

# Dokumentation AirSwimmer

|          |   |           |
|----------|---|-----------|
| <b>1</b> | <b>Anforderungen .....</b>                          | <b>3</b>  |
| <b>2</b> | <b>Vorbereitung .....</b>                           | <b>3</b>  |
| 2.1      | Versionsverwaltung .....                            | 3         |
| 2.2      | Android-Tutorium .....                              | 8         |
| 2.3      | Zeiterfassung.....                                  | 19        |
| <b>3</b> | <b>Alternativen.....</b>                            | <b>21</b> |
| 3.1      | Alternativen ohne Infrarot.....                     | 21        |
| 3.2      | Alternativen zu IRdroid .....                       | 23        |
| 3.3      | Simulation/Emulation .....                          | 23        |
| 3.3.1    | Arduino Board .....                                 | 23        |
| 3.3.2    | Raspberry Pi.....                                   | 24        |
| <b>4</b> | <b>Projektleitung .....</b>                         | <b>25</b> |
| 4.1      | Erstellen eines Projektplans.....                   | 25        |
| 4.2      | Projektbesprechungen.....                           | 27        |
| 4.3      | Weitere Aufgaben.....                               | 28        |
| <b>5</b> | <b>Empfangen .....</b>                              | <b>30</b> |
| 5.1      | Analyse der Übertragungsstrategie .....             | 30        |
| 5.2      | Aufnehmen des eigenen Infrarotsignals.....          | 32        |
| 5.2.1    | Versuch 1: Oszilloskop .....                        | 33        |
| 5.2.2    | Versuch 2: Raspberry Pi.....                        | 33        |
| 5.2.3    | Versuch 3: Soundkarte .....                         | 34        |
| 5.2.4    | Versuch 4: Schnittstelle des Computers (RS232)..... | 35        |
| 5.2.5    | Versuch 5: Arduino .....                            | 40        |
| 5.3      | Lirc-File .....                                     | 43        |
| 5.3.1    | Analyse durch WinLirc .....                         | 43        |
| 5.3.2    | Analyse und Verbesserung der Lirc Files .....       | 44        |
| <b>6</b> | <b>Senden .....</b>                                 | <b>49</b> |
| 6.1      | Irdroid Sender .....                                | 49        |
| 6.2      | Analyse der Irdroid App .....                       | 50        |
| 6.2.1    | Java Code .....                                     | 50        |
| 6.2.2    | C-Code .....  | 50        |
| 6.2.3    | Die Irdroid Klasse.....                             | 51        |
| 6.3      | Eigene App .....                                    | 52        |
| 6.4      | Weiterentwicklung der App.....                      | 53        |
| <b>7</b> | <b>Oberfläche.....</b>                              | <b>55</b> |
| 7.1      | Konzept .....                                       | 55        |
| 7.2      | Klassenstruktur .....                               | 56        |
| 7.3      | Startseite.....                                     | 58        |
| 7.4      | Steuerung mittels Buttons .....                     | 62        |
| 7.4.1    | Variante A (erste Testversion) .....                | 62        |
| 7.4.2    | Variante B (ImageButtons) .....                     | 64        |
| 7.4.3    | Reaktion auf Druck und visueller Effekt .....       | 66        |
| 7.5      | Steuerung durch Wischen.....                        | 67        |
| 7.6      | Steuerung durch Kippen .....                        | 70        |
| 7.7      | Lautstärkenkalibrierung .....                       | 77        |
| 7.7.1    | Implementierung .....                               | 77        |
| 7.7.2    | Oberfläche .....                                    | 79        |

|           |   |            |
|-----------|---|------------|
| 7.7.3     | Einbau und permanentes Speichern .....            | 82         |
| 7.7.4     | Einbau der Senden-Logik .....                     | 83         |
| 7.8       | Menü .....  | 84         |
| 7.9       | Infoseite .....                                   | 89         |
| <b>8</b>  | <b>Verbinden von Logik und Oberfläche.....</b>    | <b>94</b>  |
| 8.1       | Buttons.....                                      | 94         |
| 8.2       | Slide.....  | 95         |
| 8.3       | Kippen .....                                      | 95         |
| 8.4       | Permanente Bewegung.....                          | 96         |
| 8.5       | Abschließende Optimierungen .....                 | 99         |
| <b>9</b>  | <b>Graphische Gestaltung der Oberfläche .....</b> | <b>100</b> |
| 9.1       | Konzept .....                                     | 100        |
| 9.2       | Launcher-Icon .....                               | 102        |
| 9.3       | Animierter Hintergrund .....                      | 103        |
| 9.4       | Hintergrundbilder .....                           | 107        |
| 9.5       | Graphiken.....                                    | 107        |
| 9.5.1     | Startseite (Buttons) .....                        | 107        |
| 9.5.2     | Steuerungsseiten (Buttons und Hai).....           | 109        |
| <b>10</b> | <b>Sonstiges .....</b>                            | <b>111</b> |
| 10.1      | Linux Library .....                               | 111        |
| 10.2      | Bau eines eigenen Senders .....                   | 112        |
| 10.3      | Fehlersuche alter Fisch .....                     | 113        |
| 10.4      | Gehäuse .....                                     | 113        |
| 10.4.1    | Recherche Gehäusemöglichkeiten .....              | 113        |
| 10.4.2    | Gehäuse bemalen und verschönern.....              | 114        |
| <b>11</b> | <b>Tests .....</b>                                | <b>114</b> |
| <b>12</b> | <b>Ausblick .....</b>                             | <b>121</b> |
| <b>13</b> | <b>Anhang .....</b>                               | <b>122</b> |
| 13.1      | Anforderungsanalyse .....                         | 122        |
| 13.2      | Zeiterfassung.....                                | 122        |
| 13.3      | Projektplan.....                                  | 122        |
| 13.4      | Protokolle.....                                   | 122        |
| 13.5      | Testprotokolle .....                              | 122        |
| 13.6      | Klassendiagramm der Oberfläche.....               | 122        |

# 1 Anforderungen

Folgende Aufgaben sollten je nach Projektfortschritt bearbeitet werden:

1. Abbild der konventionellen Fernbedienung
2. Fernbedienung mittels Gestensteuerung
3. Flugmanöver mittels Gestensteuerung
4. Flug eines Pacours
5. Konzept zum Ausstatten des AirSwimmers mit Sensorik

Darauf basierend wurde von Anja Hafner, Belgüzar Kocak und Melanie Knappe eine Anforderungsanalyse erstellt, die sich im Anhang befindet.

# 2 Vorbereitung

## 2.1 Versionsverwaltung

*Von Andreas Gerken*

Bei der Softwareentwicklung (vor allem bei größeren Projekten im Team) lohnt es sich auf jeden Fall ein Versionsverwaltungsprogramm einzurichten. Durch die Versionsverwaltung können Änderungen nachvollzogen werden (wann hat wer welche Änderung vorgenommen) und alte Versionen geladen werden. So lässt sich, wenn ein Fehler vorkommt, schnell nachvollziehen seit welcher Version der Fehler im Programm ist und wer ihn eingebracht hat. Im Normalfall kann der „Verursacher“ den Fehler am schnellsten beheben. Ein weiterer großer Vorteil von Versionsverwaltung ist, dass alle Teammitglieder gleichzeitig entwickeln können und ihre Änderungen immer wieder zusammenführen. Das Programm zur Verwaltung achtet dabei darauf, dass keine Änderung verloren geht und weist die Benutzer auf eventuelle Konflikte (wenn mehrere Nutzer eine Datei geändert haben) hin.

Es gibt viele verschiedene Versionsverwaltungsprogramme die sich in Bedienung und Funktionsweise zum Teil stark unterscheiden. Die meisten Systeme speichern alte Versionen auf einem zentralen Server ab und die Benutzer können auf diese über spezielle Programme direkt abrufen. Dabei unterscheiden sie sich insofern, dass der Nutzer bei manchen Systemen bei jeder Verwendung auf den Server zugreifen muss und bei Anderen immer alle alten Versionen in Form eines Repositorys herunterladen kann und auch ohne Internet Verbindung Zugriff auf alle Daten hat. Der Vorteil der „Online-Systeme“ ist, dass immer sichergestellt ist, dass der Benutzer immer Zugriff auf die neusten Versionen hat. Der Vorteil der Systeme die Repositorys herunterlädt ist, dass der Server keinen Dienst anbieten muss sondern nur einen Webspace auf den man von Überall zugreifen kann. Außerdem ist man unabhängig vom Server und durch die lokalen Kopien auf allen Rechnern können sogar alle Daten auf dem Server verloren gehen, ohne dass der komplette Projektfortschritt verloren ist.

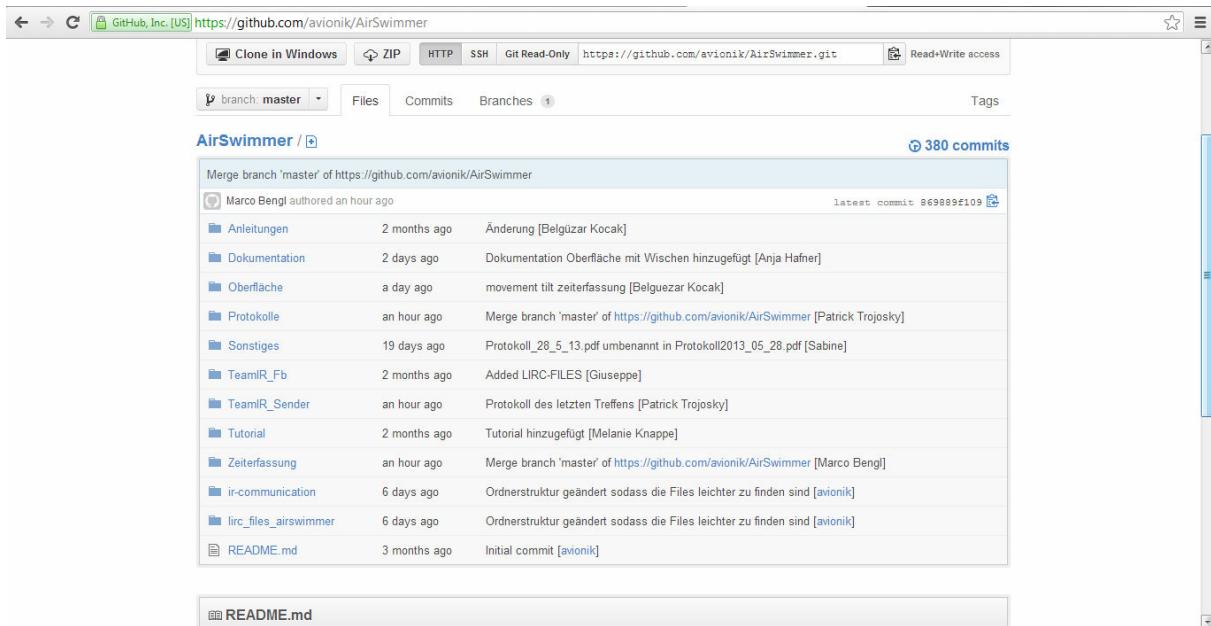
Wir haben uns für ein System namens „GIT“ entschieden, das keine dauerhafte Internetverbindung benötigt, sondern immer die komplette Historie von einem Webspace lädt und lokal auf jedem Entwicklerrechner speichert. Dadurch benötigen wir keinen teuren Server auf dem ein Programm laufen muss sondern können einen kostenlosen Dienst namens „GIT Hub“ verwenden. Um ihn nutzen zu können, muss

man sich nur registrieren und ein Repository auf dem Server anlegen. Man bekommt eine URL, von der man sich das komplette Repository herunterladen kann. Auf der Homepage von Git Hub kann man sich außerdem einloggen und Änderungen über eine Weboberfläche nachvollziehen, ohne ein Programm auf dem PC installiert haben zu müssen.

Zum verwalten des Repositorys gibt die GIT Software die man kostenlos herunterladen kann. Sie bietet zwar alle nötigen Funktionen an, ist jedoch Kommandozeilenbasiert. Somit müsste jedes Teammitglied erst die Befehle lernen und der Aufwand wäre recht groß. Deswegen gibt es mehrere GUI basierte Programme die GIT steuern. Wir haben uns für Tortoise GIT entschieden, das direkt in das Windows Filesystem eingebunden ist. So bekommt man alle Funktionen über einen Rechtsklick auf Ordner über ein Kontextmenü zur Verfügung gestellt. Die Verwendung von GIT ist dadurch sehr intuitiv und kann schnell erlernt werden.

Das GIT System beruht auf dem runterladen, verändern und wieder hochladen des Repositorys. Die beiden Operationen dazu heißen Pull (runterladen) und Push (hochladen) wobei beim herunterladen Änderungen die auf dem Server gespeichert sind mit denen im lokalen Repository verschmolzen werden (merge).

Um das Repository das erste Mal herunter zu laden, muss die Funktion clone aufgerufen werden. Hier kann man die URL des Repositorys auf dem GitHub-Server eingegeben werden. Für unser Projekt wurden zwei Repositorys eingerichtet. Ein Produktives namens „AirSwimmer“ und Eins das zum Testen der GIT Funktionen gedacht ist („TestRepository“). Somit kann Jeder alle Funktionen testen ohne Angst haben zu müssen etwas falsch zu machen.



**Abbildung 1**  
Weboberfläche von GitHub

Hier sieht man direkt auf den ersten Blick, wer in welchem Bereich aktuell entwickelt. Die Messages der neusten Änderung in einem Ordner werden mit dem Namen des Benutzers der die Änderung getätigt hat angezeigt.

## Bibliothek "Dokumente"

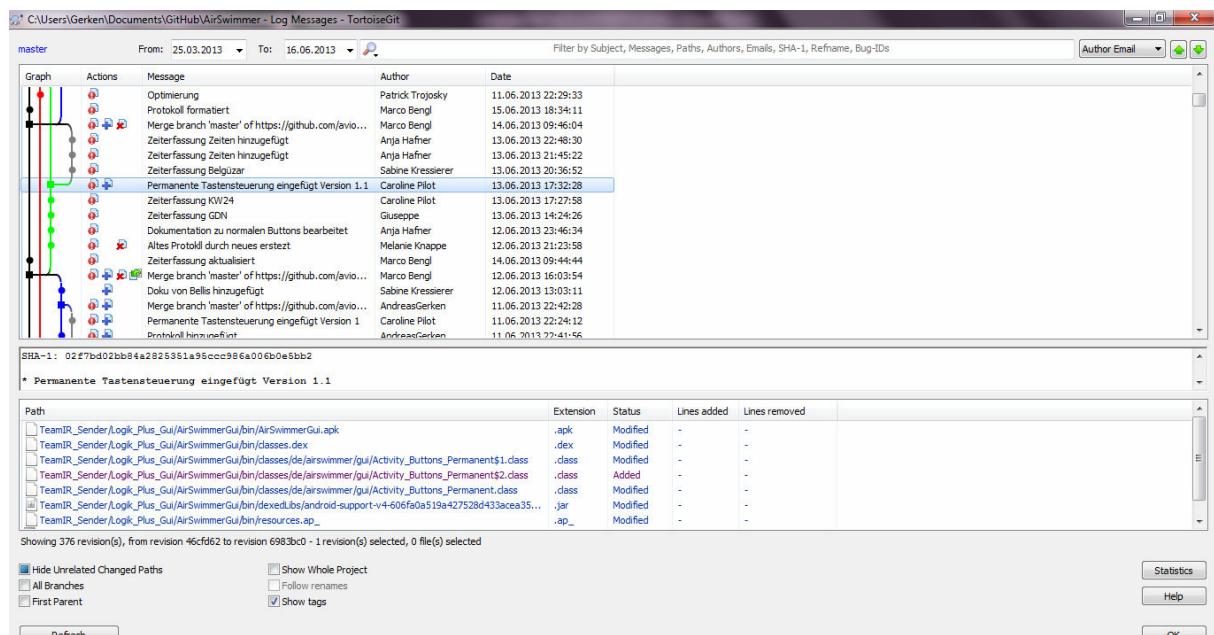
### Oberfläche

| Name                                    | Änderungsdatum   | Typ                  | Größe |
|---|------------------|----------------------|-------|
| ✓ Oberflächen_Entwurf                   | 05.05.2013 15:01 | Dateiordner          |       |
| ✓ src                                   | 29.05.2013 21:38 | Dateiordner          |       |
| ✓ Beschreibung_einzelneOberflaechen.doc | 04.06.2013 12:07 | Microsoft Office ... | 25 KB |
| ✓ Neues Textdokument.txt                | 17.06.2013 19:10 | TXT-Datei            | 0 KB  |
| ✓ ZurInfo.txt                           | 17.06.2013 19:09 | TXT-Datei            | 1 KB  |

**Abbildung 2**

Einbindung von GIT in das Windows Dateisystem durch TortoiseGit

Dateien und Ordner mit grünem Haken sind mit der aktuellsten Version im Lokalen Repository identisch. Dateien mit rotem Ausrufezeichen wurden geändert und müssen noch in das Repository „committed“ werden. Dateien ohne Symbol wurden neu angelegt und noch nicht ins Repository hinzugefügt.



**Abbildung 3**

Exemplarische Änderungshistorie mit der Möglichkeit genau zu sehen, was bei einer Version geändert wurde.

# Anleitung zur Benutzung des Git Systems

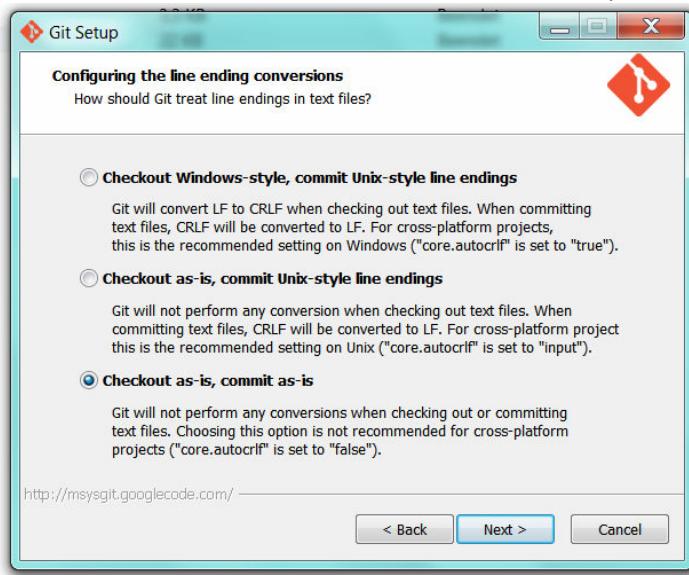
Von Andreas Gerken, Andreas Weber, Caroline Pilot

## 1. Schritt

Git runterladen und installieren:

<http://git-scm.com/>

Hierbei ist darauf zu achten, dass beim Installationsprozess folgendes eingestellt ist:



**Abbildung 4**  
„Git Setup“-Dialog

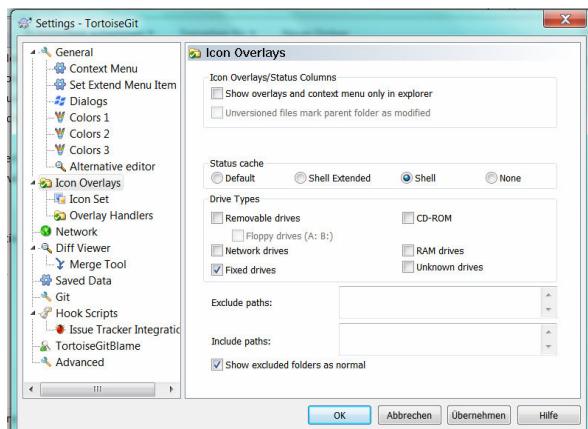
TortoiseGit runterladen und installieren:

<https://code.google.com/p/tortoisegit/wiki/Download>

## 2. Schritt

### Git einrichten

- Ordner auf eigenem Rechner anlegen (in dem die Dateien des Projekts gespeichert werden sollen) z.B. Avionik Projekt (nicht AirSwimmer!)
- Rechter Mausklick im leeren Ordner → „TortoiseGit“ → „Settings“
  - o → „Git“, Name und E-Mail einfügen
  - o → „IconOverlays“, Einstellungen siehe Bild



**Abbildung 5**  
Einstellungen von „TortoiseGit“

### **3. Schritt**

#### Klonen der Server-Ordner auf den Rechner

- Rechter Mausklick im Ordner → „Git Clone“
- Folgende URL ist einzugeben
  - o <https://github.com/avionik/TestRepository.git>
    - Dieser Ordner ist zum Ausprobieren gedacht‘
- Letzte zwei Punkte sind zu wiederholen, diesmal mit folgender URL
  - o <https://github.com/avionik/AirSwimmer.git>
    - Dieser Ordner ist für die richtigen Dateien des AirSwimmers gedacht

### **4. Schritt**

#### Arbeiten mit dem GitHub Server

##### **Grundbegriffe und Funktionen:**

- Pull
  - o Lädt die neuste Version vom Server herunter und verschmilzt (Merge) diese mit der lokalen Version
    - Merge kann evtl. zu Problemen führen, da dieser nicht trivial ist, man muss die zu übernehmende Version manuell über den Menüpunkt „Conflict“ auswählen
- Push
  - o Lädt die lokal gespeicherte Version auf den Server
- Commit
  - o Speichert die Änderung als neue Version auf dem lokalen Repository

**Vor Änderungen ist es sinnvoll erst die neuste Version herunterzuladen um Konflikte zu vermeiden.**

##### **Änderungen ins lokale Repository speichern**

- Änderungen auf Datei ausführen
- Rechter Mausklick auf geänderte Datei → „Git commit -> master“
- Message hinzufügen (ausdrucksstarke Info!)
- Bestätigen

##### **Lokales Repository auf den Server Laden**

- „Git commit -> master“ falls noch nicht geschehen
- Im Order, in dem geänderte Datei liegt rechter Mausklick
- Git Sync
- Pull
- Push (Eingabe von Nutzernamen und Passwort erforderlich)

Wenn das Pull dabei nicht durchgeführt wird, kann beim Push die Meldung kommen, dass das lokale Repository gegenüber dem aktuellen Repository (auf dem Server) zurück liegt. Da die Push Operation dann nicht durchgeführt werden kann, ist es also sehr wichtig den Pull Schritt nicht zu vergessen.

## 2.2 Android-Tutorium

Von Melanie Knappe

Die Applikation soll für Android basierte Smartphones und Tablets entwickelt werden. Die meisten Projektteilnehmer haben noch nicht mit Android gearbeitet. Mit Hilfe eines Android Tutorial soll der Einstieg erleichtert werden.

### Installation von Eclipse

Um für Android basierte Systeme entwickeln zu können, muss die Entwicklungsumgebung eingerichtet werden. Wir verwenden Eclipse mit Android Development Kit und Android Software Development Kit.

### Java Software Development Kit (SDK) runterladen und installieren:

Als erstes lädt man sich das Java SDK runter. Unter dem folgendem Link findet man die Downloadseite von Oracle [www.oracle.com/technetwork/java/javase/downloads/index.html](http://www.oracle.com/technetwork/java/javase/downloads/index.html). Hier wird das Java Development Kit (JDK), für das jeweilige Betriebssystem, heruntergeladen. (Stand 14.11.12 Java SE 7u9)



Abbildung 6

Java SE Downloads der oracle-Webseite (Stand 14.11.12)

Nach dem Runterladen wird das Java Development Kit installiert.

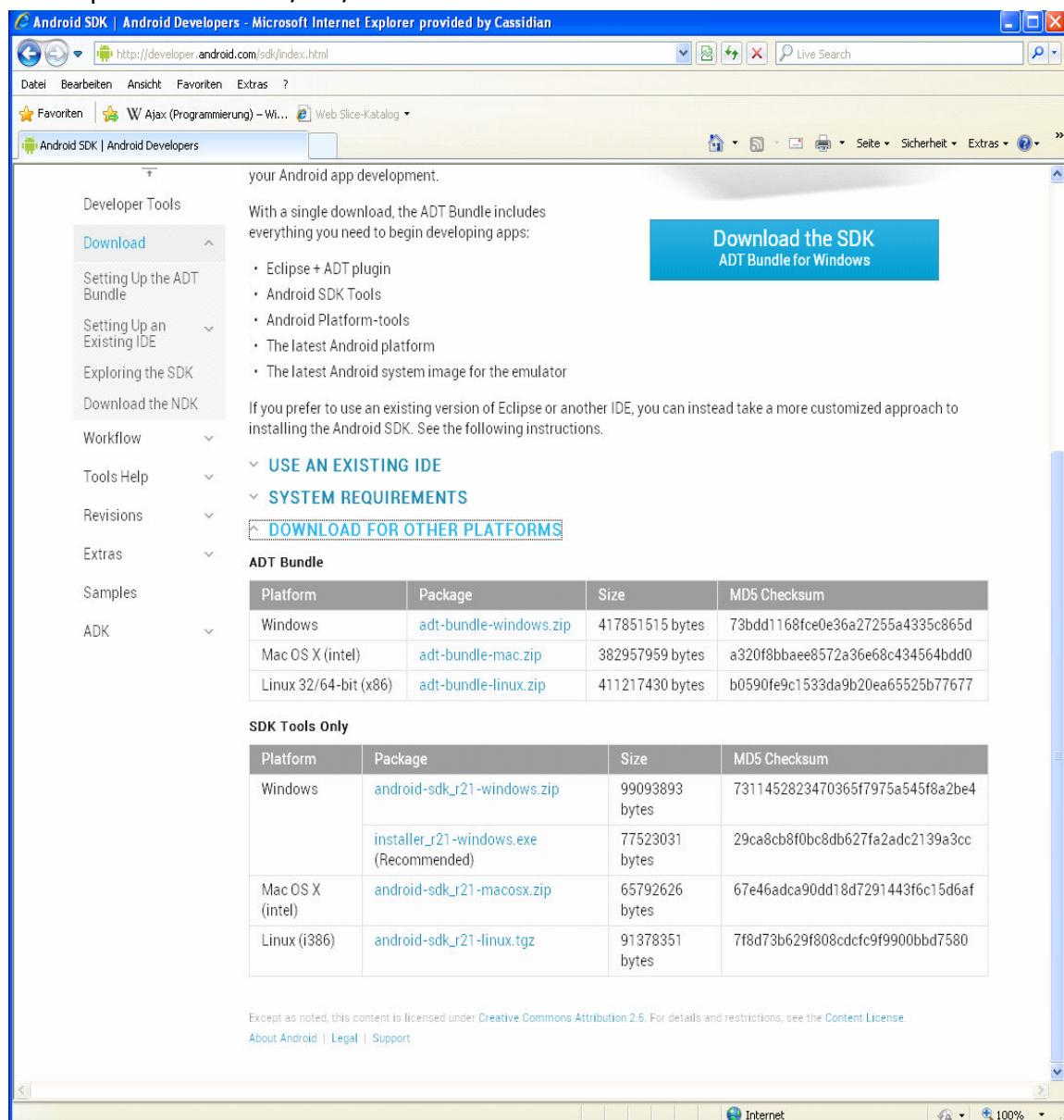
Bei Custom Setup wird das Development Tools ausgewählt. Dann klickt man auf "Next" um fortzufahren. Danach den gewünschten Zielordner angeben und wieder auf "Next" klicken. Dann noch auf "Continue" klicken und das Java SDK ist installiert.

Setup-Schritt von Java SDK. Erst einmal auf "Next" klicken dann einen Zielordner angeben und mit "Next" bestätigen. Dann einfach mit "close" das Ganze schließen.

Jetzt öffnet sich automatisch eine Webseite auf der man sich Registrieren kann, aber nicht muss.

## **Android Software Development Kit (SDK) herunterladen und installieren:**

Nun laden wir das Android SDK herunter. Das geht über die Seite [developer.android.com/sdk/index.html](http://developer.android.com/sdk/index.html).



**Abbildung 7**

Webseite „[developer.android.com](http://developer.android.com)“ mit Android SDK zum downloaden

Hier die aktuelle Version für die jeweilige Plattform herunterladen. (Stand: 14.11.12: Windows installer\_r21-windows.exe). Setup starten. Mit "Next" fortfahren. Android SDK Setup überprüft, ob die installierte Java Version aktuell genug ist. Mit "Next" kommt man zur Eingabe des Zielordners. Mit "Next" weiter. Jetzt kann man eine Start Menu Folder eingeben und dann auf Installieren klicken. Nach dem installieren den Start SDK Manager aktivieren und mit "Finish" bestätigen. Im Android SDK Manager die jeweiligen Parkette runterladen. (Stand: 14.11.12: Es wurden alle bis dahin verfügbaren Parkette runtergeladen).

### **Eclipse herunterladen und installieren:**

Eclipse kann man unter [www.eclipse.org/downloads/](http://www.eclipse.org/downloads/) herunterladen. Die Version „Eclipse IDE for Java Developers“ wird benötigt.

The screenshot shows the Eclipse Downloads page in Microsoft Internet Explorer. The main content area displays a list of Eclipse packages for Windows, each with a download link for Windows 32 Bit and Windows 64 Bit. The packages listed include:

- Eclipse IDE for Java EE Developers, 221 MB
- Eclipse Classic 4.2.1, 183 MB
- Eclipse IDE for Java Developers, 150 MB
- JRebel for Eclipse
- Eclipse IDE for C/C++ Developers, 129 MB
- Eclipse for Mobile Developers, 144 MB
- Eclipse IDE for Java and DSL Developers, 260 MB
- Eclipse Modeling Tools, 274 MB
- Eclipse for RCP and RAP Developers, 227 MB
- Eclipse IDE for Java and Report Developers, 260 MB
- Eclipse for Parallel Application Developers, 199 MB
- Eclipse for Testers, 95 MB
- Eclipse IDE for Automotive Software Developers (includes Incubating components), 183 MB

The sidebar on the right contains links for "Installing Eclipse" (Install Guide, Compare/Combine Packages, Known Issues, Updating Eclipse), a Jazz advertisement (Innovation through Collaboration), and "Related Links" (Documentation, Make a Donation, Forums, Eclipse Juno (4.2), Eclipse Indigo (3.7), Older Versions). A "Hint" section at the bottom explains that a Java runtime environment (JRE) is required to use Eclipse, and provides a link to the Eclipse Foundation Software User Agreement.

**Abbildung 8**

Download-Seite von Eclipse

Einfach runterladen und entpacken. Fertig!

Beim ersten Start muss noch der Workspace angegeben werden.

### **Eclipse aktualisieren:**

Beim ersten Aufruf kommt ein Dialogfeld „Help/Check for Updates“, dieses wird mit „Yes“ bestätigt. Darauf hin öffnen sich die Preferences/Available Software Sites. Über die Schaltfläche „Add...“ kann neue Seite dazu gefügt werden. Auf dieser Seite wird dann nach Updates gesucht.

### **ADT (Android Developer Tools) in Eclipse einbinden:**

Dafür gibt es eine Anleitung auf der Seite [developer.android.com/sdk/index.html](http://developer.android.com/sdk/index.html). Auf dieser Seite findet sich ein Link (Stand 14.11.12: <https://dlssl.google.com/android/eclipse>). Dieser Link wird kopiert und bei Eclipse unter „Help/Install new Software“ bei „Add...“ hinzugefügt. Daraufhin sucht Eclipse die neue Software. Die gewünschte Software auswählen und durch einen Klick auf „Next“ wird die Software geladen. Sollte die Installation fehlgeschlagen, kann es sein, das Eclipse nicht als Administrator ausgeführt wird.

Wenn die Installation erfolgreich war sollte Eclipse neu gestartet werden. Nach dem Neustart kommt ein neues Fenster „Welcome to Android Development“. Bei diesem Fenster können wir „Use existing SDKs“ auswählen und den Ordner angeben wo unser Android Software Development Kit liegt. Mit „Next“ kommt man weiter auf eine Seite wo man entscheiden muss, ob Daten an Google gesendet werden dürfen oder nicht. Mit „Finish“ wird das ganze fertiggestellt.

### ***Die Ersten Schritte in Eclipse***

Hilfreiche Internetseite für die Entwicklung von Android Applikationen ist <http://developer.android.com/index.html>. Diese Seite wurde speziell für Android Applikationen Entwickler ins Leben gerufen.

Die Entwicklung mit Eclipse bietet viele Vorteile. Zum Beispiel lassen sich Emulatoren leicht erstellen. Emulatoren simulieren Android Gerät. Die simulierten Android Geräte lassen sich komplett konfigurieren. Zum Beispiel kann die Auflösung, die API und der Speicherplatz des Android Gerätes eingestellt werden. Die Emulatoren sind somit ein sehr hilfreiches Werkzeug, wenn Android Applikationen entwickelt werden. Gerade dann, wenn die Applikation auf mehreren API's laufen soll. Da mehrere Emulatoren mit unterschiedlichen API's erstellt werden können und so die Applikation, für verschiedene API's, bequem getestet werden kann.

Bei der Android Applikation Entwicklung bekommt die Oberfläche immer mehr Aufmerksamkeit. Die Oberflächen sollen ansprechend und leicht zu bedienen sein. Die Bedienung soll so gestaltet sein, das keine langen Erklärungen notwendig sind.

Für die Gestaltung der Oberflächen bietet Eclipse mit dem Android Development Tool Hilfen an. Es gibt schon fertiges Layout Strukturen, die entweder einzeln bzw. in Kombination verwendet werden können. Einige Beispiele sind hier:

- GridLayout

Bei einem GridLayout werden die einzelnen Views in ein zweidimensionales, rollbaren Raster gezeigt. Die Grid-Elemente werden automatisch in das Layout eingefügt.

- LinearLayout

Die gesamte Ausrichtung beschränkt sich auf horizontale bzw. vertikal. Die Richtung lässt sich mit dem android:orientation Attribut einstellen.

- Relative Layout

Kinder Ansichten können relative Positionen haben. Die Position der jeweiligen Ansicht kann relativ zu einem Geschwister Element angegeben werden oder relativ zum übergeordneten Bereich angegeben werden.

## **Erstellen von einem Projekt**

Ein neues Projekt erstellen geht über „File/New/Other...“. Es öffnet sich ein Dialogfenster, indem das genaue Projekt ausgewählt werden kann. Um ein Android Projekt zu erstellen muss „Android/Android Applikation Project“ ausgewählt werden und mit „next“ bestätigt werden. Auf der nächsten Seite, des Dialogfensters, kann der Application Name, Project Name, Package Name und das SDK angegeben und gewählt werden. Auf den nächsten Seiten lassen sich noch optische Einstellungen (z.B. Launcher icon) vornehmen, aber die später jeder Zeit noch verändert werden können. Ein Android Applikation Projekt hat mehrere Bestandteile, auf die wichtigsten wird kurz eingegangen.

- Src

Im Src-Ordner liegen der Java Quellecode.

- Gen

Der Ordner Gen, wird automatisch angelegt. Die Files in diesem Ordner stellen eine Brücke zwischen Javacode und Ressourcen da.

- Libs

zusätzliche Bibliotheken können hier eingefügt werden.

- Res

in diesem Ordner sind alle Ressourcen für die Applikation untergebraucht.

- o Drawable

Alle Grafiken und Bilder sind in den Drawable-Ordnern.

- o Layout

Die Oberflächen, die später von der Applikation angezeigt wird, befinden sich in dem Layout-Ordner.

- o Menu

Alle Menus, die erstellt werden befinden sich im Menu-ordner.

- o Values

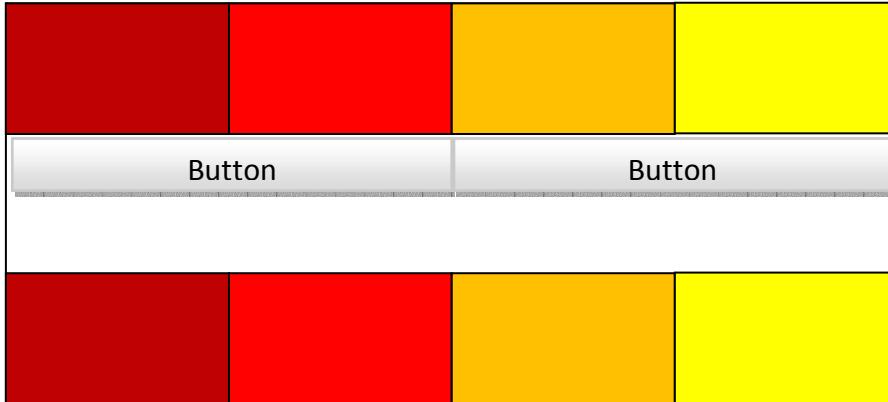
Im Values-Ordner sind alle Definierten Größen, wie zum Beispiel Farben, Strings und Arrays.

- Manifest

Manifest Datei bedeutet so viel wie „Öffentliche Erklärung von Zielen und Absichten“. In dieser Datei finden sich alle Informationen über die Ziele der Applikation. Berechtigungen werden hier definiert.

## **Erstes Beispiel: Oberflächen Gestaltung**

Das erste Beispiel beschäftigt sich mit der Oberflächen Gestaltung. Dieses Beispiel wurde gewählt, weil die Oberflächen Gestaltung bei der Entwicklung von Android Applikationen einen zentralen Punkt spielt.



Dieses Layout soll im ersten Beispiel nachgebaut werden.

### **Farben definieren:**

Bevor das Layout gestaltet werden kann, müssen davor noch die Farben definiert werden. Farben werden an der folgenden Stelle definiert: Projekt/res/values/color.xml. Da das „Android XML File Values File“ color.xml noch nicht existiert muss dieses noch hinzugefügt werden. Die vier Farben die hier benutzt werden sind wie folgt definiert:

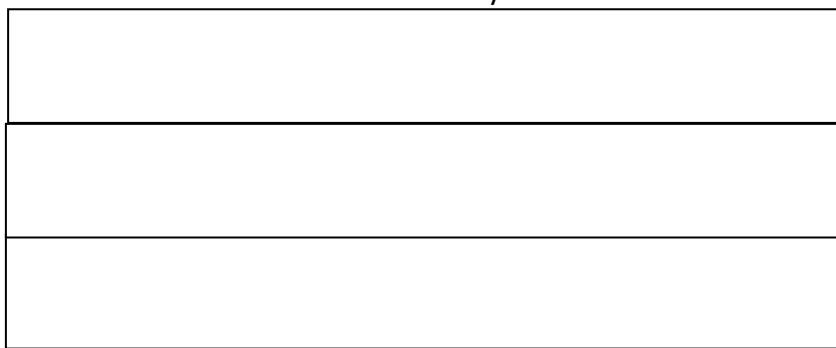
```
<?xml version="1.0" encoding="utf-8"?>
<resources>

    <color name="Rot">#8B0000</color>
    <color name="HellRot">#FF0000</color>
    <color name="Orange">#FF6600</color>
    <color name="Gelb">#FFFF00</color>

</resources>
```

### **Aufbau des Layouts:**

Die Grundstruktur bildet ein LinearLayout mit vertikaler Orientierung.



