Projektgruppe 3D-Spieleprogrammierung auf mobilen Endgeräten

Philipp Bertram, Sascha Henke, Sergiy Krutykov, Daniel Künne, Andre Schemschat, Sebastian Stock, Peer Wagner

Universität Osnabrück

Die Welt der mobilen Telefone wächst rasant. Die immer leistungsfähiger werdenden Geräte ermöglichen es anspruchsvolle Applikationen zu entwickeln, die hohe Anforderungen an die Hardware stellen. Einen großen Anteil der Applikationen machen dabei Spiele aus, wobei sich die Entwickler häufig auf die Portierung bekannter Desktop-Spiele beschränken. Dies hat zur Folge, dass einerseits die Spiele gegenüber dem Original stark an Qualität verlieren, da sie mit der beschränkten Hardware auskommen müssen und andererseits die typischen Eigenschaften solcher Geräte nicht ausgenutzt werden. Die modernen Smartphones bieten jedoch eine Vielzahl neuer, bisher nicht vorhandener Eigenschaften. Hierzu zählen unter anderem Kameras, Beschleunigungssensoren, GPS und mobiles Internet.

Das Ziel der Projektgruppe *3D-Spieleprogrammierung auf mobilen Endgeräten* war es daher möglichst viele Eigenschaften der Smartphones zu nutzen und zu zeigen, wie sich diese Eigenschaften sinnvoll nutzen und kombinieren lassen. Das Ergebnis dieser Studie ist ein Spiel, das diese Anforderungen in sich vereint und eine der Möglichkeiten darstellt.

8.4.2011

Inhaltsverzeichnis

1. Spielidee und Einleitung

Sascha Henke

Grundbestandteil des Projekts ist es alle Sensoren, die in einem Smartphone vorhanden sind, wie *GPS*, *Kamera*, *Lagesensor* und *Kompass*, in einem Spiel zu nutzen. Durch die Möglichkeit GPS-Daten auslesen zu können, entstand die Idee das Konzept des *Geocachings* teambasiert auf Smartphones zu portieren. So kam es recht schnell zu der Vorstellung Teams zu bilden, welche sich zusammenfinden um, *Schätze* zu suchen, die von einem zentralen Server verteilt werden. Als primäre Navigationsmöglichkeit sollen Karten angezeigt werden, in denen man neben der eigenen Position eventuelle Mitspieler sowie Schätze unter den richtigen Umständen ausfindig machen kann. Weiterhin soll der "Multiplayer-Aspekt" verstärkt werden, indem zusätzlich zu den "Schätzen" auch noch Gegenschätze verteilt werden die in das Spielgeschehen eingreifen bzw. andere Mitspieler beeinflussen. Als weiteres Spielelement soll eine *Augmented Reality* eingebaut werden. So soll, falls man sich in der Nähe eines Gegenstandes oder Schatzes befindet, ein dreidimensionales Bild von diesem möglichst perspektivisch über das Kamerabild gelegt werden, so dass der Eindruck entstehe das Spielelement sei auch in der Realität vorhanden.

Über die Entwicklungszeit entstand die Idee das Spielprinzip in eine "Räuber und Gendarm" Umgebung zu bringen. So finden sich die Spieler in den Teams "Räuber" und "Polizisten" ein und wetteifern um eine begrenzte Ressource, welche als Tresor dargestellt werden. Den Räubern geht es nun darum die verteilten Tresore zu ergattern und den Inhalt an sich zu bringen. Die "Polizisten" versuchen hingegen die Tresore zu sichern, sodass diese für die Räuber nicht mehr zu finden sind. Um ein weiteres forderndes Element dem Spiel hinzuzufügen, muss beim Sichern bzw. Knacken eines Tresors ein Minispiel gelöst werden. Spielbeeinflussende Gegenstände werden in Form von Geschenken, welche einem Nutzer auf der Karte immer sichtbar sind und einen zufälligen, von Außen nicht sichtbaren, nützlichen Inhalt besitzt.

2. Spielanleitung

Phillipp Bertram

Run4Gold zeigt, wie die Funktionen und Eigenschaften moderner Smartphones verpackt in einem Spiel miteinander kombiniert werden können. GPS, Internet und diverse Sensoren sind einige dieser Eigenschaften, aus denen letztendlich unsere Spielidee entstanden ist. *Run4Gold* ist keine reine Schnitzeljagd, in der es gilt Hinweisen zu folgen; es differenziert sich dennoch stark vom weit verbreiteten *Geocaching*[1].

Die folgenden Abschnitte erläutern die Funktionen des Spiels. Zunächst wird das Anlegen eines neuen Benutzer-Accounts mit entsprechendem Login exerziert. Anschließend wird das Beitreten und Anlegen eines Spiels gezeigt, sowie die das eigentliche Spiel vorgestellt.

2.1 Startseite

Phillipp Bertram

Abb. 1: Startbildschirm

Wird die Applikation gestartet, gelangt der Benutzer zunächst auf die in der Abbildung gezeigte *Startseite*. Die Applikation speichert während der Benutzung gewisse Daten, lädt und verifiziert diese beim Server, damit der Benutzer beim Starten nicht ständig seine Daten erneut eingeben muss. Daher kann der Benutzer schon an dieser Stelle über `Resume` direkt sein Spiel fortsetzen, sofern er sich derzeit in einem befindet. Hierbei gelangt er zu den im späteren Abschnitt beschriebenen *Userinformationen*. Ist dies nicht der Fall wird `Login` statt `Resume` angezeigt, sodass der Benutzer zum Login aufgefordert wird.

Darüber hinaus können über `Options` einige Einstellungen vorgenommen, sowie über `Help` und `About` diverse Informationen zum Spiel bzw. zur Projektgruppe aufgerufen werden (siehe Abschnitt *Options, Help, About*).

2.2 Login & Register

Phillipp Bertram

1 http://www.geocaching.de/index.php?id=74

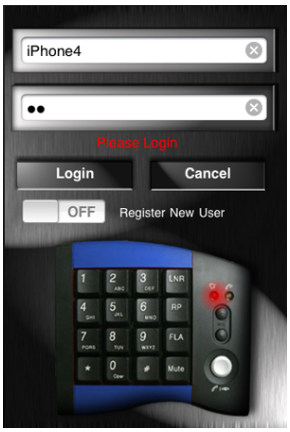


Abb. 2: Start

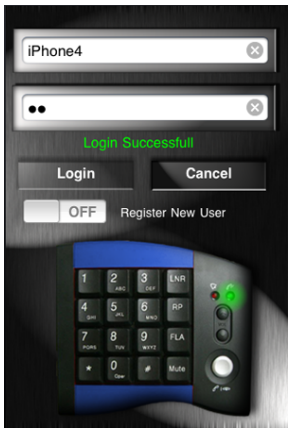


Abb. 3: Successfull

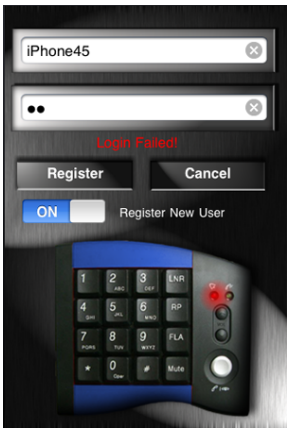


Abb. 4: Failed & Register

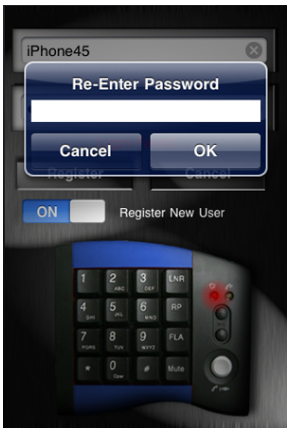


Abb. 5: Re-enter Password

Über das Login Menü kann der Benutzer sich mit seinem bereits angelegtem Account anmelden oder sich auch ggf. einen neuen anlegen. Das Menü besteht aus zwei Textfeldern, in denen der Username und das Passwort eingegeben werden, sowie aus einem Infofeld, einem `Login-/Register-` , einem `Cancel-` und einem `Toggle-`Button. Das Tastenfeld in der unteren Hälfte besitzt keine Funktionalität und dient nur der Optik.

Bei bereits vorhandenem Account kann der Benutzer also seine Daten eingeben und auf `Login` drücken. Stimmen Username und Passwort überein, sollte die rote Lampe auf dem Tastenfeld nun grün leuchten und das Infofeld "Login Successfull" anzeigen. Tritt beim Einloggen ein Fehler auf, so wird eine Meldung geworfen und das Infofeld zeigt "Login Failed!" an .

Möchte man einen neuen Account anlegen, so muss der `Toggle-`Button auf *On* geschoben werden. Der `Login-` wird nun zu einem `Register-`Button und bei Eingabe der Daten wird man zur Sicherheit aufgefordert, sein Passwort zu wiederholen. Usernamen dürfen nicht mehrfach vergeben werden, daher kann es sein, dass auch hier eine Meldung erscheint und den Benutzer auffordert einen neuen Namen einzugeben.

2.3 Userinfo

Phillipp Bertram

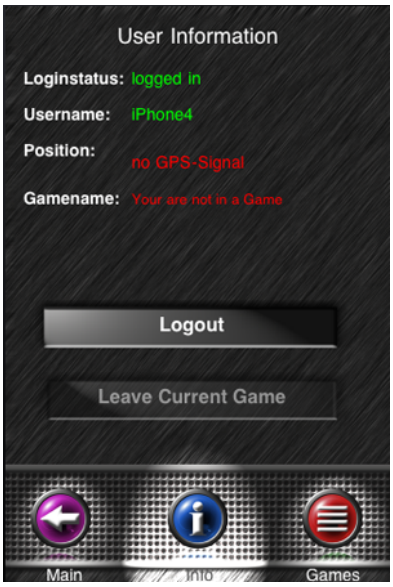


Abb. 6: Kein GPS-Signal, kein Spiel

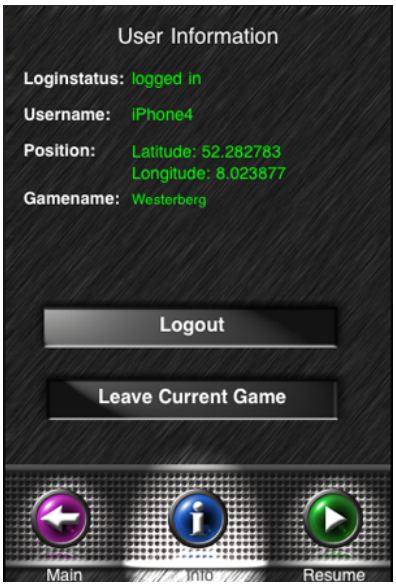


Abb. 7: GPS-Signal, bestehendes Spiel

Die *Userinfo* ist die erste Ansicht, zu der man nach erfolgreichem Login gelangt. Hier stehen einige generelle Informationen wie Loginstatus, Username, Position und Spielname. Ist man bereits in einem Spiel eingeloggt und das GPS Gerät ist eingeschaltet sollte in etwa das in der rechten Abbildung gezeigte Bild erscheinen, andernfalls in etwa die linke Abbildung.

Hier besteht außerdem die Möglichleit sich komplett auszuloggen (`Logout`) oder ggf. nur das aktuelle Spiel zu verlassen (`Leave Current Game`). Ist man ausgeloggt, ändert sich der `Logout`-Button in `Login`, sodass der Benutzer sich auch hier wieder einloggen kann.

Im unteren Bereich der Ansicht befinden sich drei weitere Buttons. Mit dem linken gelangt man wieder zurück zur Startseite. Mit den anderen beiden Buttons wechselt man zwischen dem *Gameselector* und der *Userinfo*. Welche Ansicht gerade aktiv ist, erkennt man in erster Linie durch die Überschrift, aber auch an der Beleuchtung des jeweiligen Buttons. Der Button der aktiven Ansicht ist durch einen Spot hervorgehoben. Zu beachten ist, dass sich der rechte untere Button ändert, je nachdem ob sich der Benutzer in einem Spiel befindet oder nicht.

2.4 Gameselector & Gamecreator

Phillipp Bertram

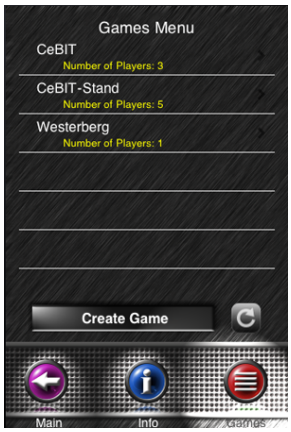


Abb. 8: Liste aller Spiele

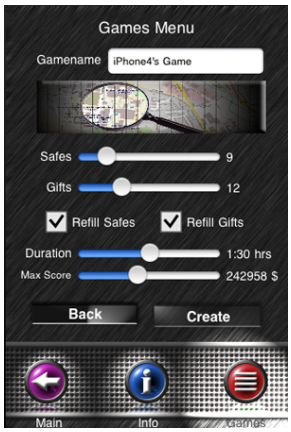


Abb. 9: Spiel erstellen

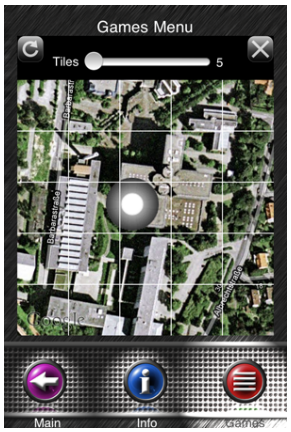


Abb. 10: Spielfeld definieren

Abb. 11: Detailansicht

Über den *Gameselector* kann man, wie der Name bereits verrät, entweder einem bestehenden Spiel beitreten oder sich ein neues anlegen. Alle bestehenden Spiele werden in der Tabelle aufgeführt und der Benutzer kann durch Berühren eines Eintrags in die Detailansicht wechseln.

Die Detailansicht gibt etwas mehr Informationen zu dem jeweiligen Spiel wieder. Hier wird der Name des Spiels, die Spieleranzahl und die Spielernamen mit Team angezeigt. Über die drei Buttons `Cop`, `Random`, `Thief` tritt man diesem Spiel entweder als *Polizist*, *zufällig* oder als *Dieb* bei.

Möchte man ein eigenes Spiel erstellen, geschieht das über den *Gamecreator*. Per `Create Game` öffnet sich eine neue Ansicht, in der alle Einstellungen vorgenommen werden können. Standardmäßig heißt das Spiel wie der Benutzername, kann aber beliebig geändert werden. Der Button mit der Lupe öffnet eine weitere Ansicht, auf der das Spielfeld definiert werden kann. Der Kartenausschnitt gibt hierbei genau die Grenzen des Spiels an. Durch raus- bzw. reinzoomen (zwei Finger mittig platzieren und voneinander entfernen bzw. auch umgekehrt) kann das Spielfeld vergrößert oder verkleinert werden. Die Anzahl der "Kacheln", also die Bereiche, die später durch den "Nebel" verdeckt sind und aufgedeckt werden müssen, kann im oberen Bereich durch einen *Slider* variiert werden. Beim Definieren des Spielfeldes kann es vorkommen, dass für den ausgewählten Bereich keine Daten vorhanden sind. Dies wird aber ggf. durch eine Meldung angemerkt.

Des Weiteren lassen sich die Anzahl der Tresore und Geschenkboxen über einen Slider einstellen. Im Normalfall verschwinden geknackte bzw. gesicherte Tresore und eingesammelte Geschenke. Über die Häkchen bei `Refill Safes` und `Refill Gifts` wird anschließend wieder ein neuer Tresor zufällig in das Spiel eingefügt. Die letzten beiden Slider legen mehr oder weniger das Ziel des Spiels fest. Wird eine Spielzeitdauer angegeben, endet das Spiel nach eben genau dieser Zeit und das Team mit dem höchsten Score gewinnt. Wird ein `Maximal Score` angegeben, hat das Team gewonnen, das diesen überschritten hat. Zieht man die Slider ganz nach links, gibt es keine Grenzen.

Wenn alle Einstellungen vorgenommen worden sind, kann über `Create` das Spiel erstellt und beigetreten werden. Welchem Team man angehören möchte, kann direkt in einem *Popup* entschieden werden.

2.5 Game

Phillipp Bertram

Je nachdem für welches Team man sich entschieden hat oder welches Team zufällig ausgewählt wurde, sieht auch die *GUI* anders aus. Während die Hintergründe und die Statusbar bei Polizisten blau gefärbt sind, erhalten die Diebe eine rote Farbe. Im Prinzip verfolgen beide Teams das selbe Ziel: Sie müssen Tresore finden und deren Score über ein kleines Zahlenpuzzle erlangen. Der Unterschied besteht lediglich darin, dass die Diebe den Tresor knacken und das darin befindliche Gold stehlen. Die Polizisten sichern diesen, sodass kein Dieb mehr in der Lage ist, sich an diesem Tresor zu vergreifen. Auch hier erhält der Polizist den selben Score, den auch der Dieb beim Ausrauben erhalten hätte. Über diverse Bonusgegenstände können sich die Teams

gegenseitig beeinflussen. Es gilt stets immer als Erster an einem Tresor zu sein und diesen über die *Augmented Reality* Perspektive zu knacken (als Dieb) bzw. zu sichern (als Polizist).

2.5.1 Karte

Phillipp Bertram

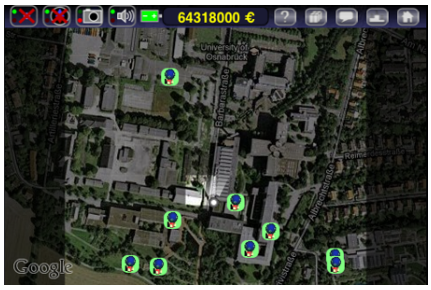


Abb. 12: Ausgangsansicht

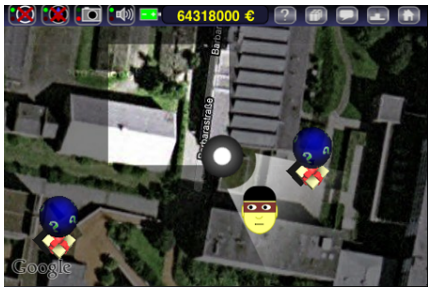


Abb. 13: Vergrößert

Abb. 14: Kacheln aufgedeckt

Ausgangspunkt der Anwendung ist eine Karte, auf der das Spielfeld, hevorgehoben durch einen etwas hell ausgegrauten Bereich, visualisiert ist. Der Benutzer kann an dieser Stelle nur bestimmte Bonus-Gegenstände (*Items*) - die *Geschenkboxen* - sehen. Diese enthalten einen zufällig vom Server generierten Bonusgegenstand, der die Suche vereinfachen oder durch Interaktion mit seinen Teammitgliedern dem gegnerischen Team die Suche erschweren kann. Eine Auflistung aller Bonus-Gegenstände befindet sich in Abschnitt *"Inventory"* und eine Auflistung aller Symbole auf der Karte ist in diesem Abschnitt weiter unten aufgeführt.

Um an die bisher durch den "Nebel" verdeckten Tresore zu gelangen, kann sich der Spieler zunächst zu dem sichtbaren Geschenkboxen begeben. Auf seinem Weg werden bisher verdeckte *Kacheln* aufgedeckt, sodass unter Umständen ein Tresor oder auch gegnerische Spieler, die sich innerhalb dieser Kachel befinden, angezeigt werden. Zoomen ist ebenfalls möglich und funktioniert genauso wie bei *GoogleMaps*. Befindet man sich in der Nähe einer Geschenkbox oder eines Tresores, muss nun in die *Augmented Reality*-Perspektive gewechselt werden, indem das Smartphone wie eine Kamera nach oben gehalten wird.

Am oberen Rand befindet sich eine Statusbar. Diese enthält diverse Buttons und in der Mitte ein Feld, auf dem der aktuelle Highscore des Spielers angezeigt wird. Links daneben sind verschiedene Anzeigen, wie Batteriestatus, Sound, Camera, Kompass und GPS, angeordnet. Hier besteht die Möglichkeit, deren Funktionen ein- und auszuschalten. Rechts daneben befinden sich fünf kleine Buttons, mit denen man zur Hilfe, in das Inventar, zum Chat, zum Highscore (siehe Abschnitt *Inventar, Chat. Highscore*) oder zurück zur Userinfo gelangt.

Durch Berühren eines Tresors auf dieser Karte wird ein kleines Feature aktiviert. Dieses zeigt die Entfernung zu dem Tresor bezüglich seiner eigenen Position an. Allerdings wird hier die Entfernung der Luftlinie gemessen. Der wahre Weg dorthin kann unter Umständen deutlich länger sein.

Legende

Deutet die Position eines Polizisten an. Polizisten sind nur innerhalb desselben Teams oder über spezielle Bonus-Gegenstände sichtbar.

Deutet die Position eines Diebes an. Diebe sind nur innerhalb desselben Teams oder über spezielle Bonus-Gegenstände sichtbar.

Zeigt die Position eines Tresors an. Tresore sind nur innerhalb aufgedeckter Bereiche oder über spezielle Bonus-Gegenstände sichtbar. Um die Punkte eines Tresors zu bekommen, muss ein Zahlenpuzzle in vorgegebener Zeit gelöst werden. Über entsprechende Bonus-Gegenstände kann das Puzzle auch übersprungen werden. Läuft die Zeit ab, bevor das Puzzle gelöst worden ist, verliert man die Punkte und der Tresor zerstört sich selbst.

Zeigt die Position einer Geschenkbox an. Geschenkboxen sind im Gegensatz zu den Tresoren oder Spielern immer sichtbar. Sie enthalten nützliche Gegenstände, die den eigenen Spielfluss erleichtern oder den des gegnerischen Teams erschweren. Anders als bei den Tresoren muss hier kein Zahlenpuzzle gelöst, sondern kann dieser direkt eingesammelt werden.

Dieses Symbol zeigt die eigene Position an.

2.5.2 Augmented Reality Perspektive

Phillipp Bertram

Abb. 15: AR mit Geschenkbox und Tresor

Abb. 16: Geschenkbox geöffnet

Abb. 17: Tresor Popup

Ein besonderes Feature stellt die *Augmented-Reality*-Perspektive dar. Die auf der Karte aufgeführten Tresore und Geschenkboxen werden in dieser Ansicht dreidimensional in das Kamerabild projiziert. Sind keine Objekte zu sehen, muss der Spieler ggf. die Kamera etwas herumschwenken, bis die in der Nähe befindlichen Items im Kamerabild erscheinen. Die im oberen Bereich abgebildete Kompassnadel zeigt hierbei die Blickrichtung des Spielers an.

