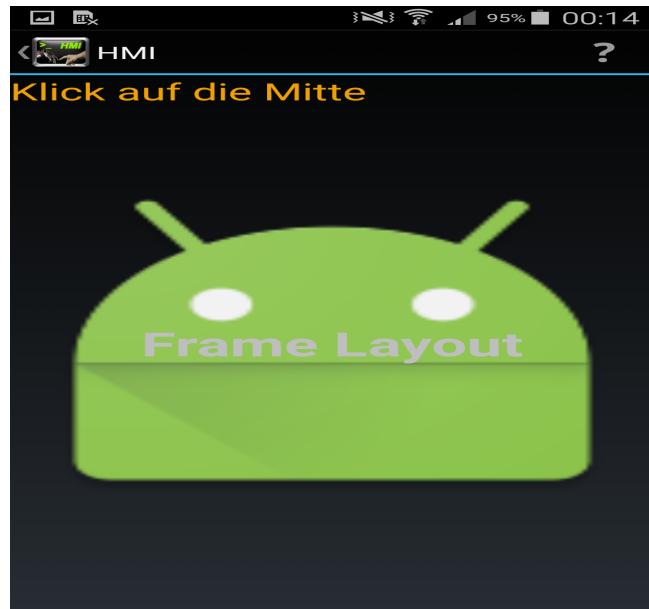


Abbildung 6.7: FrameActivity Oberfläche; frame_layout.xml

6.5.3 Relative Layout



Abbildung 6.8: RelativeActivity Oberfläche; relative_layout.xml

Positioniert die eingebetteten Steuerelemente relativ zueinander (Bildschirmrand, zu sich

selbst, View). Die Relation zueinander wird mit folgenden Parametern beeinflusst: android:layout_above, below, toLeftOf, toRightOf, alignLeft, alignTop, alignRight, alignBottom.

6.5.4 Linear Layout

Ordnet die eingebetteten Steuerelemente entweder horizontal oder vertikal an. Die Ausrichtung wird über den XML Parameter android:orientation festgelegt.

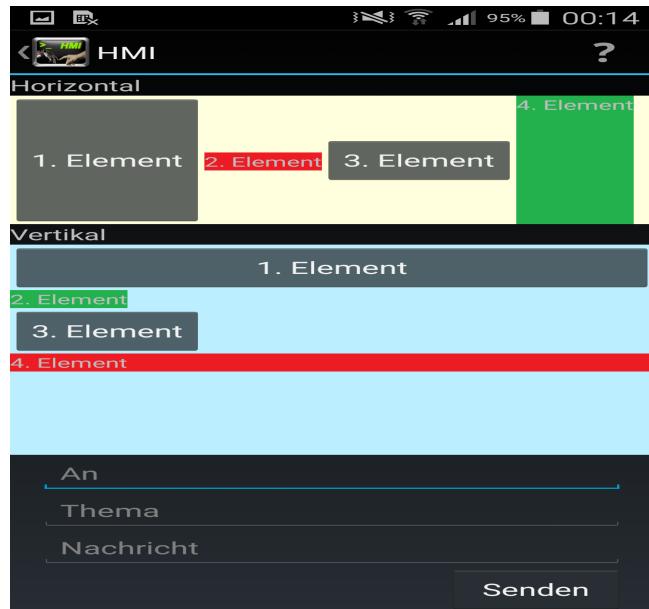


Abbildung 6.9: LinearActivity Oberfläche; linear_layout.xml

6.5.5 Grid Layout

Die eingebetteten Elemente werden auf einem Raster positioniert, wobei sich ein Steuerelement über mehrere Zeilen und Spalten erstrecken kann.

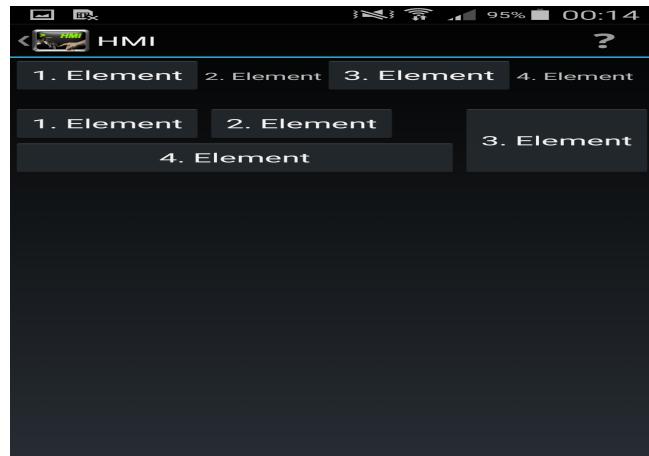


Abbildung 6.10: GridActivity Oberfläche; grid_layout.xml

6.5.6 Table Layout

Ähnelt einer Tabelle und gruppiert die eingebetteten Steuerelemente innerhalb der Elemente vom Typ TableRow. Mittels den Parametern android:layout_width und android:layout_weight kann die Größe der Steuerelemente relativ zueinander festgelegt werden.



Abbildung 6.11: TableActivity Oberfläche; table_layout.xml

6.5.7 List View

Mit einem ListView lassen sich große und komplexe Listen sehr effizient und benutzerfreundlich darstellen. Die Listen sind scrollbar und registrieren Benutzereingaben. Die Listeneinträge werden automatisch in die Liste mit einem Adapter eingefügt. Die Inhalte werden aus einer Quelle wie einem Array oder einer Datenbank bezogen.



Abbildung 6.12: ListView Oberfläche mit 2 Listen, die obere beinhaltet nur Text, die untere besitzt darüber hinaus Grafiken; list_layout.xml

6.5.8 Swipe View

Mit Android 5.0 eingeführtes Material Theme Design. Bietet die Funktion, mittels Swipe Gestik, also dem wischen von links nach rechts beziehungsweise rechts nach links zwischen den Bildschirminhalten zu wechseln.

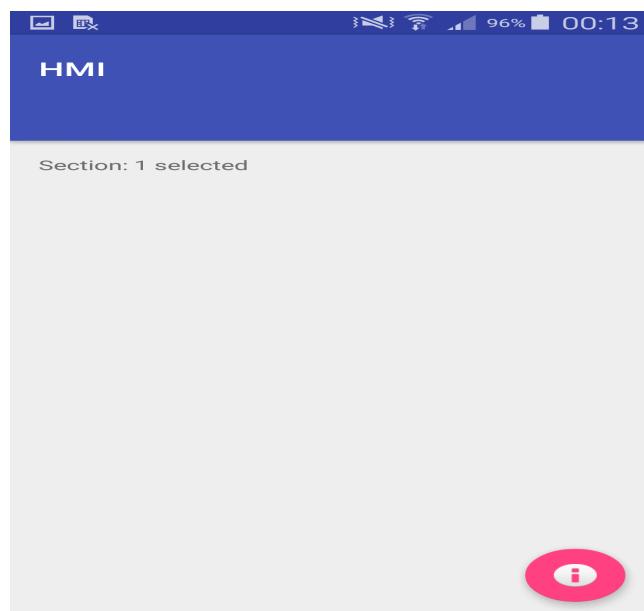


Abbildung 6.13: SwipeviewActivity Oberfläche in Android 5.0 Material Design; tab_layout.xml

6.5.9 Action Bar tabs

Ähnlich zu dem Swipe View wird mittels Swipe Gestik der Bildschirm verändert, hierbei wird aber in der Action Bar eine Tab Ansicht realisiert, die die Verfügbaren Bildschirme anzeigt und ebenso den zurzeit befindlichen Bildschirm.

