HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL
FHO Fachhochschule Ostschweiz

NeoMap App Reloaded

Bachelorarbeit

Abteilung Informatik
HSR Hochschule für Technik Rapperswil
Frühlingssemester 2013

Autoren:	Simon Stähli
	Patrick Zenhäusern
Betreuer	Prof. Stefan Keller, HSR
Experte:	Claude Eisenhut, Eisenhut Informatik AG
Gegenleser:	Prof. Dr. Andreas Rinkel, HSR
Durchführungszeitraum:	18.02.2013 – 14.06.2013

Abstract

Mit der rasanten Verbreitung von Smartphones nehmen auch Karten-Applikationen einen immer wichtigeren Standpunkt im Alltag ein. Dies nicht zuletzt dank des allseits bekannten und sehr verbreiteten Google Map Apps, welches auf diversen Plattformen verfügbar ist. Im Gegensatz zu Google Maps stellt der Kartendienst OpenStreetMap Kartenmaterial bereit, welches für jedermann frei verwendbar und auch zur konsequenten Offline-Nutzung verfügbar ist. Nebst lizenztechnischen Nachteilen bietet Google Maps auch keine Möglichkeit, eigene Karten, welche zuvor geolokalisiert wurden, einzubinden.

Mit dem Projekt NeoMap, welches am Institut für Software der HSR ins Leben gerufen wurde, wird das Ziel verfolgt, eine vollständige offline-fähige Karten-Applikation auf der Basis von Android und OpenStreetMap zu entwickeln. Diese soll es dem Benutzer unter anderem ermöglichen, eigene geolokalisierte Karten einzubinden. Um hierbei eine möglichst leistungsfähige Lösung zu bieten, wird auf die hochperformante Grafikbibliothek OpenGL ES zur Darstellung der Karte gesetzt. Eine erste Version der App wurde bereits in einer Semesterarbeit entwickelt.

Der Fokus dieser Bachelorarbeit lag darin, die bestehende Version des NeoMap Apps zu überarbeiten und auf den neusten Stand der Technik zu bringen. Die Migration betraf vorwiegend die Umstellung von OpenGL ES 1.1 auf die Version 2.0 sowie die Verwendung des modernen Android 4.0 API.
Ein weiteres Hauptziel war es, die schlecht wart- und erweiterbare App zurück auf einen guten Stand zu bringen. Um dieses Ziel zu erreichen, wurde ein grosses Refactoring durchgeführt. Nachdem diese erste Phase erfolgreich abgeschlossen werden konnte, wurde die Applikation um weitere nützliche Features ergänzt, welche die Attraktivität der App weiter steigern sollen. Dazu gehören unter anderem eine 3D-Schrägansicht als Orientierungshilfe, das Erfassen von Notizen in Form von Punkten oder Routen auf der Karte sowie eine Geonamen-Suche. Bei all diesen Aspekten wurde der Grundvorsatz der Offline-Fähigkeit immer eingehalten.

Webseite: http://neomap.hsr.ch

Management Summary

Ausgangslage und Problematik
Mittels des Projektes NeoMap, welches aus einer Webseite sowie einer Android-Applikation besteht und in einer ersten Version von Studenten der HSR entwickelt wurde, wird eine Karten-Applikation geboten, welche eine konsequente Offline-Nutzung sowie das Einbinden von eigenen zuvor geolokalisierten Karten ermöglicht. Als Basis für das Kartenmaterial wird dabei auf das Open-Source und Community-Projekt OpenStreetMap gesetzt, welches erlaubt, Kartenmaterial frei von jeglichen Lizenzen einzusetzen. Diese beschriebenen Funktionen sind Kern der NeoMap App und heben diese damit auch entscheidend von anderen Karten-Apps ab. In der bereits vorhandenen Version der NeoMap App wurden leider grundlegende Prinzipien der Softwareentwicklung nicht oder nur am Rande eingehalten. Die Applikation ist aus diesem Grund sehr schlecht wart- und erweiterbar. Weiter ist eine ungenügende Dokumentation vorhanden, was die Pflege und Weiterentwicklung des Projektes zusätzlich erschwert. Die Funktionalität der App ist bisher sehr begrenzt, was für die Marktfähigkeit der App ein weiterer Negativpunkt ist. Durch diese Arbeit sollen alle beschriebenen Punkte verbessert werden. Zusätzlich soll auf das moderne Android 4.0 API gesetzt werden und eine Migration von OpenGL ES 1.1 auf 2.0 stattfinden. Danach soll die App durch nützliche Funktionen ergänzt werden.

Vorgehen
Als Vorgehensmodell wurde eine vereinfachte Version des bekannten Prozessmodells RUP (Rational Unified Process) angewendet. Dazu wurde das Projekt am Anfang zuerst in Phasen eingeteilt und eine Grobplanung für das Vorgehen erstellt. Bei der Planung wurde berücksichtigt, dass in den ersten Phasen das Aneignen von Wissen - im Bereich der Grafikprogrammierung allgemein sowie OpenGL im Speziellen - im Vordergrund stand. Anschliessend wurden eine ausführliche Analyse sowie ein Refactoring und schliesslich eine Migration des bestehenden Quellcodes auf eine gute Basis für weiterführende Entwicklungen durchgeführt. Gleichzeitig wurde dabei - vorausschauend auf weitere Features - die grundlegende Architektur der Applikation definiert und entsprechend umgesetzt.
In späteren Phasen wurde die Applikation durch zusätzliche Features erweitert. Dabei wurde der Aspekt der Offline-Fähigkeit nie vernachlässigt.

Ergebnisse
Am Schluss dieser Arbeit steht nun ein komplett überarbeitetes App, welches auf den neusten Technologien, Android in der Version 4, sowie Open GL ES 2.0, basiert. Ergänzt wurde das bestehende Feature-Set mit einer 3D-Schrägansicht der Karte, welche dem Benutzer als Orientierungshilfe zur Seite stehen soll. Dabei wird die Karte mittels im Gerät verbauten Sensoren ausgerichtet. Weiter wurde die Möglichkeit eingeführt, Notizen (als „Waypoints“) auf der Karte zu erfassen sowie ganze Routen einzuzeichnen und diese in das bekannte GPX-Format zu exportieren. Ausserdem wurde die Funktion der geografischen Namenssuche eingeführt: Möchte der Benutzer erfahren, wo sich ein Ort oder was sich an einer bestimmten Stelle auf der Karte befindet, so werden diese Informationen über einen von OpenStreetMap zur Verfügung gestellten Webservice abgerufen. Um die Offline-Fähigkeit nicht zu verletzen, werden einmal abgerufene Ergebnisse lokal gespeichert und stehen dem Benutzer bei nächster Verwendung auch ohne Internet zur Verfügung.

Ausblick
Die NeoMap App wurde nun auf eine gute Basis gebracht und hat durchaus Potential eine Alternative zu bestehenden Karten-Apps zu werden. Dieser Prozess ist jedoch noch nicht abgeschlossen und es sollten diesbezüglich Weiterentwicklungen stattfinden.

Als wohl erste und gleichzeitig wichtigste Erweiterung sollte eine Funktion zur Verfügung gestellt werden, welche das Herunterladen von definierbaren Kartenregionen erlaubt.

Dadurch können Benutzer bereits im Voraus alle relevanten Karten zu einer Region herunterladen, um diese beispielsweise später auf einer Wanderung zu nutzen.
Im Weiteren sollte das Zusammenspiel mit der NeoMap-Webseite besser ausgenutzt werden. Eine Möglichkeit wäre die Einführung einer Funktion, um erfasste Notizen über die Webseite zu teilen. Damit verbunden könnte auch OAuth für die Authentifizierung verwendet werden. Dazu sollten sicher Google- sowie Facebook-Accounts in Betracht gezogen werden, um mehr Benutzer zu ermutigen, sich auf der NeoMap-Seite zu registrieren und gleichzeitig eine Integration in die sozialen Netzwerke von Facebook und Google+ zu ermöglichen.
Ein anderes nützliches Feature wäre GPS-Tracking, welches die automatische Aufzeichnung von Routen ermöglicht. Auch dies sollte über die NeoMap-Webseite teilbar sein. Werden diese Funktionen umgesetzt, so könnte sich möglicherweise eine Community rund um die NeoMap App entwickeln.
Designtechnisch sollte das Navigationsprinzip weiter optimiert werden. An der diesjährigen Google I/O wurde eine neue Sidebar-Navigation unter dem Namen „Navigation-Drawer“ vorgestellt, welche gut in die App passen würde.

Webseite: http://neomap.hsr.ch

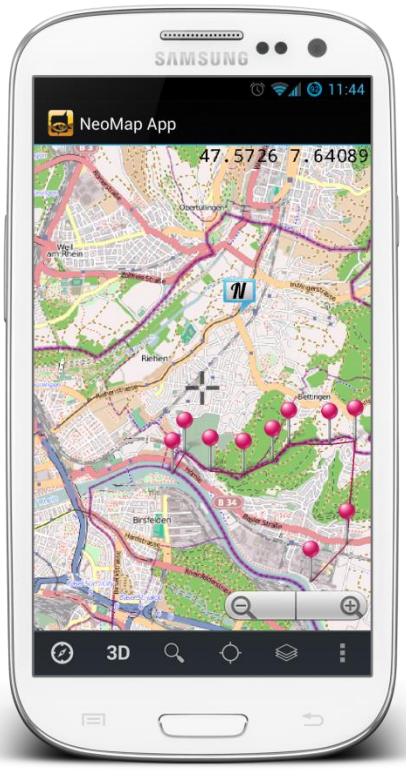


Abbildung 1: Karten Ansicht der NeoMap App

Abbildung 2: 3D-Schrägansicht der NeoMap App

Inhaltsverzeichnis

Aufgabenstellung

Jeder kennt Google Maps, das es auch auf Android Smartphones gibt. Doch fehlen diesem App drei entscheidende Funktionen:

- Es können dort keine eigenen Pläne und Karten dargestellt werden.
- Es ist nicht konsequent Offline.
- Es ist es nicht Open Source und man gibt einiges von seiner Privatsphäre preis.

Aufgaben

Das NeoMap-Projekt erfüllt diese Anforderungen (vgl. http://neomap.hsr.ch/). Das Projekt, insbesondere das App soll nochmals erweitert werden durch:

- Code Refactoring und Umstellen auf OpenGL ES 2
- Gerät kippen für Schrägansicht
- Notizen erfassen
- Geometrie erfassen und als GPX speichern
- Eigene Karten ein und ausblenden (auf Tablets mit Fragments)
- Optional: Geonamen-Suche (Offline-Fähigkeit)

Trotz vorhandener Software soll dies eine eigenständige Arbeit sein (Vorhandenes dokumentieren) mit verschiedenen Aspekten, von der Client-Programmierung (3D mit OpenGL) bis und zur Schnittstelle (RESTfull API).

Vorgaben/Rahmenbedingungen

- Die Projektabwicklung ist „RUP light“. Meilensteine werden bezüglich Termin und Inhalt mit dem Betreuer vereinbart.
- Die Kommunikation in der Projektgruppe, in der Dokumentation und an den Präsentationen erfolgt auf Deutsch.
- Je nach Zeitplanung soll ein Video gemäss den Vorgaben des Studiengangs realisiert und publiziert werden.
- Die Nutzungsrechte an der Software: MIT Lizenz.
- Ansonsten gelten die Rahmenbedingungen, Vorgaben und Termine der HSR

Inhalt der Dokumentation

- Die Projektdokumentation (Prosa) und die Benutzerschnittstelle sind in Deutsch. Der Code, die Kommentare und die Versionsverwaltung sind in Englisch.
- Die fertige Arbeit muss folgende Inhalte haben:
  1. Abstract, Management Summary, Aufgabenstellung
  2. Technischer Bericht
  3. Projektdokumentation
  4. Anhänge (Literaturverzeichnis, Glossar, CD-Inhalt)
- Die Abgabe ist so zu gliedern, dass die obigen Inhalte klar erkenntlich und auffindbar sind.
- Zitate sind zu kennzeichnen, die Quelle ist anzugeben.
- Verwendete Dokumente und Literatur sind in einem Literaturverzeichnis aufzuführen.
- Dokumentation des Projektverlaufes, Planung etc.
- Weitere Dokumente (z.B. Kurzbeschreibung, Poster) gemäss www.hsr.ch und gemäss Absprache mit dem Betreuer.

Form der Dokumentation

- Bericht (Struktur gemäss Beschreibung) gebunden (2 Exemplare) und in Ordner (1 Exemplar „kopierfähig“ in losen, gelochten Blättern).
- Alle Dokumente und Quellen der erstellten Software auf sauber angeschriebenen CD (3 Ex.).

Bewertungsschema

Es gelten die üblichen Regelungen zum Ablauf und zur Bewertung der Arbeit des Studiengangs Informatik der HSR mit besonderem Gewicht auf moderne Softwareentwicklung.

Beteiligte

Diplomanden

Patrick Zenhäusern
Simon Stähli

Industriepartner

- (Open Community / Open Source).

Betreuung HSR

Verantwortlicher Dozent: Prof. Stefan Keller (sfkeller@hsr.ch), Geometa Lab am IFS der HSR

Lizenzvereinbarung

Vereinbarung

1. Gegenstand der Vereinbarung

Mit dieser Vereinbarung werden die Rechte über die Verwendung und die Weiterentwicklung der Ergebnisse der Bachelorarbeit „NeoMap App Reloaded“ von Patrick Zenhäusern und Simon Stähli unter der Betreuung von Prof. Stefan Keller geregelt.

2. Urheberrecht

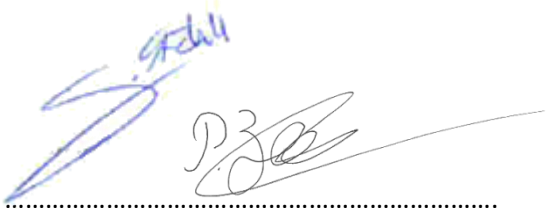
Die Urheberrechte stehen den Studenten zu.

3. Verwendung

Die Ergebnisse der Arbeit dürfen sowohl von der Studentin / dem Student und von der HSR nach Abschluss der Arbeit verwendet und weiterentwickelt werden.
Für die Software wird die MIT-Lizenz verwendet.

Beilage/n:

MIT Lizenz (deutsche Übersetzung): http://de.wikipedia.org/wiki/MIT-Lizenz



Rapperswil, den 02.04.2013 .....

Die Studenten

Rapperswil, den 02.04.2013 .....

Der Betreuer

Eigenständigkeitserklärungen

Simon Stähli

Ich erkläre hiermit,

- dass ich die vorliegende Arbeit selber und ohne fremde Hilfe durchgeführt habe, ausser derjenigen, welche explizit in der Aufgabenstellung erwähnt ist oder mit dem Betreuer schriftlich vereinbart wurde,
- dass ich sämtliche verwendeten Quellen erwähnt und gemäss gängigen wissenschaftlichen Zitierregeln korrekt angegeben habe.
- das ich keine durch Copyright geschützten Materialien (z.B. Bilder) in dieser Arbeit in unerlaubter Weise genutzt habe.

Ort, Datum: Rapperswil, 08.06.2013

Name, Unterschrift: Simon Stähli, 

Patrick Zenhäusern

Ich erkläre hiermit,

- dass ich die vorliegende Arbeit selber und ohne fremde Hilfe durchgeführt habe, ausser derjenigen, welche explizit in der Aufgabenstellung erwähnt ist oder mit dem Betreuer schriftlich vereinbart wurde,
- dass ich sämtliche verwendeten Quellen erwähnt und gemäss gängigen wissenschaftlichen Zitierregeln korrekt angegeben habe.
- das ich keine durch Copyright geschützten Materialien (z.B. Bilder) in dieser Arbeit in unerlaubter Weise genutzt habe.

Ort, Datum: Rapperswil, 08.06.2013.

Name, Unterschrift: Patrick Zenhäusern, 

Teil I
Technischer Bericht

1 Einführung

1.1 Projekt Fachbegriffe

In diesem Kapitel sind Projektfachbegriffe beschrieben. Dabei geht es nicht um die eigentliche Erklärung bzw. Bedeutung der Begriffe, sondern diese werden hier aufgeführt, um eine konsistente Verwendung im Dokument zu erhalten.

Begriff	Beschreibung
NeoMap	Bezeichnet eine eigene geolokalisierte Karte
NeoMap App / App	Steht für die NeoMap App, welche für die Android-Plattform entwickelt wurde
NeoMap Webseite	Steht für die Webseite zum NeoMap App Verfügbar unter http://neomap.hsr.ch

Tabelle 1: Projektfachbegriffe

1.2 Vision

Der Quellcode von NeoMap soll wieder auf einen guten Stand gebracht werden, so dass eine einfache Erweiterbar- sowie Wartbarkeit gegeben sind. Dabei soll auf moderne, zukunftsorientierte APIs zugegriffen werden. Altlasten wie die Kompatibilität zu alten Geräten sollen dabei nicht mitgetragen werden.
Weiter sollen noch mehr für den Benutzer nützliche Funktionen, wie zum Beispiel eine 3D-Schrägansicht als Orientierungshilfe, eingeführt werden. Damit soll die Attraktivität der App weiter gesteigert und die Konkurrenzfähigkeit erhöht werden.
Der Hauptfokus der NeoMap App - alle Funktionen online sowie offline anzubieten - soll im Rahmen dieser Arbeit beibehalten werden.

1.3 Ziele und Unterziele

Folgende Ziele sind hauptsächlich aus der Aufgabenstellung extrahiert. Zusätzlich aufgeführt sind auch Merkmale, welche uns persönlich wichtig sind.

- Migration der bestehenden Rendering-Engine von OpenGL ES 1.1 zu OpenGL ES 2.0
  - Eine Challenge stellt sich hierbei auch bei der Aneignung von Wissen über das Zeichnen von Computer-Grafiken und OpenGL im Speziellen
- Migration von Android-Version 2.3 auf die Version 4.0
  - Einführen / Einsetzen von Fragments wo sinnvoll
  - Nutzen der neusten Features der Android-APIs ab Version 4.0 (z.B. Action Bar)
- Grosses Refactoring des Quellcodes (vor allem im Bereich der Karten-Darstellung)
  - Klares Ziel ist es, eine gute Erweiterbarkeit sowie auch Wartbarkeit sicherzustellen
- Architekturdokumentation der wichtigsten, auch bereits bestehenden Teile der Applikation erstellen.
- Aufräumen / Bereinigen des offiziellen NeoMap Projektverwaltungssystems Redmine
  - Commits direkt mit Tickets verknüpfen → dies bietet eine schnell ersichtliche Dokumentation direkt über das Redmine
- Schrägansicht / 3D-Ansicht als Orientierungshilfe beim Kippen des Gerätes einführen
- Notizen sollen auf der Karte als einzelne Punkte oder Routen erfasst werden können
- Erfasste Punkte oder Routen sollen mittels GPS Exchange Format (GPX) in eine Datei exportieren werden können

- Eigene NeoMap-Karten einfach ein- und ausblenden, sowie deren Darstellungsreihenfolge bestimmen können
  - Für diese Einstellungen sollen Fragments verwendet werden
- **Optional**: Geografische Namenssuche einführen
  - Bei der ersten Suche wird dabei eine Internetverbindung benötigt, danach offline verfügbar

1.4 Rahmenbedingungen

Diese Arbeit wird im Rahmen einer Bachelorarbeit an der Hochschule für Technik in Rapperswil (HSR) durchgeführt. Somit gelten die grundsätzlichen Vorgaben, Termine und Beurteilungskriterien der HSR.

1.5 Aufbau der Arbeit

Die Dokumentation dieser Arbeit wird wie folgt in vier Teile gegliedert.

Aufteilung	Beschreibung
Teil I Technischer Bericht	Dieser Teil beinhaltet eine Einführung in die Thematik, eine Übersicht über die gesamte Arbeit sowie einen Abschnitt über die Resultate dieser Arbeit.
Teil II SW-Projektdokumentation	In diesem Teil befindet sich die Dokumentation über den Prozess der Entwicklung. Dazu gehören Anforderungsspezifikationen, Architektur-Dokumentation sowie das Testen der Applikation.
Teil III Projektmanagement	In diesem Teil finden sich Details zur Planung und Ausführung des Projektes.
Teil IV Anhänge	Beschreibt die Inhalte der abgegebenen CD und enthält des Weiteren das Glossar und alle Verzeichnisse.

Tabelle 2: Aufbau der Arbeit

2 Stand der Technik

2.1 Android

Android ist heutzutage das auf Smartphones meist genutzte Betriebssystem. An der Google I/O 2013 gab Google bekannt[1], dass bereits über 900 Millionen Android-Geräte aktiviert sind. Dies ist ein Wachstum von 125% im Vergleich zum Jahr 2012 (400 Millionen) und ein Wachstum von 800% zum Jahr 2011 (100 Millionen). Das Wachstum scheint momentan kein Ende zu nehmen und die Anzahl Aktivierungen wird vermutlich weiter stark ansteigen.

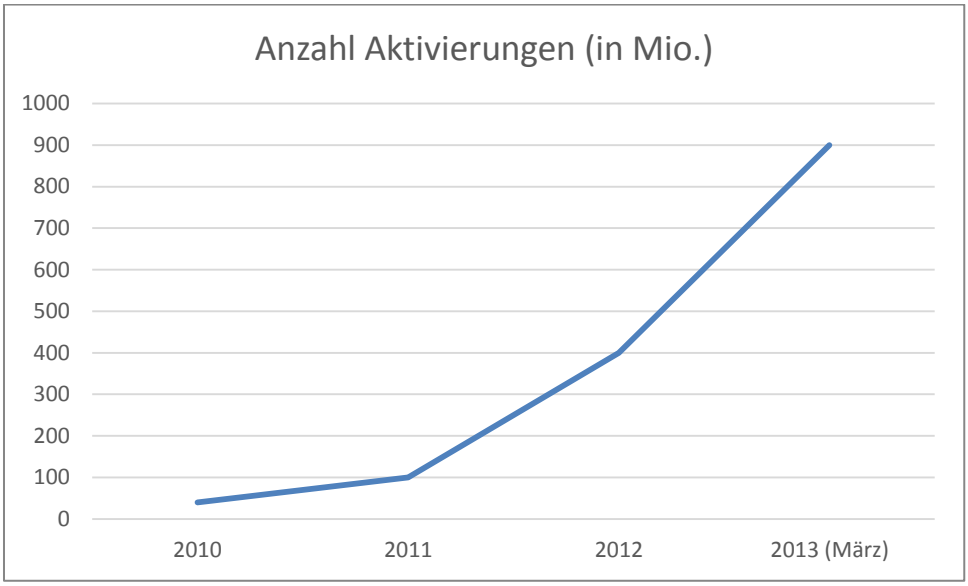


Diagramm 1: Anzahl Aktivierungen von Android-Geräten pro Jahr

Google gab auch bekannt, dass die Reise erst begonnen habe und es das Ziel sei, möglichst viele Leute online zu bringen. Nicht zuletzt auch in Schwellen- und Entwicklungsländern.

Im Weiteren hat Google Play, der Marktplatz für Apps, laut offiziellen Angaben von Google die Marke von 48 Milliarden App-Installationen durchbrochen. Auch gab Google bekannt, dass von Android-Benutzern in den ersten vier Monaten des Jahres 2013 bereits mehr Geld für Apps ausgegeben wurde als insgesamt im Jahr 2012.

Diese beeindruckenden Zahlen zeigen, dass die Entwicklung von Apps für die Android-Plattform immer lukrativer und interessanter wird. Android ist momentan die Plattform, auf welcher durch Apps am meisten Menschen erreicht werden können.

[1] Google I/O 2013: Keynote, «youtube.com,». [Online]. http://www.youtube.com/watch?v=9pmPa_KxsAM.

2.2 Karten auf mobilen Geräten

Ein wichtiger Bestandteil jedes mobilen Gerätes ist heutzutage eine Karten-Applikation. Gerade durch die einfache Lokalisierungsmöglichkeit mittels eingebauten GPS-Sendern, hat sich die Popularität stark gesteigert. In einigen Fällen können die Karten-Apps sogar direkt zur Navigation verwendet werden. Es existiert bereits eine Vielzahl von bekannten Lösungen. Einige davon sollen hier kurz beschrieben und verglichen werden. Bei den Vergleichskandidaten wurden möglichst populäre Lösungen ausgewählt, welche einen ähnlichen Funktionsumfang wie die NeoMap App bieten.
Damit ein guter Vergleich stattfinden kann, soll an dieser Stelle nochmals die NeoMap App, im Zustand nach Abschluss dieser Bachelorarbeit, beschrieben werden.

<table>
<tr><th>NeoMap App</th><th>Funktionalität</th><th>Karten: OpenStreetMap</th></tr>
<tr><td></td><td colspan="2">• Offline-Verfügbarkeit von allen Funktionen
• NeoMaps – eigene Karten, die über die Webseite hochgeladen werden können
• Erfassen von Notizen in Form von Punkten (Wegpunkten) und Linien (Routen, Abgrenzungen etc.) und Export in GPX-Dateien
• Geografische Namenssuche (sowie Offline-Verfügbarkeit von Suchbegriffen)
• 3D-Schrägansicht zur Orientierung
• Viele Einstellungsmöglichkeiten: Eigene Tile-Server (Adresse für Karten), fast alles ein- und ausblendbar usw.
• Android 4.0 und OpenGL ES 2.0
• Oberfläche stark an Android-Standards angelehnt
• OpenSource
• MIT-Lizenz</td></tr>
<tr><th>Rating im Playstore</th><th colspan="2">Preis</th></tr>
<tr><td>-</td><td colspan="2">- (Wird sicher gratis sein, ist jedoch noch nicht im Playstore)</td></tr>
<tr><th colspan="3">Beschreibung</th></tr>
<tr><td colspan="2">NeoMap App ist eine gratis Applikation für Android, welche eigene Karten („NeoMaps“) anzeigen kann. Gleichzeitig können eigene Karten über die NeoMap Webseite (http://neomap.hsr.ch) mit anderen Benutzern geteilt werden.

NeoMap App ist eine vollständige Karten-App, welche OpenStreetMap-Karten als Basis-Karte nutzt. Einmal angesehene Karten werden gespeichert und können auch offline verwendet werden.

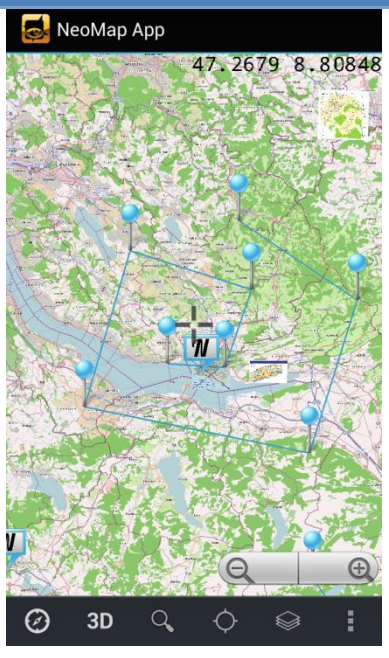
Zudem bietet NeoMap App viele weitere Funktionen wie zum Beispiel:
• Geografische Namenssuche (mit Offline-Verfügbarkeit)
• 3D-Schrägansicht als Orientierungshilfe
• Erfassen von Notizen als Punkte oder Routen auf der Karte
• Export von Notizen im GPX-Format
• Und vieles mehr…</td><td> 
</td></tr>
</table>

Tabelle 3: NeoMap App Übersicht

2.2.1 Vergleichbare Lösungsansätze

Google Maps	**Funktionalität / Vergleich mit NeoMap**	**Karten:** Google Karten
	• Populärste Kartenapplikation • Sehr stabil, mit sehr vielen Funktionen • Lizenz-Technische Nachteile (Keine Garantie, dass alles für immer gratis bleibt) • Kein OpenSource (Abhängigkeit von Google) • Keine konsequente Offline Verfügbarkeit von Karten (nur Ausschnitte können heruntergeladen werden und dieser werden nach einer bestimmten Zeit wieder gelöscht) • Richtige 3D Ansicht (Gebäude etc.) • Erfassen von Notizen nicht möglich • Keine eigenen Karte • Wenig Mitspracherecht für künftige Features etc. von der Community	
Rating im Playstore	**Preis**	
4.4 (2‘513‘924)	- (Gratis)	
Beschreibung (aus Android Playstore)		
Mit der neuesten Version von Google Maps können Sie auf Stadtpläne und Landkarten komplett verzichten. Laden Sie das Programm jetzt herunter. Google Maps für Android mit Navigation (Beta) bietet: • Detaillierte Karten mit 3D-Gebäuden • Sprachgestützte detaillierte GPS-Routenführung • Wegbeschreibungen für Autofahrer, Radfahrer, Fußgänger und öffentliche Verkehrsmittel Aktuelle Verkehrsinformationen zum Umgehen von Staus • Lokale Suche und Erfahrungsberichte zu Geschäften • Google Maps Street View • Gebäudepläne für ausgewählte Flughäfen, Hotels, Geschäfte und mehr Egal, ob Sie ein Restaurant in der Nähe suchen, die Wegbeschreibung zu einem bestimmten Ort oder Informationen zu Ihrer Umgebung brauchen, Google Maps für Android hilft Ihnen dabei.		

Tabelle 4: Google Maps

Geopaparrazi	Funktionalität / Vergleich mit NeoMap	**Karten**: OpenStreetMap
	• Viele Funktionen um Geodaten zu erfassen • Notizerfassung einfach möglich, zeigt auch sehr gut Position auf Karte an, wo sich die erfassten Notizen befinden • Gewöhnungsbedürftiges Interface: Merkwürdiger Startbildschirm, „Notfall“ Optionen sind verwirrend, etc. • Zielgruppe ist hier wohl eher die „Datensammler“ → Kleine Zielgruppe • Bietet keine Offline Funktionalität an	
Rating im Playstore	**Preis**	
4.4 (58)	- (Gratis)	
Beschreibung (aus Android Playstore übersetzt)		
Geopaparazzi ist ein Tool, welches entwickelt wurde um schnell qualitativ hochwertige und wissenschaftlich korrekte Aufnahmen von Geodaten zu machen. Seine Stärken liegen darin, dass es über eine direkte Verbindung zu BeeGIS GIS verfügt, welches dann zur Verfeinerung der Daten benutzt werden kann. Obwohl das Hauptziel der App ist Geodaten zu erfassen, verfügt es trotzdem über einen grossen Nutzen für Touristen um ein „Geo-Tagebuch“ zu führen oder sich zu orientieren. Webseite: www.geopaparazzi.eu		

Tabelle 5: Geopaparrazi

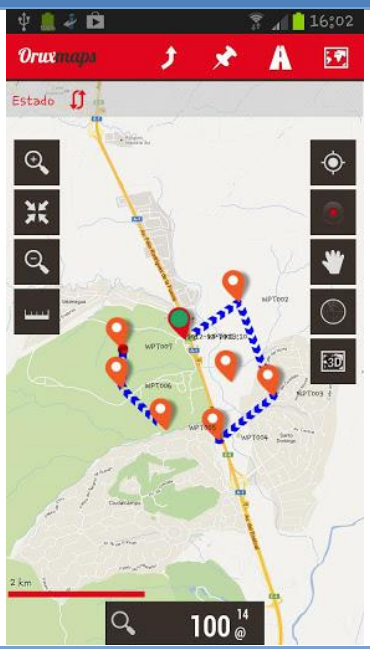
OruxMaps	**Funktionalität / Vergleich mit NeoMap**	**Karten**: OpenStreetMap /Google Maps/ Bing Maps
	• Kann Notizpunkte und Routen erfassen, auch mit Auto-Tracking • Stellt auch Offline Funktionalität zur Verfügung, allerdings nur für einen Teil der Funktionen • Bietet die Wahl zwischen verschiedenen Kartenanbietern an (sehr flexibel) • Eigene Karten können eingefügt werden • Sehr viele Funktionen • Bietet auch eine 3D Kipp Funktion, allerdings ohne die Sensoren zu beachten (kein Eigener Standort, kein Mitdrehen etc.) • Unterstützt Vektor-Grafik-Karten • Interface wirkt teilweise überladen, kein roter Faden • Leider nicht sehr international → wird vor allem von Spaniern gepflegt und es wird auch in den offiziellen Foren hauptsächlich in Spanisch kommuniziert	
Rating im Playstore	**Preis**	
4.6 (9'751)	- (Gratis)	
Beschreibung (aus Android Playstore übersetzt und gekürzt)		
OruxMaps ist eine Kartenansichts- und Routenerfassungsapplikation für Outdoor-Aktivitäten. OruxMaps funktioniert in zwei Modi: • Online mit vielen verschiedenen Karten-Typen (Google Maps, OpenStreetMap, Microsoft Maps, etc.) • Offline mit georeferenzierten Karten (Diese können selbst erstellt werden oder es können Ozi Explorer Maps konvertiert werden) Erlaubt das Aufzeichnen von Routen mittels GPS. Die Routen können im GPX oder im KML-Format gespeichert werden. Und viele weitere Funktionen. Webseite: www.oruxmaps.com		

Tabelle 6: OruxMaps

OsmAnd Karten & Navigation	Funktionalität / Vergleich mit NeoMap	Karten: OpenStreetMap
	• Offline Funktionalität für alle Karten • Oberfläche optisch nicht ansprechend und nicht aufgeräumt • App legt Fokus eher in Richtung Navigation (Fahrzeug, Velo, zu Fuss) • Es kann direkt bei OSM mitgearbeitet werden (fehlende Adressen ergänzen etc.) • OpenSource	
Rating im Playstore	**Preis**	
4.3 (5'465)	- (Gratis) → OSMAnd + Premium Version kostet 5.99 Fr.	
Beschreibung (aus Android Playstore)		
OsmAnd ist eine Landkarten- und Navigations-Applikation basierend auf dem kostenlosen, globalen, qualitativ hochwertigen OpenStreetMap (OSM) Kartenmaterial. Alle Karten können zur Offline-Verwendung auf dem Handy oder Tablet PC gespeichert werden. Mittels GPS kann OsmAnd auch navigieren (optische Richtungsanzeige und Sprachansagen), mit Auto-, Fahrrad- oder Fußgängermodus. Alle Grundfunktionen funktionieren sowohl online wie auch offline (ohne Internetzugang). Kernfunktionen: • Navigation • Landkartenfunktion (Eigene Position, Blickrichtung ,etc.) • Nutzung von OpenStreetMap und Wikipedia-Daten • Sicherheitsfunktionen (z.B. Anzeige von Geschwindigkeitsbegrenzung) • Fahrrad und Fussgängerfunktionen • Direkte Mitarbeit bei OSM Möglich		

Tabelle 7: OSMAnd Karten & Navigation

2.2.2 Fazit

Die NeoMap App erfindet das Rad sicherlich nicht neu und viele andere Apps bieten ähnliche oder sogar mehr Funktionalität an. Allerdings sind viele dieser Karten-Apps bereits sehr überladen und haben kein konsequentes Konzept. Der vermutliche grösste Konkurrent ist hierbei wohl OruxMaps. Die App deckt fast alle Features ab, die NeoMap App bietet, mit Ausnahme der geografischen Namenssuche und anderen kleinen Unterschieden. Sie bietet noch viel mehr Features an. Dies ist aber vielleicht auch gerade das Problem: OruxMaps hat zwar ein schönes User-Interface, ist aber relativ überladen und entsprechend kompliziert zu bedienen. NeoMap App hingegen überzeugt durch ein aufgeräumtes, einfaches Interface und durch ein konsequentes Konzept: Alle Funktionen sind auch offline Verfügbar. Ob dies jedoch ausreicht um Popularität zu gewinnen und ein ernsthafter Konkurrent der anderen Apps zu werden, bleibt abzuwarten und ist unter anderem auch davon abhängig, wie bzw. ob das Projekt weiter gepflegt und mit sinnvollen Features bereichert wird.

2.3 OpenGL ES

OpenGL ES (Open Graphics Library for Embedded Systems) wurde als Subset der OpenGL 3D API, eine Spezifikation für eine Plattform- und Sprachenunabhängige API zur Entwicklung von 2D- und 3D Computergrafiken, für Embedded Systeme entwickelt und im Jahre 2003 von der Khronos Group angekündigt. OpenGL ES hat über die Jahre sehr schnell an Popularität gewonnen, nicht zuletzt dank neuen und schnell wachsenden Plattformen wie Android, iPhone oder auch Play Station 3.

Im März 2007 wurde von der Khronos Group die OpenGL ES 2.0 Spezifikation freigegeben. Die grösste Änderung, welche dabei eingeführt wurde, ist das Konzept der programmierbaren Pipelines. Durch die programmierbaren Pipelines stehen den Entwicklern mehr Freiheiten sowie eine bessere Kontrolle beim Erstellen von Grafik-Elementen zur Verfügung. Das Konzept der programmierbaren Pipelines wird durch die sogenannte Shading Language realisiert, welche eine „C“ ähnliche Syntax besitzt und direkt auf der Grafikkarte bzw. deren GPU ausgeführt wird. Die zusätzliche Freiheit kommt jedoch auch mit dem Preis einer höheren Komplexität bei der Entwicklung. Viele zuvor in OpenGL ES 1.x standardmässig zur Verfügung gestellte Funktionalitäten, müssen nun erst in einem eigenen Shader implementiert werden. Dies führt auch dazu, dass dieses neue Konzept die Rückwärtskompatibilität von OpenGL ES 2.0 zu früheren Versionen bricht. (P. Rideout 2010)

2.3.1 OpenGL ES unter Android

Laut der offiziellen Statistik von zum Stand des 1. Oktobers 2012 unterstützen bereits 90.8% aller Geräte OpenGL ES 2.0, lediglich 9.2% haben nur eine Unterstützung für die Version 1.1
Dabei handelt es sich um Geräte mit einer Android-Version 2.2 oder tiefer. OpenGL ES 2.0 wird vom Android SDK seit der Version 2.2 (API 8) unterstützt. (android.com 2012)

Laut der offiziellen Android-Versions-Statistik, gemessene Zugriffe innerhalb der letzten 14-Tagen auf den Google-Play-Store, Stand 4. Februars 2013, werden lediglich noch 2.4% aller Android-Geräte mit einer tieferen Version als 2.2 betrieben. Da NeoMap jedoch sowieso als Ziel-Version Android 4.0 besitzt, können frühere Versionen in diesem Falle ausser Acht gelassen werden. Bereits 42.6% aller Android-Geräte befinden sich zum Zeitpunkt dieser Statistik (4. Februar 2013) bereits auf der Version 4.0 oder sogar höher. Somit kann davon ausgegangen werden, dass alle Geräte des Zielsegmentes der NeoMap App OpenGL ES 2.0 unterstützen und eine Rückwärtskompatibilität zu OpenGL ES 1.1 nicht beachtet werden muss.

2.3.2 Vorteile OpenGL ES 2.0

Durch die programmierbaren Pipelines und den Gebrauch von Shadern werden dem Entwickler eine erhöhte Kontrolle und mehr Freiheiten beim Erstellen von Grafiken ermöglicht.
Zusätzlich ist bei grösseren Operationen eine Performance-Optimierung vorhanden, da die GPU der Grafikkarte direkt eigenen, optimierten Code ausführen kann. In Falle der NeoMap App sind die Performance-Optimierungen von kleinerer Relevanz, jedoch soll mit der Migration auf OpenGL ES 2.0 ein solides und zukunftsorientiertes Design der Rendering-Architektur von NeoMap geschaffen werden. Dies auch im Hinblick auf das kommende OpenGL ES 3.0, welches auf der Version 2.0 aufbaut und vermutlich mit Android 4.3 oder 5.0 kommen soll.

3 Basiswissen

Um einen möglichst einfachen Einstieg in die Thematik der Kartografie sowie die verwendeten Technologien dieses Projektes zu ermöglichen, wird an dieser Stelle eine kurze Einführung zu den verschiedenen Aspekten abgehandelt.

3.1 Android

Dieses Kapitel soll als eine kurze Einführung in Android als Betriebssystem auf mobilen Geräten sowie die Möglichkeiten zur Entwicklung von Applikationen für Android darlegen. Das grundlegende Wissen zu diesem Thema wurde dem Buch „Professional Android 4 Application Development“ von Reto Meier, führender Mitarbeiter von Google im Zusammenhang mit Android-Projekten, entnommen und kann dort bei Bedarf noch genauer nachgelesen werden.

3.1.1 Kurze Historie

Das Betriebssystem Android für mobile Endgeräte wurde von Google erstmals an einer Konferenz im Jahre 2007 vorgestellt. Ursprünglich wurde es von der Firma Android ins Leben gerufen, welche jedoch von Google aufgekauft wurde. Zusammen mit 33 weiteren Mitgliedern gründete Google die Open Handset Alliance, welche die Weiterentwicklung von Android zum Ziel hatte. Seit dem 21. Oktober 2008 ist Android offiziell verfügbar.

3.1.2 Architektur

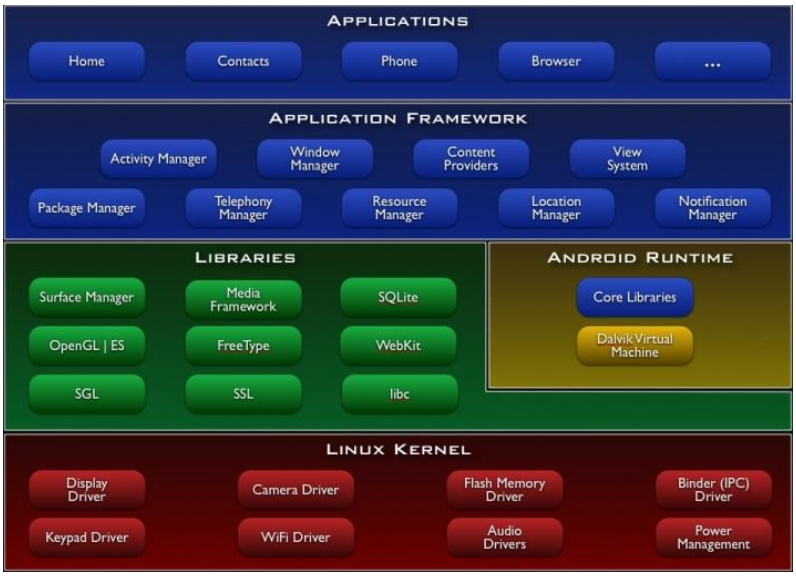


Abbildung 3: Offizielle Visualisierung der Android Architektur (developers.android.com)

3.1.2.1 Kernel

Die gesamte Architektur von Android baut auf dem Linux Kernel 2.6 auf. Er dient als Schnittstelle zur Hardware und führt die Speicher- und Prozessverwaltung aus.

3.1.2.2 Libraries und Android Runtime

Auf diesem Layer aufgesetzt befindet sich die Android Runtime, welche aus einigen Core Libraries sowie der Dalvik Virtual Machine (DalvikVM) besteht. Daneben stehen eine Vielzahl von C/C++ Libraries zur Seite, welche dem Benutzer über das Android Application Framework zur Verfügung gestellt werden.

Einige der wichtigsten Libraries, welche auch im NeoMap App zum Einsatz kommen:

- Surface Manager
- Datenbank (SQLite)
- 3D Graphics (OpenGL ES 2.0 Implementierung)

Anwendungen für die DalvikVM werden mittels Java programmiert. Beim Android-Framework selbst ist jedoch ein Grossteil in C/C++ realisiert.
Im Mittelpunkt der Android Runtime steht die DalvikVM, eine abgeleitete Form der JavaVM. Jede Android-Applikation läuft als eigener Prozess mit einer eigenen Instanz der DalvikVM. Dalvik wurde optimiert um mehrere VM-Instanzen gleichzeitig auf effiziente Weise laufen zu lassen. Ein wesentlicher Unterschied zwischen der JavaVM und der DalvikVM besteht in der verwendeten Prozessorarchitektur. Bei der JavaVM kam der Kellerautomat zum Einsatz, DalvikVM hingegen basiert auf einer Registermaschine. Dies ist auch der Grund, weshalb Java-Compiler Bytecode nicht ohne weiteres in einer Dalvik-VM laufen gelassen werden kann. Hier wird der Dalvik Cross-Assembler benötigt, welcher jedoch von Google zur Verfügung gestellt wird und bei der Entwicklung im Zusammenspiel mit entsprechenden Plugins automatisiert ausgeführt wird.

3.1.2.3 Application Framework

Ein in Java geschriebenes Framework, welches Schnittstellen zur Entwicklung von eigenen Anwendungen anbietet. Hierbei werden dieselben Funktionalitäten angeboten, wie sie auch von den Core Android Applications genutzt werden.

3.1.2.4 Applications

Android wird jeweils mit vorinstallierten Applikationen, wie Email-Client, SMS Programm, Kalender, Browser, Kontakt-Verwaltung etc. ausgeliefert.

3.1.3 Entwickeln von Android-Applikationen

3.1.3.1 Kern Applikations-Komponenten

Der Kern einer Applikation wird jeweils durch nachfolgende Elemente gebildet:

Komponente	Zuständigkeit
Activities	Repräsentieren einen einzelnen Screen mit einem User Interface. Eine Android-Applikation kann durchaus mehrere Activities besitzen, welche verschiedene Screens repräsentierten.
Fragments	Fragments können nicht ohne eine Activity laufen. Man kann sie als Module verstehen, welche in Activities eingesetzt und ausgetauscht werden können. Im Gegensatz zu Activities können mehrere Fragments gleichzeitig aktiv sein bzw. angezeigt werden.
Services	Services sind eigenständige Komponenten, welche im Hintergrund laufen, meist werden in ihnen lang dauernde Operationen durchgeführt welche periodisch immer wieder durchlaufen werden müssen. Des Weiteren besitzen sie kein User Interface.
Content Providers	Da eine strikte Trennung von Applikationen mittels eigenen DalvikVM-Instanzen erfolgt, muss eine Möglichkeit geboten werden um Daten zwischen Applikationen, bspw. Kontakte und Kalender, zu teilen. Hierfür wurden Content Providers eingeführt. Ein Content Provider ist eindeutig über seine URI adressierbar und über ihn können Daten abgefragt werden.
Broadcast Receivers	Mittels Broadcast Receivern kann auf sogenannte Intents gewartet werden. Intents können von Anwendungen oder aber auch vom System verschickt werden (Low Battery, Connected to WLAN etc.)

Tabelle 8: Android-Applikationskomponenten

3.1.3.2 Manifest

Über das Manifest wird festgelegt, aus welchen Komponenten eine Android-Applikation besteht sowie welche Geräte-Funktionen eine Applikation in Anspruch nimmt. Auch wird darin festgelegt, welche Berechtigungen eine Applikation benötigt. Das Manifest ist sozusagen die Basis-Konfiguration für ein App.

3.1.3.3 Ressourcen

Eine Android-Applikation besteht jedoch aus mehr als nur Code. Es gehören auch XML-Layout Files, Bilder, ausgelagerte Texte, Audio Files etc. zur Applikation. Um hierbei eine saubere Trennung zum eigentlichen Programmcode zu erhalten, sind diese in sogenannte Resources ausgelagert. Diese können dann über den Java-Code referenziert werden.

3.1.3.4 Activity-Lebenszyklus

Eine Eigenart von Android, welche beim Entwickeln von Applikationen beachtet werden muss, ist der sogenannte Activity-Lebenszyklus. Das Android-Betriebssystem verwaltet Activities auf einem Stack. Es ist jeweils nur gerade die oberste Activity auf dem Stack aktiv. Inaktive Activities rutschen beim Starten von neuen Activities jeweils weiter nach unten. Kommt es nun zu Engpässen im Bereich des Speicher- oder Prozess-Managements, werden Activities, welche sich zuunterst auf dem Stack befinden, als erstes beendet und entfernt.

In der nachfolgenden Abbildung ist der Lebenszyklus einer Activity noch genauer beschrieben, auch sind Lifecycle-Methoden eingezeichnet, bei welchen man informiert wird, in welchem Status sich die Applikation befindet. Dies kann beispielsweise genutzt werden, um Informationen über die zuletzt getätigten Benutzer-Eingaben zu persistieren sobald das App in den Hintergrund verdrängt wird. Bei einer erneuten Benutzung des Apps, kann so wieder genau an derselben Stelle, mit exakt denselben Daten gestartet werden.

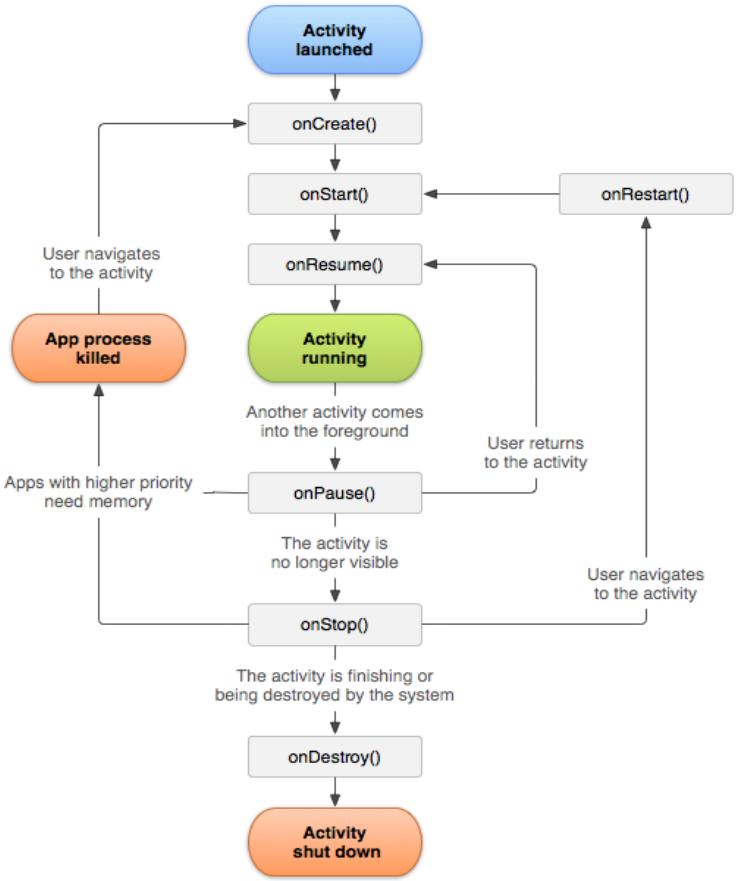


Abbildung 4: Offizielle Visualisierung des Activity Lifecycles (developer.android.com)

Hierbei können drei grundsätzliche Zyklen entnommen werden.

Lifetime	Zyklus (von – bis)	Beschreibung
Entire	onCreate – onDestroy	Die gesamte Lebenszeit einer Activity, von der Instanziierung bis zur Ressourcen-Freigabe.
Visible	onStart - onStop	Die Activity ist in diesem Zustand grundsätzlich sichtbar, wird jedoch von einer anderen Activity überdeckt. Es ist somit nicht möglich mit ihr zu interagieren.
Foreground	onResume - onPause	Während dieser Zeit ist die Activity im Vordergrund und interagiert mit dem Benutzer.

Tabelle 9: Lifetime-Zyklen von Android Apps

3.2 Grundlagen zum Aufbau von OpenStreetMap-Karten

Bei OpenStreetMap handelt es sich um ein im Jahre 2004 gegründetes Projekt, welches sich das Ziel gesetzt hat, eine freie Weltkarte zu schaffen. Seither werden alle möglichen geografischen Daten gesammelt und in Karten-Material umgesetzt. OpenStreetMap-Daten dürfen von jedermann lizenzkostenfrei eingesetzt und beliebig weiterverarbeitet werden. Es werden hierbei nicht nur die Karten als solches frei angeboten, es besteht auch die Möglichkeit die rohen Geo-Daten zu beziehen und diese so zu nutzen wie man möchte. (F. Ramm 2010)

3.2.1 Aufbau einer OpenStreetMap Slippy Map

Bei der sogenannten Slippy Map (deutsch: flinke Karte) handelt es sich um eine Karten-Ansicht, bei welcher die Karte frei nach links und rechts bzw. oben und unten bewegt sowie zwischen Zoom-Stufen gewechselt werden kann. Dies ist nichts anderes, als der bekannte, normale Kartenansichtsmodus, den man überall findet.
Beim dargestellten Karten-Material handelt es sich um von einem Server vorbereitete (pre-rendered) Kachel-Grafiken, welche dann auf einem Raster zu einer Karte zusammengesetzt werden.
Ein einzelnes Tile (Kachelgrafik) wird dabei durch dessen x- und y-Koordinaten auf dem Raster sowie dessen Zoom-Level identifiziert.

3.2.1.1 Zoom-Stufen

Die Zoom-Stufen wurden von OpenStreetMap von 0 bis 18 definiert. Wobei 0 ganz herausgezoomt ist, und 18 die genaueste Zoom-Stufe repräsentiert. Über die Zoom-Stufe kann somit berechnet werden, mit wie vielen Tiles die gesamte Weltkarte abgedeckt werden kann. Durch die verschiedenen Zoom-Stufen, ergibt sich ein Kartenaufbau, welcher die Form einer Pyramide hat.

Formel: n = 2^zoomStufe * 2^zoomStufe

Zoomstufe	Anzahl Tiles	Total
0	$2^0 \times 2^0$	1
1	$2^1 \times 2^1$	2
2	$2^2 \times 2^2$	4
12	$2^{12} \times 2^{12}$	16 777 216
18	$2^{18} \times 2^{18}$	68 719 476 736

Tabelle 10: Anzahl Tiles bei verschiedener Zoom-Stufe

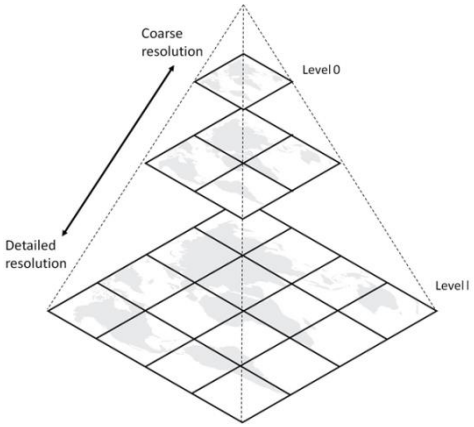


Abbildung 5: Karten-Pyramide

3.2.1.2 Tile-Koordinaten:

Wie bereits erwähnt, werden Tiles durch deren X- und Y-Koordinaten innerhalb des Rasters der aktuellen Zoom-Stufe identifiziert. Hierbei wird in der oberen linken Ecken des Rasters gestartet, X startet bei 0 und geht bis $2^{\text{^Zoom-Stufe}-1}$. Dasselbe gilt für Y, welches bei 0 startet und unten bei $2^{\text{^Zoom-Stufe}-1}$ endet.

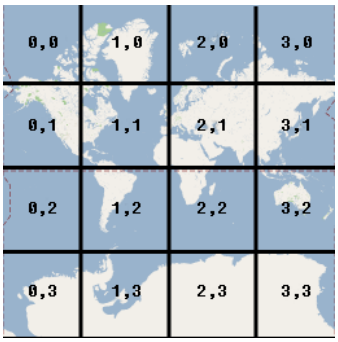


Abbildung 6: Tile-Koordinaten

3.2.1.3 Tile-Server

Einzelne Tiles können über sogenannte Tile-Server bezogen werden.

Diese liefern die einzelnen Tiles meist in einer Auflösung von 256 x 256 Pixeln. Der Zugriff auf Tiles erfolgt über einen http-Aufruf, bei welchem sich der Pfad auf dem Server stets gleich zusammensetzt. Für jede Zoom-Stufe wird auf dem Server ein eigenes Verzeichnis angelegt, in welchem wiederum für jede Spalte von Tiles (X-Koordinate) ein Verzeichnis angelegt wird. In diesem befinden sich letztendlich die einzelnen Tiles, eindeutig identifiziert durch Ihre Y-Koordinate.
Tile-Datei-Pfad: …/zoom/x/y.png

3.3 Basiswissen Kartographie

Bei der Kartendarstellung wird auf die Mercator-Projektion zurückgegriffen. Bei dieser winkelgetreuen Projektion bleiben geometrische Formen auch im Kleinen noch korrekt erhalten. Hingegen ist die Projektion nicht flächentreu, sprich Flächen derselben Grösse haben an unterschiedlichen Stellen auf der Karte nicht dieselben Abmessungen.

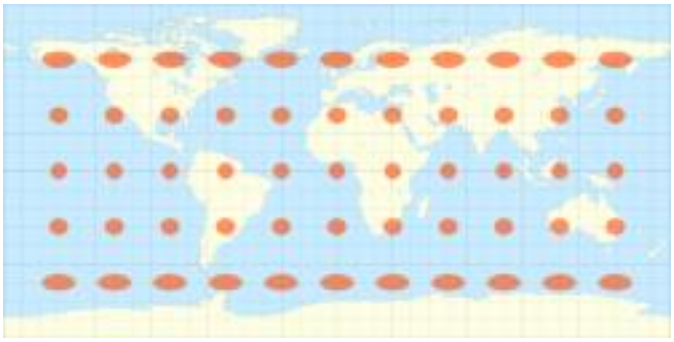


Abbildung 7: Weltkarten nach Mercator-Projektion

3.3.1 Geografische Koordinaten

Um die geografischen Koordinaten zu erhalten wird die gesamte Erde in 360 Längengrade sowie 180 Breitengrade unterteilt. Hierbei verlaufen Längengrade vom Nord- zum Südpol und Breitengrade parallel zum Äquator. Jeder Punkt auf der Erde kann somit exakt durch seine geografischen Koordinaten identifiziert werden.

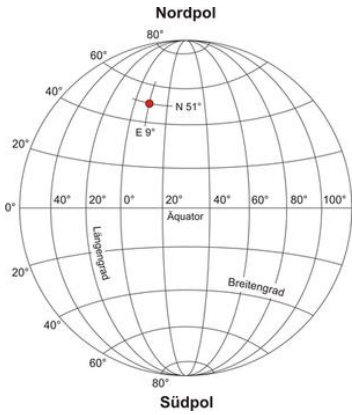


Abbildung 8: Geografische Koordinaten

3.3.1.1 Breitengrade (Latitude)

Wie bereits erwähnt, verlaufen Breitengrade parallel zum Äquator, welcher zugleich den Nullmeridian für die Breitengrade festlegt. Ausgehend vom Nullmeridian wird der Winkel in Graden nach Norden und Süden jeweils bis 90° gemessen und als nördliche bzw. südliche Breite angegeben (teils auch angegeben als negative bzw. positive Werte).

3.3.1.2 Längengrade (Longitude)

Da Längengrade um die gesamte Erde vom Nord- zum Südpol verlaufend verteilt sind, erstrecken sie sich 360° rund um die Erde. Aufgeteilt wurden sie dabei von -180° im Westen bis 180° im Osten. Da es nicht wie beim Breitengrad einen natürlichen Nullmeridian gibt, musste er definiert werden und wurde dabei so festgelegt, dass er durch Greenwich (London) verläuft.