Der Spieler ist in dieser Ansicht in der Lage mit den angezeigten Objekten zu interagieren. Um z.B. eine Geschenkbox einzusammeln, muss der Spieler diesen lediglich anklicken. Bei den Tresoren geschieht das analog, mit dem Unterschied, dass zunächst eine Bestätigung erfolgen und anschließend ein Zahlenpuzzle gelöst werden muss.

2.5.3 Tresor knacken / sichern

Phillipp Bertram

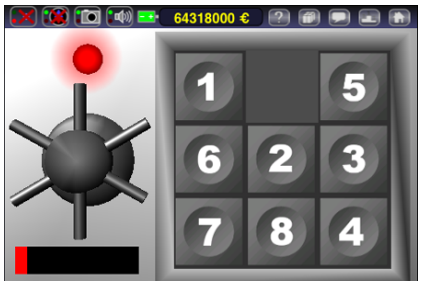


Abb. 18: 9-Puzzle

Abb. 19: Tresor gesichert

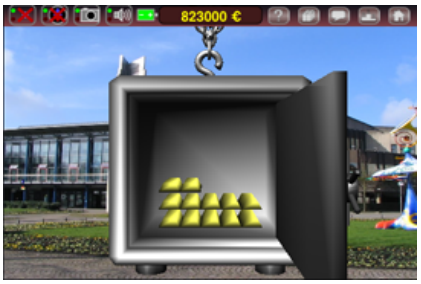


Abb. 20: Tresor geknackt

Ob ein Tresor als Dieb geknackt oder als Polizist gesichert werden soll, stellt vom Spielablauf keinen Unterschied dar. Beide Parteien müssen ein Zahlenpuzzle lösen. Hierbei handelt sich um ein gewöhnliches 8-Puzzle. Das Spiel besteht aus 8 in einem drei-mal-drei-Quadrat angeortneten Zahlen, die durch Verschiebungen aufsteigend in vorgegebener Zeit geordnet werden müssen. Schafft es ein Polizist innerhalb der vorgegeben Zeit das Puzzle zu lösen, ist der Tresor gesichert und er erhält die Punkte. Die Diebe müssen bei erfolgreichem Knacken auf das Rad drücken, damit sich der Tresor öffnet und der Inhalt entnommen werden kann.

Anschließend wechselt der Spieler wieder in die Kartenansicht und sucht sich den nächsten Tresor oder die nächste Geschenkbox.

2.5.4 Inventar, Chat, Highscore

Phillipp Bertram

Abb. 21: Inventar

Abb. 22: Chat

Abb. 23: Highscore

Das *Inventar* stellt ein zentrales Feature des Spiels dar. Nicht nur die persönlich eingesammelten Bonusgegenstände, sondern auch die des Teams werden hier angezeigt. Ist der Spieler in Besitz eines Items, das einer seiner Teammitglieder besser gebrauchen könnte, kann er diesen einfach von seinem persönlichen Inventar in das *Team Inventar* verschieben. Der Mitspieler ist so in der Lage diesen Gegenstand in sein eigenes Inventar zu überführen und ihn bei Gelegenheit auch zu aktivieren. Eine Auflistung aller Gegenstände ist weiter unten in diesem Abschnitt aufgeführt.

Über den *Chat* können die Teams untereinander, aber auch global miteinander kommunizieren und sich ggf. strategisch absprechen oder einfach nur austauschen. Im linken Bereich befindet sich eine Tabelle mit den Spielernamen, die sich in dem Spiel befinden. Wechselt man über die beiden Buttons im unteren linken Bereich zwischen `globalem` und `team`-Chat ändert sich auch entsprechend die Liste der angezeigten Spieler. Um diesen Chat wieder zu verlassen, muss man erneut auf das Chat-Symbol clicken.

Die *Highscore* ist lediglich eine Auflistung aller Spieler mit deren Score sowie der Gesamtpunktzahl der Teams und gibt einen Überblick über den Stand des Spiels.

Items

Alle Spieler

Alle Spieler werden für kurze Zeit auf der Karte angezeigt. Dieses Item ist nur in der Kartenansicht einsetzbar.

Alle Tresore

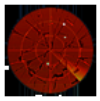
Alle Tresore werden für kurze Zeit auf der Karte angezeigt. Dieses Item ist nur in der Kartenansicht einsetzbar.

Blaupause

Mit der Blaupause kann das Zahlenpuzzle gelöst werden. Dieses Item ist nur in der Spieleansicht einsetzbar.

Taschenlampe

Die Taschenlampe deckt alle im Umkreis von 50 Metern umliegenden Kacheln auf. Dieses Item ist nur in der Kartenansicht einsetzbar.



Störsender

Mit dem Störsender wird einem zufälligen Spieler des gegnerischen Teams die Kartenansicht für zwei Minuten gestört.

Stehlen

Mit diesem Item wird ein zufälliges Item eines zufälligen gegnerischen Spielers gestohlen.

2.6 Options, Help, About

Phillipp Bertram

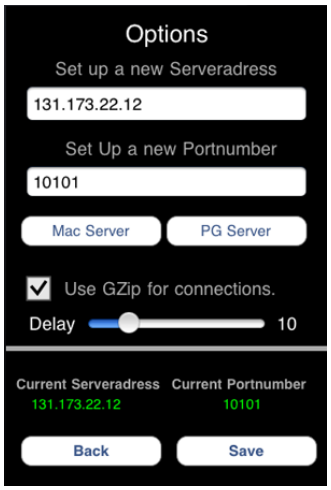


Abb. 24: Options

Über das *Options*-Menü kann der Benutzer die Adresse und den Port des Servers dynamisch ändern. Die Adressen der beiden gängigsten Server werden durch Klicken des jeweiligen Buttons `Mac Server` bzw. `PG Server` automatisch in die Textfelder übernommen. Außerdem kann die Rate der Anfragen zum Server über den Slider `Delay` variiert und die Kompression der Anfragen aktiviert bzw. deaktiviert werden.

Help öffnet ebenso wie *About* eine Internetseite, auf der eine kleine Spielanleitung bzw. Informationen zu diesem Projekt und der Projektgruppe zu finden sind.

2.7 Gameviewer

Andre Schemschat

Neben den beiden Smartphone-Clients gibt es noch eine dritte Möglichkeit ein Spiel zu verfolgen, den Gameviewer. Während die Handy-Applikationen dazu ausgelegt sind die Teilnahme an einem Spiel zu ermöglichen, ist der Gameviewer in erster Linie darauf ausgelegt von unbeteiligten Beobachtern benutzt zu werden. Neben dem reinen Beobachten bietet er jedoch auch Funktionen um Spielelemente hinzuzufügen oder zu entfernen, so dass er auch ein geeignetes Instrument für einen etwaigen Spielleiter ist. Der folgende Abschnitt gibt eine Übersicht über den Gameviewer und erläutert die einzelnen Ansichten und ihre Funktionsweise.

Realtime-Trackung und History-Tracking

Der Gameviewer unterteilt sich in zwei verschiedene Modi, zum einen das Realtime-Tracking und zum anderen das History-Tracking. Beide Modi haben dieselbe Basis, unterscheiden sich aber in wenigen Details. Das Realtime-Tracking zeigt Spiele an, die aktuell gespielt werden. Für laufende Spiele bietet die Oberfläche verschiedene Möglichkeiten in den Spielverlauf einzugreifen, wie später gezeigt wird. Im History-Tracking ist dies nicht möglich, dafür kann der Beobachter nach Belieben mit Hilfe der Zeitleiste durch den Spielverlauf springen.

Spielauswahl

Die Spielauswahl ist der Startbildschirm für den Gameviewer. Hier werden alle Spiele aufgelistet (2) und der Besucher kann sich kurze Informationen einblenden lassen, wie in (3) zu sehen. In der Titelleiste (1) wird angezeigt in welchem Modus man sich gerade befindet. Mit einem einfachen Klick auf den Play-Button startet die Ansicht zu einem Spiel.

Übersicht

Die Hauptansicht des Gameviewers ist, wie in dem obigen Bild zu sehen, in vier Bereiche unterteilt, die der Steuerung und der Anzeige dienen. Folgend wird eine Erklärung zu jedem der Bereiche geliefert.

Titelleiste (1)

Die Titelleiste zeigt in welchem Modus man sich momentan befindet. Daneben findet man einen Button um zurück zur Übersicht zu kommen. Im Realtime-Tracking ist ganz rechts das Icon eines Tresors zu finden. Dieses Icon wechselt sich bei mit einem Geschenk ab und bestimmt, was bei einem Klick auf die Karte hinzugefügt wird (Dazu im Abschnitt Karten mehr).

Karten (2)

Die Karte ist die Hauptansicht des Viewers und liefert dem Beobachter den aktuellen Stand des Spiels. Die Umrandungen definieren dabei das Spielfeld, auf dem die Spieler sich bewegen und in dem die Ziele sich befinden. Die rote bezeichnet hierbei das komplette Spielfeld, während die grauen Kästchen einzelne Spielkacheln symbolisieren. Über diesen Begrenzungen werden sowohl die Tresore als auch die Geschenke mit kleinen Icons angezeigt. Für die Spieler werden farbige Icons generiert, die jeden Teilnehmer eindeutig auf der Karte identifizieren (Siehe den Abschnitt *Informationsfenster*). Im Gegensatz zu den Ansichten der Clients ist hier alles sichtbar, der Besucher hat also den kompletten Überblick. Mit den Funktionen des Gameviewers lässt sich auch die Sicht eines Spielteilnehmers nachahmen, dazu später mehr. In dem speziellen Fall, dass einer der Spieler ein Agent ist, so wird noch seine aktuelle Route in Form einer farbigen Linie angezeigt. Befindet man sich in dem Realtime-Tracking, dann ist es möglich mit Hilfe eines Rechtsklicks auf der Karte einen neuen Tresor oder Geschenk an dieser Position zu platzieren. Was dabei gesetzt wird, lässt sich über den Schalter oben rechts in der Titelleiste einstellen. Besucher können so Einfluss auf die Ziele der Spieler nehmen.

Zeitleiste (3)

Die Zeitleiste steuert das Abspielverhalten des Gameviewers. Das Layout orientiert sich stark an einem gängigen Videoplayer und die Funktionalität ist analog. Mit dem Play-Button lässt sich die Wiedergabe starten oder anhalten. Mit den Vorwärts- und Rückwärtsbuttons springt man vor oder zurück in dem angezeigten Zeitpunkt und mit der Zeitleiste kann direkt ein Zeitpunkt ausgewählt werden. Mit dem Regler der Sprungweite lässt sich regulieren, wieviele Sekunden bei jedem Update (Ein Update wird alle fünf Sekunden durchgeführt) gesprungen werden sollen. Um das Spiel 1:1 zu verfolgen ist der Wert fünf voreingestellt. Die gesamten Sprungfunktionen

stehen jedoch nur im History-Tracking zur Verfügung. Im Realtime-Tracking sind sie deaktiviert, da hier jeweils der neueste Spielstand angezeigt wird.

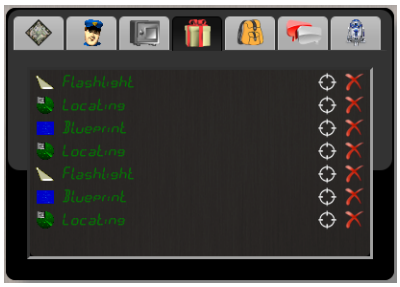
Informationsfenster (4)

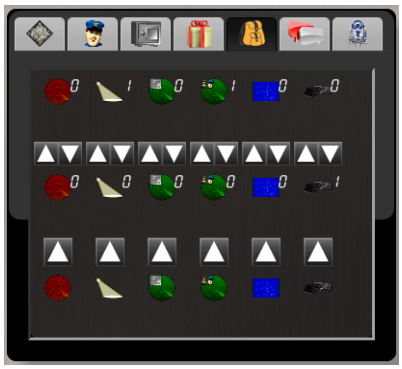
Das Informationsfenster ist der zweite Hauptbereich. In ihm finden sich in verschiedenen Reitern detailierte Informationen zu den einzelnen Bereichen eines Spiels und weitere Möglichkeiten, Einfluss zu nehmen.

Der erste, und standardmässig geöffnete Reiter enthält die allgemeinen Spielinformationen. Neben dem Namen des Spiels, dem Startzeitpunkt und der Spieldauer finden sich hier auch Informationen zu der Anzahl der Tresore, der Spielfläche oder den Fülloptionen.

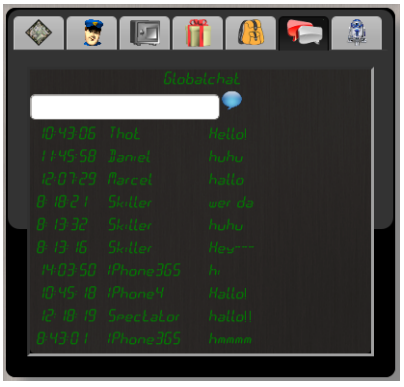
An zweiter Stelle erhält der Besucher informationen über die Teams und aktive Spieler. Die Spieler sind in einer Liste angeordnet und ihrem jeweiligen Team zugeordnet. Hinter jedem Eintrag steht der aktuelle Spielstand, sowohl für individuelle Spieler als auch der gesamte Score den ein Team bis jetzt einbringen konnte. Für jeden Spieler gibt es des Weiteren verschiedene Icons, die einem Besucher erlauben bestimmte Informationen abzurufen oder Aktionen auszuführen. Jeder Spieler hat vor seinem Namen ein eindeutiges, farbiges Icon, über das er auf der Karte identifiziert werden kann und seine Teamzugehörigkeit erkenntlich ist. Das Symbol kann benutzt werden um die Karte auf einen Spieler zu zentrieren. Das Fadenkreuz färbt sich rot und sobald sich dieser Spieler bewegt wird der Kartenausschnitt mit verschoben. Der -Button kann genutzt werden um den Gameviewer in die Sicht eines Spielers zu versetzen. Sobald dieser Modus für einen Spieler aktiviert wurde, färbt sich das Auge rot und die Karte beginnt sich zu verdecken. Abschnitte, Tresore, Geschenke und andere Spieler die für das Ziel nicht sichtbar sind, sind nun auch im Gameviewer verdeckt. Zudem wird das Inventar aktiviert, doch dazu unter *Inventar* mehr. Der letzte Button, , kann genutzt werden um einen Spieler ohne sein Zutun aus einem Spiel zu entfernen. Dieser Button ist nur im Realtime-Tracking vorhanden.

Um eine Liste aller momentan im Spiel befindlichen Tresore oder Geschenke zu erlangen, können diese Reiter genutzt werden. Neben dem Wert des Tresors bzw. dem Typ des Geschenks gibt es die Möglichkeit zu der Position auf der Karte zu springen oder den Gegenstand aus dem Spiel zu löschen. Auch hier ist die Funktion nur im Realtime-Tracking aktiviert.





Jeder Spieler kriegt ein Inventar, in dem er nützliche Gegenstände sammeln kann. Um auch diesen Aspekt des Spiels in dem Gameviewer abzubilden, gibt es diesen Reiter. Wird er ohne einen Spieler im Fokus aufgerufen (Siehe Abschnitt zu den Teaminformationen), so ist lediglich ein rotes Kreuz zu sehen. Wird jedoch ein Spieler fokussiert, dann wechselt diese Ansicht zu der links abgebildeten. In der ersten Reihe sind die Gegenstände zu sehen, die der Spieler in seinem privaten Inventar hat, die Zeile dadrunter sympolisiert das Teaminventar. Die unterste Reihe stellt eine Art Shop dar, aus dem der Spielleiter den Spielern Gegenstände zuweisen kann, indem er die Pfeile benutzt. Mit der Ansicht können außerdem einzelne Gegenstände mit Hilfe der ersten Pfeilreihe innerhalb der beiden Inventar-Typen verschoben werden. Die beiden interaktiven Möglichkeiten sind jeweils wieder nur in dem Realtime-Tracking vorhanden.



Um nicht nur als reiner Beobachter zu agieren wurde in den Gameviewer auch ein Weg integriert, um den spielinternen Chat nutzen zu können. Dieser agiert wie ein normaler Chat mit der Ausnahme, dass die Nachrichten immer von einem User "Spectator" gesendet werden. Ist ein bestimmter Spieler im Fokus, dann werden hier die Chatnachrichten innerhalb des Teams angezeigt, in diesem kann jedoch nicht geschrieben werden. Im History-Tracking ist dieser Reiter komplett entfernt.

Der letzte Reiter, der wiederrum nur im Realtime-Tracking sichtbar ist, kann genutzt werden um dem Spiel einen Agenten hinzuzufügen. Ein Klick auf einen der beiden Team-Symbole startet einen Computerspieler. Sollten schon Agenten in dem Spiel vorhanden sein, so werden sie über den Buttons in einer Liste angezeigt. Mit Hilfe des ✕ können die Agenten wieder gestoppt werden.

3. Server

Andre Schemschat

Um einen geregelten Spielablauf zu ermöglichen, müssen alle Smartphone-Klienten miteinander kommunizieren. Positionen müssen ausgetauscht, die Positionen der Tresore und Geschenke verwaltet und allgemeine Spielinformationen verarbeitet werden. Sobald die Spielfläche den Radius einer Bluetooth-Verbindung überschreitet und eine direkte Kommunikation unter den Smartphones nicht mehr möglich ist, wird eine zentrale Anlaufstelle gebraucht, die diesen Datenaustausch stellvertretend regelt. Zudem bietet eine gemeinsame Anlaufstelle den Vorteil, dass nur ein Datenbestand exisitert, der gepflegt werden muss. Eine dezentrale Datenhaltung erfordert zum einen, dass jeder Klient ständig Updates von allen anderen Teilnehmern erhält und zum anderen muss die Spiellogik redundant auf beiden Platformen implementiert werden. Aus diesem Grund wurde in diesem Projekt auf eine zentrale Verwaltungsstelle gesetzt, den Server.

Im Folgenden soll die Zusammensetzung des Servers aus den einzelnen Komponenten und ihr Zusammenspiel näher erläutert werden. Das folgende Bild zeigt einen schematischen Aufbau der Module und ihrer Wechselwirkungen.

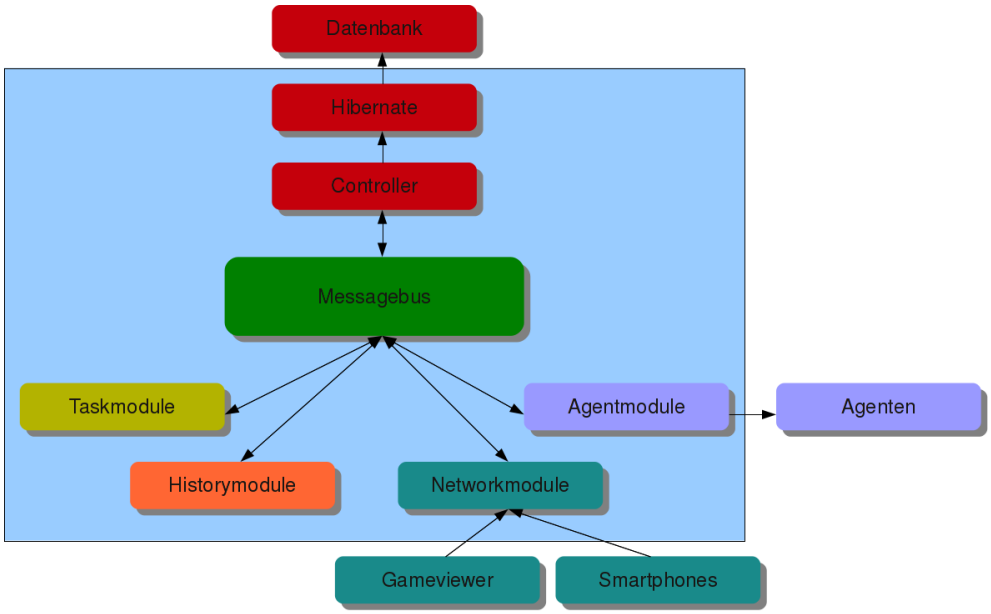


Abb. 25: Schematischer Aufbau des Servers

3.1 OpenStreetMap

Daniel Künne

Karten und Geodaten spielen eine wichtige Rolle in vielen populären Anwendungen und Webangeboten. Dabei werden die zugrundeliegenden Karten häufig um individuelle Daten, wie Anfahrtsbeschreibung oder Points of Interest, erweitert. Üblicherweise wird dabei die Basiskarte nicht selbst erstellt, sondern auf vorhandenes Material von Drittanbietern, wie *GoogleMaps*, zurückgegriffen. Diese Dienste stellen hierfür eine API zu Verfügung, die es dem Anwender ermöglicht punktgenau Marker oder andere Informationen auf der Karte hinzuzufügen.

Die Möglichkeiten, die sich bei der Verwendung eines solchen Dienstes bieten, sind jedoch beschränkt. So lässt sich weder die Farbgebung an den Webseitenstil anpassen, noch kann eine Auswahl getroffen werden welche Elemente überhaupt relevant sind und angezeigt werden sollen.

Einen anderen Ansatz verfolgt *OpenStreetMap*[2] *(OSM)* : 2004 wurde *OpenStreetMap* von Steve Coast in Großbritannien mit dem Ziel der Erschaffung einer freien Weltkarte aus usergenerierten Inhalten gegründet,

2 http://www.openstreetmap.org/

wobei "frei" nicht nur "kostenlos" meint. Vielmehr ist darunter "ohne urheberrechtliche Zugangsbeschränkung" und "ohne Relevanz-Beschränkung" zu verstehen. Dies wird deutlich, wenn man Kartenausschnitte von *OpenStreetMap* mit *GoogleMaps* vergleicht:

Abb. 26: Altstadt Osnabrück in OSM

Abb. 27: Altstadt Osnabrück in GoogleMaps

Wie die obigen Grafiken zeigen, kann der Unterschied lokal sehr groß sein. Auch wird deutlich, dass *OpenStreetMap* nicht nur Straßendaten sammelt, sondern auch Informationen zu Gebäuden und Umgebung bereitstellt. Da die Daten von Freiwilligen gesammelt werden, variiert der Detailgrad der Karten von Region zu Region. Während zum Beispiel Osnabrück zugute kommt, dass es ein regionales Projekt gibt, welches sich mit Erfassung der Geodaten innerhalb der Stadt befasst, sind in ländlichen Regionen die kommerziellen Kartensysteme überwiegend wesentlich besser.

