Entwicklung von *Mobile-Music*-Apps mit der *Pure Data* - Library „*libpd*” am Beispiel einer App für den therapeutischen Einsatz

Bachelorarbeit

von

Benjamin Stahl

im Rahmen des Seminars

„Computermusik und Multimedia 02”

im Sommersemester 2014

bei

DI Johannes Zmölnig

INHALT

ABSTRACT....2

1. VORWORT....3

2. EINLEITUNG....4

2.1. Motivation....4

2.2. Literatur....6

2.3. Definition und Erweiterung des Musiktherapiebegriffs....8

3. GESCHICHTLICHES....10

3.1. Geschichte des Smartphones....10

3.2. Geschichte von libpd....12

3.3. Vergangene und aktuelle Entwicklungen im Bereich Mobile-Music....13

3.3.1. Kategorisierung....15

4. SYSTEMBESCHREIBUNG....17

4.1. Hardware....17

4.2.1. Touchscreen....17

4.2.2. Beschleunigungssensor....18

4.2. Software....19

4.2.1. Tonerzeugung: die Pd-Library libpd....19

4.2.2. Grafikwerkzeug: Processing....20

4.2.3. Das Android-Betriebssystem....22

5. ENTWICKLUNGSPROZESS....23

5.1. Integration der Programmierschnittstellen....23

5.2. Freie Improvisation mit metallischem Klang....29

5.2.1. Überblick....29

5.2.2. Realisierung....31

5.2. Koordination des Handgelenks....36

5.3.1. Überblick....36

5.3.2. Realisierung....38

6. SCHLUSSFOLGERUNG....44

7. LITERATURVERZEICHNIS....46

8. ERKLÄRUNG....49

ABSTRACT

This work deals with the design and realization of an Android *mobile-music* app for use in a therapeutic context. The app "*Healthy Sounds"*, which has been created in the course of this work, contains two instruments. The first instrument is designed to be used in a rather conventional, psychotherapeutically oriented music therapy and allows the patient to improvise with Tibetan Singing Bowl sounds using a "pad-like" multitouch interface. The second instrument shall be used in a ergo- or physiotherapeutically oriented music therapy. It converts the movement data of the patient's wrist to sound and helps coordinating the wrist by giving audiovisual feedback.

Concerning the implementation of the *mobile-music* app, the *Pure Data* library *libpd* was used for sound synthesis, while the graphics were rendered using the Android version of the graphics tool *Processing*. The description of the realization process of *Healthy Sounds* especially focuses on the use of libpd.

As a conclusion, it was found out that the applied combination of software libraries is suitable for the implementation of smaller projects and prototypes. By careful programming, *Healthy Sounds* could be brought to also run stable on devices with average performance, making the app ready for a field test on patients' and therapists' acceptance and opinions.

1. VORWORT

Für das Seminar „Computermusik und Multimedia 02“ im Sommersemester 2014 bei Johannes Zmölnig sollte als Projektarbeit eine *Mobile-Music*-App entwickelt werden.

Die Wahl eines Themas bzw. eines Konzepts für die zu entwickelnde App blieb den Teilnehmern des Seminars selbst überlassen.

Diese Bachelorarbeit handelt von der Entwicklung einer App für den *musiktherapeutischen* Einsatz: Die Android-App *„Healthy Sounds“* enthält zwei Instrumente. Die Instrumente sollen ein möglichst breites Spektrum therapeutischer Einsatzmöglichkeiten abdecken. Deshalb wurde einerseits ein einfach zugängliches elektronisches Instrument zur freien Improvisation mit einem wählbaren Farbschema entwickelt, welches eher in der klassischen psychotherapeutischen Musiktherapie seinen Einsatz fände, andererseits ein Instrument, welches Bewegungen des Handgelenks in Klang umsetzt und somit dem Patienten ein auditives Feedback gibt und das selbständige Üben vereinfacht bzw. interessanter macht. Beide Instrumente, insbesondere das zuletzt genannte, sind als Beispiele für ihre Kategorie zu betrachten.

Die Arbeit beschäftigt sich im Allgemeinen mit dem Feld *Mobile-Music*, indem sie vergangene und aktuelle Entwicklungen aufzeigt, und dokumentiert im Speziellen den Entstehungsprozess einer *Mobile-Music*-App.

Ich widme diese Bachelorarbeit meinen Großeltern, die mich auf meinem Bildungsweg immer unterstützt haben und denen ich zu einem großen Teil meine bisherigen Erfolge zu verdanken habe.

Ich bedanke mich bei meiner Mutter Nicole Stahl, welche mir durch ihre fachkundige Beratung sehr bei der Entwicklung eines *Mobile-Music*-Instruments für den Einsatz im ergo- und physiotherapeutischen Bereich geholfen hat.

Ich danke meinem Kollegen und guten Freund Jonas Helm, welcher mit mir das o.g. Seminar besucht hat für die Motivation und Unterstützung beim Fertigstellen der Bachelorarbeit.

Vielen Dank an Johannes Zmölnig für die interessante Themenauswahl und Gestaltung des Seminars.

Außerdem möchte ich mich bei Danielle Sofer für reichlichen Input in interessanten Gesprächen zum Thema Musiktherapie bedanken.

2. EINLEITUNG

2.1. Motivation

Mobile-Music-Apps sind Anwendungen für mobile Endnutzergeräte, bei denen Klang eine zentrale Rolle spielt. Das Genre „*Mobile Music*“ reicht somit von simplen Playern und *Music-Information-Retrieval*-Apps über Apps für die Musikproduktion, virtuelle Nachbauten von akustischen oder elektronischen Instrumenten und abstrakten Instrumenten mit neuartigen Interfaces bis hin zu musikalischen *Augmented-Reality*-Apps und musikalischen Spielen (siehe Kapitel *3.3.1.* für eine genaue Differenzierung mit Beispielen). Wie immer auch ein *Mobile-Music*-Instrument geartet ist, es handelt sich oft um eine starke Vereinfachung des realen Vorbilds bzw. im Falle eines abstrakten Instruments um ein (aus einer klassisch-virtuosen Musiktradition betrachtet) sehr simples Instrument. Dies ist zum Großteil bedingt durch die Eingeschränktheit des Interfaces (kleiner Touchscreen, keine sonstigen kinästhetischen Elemente).

Ein derartiges Instrument bietet allerdings die Möglichkeit, Menschen mit keinen oder geringen musikalischen Vorkenntnissen einen Zugang zum eigenständigen Musizieren zu geben. Was aus Sicht einiger klassisch-virtuos orientierter Musiker wohl der größte Kritikpunkt an *Mobile-Music*-Apps ist, nämlich die Unfähigkeit oder starke Eingeschränktheit, eine Virtuosität auf dem entsprechenden Instrument zu entwickeln oder salopp gesagt der „Spielzeug-Charakter“, kann für den *therapeutischen* Einsatz ein großer Pluspunkt sein:

Bei der psychotherapeutisch orientierten, auf Improvisation basierenden Musiktherapie steht Virtuosität nicht im Vordergrund, sondern vielmehr die Kommunikation zwischen Therapeut und Patient in der Sprache der Musik [1]. Dabei ist es von Vorteil, wenn der Patient einen einfachen Zugang zu seinem Instrument findet und nicht durch komplizierte Interfaces, die auf musikalischem Basiswissen aufbauen, welches sich der Patient eventuell erst noch erarbeiten müsste, abgeschreckt wird.

Außerdem bieten virtuelle Instrumente die Möglichkeit, das Interface für Patienten mit fortgeschrittenen musikalischen Kenntnissen (zum Beispiel durch Änderung einer Tonleiter, auf die die Gesten auf dem Touchscreen abgebildet werden) anzupassen.

Ein weiterer Vorteil von *Mobile-Music*-Apps für den musiktherapeutischen Einsatz ist die frei gestaltbare visuelle Oberfläche, welche durch die Entwicklung einer ansprechenden grafischen Benutzeroberfläche verwendet werden kann, um ein Instrument für den Benutzer attraktiver zu machen [2] und zudem durch unterschiedliche Farbschemata an die aktuelle Stimmung des Patienten angepasst werden kann.

Zusätzlich zu den Einsatzmöglichkeiten in der klassischen, eher psychotherapeutisch orientierten Musiktherapie können *Mobile-Music*-Apps durch den heutzutage in jedem Smartphone verbauten dreidimensionalen Beschleunigungssensor auch in einer motorischen Therapie verwendet werden. Bewegungen des Patienten können sonifiziert (= in Klang umgesetzt) werden und somit dem Patienten (evtl. zusätzlich zu einem visuellen Feedback) ein auditives Feedback geben. Dies kann motorische Therapie attraktiver und leichter zugänglich machen und somit den Patienten dazu ermutigen, bestimmte motorische Übungen auch selbständig zu machen.

Hierfür kann das Smartphone mittels einer für diesen Einsatz konzipierten Konstruktion an ein bestimmtes Körperteil befestigt werden und die Bewegungen nach einer Kalibrierung aus den Daten des Beschleunigungssensors ermittelt werden. Schrittzähler-Apps, welche ebenfalls mit dem

eingebauten Beschleunigungssensor arbeiten, stellen eine Vorstufe dieser Idee dar und haben sich mittlerweile im Fitness-Bereich etabliert. Die elektronische Umsetzung von Bewegungsdaten in Klang wurde bereits in einigen Anwendungen (vor allem im Sport-Bereich) erprobt [3], [4]. Das Ziel dieser Arbeit ist es, für den Therapiebereich eine derartige Anwendung zu entwickeln.

Neben Audiobibliotheken für die herkömmlichen Programmiersprachen, welche zum Erstellen mobiler Anwendungen verwendet werden (zum Beispiel das Paket *android.media* für in *Java* programmierte *Android*-Apps oder das *Core Audio* - Framework für in *Objective C* programmierte iOS-Apps), erschienen relativ bald nach der Veröffentlichung der jeweiligen Software-Development-Kits auch Wrapper für die Audio-Programmiersprachen *Csound*, *Supercollider* und *Pure Data*, um die Programmierung von Klang in mobilen Anwendungen einfacher und flexibler zu machen. Der Pure Data - Wrapper *libpd* ermöglicht es, grafisch programmierte Pure Data Patches in verschiedenste Programmiersprachen zu integrieren, es existieren auch APIs für Android und iOS [5]. Für das Ziel, eine *Mobile-Music*-App für Android zu programmieren, erscheint libpd als Mittel der Wahl.

2.2. Literatur

Als Literaturgrundlage für den Einsatz elektronischer Hilfsmittel in der Musiktherapie wurden zwei Veröffentlichungen der Musiktherapeutin Wendy Magee verwendet.

In ihrem Artikel „Electronic technologies in clinical music therapy: A survey of practice and attitudes" (2006) beschreibt sie die technischen Möglichkeiten und Einschränkungen elektronischer Therapieinstrumente [6]. Anhand einer Umfrage von Musiktherapeuten wird erörtert, warum elektronische Instrumente bei Therapeuten bislang größtenteils auf Ablehnung stoßen. Als Hauptgrund findet Sie einen Mangel an Trainingsangeboten für Therapeuten, die die Verwendung der neuartigen Instrumente genau erklären.

Wendy Magee und Karen Burland beschreiben in „An Exploratory Study of the Use of Electronic Music Technologies in Clinical Music Therapy" (2008) ein von ihnen entwickeltes Modell zur Musktherapie mit elektronischen Hilfsmitteln [7]. Als Indikatoren für eine solche Therapie zählen sie komplexe körperliche und sensorische Störungen, Motivationsprobleme und bestimmte Erfordernisse bezüglich des Ausdrucks der eigenen Identität auf. Kontraindiziert sind elektronische Hilfsmittel in Fällen, bei denen der Patient sich der kausalen Wirksamkeit der Therapie nicht bewusst ist.

Ein weiterer Ausgangspunkt für die Verwendung elektronischer Instrumente in der Musiktherapie ist die Masterarbeit „Investigation of the use of Multi-Touch Gestures in Music Interaction" (2013) des Toningenieurs Nicholas Arner [2]. Zum Teil aufbauend auf den Forschungsergebnissen von Magee und Burland werden hier technische Aspekte und Designfragen von elektronsichen Instrumenten für den therapeutischen Einsatz erörtert. In Tests fand er heraus, dass Apps, welche Multitouch-Gesten unterstützen, als intuitiver gesehen werden.

Intensiv mit *Mobile-Music*-Apps im Allgemeinen hat sich der deutsche Musik- und Medienpädagoge und Musiker Matthias Krebs beschäftigt. Er ist Leiter des Berliner Smartphone-Orchesters und gibt Vorträge und Seminare zum Thema *Mobile-Music*. In seinen Präsentationsfolien finden sich aufschlussreiche Fakten zu den Einsatzmöglichkeiten und unterschiedlichen Kategorien von *Mobile-Music*-Apps [8].

Bezüglich der Umsetzung eines Instruments für den physiotherapeutischen Einsatz wurden folgende Werke zum Verstehen der physiologischen Eigenschaften des *Handgelenks* verwendet:

Gerhard Aumüller et al. beschreiben im Medizin-Standardwerk „Duale Reihe Anatomie" (2014) die Bewegungen des Handgelenks [9]. Sie finden auf zwei Achsen (*dorsal-palmar* und *transveral*) statt, was in 4 Grundbewegungen (*Dorsalextension (maximaler Winkel: 60°), Palmarflexion (80°), Radialabduktion (20°) und Ulnarabduktion (40°)*) resultiert.

Patrick Salvia et al. beschreiben in „Analysis of helical axes, pivot and envelope in active wrist circumduction" (2000), dass eine Kreisbewegung (Circumduction) des Handgelenks einen guten Anhaltspunkt für den Gesamtbewegungsbereich (also auch Mischformen der oben genannten Bewegungen) darstellt [10]. In einer von den beiden Grundachsen aufgespannten Ebene kann dieser durch eine Ellipse angenähert werden.

Bezüglich der Programmierung in Pure Data sind sowohl „Programmierung Elektronischer Musik in Pd" (2009) von Johannes Kreidler [11], als auch der *FLOSS Manuals* - Eintrag (2005-2012) [12] nützliche Informationsquellen. Zudem beschreiben Peter Brinkmann et al. in „Embedding Pure Data with libpd" (2011) die softwarearchitektonischen Eigenschaften von libpd [5].

Allgemeine Grundlagen der Android-Programmierungen werden auf der Android Entwicklerseite beschrieben [13]. Das *YouTube*-Tutorial „Android Development Tutorial“ (2013) von Derek Banas beschreibt die Projektstruktur eines Android-Projektes in *Eclipse* und gibt einen ersten Überblick über das Arbeiten mit dem Android-Software-Development-Kit [14].

Das Integrieren der verwendeten *Processing*-Grafikumgebung in eine Android-App wird im Online-Tutorial „Setting up a Processing Android project in Eclipse“ (2013) von Andy Li beschrieben [15].

2.3. Definition und Erweiterung des Musiktherapiebegriffs

Um die Anforderungen an ein elektronisches Instrument, welches in einem therapeutischen Kontext verwendet werden soll, besser nachvollziehen zu können, folgt eine kurze Beschreibung des Prinzips der Musiktherapie.

Bereits in Zeiten vor der Antike verwendeten Menschen Musik als Heilmethode. In der Regel waren es damals magische oder religiöse Vorstellungen, mit denen man sich die heilende Wirkung von Musik erklärte, so gab es beispielsweise schamanische Rituale, bei denen mit Musik böse Geister vertrieben werden sollten. Auch in der Bibel finden sich derartige Beschreibungen:

„Wenn nun der Geist Gottes über Saul kam, so nahm David die Harfe und spielte mit seiner Hand; so erquickte sich Saul, und es ward besser mit ihm, und der böse Geist wich von ihm.“

(1. Samuel 16, 23)

Musiktherapie wird heutzutage als Einzel- oder Gruppentherapie zur Behandlung unterschiedlichster Krankheitsbilder eingesetzt.

Als wissenschaftliche Disziplin existiert die Musiktherapie erst seit ungefähr 70 Jahren: Nach dem zweiten Weltkrieg entwickelten sich ausgehend von Wien unterschiedliche Forschungsgruppen [1] und später Universitätsstudien bzw. Ausbildungslehrgänge. Heute ist die Musiktherapie als komplexe interdisziplinäre praxisorientierte Wissenschaft zu betrachten. Die Deutsche Musiktherapeutische Gesellschaft schreibt in ihrer Begriffsdefinition:

„Musiktherapie ist eine praxisorientierte Wissenschaftsdisziplin, die in enger Wechselwirkung zu verschiedenen Wissenschaftsbereichen steht, insbesondere der Medizin, den Gesellschaftswissenschaften, der Psychologie, der Musikwissenschaft und der Pädagogik.

Der Begriff „Musiktherapie“ ist eine summarische Bezeichnung für unterschiedliche musiktherapeutische Konzeptionen, die ihrem Wesen nach als psychotherapeutische zu charakterisieren sind, in Abgrenzung zu pharmakologischer und physikalischer Therapie.“

[16]

Diese Definition muss allerdings nicht als allgemeingültige Norm angesehen werden, zunehmend wird auch der *physiotherapeutische* Einsatz von Musik diesem Begriff zugeordnet. [17]

Abzugrenzen von der Musiktherapie ist die *Musikwirkungsforschung*. Als Spezialgebiet der Psychophysik ordnet sie gewissen musikalischen Reizen psychische und körperliche Empfindungen bzw. Reaktionen zu. So werden beispielsweise die Wirkungen bestimmter musikalischer Charakteristika auf das sympathische oder parasympathische Nervensystem untersucht und damit in ergotrop (aktivierend) und trophotrop (entspannend) unterteilt [18].