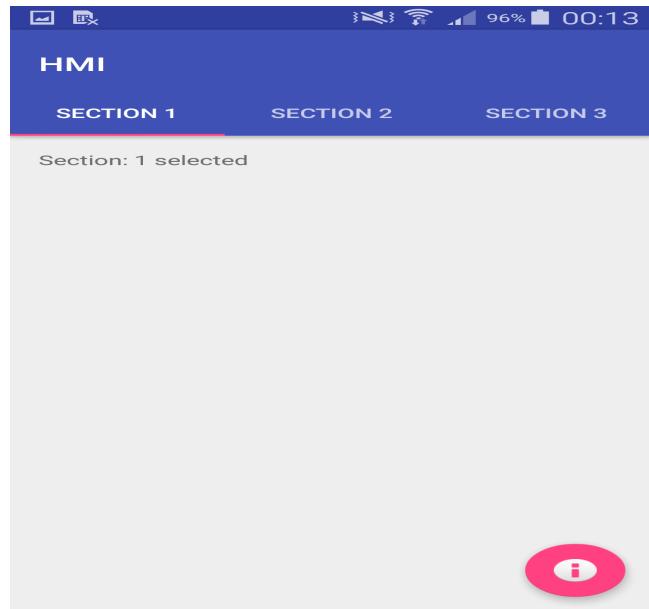


Abbildung 6.14: ActionBarTabsActivity Oberfläche in Android 5.0 Material Design; actionBar_layout.xml

6.5.10 Navigation Drawer

Bietet die Funktionalität in der Action Bar auf ein Symbol mit drei waagerechten Strichen zu klicken um ein seitliches Menü zu öffnen oder per Swipe Geste vom linken Bildschirmrand zur Mitte aufzurufen. Das Menü kann mit verschiedenen Funktionen ausgestattet sein, mit Grafiken oder Text.

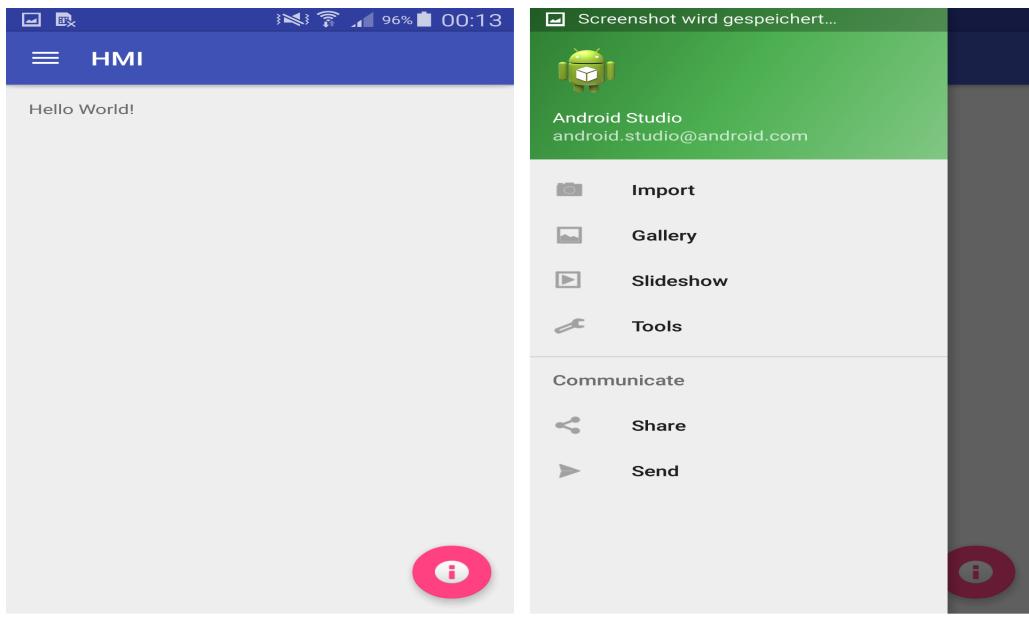


Abbildung 6.15: NavigationDrawerActivity Oberfläche in Android 5.0 Material Design; navigationdrawer_layout_layout.xml

6.6 Optische Täuschungen

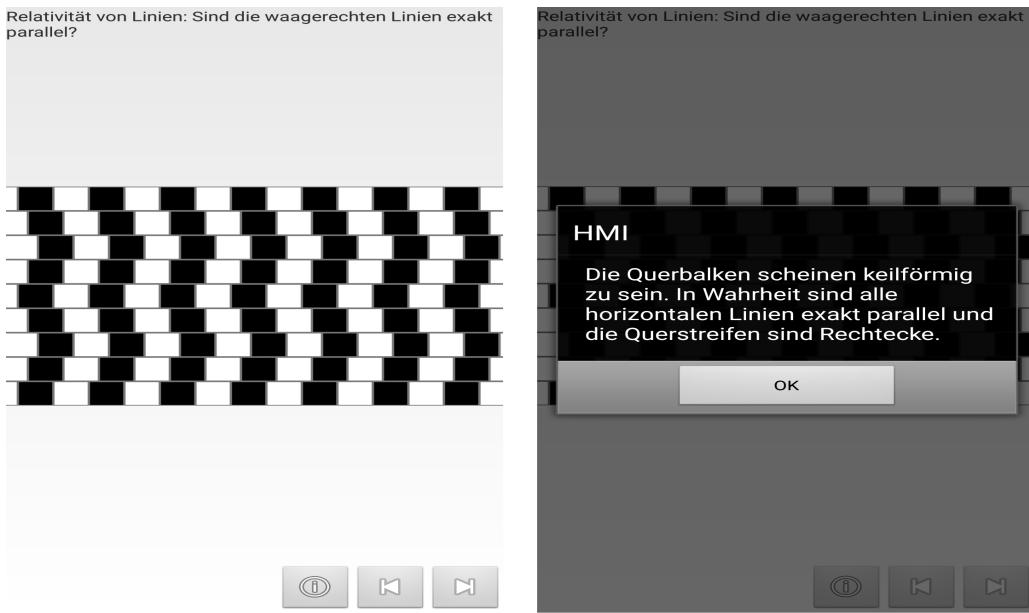


Abbildung 6.16: OptischeTaeuschungActivity; optische_taeuschung_layout.xml

In dieser Activity wird mit Hilfe eines ImageSwitcher Objekts verschiedene Optische Illusionen angezeigt. Am unteren Bildschirmrand befinden sich Buttons zum vor- und zurück klicken sowie ein Informationsschalter um Informationen zum aktuell angezeigten Bild zu erhalten.