Zugriff auf Kartendaten

Um auf die Daten von *OpenStreetMap* zuzugreifen und diese herunterzuladen, besteht neben einer API die Möglichkeit einen kompletten oder regional beschränkten Datenbankabzug des Datenbestandes zu erhalten. Die gesamten Daten, Planet-File[3] genannt, erhält man direkt bei *OpenStreetMap*. Da diese Datei aber häufig zu umfangreich ist, bietet die Geofabrik GmbH[4] Teile zum separaten Download an. Dieser Datenbestand umfasst dabei alle Länder Europas, sowie die Bundesländer Deutschlands.

Aufbau der Kartendaten

Sämtliche *OpenStreetMap*-Informationen bestehen aus den Objekttypen Nodes, Ways und Relations. Diese finden sich sowohl in den Datenbanktabellen als auch in den XML-Exports wieder.

- **Node:** Ein Node symbolisiert einen Punkt mit seiner geographischen Koordinate, sowie allgemeinen Informationen zu Erfasser und Zeitpunkt des Erfassens. Er kann komplett alleine stehen, beispielsweise bei Parkbänken oder Verkehrsampeln, oder Bestandteil eines Weges sein.
- **Way:** Ein Way ist eine geordnete Folge von Nodes, die auch mehrfach auf einem einzelnen Way liegen können. Flächen werden ebenfalls durch Ways dargestellt. Allerdings sind diese geschlossen, was dadurch ausgedrückt wird, dass Start- und Endpunkt identisch sind. Ways werden bei *OpenStreetMap* genutzt um Straßen, Flächen oder Gebäude darzustellen.

3 http://planet.openstreetmap.org/
4 http://download.geofabrik.de/osm/

- **Relation:** Eine Relation ist ein eine geordnete Folge beliebiger Objekte und dient zur logischen Verknüpfung verschiedener Einheiten. Im folgenden Ausschnitt eines XML-Exports fasst die Relation eine Kleingartensiedlung zusammen. Die einzelnen Member-Knoten stehen hier jeweils für einen Way, welcher wiederum vorher separat definiert ist und nur einen kleinen Bestandteil der gesamten Relation darstellt.

Was ein konkretes Objekt nun darstellt, ergibt sich aus seinen Attributen, den Tags. Diese sind Schlüssel-Wert-Paare aus Unicode-Zeichen wobei jeder Key pro Objekt nur einmal auftreten darf. Im Beispiel sieht man aber auch, dass Attribute weggelassen werden können, wenn sie nicht notwendig sind um ein Objekt zu beschreiben. Sind Nodes beispielsweise nur Bestandteil von Ways, so ist eine Attributierung unnötig.

```
<?xml version='1.0' encoding='UTF-8'?>
<osm version="0.6" generator="pbf2osm">
        <node id="125799" lat="53.0749606" lon="8.7867843" version="5" changeset="5922698"
                user="UScha" uid="45445" timestamp="2010-09-30T19:23:30Z" />
        <node id="125800" lat="53.0719347" lon="8.7840318" version="10" changeset="5923003"
                user="UScha" uid="45445" timestamp="2010-09-30T19:57:15Z" />
        <node id="125801" lat="53.0706761" lon="8.7819464" version="11" changeset="5922698"
                user="UScha" uid="45445" timestamp="2010-09-30T19:23:30Z" />
        [...]
        <way id="38834661" version="1" changeset="2100423" uid="59402" user="Mithi"
                timestamp="2009-08-10T18:29:38Z">
                <nd ref="457577162" />
                <nd ref="461056114" />
                <nd ref="461056117" />
                <tag k="highway" v="residential" />
                <tag k="name" v="Auf den Hohen Enden" />
        </way>
        [...]
        <relation id="33115" version="3" changeset="2022428" uid="55521" user="Ebbe73"
                timestamp="2009-08-03T08:46:05Z">
                <member type="way" ref="27148028" role="inner" />
                <member type="way" ref="15400214" role="inner" />
                <member type="way" ref="23321484" role="outer" />
                <tag k="name" v="Kleingartenanlage" />
                <tag k="type" v="multipolygon" />
        </relation>
        [...]
</osm>
```

Relevante Tags

Da der Server die Platzierung von Tresoren und Geschenkpaketen anhand der vorliegenden OSM-Daten vornimmt und sichergestellt sein muss, dass diese Positionen auch vom Spieler erreicht werden können, muss der Server Kenntniss über alle Tags haben, die eine solche Position beschreiben. In der aktuellen Version des Servers sind das 635 Schlüssel-Wert-Paare, die erreichbar und nicht gefährlich sind. So sind beispielsweise Knoten auf einer vierspurigen Autobahn erreichbar, aber sicherlich nicht als Zielpunkte in einem Spiel für Fussgänger zu wählen.

Bei der Frage welche Tags relevant sind, wurde entschieden Gebäude des öffentlichen Interesses, Naherholungsgebiete und Straßen bis zu einer bestimmten Größe zu berücksichtigen. Die Knoten, die direkt oder indirekt mit solchen Informationen verknüpft sind, werden in einer View herausgefiltert und der Verarbeitung durch den Server zugänglich gemacht.

Import der erreichbaren Knoten

Der Import der erreichbaren Knoten in ein neues Spiel besteht also insgesamt aus mehreren Schritten. Zuerst müssen die *OpenStreetMap*-Daten für das Bundesland, in dem gespielt werden soll, in das Datenbank-Schema `osm_data` importiert werden. Im Hintergrund werden diese Daten dann mit Hilfe einer View so aufbereitet, dass nur noch die relevanten Werte überbleiben. Da es sich bei diesen Werten zum einen noch nicht um geographische Objekte, sondern einfache double-Werte handelt, und zum anderen diese Informationen noch

im falschen Datenbank-Schema liegen, wird eine SQL-Funktion genutzt, die im Hintergrund die endgültige Aufbereitung vornimmt.

```
CREATE OR REPLACE FUNCTION pgsgame.import_osmnodes(
                "min_lat" double precision, "max_lat" double precision,
                "min_lng" double precision, "max_lng" double precision) RETURNS VOID AS $$
        DECLARE
                start_id INTEGER;
                end_id INTEGER;
        BEGIN
                SELECT INTO start_id max(id) FROM pgsgame.nodes;
                IF start_id IS NULL THEN
                        SELECT INTO start_id 0;
                END IF;

                -- Verbindung auf ein anderes DB-Schema
                PERFORM dblink_connect_u('myconn', 'dbname=osm_data');

                -- Ermitteln der relevanten Koordinaten, Erzeugen von Punkt-Objekten
                INSERT INTO pgsgame.nodes(id, version, coordinate)
                        SELECT
                                nextval('nodes_id_seq'), 1,
                                GeometryFromText ('POINT(' || lon || ' ' || lat || ')', 4326)
                        FROM dblink('myconn',
                                'SELECT DISTINCT lon, lat
                                 FROM export_nodes
                                 WHERE lat>' || min_lat || ' AND lat<' || max_lat ||
                                ' AND lon> ' || min_lng || ' AND lon<' || max_lng || ''
                        ) AS t(lon DOUBLE PRECISION, lat DOUBLE PRECISION);
                SELECT INTO end_id max(id) + 1 FROM pgsgame.nodes;

                -- Anpassen der neuen ID's
                INSERT INTO pgsgame.osmnodes(id)
                        SELECT id FROM pgsgame.nodes WHERE id > start_id AND end_id > id;
        END;
$$ language 'plpgsql';
```

Die obige Funktion erwartet vier Parameter mit denen angegeben werden muss, für welchen geographischen Bereich die Aufbereitung erfolgen soll. Anschließend werden anhand der vorliegenden Informationen *Points* generiert und in der Tabelle `nodes` eingefügt. Um eine korrekte Vererbung innerhalb des *ORMs* sicherzustellen, müssen die IDs der so erzeugten Punkte zusätzlich noch in die Tabelle `osmnodes` eingefügt werden. (siehe auch: Entity-Relationship-Modell)

Nach Ablauf dieser Funktion können Spiele in dem aufbereiteten Bereich gespielt werden, da der Server nun weiß, wo er sinnvoll Tresore und Geschenkpakete platzieren kann.

3.2 Datenbank

Daniel Künne

Aufgrund des hohen Datenvolumens erfolgt die Datenhaltung datenbankbasiert. Um keine zusätzlichen Lizenzkosten durch die Anschaffung proprietärer Software zu haben, kommt eine Open Source-Datenbank zum Einsatz. Zur Zeit stehen mit *PostgreSQL* und *MySQL* Opensource-Datenbanken mit nativer Unterstützung von Geometrien zur Verfügung. Beide Datenbanken können sowohl unter Linux als auch unter Windows betrieben werden und sind mit vergleichbarem Aufwand zu installieren. Zudem handelt es sich bei beiden Datenbanken im Kern um relationale Datenbanksysteme, die um einen zusätzlichen Objektdatentyp (`Geometry`) und entsprechende Funktionen erweitert wurden.

MySQL

Bei *MySQL* handelt es sich, wie bereits erwähnt, um eine relationale Datenbank. Sie wird seit 1995 vertrieben und beim Einsatz in nicht kommerziellen Produkten kostenlos. Durch die einfache Struktur und die zahlreichen

verfügbaren Schnittstellen zu fast allen verbreiteten Programmiersprachen ist *MySQL* die am häufigsten eingesetzte Datenbank bei Web-Projekten.

MySQL wird über die Datenbanksprache SQL gesteuert. Allerdings ist der Befehlssatz im Vergleich zum SQL92-Standard etwas eingeschränkt, da zum Beispiel statische Integritätsbedingungen akzeptiert, aber bei der Auswertung nicht berücksichtigt werden.

PostgreSQL

Die objektrelationale Datenbank *PostgreSQL* hat ihren Ursprung im Datenbank-Projekt *Postgres*, das 1986 an der Universität Berkeley (USA) ins Leben gerufen wurde. Die Ergebnisse des Projekts wurden 1988 erstmals vorgestellt und 1989 externen Benutzern zur Verfügung gestellt. Bis 1993 wurde *Postgres* in Berkeley weiterentwickelt, aufgrund des zu hohen Support-Aufwandes jedoch eingestellt und als Open Source Projekt unter dem Namen *Postgres95* weitergeführt. Die Implementierung eines SQL-Interpreters brachte die Namensänderung in *PostgreSQL* mit sich.

PostGIS ist eine Erweiterung von *PostgreSQL* um geographische Objekte. *PostGIS* wurde von Refractions Research Inc.[5] entwickelt und bietet die Möglichkeit *PostgreSQL* als Datenserver eines *Geographischen Informationssystems (GIS)* zu nutzen.

Als Datenbanksprache nutzt *PostgreSQL* den SQL92-Standard und wird mit *PostGIS* um die Sprache *PSQL* erweitert, die räumliche Abfragen ermöglicht und einige Funktionen zur Arbeit mit Geometrie-Objekten mitbringt.

Vergleich und Auswahl

Die Funktionen, die von *PostgreSQL/PostGIS* und *MySQL* angeboten werden, sind in der folgenden Tabelle dargestellt. Zusammenfassend lässt sich für *PostgreSQL/PostGIS* sagen, dass eine volle Unterstützung der ISO 19125-Funktionalität gegeben ist. Außerdem wird eine Auswahl von weiteren Funktionen des *Simple Feature Modells* angeboten. Dazu gehören beispielsweise die Validierung von Geometrien oder die Transformation von Geometrien und Koordinaten. Darüber hinaus stehen auch weitergehende Konvertierungsfunktionen nach SVG und seit *PostGIS* 1.2 auch nach KML zur Verfügung.

Die Funktionalität von *MySQL* ist deutlich eingeschränkter. *MySQ*L implementiert nur eine Untermenge der Funktionen des *Simple Feature Models*. So sind die Testfunktionen `IsSimple` und `IsRing` zwar implementiert allerdings nicht funktionsfähig. Bei den geometrischen Funktionen auf einer Geometrie stehen weder `Buffer` noch `ConvexHull` zur Verfügung. Am Gravierendsten gegen einen Einsatz als Datenbank-Backend bei der Umsetzung des Servers spricht das Fehlen von geometrischen Funktionen für mehrere Geometrien.

Vergleich der Funktionen von PostgreSQL/PostGIS und MySQL

Funktion	PostgreSQL/PostGIS	MySQL
Zugriff auf Basiseigenschaften (`Dimension`, `GeometryType`, Zugriff auf die Geometriebestandteile)	X	X
Test von Geometrieeigenschaften (`IsSimple`, `IsClosed`, `IsRing`)	X	(X)
Geometrische Funktionen auf Geometrien (`Distance`, `Centroid`, `Buffer`)	X	
Schnittfunktionen (`Intersection`, `Contains`)	X	
Validierung von Geometrien	X	
Geometrische Aggregatfunktionen	X	

Zu Beginn der Implementation wurde auf *MySQL* als Datenbank-Managementsystem zurückgegriffen. Allerdings zeigte sich bereits bei ersten Tests, dass *MySQL* mit dem zu erwartenden hohen Volumen an geographischen Daten und den damit verbundenen Anfragen Probleme bekommt. Aus diesem Grund wurde das Datenbank-Backend dahingehend umgestellt, dass nun *PostgreSQL* mit der *PostGIS*-Erweiterung zum Einsatz kommt.

5 http://www.refractions.net

3.2.1 Postgis

Daniel Künne

PostGIS ist eine Erweiterung für das objekt-relationale Datenbank-Managementsystem *PostgreSQL* um geographische Funktionen und Objekte. Somit kann der Datenbank-Server in Verbindung mit einem *Geoinformationssystem (GIS)* genutzt werden. Außerdem folgt *PostGIS* im Gegensatz zu den Implementation von *MySQL* oder *Oracle* der *OpenGIS Simple Feature Specification for SQL*[6].

Installation

Da die Installation der *PostGIS*-Funktionalität sich nur auf ein Datenbank-Schema auswirkt, muss bei jedem neuen Anlegen die komplette Installationsroutine durchlaufen werden.

```
-- Erstellung eines neuen Datenbank-Schemas
createdb -O pgsgame pgs_prod
-- Erweiterung des Befehlssatzes um die Sprache PSQL
createlang plpgsql pgs_prod
-- Import der GIS-Funktionen und -Objekte mit dem aktuellsten Update
psql -d pgs_prod -f lwpostgis.sql
psql -d pgs_prod -f lwpostgis_upgrade.sql
-- Import der Kommentare für obige Funktionen
psql -d pgs_prod -f postgis_comments.sql
-- Import der Referenzinformationen für die System
psql -d pgs_prod -f spatial_ref_sys.sql
```

In obigem Beispielcode erzeugt der Befehl `createdb` ein neues Datenbank-Schema mit dem Bezeichner `pgs_prod` für den Benutzer `pgsgame`. Um nun den vollen Funktionsumfang der *PostGIS*-Erweiterung zu aktivieren, muss zuerst die Sprache *PSQL* als Erweiterung für SQL aktiviert werden. Anschließend können die SQL-Skripte mit den `create`-Statements für die GIS-Funktionalität gegen die Datenbank abgesetzt werden. Das Importieren der Kommentare zu den Funktionen und Objekten ist nicht zwangsläufig erforderlich, liefert aber eine bessere Verständlichkeit und hat keine negativen Auswirkungen auf die Performance des Servers. Der letzte Schritt bei der Installation ist der Import der Referenzinformationen für das System. Dies ist erforderlich, da PostGIS die Geoinformationen nicht in den vom Benutzer angelegten Tabellen direkt, sondern in einer separaten Tabelle für Geodaten speichert.

Speicherung

Die Speicherung der Geometrien erfolgt innerhalb einer *PostgreSQL*-Datenbank, die die *PostGIS*-Erweiterung installiert hat, im *Well-Known Binary (WKB)* Format in einer speziellen Tabelle, welche nur die Geoinformationen enthält. Die Spalten aus dieser Tabelle werden durch *PostGIS* den eigentlichen Daten zugeordnet.

Bei *WKB* handelt es sich um ein binäres Datenformat, welches vom *Open Geospatial Consortium*[7] extra standardisiert wurde, um Geodaten in einer Datenbank zu speichern. Neben der Repräsentation der Daten im *Well-Known Binary* Format besteht auch die Möglichkeit das textbasierte *Well-Known Text (WKT)* Format zu nutzen. Dieses dient lediglich zur Verbesserung der Lesbarkeit eines Datensatzes und nicht zur Speicherung.

```
-- WKT Repräsentation eines Punktes
POINT(8.0000116 52.4779105)
-- WKT Repräsentation einer Spielfläche
POLYGON((8.02003422600057 52.2860547779375, 8.02738577399943 52.2860547779375,
         8.02738577399943 52.2815632220625, 8.02003422600057 52.2815632220625,
         8.02003422600057 52.2860547779375))
```

6 http://portal.opengeospatial.org/files/?artifact_id=829
7 http://www.opengeospatial.org/

Indizierung

Neben den Funktionen zur einfachen Analyse bietet *PostGIS* Funktionen an, mit denen sich ein Zugriff auf die vorhandenen Daten optimieren lässt. Damit bei der Abarbeitung einer Anfrage nicht alle Daten nacheinander eingelesen werden müssen, sondern möglichst nur die Daten, die für eine Anfrage in Betracht kommen, bietet es sich an, einen Index auf den Daten zu erzeugen. Indizes, die den Zugriff auf vorhandene Daten optimieren, gibt es in verschiedenen Systemen, nicht nur im Geodatenbereich. Beispielsweise können bei einer normalen *PostgreSQL* Datenbank Indizes angelegt werden, die das Auffinden von Werten in bestimmten Spalten erheblich beschleunigen. Allerdings entsteht durch das Anlegen eines Indizes immer ein Overhead, der das System zusätzlich belastet und es sollte vom Entwickler genau abgewogen werden, ob dieser Aufwand gerechtfertigt ist.

Als Index-Typen stehen *B-Tree*s, *R-Trees* und *Generalized Search Trees* (GiST) zur Verfügung. Einen Index für Geodaten in *PostGIS* sollte man mit Hilfe eines *GiST* erzeugen.

Da diverse Anfragen auf die Geometrien abgesetzt werden, die die Gesamtfläche des Spieles sowie die einzelnen Teilbereiche beschreiben und die Anzahl der Datensätze mit jedem neu erstellten Spiel erheblich steigt, ist hier eine Indizierung innerhalb der Serverimplementation durchaus sinnvoll.

```
-- GIST-Index auf die Spalte boundingBox in der Tabelle gamefields
CREATE INDEX idxBoundingbox ON pgsgame.gamefields USING gist(boundingBox GIST_GEOMETRY_OPS)
```

SQL-Queries

Aufgrund des Einsatzes von *Hibernate* als *objektrelationaler Mapper* und der Verwendung des Aufsatzes *Hibernate Spatial* müssen Im Java-Code keine SQL-Anfragen für geometrische Objekte formuliert werden. Allerdings werden im Hintergrund sehr wohl SQL-Anfragen an die Datenbank durchgereicht.

```
-- Liefert alle Knoten innerhalb einer gegebenen Fläche
SELECT n.coordinate FROM nodes n, areas a
        WHERE Contains ( a.boundingBox, n.coordinate) = TRUE;
-- Liefert alle Spieler die sich in einem Umkreis von maximal
-- 0,05 Grad um die Spielfläche befinden
SELECT p.* FROM areas a, player p
        WHERE isempty(intersection(a.boundingBox, st_buffer(p.coordinate, 0.05))) = false;
-- Aggregatfunktion zum Zusammenfassen einzelner Areas
SELECT SUM(ST_Area(boundingBox)) AS total_area FROM areas;
```

Der obige Auszug zeigt drei Beispielanfragen, wie sie von *Hibernate* generiert werden. Alle drei benutzen Funktionen der *PostGIS*-Erweiterung und im letzten Fall wird die `ST_Area`-Funktion, die eine Fläche erzeugt, mit den Aggregatfunktionen von SQL verknüpft. Die `SUM`-Funktion ist hier aber überladen und liefert keinen Zahlenwert, sondern erzeugt im Hintergrund eine neue Geometrie.

3.2.2 Hibernate

Andre Schemschat

Hibernate ist ein Framework für Java und .Net, dass die Benutzung einer Datenbank kapselt und abstrahiert. Die unten stehende Abbildung verdeutlicht dessen Aufbau. An unterster Stelle steht hierbei die Datenbank. Der Zugriff auf diese wird dabei durch die drei Standard-APIs *Java Naming and Directory Interface (JDNI)*[8], *Java Transaction API (JTA)*[9] und *Java Database Connectivity (JDBC)*[10] geregelt. Hibernate setzt auf diese APIs auf, so dass der Zugriff von der spezifischen Struktur abstrahiert werden kann. Die meisten

8 http://www.oracle.com/technetwork/java/index-jsp-137536.html
9 http://java.sun.com/javaee/technologies/jta/index.jsp
10 http://java.sun.com/products/jdbc/

Datenbankhersteller bieten ihre eigenen Implementationen für diese Interfaces an, so dass das Framework unabhängig von der benutzten Datenbank ist.