	Musikalische Charakteristika	Wirkungen
ergotrope Musik	• Rigider, harter Rhythmus • Beschleunigend • Überwiegend Dur-Tonart • Dissonant • Laut Bsp.: Rockmusik, Techno, Marschmusik (in der Regel decken alle Genres beide Forman ab)	• Erhöhung des Blutdrucks • Beschleunigung der Atemfrequenz und des Pulses • Rhythmische Kontraktion der Muskulatur • Pupillenerweiterung • Aktivierung • Rauschzustände/Trance
trophotrope Musik	• Wenig akzentuierter Rhythmus • Überwiegend Moll-Tonart • Konsonant • Leiser Bsp.: New Age, Meditationsmusik u.a. (in der Regel decken alle Genres beide Forman ab)	• Abfall des Blutdrucks • Verlangsamung und Abflachen von Atemfrequenz und Puls • Entspannung der Muskulatur • Pupillenverengung • Beruhigung

Tabelle 1*: Charakteristika und Wirkungen von ergotroper und trophotroper Musik (aus „Decker-Voigt: Aus der Seele gespielt“ [18])*

Die obige Tabelle zeigt einige dieser Charakteristika und Wirkungen von ergotroper bzw. trophotroper Musik auf den körperlichen, mentalen und psychischen Zustand.

Obwohl die Musikwirkungsforschung eng mit der Musiktherapie verwandt ist, entspricht es im Allgemeinen nicht der Arbeitsweise eines Musiktherapeuten, mit bestimmten musikalischen Mitteln gegen bestimmte Krankheiten vorzugehen. Trotzdem macht die im Rahmen dieser Arbeit entwickelte App von diesen Mitteln Gebrauch, um die Möglichkeiten einer freien elektronischen Gestaltung des Klanges auszunutzen.

Doch nicht nur im Bezug auf Klang sollte der klassische Begriff der Musiktherapie erweitert werden, um die Möglichkeiten elektronischer Hilfsmittel besser ausnutzen zu können: Die grafische Oberfläche des Instruments kann durch ansprechende Gestaltung ebenfalls eine bestimmte Wirkung hervorrufen. Obwohl neueren Erkenntnissen zufolge kein „fix-verdrahteter“ Zusammenhang zwischen Farbe und Stimmung besteht [19], existieren sowohl popkulturelle Annahmen als auch psychophysische Studien, die gewisse Trends bezüglich der Wirkung von Farben bestätigen und damit den Einsatz bestimmter Farben in Kombination mit bestimmten Klängen zum Erzeugen einer erwünschten Wirkung nahelegen. Eine Studie von Valdez und Mehrabian (1994) ergab, dass die Farbkombination violett-blau am ehesten als beruhigend und grün-gelb am ehesten als aufweckend wahrgenommen werden [20].

3. GESCHICHTLICHES

3.1. Geschichte des Smartphones

Heutige Smartphones und Tablets ermöglichen es uns, die wichtigsten Funktionen eines Computers immer mobil verfügbar zu haben. Seit den ersten Smartphones, die Ende des vergangenen Jahrtausends auf den Markt kamen, haben sich die Geräte stetig weiterentwickelt. Wurden Smartphones zu Beginn ihrer Ära noch hauptsächlich zu Kommunikationszwecken (Mobiltelefon, Pager, Faxgerät) eingesetzt, so ermöglichten es spätestens seit der Markteinführung des *Apple iPhones* 2007 immer ausgereiftere Hardwarekomponenten und Sensoren, die Geräte auch zu anderen alltäglichen Zwecken zu verwenden (Kamera, Unterhaltungsmedien wie Videos und Spiele, Kompass, Wasserwaage, ...).

Das Prinzip, ein Telefon mit intelligenter, rechnergestützter, elektronischer Datenverarbeitung und einem dafür geeigneten Bildschirm zu kombinieren, ließ Theodore Paraskevakos bereits im Jahr 1973 patentieren [21], er war jedoch mit seiner Idee den damaligen technischen Möglichkeiten voraus und somit konnte sein Konzept nicht in die Realität umgesetzt werden.

Erst 1983, als *Motorola* mit dem *Dynatax 8000x* das erste kommerziell zu erwerbende, einfach zu transportierende Mobiltelefon auf den Markt brachte, existierte eine reelle technische Grundlage für die Realisierung von Mobiltelefonen mit eingebauten Datenverarbeitungs- und Kommunikationssystemen [22].

Langsam entwickelte sich in den folgenden Jahren die mobile Kommunikationstechnik weiter, bis 1992 *IBM* mit „*Simon*“ das erste Gerät entwickelte (Markteinführung 1994), das Paraskevakos' Idee in die Realität umsetzte [23]. Simon war ein Mobiltelefon mit integriertem Kalender, Adressbuch, E-Mail-Software und Spielen. Die Bedienung von Simon erfolgte bereits via Touchscreen. Das Gerät war allerdings wohl mit einem Preis von 1100 US-Dollar ohne Vertragsbindung einfach zu teuer, um einen großen Erfolg auf dem Markt zu haben [24].

Abb. 1: IBM Simon

(Quelle: „http://cdn02.androidauthority.com/wp-content/uploads/2012/11/IBM_Simon.jpeg“)

Das erste derartige Gerät, welches auf dem Markt erfolgreich war, war der 1996 vorgestellte „*Nokia 9000 Communicator*“ [24]. Im Gegensatz zu Simon verzichtet dieser auf einen Touchscreen. Von außen sah das Gerät aus wie ein gewöhnliches Mobiltelefon mit einem Tastenfeld und einem kleinen Display, ein Aufklappen des Geräts offenbarte jedoch den viel größeren Innenbildschirm für Organizer- und E-Mail-Funktionen und eine QWERTY-Tastatur.

Abb. 2: Nokia 9000 Communicator

(Quelle: „http://cdecas.free.fr/computers/pocket/images/nokia9000.jpg“)

Die Integration von Multimedia-Geräten in Mobiltelefone wurde erstmals mit den Modellen *Toshiba Camesse* (erstes Handy mit integrierter Digitalkamera, 1999) [25] und *Samsung SGH-M100* (erstes Handy mit mp3-Player, 2000) [26] realisiert.

2007 führte *Apple* das *iPhone* auf dem Markt ein. Das Mobiltelefon brachte einige Neuerungen mit sich [27]. Beispiele sind die Bedienung mit Multitouch-Gesten und die Verwaltung und Vermarktung von Apps in einem einfach durchsuchbaren „*App-Store*“.

Abb.3: iPhone der ersten Generation

(Quelle: „http://cdn.redmondpie.com/wp-content/uploads/2012/08/iphone-original.jpg“)

2008 wurde als Konkurrenzbetriebssystem zu dem auf den iPhones laufenden iOS das ursprünglich zur Steuerung von Digitalkameras gedachte Android als Betriebssystem für Mobiltelefone auf dem Markt eingeführt. Android entwickelte sich bis zum Jahr 2010 zum Marktführer der mobilen Betriebssysteme [28].

Gegenwärtig sind *gebogene* Smartphones im Kommen. Geräte wie das *Samsung Galaxy Round* oder das *LG G Flex* versuchen durch eine Krümmung die Bedienung ergonomischer zu machen.

3.2. Geschichte von libpd

Als eine signalfluss- und datenstromorientierte Programmiersprache, die dem Anwender das interaktive Verbinden von Objekten zum Zweck des Erstellens unterschiedlichster DSP- und Klangsynthese-Anwendungen ermöglicht, ist *Pure Data* eine hervorragende Möglichkeit für Künstler und Entwickler, ihre Ideen umzusetzen, und wäre eine sehr gute Entwicklungsumgebung für mobile Anwendungen. In seiner ursprünglichen Form ist Pure Data jedoch auf den Standalone-Betrieb auf Linux, Windows und Mac OSX Systemen beschränkt. Diese Einschränkung brachte 2010 einige Pure Data - Entwickler dazu, eine Pure Data API für Android zu schreiben, welche den Startpunkt des Projektes *libpd* markierte [5]. Das eigentliche Ziel der Programmierer war es, eine Pure Data - Library für die native Android-Programmierung in C zu entwickeln, allerdings ermöglichte das damalige Software Development Kit für native Android-Programmierung (*NDK*) nicht den Zugriff auf Audio-Ein- und Ausgänge. Man entschied sich also bei libpd für Android für einen zweiteiligen Aufbau, nämlich einerseits der Low-Level-C-API von libpd und andererseits dem *Java Native Interface (JNI)* als Wrapper für Java.

Aufbauend auf der Low-Level Version von libpd wurden später Wrapper für andere höhere Programmiersprachen wie Objective C und Python und auch eine iOS-Version von libpd entwickelt. Die Multimedia Programmierumgebungen *Processing* und *OpenFrameworks* erhielten ebenfalls eigene libpd-APIs (*PureDataP5* und *ofxPd*).

Seit der Veröffentlichung von libpd wurde es vielfach in nichtkommerziellen und aufgrund der BSD-Lizensierung auch in kommerziellen Projekten als Audiowerkzeug verwendet.

2012 veröffentlichte Peter Brinkmann das Buch „Making Musical Apps – Real-time audio synthesis on Android and iOS" in dem er Grundlagen zum Umgang mit den mobilen libpd-APIs erklärt und Denkanstöße für *Mobile-Music*-Apps gibt [30].

Auf der libpd-Community-Website („*http://libpd.cc/*") werden regelmäßig neue Apps vorgestellt, die mithilfe von libpd erstellt wurden. Es folgen einige Beispiele:

- ***NodeBeat*** (2013, *AffinityBlue*)

NodeBeat ist eine Anwendung für Windows, Mac, Android und iOS, bei der man durch das Erstellen und Verschieben grafischer Elemente auf dem Bildschirm ein Musikstück kreiert. Elemente senden Signale, die bei anderen Elementen bestimmte rhythmische und melodische Klänge erzeugen. Der Quellcode wird mit der kostenlosen Desktop-Version mitgeliefert.

- ***Inception*** (2010, *RjDj & Remote Control Productions*)

Diese an den gleichnamigen Film angelehnte kostenlose *Augmented-Reality*-App für iOS nimmt Umgebungsklänge auf und gibt sie in modifizierter Form wieder. Auch Teile der Filmmusik des Komponisten Hans Zimmer werden wiedergegeben. Der Benutzer, welcher die von der App wiedergegebenen Klänge über Kopfhörer hört, kann zwischen mehreren Szenen wählen, welche die akustische Umgebung auf verschiedene Arten modifizieren.

- ***FRACT OSC*** (2014, *Phosfiend Systems Inc.*)

In diesem Spiel für Windows und Mac hat der Spieler die Aufgabe eine fiktive, vergessene und wüste Welt, die einst aus Musik bestand, wieder aufzubauen. Durch das Lösen von Rätseln können die musikalischen Einzelteile wieder zusammengesetzt werden. Auch eigene Musik kann im Spiel kreiert werdem. *FRACT OSC* kostet 14,99€.

3.3. Vergangene und aktuelle Entwicklungen im Bereich Mobile-Music

Schon vor der Markteinführung der „modernen“ mobilen Betriebssysteme iOS und Android gab es Entwicklungen im Bereich *Mobile-Music*. Erste Unterhaltungsanwendungen auf Mobiltelefonen waren Spiele wie *Snake*, *Pong*, *Tic-Tac-Toe* oder *Tetris* und – bereits zur Zeit der Jahrtausendwende – Klingeltoneditoren. Mithilfe dieser Anwendungen konnte der Nutzer seine eigenen Ruftöne erstellen.

Abb. 4: Klingeltoneditor bei einem Nokia-Handy

(Quelle: „http://www.irkitated.com/2014/08/happened-ringtones.html/nokia-ringtone-composer“)

Als 2008 mit *Xcode 3* die erste IDE zur Entwicklung mobiler Apps für iOS und 2009 das *Android Software Development Kit* erschienen, ergaben sich sowohl für professionelle App-Studios als auch für Hobby-Entwickler kreative Möglichkeiten in etlichen Genres.

Ein weiterer Fortschritt bezüglich der Popularisierung des App-Entwickelns war die Bewegung weg von proprietären Audio- und Video-Libraries, hin zu speziellen Tools wie *openFrameworks*, *Unity*, *kivy*, *Csound* und *libpd* für Android und iOS.

Es folgt anhand des Apple-App-Stores eine kurze Marktanalyse bezüglich *Mobile-Music*-Apps.

Für *Mobile-Music*-Apps jeglicher Art gibt es im Apple-App-Store die Kategorie „Musik“. Allerdings befinden sich weitere *Mobile-Music*-Apps in der Kategorie „Spiele“ in der Unterkategorie „Musik“ und auch in anderen Kategorien finden sich vereinzelt Apps, bei denen Musik oder zumindest Klang eine zentrale Rolle spielt. Aufgrund dieser Verteilung von *Mobile-Music*-Apps über mehrere Kategorien ist eine genaue Marktanalyse anhand von Statistiken, die beispielsweise die Anzahl der Downloads pro Kategorie angeben, zwar nicht möglich, allerdings können durchaus Trends beobachtet werden. Auf eine Differenzierung nach *Mobile-Music*-Kategorien (siehe Kapitel *3.3.1.*) oder kostenlosen und nicht kostenlosen Apps wird hier der Einfachkeit halber verzichtet.

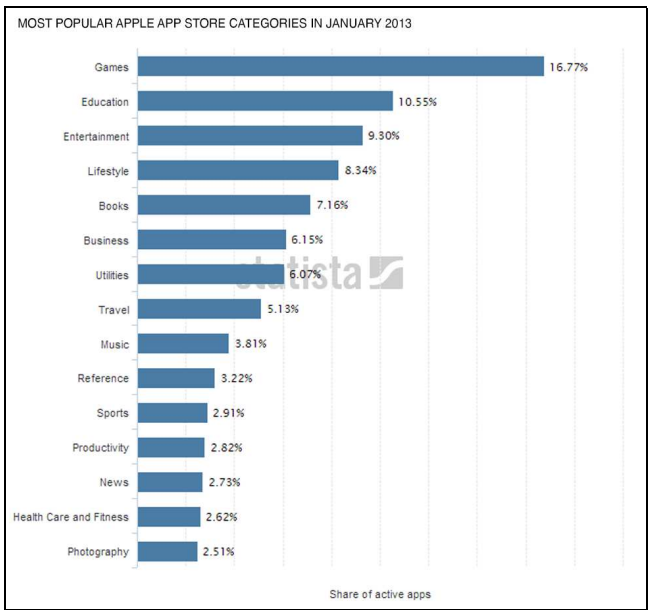

Abb. 5: Anteil der jeweiligen Kategorien an den gesamten Downloads im Apple-App-Store bis Januar 2013

(Quelle: „http://4.bp.blogspot.com/--KKRkUB-JYM/UYoDLfMf8JI/AAAAAAAAAFY/CdLg8Qgl4Rg/s1600/MOST.png“)

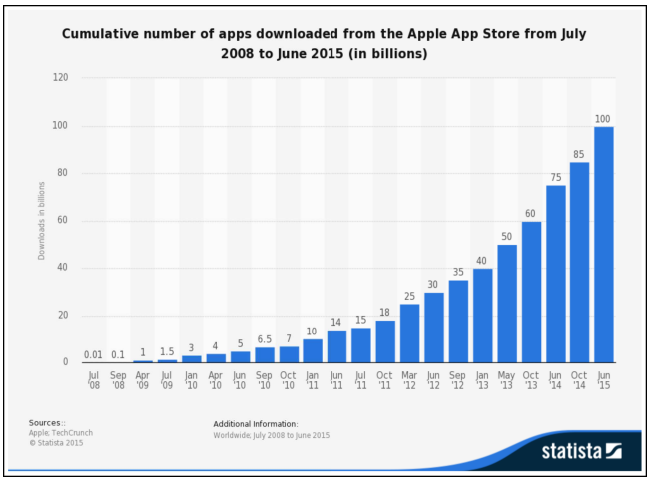

Abb. 7: Anzahl der insgesamt heruntergeladenen vom Apple-App-Store heruntergeladenen Apps von 2008-2015

(Quelle: „http://www.statista.com/graphic/1/263794/number-of-downloads-from-the-apple-app-store.jpg“)

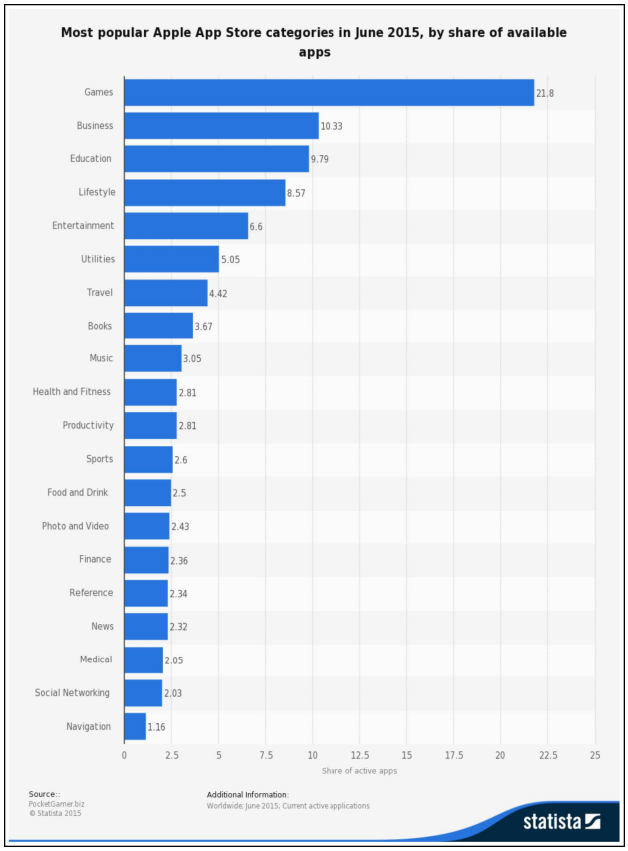

Abb. 6: Anteil der jeweiligen Kategorien an den gesamten Downloads im Apple-App-Store bis Juni 2015

(Quelle: „http://www.statista.com/graphic/1/270291/popular-categories-in-the-app-store.jpg“)

Die Zusammenstellung von Statistiken (Abb. 2, 3 und 4) zeigt folgende Marktsituation:

Während der Anteil von heruntergeladenen Apps der Kategorie „Musik“ im Januar 2013 noch 3,81% betrug, waren es im Juni 2015 nur noch 3,05%. Insgesamt kann jedoch festgehalten werden, dass der Anteil von Multimedia-Anwendungen stark zugenommen hat: Nahmen Spiele 2013 noch 16,77% der Downloads ein, so waren es 2015 21,8%. Über die Entwicklung des Gesamtanteils von *Mobile-Music*-Apps lassen sich daraus zwar keine Schlüsse ziehen, allerdings zeichnet sich ab,

dass musikalische Spiele (siehe Kapitel *3.3.1.*) innerhalb der Kategorie *Mobile-Music* einen zunehmenden Anteil verzeichnen.