Der Code sieht im XML wie folgt aus:

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >
</LinearLayout>
```

Die weitere Unterteilung ist wieder LinearLayout, aber hier mit horizontaler Orientierung.

|  |  |  |  |
|--|--|--|--|
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >

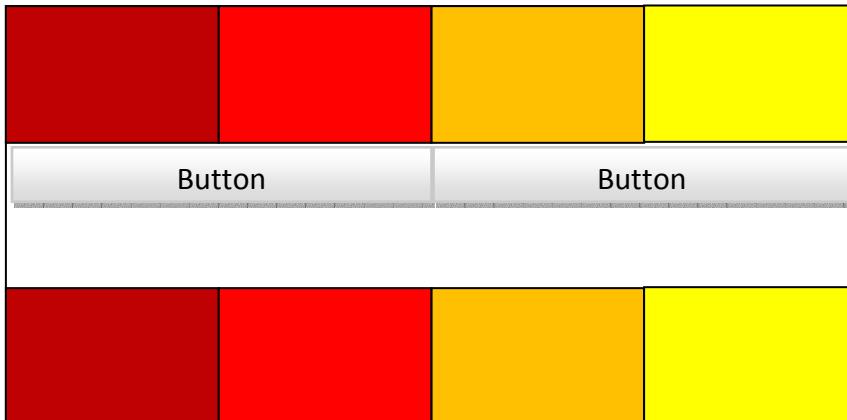
    //1.Zeile des vertikalen Layouts (beinhaltet ein LinearLayout mit
    //horizontaler Orientierung)
    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_weight="1"
        android:orientation="horizontal" >
        </LinearLayout>

    //1.Zeile des vertikalen Layouts (beinhaltet ein LinearLayout mit
    //horizontaler Orientierung)
    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_weight="1"
        android:orientation="horizontal" >
        </LinearLayout>

    //1.Zeile des vertikalen Layouts (beinhaltet ein LinearLayout mit
    //horizontaler Orientierung)
    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_weight="1"
        android:orientation="horizontal" >
        </LinearLayout>

</LinearLayout>
```

Also nächster Schritt werden die Views mit den jeweiligen Farben zugefügt und die zwei Button. Danach sollte das Layout wie gewünscht ausschauen:



Der Programmcode sieht wie folgt aus:

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_weight="1"
        android:orientation="horizontal" >

        <View
            android:id="@+id/view1"
            android:layout_width="0dp"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:background="@color/Rot" />

        <View
            android:id="@+id/view2"
            android:layout_width="0dp"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:background="@color/HellRot" />

        <View
            android:id="@+id/view3"
            android:layout_width="0dp"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:background="@color/Orange" />

        <View
            android:id="@+id/view4"
            android:layout_width="0dp"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:background="@color/Gelb" />

        <Button
            android:id="@+id/button1" />
        <Button
            android:id="@+id/button2" />
    </LinearLayout>
</LinearLayout>
```

```

</LinearLayout>

<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="0dp"
    android:layout_weight="1"
    android:orientation="horizontal" >

    <Button
        android:id="@+id/button1"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:text="Button" />

    <Button
        android:id="@+id/button2"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:text="Button" />
</LinearLayout>

<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="0dp"
    android:layout_weight="1"
    android:orientation="horizontal" >

    <View
        android:id="@+id/view5"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:background="@color/Rot" />

    <View
        android:id="@+id/view6"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:background="@color/HellRot" />

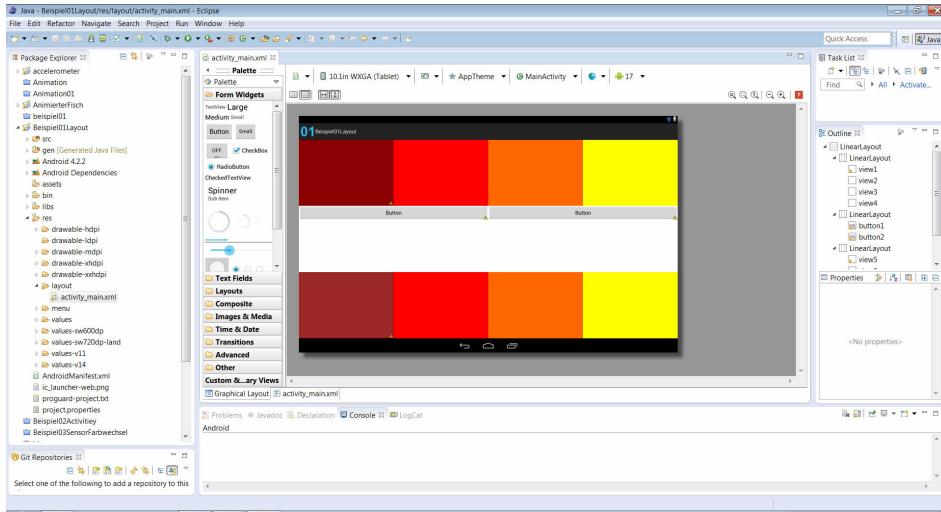
    <View
        android:id="@+id/view7"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:background="@color/Orange" />

    <View
        android:id="@+id/view8"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:background="@color/Gelb" />
</LinearLayout>

</LinearLayout>

```

Das fertige Layout schaut in Eclipse wie folgt aus:



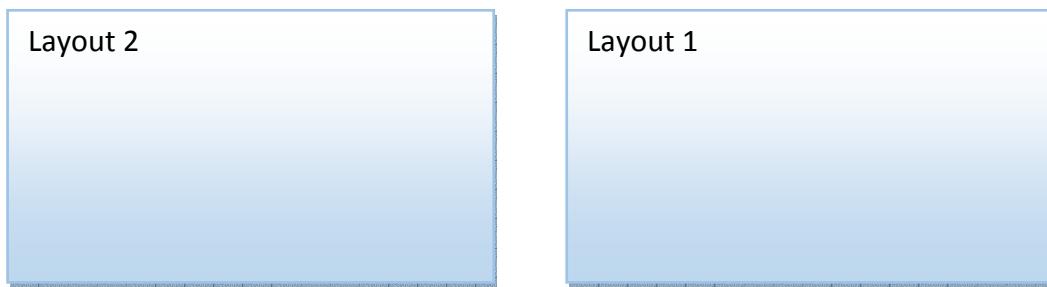
### Was wurde gelernt:

- Kombination von verschiedenen Layouts
- Definition von Farben

### **Zweites Beispiel: Layouts von Activities aus aufrufen**

Die gestalteten Layouts müssen nun auch angezeigt werden. Das geschieht mit Hilfe von Activities. Activities rufen Layouts auf und übernehmen zum Beispiel Event Handling. Jede Aktivität muss in der Manifestdatei bekannt sein.

Das zweite Beispiel erklärt, wie Layouts mit Hilfe von Activities angezeigt werden können. Außerdem wird ein Click-Event auf einen Button ausgewertet und zu einem zweiten Layout weitergeleitet.



Das „Layout 1“ soll gestartet werden, sobald die Applikation gestartet wird. Klickt der Nutzer auf den Button soll das „Layout 2“ geöffnet werden.

Als erstes werden die zwei Layouts gestaltet:

### Layout 1:

Dieses Layout braucht einen Button. Der Button bekommt ein zusätzliches Attribut „android:onClick="onButtonClick"“. Wird auf den Button geklickt wird die Funktion „onButtonClick“ aufgerufen.

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity" >

    <Button
        android:id="@+id/button1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentTop="true"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="26dp"
        android:text="Zur nächsten Seite"
        android:onClick="onButtonClick"/>

</LinearLayout>
```

Das zweite Layout bekommt nur ein Textfeld „Willkommen auf der Seite 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"
    android:orientation="vertical" >

    <TextView
        android:id="@+id/textView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Willkommen auf der Seite 2"/>

</LinearLayout>
```

Nachdem beide Layouts erstellt worden sind müssen die Activities gestaltet werden.

MainActivity:

```
package de.tutorial.beispiel02activitiey;
import android.os.Bundle;
import android.app.Activity;
import android.content.Intent;
import android.view.Menu;
import android.view.View;

public class MainActivity extends Activity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main); //Hier kommt der jeweilige Layoutname rein
    }
    public void onButtonClick(View view){      //heißt genauso wie die Funktion im XML-File
        switch (view.getId()){
            case R.id.button1:                  //Heißt wie der Button im XML-File
```

```

        /*Aufruf einer neuen Activity, die dann das Layout 2 anzeigt. In diesem Fall
        hat die neue Activity den Namen Unteractivity.class */
        startActivity(new Intent(this, Unteractivity.class));
        break;
    }
}
}

```

Die Activity, die das Layout 2 Aufruft sieht wie folgt aus:

```

package de.tutorial.beispiel02activitiey;
import android.os.Bundle;
import android.app.Activity;
import android.view.Menu;
import android.view.View;

public class Unteractivity extends Activity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.unteractivity);
    }
}

```

### Was wurde gelernt?

- Wie werden Layouts von Activitys aufgerufen
- Wie werden Click-Events behandelt

## 2.3 Zeiterfassung

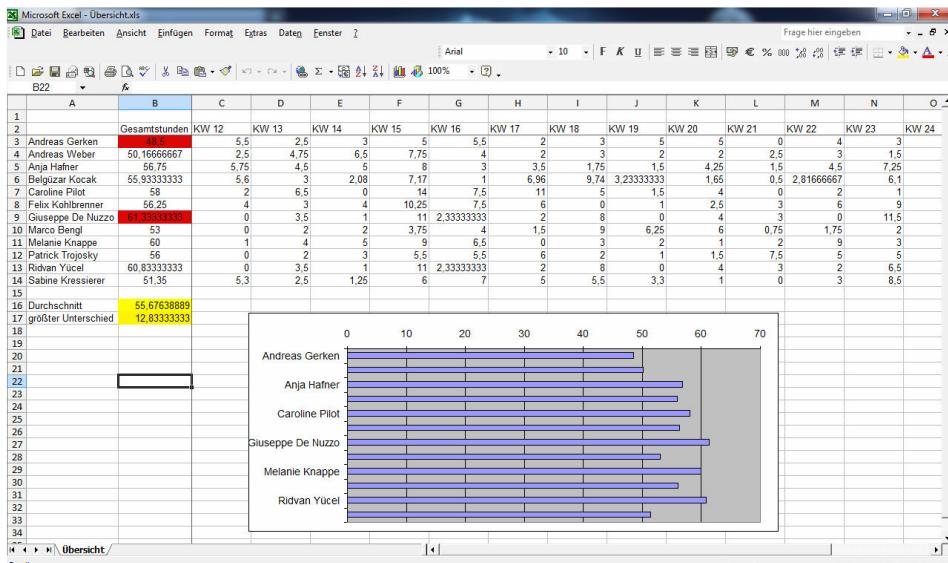
*Von Sabine Kressierer*

### **Ziel**

Ziel war es, ein Tool zu finden, dass es ermöglicht die gearbeiteten Zeiten aller Teammitglieder zu erfassen und übersichtlich darzustellen. Dazu sollte jeder seine Zeiten, und die dabei erledigten Arbeiten, selbst eintragen können.

### **Ergebnis**

Ein Großteil der im Internet vorhandenen Zeiterfassungstools ist für mehr als einen Nutzer kostenpflichtig. Zudem handelt es sich dort meist um komplettete Projektmanagement-Tools, bei denen der Aufwand zur Einrichtung und Konfiguration des Tools sowie die Einarbeitungszeit zu hoch wäre, um eine einfache Zeiterfassung für 12 Personen zu erhalten. Aus diesem Grund wird stattdessen eine Excel-Tabelle, die Andreas Weber zur Verfügung stellte, verwendet. Dort ist es möglich für die einzelnen Tage, oder ganze Wochen, gearbeitete Stunden oder Uhrzeiten einzutragen, aus denen die Arbeitszeit einer Woche sowie über die gesamte Projektphase berechnet wird. Diese Datei wurde auf den Projektzeitraum angepasst und für jedes Teammitglied dupliziert. Zudem wurde eine Übersicht erstellt, die automatisch die Gesamtstunden der einzelnen Teammitglieder, sowie die einzelnen Wochenstunden anzeigt. Die Gesamtstunden werden zudem in einem Balken-Diagramm graphisch dargestellt(Abbildung 9).



**Abbildung 9**

Übersicht Zeiterfassung

So ist ein schneller Vergleich der verschiedenen Zeiten möglich, die später bei der Verteilung der neuen Arbeitspakete berücksichtigt werden. Um die Zeiten nachvollziehen zu können, wird zu der entsprechenden Zeitangabe eingetragen, was bearbeitet wurde. Zu Beginn befand sich die gesamte Zeiterfassung in einer einzelnen Excel-Datei. Da das Versionsverwaltungstool GIT jedoch keine Excel-Dateien mergen kann, führte dies immer wieder zu Konflikten. Aus diesem Grund sind die Zeiterfassungstabellen der einzelnen Teammitglieder nun in extra Dateien, die sich zusammen mit der Übersicht in einem Ordner befinden.

| Zeiterfassung         |                  |                       |       |
|-----------------------|------------------|-----------------------|-------|
| Name                  | Änderungsdatum   | Typ                   | Größe |
| Andreas Gerken.xls    | 20.06.2013 20:33 | Microsoft Excel-Ar... | 42 KB |
| Andreas Weber.xls     | 20.06.2013 23:00 | Microsoft Excel-Ar... | 47 KB |
| Anja Hafner.xls       | 20.06.2013 20:33 | Microsoft Excel-Ar... | 40 KB |
| Belgezar Kocak.xls    | 20.06.2013 20:33 | Microsoft Excel-Ar... | 46 KB |
| Caroline Pilot.xls    | 20.06.2013 20:33 | Microsoft Excel-Ar... | 54 KB |
| Felix Kohlbrenner.xls | 20.06.2013 20:33 | Microsoft Excel-Ar... | 45 KB |
| Giuseppe De Nuzzo.xls | 20.06.2013 20:33 | Microsoft Excel-Ar... | 51 KB |
| Marco Bengl.xls       | 20.06.2013 20:33 | Microsoft Excel-Ar... | 49 KB |
| Melanie Knappe.xls    | 20.06.2013 23:00 | Microsoft Excel-Ar... | 48 KB |
| Patrick Trojosky.xls  | 20.06.2013 20:33 | Microsoft Excel-Ar... | 47 KB |
| Ridvan Yücel.xls      | 20.06.2013 20:33 | Microsoft Excel-Ar... | 44 KB |
| Sabine Kressierer.xls | 20.06.2013 20:33 | Microsoft Excel-Ar... | 45 KB |
| Übersicht.xls         | 20.06.2013 20:33 | Microsoft Excel-Ar... | 40 KB |

**Abbildung 10**

Ordnerstruktur Zeiterfassung

Da so jeder eine eigene Datei zum Ändern hat, können keine Versionskonflikte mehr auftreten. Die Übersicht ist weiterhin mit den einzelnen Dateien verlinkt. Da in Excel alle Pfade relativ gespeichert werden, können die verlinkten Felder auch aktualisiert werden, wenn der gesamte Ordner verschoben wurde. Da sich jedoch alle Dateien in einem Ordner befinden müssen ist es so nicht mehr möglich den Stand direkt auf [github.com](https://github.com) zu aktualisieren. Dort wird nur der zuletzt gepushte Stand angezeigt. Die abschließende Version der Zeiterfassung befindet sich im Anhang.

## 3 Alternativen

### 3.1 Alternativen ohne Infrarot

von Anja Hafner

Für den Fall, dass die Ansteuerung des AirSwimmers mit Infrarot nicht realisiert werden kann, wird nach alternativen Geräten gesucht.

Diese Geräte sollten mittels Bluetooth ansteuerbar sein, nicht mehr als 100 € kosten und fliegen können.

#### Warum Bluetooth?

Die Steuerung per Bluetooth ist gewünscht, da generell alle aktuellen Tablets und Smartphones Bluetooth besitzen und keine zusätzliche Hardware (wie bei der Steuerung mit Infrarot) nötig ist. Außerdem ist die Umsetzung dieser Kommunikation mit dem Fisch deutlich einfacher als mit Infrarot, da die Kommunikation mit Bluetooth besser dokumentiert ist.

#### Warum soll es fliegen?

Da es sich um ein Avionik-Projekt handelt, sollte der Gedanke, etwas flugfähiges zu steuern, nicht verloren gehen.

#### Suche nach Alternativen

Zu Beginn wird nach einem AirSwimmer gesucht, der mit Bluetooth gesteuert werden kann. Jedoch gibt es keinen alternativen AirSwimmer, der mit Bluetooth gesteuert wird, lediglich funkgesteuerte AirSwimmer werden angeboten.

Aus diesem Grund wird die Suche auf alle flugfähigen, mit Bluetooth und Android gesteuerten Geräte, erweitert.



**Abbildung 11**

Bluetooth-gesteuerter Hubschrauber

Auch hier ist die Auswahl nicht groß, lediglich eine flugfähige Alternative der Firma BeeWi wird angeboten. Dieser Hubschrauber ist per Bluetooth 3.0 ansteuerbar und bietet bis zu 8 Minuten Flugzeit. Die Ladezeit hingegen beläuft sich auf ca. 40 Minuten und erfolgt per USB. Ein weiterer Nachteil ist die schwere Steuerbarkeit des Hubschraubers. Die Kosten des BeeWi BB/301-A0 Bluetooth Controlled Hubschraubers in schwarz belaufen sich auf 50,37 €.

## **Fahrfähige Alternativen**

Da nur eine flugfähige Alternative mit Bluetooth gefunden wurde, wird ebenfalls nach Bluetooth-gesteuerten Fahrzeugen gesucht, auch wenn dies nicht ganz den Anforderungen entspricht. Der Vorteil von Fahrzeugen ist die einfachere Steuerbarkeit. Das erste Modell stammt ebenfalls von der Firma BeeWi und ist ein Mini Cooper. Dieses Modellauto wird mit Bluetooth 2.0 gesteuert und hat eine Reichweite von bis zu 10 Metern. Für den Betrieb sind 3 AA Batterien erforderlich und die Kosten betragen 30,54 € bei Amazon.



**Abbildung 12**  
Bluetooth gesteuerter Mini Cooper

Eine weitere Alternative sind die so genannten Tankbots. Die kleinen Fahrzeuge werden per USB aufgeladen und besitzen 3 bereits vorhandene Fahrmodi: sie können Hindernisse selbstständig ausweichen, in freien Bewegungen "tanzen" oder per Smartphone gesteuert werden. Die Kosten belaufen sich auf 24,99 € bei Amazon.



**Abbildung 13**  
Tankbots

Für den Fall, dass die Ansteuerung per Infrarot nicht realisiert werden kann, wurde der Hubschrauber als mögliche Alternative ausgewählt.

## 3.2 Alternativen zu IRdroid

Von Marco Bengl

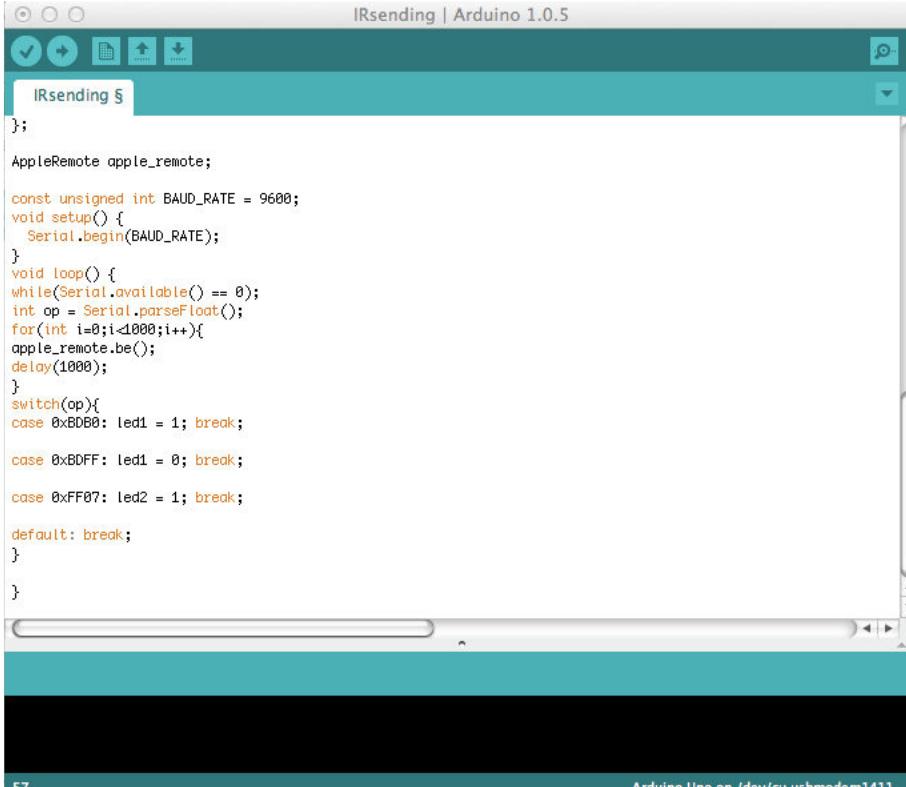
Bei der Suche nach einem Infrarotsender für ein Tablet wurde man schnell bei dem Irdroid-Infrarot-Sender fündig. Da der Versand des Irdroid-Senders zwei bis drei Wochen dauern könnte wurde sich auch nach Alternativen erkundigt. Dabei wurde klar, dass es bereits mehrere Apps gab, die als Infrarotsteuerung verwendet werden konnten. Hardware die als Alternative für den Irdroid infrage kam, gab es nicht. Grund für die fehlenden Alternativen ist, dass Infrarot in Smartphones und Tablets in den vergangenen Jahren für veraltet gehalten wurde. Infrarot wurde weitgehend durch Bluetooth ersetzt und war somit nicht mehr weiter von nutzen.

## 3.3 Simulation/Emulation

### 3.3.1 Arduino Board

Von Giuseppe De Nuzzo und Ridvan Yücel

Die Backup Lösung dient als Absicherung des Projekts, sollte dieses scheitern. Die Android Applikation soll hierbei nicht mehr den Fisch steuern, sondern mit dem Arduino Board kommunizieren. Die Aktionen LEFT, RIGHT, DIVE und CLIMB werden durch LED's simuliert. Jede Taste die auf der App gedrückt wird, bzw. das dadurch emittierte Infrarotsignal soll vom Arduino Board verarbeitet werden. Hierbei ist es nicht mehr notwendig die originale Fernbedienung aufzuzeichnen, es werden Hex-Werte vordefiniert. (Bsp. 0x0F für LEFT).



The screenshot shows the Arduino IDE interface with the title bar "IRsending | Arduino 1.0.5". The code editor contains the following C++ code for an Arduino sketch:

```
IRsendering | Arduino 1.0.5

IRsendering §

};

AppleRemote apple_remote;

const unsigned int BAUD_RATE = 9600;
void setup() {
  Serial.begin(BAUD_RATE);
}
void loop() {
while(Serial.available() == 0);
int op = Serial.parseFloat();
for(int i=0;i<1000;i++){
apple_remote.be();
delay(1000);
}
switch(op){
case 0xB0B0: led1 = 1; break;
case 0xBDFF: led1 = 0; break;
case 0xFF07: led2 = 1; break;
default: break;
}
}

