Kollegium Spiritus Sanctus Brig

Maturaarbeit 2012/13

Entwicklung von Android Applikationen für Menschen mit geistigen und körperlichen Behinderungen

Von:

Ringeisen Simon, 5D

Eingereicht im **Fachbereich Informatik**

Betreut durch:

Britsch Bernhard

Brig, den 22. Januar 2013

Nummer 2012/75

Bemerkungen

Datenschutz

Der Datenschutz ist im Umgang mit Personen mit Behinderungen sehr ernst zu nehmen, da diese oft nicht selber entscheiden können, wo ihre Daten verwendet werden dürfen. Aus diesem Grund werden in meiner gesamten Arbeit Namen durch zufällig ausgewählte Grossbuchstaben ersetzt. Für Personen, welche genannt wurden, ist im Anhang eine Erlaubnis angefügt. Diese wurde bei nicht mündigen Personen durch den Vormund unterschrieben.

Optimierung für das Entwicklungsgerät

Um die Programmierarbeiten nicht weiter zu verkomplizieren wurden die Applikation jeweils auf die Ausstattung (Bildschirmauflösung, Softwareversion,…) des Testgerätes, dem Nexus 10 [1] von Samsung und Google, angepasst. Eine Unterstützung von anderen Geräten ist im Moment nicht notwendig, da auch die Testphase in der Tagesstätte der ‚insieme Oberwallis'[2] in Visp mit diesen Geräten durchgeführt werden wird. Es ist deshalb möglich, dass die Applikationen auf anderen Geräten nicht oder nur eingeschränkt lauffähig sind.

[1] Information zum Gerät: http://www.google.com/nexus/10/ (19.01.2013)
[2] Weitere Informationen zum Verein: http://insieme-oberwallis.ch

Inhaltsverzeichnis

Vorwort

Ich habe mir schon sehr früh Gedanken über meine Maturaarbeit gemacht. Von Beginn an war für mich klar, dass ich eine praktische Arbeit im Bereich der Informatik machen will. Da ich mich zu dieser Zeit viel mit Android auseinandergesetzt habe und von der ganzen Umgebung begeistert war und immer noch bin, lag es nahe, dass ich eine Android Applikation entwickeln wollte. Ganz nach dem Motto „Fortunately, you can easily create components that look and behave in any way you like, limited perhaps only by your imagination“ [3] lag mir damit ein riesiges Feld zu Füssen.

Für mich war aber auch klar, dass ich kein Produkt entwerfen wollte, welches nach der Maturaarbeit in einer Schublade verschwindet und verstaubt. Zu diesem Zeitpunkt kamen deshalb hauptsächlich zwei Themen für mich in Frage: die Entwicklung einer App für das Kollegium Spiritus Sanctus oder die Entwicklung einer App für behinderte Menschen. Nach einigen Gesprächen mit meinen Eltern, mit Angehörigen und Betreuern von behinderten Personen und meinem zukünftigen Betreuer war mir klar, dass ich mich für letzteres entscheiden würde. Das aus diversen Gründen:

Angehörige wie auch Betreuer waren begeistert zu hören, dass sich jemand für dieses Thema interessiert. Viele von ihnen nutzen privat selber Tablets und Smartphones und sahen natürlich auch Möglichkeiten, diese für die Betreuten einzusetzen. Aber gerade in den Betreuungsinstitutionen gab es noch sehr wenige Produkte, welche auch eingesetzt werden konnten. Das lag einerseits am immensen Aufwand, welche einige Geräte mit sich brachten und andererseits an den enorm hohen Anschaffungskosten. Ein Sprachcomputer etwa kostet einen fünfstelligen Betrag, einfache Tastaturen mittlere dreistellige Beträge. Der Gedanke, dass ein Produkt, welches für den Massenmarkt gedacht ist, nur einen Bruchteil dieser Summe kostet, ist erschreckend.

Als ich in der Tagesstätte der ‚insieme Oberwallis‘ in Visp zu Besuch war, konnte ich mir selber ein genaueres Bild machen. In der Gruppe, bestehend aus ca. 10 behinderten Personen, gab es einen Betreuten, der einen Sprachcomputer besitzt. Der Computer stand am Ladekabel angeschlossen auf einem Regal. U.[4], dem das Gerät gehört, kann selber nur eingeschränkt sprechen. Das Gerät nutzte er während meinem Besuch aber nicht, die Betreuerin erklärte, dass es zu schwer für ihn sein, er könne es nur benutzen, wenn er es auf einem Tisch abstellen kann. Gegen Ende meines Besuches hat die Gruppe begonnen, ein Tagebuch auszufüllen. Früher wurde jeweils von der Betreuerin ein kurzer Text verfasst, welcher jeweils an die Eltern gerichtet war. Seit kurzem konnten die behinderten Betreuten ihre Tagebücher selber gestalten. Aus einer Kiste mit vielen Piktogrammen konnten sie sich einige aussuchen, diese wurden selbstständig oder mit der Hilfe von einer Betreuerin in eine Agenda geklebt. Somit konnten die behinderten Personen selber in ihrem Tagebuch lesen.

Bis zum Beginn der Arbeit hatte ich mir viele Gedanken darüber gemacht, was ich genau programmieren wollte. Ich wollte nicht einfach ein anderes Produkt imitieren und billiger verkaufen, ich wollte eine andere Richtung einschlagen als die meisten erhältlichen Produkte. Ich wollte, dass die behinderten Personen Freude an der Nutzung der App und die Betreuer dank der App eine Entlastung haben. So habe ich mich entschieden, das Tagebuch, welches sie ohnehin schon führen, digital umzusetzen. Es sollte aber neben den einfachen

[3] Android Open Source Project, Custom Components - Fully Customized Components, 2013
[4] Vollständiger Name darf nicht angegeben werden, siehe Datenschutz

Mitteln wie Zeichnen auch für Tablets spezifische Elemente beinhalten. Dazu habe ich etwa eine für behinderte Personen einfach zu benutzende Kamera entwickelt.

Damit meine Arbeit auch einen analytischen Teil besitzt, habe ich noch eine 2. Applikation entwickelt. Diese Applikation habe ich zu Beginn der Arbeit entwickelt. Ich hatte die Möglichkeit sie regelmässig mit behinderten Personen zu testen.

Einleitung

Die Fortschritte, welche in den letzten Jahren im Bereich der mobilen Geräte gemacht wurden, sind immens. Diese Fortschritte bleiben Nutzern, welche ausserhalb des Massenmarktes stehen, verwehrt. Das liegt oftmals nicht am verkleinerten Markt, sondern daran, dass die Hersteller das Rad immerzu neu erfinden wollen. Anstatt die Software von einem bereits am Massenmarkt erhältlichen Gerät so zu optimieren, dass sie für die speziellen Bedürfnisse angepasst ist, wird oft Hard- und Software komplett neu entwickelt. Das wirkt sich stark auf die Qualität und den Preis dieser Systeme aus. Um diesem Trend entgegenzuwirken, wird bereits auf dem Massenmarkt verfügbare Hard- und Software benutzt. Als Software Plattform wurde Android gewählt, da verschiedene Geräte zur Verfügung stehe (darunter auch Schlag- und Wasserfeste) und auch die Software relativ einfach angepasst werden kann.

Android

Entstehungsgeschichte

Im Oktober 2003 wurde Android Inc. unter anderen von Andy Rubin gegründet. Ausser das an Software für mobile Telefone gearbeitet wurde, war über das gegen aussen verschlossene Unternehmen nicht viel bekannt. In einem Interview mit ‚Bloomberg BusinessWeek' zwei Monate vor der Gründung des Unternehmens sagte Rubin, dass in der Nutzung von Positionsdaten und Beachtung der Einstellung des Benutzers enormes Potenzial für intelligente, mobile Geräte liege.[5]

Im August 2005 wurde Android von Google Inc. gekauft. Google gab die talentierten Entwickler und die grossartige Technologie als Grund für die Akquisition an. [6]

Wiederum zwei Jahre später, im November 2007, wurde die ‚Open Handset Alliance' veröffentlicht.[7] Sie besteht aus namhaften Mobilfunkgesellschaften, Software Firmen, kommerziellen Firmen, Herstellern von Halbleitern und Geräteherstellern. Ihr Ziel war es mit Android etwas Offenes zu gestalten. Dieses Öffnen ist in vielerlei Hinsichten gemeint. Einerseits ist die Software ‚Open Source', andererseits sind es auch viele Geräte und selbst das Ökosystem, welches die Geräte umgibt. Damit gemeint ist etwa die Freiheit, zwischen einer Vielzahl von Geräten und Herstellern wählen zu können. Aber auch die Austauschbarkeit innerhalb der Software auf den Geräten ist gewährleistet. Zudem soll es jedem Entwickler möglich sein, Software für diese Geräte entwickeln zu können. Es sind weder teure Entwicklungsumgebungen nötig noch existieren strenge Richtlinien für die Entwicklung und die Veröffentlichung im Google Play Store. Diese Offenheit erstreckt sich über das ganze ‚Projekt Android'.