Ein absoluter Rückgang der Downloads muss in keiner der App-Store Kategorien befürchtet werden: Wie in Abb. 13 deutlich zu erkennen ist, unterliegt die Zahl der insgesamt heruntergeladenen iOS-Apps einem exponentiellen Wachstum.

3.3.1. Kategorisierung

Diese Unterscheidung, welche aktuelle *Mobile-Music*-Apps in die folgenden 7 grundlegend verschiedenen Kategorien unterteilt, orientiert sich teilweise an der Kategorisierung von Matthias Krebs [8].

- ***Player/Streaming***

Die Idee, einen mp3-Player in ein Mobiltelefon zu integrieren, wurde bereits 2000 mit dem *Samsung SGH-M100* in die Realität umgesetzt. In modernen Smartphone-Betriebssystemen ist die Wiedergabe von Musikdateien standardmäßig möglich, in der Regel sind bereits einfache Player vorinstalliert. Viele Apps machen sich Streaming und Cloud Technologien zu eigen, um dem Benutzer mobilen Zugriff auf große Musikbibliotheken zu ermöglichen. Beispiele hierfür sind die Apps der Streamingportale *Grooveshark* (existiert nicht mehr), *Spotify* und *Soundcloud*. Zudem existieren Apps wie *TuneIn Radio* die dem Benutzer Zugriff auf diverse AM/FM- und Internet-Radiosender bieten.

- ***Music Information Retrieval***

Eine weitere Kategorie von *Mobile-Music*-Apps sind Anwendungen, die dem Benutzer dabei helfen, gehörte Musik mithilfe des Mikrofons des Smartphones zu identifizieren. Die Anwendungen basieren entweder auf Extraktion bestimmter Merkmale (wie Melodieverlauf, Tempo, Text, ...) aus dem aufgenommenen Audiomaterial und Abgleichen (mithilfe von Mustererkennungsalgorithmen) dieser Features mit Datenbänken oder auf direktes Abgleichen des Audiomaterials mit Datenbänken. Bekannte Beispiele für solche Apps sind *Shazam*, *SoundHound* und *TrackID™*. Musikerkennung mit Mobiltelefonen existierte bereits vor dem Smartphone. *Shazam* bot zur Zeit der Jahrtausendwende eine Kurzwahlnummer an, die der Benutzer anrufen konnte und das zu erkennende Musikstück via Telefonleitung an einen Musikerkennungsdienst sendete. Der Benutzer bekam dann eine SMS mit dem Ergebnis der Musikerkennung zugesandt.

- ***Hilfsmittel***

Wie für viele andere Lebensbereiche existieren auch für den Musikbereich unzählige Gadget-Apps. Diese können dem Benutzer beim Üben oder bei Aufführungen behilflich sein. Beispiele hierfür sind Stimmgeräte, Metronome oder Fernsteuerungen wie *TouchOSC* für elektronische Instrumente oder Medien. Bei Aufführungen elektronischer Musik ist es mittlerweile nicht mehr unüblich, dass der Performer bzw. Instrumentalist eine Fixed-Media Begleitung oder einen elektronischen Effekt mit dem Smartphone startet.

- ***Musikproduktion***

Auch die Hersteller von Musikproduktions-Software nutzten die Chance und entwickelten Smartphone-Apps, die es Musikern ermöglichen, unterwegs an ihren elektronsich produzierten Musikprojekten zu arbeiten. Sequencer-Anwendungen wie *Cubasis* und *nTrack Studio* eignen sich insbesondere für Tablets. Spezielle Audiointerfaces wie das *Behringer iStudio IS202* geben Musikern und Toningenieuren die Möglichkeiten einer mobilen Mehrspurmaschine.

- ***Instrumente***

Eigenständige *Mobile-Music*-Instrumente, seien es virtuelle Nachbauten von akustischen oder elektronischen Instrumenten oder Instrumente mit neuartigen Interfaces, welche besonders auf die Eingabemöglichkeiten und Sensoren von mobilen Endnutzergeräten eingehen (Multitouch, Beschleunigungssensor, Gyroskop), sind – wenn man das Stichwort „*Mobile-Music*“ hört – die wohl nächstliegendsten Anwendungen. Andererseits ist diese Kategorie aber auch von den hier genannten sowohl in der U- als auch in der E-Musik die am wenigsten ernst genommene, zu viele der angebotenen Apps sind eher spielzeugartig und zu groß sind demzufolge die Hemmungen bei klassisch-virtuos orientierten Musikern. Beispiele sind die Android-Apps *Otoone* (Multitouch-Synthesizer mit intuitivem Interface) und *Real Guitar* (Gitarren-„Emulator“), sowie die iOS-Apps *Geo Synthesizer* (Multitouch-Synthesizer mit intuitivem Interface) und *Orphion* (perkussives Multitouch-Pad-Instrument). Aufgrund der deutlich niedrigeren Latenz bei iOS können die iOS-Apps in dieser Kategorie im Allgemeinen als „virtuoser“ betrachtet werden.

- ***Musikalische Augmented Reality Apps / Reactive Music***

In diese Kategorie fallen Apps, welche die erzeugten Klänge in Echtzeit an die Umgebung des Benutzers anpassen. Ein App-Studio, das hauptsächlich mit diesem Genre in Verbindung gebracht wird, ist das 2008 vom österreichischen Künstler und Ingenieur Michael Breidenbrücker gegründete RjDj (Reality Jockey Ltd.), welches die gleichnamige App veröffentlichte. Die Nutzer haben in dieser App die Möglichkeit, ein *Reactive-Music*-Album herunterzuladen, die Musik wird dann nach bestimmten Regeln aus dem Mikrofonsignal und verschiedensten Sensordaten berechnet. Weitere Beispiele sind die in Kapitel 3.2 beschriebene App *Inception* und die ähnlich konzipierte App *The Dark Knight Rises Z*.

- ***Musikalische Spiele***

In diese letzte Kategorie fallen jene *Mobile-Music*-Apps, die einen Spiele-Charakter haben, demnach also Spieleapps, bei denen Klang eine zentrale Rolle spielt, zum Beispiel Songquiz-Spiele, bei denen die Spieler ein Lied oder einen Interpreten erkennen müssen (Beispiel sind die iOS-App *Guess The Song Game – Music pop quiz* und die Android-App *SongPop*) und Geschicklichkeitsspiele, bei denen der Spieler „nebenbei“ eine Melodie erzeugt (wie die iOS/Android-App *Piano Tiles (Don't Tap The White Tile)*), aber auch komplexere Apps, wie das oben beschriebene *FractOSC*.

4. SYSTEMBESCHREIBUNG

4.1. Hardware

Im Folgenden werden diejenigen Hardwarekomponenten beschrieben, welche bei den Interfaces der Instrumente in der App *Healthy Sounds* eine zentrale Rolle spielen, nämlich Touchscreen und Beschleunigungssensor.

4.1.1. Touchscreen

Der Touchscreen ist das Standard-Ein- und Ausgabegerät von modernen Smartphones. Er besteht bei Smartphones in der Regel aus folgenden zwei Komponenten: einem Flüssigkristallbildschirm (LCD) zur Anzeige und einem darüber angebrachten Touch-Modul. Dieses Touch-Modul kann auf unterschiedlichen Funktionsprinzipien basieren: Bei Smartphones kommt hierbei die resistive (druckempfindlich, zwei leitfähige Oberflächen werden zusammengedrückt) oder die kapazitive (kein Druck nötig, leitfähiges Material in der Nähe (z.B. Finger) beeinflusst elektrisches Feld) Berührungssensor-Technik zum Einsatz. Der heutzutage am meisten verwendete Ansatz ist der projiziert-kapazitive [30] nach dem Prinzip der Gegenkapazität:

Unter einer Schutzoberfläche sind zwei von einer Isolatorschicht getrennte Schichten des transparenten und halbleitenden Stoffes Indiumzinnoxid (ITO) angeordnet, welche zuvor durch Ätzen zu einem leitfähigen Muster verarbeitet wurden. Das iPhone beispielsweise verwendet hier eine einfache matrixartige Anordnung (Spalten in der einen Schicht, Zeilen in der anderen). Von einem integrierten Schaltkreis werden durch Anlegen einer Spannung die Gegenkapazitäten aller Leiterkombinationen ermittelt. Kommt ein leitfähiger Gegenstand in die Nähe einer Leiterkreuzung, so vermindert er die Gegenkapazität der sich kreuzenden Leiter [31].

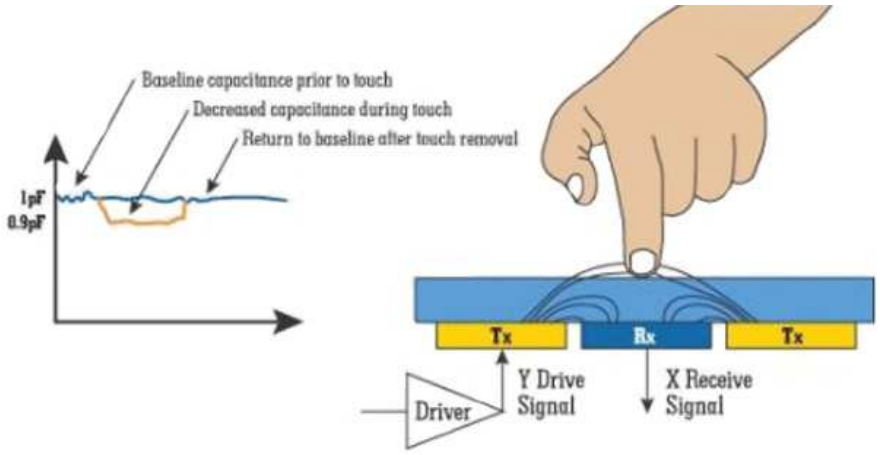

Abb. 8: Funktionspinzip projiziert-kapazitiver Touchscreens

(Quelle: „http://www.digikey.com/es/articles/techzone/2012/sep/from-touch-to-call-tracing-the-path-of-a-touch-gesture“)

4.1.2. Beschleunigungssensor

In praktisch jedem modernen Smartphone ist ein Beschleunigungssensor integriert, der die Hauptaufgabe besitzt, die Ausrichtung des Bildes (Hochformat / Querformat) an die Ausrichtung des Gerätes anzugleichen. Doch auch in vielen Spielen und anderen Unterhaltungsanwendungen findet der Beschleunigungssensor Anwendung.

Die in Smartphones verbauten Accelerometer-Chips besitzen üblicherweise drei Beschleunigungssensoren, welche entsprechend den kartesischen Koordinaten zueinander angeordnet sind. Die Beschleunigungssensoren sind in MEMS (microelectromechanical system) – Bauweise ausgeführt und beruhen alle auf dem Prinzip der Messung der Trägheitskraft auf eine Probemasse. Hierbei kommen unterschiedliche Methoden zum Einsatz.

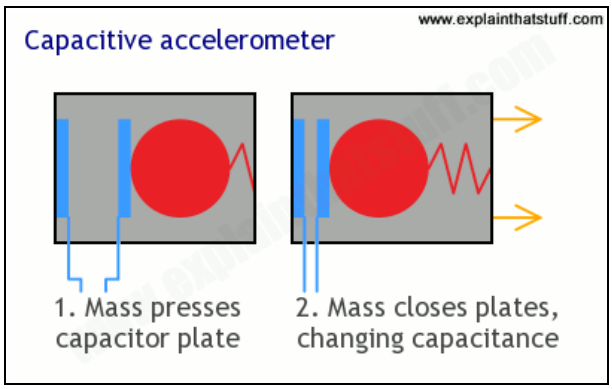

Abb. 9: kapazitive Beschleunigungsmessung

(Quelle: „http://www.explainthatstuff.com/accelerometers.html“)

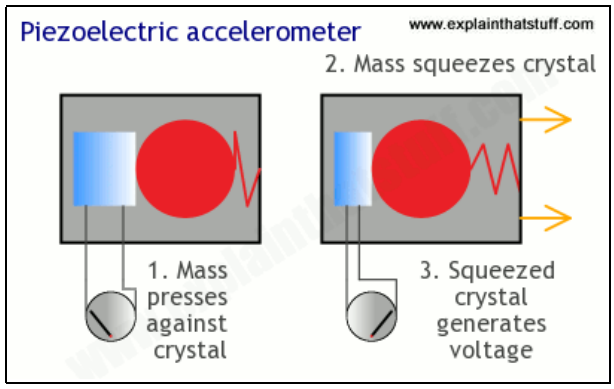

Abb. 10: piezoelektrische Beschleunigungsmessung

(Quelle: „http://www.explainthatstuff.com/accelerometers.html“)

Eine sich bewegende Masse ist an einer Feder aufgehängt, somit führt eine durch Beschleunigung des Sensors hervorgerufene Kraft auf die Masse zu einer von dieser abhängigen Auslenkung [32].

Bei der kapazitiven Methode (siehe Abb. 9) wird dadurch die Kapazität eines Plattenkondensators beeinflusst (Platten werden zueinander hin oder voneinander weg bewegt).

Bei der piezoelektrischen Methode (siehe Abb. 10) wird ein Piezokristall zusammengedrückt, was in einer Spannung zwischen der Ober- und Unterseite des Kristalls resultiert.

In einem integrierten Schaltkreis wird die Spannung bzw. Änderung der Kapazität in einen Beschleunigungswert umgewandelt. Oft enthalten die Accelerometer-Chips auch Mikroprozessoren, welche z.B. direkt die Ausrichtung des Bildschirms berechnen. Die Sensordaten werden üblicherweise mittels des standardisierten I^2C-Bus weitergegeben.

4.2. Software

Bei der Programmierung von *Healthy Sounds* wurden für die Klangsynthese die *Pd*-Library *libpd* und für das Erstellen der grafischen Oberfläche das Grafikwerkzeug *Processing* verwendet.

4.2.1. Tonerzeugung: die *Pd*-Library *libpd*

Bei *libpd* handelt es sich um eine Library, welche es ermöglicht, grafisch programmierte *Pure Data* Patches in unterschiedlichen Programmiersprachen einzubinden.

Pure Data (*Pd*) ist eine grafisch aufgebaute Programmierumgebung, die Programmierern, Künstlern und Sound-Designern die digitale Signalverarbeitung mit Klang erleichtert. Sie wurde in den 1990ern vom Computermusiker Miller Puckette, welcher auch der Erfinder des Pure Data - Vorgängers *Max* war, enwickelt [33].

Im Gegensatz zu Max ist Pd eine quelloffene Software, die ständig von einer offenen Community weiterentwickelt wird.

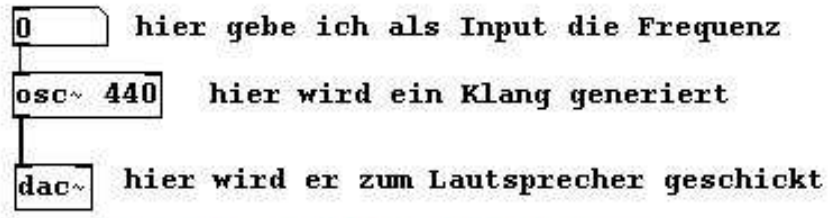

Abb. 11: Einfacher PD-Patch zum Erzeugen eines Sinustones

(Patch aus: „http://www.pd-tutorial.com/german/ch02.html“ [11])

Abb. 11 zeigt einen Beispielpatch und beschreibt die Funktionen der Elemente. In Pure Data gibt es unterschiedliche Elemente (Zahlen, Oszillatoren, Filter, Schieberegler, mathematische Operatoren, …), die der Benutzer durch Ziehen von Verbindungen mit der Maus verknüpfen kann.

Pure Data als datenstromorientierte Programmiersprache besitzt Datenströme auf zwei Ebenen: der Kontroll-Ebene und der Audio-Ebene. Berechnungungen in der Kontroll-Ebene werden beim Empfangen eines Bangs oder einer Message ausgeführt, in der Audio-Ebene hingegen blockweise für jedes Sample. Objektverbindungen zwischen Kontroll-Objekten (im obigen Beispiel die Verbindung zwischen der Zahl und dem [`osc~`]-Objekt) erscheinen als dünne Striche. Objektverbindungen, die in Audiorate aktualisiert werden (im obigen Beispiel die Verbindung zwischen dem [`osc~`]- und dem [`dac~`]-Objekt) erscheinen als dicke Striche [11], [12].