57
```

At the bottom right of the IDE, it says "Arduino Uno on /dev/cu.usbmodem1411".

Abbildung 14  
Code der Simulation

### 3.3.2 Raspberry Pi

Von Andreas Gerken

Zur Emulation der Funktionalität des Air Swimmers ist eine Möglichkeit einen Raspberry Pi (RPi) zu verwenden. Das ist ein kleiner, reduzierter aber auch vollwertiger Computer, der dazu gedacht ist, dass man auf ihm Sachen ausprobieren kann ohne Angst haben zu müssen, dass man etwas kaputt macht. Auf dem Rechner läuft eine spezielle Linux Variante (Raspbian; abgewandelt von Debian) die auf die Leistungsmerkmale des Rechners zugeschnitten ist. Auf der Platine ist ein 700MHz ARM-Prozessor und 512MB Arbeitsspeicher verbaut. Die Platine verfügt über zwei USB Anschlüsse, einen HDMI ausgang, einen Ethernetsteckplatz und weitere Verbindungen. Vorallem aber einen „GPIO“ = General Purpose Input Output Steckplatz mit insgesamt 26 Pins über die Peripherie wie Sensoren, Servos, Motoren oder Ähnliches angeschlossen werden können. Somit könnte man über diesen Steckplatz auch einen Infrarot Empfänger anschließen und LEDs oder Motoren. Somit könnte man die Aktivitäten des Fisches Emulieren. Der RPi könnte auf die gleichen Signale reagieren, die im LIRC File definiert sind und auf die Signale mit leuchten der LED/ drehen der Motoren reagieren. Damit könnte auch bei Ausfallen des Fisches der Erfolg des Projekts demonstrativ gezeigt werden.



**Abbildung 15**

Die Platine des Raspberry Pi's die 26 Pins auf der linken Seite der Abbildung sind die GPIO Pins

Da das Betriebssystem Raspbian nur eine reduzierte Linux Variante ist, unterstützt es leider nicht wie die meisten Linux Distributionen. Das Programm kann auch nicht aus einem Repository nachgeladen werden sondern muss in den Linux Kernel eingebaut werden und das komplette Betriebssystem muss neu kompiliert werden. Dazu gibt es eine umfangreiche Anleitung im Internet, die Schritt für Schritt durchgeführt wurde. Die Lirc Komponente konnte zwar eingebaut werden, jedoch dauerte der Kompiliervorgang sehr lange (da das gesamte Betriebssystem gebaut werden muss).

Dadurch ist die Fehlersuche sehr langwierig und es ist sinnvoller den Arduino als Emulationsplatine zu verwenden.

## 4 Projektleitung

### 4.1 Erstellen eines Projektplans

#### Ziel

Erstellen eines realistischen Zeitplans für die gesamte Projektphase.

#### Vorgehen

Zuerst wurde nach einer Möglichkeit gesucht einen Projektplan zu entwickeln und übersichtlich darzustellen. Da an der Hochschule das Programm „MS Project“ vorhanden ist und dieses zudem in einer anderen Vorlesung im Laufe des Semesters behandelt werden sollte, wurde dieses ausgewählt. Dort ist es unter anderem möglich verschiedene Aktivitäten und Meilensteine einzutragen, deren Abhängigkeiten zu definieren und alles graphisch darzustellen.

Um einen groben Überblick über das Projekt zu erhalten wurden zu Beginn alle bekannten Elemente eingetragen:

Bekannte Daten:

- Projektbeginn: 19.3.2013
- Projektende: 25.6.2013
- Bekannte Aktivitäten
  - o Vorbereitungen
  - o Entwicklung App
  - o Senden
  - o Oberfläche
    - Tasten
    - Gesten
    - Flugmanöver
    - Pacour
  - o Konzept für Sensorik

Die bekannten Aktivitäten wurden anschließend, soweit möglich, in kleinere Aufgaben unterteilt und deren Zeitdauer geschätzt. Realistische Zeiten abzuschätzen gestaltete sich schwierig, da nicht alle Tätigkeiten bekannt waren und bei vielen Tätigkeiten der Aufwand nicht vorherzusehen war. Zudem führten unterschiedliche Vorkenntnisse der Teammitglieder zu zeitlichen Schwankungen abhängig von den bearbeitenden Personen. Da jede Woche eine Projektbesprechung stattfindet, wurden zur Vereinfachung die meisten Vorgänge auf ganze Wochen ausgelegt.

|    |  | Vorgang | Vorgangsname                    | Dauer   | Anfang       | Ende         |
|----|--|---------|---------------------------------|---------|--------------|--------------|
| 3  |  |         | ▷ Vorbereitung                  | 21 Tage | Die 19.03.13 | Die 16.04.13 |
| 4  |  |         | Einrichten Versionsverwaltung   | 4 Tage  | Die 19.03.13 | Fre 22.03.13 |
| 5  |  |         | IR Sender besorgen              | 16 Tage | Die 19.03.13 | Die 09.04.13 |
| 6  |  |         | Senden analysieren              | 21 Tage | Die 19.03.13 | Die 16.04.13 |
| 7  |  |         | Testoberfläche mit Pfeilen      | 6 Tage  | Die 19.03.13 | Die 26.03.13 |
| 8  |  |         | ▷ App                           | 51 Tage | Die 19.03.13 | Die 28.05.13 |
| 9  |  |         | Konzept App                     | 21 Tage | Die 19.03.13 | Die 16.04.13 |
| 10 |  |         | fertiges Konzept mit IR         | 0 Tage  | Don 18.04.13 | Don 18.04.13 |
| 11 |  |         | Senden implementieren           | 14 Tage | Mit 10.04.13 | Mon 29.04.13 |
| 12 |  |         | AirSwimmer reagiert auf Befehle | 0 Tage  | Die 30.04.13 | Die 30.04.13 |
| 13 |  |         | Steuerung mit Pfeilen fertig    | 0 Tage  | Die 07.05.13 | Die 07.05.13 |
| 14 |  |         | Oberfläche Gesten               | 14 Tage | Mit 17.04.13 | Mon 06.05.13 |
| 15 |  |         | Steuerung über Gesten           | 14 Tage | Die 30.04.13 | Fre 17.05.13 |
| 16 |  |         | Steuerung mit Gesten fertig     | 0 Tage  | Die 14.05.13 | Die 14.05.13 |
| 17 |  |         | Flugmanöver                     | 16 Tage | Die 07.05.13 | Die 28.05.13 |

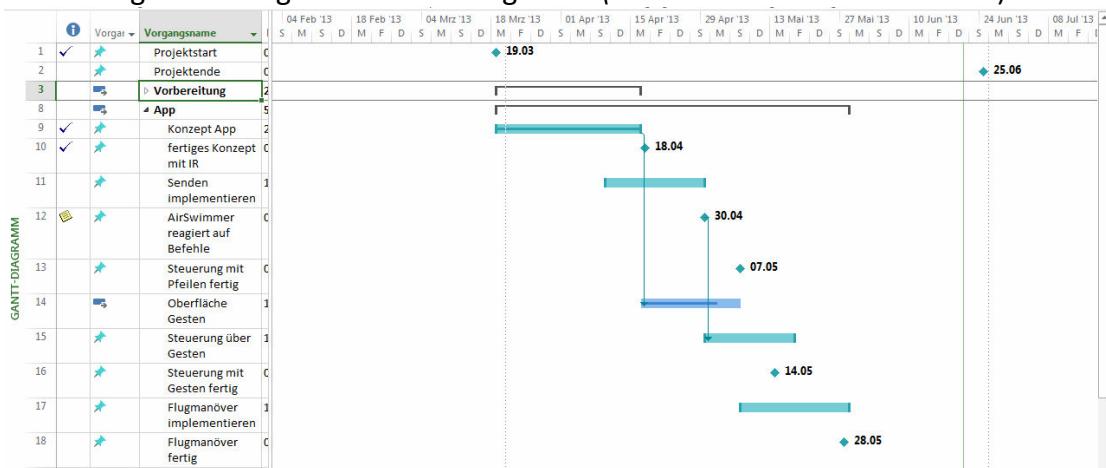
BEREIT NEUE VORGÄNGE : MANUEL GEPLANT

**Abbildung 16**

Ausschnitt Vorgänge und Zeiten des Projekts

Zur Darstellung der tatsächlichen Projektdauer wurden anschließend noch Abhängigkeiten zwischen den einzelnen Vorgängen (z.B. Beschaffung IR-Sender u. Senden implementieren) definiert, sowie bei Häufung vieler Aktivitäten einzelne verschoben.

Daraus ergab sich folgendes Gantt-Diagramm (*Bild bereits mit Meilensteinen*):



**Abbildung 17**

Gantt-Diagramm AirSwimmer-Projekt

## Meilensteine



**Abbildung 18**

Meilensteine des Projekts

Als erste Meilensteine sind ein fertiges Konzept für die Applikation einschließlich Senden der Infrarot-Befehle, sowie eine erste Reaktion des Fisches auf die Befehle vom Tablet definiert. Diese sollen sicherstellen, dass bei großen Schwierigkeiten der Infrarotübertragung rechtzeitig eine andere Alternative verwendet wird.

Von den oben genannten Unterpunkten der Oberfläche sollen je nach Verlauf des Projekts mehr oder weniger implementiert werden. Um dies besser überwachen zu können gibt es für die Fertigstellung jeder Oberfläche einschließlich dazugehöriger Logik einen Meilenstein. Die Lücke zwischen dem letzten Meilenstein und dem Projektende sollte zuerst für die Entwicklung eines Konzepts zum Ausstatten des AirSwimmers mit Sensorik genutzt werden. Da sich aber bereits zu Beginn des Projektes abzeichnete, dass dazu die Tragfähigkeit des Ballons nicht ausreichen würde, wird dieser Zeitraum als Puffer genutzt. Um die zu erreichenden Meilensteine übersichtlich darzustellen, werden diese auf der Zeitachse dargestellt (Abbildung 18 Meilensteine des Projekts).

## Ergebnis

Der fertige Projektplan befindet sich im Anhang.

## 4.2 Projektbesprechungen

Während der Bearbeitung des Projekts findet einmal wöchentlich eine Projektbesprechung statt, zu der jede Woche ein anderer ein Protokoll erstellt. Diese Protokolle wurden in einem eigenen Ordner auf dem Server abgelegt und befinden sich im Anhang. In den Besprechungen wird der aktuelle Stand aller laufenden Tätigkeiten, neue Anforderungen und Organisatorisches besprochen sowie neue Arbeiten für die folgende Woche verteilt. Die Aufgabe der Projektleitung besteht hier darin, die Besprechungen zu leiten. Zu Beginn des Projekts besteht das Problem, dass aufgrund eines fehlenden roten Fadens häufig zwischen den einzelnen Themen gewechselt wird und unwichtige Themen thematisiert werden. Deshalb wird später für jeden Termin eine feste Tagesordnung aufgestellt, in der immer zuerst der aktuelle Statusbericht stattfindet und danach die Aufgaben verteilt werden. Wenn vorhanden wird zu Beginn oder am Ende noch Organisatorisches besprochen. Aufgrund erster Erfahrungen werden längere Diskussionen einzelner Teilnehmer auf nach der Besprechung verschoben, wodurch die Aufmerksamkeit der anderen Teilnehmer etwas gesteigert werden konnte. Durch diese Änderungen konnten die Termine besser strukturiert und verkürzt werden.

### Statusbericht

Um den aktuellen Status des Projekts zu ermitteln, berichtet jedes Team was in der letzten Woche erreicht wurde und welche Aufgaben noch übrig wären. Dabei muss entschieden werden ob diese weiteren Aufgaben erledigt werden, oder wichtigere Themen anstehen.

### Aufgabenverteilung

Hat jedes Team seinen Stand berichtet wird entschieden, wer seine bisherigen Aufgaben weiter führt, welche Teams durch weitere Leute unterstützt werden, und wer neue Aufgabenpakete beginnt. Bei der Verteilung der neuen Aufgaben ist zudem darauf zu achten, dass die gearbeiteten Stunden aller Teammitglieder am Ende ungefähr ausgeglichen sind. Da das Projekt zu vielen Zeitpunkten zu wenig Arbeit bietet, und zu große Teams für einzelne Aufgaben zu Problemen führen würden, werden immer wieder Teilaufgaben gefunden, die zusätzlich von Einzelnen bearbeitet werden können (z.B. Features der Oberfläche). Während dem Großteil des Projektes war ca. die Hälfte des Teams mit der tatsächlichen Programmierung der App beschäftigt, während die andere Hälfte die Analyse der Infrarot-Signale und andere anstehende Aufgaben übernahm. Da diese Aufgaben jedoch zum Ende hin weniger wurden, mussten hier Aufgabenpakete an der App gefunden werden, bei denen die Einarbeitungszeit gering ist. Um größere Unterschiede auszugleichen mussten manche zusätzliche Aufgaben übernehmen, während andere in manchen Wochen pausieren mussten.

## 4.3 Weitere Aufgaben

### Überwachung des Projektstatus

Während der gesamten Zeit wurde der Fortschritt des Projekts, sowie die im Abschnitt "Erstellen eines Projektplans" beschriebenen Meilensteine, überwacht. Die ersten beiden Meilensteine (Konzept Senden/Fisch reagiert) wurden termingerecht erreicht, wodurch keine Alternative zum AirSwimmer benötigt wurde. Danach kam es zu Verzögerungen, da unter anderem aufgrund von Schwächen in der Infrarot-Übertragung neue Aufgaben hinzukamen und Schwierigkeiten beim Senden auftraten. Zudem war der AirSwimmer kurz vor Fertigstellung der Implementierung des Sendens, einige Zeit nicht funktionsfähig. Während dieses Ausfalls konnte nicht getestet werden und es mussten Alternativen gefunden werden, falls vor Projektende kein Ersatz beschafft werden konnte. Aus diesem Grund wurde die eingeplante Pufferzeit aufgebraucht, sowie der Funktionsumfang der App gekürzt.

### Projektcontrolling

Da anfangs in den Projektbesprechungen oft Probleme aufkamen, die von anderen sehr schnell gelöst werden konnten, wurde später zwischen den Projektbesprechung öfter der Status der einzelnen Teams abgefragt und wenn nötig zwischen den Teams vermittelt. Stellten sich einzelne Arbeitspakete als leichter oder aufwändiger heraus, wurden wenn möglich Aufgaben anders verteilt oder der Umfang der Aufgaben verringert.

Um die Einarbeitungszeit in die App gering zu halten, wird regelmäßig der Code strukturiert und überprüft, ob im Source-Code der funktionierenden App Kommentare vorhanden sind. Ebenfalls wurde überprüft, ob Kommentare und Code, wie zu Beginn des Projekts festgelegt, auf Englisch geschrieben wurde.

### Organisatorisches

Für die richtige Verteilung der Aufgaben ist es nötig, dass die Zeiterfassung aktuell ist. Deshalb wurde regelmäßig die Aktualität geprüft, sowie bei Bedarf einzelne Personen daran erinnert, diese zu aktualisieren. Für die Nachvollziehbarkeit der Stunden geschah dies stichprobenartig auch für die Beschreibung der Tätigkeiten in der Zeiterfassung.

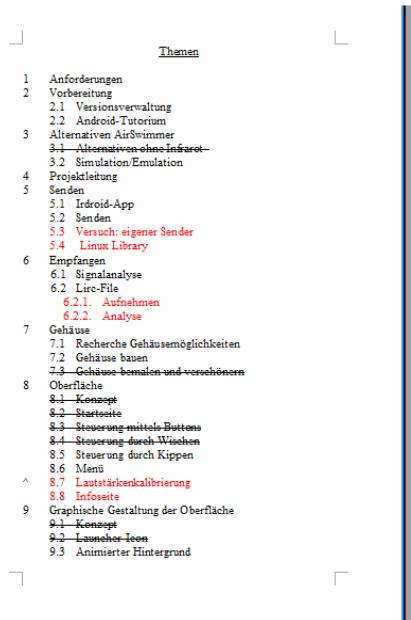
### Dokumentation

Für die Abschlussdokumentation wurde ein eigener Ordner auf dem Server erstellt, in den jeder seine Teile der Dokumentation legt. Diese Teile werden anschließend von der Projektleitung zu einer einheitlichen Dokumentation zusammengefügt. Auf diese Weise wurde verhindert, dass der gleichzeitige Zugriff mehrerer Personen auf eine Datei zu Konflikten führt. Die Erstellung der gesamten Dokumentation findet gleichzeitig mit den einzelnen Themen statt, somit werden neue Themen laufend eingefügt. Damit jeder einen Überblick über den aktuellen Stand der gesamten Dokumentation hat, werden bereits eingefügte Teile im Namen markiert (Abbildung 19)

| Bibliothek "Dokumente"   |                  |                     |          |
|--|------------------|---------------------|----------|
| Anordnen nach:   |                  |                     |          |
| Name   | Änderungsdatum   | Typ                 | Größe    |
| [eingefügt]2c Alternativensuche.odt  | 20.06.2013 20:33 | OpenDocument-T...   | 335 KB   |
| [eingefügt]6_a Konzept zur Oberflächensteuerung.doc                                | 20.06.2013 20:32 | Microsoft Word-D... | 25 KB    |
| [eingefügt]6_h Erstellen von Grafiken für die Oberfläche.doc                       | 20.06.2013 20:33 | Microsoft Word-D... | 554 KB   |
| [eingefügt]6_b Oberfläche mit Buttons.odt  | 20.06.2013 20:32 | OpenDocument-T...   | 402 KB   |
| [eingefügt]6_d Oberfläche mit Wischen.odt  | 20.06.2013 20:32 | OpenDocument-T...   | 2.146 KB |
| [eingefügt]8.7 Lautstärkenkalibrierung.odt   | 23.06.2013 01:02 | OpenDocument-T...   | 987 KB   |
| [eingefügt]8.6 Oberfläche mit Buttons.doc  | 20.06.2013 20:32 | Microsoft Word-D... | 182 KB   |
| [eingefügt]8.6_a Buttons der Startseite.doc  | 20.06.2013 20:32 | Microsoft Word-D... | 858 KB   |
| [eingefügt]Kapitel 5 Gehäuse.doc   | 20.06.2013 20:33 | Microsoft Word-D... | 141 KB   |
| [eingefügt]Kapitel 5.1 Konzeptentwicklung für das Oberflächendesign.doc            | 20.06.2013 20:33 | Microsoft Word-D... | 2.669 KB |
| [eingefügt]Kapitel 5.2 Tasten - Druck und visueller Effekt.doc                     | 20.06.2013 20:33 | Microsoft Word-D... | 31 KB    |
| [eingefügt]Kapitel 5.5 Erstellen der Startseite und Einbinden der Benutzermodi.doc | 20.06.2013 20:33 | Microsoft Word-D... | 741 KB   |
| [eingefügt]Kapitel 5.6 Menü - Steuerungsart ändern.doc                             | 20.06.2013 20:33 | Microsoft Word-D... | 291 KB   |
| [eingefügt]Kapitel 5.7_1 Launcher Icon.doc   | 20.06.2013 20:33 | Microsoft Word-D... | 104 KB   |
| Bericht AndreasWeber.docx  | 20.06.2013 23:00 | DOCX-Datei          | 2.184 KB |
| Dokumentation AirSwimmer.pdf   | 22.06.2013 22:49 | Adobe Acrobat D...  | 2.177 KB |
| info.tu  | 20.06.2013 20:33 | Textdokument        | 1 KB     |
| Kapitel 2 Git.doc  | 25.06.2013 11:21 | Microsoft Word-D... | 878 KB   |
| Kapitel 2 Vorbereitungen.doc   | 20.06.2013 20:33 | Microsoft Word-D... | 67 KB    |
| Kapitel 2.6.2 LIRC analyse.doc   | 25.06.2013 11:21 | Microsoft Word-D... | 135 KB   |
| Kapitel 10.3 Permanente Bewegung.doc   | 24.06.2013 21:10 | Microsoft Word-D... | 575 KB   |
| Themen.doc   | 25.06.2013 11:21 | Microsoft Word-D... | 14 KB    |

**Abbildung 19**  
Ordnerstruktur „Dokumentation“

Um den Inhalt der Dokumentation festzulegen, wurde anhand der Protokolle und Besprechungen eine Themensammlung erstellt, die immer wieder erweitert wurde. Diese Sammlung umfasste Themen aus den Bereichen Vorbereitung, Signalanalyse, Senden und Oberfläche. Die Verteilung der Unterthemen dieser Bereiche sollte von den beteiligten Personen untereinander abgesprochen werden. Da dies leider nicht überall funktionierte, mussten am Ende noch Missverständnisse und Probleme bei Überschneidungen gelöst werden. Für eine übersichtliche Darstellung wurden neue Themen farblich markiert und bereits eingefügte und vollständig geklärte Themen durchgestrichen.



**Abbildung 20**

Zwischenstand der Themensammlung für die Dokumentation

## 5 Empfangen

### 5.1 Analyse der Übertragungsstrategie

Von Andreas Weber

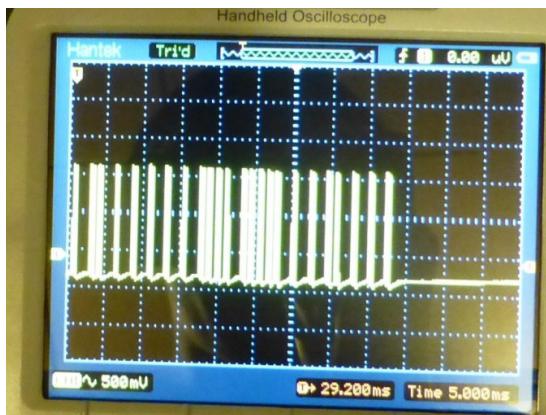
#### Signalaufbau

IR-Fernbedienungen senden ein Signal im unsichtbaren Infrarotbereich aus. Als Strahlungsquelle dienen häufig Infrarotleuchtdioden. Das Signal wird mit einer Frequenz um 40 kHz aus- und eingeschaltet. Dadurch erhöht sich die Störsicherheit des Empfängers: Ein Bandpassfilter lässt nur diese Frequenzen passieren und sperrt zufällige Störsignale aus. Durch Modulation dieses Sendesignals werden Informationen zum Empfänger übertragen.



**Abbildung 21**

Grundfrequenz mit 36 – 40 kHz (gleich bleibende Frequenz eines „1“ Impulses)



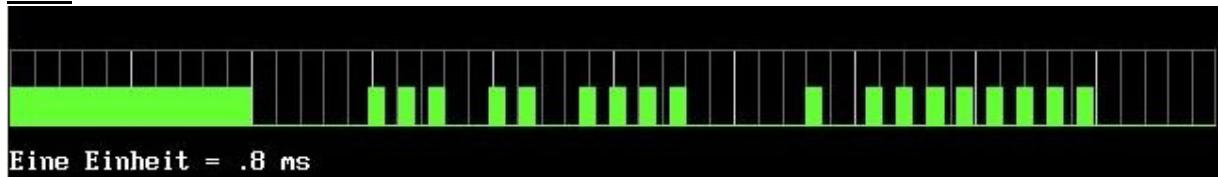
**Abbildung 22**

Grundfrequenz wird passend ein und ausgeschaltet  
(Je nach Codierungsverfahren)

### **Codierverfahren der IR Übertragung**

Es gibt wohl nahezu so viele Codierverfahren, wie Hersteller von IR Geräten, leider kein einheitlicher Standard.

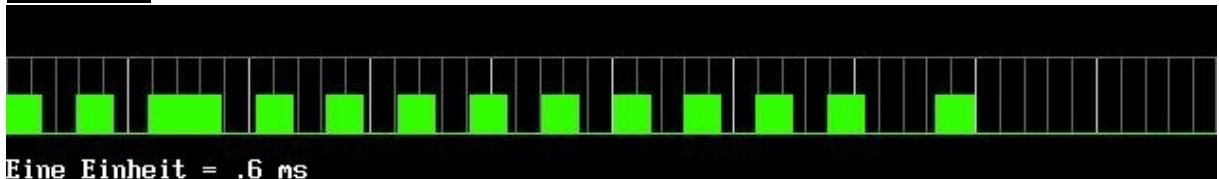
x-Sat



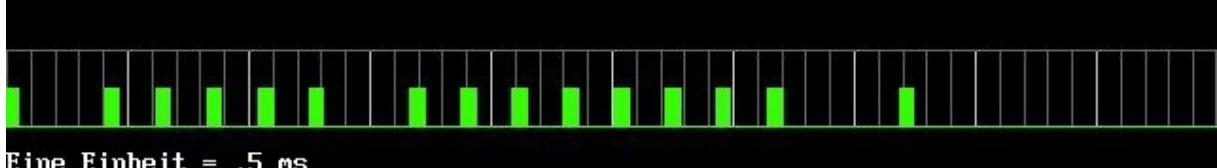
Gedrückt wurde die Taste "1" fürs erste Programm. Man kann sehr schön ablesen, dass zu Beginn ein 8ms langer Impuls gesendet wird, gefolgt von einer 4ms-Lücke. Das ist eindeutig das auf der oben angesprochenen Seite als "X-Sat" bezeichnete Protokoll. Dem Startsignal folgen zwei Gruppen kürzerer Impulse, wieder getrennt durch eine 4ms-Pause. Bei diesem Protokoll sind jeweils die ersten acht dieser Kurzen Impulse das Byte, das den Gerätecode enthält, der Letzte ist sozusagen ein Stoppimpuls. Ein Impuls mit kurzer Pause danach ist eine Null, einer mit langer Pause eine Eins, gelesen wird von links nach rechts. Der Gerätecode ist also 20. Das zweite Byte ist 1, was der Code für Taste 1 ist. Auch hier ist der 9. Impuls der Stoppimpuls.

Viele Protokolle arbeiten mit dieser Aufteilung in Gerät- und Funktionscode. So stören sich die Fernbedienungen z.B. von Fernseher und Videorecorder des gleichen Herstellers nicht, auch wenn sie ihre Daten nach dem gleichen Verfahren übertragen. Hier noch zwei weitere Beispiele, bei beiden wurde jeweils die Taste "1" gedrückt:

RC-5 Code:



Sharp Code:



Quelle: Skilltronics.de [http://www.skilltronics.de/versuch/elektronik\\_pc/ir.html](http://www.skilltronics.de/versuch/elektronik_pc/ir.html)

Bei dem von unserem AirSwimmer Fernbedienung verwendeten Codierung handelt es sich jedoch um ein Unbekanntes Codier verfahren, über das keine weiteren Informationen gefunden wurden. Bei diesem Codier verfahren werden die Informationen rein über die Länge des „0“ Impulses Codiert, es gab keine Messbare Startsequenz.

## 5.2 Aufnehmen des eigenen Infrarotsignals

*Von Andreas Weber*

Im Internet wurde nach einer Möglichkeit gesucht, das eigene Infrarotsignal aufzunehmen. Dazu wurden mehrere Möglichkeiten gefunden. Für Apple Geräte gibt es einen fertigen Aufsatz mit dem unter anderem auch die Hochschule Rosenheim bereits den AirSwimmer gesteuert hat und dazu eine passende app geschrieben hat. Diese Version ist sehr einfach, da das Gerät und die beiliegende Software alles selbst übernimmt, es ist für die Kommunikation kein weiterer Programmieraufwand nötig.



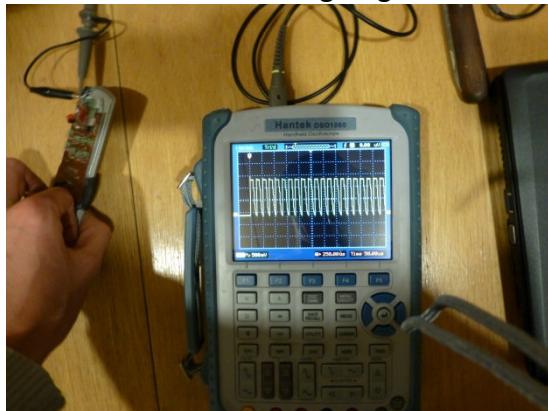
Im Internet gibt es weitere ausgereifte Geräte, um infrarot Signale aufzunehmen. Leider sind diese recht teuer und deshalb wurde eine günstigere Variante gesucht.

Bei der Suche sind wir auf den so genannten TSOP Empfänger gestoßen. Dieser wird an 5V Gleichspannung angeschlossen und wandelt das empfangene Lichtsignal in eine digitale Spannung um. Außerdem filtert er bereits die Grundfrequenz von 40HZ heraus.

### **5.2.1 Versuch 1: Oszilloskop**

Von Andreas Weber

Zum Untersuchen eines Signals, in diesem Fall ein Spannungssignal dass eine Infrarote Leuchtdiode (LED) ansteuert, kann ein handelsübliches Oszilloskop hergenommen werden. Dies wird einfach zu den beiden Anschlüsse der Infrarot LED parallel angeschlossen. Dazu wurde eine Universalfernbedienung, die auf nahezu jedes Fernsehgerät angelernt werden kann, verwendet. Diese wurde dazu zerlegt, und in den Versuchsaufbau eingefügt:



**Abbildung 23**

Versuchsaufbau „Signalaufnahme mit Oszilloskop“

Anschließend konnte bereits eine Vorabanalyse des Signals vorgenommen werden. Leider war die Auflösung des Oszilloskops nicht ausreichend genug, bzw. das Fenster in dem Das Signal angezeigt wurde nicht breit genug, deswegen wurde nach einer Möglichkeit gesucht, das Signal über einen längeren Zeitraum (ca. 0.5s) aufzuzeichnen.

Im Internet werden solche Signale über den Mikrofoneingang des Notebooks aufgezeichnet. Mit geeigneter frei verfügbarer Software, kann das Signal dann als Spannungsamplitude über die Zeit angesehen werden.

Das Problem dabei ist, dass der Mikrofoneingang hardwaremäßig auf der Soundkarte bereits über entsprechende Filterschaltungen verfügt, und somit alles was nicht im hörbaren Bereich liegt (also von der Frequenz kleiner als ca 20kHz) herausfiltert. Da das Infrarotsignal über eine Grundfrequenz von 40kHz verfügt, kann es zu falschen Messpunkten kommen.

### **5.2.2 Versuch 2: Raspberry Pi**

Von Andreas Gerken

Wie in 2.3 (Emulation) beschrieben, kann über die GPIO Pins das Signal des Infrarotempfängers über LIRC analysiert werden. Es kann jedoch nicht nur zur Erkennung der Befehle verwendet werden, sondern genauso um ein Signal aufzunehmen. Parallel zur Bearbeitung der anderen Lösungsmöglichkeiten (Analyse per Oszilloskop, WinLirc) wurde auch für die Aufnahme, LIRC in das Raspbian Betriebssystem eingebaut. Dabei gab es zunächst zwei relativ „einfache“ Probleme:

- Bildschirme werden normalerweise über den HDMI-Port an den Raspberry PI angeschlossen. Die Bildschirme in der Hochschule haben aber keinen solchen Eingang. Die Lösung stellte ein HDMI-DVI Adapter von Herr Schäffer dar.
- Um die benötigten Programme zu installieren benötigt man einen Internetzugang. In der Hochschule gibt es aber keine frei zugänglichen Ethernet Stecker, an die man den Raspberry anschließen könnte. Es gelang über einen WLAN Stick, dessen Einrichtungsumgebung aber auch nicht auf die Authentifizierung mit Username und Passwort ausgelegt war.

Diese Verzögerungen führten dazu, dass kaum noch Zeit war um LIRC einzurichten und die Teammitglieder die es mit WinLirc probiert haben schneller zu einem Ergebnis gekommen sind.

### 5.2.3 Versuch 3: Soundkarte

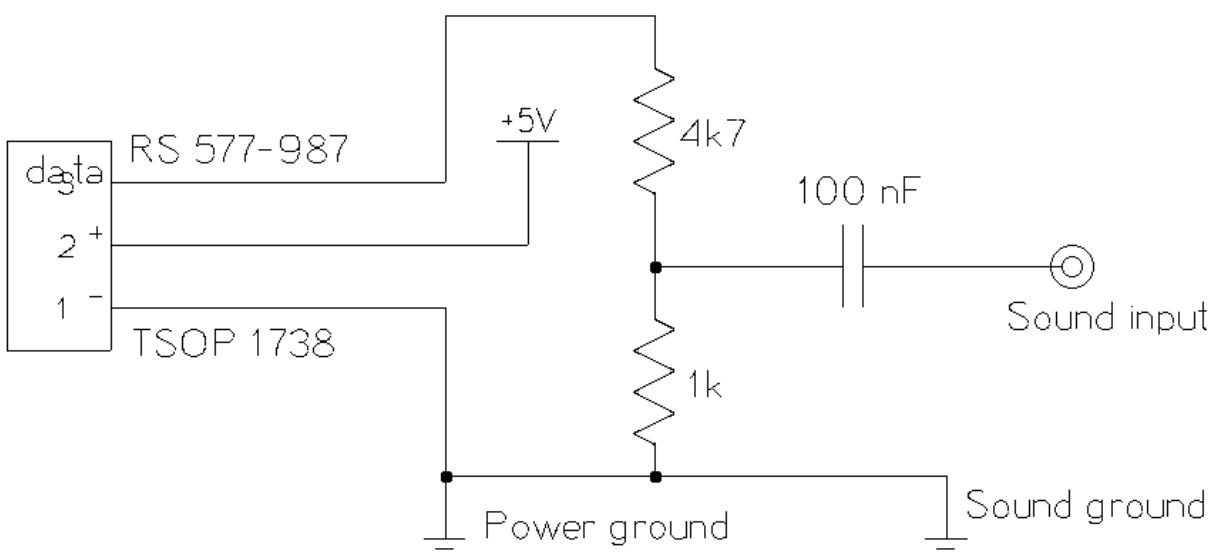
Von Giuseppe De Nuzzo und Ridvan Yücel

In diesem Abschnitt wird die Aufzeichnung der Infrarotsignale über die Soundkarte eines Computers beschrieben. Resultat dieser Aufzeichnung ist ein Audio-File im Wav-Format.

Für den Aufbau der Schaltung werden folgende Bauteile benötigt:

- Ein Infrarotempfänger wie z.B. TSOP 1738
- Zwei Widerstände  $4k7\ \Omega$  und  $1k\ \Omega$
- Ein Kondensator  $100\ nF$
- Ein 3,5mm Klinkenstecker für die Soundkarte

Die zwei Widerstände werden in Reihe geschalten um somit einen Spannungsabfall am Ausgangssignal zu erzeugen. Der hier eingesetzte Kondensator eliminiert den Gleichspannungsanteil. Als Spannungsversorgung wird der USB-Port des Computers verwendet.

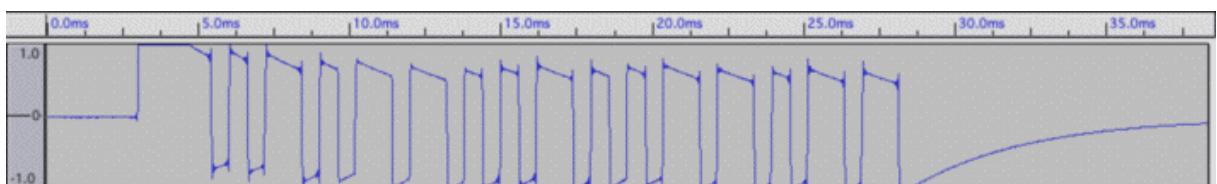


**Abbildung 24**

Schaltplan der oben beschriebenen Schaltung

## **Vorgehensweise zur Aufzeichnung**

Nachdem die Spannungsversorgung durch den USB-Port hergestellt und die Schaltung über den Mikrofoneingang der Soundkarte verbunden wurde, kann mit der Aufzeichnung der Infrarotsignale begonnen werden. Dazu wurde das bereits in Windows vorinstallierte Programm Audiorecorder verwendet. Die nun gesendeten Infrarotsignale werden vom Infrarotempfänger verarbeitet und an die Soundkarte weitergegeben. Das erzeugte Wav-File enthält ein Rauschen für jedes eingelesene Infrarotsignal.



**Abbildung 25**

Verlauf eines aufgezeichneten Audiosignals

### **5.2.4 Versuch 4: Schnittstelle des Computers (RS232)**

*Von Andreas Weber, Giuseppe De Nuzzo, Ridvan Yücel*

Eine bessere Möglichkeit Infrarotsignale zu analysieren, besteht darin ein LIRC-File zu erzeugen. Dazu wird eine alternative TSOP-Schaltung benötigt.

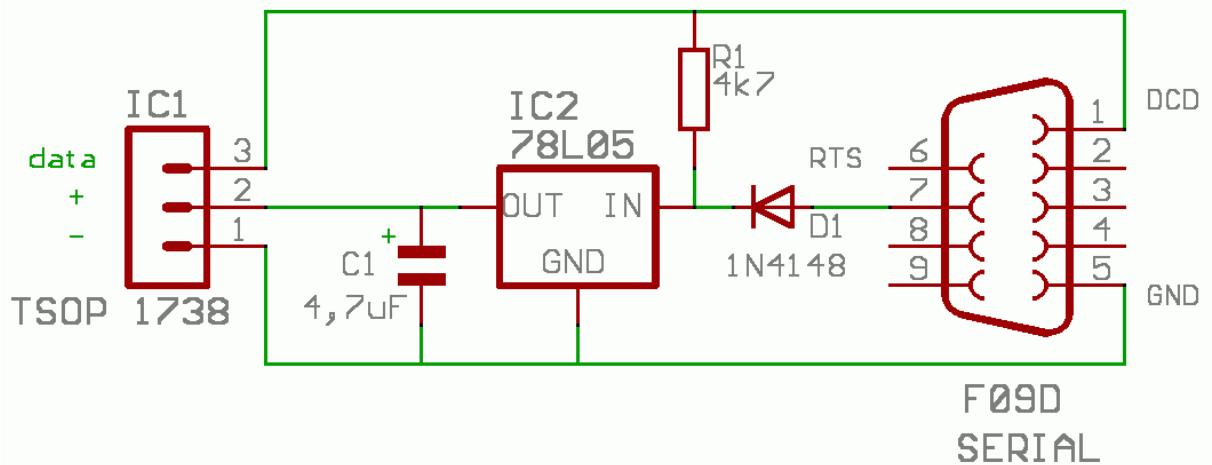
Für den Aufbau der Schaltung werden folgende Bauteile verwendet:

- Infrarot Empfänger-Modul TSOP 1738
- IC 78L05 (Linearer Spannungsregler 5V)
- Kondensator 4,7 $\mu$ F
- Widerstand 4,7k
- Diode 1N4148
- RS232-Kabel und einen seriellen Anschluss

#### **Pinbelegung RS232**

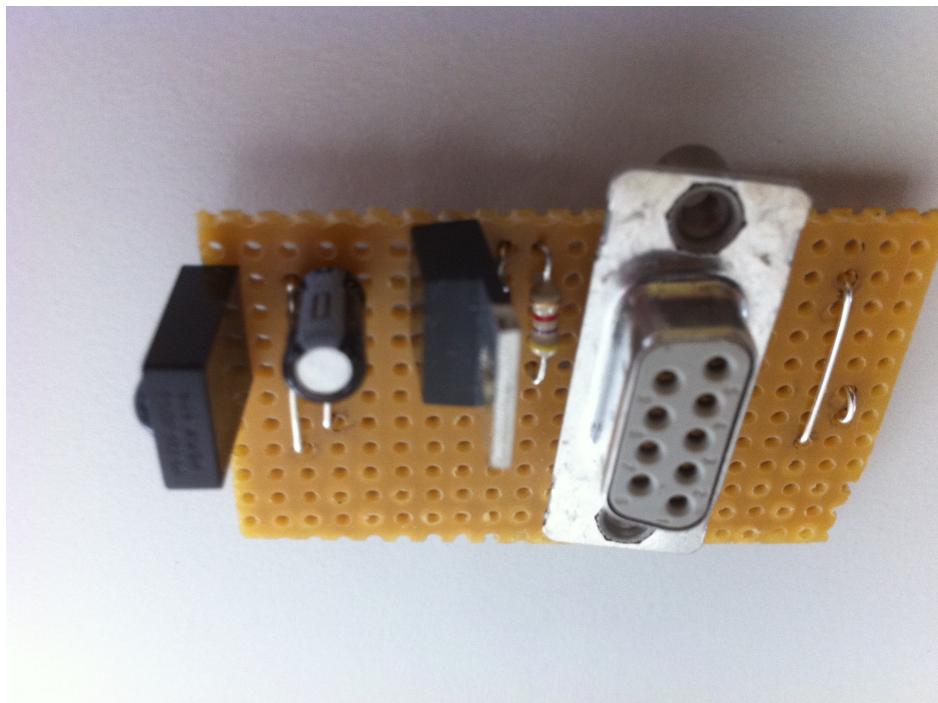
---

|     |   |   |
|-----|---|---|
| TxD | 3 | transmit data                           |
| RxD | 2 | receive data                            |
| RTS | 7 | request to send (here: power source)    |
| CTS | 8 | clear to send                           |
| DSR | 6 | data set ready                          |
| GND | 5 | ground                                  |
| DCD | 1 | data carrier detect (here: signal line) |
| DTR | 4 | data terminal ready                     |



**Abbildung 26**  
Schaltplan TSOP-Schaltung

Die Spannungsversorgung erfolgt über den 7. Pin (RTS) des seriellen Ports. Die Diode schützt die Schaltung vor negativen Spannungen, die am RTS-Pin auftreten können. Der LIRC-Treiber versorgt die Schaltung mit der notwendigen Spannung. Standardmäßig beträgt diese Spannung 10V. Der Spannungsregler IC2 regelt die Eingangsspannung auf konstante 5V. Die Eingangsspannung muss mindestens 2V höher als die Ausgangsspannung betragen, damit eine zuverlässige Arbeitsweise des Bauteils gewährleistet ist. Daher sollte sichergestellt werden, dass die serielle Schnittstelle mindestens 8V liefert. Bei einer geringeren Ausgangsspannung muss ein alternativer Spannungsregler eingesetzt werden. Der Kondensator C1 dient zur Stabilisierung der Spannung und verhindert kurzzeitige Spannungseinbrüche. Der Widerstand R1 ist als Pullup-Widerstand verschaltet und stellt somit sicher, dass der 1. Pin (DCD) mit einem gültigem Spannungspiegel versorgt wird. Dies ist dann der Fall, wenn keine Infrarotsignale empfangen werden. Wird ein Infrarotsignal erkannt, so setzt der Infrarotempfänger die Spannung auf GND. Der DCD-Port interpretiert dieses Signal als logische „1“ (DCD = 0).



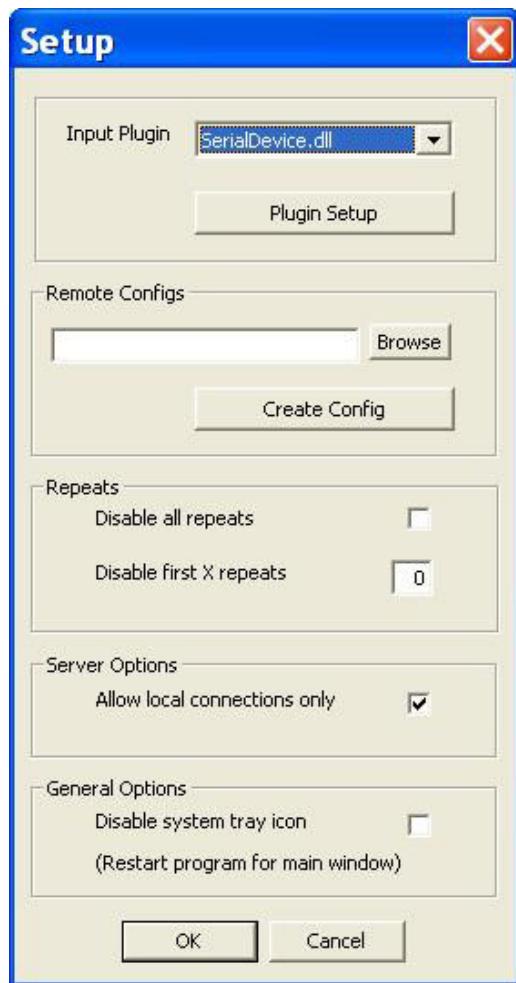
**Abbildung 27**  
Platine TSOP mit RS232

### **Vorgehensweise zur Erzeugung eines LIRC-Files:**

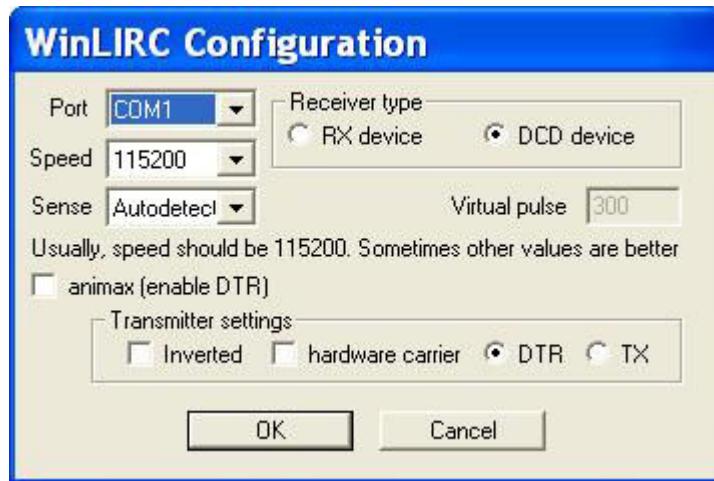
*Von Andreas Weber, Giuseppe De Nuzzo, Ridvan Yücel*

Benötigt wird hier die Software WinLirc (siehe [www.lirc.org](http://www.lirc.org)) Version 0.9.0e. Nach dem Herunterladen, befindet sich im entpackten Verzeichnis das ausführbare Programm. Eine Installation ist hierbei nicht erforderlich.

Bevor man mit der Aufzeichnung beginnt muss das LIRC-Programm konfiguriert werden:

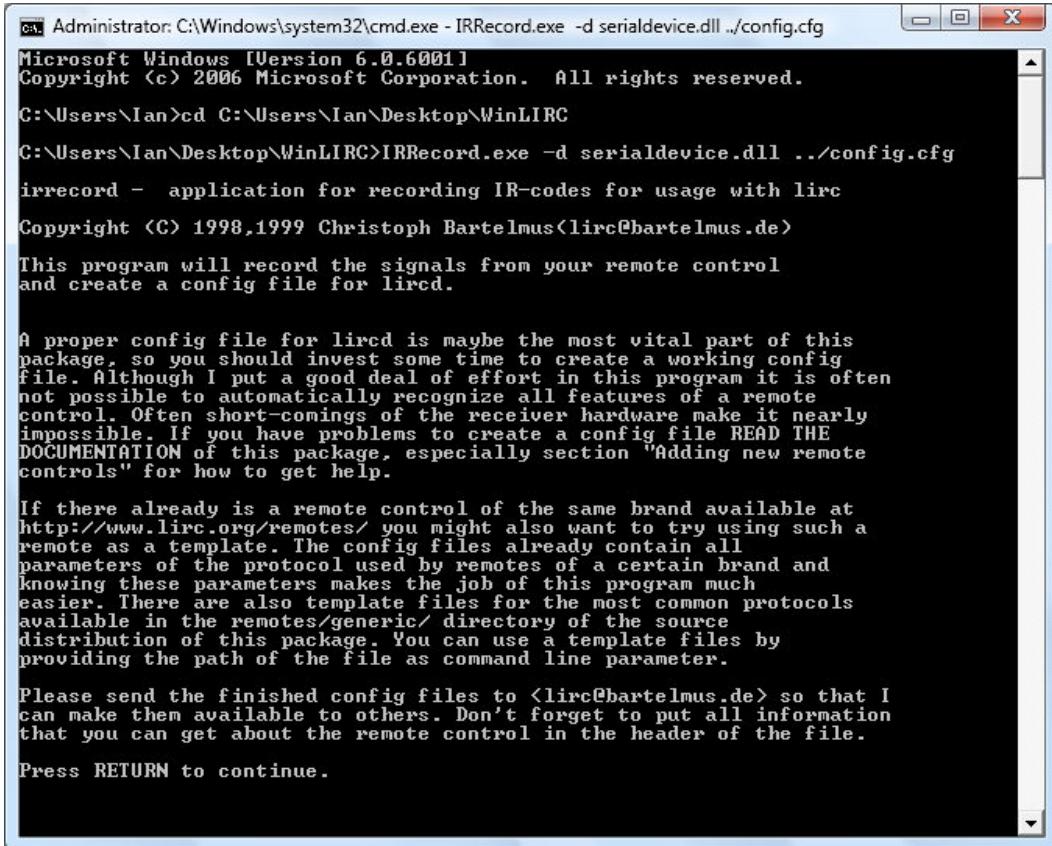


Als erstes erfolgt die Auswahl des Input Plugins, hier muss SerialDevice.dll gewählt werden, da wir über die serielle Schnittstelle mit der TSOP-Schaltung kommunizieren.



Zusätzlich muss sichergestellt werden, dass der richtige COM-Port ausgewählt wurde. Dies kann unter Windows im Geräte-Manager überprüft werden. Die Geschwindigkeit sollte bei 115200 Baud eingestellt werden, falls diese nicht schon standardmäßig vorgegeben ist. Die übrigen Einstellungen sollten für kleinere Anwendungen nicht verändert werden.

Mit dem Button **Browse** kann man nach Auswahl des Verzeichnisses und Eingabe des Dateinamens den Speicherort des LIRC-Files festlegen. Um die Aufzeichnung zu starten, muss auf **Create Config** geklickt werden. Nun wird die IRRecord in der Kommandozeile aufgerufen.



The screenshot shows a Windows Command Prompt window titled "Administrator: C:\Windows\system32\cmd.exe - IRRecord.exe -d serialdevice.dll ./config.cfg". The window displays the following text:

```
Microsoft Windows [Version 6.0.6001]
Copyright <c> 2006 Microsoft Corporation. All rights reserved.

C:\Users\Ian>cd C:\Users\Ian\Desktop\WinLIRC
C:\Users\Ian\Desktop\WinLIRC>IRRecord.exe -d serialdevice.dll ./config.cfg
irrecord - application for recording IR-codes for usage with lirc
Copyright <C> 1998,1999 Christoph Bartelsmus<lirc@bartelsmus.de>

This program will record the signals from your remote control
and create a config file for lircd.

A proper config file for lircd is maybe the most vital part of this
package, so you should invest some time to create a working config
file. Although I put a good deal of effort in this program it is often
not possible to automatically recognize all features of a remote
control. Often short-comings of the receiver hardware make it nearly
impossible. If you have problems to create a config file READ THE
DOCUMENTATION of this package, especially section "Adding new remote
controls" for how to get help.

If there already is a remote control of the same brand available at
http://www.lirc.org/remotes/ you might also want to try using such a
remote as a template. The config files already contain all
parameters of the protocol used by remotes of a certain brand and
knowing these parameters makes the job of this program much
easier. There are also template files for the most common protocols
available in the remotes/generic/ directory of the source
distribution of this package. You can use a template files by
providing the path of the file as command line parameter.

Please send the finished config files to <lirc@bartelsmus.de> so that I
can make them available to others. Don't forget to put all information
that you can get about the remote control in the header of the file.

Press RETURN to continue.
```

Nachdem man zweimal die RETURN-Taste betätigt hat wird mit der Aufzeichnung begonnen. Es wird verlangt, alle aufzuzeichnenden Tasten der Fernbedienung nacheinander zu drücken. Hierbei wird die konstante Signallänge durch Vergleich der unterschiedlichen Signallängen ermittelt. Nach diesem Vorgang wird der erste Schritt wiederholt. Erkennt das Programm das von der Fernbedienung verwendete Protokoll wird zusätzlich eine Headerinformation im LIRC-File hinterlegt. Anschließend wird vom Programm jede Taste einzeln eingelesen. Hierzu gibt man den Namen der gewünschten Taste ein und betätigt diese. Nach erfolgreichem Einlesen wird die Signallänge der jeweiligen Taste angezeigt und die Bitfolge im LIRC-File abgespeichert. Für jede gewünschte Taste wird dieser Vorgang wiederholt. Um die Aufzeichnung zu beenden muss zweimal RETURN betätigt werden.

Die Darstellung der Bitfolge im LIRC-File sowie die Analyse werden in Kapitel 5.3 näher beschrieben.

### **5.2.5 Versuch 5: Arduino**

*Von Giuseppe De Nuzz und Ridvan Yücel*

Ein Arduino Board besteht aus einem Atmel AVR Microcontroller und zusätzlichen Komponenten die Programmierer unterbringen und mit anderen Schaltkreisen zusammenarbeiten. Das Board enthält einen 5-Volt linearen Spannungsregulator und einen 16 MHz Oszillator. Der Microcontroller ist vorprogrammiert mit einem Bootloader, so dass ein externer Programmer nicht notwendig ist. Die Boards werden alle über eine serielle RS232 Verbindung programmiert, hierbei unterscheidet sich nur die Hardwareimplementierung. Aktuelle Boards können allerdings per USB programmiert werden.

Das Arduino Board verfügt über Input/Output Pins, welche an der Front angebracht sind. Es gibt viele Shields, die zusätzlich als Plugin-Boards auf diese Pins montiert werden können (z.B. Ethernet, Wireless).

Die Arduino IDE bietet einen Editor und einen Compiler zur Erstellung von Software. Diese kann kostenlos benutzt werden. Die Programmiersprache basiert auf Wiring, eine C-ähnliche Sprache.

Das Arduino Board stellt eine weitere Möglichkeit dar, Infrarotsignale aufzuzeichnen.

Hierbei wird der bereits in den vorherigen Kapiteln beschriebene Infrarotempfänger TSOP 1738 mit dem Eingang des Boards verbunden.



**Abbildung 28**  
Arduino Board

Die folgenden Abbildungen zeigen den Code, der für die Infrarotsignalanalyse umgesetzt wurde.

```
#include <IRremote.h>

int RECV_PIN = 11;

IRrecv irrecv(RECV_PIN);
IRsend irsend;

decode_results results;

void setup()
{
    Serial.begin(9600);
    irrecv.enableIRIn(); // Start the receiver
}

// Storage for the recorded code
int codeType = -1; // The type of code
unsigned long codeValue; // The code value if not raw
unsigned int rawCodes[RAWBUF]; // The durations if raw
int codeLen; // The length of the code
int toggle = 0; // The RC5/6 toggle state

// Stores the code for later playback
// Most of this code is just logging
void storeCode(decode_results *results) {
    codeType = results->decode_type;
    int count = results->rawlen;
    if (codeType == UNKNOWN) {
        Serial.println("Received unknown code, saving as raw");
        codeLen = results->rawlen - 1;
        // To store raw codes:
```

**Abbildung 29**

Code Infrarotsignalanalyse

Der Eingangspin für die Infrarotdaten wird auf PIN 11 gesetzt und die Baudrate auf 9600. Anhand der Eingangsdaten wird das verwendete Protokoll ermittelt.

```
for (int i = 1; i <= codeLen; i++) {
    if (i % 2) {
        // Mark
        rawCodes[i - 1] = results->rawbuf[i]*USECPERTICK - MARK_EXCESS;
        Serial.print(" m");
    }
    else {
        // Space
        rawCodes[i - 1] = results->rawbuf[i]*USECPERTICK + MARK_EXCESS;
        Serial.print(" s");
    }
    Serial.print(rawCodes[i - 1], DEC);
}
Serial.println("");
}
else {
    if (codeType == NEC) {
        Serial.print("Received NEC: ");
        if (results->value == REPEAT) {
            // Don't record a NEC repeat value as that's useless.
            Serial.println("repeat; ignoring.");
            return;
        }
    }
    else if (codeType == SONY) {
        Serial.print("Received SONY: ");
    }
    else if (codeType == RC5) {
        Serial.print("Received RC5: ");
    }
}
```

Wird kein Standardprotokoll verwendet, so werden die Infrarotsignale als RAW-Codes einem Integer Array zugewiesen.

```
else if (codeType == RC6) {
    Serial.print("Received RC6: ");
}
else {
    Serial.print("Unexpected codeType ");
    Serial.print(codeType, DEC);
    Serial.println("");
}
Serial.println(results->value, HEX);
codeValue = results->value;
codeLen = results->bits;
}

int lastButtonState;

void loop() {
// If button pressed, send the code.

if (irrecv.decode(&results)) {
    storeCode(&results);
    irrecv.resume(); // resume receiver
}
}
```

In der Funktion **void loop** werden die empfangenen Infrarotsignale in Hex-Werte umgewandelt und im Serial Monitor angezeigt. Hält man die Taste länger gedrückt, so wird dies mit dem Befehl „Repeat“ erkannt. Falls ein Protokoll erkannt wurde, wird diese Information ebenfalls angezeigt. Diese Hex-Werte können dann im LIRC-File eingesetzt werden.

## 5.3 Lirc-File

### 5.3.1 Analyse durch WinLirc

Von Andreas Gerken und Andreas Weber

```
begin remote

    name  \\rz-home\ff9315\Desktop\FbLeftSend
    flags RAW_CODES|CONST_LENGTH
    eps      30
    aeps     100

    gap      210903

    begin raw_codes

        name DIVE
        | 309   712   370   657   352   130
        | 360   162   304   718   325   215
        | 285   205   316   707   303   728
        | 298   216   302   715   358   672
        | 332   205   276   215   337   199
        | 297   703   330   695   293   220
        | 298   210   360   159   355   159
        | 324   701   306   712   308   719
        | 317   96223  296   724   308   719
        | 306   205   302   218   302   719
        | 304   211   325   190   289   735
        | 305   777   267   210   285   711
        | 312   719   329   207   315   202
        | 302   182   298   732   304   715
        | 337   203   333   153   312   206
        | 311   202   364   652   339   680
        | 316   717   306

    end raw_codes

end remote
```

**Abbildung 30**

Raw File mit exemplarischem Befehl

Das obere Bild zeigt ein Lirc File, wie es direkt nach dem einlesen aussieht. Die Infrarotsequenz wird dabei nur als „raw code“ aufgezeichnet. Die in der Datei abgespeicherten Werte zeigen immer abwechselnd die Dauer eines High Pegels und eines Low Pegels. Bei der Abbildung wurde nur die Sequenz einer Taste dargestellt. Es handelt sich um die Taste um das Gewicht nach vorne zu bewegen und den Fisch somit nach vorne zu kippen.

Die dargestellten Werte zeigen immer abwechselnd die Dauer eines High Pegels und eines Low Pegels. Man sieht jedoch, dass die Werte stark schwanken und kaum geeignet sind, um erneut gesendet zu werden. Deshalb bietet Lirc die Möglichkeit, Signale in Hexadezimalen Werten abzuspeichern und einheitlich das Signal für ein 1 Bit und ein 0 Bit abzuspeichern.

Um ein Signal vom „Raw Code“ in Hexadezimale Werte umzuwandeln muss die irrecord.exe (in Winlirc inbegriffen) mit dem Attribut –a für Analyse und dem Dateinamen der Raw Datei aufgerufen werden. Das Ergebnis wird auf die Konsole ausgegeben und muss somit noch in ein File umgeleitet werden. Der gesamte Konsolenaufruf sieht folgendermaßen aus:

Irrecord.exe –a rawfile.txt > hexfile.txt

Das Programm sucht intern nach einem Muster in den Pegelzeiten und findet zum Beispiel dass die ersten beiden Signale (309ms, 712ms und 370ms, 657ms) relativ ähnlich sind. Die darauf folgenden Signal (352ms, 130ms und 360ms, 162ms) unterscheidet sich von den ersten Beiden, sind aber untereinander wiederum Ähnlich. Wenn man die beiden ersten beiden Bits als Eins definiert sind die anderen automatisch Nullen. So erhält man  $1100 = 0xC$ . So werden alle Pegelzeiten analysiert und zusammengesetzt. Das Signal aus der Abbildung besteht aus 3 Bytes und wird nach der Umwandlung durch `0xC9B187` beschrieben. Zusätzlich werden alle Pegeldauern einer Art (z.B. der Low Pegel einer 0) gemittelt und abgespeichert. Das neue File ist dadurch viel kompakter und einfacher zu lesen. Außerdem werden die Signale viel genauer beschrieben, weil es keine zeitlichen Schwankungen zwischen den einzelnen Pegeldauern gibt.

### **5.3.2 Analyse und Verbesserung der Lirc Files**

*Von Andreas Gerken*

Die „Hex Files“ sehen folgendermaßen aus:

```

#
# this config file was automatically generated
# using lirc-0.9.0(emulation) on Thu Apr 18 15:38:59 2013
#
# contributed by
#
# brand:                               \\rz-home\ff9315\Desktop\FbLeftSend
# model no. of remote control:
# devices being controlled by this remote:
#



begin remote

    name  \\rz-home\ff9315\Desktop\FbLeftSend
    bits      24
    flags  SPACE_ENC|CONST_LENGTH
    eps       30
    aeps      100

    one       315    709
    zero      315    198
    ptrail    319
    gap       114972
    toggle_bit_mask 0x0

        begin codes
            DIVE           0xC9B187
            CLIMB          0xC9B14B
            TAILRIGHT      0xC9B11E
        end codes

end remote

```

**Abbildung 31**  
Lirc-File als „Hex-File“

In der Abbildung erkennt man einige Daten, die im Folgenden erklärt werden:

- Name: Name unter dem die „raw Datei erstellt wurde“
- Bits: Bits die pro Signal gesendet werden. In unserem Fall 3 Bytes.
- Flags:
  - o SPACE\_ENC: Das Flag heißt, dass Bits durch die Abstände (SPACE) zwischen den Flanken kodiert (ENCryption) werden. Es gibt zwar noch andere Kodierungen (RC5,...) jedoch wurde diese Kodierung in allen analysierten LIRC Files automatisch gesetzt.
  - o CONST\_LENGTH:
    - Alle Befehle haben die gleiche Länge. (Auswirkung wird bei gap beschrieben)
- Eps, Aeps: Kennzahlen zur Toleranz beim Empfangen von Infrarotsignalen. Eps ist dabei ein relativer Wert (30% Toleranz) und Aeps ein absoluter Wert in microsekunden (100µs).
- One, Zero: gibt die Dauer der verschiedenen Pegel an. Zuerst die Dauer des „High Pegels“ und dann die Dauer des „Low Pegels“ im obigen Beispiel wird also eine Eins durch 315ms an und 709ms aus kodiert. Äquivalent dazu die Null mit 315ms an und 198ms aus.
- Ptrail: Nach den letzten Daten wird ein „High Pegel“ von 319ms gesetzt
- Gap: Der Gap ist die Lücke zwischen zwei Befehlen. Da bei Flags CONST\_LENGTH gesetzt ist, gibt der Wert die Pause in Millisekunden an, die zwischen Ende eines Befehls bis Anfang des nächsten Befehls gewartet werden soll. Ohne CONST\_LENGTH würde der Wert die Pause zwischen jeweils den Anfängen zweier Befehle festlegen.
- toggle\_bit\_mask 0x0Laut (<http://winlirc.sourceforge.net/technicaldetails.html>) können über die Maske Bits, bei jedem erneuten drücken einer Taste gekippt werden.

#### Zusätzliche Möglichkeiten

1. pre\_data, pre\_data\_bits: Über pre\_data können Bits angegeben werden, die vor jedem Befehl gesendet werden sollen. Die Länge wird über pre\_data\_bits angegeben
2. post\_data, post\_data\_bits: Äquivalent zu pre\_data nur nach dem Befehl.

#### Codes:

Zwischen begin codes und end codes werden nun die einzelnen Befehle Angegeben. Im vorherigen Beispiel sind nur drei Befehle aufgezeichnet worden. Der fehlende (TAILLEFT) wurde fehlerhaft aufgezeichnet und konnte nicht Analysiert werden.

### **Optimierung**

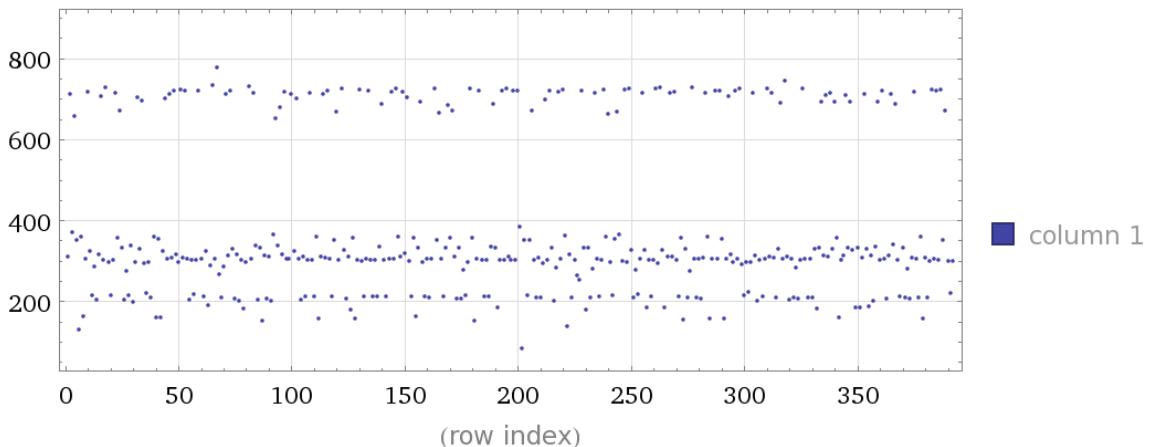
Die aufgenommenen LIRC Files funktionierten leider nur sehr schlecht, da sie Fehlerhaft waren. Durch das „zusammenbauen“ mehrerer Files konnten die Fehler korrigiert werden. Die Fehler kommen dadurch, dass erstens die Fernbedienung keine präzisen Signale sendet, die Abtastrate des Seriellen Eingangs zu langsam ist oder der Prozess auf dem Computer zu spät dran kommt.

Dadurch treten folgende Fehler auf:

- Die Dauer der Pegel wird nur ungenau aufgenommen

- Dadurch können manche Signale nicht analysiert werden oder es werden die falschen Hex Werte erkannt
- Die Pegeldauer (One, Zero) werden falsch übernommen. Da der falsche Wert auch zum Senden benutzt wird werden die Signale dadurch vom Empfänger zum Teil nicht erkannt.

Raw data plot :



Computed by Wolfram|Alpha

In der Abbildung ist die Verteilung der Dauer einzelner Pegel (Raw values vor der Analyse) zu sehen hier fällt schon auf, dass die Werte stark variieren. 4 Werte wurden bereits aus der Graphik entfernt die ca. 100.000ms lang waren da sie sonst die Graphik zu sehr verschoben hätten. Rein logisch waren sie nicht zu erklären und im Internet ließ sich nichts über solch große Werte finden. Sie werden vom Analysetool anscheinend nicht beachtet, da sie anscheinend keinen Einfluss auf die Hex Werte haben.

Da auch beim Senden mit der Irdroid weitere Ungenauigkeiten auftreten, summieren sich alle Fehler und die Befehle werden nur schlecht oder gar nicht vom Fisch erkannt. Deshalb wurde das LIRC File optimiert was im Endeffekt auch Wirkung zeigte:

Zunächst wurden die Kommandos aus allen Files verglichen und die ausgewählt, die in den meisten Files vorgekommen sind. Dass manchmal Befehle nicht erkannt wurden, war dadurch kein Problem, da jeder Befehl in mindestens einer Aufnahme vorhanden war.

Um die Richtigkeit der Hexwerte zu verifizieren, wurde das Bitmuster analysiert:

Da die ersten beiden Bytes für alle Befehle gleich waren (0xC9B1) wurde nur das letzte betrachtet:

|           |      |          |          |          |          |          |          |          |          |
|-----------|------|----------|----------|----------|----------|----------|----------|----------|----------|
| DIVE      | 0x87 | <b>1</b> | 0        | 0        | 0        | <b>0</b> | 1        | 1        | 1        |
| CLIMB     | 0x4B | 0        | <b>1</b> | 0        | 0        | 1        | <b>0</b> | 1        | 1        |
| TAILLEFT  | 0x2D | 0        | 0        | <b>1</b> | 0        | 1        | 1        | <b>0</b> | 1        |
| TAILRIGHT | 0x1E | 0        | 0        | 0        | <b>1</b> | 1        | 1        | 1        | <b>0</b> |

Wie man in der Tabelle sieht, sind die Bits immer entweder bei drei Befehlen 1 und bei den anderen 0 oder genau andersrum. Somit hat jeder Befehl zwei einzigartige „Bit-Merkmale“ und beim Empfangen können Bit-Fehler ausgeglichen werden. Dass die Anordnung zufällig entstanden ist, ist sehr unwahrscheinlich und so, konnte bestätigt werden, dass die Hex Werte stimmen.

Des Weiteren wurde die Pegeldauer von Zero und One von allen Aufnahmen gemittelt um den Fehler auch hier zu reduzieren. Die gemittelten Werte waren:

|        |     |     |
|--------|-----|-----|
| One    | 296 | 720 |
| Zero   | 296 | 221 |
| Ptrail | 296 |     |

Die gemessenen Zeiten weichen bis zu 12% von diesen Durchschnittswerten ab. Zum Testen wurden drei Versionen bereitgestellt: die Original-Zeiten, die Durchschnittswerte und gerundete Werte (auf 300, 720 und 220). Durch Testen mit dem Fisch stellte sich heraus, dass der Fisch auf die gerundeten Werte am besten reagierte.

Die gleiche Analyse wurde genauso für den neuen Fisch durchgeführt, bei dem sich die Analyse als schwerer herausstellte, da die aufgenommenen LIRC-Files stark voneinander abwichen und viele Fehler dabei waren. Das kann entweder daran liegen, dass die Fernbedienung ungenauer schickt oder das Aufnahmesetup ungenauer war (Anderer PC, weitere Entfernung vom Sender zum Empfänger, etc).

### **Änderungen bei der neuen Fernbedienung:**

Es gibt einen Befehl mehr, bei dem der Fisch mit der Schwanzflosse wedelt und somit nach vorne schwimmt. Die Daten sind 11 Bits lang wobei die ersten 2 immer Nuller sind, die restlichen 9 Bits sind ähnlich wie bei der alten Fernbedienung aufgebaut:

|           |       |   |   |   |   |   |   |   |   |   |   |
|-----------|-------|---|---|---|---|---|---|---|---|---|---|
| DIVE      | 0x0FF | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| CLIMB     | 0x17C | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 0 |
| TAILLEFT  | 0x1B5 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 1 |
| TAILRIGHT | 0x1D9 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| SWIM      | 0x1FD | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 |

Auch hier gibt es für 8 Bits jeweils eine einmalige Belegung. Die Ausnahme beim vorletzten Bit, bei dem Dive und Forward beide eine Eins belegen muss daher kommen, dass die Befehle in 9 Bits kodiert werden. Mit 10 Bits wäre es genau aufgegangen, weswegen nicht ganz verständlich ist, warum das eine Bit nicht noch spendiert wurde und solche krummen Zahlen verwendet werden.

Die gemittelten Werte für One und Zero sind bei dem neuen Fisch nicht so einheitlich und intuitiv „rundbar“

|      |     |     |
|------|-----|-----|
| One  | 777 | 314 |
| Zero | 382 | 314 |

Ptrail entfällt bei der Fernbedienung

Da bei der alten Fernbedienung Änderungen der Zeiten um wenige Millisekunden reichten um das Signal zu verbessern versuchte ich auch bei den neuen Werten ein System zu entdecken.

$$\frac{t_{One, on}}{t_{Zero, on}} = \frac{777}{382} = 2,034$$

$$\frac{t_{One, ges}}{t_{Zero, ges}} = \frac{777 + 314}{382 + 314} = \frac{1091}{696} = 1,5675$$

Es kann natürlich Zufall sein, dass das Verhältnis zwischen den „An Zeiten“ ungefähr Zwei ist und das Verhältnis zwischen den Gesamtzeiten ca. 1,5. Ein Versuch war es auf jeden Fall wert, Werte zu finden, die die genauen Verhältnisse bringen und ähnlich zu den Gemittelten sind.

Durch lösen der Gleichungen  $\frac{x}{y} = 2$  und  $\frac{x+z}{y+z} = 1.5$  resultierte die Lösung  $y = z = \frac{x}{2}$ .

Somit müsste man die „An Dauer“ von Zero und die „Aus Dauer“ um jeweils mindestens 20% verändern um das Verhältnis zu erlangen was zu unrealistisch erschien.

Die Lösung des zweiten Versuchs (mit  $\frac{x+z}{y+z} = 1.6$ ) ist  $y = \frac{x}{2}$  und  $z = \frac{x}{3}$

Eine Lösung dafür, bei der jeder Wert um 10% geändert werden würde (weniger war nicht möglich) ist

|      |           |           |
|------|-----------|-----------|
| One  | 840 (+63) | 280 (-34) |
| Zero | 420 (+38) | 280 (-34) |

Die Veränderung wäre immer noch sehr drastisch also müssen die Anforderungen reduziert werden.

Wenn man die zweite Bedingung ( $\frac{x+z}{y+z} = 1.5$  oder  $1.6$ ) vernachlässigt und nur versucht die Erste zu erreichen, ist die Lösung mit sehr kleinen Korrekturen zu erreichen. Wenn danach noch der „Low“ Wert auf 315 aufgerundet wird, stellt sich genau der Faktor zwischen den Gesamtzeiten 1.55. Die beiden geraden Faktoren (ohne viele Nachkommastellen) und die einzelnen Werte die auf 0 und 5 enden scheinen gefühlsmäßig plausibel.

|      |          |          |
|------|----------|----------|
| One  | 770 (-7) | 315 (+1) |
| Zero | 385 (+3) | 315 (+1) |

Die geänderten Zeiten werden auf den Server geladen und sie müssen getestet werden. Das gefundene „System“ ist eventuell ein wenig aufgezwungen doch es wird sich zeigen, ob die Werte besser funktionieren.

### **Schönheitsänderungen:**

Da bei der ersten Fernbedienung die ersten 2 Bytes immer gleich waren (0xC9B1) wurden sie in pre\_data geschrieben, sodass nur noch der veränderliche Teil pro Befehl angegeben werden muss.

Bei manchen LIRC Files war keine CONST\_LENGTH angegeben. Das Flag wurde hinzugefügt und der Gap entsprechend geändert.

## **6 Senden**

*Von Felix Kohlbrenner und Patrick Trojosky*

### **6.1 Irdroid Sender**

Für das Ziel den AirSwimmer mit einem Tablet fernsteuern zu können, muss das entsprechende Endgerät infrarot fähig sein. Da das zu verwendende Tablet der Hochschule über keine integrierte Infrarotschnittstelle verfügt, gilt es zu prüfen, in wie fern sich die Aufgabenstellung umsetzen lässt.

Eine ausführliche Internetrecherche führt zu verschiedenen Android Geräten, welche Infrarot unterstützen (z. B. das HTC One). Die Anwendung zum Steuern des AirSwimmers auf einem anderen Gerät als dem Tablet der Hochschule zu verwenden, ist allerdings nur eine Notlösung für den Fall, dass es nicht möglich ist, das Tablet mit Hilfe eines Moduls nachzurüsten.

Neben den Geräten mit eingebautem Infrarot, tauchte auch immer wieder die Umsetzung der Firma „Irdroid“ (<http://www.irdroid.com/>) auf. Das Produkt welches diese Firma vertreibt, scheint für die Ziele des Projektes genau das Richtige zu sein, sodass weiteren Recherchen hierauf fixiert wurden. Bei dem nachrüstbaren Infrarot Modul handelt es sich um eine Universalfernbedienung, welche man wie einen Kopfhörer in den Audioausgangs des Geräts stecken kann.



**Abbildung 32**  
Irdroid – Adapter

Diese Fernbedienung wurde speziell für Android Geräte entwickelt und ist dazu gedacht Fernseher per Infrarot fernzusteuern. Dazu gibt es verschiedene Applikationen, welche über den Google Play Store installiert werden können. Die offizielle App der Irdroid Hersteller ist Open Source, sodass einer Eigenentwicklung der gewünschten Funktionalitäten nichts im Wege steht. Gemeinschaftlich wird somit beschlossen ein

Irdroid Modul zu erwerben. Dieses kommt nach relativ kurzer Versandzeit bei uns an, leider mit einer schon fast leeren Batterie. Der nächste Schritt besteht daher darin eine neue Batterie zu besorgen, was sich als nicht trivial herausstellt, da kein Elektronikfachmarkt eine entsprechende Batterie vorrätig hat. Schlussendlich wird über das Internet eine zehner Packung bestellt, da diese unwesentlich teurer ist als eine einzelne Batterie.

## 6.2 Analyse der Irdroid App

Die Funktionalität der App von Irdroid kommt der gewünschten Funktion der geplanten App schon relativ nahe, daher bestand der nächste Schritt darin, den Quellcode der App zu analysieren. Mit den Ergebnissen dieser Analyse wurde entschieden, welche Teile davon später in unsere App einfließen können und was noch selbst implementiert werden muss.

Generell lässt sich die Irdroid App in drei Komponenten aufteilen:

1. Java Code der eigentlichen App (im „src“ Ordner)
2. Native C-Funktionen, die das Senden hardwareseitig realisieren und das Lirc-File lesen
3. Lirc – File, siehe Kapitel 6.2

### 6.2.1 Java Code

Die Irdroid App besteht hauptsächlich aus drei Klassen („Iconic.java“, „Irdroid.java“ und „Lirc.java“). Die Anwendung bietet das Feature Lirc-Files für bestimmte Geräte aus dem Internet über eine Datenbank abzurufen. Die Implementierung dieser Funktion findet sich in der „Iconic“ Klasse wieder. Sie ermöglicht es eine Verbindung mit dem Internet aufzubauen und jeweils die aktuellsten Lirc - Files anzuzeigen, sodass der Benutzer diese auswählen und herunterladen kann. Da sich unser Projekt auf die Fernbedienung des zu steuernden Fischs fokussiert, sind diese Funktion und damit auch die Klasse Iconic für weitere Schritte nicht relevant.

Deutlich wichtiger ist die Klasse „Lirc.java“, welche das Handling des Java Native Interfaces (JNI) beinhaltet. In Java ist es grundsätzlich möglich Code aus anderen Programmiersprachen einzubinden und zu verwenden. JNI bildet dabei ein standardisiertes Interface. Bei der Irdroid App werden mit diesem Hilfsmittel C-Dateien verwendet, auf die im nächsten Abschnitt genauer eingegangen wird. Um die Dateien verwenden zu können, ist die Definition einer Schnittstelle zwischen Java und C notwendig. Dafür werden in der Lirc Klasse „native“ Java Methoden deklariert. Somit weiß der Compiler, dass es sich bei diesen Methoden um „fremden“ Code handelt. Die Funktionalität zu diesen „native“ Java Methoden ist in der ins Projekt eingebundenen C-Bibliothek abgelegt. Die Hauptaufgabe der Lirc Klasse ist sowohl das Laden der .so Datei (Linux Bibliothek) über den Befehl „system.LoadLibrary(„irdroid“)“, als auch die Deklaration der vier „native“ Methoden.

### 6.2.2 C-Code

Die C-Dateien, welche die Irdroid App verwendet, machen uns zu Beginn des Projekts relativ große Probleme. Es ist sehr schwierig nachzuvollziehen, wie genau die Aufrufe erfolgen und welche Funktionen ausgeführt werden. Ein Debuggen der Funktionen ist

in der Entwicklungsumgebung Eclipse nicht möglich, da im Programmablauf nur noch die Java „native“ Methoden verwendet werden. Dies bedeutet, dass lediglich ein Methodenprototyp zur Analyse zur Verfügung steht, da die eigentliche Logik in der .so Datei (C-Code) steht. Diese Datei kann nicht genauer betrachten werden, da sie bereits kompiliert ist. Nach weiterer Auseinandersetzung mit dem Programm stellte sich heraus, dass die Methoden für das Handling des Lirc-Files eingesetzt werden. Dies betrifft das Lesen, das Verarbeiten und die Bereitstellung der Daten in einem Puffer. Die „native“ Java Methode „parse(String filename)“ liest das Lirc-File ein und bereitet die verschiedenen Kommandos mit dem zugehörigen Hexadezimalen Code auf. Die Methode „getCommandList()“ liefert dann eine Liste mit allen Kommandos, die im Lirc-File gefunden werden. Die Methode „getIrBuffer(device,command)“ sucht dann nach einem als Parameter angegebenen Kommando, welches zum Beispiel das Erhöhen der Lautstärke ist und liefert den passenden Puffer zurück. Dieser enthält dann die hexadezimale Kodierung des Signals. Welche Schritte dazu genau in C durchgeführt werden, konnte nicht genau herausgefunden werden. Daher wurde die Vorgehensweise der Irdroid App als Richtlinie angenommen.

Für den App Prototypen muss die .so Datei eingebunden werden (.so ist die Dateiendung einer Linux Bibliothek und steht für shared object). Somit kann auf die C-Funktionen zugegriffen werden, ohne deren genaue Logik zu kennen.

### **6.2.3 Die Irdroid Klasse**

Die größte und auch wichtigste der drei Java Klassen ist die Klasse „Irdroid.java“. Sie ist knapp 1000 Zeilen lang und beinhaltet die meisten Funktionen der App. Die Herausforderung besteht darin herauszufinden welche Teile für unsere App von Bedeutung sind. Die Klasse enthält neben den Funktionsaufrufe um das Senden der Signale zu realisieren auch die Methoden und Variablen der Benutzeroberfläche. Dies schließt auch Listener für die Buttons und Menüs mit ein. Generell betrifft mindestens die Hälfte aller Funktionen die grafische Oberfläche. Diese ist nicht von Interesse, da im Rahmen des Projektes eine eigene Oberfläche erstellt wird.

Während der Analyse des Codes stellt sich heraus, dass die Irdroid App Runnables implementiert, um die wichtigsten Kommandos zu senden. Ursprünglich besitzt die App sechs Tasten (Lautstärke plus/minus Sender plus/minus, Power und Mute). Da vier dieser Tasten im alltäglichen Gebrauch länger gedrückt werden können, zum Beispiel beim Senderwechsel, werden diese Sendevorgänge als Runnables implementiert.

Der Aufruf der Logik die das Senden realisiert erfolgt in der Methode „sendSignal(device,comand)“. Diese Erkenntnis ist wichtig, da sie zeigt, dass es auch möglich ist Kommandos ohne großen Aufwand schnell hintereinander zu versenden. Die „sendSignal()“ Methode stellt dabei einen der wichtigsten Teile der App dar und ist somit für die AirSwimmer App essentiell. Daher wird diese Methode auch mit leichten Anpassungen später in unserer App verwendet.

Zudem werden die Abschnitte im Code identifiziert, die das Handling des Lirc - Files ermöglichen. Für das Projekt ist es ausreichend ein einziges File zu benutzen, weshalb nur ein geringer Teil der vorhandenen Implementierung verwendet werden muss.

Zusammenfassend kann man sagen, dass die Analyse der Irdroid App sehr erfolgreich ist. Es gelingt die essentiellen Teile ausfindig zu machen und weiterzuverwenden.

Durch den bereits vorhandenen Code zum Senden der Kommandos und Einlesen und verwenden des Lirc - Files werden die nächsten Implementierungsschritte erleichtert.

### **Grundsätzliche Funktionsweise des Sendens:**

Das Lirc-File wird von der App eingelesen und alle gefundenen Kommandos in einer Liste gespeichert. Wenn eine Taste (z.B. Mute) gedrückt wird, wird die „sendSignal“ Methode aufgerufen. In dieser wird mithilfe der C-Library das passende hexadezimal kodierte Kommando zum gewünschten Befehl gesucht und in einen Puffer geschrieben. Danach wird ein Android Audiotrack erstellt, der den Inhalt des Puffers übergeben bekommt. Dieser Audiotrack muss korrekt konfiguriert werden (siehe Abbildung 2). Jetzt kann dieser Audiotrack den Puffer abspielen und so das Signal über den Aux - Ausgang in den Irdroid Adapter leiten. Dieser verstärkt das Signal, da dieses am Aux - Ausgang sehr schwach ist. Schlussendlich sendet der Adapter das Signal über die beiden Dioden an den Empfänger.

```
ir = new AudioTrack(AudioManager.STREAM_MUSIC, 48000,
    AudioFormat.CHANNEL_CONFIGURATION_STEREO,
    AudioFormat.ENCODING_PCM_8BIT, bufferSize, AudioTrack.MODE_STATIC);
```

**Abbildung 33**  
Audiotrack

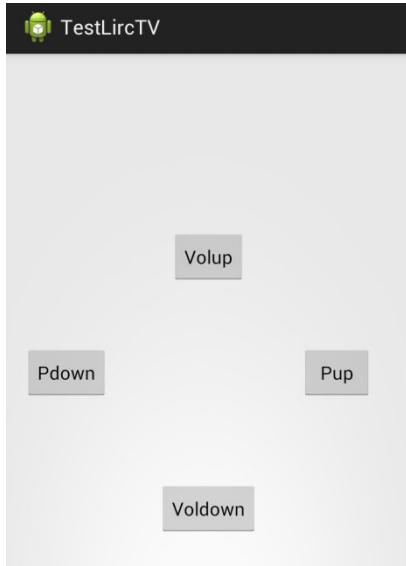
## **6.3 Eigene App**

Nach der Analyse der Irdroid-App und mit dem Adapter, soll diese zunächst an einem Fernseher getestet werden, um zu sehen ob er sich so verhält wie erwartet. Dazu wird ein Samsung Fernseher und ein Samsung Galaxy S3 verwendet. Das benötigte Lirc-File für den Fernseher kann im Internet gefunden werden (unter <http://lirc.sourceforge.net/remotes/> sind viele Lirc-Files für diverse Fernbedienungen vorhanden). Leider zeigt der Fernseher keine Reaktion auf die Tasten die in der App betätigt werden. Die Ursache dafür ist zunächst nicht offensichtlich. Es ist im ersten Moment nicht festzustellen, ob der Fehler vom Sender, vom Empfänger oder vom Lirc-File ausgeht. Der Versuch mit einem anderen TV Gerät ist auch erfolglos. Als nächstes wird die Batterie des Irdroid-Moduls ausgetauscht, erneut ohne Erfolg. Da zu diesem Zeitpunkt noch keine Möglichkeit besteht, die Fernbedienung selbst aufzuzeichnen, um unser eigenes Lirc-File zu erstellen, muss auf dessen Richtigkeit vertraut werden. Somit kann es nur noch am Telefon liegen. Daher wurde die App auf einem Samsung Galaxy S2 getestet. Doch auch hier ist keine Reaktion zu sehen. Es konnte aber herausgefunden werden, dass der Sender zumindest Signale sendet. Diese kann mittels einer auf die LEDs gerichteten Kamera erkannt werden.

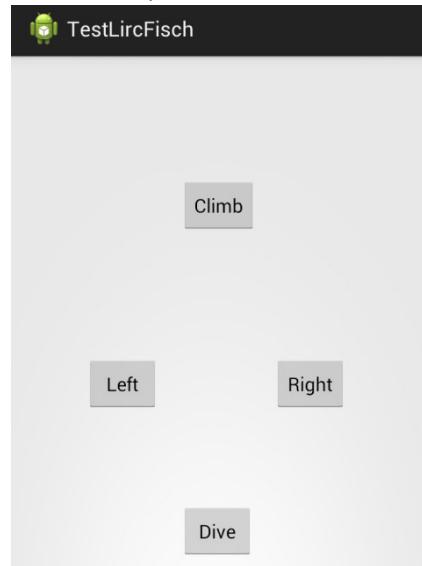
In der mitgelieferten Beschreibung, welche zunächst nicht vorliegt, kann die Ursache für das Problem gefunden werden. Die Lautstärke des Senders muss passend eingestellt werden. Deshalb wird im nächsten Versuch mit den verschiedenen Lautstärken (Android Geräte besitzen 15 Stufen) des Galaxy S2 experimentiert, bis das passende Level gefunden ist und der Fernseher auf die Signale reagiert. Die korrekte Funktion der App ist damit sichergestellt und die Anwendung erfüllt mittels passendem Lirc-File unsere Zwecke.

Der nächste Schritt besteht nun darin, die für die Anwendung benötigten Teile der Irdroid App zu extrahieren und eine Test App zu schreiben. Da es noch kein Lirc-File für

den AirSwimmer gibt, wird beschlossen eine App für den Fernseher zu schreiben, da dieser bereits auf erste Sendevereuche reagierte. Dafür wird die Irdroid-Anwendung analysiert und der Code, der unserer Ansicht nach nicht benötigt wird, gelöscht. Dazu gehören verschiedene Funktionen, wie das Downloaden eines Lirc-Files oder eine Infoseite. Für das Testen des Prototyps entsteht eine einfache Oberfläche mit vier Buttons (siehe Abbildung 36, Volup: Erhöhen der Lautstärke, Voldown: Verringern der Lautstärke, Pdown: Sender erniedrigen, Pup: Sender erhöhen).



**Abbildung 34**  
TVapp Prototyp



**Abbildung 35**  
AirSwimmer Prototyp

Der erste Prototyp ist auf Anhieb erfolgreich. Der Fernseher lässt sich jetzt mit unserer Anwendung steuern. In der Zwischenzeit gelingt es unseren Kollegen das Lirc-File der AirSwimmer Fernbedienung aufzuzeichnen. Um zu testen, ob dieses Lirc-File funktioniert, muss im Anschluss ein Prototyp geschrieben werden, mit dem der Fisch ferngesteuert werden kann. Dies ist aufgrund der bereits funktionierenden Fernseher Anwendung relativ schnell erledigt. Die Oberfläche der AirSwimmer Anwendung ist in Abbildung 4 zu sehen.

Es müssen im Prinzip nur die Buttons umbenannt und die Listener angepasst werden. Leider scheint es doch nicht so einfach zu sein, denn mit diesem Prototyp lässt sich der AirSwimmer nicht fernsteuern. Das Lirc-File kann als Fehlerquelle ausgeschlossen werden, da dieses in die Irdroid-App eingebunden werden kann und hier die gesendeten Signale beantwortet werden, wenn auch nur sehr sporadisch.

## 6.4 Weiterentwicklung der App

Da nicht klar ist wo das Problem liegt, ist das Schreiben einer zweiten Testapplikation notwendig. Im ersten Prototyp werden die Signale pro Tastendruck nur einmal über die „sendCommand()“ Methode gesendet. Diese Methode ist die aus der Irdroid App bekannten „sendSignal()“ Funktion mit leicht veränderter Syntax. So hat unsere Variante nur noch einen Parameter, das zu schickende Kommando. Im zweiten Versuch werden vier Runnables implementiert, jeweils eine pro Bewegungsrichtung des Fischs. Durch die Runnables, die auch in der Irdroid App für spezielle Tasten

verwendet werden, ist es nun möglich eine Taste länger zu drücken und damit das Signal mehrfach hintereinander zu senden. Für Vergleichszwecke wurde parallel die TV App weiterentwickelt. Erneut funktioniert diese Variante ohne Probleme, auch bei schrägen Einfallswinkeln des Signals.

Da der AirSwimmer mittlerweile zerlegt ist, besteht die Möglichkeit das Empfänger Modul mitzunehmen und intensiver zu testen. Nach vielen kleinen Änderungen, die hauptsächlich im Try-and-Error Verfahren ermittelt werden, gelingt es den Fisch in Bewegung zu setzen. Während des Testens wird versucht die Signalstärke softwareseitig zu verstärken, da vermutet wird, dass das Signal zu schwach ist. Selbst wenn es uns gelingt das Empfängermodul zu steuern, ist dies nur mit genauem Zielen auf die LED möglich. Sobald der Sender nicht exakt auf den Empfänger zeigt, ist keine Bewegung mehr hervorzurufen. Es werden verschiedene Sample Rates im verwendeten Audiotrack getestet, jedoch ohne eine Verbesserung festzustellen. Da bereits herausgefunden wurde, dass die Lautstärke des Geräts angepasst werden muss, wird entschieden, die „sendCommand()“ Methode zu erweitern und die Lautstärke fest bei jedem Sendevorgang zu setzen. Man muss dabei nur darauf achten, die richtige Lautstärke zu regeln, da Android unterschiedliche Lautstärken besitzt. Es gibt die System, Anruf und Medien Lautstärke. In diesem konkreten Fall die Stream\_Music Lautstärke, also die Lautstärke der Medien. Dazu wird die Codezeile:  
audio.setStreamVolume(AudioManager.STREAM\_MUSIC, 5, 0) verwendet. Durch diese Änderung kann eine konstant funktionierende App erstellt werden. Bei weiteren Tests, diesmal mit dem Tablet, wird festgestellt, dass nicht bei allen Geräten die gleiche Lautstärke funktioniert. Weiteres Testen führt dazu, dass für das Galaxy Tab 1 und 2 eine Lautstärke von sechs zu wählen ist (beim Galaxy S2 ist es fünf). Aus dieser Beobachtung entsteht die Idee eine initiale Lautstärke Kalibrierung in die Oberfläche zu integrieren (siehe 7.7).

Weiterführende Tests mit dem Galaxy Tab 1 führen zu der Erkenntnis, dass eine Lösung für die Integration des Lirc-Files in die App gefunden werden muss. Bisher ist es so, dass das Lirc-File in der obersten Ebene des internen Speichers abgelegt wird. Der Pfad des Lirc-Files, wurde zum Testen und der Einfachheit halber fix codiert. Nun tritt allerdings das Problem auf, dass der Pfad auf verschiedenen Geräten anders lautet. Es wird erkannt, dass der „assets“ Ordner innerhalb eines Android Projekts dazu da ist, Dateien, die die Anwendung benötigt, hier abzulegen. DasLirc-File wird daher in diesen Ordner kopiert und kann über „caller.getAssets().open("lirc.txt");“ darauf zugreifen. Allerdings liefert dieser Methodenaufruf einen Inputstream zurück. Das heißt, das File muss erst eingelesen und der resultierenden Inputstream dann in ein neues File geschrieben werden. Dazu wird ein Ordner mit dem Namen AirSwimmer im Dateisystem des Geräts erstellt. In diesen Ordner wird eine Datei „Lirc.txt“ abgespeichert, welche über einen BufferedReader erstellt wird. Dabei ist darauf zu achten, dass das Format des Lirc-Files nicht geändert wird, es dürfen also keine zusätzlichen Zeilenumbrüche oder ähnliches hinzugefügt werden.  
Nach hinzufügen dieser Funktion ist das Testen der Anwendung auf verschiedene Geräten um einiges einfacher, da nicht jedes Mal das aktuellste Lirc-File auf das Gerät kopieren werden muss. Ab jetzt genügt es, die neuste Version der Anwendung zu installieren. Allerdings muss von nun an für eine neue Version des Lirc-Files, das alte im assets Ordner ersetzt werden. (Lirc-File Pfadanpassung: Felix Kohlbrenner)

# 7 Oberfläche

## 7.1 Konzept

*Von Anja Hafner, Melanie Knappe, Belgüzar Kocak*

### **Zielsetzung**

Ziel war es mit der Oberflächengruppe Anja Hafner, Melanie Knappe & Belgüzar Kocak Konzepte zur Steuerung des Air Swimmers zu entwickeln.

### **Vorüberlegung und Analyse**

Die Ansteuerung des Air Swimmers erfolgt durch eine Infrarot Verbindung. Diese enthält folgende Funktionen:

1. Ansteuerung der hinteren Flosse (nach links oder nach rechts)
2. Ansteuerung des Gewichtes am AirSwimmer (Flug nach oben oder nach unten)

Auf diese Funktionen soll die Oberflächensteuerung aufbauen. Somit wurden drei verschiedene Ansteuerungsvarianten zur Realisierung der Oberfläche erstellt, die wiederum die Steuervarianten Single und Permanent enthalten.

Bei der Ansteuerungsart Button stellt die Bewegungsart Single die Funktion der Fernbedienung nach. Die Ansteuerungsart Permanent enthält einen zusätzlichen Button Start. Durch betätigen des Start Buttons bewegt sich der AirSwimmer aufgrund der permanenten Ansteuerung der hinteren Flusse vorwärts. Die Buttons Links & Rechts ermöglichen eine Links-/ Rechtskurve.

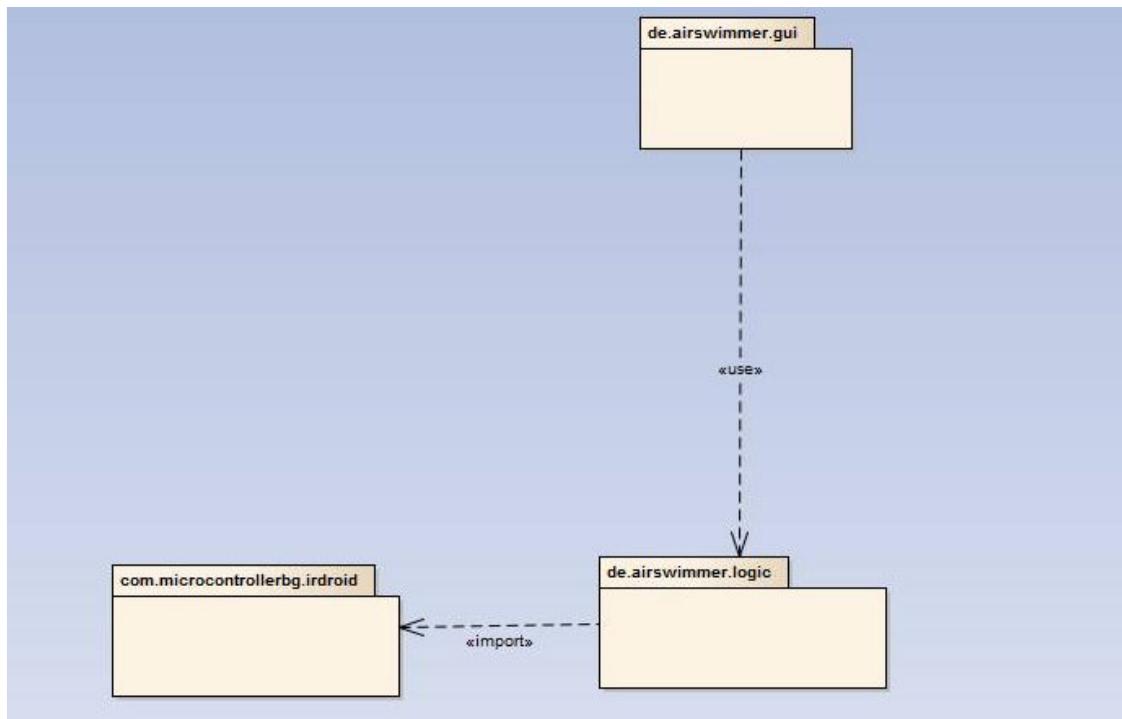
Beim Wischen stellt die Single Bewegungsart folgendes dar: durch eine Wischbewegung nach Links bewegt sich die Flosse einmal nach links. Durch die Wischbewegung nach rechts bewegt sich die Flosse einmal nach rechts. Bei der Permanentfunktion fliegt der AirSwimmer durch eine Berührung der Oberfläche nach vorne. Durch das Wischen nach rechts/links fliegt der AirSwimmer die entsprechende Kurve.

Bei der Nutzung der Ansteuerungsart Kippen erscheint beim Start ein Start Button. Durch Drücken auf diesen startet der Fisch. Bei der Bewegungsart Single bewegt sich die Hinterflosse einmal nach links oder rechts, je nach Kipprichtung. Durch das Kippen nach oben/unten wird das Gewicht am AirSwimmer angesteuert. Wird der Bildschirm berührt, reagiert der AirSwimmer nicht mehr und der Startbildschirm erscheint. Bei der Bewegungsart Permanent erscheint auch zunächst der Start Button, nach Betätigung beginnt der Air Swimmer geradeaus zu schwimmen. Durch das Kippen nach links/rechts wird eine Links-/Rechtskurve geflogen. Auch hier erzeugt die Berührung an den Bildschirm ein Stoppen und der Startbildschirm erscheint.

Die Realisierung der verschiedenen Konzepte wird in den nächsten Kapiteln eindeutig erklärt.

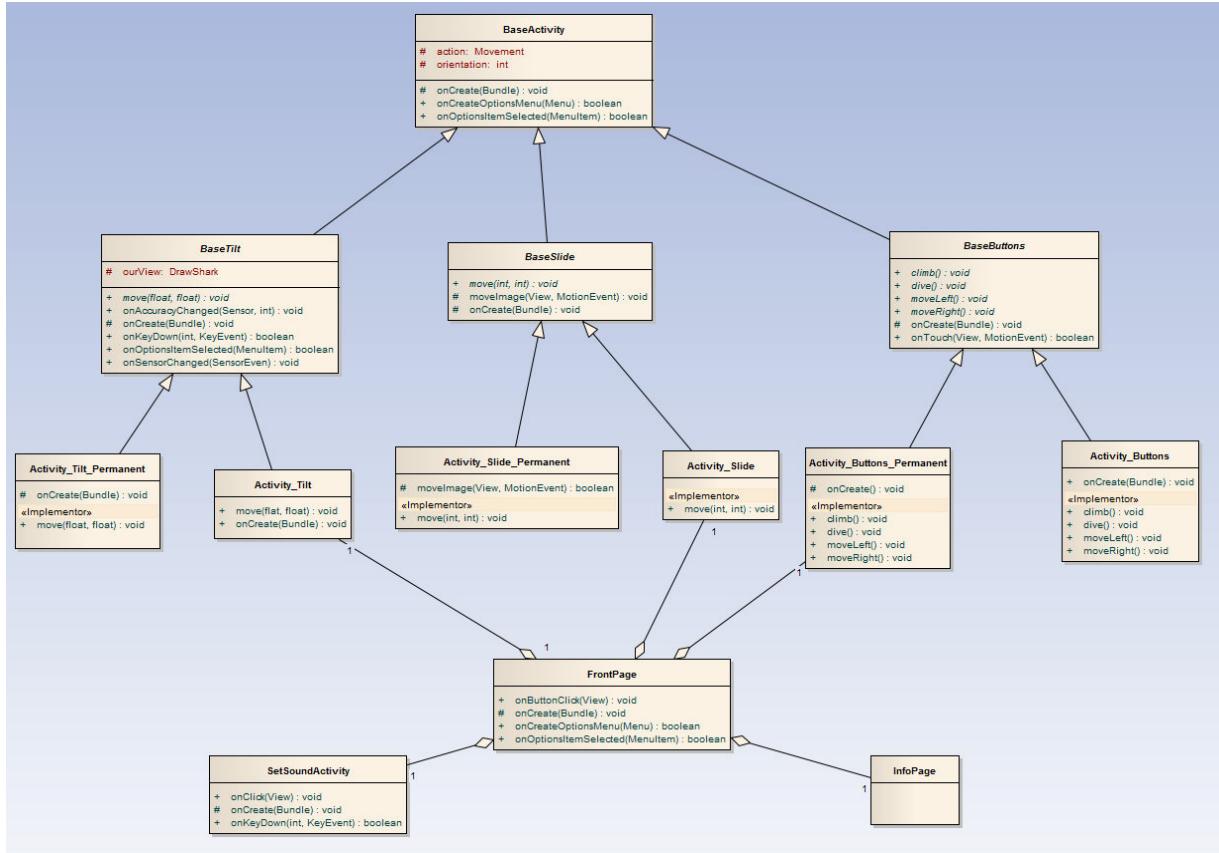
## 7.2 Klassenstruktur

Von Sabine Kressierer



**Abbildung 36**  
Package-Diagramm AirSwimmer

Die gesamte Applikation ist in drei Pakete unterteilt (siehe Abbildung 36). Das Paket **de.airswimmer.logic** enthält dabei die Implementierung zum Senden der vier verschiedenen Infrarotsignale. Dazu wird das Paket **com.microcontrollerbg.irdroid** benötigt. Dieses enthält eine Klasse **Lirc.class**, die als Schnittstelle zu nativem Code dient, der das eigentliche Senden übernimmt. Dieser Programmteil wird in den Kapiteln 6 (Senden) und 8 (Verbinden von Logik und Oberfläche)<sup>8</sup> näher beschrieben. Die Oberfläche befindet sich im Paket **de.airswimmer.gui** und enthält die in folgender Abbildung dargestellten Klassen.



**Abbildung 37**

Klassendiagramm der Oberfläche

Dieses Klassendiagramm zeigt nur die für die Hierarchie wichtigen Attribute und Methoden, ohne die interne Implementierung zu berücksichtigen. Auf die konkrete Implementierung wird in den nachfolgenden Kapiteln eingegangen. Eine vergrößerte Darstellung des Diagramms befindet sich im Anhang.

Als zentrales Element wird bei Starten der App durch die Klasse `FrontPage.class` eine Startseite angezeigt. Von hier aus lässt sich eine Oberfläche zum Kalibrieren der Lautstärke sowie eine Infoseite öffnen. Die Infoseite „`InfoPage`“ setzt sich eigentlich aus drei Klassen zusammen, deren genaue Implementierung im Kapitel 7.9 (Infoseite) erläutert wird. Von der Startseite aus gelangt man zudem auf die verschiedenen Oberflächen zum Steuern des AirSwimmers.

Diese Steuerungsoberflächen leiten alle von einer gemeinsamen Basisklasse „`BaseActivity`“ ab, die wiederum von „`Activity`“ abgeleitet ist. In der Klasse „`BaseActivity`“ sind alle Gemeinsamkeiten der Oberflächen zusammengefasst. Dazu gehört das Erzeugen und Behandeln des gemeinsamen Menüs, sowie die initialisierenden Tätigkeiten, die zum Senden von Befehlen nötig sind. Das Attribut `action` ist ein Objekt der Klasse `Movement`, die sich im Package „`de.airswimmer.logic`“ befindet und dient als Schnittstelle zum Senden der Befehle. Zudem wird hier für jede Activity gespeichert, ob die App im Landscape- oder im Portrait-Modus laufen soll.

Die von „ `BaseActivity`“ abgeleiteten Klassen „ `BaseTilt`“, „ `BaseSwipe`“ und „ `BaseButtons`“ enthalten die eigentliche Implementierung der Oberfläche. Diese Klassen werden später in den entsprechenden Kapiteln näher beschrieben.

Da es, wie im letzten Kapitel beschrieben, zu jeder Steuerungsoberfläche (Tasten, Kippen und Wischen) zwei Steuerungsmodi (single, permanent) geben soll, leiten von jeder Oberfläche zwei entsprechende Activitys ab, die die entsprechende Bewegung implementieren. Dabei entsprechen die Klassen `Activity_tilt`, `Activity_buttons` und `Activity_wipe` der einfachen Bewegung. Diese sind direkt von der Startseite aus erreichbar. Die Klassen mit dem Zusatz „permanent“ stellen die Steuerungen mit permanenter Bewegung dar und sind nur über das Menü der einfachen Steuerung erreichbar.

## 7.3 Startseite

*Von Caroline Pilot*

### Zielsetzung

Jede App benötigt eine Startseite, die es dem Benutzer ermöglicht, eine Auswahl zu treffen, zwischen verschiedenen Seiten und Funktionen zu wechseln und somit den Benutzer empfängt und leitet.

Da die Startseite einen Knotenpunkt für alle Teilanwendungen (Tasten, Kippen, Wischen) darstellt, müssen diese hiermit verknüpft werden.

### Vorüberlegung

#### Aussehen

Da die Startseite in erster Linie die Navigation zu den verschiedenen Benutzermodi übernehmen soll, werden drei Buttons gebraucht.

Außerdem soll die erste Seite der App natürlich optisch ansprechend wirken, deshalb wäre eine Eröffnungsanimation oder eine einladende Graphik des AirSwimmers von Vorteil.

#### Programmcode

Mit Hilfe der Java-Klasse „`Activity`“ können die Buttons mit Aktionen belegt werden, die zu den einzelnen Teilfunktionen führen.

Der Programmcode soll in englischer Sprache erstellt werden.

Außerdem soll für ein Tablet der Größe 10,1 Zoll mit einer Auflösung von 1280 x 800 Pixeln programmiert werden.

### Vorgehensweise

#### Erstellen der Startseite

##### a) Konfiguration

Um die Einbindung der Teilfunktionen einfacher zu gestalten, wird ein neues Android Application Project aufgesetzt („AirSwimmer“) und mit folgenden Einstellungen konfiguriert:

Minimum Required SDK → API 7: Android 2.1 (Eclair)

Target SDK → API 7: Android 2.1 (Eclair)

Compile with → API 7: Android 2.1 (Eclair)

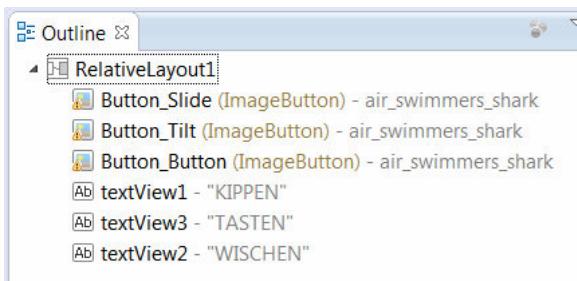
Um Probleme bei der Abwärtskompatibilität zu vermeiden, werden niedrige SDK-Versionen verwendet, damit die App auch auf Endgeräten mit älteren Android-Versionen lauffähig ist.

Die Launcher-Konfiguration wird über ein „Image“ der Form „Circle“ mit der erstellten Graphik gespeist.

Der „Activity-Name“ ist passend dazu „FrontPage“ sowie der zugehörige „Layout Name“ „front\_page“ lautet.

### **b) Layout Erstellung**

Im Unterordner „Layout“ kann nun per Drag&Drop das Erscheinungsbild der Startseite, wie in der Vorüberlegung angesprochen, erstellt werden. Diese setzt sich aus folgenden Komponenten zusammen:



**Abbildung 38**

Komponenten des Layouts der Startseite

Die Buttons werden je nach Funktion mit einem String beschriftet.

Nach Erstellen der Buttons befindet sich für jeden Button ein Codeblock in der xml-Datei. Dieser wird jeweils um folgende Codezeile erweitert, damit der Benutzer diesen Button tatsächlich anklicken kann.

`android:onClick="onButtonClick"`

Außerdem wird ein passender Hintergrund gewählt, der das „AirSwimmer-Thema“ aufgreift

und die Buttons werden in ihrer Größe und Ausrichtung auf die Maßen eines Tablets ausgerichtet.

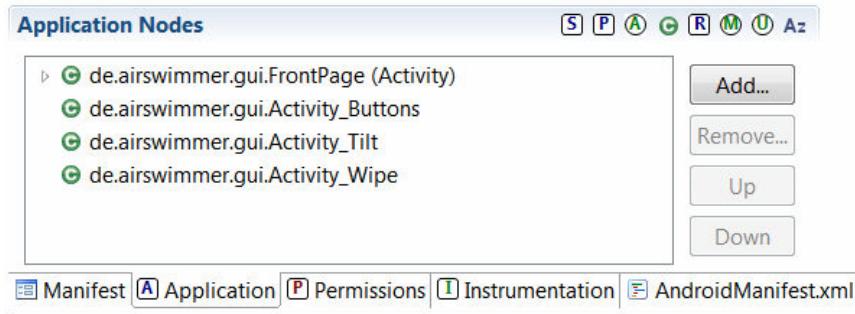
## **Einbinden der Teilfunktionen**

### **a) Layout und Source-Code einbinden**

Durch Erzeugen neuer xml-Dateien („activity\_layout\_buttons“, „activity\_layout\_wipe“ und „activity\_layout\_tilt“) im Unterordner „Layout“ und Einfügen des bereits vorhandenen Codes für die Modi, sind die Layouts der Unterfunktionen nun Bestandteil des Hauptprogramms.

Dasselbe Vorgehen findet auch für den Source-Code im Unterordner „src“ statt, mit neuen Java-Dateien („Activity\_Buttons“, „Activity\_Wipe“ und „Activity\_Tilt“).

Diese Funktionen müssen nun im AndroidManifest unter „Application“ als Knoten hinzugefügt werden:



**Abbildung 39**

Unterpunkt „Application“ im AndroidManifest

### b) Funktion programmieren

Da durch den Buttonklick zu der jeweiligen Funktion gewechselt werden soll, wird der Source-Code der Startseite um switch-case Anweisungen erweitert:

```
// What will happen when you click on a button
public void onButtonClick(View view){
    switch(view.getId()){
        case R.id.Button.Buttons:
            startActivity(new Intent(this, ActivityButtons.class));
            break;
        case R.id.Button_Tilt:
            startActivity(new Intent(this, ActivityTilt.class));
            break;
        case R.id.Button_Wipe:
            startActivity(new Intent(this, ActivityWipe.class));
        default:
    }
}
```

Insgesamt muss beachtet werden, dass alle Namenskonflikte aufgelöst werden, alle zusätzlich genutzten Pakete ebenfalls implementiert werden und alle eingefügten Graphiken in das neue Projekt eingepflegt werden.

### Ergebnis

Der Benutzer gelangt mit Hilfe dieser App nun über die Startseite hin zu den Teiltfunktionen (Tasten, Wischen, Kippen) und kann somit den AirSwimmer steuern.

### Abfrage „IrDroid angeschlossen.“

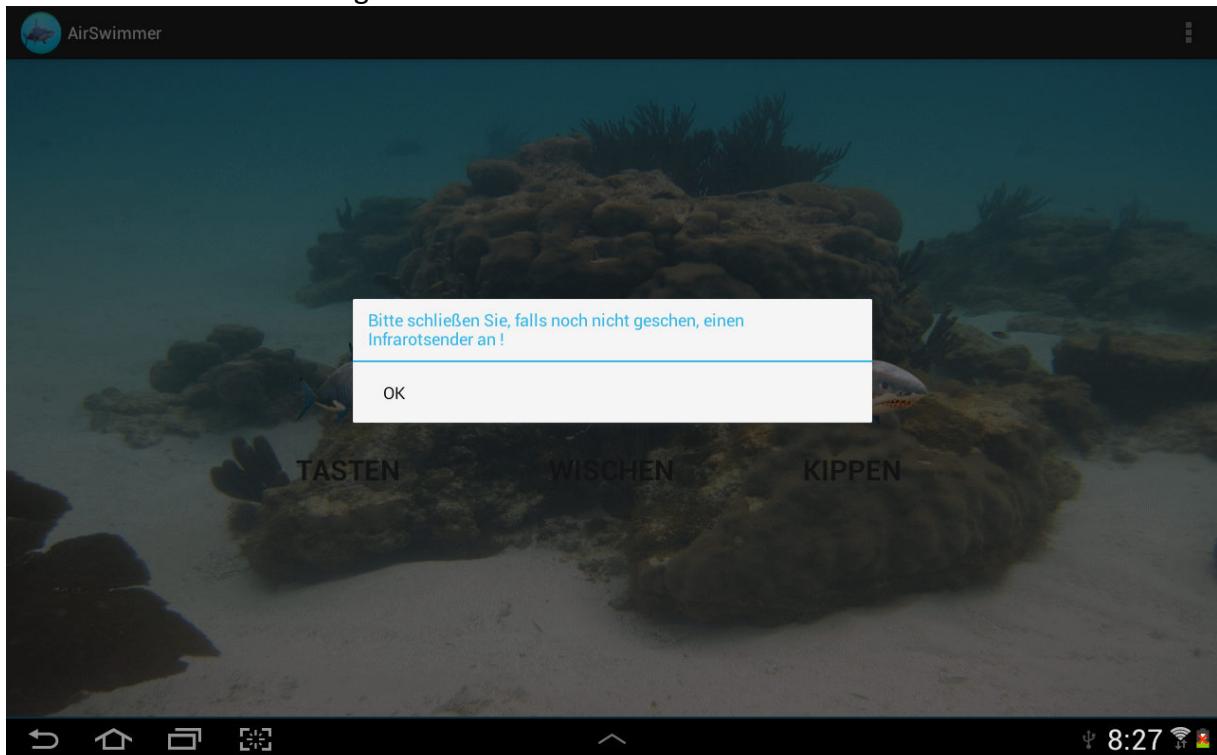
Von Marco Bengl

### Zielsetzung

Ziel der Abfrage ist es den Anwender daran zu erinnern den IrDroid-Sender anzuschließen falls noch nicht geschehen. Ohne den IrDroid-Sender kann der AirSwimmer nicht gesteuert werden.

## Umsetzung

Bei jedem Öffnen der App wird ein Dialog geöffnet, in dem darauf hingewiesen wird, dass der IrDroid-Sender angeschlossen werden soll.



**Abbildung 40**

Startseite mit Erinnerung an IRdroid

Dies geschieht mit Hilfe eines AlertDialog, der in der onCreate()-Methode der FrontPage-Activity erzeugt und angezeigt wird. Mit einem Klick auf den OK Button verschwindet der Hinweis wieder.

```
protected void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    setContentView(R.layout.front_page);  
    preferences = getSharedPreferences("AirSwimmerPrefs",  
        Context.MODE_WORLD_READABLE); //get preferences in file AirSwimmerPrefs  
    int layout = preferences.getInt("layout", -1); //get value for key "layout" or -1 if "layout" does not exist  
  
    //reference for infrared transmitter  
    final CharSequence[] answer = { "OK" };  
    AlertDialog.Builder builder2 = new AlertDialog.Builder(this)  
        .setTitle("Bitte schließen Sie, falls noch nicht geschen, einen Infrarotsender an !").setCancelable(false)  
        .setItems(answer, new DialogInterface.OnClickListener()  
  
            @Override  
            public void onClick(DialogInterface dialog, int item) {  
                String choice = (String) answer[item];  
                if (choice == answer[0]) {  
  
                } else if (choice == answer[1]) {  
                }  
            }  
        );  
}
```

Diese Abfrage enthält keine Funktionalität die abfragt ob der Irdroid tatsächlich eingesteckt ist. Grund dafür ist, dass man nicht ermitteln kann was genau am AUX-Ausgang angeschlossen ist.

## 7.4 Steuerung mittels Buttons

### Ziel:

Ziel ist es, eine Oberfläche mit Buttons zu erstellen. Diese soll vier Buttons zur Steuerung enthalten. Somit wird die Steuerungsarten rechts, links, oben und unten realisiert. Auch soll diese Oberfläche die Fernbedienung des AirSwimmers ersetzen. Zu Beginn werden zwei Ausführungen erstellt: Einerseits mit Buttons mit Beschriftung(Variante A) und andererseits mit Buttons mit Bildern (sogenannten ImageButtons) (Variante B). Als endgültige Version der Buttonssteuerung wird Variante B gewählt.

### 7.4.1 Variante A (erste Testversion)

von Anja Hafner

#### Vorgehensweise:

Die Oberfläche des Projektes „Android Fernsteuerung des AirSwimmers“ wird in der Entwicklungsumgebung „eclipse“ mit Add-Ons für die Android-Programmierung entwickelt.

Zu Beginn wird ein neues „Android Application Project“ in eclipse angelegt (zu finden unter File → New → Other... → Android → Android Application Project).

Die „main\_activity“, die nach der Erstellung des Projekts erscheint, zeigt eine Darstellung eines Handybildschirms mit dem Projektnamen als Überschrift und „Hello World“ als Text im Bildschirm an.

Um nun den ersten Entwurf mit vier einfachen Knöpfen mit Beschriftung zu erzeugen, wird aus der Palette, die sich neben der Darstellung der zukünftigen App befindet, im Abschnitt „Form Widgets“ der Button ausgewählt.

Die Beschriftungen der Buttons werden zunächst in „strings.xml“ definiert (res → values → strings.xml). Für die Initialisierung wird der Button „Add...“ gedrückt. Im erscheinenden Fenster wählt man den Typ „String“ aus und gibt als Namen den Namen des Strings ein, mit dem er angesprochen werden soll. In Value wird die zukünftige Beschriftung (zum Beispiel: „up“) definiert.

Anschließend wird der Button viermal mit der Maus in den Handybildschirm gezogen. Mit dem Befehl „android:text="@string/\*“ wird der String dem Button zugeordnet, wobei der Stern für den Namen des definierten Strings steht.

Für eine einfachere Benutzung der Knöpfe während dem Programmieren, werden die Namen der vier Buttons in der „main\_activity.xml“ in „button\_up“, „button\_down“, „button\_left“ und „button\_right“ geändert.

Um die Abstände der Knöpfe zueinander anzupassen, markiert man den Button, den man verschieben möchte. Danach wird in der Symbolleiste des „activity\_main.xml“ Graphical Layout-Fensters der Button, der einen gestrichelten Rahmen zeigt („Change Margins...“), angeklickt. In dem erscheinenden Fenster trägt man die gewünschte Position ein. Dabei ist zu beachten, dass die Abstände mit der Endung „dp“ (Density Independend Pixel) angegeben werden müssen, damit die Oberfläche richtig angezeigt wird.

Nun kann die App getestet werden. Dazu wird sie zunächst gespeichert. Anschließend folgt ein rechts-Klick auf den Projektordner und unter „Run As“ wird die Option „Android Application“ ausgewählt.

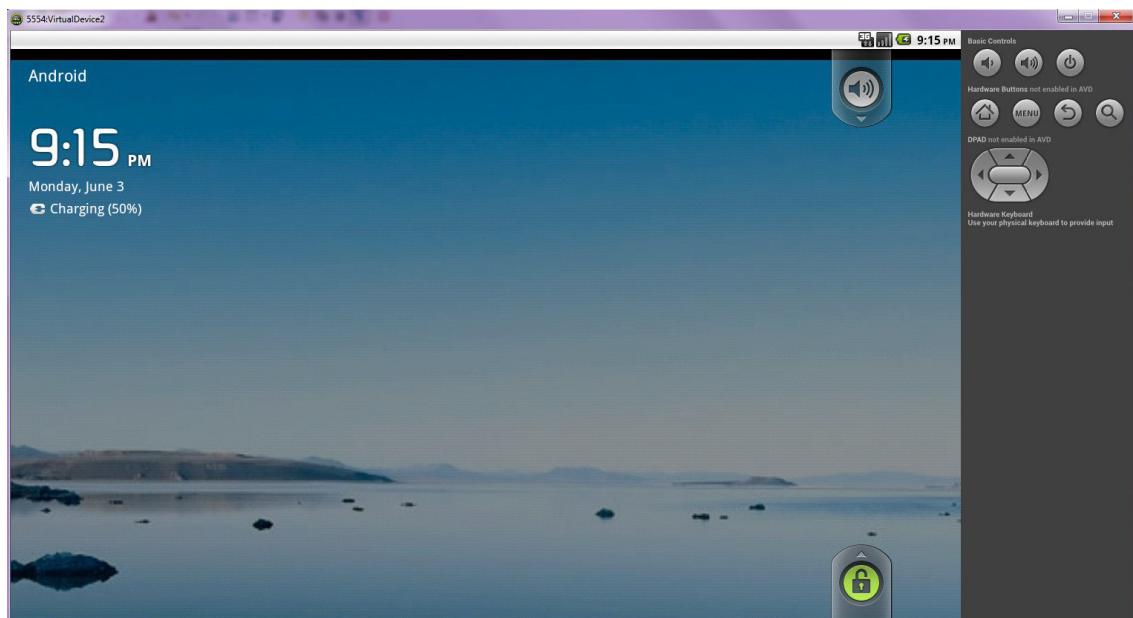
Besitzt man ein Android-fähiges Handy oder Tablett, wird dieses angeschlossen und von eclipse erkannt. Sollte es nicht erkannt werden, müssen zur Erkennung zusätzliche Treiber für das Handy / Tablett installiert werden. Die .apk-Datei der App wird danach auf das Gerät installiert und es kann direkt auf dem Handy / Tablett ausgeführt werden. Eine .apk-Datei ist eine Art Ordner, die alle erforderliche Daten für die Ausführung der App enthält.

Soll die App auf dem Computer ausgeführt werden, ist ein „Virtual Device“ (ein virtuelles Handy, auf das die App gespielt wird) nötig.

Dieses wird im „Android Virtual Device Manager“ erzeugt. Dorthin gelangt man über „Window → Android Virtual Device Manager“ oder in der Symbolleiste über das Handysymbol. Per Mausklick öffnet sich ein Fenster. In diesem Fenster drückt man den Button „New...“ und ein neues Virtual Device kann erstellt werden.

Da die App für Tablets gedacht ist, wird als darzustellendes Gerät „10.1“ WXGA (Tablet) (1280 x 800: mpdi) ausgewählt. Als Target (die minimale Android-Version, auf der die App laufen wird) wird die API „Android 2.2 – API Level 8“ festgelegt.

Nachdem das „Android Virtual Device“ gestartet ist, kann die App, die „AirSwimmer“ genannt wird, gestartet und die Steuerung mit Buttons getestet werden.



**Abbildung 41**

Android Virtual Device

## 7.4.2 Variante B (ImageButtons)

Von Belgüzar Kocak

### Vorgehen

Zuerst stellt uns eclipse beim Erstellen von Projekten ein Standard Layout zur Verfügung.

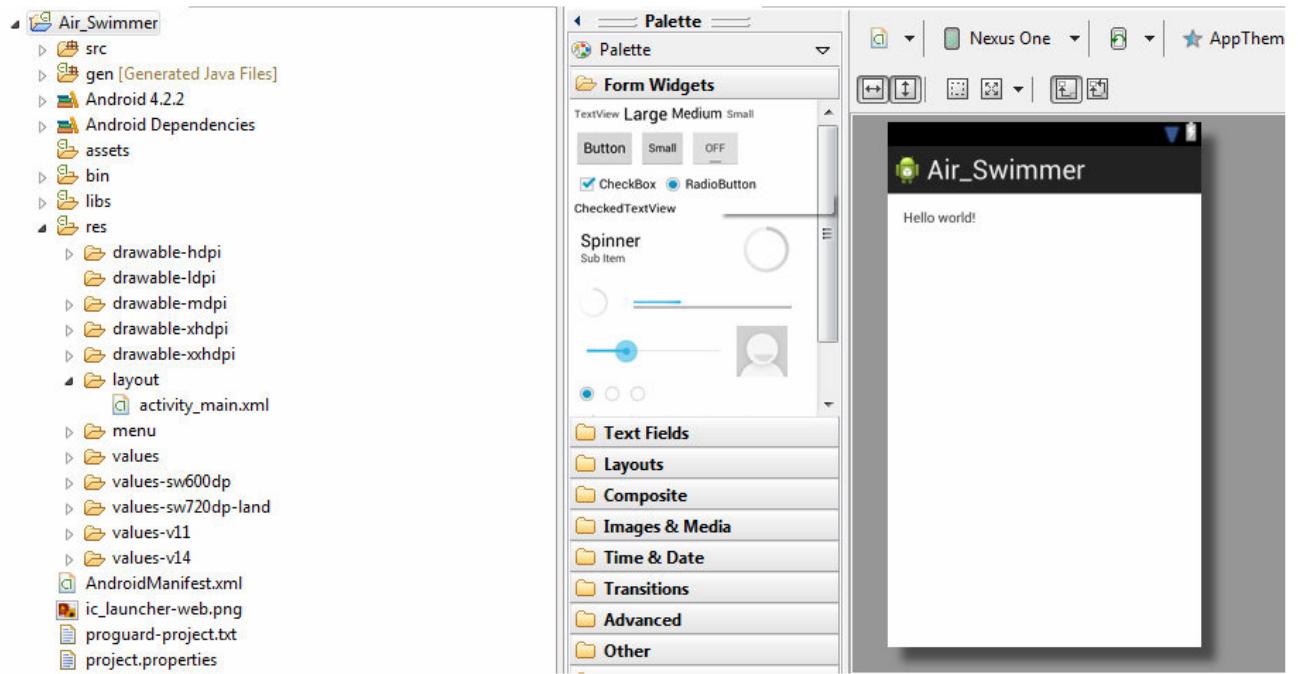


Abbildung 42

Oberflächenentwicklung in Eclipse

Dieses enthält folgende XML-Struktur:

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    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=".MainActivity" >

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/hello_world" />

</RelativeLayout>
```

Zu allererst wird dabei die TextView entfernt. Unter dem Punkt Image & Media werden 4 ImageButtons in das Layout gezogen und durch einen Doppelklick gelangt man direkt in die XML-Datei. Anhand dieser Zeile in der XML-Datei wird die entsprechende Grafik in den Button gesetzt.

`android:src="@drawable/up"`

Auch wird der Hintergrund des Layouts geändert indem beim RelativeLayout die folgende Zeile hinzugefügt wurde:

```
android:background="@drawable/ic_sky"
```

Diese Grafik stellt ein Wolkenhintergrund dar und wird vom drawable-hdpi aufgerufen. Um die Buttons transparent zu gestalten, wird folgender Code editiert.

```
android:background="#00000000"
```

Um auch später die Bewegungssteuerung für die Buttons schnell und einfach zu implementieren wurde für jeden Button ein ButtonListener eingebaut. Ein Listener überprüft wann und ob ein Button gedrückt wurde. Falls ein Button gedrückt wurde kann man eine Funktion zur Steuerung implementieren.

```
button_up = (ImageButton) findViewById(R.id.imageButtonUp);  
button_up.setOnTouchListener(this);
```

### Ergebnis

Durch diese Vorgehensweise wurde ein Layout mit Buttons-Steuerung erstellt. Somit wurde der Start zur Oberflächenprogrammierung gegeben. Auf diese Oberflächen wurden die folgenden Ansteuerungsarten gebaut.

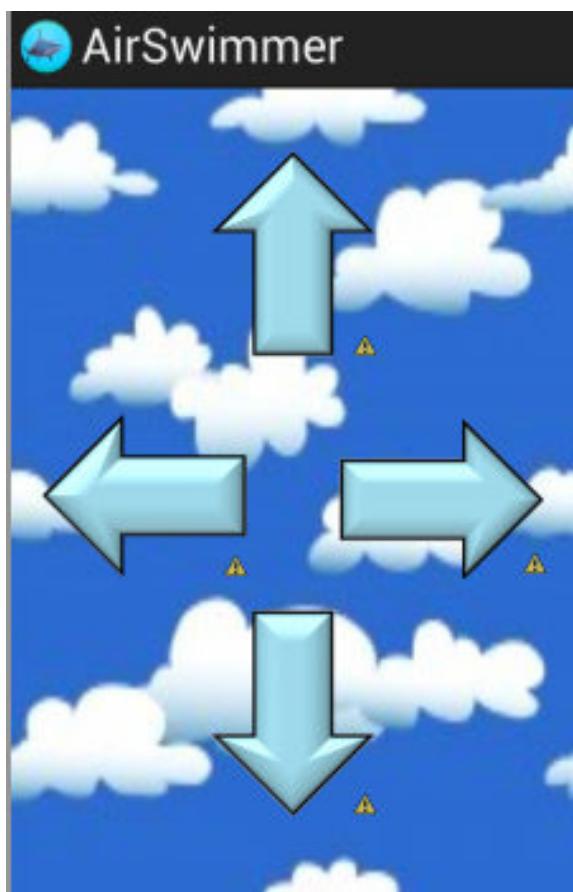


Abbildung 43  
Oberfläche mit Image-Buttons

### **7.4.3 Reaktion auf Druck und visueller Effekt**

Von Caroline Pilot

#### **Zielsetzung**

Da die Tasten bei einer einfachen „if-Abfrage“ in den „Activtiy\_Button“ Klassen nur auf ein Loslassen der Tasten vom Benutzer reagieren, muss hier, vor allem aus Logikgründen, eine Reaktion beim Drücken der Tasten implementiert werden. Zudem soll ein visueller Effekt dem Benutzer deutlich machen, wann ein Signal gesendet wird und wann nicht.

#### **Vorüberlegung**

Die Reaktion beim Drücken bedarf keiner weiteren Überlegung.

Der visuelle Effekt jedoch, kann verschieden implementiert werden:

- Ein Text mit der Aufschrift der jeweiligen Richtung besteht bereits, dieser wird allerdings nur für eine kurze Zeitdauer angezeigt
- Ein farblicher Rahmen um die Tasten beim Druck
- Die gedrückte Taste an sich, könnte die Farbe verändern

**Ergebnis:** Der Text soll weiterhin kurz angezeigt werden, dies soll der reinen Information für den Benutzer dienen und außerdem wird so sichergestellt, dass auch tatsächlich die Signale für beispielsweise die Aufwärtsbewegung ausgegeben werden. Auch die Taste an sich soll keine Farbänderung erfahren, denn dies könnte möglicherweise nicht direkt gesehen werden, wenn der Benutzer seinen Finger darauf legt.

So bleibt die gebräuchliche Möglichkeit, einen dickeren Rahmen um jede Taste zu legen und diesen beim Druck einzufärben und beim Loslassen verblassen zu lassen.

#### **Vorgehensweise**

##### **Reaktion beim Druck der Tasten**

Durch eine einfache Erweiterung der „If-Anweisung“ in der Base-Class der Buttons kann die Reaktion nur beim Druck erreicht werden.

```
if (event.getAction() == MotionEvent.ACTION_DOWN){...}  
bzw.  
if(event.getAction() == MotionEvent.ACTION_UP){...}
```

#### **Visueller Effekt**

Mit dem Befehl lässt sich der Hintergrund der Buttons einfärben:

```
button_up.setBackgroundColor(Color.BLUE);
```

Dies setzt den Hintergrund des „button\_up“ Bildes auf Blau. Diese Farbe wurde gewählt, weil man sie auf jedem Hintergrundbild gut sehen kann und auch zu den Farben des AirSwimmers passt.

Da dies erst geschehen soll, sobald der Nutzer die Taste drückt (`ACTION_DOWN`), wird dieser Befehl in die erste „if-Anweisung“ eingefügt und ein passender Befehl (`Color.TRANSPARENT`), welcher den Hintergrund der Taste wieder farblich zurücksetzt, in die zweite „if- Anweisung“ eingefügt.

## **Ergebnis**

Der Benutzer erhält nun visuelle Rückmeldung, dass der Button gedrückt wird, diese Farbänderung hält die ganze Zeit an, bis der Benutzer den Finger wieder hebt. In genau dieser Zeit sollen auch die Signale an den Fisch gesendet werden.

## **7.5 Steuerung durch Wischen**

*von Anja Hafner*

### **Ziel:**

Ziel ist es, eine Ansteuerung des AirSwimmers per Drag and Drop Prinzip zu realisieren. Dabei soll ein Bild, das in dem „drawable“-Ordner im „res“-Ordner des Projektes gespeichert ist, per Drag and Drop Prinzip über den Handybildschirm bewegt werden. Unter Drag and Drop versteht man, dass mit dem Finger das Bild des Fisches in der App berührt wird und dieses in die gewünschte Position gezogen wird. Der AirSwimmer ahmt anschließend die Bewegung des Fisches in der App nach.

### **Vorgehensweise:**

Für die Steuerung der App durch Wischen wird ein neues Projekt mit dem Namen „SlideApp“ in eclipse erstellt.

Zunächst wird im „Graphical Layout“ der „activity\_layout\_slide.xml“ die Oberfläche erstellt. Dazu wird die Palette eingesetzt. In dem Ordner „Images & Media“ der Palette wird die ImageView in den Handybildschirm gezogen. In dem erscheinenden Dialogfenster wählt man in den Projektressourcen das Bild des Fisches aus, das eingefügt werden soll.

Alternativ kann das Bild auch als „ImageView“ direkt in der „activity\_layout\_slide.xml“ Datei mit dem Befehl „`android:src="@drawable/fish"`“ eingefügt werden (hier ist der Name des Bildes: fish).

Wird das Bild direkt in der xml-Datei eingefügt, befindet sich das Bild im oberen linken Rand des Bildschirms. Wird es hingegen per Hand in den Bildschirm gezogen, ist die Position frei wählbar. Das zu bewegende Bild soll zentriert angezeigt werden. Dazu wird im ImageView der Befehl „`android:layout_centerInParent="true"`“ hinzugefügt. In diesem Fall ist der Parent die View, in dem das Bild nun zentriert plaziert ist.



**Abbildung 44**

Bild des Graphical Layouts mit der Palette zur Erstellung der Oberfläche

Der Befehl „centerInParent“ existiert nur im „RelativeLayout“, darum wird dieses Layout gewählt.

Um auch den Hintergrund dem Thema entsprechend anzupassen, wird hier ein Bild mit dem Befehl: „android:background="@drawable/sea"" eingefügt. Dieses Bild befindet sich ebenfalls im „drawable“-Ordner des Projektes.

```

BaseSlide.java *activity_layout_slide.xml
1 <RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
2     xmlns:tools="http://schemas.android.com/tools"
3     android:layout_width="match_parent"
4     android:layout_height="match_parent"
5     android:background="@drawable/sea"
6     android:id="@+id/layout"
7     tools:context=".MainActivity" >
8
9     <ImageView
10         android:id="@+id/img"
11         android:layout_width="wrap_content"
12         android:layout_height="wrap_content"
13         android:layout_centerInParent="true"
14         android:src="@drawable/fish" />
15
16 </RelativeLayout>

```

**Abbildung 45**

Screenshot der „activity\_layout\_slide.xml“ Datei

Die Bewegung des Fisches wird in der „BaseSlide.java“-Datei implementiert. Sie wird mit Hilfe eines sogenannten „onTouch“-Listeners realisiert. Dieser ist in zwei Ausführungen im Programm vorhanden: Einmal ein „onTouch“-Listener für die ViewGroup (eine View, die andere Views beinhalten kann, in diesem Fall das Bild des Fisches) und für das Bild, welches bewegt werden soll.

Im „onTouch“-Listener des Bildes wird mit „MotionEvent.ACTION\_DOWN“ überprüft, ob der Benutzer das Touch-Display berührt. Ist dies der Fall, wird ein neues Offset ermittelt. Mit diesem Offset wird im „onTouch“-Listener der ViewGroup die aktuelle Position des Bildes berechnet.

Findet keine Berührung statt, passiert nichts.

```
ImageView img = (ImageView) findViewById(R.id.img); // fish image is initialized
img.setOnTouchListener(new View.OnTouchListener() { // onTouchListener for fish

    @Override
    public boolean onTouch(View view, MotionEvent event) {
        switch (event.getActionMasked()) {
            case MotionEvent.ACTION_DOWN: // if someone touches the fish
                offset_x = (int) event.getX(); // calculate new offset
                offset_y = (int) event.getY();
                selected_item = view; // view is selected
                break;

            default: // fish is not touched
                break;
        }
        return false;
    }
});
```

**Abbildung 46**  
onTouch-Methode

Der „onTouch“-Listener der ViewGroup wird in der Methode „onCreate“ gesetzt. Der Listener wird in dieser Methode aufgerufen, damit der Fisch nicht bei jeder beliebigen Berührung an die berührte Stelle springt.

In diesem Listener wird ebenfalls überprüft, ob eine Bewegung innerhalb des Bildes während einer Berührung stattfindet. Hierbei werden die Koordinaten des Bildes mit der Methode „getLocationOnScreen“ ermittelt. Diese Koordinaten werden von den Koordinaten des gedrückten Punktes abgezogen. Anschließend wird überprüft, ob die Bewegung innerhalb eines bestimmten Toleranzbereiches liegt.

```
(if (MotionEvent.ACTION_MOVE >= imageX - 70 &&
    MotionEvent.ACTION_MOVE < imageX + 70 || 
    MotionEvent.ACTION_MOVE >= imageY - 30 &&
    MotionEvent.ACTION_MOVE < imageY + 30).
```

Wird das Bild berührt, wird die Methode zum Bewegen des Fisches („moveImage“) aufgerufen. Die Methode „moveImage“ ermöglicht die Bewegung des Fischbildes. Wird das Bild bewegt, wird die neue Position des Fisches in der x- und der y-Achse jeweils mit dem im „onTouch“-Listener ermittelten Offsets berechnet. Um zu verhindern, dass der Fisch an allen vier Seiten aus dem Bildschirm herausgezogen werden kann, wird mit „get WindowManager().getDefaultDisplay().getWidth“ und „get WindowManager().getDefaultDisplay().getHeight“ die Breite und Höhe des Displays ermittelt. Mit diesen Werten ist es möglich, mit einer einfachen if-Abfrage zu ermitteln, ob der Fisch zu weit nach links/rechts oder oben/unten gezogen wurde. Wird das Bild nicht bewegt, passiert nichts.

Es fällt auf, dass der Fisch bei Annäherung an den rechten und unteren Rand der App kleiner wird.

Durch Ersetzen von „`android:layout_width="fill_parent"`“ und „`android:layout_height="fill_parent"`“ durch „`android:layout_width="match_parent"`“ und „`android:layout_height="match_parent"`“ wird das Verkleinern des Fisches verhindert.



**Abbildung 47**  
Bild der SlideApp

## 7.6 Steuerung durch Kippen

Von Belgüzar Kocak

### Zielsetzung

Ziel war es eine Grafik anhand der Sensoren einer Android Hardware je nach Neigung nach rechts, links, oben und unten über den Bildschirm zu bewegen.

### Vorgehen

Bevor die Implementierung der Funktionen starten konnte, musste eine Lösung zur Aufgabe gefunden werden. Dafür wurde eine Recherche die über längere Zeit andauert geführt um die Zielsetzung zu realisieren. Dabei wurde die Sensorfunktion der Hardware untersucht. Sowie Methoden für der Android Programmierung zur Realisierung erforscht.

*Wir werden eine Oberfläche erstellen, die ein Bitmap Image auf ein Canvas<sup>1</sup> zeichnet und es anhand des Accelerometer<sup>2</sup> Sensors bewegt.*

Die Sensoren werden von einem zentralen Android-Dienst verwaltet, dem `SensorManager`. Um sich eine Referenz auf diesen Dienst zu holen wird die Methode `getSystemService()` verwendet.

---

<sup>1</sup> **Canvas** ist eine Leinwand, die in der Android Programmierung genutzt wird um Linien, Kreise, Rechtecke, Texte & Bilder zu malen.

<sup>2</sup> **Accelerometer** ist ein Beschleunigungssensor, der schon kleinste Neigungen und Erschütterungen registriert.

```
sm = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
```

sm stellt hier den SensorListener dar.

Auch ist es wichtig sich vom Sensor eine Referenz auf den richtigen Sensor zu verschaffen.

```
sm.getSensorList(Sensor.TYPE_ACCELEROMETER)
```

Diese Funktionen werden in die OnCreate-Methode gesetzt.

Als nächstes gilt es eine Grafik auf der Oberfläche bewegen und auch abilden zu lassen.

Vorab wird noch eine Methode eingebaut, um die Sensoränderungen zu detektieren und somit die Grafik je nach Sensoränderung zu bewegen.

```
public void onSensorChanged(SensorEvent event) {  
    sensorY = event.values[0];  
    sensorX = event.values[1];  
}
```

Um das Zeichnen des Images zu beginnen implementieren wir die Methode drawSharkImage(). In dieser Methode wird ein Canvas erstellt und die Bitmap darauf abgebildet. Vorher wird die Grafik als eine Bitmap definiert.

Dies wird in der onCreate() Methode definiert:

```
shark =  
BitmapFactory.decodeResource(getResources()R.drawable.bellishark_1_medium)
```

Die drawSharkImage() Methode:

Wie vorab schon erläutert muss eine neue Canvas definiert werden:

```
Canvas canvas = ourHolder.lockCanvas();
```

Danach wird der Startpunkt der Grafik bestimmt

```
float startX = 350;  
float startY = 150;  
float offsetX = 0;  
float offsetY = 0;
```

In den beiden Variablen offsetX und offsetY wird die winkelabhängige Verschiebung des Fisches von der Startposition gespeichert. Wie die Berechnung dieser Werte so durchgeführt wird, dass die Grafik nicht bei der minimalsten Erschütterung zittert wird im Absatz „Schwierigkeiten der Oberfläche“ näher erläutert.

Um die Grafik auf dem Canvas (Leinwand) abzubilden muss dieser gemalt werden.

```
canvas.drawBitmap(shark, startX + offsetX, startY + offsetY, null);
```

Zusätzlich wird ein Thread geschrieben um das bewegte Image vom Hai zu malen. Dort benötigen wir einen SurfaceHolder, der mit der View über ein Interface kommuniziert.

```

@Override
public void run() {
    while (isRunning) {
        if (!ourHolder.getSurface().isValid()) {
            continue;
        }
        drawSharkImage();
    }
}

```

Der Thread, der unabhängig vom Main-Thread läuft, wird durch das Runnable-Interface implementiert und gestartet, wenn die Activity erzeugt wird. Zum Steuern des Threads werden zusätzlich eine „pause“-Methode, sowie eine „resume“-Methode, die nach der Pause DrawShark fortsetzt, und eine „stop“-Methode eingebaut.

```

// pauses DrawShark thread
public void pause() {
    isRunning = false;
    while (true) {
        try {
            ourThread.join();

        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        break;
    }
    ourThread = null;
}

// resumes DrawShark thread
public void resume() {
    isRunning = true;
    ourThread = new Thread(this);
    ourThread.start();
}

// Stops DrawShark thread
public void stop() {
    isRunning = false;
}

```

Somit wird die Kippen Oberfläche realisiert.

### **Schwierigkeiten der Oberfläche**

Von Sabine Kressierer

Da in dieser Oberfläche ein Bild dargestellt wird, dass sich permanent bewegen muss, unterscheidet sich die Implementierung etwas von den anderen Oberflächen. Dies hat einige Besonderheiten zur Folge, die berücksichtigt werden mussten.

Die Besonderheiten beim Ändern des Hintergrunds werden im Kapitel 7.8 (Menü) beschrieben.

## Thread

Da für das Zeichnen des Fisches ein Thread verwendet wird muss dieser an den richtigen Stellen beendet und gestartet werden. So muss er bei jedem Verlassen der Oberfläche gestoppt werden, da sonst bei einigen Geräten eine Fehlermeldung auftritt. Dies geschieht mithilfe der oben beschriebenen Stop-Methode des Threads, die an den richtigen Stellen eingefügt werden musste.

Es gibt mehrere Möglichkeiten die Kippen-Oberfläche zu verlassen:

1. Menüpunkte „Startseite“, „Modus wählen“, „Steuerungsart wählen“
2. Betätigen der „zurück“-Taste
3. Betätigen der „home“-Taste und somit Schließen der App

Da für die Änderung des Hintergrundes bereits die Methode

`onOptionsItemSelected(MenuItem item)` überschrieben wurde, musste für Punkt eins dort nur noch eine Abfrage auf die entsprechenden Menüpunkte und die Methode zum Beenden eingefügt werden:

```
if (item.getGroupId() == R.id.submenu_changeMode  
    || item.getGroupId() == R.id.submenu_changeMove  
    || item.getItemId() == R.id.frontPage) {  
    ourView.stop(); // stops drawing thread before mode is changed  
}
```

Um die Reaktion auf die „zurück“-Taste zu verändern, wird die Funktion

```
public boolean onKeyDown(int keyCode, KeyEvent event)
```

der Activity-Klasse überschrieben. Diese wird aufgerufen, wenn eine Taste gedrückt wurde, die durch keine der Views innerhalb der Activity behandelt wurde. Der Parameter `keyCode` repräsentiert die gedrückte Taste und entspricht im Falle der „zurück“-Taste der Konstanten `KeyEvent.KEYCODE_BACK`. Somit wird in der `OnKeyDown`-Funktion der `KeyCode` mit dieser Konstanten verglichen und bei Gleichheit der Thread gestoppt.

Die Veränderung der Reaktion auf die „home“-Taste könnte prinzipielle genauso funktionieren, wie bei der „zurück“-Taste. Dies kann jedoch bei Fehlern dazu führen, dass eine App nicht mehr beendet werden kann. Aus diesem Grund kann diese Taste nicht so leicht überschrieben werden. Im Internet gibt es zwar einige Lösungsvorschläge dazu, diese funktionierten jedoch alle nicht. Da bei Verlassen der App der Thread sowieso beendet wird, und die Fehlermeldung auf dem Zieltablet nicht auftrat, wurde von weiteren Versuchen abgesehen.

Damit bei Rückkehr zur Oberfläche durch Betätigen des Zurück-Knopfes die im Thread realisierte Animation wieder stattfindet, wird dieser in der Methode `onResume()`, die bei Rückkehr zur Activity aufgerufen wird, wieder gestartet.