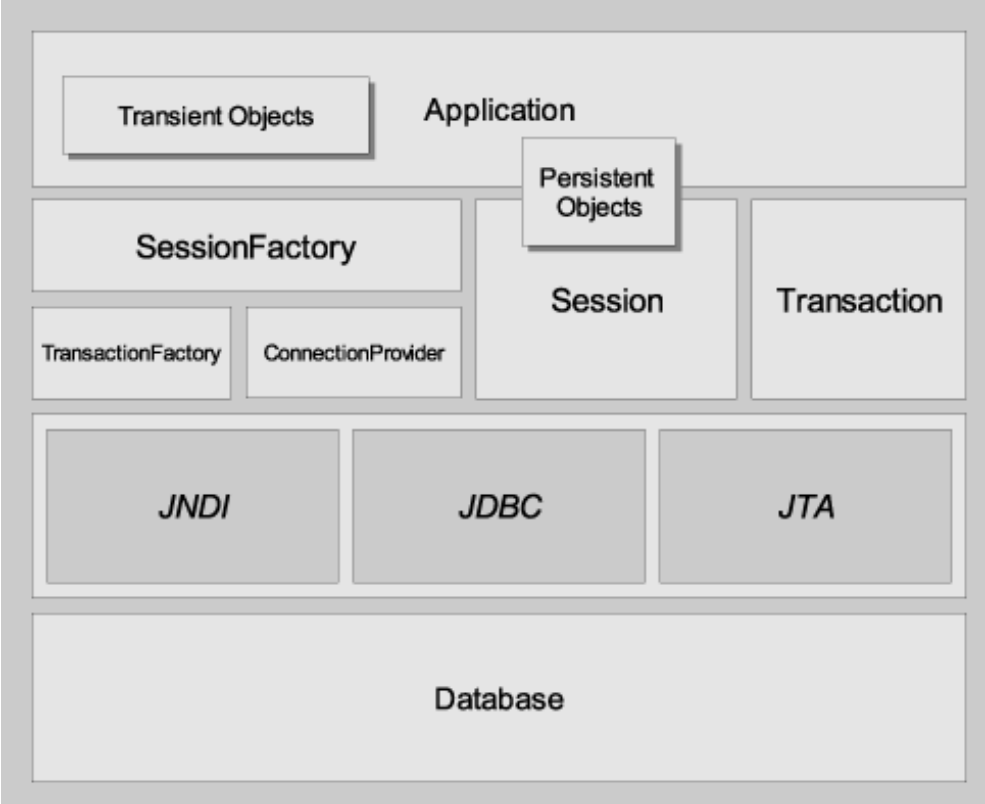


Abb. 28: Übersicht der Hibernate-Architektur[11]

Auf die *JDBC*-Schnittstellen setzt dann ein *ConnectionProvider* auf. Dieser verwaltet die Verbindungen zu der Datenbank über einen ConnectionPool. Aus diesem Pool werden bei neuen Anfragen die offenen Verbindungen entnommen oder neue angelegt, falls keine verfügbar sind. Der Provider ist außerdem dafür verantwortlich nicht mehr benötigte Verbindungen wieder freizugeben. Neben dem *ConnectionProvider* arbeitet die *TransactionFactory*. Gerade in größeren Applikationen, in denen zu einer Anfrage mehrere zusammenhängende Datenbankquerys erforderlich sind, bietet es sich an dem *ACID*-Prinzip[12] zu folgen. Um dies einfach umzusetzen bietet Hibernate mit der Factory eine Möglichkeit, Transaktionen einfach und sicher zu verwalten. Um beide Interfaces in der Applikation komfortabel zu nutzen hat Hibernate eine weitere Abstraktionsebene, die *SessionFactory* eingefügt. Je nach Kontext der Nutzung stellt diese Klasse automatisch eine Transaktion auf einer offenen Verbindung bereit, ohne dass dies explizit angefordert wird. Ein Kontext kann dabei eine gesamte Applikation, aber auch ein einzelner Thread sein. Gerade letzteres ist bei einem *One-Thread-per-Request*-Model sehr nützlich, da pro Thread eine neue, atomare Session geöffnet wird.

Die so geöffnete Session ist das Objekt, mit dem innerhalb der Geschäftslogik gearbeitet werden kann. Mit diesem lassen sich Anfragen auf das Model ausführen. Dieses besteht dabei aus Java-Beans (oder *POJOs*), die über XML-Mappings oder Annotations auf ein relationales Schema überführt werden (Ein detaillierteres Beispiel wird unter Beans vorgestellt). Hibernate agiert hier also auch als *Object-relational Mapper*[13]. Mit Hilfe der Session und den Beans können dann Anfragen an die Datenbank gestellt werden. Dies geschieht zum einen mit der *Hibernate Query Language (HQL)*. Diese Sprache ist ein Dialekt der *Structured Query Language*, die auf die objektorientierte Basis angepasst wurde. Die zweite Möglichkeit ist die `Criteria`-API. Mit dieser lassen sich Schritt für Schritt Ergebnismengen einengen und Queries mit vordefinierten Methoden erstellen. Für beides ist im Folgenden ein kurzes Beispiel gegeben.

```
Session session = HibernateUtil.getSessionFactory().getCurrentSession();
session.beginTransaction();
```

11 http://docs.jboss.org/hibernate/core/3.6/reference/en-US/html/architecture.html#architecture-overview-comprehensive
12 http://de.wikipedia.org/wiki/ACID
13 http://de.wikipedia.org/wiki/Objektrelationale_Abbildung

```
Person aPerson = (Person) session
    .createQuery("select p from Person p left join fetch p.events where p.id = :id")
    .setParameter("id", 5)
    .uniqueResult();

...
session.getTransaction().commit();


Session session = HibernateUtil.getSessionFactory().getCurrentSession();
session.beginTransaction();

List<Person> list = (List<Person>)session
     .createCriteria(Person.class)
     .addRestriction(Restrictions.eq("name", "Mustermann"))
     .list();

...
session.getTransaction().commit();
```

Die so geladenen Daten sind als persistente Objekte in der Session verankert. Solange ein Objekt an eine Session *attached* ist, werden Änderungen automatisch in die Datenbank übernommen, sobald die Transaktion beendet wird. Gleichzeitig ist Hibernate auch in der Lage durch optimistisches oder pessimistisches Locking Dateninkonsistenzen zu vermeiden oder Referenzen dynamisch bei Bedarf nachzuladen.

Neben diesen Kernfeatures bietet das Hibernateframework eine sehr flexible Architektur. So lassen sich weitere Caching-Mechanismen einbinden oder der Connectionpool auswechseln, um nur zwei Beispiele zu nennen.

3.2.2.1 Hibernate Spatial

Daniel Künne

Hibernate Spatial bietet eine generische Erweiterung für das Datenbank-Framework *Hibernate* für die Arbeit mit geographischen Daten. Es unterstützt dabei die meisten Funktionen der *OpenGIS Simple Feature Specification for SQL* und kann mit allen großen Datenbank-Managementsystemen verwendet werden.

Um *Hibernate Spatial* verwenden zu können, muss die entsprechende Bibliothek in das Projekt eingebunden werden. Zusätzlich wird auch noch eine Bibliothek mit den *PostGIS*-Klassen benötigt, damit *Hibernate* ein korrektes Mapping vornehmen kann. Ist dies geschehen, stehen zusätzlich zu den normalen `Restrictions` noch die `SpatialRestrictions` für Datenbankanfragen zur Verfügung.

```
Polygon boundingBox = calculateBoundingbox(msg.getCenter(), msg.getWidth(), msg.getHeight());
...
List<OSMNode> list = (List<OSMNode>)s
     .createCriteria(OSMNode.class)
     .add(SpatialRestrictions.within("coordinate", boundingBox))
     .list();
```

In obigen Code-Beispiel werden alle Einträge aus der Tabelle `osmnodes` ermittelt, die innerhalb eines Polygons liegen. Die Klasse `Polygon` ist dabei Bestandteil der bereits erwähnten *PostGIS*-Bibliothek.

Neben der Möglichkeit Abfragen um geographische Funktionen zu ergänzen bietet *Hibernate Spatial* auch die Möglichkeit in Java-Klassen Attribute zu verwenden, die auf eine Spalte mit geographischen Objekten in der Datenbank gemappt sind. Das folgende Beispiel ist Teil eines solchen Mappings und zeigt ein Attribut mit dem Bezeichner `coordinate` welches vom Typ `org.hibernatespatial.GeometryUserType` ist. Wird einer Datenbank-Spalte dieser Typ zugewiesen, so kann sie alle Formen von geographischen Objekten im *Well-Known Binary* Format speichern.

```
<property name="coordinate" type="org.hibernatespatial.GeometryUserType" not-null="true" />
```

3.2.3 ERM

Daniel Künne

Das *Entity-Relationship-Modell* von Peter Chen geht davon aus, dass die Welt nur aus Entitäten und deren Beziehungen zueinander besteht. Es bietet somit einen sehr einfachen Ansatz Datenbanken zu modellieren, da diese ebenfalls aus Tabellen (den Entitätstypen) und Schlüsseln (den Beziehungen) bestehen.

Import-Schema der OpenStreetMap-Daten

Wie bereits unter *OpenStreetMap* erwähnt ist, werden zuerst die geographischen Daten von den Spielregionen in den Server importiert und dort aufbereitet. Damit es zu keinen Inkonsistenzen zwischen Original-*OpenStreetMap*-Daten und den innerhalb eines Spiels verwendeten Koordinaten kommt, werden die Daten in unterschiedlichen Schemata auf der Datenbank vorgehalten.

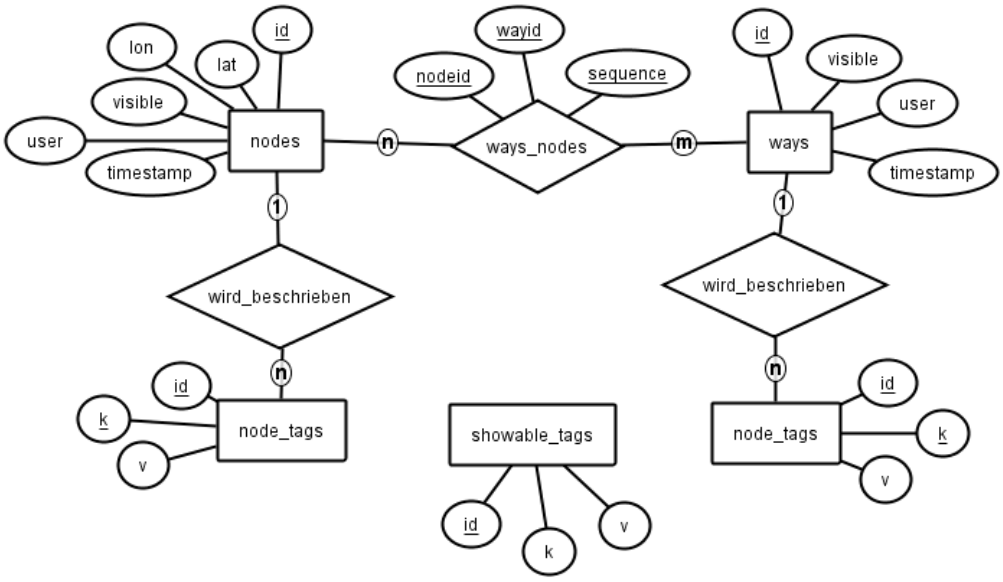


Abb. 29: Datenbank-Schema `osm_data`

Die obige Grafik zeigt die vier Entitäten, die die importierten *OpenStreetMap*-Daten beinhalten. Dabei ist die Anzahl der Attribute von `nodes` und `ways` bewusst gering gehalten um keinen unnötigen Overhead durch die Vielzahl der Datensätze zu erzeugen. Allein die aktuell in der Datenbank gespeicherten Werte von Niedersachsen und Berlin umfassen circa 20 Millionen Datensätze und haben ein Datenvolumen von etwa 2,5 Gigabyte.

Die Tabelle `showable_tags` ist nicht direkt mit den *OpenStreetMap*-Daten verbunden, sondern beinhaltet die Tags, die für den Spieler erreichbare Knoten definieren.

Schema mit den Daten des Spiels

Die Informationen, die für ein laufendes Spiel gespeichert werden, liegen auf einem eigenen Datenbank-Schema `pgs_prod`. Dabei lassen sich zwei große Bereiche voneinander abgrenzen. Auf der einen Seite sind die Informationen, die zu einem konkreten Spiel gehören und auf der anderen die Daten eines konkreten Spielers.

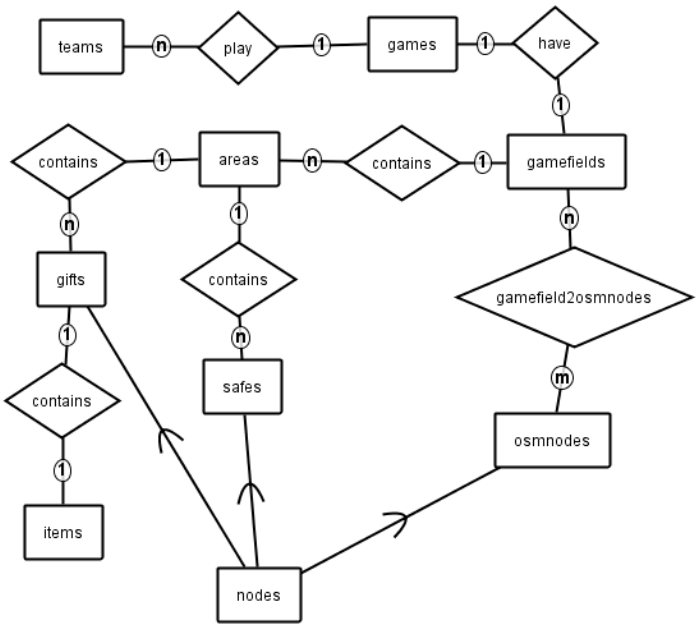


Abb. 30: spielbezogene Daten des Datenbank-Schemas `pgs_prod`

In obigem ER-Diagramm sieht man, dass von dem Entitätstypen `games` zwei Relationships ausgehen. Zum einen ist dies die Verbindung zu der Spielerverwaltung mit dem Entitätstypen `teams` und zum anderen die Verwaltung der relevanten Informationen zu einer Spielfläche in `gamefields`. Zu den dort vorgehaltenen Daten gehören dabei die einzelnen Teilbereiche des Spielfeldes (`areas`), die Menge der für den Spieler erreichbaren Punkte (`osmnodes`) sowie die Informationen zu Tresoren (`safes`) und Geschenkpaketen (`gifts`). Im gezeigten Diagramm sieht man auch, dass intern alle Entitätstypen mit einem Positionsbezug von `nodes` abgeleitet sind und somit intern einfacher zu vergleichen sind.

Die Schnittstelle von spiel- zu spielerbezogenen Daten bilden die Entitätstypen `teams` und `players`. Beide verfügen über eine Beziehung mit einem Inventar, welche wiederum mit den unterschiedlichen gefundenen Gegenständen verbunden ist.

Eine Entität des Typen `players` ist immer genau einem Benutzer (`users`) zugeordnet, da es diesem erlaubt ist zeitgleich in nur einem Spiel aktiv zu sein. Außerdem wird zu einem Spieler gespeichert welche Effekte (`effects`) gerade für ihn aktiv sind. Dabei wird unterschieden zwischen den Sichtbarkeitseffekten der einzelnen Spielfelder und der Sichtbarkeit von Mitspielern, Tresoren und Geschenkpaketen.

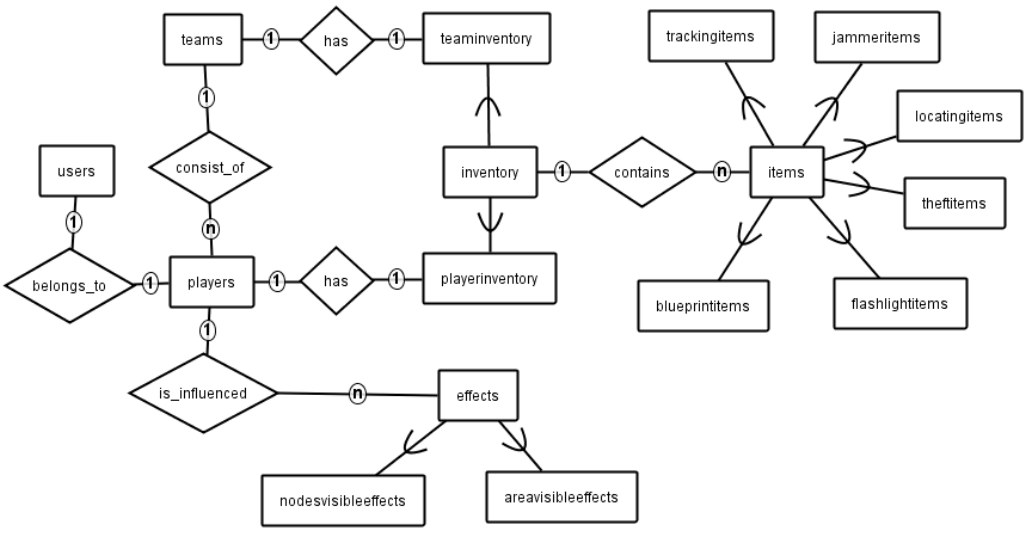


Abb. 31: spielerbezogene Daten des Datenbank-Schemas `pgs_prod`

3.3 Architektur

Andre Schemschat

Nach den Einführungen in die verschiedenen Bestandteile der Datenbank folgt in diesem Kapitel eine Übersicht über alle Komponenten der eigentlichen Serverarchitektur. Diese unterteilt sich anhand der verschiedenen gefärbten Bereiche, die in der Einführung zu sehen sind.

Das Laufzeitverhalten des Servers lässt sich in drei Phasen unterteilen. An erster Stelle steht die Initialisierungsphase. In dieser werden die zentralen Komponenten geladen und alle Module hochgefahren. Dieser Schritt darf und kann vergleichsweise viel Zeit in Anspruch nehmen, da oft dynamisch per *Reflection* Plugins nachgeladen werden. Ist dieser Teil abgeschlossen, befindet sich der Server in seiner Laufzeit-Phase. In dieser sollten alle Aktionen performant implementiert sein, damit eine geringe Latenzzeit garantiert werden kann, wenn es um die Kommunikation mit den Clients geht. Die durchschnittliche Verarbeitungszeit liegt hier in einem geringen dreistelligen Millisekundenbereich. Erhält der Server das Kommando zum Herunterfahren, so wird diese Nachricht an alle laufenden Module weitergeleitet. Diese bekommen dann die Chance sich geordnet zu beenden. Sind alle Module beendet fährt sich auch der Hauptthread herunter und der Server ist gestoppt.

3.3.1 Beans

Andre Schemschat

Der Server setzt zur Datenhaltung auf das ORM-Framework Hibernate. Dies basiert darauf, dass sogenannte Java-Beans oder POJOs (Plain old java objects) mit Hilfe von Annotations oder XML-Mappings auf eine relationale Datenbanktabelle abgebildet werden. Diese Beans bilden die Modelschicht der Architektur. Alle Objekte die im Rahmen der Spiellogik verarbeitet werden sind mit einer Bean und einem Eintrag in einem Mapping vertreten. Im Zusammenhang mit Hibernate gibt es jedoch bei den Beans einige Besonderheiten, die sie von normalen Java-Beans abheben. Jede Bean muss einen öffentlichen Standard-Konstruktor enthalten, der keine Parameter erwartet. Zudem muss für jede Variable, die gespeichert werden soll, ein `Getter` und `Setter` vorhanden sein. Im Falle der `Setter` dürfen weiter keine Gültigkeitsprüfungen, im speziellen eine Überprüfung auf `Null`, stattfinden, da Hibernate nicht zusichert, dass ein Objekt direkt korrekt initialisiert wird. Die beiden Beispiele unten zeigen eine exemplarische Bean und ein Mapping für diese Bean.

```
public class Player extends Node {
    private User user;
    private Team team;
    private int highscore;

    protected Player() {
        this.user = null;
        this.team = null;
        this.highscore = 0;
    }

    public Player(User user, Point coordinate) {
        this();
        setUser(user);
    }

    // getter und setter
}


<class name="Player" table="players">
    <id name="id" column="id" >
        <generator class="sequence">
            <param name="sequence">nodes_id_seq</param>
        </generator>
    </id>

    <many-to-one name="user"
                 column="user_id"
                 class="core.data.authentication.User"
                 not-null="true"
                 unique="true" />

    <many-to-one name="team"
                 column="team_id"
                 class="core.data.game.Team" />

    <property name="highscore" />
</class>
```

3.3.2 Messagebus

Andre Schemschat

Wie in jeder komplexen Architektur müssen die einzelnen Komponenten untereinander verbunden sein, um Daten und Nachrichten auszutauschen. Je fester diese Koppelung ist, um so schneller können untereinander Daten ausgetauscht werden, die Entwicklung ist also schneller und einfacher. Dieser vermeintliche Vorteil wandelt sich jedoch mit steigender Komplexität zum Nachteil, da es schwerer wird den Code sauber zu trennen und die Übersicht zu behalten. Eines der Ziele der gewählten Architektur ist es, neue Erweiterungen so einfach wie möglich zu erlauben, ohne dabei etwas an dem Grundsystem zu ändern. Zu diesem Zweck wurde ein Messagebus integriert. Während der Initialisierung können sich alle Teile an diesem Bus registrieren und mitteilen, auf welche Nachrichten sie reagieren wollen. Während der Laufzeit können dann Nachrichten verschickt werden, die der oder die Empfänger verarbeiten. Die Empfänger sind dabei in ihrer Anzahl nicht eingeschränkt, es können also mehrere Quellen auf die gleiche Nachricht reagieren, so dass gerade Statusmeldungen sehr einfach zu verwenden sind. Das Versenden der Nachrichten läuft dabei synchron ab, dass heißt eine Nachricht wurde erst vollständig von allen Emfpfängern verarbeitet, bevor der Aufruf zurück zu dem Urheber springt. Exceptions, die während der Ausführung auftreten, werden ebenfalls an den Urheber weitergeleitet, so dass dieser auf etwaige Fehler reagieren kann.

```
@Receive public void getGame(MGetGame m) {
    Session s = Database.session();
    Game game = (Game)s.get(Game.class, m.getID());

    if(game == null)
        throw new ControllerException("game_not_found");
    m.setGame(game);
```

```
}


MGetGame getgame = new MGetGame(gameId);
send(getgame);
Game game = getgame.getGame();
```

Wie im obigen Beispiel gesehen werden bestimmte Typen von Objekten über den Messagebus geschickt, so genannte `Messages`. Die Nachrichten fungieren als eine Art Umschlag für die Informationen, die benötigt werden um die Anfrage auszuführen. Sie bestehen meistens aus einem Konstruktor, der die erwarteten Parameter entgegennimmt und aus `Settern` und `Gettern`, um die Parameter zu lesen und die Antwort zu setzen.