[5] Elgin, Ben, 2005
[6] Google-Pressesprecher, 2005
[7] Open Handset Alliance , 2007

Kurz nach der Ankündigung durch die ‚Open Handset Alliance' wurde auch die erste Version des SDK [8] veröffentlicht.[9] Im Oktober 2008 folgte das erste Android Smartphone, das HTC Dream alias T-Mobile G1.

Versionsgeschichte[10]

Bereits in der Version 1.0 waren viele für Android wichtige Komponenten vorhanden. Dazu zählen neben wichtigen Applikationen wie dem Market (welcher für das Herunterladen und Aktualisieren von Applikationen verantwortlich ist) und dem Browser auch generelle Merkmale wie Widgets (kleine Programme, die direkt Informationen auf dem Home Bildschirm anzeigen) und die Notifications Liste (welche Nachrichten aus dem ganzen System zusammenfasst). In den weiteren Versionen wurde der Funktionsumfang für die Smartphones immer weiter ausgebaut.

Im Februar 2011 wurde die erste offizielle Version für Tablets vorgestellt. Die für die grösseren Bildschirme entwickelte Version wurde zunächst parallel zur Smartphone Version entwickelt. Neu waren auch die sogenannten ‚Soft buttons'. Mit diesen auf dem Bildschirm dargestellten Tasten für ‚Zurück', ‚Menü' und ‚laufende Applikationen' wurden die Geräte flexibler. Dabei konnte der Platz für die physischen Tasten eingespart werden und besser auf die Ausrichtung des Gerätes eingegangen werden. „There is no ‚wrong' way to hold it"[11], wie Andy Rubin sagt.

Den nächsten grossen Schritt macht Android in der Version 4. Die bislang parallel entwickelten Tablet und Smartphone Betriebssysteme wurden zusammengelegt. Damit kam auch die Darstellung der ehemals physischen Tasten auf den Bildschirm auf die Smartphones.

In der darauffolgenden Version 4.1 alias ‚Jelly Bean' wurde vor allem an einer Steigerung der Performance gearbeitet. In Version 4.2 wurden viele kleine Änderungen vorgenommen. Die Notifications Bar wurde um eine Einstellungsseite erweitert. Zudem wurde das Layout für Tablets dem von Smartphones angeglichen. Ebenfalls neu ist die Möglichkeit, mehrere, voneinander getrennte Accounts einzurichten. Diese Funktion ist aber nur auf Tablets erhältlich.

Nexus

Seit der ersten Version von Android arbeitet Google abwechslungsweise mit verschiedenen Geräteherstellern zusammen, um Geräte zu entwickeln, welche nicht nur für die Entwicklung des Betriebssystems genutzt werden, sondern auch in den Verkauf gelangen. Mit dem HTC Dream, dem Motorola Xoom und der Nexus Serie demonstriert Google jeweils die neuste Android Version. Es wird dabei jeweils eine nicht modifizierte Version des Betriebssystems verbreitet, wodurch die neuste Version meist direkt nach der Vorstellung verfügbar ist.

[8] engl. Software Development Kit – Software Entwicklungspaket
[9] androiddevelopers, 2007
[10] Zusammengestellt aus Beiträgen von [X]CUBELABS, Chris Ziegler und Wikipedia.
[11] electronista, 2010

Entwicklungsmöglichkeiten in Android

Einer der grössten Vorteile für Entwickler ist, dass Google mehrere Möglichkeiten zur Entwicklung zur Verfügung stellt. Eine Unterteilung ist grundsätzlich in HTML, Java und native Apps möglich. Jede dieser Methoden hat ihre Vor- und Nachteile.

HTML Apps oder auch Web Apps genannt sind nichts anderes als auf mobile Bildschirme angepasste Websites. Diese sind grundsätzlich nicht auf den Geräten selber installiert, sondern werden mit Hilfe einer Verbindung mit dem Internet im Browser angezeigt. Es ist aber auch möglich, die Daten lokal abzuspeichern und mit einer installierten Applikation aufzurufen. Die Vorteile einer Web App liegen auf der Hand: Der Entwickler muss nicht über Kenntnisse in Java oder C/C++ verfügen, es genügen Kenntnisse in HTML, CSS und JavaScript. Zudem kann die Web App ohne grosse Anpassungen auch für andere mobile Betriebssystemen wie iOS oder Windows Phone verwendet werden. Die Nachteile, welche diese Art von Applikationen mit sich bringen, sind für viele Entwickler zu gross. Zum einen ist die Performance eigeschränkt. Besonders Apps, welche viele graphische Elemente benutzten oder rechenintensiv sind, leiden darunter. Es fehlen auch die von Android zur Verfügung gestellten Elemente, wie die Views (z.B. ein Button), die Bedienungselemente (zB. die Navigationliste) und die Animationen. Diese müssen vom Entwickler selber umgesetzt werden.[12]

Auf der Android Developer Website[13] merkt man schnell, welche Methode Google den Entwicklern nahelegt. Es handelt sich um die Entwicklung mit Java. Sie kann mit den Android SDK direkt in der IDE (Integrated Development Environment - integrierte Entwicklungsumgebung) Eclipse entwickelt werden. Es kann aber auch mit anderen Umgebungen entwickelt werden. Das Android SDK wird aber in jedem Fall benötigt. Die Vorteile gegenüber der Web App sind die bessere Performance und die vordefinierten UI-Elemente (User Interface, den für den Nutzer sichtbaren Bereich). Der Entwickler muss sich dafür in den Java und Android Bibliotheken auskennen.

Die Möglichkeit, nativen Code zu entwickeln, ist in Grunde eine Erweiterung der Java Applikation. Die ermöglicht dem Entwickler bestehende C/C++ Bibliotheken zu nutzen oder direkt in C/C++ zu programmieren. Auf der Android Developer Website[14] wird aber dazu geraten, nativen Code nur zu verwenden, wenn es aufgrund einer nötigen Bibliothek nicht anders möglich ist oder aber sehr rechenintensive Aufgaben gelöst werden müssen. Der Nachteil ist, dass der Code durch den Einsatz des NDK (Native Development Kit – Paket für das Entwickeln von nativem Code) wesentlich komplexer wird.

Entwicklungsumgebung und Tools

Eclipse

Eclipse ist eine Open-Source IDE, welche sich durch Plug-Ins erweitern lässt. Um mit Eclipse Android Applikationen zu entwickeln, ist es sinnvoll, die ADT[15] und das SDK einzubinden. Damit werden die Android Bibliotheken wie auch viele weitere Tools, wie z.B. der Emulator, in Eclipse integriert.

[12] Yao, Mariya, 2011
[13] Android Open Source Project, Getting Started
[14] Android Open Source Project, Android NDK
[15] Android Developer Tools

Debugging

Wer bereits Software entwickelt hat weiss, dass man beim Debuggen[16] sehr viel Zeit verlieren kann. Durch das Nutzen der integrierten Tools lässt sich dieser Aufwand minimieren. Während dem Programmieren habe ich hauptsächlich 2 Elemente benutzt. Die LogCat Ausgabe gibt (neben diversen weiteren Parametern wie dem benutzten Memory) bei einem Programmabsturz eine Fehlermeldung aus. Aus dieser Fehlermeldung lassen sich viele wertvolle Information herauslesen, etwa wo (in welchen Zeile) sich welcher Fehler ereignet hat. Einfache Fehler lassen sich durch diese Anhaltspunkte sehr leicht aufspüren. Bei komplexeren Fehlern lässt sich die App zusätzlich im „debug mode“ starten. Es lassen sich damit Parameter aus der laufenden App herauslesen. Falls gewünscht können auch sog. Breakpoints an einer bestimmten Stelle im Code platziert werden. An diesen Stellen wird die Ausführung des Codes pausiert. Der Entwickler hat nun die Möglichkeit, die bis zu diesem Punkt gesetzten Variablen einfach zu überprüfen und festzustellen, welche Zeilen wirklich ausgeführt wurden.