## Zittern des Fisch-Bildes

Reagiert der Fisch auf die kleinsten Bewegungen hat dies zur Folge, dass das Bild zittert, wenn das Tablet ruhig auf dem Tisch liegt. Um dies zu verhindern wird eine Variable `delay` eingeführt, mit der kleinere Bewegungen unterdrückt werden:

```
double delay = 0.4; // sensitivity of sensor  
float offsetX = 0;  
float offsetY = 0;  
// ignore tilt between ]-delay;delay[  
if (sensorX > delay) {  
    offsetX = (float) ((100.0 / 360.0 * (sensorX - delay)) * width / 4);  
} else if (sensorX < delay * -1) {
```

```
        offsetX = (float) ((100.0 / 360.0 * (sensorX + delay)) *
width / 4);
}
```

Die Belegung der Variablen mit 0,4 ergab sich aufgrund einiger Tests, die zeigten, dass dies der kleinste mögliche Wert ist, ab dem das Zittern akzeptabel ist, ohne die Bewegung zu stark zu beeinflussen. Der Wert `offsetX` wird nur mit einem Wert ungleich null belegt, wenn der Betrag des entsprechenden Sensorwertes größer als der in `delay` gespeicherte Wert ist. Somit verschiebt sich der Fisch bei kleineren Bewegungen nicht. Um zu verhindern, dass der Fisch bei Überschreiten dieser Grenze plötzlich an eine andere Position springt, wird dieser Wert vom eigentlichen Sensorwert abgezogen, beziehungsweise bei negativem Sensorwert addiert. Die Berechnung des tatsächlichen `offset`-Wertes erfolgte unter der Annahme, durch den Sensor die Angabe des Neigungswinkels zu erhalten. Damit würde die Neigung proportional auf die Bildschirmbreite umgerechnet werden. Da dies nicht der Fall ist, aber die Beschleunigungswerte auch nicht innerhalb eines bestimmten Bereichs lagen, wurde diese Formel beibehalten, und durch mehrere Versuche mithilfe eines Faktors an die Bildschirmbreite angepasst. Die Bestimmung von `offsetY` erfolgt analog dazu.

## Orientierung der Achsen

Die im Abschnitt „Vorgehen“ beschriebene Zuweisung der SensorEvent Werte zu den entsprechenden Achsen funktioniert nur im Landscape-Modus. Da die App anfangs nur darauf ausgelegt war, konnte dies einfach so zugewiesen werden. Später sollte es möglich sein die Orientierung zu Ändern, wodurch auch diese Zuweisung entsprechend der aktuellen Orientierung geschehen musste. Dazu wurde eine in der `onSensorChanged`-Methode folgende Abfrage eingebaut, die aufgrund des zuvor gespeicherten Wertes für die Orientierung des Bildschirms, die Werte den entsprechenden Achsen zuweist. Zudem wurde später festgestellt, dass bei einigen Tablets, so auch dem vorgegebenen Zieltablet, die Achsen vertauscht sind. Da auf den meisten anderen Geräten obige Zuweisung stimmt, wird über die Konstanten `android.os.Build.BRAND` und `android.os.Build.DEVICE` ermittelt, ob es sich um das entsprechende Zieltablet handelt und in dem Fall die Zuweisung entsprechend angepasst.

## Einbau eines Start-Buttons

Von Marco Bengl

### Motivation

Da der Kippenmodus sofort nach dem Drücken des Buttons auf der Startseite aktiv wird, bewegt sich der AirSwimmer zu Beginn völlig unkontrolliert. Um diesen plötzlichen Start zu vermeiden wurde beschlossen, dass im Kippenmodus ein Start/Stopp Button entstehen soll. Somit bietet sich die Möglichkeit den Kippenmodus gezielt starten, oder auf beenden zu können.

### Zielsetzung

Ziel war es einen Button in die Oberfläche des Kippenmodus zu integrieren um die Bedienung zu erleichtern.

### Vorgehensweise

Die Oberfläche des Kippenmodus ist keine klassische Android-Oberfläche, wie der Wischen- oder Tastenmodus, sondern wird aus einer Bitmap erstellt. Somit gestaltete sich die Integrierung eines Buttons in diese Oberfläche schwieriger als gedacht. Um das Vorhaben zum Erfolg zu führen wurden mehrere Ansätze ausprobiert die im Folgenden erläutert werden:

### 1. Integrierung des Buttons in die Bitmap

- Die allgemeine Problematik die sich ergab war, dass man in einer Bitmap keine Buttons sichtbar machen konnte. Somit wurde nach etlichen versuchen klar, dass entweder die Bitmap sichtbar war, oder nur ein weiser Hintergrund und der gewünschte Button.
- Ein weiterer Gedanke war anschließend den Button direkt mit der Bitmap zeichnen zu lassen. Dies war allerdings nicht zielführend da man den Button dann nicht als solchen erfassen könnte.

### 2. Platzierung des Buttons außerhalb der Bitmap

Als nächstes wurde versucht die Bitmap zu verkleinern, sodass am unteren Rand eine Fläche frei wurde um dort den Button zu setzen. Es stellte sich jedoch heraus, dass die Bitmap das gewünschte Bild zwar verkleinerte, jedoch trotzdem den ganzen Bildschirm überschrieb. Somit war auch diese Möglichkeit ein Fehlversuch

### 3. Erstellung einer neuen Ebene über der Bitmap

Aufgrund der beiden oben genannten Fehlschläge wurde beschlossen den Button nicht direkt in die Oberfläche des Kippenmodus zu integrieren. Vielmehr wurde eine Ebene zwischen Starseite und Kippenmodus erstellt in der ein Startbutton integriert wurde.

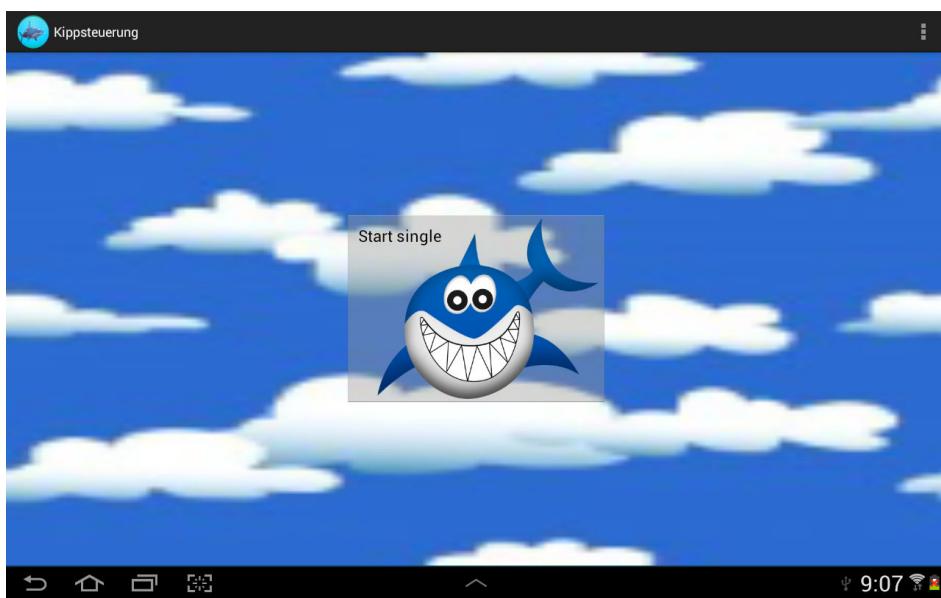
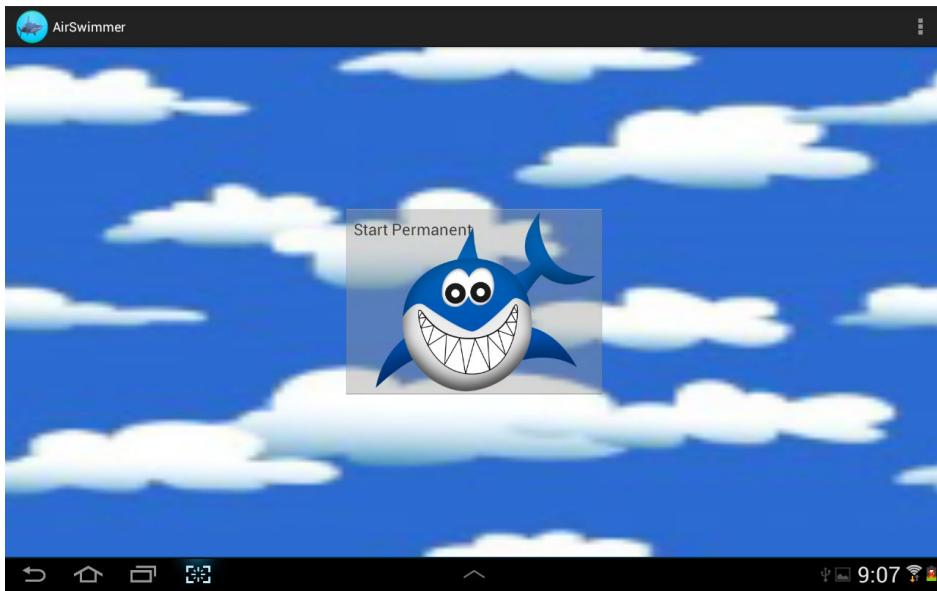


Abbildung 48

Startbutton mit Titelleiste „Kippsteuerung“



**Abbildung 49**

Startbutton mit Titelleiste „AirSwimmer“

Dieser Ebene wurde für den Single-Kippen und Permanent-Kippen separat erstellt, da das Starten über eine Ebene einen deutlich höheren Programmietechnischen Aufwand erfordert hätte. Das Prinzip wie der Button funktioniert ist sehr einfach: Beim drücken des Buttons wird die jeweilige Kippenklasse aufgerufen und gestartet.

```

public void onButtonClick(View view) {
    switch (view.getId()) {
        case R.id.Button_Button:
            startActivity(new Intent(this, Activity_Buttons.class));
            break;
        case R.id.Button_Tilt:
            startActivity(new Intent(this, Start_button_tilt.class));
            break;
        case R.id.Button_Slide:
            startActivity(new Intent(this, Activity_Slide.class));
            break;
        default:
            ;
    }
}

```

**Problem:** Es wurde jetzt „nur“ ein Startbutton realisiert. Und man war wieder mit dem Problem konfrontiert, einen Button in die Bitmap zu integrieren.

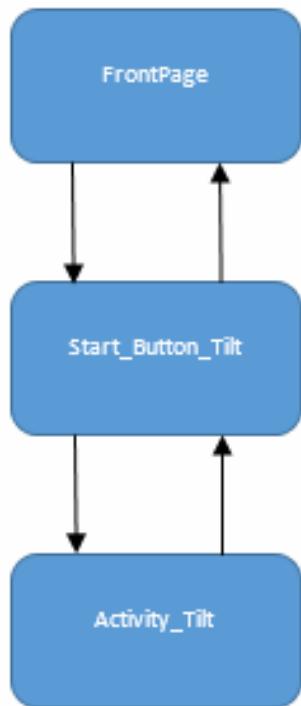
**Lösung:** Schließlich wurde klar, dass die Oberfläche auf die „onTouch-Funktion“ reagiert

```

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    // returns to StartPage if screen is touched
    ourView.setOnTouchListener(new View.OnTouchListener() {
        @Override
        public boolean onTouch(View v, MotionEvent event) {
            ourView.stop();
            startActivity(new Intent(ourView.getContext(),
                Start_button_tilt.class));
            return true;
        }
    });
}

```

Somit wurde der Backbutton realisiert in dem man auf eine beliebige Stelle auf dem Bildschirm drücken kann. Durch das Drücken auf den Bildschirm kommt man wieder auf die Zwischenebene mit dem Startbutton.



**Abbildung 50**  
Ablaufstruktur Kippen

## 7.7 Lautstärkenkalibrierung

### 7.7.1 Implementierung

Von Giuseppe De Nuzzo und Ridvan Yücel

Nachdem die Sendefunktion der AirSwimmers-App implementiert wurde, ergaben sich noch zusätzliche Anforderungen, die noch umgesetzt werden müssen. Durch Testen der App auf unterschiedlichen Android-Geräten, erwiesen sich deutliche Unterschiede.

Die Lautstärke muss so eingestellt werden, dass eine optimale Sendeleistung erreicht wird. Diese Einstellungen sind auf jedem Gerät verschieden.

Daher soll eine Kalibrierungsfunktion implementiert werden, die die Lautstärke optimal justiert. Diese Funktion erhöht stufenweise die Lautstärke und sendet dabei ein Befehl zum Airswimmers. Empfängt der Airswimmers dieses Signal, so bestätigt der Benutzer mit Drücken auf OK. Die Lautstärke wurde nun auf einen optimalen Pegel eingestellt.

```
protected void onStart(){
    super.onStart();
    bar.setProgress(0);

    new Thread(new Runnable() {
        @Override
        public void run() {
            try {
                Thread.sleep(5000);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }

            bar.setProgress(1);

            for(int i=1;i<16;i++){
                if(stopped==true){
                    break;
                }

                bar.setProgress(i);
                //comand send signal move_fish

                try {
                    Thread.sleep(2000);
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }

                value = i;
            }

            runOnUiThread(new Runnable() {
                @Override
                public void run() {
                });
        });
    }).start();
}
```

**Abbildung 51**  
Code der Lautstärken-Kalibrierung

Die Funktion wurde in der onStart Methode implementiert und startet daher genau nach Aufrufen dieser Funktion im Menü der Airswimmers-App. Die Kalibrierung wird durch Threads realisiert, um so eine parallele Zusammenarbeit mit anderen Funktionen, sowie die Kommunikation mit dem Benutzer zu gewährleisten. Das Erhöhen der Lautstärke und das Senden eines Befehls geschieht in einer For-Schleife, die bei 1 anfängt und bis zum maximalen Wert der Lautstärke durchläuft. Wird die OK-Taste durch den Benutzer betätigt, so wird die Variable **Stopped** auf true gesetzt und

die Schleife vorzeitig beendet. Der aktuelle Schleifenzählerwert wird in die Variable „Value“ gespeichert und zukünftig als Standardlautstärke für die AirSwimmers-App gespeichert.

## 7.7.2 Oberfläche

von Anja Hafner

### Ziel:

Ziel ist es, eine Oberfläche zur Lautstärkeneinstellung zu entwickeln. Das Fenster soll beim erstmaligen Öffnen der App erscheinen und die ausgewählte Lautstärke im Gerät einstellen und diese speichern. Nach dem erstmaligen Einstellen besteht die Möglichkeit, sie über das Menü bei Bedarf erneut zu ändern.

### Vorgehensweise:

Die Anzeige des momentanen Lautstärkenlevels erfolgt über eine sogenannte „SeekBar“. Diese ist in der Palette der „activity\_set\_sound.xml“ Datei unter „Form Widgets“ zu finden. Die SeekBar wird mittig in den Handybildschirm platziert. Darüber steht als kurze Erklärung in einer TextView, dass bei Bewegung des Fisches auf OK gedrückt werden soll und unter der SeekBar ist der OK-Button platziert.

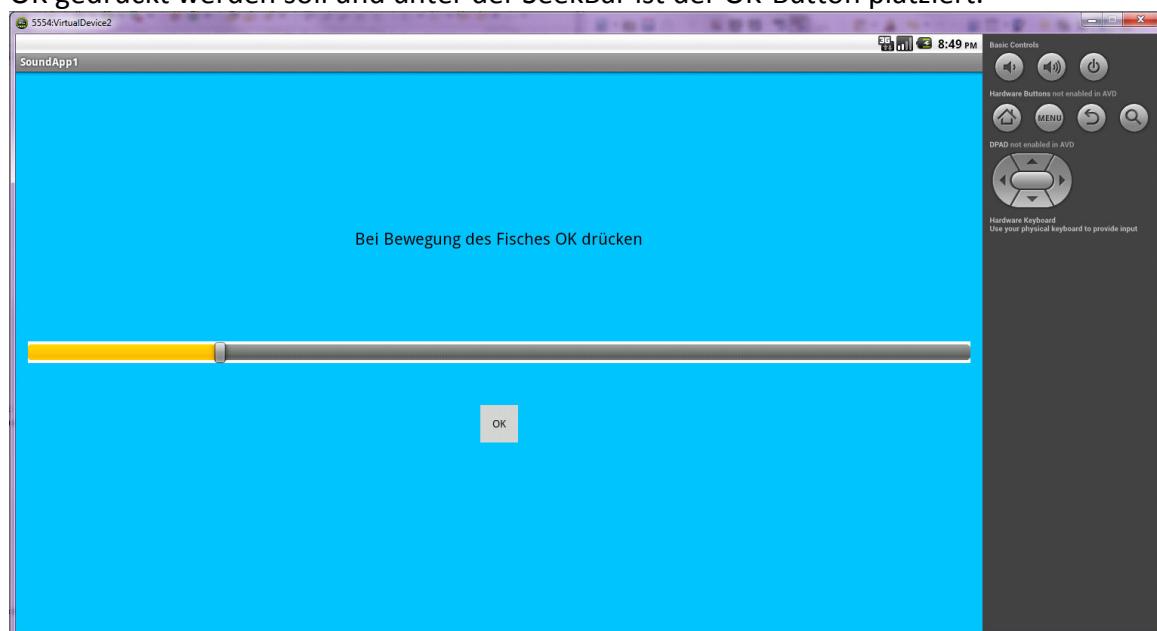


Abbildung 52

App zur Ermittlung der richtigen Lautstärke

Mit dem Befehl „`android:max="15"`“ in der „activity\_set\_sound.xml“ Datei wird der SeekBar ein maximaler Einstellungswert von 15 zugewiesen (da die Lautstärkenangabe bei Android-Geräten in einem Bereich von 0 bis 15 angegeben und eingestellt wird).

Um den ermittelten Wert der SeekBar zu erhalten, wird der SeekBar ein sogenannter „OnSeekBarChangeListener“ zugewiesen. Dieser zeigt an, ob der Fortschritt der SeekBar verändert wurde. Diese Änderung wird mit der Methode „`onProgressChanged(SeekBar seekBar, int progress, boolean fromUser)`“ erkannt.

Mit einem „OnTouchListener“ und mit Hilfe der Methode „onTouch“, wird der Button zum Bestätigen der Bewegung implementiert.

Wird der Button gedrückt (if (event.getAction() == MotionEvent.ACTION\_DOWN)) ändert er seine Farbe (als Feedback dass gedruckt wurde) und ruft den Fortschritt der SeekBar mit „bar.getProgress()“ auf.

Mit Hilfe des AudioManager kann die Lautstärke am Audiojack-Ausgang geregelt werden. Der Befehl

„audioManager.setStreamVolume(AudioManager.STREAM\_SYSTEM, value, 0);“ setzt die Lautstärke auf den ermittelten Wert (value).

Wird der Button losgelassen (if (event.getAction() == MotionEvent.ACTION\_UP)), ändert er lediglich seine Farbe.

```
61@     @Override
62@     public void onClick(View v) {
63@ 
64@         if (v == okbutton) {
65@             okbutton.setOnTouchListener(new OnTouchListener() { //create OnClickListener for okbutton
66@ 
67@                 @Override
68@                 public boolean onTouch(View v, MotionEvent event) { //called when screen is touched
69@ 
70@                     if (event.getAction() == MotionEvent.ACTION_DOWN) { //if button is pressed
71@                         okbutton.setBackgroundColor(Color.GRAY); //change button colour (so you can see that button has been pushed)
72@                         int value = bar.getProgress();
73@                         AudioManager audioManager = (AudioManager) getSystemService(AUDIO_SERVICE);
74@                         audioManager.setStreamVolume(AudioManager.STREAM_SYSTEM, value, 0);
75@ 
76@                         return true;
77@                     }
78@                     else if (event.getAction() == MotionEvent.ACTION_UP){ //if button is released
79@                         okbutton.setBackgroundColor(Color.LTGRAY); //change button colour again (so you can see that button has been released)
80@                         return true;
81@                     }
82@ 
83@                     return false;
84@                 }
85@             });
86@         }
87@     }
88@ }
89@ }
```

### Abbildung 53

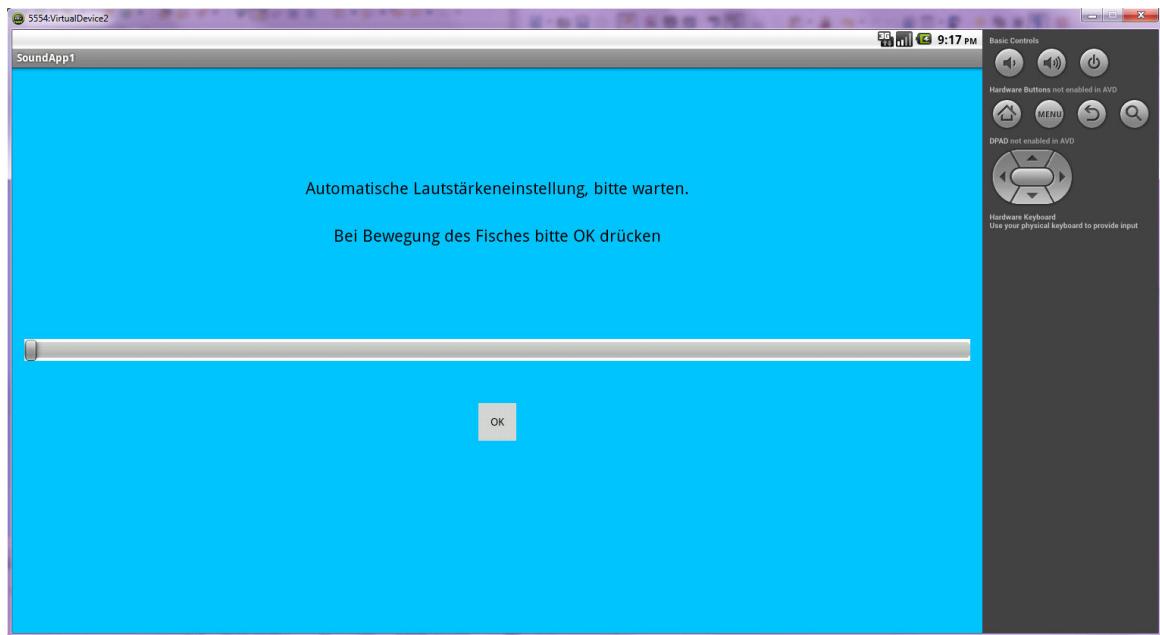
Bild des „onTouch“-Listeners mit Setzen der Lautstärke

### Korrekturen:

Es folgen noch ein paar kleinere Korrekturen.

So soll dieSeekBar nicht per Hand veränderbar sein. Mit Hilfe des Befehls „setEnabled(false)“ ist eine manuelle Steuerung nicht mehr möglich.

Auch ist die momentane Beschreibung über derSeekBar nicht sehr eindeutig und der Benutzer weiß nicht genau, wie er sich verhalten soll. Zur Vermeidung unnötiger Verwirrung wird eine zusätzliche TextView mit einer näheren Beschreibung („Automatische Lautstärkeneinstellung, bitte warten.“) hinzugefügt.



**Abbildung 54**

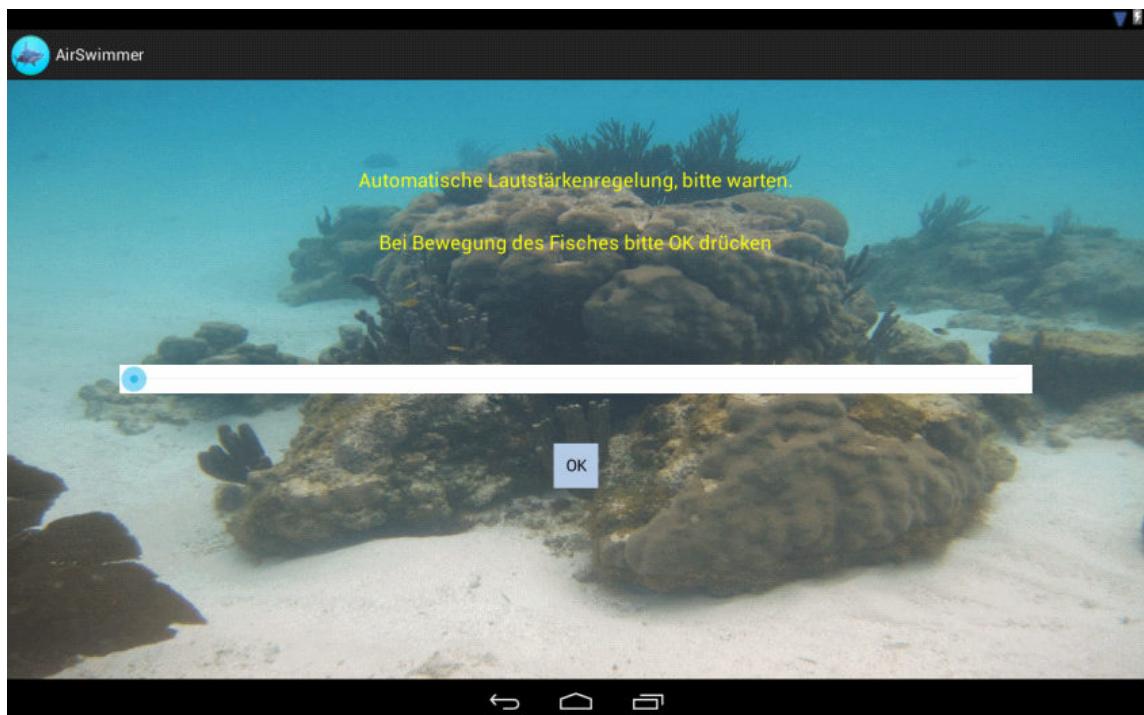
Bild der SoundApp mit erweiterter Erklärung

Für eine ansprechendere Optik wird der Hintergrund der Lautstärkeneinstellung an den Hintergrund der Startseite angepasst.

Mit dem Befehl „`android:background="@drawable/frontpage_background"`“ wird das Bild eingefügt.

Da die schwarze Schrift nicht sehr gut auf dem dunklen Hintergrund zu sehen ist, wird sie mit „`android:textColor="#f6f60e"`“ in gelb umformatiert.

Der Balken der SeekBar muss eine helle Farbe besitzen, da er sonst nicht erkennbar ist.



**Abbildung 55**

Lautstärkeneinstellung mit neuem Hintergrund

### **7.7.3 Einbau und permanentes Speichern**

Von Marco Bengl

#### **Erstellung des Startmenüpunktes**

Die Lautstärkeregelung sollte das Menü in der Startseite erreichbar sein.



**Abbildung 56**

Startseite mit Lautstärkenregulierung im Menü

Somit wurde zunächst der Menüpunkt „Info“ dem Menü auf der Startseite hinzugefügt. Dies wird erreicht indem man in der Klasse „FrontPage“ in der Funktion `onOptionsItemSelected` den Fall

```
else if (item.getItemId() == R.id.sound_calc) {  
    startActivity(new Intent(this, SetSoundActivity.class));  
}
```

hinzufügt.

#### **Integration in die Oberfläche**

Da die Funktionen der Lautstärkeregulierung separat von der Oberfläche entwickelt wurde, musste der Code noch in die Oberfläche integriert und mit den entsprechenden Funktionen verknüpft werden (Oberfläche siehe Abbildung 56(oben)).

#### **Dauerhafte Abspeicherung des Lautstärkewertes.**

Da es sehr lästig ist die Lautstärke für den IrDroid Sender jedes mal neu zu Kalibrieren wurde beschlossen, dass der Wert der bei der ersten Kalibrierung ermittelt wird, dauerhaft gespeichert wird. Dazu wird in der Funktion

```
protected void onCreate(Bundle savedInstanceState)
```

die Methode SharedPreference verwendet. Diese speichert einen Wert dauerhaft ab. Somit wird der bei der ersten Kalibrierung ermittelte Wert in einer Preference mit dem Schlüssel „voice“ abgespeichert.

```

prefs = getSharedPreferences("AirSwimmerPrefs", Context.MODE_PRIVATE);
editor = prefs.edit();
editor.putInt("voice", 0);
editor.commit();

```

Bei jedem weiteren Start der App wird dann auf diesen abgespeicherten Wert zurückgegriffen. Durch eine erneute Kalibrierung kann der Wert jedoch überschrieben werden.

#### **7.7.4 Einbau der Senden-Logik**

*Von Patrick Trojosky*

Nachdem die Oberfläche von Anja, Giuseppe und Marco erstellt wurde, konnte die Kalibrierung der Lautstärke in die Oberfläche eingebunden werden. Damit der Fisch aber überhaupt reagiert, muss noch die Senden Funktion hinzugefügt werden. Die Kalibrierung soll einmal alle Lautstärkelevel durchschalten und für zwei bis drei Sekunden ein Signal an den AirSwimmer senden. Dies wird relativ einfach über eine „For“-Schleife realisiert, die fünfzehnmal ausgeführt wird. Falls der Benutzer vorher eine Bewegung erkennt und ok drückt, bricht die Schleife vorher ab und der aktuelle Lautstärke Wert wird gespeichert. Zwischen den Sendevorgängen werden Verzögerungen eingefügt, sodass der Benutzer auch Zeit hat die ok Taste zu betätigen. Die genaue funktionsweise ist in Abbildung 62 zu erkennen.

```

for (int i = 1; i < 16; i++) {

    if (stopped == true) {
        break;
    }

    bar.setProgress(i);
    // command send signal move_fish
    for (int j = 0; j < 5; j++) {

        if (stopped == true){ //stop sending if ok-click occurred while going through for-loop
            break;
        }

        action.diving();

        try {
            Thread.sleep(400);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}

```

**Abbildung 57**  
Lautstärke Kalibrierung

## 7.8 Menü

### **Steuerungsoberflächen**

von Sabine Kressierer

#### **Zielsetzung**

In den verschiedenen Oberflächen zum Steuern des AirSwimmers soll ein Menü eingefügt werden, dass das Umschalten zwischen den einzelnen Steuerungsmodi sowie eine Auswahl des Hintergrund ermöglicht.

#### **Vorgehensweise**

Im Ordner „res“ wurde ein Ordner „menu“ angelegt, der die XML-Dateien zum darstellen der Menüs enthält. Da die Menüpunkte der verschiedenen Oberflächen zum Steuern des AirSwimmers gleich sind, wurde für sie nur eine XML-Datei „control.xml“ erstellt. Diese enthält die Menüpunkte mit allen Unterpunkten wie hier für den Menüpunkt „Hintergrund ändern“:

```
<item android:title="@string/change_background"
      android:id="@+id/change_background">
    <!-- Submenu with different background pictures -->
    <menu>
        <group android:id="@+id/submenu_changeBackground" >
            <item
                android:id="@+id/water"
                android:title="@string/water_picture"/>

            <item
                android:id="@+id/sky"
                android:title="@string/sky_picture"/>
            <item android:id="@+id/th_picture"
                  android:title="@string/th_picture"/>
        </group>
    </menu>
</item>
```

In einer Basisklasse „ BaseActivity.java“, von der alle Activities zum Steuern des Airswimmers ableiten, wird die Logik für das Menü implementiert. Das Erzeugen des richtigen Menüs erfolgt durch Überschreiben der onCreateOptionsMenu-Methode, in der den Activities mit Hilfe des Befehls  
getMenuInflater().inflate(R.menu.control, menu)  
das Menü der Datei „control.xml“ zugewiesen wird. Die Reaktion auf die Auswahl der Menüpunkte erfolgt in der Funktion onOptionsItemSelected, die ebenfalls in der Basisklasse überschrieben wird. Auf diese Weise werden die nachfolgenden Menüpunkte realisiert.

Bei der Oberfläche Kippen sind einige Besonderheiten zu beachten, die im Abschnitt „Schwierigkeiten der Oberfläche“ von Kapitel 7.6 (Steuerung durch Kippen) beschrieben werden.

## a) Hintergrund ändern

von Sabine Kressierer

### Zweck

Dem Benutzer soll die Möglichkeit zur Verfügung stehen zwischen verschiedenen Hintergründen für die Steuerungsoberflächen zu wählen.

### Umsetzung

Der Menüpunkt „change\_background“ enthält ein Untermenü mit den verschiedenen Hintergrundbildern. Wird einer dieser Punkte ausgewählt wird mittels `background.setBackgroundDrawable(source.getDrawable(R.drawable.ic_sky));` das entsprechende Bild (in diesem Fall ein Himmel) als Hintergrund verwendet. Hier muss die veraltete Methode verwendet werden, da dass Zieltablet die empfohlene, neuere Methode nicht unterstützt. „background“ ist in diesem Fall das Layout der Activity. Zur einheitlichen Verwendung hat das Layout bei allen Activities die Id „layout“.

Besonderheit der Oberfläche Kippen: Da hier der Hintergrund anstatt durch ein einfaches Drawable mithilfe einer BitMap dargestellt wird, muss die Funktion der Menüauswahl dort überschrieben werden. In der überschreibenden Funktion, wird der Hintergrund geändert, indem die Id des gewünschten Bildes mithilfe der Funktion `ourView.changeBackground(id)` in einer Variablen „background“ des Threads `ourView` gespeichert wird. Aufgrund der in „background“ gespeicherten id, wird bei jedem Durchlauf der Hauptschleife des Threads der Hintergrund neu gezeichnet. Somit wird beim nächsten Durchlauf das neue Bild angezeigt.

## b) Modus auswählen

von Sabine Kressierer

### Zweck

Um die Bedienung zu vereinfachen soll man nicht nur über die Startseite sondern auch über das Menü zwischen den einzelnen Steuerungsmodi (Kippen, Wischen, Tasten) wechseln können

### Umsetzung

Der Menüpunkt „change\_mode“ enthält ein Untermenü mit den Namen der einzelnen Steuerungsarten. Wird ein Punkt ausgewählt, wird mittels `startActivity(new Intent(this, Activity_Buttons.class))`; die entsprechende Activity gestartet. Um ein erneutes Auswählen der bereits laufenden Activity zu verhindern, wird bei Erzeugung des Menüs, der entsprechende Menüpunkt ermittelt und unsichtbar gemacht.

## c) Steuerungsart ändern

von Caroline Pilot

### Zielsetzung

Der Benutzer soll die Möglichkeit haben das Versenden von Signalen an den AirSwimmer über verschiedene Steuerungsarten zu realisieren.

Zum einen soll genau ein Signal (ein einziger kompletter Bewegungsbefehl) je Tastendruck ausgeführt werden (→ Einfach), zum anderen soll eine permanente Bewegung mit nur einem Klick ebenfalls möglich sein (→ Permanent). Diese Steuerungsart soll sowohl im Tastenmodus als auch im Wisch- und Kippmodus möglich sein.

Im Folgenden wird das Einfügen des Menüpunkts für die Tastensteuerung beschrieben, da die anderen Steuerungsmodi analog ablaufen.

### Vorüberlegung

Da es sich zwar bei der Optik jeweils um die gleichen Klassen handelt (Einfache Tastensteuerung und Permanente Tastensteuerung), jedoch die Logik dahinter anders ist, muss die vorhandene Klasse geklont werden, damit ein Menüpunkt später dorthin verweisen kann.

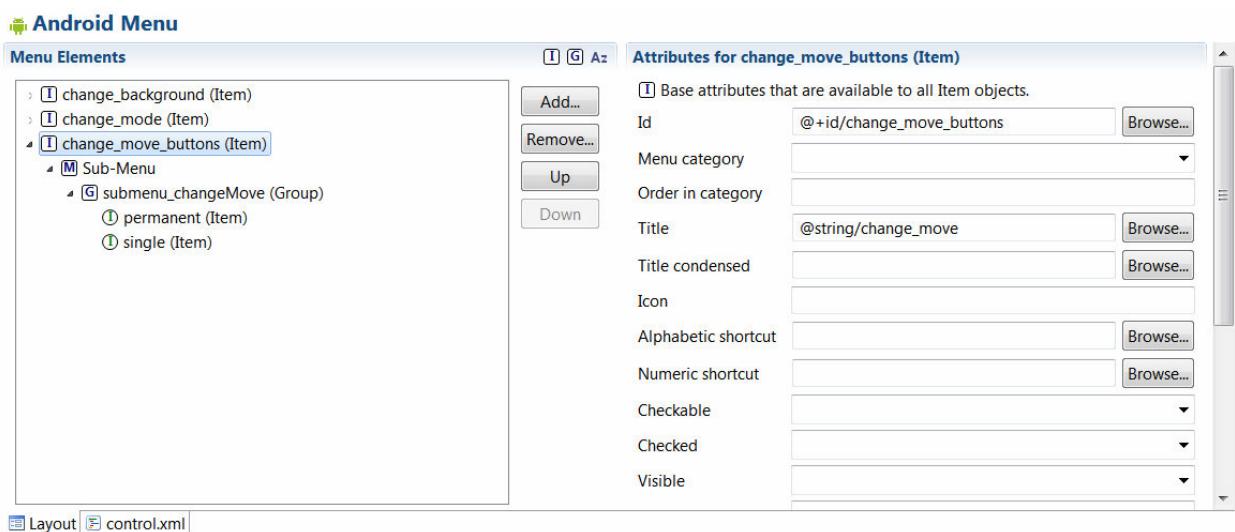
Dieser Menüpunkt („Steuerungsart ändern“) muss als ein weiterer in die Liste eingetragen werden, wobei ein Untermenü zum Wechseln zwischen „Einfach“ und „Permanent“ ebenfalls implementiert werden muss.

Damit der Benutzer auch weiß, in welchem Modus er sich befindet, muss der Android Action Bar Text (Wird am oberen Bildschirmrand während jeder Aktivität angezeigt) angepasst werden.

### Vorgehensweise

#### **Erstellen des Menüpunkts „Steuerungsart ändern“**

In der bereits vorhandenen *control.xml* Datei werden im Layout folgende Punkte hinzugefügt:



**Abbildung 58**

Menüpunkt für Steuerungsmodus hinzugefügt

Passende „id's“ und „strings“ werden eingetragen. So findet sich unter *@string/change\_move* der Text „Steuerungsart ändern“.

Hiermit hat man nun im Menü diesen Punkt, sollte dieser durch Klicken aufgerufen werden, so öffnet sich ein Untermenü mit den Unterpunkten „Einfach“ und „Permanent“.

Um auch die Logik dahinter zu realisieren wird in der BaseActivity Klasse strukturiert abgefragt, um welche Klasse es sich aktuell handelt:

```
if (currentActivityName.equals(getResources().getString(R.string.title_activity_activity_buttons))){...}
```

und ob der „Steuerungsart ändern“ Menüpunkt aufgerufen wurde:

```
if (item.getGroupId() == R.id.submenu_changeMove){...}
```

Sollte dies der Fall sein, so wird die Nutzereingabe über **switch** (item.getItemId()) eingelesen und zur jeweils zugehörigen Klasse gesprungen:

```
case R.id.permanent:
    startActivity(new Intent(this,Activity_Buttons_Permanent.class));
    return true;
case R.id.single:
    startActivity(new Intent(this, Activity_Buttons.class));
    return true;
default:
    return false;
```

Ändern des Android Action Bar Texts

Der Android Action Bar Text kann im *AndroidManifest.xml* verändert werden.

Da hier ohnehin schon zu jeder Activity ein Codeblock mit einem Label (default: Namen der Klasse) generiert wurde kann dieser mit gebräuchlicheren und verständlicheren Namen ersetzt werden.

```
<activity
    android:name="de.airswimmer.gui.Activity_Buttons"
    android:label="Einfache Tastensteuerung"
    android:screenOrientation="Landscape" >
</activity>
```

Hier wird unabhängig von definierten Strings gearbeitet und macht ein Modifizieren und Erweitern einfach.

## Ergebnis

Der Benutzer gelangt nun mit Hilfe eines „Steuerungsart ändern“ Menüpunkts zu der Auswahl, ob der aktuelle Modus in einer Permanenten oder Einfachen Steuerungsart bedient werden soll.

Optisch unterscheiden sie sich kaum, jedoch ist die Logik dahinter jeweils eine andere. Für eine bessere Orientierung sieht der Benutzer am oberen Bildschirmrand nun in welchem Modus und welcher Steuerungsart er sich befindet.

## **Startseite**

### **a) Abfrage Aux-Anschluss**

*Von Sabine Kressierer*

#### **Zielsetzung**

Die Ausrichtung der Oberfläche soll so erfolgen, dass sich der Aux-Anschluss immer an der oberen Seite des Tablets befindet. Somit kann der IrDroid Sender direkt auf den AirSwimmer gerichtet werden. Bei diesbezüglichen Recherchen wurden nur Geräte gefunden, deren Aux-Anschluss sich entweder an der oberen oder an der linken Seite befindet. Darauf basierend wird nur abgefragt, ob sich der Anschluss an der langen oder der kurzen Seite befindet und die Oberfläche dementsprechend ausgerichtet.

#### **Umsetzung**

Bei erstmaligem Öffnen der App wird zuerst ein Dialog geöffnet, in dem angegeben werden muss, wo sich der Aux-Anschluss befindet. Dies geschieht mit Hilfe eines AlertDialog, der in der onCreate()-Methode der FrontPage-Activity erzeugt und angezeigt wird. Die Reaktion auf die Auswahl eines Punktes erfolgt mittels onClick-Listener, in dessen onClick-Methode der Index des gewählten Punktes bekannt ist. Entsprechend dieser Auswahl wird die ActivityInfo-Konstante für das entsprechende Layout gespeichert.

```
if (item == 0) { //get value for requested screen orientation  
    layout = ActivityInfo.SCREEN_ORIENTATION_PORTRAIT;  
} else if (item == 1) {  
    layout = ActivityInfo.SCREEN_ORIENTATION_LANDSCAPE;  
}1
```

Da sich die Position des Aux-Anschlusses nicht verändert, wird dieser Wert dauerhaft gespeichert. Dazu werden shared-preferences verwendet, die es ermöglichen innerhalb der gesamten App auf den Wert zuzugreifen. Mit Preferences = getSharedPreferences("AirSwimmerPrefs", Context.MODE\_WORLD\_READABLE); werden die preferences in der Datei „AirSwimmerPrefs“ ermittelt. In diesen Preferences wird dann mit dem SharedPreferences.Editor der int-Wert zum Schlüssel „layout“ auf die oben ermittelte Konstante gesetzt. Am Ende wird in der onCreate-Methode der Startseite mittels setRequestedOrientation(layout); die Orientierung entsprechend verändert. Mit der selben Funktion wird in der onCreate-Methode der „ BaseActivity“ die Orientierung gesetzt. Um dort den in den Preferences gespeicherten Wert zu erhalten, werden wie oben die Preferences ermittelt und anschließend preferences.getInt("layout", -1) aufgerufen.

Bei erneutem Starten der App wird der Dialog nur noch gestartet, wenn in den Preferences kein Wert gespeichert ist.