Neben dieser allgemeinen Art von Nachricht gibt es noch zwei spezielle Subtypen, die sich in einigen Punkten unterscheiden. Zum einen die `PassiveMessage`, zum anderen die `SingleResponseMessage`. Ersteres markiert eine Nachricht, die lediglich für passive Zwecke eingesetzt wird. Passive Nachrichten werden immer dann verschickt, wenn sich etwas geändert hat, was andere interessieren könnte, wie z.B. das ein Spieler ein Spiel verlassen hat. Letztere sind Anfragen, die nur ein einziges Ergebnis zurückliefern und dadurch vereinfacht gehandhabt werden können. Für beides ist im folgenden ein Beispiel gelistet.

```
public class MGameCreated extends PassiveMessage {
        private Game game;

        public MGameCreated(Game g) {
                this.game = g;
        }

        public Game getGame() {
                return this.game;
        }
}


public class MGetGame extends SingleResponseMessage<Game> {
        private Game game;
        private int id;

        public MGetGame(int id) {
                if(id < 1)
                        throw new IllegalArgumentException();
                this.id = id;
                this.game = null;
        }

        public int getID() {
                return this.id;
        }

        @Override
        public Game getResponse() {
                return game;
        }

        @Override
        public void setResponse(Game game) {
                if(game == null)
                        throw new IllegalArgumentException();
                this.game = game;
        }
}

// das vereinfachte senden der Nachricht
Game g = (new MGetGame(5)).send(this);
```

3.3.3 Controller

Andre Schemschat

Das Zentrum der Architektur befindet sich in den *Controllern*. In ihnen befindet sich die gesammelte Geschäftslogik, sowie die einzige Interaktion mit der Datenhaltung. Alle Aktionen, die zum Spielverlauf ausgeführt werden müssen dementsprechend in dieser Schicht implementiert werden. Jeder *Controller* wird dafür initialisiert und mit dem Messagebus verknüpft, um auf eingehende Nachrichten zu reagieren und eine entsprechende Aktion auszuführen. Dabei können selbst wieder weitere Nachrichten versendet werden, um eine Aufgabe in kleinere Unterportionen zu zerteilen. Weiter sind Controller der einzige Teil der Applikation, der mit der Datenbank interagieren darf. Dies sorgt für einen einheitlichen Umgang mit der Datenschicht. Das abgebildete Sequenz-Diagramm zeigt einen typischen Verlauf den eine versendete Nachricht nimmt. Um die Abgeschlossenheit der Transaktion sicherzustellen dürfen Controller-Objekte jedoch selbst keine Transaktion öffnen, sie nutzen lediglich eine geöffnete Session. Tritt innerhalb der Verarbeitung ein Fehler auf, so werfen die Methoden eine `ControllerException`.

Das Sequenzdiagramm zeigt einen kurzen Ausschnitt einer Kommunikation innerhalb verschiedener *Controller*. Aus einem externen Teil kommt die Anfrage einen neuen Spieler anzulegen. Diese Bitte wird von dem Bus an den `Playercontroller` weitergeleitet, der seinerseits wieder eine Nachricht an den `TeamController` schickt um ein Team zu laden. Beide Methoden interagieren mit der Datenbank, um bestimmte Objekte zu laden oder zu speichern, jedoch kennt keiner die Implementationsdetails des jeweils Anderen. Der gesamte Ablauf erfolgt dabei synchron, die Quelle der Anfrage bekommt also erst dann eine Antwort, wenn alle Unteraufrufe erfolgreich beendet worden sind.

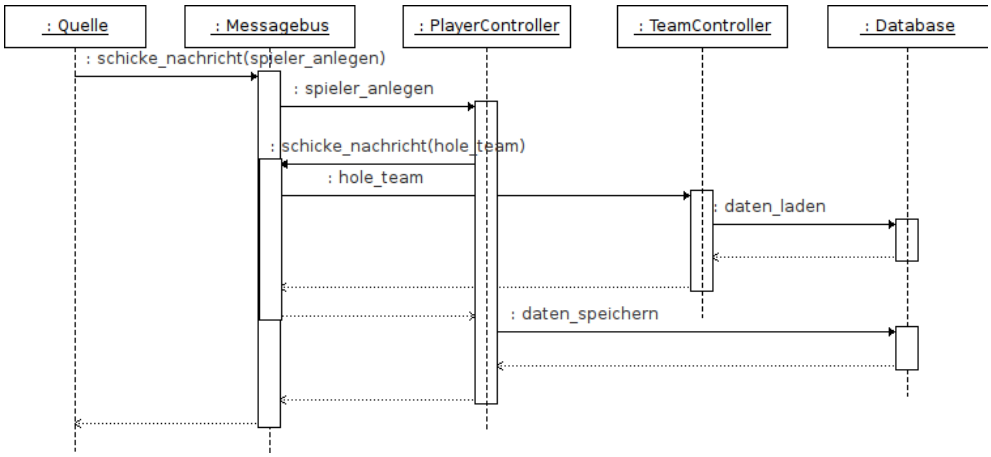


Abb. 32: Ablauf einer Anfrage an die Controller

3.3.4 Taskmodule

Andre Schemschat

Neben den Anfragen der Klienten gibt es auch verschiedene Routine-Aufgaben, die in regelmäßigen Abständen ausgeführt werden müssen. Um diese Anforderung abzubilden, wurde ein Modul in den Server integriert, dass sich genau um diese Aufgaben kümmert. Während der Initialisierungsphase lädt das Hauptmodul alle Tasks und startet für jeden einen einzelnen Thread. Jeder der Tasks gibt dabei entweder ein Interval an, in dem es ausgeführt wird, oder ob er am Ende bzw. Anfang einmalig ausgeführt werden möchte. In kontinuierlichen Abständen werden dann die Verabeitungsroutinen gestartet, die z.B. neue Tresore oder Geschenke auffüllen oder Spiele, deren Spielzeit abgelaufen ist, schließen.

3.3.5 Networkmodule

Andre Schemschat

Um nicht nur intern die Daten verarbeiten zu können, sondern auch externe An- und Abfragen, wird eine weitere Komponente benötigt. Um diesen Teil kümmert sich das Netzwerkmodul. Es ist wiederum modular um einen allgemeinen Teil angeordnet, um flexibel neue Kommandos implementieren zu können. An der Basis lauscht ein Socket auf neue Anfragen. Verbindet sich einer der Klienten mit dem Server, so wird ein neuer Thread aus einem Threadpool geholt um den Mehrbenutzerbetrieb zu ermöglichen. Innerhalb dieses Threads werden dann verschiedene, auf einander aufbauende Verarbeitungsschritte ausgeführt. Angefangen wird damit, die Anfrage zu lesen und anschließend das JSON-Format zu parsen. Die umgewandelten Informationen werden dann an sogenannte `Annotations` weitergereicht. Diese stellen eine Art Vorverarbeitungsschicht dar. Die `Annotations` können flexibel erweitert werden und werden in der Initialisierungsphase dynamisch in die Netzwerkschnittstelle eingeladen. Durch sie wird z.B. die mitgeschickte Session in ein gültiges Benutzerobjekt umgewandelt oder Informationen über das betrachtete Spiel eingefügt. Nachdem die Anfrage an alle `Annotations` durchgereicht wurde kommt die Hauptebene, die `Actions`. Diese werden ebenfalls, ähnlich zu der Vorstufe, dynamisch geladen. Der nebenstehende Codeausschnitt zeigt eine beispielhafte Implementierung einer Action. Sowohl `Annotations` als auch `Actions` können eine Liste von Abhängigkeiten angeben und das Netzwerkmodul garantiert, dass diese eingehalten werden. Ist dies nicht möglich, der entstandene Graph also nicht topologisch sortierbar, schlägt die Initialisierungsphase fehl.

```
{   "meta": {
        "type": "get_games",
        "id": "spectator"
    },
    "data": {}
};

{   "get_games": [
        {   "id":1,
            "players": {
                "cops":[],
                "thieves":["Spielleiter_1", "Android"]
            },
            "name":"Westerberg",
            "numberofplayers":2,
            "maximum_safes":5
        },
        {   "id":4,
            "players": {
                "cops": ["iPhone4", "Aton"],
                "thieves": ["daniel", "Spielleiter_4", "andre", "iPhone3GS"]
            },
            "name": "CeBIT-Stand",
            "numberofplayers": 6,
            "maximum_safes": 3
        },
        {
            "id": 3,
            "players": {
                "cops": ["Sebastian"],
                "thieves": ["Spielleiter_3"]
            },
            "name": "CeBIT",
            "numberofplayers": 3,
            "maximum_safes": 5
        }
    ]
}
```

3.3.6 Historymodule

Daniel Künne

Um bereits komplett abgeschlossene Spiele im Nachhinein betrachten zu können, müssen im Verlauf des Spieles bestimmte Aktionen gespeichert werden. Hierfür wird das Historymodule eingesetzt, welches sich am *Messagebus* registriert und auf die zu den Aktionen gehörigen Nachrichten reagiert.

Solange für ein Spiel noch nicht die Nachricht der Beendigung beim Historymodule angekommen ist, werden die Daten in einem allgemeinen Format in eine Tabelle der Datenbank geschrieben. Die wesentlichen Informationen eines Datensatzes sind der Zeitpunkt des Auftretens, die ID eines möglichen Elternelementes, die ID des aktuellen Elementes, der Typ dieses Elementes sowie die Art der Nachricht mit möglichen ergänzenden Informationen.

Beispielsdatensätze aus der History

timestamp	parent_id	child_id	type	message	params
2011-02-27 17:42:44.303	1	718258	gift	created	position:POINT (8.0214282 52.2839609);item:1;
2011-02-27 17:48:49.882	4	718318	player	moved	position:POINT (8.07649612664144 52.25366454280925);

In obiger Tabelle sind zwei Beispiele von Einträgen zu sehen, wie sie das History-Module erstellt hat. Die erste Zeile gibt den Zeitpunkt und die geographische Koordinate einer neu erstellten Geschenkbox an, während in der zweiten Zeile die Positionsveränderung eines Spielers weggeschrieben wurde.

Empfängt das Historymodule die Nachricht, dass das Spiel beendet wurde, so startet es im Hintergrund eine Funktion auf der Datenbank, die die gesammelten Informationen wieder für die Anzeige im Gameviewer aufbereitet. Diese Aufbereitung kann erst nach Abschluss des Spiels erfolgen, da ansonsten die Bearbeitungszeit der einzelnen Anfragen an den Server unnötig verlängert würde und es zu Verbindungsunterbrechungen bei den Clients kommen würde.

3.3.7 Agentmodule

Andre Schemschat

Das Agentenmodule beinhaltet die Schnittstelle zu dem Agenten, also den künstlichen Gegenspielern. Dieser funktioniert in einer Standalone-Umgebung, wie in Agent beschrieben. Das Module bietet jedoch eine weitergehende Integration in den Server an. So ist es möglich, wie unter Gameviewer erläutert, Agenten einem Spiel beitreten zu lassen, oder direkt zu Spielbeginn eine Spieleranzahl festzulegen, die dann im Notfall mit Agenten aufgefüllt wird. Zur Integration wird ein Interface genutzt, dass die nötigen Start- und Status-Methoden anbietet. Erhält das Modul den Auftrag einen neuen Agenten zu starten, dann wird zuerst eine neue Instanz der Hauptklasse angelegt. Alle Fehlerbehandlungen und Log-Nachrichten werden dabei an einen definierten Handler weitergereicht, der die volle Kontrolle über den Agenten übernimmt und im Falle eines Fehlers die nötigen Shutdown-Routinen einleiten kann.

4. Smartphones

Bei der Auswahl der Mobiltelefone für das Projekt, hat sich angeboten, Smartphones wegen ihren wesentlich überragenden Funktionalitäten, insbesondere bezogen auf bisherige Handys oder auch sogenannte PDAs, zu nehmen. Ein Smartphone ist durch unterschiedliche Software erweiterbar und ist oftmals auch durch seinen großen Bildschirm gekennzeichnet. In solchen Smartphones sitzen außerdem relativ potente Prozessoren, Sensoren wie Accelerometer oder auch Magnetometer und in den meisten Fällen sogar eigene Grafikprozessoren.

Als moderne Smartphones lassen sich unter anderem iPhones jeglicher Generation zählen, aber auch andere Geräte der verschiedenen Hersteller wie das HTC Desire oder das Samsung Galaxy S, um nur einige zu nennen. Am meisten verbreitet auf Smartphones sind die drei Betriebssysteme: iOS (bei iPhones), Android (HTC Desire, Samsung Galaxy S, etc.) und Windows Mobile (HTC HD2, etc.).

Es wurde beschlossen, in diesem Projekt sowohl für Android als auch für iOS zu entwickeln. Da die Systeme selbst als auch die Entwicklungsumgebungen sowie die Vorgehensweise beim Entwickeln sich ziemlich unterscheiden, werden die beiden Systeme im Weiteren gesondert behandelt.

4.1 Android

Sascha Henke, Peer Wagner

Android[14] ist anders als man es im ersten Moment vermuten könnte nicht ein einziges oder auch nur eine einzige Art von Handys, sondern es ist ein Linux-basiertes Betriebssystem für Smartphones und zunehmend auch Netbooks, welches maßgeblich von *Google* und der *Open Handset Alliance*[15] entwickelt wird. Es handelt sich um ein Open-Source System, das seit dem 22. Oktober 2008 der Öffentlichkeit zur Verfügung steht und bis heute weiterentwickelt wird. Aktuell liegt es in der Version 2.3.3 beziehungsweise 3.0 vor, wobei letztere jedoch allein Tablet-PC's zur Verfügung steht und in der der nächsten Version mit dem eigentlichen Handy-Betriebssystem wieder zusammengeführt werden soll.

Das Android Betriebssystem wird in erster Linie über einen Touchscreen bedient, jedoch werden auch eine Reihe von Hardwaretasten definiert, die in jedem Android-Gerät verbaut sein müssen. Pflichttasten sind hierbei die "Menü"-, "Home"- und "Zurück"-Taste. Das Android System steht unter der Apache 2.0, GPLv2 Lizenz.

4.1.1 Geschichte

Sascha Henke

Marktanteile

Marktanteile Smartphones[16]

Company	Units	3Q10 Marked Share	Units	3Q09 Marked Share
Symbian	29,480.1	36.6	18,314.8	44.6
Android	20,500.0	25.5	1,424.5	3.5
IOS	13,484.4	16.7	7,040.4	17.1
Research In Motion	11,908.3	14.8	8,522.7	20.7
Microsoft Windows Mobile	2,247.9	2.8	3,259.9	7.9

14 http://www.android.com
15 http://www.openhandsetalliance.com/
16 http://www.gartner.com/it/page.jsp?id=1466313

Linux	1,697.1	2.1	1,918.5	4.7
Other OS	1,214.8	1.5	612.5	1.5
Total	80,532.6	100.0	41,093.3	100.0

Die vorangehende Tabelle verdeutlicht zum einen die deutliche Vergrößerung des Smartphone Marktes in den letzten Jahren und zum anderen den wachsenden Marktanteil von Android-Geräten. So ist der Marktanteil an Smartphones mit Android-Betriebssystem vom dritten Quartal 2009 bis zum dritten Quartal 2010 von 3.5% auf 25.5% gewachsen.

Versionenübersicht

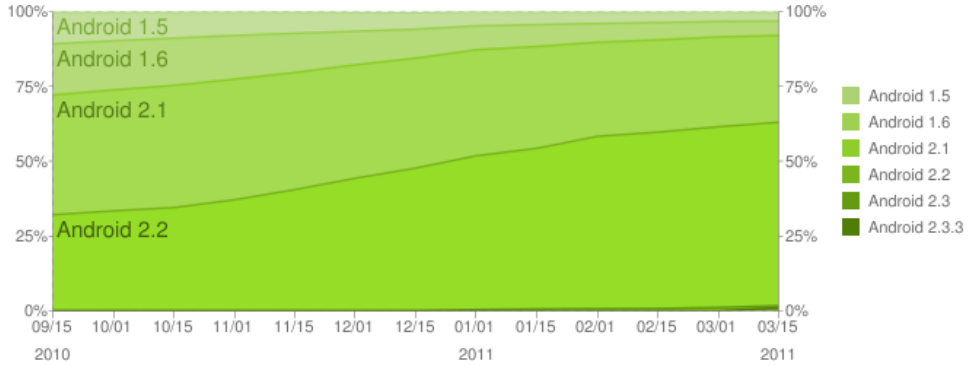


Abb. 33: Historische Versionsverteilung

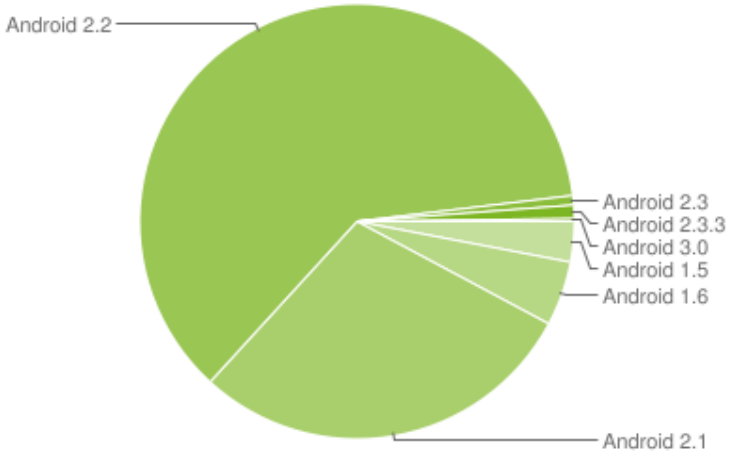


Abb. 34: Derzeitige Android Versionsverteilung

Beide Grafiken zeigen eine Google Statistik vom 15. März 2011 [17]. Die oberste Grafik zeigt die historische Verteilung der bisher vorhandenen Android Versionen, gemessen an den Zugriffen auf den Android-Market . Deutlich zu erkennen ist die Zunahme der Version 2.2, welche, wie im unterem Bild zu erkennen ist, momentan mit 61.3% die meißt verbreitete Android Version darstellt.

Versionenübersicht

Seit der Android Version 1.5 vom April 2009 trägt jede Android Version den Namen einer Süßspeise. Im Folgenden soll ein Überblick über die verschiedenen Versionen und ihre Neuerungen geschaffen werden:

17 http://developer.android.com/resources/dashboard/platform-versions.html -Android Dashboard, Platform Versions

- 1.5 **Cupcake**
  - Abspielen von Videos
  - Bluetooth Unterstützung
  - Bildschirmtastatur
  - Automatischer Wechsel zwischen Hoch- und Querformat
- 1.6 **Donut**
  - Gestenerkennung
  - VPN Konfiguration
- 2.0 **Eclair**
  - Bluetooth 2.1 Unterstützung
- 2.1 **Eclair**
  - Bugfixes der vorigen Version
  - HTML 5
  - Linux Kernel 2.6.32
  - JIT-Compiler
  - OpenGL ES 2.0 Erweiterung
- 2.2 **Froyo**
  - Flash 10.1 Unterstützung
  - Appinstallation auf SD Karte
- 2.3 **Gingerbread**
  - Gyroskop Unterstützung
  - Höhere Auflösung wird unterstützt
  - Linux Kernel 2.6.35
  - Ext4 Unteratützung
- 3.0 **Honeycomb**
  - Tablet Oberfläche
- 3.1 **Ice Cream** (noch kein Release)
  - Zusammenführung der 2er und 3er Entwicklungslinie

Der Entwicklungssprung am 26. Oktober 2009 auf die Android Version 2.0 stellte grade im grafischen Bereich eine große Neuerung dar, da nun die OpenGL ES 2.0 API im *Native Development Kit (NDK)* zur Verfügung gestellt wurde. Dies ermöglicht es Hardwarenah in C oder auch C++ zum Beispiel Opengl-Grafiken zu berechnen und auf dem Android Gerät darzustellen.

Android Market

Der Android Market ist, wie auch seine Konkurrenten *App Store* und *Windows Phone Marketplace*, eine Plattform um Programme für das System zu veröffentlichen bzw. sich Applikationen zu laden. Die Android Market-Applikation ist bei derzeitigen Geräten mit Android-System vorinstalliert, sodass schon nach Auslieferung des Gerätes zusätzliche Applikationen nachinstalliert werden können.

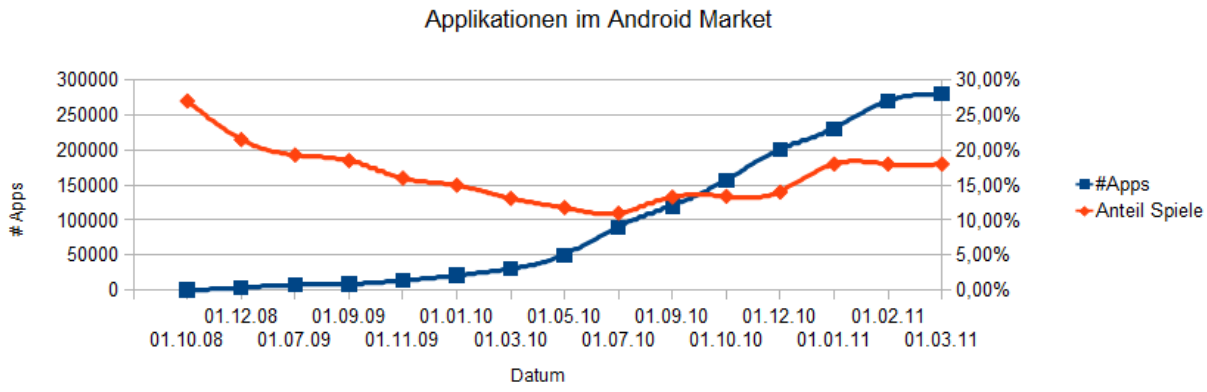


Abb. 35: Apps im Android Market

Der Android Market wurde im Oktober 2008 in betrieb genommen. Von den anfänglichen 167 Apps im Market sind aufgrund der Entwicklung von Drittanbietern und freien Programmierern die Anzahl auf momentan etwa 280.000 Applikationen gestiegen [18]. Von diesen Applikationen sind etwa 18% Spiele. Dies zeigt, dass das Interesse an Mobilen Unterhaltungsprogrammen deutlich zugenommen hat und Marktpotential besitzt.

Um als Entwickler eine Applikation in den Android Market zu stellen, ist lediglich eine einmalige Registrierungsgebühr von 25 Dollar fällig. Beim Verkauf einer Applikation verlangt Google, wie auch die Konkurrenten Apple und Microsoft, eine Transaktionsgebühr in Höhe von 30% [19].

4.1.2 Programmieren auf Android

Sascha Henke

Dalvik Virtual Machine

Das Android-System beruht in erster Linie auf der *Dalvik Virtual Machine*[20], einer von Dan Bornstein entwickelten Portierung der *Java Virtual Machine (JVM)* für mobile Endgeräte. So beruht diese im Gegensatz zur JVM auf einer virtuellen Registermaschine anstatt eines übliche Kellerautomaten. Weiterhin werden beim Kompilieren alle `.class` Dateien in eine `.dex` (*Dalvik Executable*) Datei zusammengeführt, wobei einige Optimierungen vorgenommen werden, um zum Beispiel den Speicherbedarf zu minimieren.