6.7 Dialoge

Wenn der Benutzer ein Dialog anklickt, öffnet sich ein kleines Fenster mit einem Titel und einer Message oder optional mit Schaltflächen wie Eingabefeldern usw. Solche modalen Dialoge werden über der aktuellen Activity angezeigt. Sie erhalten den Fokus und nehmen alle Eingaben entgegen, der Anwender wird also in seiner Tätigkeit unterbrochen."Toasts" sind Kurznachrichten für den Benutzer. Diese Kurznachricht schwiebt quasi über der normalen Applikations-View und kann keinen Fokus erlangen. Bei der Anzeige eines Toasts ist es völlig egal, was der Benutzer gerade macht, der Toast wird in jedem Fall angezeigt. Die Idee dahinter ist, den Benutzer so wenig wie möglich zu stören, ihm aber die im Toast enthaltene Information jedenfalls zu präsentieren.

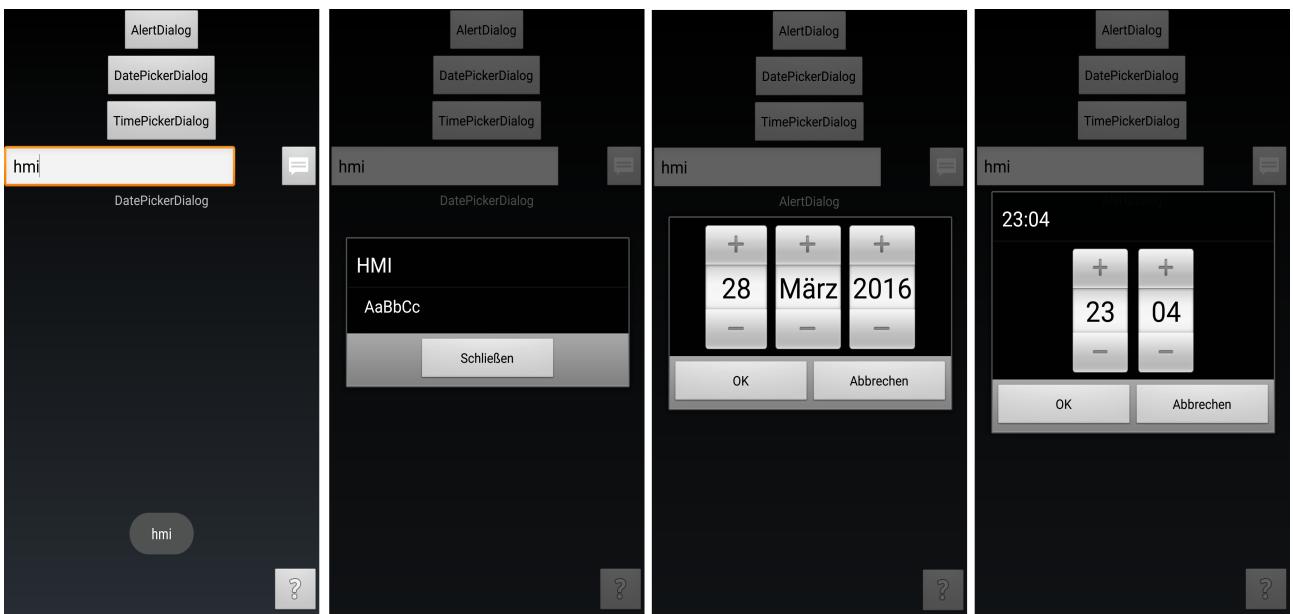


Abbildung 6.17: DialogActivity, im linken Teil sieht man die Toast-Nachricht mit dem Inhalt aus dem EditText Feld. Rechts daneben sieht an einen AlertDialog, auf den der Nutzer reagieren muss um zur normalen App-Ansicht zurückzukehren. Im vorletzten Bild ist ein DatePickerDialog gezeigt, der den Nutzer zum Auswählen eines Datum bittet. Das letzte Bild zeigt ein TimePickerDialog der den Anwender um eine Uhrzeit Eingabe bittet; dialog_layout.xml

6.8 Sensoren

Android-Geräte enthalten zahlreiche Sensoren. Diese lassen sich mit geringem Aufwand in eigenen Apps nutzen. Beispiele: Abschalten des Bildschirmes beim Telefonieren, Darstellung auf dem Bildschirm passt sich der Ausrichtung an, Spiele reagieren auf Bewegungsänderungen, Karten-Apps erkennen automatisch den Standort uvm. Zusätzlich lassen sich die Sensoren auch als Eingabeschnittstelle zur Kommunikation nutzen, beispielweise das Gerät schütteln um Eingabe zu korrigieren.



Abbildung 6.18: SensorActivity, menü zur Auswahl einiger Sensoren. Am unteren Rand werden alle verfügbaren Sensoren samt Daten angezeigt; sensor_layout.xml

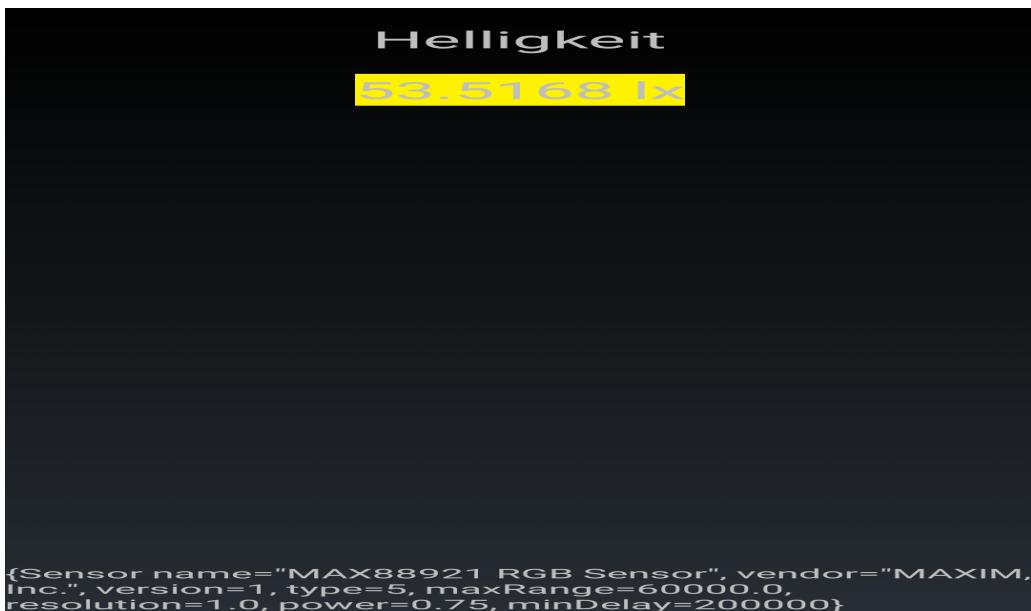


Abbildung 6.19: HelligkeitActivity, zeigt über den Smartphone Sensor die Helligkeit der Umgebung an; helligkeit_layout.xml



Abbildung 6.20: BeschleunigungActivity, zeigt auf dreidimensionalen Achsen: x, y, z die Beschleunigung an. Beim Hochhalten bzw liegend fällt die Erdanziehungskraft auf den Achsen auf; beschleunigung_layout.xml