4 Resultate, Bewertung und Ausblick

4.1 Zielerreichung

Alle im Kapitel „1.3“ definierten Ziele wurden erreicht. Dies beinhaltet alle Muss- sowie alle Optionalen-Ziele. Auch das nicht direkt ersichtliche Ziel, eine gute Wart- sowie Erweiterbarkeit herzustellen, wurde aus unserer Sicht gewährleistet.
Zusätzlich wurden weitere Features, die während dem Projekt entweder als zusätzliche Anforderungen hinzu kamen oder aber uns selber gestört haben, umgesetzt:

- Einführen einer Einstellung zur Auswahl bzw. Eingabe verschiedener Tile-Server (Kapitel „Kartenmaterial“)
- News (vom RSS-Feed der NeoMap Webseite) anzeigen und abspeichern (Kapitel „News“)
- Überarbeitung des kompletten Design und Look and Feels nach Android 4.0 Standards (Hole Theme)
- Diverse kleine Tweaks: Ausschalten des Bildschirmes bei Bedarf deaktivieren, Löschen von heruntergeladenen Tiles, Ausblenden von beliebigen Ebenen, Einführen von Zoom-Buttons usw.

4.2 Schlussfolgerung und Ausblick

Wir denken, dass wir die NeoMap App zurück auf eine gute Basis bringen und nützliche Zusatz-Features einbauen konnten. Vor allem durch das grosse Refactoring, die Verwendung der Android Version 4.0 API Features sowie die Migration von OpenGL ES 1.1 auf 2.0, wurde die App auf einen modernen und gut erweiterbaren Stand gebracht. In diesen Belangen ist sie sicherlich auch vielen anderen Apps einen grossen Schritt voraus. Eine Mehrzahl anderer Karten-Apps läuft noch mit OpenGL ES 1.1 und trägt viele Altlasten mit sich. Damit dieser „Vorsprung“ beibehalten werden kann, muss jedoch eine kontinuierliche Weiterentwicklung stattfinden. Die App ist zwar grundlegend voll funktionsfähig, jedoch kann und sollte sie noch um einige wichtige Funktionalitäten ergänzt werden. Die folgenden Kapitel beschreiben die Weiterentwicklungsmöglichkeiten vorwiegend aus Sicht des Benutzers. Für einen Einblick in die technische Ebene, sollte der „Teil II15 Weiterentwicklung“ gelesen werden. Nebst den hier aufgeführten Punkten, gibt es noch viele weitere Funktionen die eingebaut werden könnten. Einige davon sind auch im offiziellen Redmine (http://dev.ifs.hsr.ch/redmine/projects/neomap/) erfasst.

4.2.1 Ganze Karten-Regionen herunterladen

Ein Feature welches wir selber während der Entwicklung vermisst haben. Oft wäre es wünschenswert, wenn man bereits im Voraus, bevor man beispielsweise auf eine Wanderung geht, einen bestimmten Kartenabschnitt über alle Zoom-Stufen herunterladen könnte. Google Maps bietet diese Funktionalität mittlerweile auch an.

4.2.2 Auto-Tracking bzw. Aufzeichnen von Routen

Für uns ist das eine wichtige Funktion die fehlt, vor allem da manuell bereits Routen (mittels Liniengeometrien) eingezeichnet werden können. Dieses Feature wird sicherlich schnell vermisst werden.

4.2.3 Import von GPX-Dateien

GPX-Dateien können nun zwar exportiert, jedoch noch nicht importiert werden. Dieses Feature bringt einen relativ kleinen Aufwand mit sich. Trägt aber jedoch sicher wesentlich dazu bei, die Attraktivität der App zu steigern.

4.2.4 Zusammenspiel mit der NeoMap Webseite

Der Vorteil, dass eine Webseite zur App besteht, sollte besser ausgenützt werden. Vor allem der Aspekt, dass soziale Netze momentan sehr populär sind, müsste besser genutzt werden. Bietet man eine einfachere Registrierungsmöglichkeit, beispielsweise mittels Facebook- oder Google-Konto, können sicherlich mehr Benutzer dazu ermutigt werden einen Account anzulegen. Dann können weitere Funktionen wie z.B. das Teilen von Notizen und erfassten Routen eingebaut werden. Zusätzlich sollte auf der Webseite definitiv erklärt werden, wie man Karten georeferenziert bzw. dieser ganze Ablauf sollte stark vereinfacht werden, denn nur so können Benutzer dazu animiert werden, eigene Karten zu erstellen.

4.2.5 Navigationsart anpassen

Die Navigation der App findet momentan über das Overflow-Menü der Action Bar statt. Dies ist sicherlich nicht optimal und von Android auch nicht so angedacht. Ideen für eine andere Navigationsart finden sich bei „Teil II15 Weiterentwicklung“.

4.2.6 Ausgabe von Meldungen überarbeiten

Meldungen werden momentan als sogenannte „Toasts“ ausgegeben. Diese Popup-Meldungen erscheinen immer am gleichen Ort auf dem Bildschirm. Schöner wäre in vielen Fällen jedoch, wenn die Meldungen mit einem Pfeil oder etwas ähnlichem auch dort erscheinen würden, wo sie hingehören. Zum Beispiel eine Meldung zu „Was ist hier?“ sollte idealerweise mit einem Pfeil an die Stelle zeigen, auf welche mit dem Finger gedrückt wurde.

4.3 Grundgesetze bei der Weiterentwicklung

Zusätzlich sollten bei der Weiterentwicklung der App folgende Aspekte auf jeden Fall immer eingehalten werden:

- Offline-Fähigkeit aller (zusätzlicher) Features immer gewährleisten
- Vorgegebene Design-Guidelines von Android einhalten (http://developer.android.com/design/index.html)
- App nicht mit unnötigen Features überladen
- Einfachheit bzw. Simplizität der Benutzeroberfläche beibehalten (User Interface „clean“ halten)

4.4 Persönliche Berichte

4.4.1 Simon Stähli

4.4.1.1 Projektverlauf

Da ein Grossteil dieser Arbeit auf Themen basierte, mit welchen wir nicht im Voraus vertraut waren und keine Erfahrungen hatten, war zu Beginn der Arbeit die Einarbeitung in die Thematik der Grafikprogrammierung sowie der Kartographie die wichtigste Tätigkeit. Im Speziellen die Grafikprogrammierung ist ein Thema, welches im Studium keinen Platz in einer Vorlesung findet, so dass bei null begonnen werden musste.
Die Theorie zu verstehen war das Eine, konnte jedoch mit guter Literatur und guten Online-Ressourcen bewältigt werden. Jedoch war es nochmals schwieriger, selbst mit diesem Wissen, das bestehende App zu verstehen. Dies wäre viel einfacher gewesen, wenn es eine Dokumentation dazu gegeben hätte. Glücklicherweise konnten wir viele unserer ersten Fragen durch ein Meeting mit einem früheren Entwickler klären. Da es sich bei dieser App jedoch um ein komplett neues Gebiet handelte, mussten auch während der gesamten Entwicklung frühere Umsetzungen überdacht und überarbeitet werden, da stets neues Wissen erarbeitet und neue Erfahrungen gesammelt wurden. Eine exakte Planung und Konzeption der Architektur des Apps im Voraus wäre an unserer Stelle wohl unmöglich gewesen, da uns zu viele Themen fremd waren. Als kritischen Zeitpunkt würde ich vor allem die ersten Wochen des Projektes und die anfängliche Migration von OpenGL 1.1 auf 2.0 beschreiben. Nach einigen Schwierigkeiten konnten wir dies jedoch erfolgreich abschliessen. Ein wesentliches Problem war dabei, dass ein Debugging fast nicht möglich ist und beispielsweise leichte Verschiebungen bei der Projektion der Karte darin enden, dass auf dem Bildschirm gar nichts sichtbar ist. Hier ist ein langsames Herantasten an die optimale Lösung meist nicht möglich.

4.4.1.2 Lessons Learned

Durch die Schwerpunkte dieser Arbeit, die Bereiche Android und OpenGL, konnte ich viel über die Grafikprogrammierung und Android im Allgemeinen lernen. Grundlegendes über Android war mir bereits im Voraus bekannt, jedoch stellten sich im Zusammenhang mit dem Karten-App und der intensiven Verwendung von Bitmap-Grafiken neue Herausforderungen im Bereich des Memory-Managements und der Performance. Hierbei stiessen wir teilweise an Grenzen und Probleme, für welche weder in Literatur, noch online Lösungen gefunden werden konnte.
Auch die Grafikprogrammierung hat uns immer wieder neue Hürden gestellt, doch war es alles in allem ein sehr interessantes Thema. Nicht zuletzt auch, da die verwendete Literatur teilweise auch aus der Game-Entwicklung stammte und die grundlegenden Konzepte, in Bezug auf die Grafik-Darstellung, ähnlich sind wie bei einem Spiel.
Durch die erstmalige Verwendung von Git in einem Projekt konnte das theoretische Wissen in der Praxis angewendet werden. Git hat mich dabei vollumfänglich überzeugt, gerade in Bezug auf Branches erstellen und späteres Mergen von Branches bzw. von einzelnen Commits. Hier überzeugten sowohl die Performance als auch die guten Merge-Resultate.
Insgesamt kann gesagt werden, dass die Lernkurve bei diesem Projekt zu Beginn sehr steil war, jedoch nie allzu gross abflachte und somit bis am Schluss immer wieder neue Erkenntnisse gewonnen werden konnten. Vor allem in Bezug darauf wie etwas doch besser gelöst werden könnte als anfangs gedacht.

4.4.1.3 Rückblick

Am Abschluss dieser Bachelorarbeit stehend und rückblickend gesehen auf die gesamte Arbeit, kann ich ein durchwegs positives Resümee ziehen. Die Arbeit insgesamt habe ich als sehr fordernd empfunden und konnte in verschiedenen Bereichen nochmals sehr viel Neues und Spannendes lernen. Die Arbeit an einem Android-App hat mir meist Spass gemacht, nicht zuletzt da es sich um ein interaktives App handelt, bei welchem man meist ein direkt sichtbares Resultat bei der Entwicklung von neuen Features hat.

Das Teamwork hat stets gut funktioniert und allfällige Meinungsverschiedenheiten konnten gut ausdiskutiert und dadurch immer gelöst werden. Dies war bereits die zweite grosse Arbeit, die ich mit Patrick zusammen durchgeführt habe und ich denke, ich werde diese Arbeiten, auch gerade wegen dem Teamwork, als eines der Highlights im Studium in Erinnerung behalten. Da einem doch zwischendurch die Motivation fehlen kann, ist es umso wichtiger einen guten Arbeitspartner zu haben, um sich gegenseitig zu motivieren oder bei gemeinsamen Arbeiten eine lockere Stimmung haben zu können. Auch die Zusammenarbeit mit Herrn Keller hat gut geklappt. Er hat den Fortschritt der Arbeit stets mit Interesse erwartet.

4.4.2 Patrick Zenhäusern

4.4.2.1 Projektverlauf

Am Anfang, bei der Ausschreibung der Aufgabenstellung, hatte ich gemischte Gefühle. Weder Simon noch ich hatten grosse Erfahrung mit Android, geschweige denn mit OpenGL (der Aufgabentitel lautete am Anfang noch „NeoMap App mit 3D“ oder so ähnlich, was natürlich zusätzlichen „Respekt“ erbrachte). Dennoch war für mich klar: Dies ist eine der interessantesten Aufgaben und eine Bachelorarbeit sollte ja auch eine gewisse Challenge bieten.
Nach dem Kick-Off Meeting war mein Gefühl jedoch noch nicht viel besser. Herr Keller berichtete uns, dass M. Wolski (Masterstudent) bereits mit der Migration von OpenGL ES 1.1 zu 2.0 gescheitert sei und das Ganze vermutlich recht schwierig sei. Umso grösser war jedoch meine Motivation mich ausführlich in die Materie einzulesen und einzuarbeiten. Tatsächlich stellte sich OpenGL ES 2.0 als recht komplex heraus. Für mich war das grosse Ziel ganz klar: Die Migration so schnell wie möglich anzufangen und abschliessen zu können. Denn dieses Kriterium war für mich der Stein auf der Waage, der aufzeigen würde ob die Bachelorarbeit ein Erfolg werden wird oder nicht. Das dieses Kriterium bereits am Anfang auftrat, konnte sich natürlich als Fluch, wie auch als Segen herausstellen. Glücklicherweise war das Zweite der Fall, denn nach ca. ein-zwei Wochen Einlesen schafften wir es innerhalb einer Woche die Migration grösstenteils erfolgreich abzuschliessen (natürlich war während dieser Zeit der Aufwand entsprechend hoch). Ab da stand für mich fest: Jetzt geht es aufwärts! Ein weiterer Stolperstein war die Qualität des Codes. Diese war teilweise erschreckend schlecht bzw. unverständlich. Dadurch dass wir uns genügend Zeit nahmen, konnte der Code jedoch auf eine gute Basis zurück gebracht werden. Viele Stellen musste ich jedoch mehrmals überarbeiten, dies ist vor allem auf mein beschränktes Wissen von Android zurückzuführen.
Während des Projektverlaufes hatte ich immer wieder mehrere „Aha!“-Momente bei denen mir klar wurde, dass dies eigentlich „so“ gemacht werden müsste. Dies zog dann natürlich immer eine Änderung von bestehendem Code mit sich. Nach der Migration und dem Refactoring lief dann meistens alles wie am Schnürchen. Die geplanten Features konnten schnell und meist ohne Probleme umgesetzt werden. Es blieb sogar genug Zeit um noch zusätzliche Anforderungen wie z.B. eine Auswahl von verschiedenen Tile-Servern oder ein News-Feature umzusetzen. In der Mitte des Projektes war für mich dann bereits klar: Das optionale Ziel (geografische Namenssuche) MUSS auch umgesetzt werden. Einerseits weil wir gut in der Zeit lagen und andererseits weil ich schnell merkte, dass eine geografische Namenssuche ein MUSS-Kriterium für eine Karten-App ist. Immer wieder vermisste ich die Möglichkeit nach Orten zu suchen. Der aus diesem Feature entstandene praktische Nutzen ist in meinen Augen enorm! Nachdem auch dieses Feature relativ gut implementiert wurde, ging es dann vor allem darum noch Tests zu schreiben und an mein Lieblings-Thema – die Dokumentation! Glücklicherweise hatten wir bereits während dem gesamten Projektverlauf die Dokumentation relativ gut mitgeführt und mussten deswegen nicht alles nachführen. Trotzdem entstand ein grosser Aufwand. An dieser Stelle war es für mich teilweise schwer genügend Motivation zu finden. Schlussendlich schaffte ich es aber mich selber zu disziplinieren und die Dokumentation, in einer wie ich finde, guten Form abschliessen zu können.

4.4.2.2 Lessons Learned

Den wohl grössten Punkt den ich hier erwähnen möchte ist: Grafikprogrammierung (OpenGL) ist (ohne Framework) sehr schwierig! Ich hatte diese Vermutung bereits vor der Arbeit und bekam diese während der Arbeit bestätigt. Es geht auf dieser Ebene der Programmierung wirklich sehr viel um Matrizenumrechnungen, Projektionen und andere mathematisch anspruchsvolle Themen. Ich habe durch diese Arbeit einen guten Einblick in diese Welt bekommen und konnte mir so auch klar machen, dass dies vermutlich nicht das Gebiet ist, in welchem ich später arbeiten möchte. Ich bevorzuge etwas „höhere Programmierung“, bei der es vor allem um gute Konzepte usw. geht.
Weiter wurde mir wieder einmal klar, wie schwierig es ist bestehende Applikation (vor allem ohne Dokumentation!) zu verstehen und weiter zu entwickeln. Ich denke es ist um einiges schwieriger ein bestehendes Produkt umzubauen, anstatt ein neues zu entwickeln. Gleichzeitig wurde mir natürlich auch wieder die Relevanz einer (nur schon minimalen) Dokumentation bewusst. Mit Dokumentation wäre das Einarbeiten in den Code sicher um ein Vielfaches einfacher gewesen.
Im ganzen Bereich Android-Entwicklung / Designpattern konnte ich auch einige neue Konzepte mitnehmen und erlernen. Gleichzeitig merkte ich, dass mir Android Entwicklung sehr viel Spass macht: Einerseits weil man sehr schnell sichtbare Resultate erzielen kann und andererseits weil diese Plattform ständig und relativ schnell weiter entwickelt wird und so immer neue interessante Aspekte hinzukommen.
Auch im Bereich von Git konnte ich mir einiges zusätzliches Wissen aneignen. So lernte ich, wie man möglichst effizient und gut mit „Branches“ arbeiten kann oder wie man auf einfachste Art und Weise Fehler in Files finden kann („Git blame“).

4.4.2.3 Rückblick

Rückblickend bin ich mit den erzielten Resultaten im Projekt sehr zufrieden. Natürlich gibt es einiges was ich jetzt anders bzw. besser machen würde. So bin ich beispielsweise nicht ganz zufrieden mit der Umsetzung des Fragments-Teils. Dieser ist etwas schwerfällig geworden und kann sicherlich noch vereinfacht und optimiert werden. Natürlich gibt es im Nachhinein immer vieles, das man anders machen würde. Leider fehlte dazu teilweise die Zeit, denn während es Projektes kamen auch immer wieder neue, wenn meist auch kleine, Anforderungen von Herrn Keller hinzu. Die Zusammenarbeit mit Herrn Keller klappte aber trotzdem meistens sehr gut. Er war entgegenkommend und ging auch auf unsere Vorstellung ein. So konnten wir Ihn beispielsweise davon überzeugen eine (grosse) Änderung bei den Notizen am Ende wegzulassen, weil wir schon zu weit fort geschritten waren und keine solchen grossen Änderungen in der Architektur mehr vornehmen wollten. Ein kleines Problem lag jedoch in den unterschiedlichen Vorstellungen, die Herr Keller und wir von der App hatten. Herr Keller sah das Zielpublikum der App auch im Bereich von Geo-Erfassung. Wir hingegen sahen den Nutzen vor allem für den „normalen Gebrauch“ (siehe User Szenarien). Dadurch entstanden manchmal gewisse Unstimmigkeiten für die Art der Umsetzung von Features. Dies hätte vermutlich vermieden werden können, wenn wir die User Szenarien gemeinsam besser angeschaut und erarbeitet hätten.
Etwas mühsam war auch die Arbeit mit der NeoMap-Webseite: Jedes Mal wenn eine Funktion der Webseite gebraucht wurde, funktionierte irgendetwas nicht. So konnte man sich zuerst nicht registrieren, dann funktionierte der Upload von NeoMaps nicht und das Login über die NeoMap App bzw. das Abrufen von privaten NeoMaps gab immer falsche Resultate zurück. Die Punkte wurden zwar irgendwann immer behoben, jedoch mussten wir dafür trotzdem oft 1-2 Wochen warten.
Sehr gut war hingegen die Zusammenarbeit mit Simon. Wir hatten schon bei mehreren Projekten zusammen gearbeitet und kannten uns deshalb schon relativ gut. Wir hatten selten grosse Unstimmigkeiten und konnten auch eine sehr gute Arbeitsteilung realisieren. Vor allem im Bereich von OpenGL bin ich sehr froh, dass Simon dabei war. Er hatte die Thematik um einiges schneller als ich intus und hatte in diesem Teil sicher auch die Führung übernommen. So entstand auch eine, implizite, Arbeitsaufteilung: Währendem Simon sich primär auf OpenGL konzentrierte, kümmerte ich mich um den ganzen Android-Teil.

Diese Aufteilung zogen wir mehr oder weniger das ganze Projekt so durch, wobei wir durch Reviews einander immer wieder erklärten, warum was wie umgesetzt wurde.
Insgesamt hat mir das Projekt doch zum grössten Teil viel Spass bereitet. Vor allem habe ich mich sehr gefreut, dass ich, zum ersten Mal in einer von mir entwickelten Applikation, an zwei Stellen, „EasterEggs“ einbauen konnte. Wo dies genau ist, soll an dieser Stelle jedoch nicht verraten werden☺.

Teil II
SW-Projektdokumentation

1 Analyse des Apps bei Projektbeginn

Da es sich bei der NeoMap App nicht um ein neues Projekt handelt, war bereits viel Code vor diesem Projekt vorhanden. Somit bestand einer der ersten Schritte in der Analyse des Quellcodes sowie der existierender Applikation aus Benutzersicht.
Folgend sind die wichtigsten Resultate der Analyse aufgelistet. Der grösste Teil davon wurde in der Construction 1 und 2 durch Refactoring und Fehler-Behebung verbessert. Detailliertere Angaben zum Refactoring befinden sich im Kapitel „Refactoring".
In den ersten zwei Wochen des Projektes stand leider noch nicht der aktuelle Code zur Verfügung. M. Wolski hatte in Form einer Projektarbeit während des Masterstudiums im Herbstsemester 2013 an der App gearbeitet, seinen Code jedoch nie ins GIT Repository eingecheckt. Da wir jedoch nicht warten wollten bis er dies gemacht hatte, haben wir die Analyse in zwei Phasen gegliedert:

Analysen-Phasen	Zeitraum	Beschreibung
P1: Ohne Änderungen M. Wolski	Woche 1 – Mitte Woche 2	Offizieller Code, welcher auch auf der NeoMap Seite heruntergeladen werden kann
P2: Mit Änderungen M. Wolski	Woche 2 – Woche 4	Bestehender Code wurde durch eine nicht funktionierende POI-Suche und Anzeige ergänzt

Tabelle 11: Analysen-Phasen

1.1 Benutzeroberfläche

Folgend sind Fehler und Optimierungen aufgelistet, welche den Benutzer der App direkt betreffen.

1.1.1 Fehler aus Benutzersicht

Beschreibung	Priorität
Eigener Standort (roter Kreis) auf der Karte wird „mit verschoben" wenn die Karte bewegt wird, anstatt das dieser fixiert bleibt	**1**
Zu nahes hineinzoomen in die Karte führt zu einem weissen Bild	**1**
Zu weites raus zoomen führ dazu, dass die Karte sehr klein ist und rund herum ein weisses Rasternetz erscheint.	**1**
Unerklärliche Abstürze, welche sporadisch während der Benutzung des Apps auftreten	**1**
Einschalten von POI Anzeigen (Kultur, Tankstellen, etc.) führt zu App Absturz	**2**
Suche nach eine POI bzw. schon das Eingeben eines Zeichens führt zu App Absturz	**2**
Ursprünglich wurden einmal Koordinaten des aktuellen Standpunktes angezeigt, dies ist jedoch nicht mehr vorhanden in der letzten Version	**2**

Tabelle 12: Fehler aus Benutzersicht

1.1.2 Optimierungen aus Benutzersicht

Beschreibung	Priorität
Der Aufbau der Karte ist relativ langsam und beginnt jeweils oben links. Im Interesse des Benutzers steht jedoch meist als erstes, was sich im Zentrum der Karte befindet. Siehe dazu Kapitel „2.2.10 Laden von Tiles optimieren“.	**1**
Fehlende Buttons (Suche starten, etc.) in den Einstellungen, die Suche wird an diesen Orten entweder implizit oder mittels betätigen der „Enter“ Taste auf dem Smartphone gestartet	**1**
Das Navigations-Menü (Eigener Standort festlegen, Kompass einschalten etc.) ist auf Geräten mit kleinem Bildschirm zu gross	**1**

Tabelle 13: Optimierungen aus Benutzersicht

1.2 Quellcode

Einen Überblick über den zu Projektbeginn vorhandenen Code und dessen Zustand wurde unter anderem auch mittels den Stan4J, Eclipse-Plugins Findbug, PMD und CodePro AnalytiX verschaffen. Nachfolgend einige Code-Metriken.

1.2.1 Überblick

Eine Analyse der bestehenden Package-Struktur und den dabei auftretenden Abhängigkeiten hat ergeben, dass sehr viele bidirektionale Abhängigkeiten vorhanden sind. Diese Abhängigkeiten sind unter anderem auch zwischen dem Presentation-Package und den allen restlichen Packages vorhanden. Eine saubere Trennung der Präsentations-Schicht wurde hierbei nicht verfolgt.

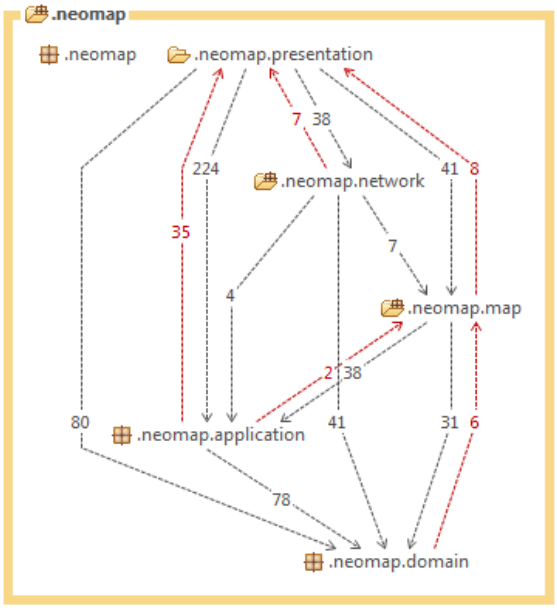


Abbildung 9: Dependencies bei Projektübernahme

1.2.2 Code-Metriken

Folgende Code-Metriken sollen den Stand anfangs des Projektes dokumentieren.
Stand 9.03.2013

1.2.2.1 Quellcode Überblick

Kategorie	Werte
Lines of Code	9065
Number of Packages	31
Number of Types	191
Efferent Couplings	94

Tabelle 14: Code-Metrik Übersicht der gesamten Applikation bei Projektbeginn

1.2.3 Generell

Fehler:

- **P2:** Teilweise sind Klassen doppelt vorhanden in verschiedenen Paketen und werden dann falsch referenziert in Imports

Verbesserungswürdig:

- **P1:** Zu wenig Packages (Keine klare Strukturierung nach Zusammengehörigkeit)
- **P1:** Klassen haben teilweise keine guten Namen (nicht direkt klar was die Klasse macht)
- **P1:** Viele auskommentierte Zeilen, teilweise auch mit „old“ etc. deklariert
- **P1:** Klassen haben oft viele innere Klassen, welche besser in ein eigenes File verschoben werden sollten
- **P1:** Oft „Over-Engineering“ vorhanden (Methode xy mit Kommentar „könnten wir ja mal brauchen“ oder „do3D“ (was aber nicht funktioniert))
- **P2:** Einige Klassen eingeführt welche nirgends aufgerufen werden
    - Beispielsweise Klasse Circle oder auch CircleRenderer. Code kann so wie er darin enthalten ist nicht mit der Applikation interagieren.
- **P2:** Teilweise Klassen eingeführt welche sehr viele Variablen enthalten, welche besser eine eigene Klasse bzw. eine Enum-Klasse haben sollten

Bei der Analyse haben sich vor allem die beiden Packages ch.hsr.neomap.map und ch.hsr.neomap.presentation.activity als überladen und unstrukturiert hervorgehoben.

1.2.3.1 Package ch.hsr.neomap.map

Kategorie	Werte
Lines of Code	2170
Number of Types	40
Efferent Couplings	22

Tabelle 15: Code-Metrik Package ch.hsr.neomap.map bei Projektbeginn

Um den Quellcode übersichtlich zu halten, sollten hier die Klassen nach einer neu definierten Package-Struktur aufgeteilt werden. Auch scheint das Package mit einem Efferent Coupling Werte von 22 von vielen externen Klassen direkt abhängig zu sein.

Konkrete Klassen die unbedingt ein Refactoring benötigen:

- OSMLayerRenderer
- NeoMapLayerRenderer

Beide Klassen besitzen teils duplizierten Code, welcher in die gemeinsame Basis-Klasse AbstractLayerRenderer ausgelagert werden sollte.
Des Weiteren besitzen beide Klassen mehrere interne Klassen welche auch ausgelagert werden sollten um die Übersicht zu erhöhen.

1.2.3.2 Package ch.hsr.neomap.presentation.activity

Kategorie	Werte
Lines of Code	2450
Number of Types	60
Efferent Couplings	12

Tabelle 16: Code-Metrik Package ch.hsr.neomap.presentation.activty bei Projektbeginn

Konkrete Klassen die unbedingt ein Refactoring benötigen:
MapActivity

- Beinhaltet 578 Lines of Code. Vieles davon, im Speziellen die momentan darin integrierte POI-Suche und Anzeige, sollte und kann problemlos in separate Klassen ausgelagert werden. Diese Arbeiten sollten ausgeführt werden um dem Prinzip der High-Cohesion wieder gerecht zu werden.

MapSearchActivity

- Besteht aus 374 Lines of Code. Besitzt eindeutig zu viele private Felder, welche in eigene Objekte (beispielsweise ein User-Objekt, welches User-Details kapselt), ausgelagert werden sollten.

1.3 Offizielles NeoMap Redmine

Infolge der Analyse wurde auch das offizielle Redmine des NeoMap Apps angeschaut, denn dieses stellt das eigentliche Projektmanagement Werkzeug von NeoMap dar und ist daher von relativ hoher Wichtigkeit.

- **P1:** Viele Feature Requests im Wiki anstatt als Tickets eingetragen
- **P1:** Viele Tickets nicht aussagekräftig, es ist nicht klar, was damit gemeint ist
- **P1:** Sprache Deutsch / Englisch gemischt, anstatt konsequent Englisch
- **P1:** Sinnlose Versionen eingetragen („Sitzung“, „Inception“) etc. → solche Versionen haben in einem solchen Redmine nichts verloren
- **P1:** Als News sind oft Sitzungsprotokolle und ähnliches erfasst, anstatt richtige News

1.4 Sonstiges

- **P1/P2:** Dokumentation sozusagen inexistent
  - **P1:** Die Studienarbeit von D. Lüchinger enthält nichts, was wirklich hilfreich für das verstehen der Architektur ist. Es existiert zwar ein Klassendiagramm dies ist aber wohl autogeneriert worden und enthält keinerlei Erklärungen
  - **P1:** P. Recher hat gar keine Doku erstellt, dies weil die Arbeit für ihn nur ein Miniprojekt für das Programmieren 1 war und dort auch keine Dokumentation gefordert wurde
  - **P2:** M. Wolski hat zwar eine Dokumentation geliefert. Jedoch beinhaltet diese kein Wissen über die Architektur, sie hilft für die Weiterentwicklung also sozusagen nichts.

1.5 Fazit

Die Analyse der App hat ergeben, dass diese in einem sehr schlechten Zustand ist. Es funktioniert zwar fast alles (abgesehen vom POI-Teil), jedoch sind die Erweiterbarkeit sowie die Wartbarkeit der Applikation in dieser Form nicht gegeben.
Der Fokus des Refactorings wird deshalb vor allem darin liegen, diese beiden Punkte stark zu verbessern. Der Schwerpunkt liegt dabei insbesondere bei den Komponenten, welche für die Darstellung der Karte verantwortlich sind, da diese den Kern dieses Apps bilden.
In den obigen Kapiteln sind natürlich nur die wichtigsten Punkte aufgelistete, grundsätzlich bedarf es aber sozusagen bei allen Klassen einem Refactoring!
Zusätzlich wird für alle relevanten Teile eine Architektur-Dokumentation erstellt.

2 Refactoring

Die Analyse des bestehenden Quellcodes hat, wie bereits im vorhergehenden Kapitel erwähnt, ergeben, dass zwingend ein grosses Refactoring nötig ist, um eine stabile Grundlage für weitere Entwicklungen zu legen. Da diese Aufgabe ein doch beachtlicher Teil der Projektzeit in Beanspruchung nahm, wurde ihr ein eigenes Kapitel gewidmet.

2.1 Offizielles NeoMap Redmine

Bereits vor Projektbeginn wurde Redmine als Projektmanagement-Tool für die Verwaltung des NeoMap Projektes eingesetzt. Dieses sollte nun, um an Übersichtlichkeit zu gewinnen, aufgeräumt werden. Zusätzlich sollten einige Regeln zur Benutzung festgelegt werden.
Unter folgendem Link ist das Redmine-Projektmanagement-Tool erreichbar:
http://dev.ifs.hsr.ch/redmine/projects/neomap

Dabei wurden folgende Hauptänderungen vorgenommen:

- Als Sprache wurde konsequent Englisch verwendet, dabei wurden alle existierenden deutschen Texte nach Englisch übersetzt
- Alle vorhandenen Versionen wurden geschlossen und es wurden zwei neue Versionen eingeführt:

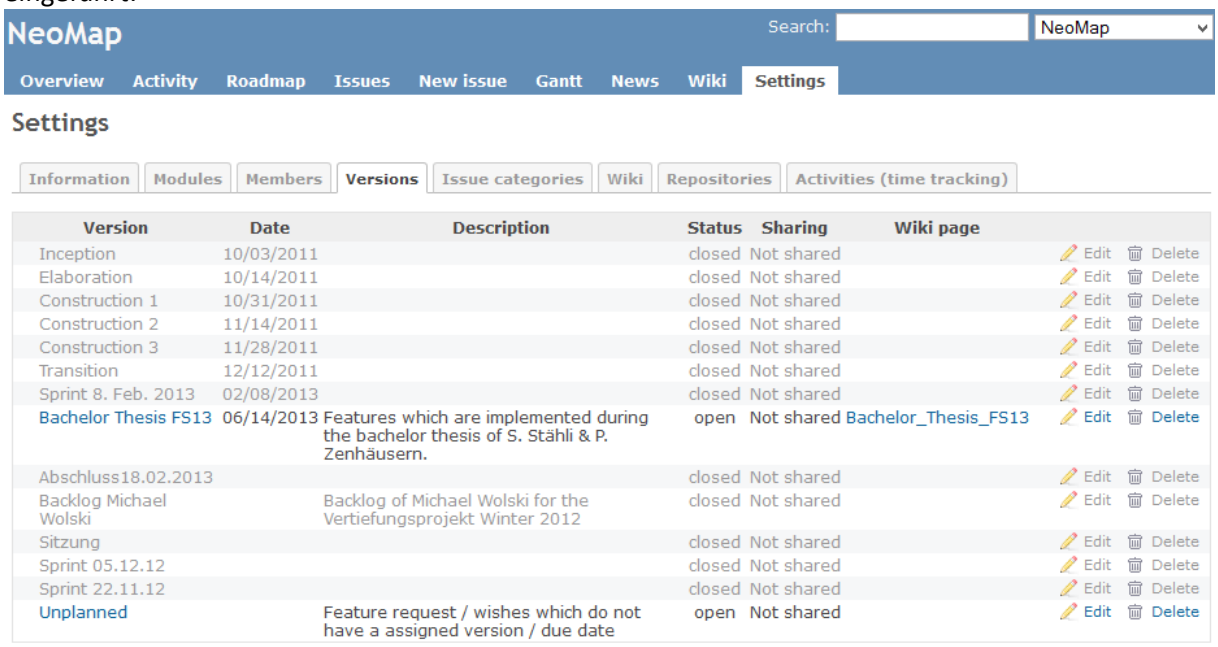


Abbildung 10: NeoMap Redmine Versionen

- Die Version „Bachelor Thesis FS13“ deckt dabei alle Features ab, welche im Rahmen von der Bachelorarbeit bearbeitet werden. Zu dieser Version existiert zusätzlich ein Wiki Artikel, welcher detaillierter erklärt was in dieser Version gemacht wurde und schlussendlich diese Bachelor Dokumentation als Anhang enthalten soll.
  Zusätzlich wurde die Version „Unplanned“ eingeführt. Wie die Beschreibung bereits angibt, werden alle Features / Wünsche welche keiner konkreten Version zugeordnet werden können dieser Version angefügt. Arbeitet später ein weiteres Team an einer neuen Version, können Tickets von der „Unplanned“ Version einfach in die neue Version verschoben werden. Durch dieses Vorgehen ist sichergestellt, dass keine Tickets verloren gehen. Ausserdem sollte so immer klar sein, was in welcher Version geändert wurde. Natürlich muss das Konzept weiterhin eingehalten werden und nicht wieder sinnlose Versionen wie „Sitzung“ etc. erstellt werden.
- Alle vorhandenen Tickets wurden entweder einer der beiden neuen Versionen zugewiesen oder gelöscht bzw. als abgeschlossen markiert

- Alle Feature-Requests aus dem Wiki wurden von Deutsch auf Englisch übersetzt, Tickets erstellt und der Version „Unplanned“ zugeordnet
- Git Repository des NeoMap App wurde mit dem Redmine verknüpft. Auf diese Weise kann zum einen das gesamte Repository bzw. der Git Log direkt in Redmine angeschaut werden und zum anderen können Commits mit Tickets verknüpft werden (z.B. durch den Text „refs #122“, wobei 122 die Ticketnummer darstellt) und werden so direkt in den zugehörigen Tickets angezeigt. So kann man aus den Tickets direkt erkennen, welche Code-Änderungen diese mit sich brachten.

2.2 NeoMap App Quellcode

2.2.1 Package Struktur

Die bestehende Package-Struktur hatte kein durchgezogenes Konzept trotz einigen wenigen Packages, wurden Klassen teils sogar auch ohne Berücksichtigung dieser "Struktur" abgelegt. Beispielsweise fanden sich View oder Adapter Klassen im „application“ Package anstelle des „presentation“ Packages. Noch während der Elaboration-Phase wurde eine neue Package Struktur ausgearbeitet und eingeführt. Diese ist Im Kapitel „Tabelle 18: Package-Struktur-Beschreibung Package-Struktur“ dokumentiert.

2.2.2 Refactoring Generell

Beim Refactoring wird vor allem das Ziel der Separation of Concerns verfolgt. Durch das eindeutige trennen von Funktionalitäten, sollte es möglich sein, den aktuellen Quellcode wieder übersichtlicher zu gestalten und die hohe Kopplung zwischen verschiedenen Klassen zu verringern.
Teils wurden bereits Interfaces sowie auch abstrakte Klassen eingeführt um Abhängigkeiten zwischen Klassen zu brechen, es wurde aber selten auch gegen diese programmiert. Meist wurde stattdessen doch die konkrete Klasse verwendet, was zu erhöhter Kopplung zwischen Klassen führte. Dies soll wenn möglich geändert werden.
Konkrete Klassen hatten teilweise generische Namen, wie zum Beispiel „Grid“. Dieses Grid wurde jedoch nur für den OpenStreetMap-Karten Teil verwendet. Daher ist es in diesem Fall besser als „OSMGrid“ zu bezeichnen. Mit solchen aussagekräftigen Namen für Klassen kann die Lesbarkeit und Verständlichkeit des Quellcodes erhöht werden.

2.2.3 Preferences / Settings

Die benutzerdefinierten Einstellungen wie z.B. der Benutzername oder das Passwort waren an verschiedenen Orten gespeichert, d.h. sie waren mehrfach vorhanden. Einerseits waren sie in verschiedenen SharedPreference Dateien gespeichert und andererseits auch noch in der Datenbank. Solche Einstellungen gehören jedoch klar in ein einziges SharedPreference-File. Vorzugsweise benutzt man dazu „PreferenceManager.getDefaultSharedPreferences(Context)“. Damit muss man sich nicht mal mehr selber um den Namen der Datei kümmern. Diese wird automatisch im privaten Datenordner der NeoMap App angelegt.
Für die Settings wird neu eine Service-Klasse benutzt, die im Wesentlichen aus den Methoden „saveLastSettings()“ und „loadLastSettings()“ sowie weiteren Hilfsmethoden besteht. Durch diese Klassen können, mit Hilfe des neuen ApplicationControllers, Einstellungen von beinahe überall her, ohne grossen Aufwand, abgefragt und eingefügt werden.

2.2.4 Dead Code

In der gesamten App war viel Dead Code vorhanden. Teilweise waren dies ganze Klassen welche nicht mehr benötigt werden, jedoch nie gelöscht wurden. Dies schien vor allem mit der letzten Umstellung des User-Interfaces zusammen zu hängen.
Des Weiteren wurde Code häufig auskommentiert, jedoch in der Klasse belassen. Bei der Verwendung eines Versionsverwaltungs-Systems hat auskommentierter Code keinen Verwendungszweck, sondern erschwert die Übersichtlichkeit.
In einer ersten Aufräumungs-Aktion konnten bereits etwa 25% aller Klassen gelöscht werden, die nirgends mehr verwendet wurden.

2.2.5 Activities

Obwohl es unter Android relativ schwierig ist, im Presentation-Layer den Code, welcher rein zur Darstellung von Informationen benötigt wird, von der eigentlichen Logik zu trennen, sollte dies doch zu einem bestimmten Grade getan werden um Activity-Klassen übersichtlicher zu gestalten.
Um dies zu bewerkstelligen, sollten Service-Klassen eingeführt werden, welche vorbereitete Daten an die Activity-Klasse liefern, welche dann nur noch UI-Controls entsprechend mit Information füllen sowie User-Inputs vorbereitet und entgegen nehmen.
Listener-Klassen welche auf User-Inputs, wie Touch o.ä., warten sollte jedoch meist auch in eigene Klassen ausgelagert werden.

2.2.6 Kartendarstellung

Der Code für die Anzeige von Karten ist überkompliziert implementiert und ist überhaupt nicht lesbar. Im Weiteren ist er so ausgelegt, dass es so gut wie unmöglich ist, andere Elemente auf der Karte zu zeichnen (wie z.B. Notizen). Aus diesem Grund wurde dieser Teil komplett überarbeitet. Der neuen Kartendarstellung ist ein ganzes Kapitel gewidmet: „5 Karten“.

2.2.7 Layout in XML auslagern

Alle früheren Entwickler der NeoMap App gingen davon aus, dass es besser sei das Design der Applikation bzw. des GUIs direkt in Java zu machen, anstatt im XML, wie es die Design-Richtlinien von Android vorschlagen und empfehlen. Als Grund wurde „Flexibilität“ angegeben. Dies ist natürlich völlig falsch. Denn einerseits ist es immer besser das Design von der Logik zu trennen (bestes Beispiel: HTML und CSS) und zum anderen hat man keinerlei Einbussen an Flexibilität, sondern gewinnt sogar dazu bei der Einsetzung von XML-Layouts. Sind Styles in XML ausgelagert, können für eine Designänderung nur diese Dateien bearbeitet werden, dabei ist ein neues Kompilieren des Java-Codes theoretisch nicht nötig. Zudem können alle Komponenten welche per XML definiert wurden immer noch über Java angesprochen und bearbeitet werden (z.B. um ein Element auszublenden).
Aus diesem Grund wurden fast alle Designs nochmals neu in XML geschrieben. Der frühere Java-Code wurde soweit es ging entfernt. Durch das Schreiben der XMLs konnten einige Elemente, die vorher im Java Code doppelt vorhanden waren, wieder verwendet werden. An einigen wenigen Orten bzw. für kleine Sachen, wie z.B. Alert Dialoge, wurde auf eine Auslagerung in XML verzichtet. Der Grund dafür ist, dass zwar UI-Elemente direkt in Java angelegt werden, jedoch kein spezielles Design haben, sondern von ihrer Parent-Views (welche in XML definiert sind) erben.

2.2.8 Entfernen von POI-Suche und Anzeige

Da der Versuch eine POI-Suche und Anzeige der verschiedenen Punkte auf der Map in der Projektarbeit von M. Wolski im HS 2012 gescheitert war, diese unvollständigen Klassen jedoch noch in der App vorhanden waren, galt ein spezielles Augenmerkt bei der ersten Aufräumungs-Aktion am Ende der Construction Phase genau diesen POI-Elementen. Nach gemeinsamem Beschluss mit Herrn Keller wurden alle Klassen welche mit der Verbindung zu POI-Suche / -Anzeige wieder aus der App entfernt.

2.2.9 Generalisierung der NeoMapManagementKlasse

Die NeoMapManagement-Klasse war so ausgelegt, dass diese ausschliesslich für NeoMaps verwendet werden konnte. Da Notizen jedoch nach dem gleichen Prinzip wie NeoMaps funktionieren, müssen diese auch gleich verwaltet werden können.
Aus diesem Grund wurde versucht die Klasse generisch zu gestalten. Dazu wurde zum einen eine Abstrakte Klasse-ManagementFragment (zusammen mit einem ManagementAdapter) eingeführt und zum anderen eine abstrakte Klasse ManagableItem. Details zu den neuen Klassen finden sich im Klassenbeschrieb in dieser Dokumentation.

2.2.10 Laden von Tiles optimieren

Bewegt man die Karte, so wird direkt immer alles geladen d.h. „scrollt“ man über einen grossen Kartenausschnitt oder zoomt man tief in die Karte hinein/hinaus, so werden der komplette Weg bzw. alle Zoomstufen geladen. Dies ist im Prinzip nicht weiter schlimm, jedoch ist das System auf „First in, first out“ ausgerichtet, was so viel heisst wie es wird alles in der Reihenfolge geladen in der es gesehen wird. Für die Verständlichkeit ein Beispiel: Wenn man sich auf der Karte bei Jona befindet, möchte jetzt aber Rapperswil sehen, so bewegt man die Karte Richtung Rapperswil. Es wird der komplette Weg geladen, befindet man sich dann bei Rapperswil, muss man gegebenenfalls warten bis der komplette Weg von Jona bis Rapperswil geladen ist, bis man überhaupt etwas sieht.
Es sollte also eine Art „Stack“ („Last in, First Out“) als Ladestrategie genutzt werden.
Dies wurde während dem Refactoring auch so umgesetzt. Falls sich zu viele Tiles in der Queue befinden, werden die ältesten gelöscht und die neuen heruntergeladen. Dadurch ist sichergestellt, dass die Tiles an der Stelle, an welcher sich der Benutzer gerade befindet, als erstes heruntergeladen werden.

3 Anforderungsspezifikation

3.1 Überblick

Um sicher zu stellen, dass neue Funktionen auch im Sinne von zukünftigen Nutzern entwickelt werden sowie deren Anforderungen abdecken, wurden User Szenarien erstellt. Es wurde auf Use Cases verzichtet, da diese für die NeoMap App unnötig formal und detailliert sind. Mittels User Szenarien sollten die Anforderungen klar ausgedrückt werden, ohne dass bereits im Voraus eine zu starke Einschränkung in Bezug auf die Entwicklung entsteht.
Da ein grosser Teil der Arbeit aus der Migration von OpenGL ES 1.x zu Open GL ES 2.0 sowie Refactoring des bestehenden Quellcodes besteht, sind nicht ganz so viele Szenarien vorhanden.

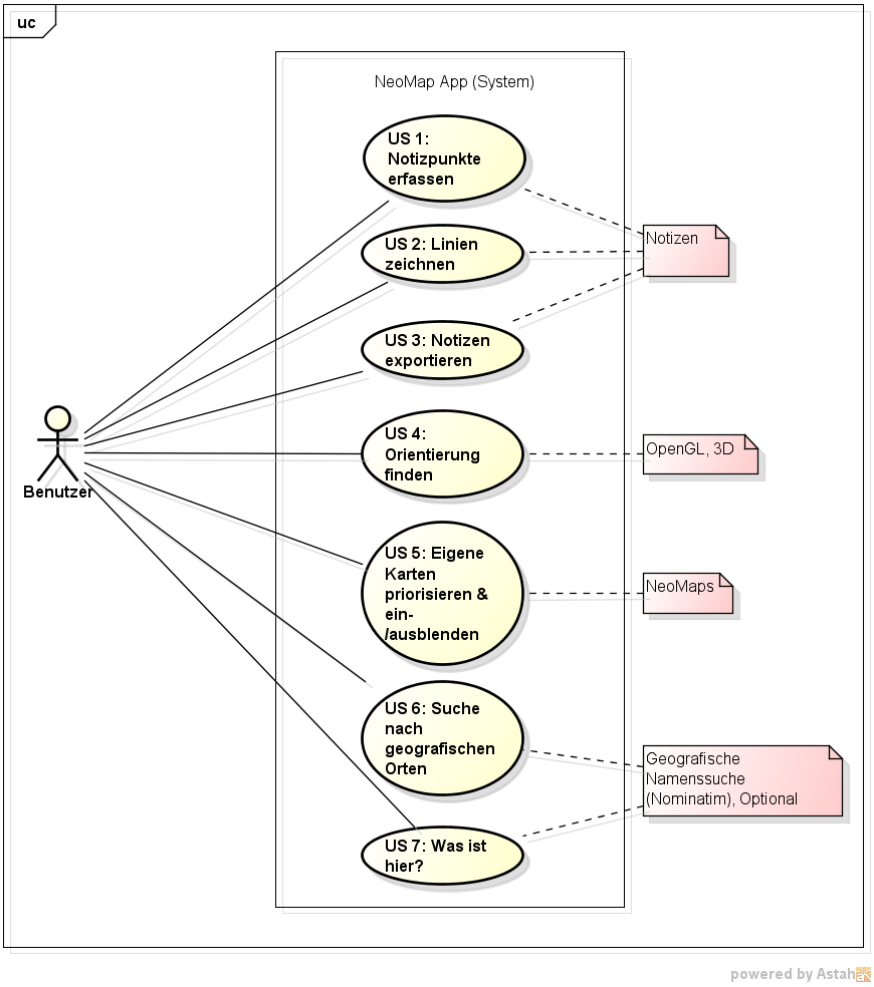


Diagramm 2: User Szenarien

3.2 User Szenarien

3.2.1 US 1: Notizpunkte erfassen

Patrick ist heute wieder mal in Wanderstimmung und macht eine kleine Tour in die Berge. Unterwegs trifft er auf einen besonders schönen, aber versteckten Rastplatz. Dieser ist natürlich auf keiner seiner Karten ersichtlich. Glücklicherweise hat er jedoch sein Smartphone dabei, auf welchem er zuvor die NeoMap App installiert hatte. Kurzerhand öffnet er die App, lässt ihr kurz Zeit um seinen Standort zu lokalisieren, drückt dann lange auf die Stelle an der er sich gerade befindet, wählt „Notiz für aktuellen Punkt hinzufügen“ und erfasst eine Notiz mit dem Namen „Super schöner Rastplatz“.

Nachdem Patrick die Notiz gespeichert hat, befindet sich an dieser Stelle ein kleines Symbol, welches den Standort der Notiz kennzeichnet. Zu Hause angekommen, möchte Patrick seinem Freund Simon den gefunden Platz nun zeigen, er öffnet die App und wählt aus dem Menü „Notizen verwalten“ die neu erfasste Notiz aus. NeoMap lokalisiert automatisch den erfassten Standort der Notiz und zeigt wiederum einen Pin an.

Ziele:

- Notizpunkte an beliebigen Orten erfassen
- Erfasste Notizpunkte können entweder durch das Menü „Notizen verwalten“ oder direkt auf der Karten angezeigt werden

3.2.2 US 2: Linien zeichnen

Patrick plant eine Wanderung und möchte diese im Voraus planen. Deshalb zückt er sein Smartphone und öffnet die NeoMap App. Dort sucht er sich den Ort für seine Wanderung aus. Am gewünschten Startpunkt drückt er anschliessend lange auf den Bildschirm und wählt den Punkt „Liniengeometrie erfassen“. Somit befindet er sich im Erfassungsmodus. Mittels Klicks auf verschiedene Stellen auf der Karte, kann er Linien ziehen und schlussendlich noch eine Notiz dazu erfassen.
Die erfasste Route ist nun jederzeit auf der Karte sichtbar oder kann über das Menü „Notizen verwalten“ gefunden werden.

Ziele:

- Linien bzw. Routen erfassen und mit einer Notiz versehen
- Erfasste Linien können entweder durch das Menü „Notizen verwalten“ oder direkt auf der Karte angezeigt werden

3.2.3 US 3: Notizen exportieren

Patrick hat bereits einige Notizen in Form von Punkten sowie Linien erfasst. Cyrill möchte diese gerne für sein Garmin-Gerät benutzen können. Für Patrick ist dies kein Problem, er exportiert alle erfassten Notizen kurzerhand über den Menüpunkt „Notizen verwalten“. Die Notizen werden dabei in einzelne Dateien im GPS Exchange Format (GPX) gespeichert. Jetzt kann Patrick die exportierten Dateien einfach an Cyrill weiter schicken.

Ziele:

- Alle Arten von Notizen exportieren können (einzeln oder alle zusammen)
- Die Export Dateien sollen sich in einem bekannten, wohl definierten Format befinden (GPX)

3.2.4 US 4: Orientierung finden

Simon ist zum ersten Mal in Rapperswil. Zur Orientierung benutzt er sein Smartphone mit der NeoMap App. Leider ist Simon nicht so begabt im Lesen von Karten, deshalb hilft ihm auch die Kompassfunktion der App nicht weiter. Zum Glück schafft NeoMap auch dagegen Abhilfe. Simon schaltet mit einer Berührung auf das entsprechende Symbol die 3D-Schrägansicht (welche eine Schräg-Ansicht auf die Karte ist) ein. So sieht Simon auf der Karte etwa dasselbe, was sich auch real in seiner Blickrichtung befindet. Er kann jetzt leicht sehen, was sich rechts und links von der Strasse auf der er gerade geht befindet und kann sich so orientieren.
Später geht Simon noch an die HSR. Auf dem HSR Gelände und mit der NeoMap App nützt ihm die Schrägansicht nichts mehr, er schaltet die Kippfunktion deshalb einfach wieder aus.

Ziele:

- Orientierung finden
- Funktion jederzeit abschalten können

3.2.5 US 5: Eigene Karten priorisieren und ein-/ausblenden

Simon hat für die HSR zwei verschiedene NeoMaps. Eine davon ist die offizielle Gebäude-Ansicht der HSR, die andere ist eine Karte der HSR in der er und seine Clique ihre „geheimen“ Treffpunkte eingezeichnet haben. Logischerweise kann er nicht beide gleichzeitig anzeigen, denn diese überlappen sich fast vollständig. Glückerweise kann Simon mit einem Klick in die NeoMap Verwaltung wechseln und eigene Karten leicht ein- und ausblenden. So kann er jederzeit zwischen seinen eigenen Karten hin-und-her wechseln.

Simon hat zusätzlich auch noch eine eigene Karte von Rapperswil. Er will grundsätzlich diese Karte anzeigen und zusätzlich auch noch seine eigene Karte von der HSR. Damit er die Karte von der HSR sieht, könnte er die Karte von Rapperswil ausblenden, was jedoch etwas unnötig scheint, da die Überlappung schliesslich nur genau an der HSR stattfindet. Statt die Karten ein- und auszublenden kann Simon, im gleichen Menü in dem er die Karten ein und ausschaltet, die Karten mittels Drag & Drop einfach priorisieren. Karten welche zuoberst in der Liste kommen, haben eine höhere Priorität als Karten die tiefer in der Liste sind und werden in der Kartenansicht damit auch oberhalb den anderen Karten dargestellt.

Ziele:

- Karten über das Ebenen-Menü standortabhängig ein- und ausschalten (auf Tablets mittels Fragments)
- Karten die sich überlappen priorisieren können (Welche Karte ist zuoberst und überlappt allfällige tiefere Karten am gleichen Ort)

3.2.6 US 6: Optional: Suche nach geografischen Orten

Simon möchte wissen wie genau die Karte von London aussieht. Er öffnet NeoMap und gibt in der Suche „London“ ein. Da Simon vorher noch nie nach „London“ gesucht hat ist eine Internetverbindung von Nöten. Die Suche erzielt einen Treffer, die Karte fokussiert London und markiert die Stelle mit einem Icon. Simon kann sich nun London in Ruhe anschauen.

Ein paar Tage später trifft Simon Patrick und möchte ihm etwas zeigen, dass er in London gesehen hat. Patrick lehnt ab und sagt, er habe selber ein Smartphone mit Google Maps. Doch einen Moment später wird Patrick plötzlich ganz kleinklaut, denn so wie es aussieht funktioniert seine Internetverbindung im Moment nicht und er kann London auf Google Maps nicht suchen. Simon muss schmunzeln, öffnet dann aber trotzdem seine NeoMap App und sucht nach London. Da in der NeoMap App alles offline verfügbar ist, funktioniert die Suche einwandfrei.

Ziele:

- Geografische Orte (Länder, Städter, Strassen, etc.) per Suche finden
- Für die erste Suche eines bestimmten geografischen Ort muss eine Internetverbindung vorhanden sein
- Für alle weiteren Male soll die Suche komplett offline funktionieren

3.2.7 US 7: Optional: Was ist hier?

Patrick befindet sich gerade an einer Strasse in Rapperswil deren Namen er vergessen hat, will diesen aber unbedingt wissen. Für ihn kein Problem: Er zückt sein Smartphone, öffnet die NeoMap App und lässt seinen Standort lokalisieren. Leider ist auf der angezeigten Karte der Name der Strasse nicht ersichtlich – jedoch gibt es natürlich auch für das eine Lösung. Patrick drückt lange auf die Strasse und wählt im Menü „Was ist hier?“. Ihm wird angezeigt, was für eine Strasse sich an diesem Ort bzw. diesen Koordinaten befindet. Ab sofort stehen diese Informationen natürlich auch offline zur Verfügung und können auch über die normale Namenssuche gefunden werden.

Ziele:

- Koordinaten auflösen können (Funktion „Was ist hier?“, Reverse Geografische Namenssuche)
- Für die erste Suche muss eine Internetverbindung vorhanden sein
- Für alle weiteren Male (für die gleiche Stelle!) soll die Suche komplett offline funktionieren

4 Architektur

4.1 Gesamtübersicht

Die NeoMap App kommuniziert mit verschiedenen Servern, um alle Informationen von der Darstellung der Grundkarte, den zusätzlichen NeoMap-Karten sowie zur Durchführung von Geonamen-Suchen zu erhalten.

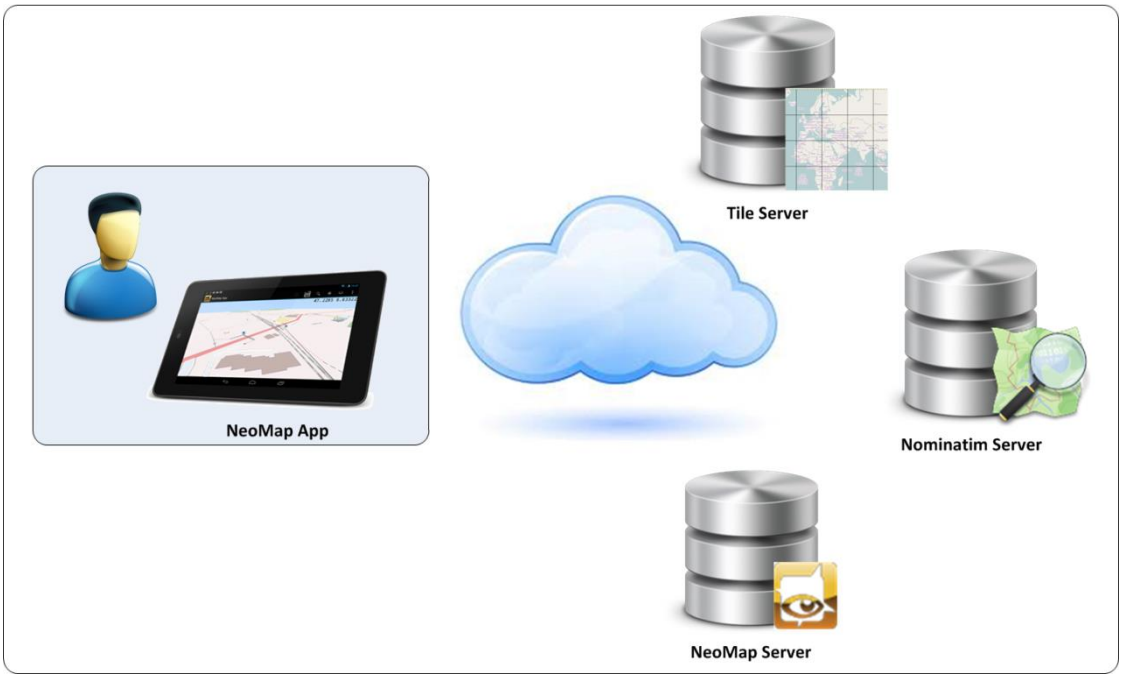


Diagramm 3: Gesamtübersicht der Architektur

4.2 Projektstruktur

Die Projektstruktur ist massgeblich durch den Einsatz des Android SDKs vorgegeben. In der folgenden Tabelle ist jedoch noch eine Beschreibung der wichtigsten Elemente und Verzeichnisse des Projekts ersichtlich.