Dem Programmierer ermöglicht nun libpd, einen solchen Patch beispielsweise in ein *Java*-Programm einzubinden. libpd ist als Library für *Java*, *Processing*, *Objective C* und *Python* verfügbar und unterstützt zudem die Betriebssysteme Android und iOS.

Zum Einbinden gibt der Programmierer oder Nutzer den Pfad des Pd-Patches an das Hauptprogramm, welches den Patch lädt und ihn parallel zum Hauptprogramm ausführt. Die Kommunikation

zwischen Hauptprogramm und Pure Data Patch funktioniert mit den Objekten `[send]` und `[receive]` seitens des Pd-Patches (zu verwenden wie beim „Standalone“-Pd) und den Methoden bzw. Funktionen `sendBang()` / `sendFloat()` / `sendList()` usw. und `receiveBang()` / `receiveFloat()` / `receiveList()` usw. Somit können dynamisch Vorgänge getriggert bzw. gesteuert werden.

4.2.2. Grafikwerkzeug: *Processing*

Bei *Processing* handelt es sich um eine einfach erlernbare Programmiersprache für Grafikanwendungen mit dazugehöriger IDE. Die Syntax von Processing ist an die Programmiersprache Java angelehnt, genau genommen handelt es sich bei Processing um eine in Java programmierte Anwendung, die den Java-Wortschatz durch eine Library erweitert.

In Processing erstellt der Programmierer sogenannte Sketches (= Skizzen). Hierbei sind die beiden Methoden `setup()` und `draw()` essentiell [34]. Abb. 12 zeigt einen einfachen Sketch in der Processing-IDE.

Die Methode `setup()` wird zu Beginn des Programms einmal aufgerufen. Hier können zum Beispiel benötigte Objekte initialisiert werden.

Die Methode `draw()` wird pro angezeigtem Frame einmal ausgeführt. Die Grafik wird mit einer Standard-Framerate von 60 fps aktualisiert. Je nach Prozessorauslastung kann diese jedoch erheblich niedriger werden. Falls erwünscht, kann der Anwender jedoch auch manuell programmatisch eine niedrigere Framerate festlegen. In der Methode *draw()* finden sich typischerweise Befehle zum Anzeigen unterschiedlicher Formen, aber auch beispielsweise Filteroperationen.

Die Processing-IDE besitzt seit der Revision 0193 (2011) einen Android-Modus, welcher das Kompilieren von Processing-Sketches für Android-Geräte ermöglicht. Dieser Modus baut auf der Processing-*Android*-*Core*-Library auf (welche beispielsweise in der Processing-Version 2.0b9 enthalten ist). In neueren Processing-Versionen ist der Android-Modus jedoch nicht mehr standardmäßig enthalten.

Das Integrieren dieser Libarary in anderen Android-Code wird vom Processing-Benutzersupport als „nicht triviale Aufgabe“ bezeichnet [35]. Dies hat unter anderem damit zu tun, dass das Verhalten des Sketches nicht mit dem Activity-Lebenszyklus (siehe Kapitel *4.2.3.*) abgestimmt ist.

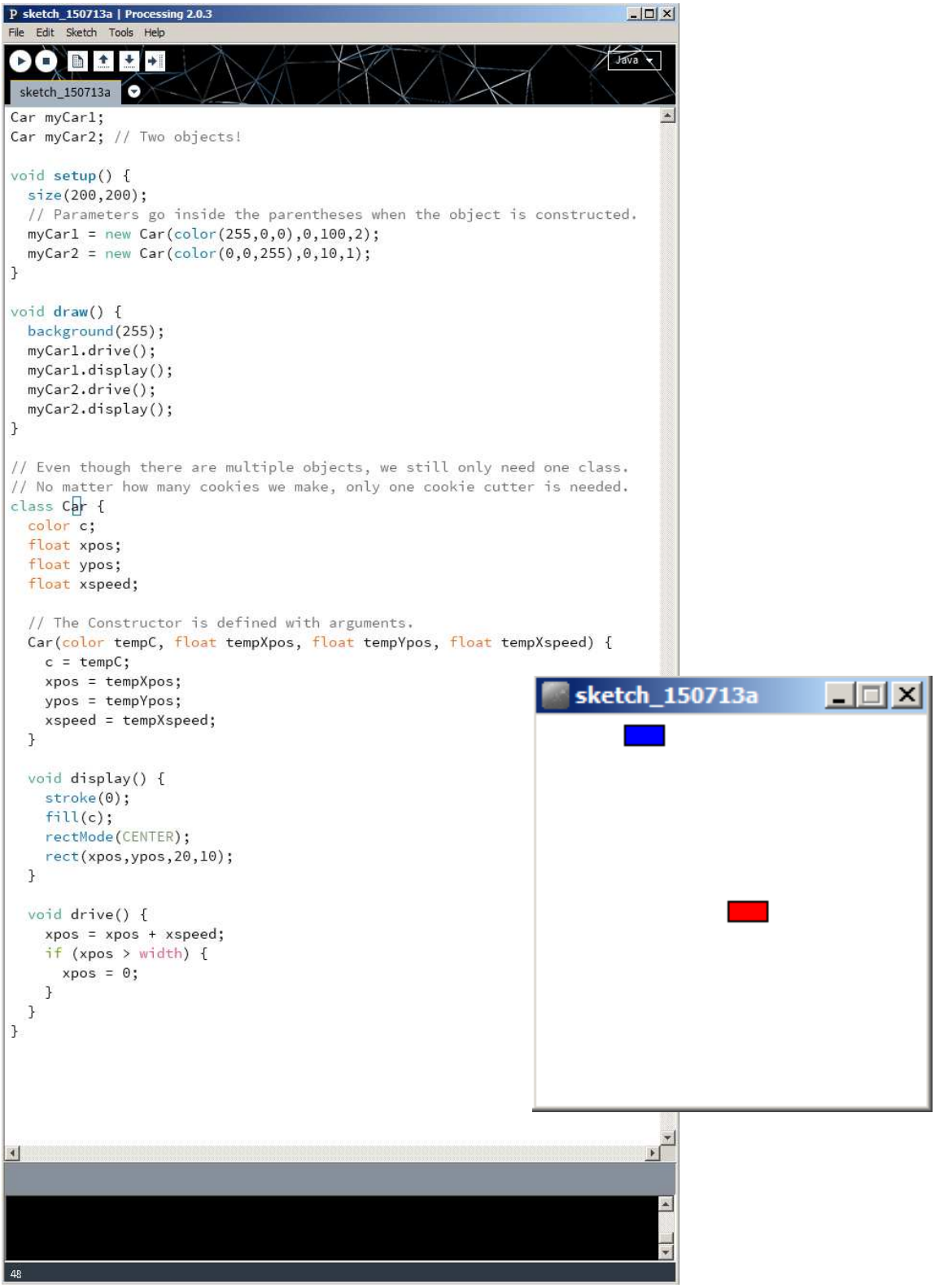

Abb. 12: Einfacher Processing-Sketch: Hier werden die Verwendung der Methoden setup() und draw() und die Möglichkeiten der objektorientierten Programmierung gezeigt. Das Fenster rechts zeigt das grafische Ergebnis: die beiden Rechtecke bewegen sich von links nach rechts durchs Bild, wobei sich das rote Rechteck doppelt so schnell wie das blaue bewegt.

(Sketch aus: „https://processing.org/tutorials/objects/“)

4.2.3. Das Android-Betriebssystem

Da eine ausführliche Beschreibung der Architektur des Android-Betriebssystems für diese Arbeit zu tief ins Detail gehen würde beschäftigt sich dieses Kapitel nur mit der obersten und benutzernächsten Ebene: den *Applications*.

Applications sind – wie der Name bereits sagt – Benutzeranwendungen. Sie greifen (jede unter einer eigenen User-ID und mittels einer eigenen virtuellen Maschine) auf den Linux-Systemkern zu, auf welchem Android basiert [36].

Eine Application kann aus vier Arten von Komponenten bestehen: *Activities* (Bildschirmansichten mit Benutzeroberfläche), *Services* (laufen im Hintergrund und führen dort länger andauernde Operationen (z.B. Abspielen eines Musikstücks) aus), *Content providers* (stellen Daten (z.B. für andere Applications) zur Verfügung) und *Broadcast receivers* (empfangen systemweite „Durchsagen“ und dienen als Kommunikationsmittel zwischen den Anwendungskomponenten).

Für die Implementierung der in dieser Arbeit beschriebenen App werden nur Activities verwendet: Activities sind aktiv, wenn sie im Vordergrund erscheinen (die Benutzeroberfläche sichtbar ist), können allerdings aber auch inaktiv im Hintergrund existieren [37]. Dies geschieht dann, wenn eine Activity mit höherer Priorität in den Vordergrund tritt (z.B. aufgrund eines eingehenden Anrufs). Abb. 13 zeigt den Lebenszyklus einer Instanz der Klasse `android.app.activity` mit den zu bestimmten Zeitpunkten aufgerufenen Methoden. Die Methoden können überschrieben werden, damit beispielsweise Ressourcen zur richtigen Zeit freigegeben werden können.

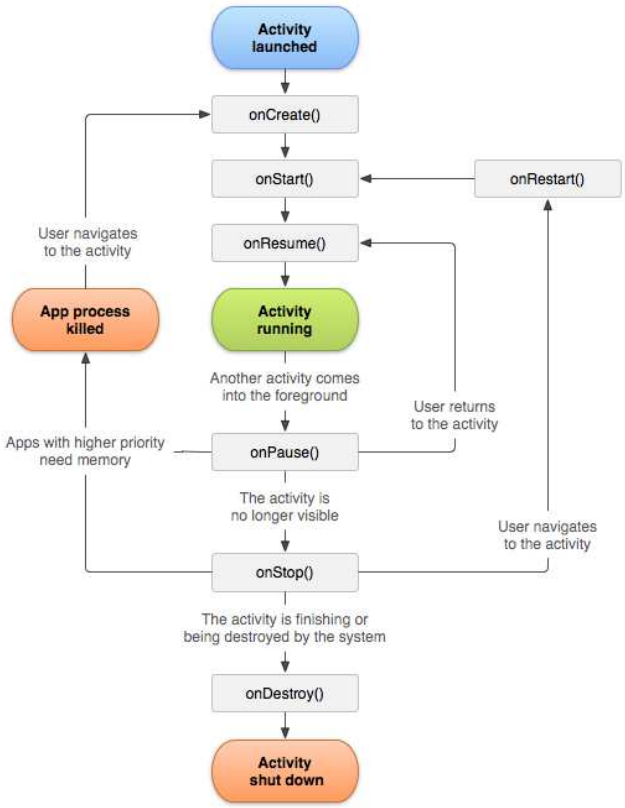

Abb. 13: Lebenszyklus einer Android-Activity

(Quelle: „http://developer.android.com/images/activity_lifecycle.png“)

5. ENTWICKLUNGSPROZESS DER APP „HEALTHY SOUNDS“

5.1. Integration der Programmierschnittstellen

Als IDE wurde *Eclipse* verwendet. Das *ADT (Android Development Tools) - Plugin* kann in Eclipse unter *„Help“ → „Install New Software”* installiert werden, die dafür benötigte URL findet man auf der Android-Entwickler-Website. Mittlerweile bietet Android mit *Android Studio* eine eigene IDE zur Entwicklung von Apps an, früher konnte man auf der Entwickler-Website das sogenannte „*ADT-Bundle*“ herunterladen, eine Eclipse-Version mit vorinstalliertem Plugin.

Nachdem das Plugin in Eclipse installiert wurde, muss nun zumindest eine Version des *Android Software Development Kits (SDK)* heruntergeladen werden. Hierfür kann der *SDK-Manager* verwendet werden *(„Window“ → „Android SDK Manager“*). Das SDK enthält die benötigten Ressourcen, um ein Java-Programm für ein Android-Gerät zu entwickeln und lesbar zu machen.

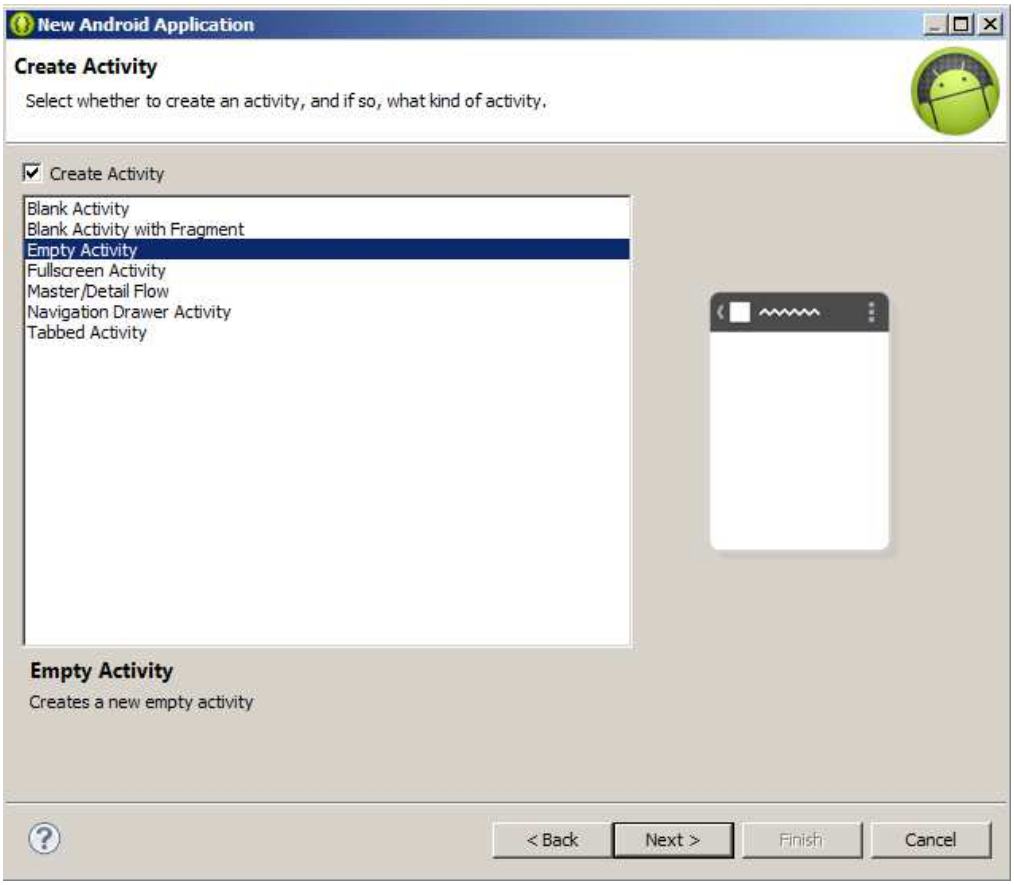

Abb. 14: „Assistent zum Erstellen eines neuen Android-Anwendungsprojektes“

Nun kann eine erste Test-App erstellt werden. Hierfür wählt man *„File“ → „New“ → „Android Application Project“*. Im erscheinenden Dialog kann der Name der App, die Zielversion und die minimal benötigte Version des Betriebssystems und die SDK Version, mit der die App kompiliert wird, gewählt werden. Des weiteren kann eine leere Android-Activity erstellt werden. Wenn dieser

Dialog beendet ist hat man bereits eine kompilier- und ausführbare App, allerdings ohne Inhalt und mit der Testüberschrift „Hello World“.

Im *Project-Navigator* erscheint nun das neu erstellte Projekt mit seiner Ordnerstruktur (siehe Abb. 15)

Im Folgenden wird grob die Ordnerstruktur eines Android-Anwendungsprojekts beschrieben [14]:

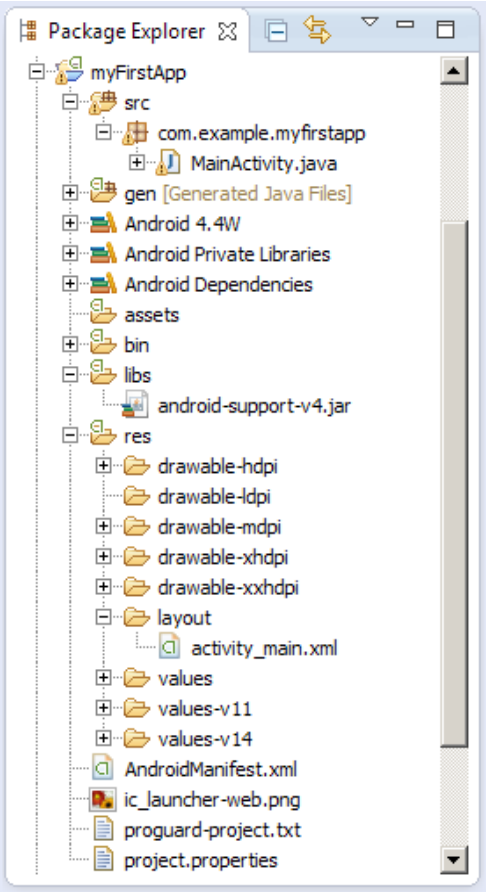

Abb.15: Project Navigator

- ***src* - Ordner**

In diesem Ordner befindet sich der eigentliche Programmcode in Form von java-Dateien. Besteht, die App aus mehreren Activities, so wird pro Activity eine java-Datei erstellt. Auch in separate Dateien ausgelagerte Klassen oder ganze (nicht archivierte) Libraries können in diesem Ordner gespeichert werden.

- ***gen* - Ordner**

Der gen-Ordner enthält die automatisch generierten Dateien *BuildConfig.java* (Konfigurationsdatei) und *R.java* (ermöglicht Zugriff auf verwendete Ressourcen). Diese Dateien sollten nicht manuell verändert werden.

- ***assets* - Ordner**

Hier werden Textdateien und Datenbänke gespeichert. Bilddateien, auf welche das `PApplet`-Objekt (s. u.) mit der `loadImage`-Methode zugreift, sollten ebenfalls hier gespeichert werden.