Abbildung 6.21: InputActivity, übersicht der Eingabe-Methoden und verzweigung per Buttons; input_layout.xml

6.9 Eingabe

6.9.1 Button

Klassische Buttons dienen als gewöhnliche Schaltflächen mit Beschriftung. Ein normaler Button besitzt ein Label zur Beschriftung, was dieser bei einem klick realisiert. ImageButtons zeigen ein Bild anstelle einer Beschriftung an. Beide Buttons lassen sich auch kombinieren und somit

ein Button mit einer Beschreibung und einem Icon erstellen. Buttons können als Hintergrund eine Grafik enthalten, z.B. eine png-Grafik. Platzsparender aber aufwendiger zu erstellen sind xml-Hintergründe die sich als Buttons nutzen lassen. Dafür skalieren diese für verschiedene Geräte besser. Buttons lassen sich auch rotieren sowie drehen und somit wie Analoge Drehregler erscheinen lassen. Zur Reaktion auf Benutzereingaben wird einem Steuerelement eine Referenz

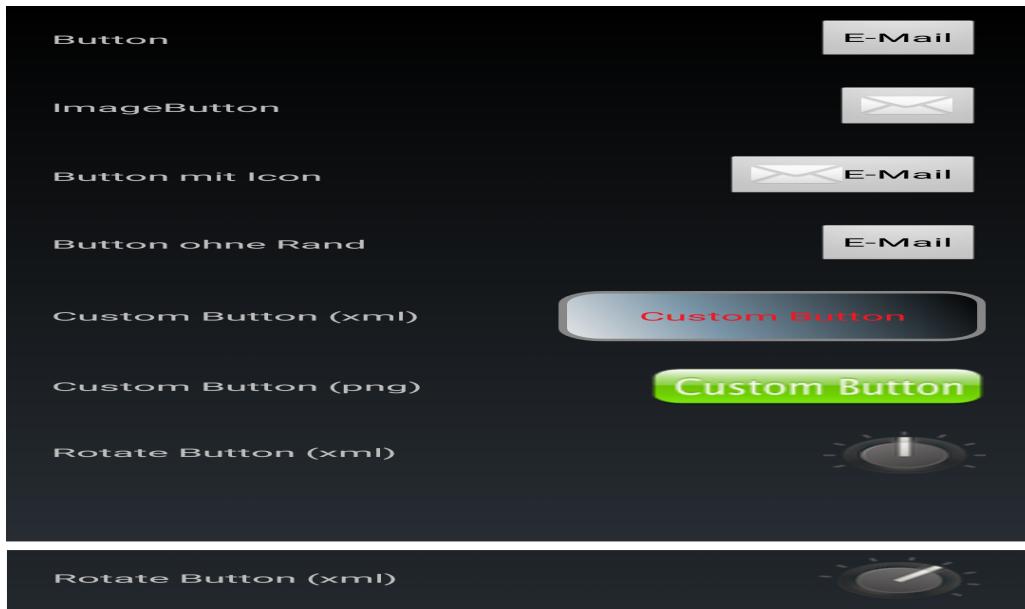


Abbildung 6.22: ButtonActivity, übersicht der verschiedenen Button Gestaltungsmöglichkeiten. Der untere Rotate Button dreht sich wie ein Drehregler beim anklicken; button_layout.xml

einer Listener-Schnittstelle übergeben. Beispielsweise *View.OnClickListener*

```
/* Listener als anonyme Klasse Beispiel */

Button b = (Button) findViewById(R.id.button1);
4 b.setOnClickListener(new OnClickListener() {
    @Override
    public void onClick(View v) {
        //Button button1 wurde angeklickt
8    }
});
```

Listing 6.2: Das Listing zeigt einen Ausschnitt wie Benutzereingaben von Buttons registriert werden

6.9.2 Textfelder

Ein Textfeld ermöglicht es dem Benutzer Text in der Anwendung zu geben. Es kann entweder eine Zeile oder mehreren Zeilen lang sein. Sobald ein Textfeld berührt wird zeigt sich automatisch der Cursor und die Tastatur. Neben der Eingabe, ermöglichen Textfelder eine Vielzahl von anderen Aktivitäten, wie zum Beispiel Textauswahl (Ausschneiden, Kopieren, Einfügen)

und Daten-Look-up über Auto-Vervollständigung, ein Name oder bereits getipptes Wort wird über der Tastatur empfohlen.

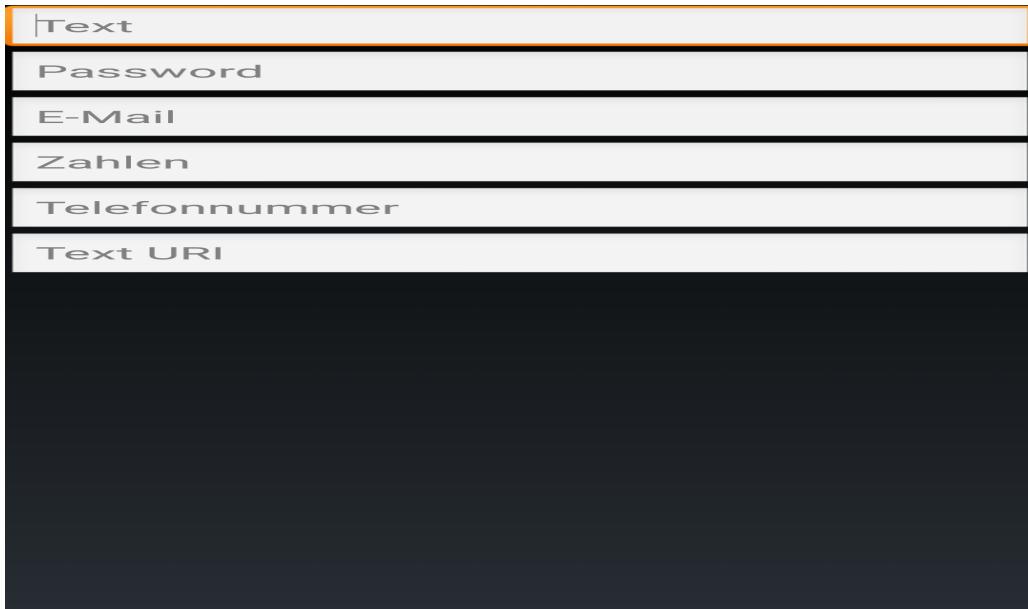


Abbildung 6.23: EdittextActivity, übersicht der Eingabe-Methoden für verschiedene Textfelder; edittext_layout.xml

Textfelder können unterschiedliche Eingabetypen aufweisen, wie Nummer, Datum, Passwort oder E-Mail-Adresse. Der Typ bestimmt, welche Art von Zeichen in das Feld zulässig sind und bestimmt welche Zeichen auf der virtuellen Tastatur angezeigt werden.