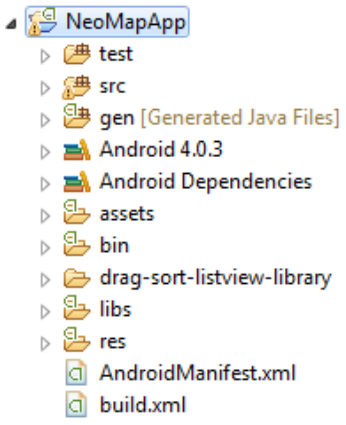


Abbildung 11: Projektstruktur

Verzeichnis	Beschreibung
test	Test-Klassen befinden sich alle unter diesem Verzeichnis.
src	Der gesamte Quellcode wird in diesem Verzeichnis abgelegt.
gen	Enthält die von den Android Development Tools (ADT) generiert und verwaltete „R“ Datei. Diese wird benötigt, um direkt im Quellcode auf Ressourcen welche sich im res Ordner befinden zuzugreifen.
assets	Zusätzliche Elemente, wie beispielsweise Icons welche nicht direkt über die „R“ Datei als UI-Elemente eingebunden werden, sind in diesem Verzeichnis abgelegt.
res	Ausgelagerte Elemente wie Layout-Definitionen oder auch übersetzte Texte werden hier zentral abgelegt.
AndroidManifest.xml	Zentrale Beschreibungs- und Konfigurations-Datei einer Android-Anwendung. Hier werden unter anderem die Haupt-Komponenten, benötigte Berechtigungen und Hardware-Anforderungen definiert.
build.xml	Ant-Script zur Generierung eines APK-Files, welches zur Installation auf einem Gerät verwendet wird.

Tabelle 17: Projektstruktur

4.3 Grundlegende Architektur Übersicht

Das App wurde grundsätzlich in drei Layer aufgeteilt und die Klassen nach entsprechender Zugehörigkeit zugeteilt. Dabei wurde darauf geachtet, dass der Presentation-Layer sauber von den restlichen Layern getrennt wurde und keine Zugriffe von unteren Layern auf Klassen des Presentation-Layers stattfinden.

Layer	Beschreibung
Presentation-Layer	Der Presentation-Layer beinhaltet alles was für die Darstellung sowie die direkte Interaktion mit dem Benutzer benötigt wird.
Application-Layer	Die eigentliche Logik des Apps wurde in den Application-Layer gekapselt.
Data-Layer	Über den Data-Layer können Daten aus dem lokalen Storage, einer lokalen Datenbank, oder aber auch über das Internet bezogen werden. Alle Zugriffe des Apps auf externe Quellen werden somit über diesen Layer durchgeführt.

Tabelle 18: Package-Struktur-Beschreibung

4.3.1 Package-Struktur

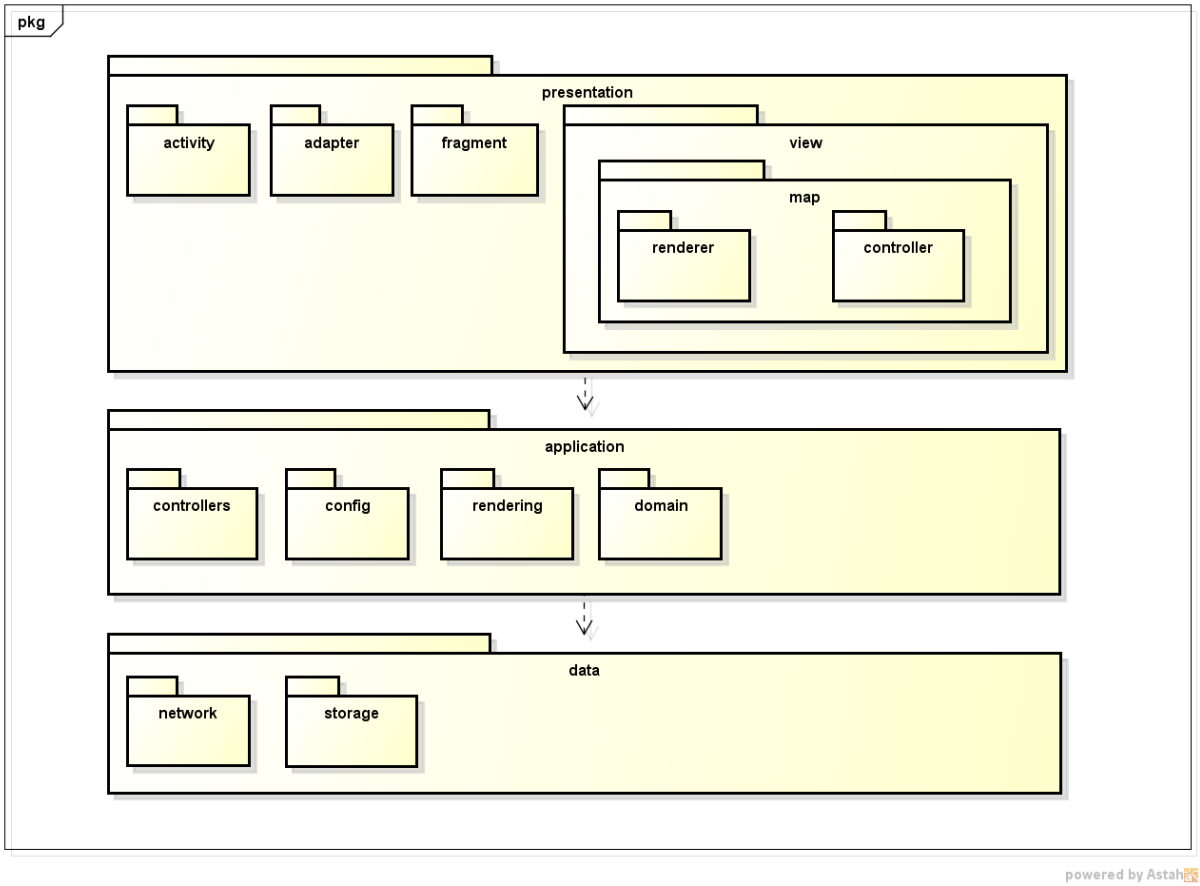


Diagramm 4: Package-Struktur

4.4 Persistenz

Zur Persistierung von Daten werden folgende von Android angebotene Möglichkeiten genutzt.

4.4.1 SharedPreferences

Ein von Android angebotener Key-Value Storage, welcher es ermöglicht, Daten separat privat pro Applikation auf dem Gerät zu speichern. Diese Option wird vor allem für benutzerdefinierte Daten verwendet.

4.4.2 External Storage

Das Android Framework bietet auch die Möglichkeit, Daten auf dem externen Speicher eines Gerätes abzulegen. Hierbei handelt es sich beim NeoMap App vorwiegend um zwischengespeichertes Karten-Material, wie einzelne NeoMaps oder OSM-Tiles.

4.4.3 SQLite Database

Des Weiteren können strukturierte Daten auch in privaten Datenbanken auf der Basis von SQLite abgelegt werden. Dies wird verwendet um detaillierte Informationen zu Karten-Overlays, bspw. NeoMap-Karten oder Notizen, zu persistieren. Alle Datenbankoperationen finden über die Klasse „DbController“ statt. Diese ist momentan die einzige Klasse, mit der ein Zugriff auf die Datenbank möglich ist. Dabei gibt es auch keine Subklassen für Notizen oder NeoMaps. Stattdessen befinden sich alle Methoden direkt in dieser Klasse. Dies ist ein Punkt, der sicherlich verbessert werden kann (siehe dazu „Datenschicht besser aufteilen (DAOs / ORM)“).

4.5 Grundlegende Klassen

MapActivity
Activities repräsentieren unter Android ein sichtbares User-Interface. Ein App kann aus mehreren Activities bestehen, jedoch muss über das Android-Manifest festgelegt werden, welches die Startup-Activity ist. Die NeoMap App besteht neu aus einer einzigen Activity. Die verschiedenen Teile der Benutzeroberfläche werden jeweils durch die Verwendung von Fragments realisiert.
Die Klasse MapActivity stellt somit den Eintrittspunkt des Apps dar und ist eine abgeleitete Klasse der Android Basis-Klasse Activity.

ApplicationController
Um in Android einen globalen Applikations-Status zu erhalten, besteht die Möglichkeit die abgeleitete Klasse von der Android Basis-Klasse Application zu erstellen. Diese Klasse muss im Android-Manifest deklariert werden und wird somit auch instanziiert sobald die Applikation gestartet wird. Die Instanz bleibt solange erhalten, bis die Applikation vom System geschlossen wird. Sie bietet somit die Möglichkeit, Daten zwischen verschiedenen Activities auszutauschen oder zwischen zu speichern im Falle dass eine Activity, bspw. Aufgrund einer Display-Rotation, neu gestartet werden.

DbController
Der DbController ist ein Singleton Objekt, welche von der Android-Klasse „SQLiteOpenHelper“ abgeleitet ist. Sie enthält alle Methoden und SQL-Statements welche für den Datenzugriff und das Erstellen von Tabellen nötig sind. Datenbank-Operationen sollten jeweils immer über diesen Controller durchgeführt werden.

PreferencesController
Einstellungen sowie Zustände und Informationen der letzten App Benutzung werden in den von Android angebotenen SharedPreferences abgelegt. Um den Zugriff, auf diese immer wieder verwendeten Einstellungen, zu vereinheitlichen wurde die Klasse PreferencesController eingeführt.

4.6 Techniken

4.6.1 AsyncTask und Callback

Um ein möglichst responsives und flüssiges UI zu bieten, ist es in Android unvermeidlich mit zusätzlichen Threads zu arbeiten. Die Klasse AsyncTask ist eine einfache Template-Klasse, welche es erlaubt, Aufgaben auf einfache Weise ausserhalb des UI-Threads zu erledigen. Somit blockieren langandauernde Aufgaben (wie z.B. ein Login-Request) nicht den UI-Thread. In der NeoMap App werden alle AsyncTasks über eine eigene „Executor“-Klasse ausgeführt. Dies hat den Grund, dass von Haus aus AsyncTasks zwar parallel zum UI-Thread laufen, jedoch nicht mehrere AsyncTasks parallel zueinander. Dies bedeutet, dass standardmässig immer nur ein AsyncTask gleichzeitig laufen kann. Um nun mehrere simultane AsyncTasks zu erlauben, wurde deshalb die Klasse „AsyncTaskController“ eingeführt. Diese startet jeden AsyncTask direkt als eigenen Thread und hebt die Limitierung, dass nur ein AsyncTask gleichzeitig aktiv sein kann, auf.
Alle Klassen die von AsyncTask ableiten, müssen mindestens die Methode „doInBackGround“ überschreiben. Diese läuft in einem eigenen Thread, die anderen Methoden laufen alle im UI-Thread (natürlich nur sofern sie nicht innerhalb der „doInBackGround“ Methode aufgerufen werden).In den allermeisten Fällen überschreibt man auch die Methode „onPostExecute“ und setzt einen eigenen Konstruktor. Über die „onPostExecute“ Methode kann das Resultat, dass die „doInBackGround“ Methode liefert weiter verwendet werden. Man könnte hier auch direkt Updates auf das GUI machen um Dialoge und ähnliches anzeigen (da die Methode im UI-Thread läuft). In den meisten Fällen ist dies jedoch nicht sehr sinnig.

Denn einerseits sollte ein AsyncTask sich wirklich ausschliesslich um seine eigentliche Aufgabe kümmern (Single-Responsibility-Principle) und andererseits will man meist nicht die konkrete GUI-Klasse (d.h. Activity oder Fragment) als Parameter mitgeben. Stattdessen wird mit Interfaces gearbeitet, um eine hohe Kopplung an den Presentation-Layer zu vermeiden.
An dieser Stelle kommt auch das Callback ins Spiel: In der NeoMap App gibt es ein einfaches Interface namens „IOnTaskCompleted<T>“ welches nur eine einzige Methode enthält: „onTaskCompleted(T result)“. Klassen welche einen oder mehrere AsyncTasks brauchen implementieren dieses Interface und stellen somit eine Callback-Methode zur Verfügung. Beim Erstellen des AsyncTasks gibt man dann die Klasse selbst (als Interface!) als Parameter mit. Die AsyncTask-Klasse kann dann bei „onPostExecute“ ganz einfach einen Callback auf das Interface machen um das Resultat mitzuteilen.

4.6.2 Observer-Pattern

In der ursprünglichen Fassung des Apps wurde anstelle der offiziellen Java-Observer Klassen, eigene Klassen verwendet. Dabei wurden meistens Interfaces mit dem Namen „XXXDelegate“ gebraucht. Dies hatte folgende grosse Nachteile:

- Es ist nicht direkt ersichtlich, was dieses „Delegate“ Interface macht
- Fehleranfälligkeit (kein synchronized beim Entfernen von Observern, etc.)
- Keine detaillierte Unterscheidung von verschiedenen Eventtypen (Beispielsweise hat die Klasse Perimeter mehrere Observer, welche an verschiedenen Ereignissen interessiert sind. Es gibt jedoch nur eine Methode „onPermiterChanged“, welche immer alle Observer informiert. Diese haben keine Möglichkeit zu evaluieren, ob das Ereignis für sie überhaupt relevant ist.)

Aus diesem Grund wurden die „xxxDelegate“-Klassen grösstenteils entfernt und die offiziellen Java Observer-Klassen benutzt. Mittels drei Hilfsklassen werden die Benachrichtigungen erstellt und wieder ausgelesen.

Klasse	Beschreibung
UpdateBundle	Eine Wrapper-Klasse für die Klasse „Bundle“ von Android. In ihr wird zum einen immer der Grund für die Benachrichtigung gespeichert (UpdateReason) und zum anderen gegebenenfalls die neuen Werten (UpdateValues) gespeichert.
UpdateReason (Enum)	Enthält alle möglichen Update-Gründe, wie z.B. „SENSOR_AZIMUTH_CHANGE“. In diesem Fall wäre dies eine Änderung der Kompass-Ausrichtung.
UpdateValue (Enum)	Enthält Enums für die Zuordnung von geänderten Werten, wie z.B. „AZIMUTH“. In diesem Fall wäre dies der konkrete Wert der Kompass Ausrichtung. Dieser Enum wird im UpdateBundle immer als Paar mit dem zugehörigen Wert abgelegt z.B. „AZIMUTH, 321.23“. Dieser Wert kann von den Observern dann ausgelesen werden.

Tabelle 19: Observer Hilfsklassen

4.7 Internationalisierung

Die Internationalisierung von Texten wurde mittels des vom Android SDK gebotenen Mechanismus gelöst. Dieser erlaubt es mehrere XML-Files für verschiedene Sprachen zu erstellen, in welchen Texte verwaltet werden können.
Im Verzeichnis „res“ können die verschiedenen Sprachen in einem Ordner Namens "values-<Sprache>" abgelegt werden. Dies wird von Android selbst entsprechend aufgelöst. Ist keine Version für die Sprache des Benutzers vorhanden, wird jeweils das Standard Verzeichnis "values" verwendet.

5 Karten-Darstellungsgrundlagen (Rendering)

5.1 Konzept

Die sichtbare Karte ist modular aufgeteilt und kann aus mehreren Layern bestehen. Basis der Karte ist dabei der OSM-Layer. Alle übergeordneten Layers werden daher als Overlays angesehen und werden in weiteren Kapiteln auch so angesprochen. Alle aktuell möglichen Layer werden nachfolgend aufgelistet. In der App besteht die Möglichkeit, diese Layers auch einzeln ein- und auszublenden.

ID	Layer-Name	Funktion
0	OSM	Stellt die Grundfunktion der Karten-App dar, zeigt die OpenStreetMap-Karte an.
1	NeoMap	Über den NeoMap-Layer werden NeoMaps als Overlay auf die OSM-Karte gelegt.
2	Notes	Ein weiteres Overlay, welcher für das Darstellen von Notiz-Pins verantwortlich ist.
3	Nominatim	Ergebnisse der geografischen Namenssuche werden mittels eines Icons auf der Karte über diesen Layer angezeigt.
4	OwnPosition	Bei aktivierter Standort-Bestimmung wird hierüber ein Icon als Overlay an der entsprechenden Position angezeigt.
5	Coordinates	Das Fadenkreuz in der Mitte sowie die Koordinaten in der oberen rechten werden über diesen Layer im Vordergrund der Karte angezeigt

Tabelle 20: Aufzählung möglicher Karten-Layer

Wie bereits erwähnt, können diese Layers einzeln hinzugefügt oder von der Darstellung entfernt werden. Nachfolgend eine Illustration, die das sogenannte Layering-Konzept veranschaulicht.

Abbildung 12: Eine Auswahl an verschiedenen Karten-Ebenen

5.2 Umsetzung

Im nachfolgenden Klassendiagramm sind alle Klassen aufgezeichnet, welche massgeblich am Rendering-Prozess beteiligt sind. Auch die Umsetzung des Layering-Konzeptes ist darin abgebildet. Es sind dabei nur die wichtigsten Klassen und Methoden, in einer vereinfachten Form, aufgeführt.

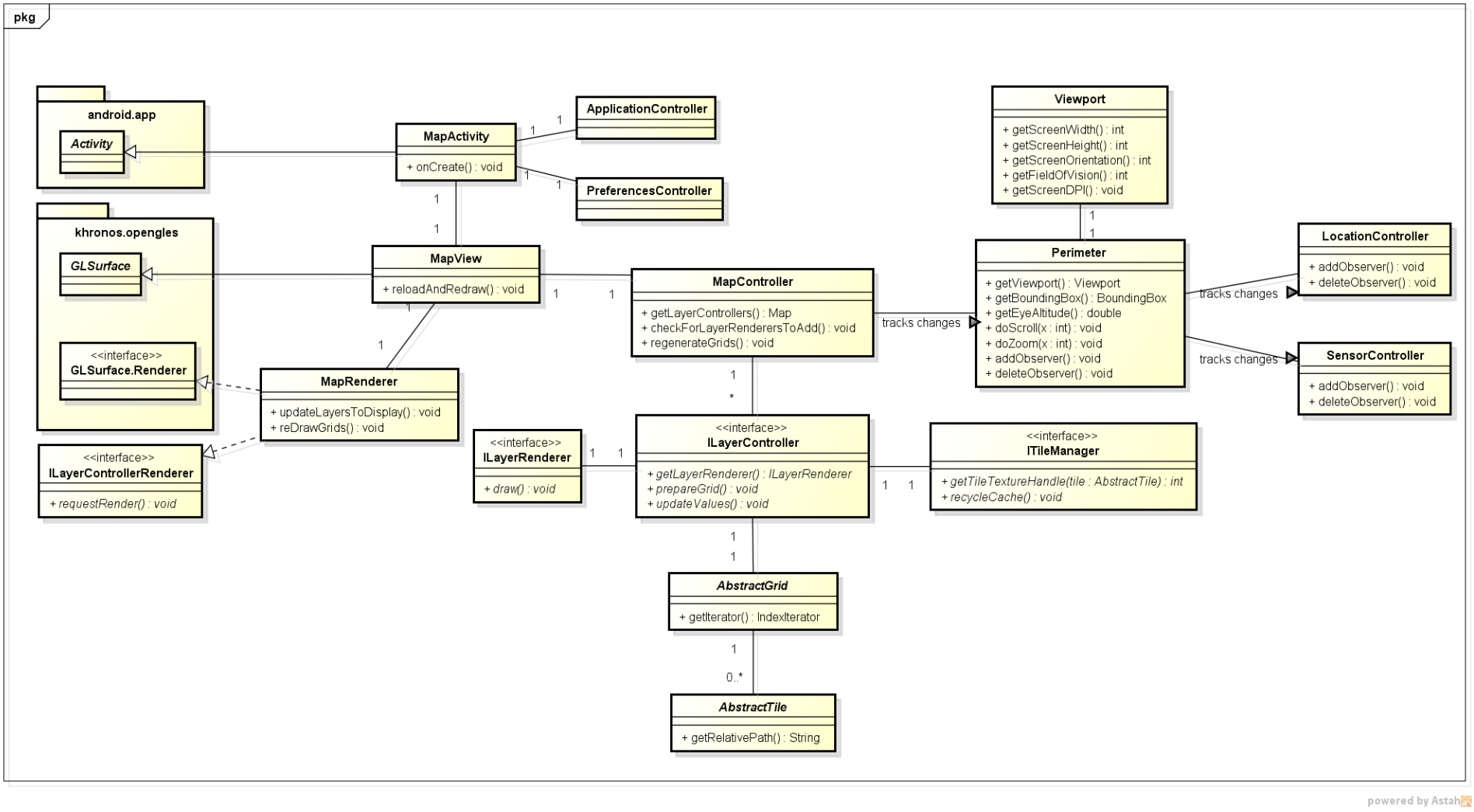


Diagramm 5: Rendering Klassendiagramm

5.2.1 Generelle Rendering Klassen

MapView
Diese Klasse ist von der GLSurfaceView-Klasse aus der Android-API abgeleitet und stellt die Oberfläche dar, auf welcher später mittels eines Renderers die Karte gezeichnet werden kann. Alle Touch-Inputs des Users werden in dieser Klasse mittels diverser Gesture-Listeners entgegengenommen und verarbeitet. Meist enden diese Touch-Inputs in einer Anpassung von Eigenschaften der Perimeter-Klasse, das zentrale Kontrollinstrument im Bezuge auf die Darstellung der Karte. Touch-Inputs sind dabei beispielsweise die Pinch-Geste oder ein Double-Tap um zu zoomen.

MapController
Die Klasse MapController enthält die Logik, um einzelne Overlays zu erstellen und zu verwalten. Da bei sämtlichen Änderungen wie Zoom-Stufe, Kompass-Ausrichtung der Karte oder auch Standort-Lokalisierung die Karte neu ausgerichtet und gezeichnet werden muss, meldet sich der MapController als Observer des Perimeters an und erhält somit eine Benachrichtigung bei entsprechenden Änderungen. Je nach Änderung muss ein Layer neu erstellt werden oder ein bestehender aus den aktiven Overlays gelöscht werden. Danach wird jeweils der Vorgang des Neu-Zeichnens der gesamten Karte über den MapRenderer angestossen.

MapRenderer
Der MapRenderer ist das zentrale Element beim Darstellen der Karte. Die Klasse ist von der Android-API Klasse GLSurfaceRenderer abgeleitet und enthält Code, welcher das Rendern der Karte vorbereitet und letztlich durchführt. Unter anderem wird hier die Perspektive, in welcher die Karte angezeigt wird, definiert, sei dies nun beispielsweise eine leicht gekippte Schräglagen-Perspektive oder eine orthogonale Ansicht.

LayerType
Definiert die zur Verfügung stehenden Map-Layers und deren Priorität beim Zeichnen.

Perimeter
Die Perimeter-Klasse ist das Kontroll-Instrument der Karten-Anzeige. Darüber kann gesteuert werden, welche Zoom-Stufe verwendet, ob die aktuelle Position geortet oder auch ob die Karte mittels des Kompasses ausgerichtet werden soll. Andere Objekte, welche an Änderungen dieser Werte interessiert sind, können sich bei dieser Klasse als Observer anmelden. Um Informationen wie die aktuelle Position oder die Haltung des Gerätes zu erhalten, meldet sich die Perimeter-Klasse als Observer beim LocationController und dem SensorController an.

BoundingBox
Die BoundingBox-Klasse enthält immer genau zwei Koordinaten. Dies sind eine Nordwest- und eine Südost-Koordinate. Durch diese beiden Koordinaten entsteht eine rechteckige Begrenzungsbox, in welche z.B. eine NeoMap eingebettet werden kann. Zum Verständnis ist dies hier beispielhaft dargestellt. Auch wird diese Klasse dazu verwendet, den sichtbaren Bereich einer Karte zu defnieren.

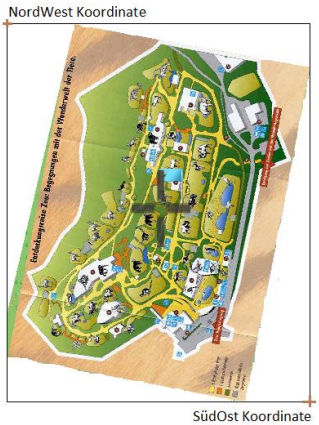


Abbildung 13: BoundingBox mit eingebetteter NeoMap

Viewport
Die Viewport-Klasse enthält spezifische Informationen, welche für die finale Darstellung der Karte relevant sind. Unter anderem enthält sie die DPI-Anzahl des Gerätes und die Breite sowie die Höhe des verfügbaren Karten-Bereiches.

LocationController
Die aktuelle Position des Users wird über den LocationController ermittelt und verwaltet. Andere Klassen können sich bei ihm als Observer anmelden, um Informationen oder Änderungen des aktuellen Standpunktes zu erhalten. Die Logik, ob eine Ortung per Mobilfunk-Netz oder GPS, wie häufig ein Update des Standortes abgerufen wird und wie genau ein Signal sein muss, um als aktueller und besser zu gelten, wird über den LocationController ausgeführt.

SensorController
Um die Karte entsprechend der aktuellen Haltung des Gerätes auszurichten, ist die Klasse SensorController verantwortlich. Sie überwacht die im Gerät integrierten Sensoren und gibt Informationen wie z.B. die Kompass-Ausrichtung oder auch die Schräglage des Gerätes an angemeldete Observer weiter.

5.2.2 Generischer Aufbau eines Layers

Ein kompletter Layer setzt sich dabei aus Implementierungen der folgenden Interfaces respektive abstrakten Klassen zusammen. Die einzelnen Layers werden, wie bereits zuvor erwähnt, vom MapController verwaltet.

ILayerController
Jeder Layer besitzt als Hauptklasse einen eigenen, sogenannten Layer-Controller, welcher die Logik enthält um das Zusammenspiel der verschiedenen Komponenten eines Layers zu gewährleisten. Über ihn wird definiert, in welchem Fall ein neues Grid gezeichnet werden soll, welche Daten geladen und wie genau diese Informationen dargestellt werden sollen.

ILayerRenderer
Wird beim effektiven Zeichnen eines Layers benötigt. Da das Rendering in einem eigenen Thread durchgeführt wird, wird die draw-Methode stets über den OpenGL-Thread aufgerufen. Seine Logik ist darauf beschränkt, alle Elemente des aktuellen Grids seines Layers zu zeichnen.

ITileManager
Enthält das Wissen und die Möglichkeiten Bitmaps zu den einzelnen Tiles zu laden und wenn benötigt auch zu cachen. Allgemein gesagt, führt er die Verwaltung der Texturen eines Layers durch. Ein Handle zu einem Tile kann über die getTileTexture Methode angefordert werden. Falls kein Handle zur Verfügung steht, wird jeweils -1 als Rückgabewert verwendet. Ist ein noch kein Handle verfügbar, wird jeweils ein asynchrones Laden des Bitmaps angestossen.
Da teilweise ein interner Cache vorhanden ist und dieser von Zeit zu Zeit aufgeräumt werden muss, wurde die Methode recycleCache eingeführt, welche jeweils zu Beginn eines Rendering-Calls aufgerufen wird und somit zyklisches Aufräumen des Caches ermöglicht.

AbstractGrid
Ein sogenanntes Grid verwaltet die genauen Positionen der Elemente des jeweiligen Layers und führt eine Liste der einzelnen Tiles des Grids. Jeder Punkt in einem Grid wird dabei einem Tile zugeordnet oder ist leer.

AbstractTile
Enthält Informationen über ein einzelnes Tile. Sei dies ein Kartenausschnitt, eine einzelne Notiz oder auch der aktuelle Standort. Dies ist je nach Layer variierend. Ein Tile muss jedoch zwingend einige Informationen, wie z.B. die Koordinate zu der es gehört, beinhalten um sich genauer zu identifizieren.

Das Zusammenspiel der verschiedenen Komponenten soll anhand des nachfolgenden Sequenzdiagramms vereinfacht aufgezeigt werden.

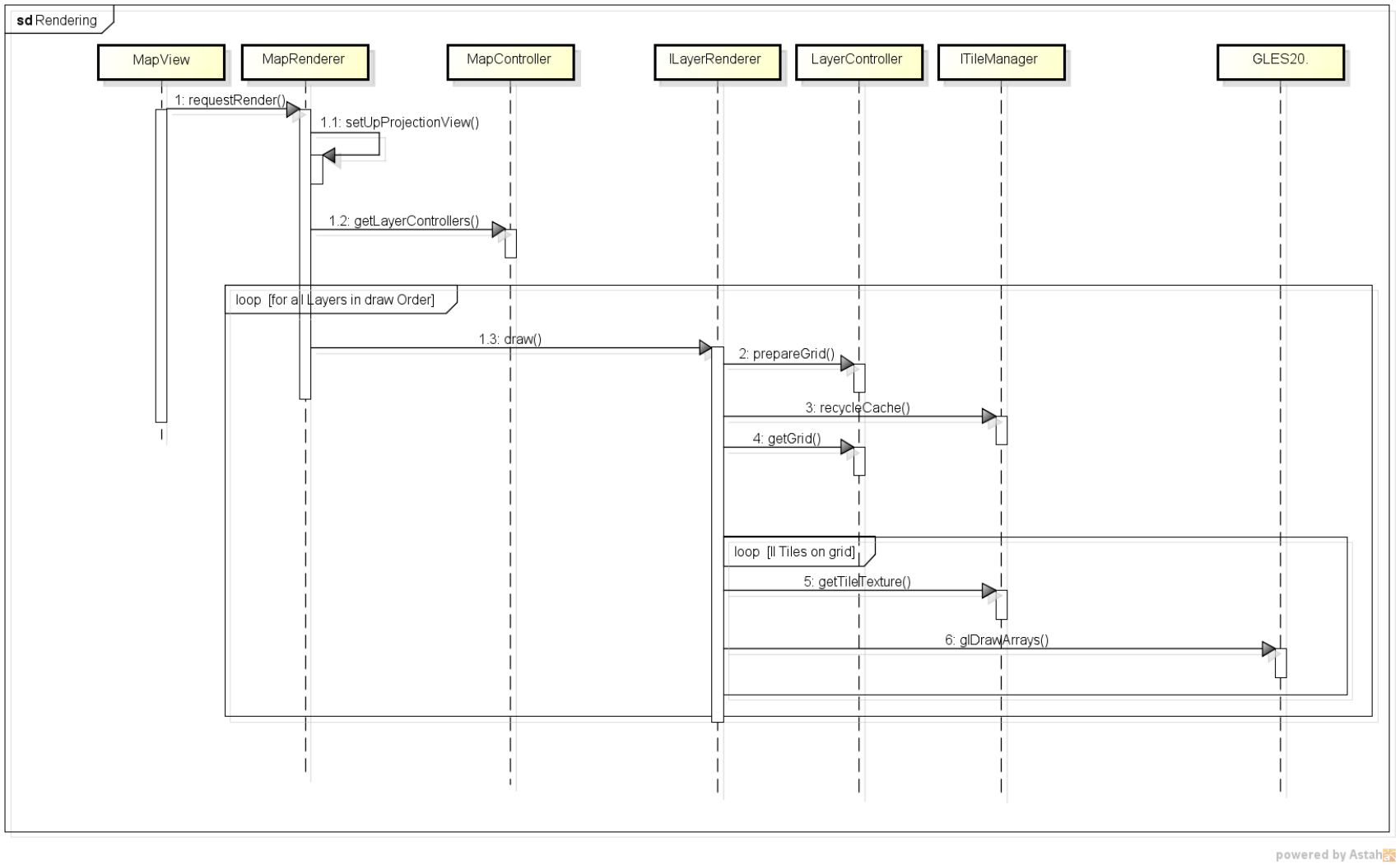


Diagramm 6: Rendering Sequenzdiagramm

5.3 OSM-Layer

5.3.1 Konzept

Aufgrund der aktuellen Koordinaten, welche sich im Mittelpunkt des Sichtfeldes befinden, und der aktuellen Zoom-Stufe wird berechnet, welcher Bereich der Karte für den Benutzer sichtbar ist. Für diesen sichtbaren Bereich wird dann ein Raster generiert, in welchem später die einzelnen Tiles als Texturen geladen werden können. Nachfolgend eine kleine Illustration dazu.

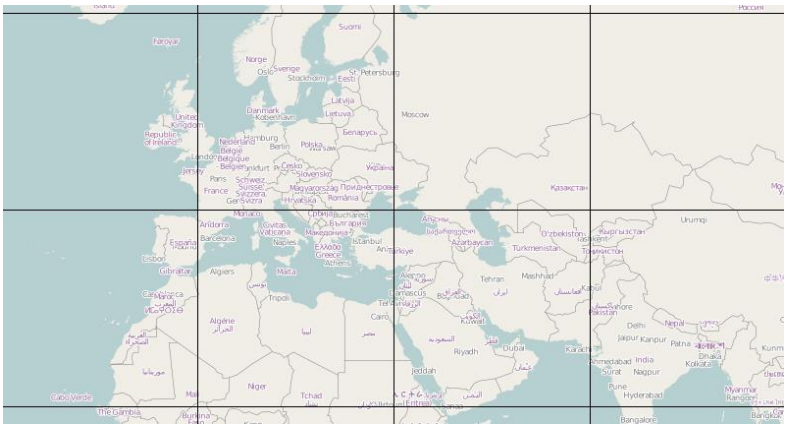


Abbildung 14: Raster für die Basis-Karte

Bei der ersten Anzeige einer Region, müssen diese Tiles erst von einem sogenannten Tile-Server angefordert und heruntergeladen werden. Danach werden diese, um die Offline-Nutzung zu gewährleisten, auf der SD-Karte gespeichert. Bei Bedarf kann das heruntergeladene Kartenmaterial über die Einstellungen wieder gelöscht werden.

5.3.1.1 Kartenmaterial

Durch die Verwendung des Pyramiden-Prinzips beim Kartenaufbau, bei welchem jedes Tile mittels dessen X und Y-Koordinaten sowie Zoomstufe identifiziert wird, besteht die Möglichkeit, der Benutzung von verschiedenen Tile-Servern. Diese Tile-Server können dabei völlig unterschiedliches Kartenmaterial anbieten. Natürlich muss ein gewisses Schema eingehalten werden (siehe auch „Aufbau einer OpenStreetMap Slippy Map“).
Um dies im NeoMap App zu ermöglichen, wurde ein Feature eingebaut, welches erlaubt zwischen verschiedenen vorgegebenen Tile-Servern auszuwählen oder aber einen benutzerdefinierten Tile-Server anzugeben. Zwischen diesen Tile-Servern kann im laufendend App-Betrieb gewechselt werden, die Karte wird dann sofort neu gezeichnet. Bereits heruntergeladene Tiles eines anderen Servers gehen dabei nicht verloren, sondern bleiben auch bei einem Wechsel des Servers auf dem Gerät gespeichert. Hierzu werden die Tiles jedes Servers in einem eigenen Verzeichnis abgelegt. Für die vorgegebenen Tile-Server existieren dabei klar definierte Ordnernamen, für eigene Tile-Server wird die URL so kodiert, dass sie als Filenamen verwendet werden kann. Dadurch ist sichergestellt, dass ein Ordnername in allen Fällen absolut eindeutig ist.

Folgende Tile-Server sind vorgegeben:

Name	Tile-Server URL
OSM Standard	http://tile.openstreetmap.org/
MapQuest	http://otile1.mqcdn.com/tiles/1.0.0/map/
OpenCycleMap	http://c.tile.opencyclemap.org/cycle/
HikeBike	http://a.www.toolserver.org/tiles/hikebike/

Tabelle 21: Vorgegebene Tile-Server

Der Tile-Server kann in den Einstellungen der NeoMap App gewechselt werden.

In der folgenden Abbildung ist der gleiche Kartenabschnitt auf verschiedenen Tile-Servern dargestellt.
Im ersten Bild wurde der „OSM Standard“ Server gewählt. Im mittleren Bild ist „MapQuest“ ersichtlich und ganz rechts wurde eine eigene Tile-Server-URL (http://a.www.toolserver.org/tiles/osm-no-labels) eingegeben, welche fast das gleiche Kartenmaterial wie OSM-Standard anbietet, abgesehen davon, dass keine Beschriftungen auf der Karte angezeigt werden.

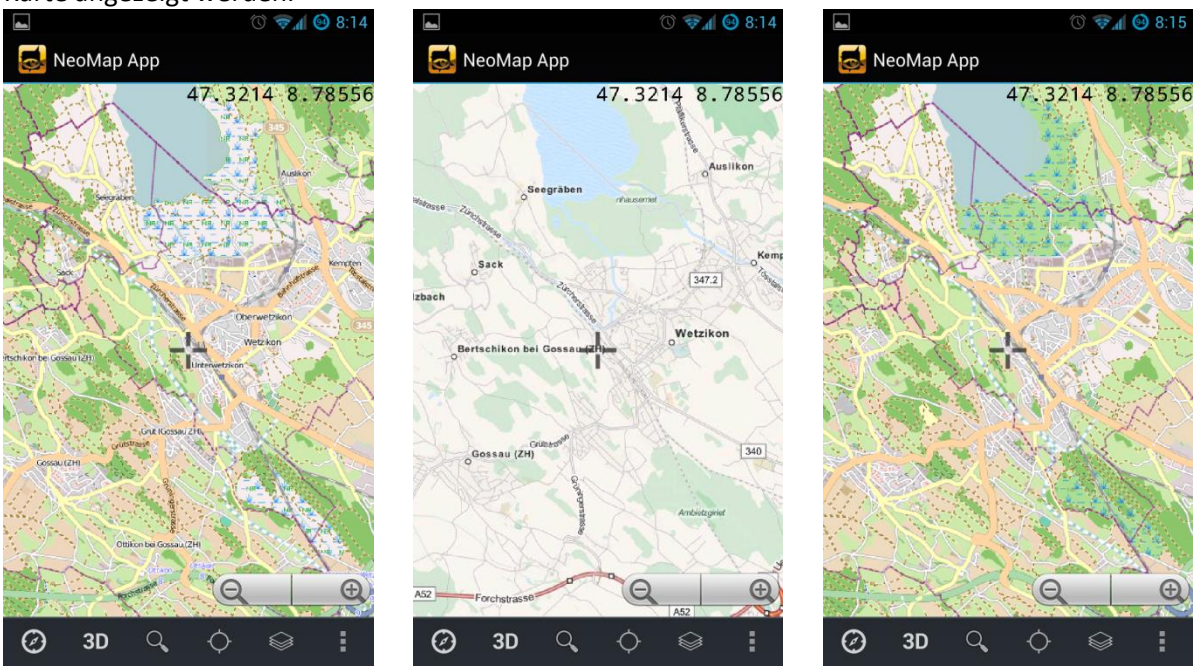


Abbildung 15: Gleicher Kartenabschnitt mit verschiedenen Tile-Servern

5.4 Umsetzung

Um eine Karte anzuzeigen, muss erst ein Raster für den aktuell sichtbaren Kartenausschnitt generiert werden. Die einzelnen Tiles dieses Rasters müssen jeweils beim ersten Aufruf eines Kartenausschnittes erstmals über das Internet von einem Tile-Server heruntergeladen werden. Danach werden die Tiles auf dem externen Speicher des Gerätes abgelegt und bei erneutem Betrachten von dort geladen.

5.4.1 Klassen

Die wichtigsten Klassen, welche am Prozess von der Generierung eines Karten-Rasters bis zur Darstellung der kompletten Karte beteiligt sind, werden nachfolgend aufgelistet und deren wichtigsten Eigenschaften kurz beschrieben.

OSMCalculator
OpenStreetMap spezifische Berechnungen, wie die genaue Berechnung der Karten-Auflösung und der Tile-Koordinaten abhängig des Längen- und Breitengrades wurden, in diese Klasse ausgelagert. Die Klasse basiert auf den offiziell bereitgestellten Berechnungsgrundlagen von OpenStreetMap. http://wiki.openstreetmap.org/wiki/Slippy_map_tilenames#Resolution_and_Scale

Perimeter
Über den Perimeter wird ermittelt, welches die aktuelle BoundingBox der Karte ist. Dies basiert auf der aktuellen Zoom-Stufe, der aktuellen Center-Koordinaten, der DPI-Anzahl des Bildschirmes sowie des Sichtwinkels auf die Karte.

OSMTile
Wichtige Attribute: x, y und zoom
Eine einzelne Kachel wird durch die Klasse OSMTile repräsentiert. Wie bereits erwähnt, kann ein Tile durch dessen x- und y-Koordinaten sowie die Zoom-Stufe identifiziert werden. Der eindeutige Dateiname eines Tiles kann über diese Attribute zusammengesetzt werden. (zoom/x/y.png)

OSMGrid
Ein kompletter Kartenausschnitt wird durch ein Grid repräsentiert. Beim Grid handelt es sich um eine Raster-Einteilung, welche alle Tiles sowie deren exakte Eckpunkt-Koordinaten beinhaltet.
Ein Grid wird jeweils für den aktuell sichtbaren Bereich, welcher im Perimeter als BoundingBox hinterlegt ist, generiert. Dabei wird ein Offset berücksichtigt, welcher etwas grösser als die eigentlichen BoundingBox ist. So wird auch eine Reihe von Tiles ausserhalb des sichtbaren Bereiches geladen. Damit muss bei kleinen Verschiebungen der Karte nicht gleich wieder alles nachgeladen werden. Verlässt ein Benutzer nun jedoch auch den Bereich des Offsets, muss wieder ein neues Grid erstellt werden.

5.5 Laden von Tiles zur Darstellung

Tiles werden jeweils über den TileManager angefordert. Dieser verfügt über das Wissen, wie Tiles eines einzelnen Layers geladen und verwaltet werden müssen. Im Falle des OSM-Layers muss hier ein Caching durchgeführt werden, um die App möglichst performant zu halten.
Im nachfolgenden Diagramm wurden mögliche Abläufe aufgezeichnet, welche beim Anfordern eines Tiles auftreten können.

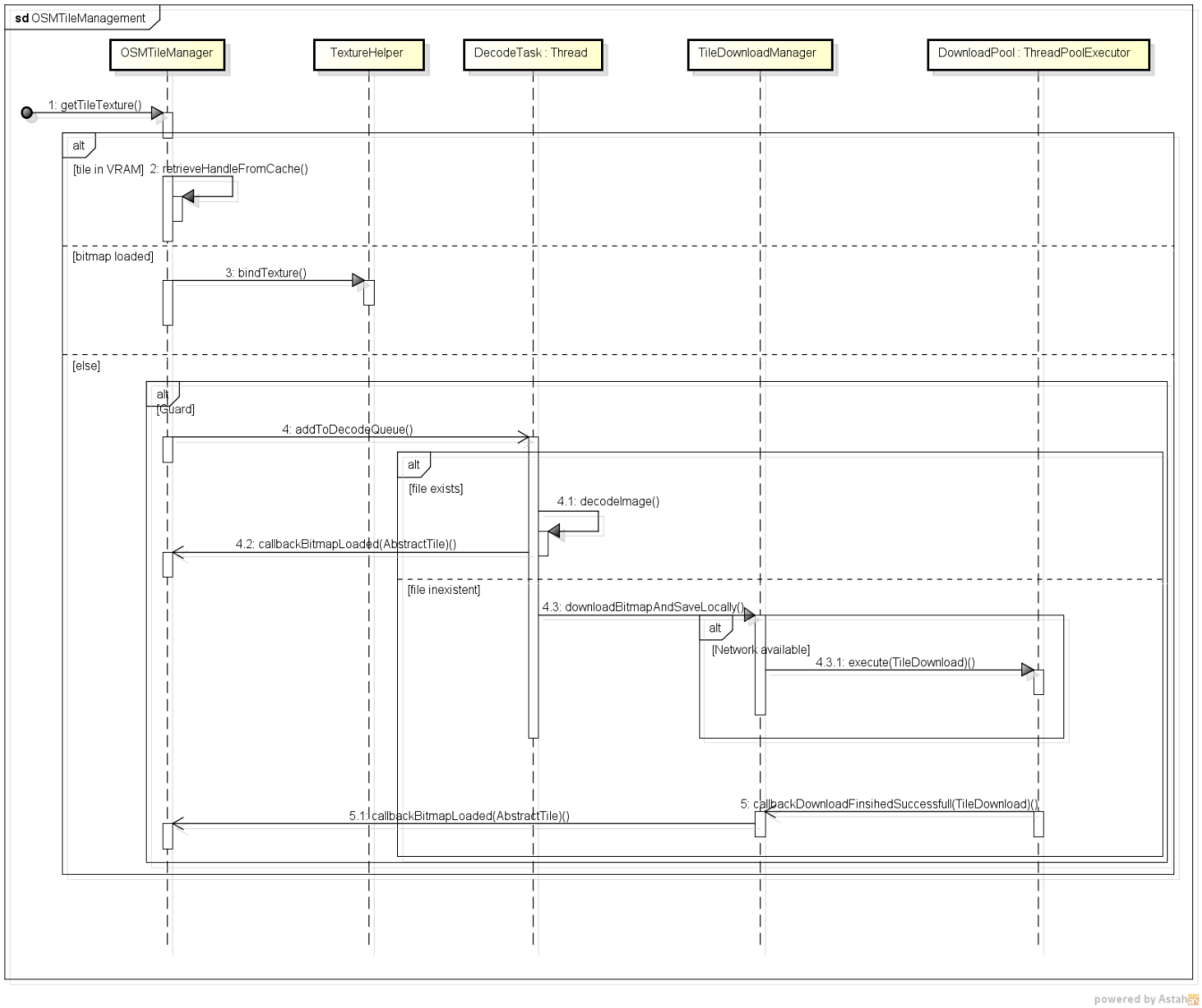


Diagramm 7: Ablauf: Laden von Tiles

OSMTileManager
Über den TileManager können Texturen zum Zeichnen des aktuellen Tiles angefordert werden. Von bereits geladenen Texturen wird ein Handle im eigenen Cache verwaltet.
Um Bitmaps möglichst effektiv zu laden, wurde eine DecodeQueue als Blocking-Queue eingerichtet, welche von einem separaten DecodeTask-Thread abgearbeitet wird. Über geladene Bitmaps wird der OSMTileManager wiederum mit einem asynchronen Callback informiert. Ist ein Tile lokal nicht verfügbar, wird es an den TileDownloadManager weitergeben.

TextureHelper
Stellt Convenience-Methoden zur Verfügung, um beispielsweise Texturen über einen OpenGL-Aufruf ins Video-RAM zu laden.

TileDownloadManager
Lokal noch nicht vorhandene Tiles werden jeweils über den TileDownloadManager angefordert. Sofern eine Internet-Verbindung besteht, werden neue TileDownloader-Tasks in den Download-Pool geschrieben. Aufgrund einer Richtlinie von OpenStreetMap zum Thema Bulk-Downloads wurde eine maximale Anzahl von vier parallelen Downloads festgelegt.

TileDownloader
Ein Download wird jeweils als TileDownload Task realisiert. Erst muss das Bitmap heruntergeladen und anschliessend lokal gespeichert werden.

CacheController
Verwaltet und legt Caches anhand des zur Verfügung stehenden RAM-Bereiches fest.

5.6 OpenGL 2.0

5.6.1 Warum und wo OpenGL benutzt wird

Bei der Benutzung der NeoMap App, stellt sich die Frage „Wo und warum wird überhaupt OpenGL benutzt?“. Auf diese Frage soll in diesem Kapitel eingegangen werden und das Funktions-Prinzip von OpenGL kurz erläutert werden.

OpenGL wird in der App bei den folgenden Vorgängen benutzt:

- Darstellen der OSM-Tiles (d.h. Anzeige der heruntergeladenen Bitmaps)
- 3D-Schrägansicht
- Darstellen der NeoMap Karten
- Darstellen des Icons für die eigene Position
- Darstellen der Koordinaten
- Nominatim-Icon anzeige

Dies alles sind hauptsächlich Operationen im 2D-Bereich und könnten theoretisch auch anders dargestellt werden. Durch OpenGL ist dies jedoch viel performanter und bietet einen grösseren Freiheitsgrad sowie eine gute Erweiterbarkeit.

5.6.2 Geometrie

Die gesamte Darstellung von Objekten und Grafiken unter OpenGL beruht letztlich auf Koordinaten. Koordinaten werden jeweils im 3D-Raum mittels deren X/Y/Z Komponenten definiert. Als Koordinatensystem verwendet OpenGL ES das kartesische System. OpenGL bietet die Möglichkeit, mehrere solcher definierter Punkte (Vertices) zu einer Linie oder einem Dreieck (Triangle) zusammenzusetzten. Soll ein Rechteck gezeichnet werden, wird dieses durch zwei Triangles, bestehend aus drei einzelnen Punkten, konstruiert.

Es besteht die Möglichkeit, welche auch in diesem Projekt genutzt wurde, ganze Reihen von Punkten zu definieren und diese mittels Triangle-Strip-Verfahren zu einer Fläche zu verbinden.

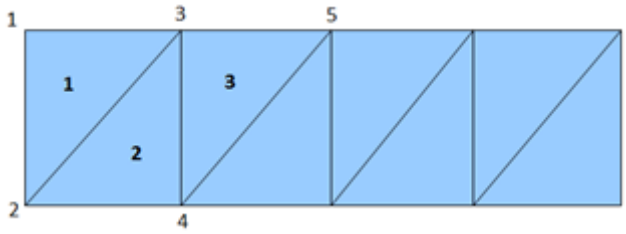


Abbildung 16: Triangle Strip

Mit diesem Verfahren wird beispielsweise jeweils der komplette Raster für die Darstellung der Karte erzeugt. Pro Rechteck wird dann ein OSM-Tile als Textur verwendet.

5.6.3 Matrizen

Matrizen werden verwendet, um Vertices oder Figuren zu verschieben und zu rotieren. Dabei spielt die Reihenfolge der Operationen eine wichtige Rolle, da jeweils der Ursprung des Koordinaten-Systems als Translations- bzw. Rotations-Punkt dient. Da Matrizen-Multiplikationen von rechts nach links durchgeführt werden, müssen Matrizen-Operationen in umgekehrter Reihenfolge festgelegt werden.

Android bietet einige Helper-Klassen, um Matrizen-Operationen durchzuführen. Als Matrix werden jeweils Standard 4x4-Matrizen verwendet.

```
float[] viewMatrix = new float[16];

// set matrix do identity matrix
Matrix.setIdentityM(rotationMatrix, 0);

// 3. translate back to original position
Matrix.translateM(rotationMatrix, 0, centerX,  centerY, 0);
// 2. rotate around origin
Matrix.rotateM(rotationMatrix, 0, -angle, 0f, 0f, 1f);
 // 1. translate center point to origin
Matrix.translateM(rotationMatrix, 0, -centerX,  -centerY, 0);
```

Quellcode 1: OpenGL Operationen für Matrizen

Wie im Quellcode als Kommentar vermerkt wurde, wird die Reihenfolge dieser Matrizen-Operationen in umgekehrter Reihenfolge, aufgrund der rechts-nach-links Multiplikation, durchgeführt.

5.6.4 Geometrie Pipeline

Nachfolgend soll der Ablauf illustriert und erklärt werden, welchen eine Geometrie durchläuft, von der anfänglichen Definition bis zur Darstellung auf dem Bildschirm.

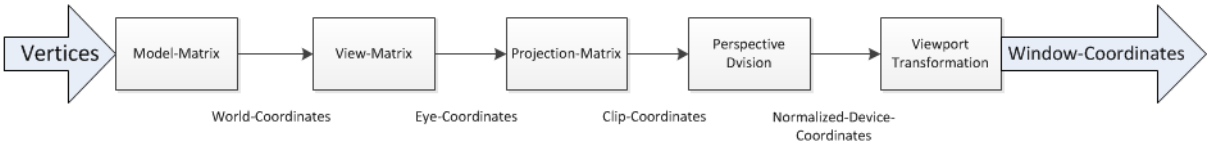


Diagramm 8: Geometrie Pipeline

Eine Figur, welche nicht in Relation zu anderen Figuren definiert wird, wird in den sogenannten **Object-Coordinates** definiert und steht am Anfang der oben abgebildeten Pipeline.
Wird nun eine Figur in Relation zu anderen Figuren transformiert, um die endgültige Platzierung in der virtuellen Welt zu erhalten, handelt es sich um die **World-Coordinates**.

Der nächste Schritt ist, die Welt in Relation zur Kamera zu verschieben. Grundsätzlich wäre es möglich, die Kamera frei im Koordinaten-System zu bewegen. Aufgrund mathematischer Vorteile befindet sich die Kamera jedoch stets am Ursprung des Koordinaten-Systems und alle Objekte werden dazu verschoben. Durch diese Transformation resultieren **Eye-Coordinates**.
Clip-Coordinates erhält man, nachdem eine Projektion durchgeführt und der Viewpoint entsprechend gesetzt wurde. Hierbei entfallen Objekte, welche nicht mehr im sichtbaren Frustum liegen.
Normalized-Device-Coordinates bilden den Bereich des Ausgabemediums in der Koordinatenspanne -1 bis 1 ab. Wird nun die Auflösung des Displays und dessen Verhältnis berücksichtigt, erhält man die finalen **Window-Coordinates**.
Die Model-, View- sowie Projection-Matrix sind massgeblich an dieser Transformation beteiligt und werden an OpenGL in Form einer einzigen, multiplizierten Matrix übergeben.

5.6.5 Grundsätzliche Funktionsweise

OpenGL ist grundlegend als Zustandsautomat aufgebaut. Dies bedeutet, dass sich OpenGL immer in einem definierten Zustand befindet und solange an den internen Zuständen von aussen her nichts geändert wird, werden alle Vertices mit denselben Einstellungen abgearbeitet. Da bei der Grafikprogrammierung meist viele verschiedene Vertices mit denselben Einstellungen bearbeitet werden, ist dies ein ausgesprochen angenehmes Konzept. Einerseits muss der Programmierer für verschiedenen Szenen die Einstellungen nur einmal und nicht bei jedem Funktionsaufruf setzten und andererseits können Vertices von OpenGL somit sehr effektiv verarbeitet werden.
Da es sich bei OpenGL ES um eine hardwarenahe Grafikbibliothek handelt, welche mit C++ realisiert wurde, ist auch kein direkter Zugriff auf deren Objekte möglich. Es wird hierbei immer mit statischen Aufrufen sowie sogenannten Handles gearbeitet. Handles sind nichts anderes als Zeiger, welche von OpenGL generiert und verwaltet werden und genutzt werden können, um OpenGL-Objekte zu referenzieren. Beispielsweise wird ein solches Handle beim Erzeugen eine Textur generiert. Über dieses Handle kann im weiteren Verlauf nun diese Textur angesprochen und in Folgeaufrufen verwendet werden. Um beispielsweise ein zuvor geladenes Bitmap zu löschen, wird OpenGL das entsprechende Handle der Textur beim Aufruf der Delete-Funktion übergeben.

5.6.6 Ablauf beim Darstellen von OSM-Tiles

In diesem Abschnitt wird beschrieben wie der Ablauf zum Anzeigen von OSM-Tiles vor sich geht. Dieser wurde dabei vereinfacht, sollte aber trotzdem eine Übersicht über die Funktionsweise geben. Der Ablauf für die Schritte 1 – 4 ist dabei generisch und gilt somit für alle Layers. Ab Schritt 5 ist dieser speziell auf OSM-Tiles bezogen, kann aber leicht auf NeoMaps sowie die anderen Layers übertragen werden. Auf diese wird deshalb nicht weiter eingegangen.

#	Was	Wo (Klasse: Methode)	Wann (Am Anfang, Einmalig,…)
1	Initial Konfiguration von OpenGL: Version setzen, Haupt Renderer setzen (MapRenderer), Rendering Modus, etc.	MapView: initializeGLSurface	Einmalig bzw. beim Öffnen der App
2	Hintergrundfarbe setzen bzw. Bildschirm aufräumen, Aufruf der Hilfsprogramme zum Erstellen der Shader	MapRenderer: onSurfaceCreated	Beim Erstellen des MapRenderers (d.h. im Normalfall einmal)
2.1	Textur Shader (d.h. Vertex und Fragment-Shader für Texturen) erstellen, dann alle Handles (zu uniform und attributes) auslesen und als Member-Variablen für die weitere Verwendung speichern	TextureShaderProgram: Konstruktor	Beim Erstellen des MapRenderers

2.2	Color Shader (d.h. Vertex und Fragment-Shader für Farben) erstellen, dann alle Handles (zu uniform und attributes) auslesen und als Member-Variablen für die weitere Verwendung speichern	ColorShaderProgram: Konstruktor	Beim Erstellen des MapRenderers
3	Setzen des Viewports (beschreibt das aktuelle Betrachtungsfenster, rechnet Koordinaten zu Geräte spezifischen Koordinaten um). Die Projektionsmatrix durch die Methode „frustumM“ bereitstellen (gibt an was weiter hinten, und somit kleiner und was vorne und somit grösser ist, hier sehr einfach gehalten, da nur bei 3D wirklich wichtig)	MapRenderer: onSurfaceChanged	Direkt nach onSurfaceCreated und immer wenn die Grösse des Surface ändert
4	Mittels rotateM, translateM und multiplyMM wird die Ansicht auf die Mitte zentriert bzw. so verschoben, dass der gewünschte Kartenausschnitt sichtbar ist. Danach wird die draw-Methode des jeweiligen Layers aufgerufen	MapRenderer: onDrawFrame	Jedes Mal wenn ein Frame gezeichnet wird (also bei jeder Verschiebung der Karte etc. mehrmals)
5	Aufruf von prepareGrid des jeweiligen LayerControllers.	LayerRenderer: draw	Jedes Mal wenn ein Frame gezeichnet wird
5.1	Falls notwendig (Zoom-Stufen-Wechsel oder grössere Verschiebung der Karte) wird ein neues Grid generiert	OSMLayerController: generteGridAndActivate	Wird aufgerufen von der prepareGrid-Methode des jeweiligen LayerControllers
5.2	Es werden alle Tiles des aktuellen Grids durchlaufen und die entsprechenden Texturen über den OSMTileManager geladen. Es wird dabei nicht gewartet, bis eine Textur geladen ist, sondern nur Texturen dargestellt, welche bereits bereit sind. Alle anderen Texturen werden asynchron in einem anderen Thread geladen.	LayerRenderer: draw	Jedes Mal wenn ein Frame gezeichnet wird

Tabelle 22: Grobablauf der Darstellung von OSM-Tiles

5.6.7 Shader

Frühere Versionen von OpenGL ES waren jeweils noch nach dem Prinzip der „fixed-function“ Pipeline aufgebaut. Die Pipeline-Architektur von OpenGL ES 2.0 erlaubt es nun, an zwei vordefinierten Stellen Einfluss auf den Rendering-Prozess zu nehmen. Aus diesem Grunde gibt es auch die zwei verschiedenen Shader-Typen: Vertex- und Fragment-Shader. Shader sind nichts anderes als kleine Programme, welche schlussendlich direkt auf der GPU ausgeführt werden. Zur Implementierung solcher Shader steht einem die OpenGL Shading Language (GLSL) zur Verfügung. Von der Struktur her sind Vertex- und Fragment-Shader ähnlich aufgebaut und gleichen auf den ersten Blick einem kurzen C-Programm. Diese, nun variablen Abläufe, waren in früheren Versionen von OpenGL ES fix programmiert und schränkten daher die Möglichkeiten ein.