TouchTest

Funktionen

Die Applikation TouchTest habe ich zuerst entwickelt. Es handelt sich um eine App, welche fortlaufend auf dem Bildschirm zufällig Elemente darstellt, welche der Proband mit dem Finger antippen muss. Diese Applikation konnte ich regelmässig an einer behinderten Person testen und dadurch sehr gut weiterentwickeln. Das ist auch der Grund, warum die App zum Schluss viel umfangreicher geworden ist, als eigentlich erwartet. Zunächst zu den Funktionen:

In der TestActivity werden auf dem Bildschirm Elemente angezeigt, welche ihren Standort zufällig verändern. Der Proband muss nun diese Elemente mit dem Finger antippen. Über das Einstellungsmenü kann genau spezifiziert werden, wie die Activity auf den TouchEvent reagiert. Beim resultatabhängigen Speichern wird der Versuch nur gezählt, wenn er in einer gewissen Nähe zum zu treffenden Element ist. Diese lässt sich über einen Regler verändern. Ist der TouchEvent zu weit entfernt, bleibt das Element bestehen und der Schritt wird nicht gezählt. Diese Funktion habe ich eingeführt, nachdem ich bei einem Test bemerkt habe, dass die behinderten Probanden sonst mit der Zeit einfach nur noch auf dieselbe Stelle tippen. Ist das resultatabhängige Feedback aktiviert, wird die haptische bzw. akustische Rückmeldung aufgrund der Distanz

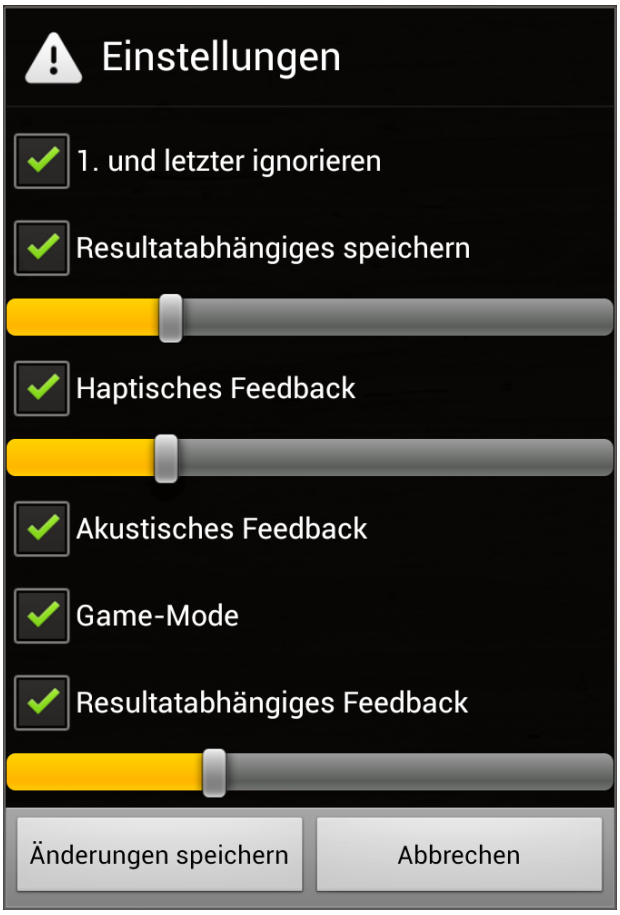

Abbildung 1: Einstellungsmenü von TouchTest

[16] Bugs (engl. Käfer, steht für Fehler im Programmcode) beseitigen.

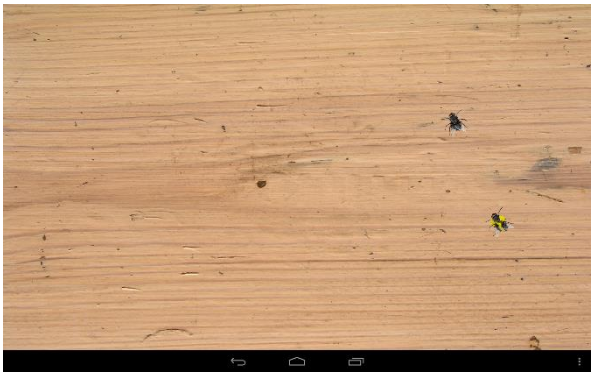

Abbildung 2: TestActivity im Game-Mode

verkürzt, bzw. leiser. Damit werden die behinderten Personen, welche ein lauteres Geräusch als wesentlich lustiger empfinden, zusätzlich animiert, das Element möglichst genau zu treffen. Die maximale Distanz, welche noch gezählt wird, lässt sich über einen Regler wählen. Über den Game-Mode wird der ganze Testverlauf zu einem Spiel. Die Probanden müssen nun nicht mehr einem roten Punkt, sondern einer Fliege nachjagen. Diese bleibt zudem einen Schritt lang zerquetscht auf dem Bildschirm zurück. Der Hintergrund, eine Holzplatte mit Astlöchern, erhöht zudem die Schwierigkeit, den Punkt zu finden. Über die Felder haptisches Feedback sowie akustisches Feedback lassen sich Rückmeldungen über Vibration, bzw. über einen Ton zuschalten. Die Dauer der Vibration kann über einen Regler verändert werden.

In der AnalyseActivity lassen sich die Resultate auswerten. Die Analyse besteht aus einer Karte, auf welcher der Bildschirm verkleinert dargestellt wird. Diese Karte wird eine beliebige Anzahl Felder unterteilt. Die Felder wiederum werden mit einer Farbe eingefärbt, welche dem durchschnittlichen Wert in diesem Bereich entspricht. Unterhalb dieser Karte lassen sich diverse Einstellungen vornehmen.

- Es kann verändert werden, in wie viele Felder die Karte aufgeteilt wird. Dies kann in der Länge und in der Breite unabhängig voneinander geregelt werden.
- Über einen Spinner[17] lässt sich auswählen, welche Resultate von welchem Probanden betrachtet werden sollen.
- Die Karte kann nach Genauigkeit oder nach Geschwindigkeit eingefärbt werden.
- Die Farbe, in welcher die Karte eingefärbt wird, kann verändert werden.
- Es können über zwei Regler die Grenzen eingestellt werden, in welchem Bereich die Werte betrachtet werden sollen.

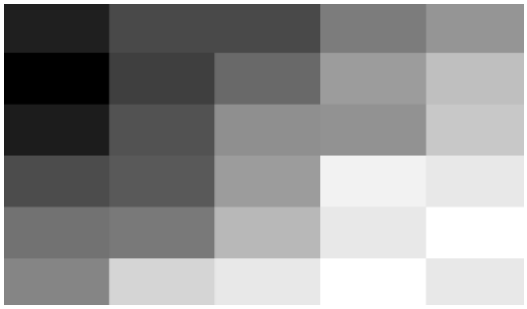

Abbildung 3: Testresultat von einer nicht behinderten Person, Rechtshänder. Aufgeteilt in 5 vertikale und 6 horizontale Reihen, je heller das Feld, desto besser die Genauigkeit. Durchschnittlich 82 Pixel Distanz (ca. 7mm).

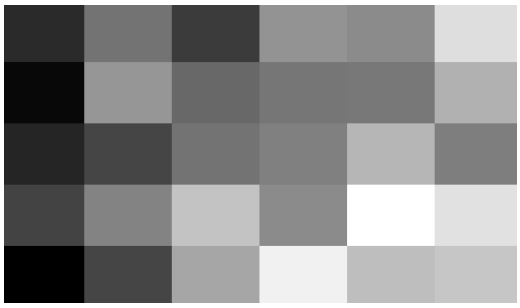

Abbildung 4: Testresultat von einer behinderten Person, Rechtshänder. Aufgeteilt in 6 vertikale und 5 horizontale Reihen, je heller das Feld, desto besser die Genauigkeit. Durchschnittlich 107 Pixel Distanz (ca. 9mm).

Durch das Berühren der 2 Buttons „Anwenden“ und „Anpassen“ lässt sich die Karte generieren. Wird „Anpassen“ ausgewählt, werden zusätzlich noch die Grenzen generiert, so dass dem Feld mit dem schlechteste Wert der hellste und dem mit dem besten Wert der dunkelste Farbton zugeordnet wird.

[17] Android UI-Widget, ermöglicht es aus einer vorgegebenen Anzahl Elemente ein Element auszuwählen(vgl. Drop-Down Menü).

Durch Tippen auf ein Feld in der Karte lässt sich auswählen, zu welchem Feld genauere Informationen angezeigt werden. Im rechts neben der Karte platzierten Feld lassen sich Position, Grösse, Anzahl Werte, durchschnittliche Distanz, durchschnittliche Dauer sowie der durchschnittliche Verschiebungsvektor in X und Y Richtung ablesen.