Es gibt mehrere verschiedene Eingabearten für verschiedene Situationen. Hier sind einige der häufigsten Werte für *android:inputType*:

- text: Normale Text Tastatur
- textEmailAddress: Normale Text Tastatur mit dem @ Symbol
- textUri: Normale Text Tastatur mit / Zeichen für Verzeichnisse. URI - Uniform Resource Identifier
- number: Grundzifferntastatur (Zahlen von 0 - 9)
- phone: Telefon-Stil Tastatur mit Wählertasten und Schriftzeichen um Kontakte unter deren Namen zu finden
- password: Normale Text Tastatur ohne Wortvorschläge und jedes eingegebene Zeichen wird verdeckt

6.9.3 Checkbox

Checkboxen sind eine Gruppe von ankreuzbaren Kontrollfeldern, bei denen der Anwender keine, eine oder mehrere auswählen kann. Radio-Buttons sind eine Gruppe von beschrifteten Knöpfen,

von denen der Anwender einen auswählen kann. Es kann immer nur einer ausgewählt sein. Toggle-Buttons erlauben den Nutzer Einstellungen zwischen zwei Zuständen (Ein und Aus) zu schalten. Spinner bieten eine schnelle Möglichkeit einen Wert aus einer Menge zu wählen. In der Grundeinstellung zeigt ein Spinner den aktuell ausgewählten Wert. Beim Berühren zeigt sich ein Dropdown-Menü mit allen anderen verfügbaren Werte, aus denen der Benutzer eine neue auswählen kann.

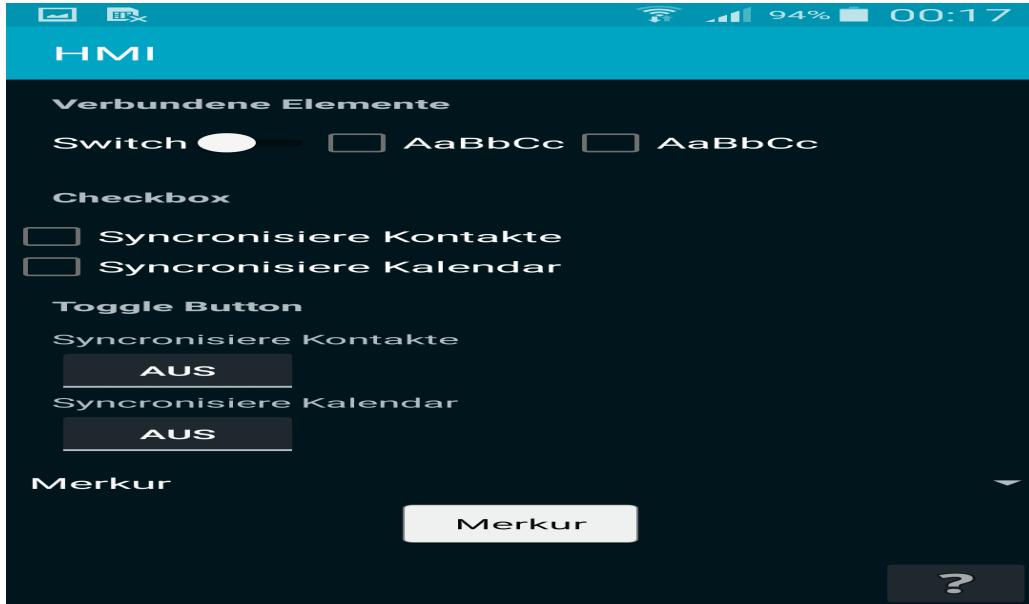


Abbildung 6.24: CheckboxActivity, übersicht von verschiedenen Kontrollfeldern; checkbox_layout.xml

6.9.4 Radio Button

Radio-Buttons sind eine Gruppe von beschrifteten Knöpfen, von denen der Anwender einen auswählen kann. Es kann immer nur einer der Radio-Buttons ausgewählt sein. Oft wird ein Initialzustand programmiert, damit keine ungültige Zustände Zustandekommen und somit ein fehlerhaftes Verhalten der App provoziert wird.

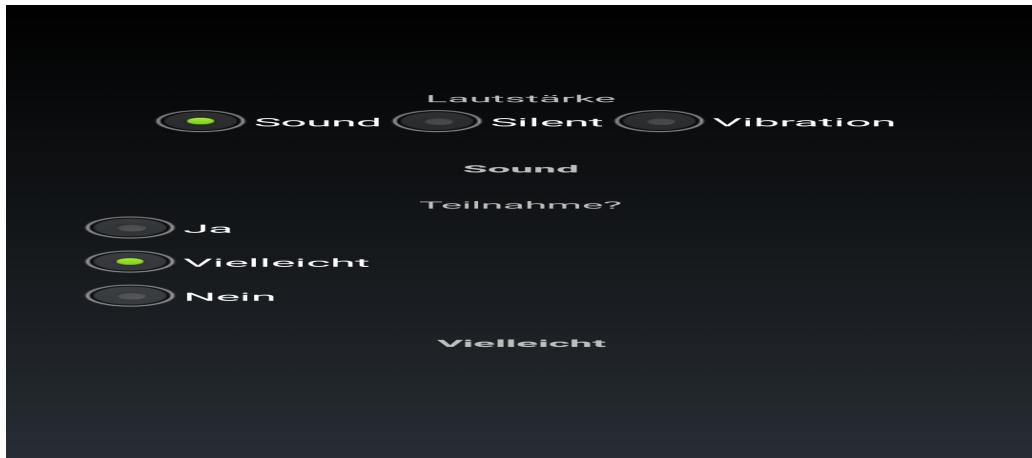


Abbildung 6.25: RadioButtonActivity, gruppierungen von verschiedenen Radio-Buttons zur Auswahl; radiobutton_layout.xml

6.10 Games

Ob in der S-Bahn, beim warten auf der nächsten Verkehrsbindung, im Wohnzimmer, oder im Wartezimmer beim Arzt - überall wird mit dem Smartphone gespielt. Besonders das Smartphone macht es interessant, denn es bietet zahlreiche Sensoren die in Kombination mit Spielen viel Spaß bereiten.



Abbildung 6.26: GameActivity, menü zum starten der Spiele; game_layout.xml

6.10.1 Spiel mit Zeit

Das Spiel zeigt eine zufällig gewählte Zeit im Intervall von 1 und 10 Sekunden an. Der Benutzer muss zweimal auf den Button drücken und versuchen die Zeit rechtzeitig zu stoppen: -Der erste Klick startet den Zähler. -Der zweite Klick beendet den Zähler. Das Spiel gibt die Absolute