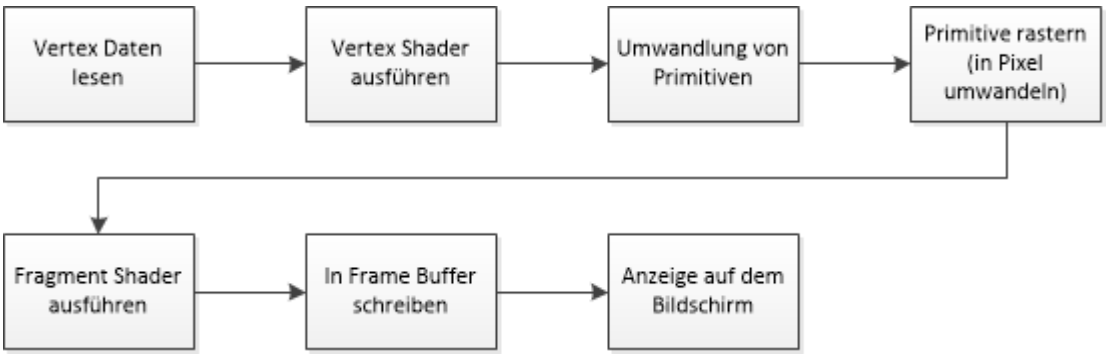


Diagramm 9: OpenGL Pipeline

Der Ablauf der Pipeline ist wie folgt:

1. Zuerst werden die Vertex-Koordinaten ausgelesen. Diese sind dabei definiert und geben die Positionen der Eckpunkte an. Z.B. eines OSMTile
2. Dann wird der Vertex-Shader ausgeführt, dieser berechnet den (auf dem Bildschirm) korrekten Ort für die einzelnen Vertices
3. Als nächstes werden die Primitiven umgewandelt, d.h. im Prinzip, dass die einzelnen Vertices zu geometrischen Figuren (d.h. Dreiecke, Linien oder Punkte) umgewandelt werden.
4. Dann werden diese in Pixel umgewandelt. Dabei findet keine 1:1 Umrechnung statt, es kann sein, dass ein Vertex mehrere Pixel gross ist.
5. Jetzt wird der Fragment-Shader ausgeführt, damit wird die finale Farbe bzw. die Texturen auf die Figuren gelegt.
6. Die ganzen Daten werden jetzt in einen Frame Buffer geschrieben, welcher dann von der Grafikkarte auf dem Bildschirm dargestellt wird.

In den folgenden beiden Unterkapiteln werden die beiden Shader für Texturen, d.h. für die Darstellung von vorgegebenen Grafiken, erklärt. Für reine Farben, d.h. ohne die Verwendung von Texturen, werden andere Shader benötigt. Auf diese wird hier jedoch nicht eingegangen, da diese nach dem gleichen Prinzip funktionieren.

Um den Code der Shader zu verstehen, müssen zuerst noch die wichtigsten Datentypen erläutert werden:

Bezeichnung	Art	Beschreibung
`uniform`	Typ	Dies bedeutet, dass damit deklarierte Variablen zum einen „Read-Only“ sind, d.h. sie können von OpenGL nicht geändert werden und zum anderen können die Werte, welche in diesen Variablen gespeichert sind, während des gesamten Rendering-Prozesses auch von Java nicht mehr verändert werden. Die Variablen müssen am Anfang des Rendering-Prozesses von Java an OpenGL übergeben werden.
`Attribute`	Typ	Dieser Typ bedeutet wiederum, dass die Variablen von Java übergeben werden und auch sie sind „Read-Only“. Jedoch können sich, im Gegensatz zu „uniform“ Variablen, ihre Werte für jeden Vertex ändern. Ausserdem kommt dieser Typ nur in Vertex-Shader vor.

`Varying`	Typ	Variablen dieses Typs werden benutzt um Daten von Vertex-Shader zu Fragment-Shader zu übergeben. Dabei können Vertex-Shader die Variablen lesen und schreiben, Fragment-Shader jedoch nur lesen. Wichtig ist es, dass die Variablen in beiden Shader genau gleich deklariert sind, ansonsten funktioniert die gemeinsame Verwendung nicht.
`Mat2, mat3, mat4`	Datentyp	Matrizen welche entweder 2x2, 3x3 oder 4x4 gross sind und Float-Werte enthalten.
`Float, vec2, vec3, vec4`	Datentyp	Vektoren welche entweder 1, 2,3 oder 4 Float-Werte enthalten.
`Sampler1D, sampler2D, sampler3D`	Datentyp	Enthält 1,2 oder 3 dimensionale Texturen.

Tabelle 23: OpenGL Variablentypen

5.6.7.1 Textur Vertex-Shader

Der Vertex-Shader ist dafür zuständig, dass die definierten Vertices an den richtigen Positionen (d.h. am richtigen Ort auf dem Bildschirm) dargestellt werden.

```
Uniform mat4 uMVPMatrix ;                          //1
attribute vec4 a_position ;                        //2
attribute vec2 a_texCoords ;                       //3
varying vec2 v_texCoords ;                         //4
void main() {
      gl_Position = uMVPMatrix  * a_position ;     //5
      v_texCoords = a_texCoords ;                  //6
}
```

Quellcode 2 : Textur Vertex-Shader

Erklärungen zum Code:

- Auf Linie 1 wird eine 4x4 Model-View-Projektions-Matrix definiert. Diese Matrix wird benötigt, um weiter unten die Position für jeden Vertex zu berechnen. Da sie „uniform“ ist, bleibt sie immer gleich. Diese wird in Java durch den ViewPort, das Frustum sowie eine Rotations-Matrix festgelegt und dem Shader als Parameter übergeben.
- Auf den Linien 2 und 3 finden sich zwei Variablen, welche als „attribute“ deklariert sind. Diese Variablen enthalten die Positionen für einzelne Vertices. Sie legen fest, an welcher Stelle Punkte gezeichnet werden müssen. In diesem Fall geben sie die Positionen der OSMTiles an.
- Auf Linie 4 wird eine Variable deklariert, welche zur gemeinsamen Verwendung von Vertex und Fragment-Shader dient. Es handelt sich hierbei um die finale Position eines Vertex-Punktes im 2D-Raum, abhängig von der Rotation- und Projektion-Matrix.
- Auf Linie 5 wird die oben erwähnte Umwandlung eines Vertices, mittels einer Matrix-Multiplikation, aus dem 3D in den 2D-Raum vorgenommen.
- Auf der Linie 6 werden die Positionen einer Textur, bzw. eines Punktes der Textur, in die auf Linie 4 definierte Variable gespeichert. Dies ist nötig, damit der Fragment-Shader diese später auslesen kann.

5.6.7.2 Textur Fragment-Shader

Der Fragment-Shader kümmert sich um die finale Generierung der Farbe, in diesem Falle unter der Berücksichtigung der vorgegebenen Textur. Somit wird er auch für jedes Fragment genau einmal aufgerufen. Ein Fragment steht hierbei in OpenGL als Synonym für einen Pixel.

```
Precision mediump float;                                    //1
varying vec2 v_texCoords;                                   //2
uniform sampler2D u_texId;                                  //3
void main(){
      gl_FragColor = texture2D(u_texId, v_texCoords);//4
}
```

Quellcode 3: Textur Fragment-Shader

Erklärungen zum Code:

- Auf Linie 1 wird die Präzision der Float-Variablen für diesen Shader bestimmt. „mediump“ bedeutet dabei „mittlere Präzision“. Diese reicht meistens aus und braucht weniger Ressourcen als eine grosse Präzision erfordern würde.
- Die „v_texCoords“ Variable auf Linie 2 enthält die Positionen der Textur welche im Vertex-Shader zugewiesen wurde und von diesem an den Fragment-Shader übergeben wurde.
- Auf Linie 3 wird die Variable u_texId definiert, welche die Position der zurzeit aktiven Textur enthält. Dies erfordert, dass zuvor eine Textur erzeugt und als aktiv gesetzt werden musste.
- Schlussendlich wird auf Linie 4 die Farbe definiert, welche auf dem Bildschirm angezeigt wird. Dazu wird in der übergebenen Textur (u_texId) an den übergebenen Positionen (v_texCoords) die Farbe ausgelesen.

5.6.8 Texturen

Texturen sind im Grunde nichts anderes als Grafiken, welche über die zuvor kreierten Triangeln gelegt werden. Texturen haben ihr eigenes, normalisiertes, Koordinaten System, welches sich immer im Bereich von 0 bis 1 bewegt, unabhängig wie gross das Bild ist. Der Ursprung (0, 0) befindet sich dabei unten links.
Ein Vereinfachter Ablauf, wie ein Bitmap vom Filesystem geladen und später im OpenGL-Context gebunden wird, ist nachfolgenden vorzufinden:

```
bitmap = BitmapFactory.decodeFile(file.getAbsolutePath(),
bitmapFactoryOptions);

// generate a new texture and retrieve a handle
int[] textureHandle = new int[1];
glGenTextures(1, textureHandle, 0);

// Bind to the texture in OpenGL
glBindTexture(GL_TEXTURE_2D, textureHandle[0]);

// Set filtering
glTexParameteri(GLES20.GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE);

// Load the bitmap into the bound texture.
GLUtils.texImage2D(GL_TEXTURE_2D, 0, bitmap, 0);

// Recycle the bitmap, since its data has been loaded into OpenGL.
Bitmap.recycle();

// setting the loaded bitmap as active for further usages
GLES20.glActiveTexture(GLES20.GL_TEXTURE0) ;
GLES20.glBindTexture(GLES20.GL_TEXTURE_2D, handle);
```

Quellcode 4: Laden einer Textur mit OpenGL

Nach diesen Schritten kann die geladene Textur beim Rendern verwendet werden.

5.6.9 Orthogonale Projektion

Bei der orthogonalen Projektion werden die X- und Y-Koordinaten unabhängig der z-Komponente eines Punktes zu Bildschirmkoordinaten umgerechnet. Es findet im Wesentlichen eine einfache Skalierung statt. Durch diese Projektion entsteht ein flaches Bild (2D) der dargestellten Objekte. Die orthogonale Projektion wird im NeoMap App verwendet, um die Koordinaten und das Fadenkreuz als Overlay über die gesamte Karte zu zeichnen.
Damit die Ansicht nicht verzogen wird, gilt es dabei das Höhen-/Breiten-Verhältnis des Bildschirms zu beachten. Die Projektion wird dabei mittels einer Matrix wie folgt definiert.

```
Float[] orthoMatrix = new float[16];
Matrix.orthoM(orthoMatrix, 0, -right, right, -top, top, 0, 1);
```

Quellcode 5: Definition einer orthogonalen Projektions-Matrix

5.6.10 Perspektivische Projektion

Um eine Ansicht zu erzeugen, welche den Anschein macht 3D zu sein, wird eine erweiterte Projektionsmatrix benötigt, welche ein Sichtfeld in Form eines Pyramidenstumpfes (Frustum) erzeugt.

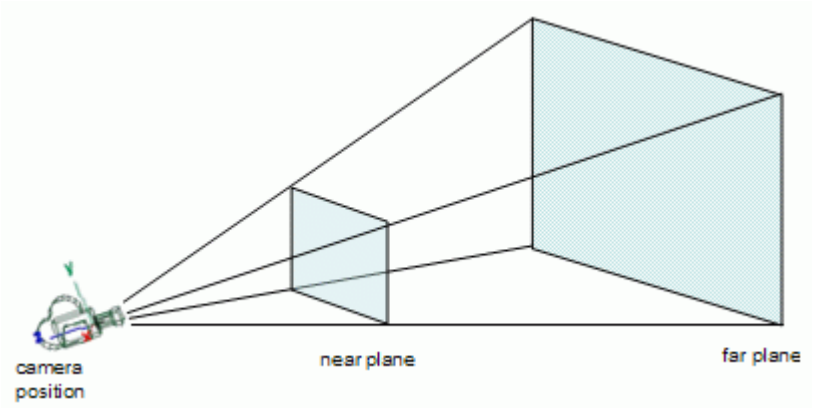


Abbildung 17: Projektions-Frustum

Hierbei werden zwei Flächen definiert, an welchen jeweils der Sichtbereich endet. Alles was sich vor dem Near-Plane sowie alles was sich hinter dem Far-Plane befindet wird abgeschnitten und ist für die Kamera nicht sichtbar.

Der wohl wichtigste Parameter bei der Projektion der Karte ist das Attribut „eyeAltitude“ der Perimeter-Klasse. Bei diesem Attribut handelt es sich um den Wert, um welchen die Karte im Projektions-Frustum auf der Z-Achse nach hinten verschoben wird. Darüber kann gesteuert werden, in welcher Grösse und in welcher Auflösung der dargestellte Kartenausschnitt letztlich auf dem Bildschirm erscheint.
Einen direkten Einfluss auf den Darstellungsbereich hat jedoch auch der Blickwinkel, welcher gesetzt werden kann, wenn die perspektivische Matrix verwendet wird.

Über das zuvor definierte Frustum, kann die Fläche des sichtbaren Bereiches bestimmt werden, für welchen später ein entsprechender Raster generiert und mit Kachelgrafiken gefüllt wird.

```
viewWidthMeter = tan(viewAngle / 2 ) * 2 * eyeAltitude
```

Quellcode 6: Berechnung des sichtbaren Bereiches

Die Projektions-Matrix kann wiederum mittels einer Android-Helper-Methode wie folgt zusammengestellt werden.

```
Float[] projMatrix = new float[16];
Matrix.perspectiveM(projMatrix, 0, angleOfVision, aspectRatio, zNear,
zFar);
```

Quellcode 7: Definition einer perspektivischen Projektions-Matrix

5.7 Eigener Standort

Mittels des von der Android API angebotenen LocationManagers kann die aktuelle Position des Gerätes bestimmt werden. Hierbei stehen die beiden Möglichkeiten der Ortung via Mobilnetz und WLAN oder aber über GPS zur Verfügung. Diese beiden Möglichkeiten unterscheiden sich durch unterschiedliche Genauigkeiten. Da es vorkommen kann, dass ein Benutzer GPS oder gar die gesamte Lokalisierung über die Einstellungen seines Smartphones deaktiviert hat, wird bei einem solchen Fall mittels eines Dialoges darüber informiert und nachgefragt, ob dies aktiviert werden soll.

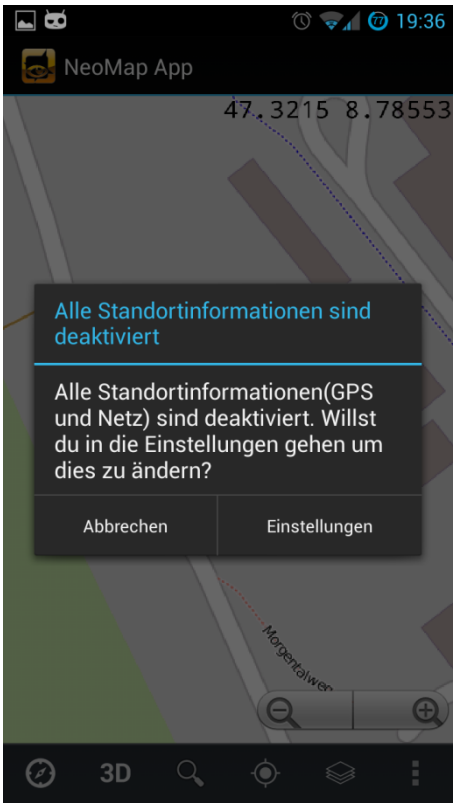


Abbildung 18: Dialog wenn Standortinformationen komplett deaktiviert

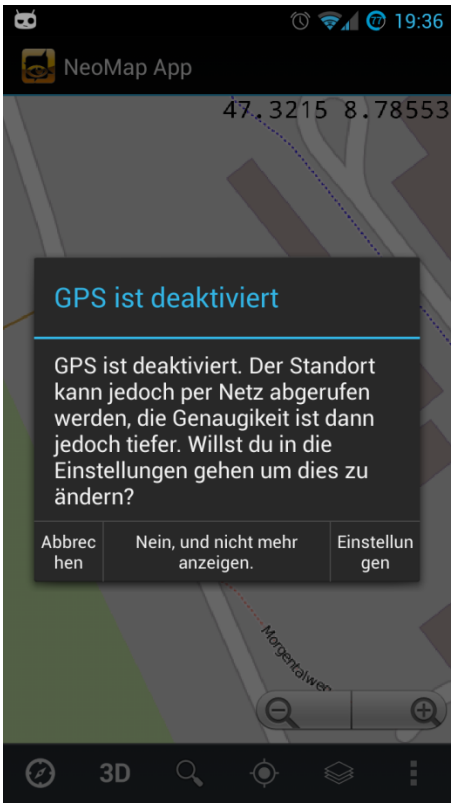


Abbildung 19: Dialog wenn GPS deaktiviert

Der eigene Standort auf der Karte wird mittels folgendem Icon dargestellt:

Abbildung 20: Icon zur Darstellung des eigenen Standorts

5.7.1 Sensor Fusion

Unter dem Begriff Sensor Fusion versteht man das Kombinieren von Werten der verschiedenen Arten von Sensoren, um stabilere und präzisere Resultate zu erhalten.
Die Android API bietet hierfür einen virtuellen Sensor unter dem Namen „TYPE_ROTATION_VECTOR“ an.

Hierbei wird auch das Sensor Fusion Prinzip angewendet und es werden, falls vorhanden, der Geschwindigkeitssensor, der Magnetismus-Sensor und das Gyroskop kombiniert, um bessere Resultate zu erreichen.
Aus diesen aktuellen Positionsinformationen lässt sich nun eine Translations-Matrix erstellen, welche später beim Zeichnen der Karte verwendet werden kann.

5.8 Performance beim Rendering

5.8.1 Allgemein

Da es sich beim Rendering von Bitmap-Grafiken um einen Ressourcen intensiven Vorgang handelt, gilt es dabei einige Dinge zu berücksichtigen, um den Karten-Aufbau sowie die Interaktion des User-Interfaces flüssig zu halten. Dies im Speziellen da die Ressourcen eines mobilen-Gerätes aktuell doch noch etwas eingeschränkt sind.

Da beim Rendern jeweils Aufrufe in einer Schleife gemacht werden, bis die Karte vollständig geladen wurde, gilt es darauf ein besonderes Augenmerk zu legen.
Es gilt möglichst wenige Objekte auf dem Heap zu erstellen, da Objekte, welche auf dem Heap erstellt werden, später auch wieder aufgeräumt werden müssen. Hierbei kommt jeweils der Garbage-Collector zum Zuge. Eine Runde Garbage-Collection unterbricht jeweils alle Threads und benötigt einige Milli-Sekunden. Primitive-Variablen stellen hierbei jedoch kein Problem dar, da sie auf dem Stack angelegt werden. Wenn der Garbage-Collector zu oft arbeiten muss, so hat dies einen spürbaren Einfluss auf die Performance.

Aktuell, Stand 8.04.2013, besitzen die meisten Android-Geräte einen Heap im Bereich von 130 MB bis 200 MB. Jedoch muss dieser für alle offenen Apps ausreichen. Das Android-System verwaltet diesen Speicher und schliesst bei erhöhtem Bedarf des aktuellen Apps andere inaktive Apps im Hintergrund. Ist ein App jedoch zu Ressourcen hungrig, kann es vorkommen, dass es vom System ohne weitere Meldung beendet wird. Dies tritt vor allem auf, wenn nur noch wenig freier RAM-Speicher vorhanden ist.

Um den Rendering-Vorgang zu beschleunigen, sollte alles was nur einmal initialisiert werden muss, zwischengespeichert werden. Der Rendering-Prozess selbst sollte nichts anderes tun, als Vertices zu zeichnen und Texturen zu binden. Somit sollte das Laden von Bitmaps oder auch das Herunterladen von nicht vorhandenen Tiles in separate Threads ausgelagert werden. Dadurch wird der OpenGL-Thread, welcher für das Zeichnen verantwortlich ist, nicht von anderen Aufgaben aufgehalten.

5.8.1.1 Tracing

Mit dem Tool-DDMS, welches beim Eclipse-Plugin Android Development Tools integriert ist, können unter anderem Traces erstellt werden, welche Angaben über die Anzahl Methoden-Aufrufe und deren Zeitverbrauch auswerten.
Um das Laden der Karte noch zu optimieren, wurden jeweils solche Traces erstellt und analysiert. Hierbei wurde darauf geachtet, welche Aufrufe am meisten Zeit benötigen und wie oft diese Aufrufe stattfinden.
Konsumiert eine Funktion viel Zeit, sollte überprüft werden, ob diese Funktion verbessert werden kann (beispielsweise bei Berechnungen durch temporäres Zwischenspeichern) oder ob die Anzahl der Funktionsaufrufe gerechtfertigt ist. Ein Augenmerk kann hierbei auch auf die Anzahl der Aufrufe des Garbage-Collectors geworfen werden.
Des Weiteren kann über das DDMS-Tool auch der aktuell verbrauchte Heap-Speicher ausgelesen oder analysiert werden. Dort ist ersichtlich, welches Objekt aktuell wie viel Memory benötigt.
Um noch eine noch genauere Analyse des Memory-Managements durchzuführen, steht auch die Option „Dump HPROF-File“ zur Verfügung. Dadurch wird ein Abbild des Heaps erstellt.

Dieses File kann beispielsweise mittels Eclipse Memory Analyzer eingelesen und auf Memory-Leaks ausgewertet werden.

5.8.2 Grafiken

In OpenGL wird das RGB-Farbmodell verwendet, welches auf dem Prinzip der Dreifarben-Theorie und dem additiven Farbraum beruht. Dabei gibt es verschiedene Encodings für diesen Farbraum. Gebräuchlich sind hierbei das 24-Bit sowie das 16-Bit Integer RGB Encoding.

5.8.2.1 24-bit Encoding

Hierbei werden 24-Bit zur Encodierung jedes Pixels verwendet bzw. 8-Bit pro Farbkanal. Es besteht auch die Möglichkeit die Intensität (Transparenz) in einem separaten Kanal festzulegen. Somit werden Farben sowie die Intensität pro Kanal mit Werten zwischen 0 bis 255 angegeben. Insgesamt kann ein Pixel mit 2^24 Bit encodiert werden. Somit stehen 16‘777‘216 Farben zur Verfügung. Diese Art von Encoding ist unter dem Namen RGB888 bzw. RGB8888 (mit Intensität) bekannt. In diesem Format benötigt eine geladene Textur also 3 bzw. 4 Bytes pro Pixel.

5.8.2.2 16-bit Encoding

Es besteht auch die Möglichkeit Farben mittels 16-Bit zu encodieren. Dabei werden 5-Bit für den roten Kanal, 6-Bit für den grünen Kanal und wiederum 5-Bit für den blauen Kanal verwendet. Grün wird hierbei mit 6-Bit encodiert weil das Auge Grünwerte besser differenzieren kann als blau und rot. Insgesamt wird mit diesem Encoding ein Pixel in 2^16 (65‘536) Farben unterteilt. Dieses Format ist auch unter dem Namen RGB565 geläufig.

5.8.2.3 Vergleich bezüglich Datenmenge

Wird von einem Bild mit der Auflösung 1024 * 1024 Pixel ausgegangen, rechnet sich die benötigt Datenmenge eines Bildes wie folgt:

Encoding	W * H * Bytes per Pixel	Datenmenge
RGB888	1024 * 1024 * 3 Bytes	3 MB
RGB8888	1024 * 1024 * 4 Bytes	4 MB
RGB565	1024 * 1024 * 2 Bytes	2 MB

Tabelle 24: Vergleich der Datenmenge verschiedener Grafik-Encodings

5.8.3 Grafik-Formate und Kompressions-Verfahren

Um nun Bilder möglichst effektiv zu speichern, wurden verschiedene Grafik-Formate und Kompressions-Verfahren erfunden. Der grösste Unterschied besteht dabei zwischen Verlust-freier und Verlust-behafteter Kompression. Die bekanntesten Vertreter je einer dieser Kategorie sind wohl PNG und JPEG.
Wird ein Bild geladen, muss es erst dekomprimiert und in seiner vollen Grösse, unabhängig des gespeicherten Formates respektive der zuvor verwendeten Komprimierung, ins RAM geladen werden.

5.8.3.1 Alpha-Blending / Transparenz

Transparenz kann, wie bereits zuvor erwähnt, über einen separaten Kanal im RGB8888 Encoding gespeichert werden. Hierbei reicht der Wert wieder von 0 (transparent) bis 255 (undurchsichtig).

5.8.4 Ressourcen

Um nun Texturen möglich effizient und schnell zu laden, sollte jeweils ein angemessenes Format verwendet werden. Im Falle dass viele Grafiken geladen und mittels der OpenGL API ins VRAM (Video RAM) geladen werden, spielt nicht nur Speicherverbrauch selbst eine Rolle, es kommt auch noch die Daten-Durchsatzrate ins Spiel.

Um dies ein wenig genauer aufzuzeigen, wurden Messungen auf verschiedenen Geräten mit verschiedenen Grafiken und Encodings durchgeführt.
Dabei wurde eine PNG-Grafik mit der Dimension 256x256 Pixeln verwendet und jeweils 500x geladen. Um verlässliche Resultate zu erhalten, wurde dies jeweils fünfmal wiederholt und der Mittelwert als Messwert verwendet.

Grafik-Encoding	Surface-Encoding	Nexus 7	Google Nexus
RGB8888	RGB8888	9299 MS	11985 MS
RGB8888	RGB565	9230 MS	11800 MS
RGB565	RGB8888	**5826 MS**	**7940 MS**
RGB565	RGB565	**5847 MS**	**7854 MS**

Tabelle 25: Durchsatz-Messung beim Laden einer Grafik

5.8.4.1 Laden einer Grafik

Eine Grafik wird jeweils erst als Bitmap in den RAM Bereich geladen. Mittels der OpenGL API kann das Bitmap dann ins VRAM übertragen werden. Danach sollte das Bitmap im RAM Bereich unbedingt wieder gelöscht werden und nur noch das sogenannte Handle, ein einfacher int-Pointer, der Grafik im VRAM für die weitere Verwendung zwischengespeichert werden. Wird eine Grafik gar nicht mehr benötigt, sollte sie, mittels der OpenGL API, auch aus dem VRAM gelöscht werden.
Bereits beim Laden der Grafik kann nun das zu verwendende RGB-Encoding angegeben werden.

5.8.5 RAM / VRAM Size

Da unter Android RAM und VRAM denselben Speicher-Bereich teilen, ist es leider nicht möglich, die Grösse des zur Verfügung stehenden VRAM‘s zu ermitteln. Der Bereich des VRAM‘s wird laufend während der Laufzeit angepasst und bei Bedarf solange vergrössert, bis kein Speicher mehr zur Verfügung steht. Dies äusserte sich jeweils dadurch, dass der allozierende Prozess vom System ohne jegliche Warnung oder Fehlermeldung geschlossen wurde.
Leider gibt es auch keine saubere Android-API Methode um herauszulesen, wie viel RAM insgesamt zur Verfügung steht. Es kann jedoch ermittelt werden, wie viel RAM momentan frei ist und dem App somit zur Verfügung steht. Wird mehr RAM benötigt, beginnt die Dalvik-VM inaktive Apps und Prozesse zu schliessen, um mehr Ressourcen freizugeben. Es ist dabei jedoch nicht garantiert, dass noch mehr Ressourcen verfügbar werden.
Um dieses Verhalten genauer zu analysieren, wurden verschiedene Tests auf unterschiedlichen Geräten durchgeführt, um das Limit des VRAMs in Relation zum insgesamt verfügbaren Speicher zu erörtern.
Als Grafik wurde ein Bild im PNG-Format mit den Abmessungen 256 x 256 Pixel mittels dem RGB565 Encoding ins VRAM geladen. Dies wurde solange wiederholt, bis die NeoMap App vom System beendet wurde.

Testgerät	Free Memory (zu Beginn des Tests)	Maximale Anzahl geladener Grafiken	Berechneter Memory Verbrauch
Galaxy S3	375 MB	2042	510.50 MB
Asus Transformer	494 MB	3103	775.75 MB
Google Nexus	135 MB	851	212.75 MB
Nexus 7	587 MB	3008	752.00 MB

Tabelle 26: Verfügbarer VRAM

Aufgrund dieser Test-Ergebnisse wurde entschieden, dass für das Caching von Grafiken jeweils der zu Beginn verfügbare Speicher problemlos komplett ausgenutzt werden kann. Wird jedoch mehr als dieser Speicher verwendet, kann es sein, dass bei gleichzeitiger Verwendung von mehreren Applikationen, welche auch viel Speicher benötigen, Engpässe auftreten.

5.8.6 Caching

Um das Caching von Tiles zu ermöglichen, wurde die Klasse TileLRUCache eingeführt. Bei der Initialisierung kann eine vorgegebene maximale Kapazität gesetzt werden. Wird diese erreicht, werden automatisch die Tiles, welche am längsten im Cache sind wieder aus dem Cache gelöscht. Intern arbeitet der Cache mit einer LinkedHashMap, welche nach dem Last-Access-Verfahren geordnet ist.
Performance- und Memory-mässig ergibt sich bei der Benutzung der LinkedHashMap im Vergleich zu einer normalen HashMap nur ein kleiner Nachteil durch das zusätzliche Führen einer LinkedList, um die Zugriffs-Ordnung beizubehalten. Operationen wie „put“ oder“ get“ werden jedoch weiterhin mit der Laufzeit von O(n) ausgeführt.
Der Cache wird nur für OSMTiles und NeoMaps erstellt. Diese beiden Layer sind die einzigen, welche eine grosse Anzahl an verschiedenen Grafiken haben und somit den Einsatz eines intelligenten Caches erfordern. Andere Layer, wie z.B. Notizen, haben jeweils nur ein bzw. zwei Symbole, welche immer wieder verwendet und somit auch nur einmal am Anfang geladen werden.
Der Cache wird initialisiert, sobald der TileManager der beiden Layer initialisiert wird. Dies ist gleich nach dem Starten des Apps der Fall. Die Cache-Grösse ist abhängig davon, wie viel RAM aktuell auf dem Gerät zur Verfügung steht und wird mittels des nachfolgenden Code-Fragments ausgelesen.

```
ActivityManager.MemoryInfo mInfo = new ActivityManager.MemoryInfo();
actvityManager.getMemoryInfo(mInfo);
mInfo.availMem / 1024 / 1024;
```

Quellcode 8: Berechnung des freien RAM in MB

Diese Grösse wird dann zwischen OSM und NeoMaps im Verhältnis 1:3 aufgeteilt.

5.8.7 Ladestrategie der Tiles

Vor dem Refactoring der NeoMap App wurden die Tiles vom oberen linken Rand der Karte, Tile für Tile bzw. Reihe für Reihe, bis nach unten rechts geladen. Folgende Abbildungen illustrieren dies.

Abbildung 21: Ladestrategie vor Refactoring

Abbildung 22: Ladestrategie vor Refactoring (nummeriert)

Dies ist sehr unvorteilhaft, denn der Benutzer möchte meist als erstes die Mitte der Karte sehen, da sich dort meist das von Ihm gesuchte befindet.

Die Ladestrategie wurde deshalb so geändert, dass Tiles jeweils von der Mitte aus, abwechselnd links und rechts, nach aussen geladen werden. Realisiert wurde das mit einem eigenen Iterator (CenterOutIterator).

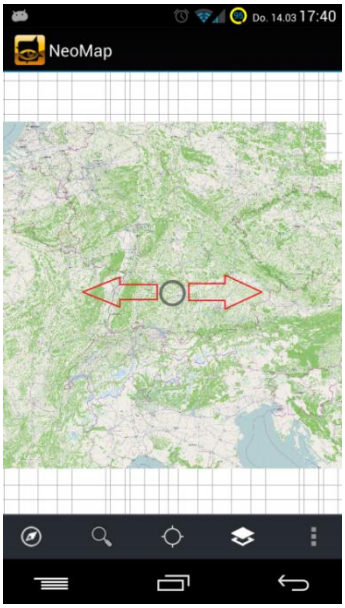


Abbildung 23: Ladestrategie nach Refactoring

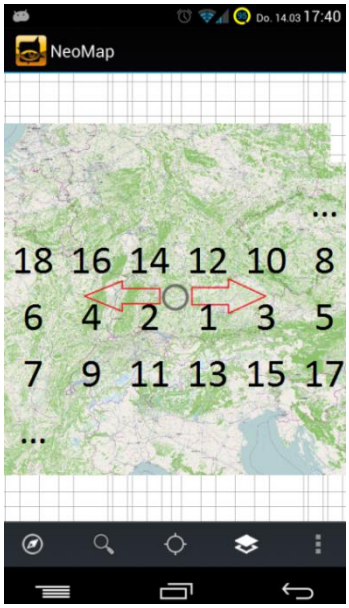


Abbildung 24: Ladestrategie nach Refactoring (nummeriert)

6 Karten-Schrägansicht (3D)

6.1 Konzept

Zur einfacheren Orientierung entstand die Idee, eine Schrägansicht der Karte einzubauen. Um die Orientierung möglichst einfach zu gestalten, soll die Karte dabei stets nach Norden, auch wenn das Gerät sich in einer Drehbewegung befindet, ausgerichtet werden. Somit wird ermöglich, dass der Benutzer auf der Karte denselben Bereich sieht, welchen er auch real in seinem Blickfeld hat. Durch Kippen des Gerätes soll der Grad der Schrägansicht gesteuert werden können. Ein weiterer wichtiger Punkt ist dabei, dass die aktuelle Position des Benutzers erörtert werden kann, damit genau der Kartenausschnitt anzuzeigen werden kann, welchen der Benutzer real vor sich sieht. Da auf der Karte auch Strassennamen angezeigt werden, ist eine Orientierungshilfe somit bewerkstelligt.

Abbildung 25: Schrägansicht

6.2 Umsetzung

6.2.1 Sichtbarer Kartenbereich

In der 3D-Schrägansicht wird der sichtbare Bereich der Karte nicht mehr aufgrund des Sichtwinkels und des Abstandes der Kamera zur Karte berechnet, da dies kaum brauchbare Resultate liefert.
Die Berechnung der BoundingBox wurde angepasst und beruht nun auf folgendem Prinzip.

Minimale Sichtweite auf höchster Detail-Stufe: **1400** Meter
Zusätzliche Sichtweite pro weitere Zoom-Stufe: **1800** Meter

Auf der untersten Zoom-Stufe, ganz nahe am Gelände, wird somit ein Kartenausschnitt in der Grösse von 1400x1400 Meter dargestellt. Zusätzlich zu dieser Sichtweite wird ein Horizont in Form von halbtransparenten Tiles am Rande des Kartenausschnittes dargestellt.
Auf weiteren Zoom-Stufen wird die Sichtweite jeweils um 1800 Meter erweitert.

6.2.2 Kartenausrichtung

Der in mittlerweile jedem Android-Gerät eingebaute Magnetismus-Sensor liefert die Abweichung der Y-Achse zur Nord-Achse und somit den Wert, um welchen die Karte gedreht werden muss, um die Nord-Ausrichtung zu ermöglichen. Dessen Verwendung ermöglicht eine optimale Ausrichtung der Karte.

6.2.3 Aktuelle Lage des Gerätes

Um dem Benutzer die Möglichkeit zu geben, mittels Kippen den Grad der Schrägansicht zu steuern, muss die aktuelle Lage des Gerätes bestimmt werden. Auch hier werden von Android-Geräten bereits Sensoren angeboten, welche diese Werte liefern. Es kann damit ermittelt werden, wie das Gerät aktuell im dreidimensionalen Koordinaten-System positioniert ist.

6.2.4 Projektion

Die zuvor beschriebenen Werte müssen folglich bei der Projektion der Karte mitberücksichtigt werden, um schlussendlich die gewünschte Darstellung zu erhalten. Bei der Projektion handelt es sich um eine perspektivische Projektion, welche im Kapitel 5.6 OpenGL 2.0 genauer erläutert ist. Kurz gesagt handelt es sich bei der perspektivischen Projektion um eine Darstellung, bei welcher Objekte proportional zu Ihrem Abstand zur Kamera dargestellt werden und dadurch ein Fluchtpunkt entsteht.

6.2.5 Drehen der Icons zur Kamera

Da es sich bei den Icons um 2D-Texturen handelt, werden diese jeweils neu ausgerichtet. Ohne dies würden die Icons je nach Dreh- bzw. Blickrichtung des Benutzers für ihn unsichtbar. Als Abweichungswert wurden hierbei 5° gewählt. Sobald sich der Benutzer um mehr als diesen Wert gedreht hat, werden die Icons neu ausgerichtet, um gut ersichtlich zu bleiben.

6.2.6 Skybox

Um den Effekt eines Horizonts zu erzeugen, wird eine sogenannte Skybox verwendet. Bei dieser handelt es sich um eine Box, welche die gesamte Karte umschliesst. Die Karte stellt dabei den Boden dieser Box dar. Die Wände und die Decke können dann mit Texturen belegt werden, um einen realen Effekt zu erzeugen. Der äusserste Bereich der Landkarte ist jeweils halbtransparent dargestellt, um den Effekt eines entfernten Horizonts zu imitieren.

6.2.7 Antippen von Objekten

Bei der Umsetzung des Antippens von Objekten im 3D-Modus, hat sich als Problem ergeben, dass sich Koordinaten nicht mehr über das Verhältnis Meter/Pixel berechnen lassen, da es sich hierbei um eine perspektivische Ansicht handelt und somit keine Modell getreue Ansicht mehr dargestellt wird. Die perspektivische Projektion hat die Eigenschaft, dass ein Fluchtpunkt existiert und entfernte Objekte kleiner erscheinen als Objekte, die sich nahe an der Kamera befinden. Diese Darstellungsform ist dem Mensch bereits bekannt, da es genau wiederspiegelt, was der Mensch durch sein Auge sieht. Um dies zu ermöglichen, findet die sogenannte perspektivische Division statt. Dieser Vorgang wird durch OpenGL vorgenommen, sobald das Modell der perspektivischen Projektion verwendet wird.

Es existiert keine Funktion um zu ermitteln, welchen Punkt ein Benutzer auf dem Bildschirm angetippt hat. Dies ist aber nötig, um direkt dessen ursprüngliche World-Coordinates zu erhalten. Deswegen muss folgender Ablauf stattfinden.

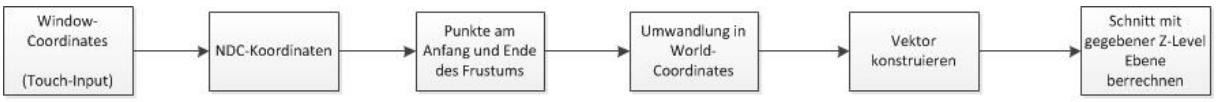


Diagramm 10: Ablauf beim Antippen eines Punktes im 3D-Modus

Android stellt eine API-Funktion bereit, mit welcher sich ein Touch-Input erkennen lässt. Diese Koordinaten (Pixel), müssen zu Normalized Device Coordinates umgewandelt werden.
Danach werden zwei Punkte gebildet, einer davon am Anfang des Viewing-Frustums (z = -1) und einer am Ende des Viewing-Frustums (z = 1). Diese Punkte können mittels der inversen Projektions-Matrix in die World-Coordinates zurück gerechnet werden. Dabei erhält man einen vierdimensionalen Vektor, wobei die letzte Komponente (w) den perspektivischen Aspekt darstellt. Um nun wiederum einen Vektor im dreidimensionalen Raum zu erhalten, muss zusätzlich eine perspektivische Division durchgeführt werden.

Mit diesen zwei Punkten in Welt-Koordinaten, kann nun ein Vektor konstruiert und berechnet werden, auf welcher Ebene (gegebener Z-Koordinate) er welche X- und Y-Komponenten hat. Dies sind zugleich die End-Koordinaten, welche von Interesse sind.

Dies alles ist notwendig, da bei der perspektivischen Division und dem von OpenGL gegebenen Depth-Buffer die Präzision der Z-Komponente verloren geht. Der Depth-Buffer hat eine maximale Grösse von 16 Bit (65‘536). Unter der Verwendung der Normalized Device Coordinates werden diese Werte nun zwischen -1 und 1 gemappt, wobei die Wahl der Clipping-Planes (vor allem z-Near) einen grossen Einfluss hat.

7 Allgemeine Karten-Overlays

7.1 Klassen

ManagableItem
Die abstrakte Klasse ManagableItem ist eine Basis-Klasse für Objekte, welche verwaltet werden können. Verwaltet heisst hierbei, dass sie in der Datenbank gespeichert werden und ein dazugehöriges Verwaltungsfragment haben. Die abstrakte Klasse besitzt die Attribute und Methoden, welche all diese Objekte gemeinsam haben, wie z.B. getName(), getDatabase(), isPublic() und isEnabled(). Die Klasse ist hauptsächlich dafür da, damit die Verwaltungsklasse nicht zu generisch gemacht werden muss, sondern einige Basisoperationen direkt eingebaut werden können.
Beispiele für ManagableItems sind z.B. NeoMaps und Notizen.

Visible Target
Diese Klasse ist im Prinzip nur ein Wrapper, welcher ein ManagableItem, das Delta zur aktuellen Tap-Position (also dort wo der Benutzer auf den Bildschirm gedrückt hat), sowie den LayerType, in welchem sich das ManagableItem befindet, beinhaltet. Die Klasse wird für die onTap/onDoubleTap Operationen auf der MapView benötigt (siehe „Antippbarkeit von Objekten (onTap / onDoubleTap / VisibleTarget)“).

VisibleTargetComparator
Der Komparator ist eine innere Klasse des VisibleTargets und dient dazu, Listen mit VisibleTargets zu sortieren. Dabei wird zum einen nach gewissen Prioritäten sortiert (z.B. LayerType Notiz ist immer vor LayerType NeoMap) und zum anderen nach der Grösse des Tap-Deltas. Je kleiner das Delta ist, desto weiter oben befindet sich ein VisibleTarget in der Liste.

ManagementFragment
Die abstrakte Klasse ManagementFragment ist eine Basis-Klasse für Klassen, welche ein konkretes ManagableItem verwalten. Bei Kindklassen muss ein Klasse vom Typ ManagableItem angegeben werden, da die Klasse mit Generics arbeitet (<T extends ManagableItem>).Die Klasse implementiert die Basisfunktionalität, sowie das Basis-GUI für das Verwalten von ManagableItems. Sie hat dabei diverse abstrakte Methoden, welche überschrieben werden müssen damit Datenbankoperationen für bestimmte Aktionen durchgeführt werden können, so z.B. getAllItems() und deleteItem(). Zudem benutzt diese Klasse eine Contextual Actionbar, mit welcher Operationen für mehrere Objekte gleichzeitig durchgeführt werden können. Siehe Kapitel „Contextual Action Mode / Contextual Action Bar“.
Eine sehr wichtige abstrakte Methode ist setInfoDialogColumns(T itemAtPosition), bei dieser Methode kann angegeben werden, welche Spalten und Inhalte beim Info Dialog angezeigt werden können. Als Beispiel sind hier die beiden unterschiedlichen Dialoge des NoteManagementFragment, sowie dem NeoMapManagementFragment abgebildet:

Abbildung 26: Management Info Dialog einer NeoMap

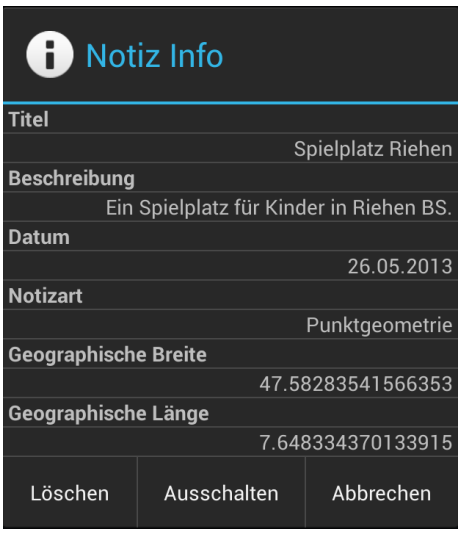


Abbildung 27: Management Info Dialog einer Notiz

Weiter ist die abstrakte Methode „setManagementAdapter“ von hoher Relevanz.

ManagementAdapter
Diese Klasse ist abstrakt und wird im ManagementFragment zum Darstellen der Liste von ManagableItems, sowie zum Suchen und Sortieren von diesen verwendet. Sie ist im Prinzip das Haupt-GUI in den Managementklassen.
Subklassen müssen zum einen zwei Methoden überschreiben, welche angeben was in der Liste für Spalten angezeigt werden sollen und eine Methode welche angibt, was für Felder bei Eingabe eines Suchtextes durchsucht werden sollen. Gleich wie das ManagementFragment arbeitet diese Klasse mit Generics (<T extends ManagableItem>). Als Beispiel sind hier zwei Implementationen des Adapters in Form des NeoMapManagementAdapters sowie des NoteManagementAdapters dargestellt:

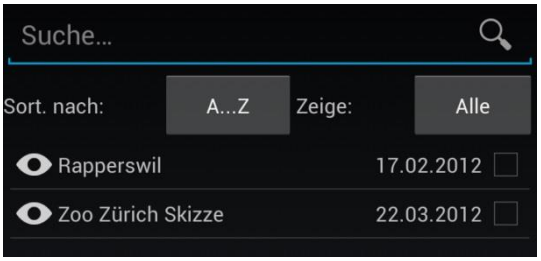


Abbildung 28: ManagementAdapter der NeoMaps

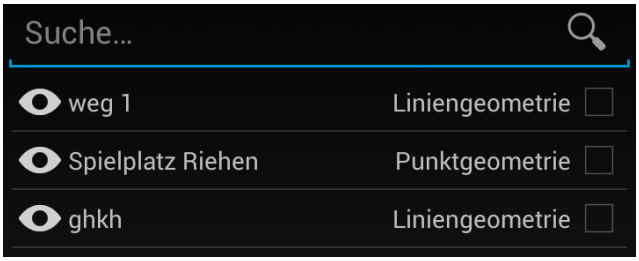


Abbildung 29: ManagementAdapter der Notizen

7.2 Contextual Action Mode / Contextual Action Bar

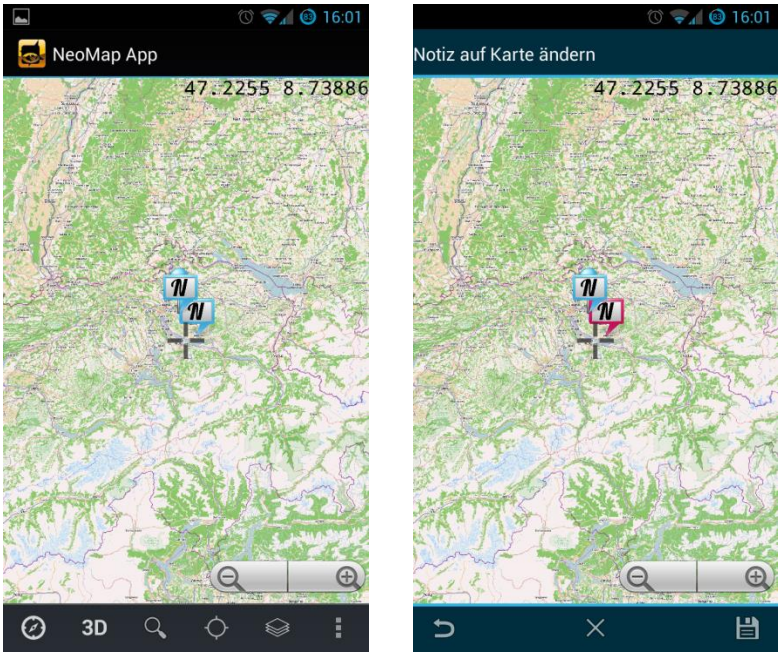
7.2.1 Allgemeines

Damit eine Operation (z.B. Löschen) für mehrere Elemente gleichzeitig durchgeführt werden kann, existiert ab Android 4.0 ein Modus namens „Contextual Action Mode“. Dieser erlaubt dem Benutzer in einen speziellen Modus zu wechseln, in welchem er gleichzeitig mehrere Elemente auswählen und bearbeiten kann. Dabei wird die normale Action Bar durch eine „Contextual Action Bar (CAB)“ ersetzt, diese enthält verschiedene „Actions“ (=Operationen) die verwendet werden können. Sie hebt sich auch farblich von der normalen Action Bar ab. Google selbst verwendet den Contextual Action Mode zum Beispiel bei der Gmail-Applikation.

Der Contextual Action Mode kommt in der NeoMap App an zwei Orten vor.

7.2.2 Kartenänderungsmodus

Dies ist eigentlich ein Missbrauch des Modus, da er an dieser Stelle nicht für das Bearbeiten von mehreren Elementen gebraucht wird, sondern um Notizpunkte auf der Karte zu verschieben bzw. hinzuzufügen. Der Modus eignet sich hier jedoch perfekt um die nötigen Funktionen einfach zur Verfügung stellen zu können und er hebt sich stark vom normalen Kartenmodus ab, dadurch sieht der Benutzer sofort, ob er sich im Änderungsmodus befindet oder nicht.
Ausgelöst wird er dabei entweder durch das LongPressMenu oder durch das Bearbeiten von einer Notiz. Siehe dazu Kapitel „Notizen“.
Der Modus kann nur verlassen werden, indem entweder das Speicherungssymbol betätigt oder aber der Modus durch das X-Symbol abgebrochen (und die Änderungen somit nicht gespeichert werden) wird.



Normaler Kartenmodus Kartenänderungsmodus

Abbildung 30: Contextual Action Bar im Kartenänderungsmodus

7.2.3 Notiz-/NeoMapverwaltung

Hier wird der Modus für das Bearbeiten von mehreren Elementen bzw. ManagableItems eingesetzt – also genau für den Zweck, wofür er auch gedacht ist. Er wird gestartet in dem der Benutzer eine beliebige Checkbox in der Notiz-/NeoMapverwaltung antippt. Danach kann der Benutzer beliebig viele weitere Elemente auswählen und dann eine Operation für all diese Elemente ausführen. Der Modus kann jederzeit verlassen werden, indem der Haken oben links angetippt wird.

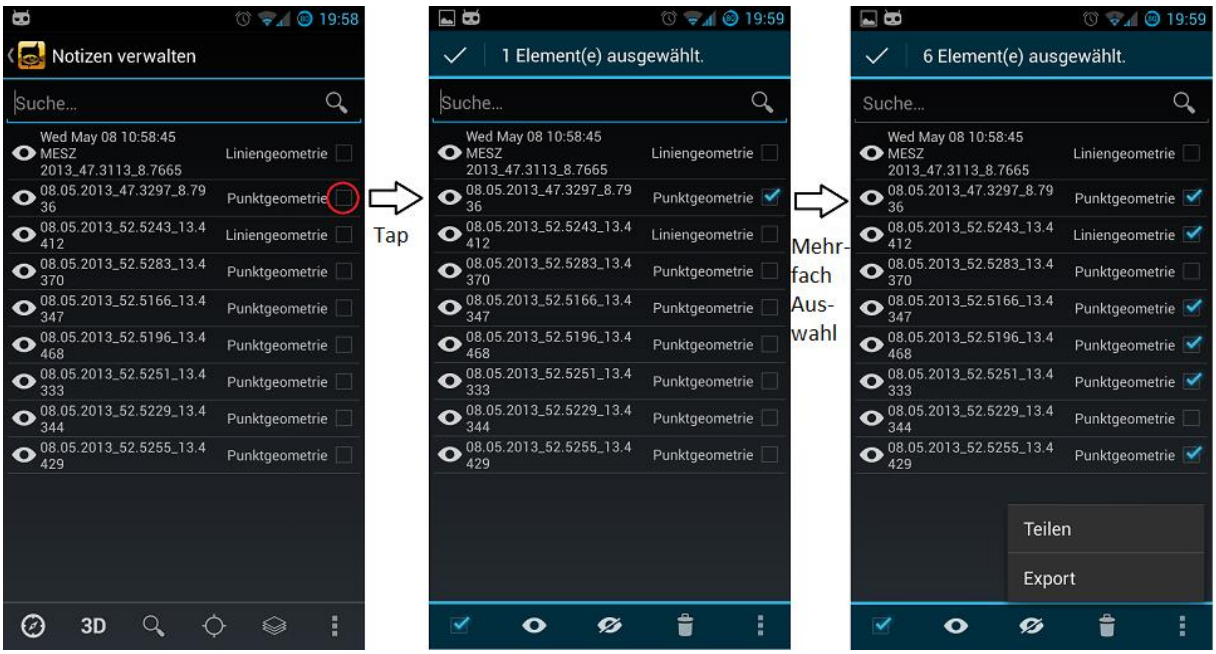


Abbildung 31: Contextual Action Bar im ManagementFragment

Der Contextual Action Bar Modus unterscheidet sich nur in einem Punkt von Notizen zu NeoMaps. Bei Notizen existieren im Gegensatz zu NeoMaps noch die beiden Funktionen „Export“ und „Teilen“, welche beide zum Export von Notizen dienen (siehe Kapitel „Export“).

Dem Benutzer stehen folgende Operationen zur Verfügung:

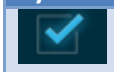
Symbol	Name	Beschreibung
	Alle markieren	Mit einem Klick auf dieses Icon werden alle Elemente markiert bzw. selektiert. Dies ist eine „Convenience-Funktion“ damit der Benutzer nicht alle selbst anklicken muss.
	Auswahl sichtbar	Dadurch werden alle selektierten Elemente auf der Karte sichtbar.
	Auswahl unsichtbar	Dadurch werden alle selektierten Elemente auf der Karte unsichtbar.
	Auswahl löschen	Dadurch werden alle selektierten Elemente gelöscht. Der Benutzer muss dies zuerst bestätigen.
-	Teilen (nur in Notizverwaltung)	Dadurch wird ein ShareIntent an alle Applikationen gesendet, welche mit GPX-Dateien umgehen können. Der Benutzer kann über ein Menü entscheiden, welche Applikation die ausgewählten Elemente empfangen bzw. weiterverarbeiten sollen. So können alle markierten Elemente zum Beispiel direkt per Mail versandt oder in die DropBox hochgeladen werden.
-	Export (nur in Notizverwaltung)	Die markierten Elemente werden auf die SD-Karte bzw. das schreibbare externe Verzeichnis des Smartphones exportiert, so dass der Benutzer diese später z.B. auf den PC kopieren kann.

Tabelle 27: Verschiedene Aktionen der Contextual Action Bar

7.3 Anzeige der aktuellen Koordinaten auf der Map

Mit der Version von M. Wolski verschwand die Anzeige der Koordinaten der Center-Position, diese befand sich vorher in der „Action Bar“ (UI-Teil). Diese Informationen sind jedoch nützlich und sollen daher wieder eingeführt werden.

Da die aktuellen Koordinaten sich bereits bei der kleinsten Verschiebung der Karte ändern, muss also einerseits stets die Karte neu gezeichnet und andererseits neue Koordinaten angezeigt werden. Werden Koordinaten nun in einem normalen UI-Element angezeigt, hat dies zur Folge, dass ständig ein Context-Switch zwischen dem OpenGL-Rendering-Thread und dem UI-Thread stattfindet, da sich immer beide Elemente – Karte und Koordinaten – zusammen ändern. Um dieser Performance-Einbusse zu entgehen, wurde entschieden, die Koordinaten direkt mittels OpenGL als Overlay auf die Karte zu zeichnen.

Abbildung 32: NeoMap App mit Koordinaten

Doch auch bei dieser Methode muss auf die Performance geachtet werden. OpenGL kann keine Schrift darstellen, d.h. es müssen Bitmaps verwendet werden. Würde jetzt für jede Koordinate bzw. jede „Stelle“ ein eigenes Bild benutzt werden, so müssten jedes Mal 14 Bitmaps geladen und dargestellt werden. Dies ist natürlich schlecht. Aus diesem Grund wird mit einem einzigen Bitmap gearbeitet, welches alle Nummern, sowie einige Sonderzeichen enthält. Mittels eines Offsets wird beim Bild dann eine bestimmte Stelle „herausgeschnitten“ und nur diese gezeichnet. Es wird das gleiche Bild also einfach 14 Mal wieder verwendet. Mit dieser Methode werden viele Ressourcen gespart. Einerseits braucht man nur ein Bild und andererseits lädt man in OpenGL das Bild nur ein einziges Mal und verwendet es dann einfach wieder.

In der folgenden Tabelle ist das Bild mit den Offsets für die einzelnen Positionen dargestellt.

Bild	0	1	2	3	4	5	6	7	8	9	.	-	'	"
Offset	0	1	2	3	4	5	6	7	8	9	10	11	12	13

Tabelle 28: Koordinatenbild mit Offsets

Die Zeichen an der Stelle 12 und 13 werden momentan nicht benutzt. Wurden aber trotzdem eingeführt, falls die Koordinaten Darstellung mal geändert werden sollte.
Um den Offset einfach zu bewerkstelligen wurde die Enum-Klasse „CoordinateImagePosition“ eingeführt. Dazu wird diese aus der Klasse „CenterPosCoordinateRenderer“ mit der statischen Methode „getCoordinateEnum(char character)“ aufgerufen. Diese Methode gibt dann einen Enum zurück, welcher unter anderem die Position des Zeichens im Bild enthält. Alles Zeichen welche im Bild nicht vorkommen werden als „SPACE“ Enum gehandhabt und gelten somit einfach als ein Abstand.