Analyse

Mit Hilfe dieser Applikation habe ich nun einige Personen, behinderte wie nicht behinderte Menschen, getestet. Interessant ist, dass bei den meisten, nicht behinderten wie behinderten Rechtshändern ein sehr ähnliches Bild entsteht. Die Genauigkeit nimmt dabei jeweils von links oben nach rechts unten ab. Zudem ist bei allen nicht behinderten Probanden die Genauigkeit bei einem kleineren zu treffenden Element wesentlich besser. Grosse Unterschiede zwischen behinderten und nicht behinderten Personen ergeben sich in der Schnelligkeit. Bei nicht behinderten Personen liegt diese im Bereich einer halben Sekunde, bei einer behinderten Person bei zwei Sekunden. Es spielt zudem eine Rolle, wo der Test durchgeführt wird. Auf einem Sofa sitzend nimmt bei einer behinderten Person vor allem die Genauigkeit auf der Y-Achse stark ab. Dies liegt daran, dass die Person den Arm nicht abstützen kann. Zudem wird, auf einem Sofa sitzend, der Punkt zuerst mit dem Finger auf dem Bildschirm „abgetastet“ und anschliessend angetippt, auf einem Tisch hingegen wird direkt getippt.

Tagebuch

Herangehensweise

Die Idee ein Tagebuch zu entwickeln ist bei meinem ersten Besuch in der Tagesstätte der ‚insieme Oberwallis‘ in Visp entstanden. Bislang wurden in eine Agenda Piktogramme geklebt und gezeichnet. Dies konnte aufgrund körperlicher Behinderungen (fehlende motorische Fähigkeiten in den Armen / Händen) nicht von allen selbst gemacht werden, ihnen wurde von einem Betreuer geholfen. Der grossen Mehrheit hat diese Aktivität aber sichtlich gefallen. Ich habe mich deshalb entschieden, diese Aktivität zum Ausgangspunkt meiner Arbeit zu machen.

Den Aufbau dieser Applikation habe ich von Beginn an sehr genau geplant. An einem Whiteboard habe ich sowohl das UI (den sichtbaren Bereich) wie auch die Logik (die Struktur des Programms) entworfen. Diesen Plan habe ich dann schrittweise in Code umgesetzt.

Aufbau der Applikation

Die Applikation besteht aus verschiedenen Funktionen. Die Hauptfunktion (das Tagebuch) ist in zwei Modi verfügbar. Mit dem Tagebuch Editor (DiaryEditor) lässt sich ein Tagebuch gestalten, mit dem Tagebuch Viewer (DiaryViewer) lässt es sich anschauen. Aus dem Viewer kann jederzeit mit einem Tipp auf einen Button in den Editor gewechselt werden. Eine weitere Funktion ist die Kamera, welche es den behinderten Nutzern erlaubt, auf einfache Art und Weise Fotos zu machen. Diese Funktion ist entweder separat oder aus dem Tagebuch Editor heraus abrufbar. Eine weitere Funktion ist der Piktogramm-Ersteller, welcher aber nur in der zukünftigen Betreuerversion enthalten sein wird.

Der Piktogramm Ersteller

Der Piktogramm Ersteller ermöglicht es, eigene Piktogramme zu definieren. Diese können durch einfaches Malen mit dem Finger erstellt werden. Anschliessend muss vom Betreuer der Name des Piktogramms eingegeben werden. Dies ist nötig, damit es einerseits wieder

gefunden werden kann, andererseits aber auch, um eine zukünftige Sprachausgabe zu ermöglichen.

In Absprache mit den Betreuern der „insieme Oberwallis“ wurde auf diese Funktion innerhalb der App für die behinderten Betreuten verzichtet. Grund dafür ist, dass die Piktogramme möglichst einheitlich sein sollten. Dies soll dadurch erreicht werden, dass der Betreuer auf seinem eigenen Gerät ein fehlendes Piktogramm erstellen kann, welches dann über einen Server auf alle Geräte in der Gruppe verteilt wird.

Die Kamera

Die Standardkamera aus dem Android Betriebssystem kann sehr viel. Die vielen Einstellungsmöglichkeiten und Menüs machen es einer behinderten Person aber unmöglich, die Kamera selbstständig zu nutzen. In der für das Tagebuch entwickelten Version habe ich deshalb die Vielzahl der Funktionen auf drei hinuntergebrochen. Über den Auslöser kann ein Bild geschossen werden, über den Blitz-Schalter die Einstellung für den Blitz verändert werden und über den Richtungswechsler zwischen der vorderen und der hinteren Kamera geschaltet werden. Die letzten zwei Buttons werden nur angezeigt, wenn die Option auch wirklich zu Verfügung steht. Der Weissabgleich sowie das Fokussieren wird (insofern vorhanden) automatisch vom System geregelt. Wird der Auslöser bestätigt, gelangt der Benutzer zu einer Bestätigungsseite, wo er das Bild bestätigen oder verwerfen kann. Bestätigt der Nutzer das Bild, wird es lokal gespeichert sowie ein Eintrag in eine Datenbank gemacht. Dieser Eintrag enthält auch eine kleine Vorschau, welche für den Bilder-Browser verwendet wird.

Der Tagebuch Viewer

Über den genauen Aufbau des Tagebuchs habe ich mir sehr viele Gedanken gemacht. Eine Überlegung war, ob das Tagebuch in verschiedene Seiten unterteilt oder als eine grosse Karte dargestellt werden soll. In Absprache mit einigen Betreuern haben wir uns darauf geeinigt, das Tagebuch in einzelne Seiten zu unterteilen. Dies vereinfacht es den behinderten Nutzern, sich im Tagebuch zu orientieren. Ebenfalls im Viewer ist ein Button vorhanden, mit welchem in den Editor-Modus geschaltet werden kann.

Der Seitenwechsler

Die Seiten werden in einem Stapel aufgeschichtet. Durch Wischen von der linken (rechten) Kante des Bildschirms in die Mitte kann die Seite geblättert werden. Dieses Wechseln der Seiten hat wesentlich mehr Arbeit benötigt als zunächst angenommen. Das liegt vor allem daran, dass das Blättern aus zwei Vorgängen besteht. Zunächst folgt das Blatt dem Finger, anschliessend muss es sich selber auf eine bestimmte Position bewegen. Die Umsetzung der ersten Bewegung war relativ einfach. Die zu bewegende Seite wird dabei jeweils um die X-Achsen-Verschiebung des Fingers verschoben. Die Seite bleibt also am Finger haften. Die zweite Bewegung soll nicht abrupt, sondern natürlich wirken. Dazu habe ich folgendes Model aufgestellt. Es berücksichtigt dabei den letzten Punkt, bevor die Seite losgelassen wird, sowie den Punkt, an dem sie losgelassen wird. Es wird zunächst entschieden, in welche Richtung die Seite bewegt werden soll. Wird die Seite um weniger als 1/3 der Bildschirmbreite bewegt, schwingt die Seite zurück. Ansonsten wandert sie auf die andere Seite. Damit sind nun drei Punkte bekannt und es kann eine quadratische Funktion berechnet werden, auf welcher sich die Seite schrittweise bewegen kann.

Beispiel

Angenommen, die Seite wurde an der Position 400 losgelassen, die letzte Position war 350 und die Breite des Bildschirms beträgt 1000 Pixel. Die Position 400 ist grösser (weiter rechts) als die Position 333 (1/3 der gesamten Breite), die Seite muss sich also auf die Position 1000 zubewegen. Um eine möglichst einfache Kurve zu erhalten wird auf der X-Achse jeweils der Schritt angegeben und auf der Y-Achse die Distanz zum Zielpunkt. Um das Beispiel einfach zu halten, nehmen wir an, es werden 10 Schritte benötigt. Wir haben also die 3 Punkte p1(-1/650), p2(0/600) und p3(10/0). Durch die Auflösung des linearen Gleichungssystems kann nun die Quadratische Gleichung $y=ax^2+bx+c$ errechnet werden, wie in Abbildung 5 dargestellt. Die Seite wird nun Schritt für Schritt in Richtung Endpunkt bewegt, es entsteht eine natürliche Bewegung.