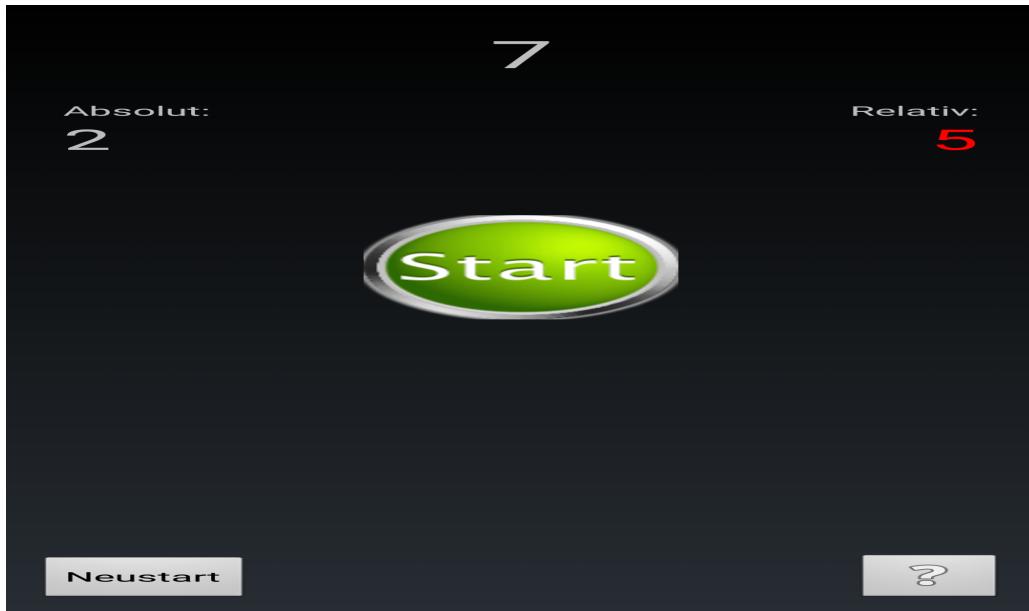


Abbildung 6.27: GameTimeActivity, spiel bei dem es um das Zeitgefühl geht; game_time_layout.xml

und Relative differenz verglichen mit der zufällig gewählten Zeit. Neuer Versuch beim klicken auf Start mit der selben Zeit oder Restart für eine neue Zeit.

6.10.2 Spiel mit Punkten

Game with Dots: Verschiedene Objekte erscheinen an zufälligen Stellen und dein Ziel ist es, das richtige Objekt anzuklicken. Sieh am oberen Bildschirmrand, welches Objekt mit welcher Farbe gesucht wird! Obere Linke Ecke: Punkte. Obere Rechte Ecke: Runden. Obere Mitte: Hinweis. Untere Mitte: Countdown. Untere Rechte Ecke: Highscore. Um das Spiel zu erschweren wurden abhängig der Runde immer mehr falsche Objekte gezeichnet, die den Fokus vom gesuchten Objekt ablenken sollten. Leider funktionierte das Prinzip durch Änderungen der API in den neueren Android Versionen nicht mehr und nur noch das gesuchte Objekt wird gezeichnet.



Abbildung 6.28: GameDotsActivity, klicke auf das gesuchte Objekt, bevor die Zeit abläuft

6.10.3 Spiel mit Analoger Steuerung

Das Spiel erlaubt es ganz klassisch mit Buttons an den Bildschirmrändern die Android Figur zu steuern.

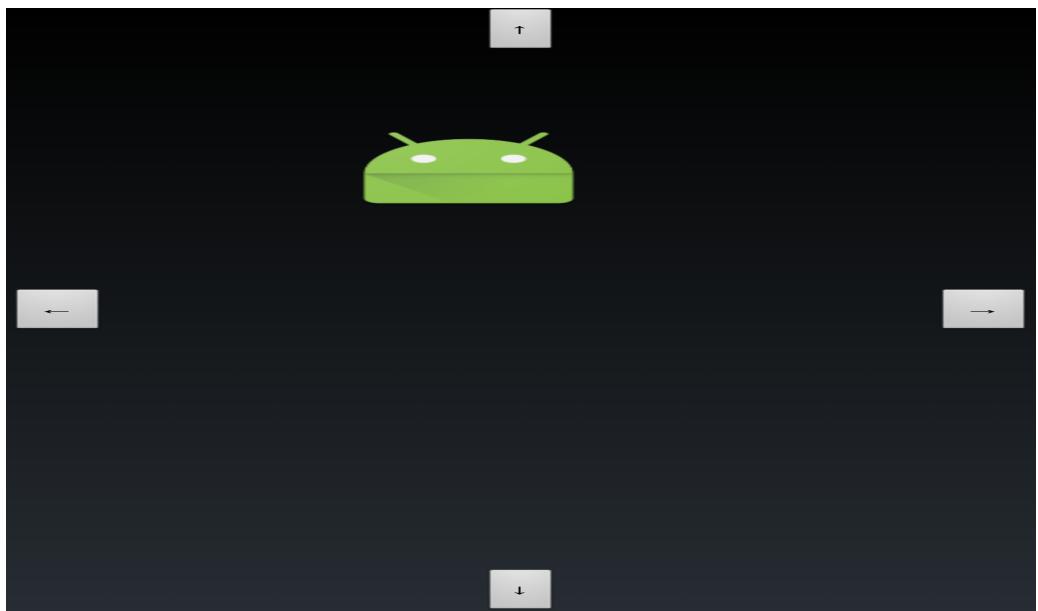


Abbildung 6.29: MainActivity, steuer die Android Figur mit Hilfe der Buttons

6.10.4 Spiel mit Touch Steuerung

Steuer das HMI Logo per Touchs auf dem Bildschirm, viele Spiele sind für Touchscreens entwickelt und kommen ausschließlich damit raus. Es gibt aber auch Nutzer, die die Steuerung mit Touch als nicht gesunde bzw. angenehme Eingabe halten.



Abbildung 6.30: MainActivity, steuer das HMI Logo mit Hilfe deiner Finger auf dem Touchscreen

6.10.5 Spiel mit Beschleunigungssensor

Als Eingabe Methode zur Steuerung der Android Figur wird das Gerät geneigt. Die Bewegung der Figur erfolgt auf den x- und y-Achsen des Gerätes. Wenn die Figur gegen die Wand läuft, prallt es an ihr ab. Die Eingabe mit dem Beschleunigungssensor ist eine tolle Erfahrung, die auch Geschick verlangt. Es ist teilweise schwer damit präzise zutreffen.

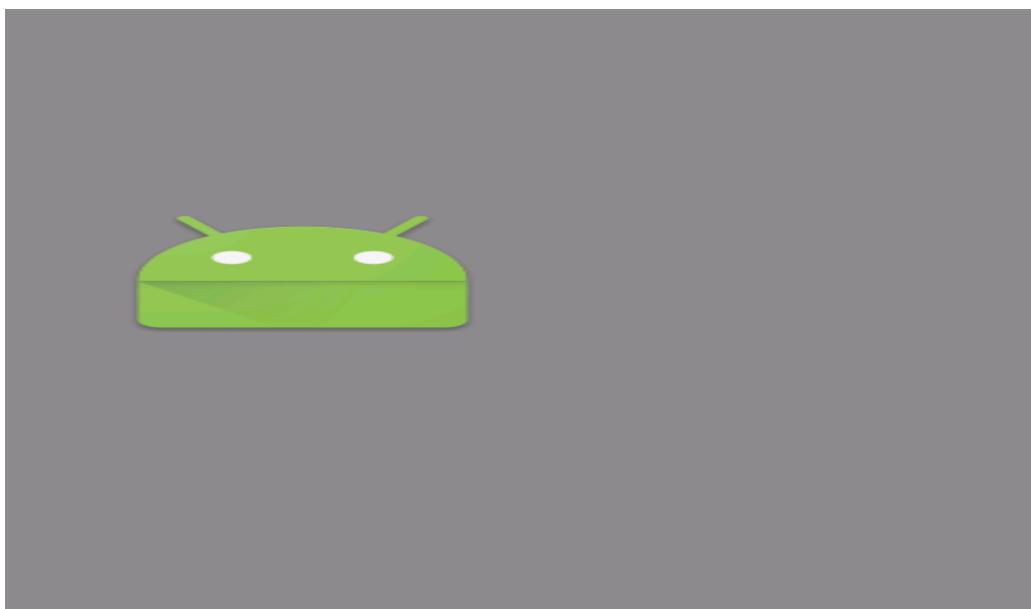


Abbildung 6.31: MainActivity, mit dem Beschleunigungssensor des Gerätes wird die Android Figur gesteuert

6.10.6 Kugel ins Loch

Die Spielsteuerung mit dem Beschleunigungs- oder Lagesensor funktioniert intuitiv und einfach. Der Sensor kann anhand der Schwerkraft messen, wie der Nutzer das Gerät gerade in der Hand hält. In diesem Spiel muss der Anwender eine Kugel ins Schwarze Loch versenken bevor die Zeit abläuft.