7.4 Antippbarkeit von Objekten (onTap / onDoubleTap / VisibleTarget)

7.4.1 Problematik

Spätestens mit der Einführung der Notizen musste ein Weg gefunden werden um gewisse, mit OpenGL gezeichnete, Objekte antippbar zu machen. Denn beispielsweise bei Notizen will man eine gesetzte Notiz auf der Karte jederzeit antippen können, umso mehr Informationen über diese zu erfahren oder diese bearbeiten zu können.
Diese ganze Thematik ist komplizierter als man denken könnte. Da man mit OpenGL Objekte nur zeichnen kann und OpenGL diese dann sozusagen „vergisst“ (da OpenGL eine reine Statemachine ist), ist es nicht möglich einen onClick-Listener oder etwas Ähnliches auf gezeichnete Objekte zu binden. Aus diesem Grund musste ein Konzept gefunden werden, welches dieses Problem möglichst elegant beheben kann.

7.4.2 Ablauf

Im folgenden Diagramm ist der Ablauf in einem Sequenzdiagramm vereinfacht aufgezeichnet. Eine Seite nach dem Diagramm wird dieses im erklärt.

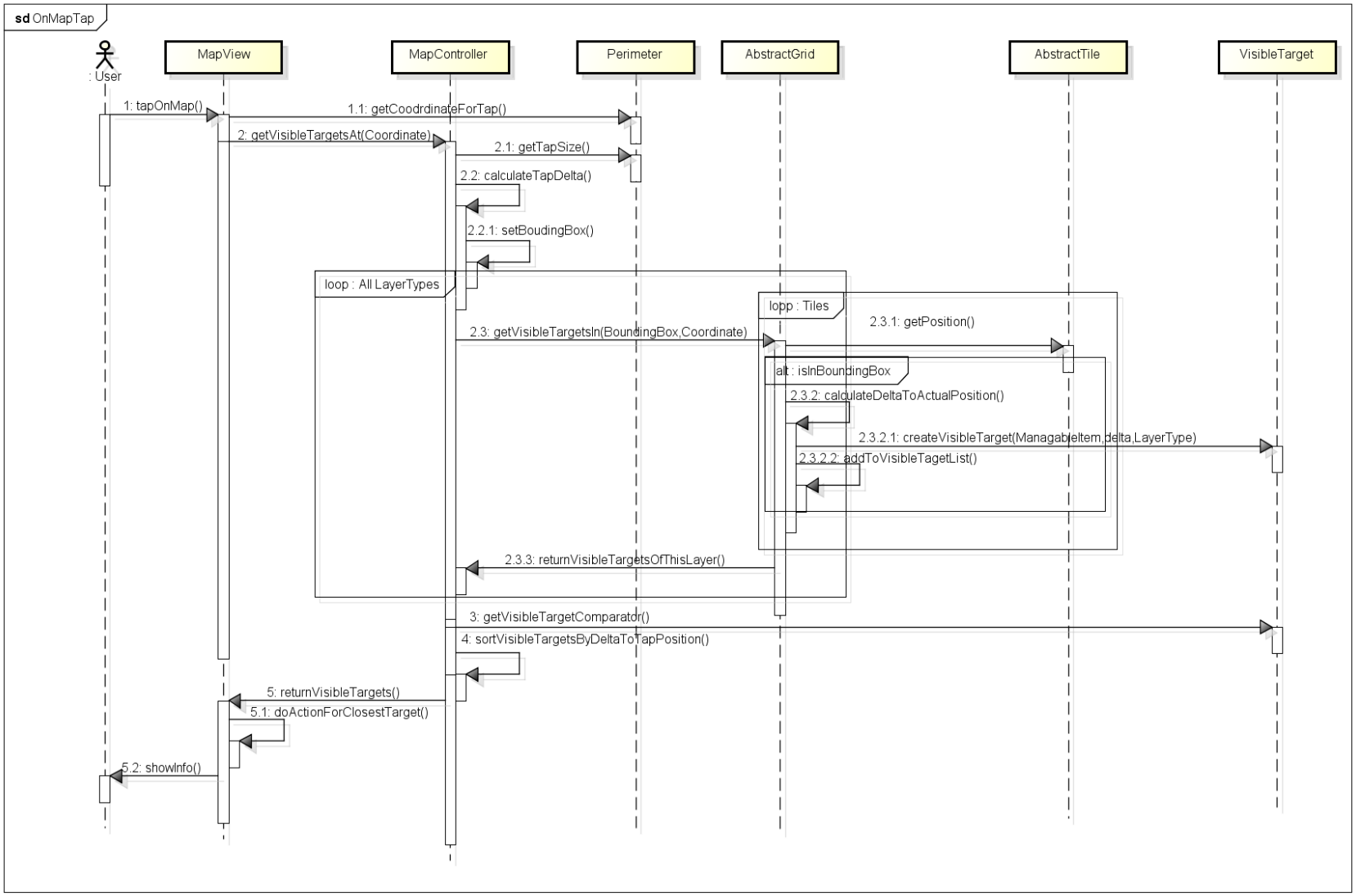


Diagramm 11: Ablauf beim Antippen eines Punktes auf der Karte

Benutzer tippt auf die Karte
Der Auslöser dieser Aktion ist jeweils der Benutzer, welcher mit seinem Finger an einer beliebigen Stelle auf die Karte tippt (1.). Im Sequenzdiagramm ist die Aktion nur für einen normalen Tap ersichtlich, selbstverständlich ist der Ablauf identisch für einen Double-Tap.
Der Tap wird in der Klasse MapView mit einem GestureListener empfangen, sobald ein Tap ausgeführt wurde, wird auf die Perimeter-Klasse einen Request gemacht (1.1), dabei werden die x und y Positionen des Taps auf dem Bildschirm übergeben, die Perimeter-Klasse rechnet diese zu Koordinaten um und gibt diese zurück an die MapView.

Sichtbare Ziele im Tapbereich holen
Als nächstes wird der MapController mit der Methode getVisibleTargetsAt(Coordinate) (2.) aufgerufen. Der Parameter Coordinate ist dabei die Koordinate, welche der Perimeter vorher berechnet hat.
Der MapController holt sich jetzt als erstes vom Perimeter die Grösse eines „Taps“ (2.1). Diese Grösse ist dabei relativ zur Bildschirmgrösse. Diese Berechnung wird im Perimeter mittels der Auflösung, sowie der DPI des Gerätes und unter der Annahme dass ein durchschnittlicher Tap von einem Finger ca. 1 cm gross ist, durchgeführt. Aus dieser Tap-Grösse wird dann mit der aktuellen BoundingBox Grösse (welche Abhängig vom angezeigten Kartenabschnitt ist) ein Delta berechnet (2.2). Dieses Delta ist dafür da, dass ein Benutzer auch etwas neben sein gewünschtes Ziel drücken kann, d.h. es nicht zu 100% treffen muss. Dies ist zwingend nötig und sehr wichtig, denn vor allem wenn man relativ weit aus der Karte heraus gezoomt ist, wäre es sonst gar nicht möglich ein Ziel überhaupt zu treffen. Das Delta stellt dabei wiederum eine BoundingBox dar (2.2.1). Die Grösse dieser BoundingBox ist jeweils abhängig von der aktuellen Zoomstufe und Skalierung. Zum Verständnis folgt das ganze grafisch:

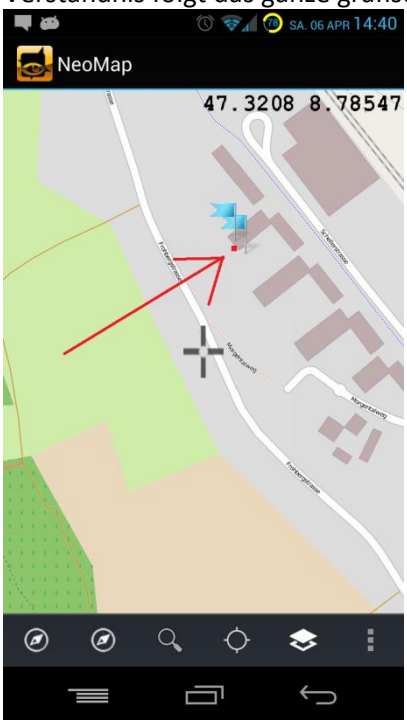


Abbildung 33: Eigentlicher Tap-Punkt

Abbildung 34: Tap –Punkt als BoundingBox

In der ersten Grafik ist die eigentliche Grösse des Tap-Punktes dargestellt. Würde man nur mit dieser Koordinate weiter rechnen, hätte der Benutzer keines der beiden Ziele (die kleinen Flaggen) getroffen, deshalb wurde wie oben beschrieben ein Delta berechnet, und dadurch eine BoundingBox erstellt, durch diese BoundingBox befinden sich beide Ziele im Tap-Bereich.

Alle LayerTypen durch gehen
Nachdem das Delta also berechnet wurde, werden nun alle LayerTypen (NeoMap, Notizen, Geometrie, etc.) durchgegangen und für all diese bzw. für ihre Grids die Methode getVisibleTargetsIn(BoundingBox, Coordinate) (2.3) aufgerufen. Die übergebene BoundingBox ist natürlich die oben berechnete und „Coordinate“ ist die genaue Tap-Position.

Alle Tiles finden, welche in der BoundingBox sind und eine Liste hinzufügen
Die Grids gehen jetzt alle ihre Tiles durch, holen für jedes Tile die genaue Position (2.3.1) und prüfen, ob dieses Tile in der übergebenen BoundingBox ist. Wenn dies der Fall ist, so wird vom aktuellen Tile das Delta zur genauen Tap-Position berechnet (d.h. wie weit entfernt ist dieses davon) (2.3.2). Anschliessend wird ein neues Objekt der Klasse VisibleTarget erstellt (2.3.2.1). Das soeben erstellte VisibleTarget wird dann einer Liste hinzugefügt (welche alle Ziele in der BoundingBox enthält) (2.3.2.2). Sind alle Tiles vom aktuellen Grid durchlaufen, wird die Liste zurück an den MapController gegeben (2.3.3).

Sortieren der Liste mit Zielen und Rückgabe an MapView
Der MapController wiederholt das Spielchen jetzt für alle verfügbaren Layers und hat schlussendlich eine Liste mit allen möglichen Zielen. Diese Liste muss er jetzt noch sortieren, dazu holt er sich einen eigenen Comparator aus der Klasse VisibleTarget (3.) und sortiert die Liste damit (4.). Die Liste wird dabei so sortiert, dass das VisibleTarget mit dem kleinsten Delta ganz vorne in der Liste ist. Dadurch ist sichergestellt, dass das Ziel welches der eigentlichen Tap-Position am nächsten ist, priorisiert behandelt werden kann.
Die Liste wird jetzt zurück an die MapView gegeben (5.). Die MapView kann jetzt die Liste durch gehen und für alle VisibleTargets, abhängig vom Typ, bestimmte Aktionen durchführen. Momentan wird nur für das Ziel eine Aktion durchgeführt, welches am nächsten bei der Tap-Position ist (also das erste in der Liste) (5.1). So wird bei Notizen z.B. der Name der Notiz angezeigt (5.2)und bei einem Double-Tap die Notiz geöffnet.

8 Fragments

8.1 Konzept

Eine weitere Vorgabe für die Umsetzung war das Einführen von Fragments. Fragments unterscheiden sich zu Activities vor allem in dem Punkt, dass diese nie alleine laufen können. Es braucht immer eine laufende Activity, welche diese startet und verwaltet. Dadurch sind Fragments auch an den Lifecycle von Activities gebunden, haben aber dennoch einen zusätzlichen eigenen Lifecycle. Durch diese Gebundenheit ergibt sich der grosse Vorteil, dass es möglich ist mehrere Fragments gleichzeitig anzuzeigen (im Gegensatz zu Activities, bei diesen kann immer nur eines gleichzeitig laufen). Dies resultiert dann z.B. darin, dass auf grösseren Bildschirmen oder im Querformat zwei Fragments nebeneinander angezeigt werden können.

8.2 Umsetzung

In folgender Tabelle ist ein Vorher-/Nachher-Vergleich von Activities und Fragments aufgelistet:

	Activities	Fragments
Vorher	• MapActivity • MapLayerChooserActivity • InfoActivity • LoginActivity • ManagementActivity • NoteManagementActivity • NeoMapManagementActivity • NeoMapSearchActivity • NoteEditActivity	-
Nachher	• MapActivity	• InfoFragment • ManagementFragment • MapLayerChooserFragment • NeoMapDrawOrderSortFragment • NeoMapManagementFragment • NeoMapSearchOnlineFragment • NominatimSearchFragment • NoteEditFragment • NoteManagementFragment • RssNewsFragment • SettingsFragment

Tabelle 29: Activities und Fragments Vorher / Nachher Vergleich

Wie zu sehen ist, wurden im Prinzip alle Activities zu Fragments umgewandelt bis auf die MapActivity, welches die Haupt-Activity darstellt.
Die Umsetzung in der NeoMap App ist so durchgeführt, dass gleichzeitig immer nur ein Fragment und jeweils die MapView (welche immer in der MapActivity läuft) angezeigt werden kann.
Wird ein neues Fragment geöffnet, so werden bereits offene Fragments immer geschlossen. Auf diese Weise kann es nicht passieren, dass im Hintergrund mehrere Fragments gleichzeitig laufen und somit eine Ressourcenknappheit entsteht.
Für die Kommunikation der Fragmente zur MapActivity steht das Interface IFragmentManager zur Verfügung. Dieses wird von der MapActivity implementiert und enthält einige Callback Methoden, welche die Fragments aufrufen können, so können die Fragments über dieses Interface z.B. ein Neu-Zeichnen der Karte oder das Öffnen eines neuen Fragments anstossen. Optional können die Fragments das Interface IFragmentClient implementieren.

Dieses bietet einige Methoden an, die der FragmentManager, d.h. in diesem Fall die MapActivity, aufruft bevor Sie gewisse eigene Funktionen wie z.B. „closeFragment()“ aufruft.

8.3 Darstellung

Wie die Fragmente und die MapView dargestellt werden, ist in der MapActivity definiert und es gelten folgende Regeln:
Ist das Gerät im Hochformat, so wird unabhängig von der Auflösung immer nur die MapView oder nur das Fragment angezeigt:

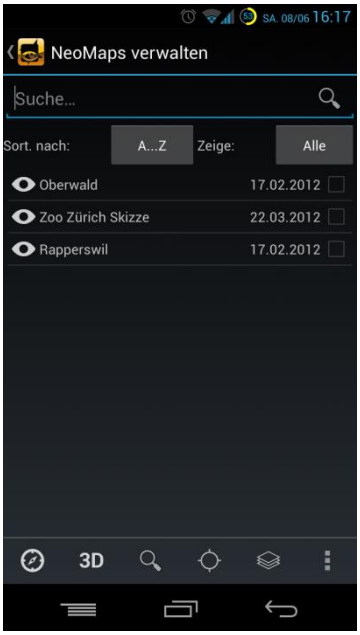


Abbildung 35: Fragment im Hochformat (Auflösung unabhängig)

Im Querformat werden ab einer Auflösung von 1000 Pixeln jeweils das Fragment und die MapView nebeneinander, mit einer Aufteilung von 50:50 angezeigt. Bei Geräten mit kleinerer Auflösung wird nur das Fragment angezeigt:

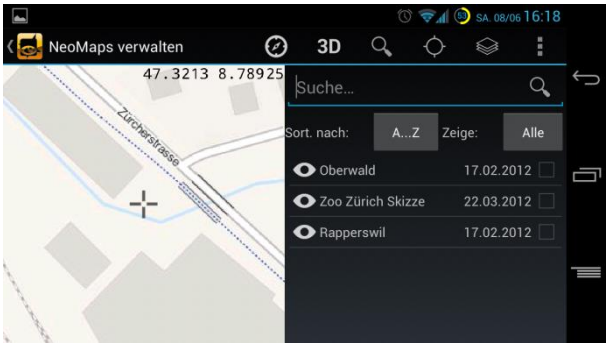


Abbildung 36: Fragment im Querformat auf Gräten mit einer Auflösung > 1000 Px

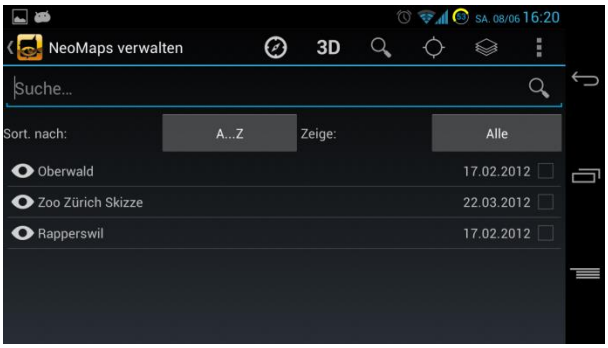


Abbildung 37: Fragment im Querformat auf Geräten mit einer Auflösung < 1000 Px

8.4 Übergabe von Daten zwischen Activities bzw. Fragments

Um Daten zwischen Activities und Fragments zu übergeben, müssen diese in sogenannte Bundles verpackt werden. Bundle ist eine vorgegebene Klasse von Android. Bekannte Datentypen wie Ints, Floats, etc. können in das Bundle einfach mittels der entsprechenden Methode wie z.B. „putFloat(String key, Float Value) übergeben werden.

Etwas komplizierter wird das Ganze aber wenn man ganze Objekte in das Bundle packen will. Dazu gibt es zwei Möglichkeiten:

- Objekt muss serialisierbar sein (implements Serialibzable)
- Objekt muss parcelable sein (implements Parcelable)

Von Google wird vor allem auf Grund der Geschwindigkeit, empfohlen Parcelable zu verwenden, denn dieses Interface wurde explizit für Android entwickelt.
Trotzdem wurde die Entscheidung zugunsten von Serialibzable gefällt. Warum dies so ist, soll nachfolgendend kurz beschrieben werden.

8.4.1 Parcelable

Ein Objekt dass das Parcelable Interface implementiert, muss mindestens folgende Methoden überschreiben:

- public int describeContents()
  - Dient dazu den Content des Parcelable mit bestimmten Flags zu überschreiben, ist in der Applikation jedoch irrelevant und gibt deshalb immer 0 zurück.
- public void writeToParcel(Parcel dest, int flags)
  - Dies ist die wichtige Methode: In Ihr müssen ALLE Felder welche später wieder ausgelesen werden sollen, in das Parcel geschrieben werden. Dazu bietet ein Parcel die Methode „writeString(…), writeInt(…)“ etc. an. Auffällig ist dabei, dass die Methoden ohne Angabe um welches Feld es sich eigentlich handelt, benutzt werden müssen. Das bedeutet aber auch, dass die Reihenfolge beim ein- und auslesen genau gleich sein muss!

Zusätzlich muss ein Objekt folgende statische Methode besitzen:

```
public static final Parcelable.Creator<Objekt> CREATOR = new
Parcelable.Creator<Objekt>() {
            public Objekt createFromParcel(Parcel in) {
                  return new Objekt(in);
            }

            public Objekt[] newArray(int size) {
                  return new Objekt[size];
            }
}
```

Quellcode 9: Statische Parcelable Methoden

Dadurch wird auch ersichtlich, dass ein weiterer Konstruktor benötigt wird, welcher das Parcel wieder einliest. Bei diesem ist es essentiell, dass das Einlesen in der gleichen Reihenfolge wie das hinausschreiben stattfindet! Sehr mühsam ist es also wenn eine Klasse ein neues Attribut erhält. Dieses muss dann in beiden Parcel-Methoden (in korrekter Reihenfolge!) hinzugefügt werden.

8.4.2 Serializable

Implementiert ein Objekt Serializable, muss in den meisten Fällen gar nichts gemacht werden. Eigentlich spielt nicht mal die Serial-ID eine Rolle, da die Objekte so oder so nur für die Übergabe serialisiert werden und nicht persistiert werden.

8.4.3 Entscheidung

Obwohl Parcelable optimiert für Android sein soll, wurde der Entscheid zu Gunsten von Serializable gefällt. Da gleichzeitig nur immer ein Objekt serialisiert wird, spielt die Performance in diesem Anwendungsfall kaum eine Rolle. Ausserdem entsteht durch Parcelable ein grosser Overhead an Code und Wartungsaufwand.

9 NeoMaps

9.1 Einleitung

Am Grundprinzip von NeoMaps wurde nichts geändert. Dies bedeutet, dass das bereits existierende Konzept der NeoMaps übernommen wurde. Auch die Masken zur Verwaltung und zum Download wurden in ihren Grundzügen beibehalten, wobei diese vor allem hinter den Kulissen einem grossen Refactoring unterzogen (Extrahierung des XML-Designs sowie Generalisierung / Generisierung der Masken) und an das Standard Android-Design angepasst wurden. Da bekanntlich aber keine Dokumentation vorhanden war, werden NeoMaps hier auch nochmal beschrieben.
Eine Ausnahme stellt das Unterkapitel „Priorisierung von NeoMaps“ dar, dieses Konzept wurde komplett neu eingeführt.

9.2 Konzept

Bei NeoMaps geht es darum, dass es jedem offen steht eigene NeoMaps zu erstellen, diese wenn gewünscht mit anderen zu teilen und in der App anzuzeigen.
Um eine eigene NeoMap hinzufügen zu können, müssen einige Schritte vollzogen werden:

1. Georeferenzieren des Bildes (http://de.wikipedia.org/wiki/Georeferenzierung)
2. Erstellen eines KMZ Files (http://de.wikipedia.org/wiki/Keyhole_Markup_Language)
3. Hochladen der Datei auf http://neomap.hsr.ch
    a. Auswahl ob Map privat oder öffentlich sein soll
4. Herunterladen der NeoMap in der App, über den Menüpunkt „NeoMaps herunterladen“

Von Relevanz in dieser Arbeit ist eigentlich nur der letzte Schritt. Die anderen Schritte stehen nur indirekt im Zusammenhang mit der Arbeit, da sich die Arbeit ausschliesslich mit dem App befasst.

Ist eine NeoMap erst einmal heruntergeladen, so ist diese standardmässig auf der Karte sichtbar. Eine NeoMap kann über die NeoMap Verwaltung jederzeit gelöscht oder ausgeblendet werden. Die Verwaltung ist dabei beinahe identisch zur Verwaltung der Notizen.

Wird eine NeoMap auf der Karte angetippt, so wird der Name der NeoMap ausgegeben. Im Gegensatz zu anderen verwaltbaren Objekten (wie Notizen), existiert für eine einzelne NeoMap keine weiteren Funktionen (d.h. keine Doppel-Tap Aktion). Sind jedoch mehrere NeoMaps überlappend vorhanden, so öffnet sich bei einem Doppel-Tap ein Fragment welches das Sortieren bzw. priorisieren dieser Maps zulässt (siehe „Priorisierung von NeoMaps“).

Abbildung 38: NeoMaps Herunterladen

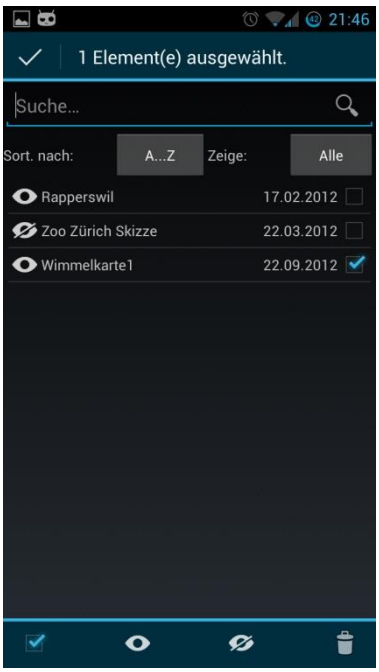


Abbildung 39: NeoMaps Verwalten

Abbildung 40: NeoMap auf der Karte

9.3 Umsetzung

9.3.1 Klassenbeschrieb

Auf der folgenden Seite ist ein Diagramm mit den für die NeoMaps relevanten Klassen abgebildet (Achtung: Klassen die nichts direkt mit NeoMaps zu tun haben, wurden weggelassen!).

Zu beachten ist auch, dass auf das Zeichnen von NeoMaps mit OpenGL gar nicht eingegangen wird, dies ist exemplarisch bereits weiter oben im Kapitel „Karten-Darstellungsgrundlagen (Rendering)“ beschrieben.

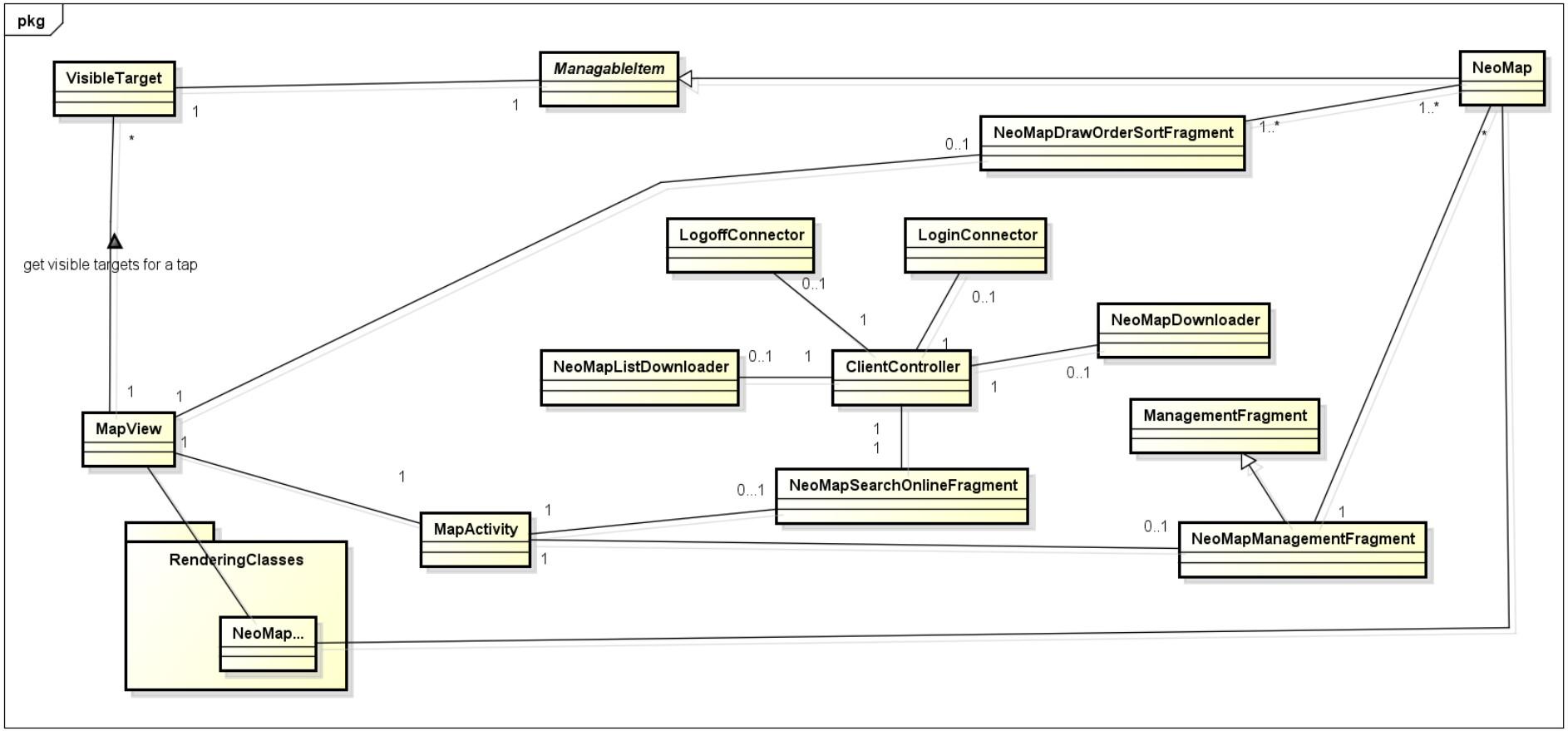


Diagramm 12: NeoMap Klassen

NeoMap
Dies ist die Domain-Klasse. Die NeoMap-Klasse enthält alle nötigen Informationen über eine Map. Sie umfasst noch weitere Subklassen wie z.B. GroundOverlay oder BoundingBox. Diese wurden im Diagramm jedoch weggelassen, da sie für das Verständnis irrelevant sind.

NeoMapManagementFragment
In diesem Fragment können NeoMaps verwaltet werden. Sie können gelöscht oder ausgeblendet werden. Ausserdem können aus dieser Klasse heraus detaillierte Informationen zu NeoMaps angezeigt und die NeoMap selbst auf der Karte angezeigt werden.

NeoMapSearchOnlineFragment
Über dieses Fragment wird online (d.h. von der NeoMap Webseite) mittels der Klasse ClientController eine Liste von NeoMaps abgerufen und angezeigt. NeoMaps können dann ausgewählt und installiert werden.

ClientController
Der ClientController baut mittels der DefaultHttpClient-Klasse (built-in Android Klasse) eine Verbindung zur NeoMap-Webseite auf. Abhängig davon, ob der Benutzer in den Einstellungen Login-Daten eingetragen hat, wird vor dem eigentlichen Download der Liste bzw. eines NeoMaps ein Login ausgeführt. Das Login ist nötig, damit ein Benutzer auch private NeoMaps sehen kann. Dazu wird die Klasse LoginConnector verwendet.

LoginConnector
Diese Klasse ist ein AsyncTask. Sie führt das eigentliche Login durch. Damit der spätere Listen Download funktioniert wird das gleiche DefaultHttpClient Objekt verwendet, welches im ClientController erstellt wurde. Dadurch wird bei dieser Verbindung serverseitig gespeichert, ob und mit welchem Account ein Benutzer eingeloggt ist. So können dann die korrekten NeoMaps angezeigt werden (d.h. private, vom eingeloggten Benutzeraccount, und öffentliche). Diese Klasse arbeitet mit CallBacks: Sie teilt dem NeoMapSearchOnlineFragment jeweils mit, ob der Benutzer mit den mitgegebenen Einstellungen erfolgreich authentifiziert werden konnte. So hat der Benutzer immer ein Feedback.

NeoMapListDownloader / NeoMapDownloader
Auch diese beiden Klassen sind AsyncTasks und führen den eigentlichen Download der NeoMapListe bzw. einer einzelnen NeoMap aus. Auch sie benutzen den im ClientController erstellten DefaultHttpClient. Der NeoMapListDownloader gibt zusätzlich nochmals die Login Daten im Body des GET-Requests mit, damit die richtigen NeoMaps angezeigt werden. Im Falle des NeoMapDownloaders wird bei erfolgreichem Download die NeoMap direkt in die Datenbank gespeichert. Diese beiden Klassen arbeiten wiederum mit CallBacks und teilen dem NeoMapSearchOnlineFragment mit wann die Downloads abgeschlossen sind.

NeoMapDrawOrderSortFragment
Dient zur Priorisierung von NeoMaps. Dies ist ausführlich in Kapitel „Priorisierung von NeoMaps“ beschrieben.

MapView
Dient als Eintrittspunkt für zwei Aktionen: Zum einen wird mit einem einfachen Tap auf eine NeoMap deren Namen ausgegeben. Zum anderen wird, falls mehrere NeoMaps überlappend vorhanden sind, mit einem Doppel-Tap das NeoMapDrawOrderFragment geöffnet.

RenderingClasses
Das Paket RenderingClasses ist nur fiktiv. Anstelle dieses Paketes würden einfach alle Renderingklassen vorhanden sein. Da diese aber bei allen ManagableItems (Notes, NeoMap, Nominatim) nach dem genau gleichen Prinzip funktionieren, sind diese hier nicht beschrieben. Generelles über die Renderklassen ist im Kapitel „Karten-Darstellungsgrundlagen (Rendering)“vorhanden.

9.4 Priorisierung von NeoMaps

9.4.1 Problem

Hat ein Benutzer für eine bestimmte Stelle mehrere NeoMaps heruntergeladen, so gab es bis anhin keine Möglichkeit die Zeichnungsreihenfolge (welche NeoMaps überlappen sich bzw. welche ist über einer anderen) zu ändern. Es war auch nirgends ersichtlich, dass sich eventuell eine zweite NeoMap unter einer anderen befindet. Die einzige Möglichkeit die bestand, war über die NeoMap-Verwaltung NeoMaps auszublenden. Da diese aber in keinem offensichtlichen Zusammenhang zueinander stehen, war es nur durch ausprobieren möglich eine bestimmte NeoMap zu sehen.

9.4.2 Lösung

Die Lösung wurde so umgesetzt, dass man bei einem einfachen Tippen (Single-Tap) auf eine NeoMap sieht, ob noch andere NeoMaps in der Nähe sind.

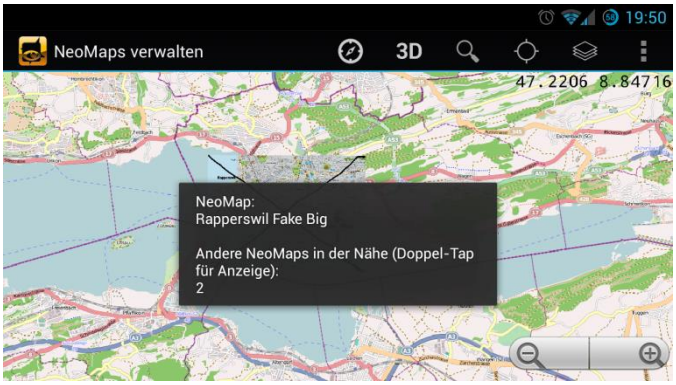


Abbildung 41: NeoMaps in der Nähe

Mit doppeltem Tippen öffnet sich dann ein Fenster, in welchem die Zeichnungsreihenfolge der NeoMaps per Drag & Drop geändert werden kann. Ausserdem können an dieser Stelle die NeoMaps direkt ein bzw. ausgeschaltet werden, dazu muss ein Eintrag lediglich angetippt werden. Befindet man sich auf einem Gerät im Querformat und hat eine genug grosse Auflösung, so ist eine Dual-Ansicht vorhanden und Sortieränderungen können direkt in der Karte beobachtet werden (siehe dazu auch Kapitel „Fragments“).

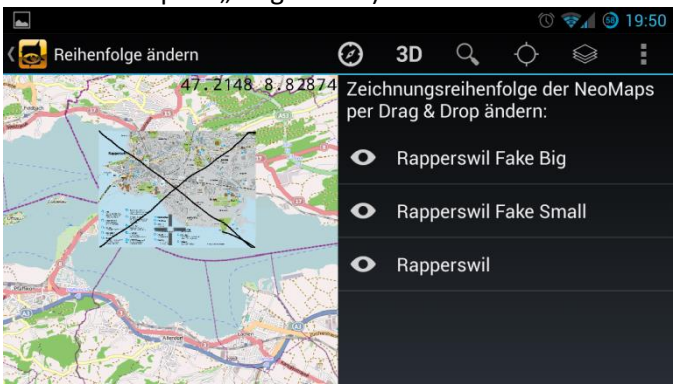


Abbildung 42: Sortierung ändern für NeoMaps in der Nähe

9.4.3 Konzept

Obwohl das Konzept einfach klingt, ist die Umsetzung etwas trickreicher. Die Hauptfrage, welche sich dabei stellte, ist:
In welchem Zusammenhang soll die Sortierung gespeichert werden?
Denn das Problem dabei ist, dass es keine absolute Sortierung (d.h. über alle NeoMaps) geben kann bzw. diese nicht sinnvoll ist. Eine Sortierung sollte immer nur für NeoMaps gültig sein, welche sich auch überlappen. Dies führt aber direkt zum nächsten Problem:
Denn falls NeoMap A, die NeoMap B überlappt, kann es sein, dass NeoMap B seinerseits wiederum NeoMap C überlappt. NeoMap C hat jedoch im Prinzip nichts mit NeoMap A zu tun, sondern die Verbindung besteht nur indirekt über NeoMap B.
Aus diesem Grund wurde eine rekursive Lösung implementiert, d.h. eine Sortierung ist immer für alle NeoMaps gültig, für welche irgendeine Verbindung besteht, dabei ist es egal ob diese Verbindung direkt, oder über beliebig viele andere NeoMaps geht.

Mit folgendem Bild soll das Prinzip noch einmal beispielhaft dargestellt und erläutert werden:

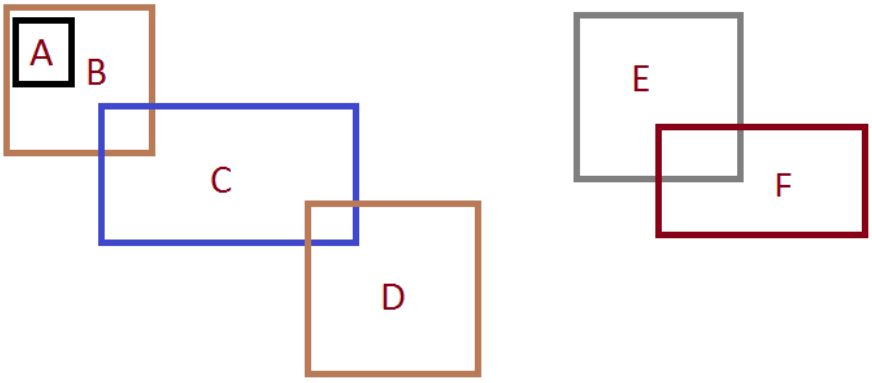


Abbildung 43: NeoMap Überlappungs-Beispiel

In der Abbildung sind verschiedene NeoMaps zu sehen. Ist für ein Benutzer jetzt die NeoMap B ersichtlich (NeoMap A befindet sich unter B) und er tippt diese an, so würde er die Nachricht erhalten, dass drei weitere NeoMaps (nämlich A, C und D) in der Nähe sind (denn diese sind alle über direkte oder indirekte Überlappung miteinander verbunden). Würde er jetzt doppelt auf die NeoMap B tippen, so könnte die Sortierreihenfolge so aussehen:

- D 1 (zu Oberst)
- C 2
- B 3
- A 4 (zu Unterst)

Diese Reihenfolge kann der Benutzer jetzt per Drag und Drop beliebig ändern. Will er jetzt, dass NeoMap A anstatt B sichtbar ist, könnte er einfach NeoMap A und B in der Liste tauschen.

Alle Änderungen die er vornimmt, haben aber keinen Einfluss auf die Sortierung der NeoMaps E und F. Diese stehen, wie zu sehen ist, in keiner Verbindung zu den NeoMaps A-D und könnten beispielsweise so aussehen:

- F 1
- E 2

9.4.4 Umsetzung

Die Prüfung, ob sich NeoMaps berühren, findet immer über die BoundingBox der NeoMaps statt. Dabei wird ausgehend von der angetippten NeoMap eine Liste mit allen NeoMaps durchlaufen. Findet sich in der Liste eine NeoMap, welche die BoundingBox der angetippten NeoMap berührt, so wird dieses in eine Resultatliste gespeichert und anschliessend für die BoundingBox von dieser neuen NeoMap der ganze Prozess wiederholt (Rekursion).

9.4.4.1 Drag & Drop Sortierung

Damit eine Liste per Drag & Drop sortiert werden kann, musste eine Third-Party-Library verwendet werden, da Android von sich aus, keine solche Liste anbietet. Die Third-Party-Library wurde mit einem Git-Submodule integriert. Das Projekt findet sich unter https://github.com/bauerca/drag-sort-listview.
Die Liste ist sehr einfach zu verwenden, es wird auf diese deshalb nicht weiter eingegangen.

9.5 Persistenz

Die Speicherung in der Datenbank von NeoMaps ist sehr simpel. Es existiert lediglich eine einzige Tabelle, welche alle Informationen enthält. Dabei enthält Sie auch eine „map_id“ welche vom Server mitgegeben wird. Sie sorgt dafür, dass NeoMaps überall eindeutig identifizierbar sind.
Die BoundingBox gibt dabei an, an welchen Koordinaten sich die Eckpunkte einer Karte befinden. Schön zu sehen ist auch, dass das Feld „draworder“ einfach nur ein Integer ist und in keinerlei Abhängigkeit zu anderen NeoMaps steht. Diese Abhängigkeit wird ausserhalb der Datenbank, also direkt in der Applikation, bestimmt.

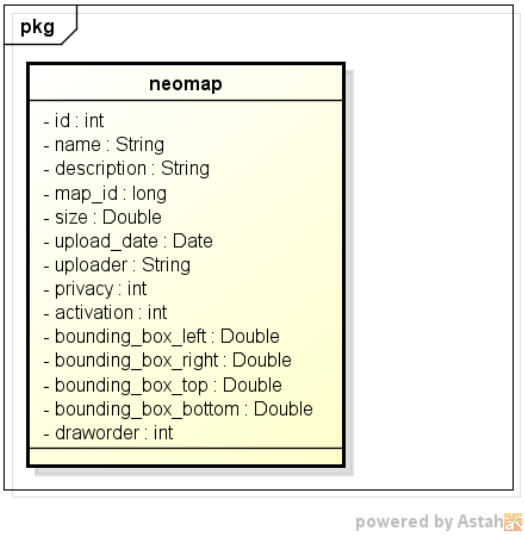


Diagramm 13: NeoMap Datenmodell

10 Notizen

10.1 Konzept

Notizen ist in der NeoMap App der Überbegriff für:

- **Punktegeometrie** (Point), diese stellen genau einen Punkt (eine Koordinate) auf der Karte dar und können mit einem Titel sowie einer Beschreibung versehen werden. Sie können zum Beispiel benutzt werden, um eine interessante Stelle auf der Karte wie z.B. einen Rastplatz zu markieren.
- **Liniengeometrie** (LineString), diese Stellen mehrere miteinander verbundene Punkte auf der Karte dar und können mit einem Titel sowie einer Beschreibung versehen werden. Der Nutzen von Liniengeometrien liegt darin, um zum Beispiel ein Gebiet abzustecken oder eine Route grob zu erfassen. Ausserdem können Liniengeometrien zur Distanzmessung gebraucht werden.

Damit Notizen auch ausserhalb der NeoMap App benutzt werden können, steht dem Benutzer eine Exportfunktion zur Verfügung, mit dieser kann er beliebig viele Notizen gleichzeitig in das bekannte Format GPX exportieren. Diese Dateien können dann in anderen Programmen importiert werden.

Eine Notiz kann jeweils aktiv (angetippt) oder inaktiv (nicht angetippt) sein (siehe auch Kapitel „Antippbarkeit von Objekten (onTap / onDoubleTap / VisibleTarget)“). Es wurde für die Darstellung der Notizen folgende Bilder gewählt:

	Inaktiv	Aktiv
Notiz Punktgeometrie		
Notiz Liniengeometrie		

Tabelle 30: Notiz Icons

Einmaliges Antippen von Notizen bewirkt immer das Anzeigen des Namens der Notiz. Bei Liniengeometrien wird jeweils noch die Länge bzw. Distanz ausgegeben, die durch die verbundenen Punkte entstanden ist.

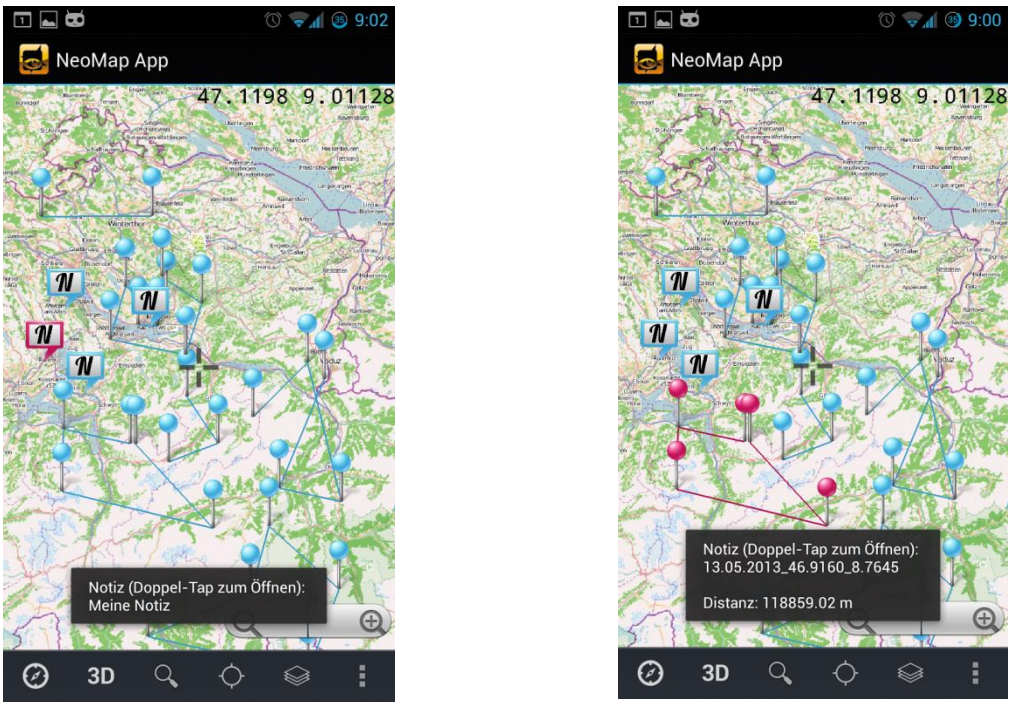


Abbildung 44: Verschiedene Notizen auf der Karte

Das neu Erfassen von Notizen kann nur über das LongPressMenu erreicht werden, währenddem das Bearbeiten von Notizen, entweder durch doppeltes Antippen oder über die Notizverwaltung ausgelöst werden kann.

Bei neuen Notizen wird zudem standardmässig automatisch ein Titel generiert, dieser besteht aus den Anfangskoordinaten und dem aktuellen Datum. Dies dient vor allem der schnellen Erfassung von Notizen. Der Titel kann mit Hilfe eines Buttons in einem Klick gelöscht und geändert werden.

10.2 Ablauf

Der Ablauf für die Erfassung bzw. das Bearbeiten für einzelne Punkte, sowie Liniengeometrien ist grösstenteils identisch. Der Unterschied besteht vor allem darin, dass beim Erfassen von einzelnen Notizpunkten anfangs immer direkt in die Bearbeitungsmaske (d.h. der Teil wo der Titel und die Beschreibung eingegeben und die Notiz dann gespeichert werden kann) gewechselt wird, währenddem bei Liniengeometrien immer zuerst der Kartenänderungsmodus (d.h. neue Punkte setzen, alte Punkte löschen, rückgängig machen des letzten gesetzten Punktes etc.) angezeigt wird. Zwischen diesen beiden Modi, kann beliebig hin und her gewechselt werden, d.h. es kann jederzeit die Position von Punkten geändert werden, oder neue Punkte können hinzugefügt werden. Die Speicherung der Notiz in der Datenbank findet schlussendlich jedoch immer über die Bearbeitungsmaske statt.

Der Grund für die beiden unterschiedlichen Einstiegspunkte ist vor allem der, dass bei einem Notizpunkt der Benutzer bereits durch das lange Drücken auf eine bestimmte Stelle entschieden hat wo er diesen einen Punkt setzen möchte. Bei Liniengeometrien möchte der Benutzer jedoch mehr als nur einen Punkt auf der Karte erfassen. Aus diesem Grund wird zuerst der Kartenänderungsmodus gestartet.

Abgesehen von einem anderen Einstiegspunkt, verhält sich auch das Bearbeiten von Notizen genau gleich wie das Erfassen.

Auf der folgenden Seite ist der Ablauf für das Erfassen von Notizen schematisch dargestellt.

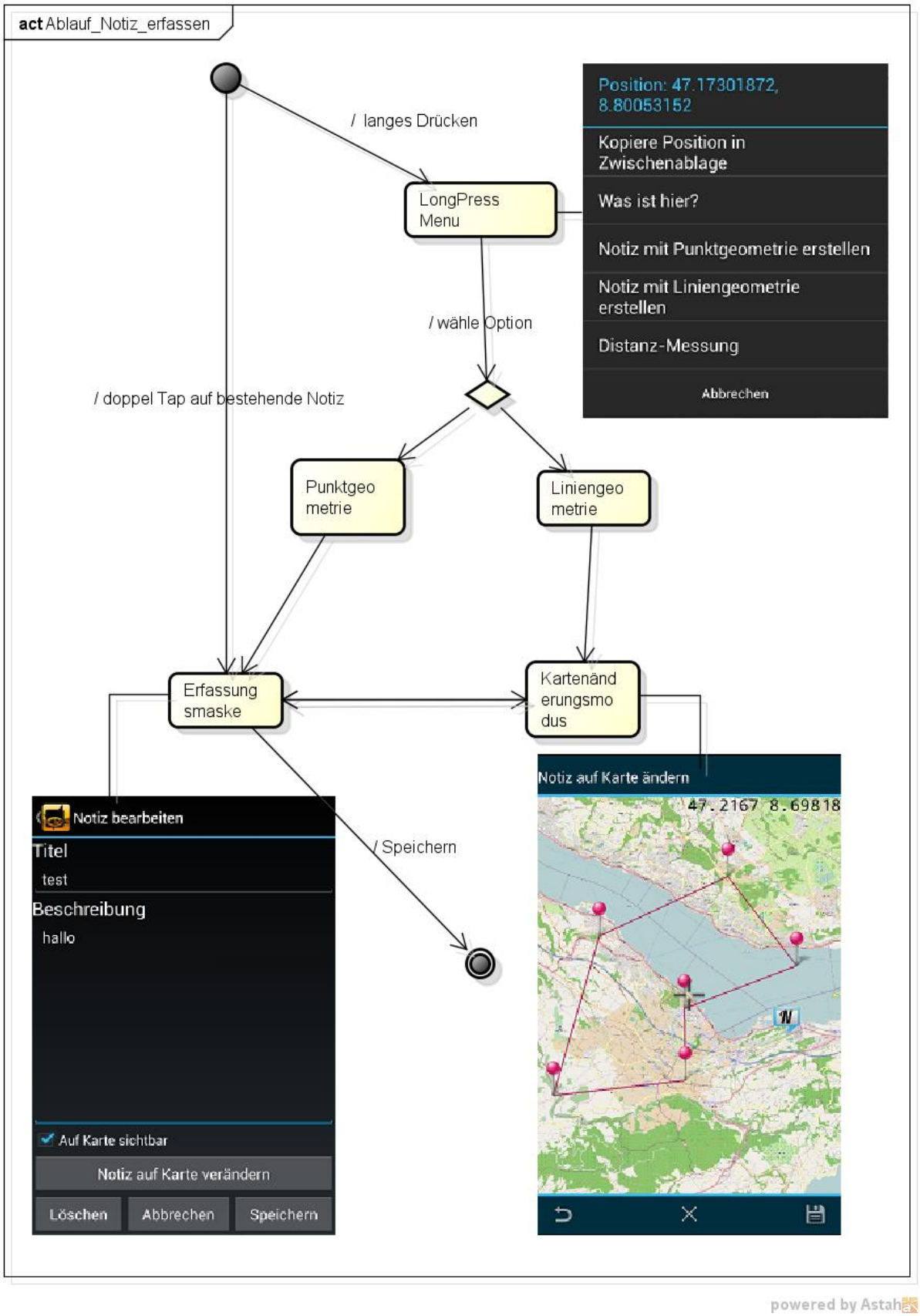


Diagramm 14: Ablauf Erfassen von Notizen

10.3 Umsetzung

10.3.1 Klassenbeschrieb

Auf der folgenden Seite ist ein Diagramm mit den für die Notizen relevanten Klassen abgebildet (Achtung: Klassen die nichts direkt mit Notizen zu tun haben, wurden weggelassen!).

Zu beachten ist auch, dass auf das Zeichnen der Notizen mit OpenGL gar nicht eingegangen wird, dies ist exemplarisch bereits weiter oben im Kapitel „Karten-Darstellungsgrundlagen (Rendering)“ beschrieben.

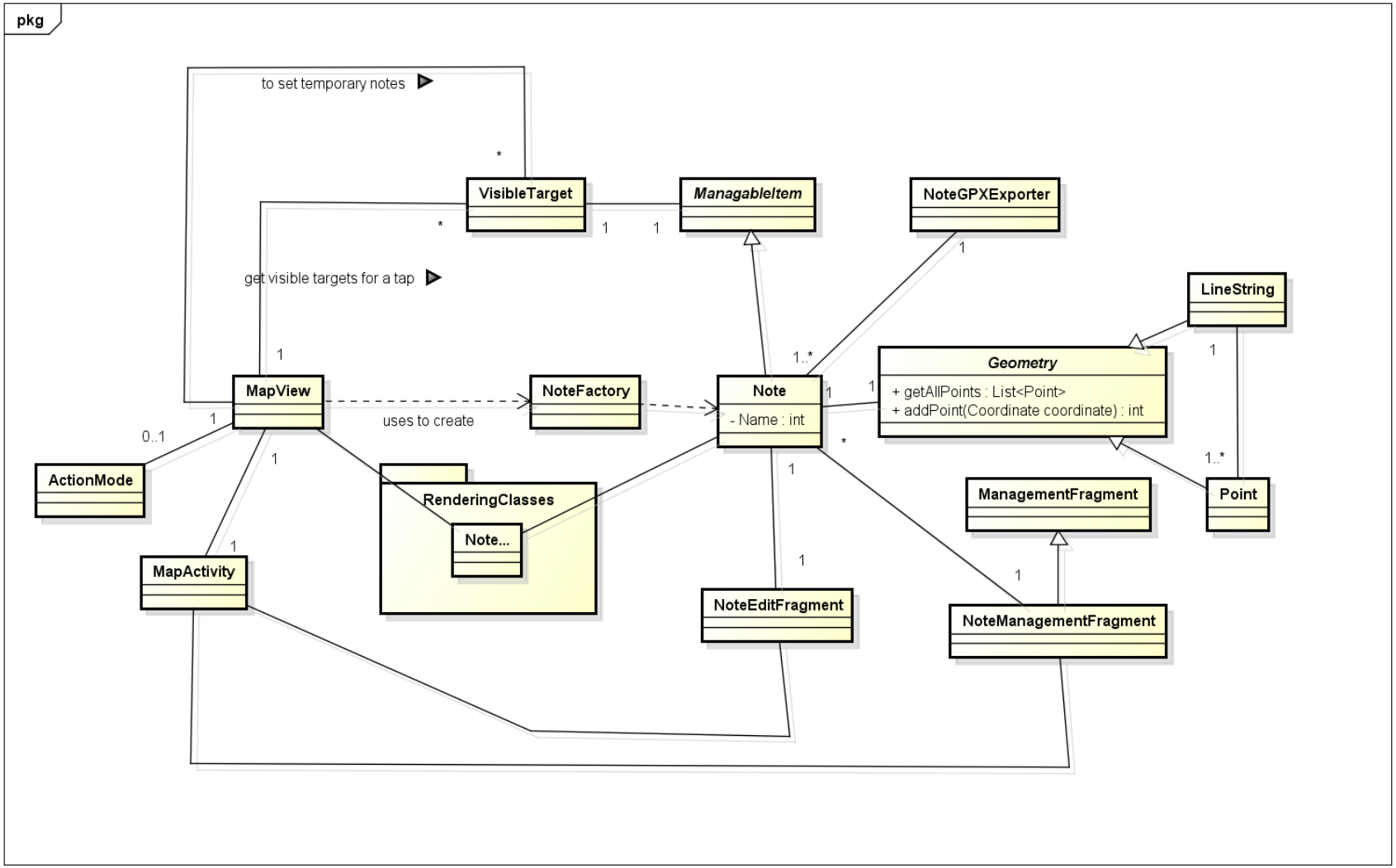


Diagramm 15: Notiz Klassen

Note
Die Klasse Note ist der zentrale Punkt der ganzen Geschichte und ist damit auch das Domain-Objekt. Sie enthält alle relevanten Informationen, wie z.B. den Namen der Notiz, die Notiz Art bzw. Geometrie usw. Sie ist von der abstrakten Klasse ManagableItem abgeleitet. Damit kann Sie von der Klasse NoteManagementFragment bzw. dessen abstrakten Klasse ManagementFragment verwaltet werden.

Geometry / Point / LineString
Die abstrakte Klasse Geometry ist die Basisklasse von Point und LineString. Wie im Diagramm zu sehen ist, enthält die Klasse LineString beliebig viele Points. Ausserdem haben beide Klassen dadurch, dass sie von Geometry erben, die gleichen „Hauptmethoden“. Dank dieser Methoden muss an vielen Orten, wie z.B. dem DB Zugriff oder in der MapView, bei der Verwendung nicht unterschieden werden um welche konkrete Klassen es sich überhaupt handelt. Es wird eine Art Composite-Pattern angewendet. Die LineString sowie die Point-Klassen enthalten beide einen Stack, welcher Undo-Operationen ermöglicht.

NoteEditFragment
Dies ist die Bearbeitungsklasse, über sie kann der Text einer einzelnen Notiz verändert bzw. erfasst werden. Sie dient hauptsächlich, dazu um den Namen, die Beschreibung, sowie die Sichtbarkeit der Notiz auf der Karte einzustellen. Diese Verwaltung erhält man einerseits wenn man über das „LongPressMenu“ eine neue Notiz erfasst und andererseits wenn man eine Notiz auf der Karte doppelt antippt (Double-Tap). In dieser kann auch der Kartenänderungsmodus für Notizen gestartet werden (Klasse MapView / Action Mode).

NoteManagmentFragment
Diese Klasse stellt eine Übersicht über alle vorhandenen Notizen dar, die Notizen können dort auch aktiviert, deaktiviert und gelöscht werden. Ausserdem kann über diese Klasse eine Notiz direkt auf der Karte angezeigt werden. Diese Fragment hat im Prinzip die gleiche Funktionalität wie die NeoMapManagementklasse, mit Ausnahme davon, dass Notizen noch exportiert werden können. Ein Export wird immer über diese Klasse gestartet.

MapView
Die MapView-Klasse stellt den zentralen Eingangspunkt für die meisten Änderungen von Notizen dar. Über diese Klasse werden neue Notizen erstellt und geändert, dabei wird eine NoteFactory verwendet, welche komfortable Methoden zur Erstellung von Punkten, sowie Geometriegebilden anbietet. Ausserdem enthält sie eine Referenz zur Action Mode Klasse, welche für den Kartenänderungsmodus zuständig ist.

Action Mode / Kartenänderungsmodus
Der Action Mode ist eine von Android vergebene Klasse. Schaltet man diesen ein, so wird die normale Action Bar ausgeblendet und die Contextual Action Bar eingeblendet. Eigentlich wird dieser Modus normalerweise für die Mehrfachauswahl von Elementen und ähnlichem verwendet (siehe auch „Contextual Action Mode / Contextual Action Bar“). In diesem Fall werden in diesem Modus jedoch Notizpunkte auf der Karte verschoben, hinzugefügt, gelöscht und rückgängig gemacht.

RenderingClasses
Das Paket RenderingClasses ist nur fiktiv. Anstelle dieses Paketes würden einfach alle Renderingklassen vorhanden sein. Da diese aber bei allen ManagableItems (Notes, NeoMap, Nominatim) nach dem genau gleichen Prinzip funktionieren, sind diese hier nicht beschrieben. Generelles über die Renderklassen ist im Kapitel „Karten-Darstellungsgrundlagen (Rendering)“vorhanden.

NoteGPXExporter
Wird zum Export von Notizen gebraucht. Siehe Kapitel „Export“.