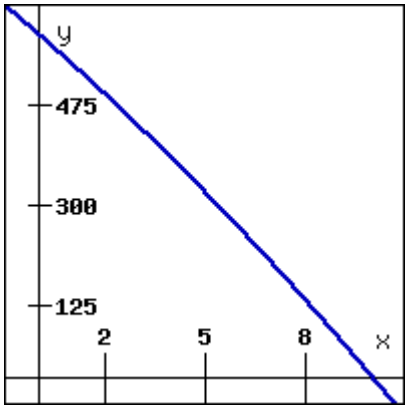

Abbildung 5: $-0.\overline{90}x^2 - 50.\overline{90}x + 600$
Erstellt auf http://rechneronline.de

Der Tagebuch Editor

Im Editor können sowohl neue Einträge erstellt, wie auch alte bearbeitet werden. In diesem Element habe ich sehr viel Wert auf Objektorientierte Programmierung gelegt. So besteht der Editor (wie auch der Viewer) aus der Oberfläche, welche die Buttons und den PageHolder anzeigt, die Klasse welche die Seiten organisiert und deren Übergang animiert. Die Seite (Page) wiederum beinhaltet sämtliche Elemente, welche auf der Seite abgelegt wurden. Die Elemente (DrawElement/ImageElement) wiederum speichern jeweils die Daten, welche nötig sind, um sich selber grafisch darzustellen. Ein DrawElement (Zeichnung) beinhaltet zum Beispiel sämtliche Punkte, welche beim Zeichnen berührt wurden.

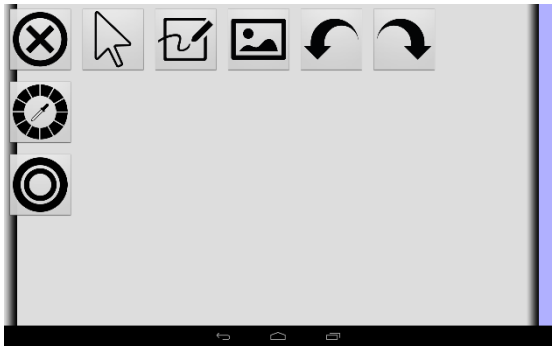

Abbildung 6: Screenshot des DiaryEditors

Die Navigation im Editor ist in zwei Leisten unterteilt. In der oberen horizontalen Leiste können die Grundfunktionen wie „Beenden“ oder „Zeichnung“ ausgewählt werden. Je nach ausgewähltem Element erscheint auf der linken Seite eine vertikale Leiste, welche weitere spezifische Optionen zur Verfügung stellt.

Memory Managment

Einer Android Applikation steht nur ein gewisser Platz im Zwischenspeicher zu. Auf dem ‚Transformers Prime‘ von Asus sind das 20MB[18]. Durch die grossen Datenmengen, welche vor allem durch Bitmap Grafiken entstehen, gerät man schnell an dieses Limit. Bitmap Grafiken werden in ARBG_8888 abgespeichert, das heisst, dass jeder Pixel in den vier Kanälen Alpha (Transparenz), Rot, Grün und Blau in je 8 Bit abgespeichert wird, das entspricht genau 4 Byte. Soll nun also ein bildschirmfüllendes Bitmap erzeugt werden, müssen bereits 1200*800*8 Byte verwendet werden. Dies entspricht 8.1 MB. Da Android beim Überschreiten der gesetzten Grenze die Applikation beendet, musste ich im Android Manifest den Wert „android:largeHeap“ auf ‚wahr‘ setzten. Dadurch erlaubt mir das System

[18] Gemäss eigenen Tests (Ausgaben des LogCat)

weiteren Zwischenspeicher zu beanspruchen. Dabei wird der maximal belegbare Zwischenspeicher dieser Applikation zugeschrieben. Beim Nexus 10 sind das (je nach bereits laufenden Apps) um die 500MB, welche zur Verfügung stehen. Dennoch muss darauf geachtet werden, nicht zu viel Zwischenspeicher zu belegen. Möglich ist es etwa, nicht benötigte Bitmaps von nicht angezeigten Seiten zu löschen und erneut zu generieren, wenn sie benötigt werden. Da dies den Rechenaufwand wesentlich erhöht und die Applikation vorerst als einzige Applikation auf den Geräten laufen wird, verzichte ich im Moment noch darauf.

Zeichnen

Beim Zeichnen habe ich mich zu Beginn an der API-Demo Applikation[19] orientiert. Dabei wird auf ein View ein onTouchListener gesetzt, dieser meldet dann jede Interaktion auf diesem View mit dem Touchscreen. Dabei gibt es 3 Events, welche für das Zeichnen wichtig sind: down (erste Berührung), move (Finger wurde bewegt) und up (Finger wurde entfernt). Diese Events kommen jeweils mit den dazugehörigen Koordinaten. Reiht man nun diese Punkte aneinander und verbindet sie miteinander entsteht ein Pfad. Dieser ist aber sehr eckig, da die Methode nicht jede, sondern nur regelmässig Werte zurückgibt. Um dieses Problem zu lösen, wird quadratische Interpolation benutzt. Dafür werden 3 Punkte benötigt. Es wird nun eine Linie durch den ersten und den letzten Punkt gezogen, diese wird aber durch den mittleren Punkt so abgelenkt, dass sich die Kurve der Verbindungsgerade zwischen den Punkten 1 und 2 und den Punkten 2 und 3 nähert. Gut zu erkennen ist das in der Abbildung 7. Die rote Linie stellt die Verbindung mit Linien und die schwarze Kurve die interpolierte Verbindung dar.

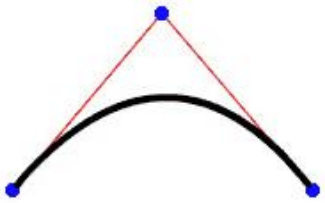

Abbildung 7: Quadratische Interpolation

Damit nun der ganze Pfad interpoliert werden kann, ist ein kleiner Trick nötig. Man darf nicht einfach zwischen jedem 2. Punkt eine Interpolation vornehmen, ansonsten entstehen an diesen Punkten Ecken und der Pfad wird unnötig ungenau. In Abbildung 8 wird genau das gemacht. Am mittleren Punkten, zwischen den 2 anderen Interpolationen, entsteht eine Ecke. Dies liegt daran, dass der bei der vorhergehenden Interpolation benutzte Punkt nicht auf einer Linie durch den nächsten liegt.

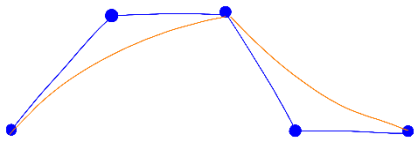

Abbildung 8: Beispiel von einer Interpolation von mehreren Punkten (ohne Nutzung von Mittelpunkten). Grafik erstellt mit Markers.

Anstatt also nur jeden 2. Punkt zu nutzen, wird jeweils zwischen 2 Punkten der Mittelpunkt (orange Punkte in Abbildung 9) bestimmt und anschliessend durch diese Punkte interpoliert (orange Linie). Dabei ist der neue Mittelpunkt jeweils der Punkt, welcher auf dem Pfad liegt. Der Pfad ist nun glatt, da jeweils 3 Punkte (je 2 blaue und 1 oranger) auf derselben Gerade liegen. Damit der Start und Endpunkt ebenfalls im Pfad sind, wird hier eine Linie bis zum jeweils nächsten Mittelpunkt gezogen (blaue Linie zwischen Mittelpunkt und Endpunkten).

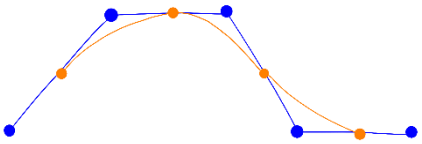

Abbildung 9: Beispiel für eine Interpolation von mehreren Punkten mit Hilfe von Mittelpunkte. Grafik erstellt mit Markers.

[19] Aus dem ‚samples' Ordner im Android SDK (4.1)

Ein Spezialfall ist der normale Punkt. Je nach Gerät gibt es zwei oder mehr Punkte, welche alle auf derselben Stelle sind. Eine Interpolation zwischen diesen Punkten ist in diesem Fall nicht sinnvoll. Damit dennoch etwas gezeichnet wird, muss die Applikation den Fall abfangen und einen einfachen Kreis zeichnen.

ColorPicker / ThicknessPicker

Abbildung 10: Screenshot des ColorPickers

Damit die Zeichnungen auch schöner gestaltet werden können, ist eine Auswahl an Farben und Stiftdicken nötig. Dies wurde mithilfe von zwei Popups, dem ColorPicker und dem ThicknessPicker, realisiert. Der ColorPicker stellt dabei aus einem Array herausgelesene Farbwerte auf einer Karte dar. Die Farbe kann durch Tippen auf eines dieser Felder verändert werden. Der ThicknessPicker stellt einen Kreis mit dem aktuellen Radius mit der aktuellen Farbe dar. Darunter befindet sich ein Dreieck, welches sich gegen rechts langsam öffnet. Nur durch Berührung dieses Dreiecks kann der Radius des Kreises verändert werden. Der Nutzer bekommt somit direkt eine Rückmeldung und kann seine Wahl noch besser justieren. Ist man mit dem neuen Wert einverstanden, kann der Finger entfernt werden und der neue Radius wird übernommen.