- ***bin* - Ordner**

Dieser Ordner enthält die beim Kompilieren erzeugte apk-Datei und andere erzeugte Dateien.

- ***libs* - Ordner**

Hier befinden sich üblicherweise die in der App verwendeten Libraries in Form von Java Archiven. Der Ordner enthält standardmäßig auch die Library *android-support-v4.jar* welche die Rückwärtskompatibilität der App gewährleistet.

- ***res* - Ordner**

Benötigte Ressourcen werden in diesem Ordner abgespeichert. Für Bilddateien (wie z.B. Icons) existieren mehrere Unterordner für unterschiedliche Bildschirmauflösungen. Bei der Verwendung von libpd werden die Patches üblicherweise in einem Unterordner namens *raw* im *res*-Verzeichnis gespeichert.

- ***layout* - Ordner**

Dieser Unterordner im *res*-Verzeichnis enthält für Activities, welchen ein bestimmtes Layout zugewiesen wird, jeweils eine xml – Datei. Hier sind die GUI Elemente und ihre Positionen abgespeichert.

- ***AndroidManifest.xml***

In dieser Datei werden der Name der App, Erlaubnisse die man der App bezüglich der Android - Systemressourcen gibt und Eigenschaften der einzelnen Activities definiert.

Nun kann in diese „leere“ App die Grafikumgebung Processing wie folgt integriert werden: Die benötigte Ressource ist das Archiv *android-core.zip*, welches zum Beispiel in der Processing Version 2.0b8 enthalten ist[1]. Das Archiv muss in *android-core.jar* umbenannt werden und in den Eclipse-Workspace (am besten in den *libs*-Ordner des Android-Anwendungsprojektes, in dem die Library zum Einsatz kommen soll) kopiert werden [15]. Nun wählt man nach einem Rechtsklick auf das Projekt im Package-Explorer „*Properties*“ und öffnet dort den Menüpunkt „*Java Build Path*“. Nachdem man unter dem Reiter „*Libraries*“ „*Add JARs…*“ gewählt hat, man zu *android-core.jar* und fügt die Library hinzu.

Der Import „`processing.core.*`“ kann nun in der Quellcodedatei *MainActivity.java* eingefügt werden und die Klasse *MainActivity* ist wie folgt zu ändern:

```
import processing.core.*
public class MainActivity extends PApplet {
  //PApplet in fact extends android.app.Activity
  public void setup() {
    /*...*/
  }
  public void draw() {
    /*...*/
  }
}
```

Codeausschnitt 1: Grundstruktur der Processing-Grundklasse `PApplet`

*(Code aus: „*http://blog.onthewings.net/2013/04/25/setting-up-a-processing-android-project-in-eclipse/*“ [15])*

Die Processing-Grundklasse `PApplet`, deren Eigenschaften von der Klasse `MainActivity` geerbt wird, ist ihrerseits eine Subklasse von `android.app.Activity`.

Die Layout-Datei *activity_main.xml* im Ordner */res/layout* kann nun gelöscht werden, da sie nicht mehr benötigt wird. Wie in einem gewöhnlichen Processing-Sketch kann nun Code in die Methoden `setup()`[2] und `draw()` eingefügt werden.

1: Processing 2.0b8 ist nicht mehr auf der offiziellen Processing-Website verfügbar und bei neueren Versionen wird der Android-Modus nicht mehr mitgeliefert. Der *Android-Core*-Quellcode kann jedoch bei *GitHub* heruntergeladen werden*: „https://github.com/processing/processingandroid/tree/master/core/src/processing“*

2: Die Verwendung der `setup()`-Methode für das Initialisieren von Variablen und Objekten ist in diesem Zusammenhang nicht sehr sinnvoll, da `setup()` nicht nur aufgerufen wird nachdem die Activity gestartet wurde, sondern auch wenn sie z.B. nach einem Telefonanruf wieder in den Vordergrund kommt. Es empfiehlt sich eher, die Initialisierung in der Methode `onCreate(Bundle)` durchzuführen.

Im nächsten Schritt kann libpd für Android integriert werden. Die hierfür benötigten Ressourcen können zusammen mit einigen Beispielprojekten bei *GitHub* heruntergeladen werden.[3] Die beiden Ordner *PdCore* und *AndroidMidi* müssen zum Eclipse-Workspace, in dem sich das Projekt befindet, in welchem man libpd integrieren will, hinzugefügt werden. Optional können auch die Beispielprojekte hinzugefügt werden. *PdCore*, welches seinerseits wiederum *AndroidMidi* verwendet, wird nun wie folgt hinzugefügt. Nach einem Rechtsklick auf das zu erweiternde Projekt wählt man – wie beim Integrieren von Processing – „*Properties*“. Dort öffnet man den Menüpunkt „*Android*“, klickt im „*Library*“-Feld auf „*Add*“ und fügt den Ordner *PdCore* hinzu.

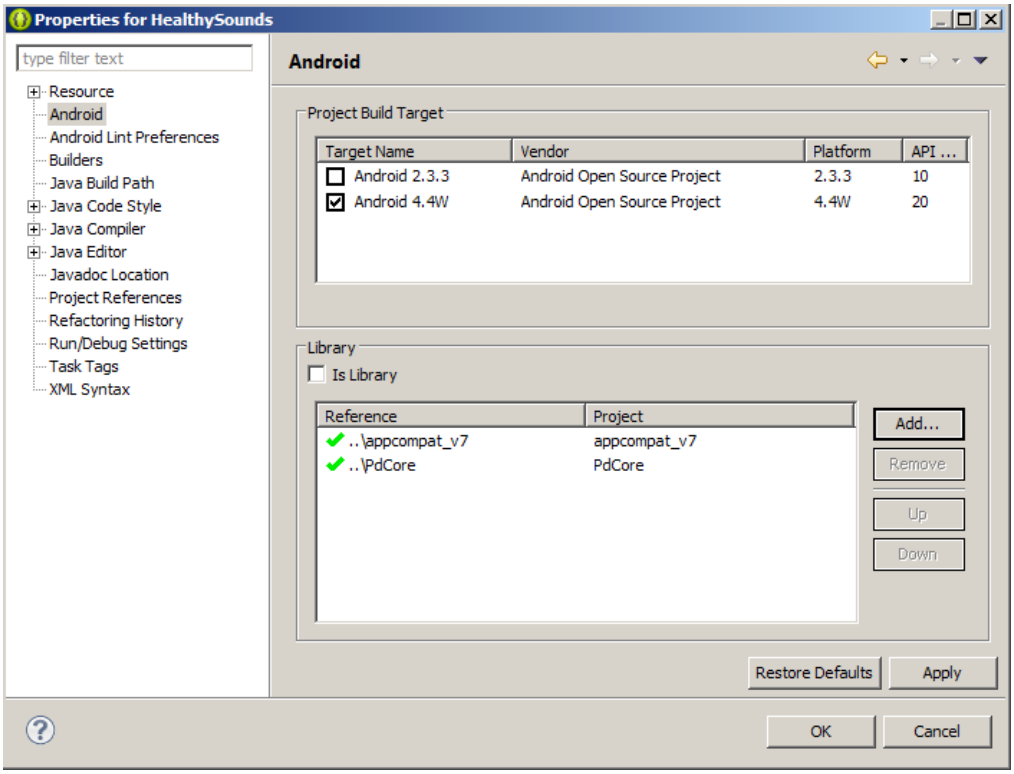

Abb. 16: Hinzufügen von Library - Projekten in den Projekteinstellungen

Wie in Abb. 16 ersichtlich ist, sollte neben *PdCore* ein grünes Häkchen erscheinen.

Zum Öffnen eines Patches und Senden und Empfangen von Nachrichten müssen nun die folgenden Importe vollzogen werden:

```
import org.puredata.android.io.AudioParameters;
import org.puredata.android.io.PdAudio;
import org.puredata.core.PdBase;
import org.puredata.core.PdReceiver;
import org.puredata.core.utils.IoUtils;
```

Codeausschnitt 2: libpd – Importe

Im Verzeichnis */res/raw* werden nun die benötigten Pure Data - Patches komprimiert in einem zip-Archiv abgelegt. Pure Data wird im Programm mit folgenden Methoden initialisiert.

3: „*https://github.com/libpd/pd-for-android*“

```
private void initPd() throws IOException {
  AudioParameters.init(this);
  int srate = Math.max(MIN_SAMPLE_RATE,
      AudioParameters.suggestSampleRate());
  PdAudio.initAudio(srate, 0, 2, TICKS_PER_BUFFER, true);
    File dir = getFilesDir();
  File patchFile = new File(dir, "mainPatch.pd");
  IoUtils.extractZipResource(
      getResources().openRawResource(R.raw.impro_patch), dir, true);
  PdBase.openPatch(patchFile.getAbsolutePath());
  PdBase.setReceiver(receiver);
  PdBase.subscribe("vu");
}

@Override
protected void onStart() {
  super.onStart();
  PdAudio.startAudio(this);
  wakeLock.acquire();
}
```

Codeausschnitt 3: Pure Data – Initialisierung

Die zur Initialisieung von Pd erstellte Hilfsmethode `initPd()` wird in der beim Start der Activity aufgerufenen Methode `onCreate(Bundle)` aufgerufen (Funktionsaufruf nicht im obigen Ausschnitt enthalten). `PdAudio.startAudio(Context)` wird typischerweise in der Methode `onStart()` aufgerufen.

In der Methode `PdAudio.initAudio(int, int, int, int, boolean)` werden die Audiokomponenten mit der vorgegebenen Samplerate, der Anzahl der Audio - Ein- und Ausgangskanäle und der Puffergröße initialisiert.

In `PdBase.openPatch(String)` wird ein Pure Data - Patch in einem gegebenen Pfad geöffnet.

`PdBase.setReceiver(PdReceiver)` und `PdBase.subsribe(String)` ermöglichen das Empfangen von Messages aus dem Pd-Patch, wobei an `PdBase.subscribe(String)` als Parameter der Name des [send] - Objektes verwendet wird, von welchem Messages empfangen werden sollen.

`PdAudio.startAudio(Context)` startet den Pd-Audiostream und muss immer nach `PdAudio.initAudio(int, int, int, int, boolean)` aufgerufen werden.

```
void rub(float x, float y, float x_velocity, float y_velocity) {
  PVector velocity = new PVector(x_velocity, y_velocity);
  if (x > minX && x < maxX && y > minY && y < maxY) {
    if (padShape.get((int) (x - minX), (int) (y - minY)) == WHITE) {
      PdBase.sendList(
          "rub",
          index,
          noteNumber,
          constrain(SENT_VELOCITY_RATIO * velocity.mag(), 0,
          1));
    }
  }
  velocity = null;
}
```

Codeausschnitt 4: Senden von Messages an den Pd-Patch

In Codeausschnitt 4 wird beispielhaft gezeigt, wie mit der Methode `PdBase.sendList(String, Object...)` eine Message mit einer Liste an den Pd-Patch gesendet wird.

Codeausschnitt 5 zeigt, wie Messages vom Pd-Patch (von dem `[send]`-Objekt mit dem zuvor an `PdBase.subsribe(String)` übergebenen Namen) empfangen und verarbeitet werden.

```
private PdReceiver receiver = new PdReceiver() {

  @Override
  public void print(String s) {
  }

  @Override
  public void receiveBang(String source) {
  }

  @Override
  public void receiveFloat(String source, float x) {
  }

  @Override
  public void receiveList(String source, Object... args) {
    if (args.length == 2) {
      if (args[0] != null && args[1] != null) {
        if (args[0].getClass() == Float.class
            && args[1].getClass() == Float.class) {
          lightUp(((Float) args[1]).intValue(),
              ((Float) args[0]).floatValue());
        }
      }
    }
  }

  @Override
  public void receiveMessage(String source, String symbol, Object... args) {
  }

  @Override
  public void receiveSymbol(String source, String symbol) {
  }
};
```

Codeausschnitt 5: Empfangen von Messages vom Pd-Patch

5.2. Freie Improvisation mit metallischem Klang

5.2.1. Überblick

Dieses Instrument in der im Rahmen dieser Arbeit entwickelten App *Healthy Sounds* bietet als Virtualisierung tibetischer Klangschalen die Möglichkeit, sowohl perkussive als auch kontinuierliche Klänge zu erzeugen und ist als Zusammenstellung mehrerer „Pads“ realisiert, welche unterschiedliche Tonhöhen besitzen.

Klangschalen sind aus Metall (meistens Bronze) gefertigte Schalen, die sowohl beim Anschlagen als auch beim Reiben mit den Fingern oder einem Klöppel einen bestimmten Ton abgeben. Sie haben ihren Ursprung im asiatischen Raum und ihre Tradition reicht bis 2000 vor Christus zurück. Damals wurden im Fernen Osten Idiophone aus Jade gefertigt (sog. Klingende Steine), welche beim Anschlagen einen schönen glockigen Klang erzeugten [38]. Ab ca. 600 vor Christus war das Metallhandwerk in Asien so weit fortgeschritten, dass exakt gestimmte metallene Klangschalen hergestellt werden konnten.

Unterschiedliche Stimmungen und Anzahlen an Pads ermöglichen es dem Patienten, seiner aktuellen Gefühlslage Ausdruck zu verleihen. Das Instrument zielt hauptsächlich auf den Einsatz in einer psychotherapeutisch orientierte Musiktherapie ab und kann sowohl in Einzel- als auch in Gruppentherapie gespielt werden.

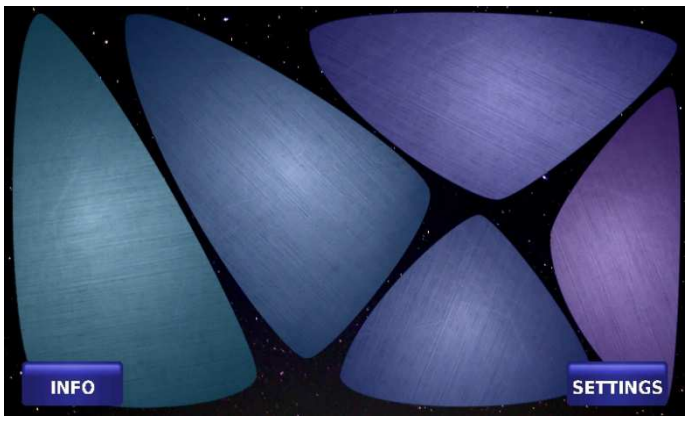

Abb. 17: Pad-Layout (5 Pads, beruhigendes Farbschema)

Die obige Abbildung zeigt das Layout des Instruments. Dieses wird mit den Fingerspitzen auf dem Touchscreen gespielt, durch Streichen über die einzelnen Pads lassen sich kontinuierliche Klänge erzeugen, durch Antippen entstehen perkussive Klänge. Die Pads leuchten auf, wenn sie einen Klang erzeugen. Die Farbe der Pads zeigt die Tonhöhe an. Im „beruhigenden“ Farbschema (siehe Abb. 17) werden tiefe Töne durch hellblau und hohe Töne durch rot-violett repräsentiert, im „anregenden“ Farbschema (siehe Abb. 18) steht orange für tiefe Töne und grün für hohe.

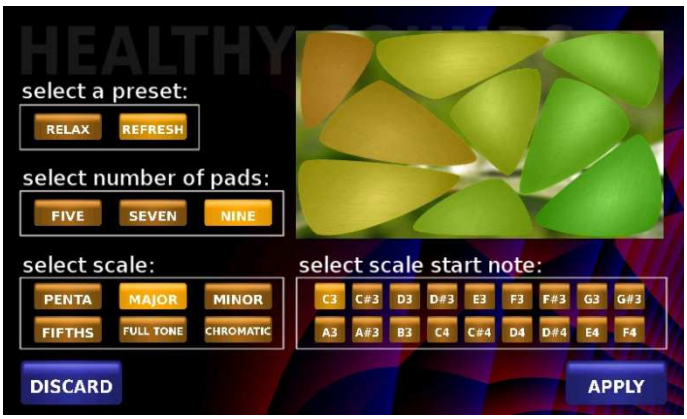

Abb. 18: Einstellungsmenü des Pad-Instruments (9 Pads, anregendes Farbschema)

In Abb. 18 ist das Einstellungsmenü des Instruments zu sehen. Der Benutzer kann zwischen zwei Presets („Relax“ und „Refresh“) wählen. Im „Relax“-Preset werden fünf Pads in einem blau-violetten Farbschema dargestellt, die Tonleiter, auf der dieses Preset basiert, ist eine Mollpentatonik ausgehend vom *c3*. Im „Refresh“-Preset werden neun Pads in einem orange-gelb-grünen Farbschema dargestellt, dieses Preset basiert auf einer Dur-Tonleiter ausgehend vom *g3*. Die Eigenschaften können allerdings auch noch separat verändert werden. Die Wahl der Charakteristika der Presets basiert auf den Erkenntnissen zu ergotroper und trophotroper Musik.