---

<sup>1</sup> 0=Aux-Anschluss an kurzer Seite → Portrait / 1= Aux-Anschluss an langer Seite → Landscape

## b) Menü der Startseite

Von Sabine Kressierer

### Zweck

Die Ausrichtung der App soll nachträglich veränderbar sein.

### Umsetzung

In der Datei „start.xml“ wurde ein Menüpunkt „change\_layout\_orientation“ eingefügt, mit einem Untermenü für die beiden Orientierungsmöglichkeiten. Das Erzeugen des Menüs erfolgt in der FrontPage-Aktivity analog zum control-Menü. Bei Auswahl einer Orientierung im Untermenü, wird dieser Wert wie beim AlertDialog in den Preferences gespeichert und die Ausrichtung der FrontPage angepasst.

## 7.9 Infoseite

### Motivation

Von Marco Bengl

Da es bei Applikationen und auch anderen Programmen gängig ist eine Seite mit den mitwirkenden Personen einzurichten, darf diese bei der AirSwimmer-App natürlich auch nicht fehlen.

### Zielsetzung

Von Marco Bengl

Gedacht ist, dass die Infoseite über einen entsprechenden Menüpunkt im Menü der Startseite erreichbar ist.

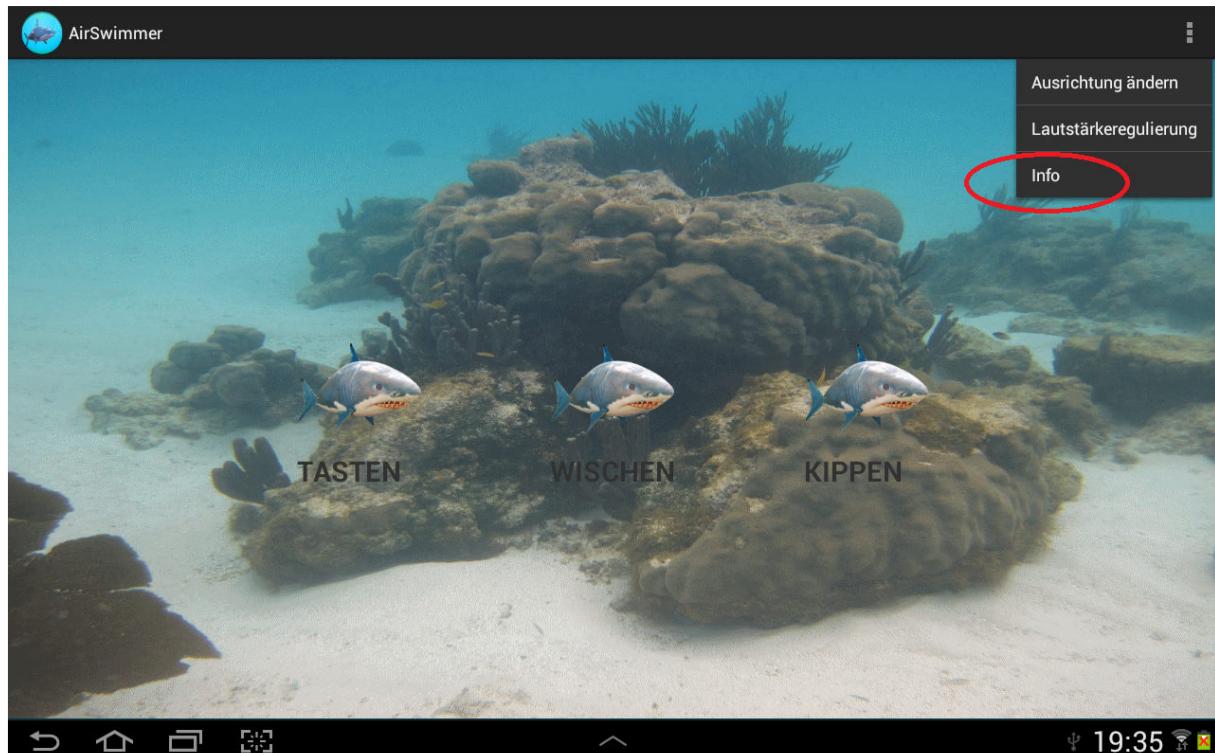


Abbildung 59

Startseite mit Menüpunkt „Info“

Zudem sollen auf der Infoseite alle Mitwirkenden Personen und die verwendete Hardware aufgeführt werden.

### Umsetzung

Von Marco Bengl

Zunächst wird der Menüpunkt „Info“ dem Menü auf der Startseite hinzugefügt. Dies wird erreicht indem man in der Klasse „FrontPage“ in der Funktion onOptionsItemSelected den Fall

```
} else if (item.getItemId() == R.id.info_page) {  
    startActivity(new Intent(this, info_page.class));  
}
```

hinzufügt.

Für die Erstellung der Infoseite wurde zunächst eine schlichte Oberfläche erstellt, in der alle benötigten Daten aufgelistet sind.



**Abbildung 60**  
Erste Version der Info Page

Diese Oberfläche soll aber noch kreativer gestaltet werden.  
Anschließend wird eine Klasse „InfoPage“ angelegt, welche die Oberfläche der Infoseite startet.

```

package de.airswimmer.gui;

import android.os.Bundle;

public class info_page extends BaseActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.info_page_layout);
    }
}

```

## Gestaltung der Info-Page

Von Melanie Knappe

Wie bereits oben erwähnt, soll die Info-Page noch schöner Gestaltet werden. Die Info-Page soll, wie ein Filmabspann aufgebaut werden. Damit ist gemeint, dass die Namen von unten nach oben bewegen werden.

Das Ganze kann mit einem Canvas realisiert werden, weil das bisherige Layout nicht erhalten bleiben muss, sondern ersetzt werden kann.

Zur Realisierung mit Canvas werden drei Klassen benötigt. Die Klasse info\_page wird im Menü aufgerufen. Die Klasse CreateView, die die Methoden für das neue Fenster beinhalten so wie die onDraw-Methode und die Klasse info\_page\_thread, die das Zeichnen auf die Canvas übernimmt. Diese drei Klassen werden im Weiteren kurz beschrieben.

- Info\_Page

Die Klasse Info\_page besitzt nur eine Methode. In der onCreate-Methode wird bei setContentView ein neues Objekt der CreateView-Klasse erstellt und der Context wird mit übergeben.

```

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(new CreateView(this));
}

```

- CreateView

In dem Konstruktor der CreateView-Klasse werden drei Methoden definiert. Die Methoden beschreiben, was passiert, wenn das Objekt erzeugt, geändert oder beendet wird.

```

public CreateView(Context context) {
    super(context);
    theGameLoopThread = new info_page_thread(this);
    surfaceHolder = getHolder();
    surfaceHolder.addCallback(new SurfaceHolder.Callback() {

```

//Die Methode surfaceDestroyed wird aufgerufen, wenn das Objekt beendet wird.

```

        public void surfaceDestroyed(SurfaceHolder holder) {
            boolean retry = true;
            theGameLoopThread.setRunning(false);

```

```

        while (retry) {
            try {
                theGameLoopThread.join();
                retry = false;
            } catch (InterruptedException e) {
            }
        }
    }
}

```

//Die Methode surfaceCreated, wird aufgerufen, wenn das Objekt erzeugt wird.

```

public void surfaceCreated(SurfaceHolder holder) {
    theGameLoopThread.setRunning(true);
    theGameLoopThread.start();

}

```

//Die Methode surfaceChanged wird aufgerufen, wenn das Objekt geändert wird.

```

public void surfaceChanged(SurfaceHolder holder, int
                           format, int width, int height) {
    // TODO Auto-generated method stub

}
});;
bmp = BitmapFactory.decodeResource(getResources(), 0x7f02000f);

}

```

Bei dieser Applikation wird als bmp das folgende Bild verwendet.

#### AirSwimmer Applikation

**Entwickler:**  
 Bengl Marco  
 DeNuzzo Giuseppe  
 Gerken Andreas  
 Hafner Anja  
 Knappe Melanie  
 Kocak Belguezar  
 Kohlbrenner Felix  
 Kressierer Sabine  
 Pilot Caroline  
 Trojosky Patrick  
 Weber Andreas  
 Yücel Ridvan

**Hardware:**  
 IrDroid Infrarotsender

**Abbildung 61**  
 Bild für animierte Info Page

Das Bild beinhaltet alle Informationen, die wir mitteilen wollen. Dieses Bild wird nach und nach von unten nach oben bewegt. Das geschieht mit Hilfe der onDraw-Methode. Diese Methode wird vom Thread immer und immer wieder aufgerufen. Bei jedem Aufruf, ändert sich die y-Position des Bildes um 2 nach oben. Daher wirkt es so, als ob sich das Bild nach oben bewege.

```

@Override
protected void onDraw(Canvas canvas) {
    canvas.drawBitmap(BitmapFactory.decodeResource(getResources(), 0x7f020006), 0, 0, null);

    if (y >= getHeight() - bmp.getHeight()) {
        y = y - 2;
    }else if(y <= getHeight() - bmp.getHeight()) {
        y = y + 2;
    }
    canvas.drawBitmap(bmp, x, y, null);
}

```

#### - Info\_page\_thread

Die letzte Klasse die für die Info-Page gebraucht wird, ist die Info\_page\_thread Klasse. In dieser Klasse wird ein Thread gestartet. Solange der Thread läuft versucht der Thread auf den Canvas zu zeichnen, indem er die onDraw-Methode aufruft.

```

private CreateView theView;
private boolean isRunning = false;

public info_page_thread(CreateView theView) {
    this.theView = theView;
}

public void setRunning(boolean run) {
    isRunning = run;
}

@Override
public void run() {
    while (isRunning) {
        Canvas theCanvas = null;

        try {
            theCanvas = theView.getHolder().lockCanvas();
            synchronized (theView.getHolder()) {
                theView.onDraw(theCanvas);
            }
        } finally {
            if (theCanvas != null) {

theView.getHolder().unlockCanvasAndPost(theCanvas);
        }
    }
}

```

Nach der Realisierung der drei Klassen, funktioniert die Infoseite wie gewünscht. Die Informationen zur unserer Applikation werden wie bei einem Filmabspann eingeblendet.

Sollten die Information nachträglich geändert werden müssen, ist das leicht zu machen. Es muss nur ein neues Bild erstellt werden und durch das jetzige Bild ausgetauscht werden. Wenn das neue Bild den gleichen Namen trägt, wie das alte Bild muss der Sourcecode nicht verändert werden.

## 8 Verbinden von Logik und Oberfläche

Von Felix Kohlbrenner

Da die Logik zum Senden der Signale an den AirSwimmer mittlerweile funktioniert, besteht der nächste Schritt darin, diese Logik in die bereits bestehende Oberfläche zu integrieren. Das Oberflächenteam hat hierzu drei verschiedene Funktionen vorbereitet:

- Steuern über Buttondruck
- Steuern durch Wischen in die gewünschte Flugrichtung
- Steuern durch Kippen des Tablets

Um die Logik des Sendens in die Oberfläche zu integrieren, muss zunächst die .so Bibliothek in den „libs“ Ordner des Projekts kopiert werden. Außerdem wird das Package mit der „Lirc.java“ Klasse benötigt, welche aus der Irdroid App übernommen werden kann.

Der nächste Schritt besteht darin eine Klasse mit dem Namen „Movement“ zu erstellen, in welche die Logik des Sendens kopiert wird. Diese kann aus dem Prototypen der Anwendung übernommen werden. Allerdings muss eine neue Methode erstellt werden, um die Aufrufe der Climb, Dive, Left und Right Befehle zu erleichtern. Diese werden bisher noch in den Listenern der Buttons des Prototyps aufgerufen.

Es werden daher die Methoden „Climbing()“, „Diving()“, „MoveLeft()“ und „MoveRight()“, sowie ein Konstruktor „public Movement(BaseActivity caller, AudioManager audio, SharedPreferences pref)“ erstellt.

Die Oberflächen Aktivität „BaseActivity“ wird um eine Instanz der Movement Klasse („action“) erweitert und es wird ein Objekt dieser Instanz in der onCreate Methode erstellt, sodass die anderen Activities auf die Funktionen der Movement Klasse zugreifen und die Steuerung des AirSwimmers möglichst einfach realisiert werden kann.

### 8.1 Buttons

Von Felix Kohlbrenner

Die Realisierung der Steuerung mittels Buttons, ist nachdem die Movement Klasse erstellt ist und auch kleinere Probleme, die beim Importieren der Logik auftreten, behoben sind, sehr einfach. Es muss lediglich der Aufruf der abstrakten Methoden dive(), climb(), moveLeft() und moveRight() an den richtigen Stellen im onTouch Listener der Elternklasse „BaseButtons“ erfolgen.

In der Kindklasse „Activity\_Buttons“ muss der Aufruf der entsprechenden Methode der Movement Klasse in den abstrakten Methoden ergänzt werden:

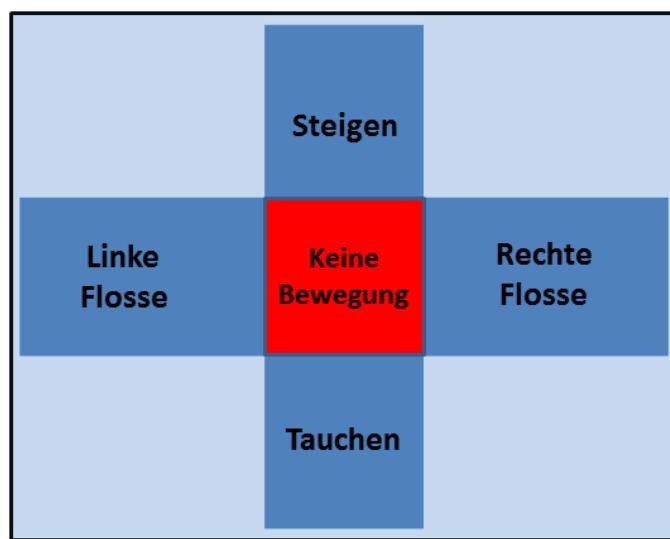
z.B.:

```
@Override  
public void dive() {  
    action.diving()  
}
```

## 8.2 Slide

Von Patrick Trojosky

Nachdem die Buttons Oberfläche von Felix implementiert sind, ist es jetzt meine Aufgabe die GUI für die Slide Funktion mit der Movement Klasse zu verbinden. Der Benutzer soll den Fisch durch Wischen auf dem Tablet bewegen können. Startposition ist die Mitte des Bildschirms. Wird das Bild des Fisches durch den Nutzer nach oben bewegt, soll der Fisch steigen, bei entgegengesetzter Bewegung sinken. Analog soll die Funktion beim Wischen nach Links und Rechts funktionieren. Damit der Fisch auch wieder zum Stillstand kommen kann, definiere ich eine Region, in der Form eines Vierecks, in der keine Bewegung ausgeführt wird. Die Aufteilung des Bildschirms ist in der folgenden Grafik zu erkennen.



Um die Funktion schließlich zu integrieren, muss ich mich nur der Methoden der Movement Klasse bedienen. Diese kann über die Objektinstanz „action“, die eine Referenz in der Grundklasse der App hat, aufgerufen werden. Daher erweitere ich den Listener der Slide Klasse um die nötigen Kommandos. Damit ich feststellen kann, welches Kommando gesendet werden soll, muss ich aus den aktuellen Koordinaten des Fisches und den Koordinaten des vorherigen Listener Aufrufs, die Bewegungsrichtung erkennen. Der Fisch soll nur eine Bewegung ausführen, wenn auch eine gewisse Strecke „überstrichen“ wird. Ansonsten wäre die Funktion sehr oft aufgerufen worden, da der Listener, die kleinsten Bewegungen des Fingers auf dem Tablet registriert. In diesem Fall wäre das Senden so oft aufgerufen worden, dass die komplette App dadurch gestört wird und nicht mehr benutzbar ist.

## 8.3 Kippen

Von Belgüzar Kocak

Nun muss noch die Oberfläche Kippen mit der Logik zum Bewegen des AirSwimmer hinterlegt werden. Dabei zeigen die positive x-Achse nach rechts und die positive y-Achse nach unten. Das heißt oben links befindet sich der Nullpunkt der Achsen somit

kann der Bildschirm in Hälften geteilt und die Funktion zum Senden aufgerufen werden.

In der Methode move werden die einzelnen Bewegungsarten realisiert.

```
public void move(float xAxis, float yAxis)
```

Falls sich die Grafik in der oberen Hälfte befindet soll sich der AirSwimmer nach oben bewegen. Um die Methode `action.climbing()` nur einmal aufzurufen und somit den Thread nicht immer wieder zu starten, wird eine Variable `bool isFirstTiltUp` hinzugefügt.

Somit werden die einzelnen Funktionen implementiert.

```
// Up
if (yAxis > 150 - delta) {
    if (!isFirstTiltUp) {
        action.climbing();
        isFirstTiltUp = true;
    }
} else {
    isFirstTiltUp = false;
}
```

## 8.4 Permanente Bewegung

*Von Caroline Pilot*

### Zielsetzung

Der Permanent-Modus soll dazu dienen, dass der AirSwimmer in einer kontinuierlichen Bewegung geradeaus schwimmt. Somit soll sich der Benutzer, nach dem Betätigen des Start/Stopp Buttons (siehe Abbildung 62Abbildung 62

**Oberfläche des Permanent-Modus, in der Mitte ist der Start/Stopp Button zu sehen** ), nicht mehr um die Fortbewegung im Allgemeinen kümmern müssen, sondern lediglich um die speziellen Richtungswechsel.

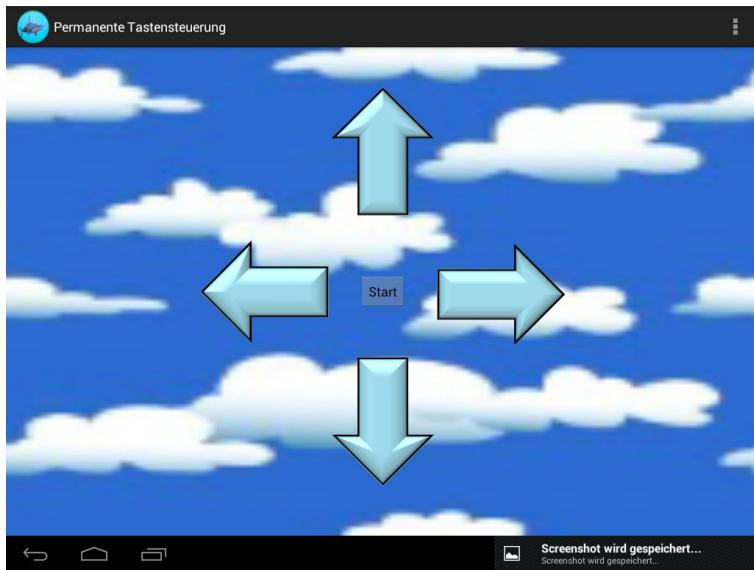


Abbildung 62

Oberfläche des Permanent-Modus, in der Mitte ist der Start/Stopp Button zu sehen

### Vorüberlegung

Im Allgemeinen muss zur permanenten Bewegung folgende Punkte in dieser Reihenfolge bearbeitet werden:

1. Eine Linksbewegung der Schwanzflosse
2. Eine kurze Wartezeit von etwa 5sec , damit die Schwanzflosse wieder zur Mitte zurückkehren kann
3. Eine Rechtsbewegung der Schwanzflosse

Wiederholen der Punkte 1 – 3, solange bis der Button erneut gedrückt wurde.

Um dies zu realisieren gibt es im Grunde zwei Möglichkeiten:

1. Eine einfache do-while Abfrage
2. Ein Runnable-Aufruf

### Vorgehensweise

#### 1. Realisierung über eine einfache do-while Abfrage

Da bereits eine Methode implementiert ist, die lediglich den Schriftzug des Buttons beim Drücken ändert, bietet es sich an, die hier markierten „// TODO“ Stellen zu füllen.

```

17 +   public void changeMoveState(View view){
18 +     Button button_start = (Button) view;
19 +     if(forwardMovement){
20 +       forwardMovement=false;
21 +       button_start.setText("Start");
22 +       //TODO stop forward movement
23 +
24 +     }else{
25 +       forwardMovement=true;
26 +       button_start.setText("Stop");
27 +       //TODO start forward movement
28 +     }
29 +   }
30 +

```

Hierfür wird eine neue Methode „swim()“ geschrieben, die genau die oben genannten Punkte abarbeitet:

```

public boolean swim() {
    moveLeft();
    action.finishMovingLeft();
    SystemClock.sleep(waiting_time);
    moveRight();
    action.finishMovingRight();

    return false;
}

```

Die verwendeten Befehle (`moveLeft();action.finishMovingLeft(); ...`) sind Funktionen der Klasse, die das Senden realisiert.

Mittels einer do-while Schleife soll nun die `swim()` Methode aufgerufen werden, solange (while) der `Button_Start` Button wieder gedrückt wird.

## **2. Realisierung über das Runnable-Interface**

Eine weitere Möglichkeit eine permanente Bewegung zu realisieren, besteht darin, ein Runnable-Interface zu starten, sobald der Button gedrückt wird.

Hierin soll die Selbe Reihenfolge der Punkte abgearbeitet werden, wie oben beschrieben. Somit läuft das Interface parallel zur restlichen Bewegung bzw. zu den anderen Aktionen der Oberfläche und beendet sich, sobald der Button erneut gedrückt wurde.

Das Verhalten eines Runnable-Interfaces, lässt sich durch das Überschreiben der „`run()`“-Methode modifizieren:

So wird auch hier eine do-while Schleife über die Signale für die Links- und Rechtsbewegung gelegt. Das Warten wird mittels einer try-catch Abfrage behandelt

und lässt das Runnable-Interface für 5 s schlafen.

### ***Ergebnis***

Das Ergebnis beider Methoden ist leider ganz und gar nicht zufrieden stellend.

#### **1. Realisierung über eine einfache do-while Abfrage**

Das Testen ergibt leider, dass die Idee, einer simplen do-while Abfrage, in einer Endlosschleife endet, da während dieser Ausführung der Rest der Oberfläche nicht mehr läuft und somit kann ein Drücken des Buttons nicht mehr wahrgenommen wird.

#### **2. Realisierung über das Runnable Interface**

Auch hier wird leider kein befriedigendes Ergebnis erreicht.

Nach vielen Modifikationen werden nie alle Fehler beseitigt und ein fehlerfreies Laufen möglich gemacht.

Zum einen ist auch hierbei wieder die restliche Oberfläche eingefroren, was dazu führt, dass das Interface nie beendet wird.

Zum anderen wird, sobald der Runnable-Aufruf in den Code eingefügt wurde, nicht mal mehr der Schriftzug des Buttons verändert, obwohl ein Runnable-Aufruf noch gar nicht stattfand.

Außerdem erlangt man undefiniertes bzw. nicht-deterministisches Verhalten, was das Senden der Befehle angeht. Sie werden, teils in großen, teils in sehr kurzen Abständen aufgerufen.

Eine weitere Realisierungsmöglichkeit für den permanenten Modus ist leider aus Zeitgründen nicht mehr möglich gewesen.

### ***Ausblick***

Um diese Funktion zu implementieren wäre ein weiterer Versuch gewesen, einen „Prioritäten-Handler“ zu implementieren, der die Oberfläche, parallel zum Permanenten Geradeaus schwimmen, parallel zum Senden der Signale, verwaltet.

## **8.5 Abschließende Optimierungen**

*Von Patrick Trojosky*

Doch mit verminderter Zahl der Aufrufe, ist ein „Ruckeln“ bei der Oberfläche zu erkennen und die App läuft nicht flüssig auf dem Tablet. Dies ist zu Beginn nicht zu erklären, da Runnables eingesetzt werden und somit die Nebenläufigkeit garantiert sein sollte. Der schlussendliche Fehler dabei, hängt aber an einem kleinen Detail. Im Android Programmierung wird mit sogenannten Handlern gearbeitet. Sie sind dafür zuständig die Kommunikation zwischen den diversen Activities und die Handhabung von Threads abzuwickeln. In der App besitzt die Basis Activity einen Handler, dieser verwaltet auch die Aktivierungen der Runnables. Das bedeutet, dass diese über den Handler, mit dem Befehl postAtTime(Name der Runnable, Zeit in Millisekunden) gestartet werden. Das Problem dabei ist jetzt allerdings, dass der Handler die Runnables in seinem Vater Thread startet, also im gleichen Thread in dem

die GUI läuft. Deshalb kommt es zu Performanz Einbrüchen, wenn wiederholt das Senden Kommando ausgeführt wird. Um dieses Problem zu umgehen, muss die App erneut umstrukturiert werden. Anstatt die Funktion in Runnables zu implementieren, werden vier Threads erstellt, die das jeweilige Kommando senden können. Drückt der Benutzer jetzt eine Taste, oder wischt über den Bildschirm, wird ein eigenständiger Thread gestartet, der unabhängig von der grafischen Oberfläche die Signale an den AirSwimmer schicken kann. Durch diese Änderung wird die Steuerung Oberfläche deutlich verbessert und „ruckelt“ nicht mehr.

Nachdem die funktionalen Teile der App nun vollständig implementiert sind, kann an den Feinschliff des Codes gedacht werden. Während den Arbeiten an der App treten teilweise Warnungen durch die Android Lint Analyse auf. Diese werden in den meisten Fällen durch den Gebrauch von veralteten Funktionen („deprecated“) hervorgerufen. Da es mittlerweile siebzehn verschiedenen Android APIs gibt, die ständig aktualisiert werden, kommt es dazu, dass das Benutzen mancher Funktionen nicht mehr empfohlen wird. Dies kann unterschiedliche Gründe haben. Oft birgt der Gebrauch alter Funktionen Sicherheitsrisiken oder die Funktionen sind nicht mehr kompatibel mit älteren Geräten. Damit der Code keine Warnungen mehr enthält, wird gezielt nach den veralteten Funktionen gesucht und diese durch das empfohlene Gegenstück aus der aktuellen Google API ersetzt.

# 9 Graphische Gestaltung der Oberfläche

## 9.1 Konzept

Von Caroline Pilot

### Zielsetzung

Ziel eines Prototyps für das Design der Oberfläche<sup>1</sup> ist es, der Gruppe einen ersten visuellen Eindruck für die App zu geben. So bekommt das gemeinsame Ziel des Teams ein Gesicht, auf das motivierter hingearbeitet werden kann.

Außerdem bieten Modelle eine bessere Diskussionsgrundlage für Wünsche, Ideen und Verbesserungsvorschläge. So kann die Oberflächenentwicklung mit konkreten Designs ausgeführt werden.

### Vorüberlegung

Zunächst muss ein Graphikprogramm gefunden werden, das den Anforderungen entspricht.

Hierbei ist folgendes zu beachten:

- Vielerlei Möglichkeiten Bilder zu bearbeiten (Kontraste, Layer, Transparenz, ...)
- Erstellen von Graphiken in gängigen Bildformaten
- Einfache Bedienung, damit keine lange Einarbeitungszeit nötig ist
- Kostengünstig, am besten ein Freeware Download Programm

Des Weiteren muss der Kreativität bei folgenden Punkten freien Lauf gelassen werden:

- Wie soll der Hintergrund aussehen?
- Wie soll das Objekt dargestellt werden?
- Welche Besonderheiten benötigen die einzelnen Benutzermodi?

### Vorgehensweise

Unter [http://www.chip.de/downloads/PhotoFiltre\\_13012070.html](http://www.chip.de/downloads/PhotoFiltre_13012070.html) ist das Freeware Graphikprogramm als Download verfügbar. Es entspricht unseren Anforderungen und enthält zudem sehr gute Nutzerrezensionen.

Aus dem Brainstorming zum Thema Oberflächendesign ergibt sich folgendes:

- Hintergrund:
  - o Himmel
  - o Unterwasserwelt
  - o Stadt/Skyline
- Objekt
  - o Bild des AirSwimmers
  - o Bild eines Fisches
  - o Bild eines Köders, dem der echte AirSwimmer „hinterher schwimmt“

---

<sup>1</sup> Hiermit sind im Kapitel 5.1 „Konzeptentwicklung für das Oberflächendesign“ die unterschiedlichen Benutzermodi zum Steuern des AirSwimmers gemeint, nicht die Startseite der App

- Besonderheiten einzelner Modi
  - o Tastensteuerung
    - Tasten in gewöhnlicher Buttonform
    - Tasten in Form eines Fisches
    - Kontrastreiche Farben <-> Gedeckte Farben
  - o Kippsteuerung
    - Angabe von Himmelsrichtungen (north, south, west, east)
    - Angabe von Bewegungsrichtung (up, down, left, right)
    - Keine Angabe
  - o Wischsteuerung
    - Angabe von Himmelsrichtungen (north, south, west, east)
    - Angabe von Bewegungsrichtung (up, down, left, right)
    - Keine Angabe

### **Ergebnis**

Bilder zum Verwirklichen dieser Ideen und zum Erstellen von Designprototypen lassen sich im Internet finden oder mit dem Graphikprogramm erstellen.

In verschiedenen Kombinationen, umgesetzt mit dem Photofiltre7 ergeben sich folgende Entwürfe:

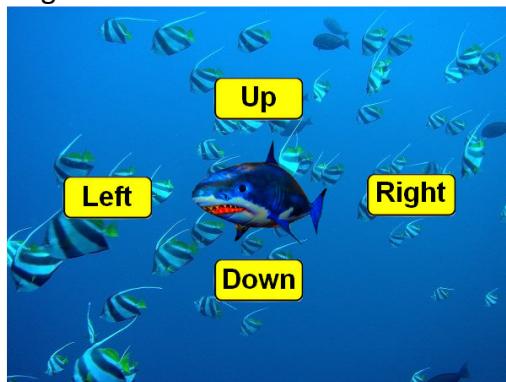


Abbildung 63

Tastensteuerung;  
Unterwasserwelt,  
Kontrastreiche Buttons



Abbildung 64

Tastensteuerung;  
Himmel, Gedeckte Buttons



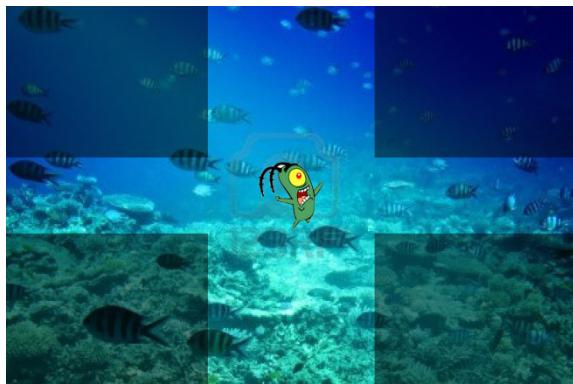
Abbildung 65

Kippsteuerung;  
Unterwasserwelt, Angabe von  
Himmelsrichtungen



Abbildung 66

Kippsteuerung;  
Skyline, ohne Angaben



**Abbildung 67**  
Wischsteuerung  
Benutzergrenzen, für definierte Bewegungen, Köder als Objekt



**Abbildung 68**  
Wischsteuerung  
Benutzergrenzen, für definierte Bewegungen

Die Gruppendiskussion bringt hervor, dass es in der App einen Punkt für einen Wechsel des Hintergrunds zwischen Himmel und Unterwasserwelt geben wird, je nach Wunsch des Benutzers.

Des Weiteren werden nicht die Himmelsrichtungen im Kippmodus angegeben, denn dies könnte zu Verwirrungen führen, stattdessen werden keine Angaben platziert, ebenso im Wischmodus.

Das Objekt in der Mitte soll ein Fisch sein.

### **Ausblick**

Da es sich bei den verwendeten Bildern um Eigentum Dritter handelt, könnte man sich durch dessen Verwendung rechtliche Probleme einhandeln. Aus diesem Grund ist es wünschenswert, eigene Graphiken, vielleicht auch dynamische, zu erstellen oder die Urheber schriftlich um eine Genehmigung zu bitten.

## **9.2 Launcher-Icon**

*von Caroline Pilot*

### **Zielsetzung**

Um einen einheitlichen Auftritt der App zu ermöglichen, wird ein Launcher-Icon benötigt. Dieser repräsentiert die App auf dem Bildschirm und ist auch bei der Ausführung am Rand zu sehen.

### **Vorüberlegung**

Da bereits das Graphikprogramm PhotoFiltre vorhanden ist, kann dieses verwendet werden, um den Launcher-Icon zu erstellen.

Des Weiteren muss der Kreativität bei folgenden Punkten freien Lauf gelassen werden:

Welche Form soll der Icon bekommen?

Welches Objekt soll darauf zu sehen sein?

Welche Farben könnten passen?

## **Vorgehensweise**

Aus dem Brainstorming ergeben sich folgende Punkte:

### Form

Rund, da es einer Wasserblase entspricht, in dem sich der AirSwimmer fortbewegt

### Objekt

Da es eine App für den AirSwimmer sein soll, liegt es nahe, dass auch auf dem Icon der Fisch abgebildet ist. Somit wäre es eindeutig auf dem Desktop zu identifizieren

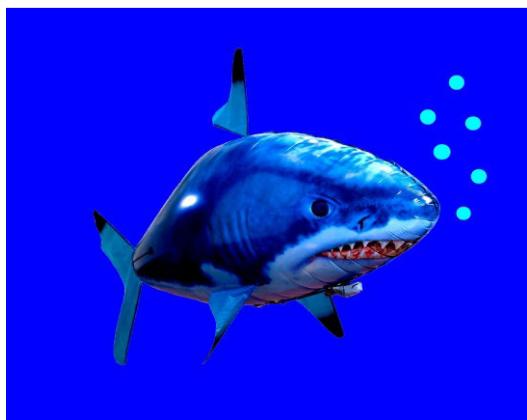
### Farben

Die Farben müssen herausstechen, sodass die App nicht auf dem Desktop untergeht und man sie nicht sehen kann (Beispiel schwarz), dennoch müsste die Farbe zum Thema Wasser/Himmel passen

## **Ergebnis**

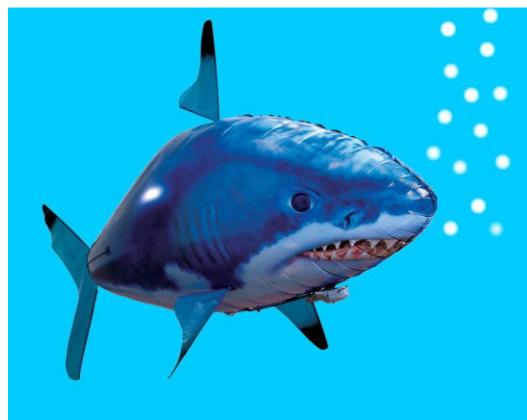
Das Bild des AirSwimmers lässt sich im Internet finden.

In verschiedenen Kombinationen mit Farben und Hintergründen, umgesetzt mit dem Photofiltre7 ergeben sich folgende Entwürfe:



**Abbildung 69**

Launcher-Icon mit dunklem Hintergrund



**Abbildung 70**

Launcher-Icon mit hellem Hintergrund

Nach dem Testen auf dem Tablet wird festgestellt, dass der dunkle Hintergrund bei einem schwarzen Desktop untergeht, daher fiel die Entscheidung auf einen runden Launcher-Icon mit hellem Hintergrund (Abbildung 71).

## **9.3 Animierter Hintergrund**

*Melanie Knappe*

Die Startseite der Applikation soll einen animierten Hintergrund bekommen. Das Hintergrundbild ist ein Unterwasserbild. Auf diesem Hintergrund sollen nun Fische herum schwimmen. Das Ganze soll ausschauen wie im echten Wasser.

Voraussetzung für den animierten Hintergrund war, dass die bereits programmierten Views erhalten bleiben müssen. Mehrere Ansätze wurden überlegt:

## Canvas

Der Erste Ansatz war, die Realisierung mit Canvas. Canvas kann man sich vorstellen, wie Tafeln, auf denen etwas drauf gezeichnet wird.

Um den animierten Hintergrund mit Hilfe von Canvas zu realisieren, werden 3 Klassen benötigt.

### 1. MainActivity

Diese Klasse beinhaltet die onCreate Methode. Allerdings wird bei setContentView kein Layout aufgerufen sondern ein neues Objekt der Klasse AnimationView aufgerufen, die als Parameter den Context mit bekommt.

```
public class MainActivity extends Activity {  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(new AnimationView(this));  
    }  
}
```

### 2. AnimationView

In dieser Klasse wird der Konstruktor mit Zusatzmethoden ausgestattet und die onDrawmethode erstellt.

#### Konstruktor:

Die Zusatzmethoden vom Konstruktor sind „surfaceDestroyed“, „surfaceCreated“ und „surfaceChanged“. Diese drei Methoden sind wichtig und geben an, was passiert, wenn die Oberfläche „zerstört“ wird, die Oberfläche erstellt wird und wenn die Oberfläche gewechselt wird.

```
public AnimationView(Context context){  
    super(context);  
    myAnimationsThread = new AnimationsThread(this);  
    mySurfaceHolder = getHolder();  
    mySurfaceHolder.addCallback(new SurfaceHolder.Callback(){  
  
        @Override  
        public void surfaceDestroyed(SurfaceHolder holder) {  
            boolean retry = true;  
            myAnimationsThread.setRunning(false);  
            while(retry){  
                try{  
                    myAnimationsThread.join();  
                    retry = false;  
                }catch(InterruptedException e){  
  
                }  
            }  
        }  
    }  
  
    @Override  
    public void surfaceCreated(SurfaceHolder holder) {  
        myAnimationsThread.setRunning(true);  
        myAnimationsThread.start();  
    }  
}
```

```

        @Override
        public void surfaceChanged(SurfaceHolder holder, int
            format, int width, int height) {
            // TODO Auto-generated method stub
        }
    });

    myBmp = BitmapFactory.decodeResource(getResources(),
        R.drawable.ic_launcher);
}

```

#### OnDrawmethode:

Diese Methode wird verwendet, um auf den Canvas zu „zeichnen“. Diese Funktion erstellt in unserem Fall die Fische, die später auf der Oberfläche schwimmen.

```

public void onDraw(Canvas canvas){
    canvas.drawBitmap(BitmapFactory.decodeResource(getResources(),
        R.drawable.meer_hintergrund), 0, 0, null);

    if(createSprite == true){
        initialSprites();
    }

    for(Fische sprite : spriteList){
        sprite.onDraw(canvas);
    }
}

```

### 3. AnimationThread

Der Thread wird gebraucht, weil die Fische nicht statisch sind, sondern sich bewegen. Die Bewegung wird dargestellt durch ein „verschieben“ des Bildes um eine gewisse Anzahl von Pixeln. Passiert das in kurzen Zeitabschnitten, dann sieht es so aus, als ob sich der Fisch bewegen würde.

Solange der Thread läuft versucht der Thread auf den Canvas mit der onDrawmethode zu „zeichnen“.

```

public class AnimationsThread extends Thread {

    static final long FPS = 20;
    private AnimationView myView;
    private boolean isRunning=false;

    public AnimationsThread(AnimationView myView){
        this.myView = myView;
    }

    public void setRunning(boolean run){
        isRunning = run;
    }

    @Override
    public void run(){
        long TPS = 1000/FPS;
        long startTime, sleepTime;

```

```

while(isRunning){
    Canvas myCanvas = null;
    startTime = System.currentTimeMillis();

    try{
        myCanvas =
            myView.getHolder().lockCanvas();
            synchronized(myView.getHolder()){
                myView.onDraw(myCanvas);
            }
    }finally{
        if(myCanvas != null){
            myView.getHolder().unlockCanvasAndPost
                (myCanvas);
        }
    }

    sleepTime = TPS-(System.currentTimeMillis()-
startTime);
    try{
        if(sleepTime > 0){
            sleep(sleepTime);
        }else{
            sleep(10);
        }
    }catch(Exception e){

    }
}
}
}

```

Die Implementierung wurde erfolgreich abgeschlossen, aber erst im Nachhinein stellte sich heraus, dass ein Canvas nicht benutzt werden kann, wenn ein Layout verwendet wird. Dies war bei uns der Fall. Es existierte bereits ein fertiges Layout und weil dieses erhalten werden sollte konnte dieser Ansatz nicht umgesetzt werden.

Aus diesem Grund wurden weitere gesucht um einen neuen Ansatz zu finden.

### ***JDroidLib***

Nach weiterem Suchen wurde die JDroidLib gefunden. Eine Bibliothek, welche sich auf Android Spiele spezialisiert hat. Nach einigen Versuchen mit der Bibliothek stellte sich heraus, dass diese ungeeignet ist. Die Bibliothek ließ sich nicht in unsere bestehende Applikation einbinden. Diese Bibliothek wurde entwickelt, damit Anfänger schnell sich ihre Spiele zusammen „puzzeln“ können. Diese Bibliothek ließ sich nicht mit unseren bestehenden Layouts zusammen fügen.

### ***Thread***

Dieser Ansatz wurde nur theoretisch ausgearbeitet und aus Zeitgründen nicht getestet. Die Idee war, dass ein eingebildetes Bild per Thread immer weiter bewegt wird. (Ähnlich wie bei den Canvas).

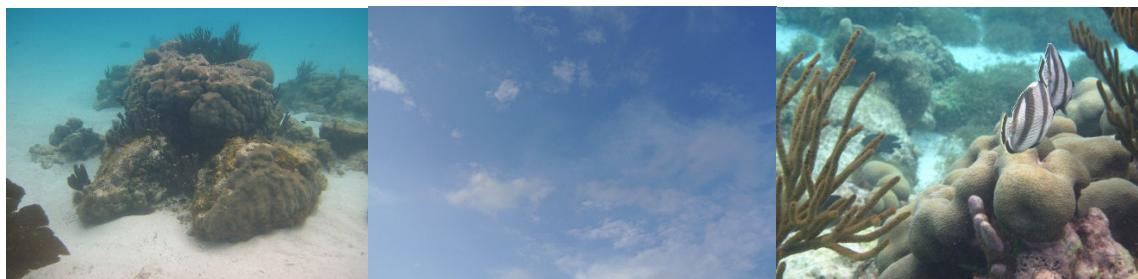
## 9.4 Hintergrundbilder

Von Melanie Knappe

Für den Hintergrund wurden privat Fotos verwendet, um keine Probleme mit Rechten zu bekommen.