ImageElement

Einen sehr hohen Stellenwert nehmen die Fotos ein. Bei Berührung des Foto-Buttons wird ein Dialog geöffnet. Der Nutzer kann hier zwischen einer neuen Aufnahme und einer bestehenden auswählen. Will der Nutzer ein neues Bild anfertigen, wird die Kamera gestartet. Sobald der Nutzer das Bild bestätigt hat, wird es an das Tagebuch zurückgegeben. Soll ein bestehendes genutzt werden, wird ein ImageBrowser geladen, welcher die Bilder aus der Datenbank anzeigt.

Durch das vertikale Menü kann der Nutzer weitere Aktionen mit dem Bild durchführen. Ein Verschieben des Bildes und eine Skalierung sind dadurch möglich. Speziell ist dabei das bei der Skalierung das Bild nicht durch zwei Finger, welche auseinandergespreizt werden, sondern durch nur einen Finger, welcher das Bild nach rechts oben vergrössert, bzw. nach links unten verkleinert.

Daten speichern auf Android Geräten[20]

Für das Sichern der Daten gibt es auf Android Geräten viele Möglichkeiten. Eine ist das Speichern von Preferences. Dieses System, welches geschaffen wurde, um Einstellungen zu speichern, ist sehr einfach und effizient nutzbar. Ich benutzte diesen Typ von Datenspeicher, um die Einstellungen der TouchTest Applikation zu sichern. Für grössere Datenmengen ist dieses System aber weniger geeignet.

Eine weitere Möglichkeit, Daten tabellarisch zu sichern, bieten SQLite Datenbanken. Der Stärke von einem solchen System ist das Laden von vielen gleich strukturierten Datensätzen. Ich benutze dieses System z.B. im Tagebuch um einen Index für die Bilder und die verschiedenen Tagebücher zu erstellen.

[20] Becker, Arno und Pant, Marcus, 2009, S. 161ff (SQLite) und S. 155ff (Dateisystem).

In Android können natürlich auch Daten direkt in den Speicher geschrieben werden. In meiner Applikation mache ich das auf zwei verschiedene Arten. Einerseits werden Bilder direkt in das Standardverzeichnis für Bilder gespeichert. Andererseits speichere ich komplexere Datensätze direkt als Serializable File[21]. Diese Möglichkeit stammt aus der Java Bibliothek. Dabei werden die zu speichernden Objekte, in diesem Fall das Tagebuch selber, direkt in eine Byte Sequenz umgewandelt. Diese Sequenz lässt sich anschliessend in den Speicher schreiben. Wird das Objekt benötigt, kann es wieder aus dem Speicher geladen werden und direkt in das benötigte Objekt umgewandelt werden.

Datensicherung im Tagebuch

Die Sicherung des Tagebuches ist die komplexeste, welche ich im Rahmen meiner Maturaarbeit verwirklicht habe. Komplex wird dieser Vorgang hauptsächlich durch die vielen verschiedenen Elemente, die gesichert werden müssen. Da beim Serialisieren ganze Objekte in Byte Sequenzen umgewandelt werden, sollten die Objekte selber möglichst klein sein und natürlich keine Objekte enthalten, welche nicht serialisiert werden können. Beim Tagebuch ist das z.B. das ‚Rect position', welches sich genau Position des Elementes mit Hilfe eines Vierecks merkt. Dieses muss in die wichtigen Komponenten links, oben, rechts und untern unterteilt werden. Um das Tagebuch zu speichern werden die Daten zu einem Objekt verbunden. Dabei enthalten die Klassen SaveDiary und Pages jeweils ein Array, um selber mehrere Elemente aufzunehmen. Dadurch können beliebig grosse Tagebücher abgespeichert werden.

Analyse von einigen interessanten Code-Stellen

Zeichnen

Der Nutzer kann das Tagebuch mit Zeichnungen gestalten. Zunächst die Klassendefinition von einem DrawingElement:

```
public class DrawingElement {
	ArrayList<DrawingStep> allSteps = new ArrayList<DrawingStep>();
	Rect position = new Rect(0, 0, 1, 1);
	int color;
	float thickness;
}
```

Es enthält:

- das Rect (Rechteck) *position*, welches die Ausmasse der Zeichnung enthält;
- das int *color*, welches den Farbwert des Zeichnung enthält;
- das float *thickness*, welches die Dicke des Pfades enthält;
- die ArrayList of DrawingSteps *allSteps*, welche sämtliche Schritte enthält.

DrawingSteps selber ist folgendermassen definiert:

```
class DrawingStep implements Serializable {
	private static final long serialVersionUID = 1L;
	int x = 0;
	int y = 0;
	boolean isFirst;
	boolean isMove;
	boolean isLast;
}
```

[21] Vidal, Jose, 2012

Ein DrawingStep ist jeweils mit einem Punkt des Pfades zu vergleichen. Die drei booleans bestimmen dabei jeweils, wo die Punkte im Pfad liegen (Beginn, Mitte oder Schluss) und die zwei Ganzzahlen x und y die genau kartesische Position (von der oberen linken Ecke aus). Speziell an dieser Klasse ist, dass die Klasse Serializable implementiert wird. Dies ist nötig, damit das Tagebuch anschliessend gesichert werden kann.
Im Tagebuch können die Zeichnungen auf 2 Arten dargestellt werden. Als direkte, sich verändernde Ausgabe (während dem Zeichnen) und als Darstellung eines bereits gezeichneten Elements. In beiden Fällen wird, sobald das DrawingElement mit den nötigen Daten gefüllt wurde, für jedes Element aus *allSteps* (Typ DrawingStep) die Methode *draw(DrawingStep ds)* in der Klasse Page (Z. 257ff) aufgerufen. Hier wird, je nach Reihenfolge des Schrittes im Pfad, ein anderes Stück Code ausgeführt.

```
if (step.isMove) {
      if (!(mX == step.x && mY == step.y)) {
            path.quadTo(mX, mY,(step.x + mX)/2,(step.y +mY)/2);
            mX = step.x;
            mY = step.y;
            touchMoved = true;
      }
} else if (step.isFirst) {

      paint.setStyle(Paint.Style.STROKE);
      paint.setColor(de.color);
      paint.setStrokeWidth(de.thickness);
      path.reset();
      mX = step.x;
      mY = step.y;
      path.moveTo(mX, mY);
} else if (step.isLast) {
      if (touchMoved) {
            path.lineTo(mX, mY);
      } else {
            paint.setStyle(Paint.Style.FILL);
            path.addCircle(mX, mY, paint.getStrokeWidth() / 2,
                                    Path.Direction.CW);
      }
      touchMoved = false;

}
```

Es wird dabei als erstes überprüft, ob der Schritt in der Mitte liegt, da dies statistisch am häufigsten vorkommt und somit am effizientesten ist. Aus logischen Gründen beginne ich dennoch beim ersten Schritt.
Bei diesem wird noch nichts gezeichnet. Es werden lediglich Grundeinstellungen vorgenommen. Der Pfad wird zurückgesetzt und die Farbe, die Dicke und der Style eingestellt. Letzteres ist nötig, damit der Pfad nicht ausgefüllt, sondern nachgezeichnet wird. Ebenfalls werden die zwei temporären Variablen *mX* und *mY* gesichert. Zum Schluss wird noch der Start des Pfades auf die Postion des ersten Schrittes gesetzt.
Ist der Schritt weder am Beginn noch am Ende des Pfades, wird zuerst überprüft, ob sich der Punkt im Vergleich zum vorhergehenden überhaupt bewegt hat. Ist dies nicht der Fall, kann der Punkt ignoriert werden. Ansonsten wird eine Quadratische Interpolation (siehe Abbildung 7) vorgenommen. Der erste Punkt entspricht dabei jeweils dem letzten im Pfad, der zweite dem letzten Mittelpunkt (*mX*/*mY*) (siehe blauer Punkt in Abbildung 9) und der letzte dem neuen Mittelpunkt. Ist dieser Schritt der zweite Schritt, entspricht der letzte Punkt im Pfad dem letzten Mittelpunkt.

Ebenfalls an dieser Stelle wird das boolean touchMoved auf ‚true' gesetzt. Dies ist nötig, damit bei Erreichen des letzten Schrittes zwischen einem Pfad (mehrere Schritte) und einem Punkt (nur ein erster und ein letzter Punkt) unterschieden werden kann. Handelt es sich um einem Pfad, wird eine Linie zum letzten Punkt gezogen. Entspricht die Zeichnung aber einem Punkt, muss der Style in dem Modus „FILL" gesetzt werden, damit anschliessend ein ausgefüllter Kreiss gezeichnet werden kann.