5.2.2. Realisierung

Berührungen und Bewegungen auf dem Touchscreen werden ausgehend von einem bestimmten Pad-Layout detektiert und in Klänge umgesetzt. Auf dem Bildschirm sind entweder fünf, sieben oder neun Pads angeordnet, deren Berühren ein Klangereignis erzeugt. Hierfür wurde die Klasse `Pad` geschrieben. Bei der Initialisierung wird ein Array erstellt, welches alle auf dem Bildschirm angezeigten Pads enthält.

```
for (int i = 0; i < 5; i++) {
  padArray[i] = new Pad(noteArray[i], i + 1);
  padArray[i].drawToBackground();
}
```

Codeausschnitt 6: Erstellen eines Arrays mit 5 Pad-Objekten

Die Form eines Pads wurde als png-Datei wie im folgenden Beispiel definiert:

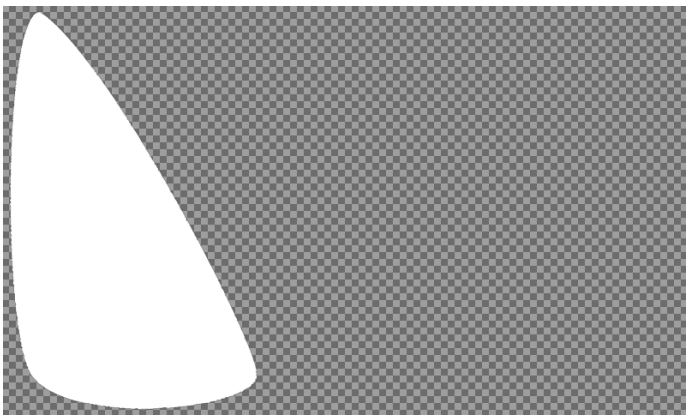

Abb. 19: Definition der Pad-Form in einer png-Bilddatei

Um bei einer Multitouch-Fähigkeit der App die unterschiedlichen Berührungspunkte verwalten zu können wurde die Klasse `Touchpoint` definiert. Bei der Initialisierung wird eine Arrayliste dieses Typs erstellt, die immer alle aktuell aktiven Berührungspunkte enthalten soll.

```
// TOUCHPOINT CLASS:
class Touchpoint {
  float x;
  float px;
  float y;
  float py;
  int id;
  float vx = 0;
  float vy = 0;
  float downTime;

  Touchpoint(float xIn, float yIn, int idIn, float downTimeIn) {
    x = xIn;
    y = yIn;
    id = idIn;
```

```
    downTime = downTimeIn;
  }

  void move(float xIn, float yIn) {
    px = x;
    py = y;
    x = xIn;
    y = yIn;
    vx = x - px;
    vy = y - py;
  }
}
```

Codeausschnitt 7: Klasse „Touchpoint“

```
ArrayList<Touchpoint> Touchpoints = new ArrayList<Touchpoint>();
```

Codeausschnitt 8: Arrayliste für aktive Berührungspunkte

In der Methode `dispatchTouchEvent(MotionEvent)` wird nun auf Berührungen und Bewegungen auf dem Bildschirm zugegriffen.

```
@Override
public boolean dispatchTouchEvent(MotionEvent event) {
  int index = MotionEventCompat.getActionIndex(event);
  // Get the pointer ID
  int mActivePointerId = event.getPointerId(index);

  float x = event.getX(index);
  float y = event.getY(index);

  int action = MotionEventCompat.getActionMasked(event);

  switch (action) {
  case MotionEvent.ACTION_DOWN:
    Touchpoints.add(new Touchpoint(x, y, mActivePointerId, millis()));
    for (int i = 0; i < event.getPointerCount(); i++) {
      int id = event.getPointerId(i);
      for (int j = 0; j < Touchpoints.size(); j++) {
        if (Touchpoints.get(j).id == id) {
          Touchpoints.get(j).move(event.getX(i), event.getY(i));
          rub(Touchpoints.get(j).x, Touchpoints.get(j).y,
              Touchpoints.get(j).vx, Touchpoints.get(j).vy);
        }
      }
    }
    break;
  case MotionEvent.ACTION_MOVE:
    for (int i = 0; i < event.getPointerCount(); i++) {
      int id = event.getPointerId(i);
      for (int j = 0; j < Touchpoints.size(); j++) {
        if (Touchpoints.get(j).id == id) {
          Touchpoints.get(j).move(event.getX(i), event.getY(i));
          rub(Touchpoints.get(j).x, Touchpoints.get(j).y,
              Touchpoints.get(j).vx, Touchpoints.get(j).vy);
        }
      }
```

```
    }
    break;
  case MotionEvent.ACTION_POINTER_DOWN:
    Touchpoints.add(new Touchpoint(x, y, mActivePointerId, millis()));
    break;
  case MotionEvent.ACTION_UP:
    for (int i = 0; i < Touchpoints.size(); i++) {
      if (Touchpoints.get(i).id == mActivePointerId) {
        if (millis() - Touchpoints.get(i).downTime < 90) {
          hit(Touchpoints.get(i).x, Touchpoints.get(i).y);
        }
        Touchpoints.remove(i);
        i--;
      }
    }
  case MotionEvent.ACTION_POINTER_UP:
    for (int i = 0; i < Touchpoints.size(); i++) {
      if (Touchpoints.get(i).id == mActivePointerId) {
        if (millis() - Touchpoints.get(i).downTime < 90) {
          hit(Touchpoints.get(i).x, Touchpoints.get(i).y);
        }
        Touchpoints.remove(i);
        i--;
      }
    }
  }
  return super.dispatchTouchEvent(event);
}
```

Codeausschnitt 9: Umgang mit Berührungen und Bewegungen auf dem Touchscreen

Wie im obigen Codeausschnitt ersichtlich ist, werden die Methoden `rub(float, float, float, float)` und `hit(float, float)` beim Bewegen eines Berührungspunkts bzw. beim Loslassen nach kurzem Tippen aufgerufen. Diese Methoden leiten die Berührungsdaten an die gleichnamigen Instanzmethoden der einzelnen Pads weiter.

```
void rub(float x, float y, float x_velocity, float y_velocity) {
  for (int i = 0; i < padArray.length; i++) {
    padArray[i].rub(x, y, x_velocity, y_velocity);
  }
}

void hit(float x, float y) {
  for (int i = 0; i < padArray.length; i++) {
    padArray[i].hit(x, y);
  }
}
```

Codeausschnitt 10: Weiterleiten der Berührungsinformationen an die Instanzmethoden der `Pad`*-Objekte*

```
void rub(float x, float y, float x_velocity, float y_velocity) {
  PVector velocity = new PVector(x_velocity, y_velocity);
  if (x > minX && x < maxX && y > minY && y < maxY) {
    if (padShape.get((int) (x - minX), (int) (y - minY)) == WHITE) {
      PdBase.sendList(
```

```
          "rub",
          index,
          noteNumber,
          constrain(SENT_VELOCITY_RATIO * velocity.mag(), 0,
          1));
    }
  }
  velocity = null;
}
void hit(float x, float y) {
  if (x > minX && x < maxX && y > minY && y < maxY) {
    if (padShape.get((int) (x - minX), (int) (y - minY)) == WHITE) {
      PdBase.sendList("hit", index, noteNumber);
    }
  }
}
```

Codeausschnitt 11: Überprüfen der Berührungsdaten und ggf. Senden an Pd-Patch

Wie im Codeausschnitt 11 zu sehen ist, wird in diesen Instanzmethoden überprüft, ob sich die übergebene Berührung bzw. Bewegung auf dem nicht transparenten Bereich des oben beschriebenen png-Bildes befindet. Ist dies der Fall, so wird eine Nachricht an den Pd-Patch gesendet, welcher in der folgenden Abbildung dargestellt ist.

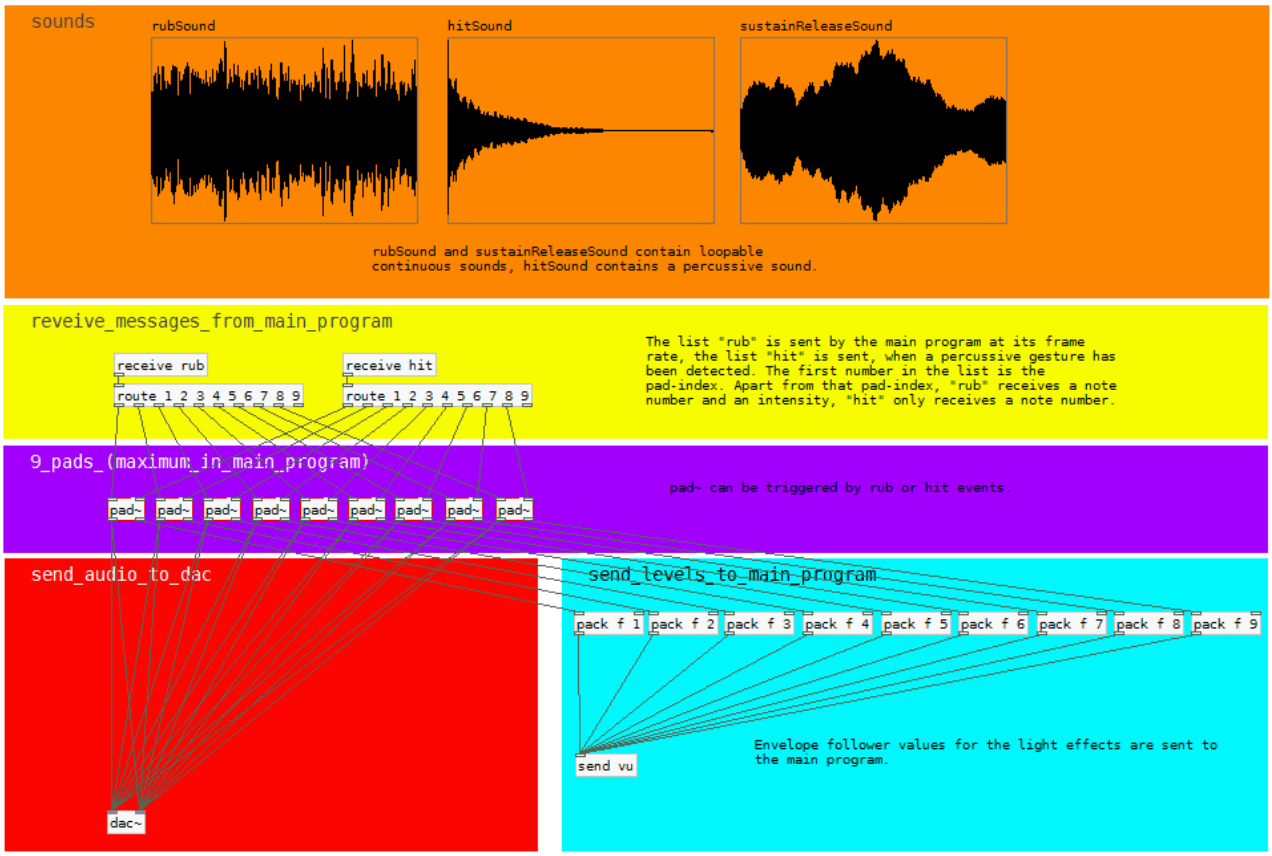

Abb. 20: Pd-Hauptpatch für das Improvisations-Instrument

Die `rub`- und `hit`-Messages werden je nach mitgesendetem Index an eines der neun [pad~]-Objekte weitergegeben, welches in Abb. 21 gezeigt und erklärt wird.

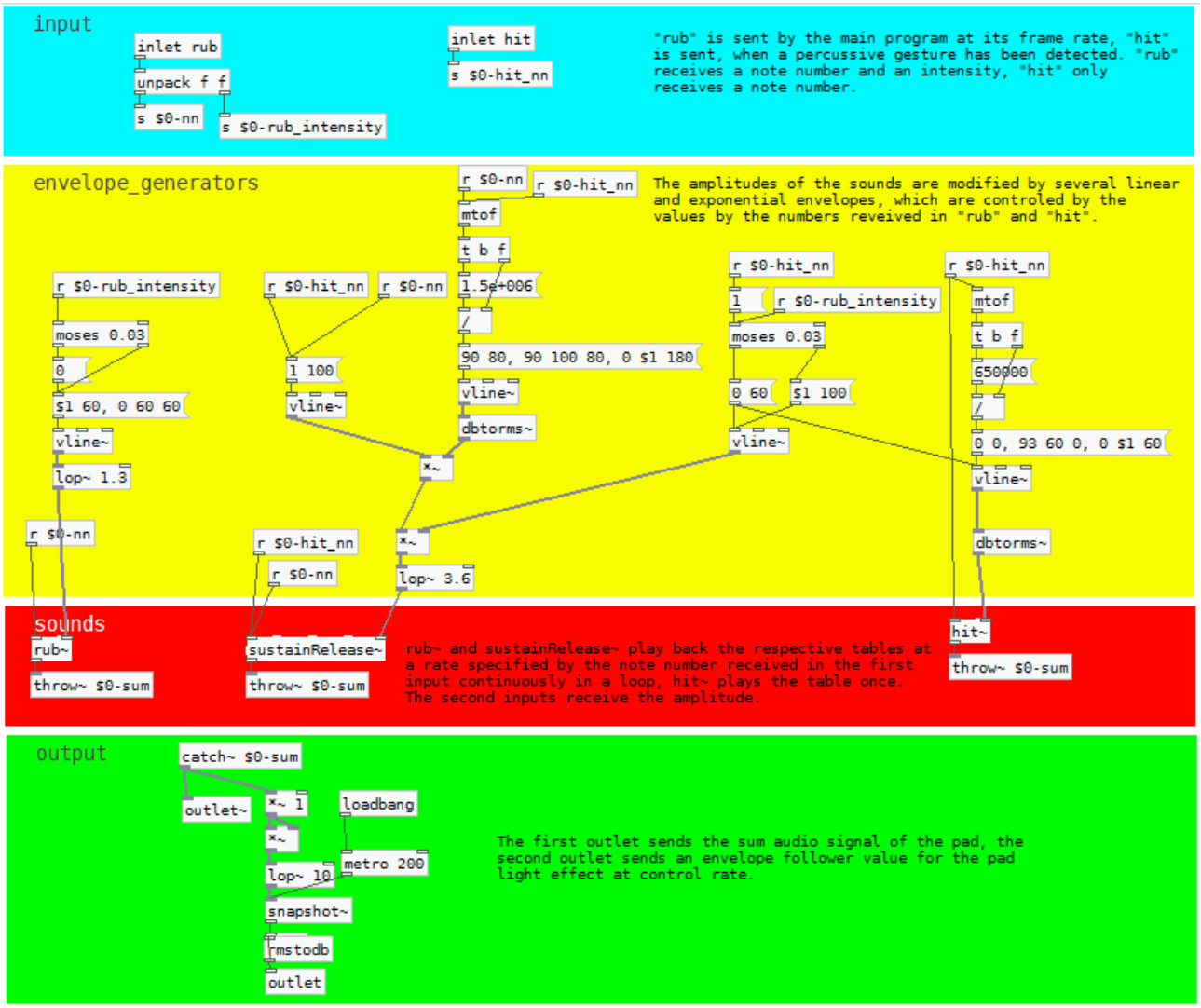

Abb. 21: Pd-Abstraction „Pad“

Zusätzlich zum Audiooutlet besitzt jedes Pad-Objekt ein Outlet auf Kontrollebene, welches alle 200 Millisekunden den Wert eines Envelope-Followers ausgibt. Während das Audiosignal im Hauptpatch an den Digital-Analog-Umsetzer gesendet wird, werden die Werte der Envelope-Followers mit den entsprechenden Pad-Indizes zurück an das Hauptprogramm gesendet, um dort für das Aufleuchten der Pads verarbeitet zu werden.

Für die Realisierung von Knöpfen, welche vor allem im Einstellungsmenü benötigt werden, wurde die Processing-GUI-Library *controlP5* verwendet. Auf Details der GUI-Programmierung geht diese Arbeit allerdings nicht ein.

5.3. Koordination des Handgelenks

5.3.1. Überblick

Dieses Instrument in *Healthy Sounds* gibt dem Patienten visuelles und musikalisch-auditives Feedback zu Bewegungen des Handgelenks, welche indirekt mit dem Beschleunigungssensor des Smartphones erfasst werden, und ist somit als Hilfsmittel einer physio- bzw. ergotherapeutisch orientierten Musiktherapie zu betrachten. Das Ziel ist es, die Bewegungen möglichst geschmeidig auszuführen. Wird eine Bewegung zu ruckartig ausgeführt, erhält der Patient einen audiovisuellen Warnhinweis.

Das Smartphone wird mit einem speziellen Handschuh (siehe Abb. 22) an der Hand des Patienten befestigt. Der Patient führt nun freie oder von der App angeleitete Bewegungen aus und bekommt audiovisuelles Feedback.

Abb. 22: Handschuh zur Befestigung des Smartphones

Bewegungen der Hand erzeugen – ähnlich wie beim Berühren eines Windspiels – zufällige Windspiel-Klänge auf einer pentatonischen Skala. Hierbei modifiziert die Winkelgeschwindigkeit der Bewegung die Klangdichte und -lautstärke. Die Stellung in der dorsal-palmaren Achse modifiziert die Balance (links – rechts) und die Klangfarbe (hölzern – metallisch) und die Stellung in der transversalen Achse den Tonhöhenbereich.

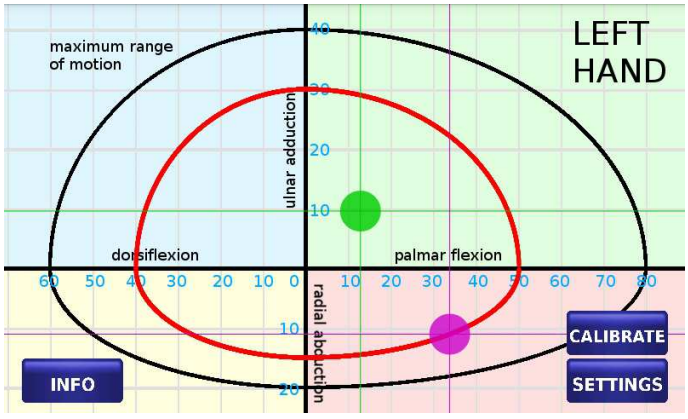

Abb. 23: Anzeige der Handstellung beim Instrument zur Koordination des Handgelenks

Wie in Abb. 23 zu sehen ist, bekommt der Patient durch einen grünen Punkt auf dem Display seines Smartphones, welcher ihm in einem durch die zwei Bewegungsachsen aufgespannten Koordinatensystem die aktuelle Stellung des Handgelenks anzeigt, visuelles Feedback.