10.3.2 Export

Um einen möglichst einfachen Export von Notizen in das GPX-Format zu realisieren, wurde zuerst nach einer passenden Library gesucht. Leider fand sich keine Library die diese Anforderungen erfüllen konnte, deshalb wurde eine Eigen-Implementation gemacht. Diese ist sehr einfach gehalten: Es wurde nur genau das implementiert was auch gebraucht wird, d.h. es wurde verzichtet eine generische Build-Klasse einzuführen, bei der man von aussen die gewünschten TAGs etc. für den GPX-Export angeben kann (dies ist bei „XML-Build-Klassen“ ansonsten üblich). Stattdessen erstellt man einfach eine neue Instanz der GPX-Export-Klasse und gibt ihr entweder eine einzelne Notiz oder aber eine Liste von Notizen mit. Die Klasse erstellt daraus dann – ohne weitere äussere Einflüsse – ein GPX-File, welches alle nötigen Angaben enthält. Notizen werden dabei folgendermassen auf die entsprechenden GPX-Elemente gemappt:

Notiz Art	GPX	Erklärung
Punktgeometrie	Waypoint, <wpt>	Ein Waypoint ist in GPX die einzig sinnvolle Implementation für einen einzelnen Punkt. Er besteht aus einer einzelnen Koordinate.
Liniengeometrie	Track, <trk> <trkseg> <trkpt>	Bei der Liniengeometrie gibt es in GPX entweder die Möglichkeit eine Route (<rte>) oder ein Track (<trk>) zu definieren. Ein Track ist sinnvoller, denn eine Route ist immer ein konkreter Weg von einem Punkt A zu einem anderen Punkt B. In der NeoMap App können Linien aber auch zur Eingrenzung von Gebieten oder ähnlichem gebraucht werden, deshalb ist es sinnvoller Track-Elemente zu verwenden. Alle Tracks enthalten jeweils nur ein Segment (<trkseg>), welches alle Punkte (<trkpt>) der Linie enthält. Mehrere Segmente würden in diesem Fall gar keinen Sinn machen, da keinerlei Informationen darüber in NeoMap App gespeichert werden.

Tabelle 31: Mapping von Notizen zu GPX

10.4 Persistenz

Die benötigten Tabellen in der Datenbank sind sehr simpel und erklären sich von selbst. Sie unterscheiden sich jedoch wesentlich von den konkreten Klassen, da kein ORMapper verwendet wurde (für Android existiert keine Library die Generics, sowie Abstrakte Klassen wirklich gut unterstützt, deswegen wurde darauf verzichtet).

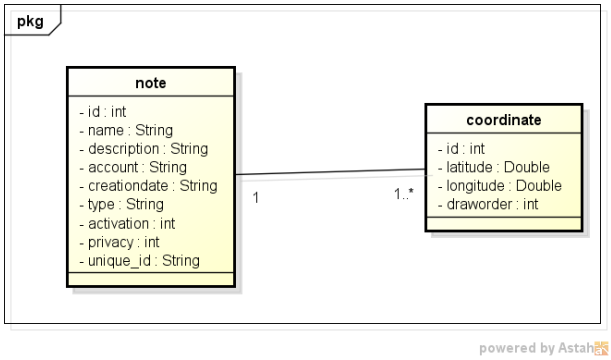


Diagramm 16: Notiz Datenmodell

Es wurde darauf geachtet, dass eine Erweiterbarkeit garantiert ist. Aus diesem Grund ist auch bereits ein Feld für die Privatsphäre, sowie eine Weltweit eindeutige ID vorhanden. Sollte später der Wunsch bestehen, dass Notizen auch über die NeoMap Webseite geteilt werden können, ist eine solche Erweiterung also einfach zu realisieren.
Jede Notiz hat in der Datenbank immer mindesten eine Koordinate (Punktgeometrie). Liniengeometrien hingegen haben im Normalfall zwei oder mehrere Koordinaten. Bei diesen ist das Feld „draworder“ von Bedeutung, es gibt die Reihenfolge an, in welcher die einzelnen Punkte gezeichnet bzw. miteinander verbunden werden müssen. Dadurch ist sichergestellt, dass alle Liniengeometrien immer wieder genauso gezeichnet werden, wie sie der Benutzer erstellt hat.

11 Geografische Namenssuche (Nominatim)

11.1 Einführung

Mittels der geografischen Namenssuche soll nach allen möglichen geografischen Elementen gesucht werden können. Dies sind z.B. Strassen, Orte, Länder und Universitäten (z.B. HSR). Ganz ähnlich also wie es zum Beispiel Google Maps erlaubt. Der Fachbegriff für diese Suche ist Query- oder Normaler-Lookup. Gleichzeitig soll auch die bekannte Funktion „Was ist hier?“ abgedeckt werden, d.h. der Benutzer soll jederzeit einen Punkt auf der Karte auswählen können und erfahren was sich dort bzw. hinter diesen Koordinaten befindet. Dieses Prinzip nennt sich Reverse- bzw. Coordinate-Lookup.
Für die geografische Namenssuche existiert von OpenStreetMap ein offizielles Projekt namens „Nominatim“, was vom Lateinischen kommt und so viel bedeutet wie „nach Namen“. Nominatim ist ein Webservice der per REST verschiedene Funktionen zur Verfügung stellt, um nach Namen zu suchen oder aber einen Reverse-Lookup (also über Koordinaten) durchzuführen.

11.2 Konzept

Da es eine sehr grosse Anzahl von geografischen Daten gibt, ist es nicht möglich alles offline bereit zu stellen. Aus diesem Grund wurde entschieden, dass nur Elemente, die einmal gesucht und vom Benutzer ausgewählt wurden, offline gespeichert werden. Mit diesem Prinzip ist lokal nur genau das gespeichert, was der Benutzer schon einmal gesucht hat und ihn somit interessiert. Elemente, welche lokal gespeichert wurden, können jeweils von der normalen Suche sowie von der Reverse-Suche aus verwendet bzw. gefunden werden. Dies bedeutet auch, dass eine Suche stets an zwei Orten durchgeführt werden muss: Lokal und im Internet.

11.2.1 Normale Suche (Query-Lookup)

Bei einer normalen Suche werden immer alle Ergebnisse in einer Liste dargestellt – natürlich sind in der Liste zuerst nur die lokalen Ergebnisse sichtbar, da diese Suche viel schneller als diejenige über das Internet abläuft. In folgender Abbildung sind die drei Stadien der Suche ersichtlich.

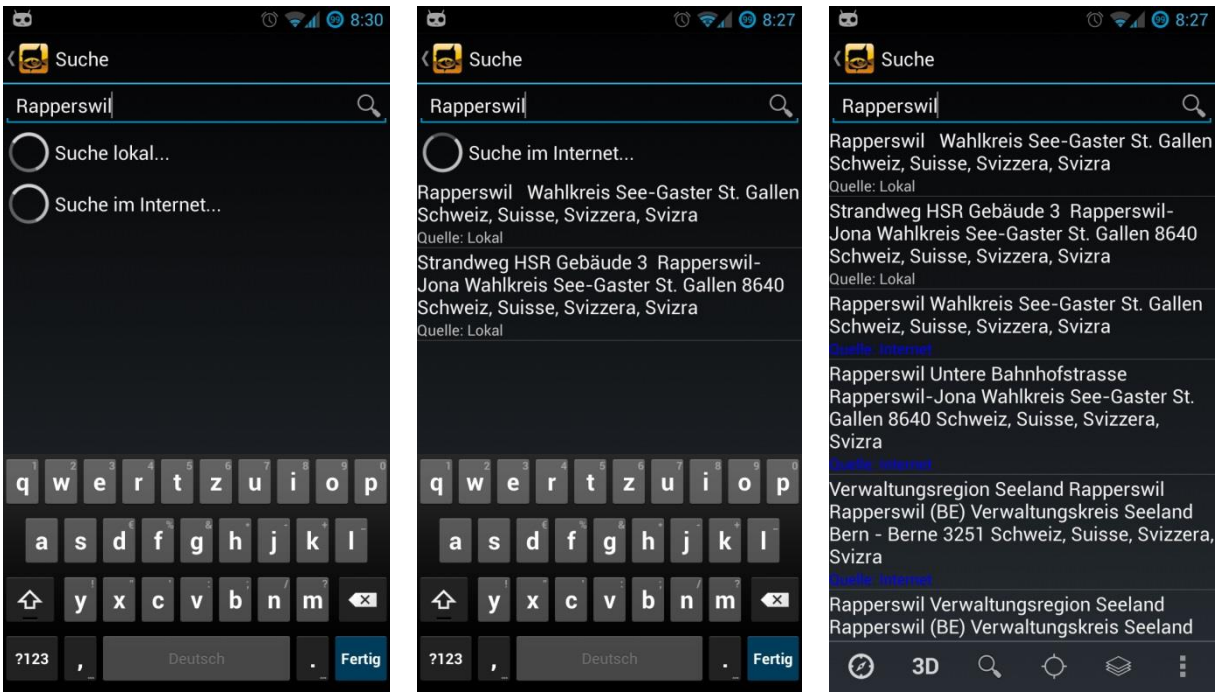


Abbildung 45: Nominatim Query-Lookup Suchstadien

Sobald ein Benutzer ein Ergebnis auswählt, d.h. antippt, so wird dieses auf der Karte angezeigt. Gleichzeitig wird geprüft ob es sich um ein Ergebnis der Internetsuche handelt, falls ja wird das Ergebnis in die lokale Datenbank eingefügt, wenn es dort noch nicht existiert.

11.2.2 Reverse Suche (Coordinate-Lookup)

Bei einem Reverse-Lookup wird immer das Ergebnis angezeigt, welches zuerst gefunden wurde. Der Grund dafür ist, dass der Benutzer sowieso nicht weiss, was er genau sucht. Denn er bittet die App im Prinzip um Hilfe und will wissen „Was ist hier denn?“ oder „Wo bin ich?“.

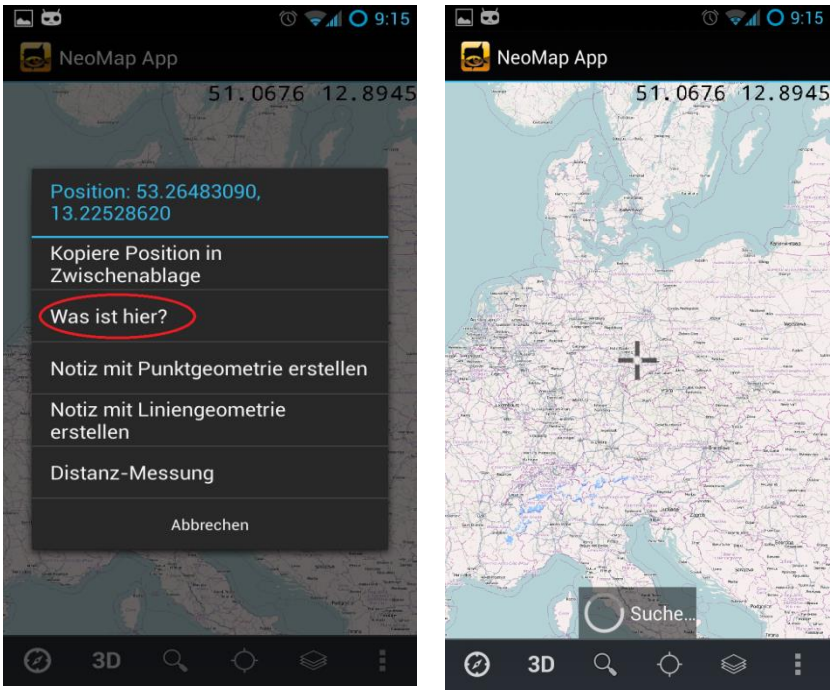


Abbildung 46: Nominatim Reverse-Lookup Suchstadien

11.2.3 Anzeige auf der Karte

Beide Sucharten zeigen ihre Ergebnisse auf der Karte genau gleich an: Die gefundene Stelle wird mit einem Icon markiert und es wird eine Toast-Nachricht mit dem Suchergebnis ausgegeben.

	Inaktiv	Aktiv
Nominatim-Suchergebnis		

Tabelle 32: Nominatim Icons

Durch erneutes, einmaliges, antippen des Icons wird der Suchergebnistext wieder ausgegeben, dies kann beliebig oft wiederholt werden. Durch doppeltes Antippen verschwindet das Suchergebnis von der Karte. Es wurde also auch hier das Bedienkonzept eingehalten (einmaliges Antippen = Information, doppeltes Antippen = Aktion). Es können beliebig viele Suchergebnisse auf der Karte vorhanden sein, diese sind allerdings nur temporär und werden entweder nach einem kompletten App-Neustart gelöscht oder falls man diese einzeln doppelt antippt. Zusätzlich können alle Suchergebnisse gleichzeitig über das „Ebenen Einstellungen“-Fragment gelöscht werden.

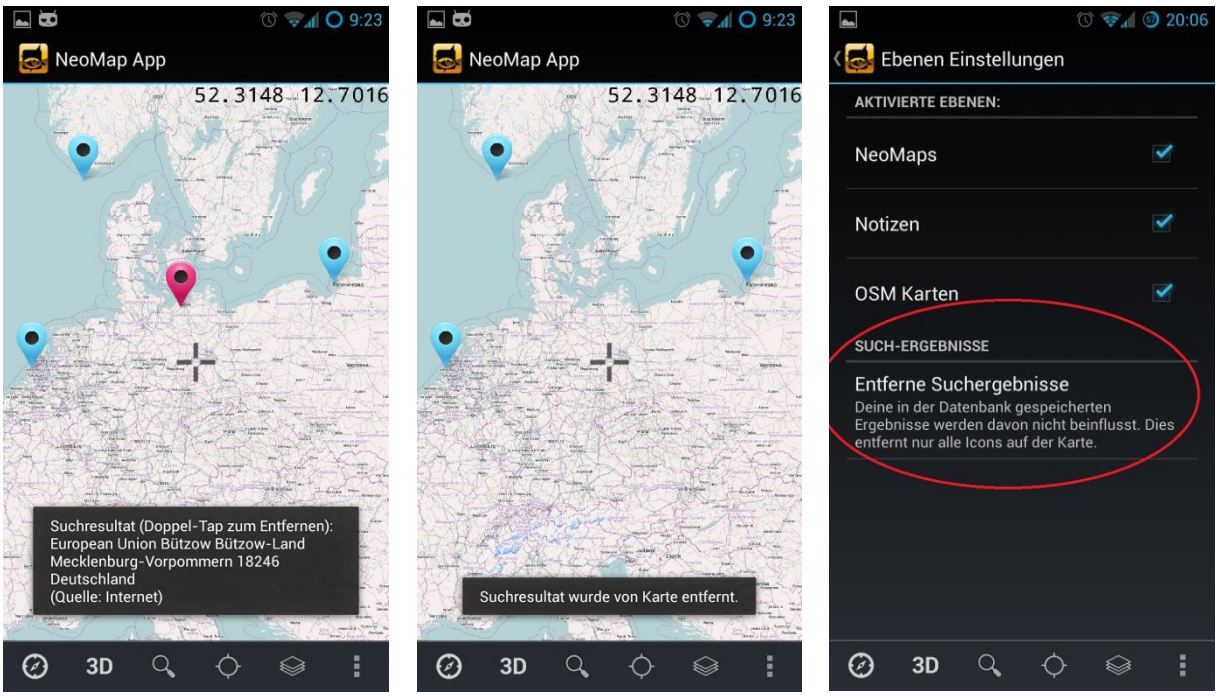


Abbildung 47: Darstellen und Löschen von Suchergebnissen auf der Karte

11.3 Umsetzung

11.3.1 Klassenbeschrieb

Auf der folgenden Seite ist ein Diagramm mit den für die geografische Namenssuche relevanten Klassen abgebildet (Achtung: Klassen die nichts direkt mit Nominatim zu tun haben, wurden weggelassen!).

Zu beachten ist auch, dass auf das Zeichnen der Nominatim-Resultate mit OpenGL gar nicht eingegangen wird, dies ist exemplarisch bereits weiter oben im Kapitel „Karten-Darstellungsgrundlagen (Rendering)“ beschrieben.

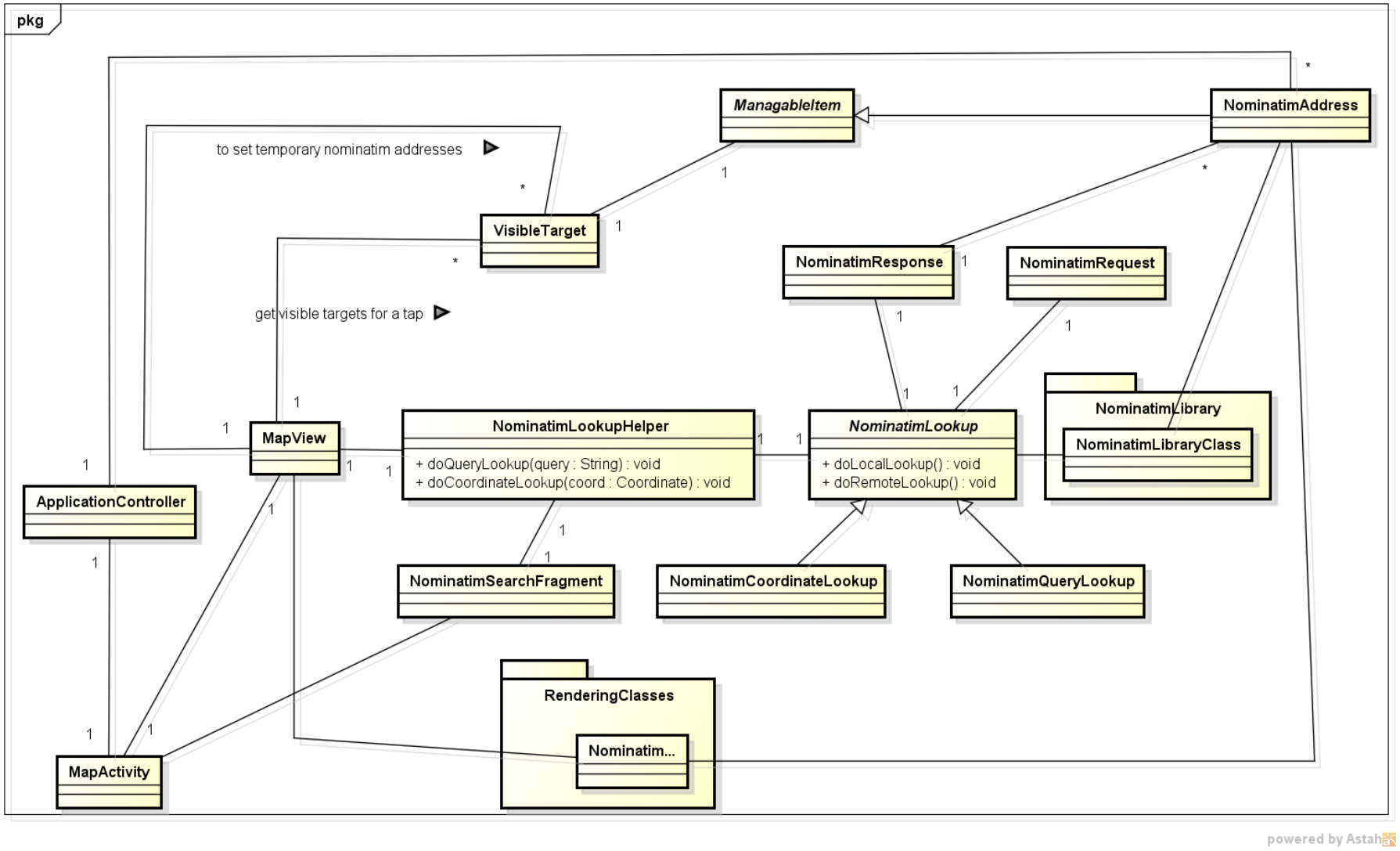


Diagramm 17: Nominatim Klassen

NominatimLookupHelper
Die Klasse NominatimLookupHelper stellt den Eintrittspunkt für das ganze Verhalten der Nominatim-Klassen dar. Sie wird entweder in der MapView (Reverse-Lookup) oder im NominatimSearchFragment (Query-Lookup) instanziiert und verwendet. Diese Klasse besitzt zwei wichtige Methoden: Eine um Reverse-Lookups zu machen und eine zweite um Query-Lookups durchzuführen. Beide Methoden erzeugen immer zwei AsyncTasks des Typs NominatimLookup: Einer für die lokale Abfrage und einer für die Internet Abfrage.

NominatimAddress
Diese Klasse ist ein Domainobjekt. Sie repräsentiert ein konkretes geografisches Element an einer bestimmten Koordinate. Gleichzeitig stellt diese Klasse ein Mapper von Objekten aus der externen Library zu lokalen Datenbankobjekten dar. Siehe auch „Umgang mit inkonsistenten Rückgabewerten“.

NominatimRequest
Dies ist lediglich eine Wrapperklasse/Datenobjektklasse für Parameter, welche für Lookups nötig sind. Sie enthält zum Beispiel einen Suchstring und die Koordinaten.

NominatimResponse
Diese Klasse enthält die Antworten der Lookups. Sie enthält immer alle Antworten eines Lookups, d.h. für den lokalen sowie denjenigen vom Internet. Mittels Convenient-Methoden können aufrufende Klassen einfach prüfen, von welchem Lookup, d.h. lokal oder remote, es bereits Ergebnisse gibt und wie diese lauten.

NominatimLookup
Dies ist eine abstrakte AsyncTask Klasse. Sie enthält das Grundgerüst, sowie die zu überschreibenden Methoden für einen Lookup. Siehe auch „Suchen“. Ihre Kindklassen setzen dann die konkreten Lookups für Query- bzw. Reverse-Lookups um. Diese Klassen werden von der Nominatim-Klasse benutzt. Je nachdem ob die Klasse für ein remote oder lokalen Lookup erstellt wurde, wird dann eine der beiden Methoden aufgerufen. Als Parameter bekommen die Lookup-Klassen immer ein existierendes NominatimResponse Objekt, in welchem Sie ihre Antworten in Form von NominatimAddress-Listen speichern.

NominatimSearchFragment
Diese Klasse stellt das User Interface für den Benutzer dar. Hier kann er Suchbegriffe eingeben, diese in einer Liste einsehen und anschliessend auswählen.

MapView
Über die MapView bzw. das LongPressMenu können Reverse-Lookups („Was ist hier?“) abgesetzt werden.

RenderingClasses
Das Paket RenderingClasses ist nur fiktiv. Anstelle dieses Paketes würden einfach alle Renderingklassen vorhanden sein. Da diese aber bei allen ManagableItems (Notes, NeoMap, Nominatim) nach dem genau gleichen Prinzip funktionieren, sind diese hier nicht beschrieben. Generelles über die Renderklassen ist im Kapitel „Karten-Darstellungsgrundlagen (Rendering)“ vorhanden.
Die Nominatim-Renderingklassen haben einen grossen Unterschied zu allen anderen Klassen: Sie zeigen nur temporäre (d.h. vom Benutzer ausgewählte) Objekte an und greifen damit auch nie auf die Datenbank zu.

ApplicationController
Der ApplicationController ist ein Singleton-Objekt. Er überlebt somit auch bei einer Rotation der App (Hochformat → Querformat) und enthält eine Liste mit verschiedenen temporären Objekten. Unter anderem enthält er eine Liste mit allen, vom Benutzer ausgewählten, Suchergebnissen. Dies sind NominatimAddresses welche auf der Karte sichtbar sind.

11.3.2 Absetzen der Internetanfragen

Damit die App sich nicht selber um das Handling von Internetanfragen etc. kümmern muss, wurde für die Internet-Suche die offizielle Nominatim Java Library verwendet. Diese erleichtert das Abfragen des Nominatim-Webservices stark, da nur noch die Suchbefehle abgesetzt werden müssen, der Rest macht die Library.

11.3.3 Umgang mit inkonsistenten Rückgabewerten

Dass der Nominatim-Service nur eine REST-Schnittstelle zur Verfügung stellt, bietet einen entscheidenden Nachteil: Es kann nicht garantiert werden, was eine Suche für Ergebnisse liefert. D.h. je nach Suchbegriff oder Koordinate kommen verschiedene Felder zurück, so muss zum Beispiel nicht jede Suche ein Feld „Ort“ oder „Strasse“ enthalten. Es ist alles sehr dynamisch. Dies stellt ein grosses Problem dar. Denn es ist somit nicht klar wie genau ein Objekt, welches die Suchergebnisse enthält, aussehen muss (die Library liefert nur eine Map mit Key-Value Paaren – ist also keine grosse Hilfe!). Aus diesem Grund wurde eine Klasse definiert, welche zum einen einfach Standardfelder, wie z.B. Strasse, Ort, Land usw. enthält und zum andern enthält sie ein Feld „various“ vom Typ JSONObject. In das JSONObject werden alle Key-Value Paare gespeichert, welche keinem anderen Member-Field zugeordnet werden können. Dadurch ist sichergestellt, dass beim Speichern von Adressdaten keine Informationen verloren gehen und diese sogar noch strukturiert (in Form eines JSONObjects) gespeichert sind.

11.3.4 Suchen

Damit die Suche möglichst schnell abläuft, werden pro Suchanfrage immer zwei AsyncTasks (asynchrone Hintergrundthreads) parallel gestartet. Einer davon sucht lokal und der andere im Internet. Beiden wird jeweils ein Callback Objekt mitgegeben, dies ist entweder das NominatimSearchFragment (Query-Lookup) oder die MapView (Reverse-Lookup) – natürlich beides Mal in Form des „IonTaskCompleted“-Interfaces. Sobald ein AsyncTask seine Suche abgeschlossen hat, macht er das Callback und teilt seine Ergebnisse in Form einer NominatimResponse mit.

11.3.4.1 Normale Suche

Die normale Suche ist relativ simpel. Für den Internet Lookup wird einfach der vom Benutzer eingegebene Suchbegriff der Nominatim-Library übergeben, diese macht den Rest. Für die lokale Suche wird der Suchbegriff in einzelne Wörter aufgesplittet (der Delimiter ist dabei einfach ein Abstand). Die einzelnen Suchbegriffe werden dann mit „AND“ verknüpft und es wird mit ihnen in der Nominatim-View im Feld „full_text_search“ gesucht.
Wird also zum Beispiel nach „HSR Rapperswil Schweiz“ gesucht, so würde das entsprechende Query so aussehen:

```
SELECT * FROM nominatim_search_view WHERE full_text_search LIKE '%HSR%' AND
full_text_search LIKE '%Rapperswil%' AND full_text_search LIKE '%Schweiz%'
```

Quellcode 10: SQL-Query bei der Suche nach Namen in der geografischen Namenssuche

Gross und Kleinschreibung wird wegen des “LIKE”-Keywords auch ignoriert.
Durch diese Suche ist sichergestellt, dass in den meistens Fällen auch lokal etwas gefunden wird. Völlig klar ist, dass diese Art von Suche bei einer grossen Anzahl von Datensätzen langsam werden wird. Jedoch sollte die Anzahl der Datensätze nie allzu gross werden, da ja immer nur Ergebnisse lokal gespeichert sind, welche der Benutzer schon einmal gesucht und ausgewählt hat.

11.3.4.2 Reverse Suche

Die reverse Suche ist etwas komplexer:
Als erstes gilt: Es wird immer nur ein Ergebnis angezeigt und zwar immer das, welches zuerst gefunden wurde.
Die Internetabfrage ist sehr simpel, es werden lediglich die Koordinaten und die Zoomstufe mitgegeben. Anhand von diesen Informationen liefert der Nominatim-Webservice eine Antwort zurück. Die Zoomstufe stellt dabei eine Art Detailgrad dar, je kleiner die Zoomstufe, desto weniger detailliert ist ein Ergebnis.
Beispiel: Ein Reverse Lookup an den Koordinaten 47.2237, 8.81675 auf Zoomstufe 18 (= maximale Nähe) ergibt als Ergebnis „HSR Gebäude 5 [….]“. Für die gleichen Koordinaten ergibt sich auf Zoomstufe 2 (= am weitesten entfernt) das Ergebnis „Schweiz“.
Dies ist natürlich sinnvoll, denn je weiter man entfernt ist, desto schwerer ist es den gewünschten Punkt exakt zu treffen. Auch ist man in diesem Falle meist nicht daran interessiert, eine detaillierte Information, wie einen Strassennamen zu erhalten, sondern interessiert sich für die aktuelle Region.
Die Schwierigkeit die sich nun stellt, ist es, dass ganze lokal auch so zu umzusetzen. Dazu wurde unter anderem der offizielle Quellcode des Nominatim-Webservices analysiert. Aus dieser Analyse ergab sich jedoch, dass so ein Suchalgorithmus im Rahmen dieser Arbeit gar nicht möglich ist, da einerseits eine spezielle GIS Datenbank verwendet wird und andererseits eine sehr komplexe Abfrage dahinter steckt.
Aus diesem Grund wird die Reverse-Abfrage lokal wie folgt durchgeführt:
Falls eine Internetverbindung besteht, so wird die lokale Datenbankabfrage lediglich mit einem Delta von 0.0001 Grad (Latitude, sowie Longitude) und auf der aktuellen Zoomstufe durchgeführt. Dies hat den Grund, dass Ergebnisse vom Internet in allen Fällen, auf Grund des Algorithmus und natürlich weil im Internet alle möglichen Daten vorhanden sind, viel besser ist. Durch dieses sehr kleine Delta wird also nur ein Ergebnis in der lokalen Datenbank gefunden, wenn man Punkt-Genau eine in der Datenbank vorhanden Koordinate angetippt hat.
Ist jedoch keine Internetverbindung vorhanden, so wird lokal erstens ein viel grösseres Delta verwendet (dieses Delta ist das Gleiche, welches beim VisibleTarget gebraucht wird und errechnet sich aus der Zoomstufe, der Bildschirmgrösse, sowie der Fingergrösse. Siehe auch „Visible Target“) und zweitens werden alle Zoomstufen durchgegangen: Zuerst werden nur Datensätze mit der aktuellen Zoomstufe gesucht, dann alle Datensätze mit höhere Zoomstufe (d.h. weniger detailliert) und anschliessend tieferen Zoomstufen (dazu wurde eine Iteratorklasse geschrieben). Dieses Verfahren ist eine starke Vereinfachung des Algorithmus, welcher das offizielle Nominatim API benutzt. Dadurch ist sichergestellt, dass ohne Internet Verbindung in vielen Fällen ein Ergebnis geliefert werden kann. Der Benutzer kann sogar sehr weit neben eine Koordinate drücken und es wird oft trotzdem ein Ergebnis gefunden – immer voraus gesetzt es sind überhaupt Daten lokal verfügbar.

11.4 Persistenz

Im folgenden Diagramm sind die Datenbanktabelle, sowie die dazu gehörige View der geografischen Namenssuche ersichtlich.

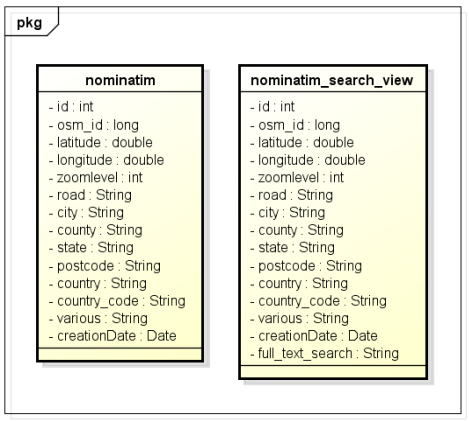


Diagramm 18: Nominatim Datenmodell

In der „nominatim“ Tabelle ist ersichtlich, dass wie oben bereits erwähnt, viele Basisinformationen, wie Strasse, Stadt, Postleitzahl etc. ein eigenes Feld haben. Zusätzlich existiert jedoch das Feld „various“ welches alle Werte enthält, die keinem Feld zugeordnet werden konnten, wie z.B. „university => HSR“ oder „footway => Strandweg“ usw. Die Werte in diesem Feld sind als JSONObject gespeichert. Auf diese Art und Weise können sie strukturiert, als Key-Value Paare, ausgelesen, abgeändert und wiederum gespeichert werden. Wegen dieser dynamischen JSONObjects ist auch gut ersichtlich, dass eine NoSQL-Datenbank hier vermutlich viel besser geeignet wäre, dort könnte sogar das ganze Ergebnis als ein einziges JSONObject gespeichert werden. Im Rahmen dieser Bachelorarbeit war es jedoch nicht möglich, sich mit dem Thema auseinander zu setzen.

Fast alle Felder können auch „NULL“-Werte enthalten. Dies ist wiederum auf die Resultate der Nominatim-API zurückzuführen: Denn dort kann es oft sein, dass keine Strasse oder keine Stadt in der Antwort vorhanden ist.

Erwähnenswert ist auch, dass fast alle Felder vom Typ „String“ sind, auch die Postleitzahl stellt keine Ausnahme dar. Dies hat den einfachen Grund, dass das offizielle Nominatim-API nicht garantieren kann, dass bei der Postleitzahl in jedem Fall eine Zahl zurückkommen muss.

Das Feld „osm_id“ enthält zudem die offizielle OSM-Node-ID, welche für jedes geografische Objekt eindeutig ist. Diese ID wird auch zum Vergleich benutzt, wenn ein Benutzer ein Resultat aus der Liste auswählt bzw. ein Resultat über die Reverse-Suche gefunden wird. Bevor der neue Wert in die Datenbank eingefügt wird, wird seine eigene OSM_ID mit der „osm_id“ in der Datenbank verglichen – ist diese vorhanden, so muss das Resultat nicht eingefügt werden.

Die „nominatim_search_view“ ist genau gleich aufgebaut wie die Tabelle. Der einzige Unterschied ist das Feld „full_text_search“. Dieses wird, wie der Name schon sagt, zum Suchen verwendet. Es enthält (fast) alle Felder der „nominatim“ Tabelle als einen einzigen String, getrennt mit Abständen. Dadurch muss bei einer Suche nur dieses Feld durchsucht werden und kein anderes. Dies vereinfacht den Ablauf enorm. Die Arbeit zu überprüfen „Was hat der Benutzer überhaupt eingegeben?“ – „War es eine Adresse?“ „Kam die Strasse zuerst?“ „Oder doch das Land?“ entfällt komplett. Es kann einfach nur das Feld durchsucht werden.
Natürlich ist klar, dass bei vielen Datensätzen die Suche extrem langsam werden wird. Jedoch wird die Tabelle vermutlich nie so stark gefüllt werden, dass dies auffällt. Denn es werden nur Datensätze gespeichert, welche der Benutzer auch wirklich zuvor ausgewählt hat bzw. über die Reverse-Suche gefunden hat.

12 News

12.1 Konzept

News können auf der NeoMap Webseite erfasst werden. Neue News sollen dem Benutzer, falls gewünscht, beim Öffnen der App präsentiert werden. Zusätzlich soll der Benutzer jederzeit die Möglichkeit haben, heruntergeladene News anzuzeigen.

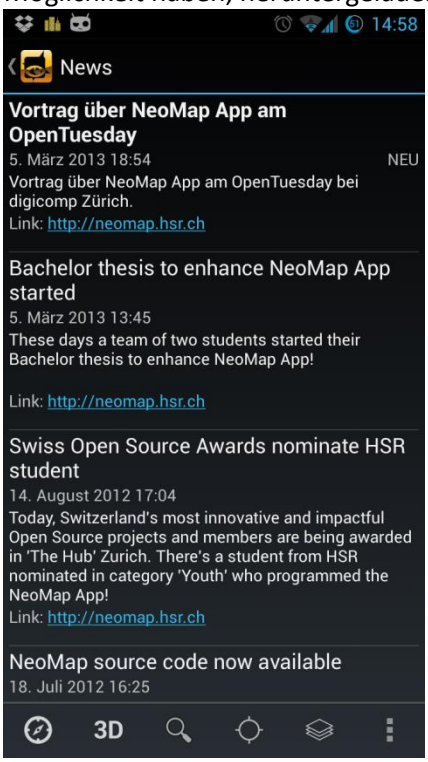


Abbildung 48: Newsanzeige

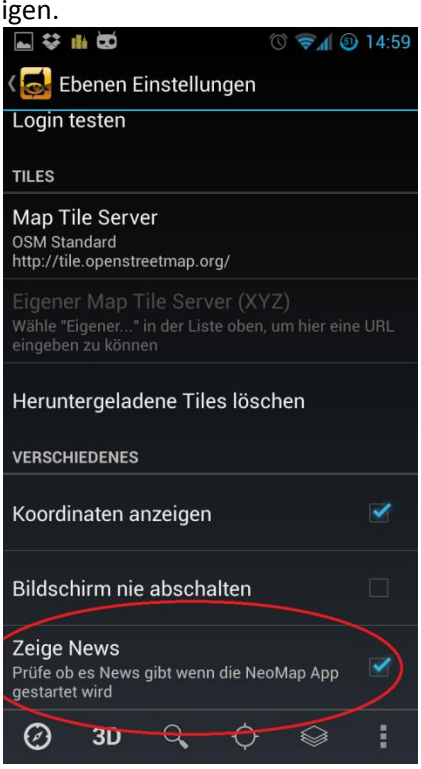


Abbildung 49: News Einstellungen

12.2 Umsetzung

Die Umsetzung ist sehr simpel. Mithilfe eines AsyncTasks (RSSNewsFeedReader) wird direkt beim Start des NeoMap Apps geprüft, ob es neue News gibt oder nicht. Dazu wird das RSS-Feed vom Link http://neomap.hsr.ch/en/news/rss/ verwendet. Mit Hilfe der externen Library „ROME“ wird eine Abfrage auf dieses RSS Feed gemacht. Anschliessend werden alle Ergebnisse durchgegangen und geprüft ob diese bereits in der Datenbank vorhanden sind. Als Vergleichsoperator wird dafür der Titel, sowie das Datum verwendet. Falls neue News vorhanden sind, wird direkt das RssNewsFragment geöffnet und dem Benutzer angezeigt.
Der RSSNewsFeedReader wird nur ausgeführt, sofern der Benutzer in den Einstellungen angegeben hat, dass auf News geprüft werden soll.
Öffnet der Benutzer die News manuell, d.h. über das Menü, so wird genau der gleiche AsyncTask wie oben beschrieben ausgeführt, der Unterschied ist jedoch, dass der Benutzer die älteren News bereits sieht, währenddem die neuen heruntergeladen werden.

12.2.1 Probleme

Das Newsfeed ist momentan nur in einer Sprache vorhanden. Meistens ist das Englisch, jedoch ab und zu auch Deutsch (siehe Abbildung oben). Ein zweisprachiges Newsfeed müsste zuerst auf der Webseite eingeführt und anschliessend das App entsprechend angepasst werden.
Ausserdem scheint der Perma-Link des Newsfeeds momentan immer nur auf http://neomap.hsr.ch zu zeigen, anstatt direkt auf den entsprechenden Artikel, dies muss auch auf der Webseite angepasst werden und sollte dann aber ohne Codeänderungen in der App funktionieren.

13 Testing

13.1 Unit Testing

Um die Qualität und Funktionalität von zentralen Klassen zu gewähren, werden Unit-Tests verwendet. UI-Elemente und Interaktionen mit dem Benutzer werden jedoch nicht direkt getestet. Der Fokus liegt hierbei auf den zentralen Elementen der Applikation, welche isoliert getestet werden. Im Rahmen dieser Arbeit wurden nur die wichtigsten Klassen getestet. Es besteht daher definitiv Potential gegen oben für eine grössere und bessere Testabdeckung (siehe auch Kapitel „15.2.4 Testing“).

Insgesamt existieren im Projekt 84 Unit Tests. Diese befassen sich vor allem damit Domain-Objekte bzw. deren Methoden zu testen. So wird zum Beispiel eine Koordinate auf ihre Gültigkeit getestet oder ob sich zwei BoundingBoxes berühren.

Wie bereits erwähnt, werden UI-Elemente nur sehr wenig getestet. Da in der NeoMap App jedoch sehr viel über das UI abläuft (das ganze Zeichnen mit OpenGL usw.) fehlen für viele Teile automatisiere Tests. Um diese Code-Teile zu testen, müsste ein Framework benutzt werden. Leider fehlte es uns an Zeit ein solches zu evaluieren. Es wurden dafür aber Systemtests durchgeführt, diese sind im Kapitel „13.2 Systemtests“ ausführlich beschrieben.

13.1.1 Mockito

Um Unit Tests ausführen zu können, wird als integrierter Teil des Android SDKs ein Android Testing Framework mitgeliefert. Das Android Testing Framework basiert auf JUnit und liefert zusätzlich spezifische Test-Klassen, um die Android-Komponenten zu testen. Obwohl dadurch bereits die Möglichkeit geboten wird, sogenannte Stub- und Mock-Objekte für isolierte Unit-Tests zu erstellen, gibt es hierfür bereits noch bessere und attraktivere Bibliotheken auf dem Markt, welche dieses Vorgehen nochmals um einiges erleichtern.
Das wohl zurzeit am besten etablierte Mocking-Framework, welches auch eine stabile Version für die Android Plattform zur Verfügung stellt, ist hierbei Mockito. (http://code.google.com/p/mockito/)

Die Einbindung von Mockito im Projekt erfolgt über das verlinken der Library (jar-File). Zusätzlich wird noch die Library Dexmaker benötigt, welche das Kompilieren von Java-Klassen für die Android-Plattform (DalvikVM) ermöglicht.
Beide dieser „.jar-Files“ können, wie jede andere Bibliothek, über den „/libs/“ Ordner ins Projekt eingebunden werden.

Mockito bietet unter anderem folgende Vorteile:

- Viel bessere Lesbarkeit im Vergleich zu Unit Tests des Android-Frameworks
- Es können Klassen und nicht nur Interfaces gemockt/gefaket werden
- Es kann sehr einfach angegeben werden, was für Output von gefakten Objekten bei gewissen Parametern erhalten werden soll, ohne das die ganze Klasse extra neu geschrieben werden muss. Um diesen Punkt zu verstehen ein einfaches Beispiel:

  ```
  doReturn(1024).when(cacheController).getAvailableMemoryMb();
  ```

  Dieses Beispiel bezieht sich auf ein Fake eines „CacheControllers“ und gibt an, dass immer wenn die Methode „getAvailableMemoryMb()“ aufgerufen wird, „1024“ zurück gegeben wird. Mit diesem Test muss sich nicht auf die eigentliche Cache-Grösse (welche vom verwendeten Gerät abhängig ist) verlassen werden, sondern es wird immer der gleiche Wert returniert. Dadurch wird der Test überhaupt erst wiederholbar.

13.2 Systemtests

Mit den Systemtests sollen komplexere Szenarien bzw. User Szenarien getestet werden, welche nicht mit Unit Tests abgedeckt werden können. Es soll ausserdem auch aufgezeigt werden, was das System kann bzw. was für eine Funktionalität es bietet. Gerade bei der NeoMap App sind diese Tests relativ wichtig, da es sehr viele GUI-Elemente besitzt, die nur schwer bzw. gar nicht mit automatisierten Tests abzudecken sind. Damit der Umfang dieses Dokumentes nicht gesprengt wird, sind einige Tests etwas reduziert worden, es werden meist nur die wichtigsten Funktionen getestet und relativ kurz und knapp beschrieben.
Die Tests sind alle in Tabellen aufgeführt, dabei wird jeweils angegeben, was getestet wird bzw. welche Aktion ausgeführt wird und was das erwartete Ergebnis ist. In der letzten Spalte wird angegeben, ob der Test in der NeoMap App das erwartete Ergebnis erbracht hat.
Zusätzlich kann die erste Spalte jeweils ein Icon enthalten, welches anzeigt, ob eine Internetverbindung für den Test nötig ist oder nicht.

Abbildung 50: Online Icon

Abbildung 51: Offline Icon

Abbildung 52: Test erfolgreich Icon

Abbildung 53: Test nicht erfolgreich Icon

13.2.1 Testumgebung

Alle Tests wurden, falls nicht anders erwähnt, auf den folgenden Testgeräten durchgeführt:

- Samsung Galaxy S3, I9300
- Galaxy Nexus S
- Asus Nexus 7

13.2.2 Grundfunktionalität

In diesem Kapitel soll die Grundfunktionalität der App getestet werden. Also das „zu Recht“ finden in der OSM Karte und das damit verbundene Zoomen und Scrollen, sowie die Basisfunktionalität wie Kompass und eigener Standort.

13.2.2.1 Abgedeckte User Szenarien

- Keine

13.2.2.2 Einfache Kartenanzeige

Vorbedingungen:

- App erfolgreich neu installiert, noch nie geöffnet

	Aktion	Erwartetes Ergebnis	Status
	Erstes Starten der NeoMap App	App gestartet, zentriert auf Rapperswil, HSR, 47.2238, 8.8158	
	-	Kartenausschnitt wird heruntergeladen	
	App schliessen	App geschlossen	
	Internetverbindung deaktivieren und App öffnen	Karte von Rapperswil wird angezeigt (ist also offline verfügbar)	

13.2.2.3 Zoomen und Scrollen

Vorbedingungen:

- 13.2.2.2 Einfache Kartenanzeige

	Aktion	Erwartetes Ergebnis	Status
	11 Mal auf den ZoomIn Button drücken	Maximale Detailstufe erreicht, ZoomIn Button ist deaktiviert, Obere Ecke HSR Gebäude 6 ist fokussiert	
	16 Mal auf den ZoomOut Button drücken	Minimale Detailstufe erreicht, ZoomOut Button deaktiviert, ganze Welt sichtbar, Europa im Zentrum	
	PinchOut (Finger spreizen) bis nächste Zoomstufe erreicht	Nächste Zoomstufe erreicht, etwas mehr Details	
	PinchIn (Finger zusammen ziehen) bis nächste Zoomstufe erreicht	Nächste Zoomstufe erreicht, etwas weniger Details, minimale Detailstufe erreicht, ZoomOut Button deaktiviert	
	Double-Tap aufs Fadenkreuz	Nächste Zoomstufe erreicht, etwas mehr Details	
	Finger gegen oben bewegen	Karte verschiebt sich von unten nach oben	
	Finger gegen unten bewegen	Karte verschiebt sich von oben nach unten	
	Finger gegen links bewegen	Karte verschiebt sich von links nach rechts	
	Finger gegen rechts bewegen	Karte verschiebt sich von rechts nach links	

13.2.2.4 Kompass

Vorbedingungen:

- 13.2.2.2 Einfache Kartenanzeige

	Aktion	Erwartetes Ergebnis	Status
	Kompass einschalten mittels Klick auf das Icon 	Kompass ist eingeschaltet, Kompass Icon ist aktiv 	
	Ablegen des Gerätes auf eine flache Oberfläche, langsames Drehen des Gerätes in alle Richtungen	Karte sollte sich automatisch immer nach Norden ausrichten	
	Kompass ausschalten mittels Klick auf das Icon 	Kompass ist ausgeschaltet, Kompass ist inaktiv  , Karte dreht sich nicht mehr mit	

13.2.2.5 Standort

Vorbedingungen:

- 13.2.2.2 Einfache Kartenanzeige

	Aktion	Erwartetes Ergebnis	Status
	In den Android-Einstellungen den Standortzugriff nur auf „Standort per WLAN und Mobilfunknetz“ stellen (GPS deaktivieren)	-	
	NeoMap App öffnen und „Eigener Standort“ mittels Klick auf das Icon einschalten	Dialog erscheint, der fragt, ob man in die Einstellungen wechseln und „GPS“ aktvieren will	
	Dialog mit „Abbrechen“ bestätigen	Eigener Standort wird lokalisiert, und Karte an dieser Stelle zentriert und mittels dem Icon angezeigt, Genauigkeit abhängig vom WLAN, sowie Mobilfunknetz, Genauigkeit sollte jedoch +- 100 Meter betragen	
	„Eigener Standort“ deaktivieren mittels Klick auf das Icon	„Eigener Standort“ wird nicht mehr angezeigt	
	In den Android-Einstellungen den „GPS-Satelliten“ aktivieren	-	
	NeoMap App öffnen und „Eigener Standort“ mittels Klick auf das Icon einschalten (Achtung: Man sollte sich für diesen Test unter freiem Himmel befinden)	Eigener Standort wird lokalisiert, und Karte an dieser Stelle zentriert und mittels dem Icon angezeigt, Genauigkeit sollte sehr gut sein, da per GPS lokalisiert	
	„Eigener Standort“ deaktivieren mittels Klick auf das Icon	„Eigener Standort“ wird nicht mehr angezeigt, Icon ist	
	In den Android-Einstellungen jegliche Standort-Bestimmung deaktivieren	-	
	NeoMap App öffnen und „Eigener Standort“ mittels Klick auf das Icon einschalten	Dialog erscheint, der fragt, ob man in die Einstellungen wechseln und den Standort aktivieren will	
	Dialog mit „Abbrechen“ bestätigen	Eigener Standort kann nicht mehr lokalisiert werden	

13.2.3 3D

13.2.3.1 Abgedeckte User Szenarien

- 3.2.4 US 4: Orientierung finden

13.2.3.2 Zoomen und Fortbewegen

Vorbedingungen:

- 13.2.2.2 Einfache Kartenanzeige

	Aktion	Erwartetes Ergebnis	Status
	Gerät flach auf eine gerade Unterlage legen	-	
	3D mittels Klick auf das Icon 3D einschalten	3D ist eingeschaltet, 3D Icon ist aktiv 3D, Kartenansicht unterscheidet sich nicht von normaler Ansicht	
	Gerät langsam aufheben und Bildschirm gegen vorne und hinten Kippen	Sicht auf Karte sollte sich ändern. Je mehr man das Gerät gegen hinten kippt, desto mehr von „oben“ sollte Sicht auf die Karte sein. Kippt man das Gerät nach vorne so wird die Sicht immer mehr „horizontal“	
	Arme bzw. Hände nach links oder rechts bewegen	Ansicht auf Karte ändert sich. Bei einer Bewegung nach links, sieht man auch die Karte die sich links vom Ausgangspunkt der Drehung befindet	
	PinchOut (Finger spreizen)	Karte kommt näher bzw. man ist näher beim Boden	
	PinchIn (Finger zusammen ziehen)	Karte ist weiter weg bzw. man ist weiter in der Luft	
	Finger gegen oben bewegen	Karte verschiebt sich Richtung Horizont, man sieht das, was vorher hinter dem Sichtpunkt lag	
	Finger gegen unten bewegen	Karte verschiebt sich gegen Hinten, man sieht das, was vorher nahe beim Horizont lag, jetzt besser bzw. näher	
	Finger gegen links bewegen	Karte verschiebt sich von links nach rechts	
	Finger gegen rechts bewegen	Karte verschiebt sich von rechts nach links	
	„Eigener Standort“ einschalten (sicher stellen, dass „Standortbestimmung“ in den Android Einstellungen aktiviert ist)	Der eigene Standort sollte nun so lokalisiert werden, dass man genau von diesem aus in alle Richtungen blicken kann, d.h. beim Drehen des Handys sollte man seine Umgebung sehen und sich so orientieren können	
	3D mittels Klick auf das Icon 3D deaktivieren	3D ist deaktiviert, Kartenansicht ist wieder normal, 3D Icon ist wieder 3D	

13.2.4 NeoMap

13.2.4.1 Abgedeckte User Szenarien

Vollständig abgedeckt:

- US 5: Eigene Karten priorisieren und ein-/ausblenden

13.2.4.2 NeoMaps herunterladen und anzeigen

Vorbedingungen:

- 13.2.2.2 Einfache Kartenanzeige

	Aktion	Erwartetes Ergebnis	Status
	Menüpunkt „NeoMaps herunterladen“ aufrufen	Liste mit verfügbaren NeoMaps	
	NeoMap „Rapperswil“ auswählen und herunterladen	NeoMap „Rapperswil“ wird heruntergeladen und als „Installiert“ angezeigt	
	Internet deaktivieren. Zurück auf die Karte wechseln und sich nach Rapperswil begeben	NeoMap von „Rapperswil“ wird angezeigt	
	NeoMap „Rapperswil“ auf der Karte einmal antippen (Single-Tap)	Toast-Ausgabe: *NeoMap:* *Rapperswil*	
	NeoMap „Rapperswil“ auf der Karte doppelt antippen (Double-Tap)	Bewirkt Zoom (da nur EIN NeoMap vorhanden)	

13.2.4.3 NeoMaps Priorisieren

Vorbedingungen:

- 13.2.2.2 Einfache Kartenanzeige
- 13.2.4.2 NeoMaps herunterladen und anzeigen

	Aktion	Erwartetes Ergebnis	Status
	NeoMap „Rapperswil Fake Small“ und NeoMap “Rapperswil Fake Big” herunterladen (oder andere NeoMaps die einander überlappen)	NeoMaps installiert	
	Internet deaktivieren. Zurück auf die Karte wechseln und sich nach Rapperswil begeben	NeoMap „Rapperswil Fake Big“ wird angezeigt („Rapperswil Fake Small“ und „Rapperswil“ nicht sichtbar)	
	NeoMap „Rapperswil Fake Big“ auf der Karte einmal antippen (Single-Tap)	Toast-Ausgabe: *NeoMap:* *Rapperswil Fake Big* *Andere NeoMaps in der Nähe (Doppel-Tap für Anzeige):* *2*	
	NeoMap „Rapperswil Fake Big“ auf der Karte doppelt antippen (Double-Tap)	„Reihenfolge ändern“ Fenster geht auf, in welchem folgende NeoMaps in der angegebenen Reihenfolge aufgelistet sind: • Rapperswil Fake Big • Rapperswil Fake Small • Rapperswil	
	Reihenfolge per „Drag & Drop“ ändern, NeoMap „Rapperswil Fake Big“ ganz ans Ende der Liste ziehen	NeoMap „Rapperswil Fake Big“ ist nun am Ende der Liste	
	Klick auf die NeoMap „Rapperswil“	Das Sichtbarkeit Status-Icon sollte sich von  auf  Icon geändert haben	
	Zurück auf die Karte wechseln	Es sollte nun die NeoMap „Rapperswil Fake Small“ über der NeoMap „Rapperswil Fake Big“ sichtbar sein, die NeoMap „Rapperswil“ sollte nicht sichtbar sein	

13.2.4.4 NeoMaps verwalten

Vorbedingungen:

- 13.2.2.2 Einfache Kartenanzeige
- 13.2.4.2 NeoMaps herunterladen und anzeigen

	Aktion	Erwartetes Ergebnis	Status
	Über das Menü die „NeoMap Verwaltung“ öffnen	„NeoMap Verwaltung“ öffnet sich, heruntergeladene NeoMaps werden angezeigt	
	Lange auf eine beliebige NeoMap drücken (welche sichtbar ist), Meldung „NeoMap auf Karte anzeigen“ kommt, diese mit „Ja“ bestätigen	NeoMap wird auf Karte angezeigt	
	Zurück in „NeoMap Verwaltung“ wechseln und eine NeoMap antippen	Dialog öffnet sich mit Eigenschaften über diese NeoMap	
	„Ausschalten“ auswählen und auf die Karte wechseln	NeoMap wird auf der Karte nicht mehr angezeigt	
	Zurück in „NeoMap Verwaltung“ wechseln und eine NeoMap antippen	Dialog öffnet sich mit Eigenschaften über diese NeoMap	
	„Löschen“ auswählen und Meldung mit „Ja“ bestätigen	NeoMap ist gelöscht	
	Beliebiges „NeoMap“ mittels der Checkbox ganz rechts auswählen	Contextual Action Mode wird gestartet, erkennbar durch eine andere Navigationsbar (siehe „Contextual Action Mode / Contextual Action Bar“)	
	Mehrere NeoMaps auswählen via Checkboxes und alle Icons (in der Leiste unten) ausprobieren	Aktionen können für mehrere NeoMaps durchgeführt werden. Aktionen haben gleiche Effekte wie bei den einzelnen Tests oben in der Tabelle	

13.2.5 Notizen

13.2.5.1 Abgedeckte User Szenarien

- 3.2.1 US 1: Notizpunkte erfassen
- 3.2.2 US 2: Linien zeichnen
- 3.2.3 US 3: Notizen exportieren

13.2.5.2 Notiz mit Punktgeometrie erstellen

Vorbedingungen:

- 13.2.2.2 Einfache Kartenanzeige

	Aktion	Erwartetes Ergebnis	Status
	Lange auf einen beliebigen Punkt auf der Karte drücken	LongPressMenu erscheint	
	Menüpunkt „Notiz mit Punktgeometrie erstellen“ auswählen	Notizbearbeitungsmaske öffnet sich, es ist bereits ein Titel vorgeben, dieser besteht aus dem aktuellen Datum, sowie der Koordinate an der vorher gedrückt wurde	
	„Abbrechen“ wählen. Im aufgehenden Dialog „Nein“ wählen	Bearbeitungsmaske immer noch sichtbar. Sie wird nicht einfach geschlossen, da es offene Änderungen gibt	
	Das Löschen-Icon antippen	Bestehender Titel wird gelöscht	
	Titel auf „Meine Notiz“ und Beschreibung auf „Meine Beschreibung“ ändern	Titel sowie Beschreibung geändert	
	„Notiz auf Karte verändern“ wählen (Tastatur muss gegebenenfalls zuerst geschlossen werden)	Kartenänderungsmodus wird gestartet, erkennbar durch eine andere Navigationsbar (siehe „Contextual Action Mode / Contextual Action Bar“)	
	Klick auf beliebige Punkte auf der Karte	Position des Notizsymboles verschiebt sich	
	Klick auf das Icon	Aktuelle Position wird rückgängig gemacht und die letzte gesetzte Position der Notiz eingenommen	
	Klick auf das Icon und im Dialog „Nein“ wählen	Erfassungsmodus wird nicht abgebrochen	
	Klick auf das Speichern Icon	Es wird zurück zur Erfassungsmaske gewechselt. Titel und Beschreibung sind immer noch gleich wie vorher eingegeben	
	Klick auf „Speichern“	Notiz wurde erfolgreich gespeichert und wird auf der Karte angezeigt	
	Einfaches Tippen (Single-Tap) auf die Notiz	Notiz wird als aktiv angezeigt und Toast-Meldung: *Notiz(Doppel-Tap zum Öffnen):* *Meine Notiz* wird angezeigt	

13.2.5.3 Notiz mit Liniengeometrie erstellen

Vorbedingungen:

- 13.2.2.2 Einfache Kartenanzeige
- 13.2.5.2 Notiz mit Punktgeometrie erstellen

Bei diesem Test ist ein grosser Teil identisch zum Test 13.2.5.2, deshalb wird auf diesen Teil nicht mehr eingegangen.