Das Rect *position* ist im aktuellen Projekt nicht von Bedeutung. Sollte aber ein Verschieben oder Skalieren von Zeichnungen gewünscht sein, liesse sich dies wie bei den ImageElements implementieren.

ColorPicker

Der ColorPicker ermöglicht es dem Nutzer, die Farbe, mit der er zeichnen möchte, zu wählen. Durch tippen auf den dazu gehörenden Button wird ein PopupWindow geöffnet. Dieses stellt ein View dar, auf das ich eingehen möchte. Am interessantesten ist dabei die onDraw[22] Methode, welche die Oberfläche des Views generiert.

```
protected void onDraw(Canvas canvas) {

	mPaint.setColor(Color.WHITE);
	canvas.drawRect(posX + 25, Height / 10, (Width - posX),
			Height * 9 / 10, mPaint);
	mPaint.setColor(Color.BLACK);
	canvas.drawRect(posX + 45, (Height / 10) + 20, (Width - posX) - 22,
			(Height * 9 / 10) - 22, mPaint);
	mPaint.setColor(Color.WHITE);
	Path path = new Path();
	path.moveTo(posX + 25, posY - 25);
	path.lineTo(posX, posY);
	path.lineTo(posX + 25, posY + 25);
	canvas.drawPath(path, mPaint);

	int i = 0;
	int h = colors.length / w;
	widthOfBlock = (int) Math.floor((Width - (2 * posX) - 75) / w);
	heightOfBlock = (int) Math.floor(((Height * 4 / 5) - 50) / h);
	for (int c : colors) {
		int row = (int) Math.floor(i / w);
		int cell = i % w;
		mPaint.setColor(c);
		canvas.drawRect((cell * widthOfBlock) + posX + 50,
			(row * heightOfBlock) + (Height / 10) + 25,
			((cell * widthOfBlock) + widthOfBlock) + posX + 50,
			((row * heightOfBlock) + heightOfBlock) + (Height / 10)
					+ 25, mPaint);
		i++;
	}
}
```

Zunächst werden dabei die weisse Oberfläche, der schwarze innere Rahmen sowie der Pfeil auf der linken Seite erstellt. Der Pfeil wird, um die Methode möglichst einfach zu halten, durch einen ausgefüllten Pfad dargestellt. Als nächstes werden die Anzahl Felder in vertikaler Richtung errechnet. Die Anzahl Felder in horizontaler Richtung werden statisch von der Aufrufenden Klasse gesetzt. *colors* entspricht einem im Vorhinein aus den Ressourcen geladenen Array, welches sämtliche anzuzeigenden Farbcodes enthält. Wichtig

[22] In der Klasse ColorPickerView ab Zeile 51

ist dabei, dass *h* * *w* der Länge dieses Arrays entsprechen muss.
In *widthOfBlock* und *heightOfBlock* werden und die Breiten, bzw. Höhen der Felder gespeichert. In der for-Schleife werden nun diese Felder mit den aus dem *colors* Array entsprechenden Farben gezeichnet.
Wird nun auf das View getippt, berechnet das Programm welches Feld sich darunter befindet:

```
if (new Rect(posX + 50, (Height / 10) + 25, (Width - posX) - 25,
                    (Height * 9 / 10) - 25).contains((int) event.getX(),
                    (int) event.getY())) {
              int x = (int) Math.floor(event.getX() - posX - 50);
              int y = (int) Math.floor(event.getY() - (Height / 10) - 25);
              int a = x / widthOfBlock;
              int b = y / heightOfBlock;
              return colors[a + b * w];
}

return -2;
```

Dabei muss zunächst überprüft werden, ob überhaupt ein Feld angetippt wurde. Ist dies nicht der Fall, wird -2 zurückgegeben (-1 ist bereits für Weiss reserviert). Ansonsten wird berechnet, in welcher Reihe b und in welcher Zeile a sich das Feld befindet. Anschliessend wird der zur Position gehörende Farbwert zurückgegeben.

Speichern und Laden des Tagebuchs

Eine letzte Stell, auf die ich eingeben möchte, ist das Sichern und Laden des Tagebuchs. Gesichert werden dabei 2 Elemente, jeweils beim Beenden des DiaryEditors. Einerseits die Vorschau und das Erstellungsdatum in einer Datenbank und andererseits die konkreten Daten in einer Datei im Dateisystem. Ich gehe hier nur auf letzteres ein.

Für das Sichern sind mehrere Klassen notwendig. Für die reine Datenstruktur sind das die Klassen SaveDiary, SavePage, SaveDrawingElement (welche wiederum DrawingStep benötigt) und SaveImageElement. Es wäre natürlich auch möglich gewesen, diese Klassen in einer Klasse zu vereinen. Ich habe aber darauf verzichtet, um die Datenstruktur möglichst Übersichtlich zu halten.
Gesichert werden die Daten in der Methode saveCurrentDiary(), welche sich in der Klasse DiaryEditor ab Zeile 306 befindet. Bis Zeile 316 wird der Datensatz in der Datenbank erstellt. Ab Zeile 318 folgt die Sicherung in einem Serializable-File. In der for-Schleife (Zeile 321-360) wird die Datenstruktur aus den bisher benutzten Elementen erstellt. Anschliessend wird dieses Objekt serialisiert.

```
ObjectOutputStream os;
try {
      os = new ObjectOutputStream(new FileOutputStream(SRC));
      os.writeObject(sd);
      os.close();
} catch (FileNotFoundException e) {
      e.printStackTrace();
} catch (IOException e) {
      e.printStackTrace();
}
```

Es wird ein ObjectOutputStream *os* erstellt, welchem den Speicherort *SRC* zugeordnet wird. Anschliessend wird das Objekt *sd* (vom Typ SaveDiary) in *os* geschrieben und *os* anschliessend geschlossen. Da dieser Prozess Fehleranfällig ist, verlangt der Compiler dass

dieser Vorgang mit Try/Catch umrahmt wird. Damit könnten abstürze zur Laufzeit verhindert werden.

Die Methode, welche die gesicherten Daten wieder öffnet, befindet sich im PageHolder. Dies weil neben dem DiaryEditor auch der DiaryViewer ein Tagebuch öffnen können muss. Die Methode heisst openDiaryFromSrc() und befindet sich ab Zeile 71.
Der Code, welche den Datensatz aus dem Dateisystem heraus öffnet, sieht dem obigen sehr ähnlich.

```
ObjectInputStream is;
SaveDiary sd = null;
try {
      is = new ObjectInputStream(new FileInputStream(SRC));
      sd = (SaveDiary) is.readObject();
      is.close();
} catch (FileNotFoundException e) {
      e.printStackTrace();
} catch (IOException e) {
      e.printStackTrace();
}
```

Wie auch für die obige Prozedur verlangt der Compiler, dass der Vorgang mit Try/Catch umrahmt wird.
In den folgenden Zeilen wird der Datensatz erneut in die benötigte Struktur gebracht.

Fazit

Einige Punkte, wie das Nutzen von Piktogrammen und weiteren Medien, konnte ich nicht implementieren. Das liegt einerseits an den sehr teuren Lizenzkosten für die Piktogramme und andererseits an den wesentlich komplexeren Implementierungen von fortlaufenden Medien wie Audio und Video. Einige Probleme gibt es auch bei der Vorschau der Kamera, welche auf den Nutzer zeigt. Sie wirkt unscharf und verzerrt. Dort liegt das Problem beim Ändern der Vorschaugrösse. Sehr zufrieden bin ich hingegen mit dem Seitenwechsler, dem Zeichnen und den Pickern (ThicknessPicker und insbesondere ColorPicker). Bei diesen Elementen ist es mir sehr gut gelungen Objekt-Orientierten und stabilen Code zu schreiben. Insgesamt habe ich mein Ziel, eine für behinderte Personen nutzbare Applikation zu erstellen, erreicht. Die behinderten Nutzer können die App nach einer Einführung selbstständig nutzen.