Falls erwünscht, kann sich der Patient die Bewegungen von der App anleiten lassen. Hierbei erscheint zu dem grünen Punkt zusätzlich ein sich bewegender violetter, dem der Patient mit dem grünen Punkt, also seiner aktuellen Handstellung folgen soll.

Der rot umrandete Bereich in Abb. 23 ist der (einstellbare) Übungs-Bewegungsbereich. Damit ein Patient mit eingeschränkter Handgelenksfunktion nicht überfordert wird, bewegt sich erstens der violette „Führungs-Punkt“ nur innerhalb dieses Bereichs, und zweitens wird das Klang-Mapping auf diesen Bereich skaliert.

Der oben erwähnte Warnhinweis wird dann gegeben, wenn die Ableitung der Winkelgeschwindigkeit (also die Winkelbeschleunigung) einen gewissen Schwellwert überschreitet, welcher von der (einstellbaren) Empfindlichkeit abhängig ist. Er besteht aus einem auditiven Warnsignal (ähnlich wie dem bei nicht angelegtem Sicherheitsgurt in neueren Autos), zusammen mit einem rötlichen Aufleuchten des Bildschirms.

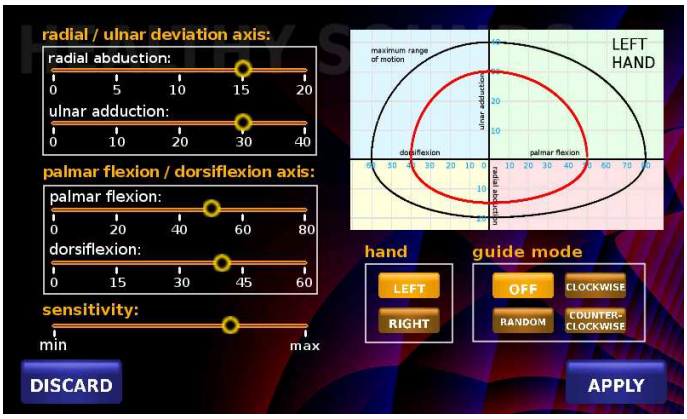

Abb. 24: Einstellungsmenü des Instruments zur Koordination des Handgelenks

Abb. 24 zeigt das Einstellungsmenü dieses Instruments. Der Übungs-Bewegungsbereich, die Empfindlichkeit auf ruckartige Bewegungen, die zu therapierende Hand und der Führungsmodus können eingestellt werden.

5.3.2. Realisierung

a) Mechanik

Bei der Therapie zeigt die Daumenseite der Hand, auf welcher mit einer speziellen Konstruktion das Smartphone (im Querformat) befestigt wird (siehe Abb. 22), zum Patienten. Der Unterarm ist fixiert (hierfür können in der Praxis spezielle Therapieauflagen verwendet werden) und der Winkel zwischen Unterarm und der Horizontalen beträgt mindestens 45°.

Bei einer Betrachtung der Position des Smartphones in Luftfahrtwinkeln (Gier, Nick und Roll, Rotation in dieser Reihenfolge) entspricht dann – wenn man die negative z-Achse des Beschleunigungssensor-Koordinatensystems (siehe Abb. 25) als Nick-Achse und die negative y-Achse als Roll-Achse betrachtet und keine Aus- und Einwärtsdrehung des Unterarms stattfindet – eine dorsal-palmare Bewegung einer Nick-Bewegung und eine transversale Bewegung einer Roll-Bewegung.

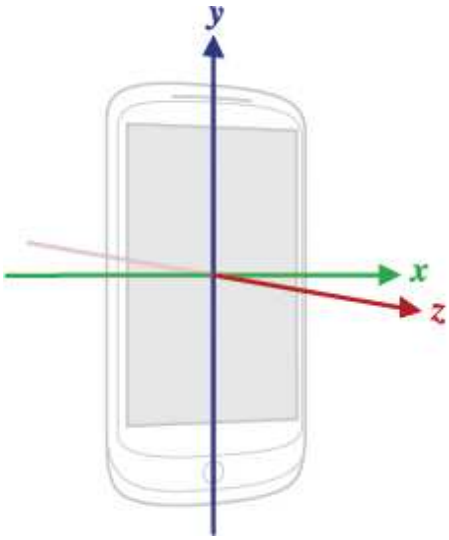

Abb. 25: Koordinatensystem des Beschleunigungssensors (Android-Konvention)

(Quelle: „http://developer.android.com/reference/android/hardware/SensorEvent.html“)

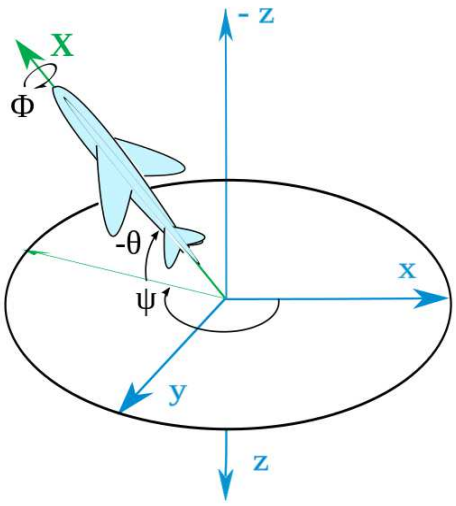

Abb. 26: Luftfahrtwinkel Gier (Ψ), Nick (θ) und Roll (Φ)

(Quelle: „https://en.wikipedia.org/wiki/File:Plane.svg“)

Geht man davon aus, dass keine anderen Beschleunigungen als die Erdbeschleunigung auf das Smartphone wirken und dass weder die Nick-, noch die Roll-Achse in Richtung des Schwerefeldvektors der Erde zeigt, so können der Nick- und der Roll-Winkel aus den Beschleuigungssensordaten wie folgt berechnet werden (die Winkel werden als Linksdrehungen um die jeweiligen Achsen angegeben; wenn sowohl Nick- als auch Roll-Winkel null sind, zeige die negative x-Achse in Richtung des Schwerefeldvektors):

$$\theta = \arctan\left(\frac{y}{-x}\right) \qquad (1),$$

$\theta ... Nick-Winkel,\ x, y ... Beschleunigungskomponenten$

$$\Phi = \arctan\left(\frac{z}{\sqrt{x^2+y^2}}\right) \qquad (2),$$

$\Phi ... Roll-Winkel,\ x, y, z ... Beschleunigungskomponenten$

Ist die Nick-Achse parallel zum Schwerefeldvektor (Rollwinkel nahe bei +/- 90°), so können Nick-Bewegungen nicht mehr erfasst werden. Das Smartphone sollte deshalb nie horizontal ausgerichtet sein. Die oben angegebenen mindestens 45° Neigung des Unterarms ergeben sich aus einer maximalen Ulnarabduktion von 40° und einem „Sicherheitsabstandswinkel“ von 5°. Ausgehend von der Neigung des Smartphones bei Grundstellung wird für die Berechnung der transversalen Stellung eine Kalibrierung durchgeführt.

Anzumerken an diese Methode zur Ermittlung der Handstellung ist, dass bei Mischformen von dorsal-palmaren und transversalen Bewegungen die obige Rotationsreihenfolge gewählt wird, weil sich die Rotationsachse transversaler Bewegungen (verläuft durch das Kopfbein) näher an den Fingern befindet als die Rotationsachse dorsal-palmarer Bewegungen (verläuft durch das Mondbein) und deshalb angenommen wird, dass erstere bei dorsal-palmaren Bewegungen mitbewegt wird.

Für die Befestigung des Smartphones an der Hand wurde aus Styropor und Holz eine Passform gebaut, welche zwischen Zeigefinger und Daumen aufliegt und an deren Oberseite ein Saugnapf befestigt ist (siehe Abb. 27). Diese wurde dann mit Stoff verkleidet und an einen Fahrradhandschuh angenäht (siehe Abb. 28).

Abb. 27: Passform mit Saugnapf

Abb. 28: Fahrradhandschuh mit Passform

b) Programmierung

Essentiell im Hauptprogramm dieses Instruments ist die recht umfangreiche `draw()`-Methode, welche pro Frame einmal ausgeführt wird. Die folgenden Codeausschnitte sind alle dieser Methode entnommen.

```
float x = ax;
float y = ay;
float z = az;
abductionAdduction = atan(-z / sqrt(pow(x, 2) + pow(y, 2)));
abductionAdduction = degrees(abductionAdduction);
if (abs(abductionAdduction) < 87){
  flexionExtension = atan(y / x);
  flexionExtension = degrees(flexionExtension);
}

smoothedAbductionAdduction += (abductionAdduction - smoothedAbductionAdduction)
    / (SMOOTHING_POS / (timeSinceLastSample));
correctedAbductionAdduction = smoothedAbductionAdduction
    - abductionAdductionCorrection;
smoothedFlexionExtension += (flexionExtension - smoothedFlexionExtension)
    / (SMOOTHING_POS / (timeSinceLastSample));

vAbductionAdduction = (smoothedAbductionAdduction - lastSmoothedAbductionAdduction)
    / (millis() - lastFrame);
vFlexionExtension = (smoothedFlexionExtension - lastSmoothedFlexionExtension)
    / (millis() - lastFrame);

absV = sqrt(sq(vAbductionAdduction) + sq(vFlexionExtension));
absA = abs(absV - lastAbsV) / (millis() - lastFrame);
smoothedAbsA += (absA - smoothedAbsA)
    / (SMOOTHING_DIFF / (millis() - lastFrame));
```

Codeausschnitt 12: Berechnung der Handstellung

Aus den Daten des Beschleunigungssensors (in den Variablen `ax`, `ay` und `az` enthalten) wird – wie in Codeausschnitt 12 zu erkennen ist – die Handstellung (die Variablen `abductionAdduction` und `flexionExtension`) berechnet. Die Vorzeichen im Code weichen von denen in (1) und (2), ab, weil für die weiteren Berechnungen ein anderer Drehungssinn benötigt wird. Die Daten werden durch Tiefpassfilteroperationen erster Ordnung mithilfe der Variable `lastFrame`, welche den Zeitpunkt des letzten Frames enthält, geglättet. Auch die Winkelgeschwindigkeit (`absV`) und die Winkelbeschleunigung (`absA`) werden berechnet.

```
float distanceSpeed = constrain((sq(correctedAbductionAdduction
    - guideAbductionAdduction) + sq(smoothedFlexionExtension
    - guideFlexionExtension)), DISTANCE_SPEED_LOWER_LIMIT,
    DISTANCE_SPEED_UPPER_LIMIT);
tau += timeSinceLastSample / distanceSpeed;
float r = 1 - (abs((GUIDE_CIRCUMDUCTION_SPEED * tau)
    % (GUIDE_CIRCUMDUCTION_ROUNDS_PER_SPIRAL * TWO_PI)
    - (0.5f * GUIDE_CIRCUMDUCTION_ROUNDS_PER_SPIRAL * TWO_PI)) / (0.5f *
      GUIDE_CIRCUMDUCTION_ROUNDS_PER_SPIRAL * TWO_PI));
```

Codeausschnitt 13: Berechnung der Geschwindigkeit des Führungspunktes und des Radius' für kreisförmige Führungsbewegungen

Der obige Codeausschnitt zeigt: Die Bewegungsgeschwindigkeit des Führungspunktes für angeleitete Handbewegungen berechnet sich anhand der Differenz zwischen Soll-Stellung und Ist-Stellung der Hand. Kann der Patient dem Führungspunkt gut folgen, so bewegt sich dieser schneller. Für kreis- bzw. ellipsenförmige Bewegungen wird der Radius abwechselnd größer und kleiner.

```
switch (guideMode) {
case RANDOM:
  guideFlexionExtension = sin(GUIDE_RANDOM_FE_SPEED * tau);
  guideAbductionAdduction = constrain(
      sin(GUIDE_RANDOM_AA_SPEED * tau),
      -sqrt(1 - sq(guideFlexionExtension)),
      +sqrt(1 - sq(guideFlexionExtension)));
  break;
case CLOCKWISE:
  guideFlexionExtension = r * sin(-tau * GUIDE_CIRCUMDUCTION_SPEED);
  guideAbductionAdduction = constrain(r
      * cos(-tau * GUIDE_CIRCUMDUCTION_SPEED),
      -sqrt(1 - sq(guideFlexionExtension)),
      +sqrt(1 - sq(guideFlexionExtension)));
  break;
case COUNTER_CLOCKWISE:
  guideFlexionExtension = r * sin(tau * GUIDE_CIRCUMDUCTION_SPEED);
  guideAbductionAdduction = constrain(r
      * cos(tau * GUIDE_CIRCUMDUCTION_SPEED),
      -sqrt(1 - sq(guideFlexionExtension)),
      +sqrt(1 - sq(guideFlexionExtension)));
  break;
}
```

Codeausschnitt 14: Berechnung der Position des Führungspunktes

In Codeausschnitt 14 wird je nach Führungsmodus die aktuelle Position des Führungspunktes berechnet.

```
if (hand == LEFT) {
  strokeWeight(1);
  stroke(CURRENT_POS_RED, CURRENT_POS_GREEN, CURRENT_POS_BLUE);
  line(0, COORDINATE_SYSTEM_CENTER_Y * height + RU_SCALE
  * correctedAbductionAdduction * height, width,
  COORDINATE_SYSTEM_CENTER_Y * height + RU_SCALE
  * correctedAbductionAdduction * height);
  line(COORDINATE_SYSTEM_CENTER_X_LEFT * width + DP_SCALE
  * smoothedFlexionExtension * width, 0,
  COORDINATE_SYSTEM_CENTER_X_LEFT * width + DP_SCALE
  * smoothedFlexionExtension * width, height);
  strokeWeight(POINT_SIZE * width);
  stroke(CURRENT_POS_RED, CURRENT_POS_GREEN, CURRENT_POS_BLUE,
  CURRENT_POS_ALPHA);
  point(COORDINATE_SYSTEM_CENTER_X_LEFT * width + DP_SCALE
  * smoothedFlexionExtension * width,
  COORDINATE_SYSTEM_CENTER_Y * height + RU_SCALE
  * correctedAbductionAdduction * height);

  if (guideMode != OFF) {
    if (guideAbductionAdduction < 0)
      guideAbductionAdduction *= ua;
    else
```

```
      guideAbductionAdduction *= ra;

    if (guideFlexionExtension < 0)
      guideFlexionExtension *= df;
    else
      guideFlexionExtension *= pf;

    strokeWeight(1);
    stroke(GUIDE_RED, GUIDE_GREEN, GUIDE_BLUE);
    line(0, COORDINATE_SYSTEM_CENTER_Y * height + RU_SCALE
        * guideAbductionAdduction * height, width,
        COORDINATE_SYSTEM_CENTER_Y * height + RU_SCALE
        * guideAbductionAdduction * height);
    line(COORDINATE_SYSTEM_CENTER_X_LEFT * width + DP_SCALE
        * guideFlexionExtension * width, 0,
        COORDINATE_SYSTEM_CENTER_X_LEFT * width + DP_SCALE
        * guideFlexionExtension * width, height);
    strokeWeight(POINT_SIZE * width);
    stroke(GUIDE_RED, GUIDE_GREEN, GUIDE_BLUE, GUIDE_ALPHA);
    point(COORDINATE_SYSTEM_CENTER_X_LEFT * width + DP_SCALE
        * guideFlexionExtension * width,
        COORDINATE_SYSTEM_CENTER_Y * height + RU_SCALE
        * guideAbductionAdduction * height);
  }

  PdBase.sendList(
      "sensor",
      constrain(absV * SENT_VELOCITY_RATIO, 0,
      SENT_VELOCITY_UPPER_LIMIT),
      constrain((smoothedFlexionExtension + df) / (pf + df), 0, 1),
      constrain((correctedAbductionAdduction + ua) / (ua + ra),
      0, 1));
}
```

Codeausschnitt 15: Visuelles Feedback und Senden der Bewegungsdaten an den Pd-Patch

Nun wird die aktuelle Handstellung durch einen Punkt auf dem Bildschirm dargestellt. Falls die Bewegungsanleitung nicht deaktiviert ist, wird auch der Führungspunkt angezeigt. Hierfür wird dessen Position, welche zunächst im Einheitskreis berechnet wurde, mit den eingestellten maximalen Übungsauslenkungen (`pf`, `df`, `ra` und `ua`) skaliert. Sowohl die aktuelle Handstellung, als auch die Winkelgeschwindigkeit wird an den Pd-Patch gesendet. Diese Werte werden mit den eingestellten maximalen Übungsauslenkungen normalisiert. Der obige Codeausschnitt zeigt das Vorgehen, falls das linke Handgelenk therapiert wird. Bei der rechten Hand sind die Variablen `pf` und `df` vertauscht.

```
if (smoothedAbsA * WARN_ANGLE_ACCELERATION_RATIO >
        WARN_ANGLE_ACCELERATION_UPPER_THRESHOLD
        - sensitivity
        * (WARN_ANGLE_ACCELERATION_UPPER_THRESHOLD -
        WARN_ANGLE_ACCELERATION_LOWER_THRESHOLD)
        && millis() - warningTime > WARN_REPEAT_TIME) {
  warningTime = millis();
  PdBase.sendBang("warn");
}

if (millis() - warningTime < WARN_LIGHT_UP_TIME) {
  noStroke();
```

```
  fill(WARN_RED,
       WARN_GREEN,
       WARN_BLUE,
       WARN_MAXIMUM_ALPHA
       - WARN_MAXIMUM_ALPHA
       * (sq(millis() - warningTime
       - (0.5f * WARN_LIGHT_UP_TIME)) / sq(0.5f * WARN_LIGHT_UP_TIME)));
  rect(0, 0, width, height);
}
```

Codeausschnitt 16: Überprüfung der Winkelbeschleunigung und ggf. Senden eines Bangs an den Pd-Patch und Färben/Aufleuchten des Bildschirms

Falls die Handbewegungen des Patienten zu ruckartig sind, wird dem Patienten ein Warnhinweis gegeben. Hierfür wird – wie in Codeausschnitt 16 zu sehen ist – überprüft, ob die Winkelbeschleunigung über einem von der eingestellten Empfindlichkeit abhängigen Schwellwert liegt. Ist dies der Fall, so wird ein Bang an den Pd-Patch gesendet und der Bildschirm leuchtet in einer zuvor eingestellten Farbe (in diesem Fall rot) auf.