Die Auflösungen der Bilder wurden auf 1280\*800 Pixel runter skaliert, damit die Größe der Bilder kleiner wird. Das umwandeln wurde mit CorelDraw durchgeführt. Es wurde festgestellt, das Bilder die im JPEG-Format abgespeichert sind eine bessere Qualität haben, als bei anderen Formaten. Daher wurden alle Bilder, bei denen es möglich war, in ein JPEG-Format umgewandelt.

In der Applikation wurden drei verschiedene Hintergrundbilder gewählt.



**Abbildung 71**

Neue Hintergrundbilder der App

Das erste Bild dient als Hintergrundbild für die Startseite und die Info Page. Die anderen beiden Bilder dienen als Hintergrund in den anderen Ansichten.

## 9.5 Graphiken

### 9.5.1 Startseite (Buttons)

Von Belgüzar Kocak

#### Zielsetzung

Die Aufgabe bestand darin, die Startseite zu verschönern und die Standard Buttons auszutauschen, da diese wie Balken über dem Bildschirm lagen. Aufgrund dessen waren größere Buttons und auch für die Startseite passende Buttons notwendig. Ziel war es Hai-Buttons für die Startseite zu erstellen.

#### Vorgehen

Bevor die Buttons ausgetauscht werden konnten, mussten sie in der XML-Datei in ImageButtons umgewandelt werden. In diese ImageButtons wurde eine Hai Grafik eingefügt und der Hintergrund der Buttons ausgeblendet damit diese die Form eines Haies haben.

In der XML-Datei wurden die normalen Buttons rausgelöscht und neue ImageButtons in die Oberfläche gezogen und die entsprechenden Images an die Buttons zugewiesen. Vorab wurde die Grafik im res Ordner unter drawable-hdpi als air\_swimmer\_shark gespeichert.

Diese wurden dann mittels folgender Anweisung den einzelnen Buttons zugewiesen:

```
android:src="@drawable/air_swimmers_shark"
```

Der Hintergrund wurde transparent gestaltet um einen Hai-Förmigen Button zu erstellen

```
    android:background="#00000000"

<ImageButton
    android:id="@+id/Button_Slide"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_centerHorizontal="true"
    android:layout_centerVertical="true"
    android:background="#00000000"
    android:onClick="onButtonClick"
    android:scaleType="center"
    android:src="@drawable/air_swimmers_shark" />
```

Die Beschriftung der einzelnen Buttons wurde anhand einer TextView dargestellt. Dabei wurde die Schriftgröße, Schriftart und der Text der View implementiert.

```
<TextView
    android:id="@+id/textView2"
    android:layout_width="1189dp"
    android:layout_height="wrap_content"
    android:layout_alignBottom="@+id/textView3"
    android:layout_alignParentRight="true"
    android:layout_below="@+id/Button_Slide"
    android:layout_toRightOf="@+id/Button_Button"
    android:text="@string/button_wischen"
    android:textSize="30sp"
    android:textStyle="bold"
    android:typeface="sans" />
```

Auch wurde der Hintergrund der Startseite verändert.

```
    android:background="@drawable/frontpage_background"
```

Am Ende hatten die Buttons der Startseite folgendes Aussehen.



Abbildung 72

Startseite

## **9.5.2 Steuerungsseiten (Buttons und Hai)**

Von Belgüzar Kocak

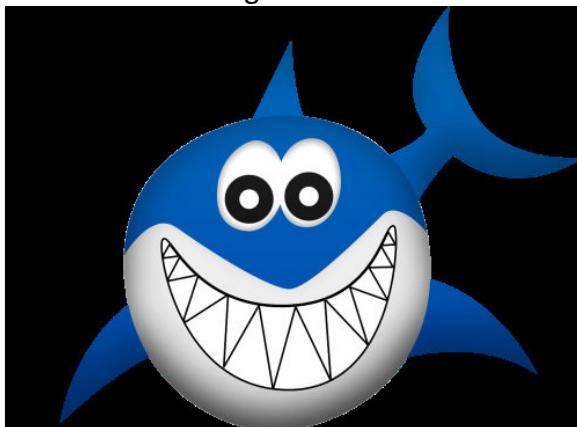
### **Zielsetzung**

Ziel war es für die Oberfläche Kippen und Wischen eine Grafik von einem Hai zu erstellen und auch Buttons für die Buttons Oberfläche zu zeichnen. Genutzt wurden hier folgende Tools: Microsoft Word, Paint.Net und Photoshop.

### **Vorgehen und Erstellen**

Verlangt wurde, dass der Hai direkt den Benutzer der Oberfläche anschaut. Aufgrund dessen wurden vorerst Zeichnungen per Hand erstellt genutzt wurden dabei ein einfaches Papier so wie Bleistift. Somit war der erste Schritt der Zeichnung getan. Jetzt war es wichtig die Zeichnung vom Papier auf den Bildschirm als Grafik zu verwirklichen. Dies gelang mit dem Photoshop Programm.

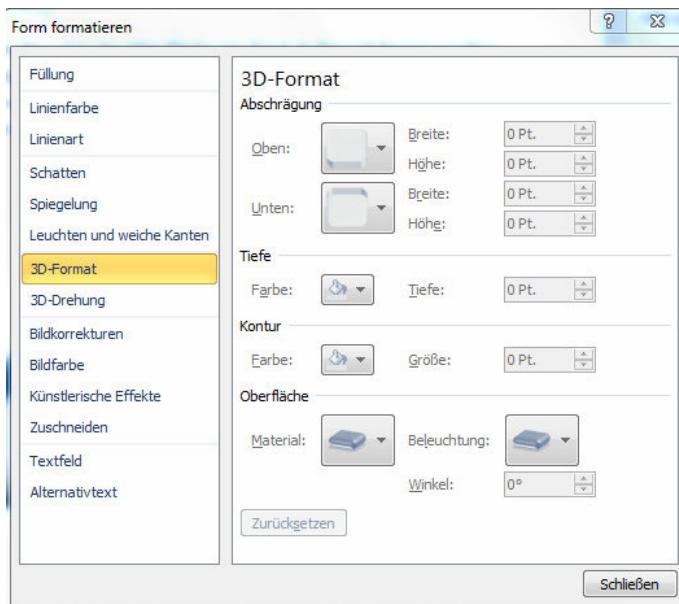
Somit entstand folgende Grafik:



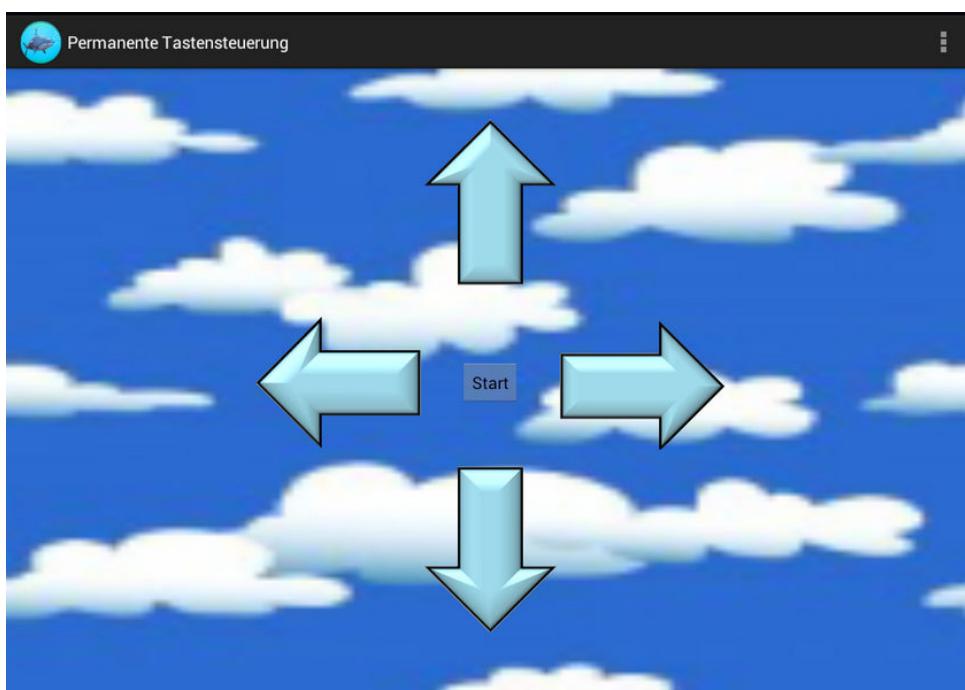
**Abbildung 73**

Hai-Bild für Oberflächen

Die Pfeile wurden in Microsoft Word erstellt, die Pfeilform wurde in die passenden Größe skaliert. Nach der Skalierung wurde unter „Form formatieren“ in 3D-Format passende Werte eingegeben sowie Farben geändert. Anschließend in Paint.Net der Hintergrund mithilfe der Zauberstabfunktion entfernt.



Nach dem Erstellen der Grafiken wurden diese in die Oberfläche eingebaut. Wie zum Beispiel in die Buttonoberfläche.



**Abbildung 74**

Button-Oberfläche für permanente Steuerung mit neuen Pfeil-Bildern

# 10 Sonstiges

## 10.1 Linux Library

Von Andreas Gerken

Um in einer App nativen Code verwenden zu können, müssen die C-Dateien zu einem .so File kompiliert werden. SO heißt Shared Object Library und ist das Linux Äquivalent zu DLLs auf Windows Systemen. Die Libraries speichern alle referenzierten Files mit ab und Verweise werden nur relativ gemacht, sodass die Library an einen beliebigen Ort verschoben werden kann. Da die Hardware Unterstützung für den Irdroid Adapter in C geschrieben ist, und die Irdroid App sie genau über so ein so File einbindet bietet sich das für unsere App genauso an.

Um die Shared Object Library zu verwenden, muss in Java das JNI (Java Native Interface) verwendet werden, das die Verbindung zwischen Java und nativ darstellt. Die Library kann über „System.loadLibrary(“<Name der Bibliothek“) geladen werden und die Prototypen der Funktionen müssen mit native deklariert werden.

Der GNU Compiler GCC kann solche .SO Dateien erstellen. Dazu muss man bei der Erstellung der Objekt dateien die Option –fPIC aufrufen. Sie macht, dass der Code unabhängig der Referenzen ist.

```
gcc -c -fPIC datei.c -o datei.o
```

Die so erzeugte Objekt Datei kann nun über den folgenden Aufruf in eine Library eingebunden werden.

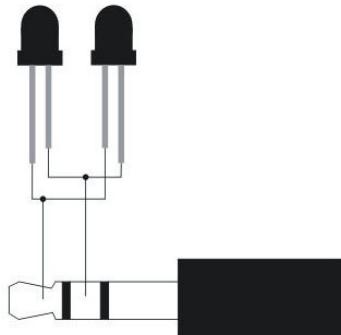
```
gcc -shared -Wl,-soname,libdatei.so.0.1 -o libdatei.so  
datei.o
```

Das Attribut –shared macht, das seine Library erstellt wird. Mit –soname kann ein Name eingestellt werden, der bei einer Referenzierung auf die Library verwendet werden muss. Er muss mit lib anfangen muss, gefolgt von einem Namen und beendet mit .so.<Versionsnummer>. Die Versionsnummer soll immer inkrementiert werden, wenn das Interface geändert wird. Libdatei.so ist der Name den die neu Kompilierte Datei bekommen soll.

So ist es möglich einzelne Dateien zu kompilieren. Wenn mehrere Dateien kompiliert werden sollen, müssen sie entweder im Aufruf angegeben werden, oder der Prozess wird über ein Makefile gesteuert. Die Irdroid App liefert einige Makefiles mit, die jedoch alle sehr kryptisch sind. Sie sind nicht dokumentiert und es gibt Mehrere die untereinander verlinkt sind. Es wird eine fertige Library mitgeliefert, die zwar auch Funktionen umfasst, die wir nicht benötigen und unserer Namenskonvention nicht entspricht, jedoch ist sie nur 180KB groß und die zusätzlichen Funktionen stören insofern nicht. Um die Namen zu ändern muss neu kompiliert werden und die Namen in die App auch noch ein gepflegt werden. Diese Erkenntnisse wurden in der Projektbesprechung vorgestellt und es wurde beschlossen, dass der Aufwand nur um die Namen zu ändern zu groß wäre. Somit wurde die orginal Library verwendet.

## 10.2 Bau eines eigenen Senders

Über den Lautsprecherausgang (AUX) des Tablets oder Smartphones soll nun ein Infrarotsignal erzeugt werden. In Internet wird behauptet, dass dies mit einer einfachen Schaltung von zwei Infrarot LEDs und eines Widerstandes geschehen kann.



Zum parallel bestellten IRdroid Adapter gehört auch eine open Source Applikation fürs Smartphone, die die Fernbedienung des Fernsehers ersetzen soll. Die erste Idee bestand also darin, mit dem selbst gebauten IR Sender, den eigenen Fernseher zu steuern, um nachzuweisen, dass das selbst gebaute Gerät funktioniert.

Die IRdroid App liest ein Lirc File ein und generiert dann mit Berühren der entsprechenden Taste ein Audiosignal, das dann vom Adapter in ein Infrarotes Signal umgewandelt wird.

Dazu kann entweder ein eigen generiertes Lirc File eingelesen werden, oder ein passendes für den jeweiligen Fernseher aus dem Internet heruntergeladen werden. Ob die Infraroten LEDs ein Lichtsignal aussenden, kann am einfachsten mit der Kamera eines weiteren Smartphones überprüft werden. Für das menschliche Auge ist nur ein begrenztes Lichtspektrum sichtbar, für die Kamera des Smartphones hingegen nicht. Leider hat diese Lösung nicht funktioniert, da das Signal, das aus dem Audioausgang des Smartphones kam zu schwach war.

Der nun verwendete IRdroid Infrarotadapter ist ein universelles Gerät, das das Audiosignal des Smartphone in ein Lichtsignal umwandelt. Außerdem verstärkt es das ankommende Signal, zur Verstärkung befindet sich deshalb eine Batterie auf der Schaltung.



**Abbildung 75**  
IRdroid-Adapter

## 10.3 Fehlersuche alter Fisch

Von Andreas Weber

Nachdem die Batterien des alten Fisches gewechselt wurden, kam keine Verbindung zwischen Fernbedienung und Fisch mehr zu Stande. Daher wurde vermutet, dass die Hardware des Fisches defekt ist. Es wurde also die Empfangsplatine des AirSwimmers demontiert, dies kann mit einem kleinen Schraubendreher geschehen. Es müssen lediglich die beiden Haken, die den Schlitten an der Zahnstange des Ballons halten, etwas auseinander gedrückt werden. Anschließend muss das Kabel, das zum Stellmotor an der Schwanzflosse geht, abgetrennt werden. Nun muss noch die Plastikverkleidung die sich zum Schutz am Schlitten befinden, vorsichtig abgenommen werden.

Nun wurde die Schaltung komplett untersucht, es wurden alle Widerstände mit einem Multimeter vermessen, alle Kondensatoren überprüft, leider ohne Ergebnis. Zuletzt wurde ein grober Schaltplan der Platine angefertigt, somit konnte das Signal, das vom Infrarotempfangsmodul kommt, über etliche Widerstände und Kondensatoren bis zu einem Bauteil auf dem lediglich ein „R“ vermerkt war, nachverfolgt werden.

Der Hersteller des Spielzeugs wies uns dann darauf hin, dass nach jedem Tauschen der Batterie, ein Vorgang der sich „paaren“ nennt, durchgeführt werden muss. Dazu wird die Fernbedienung auf das Empfangsmodul des Schlittens gerichtet, der Fisch wird eingeschaltet, innerhalb von zehn Sekunden muss nun zuerst an der Fernbedienung die „Dive“-Taste gedrückt werden und diese dann eingeschaltet werden. Die Taste muss nun gehalten werden, bis die LED am Fisch zu blinken beginnt.

Beim Zusammenbauen des Fisches, muss auf die richtige Polarität des Stellmotors für die Heckflosse geachtet werden. Wer davon ausgeht, dass die Motoren für den Schlitten und die Flosse gleich angeschlossen sind, wird enttäuscht.

## 10.4 Gehäuse

### 10.4.1 Recherche Gehäusemöglichkeiten

Von Giuseppe De Nuzzo und Ridvan Yücel

Der Irdroid-Sender besteht nur aus einer Platine und ist somit ungeschützt vor äußeren Beschädigungen. Hierbei ist vor allem der Schutz vor elektrostatischen Entladungen zu beachten. Dadurch könnten unter ungünstigen Umständen einige Bauteile beschädigt werden. Um diese Gefahren zu umgehen, wird ein Gehäuse für den Irdroid benötigt. Der erste Ansatz war, ein Spielzeug zu finden, welches passend zum Fisch und ohne großen Umbau verwendet werden kann. Aufgrund des zu hohen Gewichts der geeigneten Spielzeuge, war es nicht möglich diese als Gehäuse einzusetzen.

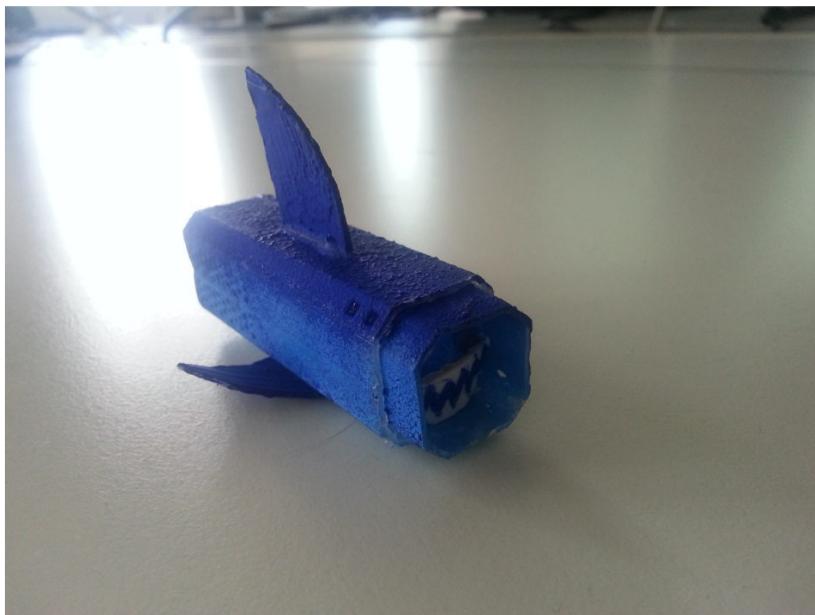
Der zweite Versuch bestand darin, einen USB-KFZ-Adapter als Gehäuse umzubauen. Aufgrund der komplizierten Form des Senders, war es nicht möglich diesen Umbau zu realisieren.

Ein aus einem Elektrofachmarkt käuflich erworbenes Produkt befand sich in einem Plastikgehäuse. Dieses Gehäuse wurde auf die Länge des Irdroids angepasst und mit einer Aussparung für die Kopfhörer-Klinke versehen.

#### **10.4.2 Gehäuse bemalen und verschönern**

*Von Belgüzar Kocak*

Nachdem ein passendes Gehäuse gefunden wurde und von Andreas Gerken begonnen wurde dieses als Fisch zu gestalten, mussten noch Flossen angebracht werden, sowie das gesamte in den passenden Farben bemalt werden. Als erstes wurden die Plastikflossen gefeilt anschließend mit Tesa Sekundenkleber angebracht. Die farbliche Gestaltung wurde mit Acrylfarbe und Naturschwamm erarbeitet.



**Abbildung 76**

fertiges Gehäuse für Irdroid-Adapter

## **11 Tests**

*Von Melanie Knappe*

Um zu überprüfen, ob die Anforderungen an die Applikation umgesetzt sind, werden Tests durchgeführt. Diese Tests werden gemacht, um die laufenden Funktionen nachzuweisen. Dies ist zum einen für die Entwicklung wichtig, damit überprüft werden kann welche Funktionen funktionieren und welche noch verbessert werden müssen. Zum anderen sind die Tests wichtig, damit dem Kunden demonstriert werden kann, dass seine gewünschte Software alle Anforderungen erfüllt.

Für die Tests wurde ein eigenes Testprotokoll erstellt. Die ausführlichen Tests befinden sich im Anhang. Allerdings werden die wichtigsten Informationen und Ergebnisse auch in der Dokumentation aufgelistet.

Als erstes wird kurz der Aufbau der Testfälle erklärt. Jeder Testfall bekommt eine ID, einen Name und eine Kurzbeschreibung. Bei jedem Testfall wird kurz die erwartete Funktionsweise beschrieben. Wenn der Test durchgeführt wird, werden Datum und der Name des Testers festgehalten. Das ganze schaut wie folgt aus.

| ID:  | Name:              | Durchgeführt am: | Durchgeführt von: |
|--|--------------------|------------------|-------------------|
| 01.01  | Aux Ausgang wählen |                  |                   |
| <b>Kurzbeschreibung:</b>   |                    |                  |                   |
| Je nachdem an welcher Seite der Aux Ausgang ist, wird die Applikation im Hochformat oder Querformat angezeigt              |                    |                  |                   |
| <b>Erwartete Funktionsweise:</b>   |                    |                  |                   |
| Aux Ausgang an der langen Seite → Applikation im Querformat<br>Aux Ausgang an der kurzen Seite → Applikation im Hochformat |                    |                  |                   |
| <b>Fehlerbeschreibung:</b>   |                    |                  |                   |
| <b>Ergebnis:</b>   |                    |                  |                   |

Beim Testen werden die Felder Fehlerbeschreibung und Ergebnis ausgefüllt. Wenn alles funktioniert, wird das Feld Fehlerbeschreibung nicht ausgefüllt.

Die ID setzt sich aus 2 Zahlen zusammen, die durch einen Punkt getrennt sind. Die erste Zahl gibt den Testbereich an. Es gibt drei Testbereiche: Oberfläche (01), Senden (02), Physikalische Werte (03). Die zweite Zahl ist ein Laufindex, damit die verschiedenen Testfälle unterschieden werden kann. Zum Beispiel bedeutet 01.05, dass der Test ein Oberflächentest ist und die Nummer 05 besitzt. Zusätzlich gibt es noch die Unterscheidung, ob das Tablet im Hoch- bzw. Querformat getestet wurde. Nach der bisherigen ID kann noch ein H bzw. Q stehen. H steht für Hochformat und Q für Querformat. Manche Funktionen funktionieren nur im Querformat, aber nicht im Hochformat, daher muss diese Information dabei stehen und unterschieden werden.

Alle Testfälle werden noch in einer Übersichtstabelle aufgelistet. In der Übersichtstabelle wird zusätzlich noch mit Farben gearbeitet. Die Farbe Grün bedeutet, dass die Funktion einwandfrei funktioniert. Gelb bedeutet, dass die Funktion noch nicht richtig funktioniert bzw. noch „Schönheitsfehler“ besitzt. Die Farbe Rot bedeutet, dass die Funktion noch nicht funktioniert.

### Oberfläche

Die Oberfläche wurde zwei Mal getestet. Die Tests haben am 20.6.13 und am 24.06.13 stattgefunden. Beide Test wurden vom Andreas Weber und von Melanie Knappe durchgeführt. Die ausführlichen Testprotokolle befinden sich im Anhang. Die Ergebnisse werden in dieser Dokumentation aufgelistet. Die Ergebnisse der beiden Tests werden zusammen in einer Tabelle aufgeführt, um die Fortschritte an der Oberfläche darstellen zu können. Der Test am 24.06.13 ist ausführlicher, als der am 20.06.13, weil alle Oberflächenfunktionen sowohl im Hochformat als auch im Querformat durchgeführt worden sind. Der Test am 20.06.13 wurde nur im Querformat durchgeführt. Daher weißt die Tabelle an manchen Stellen Lücken auf.

| ID    | Name               | Durchgeführt am 20.06.13 | Durchgeführt am 24.06.13 | Zusatzinformation    |
|-------|--------------------|--------------------------|--------------------------|----------------------|
| 01.01 | Aux Ausgang wählen | Funktioniert             | Funktioniert             | Quer- und Hochformat |

|        |  |                              |   |                      |
|--------|--|------------------------------|---|----------------------|
| 01.02Q | Infoseite                                  | Funktioniert                 | Funktioniert                              | Querformat           |
| 01.02H | Infoseite                                  |                              | Fehlerbeschreibung beachten               | Hochformat           |
| 01.03  | Infoseite Beenden                          | Funktioniert                 | Funktioniert                              | Quer- und Hochformat |
| 01.04  | Animierter Hintergrund                     | Funktion nicht implementiert | Funktion nicht implementiert              | Quer- und Hochformat |
| 01.05  | Lautstärkenregelung                        | Funktioniert                 | Fehlerbeschreibung beachten               | Quer- und Hochformat |
| 01.06Q | Fisch in alle Richtungen bewegen (Kippen)  | Funktioniert                 | Funktioniert (Achsen-Ausrichtung richtig) | Querformat           |
| 01.06H | Fisch in alle Richtungen bewegen (Kippen)  |                              | Fehlerbeschreibung beachten               | Hochformat           |
| 01.07  | Permanenter Modus (Kippen)                 | Funktioniert                 | Sendefunktion nicht implementiert         | Quer- und Hochformat |
| 01.08  | Zurück zum Startbildschirm (Kippen)        | Fehlerbeschreibung beachten  | Funktioniert                              | Quer- und Hochformat |
| 01.09Q | Fisch in alle Richtungen bewegen (Wischen) | Funktioniert                 | Funktioniert                              | Querformat           |
| 01.09H | Fisch in alle Richtungen bewegen (Wischen) |                              | Fehlerbeschreibung beachten               | Hochformat           |
| 01.10  | Permanenter Modus (Wischen)                | Fehlerbeschreibung beachten  | Sendefunktion nicht implementiert         | Quer- und Hochformat |
| 01.11  | Zurück zum Startbildschirm (Wischen)       | Fehlerbeschreibung beachten  | Funktioniert                              | Quer- und Hochformat |
| 01.12  | Alle Tastenkombinationen gehen (Tasten)    | Fehlerbeschreibung beachten  | Funktioniert                              | Quer- und Hochformat |
| 01.13  | Mehrere Tasten auf einmal Drücken          | Funktioniert                 | Funktioniert                              | Quer- und Hochformat |

|       |                                     |                             |              |                      |
|-------|-------------------------------------|-----------------------------|--------------|----------------------|
| 01.14 | Schneller Wechsel der Tasten        | Fehlerbeschreibung beachten | Funktioniert | Quer- und Hochformat |
| 01.15 | Zurück zum Startbildschirm (Tasten) | Fehlerbeschreibung beachten | Funktioniert | Quer- und Hochformat |

Bei dem Test vom 24.06.13 wurde der gesamte Test in Querformat und im Hochformat durchgeführt. Das hat zur Folge, dass eine Aussage getroffen werden kann, ob die Oberfläche besser im Hochformat oder im Querformat funktioniert. Die nachfolgende Tabelle fasst die Farbcodierung zusammen.

| Test vom 24.06.13 Hochformat vs. Querformat |                     |                     |                    |
|---|---------------------|---------------------|--------------------|
|   | Anzahl grüne Felder | Anzahl gelbe Felder | Anzahl rote Felder |
| Hochformat                                  | 10                  | 4                   | 0                  |
| Querformat                                  | 13                  | 1                   | 0                  |

Das Hochformat hat drei grüne Felder weniger als das Querformat. Zusätzlich hat das Hochformat 4 gelbe Felder und das Querformat nur eins. Die Oberfläche funktioniert im Querformat besser als im Hochformat.

### ***Verschiedene Android Versionen im Vergleich***

Nachdem die Oberfläche im Großen und Ganzen ganz gut funktioniert hat, wurde als nächstes die Kommunikation zwischen Android Gerät und AirSwimmer getestet. Es stellte sich heraus, dass das Senden mit Applikation auf verschiedenen Android Geräten, mit unterschiedlicher Android Version, unterschiedlich gut funktionierte. Der erste Test wurde von Andreas Weber, Sabine Kressierer und Melanie Knappe durchgeführt. Es standen drei verschiedenen Geräten zur Verfügung:

- Samsung Tablet mit Android Version 3.2
- Samsung Tablet mit Android Version 4.0.2
- Samsung Smartphone mit Android Version 4.1.2

Für die bessere Darstellung der Ergebnisse wurde eine tabellarische Darstellung gewählt. Die Farbcodierung ist die gleiche, wie bei der Übersichtstabelle bei den Oberflächentests.

| ID    | Name                                      | Datum      | Ergebnis                     |  |                               |
|-------|---|------------|------------------------------|--|-------------------------------|
|       |   |            | Android Version 3.2 (Tablet) | Android Version 4.0.2 (Tablet)                             | Android Version 4.1.2 (Handy) |
| 02.01 | Senden der Befehle an AirSwimmer (Kippen) | 24.06.2013 | Applikation hängt sich auf   | Funktioniert, aber nur wenn App in 1 Richtung bewegt wird. | Applikation hängt sich auf    |

|       |  |            |                       |  |                                    |
|-------|--|------------|-----------------------|--|------------------------------------|
| 02.02 | Senden der Befehle an AirSwimmer (Permanent-Kippen)    | 24.06.2013 | Nicht implementiert . | Nicht implementiert .                          | Nicht implementiert .              |
| 02.03 | Senden der Befehle an AirSwimmer (Wischen)             | 24.06.2013 | Funktion geht nicht.  | Nach 10-maligem senden reagiert der AirSwimmer | Funktion geht nicht.               |
| 02.04 | Senden der Befehle an AirSwimmer (Permanent - Wischen) | 24.06.2013 | Nicht implementiert . | Nicht implementiert .                          | Nicht implementiert .              |
| 02.05 | Senden der Befehle an AirSwimmer (Tasten)              | 24.06.2013 | Funktion geht nicht.  | Tasten müssen sehr lange gedrückt werden       | Tasten müssen oft gedrückt werden. |

Die Kommunikation zwischen Android Gerät und AirSwimmer war bei keinen der drei Geräte zufriedenstellend. Am besten der drei Geräte funktionierte es mit dem Tablet, auf dem die Android Version 4.0.2 lief.

Es wurde bis jetzt zu wenig Tests durchgeführt, damit eine Aussage getroffen werden kann, welches Android Gerät am besten mit unserer Applikation zusammen passt. Das Ergebnis aus diesem Test ist, dass die Kommunikation auf verschiedenen Geräten unterschiedlich gut funktioniert.

Wobei nicht geklärt ist, ob das unterschiedliche Verhalten mit der Android Version oder mit dem Gerät zusammen hängt.

Um diese Fragen klären zu können werden weitere Tests benötigt. Um herauszufinden, ob die Geräte das Ausschlag gebende sind, wären die Testbedingungen wie folgt:

- Unterschiedliche Geräte (Tablet, Smartphone..) und von verschiedenen Herstellern
- Auf denen die gleiche Android Version läuft.

Um zu testen, ob die Android Version Ausschlag geben ist, wären die Test Bedingungen:

- Eine Geräteart (Samstung Tab 10.1)
- Auf denen unterschiedliche Android Version laufen

Da wir leider diese Möglichkeiten nicht haben, haben wir unsere Applikation auf Geräten getestet, die uns zur Verfügung stehen.

Damit wurde der zweite Test mit folgenden Geräten und Android Versionen Durchgeführt:

- Samsung Tablet mit Android Version 3.2
- Samsung Tablet mit Android Version 4.0.2
- Samsung Tablet 2 mit Android Version 4.0.3
- EasyPad mit Android Version 4.0.4
- Samsung Galaxy 2 mit Android Version 4.1.2

Der zweite Test wurde von Andreas Weber, Knappe Melanie und Caroline Pilot durchgeführt.

Der Test wurde im Hochformat und im Querformat durchgeführt. Die Ergebnisse für das Hochformat zeigt folgende Tabelle. Pro Zeilen wird ein Modus dargestellt. Die Spalten zeigen die Android Versionen.

| Hochformat |  |                         |                         |                         |                       |  |
|------------|--|-------------------------|-------------------------|-------------------------|-----------------------|--|
| ID         | Name                                       | Android Version 3.2     | Android Version 4.0.2   | Android Version 4.0.3   | Android Version 4.0.4 | Android Version 4.1.2  |
| 02.01H     | Senden der Befehle an AirSwimmer (Kippen)  | Funktioniert sporadisch | Modus stürzt ab         | Funktioniert sporadisch | Modus stürzt ab       | Modus stürzt ab  |
| 02.03H     | Senden der Befehle an AirSwimmer (Wischen) | Funktioniert nicht      | Funktioniert nicht      | Funktioniert nicht      | Funktioniert nicht    | Schlittensteuerung funktioniert, aber Richtungssteuerung nicht |
| 02.05H     | Senden der Befehle an AirSwimmer (Tasten)  | Unbefriedigend          | Funktioniert sporadisch | Funktioniert nicht      | Funktioniert          | Tasten funktionieren bis auf „Links-Taste“                     |

Im Hochformat hat das Senden mit keiner Android Version wirklich gut funktioniert.

Bei keiner Android Version hat alle Modi funktioniert. Nur bei der Android Version 4.0.4 hat der AirSwimmer auf die Tastendrücke reagiert.

Der Modus Wischen hat am meisten Defizite. Der Modus Tasten hat am besten, auf allen Android Versionen, funktioniert.

Die folgende Tabelle zeigt das Ergebnis für Querformat.

| Querformat |  |                         |                       |  |  |                         |
|------------|--|-------------------------|-----------------------|--|--|-------------------------|
| ID         | Name                                       | Android Version 3.2     | Android Version 4.0.2 | Android Version 4.0.3  | Android Version 4.0.4                                  | Android Version 4.1.2   |
| 02.01Q     | Senden der Befehle an AirSwimmer (Kippen)  | Funktioniert sporadisch | Funktioniert          | Funktioniert sporadisch  | Funktioniert teilweise, aber Modus stürzt teilweise ab | Funktioniert sporadisch |
| 02.03Q     | Senden der Befehle an AirSwimmer (Wischen) | Funktioniert nicht      | Funktioniert          | Schlittensteuerung funktioniert, aber Richtungssteuerung nicht | Funktioniert sporadisch                                | Wurde nicht getestet    |
| 02.05Q     | Senden der Befehle an AirSwimmer (Tasten)  | Unbefriedigend          | Funktioniert          | Funktioniert sporadisch  | Funktioniert nicht                                     | Funktioniert            |

Im Querformat hat am besten die Android Version 4.0.2 abgeschnitten. Bei dieser Version gingen alle Modi. Im Querformat lässt sich schlecht beurteilen, welcher Modus am besten abgeschnitten hat.

Aus dem Test mit den unterschiedlichen Android Versionen lassen sich noch weitere Aussagen treffen. Im Folgenden werden noch die Farbcodierungen ausgewertet. Die Folgende Auswertung unterscheidet die Beiden Formatarten.

| Alle Modi zusammen |                     |                     |                    |
|--------------------|---------------------|---------------------|--------------------|
|                    | Anzahl grüne Felder | Anzahl gelbe Felder | Anzahl rote Felder |
| Hochformat         | 1                   | 6                   | 8                  |
| Querformat         | 4                   | 8                   | 2                  |

Die Tabelle listet die Anzahl der farbigen Felder für die jeweiligen Formate auf. In diese Tabelle fließen alle Modi mit ein. Das Ergebnis der Tabelle ist, das mehr Funktionen im Querformat gehen, als im Hochformat. Das Querformat hat im Vergleich zum Hochformat besser abgeschnitten.

Die gleiche tabellarische Darstellung wurde für jeden Modus erstellt.

- Kippen

| Kippen     |                     |                     |                    |
|------------|---------------------|---------------------|--------------------|
|            | Anzahl grüne Felder | Anzahl gelbe Felder | Anzahl rote Felder |
| Hochformat | 0                   | 2                   | 3                  |
| Querformat | 1                   | 4                   | 0                  |

- Wischen

| Wischen    |                     |                     |                    |
|------------|---------------------|---------------------|--------------------|
|            | Anzahl grüne Felder | Anzahl gelbe Felder | Anzahl rote Felder |
| Hochformat | 0                   | 1                   | 4                  |
| Querformat | 1                   | 2                   | 1                  |

- Tasten

| Tasten     |                     |                     |                    |
|------------|---------------------|---------------------|--------------------|
|            | Anzahl grüne Felder | Anzahl gelbe Felder | Anzahl rote Felder |
| Hochformat | 1                   | 3                   | 1                  |
| Querformat | 2                   | 2                   | 1                  |

Das Ergebnis der Tabellen ist, dass alle Modi im Querformat besser funktionieren als im Hochformat.

Das Senden im Querformat hat im Vergleich zum Senden im Hochformat besser abgeschnitten.

## 12 Ausblick

### Wo geht's hin?, Wo soll's hin gehen?, Wo kann's hin gehen?

Von Andreas Weber

Die erste Frage die sich damals an der ersten Projektrunde stellte, war: „Mit welcher Übertragungstechnologie funktioniert denn die Kommunikation zwischen dem Fisch und der Fernbedienung?“. Dass uns ausgerechnet die einfache Infrarotverbindung solche Probleme macht, hätte damals keiner gedacht. Vielleicht sollte man bei zukünftigen Versionen der Android App auf eine etwas zeitgemäßere Kommunikationstechnologie umsteigen. Es braucht ja nicht gleich W-Lan sein, aber zumindest Bluetooth wäre schon einmal eine deutliche Verbesserung. Einen Bluetooth Fisch gibt es leider nicht zu kaufen, jedoch hat heute fast jedes Tablet oder Smartphone eine eingebaute Bluetooth Schnittstelle, die also ohne Hardwareerweiterung zur Kommunikation genutzt werden kann. Entweder muss dann eben auf ein anderes Spielzeug gesetzt werden, oder muss eine zusätzliches Empfangsmodul verbaut werden. Zur Auswahl stünde zum Beispiel ein Helikopter, der mit Bluetooth angesteuert werden kann, leider sind diese Helikopter deutlich schwerer zu fliegen als ein sehr träge Helium Ballon.

Bei den abschließenden Tests wurden auch deutliche Unterschiede zwischen den Android Versionen und ob die App auf einem Tablet oder einem Smartphone installiert war, festgestellt. Beim Betrieb auf einem Smartphone gab es des Öfteren Probleme mit der Skalierung des Bildschirms, da das Ziel in erster Line war eine App für ein Tablet zu entwickeln. Die Tablets, vor allem die älteren Modelle die bereits zwei Jahre am Markt waren, erwiesen sich in einigen Fällen als zu langsam, bzw. die App zu wenig performant. Diese Unterschiede zwischen den Geräten, bzw. den Betriebssystemversionen sollten in Nachfolgeversionen untersucht und eliminiert werden.

Es wurde außerdem ein zweiter Fisch bestellt, dadurch erkannten wir, dass es Unterschiede bezüglich der infrarot Kommunikation zwischen den verschiedenen AirSwimmern gibt. Deshalb benötigt man für jedes unterschiedliche Gerät ein eigenes Lirc File. Zum einfacheren Wechseln zwischen den Lirc Files, sollte die Oberfläche dahingehend erweitern/verbessert werden.

Bereits am Anfang als die ersten Lirc Files mit der IRdroid App getestet wurden, fanden wir heraus, dass es zwischen der Lautstärkeinstellung des Android-fähigen Geräts und der Güte der Infrarotverbindung einen Zusammenhang gibt. Deshalb wurde eine Funktion in die Oberfläche integriert, mit der man die passende Lautstärkeinstellung automatisiert einstellen kann. Diese Funktion muss zurzeit unter den Einstellung manuell aufgerufen werden, der ermittelte Lautstärkewert wird dann gespeichert und steht bei weiteren Starts der App wieder zur Verfügung. Für weitere Versionen, sollte die Oberfläche zur Lautstärkenkalibrierung beim ersten Start automatisch geöffnet werden. Außerdem ist die Auflösung in der die Lautstärke verstellt wird zu grob, es sollte also eine feinere Schrittweite gewählt werden. Zudem sollte erforscht werden, warum die Lautstärkeinstellung Einfluss auf das IR Signal hat, und eventuell ein perfekter Lautstärkewert für jedes Gerät automatisch errechnet werden.

Momentan ist die App so ausgelegt, dass der Fisch über drei verschiedene Steuermethoden geflogen werden kann, es muss aber immer die Bewegung der einzelnen Bauteile vorgegeben werden (Flosse nach rechts, Schlitten nach vorne). Die

Steuerung sollte in späteren Versionen dahingehend erweitert werden, dass auch Bewegungen vorgegeben werden kann (z.B. links herum fliegen, automatisch geradeaus fliegen). Die Oberflächen für eine entsprechende Steuerung sind bereits in der jetzigen App vorhanden. Außerdem wäre eine Funktion denkbar, die nach dem betätigen eines „Record“ Buttons, alle Eingaben des Benutzers aufnimmt und diese nach betätigen eines „Play“ Buttons wiedergibt, also genau dieselbe Figur wieder nachfliegt.

Außerdem kristallisierten sich zu Ende des Projektes auch Hardwareprobleme des Fisches heraus. So musste am Fisch alle zwei Wochen Helium nachgefüllt werden, zuletzt flog er dann trotz frischer Heliumfüllung nicht mehr. Es wird vermutet, dass ein immer größerer Anteil an Luft in den Ballon gelangt ist, es wird also empfohlen, den Ballon einmal komplett zu leeren und mit neuem Helium zu befüllen. Auch die Umgebungstemperatur spielt eine wichtige Rolle beim Ausbalancieren des Fisches, wird er in einer Umgebung ausbalanciert und kommt man dann in einen Raum mit anderer Temperatur, steigt er entweder oder fällt.

## **13 Anhang**

### **13.1 Anforderungsanalyse**

*Siehe Datei „Anforderungsanalyse“ in Ordner „Sonstiges“*

### **13.2 Zeiterfassung**

*siehe Ordner „Zeiterfassung“*

### **13.3 Projektplan**

*Siehe Datei „Projektplan“ in Ordner „Sonstiges“*

### **13.4 Protokolle**

*siehe Ordner „Protokolle“*

### **13.5 Testprotokolle**

*Siehe Ordner „Test“*

### **13.6 Klassendiagramm der Oberfläche**