Nachwort / Zukunft

Persönlich habe ich während dieser Arbeit viel mehr gelernt, als ich zu Beginn erwartet hatte. Dies bezieht sich einerseits auf die behinderten Personen selber und auf ihre Bedürfnisse. Ich konnte in der Zeit, in welcher ich in der ‚insieme Oberwallis' war, sehr gut erkennen, wie die behinderten Betreuten auf technische Geräte reagierten. Als ich das Projekt einer Betreuerin vorgestellt habe, haben sich viele behinderte Personen zu uns gesetzt und aufmerksam zugehört. Sie hatten eine riesige Freude an den zu zerquetschenden Fliegen und am Fingermalen, bei dem die Finger nicht dreckig werden. Gerade diese Erfahrung hat mir gezeigt, dass auch diese Zielgruppe Freude an so einem Gerät haben kann. Umsetzten könnte ich das etwa durch das eklige Geräusch, welches ausgegeben wird, wenn eine Fliege im TouchTest zerquetscht wird oder an dem ‚natürlichen' Blättern der Seiten im Tagebuch. Vor allem habe ich erfahren dürfen, dass der Wunsch nach Hilfsgeräten, welche nicht wie Hilfsgeräte aussehen, gross ist. Dennoch darf nicht vergessen werden, für wen entwickelt wird und vor allem, welche Bedürfnisse diese Nutzer haben.

Aber auch im technischen Bereich habe ich viel mehr gelernt als erwartet. Dank früheren Arbeiten mit Android konnte ich bereits vor dieser Arbeit auf ein gewisses Grundwissen zurückgreifen. Durch verschiedene Probleme habe ich aber immer tiefere Einblicke in die Programmiertechniken erhalten, welche nötig sind, um ein derartiges System zu erstellen. Insbesondere das Erstellen von eigenen Views hat mir aufgezeigt, wie wichtig etwa ein gut strukturierter, objektorientierter Code sein kann.

Wie bereits in der Einleitung angedeutet, ist mir die Nachhaltigkeit des Projekts sehr wichtig. Ich möchte deshalb noch einige Worte über die Zukunft des Projektes verlieren.

Bereits im November habe ich erste Gespräche mit der Abteilungsleitung der Tagestätte der ‚insieme Oberwallis' in Visp geführt. Ich konnte ihnen ein Konzept unterbreiten, indem ab Februar die Applikation direkt an einer fünfköpfigen Gruppe getestet werden kann. Bis September soll die Software kontinuierlich erweitert werden. Zu diesem Zeitpunkt wird auch über den weiteren Verlauf der Testphase entschieden werden.

Mein Ziel ist es, das System soweit auszubauen, dass auch andere Unternehmen ihre Software zur Verfügung stellen können. Es soll damit eine echte Alternative zu der oftmals überteuerten und nur einseitig einsetzbaren Geräten geschaffen werden. Im Dezember habe ich seitens der ‚insieme Oberwallis' das Okay für das Projekt erhalten. Nach Findung eines oder mehreren Sponsoren kann das Projekt losgehen.

Dank

Ich bin an dieser Stelle vielen Personen zu Dank verpflichtet. An vorderster Stelle ist natürlich mein Betreuer, Herr Bernhard Britsch zu nennen. Er hatte von Beginn an Vertrauen in das Projekt und hat mich auch während dem Projekt unterstützt. Ebenfalls danken möchte ich der Betreuerin in der ‚insieme Oberwallis', Frau Jrene Hässig. Sie hat mir aufgezeigt, was das System können muss und vor allem, wie es das können muss, um bei den behinderten Personen anzukommen. Ebenfalls danken möchte ich Frau Anja Sarbach und Frau Loretan Stefanie. Ohne Sie wäre eine Weiterführung des Projekts nicht möglich geworden. Einen weiteren Dank möchte ich der Familie Abgottspon und ihrem behinderten Sohn Marco widmen. Sie waren von Anfang an offen für das Projekt und gerade durch das Beobachten von Marco, wie er mit den Applikationen umging, konnte ich sehr viele wichtige Punkte wahrnehmen und in die Arbeit einfliessen lassen.

Quellen

Literaturverzeichnis

1. Becker, Arno und Pant, Marcus, Android – Grundlagen und Programmierung, 1. Auflage, dpunkt.verlag Heidelberg 2009.

Internetquellen

1. [X]CUBELABS, http://www.xcubelabs.com/the-android-story.php (10.01.2013)
2. Android Open Source Project, Android NDK, http://developer.android.com/tools/sdk/ndk/index.html (10.01.2013)
3. Android Open Source Project, Custom Components - Fully Customized Components, 2013, http://developer.android.com/guide/topics/ui/custom-components.html#custom (27.01.2013)
4. Android Open Source Project, Getting Started, http://developer.android.com/training/index.html (10.01.2013)
5. androiddevelopers, 2007, Android Demo, http://www.youtube.com/watch?v=1FJHYqE0RDg (10.01.2013)
6. electronista, 2010, http://www.electronista.com/articles/10/12/06/rubin.shows.early.motorola.motopad.and.3d.maps/ (10.01.2013)
7. Elgin, Ben, 2005, http://www.businessweek.com/stories/2005-08-16/google-buys-android-for-its-mobile-arsenal (10.01.2013)
8. Google-Pressesprecher, 2005, http://www.businessweek.com/stories/2005-08-16/google-buys-android-for-its-mobile-arsenal (10.01.2013), „We acquired Android because of the talented engineers and great technology. We're thrilled to have them here.“
9. Open Handset Alliance , 2007, Pressemeldung „Industry Leaders Announce Open Platform for Mobile Devices“, http://www.openhandsetalliance.com/press_110507.html (10.01.2013)
10. Vidal, Jose, 2012, Java Serializable interface: Reading and writing Objects to a file Tutorial, http://www.youtube.com/watch?v=YzwiuRDgSSY (28.12.2012)
11. Wikipedia, http://en.wikipedia.org/wiki/Android_version_history (10.01.2013)
12. Yao, Mariya, 2011, Antwort auf die Frage „Mobile Applications: What are the pros and cons of writing a mobile web app rather than a native app?“, http://www.quora.com/Mobile-Applications/What-are-the-pros-and-cons-of-writing-a-mobile-web-app-rather-than-a-native-app (10.01.2013)
13. Ziegler, Chris, 2011, Android: A visual history, http://www.theverge.com/2011/12/7/2585779/android-history (10.01.2013)

Grafikverzeichnis

1. Android at Mobile World Congress 2012 (Titelbild) http://www.android.com/events/mwc/2012/
2. Quadratische Interpolation (Abbildung 6) http://help.adobe.com/de_DE/AS2LCR/Flash_10.0/help.html?content=00001293.html

Bei den anderen Grafiken handelt es sich um selbst erstellte Screenshots der Applikationen oder Skizzen, welche mit Markers (http://code.google.com/p/markers-for-android/) erstellt wurden.

In den Projekten verwendete Dateien

Audiodateien

Sämtliche Benutzen Geräusche stammen von http://www.freesound.org und wurden auf http://audio.online-convert.com/convert-to-ogg in OGG Dateien konvertiert.

Icons

Sämtliche Icons stammen von http://thenounproject.com/ und wurden mit Hilfe des Programms „Adobe Illustrator“ in PNG Dateien konvertiert.

Für das Projekt verwendete Tutorials

Während diesem Projekt habe ich sehr viele Tutorials genutzt.

- Akustisches Feedback: http://www.droidnova.com/creating-sound-effects-in-android-part-1,570.html (16.7.12)
- Dateien verschieben: http://stackoverflow.com/questions/1995320/how-to-backup-database-file-to-sdcard-on-android (15.7.12)
- Datentypen in SQLite 3: http://www.sqlite.org/datatype3.html (15.7.12)
- Haptisches Feedback: http://android.konreu.com/developer-how-to/vibration-examples-for-android-phone-development/ (16.7.12)
- onDraw()-method: http://marakana.com/s/android_2d_graphics_example,1036/index.html (25.4.12)
- PopUp Window: http://android-er.blogspot.com/2012/03/example-of-using-popupwindow.html (4.5.12)
- SQLite db: http://www.androidhive.info/2011/11/android-sqlite-database-tutorial/ (15.7.12)
- Sowie unzählige weitere auf http://developers.android.com

Anhang

Auf der beiliegenden CD befindet sich folgendes:

- Die Blog-Einträge im PDF Format
- Diese Arbeit im PDF Format
- Erlaubnis für die Erwähnung der Namen
- Quelltexte für die Projekte TouchTest und Tagebuch
- Sämtliche in dieser Arbeit verwendeten Bilder (in voller Auflösung)
- Testdaten aus „TouchTest“

Authentizitätserklärung

Ich bezeuge mit meiner Unterschrift, dass meine Angaben über die bei der Abfassung meiner Maturaarbeit benützten Hilfsmittel und über mir allenfalls zuteil gewordene Hilfe in jeder Hinsicht der Wahrheit entsprechen und vollzählig sind.

Ort und Datum: .....

Unterschrift des Schülers: .....