	Aktion	Erwartetes Ergebnis	Status
	Lange auf einen beliebigen Punkt auf der Karte drücken und im LongPressMenu „Notiz mit Liniengeometrie erstellen“ wählen	Kartenänderungsmodus wird gestartet, erkennbar durch eine andere Navigationsbar (siehe „Contextual Action Mode / Contextual Action Bar“)	
	Zusätzliche Linienpunkte mittels Tippen auf beliebige Punkte auf Karte setzen	Für jeden angetippten Punkt gibt es eine neue Verbindungslinie zu diesem Punkt. Zusätzliche wird jedes Mal die Distanz die die verbundenen Punkte einnehmen als Toast-Meldung ausgeben. Beispiel: *Distanz: 1103.48 m*	
	Klick auf das Icon	Letzter Punkt wird rückgängig gemacht	
	Klick auf das Icon und im Dialog „Nein“ wählen	Erfassungsmodus wird nicht abgebrochen	
	Klick auf das Speichern Icon	Es wird zurück zur Erfassungsmaske gewechselt.	
	Klick auf „Speichern“	Notiz wurde erfolgreich gespeichert und wird auf der Karte angezeigt. Zum Bespiel: 	
	Einfaches Tippen (Single-Tap) auf die Notiz	Notiz wird als aktiv angezeigt  und Toast-Meldung: *Notiz(Doppel-Tap zum Öffnen):* *Notiz Titel* *Distanz: 1130.48 m* wird angezeigt	

13.2.5.4 Notizen bearbeiten

Vorbedingungen:

- 13.2.2.2 Einfache Kartenanzeige
- 13.2.5.2 Notiz mit Punktgeometrie erstellen

	Aktion	Erwartetes Ergebnis	Status
	Beliebige Notiz auf der Karte doppelt antippen (Double-Tap)	Notizbearbeitungsmaske öffnet sich	
	Alle Funktionen testen	Gleiche Funktionalität wie bei 13.2.5.2	

13.2.5.5 Notizen verwalten

Vorbedingungen:

- 13.2.2.2 Einfache Kartenanzeige
- 13.2.5.2 Notiz mit Punktgeometrie erstellen
- 13.2.4.4 NeoMaps verwalten

Dieses Kapitel wird sehr kurz gehalten, da die Funktionalität, abgesehen von Export, identisch zu der von NeoMaps ist.

	Aktion	Erwartetes Ergebnis	Status
	Über das Menü die „Notiz Verwaltung“ öffnen	„Notiz Verwaltung“ öffnet sich, alle erfassten Notizen werden angezeigt	
	Alle Funktionen testen, welche auch bei „NeoMap verwalten“ vorhanden sind	Gleiche Funktionalität wie bei 0	

13.2.5.6 Notizen exportieren

Vorbedingungen:

- 13.2.2.2 Einfache Kartenanzeige
- 13.2.5.2 Notiz mit Punktgeometrie erstellen
- 13.2.5.3 Notiz mit Liniengeometrie erstellen

	Aktion	Erwartetes Ergebnis	Status
	Über das Menü die „Notiz Verwaltung“ öffnen	„Notiz Verwaltung“ öffnet sich, alle erfassten Notizen werden angezeigt	
	Beliebige „Notiz“ mittels der Checkbox ganz rechts auswählen	Contextual Action Mode wird gestartet, erkennbar durch eine andere Navigationsbar (siehe „Contextual Action Mode / Contextual Action Bar“)	
	Mehrere Notizen via Checkboxen auswählen	Mehrere Notizen sind ausgewählt, die Anzahl wird in der oberen Leiste angezeigt	
	Mittels dem Overflow-Menü „Export“ wählen	Ausgewählte Notizen werden als GPX-Dateien in das, in der Toast-Meldung angegebene, Verzeichnis exportiert. Beispiel: *Die Auswahl wurde erfolgreich in das folgende Verzeichnis exportiert: /storage/emualted/0/NeoMap/Export* Die exportierten Dateien können jetzt via Computer aus diesem Verzeichnis kopiert und verifiziert werden. Alle Dateien sollten ein korrektes GPX-Format enthalten. Test z.B. über www.gpsvisualizer.com möglich	
	Mittels dem Overflow-Menü „Teilen“ wählen, in der Auswahl dann z.B. E-Mail wählen	Die ausgewählten Notizen sollten als Attachement im E-Mail sichtbar sein. Auch diese Dateien sollten sich im korrekten GPX-Format befinden. E-Mail kann verschickt und empfangen werden	

13.2.6 Geografische Namenssuche

13.2.6.1 Abgedeckte User Szenarien

- 3.2.6 US 6: Optional: Suche nach geografischen Orten
- 3.2.7 US 7: Optional: Was ist hier?

13.2.6.2 Nach Namen Suchen

Vorbedingungen:

- 13.2.2.2 Einfache Kartenanzeige

	Aktion	Erwartetes Ergebnis	Status
	Das Such-Icon antippen	Suchmaske öffnet sich	
	Eingeben von „Rapperswil“ und Suche starten mittels Klick auf das Icon rechts vom Suchfeld	Es werden nur Ergebnisse mit der Quelle „Internet“ gefunden	
	Ein beliebiges Suchergebnis antippen	Suchergebnis wird auf der Karte mittels des Icons angezeigt und eine Toast-Meldung wird ausgegeben, z.B.: *Suchresultat (Doppel-Tap zum Entfernen): Rapperswil, Schweiz*	
	Suchresultat Icon doppelt antippen (Double-Tap)	Suchresultat ist nicht mehr auf Karte sichtbar	
	Nochmal das Such-Icon antippen	Suchmaske öffnet sich, vorher gesuchte Ergebnisse sind immer noch sichtbar	
	Suche nochmals starten mittels Klick auf das Icon rechts vom Suchfeld	Es werden Ergebnisse mit der Quelle Lokal (genau 1 Ergebnis) und der Quelle Internet angezeigt	
	Internet deaktivieren und Suche nochmals starten	Es werden nur Ergebnisse mit der Quelle „Lokal“ angezeigt	
	Das gefundene Suchergebnis antippen	Suchergebnis wird auf der Karte mittels des Icons angezeigt	

13.2.6.3 Koordinaten Suche

Vorbedingungen:

- 13.2.2.2 Einfache Kartenanzeige

	Aktion	Erwartetes Ergebnis	Status
	Lange auf einen beliebigen Punkt auf der Karte drücken und im LongPressMenu „Was ist hier?“ wählen	Es erscheint eine Toast-Meldung die ausgibt, was sich an dieser Stelle befindet. Die Quelle sollte „Internet“ sein und das Suchergebnis wird auf der Karte mittels des Icons  angezeigt	
	Mittels des Icons  die Layereinstellungen öffnen, dort „Entferne Suchergebnisse“ wählen	Alle Suchresultate sind von der Karte entfernt (dies ist nur ein anderer Weg, anstatt per Double-Tap)	
	Internet deaktivieren und nochmals auf die genau gleiche Stelle wie vorher lange drücken und im LongPressMenu wiederum „Was ist hier?“ wählen	Wenn die Stelle genug genau gedrückt wurde, so sollte wiederum eine Toast-Meldung mit dem Ergebnis ausgegeben und das Ergebnis auf der Karte mittels des Icons  angezeigt werden	

14 Bedienkonzept

14.1 Allgemein

Als berührte Position gilt immer der Punkt auf der Karte, der auch mit den Fingern berührt wurde und nicht etwa die Mitte der Karte wo sich das Kreuz befindet. Dieses dient lediglich dazu, dass der Benutzer jederzeit von jedem Punkt die Koordinaten Anzeige hat, ohne extra das LongPressMenu zu öffnen zu müssen.

14.2 Einmaliges Tippen (Single-Tap)

Bei einem Single-Tap wird dem Benutzer angezeigt, was sich an dieser Stelle auf der Karte befindet. Er liefert also nur Informationen („Read-Only“), ohne konkrete Aktionen vom Benutzer anzufordern. Bei einem Single-Tap wird dem Benutzer zudem immer angezeigt, was ein Doppel-Tap auf die gleiche Stelle für eine Wirkung hat (falls es denn einen geben sollte).
Im Beispiel von Notizen wird der Titel der Notiz angezeigt.
Ein Spezialfall ist hierbei der Kartenänderungsmodus. Bei diesem bewirkt ein Tap das Setzen oder das Verschieben eines Punktes. Dieser Modus ist auch an der Contextual Action Bar zu erkennen. Siehe auch „Contextual Action Mode / Contextual Action Bar“.

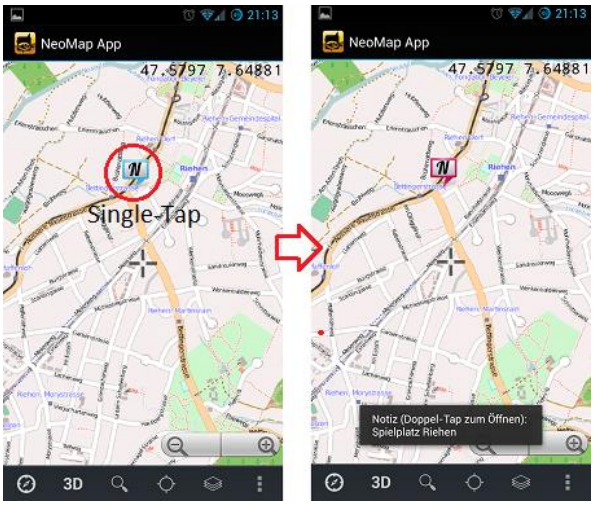


Abbildung 54: Wirkung eines Single-Taps auf der Karte (auf eine Notiz)

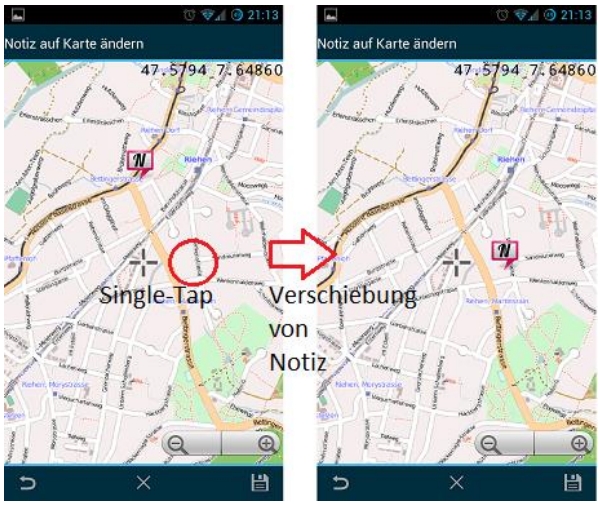


Abbildung 55: Wirkung eines Single-Taps auf der Karte während des Kartenänderungsmodus

14.3 Doppeltes Tippen (Double-Tap)

Führt der Benutzer einen Double-Tap an einer Stelle aus, an welcher sich ein Element befindet, so wird eine Aktion durchgeführt und es „ändert“ sich effektiv etwas (z.B. geht ein Fragment-Fenster auf). Bei Notizen öffnet sich die Bearbeitungsmaske (NoteEditFragment) dieser Notiz – um wieder ein Beispiel zu nennen.
Befindet sich an der Stelle nichts, so bewirkt der Double-Tap ein Hineinzoomen und Zentrieren an dieser Stelle. Während der Kartenänderungsmodus aktiv ist, bewirkt ein Double-Tap immer ein Zoomen. Dies hat den Grund, dass es während des Änderungsmodus nur Verwirrung stiften würde, falls andere Ziele angeklickt werden könnten.

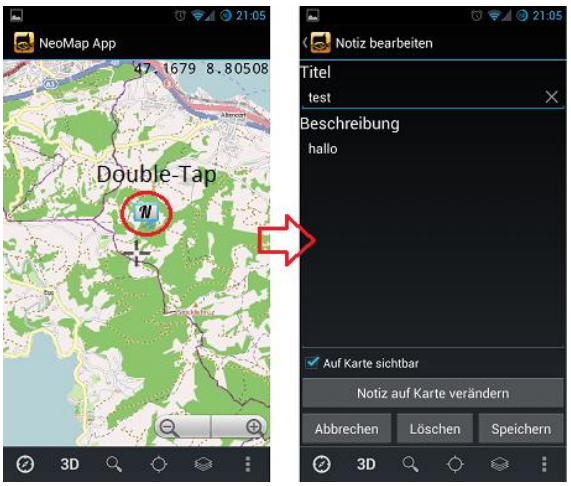


Abbildung 56: Wirkung eines Double-Taps auf der Karte (auf eine Notiz)

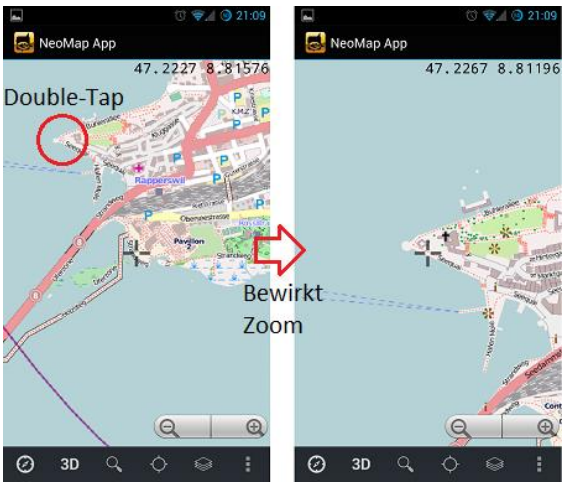


Abbildung 57: Wirkung eines Double-Taps auf der Karte (auf nichts)

14.4 Langes Drücken (LongPress)

Ein langes Drücken bewirkt das Öffnen eines Menüs (LongPressMenu), welches verschiedene Aktionen zum ausgewählten Punkt anzeigt. Im Kartenänderungsmodus wird das Menü nicht geöffnet.

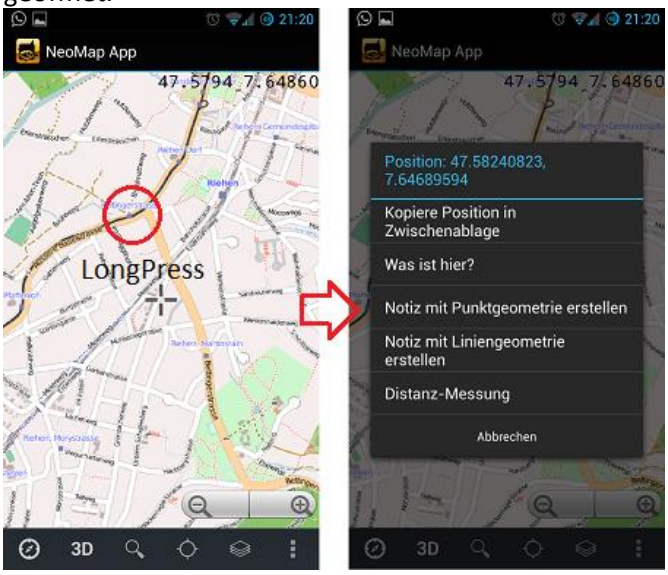


Abbildung 58: Wirkung eines LongPress auf der Karte

14.5 Navigation in der App

Um in der NeoMap App zu navigieren, wird fast ausschliesslich die „Action Bar“ verwendet. Die wichtigsten Aktionen werden mittels eines Icons dargestellt. Alle weiteren Navigationsmöglichkeiten können über das Overflow Menu aufgerufen werden.
Bei Smartphones, welche noch einen eigenen Button für das „Menü“ haben, wird der Overflow Menu Button standardmässig nicht angezeigt. Bei diesen muss die Menütaste gedrückt werden, um das Menü anzuzeigen.

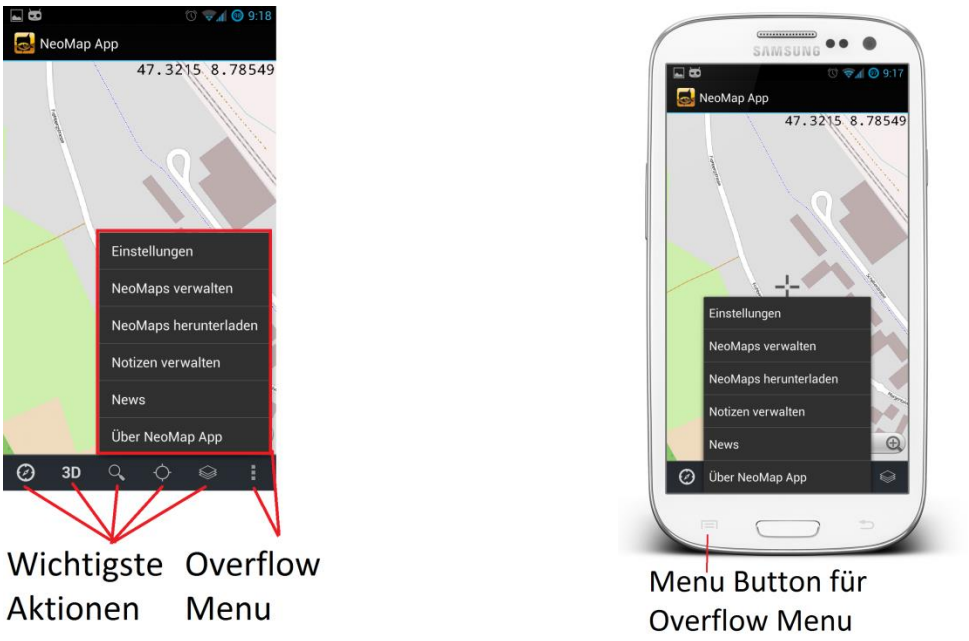


Abbildung 59: Overflow Menu

Befindet man sich nicht in der Kartenansicht, sondern in einem anderen Fenster (z.B. Notizverwaltung), so kann auf die Karte entweder durch das Drücken des Back-Buttons oder durch einen Tap auf das „Navigate-Up“-Zeichen in der oberen „Action Bar“ wieder zurückgekehrt werden.

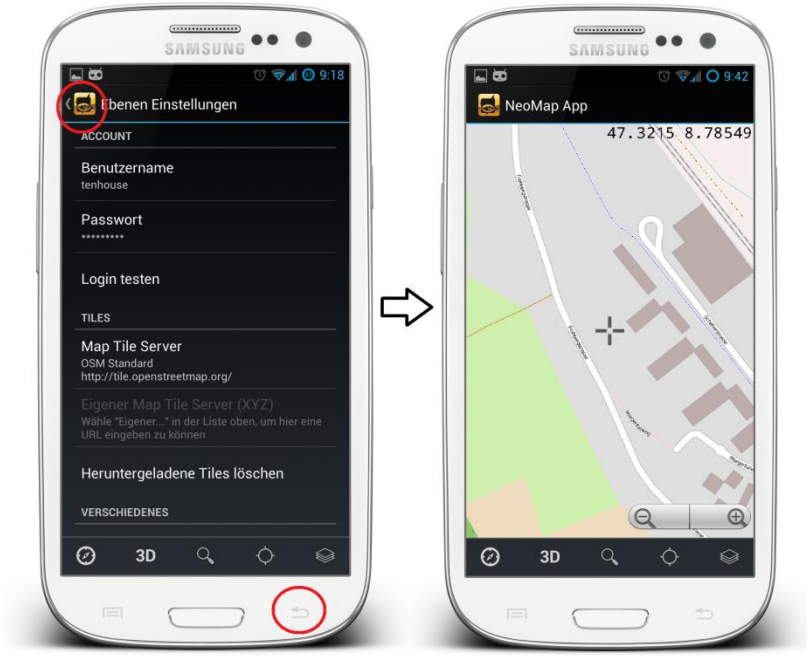


Abbildung 60: Möglichkeiten um zur Karte zurück zu navigieren

14.6 Spreizen bzw. Zusammenziehen der Finger (Pinch)

In Laufe der Zeit hat sich bei dieser Geste, welche unter dem Namen Pinch bekannt ist, als standardmässiges Verhalten das Hinein- bzw. Herauszoomen eingebürgert. Deshalb wird auch im NeoMap App diese Funktion benutzt.

15 Weiterentwicklung

In diesem Kapitel soll noch einmal auf die verschiedene Aspekte der Weiterentwicklung eingegangen werden. Dieses Kapitel ist ergänzend und technischer abgefasst als „Teil I4.2 Schlussfolgerung und Ausblick“.

15.1 Frontend

In diesem Kapitel sollen ein paar Tipps bzw. Empfehlungen zur Umsetzung von Features gegeben werden, welche für den Benutzer sichtbar sind.

15.1.1 Ganze Karten-Regionen herunterladen

Um ganze Kartenregionen herunterladen zu können, sollte dem Benutzer vermutlich am besten eine Funktion zur Verfügung gestellt werden, mit welcher er auf der Karte einen Abschnitt markieren kann (ähnliches Prinzip wie beim Linien erfassen). Von diesem könnte dann eine BoundingBox erstellt werden und alle Tiles auf allen Zoomstufen durchgegangen werden. Dann könnte im Hintergrund der Download gestartet werden. Wichtig hierbei ist sicherlich auch, dass die voraussichtliche Downloadgrösse überprüft wird. Diese sollte ziemlich genau berechnet werden können, da es sehr einfach kalkulierbar ist, wie viele Tiles insgesamt heruntergeladen werden können. Ansonsten könnte eine viel zu grosse Menge an Datendownloads anfallen. Zusätzlich sollte dies je nach Grösse vielleicht nur per WLAN oder mit sehr klarem Hinweis an den Benutzer erlaubt werden.

15.1.2 Auto-Tracking bzw. Aufzeichnen von Routen

Dieses Feature sollte nicht allzu schwer umsetzbar sein. Da vieles schon gegeben ist: Der eigene Standort kann bereits jetzt verfolgt werden und Linien können auch jetzt schon erfasst werden. Man muss im Prinzip nur einen Tracking-Modus entwickeln, welcher vielleicht jeweils bei einer Positionsänderung von 10 Metern einen neuen „Punkt“ zur Linie erfasst. Die grösste Challenge hierbei ist sicherlich, den Mittelweg zwischen zu vielen und zu wenigen Punkten beim automatischen Erfassen von Routen zu finden.

15.1.3 Import von GPX-Dateien

Der Import von GPX-Dateien wäre auch nicht schwierig zu implementieren. Die App muss als erstes „Intents“ der Art GPX empfangen können. Die GPX-Dateien müssen dann validiert und eingelesen werden. Das Mapping ist dabei gleich wie beim Export. Mit einigen Zusätzen: So sollte das Element „<rte>“ vermutlich gleich wie das Element „<trk>“ behandelt werden und allfällige „<sgt>“ ignoriert bzw. zu einem einzigen Track zusammen gefasst werden (Sieh auch Kapitel „Export“ zum Thema Mapping). Gegebenenfalls muss dem Benutzer nach dem Import noch eine Inputmaske zum Vervollständigen der Angaben (wie Einfüllen des Titels oder der Beschreibung) angezeigt werden.

15.1.4 Zusammenspiel mit der NeoMap Webseite

Zu diesem Thema soll hier nicht allzu viel gesagt werden, da die Webseite eigentlich nicht direkter Teil dieser Arbeit ist. Allerding soll erwähnt werden, dass Google an der I/O 2013 das Feature „Cross-Platform Auth mit Google+ Sign In“ (https://developers.google.com/events/io/sessions/332060543) vorgestellt hat. Dieses Feature ermöglicht es Benutzern sich nur ein einziges Mal auf der NeoMap Webseite mit einem Google+ Account einloggen zu müssen und danach automatisch die Möglichkeit erhalten, das App auf dem Smartphone zu installieren und dort direkt (ohne weitere Einflüsse) eingeloggt zu sein. Natürlich fallen für dieses Feature grössere Änderungen in der App an, es würde jedoch den Registrierungs-, sowie Login-Prozess für Benutzer erheblich vereinfachen.

15.1.5 Navigationsart anpassen

Android gibt verschiedene Navigationsarten vor – wovon keine das Overflow-Menü benutzt. Dies realisierten wir jedoch erst gegen Ende des Projektes.
Von Android empfohlene Navigationsmöglichkeiten sind:

- Six-Pack
  - Bietet eine eigene Activity an, welche eine Art Startbildschirm mit verschiedenen Menüpunkten darstellt
- Spinners (Drop-Down Menu)
  - Ein Drop-Down Menu welches oben in der Action Bar angezeigt wird und das Navigieren in der App ermöglicht
- Fixed-Tabs (mit Swipe)
  - Darstellung mittels maximal drei Tabs, zwischen welchen hin und her geswiped werden kann
- Navigation-Drawer (neu, vorgestellt auf der I/O 2013)
  - Bei Klick auf das App-Logo in der Actionbar oder bei einem Swipe am linken Bildschirmrand, wird ein Side-Menu geöffnet, welches über das unterliegende Design gelegt wird (http://developer.android.com/design/patterns/navigation-drawer.html)

Zur Navigation besseren in Apps bietet zudem dieses Video der Google I/O 2013 einen guten Überblick: https://developers.google.com/events/io/sessions/326301704

In der NeoMap App steht der Entwickler wohl vor der Wahl zwischen einem „Spinner“ und dem „Navigation-Drawer“. Wir persönlich empfehlen den „Navigation-Drawer“. Dieser bietet mehr Platz für zukünftige Navigationspunkte und kann besser strukturiert werden. Zudem ist er ein sehr neues Feature und wird so von Facebook, Google+ und YouTube benutzt. Er hat sich also bereits in einigen der grössten Apps durchgesetzt.

15.1.6 Ausgabe von Meldungen überarbeiten

Ausgabe-Meldungen an den Benutzer sollten überarbeitet werden, idealerweise sollten diese an dem Ort angezeigt werden, wo sie ausgelöst wurden. Gleichzeitig könnte auch das LongPressMenu in diesem Stil eingeführt werden, damit ein Benutzer immer klar und direkt sieht, auf welche Stelle am Bildschirm er gedrückt hat. Zu realisieren ist dies mit einer kleiner Zusatz-Library, welche ursprünglich von den Machern von Astrid (einer App) entwickelt wurde und als OpenSouce-Produkt verfügbar ist. Erste Informationen zur Implementation finden sich in diesem StackOverflow Post:
http://stackoverflow.com/questions/9833621/custom-spinners-drop-down-menu

15.2 Backend

An dieser Stelle werden Empfehlungen für Änderungen der Applikation hinter den Kulissen gegeben. Dazu gehören Dinge wie Performance und Architekturänderungen die der Benutzer nicht direkt mitbekommt.

15.2.1 Karten-Darstellung und Performance

Während dieser Arbeit wurden keine wirklichen Stress-Tests für die App durchgeführt. Diese Aussage bezieht sich vor allem auf eine grosse Anzahl von vorhandenen NeoMaps bzw. Notizen gleichzeitig auf der Karte. Bei Notizen sollte es diesbezüglich kaum Probleme geben.
Jedoch wird bei zu vielen gleichzeitig aktiven NeoMaps das Problem auftreten, dass die Geräte zu wenig Arbeitsspeicher haben. Auch können NeoMaps aktuell in unterschiedlichen Grafik-Grössen auf das Gerät geladen werden, was es erschwert, Berechnungen bezüglich eines Caching durchzuführen. Hier müsste zwingend ein Konzept erarbeitet werden, in welchem definiert wird, in welcher Grösse die Grafiken der NeoMaps verfügbar sind. Auch sollte überlegt werden, ab welcher Zoom-Stufe es überhaupt Sinn macht, NeoMaps auf der Karte darzustellen.

Es macht kaum Sinn, auf einer Weltansicht Stadtpläne von einzelnen Ortschaften zu laden. Möglicherweise wäre es auch sinnvoll, für verschiedene Zoom-Stufen Grafiken mit unterschiedlichem Detail-Grad zu laden (Stichwort MipMapping).
Gerade bezüglich der Überlegungen, auf welcher Zoom-Stufe NeoMaps angezeigt werden sollen, könnte auch auf Notizen übertragen werden. So könnten auch die Abfragen auf die SQLite Datenbank jeweils beschränkt werden.
So würde sicherlich eine Performance-Optimierung erreicht werden, wenn man sich auf einer relativ hohen Zoomstufe (d.h. näher am Boden) befindet.

15.2.2 Geografische Namenssuche verbessern

Bei der Suche sollten mehrere Dinge verbessert werden. Als erstes ist der Suchalgorithmus für Koordinaten, sowie für die Namenssuche in der lokalen Datenbank sicher nicht optimal. Bei der Namenssuche ist der Grund dafür, dass die Suchwörter momentan einfach in ein Where-Statement mit „likes“ gepackt werden (siehe Kapitel „Normale Suche“). Dies kann bei vielen Einträgen eventuell zu Performance-Einbussen führen und sollte genauer untersucht werden. Die Suche sollte so angepasst werden, dass erkannt werden kann, mit welchem Suchwort in welcher Spalte gesucht werden soll (oder so ähnlich).
Ausserdem wird bei der lokalen Suche mit Koordinaten wird momentan ein sehr grosses Delta verwendet (siehe dazu „Reverse Suche“). Dies kann sicherlich optimiert werden.
Diese beiden Punkte sind jedoch sicherlich nicht einfach umzusetzen. Ob dies überhaupt nötig ist, sollte zuerst abgeklärt werden.
Weiter eignet sich eine herkömmliche SQL-Datenbank eher schlecht für die Speicherung der Nominatim-Ergebnisse, da der RESTful-Webservice dynamische Antworten bzw. Objekte liefert. Besser wäre hier vermutlich eine NoSQL-Datenbank geeignet. Darin könnte die ganze Antwort einfach als JSON-String abgespeichert werden. Für Android existiert ein Projekt, welches die CouchDB in Apps integrierbar macht. Das Projekt findet sich unter https://github.com/couchbase/couchbase-lite-android.

15.2.3 Datenschicht besser aufteilen (DAOs / ORM)

Dies ist ein Punkt der uns persönlich an der internen Struktur momentan am meisten stört. Alle Datenzugriffe werden momentan über die Klasse „DbController“ gehandhabt. Dies ist natürlich komplett gegensätzlich zu vielen Software-Entwicklungs-Prinzipien. Vor allem wird das „Separation of Concerns“ verletzt.
Dieses Problem kann auf zwei Arten gelöst werden. Entweder müssen für alle Domain-Objekte zusätzliche DAOs (Data Access Objects) eingeführt werden, welche jeweils alle Datenbankmethoden für diese Klassen enthalten oder aber es wird eine ORM-Bibliothek eingeführt. Bekannte ORMs für Android sind zum Bespiel ORMLite (http://ormlite.com/) oder GreenDAO (http://greendao-orm.com/).
Wir empfehlen die Einführung einer ORMLibrary. Dies bringt den zusätzlichen Vorteil mit sich, dass keine nativen SQL-Abfragen etc. geschrieben werden müssen, sondern in fast allen Fällen auf das Mapping zurückgegriffen werden kann, welches einem diese Arbeit abnimmt.

15.2.4 Testing

In dieser Arbeit haben wir zwar einen Grundstock an Unit Tests eingeführt, jedoch sollte dieser stark erweitertet werden. So gibt es bis jetzt definitiv noch zu wenig Tests um alle Funktionen in der App zu testen. Allein schon bei BoundingBoxes können noch viele Tests eingeführt werden, die verifizieren, ob eine Box eine andere berührt usw. Für einige Klassen existieren auch gar keine Tests, weil diese als nicht wichtig genug angesehen wurden. Trotzdem ist auch bei diesen von Vorteil wenn man Tests hat.

Zudem sollten noch etwas komplexere, Integrationstests, eingeführt werden. Bei diesen sollte auch versucht werden zu testen, ob an einer gewissen Stelle auch wirklich das richtige Tile angezeigt wird oder ob das Zoomen in allen Regionen auf der Karte richtig funktioniert usw.

15.2.5 OpenGL ES 3.0

Mit dem kommenden Android 4.3 bzw. Android 5.0, soll, gemäss verschiedenen Quellen und Gerüchten, OpenGL ES 3.0 eingeführt werden. Der Unterschied der Version 2.0 auf 3.0 ist nicht mehr so gross wie derjenige von 1.1 auf 2.0. Damit sind ideale Bedingungen gegeben, um sich mit einem allfälligen Update auf diese Version zu befassen. OpenGL ES 3.0 bringt unter anderem Performance-Optimierungen sowie einen einfacheren Einbau von Effekten mit sich.

Teil III
Projektmanagement

1 Projekt Planung

1.1.1 Zeitlicher Rahmen

Es wird davon ausgegangen, dass jedes Projektmitglied im Durchschnitt 25 Stunden pro Woche aufwenden muss. Über die insgesamt 15 Wochen ergibt dies insgesamt 750 Personen-Stunden. Die Stunden errechnen sich an der Anzahl der Kredit-Punkte für das Modul „Bachelorarbeit Informatik“ (12 Kredit-Punkte) und dem vorgegebenen Zeitaufwand pro Kredit der HSR, welcher sich auf ca. 30 Stunden beläuft. Die Stunden sind jedoch nur eine Richtzeit. Sollten die Projektteilnehmer die geforderte Aufgabe in kürzerer Zeit lösen, so muss nicht durch sinnlose Zusatzarbeit das Zeitbudget komplett ausgeschöpft werden. Zusätzlich muss erwähnt werden, dass nach Ende des Semesters noch zwei Wochen Zeit bleiben, in welchen 100% an der BA gearbeitet werden könnte, diese wurden in die obige Berechnung jedoch nicht einbezogen, da diese als Reserve gelten sollen.
Das Projekt läuft vom 18.02.2013 bis spätestens 16.06.2013 17:00 Uhr und wird termingerecht beendet. Optionale Features können bei Bedarf und bei genügend Zeit umgesetzt werden. Der Fokus liegt jedoch ganz klar darin, alle definierten Muss-Ziele umzusetzen.

1.2 Vorgehen / Prozessmodell

Zur Umsetzung dieses Projekts wird eine vereinfachte Form des Rational Unified Process (RUP) angewendet und somit in einem iterativen und inkrementellen Vorgehen vorgegangen. Dabei wird das Projekt in vier Phasen (Inception, Elaboration, Construction, Transition) aufgeteilt, wobei die einzelnen Phasen mehrere Iterationen haben können. Eine Iteration wird jeweils mit einem Meilenstein abgeschlossen.
Genaueres zu den einzelnen Phasen und Iterationen kann im Kapitel Phasen / Iterationen nachgelesen werden.

1.2.1 Zeitliche Planung

Die Zeitliche Planung kann komplett in Redmine verfolgt werden, in welchem auch alle Phasen aufgelistet sind.
http://neomapba.no-ip.org/redmine/projects/ba_neomap_fs13/roadmap
Zusätzlich befindet sich im Kapitel Projekt Monitoring und Schlussbericht eine Auflistung und Auswertung aller Phasen und Arbeitspaketen.

1.2.1.1 Phasen / Iterationen

Da die gesamte Planung über Redmine stattfindet, sind alle Phasen und Iterationen auch dort ersichtlich.
Für den Betreuer (Prof. S. Keller) ist ein Konto eingerichtet. Zusätzlich haben öffentliche Benutzer Lese-Zugriff:
http://neomapba.no-ip.org/redmine/projects/ba_neomap_fs13

Folgend sind alle Phasen mit direkten Links zu Redmine aufgelistet:

Phase	Link
Inception	http://neomapba.no-ip.org/redmine/versions/1
Elaboration 1	http://neomapba.no-ip.org/redmine/versions/2
Elaboration 2	http://neomapba.no-ip.org/redmine/versions/4
Construction 1	http://neomapba.no-ip.org/redmine/versions/3
Construction 2	http://neomapba.no-ip.org/redmine/versions/5
Construction 3	http://neomapba.no-ip.org/redmine/versions/6
Construction 4	http://neomapba.no-ip.org/redmine/versions/7
Construction 5	http://neomapba.no-ip.org/redmine/versions/8
Transition 1	http://neomapba.no-ip.org/redmine/versions/9
Transition 2 / Reserve	http://neomapba.no-ip.org/redmine/versions/10

Tabelle 33: Phasenplan

1.3 Projektorganisation

Das Projektteam besteht aus zwei Mitgliedern und einem Betreuer.

1.3.1 Organisationsstruktur

Da das Team aus nur zwei Personen besteht, wird von der Notwendigkeit abgesehen, Rollen zu verteilen. Beide Team-Mitglieder werden als gleichberechtigt angesehen und die Arbeit wird somit zu gleichen Teilen aufgeteilt. Durch einen regelmässigen Austausch sowie gemeinsames Arbeiten, soll der gegenseitige Informations-Austausch gefördert und der Lern-Erfolg bei diesem Projekt maximiert werden.

1.3.2 Kommunikation

Die Hauptkommunikationskanäle sind:

- Face-To-Face
- Chat (Facebook, WhatsApp)
- Redmine
- Skype

Das Team trifft sich Montags von 13 bis 17 Uhr, dienstags von 9.00 Uhr bis ca. 17.00 Uhr, sowie jeden Mittwoch um ca. 12.00 Uhr bis ca. 17.00 Uhr an der HSR um am Projekt zu arbeiten und allfällige Schwierigkeiten gemeinsam zu behandeln.

1.3.3 Externe Schnittstellen

Als Ansprechperson ist **Prof. Stefan Keller (sfkeller@hsr.ch)** für dieses Projekt zuständig.

1.3.4 Besprechungen

Besprechungen/Arbeiten des Teams finden in der Regel jeweils dienstags an der HSR statt. Anwesend sind alle Teammitglieder und der Betreuer.
In späteren Phasen/Iterationen (ab Construction) können bei gutem Vorankommen und wenig Fragen die Meetings auch nur im Zwei-Wochen Rhythmus stattfinden. Ziel der Besprechungen ist es, den aktuellen Stand des Projektes zu protokollieren, Aufgaben zu verteilen und auf allfällige Probleme oder Fehler hinzuweisen. Ausserdem werden an den Meetings weitere Schritte in der Projektentwicklung geplant und festgehalten. Dazu wird jeweils ein Sitzungsprotokoll erstellt (direkt in Redmine).

1.4 Prototypen, Releases, Meilensteine

Eine detaillierte Übersicht über die Phasen ist in Redmine ersichtlich, hier sind die Meilensteine nur in Stichworten, sowie die Phasendauer aufgelistet.

Phase	Zeitraum	Meilensteine / Artefakte
Inception	18.02.2013 – 25.02.2013	• Klarstellung der Aufgabe • Erste Einblicke in OpenGL erhalten
Elaboration 1	26.02.2013 – 05.03.2013	• Einarbeitung in OpenGL • Analyse der bestehenden App • Grobplanung
Elaboration 2	06.03.2013 – 12.03.2013	• Planung abgeschlossen • Projektplan / Anforderung Spezifikation • Analyse der App (mit Änderungen M. Wolski) • Definitive Aufgabenstellung
Construction 1	13.03.2013 – 26.03.2013	• Migration von OpenGL ES 1.1 auf OpenGL ES 2.0 • Migration von Android 2.x auf Android 4.0 • Anzeige des eigenen Standortes auf der Karte bei aktivierter Lokalisierung
Construction 2	27.03.2013 – 09.04.2013	• Schrägansicht für Karte • Notizen für frei wählbare Punkte auf der Karte erfassen • Einsatz von Fragments (Darstellungsoptimierung)
Construction 3	10.04.2013 – 23.04.2013	• Geometrie (Routen) auf der Karte erfassen • Geometrie exportieren als GPX-Datei
Construction 4	24.04.2013 – 07.05.2013	• Anzeige von eigenen Karten („NeoMaps“) priorisieren können • Anzeige von eigenen Karten ortsabhängig ein bzw. ausschalten können
Construction 5	08.05.2013 – 21.05.2013	• Geografische Namenssuche (z.B. „London“) • Einmal durchgeführte Suchen lokal zur Verfügung stellen
Transition 1	22.05.2013 – 28.05.2013	• App Entwicklung abgeschlossen • Dokumentation abgeschlossen
Transition 2 / Reserve	29.05.2013 – 14.06.2013	• Reserve falls wegen des Eintreffen eines technischen Risikos alles verschoben werden muss

Tabelle 34: Phasen - Zeitraum – Meilensteine

1.5 Technische Risiken

Gewichteter Schaden: 61.2

Nr	Titel	Beschreibung	max. Schaden [h]	Eintritts-wahrschein-lichkeit	Gewichteter Schaden	Vorbeugung	Verhalten bei Eintreten
R1	Lokaler Datenverlust	Verlust der Daten einer ganzen Woche, eines einzelnen Teammitgliedes.	16	10%	1.6	Häufige Pushes auf den Server. Wenig lokale Daten.	Verlorene Arbeit wiederholen.
R2	Server-Datenverlust	Ausfall des Redmine Servers mit Datenverlust auf dem Server.	16	10%	1.6	Daten sind lokal sowie auf dem Server vorhanden. Sicherung wird jeden Tag durchgeführt.	Wiederherstellen des Servers mit lokalen Daten und Daten von Backup.
R3	OpenGL	Migration von OpenGL ES 1.x auf OpenGL ES 2.x stellt sich als unmöglich dar	100	25%	25	Einarbeitung / Lesen von Büchern. Hilfestellung P. Recher	P. Recher um Hilfe bitten. Auf OpenGL 1.1 bleiben.
R4	Aktueller Code unverständlich	Wir kommen nicht mit dem aktuellen Code klar, da dieser nicht dokumentiert ist, schlecht strukturiert und teilweise unnötig aufgeblasen ist.	80	30%	24	Analyse des Codes. Hilfestellung P. Recher	P. Recher um Hilfe bitten. Planung nach hinten verschieben.
R5	Geometrie	Bestehende App-Architektur lässt sich nicht leicht erweitern um eigene Punkte auf der Karte zu setzen.	30	30%	9	Einarbeitung in OpenStreetMap und GIS Technologien	Planung nach hinten verschieben.
Summe			**242**		**61.2**		

Tabelle 35: Technische Risiken

1.5.1 Umgang mit Risiken

Da nur eine sehr geringe Anzahl an projektspezifische Risiken vorhanden sind, existieren vorwiegend generelle Risiken, wie zum Beispiel Krankheit bzw. Totalausfall eines Teammitgliedes, vorhanden. Diese Risiken haben dafür aber ein sehr starkes Schadenspotenzial. Eine (nicht andauernde) Krankheit kann durch Zusatzaufwand des zweiten Teammitgliedes und unserer eingeplanten Reservezeit (T2) vorgebeugt werden. Der Totalausfall eines Teammitgliedes hingegen kann sicherlich nicht komplett kompensiert werden. Würde dieser Fall eintreten, so müsste das Projekt entweder abgebrochen, verlängert oder mit weniger Anforderungen versehen werden. Da dieses Risiko jedoch sehr tief ist, wird kein detaillierter Plan dafür aufgestellt.

Die technischen Risiken zeichnen sich hauptsächlich dadurch aus, dass die Einarbeitung oder Umsetzung der Materie für einen der Teammitglieder unterschätzt wird und sich dies in einem erhöhten Zeitbedarf niederschlägt. Tiefere Anforderungen an die technische Umsetzung können nicht gemacht werden, da sie als Bestandteil unserer Anforderung fest an das fertige Produkt gebunden sind. Spätestens Ende C2 sollte aber dieses Risiko eliminiert sein, da bis zu diesem Zeitpunkt die Einarbeitung abgeschlossen sein sollte. Falls nicht müsste über einen Projektabbruch bzw. eine Planänderung nachgedacht werden.

1.6 Qualitätsmassnahmen

Massnahme	Ziel	Zeitraum
DropBox-Ordner für Diagramme / Bilder / PDFs	Verfügbar für Alle, automatisches Backup	Aufsetzung zu Beginn von E1 (SW 02) Verwendung während dem gesamten Projekt
Verwendung von Git als Revision Control System	Aktueller Projektstatus und Versionshistorie für jeden zugänglich	Verwendung während dem gesamten Projekt
Automatisiertes Backup von Redmine	Verhinderung von Datenverlust bei Crash des Servers.	Verwendung während dem gesamten Projekt
Parallele Unit-Test Erstellung während Entwicklung	Fehlerminimierung und verbessertes Code Design	Während der gesamten Entwicklungsphase
Regelmässige Meetings mit dem Betreuer	Verhinderung von Entwicklung in falsche Richtung / Weiterhilfe bei Problemen	Während dem gesamten Projekt

Tabelle 36: Qualitätsmassnahmen

1.6.1 Dokumentation

Die Dokumentation befindet sich in einem DropBox Ordner (Cloud), so dass diese von überall her erreichbar ist. Die Teammitglieder besitzen alle einen persönlichen Account.
Zusätzlich ist DropBox in Redmine mittels des Plugins „Reddrop“ direkt integriert, so kann jederzeit auf alle Dokumente zugegriffen werden.

1.6.2 Code Reviews

Während des Projektes wird jeweils pro Construction-Phase ein Code-Review durchgeführt. Dabei werden die Teammitglieder zusammen, direkt an einem Laptop, wichtige Codeteile anschauen (es werden also nur Code-Reviews gemacht). Ein spezifisches Protokoll wird dafür nicht geführt, allfällige Notizen werden direkt im entsprechenden Ticket eingetragen. Kleine Änderungen werden direkt während dem Review durchgeführt. Fallen bei dem Review grössere Änderungen an, werden aus diesem direkt Tickets erstellt und einem der Teammitglieder zu gewiesen.
Die Reviews sollen jeweils ziemlich Ende einer Construction-Iteration stattfinden und werden jeweils ca. 1.5 – 2h dauern.

1.6.3 Coding Standards und Tools

- Um die Anzahl an Bugs stark zu vermindern, wird das Tool FindBugs eingesetzt. Dieses findet eine grosse Anzahl von potentiellen Fehlern und hilft so die Code-Qualität hoch zu halten. http://findbugs.sourceforge.net/
- Zu Kontrolle der Struktur des Codes bzw. Metrikanalyse des Codes wird das Tool Stan verwendet. Das Tool kann die meisten gängigen Metriken wie z.B. LOC, Komplexität, Tiefe etc. auslesen und ausgeben. Während der Entwicklungszeit werden die Metriken in regelmässigen Abständen (Wöchentlich) angeschaut und gegebenenfalls Änderungen im Code vorgenommen. Am Ende des Projektes werden die "Abschluss-Metriken" dokumentiert. http://stan4j.com/
- Um weitere Informationen über den Quellcode, wie beispielsweise Dead Code oder auch Similar Code, zu erkennen wurden unter anderem auch auf das Tool CodePro AnalytiX von Google gesetzt. https://developers.google.com/

2 Infrastruktur

Die Teammitglieder arbeiten jeweils beide mit ihren persönlichen Laptops. Dies hat den Vorteil, dass sowohl zu Hause als auch an der Schule oder aber auch unterwegs, mit derselben Umgebung gearbeitet werden kann.

2.1 Zentraler Server für Projektmanagement

Dem Team steht ein Ubuntu Server zur Verfügung. Dieser kann über die Adresse http://NeoMapBA.no-ip.org erreicht werden oder per SSH.
Jedes Projektmitglied hat einen persönlichen Account, welcher dem HSR Kürzel entspricht.
Auf diesem Server ist auch ein Redmine installiert, dieses dient primär als Planungs-und als Zeiterfassungstool. So findet die ganze RUP-Phasenplanung dort statt. Zusätzlich werden in Redmine auch alle Sitzungsprotokolle sowie allfällige Notizen zu kommenden oder erledigten Aufgaben erfasst.

2.2 Zentraler Server für NeoMap

Zusätzlich zu dem oben beschriebenen Redmine für das Projektmanagement existiert ein weiteres Redmine unter der Adresse http://dev.ifs.hsr.ch/redmine/projects/neomap/ . Dies ist das offizielle Projektmanagementtool von NeoMap. In diesem Redmine werden nur Tickets erfasst, welche direkt etwas mit der NeoMap App zu tun haben (z.B. „Migration OpenGL ES 1.1 zu Open GL ES 2.0“). Die Trennung der beiden Redmines wird als besonders Wichtig erachtet, da das offizielle NeoMap Redmine aufgeräumt und nicht durch unnötige Tickets wie z.B. „Einlesen in OpenGL“ aufgebläht werden soll.
Um noch zusätzliche Übersicht zu gewinnen, werden viele Git-Commits direkt mit Tickets aus diesem Redmine verknüpft, so ist zu Tickets direkt ersichtlich, welche Codeänderungen diese mit sich bringen. Im Weitern wird in diesem Redmine ausschliesslich in Englisch kommuniziert.

2.3 Git als Versionsverwaltungssystem

Als Git Server wird der offizielle HSR Server gebraucht. Das Repository ist unter der Adresse https://git.hsr.ch/git/NeoMap App zu erreichen.
Alle Git Commits werden in Englisch gemacht und wenn immer möglich mit Tickets aus dem Redmine verknüpft.
Da die Versionierungsmethode Git ist, müssen keine speziellen Richtlinien für das „committen“ festgelegt werden. Code wird jedoch nur auf das zentrale Git-Repo gepusht, wenn er funktionsfähig ist.

2.4 Entwicklungsumgebung

Für die Programmentwicklung wird mit Eclipse Juno mit dem Android ADT Plugin 21.1 gearbeitet.
Es werden bei allen Eclipse Installationen die Standardeinstellungen beibehalten, so dass überall die gleichen Formatierungsregeln etc. verwendet werden.

2.5 Dokumentationswerkzeug

Die Dokumentation erfolgt mithilfe von Microsoft Office 2010 und Redmine.

2.6 Testgeräte

Das Ausführen bzw. das Testen der NeoMap App wurde ausschliesslich auf echten Geräten durchgeführt. Dies vor allem aus dem Grund, dass OpenGL auf dem Emulator nicht immer richtig funktioniert und im Vergleich zu einem echten Gerät sehr langsam ist.

Bei der Auswahl der Testgeräte wurde darauf geachtet, dass nur Geräte mit einer Android Version höher als 4.0 gewählt wurden und dass eine möglichst grosse Palette an Displaygrössen und Auflösungen vorhanden war.
Folgende Tabellen sind geordnet nach der Häufigkeit, in der die Geräte zum Testen benutzt wurden.

Abbildung 61: Samsung Galaxy S3 (I9300)

Name	Samsung Galaxy S3, I9300
Art	Smartphone
Leistungstyp	High-End Gerät
Verbreitung	Hoch
Display-Grösse	4.8 "
Auflösung	1280×720 Pixel
Android Version	4.2.2
CPU	Samsung Exynos 4412 1.4 GHz (Quad Core)
GPU	ARM Mali-400-MP
Ram	1024 MB
Herkunft	Persönliches Gerät von P. Zenhäusern

Tabelle 37: Informationen Galaxy S3 (I9300)

Abbildung 62: Galaxy Nexus S

Name	Galaxy Nexus S
Art	Smartphone
Leistungstyp	Mid-Level Gerät
Verbreitung	Hoch
Display-Grösse	4 "
Auflösung	800 x 480 Pixel
Android Version	4.1.2
CPU	1 GHz ARM Cortex-A8 (Single Core)
GPU	GPU PowerVR SGX 540
Ram	512 MB
Herkunft	Ausgeliehen vom Institut für Software, HSR

Tabelle 38: Informationen Galaxy Nexus S

Abbildung 63: Asus Nexus 7

Name	Asus Nexus 7
Art	Tablet
Leistungstyp	Mid-Level - High-End Gerät
Verbreitung	Hoch
Display-Grösse	7 "
Auflösung	1280 x 800 Pixel
Android Version	4.2.1
CPU	1,3 GHz Nvidia Tegra 3 (Quad Core)
GPU	416 MHz twelve-core Nvidia GeForce ULP
Ram	1024 MB
Herkunft	Ausgeliehen vom Institut für Software, HSR

Tabelle 39: Informationen Asus Nexus 7

Abbildung 64: LG Nexus 4

Name	LG Nexus 4
Art	Smartphone
Leistungstyp	High-End Gerät
Verbreitung	Mittel
Display-Grösse	4.7 "
Auflösung	1280x768 Pixel
Android Version	4.2.1
CPU	Qualcomm Snapdragon™ S4 Pro CPU (Quad-Core)
GPU	GPU Adreno 320
Ram	2048 MB
Herkunft	Ausgeliehen vom Institut für Software, HSR

Tabelle 40: Informationen Nexus 4

Abbildung 65: HTC Wildfire S

Name	HTC Wildfire S
Art	Smartphone
Leistungstyp	Low - Mid-Level Gerät
Verbreitung	Gering
Display-Grösse	3.2 "
Auflösung	240x320 Pixel
Android Version	4.0.4
CPU	Qualcomm 528 MHz (Single Core)
GPU	GPU Adreno 200
Ram	384 MB
Herkunft	Persönliches Gerät von S. Stähli

Tabelle 41: Informationen HTC Wildfire S

Abbildung 66: Asus Eee Pad Transformer

Name	Asus Eee Pad Transformer
Art	Tablet
Leistungstyp	Low – Mid-Level Gerät
Verbreitung	Gering
Display-Grösse	10.1 "
Auflösung	1280 x 800 Pixel
Android Version	4.1.1
CPU	1 GHz Nvidia Tegra (Dual Core)
GPU	ULP GeForce
Ram	1024 MB
Herkunft	Persönliches Gerät von P. Zenhäusern

Tabelle 42: Informationen Asus Eee Pad Transformer

2.7 Einrichtung der Entwicklungsumgebung (Eclipse)

Damit das Projekt weiter entwickelt werden kann, müssen ein paar wenige Schritte ausgeführt werden:

1. Klonen des offiziellen Repositories: https://<Benutzername>@git.hsr.ch/git/NeoMapApp
2. Erstellen eines Eclipse Projektes:
   In Eclipse über *Datei → Import → Existierender Android Code In Workspace* das soeben geklonte Projekt importieren
   Wichtig:
   Da die NeoMap App eine Third-Party Drag & Drop Library verwendet, muss beim Import sowohl „NeoMapApp“ wie auch „drag-sort-listview-library\library“ und „drag-sort-listview-library\launcher“ ausgewählt werden. Bei dem „library“-Projekt sollte man am besten noch einen guten Namen geben.
3. Rechtsklick auf die NeoMap App Projekt → Eigenschaften → Android → Bei „Bibliotheken“ mittels „Hinzufügen..“ das vorher importiere „library“-Projekt auswählen
4. Das „launcher“ Projekt kann jetzt, wenn gewünscht, wieder gelöscht werden, es musste nur einmal importiert werden, damit das „R“-File für die Library generiert werden konnte.

Die letzten beiden Schritte sind zwingend notwendig, weil Android Third-Party-Libraries leider sehr mühsam handhabt. Es kann nicht einfach ein JAR hinzugefügt werden, sondern es muss ein Projekt im Workspace geben. Dies hat den Grund, dass XMLs und andere Android Ressourcen anscheinend nicht in ein JAR gepackt werden können.

3 Projekt Monitoring und Schlussbericht

3.1 Projektrückblick

ID	Task Mode	Task Name	Duration	Start
1		Inception	5 days	Tue 19.02.13
2		Elaboration #1	6 days	Tue 26.02.13
3		Elaboration #2	5 days	Wed 06.03.13
4		Construction #1	10 days	Wed 13.03.13
5		Construction #2	10 days	Wed 27.03.13
6		Construction #3	10 days	Wed 10.04.13
7		Construction #4	10 days	Wed 24.04.13
8		Construction #5	10 days	Wed 08.05.13
9		Transition 1	5 days	Wed 22.05.13
10		Transition 2 / Reserve	13 days	Wed 29.05.13

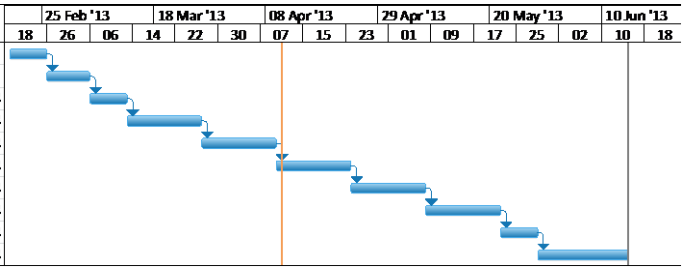


Diagramm 19: Projektüberblick

3.1.1 Inception

Zu Beginn des Projektes stand ein Kickoff-Meeting mit Herrn Keller, in welchem uns die Ausgangslage sowie eine erste Idee der Aufgabenstellung dargelegt wurden. Da es sich hierbei um eine Art Folgearbeit einer bestehenden App handelte, war der technologische Rahmen bereits recht klar definiert und es galt sich in die Thematik, vor allem Android, OpenGL ES bzw. Grafik-Programmierung im Allgemeinen sowie in OpenStreetMap einzulesen. Von Beginn weg stand uns Quellcode, jedoch noch nicht die aktuellste Version, des bestehenden Apps zur Verfügung, welcher die Grundlage unserer Arbeit festlegte. Nach knapp einer Woche, wurde noch ein Meeting mit P. Recher, einem HSR-Studenten, welcher die Grundlage des NeoMap Apps gelegt hatte, durchgeführt. Dabei konnten einige wichtige Punkte in Bezug auf die Entwicklung geklärt werden.

3.1.2 Elaboration 1

Im Fokus der ersten Elaboration-Phase stand eine erste Grobplanung des Projektes inklusive Risiko-Analyse. Viel Zeit wurde in dieser Phase auch dem Einarbeiten in die Grafik-Programmierung allgemeinen sowie OpenGL ES 2.0 spezifisch gewidmet, um uns das dazumal noch fehlende Wissen anzueignen. Da zu diesem Zeitpunkt noch ein weiteres Projekt am Laufen war (eine Projektarbeit von M. Wolski), welches sich auch mit der Weiterentwicklung des NeoMap Apps beschäftigte, lag uns leider immer noch nicht der aktuellste Stand des Quellcodes zur Verfügung.

3.1.3 Elaboration 2

Im Verlaufe der zweiten Elaboration-Phase wurde uns der aktuellste Quellcode geliefert, in welchen wir uns nochmals einarbeiteten und nach einer genaueren Analyse zum Schluss kamen, dass es sinnvoll sei, am Anfang des Projektes erstmals ein grosses Refactoring durchzuführen, um eine gute Basis für Weiterentwicklungen am App zu legen. Es wurden Refactoring-Tasks ausgearbeitet sowie neue Anforderungen mittels User-Szenarien festgelegt. In Redmine wurde eine detaillierte Planung ausgearbeitet mittels einzelner Tasks, welche den Phasen zugeordnet wurden. Nach wie vor war die endgültige Aufgabenstellung noch in Bearbeitung. Zudem wurde bereits begonnen mit OpenGL ES 2.0 erste kleine Test-Anwendungen zu erstellen. Es wurde darauf verzichtet, einen Prototypen zu erstellen, da das konkrete Produkt (die NeoMap App) ja bereits existierte.

3.1.4 Construction 1

Während der ersten Phase haben wir uns entschieden, das Refactoring in einem grösseren Stil durchzuführen als wir es ganz am Anfang geplant hatten. Es war uns wichtig, eine solide Basis zu schaffen, die wir auch im Detail verstanden, um danach darauf aufbauen zu können. Die grössten Schwierigkeiten waren hierbei beim Verständnis des bestehenden Codes und deren Architektur zu finden.

Dies vorwiegend aufgrund nicht existenter Dokumentation, diese war weder in Form eines Dokumentes, noch als Kommentare im Code vorhanden. Ausserdem war die bestehende Lösung teils doch sehr „Bastel/Hack“ mässig zusammen gestiefelt.
Die Migration von OpenGL ES 1.1 auf 2.0 konnte trotz anfänglichen Schwierigkeiten, zurückzuführen auf unzureichende Erfahrungen im Bereich der Grafik-Programmierung, alles in allem gut und in der geplanten Zeit abgeschlossen werden. Am Ende dieser Phase wurde dann auch die Aufgabenstellung zusammen mit Herrn Keller ein letztes Mal angepasst und finalisiert. Wie in einigen Diagrammen zu sehen ist, wurde in der Construction 1 sehr viel gemacht, dies war jedoch auch geplant, denn es war von Anfang an das Ziel, möglichst viel in den ersten drei Construction Phase zu erledigen, damit ein gewisses Reserve-Polster aufgebaut werden konnte.