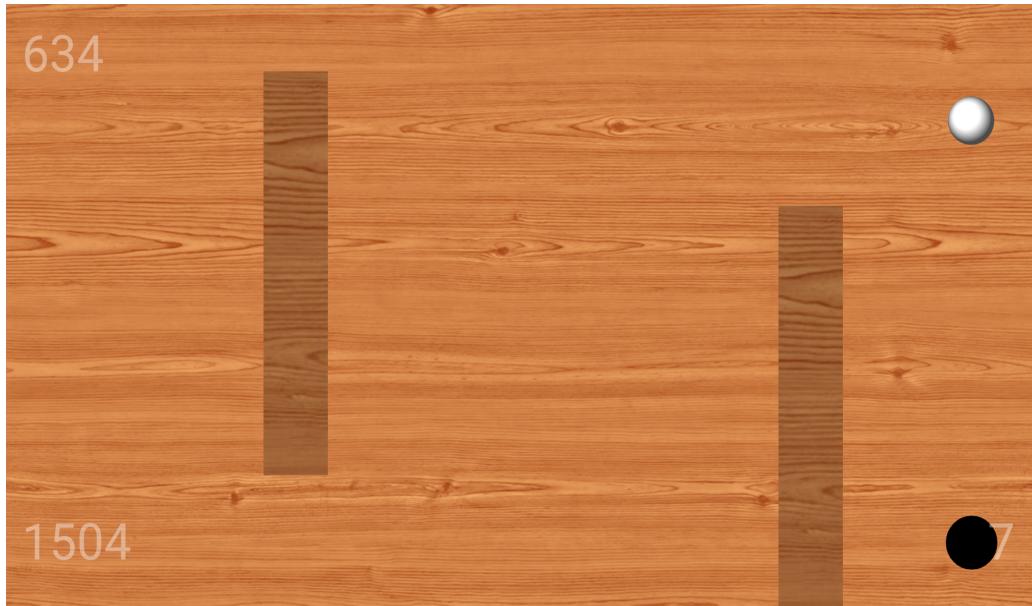


Abbildung 6.32: MainActivity, mit dem Beschleunigungssensor des Gerätes wird die Android Figur gesteuert

6.10.7 Ich packe meinen Koffer

Das Spiel besteht darin, vom "Einpacken eines Koffers" zu berichten und Gegenstände zu nennen, die bereits in den Koffer gepackt wurden. Eine zufällig gewählte Person beginnt das Spiel und legt einen beliebigen Gegenstand hinein. Im Uhrzeigersinn wird fortgefahrene. Der jeweils nächste Mitspieler wiederholt den gesamten Satz seines Vorgängers inklusive aller Gegenstände, die dieser genannt hat, und fügt einen eigenen Gegenstand hinzu. Reihum müssen die Spieler so alle bereits genannten Gegenstände in der richtigen Reihenfolge lückenlos aufzählen und am Ende der Liste einen weiteren, eigenen Gegenstand hinzufügen. Ein Spieler verliert, wenn er Gegenstände in ihrer Reihenfolge vertauscht oder weglässt.

Mit dieser Spieleanwendung sollte exemplarisch die Spracherkennung von Google genutzt werden. Auf die Spielidee sind wir in der Veranstaltung von HMI gekommen, als wir noch auf weitere Studenten warteten und Herr Deegener dieses Spiel mit uns spielte und zur anschließenden Übung eine Idee zur Umsetzung diskutierten.

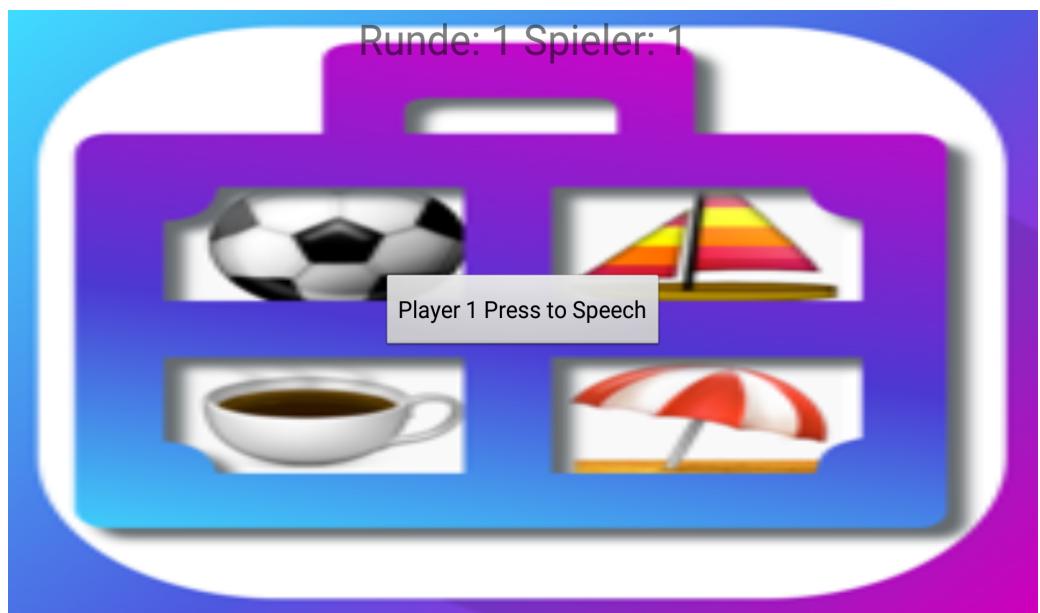


Abbildung 6.33: MainActivity, aus dem laufenden Spiel kann man entnehmen, dass es die erste Runde ist und Spieler 1 an der Reihe ist. Nach einem Klick auf den Button beginnt die "Voice Recognition", also die Spracherkennung