Activities

Eine Activity ist der elementare Bestandteil einer Applikation um eine Interaktionsmöglichkeit mit dem Nutzer zu schaffen. Innerhalb einer Activity kann mittels `setContentView` die Oberfläche angelegt bzw. importiert werden, welche üblicherweise den gesamten Schirm ausfüllt oder in eine andere Activity eingebettet ist. In der Lebensdauer einer Activity können verschiedene Ereignisse auftreten, welche den Status der Activity ändern, so kann z.B. ein Druck auf den Home Button, oder ein eingehender Anruf eine vorher im Vordergrund befindliche Activity in den Hintergrund schieben. Um diese Einflüsse in der Applikation berücksichtigen zu können, gibt es den Activity Lifecycle. Wenn nun eine Klasse von Activity erbt, so kann diese die Methoden `onCreate()`, `onStart()`, `onResume()`, `onPause()`, `onStop()`, `onDestroy()` und `onRestart()` überschreiben um auf einen Wechsel des Zustands angemessen reagieren zu können. Der Aufruf von `onStop()` sei hier gesondert aufgeführt, da dieser nicht zwangsläufig direkt vom Nutzer ausgeführt werden muss. `onStop()` kann ausgeführt werden wenn eine weitere Activity durch `onResume()` wieder in der Vordergrund tritt und diese überdeckt. Dies kann ebenfalls durch eine neu gestartete Activity geschehen.

18 http://de.wikipedia.org/wiki/Android_Market - Wikipedia, Android Market
19 http://market.android.com/support/bin/answer.py?answer=112622 - Android Market, Transaktionsgebühren
20 http://www.dalvikvm.com/ - Dalvik Virtual Machine

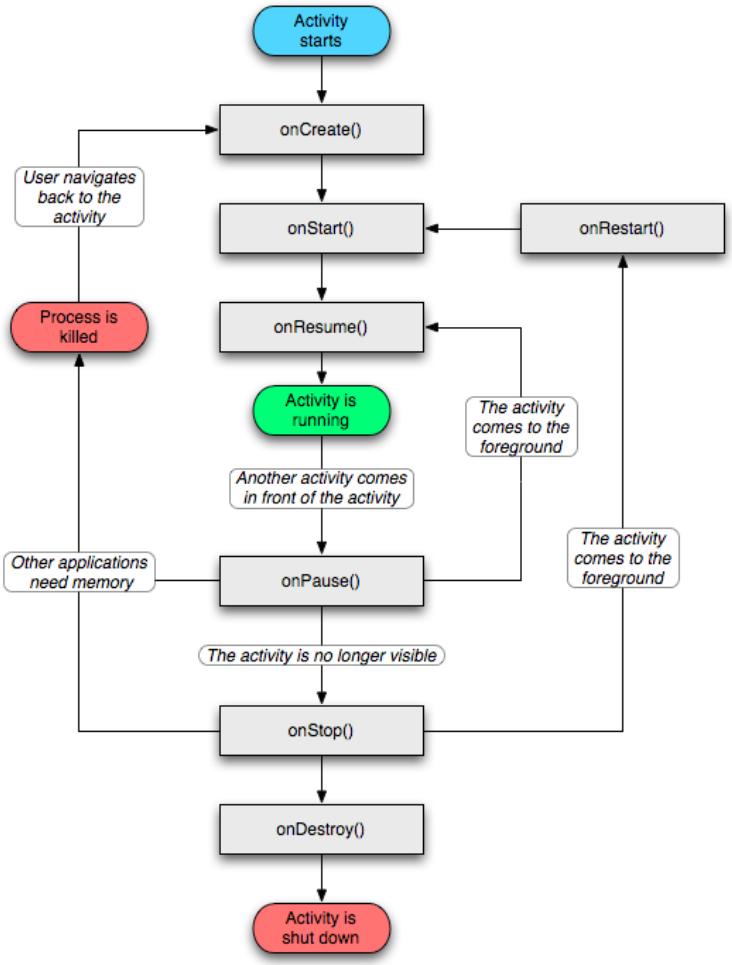


Abb. 36: Activity Lifecycle

Das vorangehende Bild[21] zeigt die verschiedenen Phasen einer Activity und die möglichen Nachfolgezustände. Der Lebenszyklus einer Activity geschieht zwischen dem ersten Aufruf von `onCreate()` und dem Aufruf von `onDestroy()`, welcher alle genutzten Ressourcen der Activity wieder freigibt. Der sichtbare Lebenszyklus geschieht zwischen den Aufrufen von `onStart()` und `onStop()`, diese Methoden können in der Laufzeit der Activity mehrfach aufgerufen werden, je nachdem wie oft diese in den Vorder- bzw. Hintergrund geschoben wird. Während dieser Zeit kann der Nutzer die Activity im Vordergrund sehen auch wenn in dieser Zeit noch ggf. keine Interaktionen möglich sind. Eine Interaktionsmöglichkeit ist lediglich zwischen den Aufrufen `onResume()` und `onPause()` möglich.

Um innerhalb einer Activity in eine andere zu wechseln werden sogenannte `Intents` benötigt. Ein `Intent` ist hierbei ein abstraktes Objekt das zum Starten einer Activity, Benachrichtigung eines `Broadcastreciever` oder zum Kommunizieren mit einem `Service` dient. So kann mit dem folgenden Beispiel:

```
Activity activity = this;
intent = new Intent(currentActivity.this, nextActivity.class);
startActivity(intent);
activity.finish();
```

von der aktuellen Activity `currentActivity` in die `nextActivity` gewechselt werden. Mit dem Aufruf von `activity.finish()` wird sichergestellt, dass die aktuelle Activity beendet wird, bzw. `onDestroy()` aufgerufen wird, damit diese vom *Garbage-Collector* abgeräumt werden kann und der Speicherplatz wieder freigegeben wird. Der Zeitpunkt des Abräumens ist auch hierbei vom System bestimmt.

21 http://developer.android.com/reference/android/app/Activity.html - Android API, Beschreibung des Activity Lifecycle

Services

Ein Service[22] ist eine Komponente die es ermöglicht langwierige Operation ohne Nutzerinteraktion oder Funktionalität für andere Applikationen bereitzustellen. Hierbei sei jedoch erwähnt das ein Service kein separater Prozess ist. Standardmäßig läuft der Service innerhalb des selben Prozesses wie die Applikation dem er angehört.

Ein Service ist somit eine Hilfsmöglichkeit dem System mitzuteilen das eine Operation im Hintergrund ausgeführt werden soll. Dazu dient die Funktion `Context.startService()` um eine Verzahnung des Services mit der Applikation zu schaffen, bis dieser explizit angehalten wird. Per `Context.bindService()` kann man eine persistente Verbindung zu einem Service erhalten. Falls dieser noch nicht gestartet wurde, so wird bei dem Aufruf der Service angestoßen und bleibt in diesem Falle so lange bestehen, wie noch eine Verbindung vorhanden ist.

Manifest

Um eine Activity oder einen Service starten zu können, müssen diese in dem Android-Manifest aufgeführt sein. Das Android Manifest ist ein XML Dokument, welches die essentiellen Bestandteile einer Android Applikation enthält. Dieses Dokument enthält:

- Den Java Package namen
- Komponenten der Applikation:
  - Activities
  - Services
  - Broadcast Reciever
  - Content Provider
- Permissions, die das Nutzen von geschützten API-Elementen erlaubt
- Minimales API-Level
- Libraries die enthalten sein müssen

Die Permissions [23] in einem Manifest erlauben einer Applikation den Zugriff auf geschützte API-Schnittstellen, über die der Nutzer bei der Installation der Applikation informiert wird. Diese Permissions umfassen den Zugriff auf GPS-Daten, Kamera, Internet und einiges mehr.

User Interface

Die Nutzeroberfläche des Androids besteht aus zwei Grundelementen:

- **Views**: Die Basiselemente der Oberfläche. Die Klasse `View` stellt die Basisklasse der sog. "widgets" dar, welche die voll implementierten UI-Elemente repräsentieren.
- **Viewgroups**: Eine Viewgroup stellt einen Container für die verschiedenen Views dar und dient als Basisklasse für die "Layouts", welche die Positionierung der Views ermöglichen.

Viewgroups können beliebig verschachtelt werden und somit einen Hierarchiebaum aufgebauen. Beim Zeichnen der Oberfläche fordert vom Wurzelknoten abwärts jede Viewgroup ihre Kindknoten auf sich zu Zeichnen. Hierbei können die Views zwar eine Größe und Position definieren, jedoch hängt das finale Aussehen eines Elementes stärker von den Parametern des zugehörigen Vaterknoten, also dem Layout, ab.

Vordefinierte Layouts:

- **Linear Layout**, ermöglicht eine lineare Anordnung der Kindelemente.
- **Relative Layout**, ermöglicht die Positionierung von Elementen relativ zueinander.

22 http://developer.android.com/reference/android/app/Service.html - Android API, Beschreibung des Service
23 http://developer.android.com/reference/android/Manifest.permission.html -Android API, Auflistung der Permissions

- **Table Layout**, stellt die Kindelemente in Zeilen und Spalten dar.
- **Grid View**, die Kindelemente werden automatisch durch einen `ListAdapter` in ein zweidimensionales, scrollbares Grid eingefügt.
- **Tab Layout**, diese Viewgroup besteht aus drei Elementen. Dem `TabHost`, welcher der Wurzelknoten in der Hirarchie ist, den `Tabwidget` der innerhalb eines `Framelayout` den Inhalt der View enthält. Durch den Tabhost kann man nun in diesem Layout je nach Implementation zwischen verschiedenen Views oder Activities hin und herwechseln.
- **List View**, die Kindelemente werden hierbei als skrollbare Liste dargestellt.

Der typische Weg eine Oberfläche in Android zu definieren ist per XML-Layout-Datei. Hierbei werden in XML-Syntax die Viewgroups und Views geschachtelt und mit Parametern für die Ausmaße, Position und weiterem versehen. Ein wichtiger Parameter innerhalb der beiden Elemente ist die ID. Mit dieser ID kann innerhalb der XML-Datei Bezug auf ein anderes Element hergestellt werden. Weiterhin kann über die ID innerhalb einer Activity auf ein View-Element bzw. dessen Events reagiert werden oder es können in einer Viewgroup dynamisch neue Elemente hinzugefügt werden.

Eine weitere Eigenheit in der UI-Gestaltung auf dem Android ist die Benutzung von *NinePatch* Grafiken. Soll eine View, z.B. ein Button, mit einem Bild versehen werden so ist die Nutzung von *NinePatch* Bildern empfohlen, da diese neben dem üblichen Bild auch noch einen 1 Pixel großen Rand besitzen, der das Verhalten beim Skalieren der Grafik definiert.

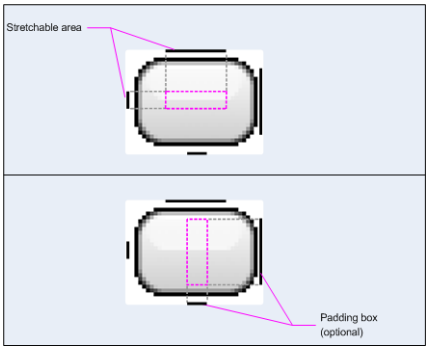


Abb. 37: NinePatch

Das nebststehende Bild [24] zeigt die Erzeugung einer *NinePatch* Grafik. In der Mitte sieht man das eigentliche Bild, der Pixelrand am linken und oberen Rand definiert den Bereich der bei einer Skalierung vergrößert bzw. verkleinert werden darf. Weiterhin kann optional am unteren und rechten Rand der Grafik noch ein Padding-Bereich angegeben werden. So wird nun durch den oberen und linken Rand der skalierbare Bereich und durch den rechten und unteren Rand der Bereich definiert in welchem der Inhalt der View angezeigt werden soll. Wenn der Inhalt nicht in diese Padding Region passt, so wird die View soweit skaliert bis der Bereich groß genug ist.

Beispiel

Um das Anlegen einer Oberfläche per XML und dynamischen hinzufügen zu verdeutlichen sei hier ein Beispiel aufgezeigt, welches zur Darstellung des Inventories auf dem Android dient. Zunächst wurde ein XML-Layout erstellt welches als Grundgerüst dient.

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

<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:orientation="vertical"
        android:keepScreenOn="true">

        <HorizontalScrollView android:id="@+id/inventory_scrollview"
                android:layout_width="fill_parent"
                android:layout_height="fill_parent">

                <LinearLayout android:id="@+id/inventory_root"
                        android:layout_width="fill_parent"
                        android:layout_height="fill_parent"
                        android:orientation="horizontal"/>
```

24 http://developer.android.com/guide/topics/graphics/2d-graphics.html - Android API, 2D Graphics Guide

```
        </HorizontalScrollView>

</RelativeLayout>
```

Hier dient als Wurzel-Viewgroup ein *Relative Layout* welches durch die Parameter *fill_parent* den gesamten zur Verfügung stehenden Platz ausfüllt. Weiterhin ist nun eine *HorizontalScrollView* verschachtelt mit einem *LinearLayout*, dies führt dazu, dass wenn nun dynamische Elemente in das *LinearLayout* eingefügt werden, so sind diese durch die *HorizontalScrollView* immer erreichbar, selbst wenn die Größe des Bildschirms nicht reicht um alle Elemente gleichzeitig anzuzeigen. Um nun dynamisch Elemente in das *Inventory* hinzufügen zu können wird zunächst einmal die ID der Viewgroup benötigt in dem die neuen Elemente eingefügt werden sollen. So kann man nun in der *Inventory Activity* per

```
root = (RelativeLayout) RelativeLayout.inflate(this, R.layout.inventory, null);
[...]
LinearLayout content = (LinearLayout)root.findViewById(R.id.inventory_root);
```

die Oberfläche zeichnen und eine Referenz zur obersten Viewgroup erzeugen und anhand dieser das *LinearLayout* bekommen in dem die neuen Elemente eingefügt werden sollen. In diesem Falle wird nun eine Schleife mit allen vorhandenen Items durchlaufen und diese in die View eingefügt:

```
[...]
for(Class<? extends Item> type: inventoryClasses){
    [...]
    entry = inventoryModel.getInventory(type);
    LinearLayout ll = (LinearLayout)View.inflate(this, R.layout.inventory_item, null);

    // ImageButton des Userinventories
    imageButton = (ImageButton)ll.findViewById(R.id.inventory_user_image);
    imageButton.setTag(type);
    imageButton.setImageResource(entry.imageResource);