3.1.5 Construction 2

In der zweiten Construction-Phase wurden erstmals auch neue Features eingebaut, dies jedoch erst nachdem nochmals eine Refactoring-Runde anfangs der Phase durchgeführt wurde. Bei der Implementierung des Features "Notizen erfassen" wurde die Grund-Architektur nochmals ein letztes Mal leicht in seinen Grundzügen angepasst, um eine gute Erweiterbarkeit in Hinsicht auf weitere Karten-Overlays zu garantieren. Weiter wurde bereits begonnen die 3D-Ansicht umzusetzen, dies gestaltete sich schwieriger als erwartet. Lediglich die Ansicht der Karte zu ändern war zwar kein Problem, jedoch mussten auch die Anzahl der zu ladenden Tiles sowie die eigene Position und vieles mehr angepasst werden, um eine ansprechende Darstellung zu ermöglichen. Zudem wurden auch noch Fragments eingeführt. Grundsätzlich beinhaltete diese Aufgabe keine grossen Probleme, jedoch musste trotzdem der gesamte Life-Cycle der Applikation angepasst werden.

3.1.6 Construction 3

Während der dritten Construction-Phase lag der Fokus vor allem in der Implementierung von Geometrie (Linien-Gebilden). Hier bestand die Herausforderung darin, einen geeigneten Bearbeitungsmodus einzuführen, der sich klar von der normalen Kartenansicht unterscheidet. Realisiert wurde dies letztlich mit dem „Contextual-Action-Mode“. Weiter kamen wir durch Herrn Keller auf die Idee in der Schrägansicht eine Art Horizont / Himmel darzustellen, dies wurde dann als SkyBox realisiert. Ausserdem kam eine weitere Schwierigkeit hinzu: Das Antippen von Objekten in der 3D-Ansicht. Da diese perspektivisch ist, gestaltete sich dies anspruchsvoller als geplant. Mit einigem zusätzlichen Zeitaufwand konnte jedoch auch dies gemeistert werden.

3.1.7 Construction 4

Hier galt es die letzten „Muss-Ziele“ umzusetzen. Als erstes wurde die Priorisierung von NeoMaps angegangen. Dies stellte sich als erstaunlich einfach heraus, da durch die Architektur bereits ein gutes Grundgerüst für das Auffinden von mehreren überlappenden NeoMaps vorhanden war. Das einzige Problem stellte hierbei die Drag & Drop Sortierung in einer Liste dar. Android bietet diese Möglichkeit von Haus aus nicht an. Aus diesem Grund wurden verschiedene externe Bibliotheken evaluiert und schliesslich eine davon eingesetzt. Gleichzeitig wurde der Export von Notizen in GPX-Dateien realisiert. Diese Aufgabe stellte keine technischen Herausforderungen. Das grösste Problem war hierbei das Konzept. Letztlich wurde das Konzept so umgesetzt, dass die komplette Verwaltung von NeoMaps und Notizen umstrukturiert wurde und neue Features, wie das Bearbeiten von mehreren NeoMaps/Notizen gleichzeitig, hinzukamen. Obwohl noch nicht in dieser Phase geplant, wurde schon mit der geografischen Namenssuche begonnen und etwa zur Hälfte bereits umgesetzt. Dies vor allem aus dem Grund, weil erstens der Fortschritt sehr hoch war und wir zum anderen nochmals ein Refactoring in der letzten Phase durchführen wollten.

3.1.8 Construction 5

Während der letzten Construction-Phase wurde zuerst die geografische Namenssuche fertiggestellt. Diese brachte vor allem für die Bereitstellung der Offline-Fähigkeit der Suche einige Herausforderungen mit sich. Nachdem diese gemeistert waren, ging es darum nochmals ein finales Refactoring durchzuführen. Dabei wurden wiederum einige Klassen stark vereinfacht. Teilweise wurden noch einmal einzelne nicht-verwendete Klassen gefunden, welche aus dem Projekt entfernt werden konnten. Auch die Benutzeroberfläche sowie die Icons wurden an vielen Stellen noch einmal überarbeitet, um ein einheitliches Erscheinungsbild der App sicherzustellen. Zusätzlich wurden viele zusätzliche Unit-Tests eingeführt und Testszenarien für User-Tests bzw. Systemtests erarbeitet. Bereits in dieser Phase wurde viel Zeit in das Dokumentieren der Arbeit investiert.

3.1.9 Transition 1

Während der ersten Transition-Phase galt es vorwiegend die Dokumentation soweit wie möglich zu erstellen, um diese Herrn Keller für ein erstes Review bereitzustellen. Dadurch sollte sichergestellt werden, dass die Dokumentation den Vorstellungen von Herrn Keller entspricht und noch genug Zeit für allfällige Änderungen vorhanden war. In der Applikation wurden nur noch kleine Fehler und kleine Refactorings durchgeführt. Neue Features wurden bereits seit Mitte C5 nicht mehr eingebaut.

3.1.10 Transition 2 / Reserve

In der Transition 2, die hauptsächlich als Reserve geplant war, wurde trotzdem noch gearbeitet. Die Dokumentation wurde noch einmal überarbeitet und gemäss dem Feedback von Herrn Keller angepasst. Ausserdem wurden das Poster, das Video und andere Abschlussarbeiten durchgeführt. Beim Video wurde zuerst ein kleines Drehbuch geschrieben und dann, meist direkt auf dem Handy, mit dem App „Screencast“, aufgenommen. Mittels dem Programm „Camtasia Studio 8.0“ wurde am Ende das Video zusammengeschnitten. Das Ganze nahm glücklicherweise nicht ganz so viel Zeit in Anspruch wie zuerst befürchtet, da der Umgang mit Camtasia bereits bekannt war. Leider war die Videoqualität des vom Smartphone aufgenommenen Bildes teilweise nicht sehr überragend, obwohl extra das leistungsstärkste Gerät (Nexus 4) benutzt wurde. Der Grund dafür ist, dass die NeoMap App zum Aufbau der Karte bzw. zum Laden von Tiles viel GPU und CPU Leistung benötigt. Das App „Screencast“ benötigt jedoch auch recht viele Ressourcen für das Aufnehmen. Aus diesem Grund entstanden dann teilweise Einbussen in der Qualität.

3.2 Projektaufwände

3.2.1 Überblick

Vorweg möchten wir anmerken, dass hier keine Statistik aufgezeigt wird, die pro Person angibt, wie viele Stunden gearbeitet wurde. Wir sahen uns stets als Team, haben beide etwa gleich viel gearbeitet und sehen keinerlei Nutzen in einer solchen Statistik.

Das erste Diagramm zeigt den effektiven Aufwand im Projekt im Vergleich zum linearen, d.h. nach den Credits vorgegebenen, Aufwand an. Der Credit-Aufwand errechnet sich dabei wie folgt: 1 Credit sind 30h, die Bachelorarbeit gibt 12 Credits, was 360 Credits pro Person ergibt, also 720 Credits total. Aufgeteilt auf 15 Wochen, ergibt das pro Woche 48h Aufwand.
Wie zu sehen ist wurde fast die ganze Zeit mehr gearbeitet, als eigentlich vorgeben. Dies ist aber nicht etwa auf eine komplett falsche Planung zurückzuführen, denn bei dieser wurde der zusätzliche Aufwand meistens eingeplant (siehe auch Diagramme weiter unten).
Speziell heben sich vermutlich noch die letzten beiden Wochen heraus. Bei diesen steigt der lineare Aufwand nicht mehr weiter an, da die 720 Credits dort bereits aufgebraucht sind. Diese zwei Wochen stellen gleichzeitig die Transition 2 dar, in welcher noch einiges an Dokumentation sowie sonstiger Abschlussarbeiten geleistet wurde.

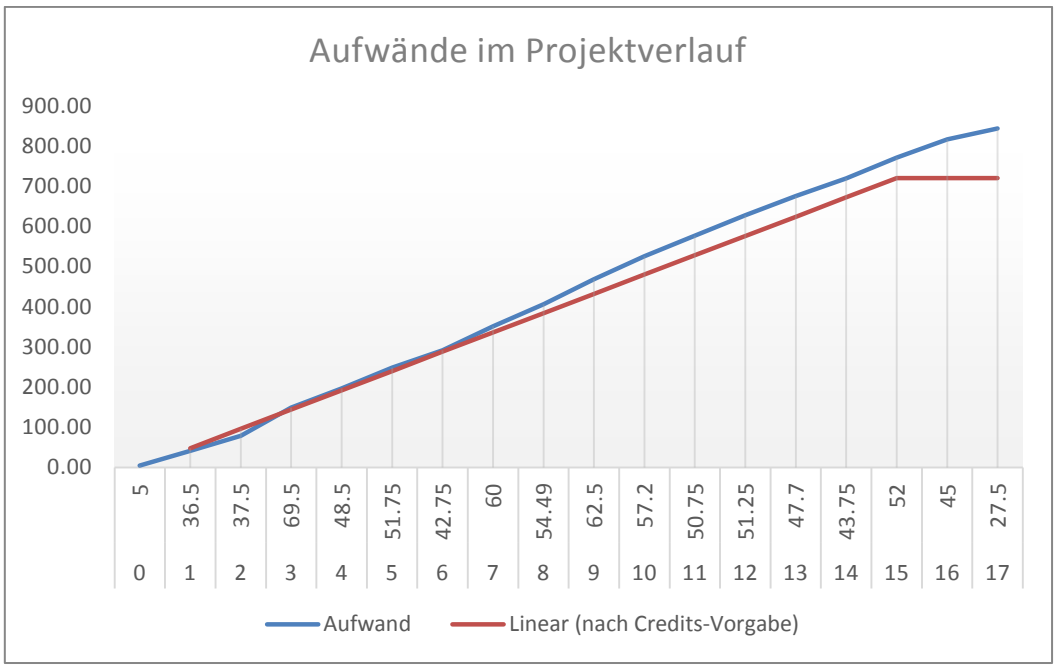


Diagramm 20: Aufwände im Projektverlauf

Durch die Planung der einzelnen Phasen, ergab sich schlussendlich eine Aufwandschätzung von 900 Stunden. Effektiv wurden 843 Stunden gearbeitet, dies ist eine Abweichung von lediglich 6%. Diese Abweichung mag sehr klein erscheinen und klingt vielleicht sogar unglaubwürdig. Tatsächlich ist die Abweichung aber wirklich so entstanden. Zurückzuführen ist diese relativ genaue Schätzung darauf, dass vor jeder Phase eine neue, detaillierte Planung für die einzelnen Arbeitspakete durchgeführt wurde. So konnte meist relativ genau geschätzt werden, wie gross der Aufwand für ein einzelnes Arbeitspacket war. Natürlich gab es trotzdem Abweichungen. Diese sind im Kapitel „Grösste Abweichungen“ aufgelistet und beschrieben.

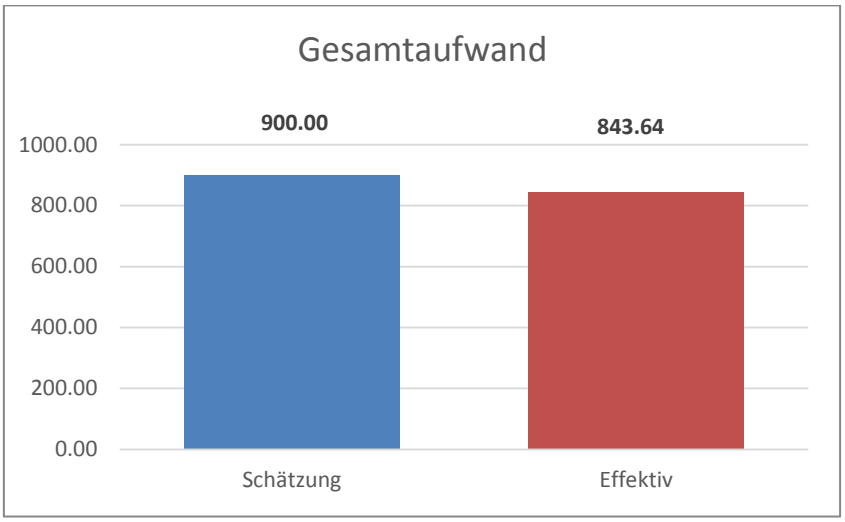


Diagramm 21: Projekt Gesamtaufwand

Das nächste Diagramm zeigt die Aufwände im Vergleich zur Planung für die einzelnen Phasen auf. Auch hier sind keine grossen Abweichungen ersichtlich. Gut sichtbar ist jedoch, dass gegen den Schluss hin die Planung eher zu hoch angelegt war, da wir eine gewisse Reserve eingeplant hatten.

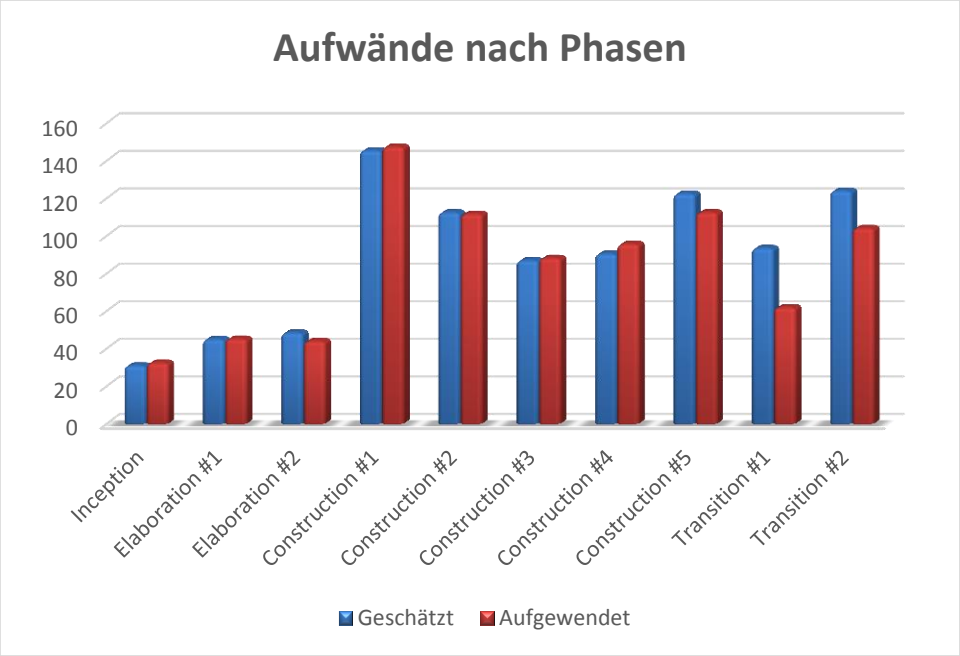


Diagramm 22: Aufwände nach Phasen

Das letzte Diagramm zeigt die jeweiligen Aufwände, verteilt auf die einzelnen Phasen und gruppiert nach der Art der Aktivität, auf. Diese Auswertung konnte dank der in Redmine geführten Zeiterfassung so genau erstellt werden. Klar ersichtlich ist, dass der mit Abstand grösste Anteil unserer aufgewendeten Zeit für die Entwicklung des Apps genutzt wurde. Mit dem Testing wurde ab ca. der Mitte des Projektes begonnen. Allerdings ist die aufgewendete Test-Zeit über das gesamte Projekt hinweg eher knapp bemessen. Im Nachhinein betrachtet hätten wir dieser Aktivität mehr Zeit widmen sollen.

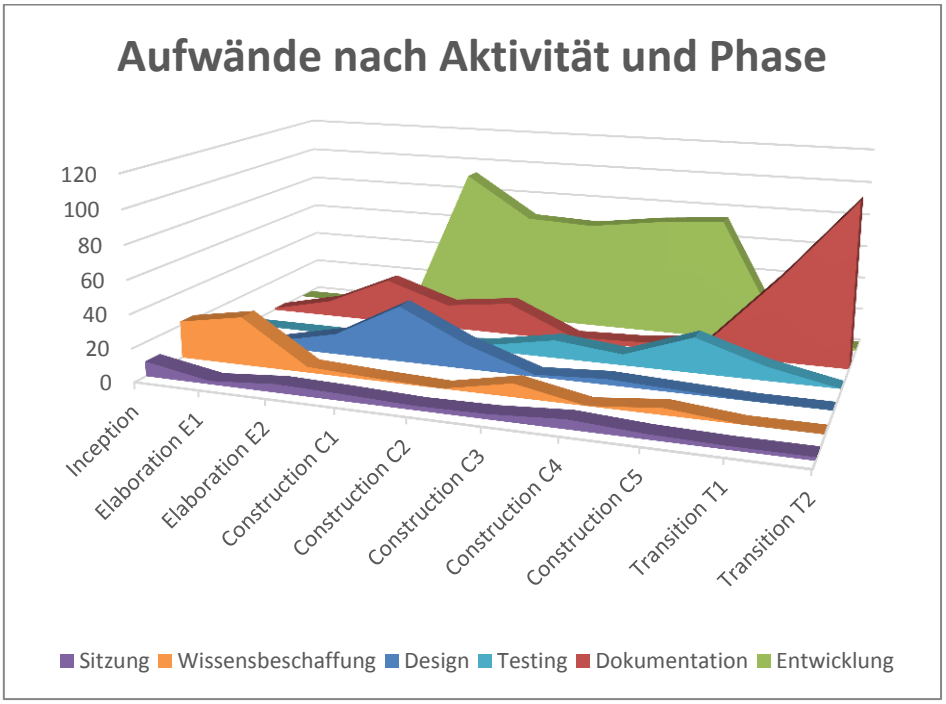


Diagramm 23: Aufwände nach Aktivität und Phase

3.2.2 Grösste Abweichungen

Wirkliche Abweichungen sind eigentlich nur in den letzten drei bzw. zwei Phasen ersichtlich. Wie schon erwähnt sind diese hauptsächlich auf eine absichtlich grössere Planung zurückzuführen, welche uns ein gewisses Reserve-Polster erbringen sollte. So wurde beispielsweise für das Video eine recht hohe Zeit einberechnet, da wir beide keine Erfahrung damit hatten, letztendlich war der Aufwand dann jedoch viel geringer.
In keinem Diagramm ersichtlich, jedoch auch bereits weiter oben beschrieben, wurde die Transition 2 ursprünglich als Reserve eingeplant. Wie aus dem obigen Diagramm hervorgeht, wurde dieses Ziel nicht erreicht, denn bereits während der Construction 5 wurde uns klar, dass wir in der Transition 2 sicher noch einige Abschlussarbeiten (Vervollständigung / Überarbeitung der Dokumentation, CD brennen usw.) erledigen müssten. Hinzu kamen auch noch das Video und die Erstellung der Abschlusspräsentation. Diese Arbeiten beanspruchten einiges an Zeit.

3.2.3 Grösste Schwierigkeiten

Die grössten Schwierigkeiten sind auf den Anfang des Projektes zurückzuführen. Einerseits war dies die Migration bzw. Einarbeitung von OpenGL und andererseits das Verstehen des bestehenden Quellcodes. Bereits während der Inception wurde uns klar, dass wir vor allem am Anfang sehr viel Aufwand erbringen müssen um unsere Ziele zu erreichen. Mit entsprechend hoher Planung und natürlich auch effektivem Aufwand waren damit vor allem die Construction 1, aber auch die Construction 2 verbunden. Wir wollten während dieser Zeit die Migration von OpenGL und das Refactoring grösstenteils abschliessen, um somit die grössten Probleme hinter uns zu lassen. Was uns auch gelang.
Während dem in den Construction 3 und 4 der Aufwand weniger wurde, kam nochmals ein grosses Stück Arbeit in der Construction 5 hinzu. Dies ist darauf zurückzuführen, dass wir unbedingt noch das optionale Feature „Geografische Namenssuche“ einführen und ein grosses Refactoring durchführen wollten. Das klare Ziel war, nach der Construction 5 keine Codeänderungen mehr vornehmen zu müssen. Dies wurde auch grösstenteils erreicht. Wegen den zusätzlichen Arbeiten wurde jedoch das Testing etwas vernachlässigt und somit nicht ganz so viele Unit Tests geschrieben wie ursprünglich geplant.

3.2.4 Arbeitspakete

Die Arbeitspakete stammen direkt aus dem Redmine. Aus Platzgründen sind die Beschreibungen teilweise abgeschnitten. Die Tickets können in Redmine mittels der Ticketnummer (erste Spalte) über die Suche direkt gefunden werden. Dort ist auch die komplette Beschreibung ersichtlich.

#	Ticket Beschreibung	Schätzung	Effektiv	Differenz	Abweichung
	Inception				
1	1. Sitzung / KickOff Meeting	2.00	4.00	2.00	100.00%
2	2. Sitzung mit P. Recher	4.00	5.00	1.00	25.00%
3	Infrastruktur einrichten	10.00	11.50	1.50	15.00%
9	Einlesen in Vorarbeiten / verwandten Arb...	5.00	3.00	-2.00	-40.00%
17	Analyse des bestehenden App-Codes (ohne ...	10.00	9.00	-1.00	-10.00%
		31.00	**32.50**		
	Elaboration E1				
4	Einlesen in OpenGL1 bzw. 2	30.00	29.25	-0.75	-2.50%
10	Grobplanung	6.00	5.50	-0.50	-8.33%
16	3. Sitzung, 05.03.2013, 14.00 Uhr	2.00	2.00	0.00	0.00%
26	Dokumentations-Vorlage mit Inhaltsstrukt...	4.00	6.00	2.00	50.00%
30	Risikoanalyse	3.00	2.50	-0.50	-16.67%
		45.00	**45.25**		
	Elaboration E2				
5	Analyse des bestehenden App-Codes (MIT Ä...	8.00	6.50	-1.50	-18.75%
28	Dokumentation des aktuellen Standes / Ar...	3.00	3.00	0.00	0.00%
34	Stand der Technik	6.00	7.25	1.25	20.83%
37	Detailplanung	5.50	5.00	-0.50	-9.09%
38	User Szenarien	5.00	3.75	-1.25	-25.00%
40	OpenStreetMap einlesen / einarbeiten	5.00	3.50	-1.50	-30.00%
56	Offizielles NeoMap Redmine aufräumen & d...	6.00	5.50	-0.50	-8.33%
62	Source-Code Analyse und Aufstellung von ...	10.00	9.50	-0.50	-5.00%
		48.50	**44.00**		
	Construction C1				
6	Migration von OpenGL 1 auf OpenGL 2	50.00	*45.50*	-4.50	-9.00%
25	Refactoring	16.00	14.00	-2.00	-12.50%
43	Bugs Behebung (ständige Abstürze, noch n...	6.00	6.00	0.00	0.00%
45	OpgenGL beschreiben / in Doku aufnehmen	8.00	6.50	-1.50	-18.75%
50	4. Sitzung, 12.03.2013, 14.00 Uhr	2.00	2.50	0.50	25.00%
59	Anzeige des eigenen Standortes auf der K...	5.00	4.00	-1.00	-20.00%
63	Dokumentation der Architektur nach Refac...	4.00	4.50	0.50	12.50%
64	Code Review C1	2.00	2.00	0.00	0.00%
66	5. Sitzung, 18.03.2013, 17.00 Uhr	2.00	1.50	-0.50	-25.00%
67	Neue Package Struktur definieren & umset...	8.00	7.00	-1.00	-12.50%
68	Überflüssiger Code/Klassen löschen	5.00	5.00	0.00	0.00%
71	Menü-Struktur und Verhalten an den Stand...	2.00	1.50	-0.50	-25.00%
73	Performance Analyse	3.00	3.50	0.50	16.67%
80	Delegates durch Observer Pattern ersetze...	5.00	3.50	-1.50	-30.00%
81	Koordinaten der aktuellen Position anzei...	15.00	14.50	-0.50	-3.33%
86	Performance Optimierung	4.50	5.50	1.00	22.22%
92	Vereinfachung der Map-Controller und Ren...	8.00	20.50	12.50	100.00%
		145.50	**147.50**		
	Construction C2				
7	Ladestrategie von Tiles ändern	5.00	5.00	0.00	0.00%
13	Fragments einsetzen wo sinnvoll	12.00	11.00	-1.00	-8.33%
18	Eigene Position auf Karte darf nicht ver...	6.00	5.50	-0.50	-8.33%
19	Gerät kippen für Schrägansicht.	10.00	9.50	-0.50	-5.00%
41	Zoom-Stufen Begrenzung beim Hineinzoomen	3.00	*2.50*	-0.50	-16.67%
46	Dokumentation Refactoring Vorher <-> Nac...	3.00	3.00	0.00	0.00%
70	TileDownload und GL zeichnen verbessern ...	18.00	17.50	-0.50	-2.78%

#	Ticket Beschreibung	Schätzung	Effektiv	Differenz	Abweichung
75	Testing auf verschiedenen Geräten	3.00	2.75	-0.25	-8.33%
89	6. Sitzung, 26.03.2013, 14.00 Uhr	1.50	1.50	0.00	0.00%
93	Download von NeoMap wird immer als erfol...	1.00	1.00	0.00	0.00%
94	SQL/ DBMethoden für Notizen erstellen	3.00	2.50	-0.50	-16.67%
95	Longtap Menü auf MapView erstellen, für ...	3.00	6.50	3.50	100.00%
96	Zu weit hinausscrollen == permanter App ...	2.00	2.00	0.00	0.00%
97	7. Sitzung, 02.04.2013, 14.00 Uhr	2.00	1.00	-1.00	-50.00%
98	Notizen erfassen	10.00	9.50	-0.50	-5.00%
103	Code Review C2	2.00	2.00	0.00	0.00%
104	Darestellen von Notizen mittels Icon auf...	6.00	4.00	-2.00	-33.33%
105	Auswahl einer bestehenden Notiz auf der ...	5.00	8.00	3.00	60.00%
106	Doku Notizen	4.00	4.00	0.00	0.00%
107	Doku onTap (visible Target / delta posti...	1.50	1.50	0.00	0.00%
108	Doku neue Renderer / Kontroller Struktur	10.50	10.50	0.00	0.00%
109	Doku Management / Generic / Managablelte...	1.00	1.00	0.00	0.00%
		112.50	**111.75**		
	Construction C3				
20	Geometrie erfassen	5.00	*5.00*	0.00	0.00%
99	Basic Ant-Build Script für das Erstellen...	3.00	*2.50*	-0.50	-16.67%
101	Unit-Tests für externe Webservice Aufruf...	5.00	*5.00*	0.00	0.00%
124	8. Sitzung, 08.04.2013, 14.00 Uhr	2.00	*2.00*	0.00	0.00%
125	UserSzernarios / Tests ausarbeiten	4.00	*4.00*	0.00	0.00%
126	Note Klasse umschreiben / Generic	4.50	*5.50*	1.00	22.22%
127	DB Erweiterung für Points/Liniene bzw. G...	4.00	*4.00*	0.00	0.00%
132	Andere Schritart für Koordinatenbild neh...	2.00	*1.25*	-0.75	-37.50%
133	Horizont in Schrägansicht andeuten	8.00	*9.50*	1.50	18.75%
134	Dialog wenn man ein "EditFragment" offen...	2.00	*2.00*	0.00	0.00%
138	MapEditModus / contextual Action Bar für...	10.00	*10.25*	0.25	2.50%
140	Doku Fragment	2.00	*1.00*	-1.00	-50.00%
152	9. Sitzung, 16.04.2013, 14.00 Uhr	2.00	2.00	0.00	0.00%
154	Anwählen von Punkten in der Schrägansich...	7.00	8.50	1.50	21.43%
155	Bessere Icons suchen & Skalierung ändern	12.00	12.00	0.00	0.00%
160	Drag & Drop Library integrieren	2.00	3.00	1.00	50.00%
163	Dokumentation NeoMaps intuitiv priorisie...	2.00	1.25	-0.75	-37.50%
165	Verwendeung eines deprecated Sensor Type...	5.50	5.00	-0.50	-9.09%
188	Icons auf der Karte sollen in Richtung d...	5.00	4.50	-0.50	-10.00%
		87.00	**88.25**		
	Construction C4				
21	Neomaps intuitiv ein und ausblendbar mac...	10.00	10.00	0.00	0.00%
39	Erfasste Geometrien als GPX exportieren	10.00	9.45	-0.55	-5.50%
42	Generelles Error-Handling bei nicht verf...	4.00	2.00	-2.00	-50.00%
85	URL's von Web-Service Calls in ein zentr...	1.50	0.75	-0.75	-50.00%
128	Zeigen ob GPS Empfang vorhanden ist	2.00	2.00	0.00	0.00%
168	Login funktioniert nicht	6.00	6.50	0.50	8.33%
170	Preference Keys in XML verschieben (anst...	1.00	1.00	0.00	0.00%
171	Aktive Icons für Actionbar erstellen und...	0.50	0.50	0.00	0.00%
172	Aktuelle Modus (3d, compass, etc.) in se...	0.50	0.50	0.00	0.00%
173	Erweiteretes Memory Management um OutOfM...	20.00	19.00	-1.00	-5.00%
174	10. Sitzung, 23.04.2013, 14.00 Uhr	2.00	2.00	0.00	0.00%
175	Action Bar für Export einführen / Layou...	3.00	10.75	7.75	100.00%
186	Warnung/Dialog anzeigen bei offenen Ände...	1.00	1.50	0.50	50.00%
187	Bei nicht verfügbarer Internet-Connectio...	3.00	3.00	0.00	0.00%
189	Doku: Laden von Grafiken und verschieden...	4.50	4.50	0.00	0.00%
191	Überarbeitung des LocationControlelrs (...	4.50	4.50	0.00	0.00%
192	11. Sitzung, 30.04.2013, 14.00 Uhr	2.00	2.00	0.00	0.00%
193	Erweiterung um verschiedenen OSM Basis K...	10.00	11.00	1.00	10.00%

#	Ticket Beschreibung	Schätzung	Effektiv	Differenz	Abweichung
194	Testen von exportierten GPX-Files in ver...	2.00	*1.75*	-0.25	-12.50%
208	Kurz-Präsentation für Kurzbetreuer vorbe...	3.00	3.00	0.00	0.00%
		90.50	**95.70**		
	Construction 5				
84	Resourcen Files und Drawable Files aufrä...	2.00	2.00	0.00	0.00%
129	Manifest überarbeiten (vor allem Berecht...	0.75	0.75	0.00	0.00%
130	Ggf. Streckenmessung einführen	10.00	8.50	-1.50	-15.00%
136	Diverse Kleinigkeiten (Mini-Todolist)	1.50	1.50	0.00	0.00%
197	Über Mögliche Einbindung recherchieren /...	2.00	2.00	0.00	0.00%
198	Reverse Suche (über Koordinaten)	10.00	10.00	0.00	0.00%
201	12. Sitzung mit Gegenleser vom 7.05.2013...	2.00	3.00	1.00	50.00%
202	Normale Namenssuche	10.00	10.00	0.00	0.00%
203	Bessere Icons suchen	2.00	2.75	0.75	37.50%
204	Geografische Namenssuche	4.00	5.50	1.50	37.50%
206	Unit Tests	20.00	11.00	-9.00	-45.00%
207	User Tests	13.00	12.00	-1.00	-7.69%
212	In den Einstellungen Möglichkeit zum lös...	1.25	1.50	0.25	20.00%
213	RSS News Feed in App integrieren	10.00	7.00	-3.00	-30.00%
214	Refactoring	23.00	27.00	4.00	17.39%
217	Doku Export	3.00	1.00	-2.00	-66.67%
218	Doku Nominatim	5.00	4.00	-1.00	-20.00%
224	ZoomButton einführen	2.00	2.00	0.00	0.00%
227	Doku News	1.00	1.00	0.00	0.00%
		122.50	**112.50**		
	Transition 1				
23	Tests auf verschiedenen Geräten	30.00	8.50	-21.50	-71.67%
29	Dokumentation überarbeiten	20.00	17.50	-2.50	-12.50%
33	Management Summary	6.00	7.50	1.50	25.00%
35	Abstract	5.00	4.00	-1.00	-20.00%
36	Projektmonitoring (Schlussbericht)	20.00	14.00	-6.00	-30.00%
117	Doku Kartographie / Openstreetmap inkl. ...	2.50	2.50	0.00	0.00%
222	Doku Android Basics	3.00	3.00	0.00	0.00%
231	Doku NeoMaps	4.00	3.00	-1.00	-25.00%
233	13. Sitzung 21.05.2012 um 14.00 Uhr	3.00	2.00	-1.00	-33.33%
		93.50	**62.00**		
	Transition 2 / Reserve				
32	CD Erstellen	4.00	1.50	-2.50	-62.50%
44	Allgemeiner Dokumentations-Aufwand	75.00	69.69	-5.31	-7.08%
239	Poster	5.00	5.00	0.00	0.00%
240	Video	20.00	16.00	-4.00	-20.00%
241	Druck & Abgabe	6.00	*4.00*	-2.00	-33.33%
242	Abschluss Präsentation erstellen	12.00	6.00	-6.00	-50.00%
244	14. Sitzung, 15:30 Uhr	2.00	2.00	0.00	0.00%
		124.00	**104.19**		
	Total	**900.00**	**843.64**	**-56.36**	**-6.26%**

Regeln für Abweichungs-Indikator:

Grün: bis 25% Abweichung
Orange: bis 25-45% Abweichung
Rot: ab 45% Abweichung

3.2.4.1 Soll- / Ist-Vergleich

Insgesamt hat sich eine Differenz von 6% zwischen Aufwandschätzung und effektiv aufgewendeter Arbeitszeit ergeben. Somit wurde etwas mehr Zeit eingeplant, als schlussendlich benötigt wurde. Mit diesem Ergebnis sind wir persönlich sehr zufrieden.
Die grössten Abweichungen sind zumeist bei kleinen Paketen, d.h. Pakete mit einem Aufwand von ca. 1 -2 Stunden. Dabei ergibt sich natürlich rasch eine Abweichung von 100%, z.B. wenn statt einer Stunde, zwei Stunden gebraucht werden. Diese Abweichungen beachten wir daher als nicht weiter tragisch.

Grosse Abweichungen traten bei folgenden Paketen auf:

- **#92 - Vereinfachung der Map-Controller und Renderer**
  - *Zeiten:*
    8h geschätzt, 20.5h effektiv → 100% Abweichung
  - *Grund:*
    Die Vereinfachung der Controller- und Renderer-Struktur stellte sich als viel schwieriger heraus als angenommen. Es bestand eine sehr hohe Kopplung zwischen diesen Klassen. Bei der Vereinfachung funktionierten dann immer wieder gewisse andere Dinge nicht mehr, wie z.B. das OSMGrid. Die Vereinfachung musste mehrmals überarbeitet und angepasst werden, was mit einem grossen zusätzlichen Aufwand verbunden war.
- **#175 - Action Bar für Export einführen / Layout ändern**
  - *Zeiten:*
    3h geschätzt, 10.75h effektiv → 100% Abweichung
  - *Grund:*
    Die Action Bar selber war sehr schnell eingeführt, jedoch ergaben sich schnell viele Anzeige sowie funktionale Probleme. So wurden beispielsweise einige Icons nicht korrekt angezeigt und die „Teilen"-Funktion hatte ein etwas sonderbares Verhalten. Für diese Probleme wurde auch ein Beitrag auf StackOverflow (http://stackoverflow.com/questions/16200242/contextual-action-bar-weird-behavior-for-menuitem) erstellt, welcher schlussendlich dann aber von uns selber beantwortet wurde.
- **#206 - Unit Tests / #23 - Tests auf verschiedenen Geräten**
  - *Zeiten:*
    20h geschätzt, 11h effektiv → 45% Abweichung bzw.
    30h geschätzt, 8.5h effektiv → 71% Abweichung
  - *Grund:*
    Hierfür gab es vor allem zwei Gründe. Einerseits wurden Tests oft direkt mit dem damit verbundenen Einbau von Features durchgeführt und wurden deshalb bereits zu anderen Zeitpunkten durchgeführt bzw. erstellt. Andererseits wurden nicht so viele Tests umgesetzt wie ursprünglich geplant, dies vor allem, weil sich das Testing als sehr schwerfällig und mühsam gestaltete. Vorwiegend durch die Verwendung von OpenGL und Android-GUI-Komponenten, welche beide das Testing erheblich erschweren. Um Testing-Frameworks zu evaluieren und einzuführen, fehlte schlussendlich die Zeit.
- **#36 - Projekt Monitoring (Schlussbericht)**
  - *Zeiten:*
    20h geschätzt, 14h effektiv → 30% Abweichung
  - *Grund:*
    Wir konnten viele Diagrammtypen usw. direkt von der Studienarbeit übernehmen. Was natürlich eine enorme Zeitersparnis ausmachte. Auch wurden bei der Studienarbeit bereits SQL-Queries für die Auswertung aus dem Redmine erstellt, diese konnten auch grösstenteils übernommen werden.

3.3 Git-Statistiken

Normalerweise sind Git-Statistiken in unseren Augen nicht wirklich aussagekräftig. Eine spezielle Statistik soll an dieser Stelle jedoch dokumentiert werden, da sie den Verlauf unseres Projektes stark reflektiert. Das Diagramm wurde mit dem Tool „gitstats“ erstellt.

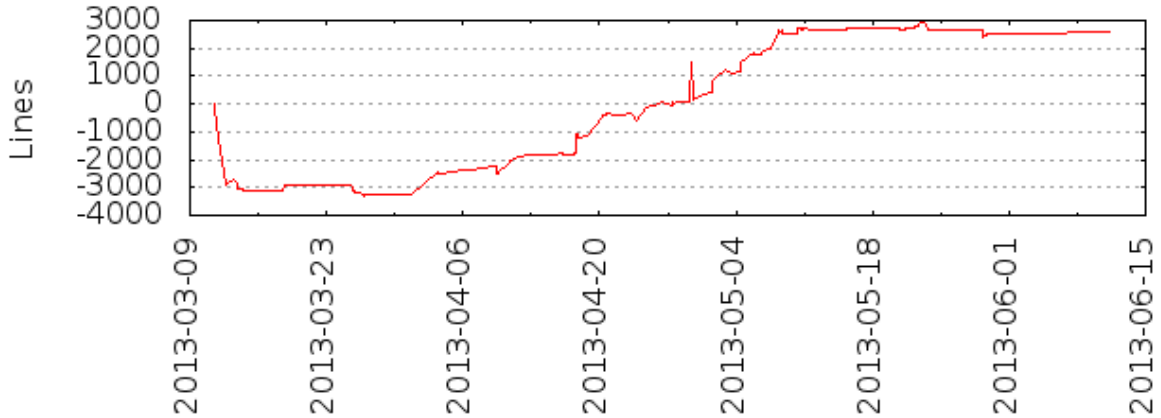


Diagramm 24: Git Statistik: Veränderung der Lines Of Code

Im Diagramm ist nur das Delta der „Lines of Code“, über die Spanne dieses Projektes, ersichtlich. In diesem Sinne wurde bei einem Line-Delta von 0 gestartet.
In den ersten Phasen, d.h. fast bis Mitte der Construction 2, wurde „DeadCode“ und nicht voll funktionsfähiger Code entfernt, sowie das Projekt generell aufgeräumt und umstrukturiert. Wie im Diagramm ersichtlich wurden ca. 3000 Zeilen-Code gelöscht! Erst dann wurde damit begonnen, neue Features einzubauen. Dabei wurden etwa 6000 Zeilen Code eingeführt (die modifizierten Zeilen sind hier nichtaufgelistet). Dies wurde bis etwa zum 12. Mai – Mitte Construction 5 - weitergeführt. Die geraden und abfallenden Linien dazwischen und vor allem am Schluss, stellen immer wieder Refactoring-Teile dar. Durch die Statistik ist auch klar ersichtlich, dass ab Mitte der Construction 5 nur noch kleine Refactorings und Fehlerbehebungen durchgeführt wurden. Ab diesem Zeitpunkt wurden keine neuen Features mehr eingebaut.

3.4 Codestatistik

3.4.1 Übersicht

Mittels des Tools Stan4J konnten die Aufrufe zwischen den verschiedenen Packages ausgewertet werden. Hierbei wurde Wert darauf gelegt, dass keine bidirektionalen Aufrufe zwischen dem Presentation-Layer und dem Data- sowie Application-Layer stattfinden, um eine saubere Trennung des Presentation-Layers zu erhalten. Im Nachfolgenden, von Stan4J generierten, Dependency-Graph wird dies noch grafisch aufgezeigt.

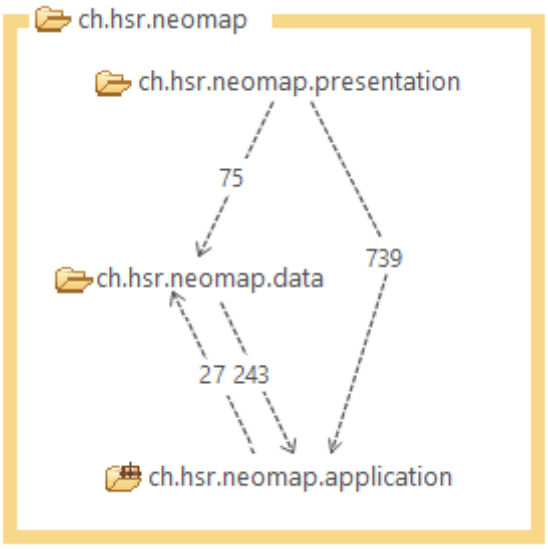


Abbildung 67: Dependency-Graph

Die Auswertung der „Lines of Code“ über das gesamte Projekt, wurde zusätzlich nach Layer und File-Type aufgeteilt. Auch ersichtlich sind die Anzahl Packages sowie die Anzahl an Klassen.

Layer	Lines of Code	Number of Packages	Number Of Types (Classes)
Presentation		7	92
Java	4287		
XML	1151		
Application		7	87
Java	4967		
Data		3	22
Java	1774		
Gesamt	**12179**	**20**	**201**

Tabelle 43: Lines of Code

In der Nachfolgend Grafik wird die Verteilung der Codezeilen auf die verschieden Packages grafisch dargestellt.

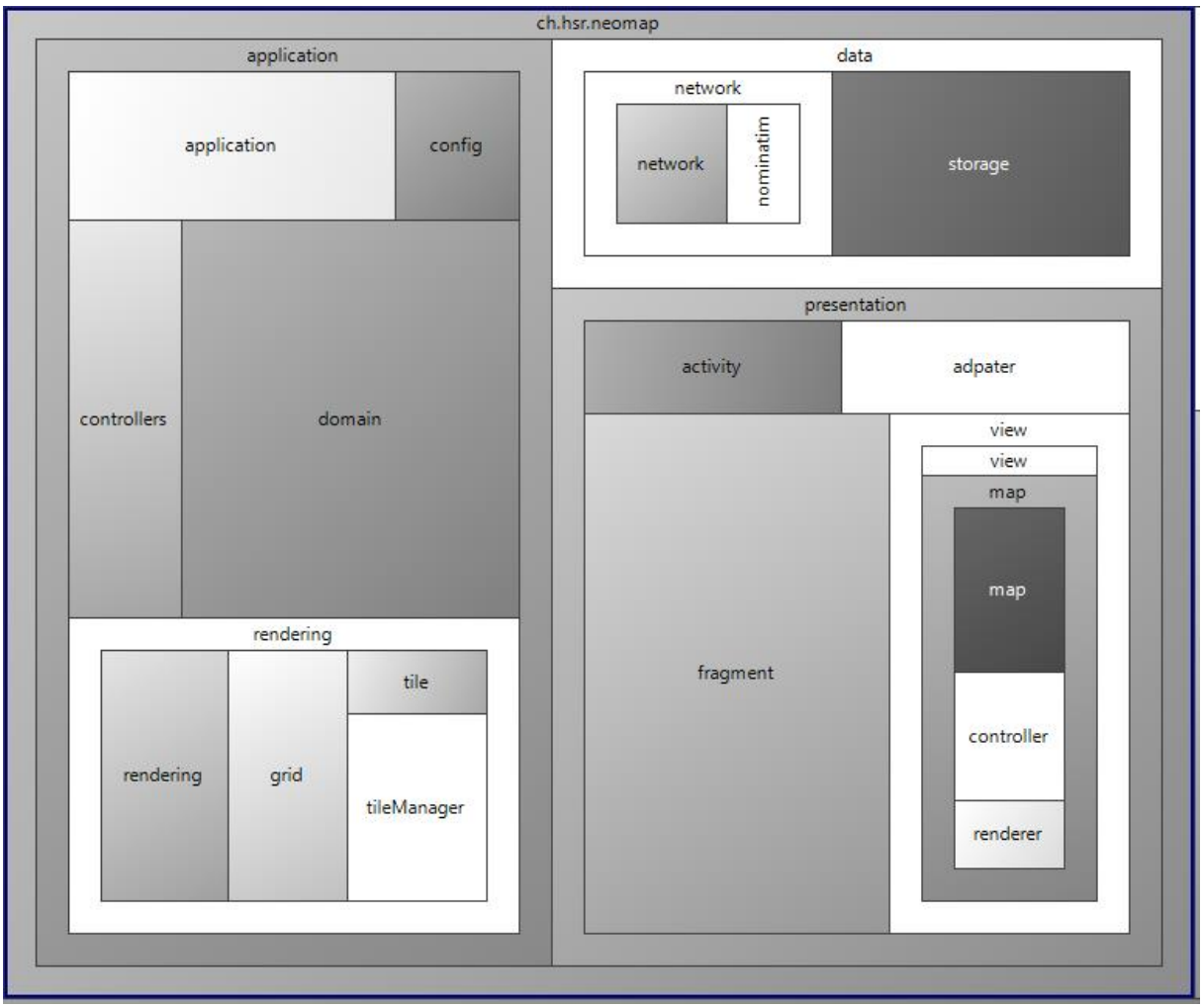


Abbildung 68: Verteilung der Codezeilen auf Packages

3.4.2 Code-Metriken

3.4.2.1 Cyclomatic Complexity

Der Messwert Cyclomatic Complexity beschreibt die Komplexität einer Methode anhand des Kontrollflusses. Die Komplexität einer Methode steigt mit der zunehmenden Anzahl von möglichen Kontrollflüssen, respektive mögliche Entscheidungen innerhalb der Methode. Entscheidungen sind dabei „if“, „while“, „switch“ Statements usw. Der niedrigste mögliche Wert hierbei ist jeweils 1 bei einer leeren Methode.
Alle Methoden im gesamten Projekt halten sich hier im Bereich von einer Komplexität bis 10 auf. Erfahrungsgemäss weisen Methoden mit höheren Werten meist eine hohe oder zu hohe Komplexität auf und sollten bei Bedarf vereinfacht werden.
Bei der nachfolgenden Grafik handelt es sich um eine Häufigkeits-Analyse der berechneten „Cyclomatic Complexity“ aller im Projekt vorhandenen Methoden.

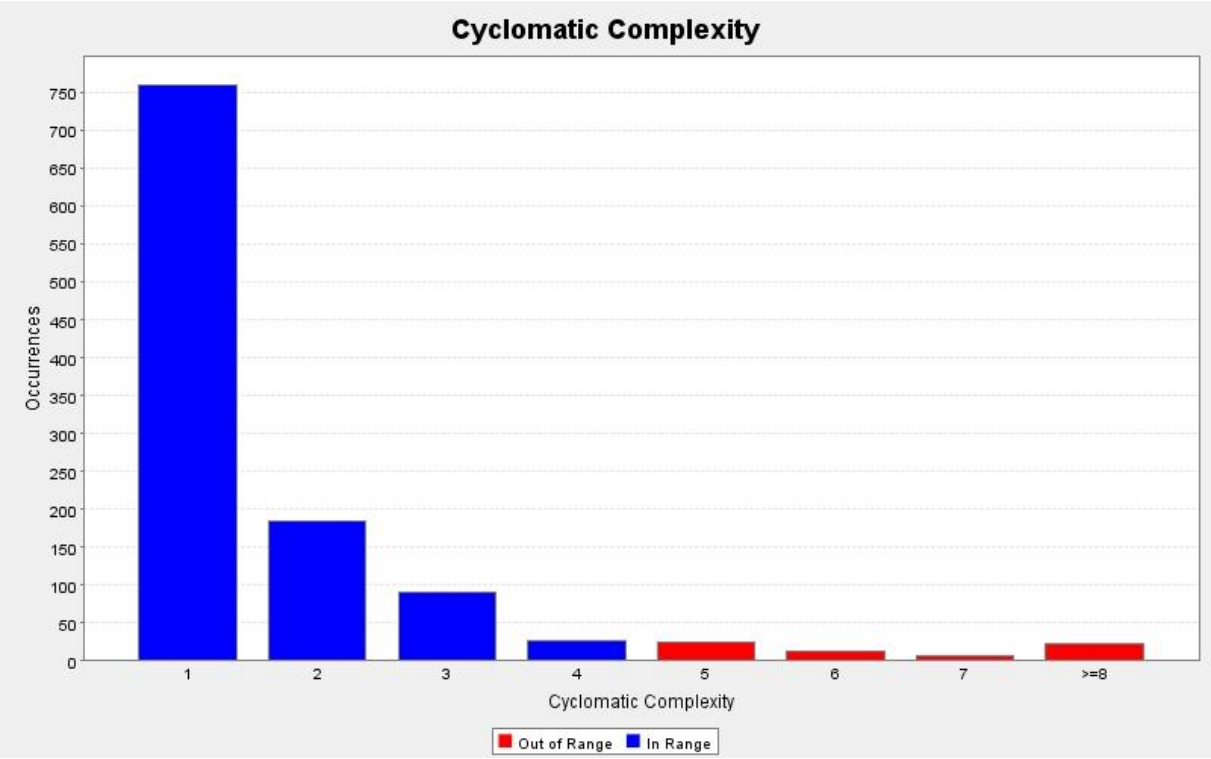


Abbildung 69: Cyclomatic Complexity

3.4.2.2 Efferent Coupling

Efferent Coupling misst den Grad der Abhängigkeit einer Klasse von weiteren Klassen. Dies beinhaltet Vererbung, Interface-Implementationen, Parameter-Typen oder auch Variablen-Typen. Ein hoher Efferent-Coupling-Wert zeigt an, dass eine Klasse nicht sehr fokussiert ist und aufgrund der vielen Abhängigkeiten, sehr anfällig gegenüber externen Änderungen sein kann. Das nachfolgende Histogramm zeigt die Verteilung der Efferent-Couplings aller im Projekt vorhandenen Klassen auf. Wie zu sehen ist, sind die Werte meistens tief und somit in Ordnung.

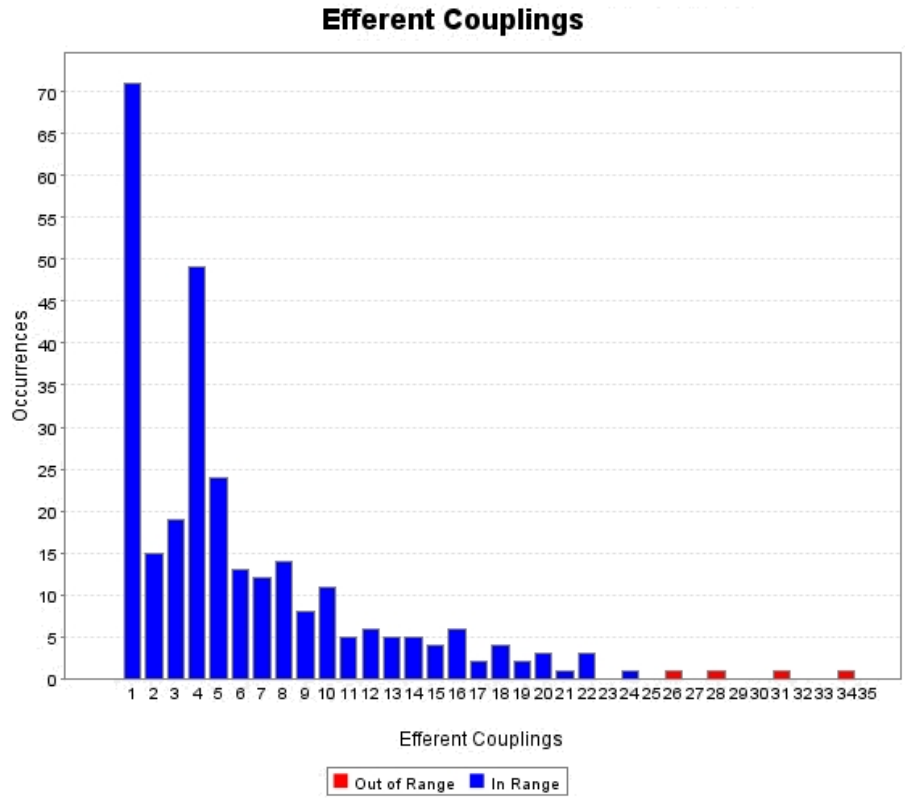


Abbildung 70: Efferent Coupling

Teil IV
Anhänge

1 Inhalt der CD

In der folgenden Tabelle findet sich der Inhalt der CD. *Kursiv* geschriebene Pfade stellen dabei Dateien dar, währenddem alle übrigen Pfade Ordner sind.

Pfad	Beschreibung
App	
KeyStore	Signier-Schlüssel sowie Anleitung zum Signieren der App
Source	Sourcen der App. Dies ist ein „git clone“ des offiziellen Repositories
NeoMapAppVersion3.0.apk	Die aktuelle und signierte Version des Apps
Bilder	
Assets	GIMP-Projektdateien etc. welche teilweise für Bilder in der App gebraucht werden
Broschüre	Alle Bilder welche in der HSR-Broschüre benutzt wurden
Screenshots	Verschiedene Screenshots, welche direkt auf einem Gerät (Nexus 4) gemacht wurden
Diagramme	
_edit	Editierbare Astah Community-Dateien
Diagramme als PNG-Dateien	
Dokumentation	
_edit	Dokumentation als „docx“ Datei (Office 2010) sowie alle Excel-Dateien (hauptsächlich Diagramme für Statistiken usw.)
Titelblatt_HSR.pdf	
NeoMap App.pdf	Gesamtdokumentation als PDF
Poster	
_edit	Editierbare InDesign (CS6) Datei des Posters sowie Bilder die für das Poster verwendet wurden
Poster.pdf	
Video	
NeoMap App.wmv	

Tabelle 44: Inhalt der CD

2 Glossar und Abkürzungsverzeichnis

Begriff	Erläuterung
API	Ein Application Programming Interface ist eine Schnittstelle, welche von einer Softwarekomponente zur Verfügung gestellt wird, um die Kommunikation mit ihr zu ermöglichen.
DalvikVM	Die Dalvik Virtual Machine ist eine für mobile Geräte entwickelte Virtual-Machine und eine der Hauptkomponenten der Android-Plattform
Fragment (Android)	Wiederverwendbare Ansicht einer Activity in Android. Mehre Fragments können kombiniert dargestellt werden, so kann bei Geräten mit grösserem Bildschirm (z.B. Tablets) mehr angezeigt werden.
Git	Verteiltes Versionsverwaltungssystem, welches mit zwei Stufen von Repositories arbeitet, nämlich lokal und entfernt.
GLES	Abkürzung für OpenGL for Embedded Systems
GPX	GPS Exchange Format Ein definiertes Dateiformat zur Speicherung von Geodaten. Es dient dem Zweck, GPS Daten zwischen verschiedenen Programmen austauschen zu können.
JavaVM	Die Java-Virtual-Machine wird für die Ausführung von Java-Bytecode benötigt.
OpenGL ES	Open Graphics Library for Embedded Systems
OSM	OpenStreetMap ist ein freies / OpenSource Kartenprojekt welches eine alternative zu Google Maps etc. bieten soll. Dabei kann jeder mit helfen Daten zu sammeln und die OSM zu verbessern.
POI	Point of Interests sind Orte welche für der Nutzer einer Karte eine Bedeutung / einen Nutzen haben könnten, wie z.B. eine Tankstelle oder ein Bankomat.
Refactoring	Bezeichnet die Tätigkeit der Restrukturierung von existierendem Quellcode, ohne dabei das Verhalten der Applikation zu verändern. Wird durchgeführt in der Software-Entwicklung, um die nicht-funktionalen Eigenschaften einer Applikation, häufig die Struktur des Quellcodes, zu verbessern.
Renderer	Im Rahmen dieses BA-Dokumentes steht ein Renderer immer für eine Klasse, welche sich um das Darstellen von einer Karte bzw. einer Textur kümmert.
RUP	Rational Unified Process – ist ein Vorgehensmodell für die Softwareentwicklung.
Shader	Shader sind Softwaremodule, welche bestimmte Bildsynthese-Effekte bei 3D-Grafiken implementieren.
Slippy Map	Beschreibt die Karten-Art, bei welcher ein Kartenausschnitt aus mehreren Tiles aufgebaut wird.
Tile	Ein Tile ist in diesem Fall immer eine kleine (quadratische) Kachelgrafik einer Karte. Eine komplette Karte wird aus vielen Tiles zusammengesetzt.
Vertex	Ein Vertex entspricht einem geometrischen Punkt, welcher durch zwei oder drei (3D) Koordinaten definiert ist
VM	Eine Virtual-Machine ist ein virtueller Computer, welcher in einer dafür vorgesehenen Laufzeitumgebung laufen gelassen werden kann.
Waypoint	Ein Waypoint ist eine elektronische Marke, welche eine eindeutige Positionsangabe mittels Koordinaten auf der Erde zulässt.

Tabelle 45: Glossar

3 Tabellenverzeichnis

4 Abbildungsverzeichnis

5 Diagrammverzeichnis

6 Quellcodeverzeichnis

7 Literatur- und Quellenverzeichnis

[1] M. Smithwick and M. Verma, Pro OpenGL ES for Android, Apress, 2012.

[2] Google I/O 2013: Keynote, «youtube.com,» 15 Mai 2013. [Online]. Available: http://www.youtube.com/watch?v=9pmPa_KxsAM. [Zugriff am 15 Mai 2013].

[3] K. Brothaler, OpgenGL ES for Android - A Quick-Start Guide, The Pragmatic Bookshelf, 2013.

[4] M. Zechner und R. Green, Beginning Android Games, Second Edition, Apress, 2012.

[5] K. Brothaler, «Learn OpenGL ES,» [Online]. Available: http://www.learnopengles.com/. [Zugriff am 16 März 2013].

[6] D. Bomfim, «All about OpenGL ES 2.,» [Online]. Available: http://db-in.com/blog/2011/01/all-about-opengl-es-2-x-part-13/. [Zugriff am 3 März 2013].

[7] P. Rideout, iPhone.3D.Programming, O'Reilly Media, 2010.

[8] R. Meier, Professional Android 4 Application Development, John Wiley & Sons, 2012.

[9] «Android Developers - Platform Versions,» 2013. [Online]. Available: http://developer.android.com/about/dashboards/index.html#Platform. [Zugriff am 1 März 2013].

[10] F. Ramm und J. Topf, OpenStreetMap, Lehmanns, 2010.