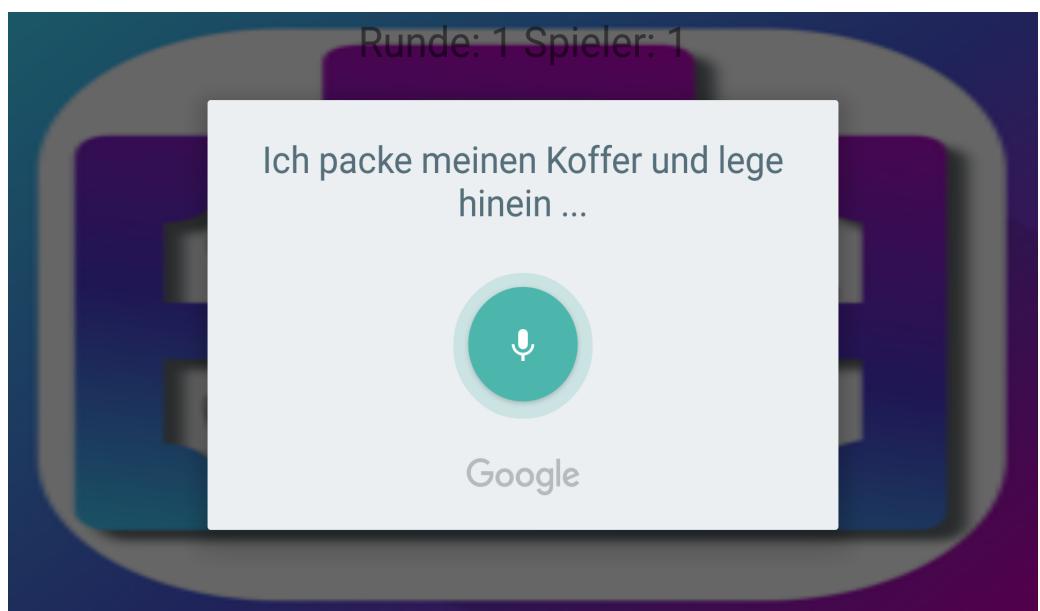


Abbildung 6.34: MainActivity, "Voice Recognition", die Spracherkennung fragt den Nutzer, was im Koffer bisher alles drinne steckt

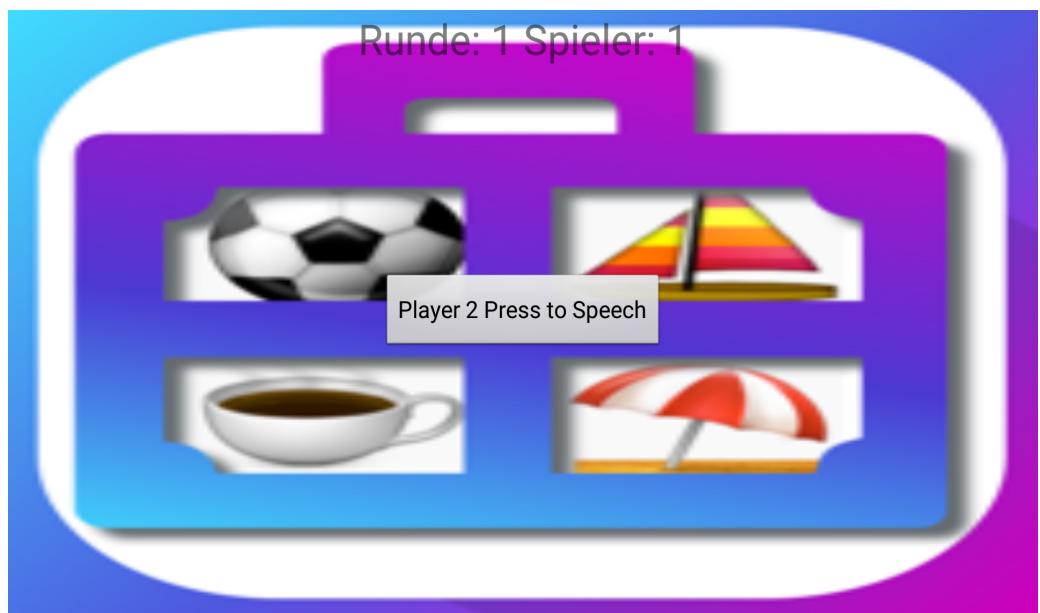


Abbildung 6.35: MainActivity, anschließend ist der nächste Spieler am Zug

Kapitel 7

Fazit

Insgesamt lässt sich hieraus der Schluss ziehen, dass Android die Benutzerschnittstelle zwischen Mensch und Maschine, wie beide miteinander kommunizieren, der Mensch seine Anweisungen an die Maschine übermittelnt und in welcher Form diese ausgeführt wird sowie die Ergebnisse ausgegeben werden, eine moderne und sichere Plattform darbietet. Android bietet zahlreiche Benutzerschnittstellen für unterschiedlichste Benutzer ohne irgendeinen Personenkreis wie eingeschränkte Nutzer auszuschließen. Daraus resultierte, dass mittlerweile Android auf einer Vielzahl verschiedener Gerätetypen läuft. Dank zahlloser Apps für Android lässt sich die Funktionsvielfalt der Geräte erweitern und an die eigene Bedürfnisse anpassen.

Dank der engen Integration von Googles Diensten, den zahlreichen Sensoren, der breiten Unterstützung durch das Open Handset Alliance Konsortium und der Offenen Plattform sowie der leichte Einstieg in die Programmierung dank Java als Programmiersprache und Android Studio als Entwicklungsumgebung, machen das Android-System so populär und und anpassungsfähig. Es lassen sich für Anwender komfortable und Lebenserleichternde Apps programmieren von denen alle profitieren. Ich habe durch das Projekt viel zu Android gelernt. Angefangen von Gedanken über usability der Oberflächenbeschreibungen, also der Benutzerfreundlichkeit, da die Bildschirme von Smartphones im Vergleich zu den Anzeigen von Notebooks oder Desktop-Systemen winzig ist. Welche Informationen beim Bau von Apps dem Benutzer präsentiert werden. Wie man das Bewegen innerhalb des Programms, die Navigation für Anwender logisch und schlüssig gestaltet. Wie ich Programmlogik und -ablauf in Anwendungen in Funktionsblöcke oder Bereiche unterteile, die genau einen Aspekt meines Programms abbilden, da zuviele Inhalte auf den kleinen Bildschirm mobiler Geräte nicht sinnvoll ist und sonst schnell Nutzer überfordern. Verschiedene Benutzereingaben verarbeite, sei es durch Felder, Schaltflächen oder Toucheingaben.

Hieraus lässt sich der Schluss ziehen, dass die verfolgten Ziele der Veranstaltung getroffen wurden und ich nun ein besseres Verständnis von der Kommunikation und Interaktion zwischen Mensch und Maschine habe und dies besser im Studium, im Berufsleben oder beim Programmieren nutzen werden kann.

Kapitel 8

Literaturverzeichnis

Android 5: Programmieren für Smartphones und Tablets. dpunkt.verlag. Arno Becker, Marcus Pant. ISBN:978-3-86490-260-4

Spieleprogrammierung mit Android Studio. Galileo Computing. Uwe Post. ISBN:978-3-8362-2760-5

Mobile Games. Hanser. Thomas Lucka. ISBN:978-3-446-41197-5

Android 4: Apps entwickeln mit dem Android SDK. Galileo Computing. Thomas Künneth. ISBN:978-3-8362-1948-8

<https://developer.android.com/develop/index.html>

Literaturverzeichnis