Die folgende Abbildung zeigt den Pd-Patch, in welchem bei diesem Instrument die Klangsynthese stattfindet:

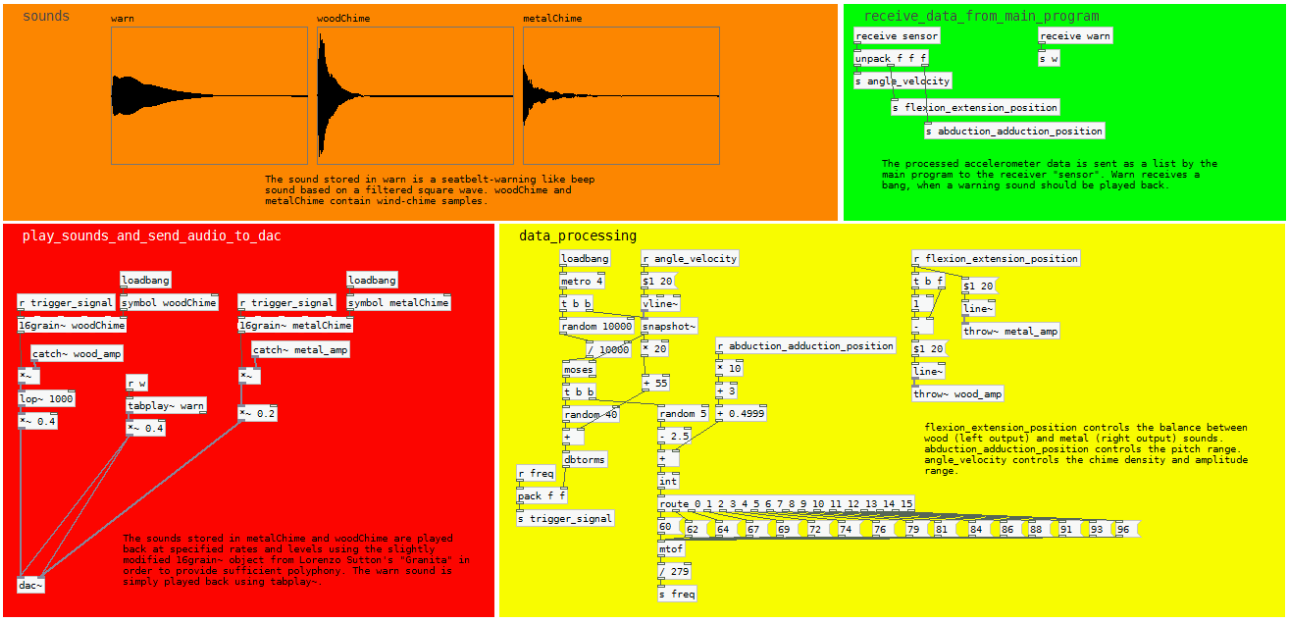

Abb. 29: Pd-Hauptpatch für das Instrument zur Koordination des Handgelenks

In zufälligem Zeitabstand werden Bangs erzeugt, die das Abspielen der Tabellen `woodChime` und `metalChime`, welche Windspiel-Klänge enthalten, bewirken. Die Winkelgeschwindigkeit der Handbewegungen bestimmt hierbei die Dichte, also die Häufigkeit der Bangs.

Die Grundfrequenz des abgespielten Klanges, also die Abspielgeschwindigkeit, ist zufällig, wobei die Frequenzen Tönen auf einer pentatonischen Skala entsprechen. Der Tonhöhenbereich der zufälligen Töne wird hierbei von der Stellung der Hand in der transversalen Achse bestimmt.

Die hölzernen Klänge werden an den linken Ausgangskanal gesendet, die metallischen an den rechten. Die Stellung in der dorsal-palmaren Achse bestimmt die Balance zwischen links und rechts.

6. SCHLUSSFOLGERUNG

Im Seminar „Computermusik und Multimedia 02“ wurden Grundlagen der Klangsynthese in *Pure Data* wiederholt und die mit der Library *libpd* erstellte Pd-Reader-App *PdDroidParty* behandelt. Einfache Interaktionsmöglichkeiten mit der klassischen „Smartphone-Hardware“ (Beschleunigungssensor, Touchscreen, Mikrofon) wurden aufgezeigt.

Mithilfe von libpd wurde eine App für den musiktherapeutischen Einsatz entwickelt. Aufgrund der Anforderungen des Projektes an Menüführung, Einstellungsmöglichkeiten und grafische Oberfläche wurde zusätzlich die Grafikumgebung *Processing* integriert.

Das Integrieren der Programmierschnittstellen in ein Android-Anwendungsprojekt in Eclipse erwies sich als relativ unkompliziert und war - wie eine Internetrechereche ergab - bereits erprobt und in Tutorials erklärt.

Bei der Programmierung traten einige Schwierigkeiten auf:

Ein Problem mit libpd war, dass bei Android aufgrund der Architektur des Betriebssystems grundsätzlich Latenzprobleme bei der Klangerzeugung und Klangverarbeitung bestehen. Die Pufferzeit muss richtig eingestellt werden, damit dem Smartphone nicht zu viel Rechenleistung abverlangt wird.

Viel problematischer als die Programmierung in Pure Data und Konfiguration von libpd erwies sich allerdings das Arbeiten mit Processing als Grafikwerkzeug für die Android-App. Da Processing in erster Linie kein Werkzeug zum einfachen Erstellen von grafischen Benutzeroberflächen ist, musste auf die Library *controlP5* zurückgegriffen werden, die das Integrieren von GUI-Elementen etwas erleichterte. Trotzdem erwies sich das Platzieren der Schaltflächen als langwierige Arbeit. Der im Android SDK enthaltenen Layout-Designer hätte diese Arbeit um einiges erleichtert, konnte allerdings nicht in Kombination mit Processing verwendet werden. Zudem ist Processing als Java-Library ein relativ hardwareentferntes Grafikwerkzeug und damit recht langsam. Grafiken werden als Bitmaps im Arbeitsspeicher gespeichert, was das Erstellen einer grafischen Benutzeroberfläche computerseitig zu einer recht speicherintensiven Aufgabe macht. Bei „unvorsichtiger“ Programmierung riskiert man somit schnell einen „Out of Memory“ - Fehler.

Für die Aufgaben, die die App an ein Grafikwerkzeug stellt, ist Processing gerade noch ausreichend, durch sinnvolle Programmierung und effizientes Speichermanagement läuft *Healthy Sounds* auch auf dem relativ leistungsschwachen *Huawei Ascend Y330* mit kleineren Bugs relativ stabil.

Die verwendete Softwareumgebung ist trotz der negativen Aspekte eine sinnvolle Lösung für das Entwerfen von Prototypen oder für kleinere Projekte. Pure Data ist durch die datenstromorientierte grafische Programmieroberfläche leicht zu verstehen und auch die libpd-Android-API ist nachdem man sich ein wenig mit den Beispielprojekten auseinandergesetzt hat einfach zu verstehen. Hinsichtlich der Klangerzeugung und Klangverarbeitung sind die Probleme (Latenz) wohl eher dem Betriebssystem Android an sich geschuldet, als libpd, welches insgesamt ein sehr nützliches und leistungsstarkes Werkzeug ist. Der Vorteil von Processing besteht in der sehr leicht zu verstehenden Syntax: Methoden wie `rect(float, float, float, float)`, `image(PImage, float, float, float, float)` und `line(float, float, float, float)` sind quasi selbsterklärend. Auch die Modifikation und Verarbeitung der Variablen pro Frame in der Methode `draw()` ist ein sehr naheliegendes und einfach zu verstehendes Konzept.

Die mit libpd, Processing und controlP5 entwickelten Instrumente in *Healthy Sounds* sind bereit für einen Praxistest bezüglich ihrer Akzeptanz und Wirksamkeit als Therapiemittel.

Vor einem solchen Test sollten eventuell in Vortests bzw. Gesprächen mit Experten noch einzelne Parameter feinabgestimmt werden.

Zudem sind die Instrumente in *Healthy Sounds* (ggf. nach einigen auf einem Praxistest aufbauenden Verbesserungen) gute Beispiele für *Mobile-Music*-Apps, welche in einer Studie bezüglich der *grundsätzlichen* Anforderungen und Wünsche von Therapeuten und Patienten an derartige elektronische Hilfsmittel zum Einsatz kommen könnten.

Ergebnisse solcher Studien treiben die Entwicklungen im Bereich elektronischer Therapiehilfsmittel weiter voran und können somit die Qualität und Wirksamkeit von Musik- bzw. Ergo- und Physiotherapie verbessern.

7. LITERATURVERZEICHNIS

[1] H.-H. Decker-Voigt und W. Eckhard, Lexikon Musiktherapie, Hogrefe Verlag, 2009.

[2] N. Arner, „Investigation of the use of Multi-Touch Gestures in Music Interaction," Masterarbeit an der Universität York, Abteilung für Elektronik, 2013.

[3] M. Eriksson und R. Bresin, „Improving Running Mechanics by Use of Interactive Sonification," in *Proceedings of ISon (3rd Interactive Sonification Workshop)*, Stockholm, 2010.

[4] A. Godbout et al., „Mobile Sonification for Athletes: A Case Study in Commercialization of Sonification," in *Proceedings of the 20th International Conference on Auditory Display (ICAD2014)*, New York, 2014.

[5] P. Brinkmann et al., „Embedding Pure Data with libpd," in *Proceedings of the Pure Data Convention*, Weimar, 2011.

[6] W. L. Magee, „Electronic technologies in clinical music therapy: a survey of practice and attitudes," *Technology and Disability,* Bd. 18, Nr. 3, pp. 139-146, 2006.

[7] W. L. Magee und K. Burland, „An Exploratory Study of the Use of Electronic Music Technologies in Clinical Music Therapy," *Nordic Journal of Music Therapy,* Bd. 17, Nr. 2, pp. 124-141, 2008.

[8] M. Krebs, „apps4music - praxiserprobte Möglichkeiten mit Apps zum Selbst-Musikmachen im Unterricht (Präsentationsfolien)," in *Fachtagung: „Medienkompetenz verbindet"*, Berlin, 2014.

[9] G. Aumüller et al., Duale Reihe Anatomie, Thieme, 2014

[10] P. Salvia et al., „Analysis of helical axes, pivot and envelope in active wrist circumduction, " *Clinical Biomechanics* Bd. 15, Nr. 2, pp. 103-111, 2000.

[11] J. Kreidler, Programmierung elektronischer Musik in Pd, 2009

[12] Floss Manuals, „Pure Data," 2005-2012. [Online]. Verfügbar auf: http://en.flossmanuals.net/pure-data/. [Zugriff am 8. August 2015].

[13] Google Inc., „Develop Apps," Android Developers. [Online]. Verfügbar auf: http://developer.android.com/develop/index.html. [Zugriff am 8. August 2015].

[14] Derek Banas, „Android Development Tutorial," YouTube, 2013. [Online]. Verfügbar auf: https://www.youtube.com/watch?v=Z149x12sXsw. [Zugriff am 8. August 2015].

[15] Andy Li, "Setting up a Processing Android project in Eclipse," 2013. [Online]. Verfügbar auf: http://blog.onthewings.net/2013/04/25/setting-up-a-processing-android-project-in-eclipse/. [Zugriff am 8. August 2015].

[16] Deutsche Musiktherapeutische Gesellschaft, „Deutsche Musiktherapeutische Gesellschaft: Definition," 2015. [Online]. Verfügbar auf: http://www.musiktherapie.de/index.php?id=18. [Zugriff am 8. August 2015].

[17] R. Spintge et al., Schriftenreihe zur Gesundheitsanalyse: Musik im Gesundheitswesen – Bedeutung und Möglichkeiten musikmedizinischer und musiktherapeutischer Ansätze, Bd. 47, Schwäbisch Gmünd: GEK - Gmünder ErsatzKasse, 2007.

[18] H.-H. Decker-Voigt und E. Weymann, Aus der Seele gespielt, Goldmann, 2000.

[19] Z. O'Connor, „Coulor Psychology and Colour Therapy: Caveat Emptor," *Color Research & Application,* Bd. 36, Nr. 3, pp. 229-234, Juni 2011.

[20] P. Valdez und A. Mehrabian, „Effects of Color on Emotions," *Jounal of Experimental Psychology: General,* Bd. 123, Nr. 4, pp. 394-409, 1994.

[21] A. Joseph, „Smartphone lawsuits how we all lose," Examiner.com, 2012. [Online]. Verfügbar auf: http://www.examiner.com/article/smartphone-lawsuits-how-we-all-lose. [Zugriff am 8. August 2015].

[22] tk/jd/dpa, „Der Urahn der Handys: Motorola Dynatac 8000X," FOCUS Online, 2008. [Online]. Verfügbar auf: http://www.focus.de/digital/handy/mobilfunkgeschichte/tid-10733/der-urahn-der-handys-motorola-dynatac-8000x_aid_310544.html. [Zugriff am 8. August 2015].

[23] I. Sager, „Before IPhone and Android Came Simon, the First Smartphone," Bloomberg Business, 2012. [Online]. Verfügbar auf: http://www.bloomberg.com/bw/articles/2012-06-29/before-iphone-and-android-came-simon-the-first-smartphone. [Zugriff am 24. Juni 2015].

[24] T. Martin, „The evolution of the smartphone," Pocketnow, 2014. [Online]. Verfügbar auf: http://pocketnow.com/2014/07/28/the-evolution-of-the-smartphone. [Zugriff am 8. August 2015].

[25] D. Steimels, „Wie alles begann: Die Geschichte des Smartphones," Pc-Welt, 2012. [Online]. Verfügbar auf: http://www.pcwelt.de/ratgeber/Handy-Historie-Wie-alles-begann-Die-Geschichte-des-Smartphones-5882848.html. [Zugriff am 8. August 2015].

[26] C. Klaß, „Samsung SGH-M100 - Handy und MP3-Player in einem," 2000, Golem.de, [Online]. Verfügbar auf: http://www.golem.de/0008/9219.html. [Zugriff am 8. August 2015].

[27] D. Frommer, „10 Ways The iPhone Changed Smartphones Forever," Business Insider, 2009. [Online]. Verfügbar auf: http://www.businessinsider.com/10-ways-the-iphone-changed-smartphones-forever-2009-6?op=1&IR=T. [Zugriff am 8. August 2015].

[28] M. Isaac, „Android OS Now World Leading Smartphone Platform," Wired, 2011. [Online]. Verfügbar auf: http://www.wired.com/2011/01/android-os-leading-smartphone/. [Zugriff am 8. August 2015].

[29] P. Brinkman, „Making Musical Apps – Real-time audio synthesis on Android and iOS," O'Reilly, 2012.

[30] Business Research Company, „The Touch Screen Market Globally 2015", Newswire, 2015. [Online]. Verfügbar auf: http://www.newswire.com/press-release/the-touch-screen-market-globally-2015. [Zugriff am 8. August 2015].

[31] S. Kolokowsky und T. Davis, „From Touch to Call: Tracing the Path of a Touch Gesture," Digi-Key Electronics, 2012. [Online]. Verfügbar auf: http://www.digikey.com/es/articles/techzone/2012/sep/from-touch-to-call-tracing-the-path-of-a-touch-gesture. [Zugriff am 8. August 2015].

[32] C. Woodford, „Accelerometers“, Explain that Stuff, 2015. [Online]. Verfügbar auf: http://www.explainthatstuff.com/accelerometers.html. [Zugriff am 8. August 2015].

[33] K. Stallman, „A Conversation with Miller Puckette: 2008 SEAMUS Award Recipient,“ *Society for Electro-Acoustic Music in the United States Newsletter,* Juniausgabe, Nr. 2, pp. 5-9, 2008.

[34] B. Fry, „Processing Overview,“ Processing, 2008. [Online]. Verfügbar auf: https://processing.org/tutorials/overview/. [Zugriff am 24. Juni 2015].

[35] U. Khan et al., „Integrating with other Android code,“ GitHub, 2014 – 2015. [Online]. Verfügbar auf: https://github.com/processing/processing-android/wiki#Integrating_with_other_Android_code. [Zugriff am 8. August 2015].

[36] Google Inc., „Appliction Fundamentals“, Android Developers. [Online]. Verfügbar auf http://developer.android.com/guide/components/fundamentals.html. [Zugriff am 8. August 2015].

[37] Google Inc., „Activities“, Android Developers. [Online]. Verfügbar auf: http://developer.android.com/guide/components/activities.html. [Zugriff am 8. August 2015].

[38] K. Humphries, „Healing Sound: Contemporary Methods for Tibetan Singing Bowls,“ *Undergraduate Library Research Award,* 2010.

8. ERKLÄRUNG

Ich erkläre, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

Graz, am 19. August 2015 .....

(Unterschrift)