    [...]
    content.addView(ll);
    [...]
}
```

Hierbei wird nun in der Schleife ermittelt um welches *Item* es sich handelt und mit Hilfe des *inventory_item* Layouts wird dann die Oberfläche eines Gegenstandes erzeugt. So wird nun mit Hilfe von `findViewByID` weiter innerhalb des *inventory_item* Layouts navigiert um den enthaltenen *imageButton* dem Typ entsprechend anzupassen. Bei diesem Handelt es sich um den Knopf zum aktivieren des *Items* auf Nutzerseite.

Werkzeuge

Emulator

Das Android SDK umfasst einen Emulator, welcher das Testen einer Applikation ohne ein vorhandenes Gerät ermöglicht. Dieser Emulator ist eine auf der freien virtuellen Maschine *Quick Emulator (QEMU)* basierende Applikation, die eine Simulation einer mobilen ARM Maschine mit kompletten Android-Betriebssystem zur Verfügung stellt. Der Emulator ahmt hierbei die meisten Eigenschaften eines Smartphones nach. Um nun eine Applikation auf verschiedenen Geräten simulieren zu können, bietet der Emulator die Möglichkeit mehrere *Android Virtual Devices (AVDs)* anzulegen, die verschiedene Hardware, so wie Auflösung, SD-Karten Support und so weiter besitzen.
Der Emulator bietet weiterhin die Möglichkeit vordefinierte GPS-Daten an das simulierte Gerät zu senden, sodass innerhalb einer Applikation der `LocationManager` bei einer Anfrage auf die GPS-Position diese vordefinierten Koordinaten ausgibt. Auf diese Weise ist es auch möglich die Lagesensoren des Gerätes zu manipulieren. Jedoch unterliegt der Emulator auch einigen Limitierungen:

- Keine richtigen Anrufe sind emulierbar.
- Keine Unterstützung von USB Verbindungen.
- Keine Kameraaufzeichnungen möglich.
- Keine SD-Karten Einlegen oder Auswurf simulierbar.
- Keine Bluetooth Unterstützung.
- Keine Möglichkeit den Batterieladestatus zu verändern.

Übersicht

Zusätzlich zum Emulator enthält das Android-SDK einige Werkzeuge zum Entwickeln bzw. Debuggen einer Android-Applikation. Nachfolgend ein kleiner Ausschnitt der gängigsten Tools die im Installationsverzeichniss des Android-SDK zu finden sind:

- **Dalvik Debug Monitor**: Hiermit können die einzelnen Threads auf dem Emulator überwacht werden, sowie die aktuelle Heap-Belegung, aktuelle Allokationen und Debug-Logs eingesehen werden.
- **Hierarchy Viewer**: Hiermit kann die Baumstruktur des aktuell betrachteten Layouts angezeigt und überprüft werden.
- **Draw 9-patch**: Erlaubt das Erstellen von *NinePatch* Grafiken.
- **sqlite3**: Ermöglicht das Auslesen und Manipulieren der internen SQLite Datenbank
- **Monkey**: Der Monkey dient zum Stress-testen einer Applikation. Dieser generiert eine zufällige Reihe von Events, so wie Klicks, Gesten, Druck auf Hardwareknöpfen usw..

Weiterhin wird ein Eclipse Plugin zur Verfügung gestellt, welches einige Tools wie den *Dalvik Debug Monitor* und den Emulator in die Entwicklungsumgebung integrieren.

4.1.3 Übersicht der Activities

Peer Wagner

In der von uns entwickelten Applikation werden vier große Activities eingesetzt, welche die verschiedenen Aufgaben des Codes kapseln. Dies ist ein für die Android-Entwicklung empfohlener Weg, da er ähnlich dem Model-View-Controller Prinzip die Applikation modularisiert und damit die unterschiedlichen Bereiche einfacher austauschbar macht. Die konsequente Anwendung der Modularisierung hat sich während der Entwicklung des öfteren bewährt, da man so verschiedene Ideen ausprobieren konnte, ohne dabei starke Änderungen am bestehenden, funktionsfähigem Code tätigen zu müssen. Das folgende Diagramm gibt nun einen Überblick über die Aufteilung der und soll verdeutlichen, welche Activities/Zustände in unserer Applikation und auch innerhalb der Activities eingenommen werden. Die Übergänge sind keineswegs vollständig, stellen aber ein möglichst gutes Verhältnis aus Korrektheit und Komplexität dar.

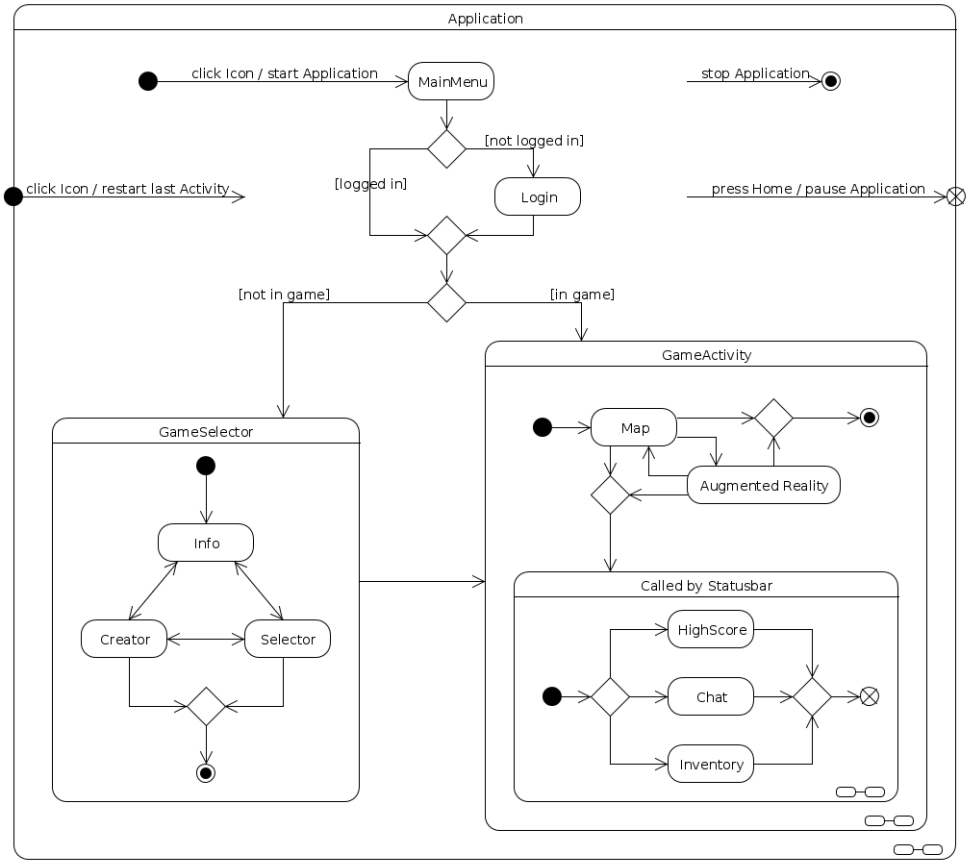


Abb. 38: Das Diagramm zeigt die Struktur der Applikation, welche in die Activities `MainMenu`, `Login`, `GameSelector` und `GameActivity` unterteilt ist

Es gibt zwei Szenarien wie die Applikation gestartet werden kann und folglich auch zwei Startpunkte. Dies liegt darin begründet, dass Applikationen nicht zwangsläufig beendet werden müssen, sondern vom System kontrolliert werden und von diesem bei Speicherproblemen auch freigegeben werden können. Es gibt damit als ersten Einstiegspunkt den Aufruf der Anwendung aus dem Menü heraus durch Anklicken des Icons. In diesem Fall liegt der Startpunkt "innerhalb" der Applikation und kennzeichnet einen echten Neustart. Die erste Activity, die aufgerufen wird ist das `MainMenu`. Von dort aus führen erneut mehrere mögliche Wege weiter. In dem Fall, dass die Anwendung beispielsweise abgestürzt ist, oder aber vom Benutzer gewollt beendet wurde ohne dass dieser sich aus dem Spiel ausgeloggt hat, wird der `Login` übersprungen. Sonst muss der Benutzer sich erst mit dem Server verbinden und einloggen. War er noch nicht eingeloggt muss er sich für eines der Spiele entscheiden, die im `GameSelector` im `Selector` präsentiert werden. Sollte dort kein Spiel dabei sein, welches er spielen möchte, so kann er sich ein neues Spiel im `Creator` anlegen. In dem Fall, dass der Benutzer schon angemeldet war, kann es auch sein, dass er noch in einem Spiel registriert ist. Er wird dann zur `GameActivity` weitergeleitet. Diese Activity bildet das eigentliche Spiel. Es enthält die Karte, als auch die *Augmented Reality*. Über diese Activity werden dann noch die anderen Teilbereiche gelegt, der Chat, das Inventar und die Highscores. Wenn die letztgenannten beendet werden, kehrt die Applikation zur Activity zurück, welche den Aufruf der Ansichten ausgeführt hat.

Nach dem gleichen Prinzip kann auch die Applikation an sich beendet beziehungsweise unterbrochen werden. Wird sie tatsächlich beendet, so wird beim nächsten Aufruf die Applikation neu gestartet, wie es oben beschrieben wurde. Wird sie allerdings unterbrochen, dies kann zum Beispiel durch einen Telefonanruf oder aber das Drücken auf den 'Home'-Button geschehen, wird die zuletzt ausgeführte Activity fortgesetzt, wenn die Unterbrechung beendet ist. Das Fortführen nach einer Unterbrechung stellt somit den zweiten Fall dar, welcher im obigen Abschnitt erwähnt wurde.

4.2 iPhone

Phillipp Bertram

Das *iPhone*[25] gehört zu einer Reihe von Internet- und Multimedia-fähigen Smartphones, entworfen und vermarktet von Apple Inc. Am 29. Juni 2007 wurde es erstmals in den USA veröffentlicht und fand am 09. November 2007 seinen Weg nach Europa. Das iPhone verbindet viele Funktionen in nur einem mobilen Gerät. Das derzeitig aktuelle Modell verfügt über eine Video-/Fotokamera, einen Media-Player, einen GPS-Empfänger, einen Kompass, ein Gyroskop, einen Beschleunigungs-, Licht-, und Näherungssensor, sowie einen Internet Clienten mit E-Mail und Web-Browsing. Hierbei steht dem iPhone das Wi-Fi und 3G als Konnektivität zur Verfügung. Die Benutzeroberfläche besteht aus einem Multi-Touch Bildschirm mit virtueller statt physikalischer Tastatur. Dies ermöglicht also eine Bedienung mit mehreren Fingern gleichzeitig. Anwendungen von Apple und Drittanbietern sind über den *App-Store* verfügbar, in welchem sich seit Januar 2011 rund 350.000 *Apps* befinden.

4.2.1 Generationen

Phillipp Bertram

Es gibt mittlerweile insgesamt vier Generationen von iPhone-Modellen, begleitet von vier Major-Releases der Plattform ***iOS*** (ehem. iPhone OS), die im folgenden vorgestellt werden.

	iPhone	**iPhone 3G**	**iPhone 3G S**	**iPhone 4**
Release	29. Juni 2007	11. Juli 2008	19. Juni 2009	24. Juni 2010
Plattform	iPhone OS 1.0	iPhone OS 2.0	iPhone OS 3.0	iOS 4.0
Display	480 x 320 px 163 ppi	480 x 320 px 163 ppi	480 x 320 px 163 ppi	960 x 640 px 326 ppi
Speicher	4, 8, 16 GB	8, 16 GB	8, 16, 32 GB	16, 32 GB
Prozessor	620 MHz	620 MHz	833 MHz	1 GHz
Arbeitsspeicher	128 MB DRAM	128 MB DRAM	256 MB DRAM	512 MB DRAM
Besonderheiten	• *Näherungssensor*: bei dem Gerät wird automatisch die Eingabefunktion sowie die Bildschirmbeleuchtung ausgeschaltet, wenn es ans Ohr gehalten wird. • *Beschleunigungssensor*: Die Anzeige wird ferner automatisch umgestellt, wenn das Gerät vertikal oder horizontal gehalten wird. • *Helligkeitssensor*: Die Bildschirmhelligkeit kann an die Lichtverhältnisse der Umgebung angepasst werden, wodurch sich die Akkulaufzeit deutlich erhöht.	• Es unterstützt zusätzlich zu EDGE die Mobilfunkstandards UMTS/HSDPA. • Mit dem Kompass und via A-GPS ist eine Standortbestimmung möglich. • Kamera mit höherer Auflösung.	• Es unterscheidet sich zu seinem Vorgänger hauptsächlich um die beschleunigte Arbeitsgeschwindigkeit und die Internetverbingung. • Auflösung Kamera von 2 auf 3-Megapixel erhöht • eine 32-Gigabyte-Version anstelle zuvor maximal 16-Gigabyte • längere Akkulaufzeiten • bessere 3D-Grafik durch Unterstüzuung von OpenGL-ES 2.0-Standards • digitaler Kompass (Magnetometer).	• Neugestaltung des Gehäuses mit EdelstahlRahmen, der als Antenne fungiert. • Kamera mit 5 Megapixel (HD-Aufnahmen möglich) • Front VGA-Kamera, die bei Videochats (FaceTime) zum Einsatz kommt. • Ein 3D-Bewegungssensor (Gyroskop) sollte vor allem bei Spielen neue Funktionen ermöglichen.

25 http://www.apple.com/de/iphone/

iPhone - 1. Generation

iPhone 3G - 2. Generation

iPhone 3GS - 3. Generation

iPhone 4 - 4. Generation

4.2.2 Plattform Grenzen

Phillipp Bertram

Wenn man über mobile Plattformen wie das iPhone spricht, ergeben sich immer eine Reihe von Einschränkungen wie Speicher, Interaktionsgrenzen, Akkulaufzeit und Performance. Mobile Plattformen können nicht den gleichen Speicher bieten, die ihre Desktop-Pendants haben. Für das iPhone kann kein großer Bildschirm mit Maus und Tastatur entworfen werden. Stattdessen muss die Entwicklung und das Design den Gegebenheiten der Plattform angepasst werden.

Speicher

Das iPhone beherbergt eine leistungsstarke und dennoch kompakte OS X Installation. Obwohl das gesamte iOS nicht mehr als ein paar hundert MB einnimmt - fast nichts im Vergleich heutiger großer Betriebssystem-Installationen -, bietet es eine umfangreiche Framework-Libary. Diese Frameworks von vorkompilierten Routinen ermöglichen dem iPhone Benutzer ein breites Spektrum an kompakten Applikationen zu verwenden, von telefonieren bis hin zu Internet surfen und Musik hören. Das iPhone bietet gerade genug Unterstützung bei der Programmierung, um flexible Schnittstellen zu erstellen und dabei die Systemdateien so zu trimmen, dass sie gut in den wenig vorhandenen Speicher passen.

Daten Zugriff

Jede iPhone Applikation ist *sandboxed*. Das bedeutet, dass sie in streng regulierten Bereichen des Dateisystems "leben". Applikationen können nicht von anderen Applikationen angesteuert werden. Es kann jedoch auf alle Daten zugegriffen werden, die über das Internet zur Verfügung stehen, sofern das iPhone mit einem Netzwerk verbunden ist.

Arbeitsspeicher

Die Speicherverwaltung ist auf dem iPhone sehr kritisch. Es unterstützt keinen Disk-Swap-basierten virtuellen Speicher; daher startet das iPhone neu, sobald man einen Speicherüberlauf hat. Ohne Auslagerungsdatei muss der Speicherbedarf sorgfältig verwaltet und ein eventueller Speicherüberlauf behandelt werden. Außerdem muss man darauf achten welche Ressourcen die Applikation verwendet. Zu hoch auflösende Bilder oder Audiodateien können die Applikation in eine *"autoterminate"* Zone bringen, sodass diese automatisch beendet wird.

Interaktion

Das Verschwinden von physikalischen Eingabegeräten und das Arbeiten mit einem kleinen Bildschirm bedeutet nicht automatisch, dass auch die Interaktions-Flexibilität geringer wird. Benutzeroberflächen können durch die Multi-Touch-Fähigkeit komplett neu designed werden, ohne sich an konventionelle Regeln halten zu müssen.

Akku Laufzeit

Bei mobilen Plattformen können die Akkulaufzeiten nicht ignoriert werden. Wie bereits erwähnt, helfen die Apple's SDK Features, die Applikation den Ansprüchen genügend zu designen.

Applikationen

Apple hat eine strenge 'One-Application-At-A-Time' Politik eingeführt. Das bedeutet, dass Programmierer keine Applikationen entwickeln können, die im Hintergrund laufen. Jedes Mal, wenn das Programm läuft, muss es aufräumen, bevor der User eine weitere Applikation startet. Es kann kein Daemon im Hintergrund laufen gelassen werden, der nach neuen Meldungen sucht oder periodische Updates sendet. Andererseits unterstützt Apple *Push-Notifications* von Web-Services. Registrierte Dienste können den User benachrichtigen, dass Daten abrufbereit auf diesen Servern liegen.

4.2.3 Programmieren auf dem iPhone

Phillipp Bertram

Voraussetzungen

- **Eine Kopie des Apple iPhone SDK**. Diese kann im *Apple's iPhone Dev Center*[26] kostenlos heruntergeladen werden, sofern man dem Apple Developer Programm beigetreten ist.
- **Ein iPhone/iPod-Touch/iPad**. Obwohl Apple einen Simulator als Teil des SDK's mitliefert, wird ein konkretes Gerät benötigt, auf dem die Software getestet werden kann.
- **Eine Apple iPhone Developer Lizenz**. Erst wenn man dem Developer Programm beigetreten ist, kann die programmierte Software auf dem iPhone getestet werden. Mitglieder erhalten ein Zertifikat, dass ihnen erlaubt ihre Applikationen auf die Endgeräte zu installieren um sie zu testen. Das Programm kostet 99$ im Jahr für Einzelpersonen und Unternehmen.
- **Ein Intel-basierter Macintosh mit Leopard**. Das SDK erfordert einen Macintosh mit Leopard OS X 10.5.3 oder höher als Betriebssystem.
- **Wenigstens einen verfügbaren USB 2.0 Anschluss**. Die eigenen Applikationen werden über einen USB 2.0 Anschluss auf das Gerät installiert.
- **Internetverbindung**. Mit einer Internetverbindung können Anwendungen mit einer *WiFi* oder *EDGE*[27] Verdingung getestet werden.
- **Kenntnisse in Objective-C**. Die Sprache ist eine Obermenge von C mit objektorientierten Erweiterungen und gehört zur Standardsprache von Apple.

26 http://developer.apple.com/iphone
27 http://de.wikipedia.org/wiki/Enhanced_Data_Rates_for_GSM_Evolution

Objective-C

Objective-C (auch *ObjC*) ist eine reflexive, objektorientierte Programmiersprache, die die Programmiersprache *C* um *Smalltalk* ähnliches *Messaging* erweitert. Sie bildet eine Obermenge von *C*, es kann also jedes in *C* geschriebenes Programm mit einem ObjC-Compiler kompiliert werden. Heute wird es in erster Linie auf Apple's Mac OS X und iOS verwendet: beide Umgebungen basieren auf dem *OpenStep*-Standard, sind mit ihm allerdings nicht konform. *ObjC* ist die primäre Sprache der Apple *Cocoa*-API und war ursprünglich die Hauptsprache auf *NeXTStep OS*[28].

Hello Objective-C

Die folgenden Beispiele sollen einen kleinen Einblick in die Programmiersprache *Objective-C* geben. Gestartet wird mit einer klassischen *Hello-Objective-C*-Anwendung.

```
#import <Foundation/Foundation.h>

int main (int argc, const char *argv[])
{
   NSLog(@"Hello, Objective-C");
   return(0);
}
```

HelloWorld Beispiel

Die *Foundation.h* ist eine Libary, die wesentliche Funktionen enthält und wird daher in den allermeisten Fällen importiert. `NSLog` gibt einen String auf die Konsole aus, der in diesem Fall 'Hello, Objective-C' lautet. Wird die *Main*-Methode mit 0 beendet, bedeutet dies nur, dass das Programm erfolgreich durchlaufen wurde.

Objekt Zugriffe

Anders als in Java werden die Methoden der Objekte nicht über eine Punkt-Notation angesprochen, sondern als *message* an das Objekt geschickt. Folgender Java-Code

```
anObject.methodeOhneParameter();
anObject.methodeMitParameter(5, 10);
```

sieht in *ObjC* also entsprechend so aus:

```
[anObject methodeOhneParameter];
[anObject methodeMitParameter:5 undWeitererParameter:10];
```

Speicherverwaltung

Wie bereits erwähnt, muss man sich bei der Programmierung auf dem iPhone selber um die Speicherverwaltung kümmern, da es - anders als in Java - keinen *Garbage Collector* gibt. Daher müssen alle allokierten Objekte auch wieder freigegeben werden. Eine Faustregel besagt, dass zu jedem `alloc`, ein `release` folgen muss, wie in dem folgenden Beispiel gezeigt ist.

28 *http://www.operating-system.org/betriebssystem/_german/bs-nextstep.htm*

```
String *myString = [[NSString alloc] initWithFormat:@"Ich bin ein String"];

// ... ganz viel Code ...

[myString release];
```

Objekte können in *ObjC* über `autorelease` automatisch wieder freigegeben werden. Allerdings sollte das nur in wenigen Fällen angewendet werden, da es hier leicht zu Fehlern kommen kann. Der folgende Code zeigt, wie `autorelease` eingesetzt wird.

```
String *myString = [[NSString alloc] initWithFormat:@"Ich bin ein String"];
[myString autorelease];

//alternativ in einer Zeile
String *another String = [[[NSString alloc] initWithFormat:@"Ich bin ein weiterer String"]
autorelease];
```

Strings in *Objective-C* müssen immer mit einem @ versehen werden. So wird zwischen *ObjC*- und *C*-Strings unterschieden, bei denen wie gewohnt das Zeichen entfällt.

```
NSLog(@"Objc String");
printf("C String");
```

Ein weiterer Unterschied zu *C* oder anderen Programmiersprachen ist der Datentyp *Boolean*. Hier kann dieser Datentyp nicht die Werte `true` oder `false` annehmen, sondern `YES` oder `NO`. Sonst ist das Verhalten in etwa gleich.

Properties und Aufbau einer Klasse

Eine noch zu erwähnende Eigenschaft von *ObjC* sind die sogenannten *Properties*. Guter Programmierstil ist, wenn Attribute nicht direkt, sondern über *Accessoren* modifiziert und abgerufen werden. Um nicht für jedes Attribut einen solchen *Getter* und *Setter* schreiben zu müssen, gibt es die *Properties*.

```
// EineKlasse.h
@interface EineKlasse:NSObject
{
   String *aString;
}

@property(retain) String *aString;

-(void) methodeOhneParameter;
-(void) methodMitParameterA:(String*) str andParameterB:(String*)
anotherString;
-(String*) methodeMitRueckgabewert;

@end

// EineKlasse.m
#import EineKlasse.h

@implementation EineKlasse

@synthesize aString;

-(void) methodeOhneParameter
{
   // mach was
}
```

```
-(void) methodMitParameterA:(String*) str andParameterB:(String*)
anotherString
{
  // mach was mit den Parametern
}

-(String*) methodeMitRueckgabewert
{
   return @"Hallo Welt";
}

-(void) dealloc
{
   [aString release];
   [super dealloc];
}

@end
```

Aufbau von Klassen und Methoden

Das Beispiel zeigt, wie in etwa eine Pseudo-Klasse aufgebaut ist. Eine Klasse besteht immer aus einem *Interface* (`EineKlasse.h`) und einer *Implementation* (`EineKlasse.m`). Für die meisten Anwendung, die für das iPhone programmiert werden, ist es sinnvoll, die Klasse von `NSObject` erben zu lassen. `NSObject` ist wie `Object` in Java eine Basisklasse, von der alle Klassen abstammen sollten, die iPhone spezifisch sind. Sie enthält die essenziellen Methoden zur Speicherverwaltung (wie `dealloc`) und noch weitere Eigenschaften.

Attribute werden in der *Header*-Datei innerhalb der geschweiften Klammern deklariert. Danach besteht die Möglichkeit zum einen *Properties* anzulegen und zum anderen Methoden-Signaturen aufzuführen, wie es schon aus *C* bekannt ist. *Properties* werden mit dem Schlüsselwort `@property` eingeleitet, gefolgt von wenigen Einstellungsmöglichkeiten innerhalb der Klammer (z.B. `retain`). Anschließend muss der Datentyp und der Name des Attributs aufgeführt werden, der der *Property* zugewiesen wird. Eine weitere Eigenschaft der *Properties* ist, dass diese Attribute - wie in Java - auch per Punkt-Notation angesprochen werden können.

Die Implementierung geschieht wie in *C* in der `.m` Datei. Hier müssen die Properties noch *synthetisiert* werden. Das bedeutet nur, dass dem Compiler mit dem Schlüsselwort `@synthesize` kenntlich gemacht wird, dass er die *Getter*- und *Setter*-Methoden beim Compilieren einfügen soll.

Die Methode `dealloc` wird immer dann aufgerufen, wenn ein Objekt dieser Klasse zerstört wird. Hier sollte der gesamte Speicher freigegeben werden, den das Objekt allokiert hat.

Apple's iPhone SDK

Apple entwickelte für das iPhone eine mobile Variante seines *Cocoa-Frameworks* namens *Cocoa Touch*. Das SDK wird zusammen mit einer neuen Version von Apple's integrierter Entwicklungsumgebung *Xcode*[29] ausgeliefert. Darin enthalten ist auch ein iPhone-Simulator, der es weitestgehend ermöglicht, die Anwendungen während der Entwicklungsphase auf dem Mac zu testen. Der Vertrieb der Programme erfolgt über den *App Store*[30] oder über iTunes. Die Entwickler können den Preis für ihre Software selbst festlegen, Apple nimmt jedoch 30 Prozent davon als Provision. Während das SDK selbst kostenlos von Apple's Entwicklerseiten bezogen werden kann, ist für die Veröffentlichung im App Store ein kostenpflichtiges Entwicklerkonto zum Preis von 99 $ (Standard) oder 299 $ (unternehmensinterne Anwendungen) pro Jahr erforderlich.

- **XCode** ist das wichtigste Werkzeug im iPhone Development Arsenal. Es bietet eine umfassende Projektentwicklung und Management-Umgebung, mit umfassender Dokumentation, einem Editor und einem graphischen Debugger.
- **Instruments** zeigt, wie iPhone Applikationen im Hintergrund ablaufen. Hiermit können Probleme gefunden und die Performance optimiert werden. Instruments bietet grafische, zeit-basierte Plots die zeigen, wo die Applikation die meisten Ressourcen verbraucht.

29 http://developer.apple.com/technologies/tools/whats-new.html
30 http://www.apple.com/de/iphone/apps-for-iphone/

- Der **Simulator** läuft auf dem Macintosh und ermöglicht das Testen der Applikationen auf dem Desktop. Dazu muss kein iPhone angeschlossen sein. Der Simulator bietet die gleiche API, die auch auf dem iPhone verwendet wird.
- Der **Interface Builder** bietet ein Rapid-Prototyping-Werkzeug mit dem man die Benutzeroberflächen designen und über vordefinierte Schnittstellen im Quellcode diese Elemente verlinken kann. Man erstellt also seine Oberflächen mit visuellen Design-Werkzeugen und verlinkt diese Bildschirmelemente mit Objekten und Methodenaufrufe aus der Applikation. Da dieser allerdings im Rahmen dieser Projektarbeit nicht benutzt wurde, wird an dieser Stelle auch nicht weiter auf ihn eingegangen.

Application Life Cycle

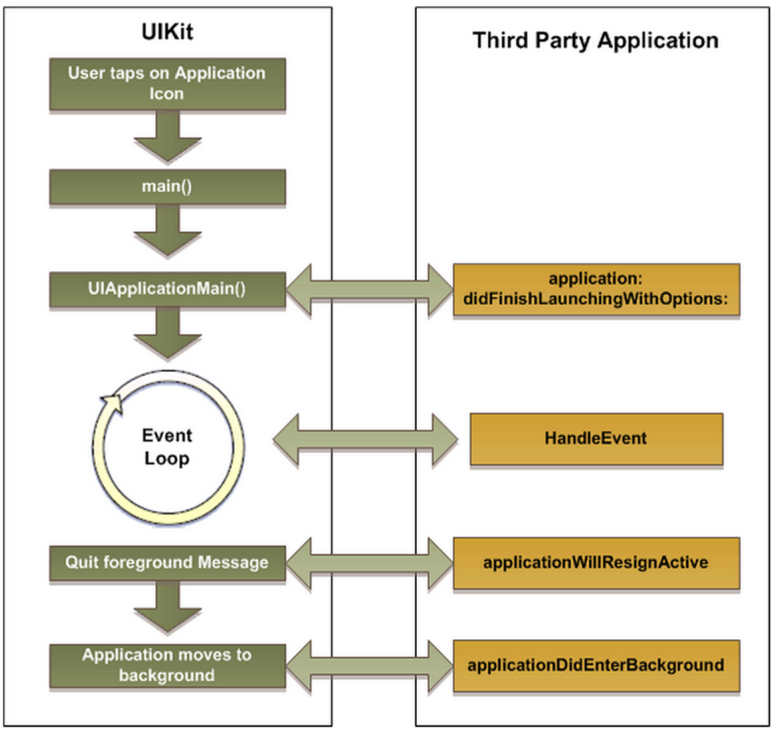


Abb. 39: iPhone Application LifeCycle. Quelle: http://www.codeproject.com/KB/iPhone/ApplicationLifeCycle.aspx

Komponenten der Applikation

Compilierte iPhone Applikationen "leben" in sogenannten *Application-Bundles*. Dies sind nichts weiter als Ordner mit einer `.app` Endung, die eben alle Ressourcen eines Programms enthalten. Die Hierarchie dieser *Bundles* ist einfach gestrikt. Alle *Materialien* befinden sich in der obersten Ebene des Ordners. Es können auch Unterordner erzeugt werden, die das Projekt besser strukturieren und organisieren, allerdings folgen diesen eigenen Unterordnern keine Standards. Das *iPhone SDK* unterstützt hierfür die `NSBundle` Klasse. Diese Klasse stellt unter anderem Methoden bereit, mit denen man auf die Dateien in dem Bundle zugreifen kann.

Die `Executable` Applikation Datei befindet sich ebenfalls in der obersten Ebene des Bundles. Sie enthält Ausführungsrechte, sodass eine Applikation nur dann geladen und gestartet werden kann, wenn es mit einem offiziellen Entwickler Zertifikat ausgewiesen wurde.

Wichtiger Bestandteil einer Applikation ist die *Property-List* (`Info.plist`). Sie beinhaltet Key-Value Daten für viele unterschiedliche Zwecke und kann diese entweder in einem lesbaren, textbasierten oder komprimierten, binären Format abspeichern. Hier wird z.B. spezifiziert, welches die auszuführende Datei ist oder wie der Name des Bundles ist.

Grundgerüst einer iPhone Applikation

Fast alle iPhone Applikationen enthalten ein paar Schlüssel-Dateien, die in der unteren Abbildung aufgeführt sind - eine `Main`, eine `ApplicationDelegate` und eine `ViewVontroller` Komponente. Diese Dateien stellen alles nötige bereit, um eine einfache Anwendung zu erstellen.

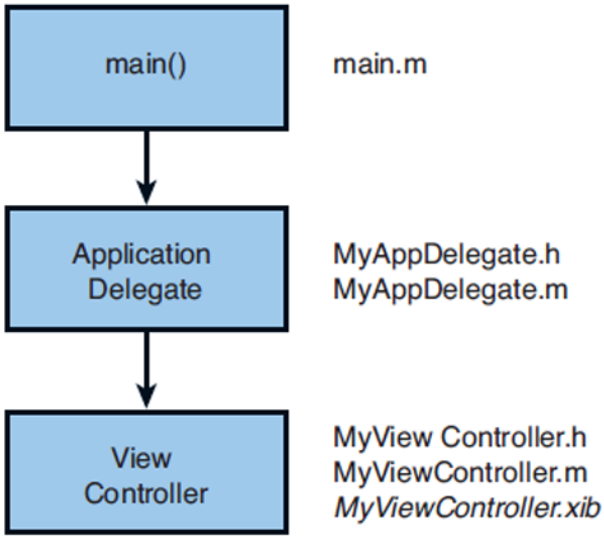


Abb. 40: Grundgerüst einer iPhone Applikation. Zu finden in *'The iPhone Developer's CookBook', S.18*

Die `main.m` hat zwei Aufgaben. Erst erstellt es einen `Autorelease-Pool` für die Applikation. Dieser Pool sammelt alle Objekte, die mit `autorelease` gekennzeichnet wurden und gibt den Speicher am Ende wieder frei. Danach wird der *Application Event Loop* aufgerufen, der unter anderem den Namen der `ApplicationDelegate` als Parameter übergeben bekommt.

```
int main(int argc, char *argv[])
{
  NSAutoreleasePool * pool = [[NSAutoreleasePool alloc] init];
  int retVal = UIApplicationMain(argc, argv, nil, @"MyAppDelegate");
  [pool release];
  return retVal;
}
```

Die `UIApplicationMain` Funktion ist eine Art Einstiegspunkt des Programms. Mehr muss man nicht über die `main.m` wissen. Diese wird automatisch von dem SDK generiert und es müssen in der Regel keine Änderungen an der Datei mehr vorgenommen werden.

Ein `ApplicationDelegate` implementiert, wie das Programm auf kritische Punkte im *Application Life Cycle* reagieren soll. Sie ist für das Initialisieren des *Window-Systems* beim Start verantwortlich und behandelt unter anderem Speicherwarnungen. Die wichtigsten Delegate-Methoden sind hier aufgeführt:

- **applicationDidFinishLaunching:** Diese Methode wird als erstes aufgerufen, nachdem die Applikation initiiert wurde. Nach dem Start ist hier der Bereich, in dem das Basis Fenster (`window`) erstellt, dessen Inhalt gesetzt und sichtbar gemacht wird.
- **applicationWillTerminate:** Diese Methode wird immer dann aufgerufen, wenn die Applikation beendet wird. In der Regel verwendet man diese Methode um z.B. Daten zu speichern, upzudaten oder auch Dateien zu schließen.
- **applicationDidRecieveMemoryWarning:** Wird diese Methode aufgerufen, muss die Applikation so viel Speicher wie möglich freigeben. Falls kein Speicher freigegeben wird oder werden kann, wird das Programm unerwartet beendet.

Im iPhone *Programmier-Paradigma* stellen die `ViewController` das Herzstück der Anwendungen dar. Hier wird normalerweise das Verhalten der Applikation implementiert, wie sie auf Userinteraktionen reagieren soll. Wenn kein *Interface Builder* verwendet wurde, werden auch hier die `Views` geladen und aufgebaut.

Beispielanwendung

Mit diesem Beispiel soll gezeigt werden, wie man eine kleine iPhone Applikation erstellt. Es wird kein *Interface Builder* verwendet, sondern die *Views* werden innerhalb eines `ViewControllers` manuell zusammengebaut und geladen. Das Programm besteht hierbei lediglich aus einem *Button*, der beim Drücken den Inhalt eines *Labels* ändert.

Mit *XCode* wird ein neues *Window-based-Applikation*-Projekt namens *'HelloWorld'* für iPhone OS angelegt. Im Ordner *Classes* sollten sich nun zwei Dateien mit der Beizeichnung `HelloWorldAppDelegate` befinden. Da kein *Interface Builder* verwendet wird, kann die '`HelloWorldViewController.xib`' gelöscht und die `main.m` wie unten aufgeführt geändert werden. Als nächstest wird ein *ViewController* namens `HelloWorldViewController` eingefügt. Dieser enthält später den Button, das Label und die nötigen Methoden. Die Beispielanwendung hat letztendlich folgendes Aussehen:

```
int main(int argc, char *argv[])
{
   NSAutoreleasePool * pool = [[NSAutoreleasePool alloc] init];
   int retVal = UIApplicationMain(argc, argv, nil,@"HelloWorldAppDelegate");
   [pool release];
   return retVal;
}
```

`main.m`

```
#import <UIKit/UIKit.h>

@interface HelloWorldViewController : UIViewController
{
        UIView   *contentView;
        UILabel  *label;
        UIButton *button;
}

- (void)buttonPressed;

@end
```

`HelloWorldViewController.h`

```
#import "HelloWorldViewController.h"

@implementation HelloWorldViewController

// Implement loadView to create a view hierarchy programmatically, without using a nib.
- (void)loadView
{
   // Init ContentView
   contentView = [[UIView alloc] initWithFrame:[[UIScreen mainScreen] bounds]];
   contentView.backgroundColor = [UIColor lightGrayColor];

   // Init Label
   label = [[UILabel alloc] initWithFrame:CGRectMake(50.0f, 250.0f, 200.0f, 30.0f)];
   label.text = @"Hello World";
   label.textAlignment = UITextAlignmentCenter;
   label.backgroundColor = [UIColor clearColor];

   // Init Button
```

```
   button = [UIButton buttonWithType:UIButtonTypeRoundedRect]; button.frame = CGRectMake(100.0,
 180.0, 100.0, 30.0);
   button.titleLabel.font = [UIFont fontWithName:@"Helvetica" size:15.000];
   [button setTitle:@"Press Me" forState:UIControlStateNormal];
   [button addTarget:self
           action:@selector(buttonPressed)
           forControlEvents:UIControlEventTouchUpInside];

   // Add label and button to contentview
   [contentView addSubview:label]; [contentView addSubview:button];

   // set the view
   self.view = contentView;
}

- (void)buttonPressed
{
   label.text = @"Button Pressed!";
}

- (void)dealloc
{
  [contentView release];
  [label release];
  [super dealloc];
}

@end
```

HelloWorldViewController.m

```
#import <UIKit/UIKit.h>

@class HelloWorldViewController;

@interface HelloWorldAppDelegate : NSObject <UIApplicationDelegate> {
   UIWindow                 *window;
   HelloWorldViewController *hwvc;
}

@property (retain) UIWindow *window;

@end
```

HelloWorldAppDelegate.h

```
#import "HelloWorldAppDelegate.h"
#import "HelloWorldViewController.h"

@implementation HelloWorldAppDelegate
@synthesize window;

- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(NSDictionary
*)launchOptions {

   window = [[UIWindow alloc] initWithFrame:[[UIScreen mainScreen] bounds]];
   hwvc = [[HelloWorldViewController alloc] init];

   [window addSubview:hwvc.view];
   [window makeKeyAndVisible];

    return YES;
}

- (void)dealloc {
   [window release];
   [hwvc release];
   [super dealloc];
```

```
}

@end
```

HelloWorldAppDelegate.m

Im `HelloWorldAppDelegate` wird zunächst ein `UIWindow` generiert. `UIWindow` ist gleichbedeutend mit `UIView` mit dem Unterschied, dass `UIWindow` nur ein einziges mal existiert und die unterste Ebene der View Hierarchie darstellt. Alle weiteren Views werden quasi auf das *window* gelegt, wie auch in diesem Fall die View des `HelloWorldViewController`'s mit der Zeile `[window addSubView:hwcv.view]`. Schließlich wird das *window* noch sichtbar gemacht.

Das Herz der Anwendung steckt jedoch in dem `HelloWorldViewController`. Im Interface sind lediglich die drei GUI-Elemente und eine Methode namens `buttonPressed` aufgeführt. Beim Laden der *View* des `HelloWorldViewController`'s wird die `loadView` Methode aufgerufen und daher befindet sich hier auch die Initialisierung der Objekte. Zunächst wird eine *content-View* erstellt, die alle grafischen Objekte aufnehmen soll. Die Hintergrundfarbe wird auf hellgrau gesetzt. Das Label erhält Größe, Position und einen initialen Text. Der Button erhält ebenfalls eine Größe, eine Position, einen Text und ist vom Typ `RoundedRect`. Über `addTarget` wird eine Methode registriert, die aufgerufen wird, sobald der Benutzer den Button drückt und wieder loslässt `UIControlEventTouchUpInside`. Diese Methode macht nichts weiter, als den Text des Labels in ' *Button Pressed!* ' abzuändern. Schließlich werden alle allokierten Objekte in der Methode `dealloc` wieder freigegeben.

4.2.4 Vektorgrafik

Sergiy Krutykov

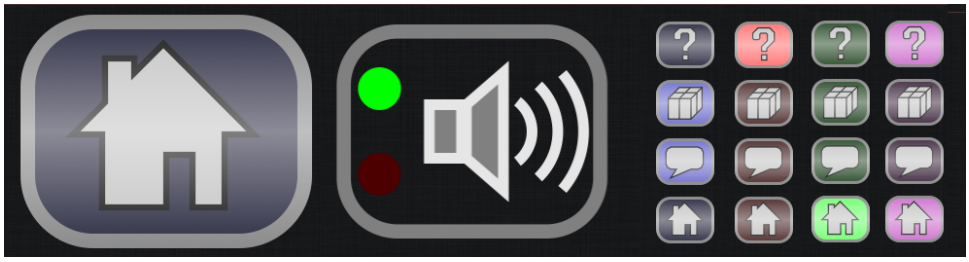
iPhone bietet eine gewisse Auswahl an Steuerelementen an, die Entwickler in ihren Programmen benutzen können. Jedoch reicht diese Auswahl für eine gemäße Gestaltung der Applikation häufig nicht aus. Bei manchen Applikationen, wie zum Beispiel bei Spielen, die in erster Linie originelles Aussehen aufweisen sollen, sind solche Standard-GUI-Elemente fast schon tabu, so dass man sowieso gezwungen ist, eigene Elemente zu implementieren. Bei manchen Elementen, wie zum Beispiel Buttons besteht noch die Möglichkeit eigene Skins in Form von Pixelgrafiken auf ihnen zu zeichnen, aber das bringt sehr unschöne Pixeleffekte mit sich, wenn man die Größe dieser Buttons ändert. Insbesondere bei den Applikationen, die sowohl auf iPhone 4 als auch auf früheren Modellen gleich aussehen sollen, sind solche Pixeleffekte wegen unterschiedlicher Auflösungen vorprogrammiert. Außerdem wird manchmal ein Element mit ganz besonderem Verhalten gewünscht, welches kein Standard-Element hat.

Alle diese Problemen kann man umgehen, wenn man die Vektorgrafik zu Darstellung von Steuerelementen benutzt, d.h., wenn die Elemente in dem Programm selbst "on the fly" gezeichnet werden. Dazu steht den Entwicklern für iPhone das Framework "Core Graphics" zu Verfügung, welches das Zeichnen von einfachen Geometrien (Linien, Rechtecke, Kreise, Polygone, etc.) mit Unterstützung von Farbeffekten und Transformationen ermöglicht. Die folgende Abbildung zeigt Fragment einer Statusbar, die komplett mit Mitteln der Vektorgrafik gezeichnet wird:

Hier werden zwei Arten von Steuerelementen benutzt, die in der Standardauswahl von iPhone so nicht zu finden sind: Toggle-Buttons (links im Bild) und Switches (rechts im Bild). Man sieht, dass der zweite Toggle-Button von links aktiviert ist, während die restlichen inaktiv sind und dass der rechte Switch (mit Sound-Symbol) ausgeschaltet ist. Darüber hinaus erscheinen rote Kreuze über den beiden linken Switches, um einen Fehler zu symbolisieren. Der Gradient im Hintergrund so wie leichtes Leuchteffekt auf den Toggle-Buttons sind im Übrigen auch Elemente der Vektorgrafik.

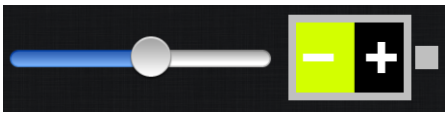
Solche grafischen Elemente sind natürlich beliebig verlustfrei skalierbar und man kann außerdem die Farben beliebig ändern und zwar zur Laufzeit: Wenn man zum Beispiel in einem Spiel für unterschiedliche Parteien spielen kann, dann kann man das Interface der Applikation, je nach dem auf wessen Seite der Benutzer spielt,

farbig anpassen. In der folgenden Abbildung sind ein Toggle-Button und ein Switch vergrößert und eine Auswahl von Toggle-Buttons in vier verschiedenen Farben dargestellt:



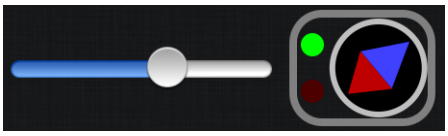
Offensichtlich müsste man ohne die Vektorgrafik in den oberen Beispielen eine große Anzahl an Pixelgrafiken benutzen. Besonderes bei animierten Elementen erweist sich die Vektorgrafik als sehr vorteilhaft. In der folgenden Animation ist ein Switch animiert dargestellt, um zu zeigen, dass der mit ihm assoziierte Dienst (in diesem Fall GPS) aktiv ist:

Außer reinen Animationen, die einfach nur zeigen, ob etwas zur Zeit aktiv ist, besteht die Möglichkeit, mit Hilfe von der Vektorgrafik wichtige Informationen zu visualisieren. Zum Beispiel sieht man in der folgenden Animation, wie feingranular angezeigt werden kann, wie viel Prozent der Batterie verbraucht ist:

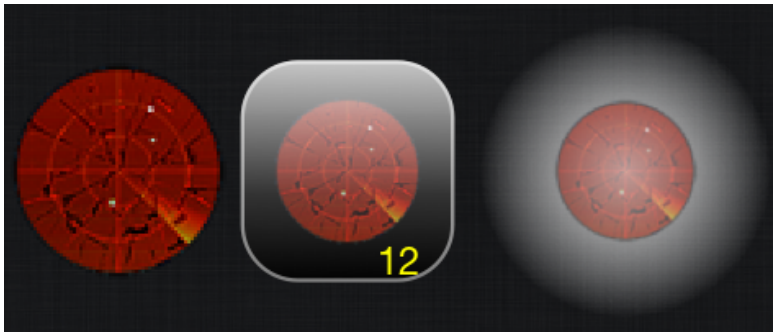


Mit Pixelgrafik würde man hier lediglich bestimmte Zustände anzeigen, wie z.B. 0%, 25%, 50%, 75%, 100% Akku-Leistung, wobei diese fünf Bilder irgendwo im Speicher abgelegt werden müssten und dann entsprechend ausgetauscht.

Je nach Fantasie kann man beliebige Indikatoren ausdenken, die wichtige Informationen veranschaulichen können. In der folgendem Video ist ein Indikator für die Ungenauigkeit von Kompass zu sehen, in welchem sich die Kompassnadel umso weiter von der vertikalen Ausrichtung wegdreht, je ungenauer der Kompass ist, wobei die Ungenauigkeit des Kompasses, die in Grad gemessen wird, dieser Drehung entspricht:



Die Vektorgrafik erlaubt auch die Pixelgrafiken zu zeichnen und insbesondere Pixelgrafiken mit Vektorgrafik zu kombinieren. In der folgenden Abbildung sieht man links eine Bitmap, die in der Mitte in einen mit Mitteln der Vektorgrafik erzeugten Button integriert ist, und rechts in eine rein "vektorgrafische" Leuchtblase:



Mit der Vektorgrafik kann man nicht nur neue Elemente implementieren, sondern auch die bestehenden ersetzen, wenn sie dem Konzept der Applikation nicht entsprechen. In der folgenden Abbildung ist dem Standard

Switch (Analogon zu Checkbox) von iPhone (links), der ziemlich unbeholfen wirkt, ein alternativer mit der Vektorgrafik gemachte (rechts) gegenübergestellt:

Programmierbeispiel

Anhand von der Checkbox von oben soll im Folgenden erklärt werden, wie man eigene Steuerelemente auf iPhone implementieren kann.

Man braucht dafür eine Klasse, die von `UIView` oder einer ihrer Unterklassen erbt. Der Zustand der Checkbox (checked/unchecked) wird in der booleschen Instanzvariable `checked` gespeichert.

```
@interface CheckBox : UIView
{
    BOOL checked;
}

@property (readonly) BOOL checked;

@end
```

In der Initialisierungsmethode `initWithFrame:` wird zuerst festgelegt (mit der Variable `singleTapGestureRecognizer`), dass bei einmaligem Tippen auf der View die Instanzmethode `hasBeenSingleTapped:` aufgerufen wird:

```
@implementation CheckBox

@synthesize checked;

- (id) initWithFrame:(CGRect)rect
{
    self = [super initWithFrame:rect];  if(!self) return self;

    checked = YES;

    UITapGestureRecognizer *singleTapGestureRecognizer
        = [[UITapGestureRecognizer alloc] initWithTarget:self
                                                  action:@selector(hasBeenSingleTapped:)];
    [singleTapGestureRecognizer setNumberOfTapsRequired: 1];
    [self addGestureRecognizer:singleTapGestureRecognizer];
    [singleTapGestureRecognizer release];

    return self;
}


- (void) hasBeenSingleTapped:(UITapGestureRecognizer *)gestureRecognizer
{
    checked = !checked;
    [self setNeedsDisplay];
}


- (void)drawRect:(CGRect)rect
{
    CGContextRef context = UIGraphicsGetCurrentContext();

    // white square
    CGContextSetRGBFillColor(context, 0.9f, 0.9f, 0.9f, 1.0f);
    CGContextFillRect(context, rect);
```

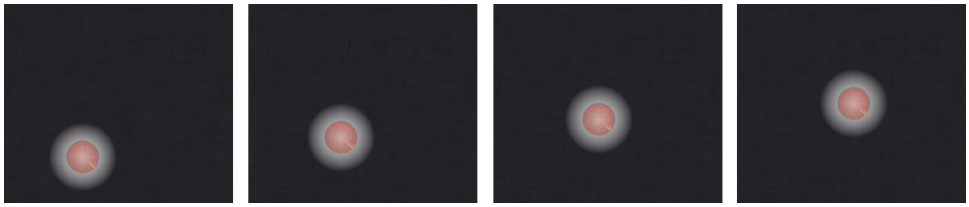
```
    // grey border
    CGContextSetLineWidth(context, rect.size.width/8.0f);
    CGContextSetRGBStrokeColor(context, 0.3f, 0.3f, 0.3f, 1.0);
    CGContextStrokeRect(context, rect);

    // checkmark
    if(checked){
        CGContextSetRGBStrokeColor(context, 0.0f, 0.0f, 0.0f, 1.0);
        CGPoint addLines[] = {
            CGPointMake(rect.size.width*0.2f,  rect.size.height*0.4f),
            CGPointMake(rect.size.width*0.45f, rect.size.height*0.7f),
            CGPointMake(rect.size.width*0.8f,  rect.size.height*0.2f),
        };
        CGContextAddLines(context, addLines, 3);
        CGContextStrokePath(context);
    }
}

@end
```

Das Zeichnen geschieht in der Methode `drawRect:`. Diese Methode wird automatisch aufgerufen, zuerst beim Erzeugen der View und sonst jedes Mal, sobald die View mit dem Aufruf der Methode `setNeedsDisplay` aufgefordert wird, sich zu erneuern, und bekommt als Parameter die Position und die Größe der View in Bildschirmkoordinaten. In dieser Methode muss man nur den graphischen Kontext `UIGraphicsGetCurrentContext()` beziehen und ihn zum Zeichnen benutzen. Mit Hilfe von speziellen Befehlen oder Kombinationen von Befehlen kann man viele primitive Objekte zeichnen: Linien, Rechtecke, Kreise, etc. Der Befehl `CGContextFillRect(context, rect);` in dem Beispiel zeichnet ein gefülltes Rechteck an der Position und in der Größe gemäß der Variable `rect`. Da diese Variable `rect` gerade der Position und Größe der ganzen View entspricht, füllt das Rechteck die ganze View aus. Wegen des Befehls `CGContextSetRGBFillColor(context, 0.9f, 0.9f, 0.9f, 1.0f);` ist die Füllfarbe des Rechtecks (R,G,B,A = 0.9, 0.9, 0.9, 1.0), also etwas abgedunkeltes weiß. Damit der Rahmen und das Häkchen bei unterschiedlichen Skalierungsfaktoren mit passender Linienbreite gezeichnet werden, muss diese Linienbreite in Abhängigkeit von der Größe der View gesetzt werden. `CGContextSetLineWidth(context, rect.size.width/8.0f);` setzt die Linienbreite auf ein Achtel der Breite der View. Mit `CGContextSetRGBStrokeColor(context, 0.3f, 0.3f, 0.3f, 1.0);` wird die Vordergrundfarbe auf dunkelgrau gesetzt und mit anschließendem `CGContextStrokeRect(context, rect)` wird ein Rechteck in dieser Farbe mit der oben definierten Linienbreite und von der Größe der View gezeichnet. Damit ist die ungecheckte Checkbox fertig und es bleibt nur noch das Häkchen zu zeichnen. Dieses Häkchen ist einfach eine gebrochene Linie mit drei Ankerpunkten, die der Inhalt des Arrays `addLines` sind. Mit `CGContextAddLines(context, addLines, 3);` werden alle drei Punkte zu dem Kontext hinzugefügt und mit Linien verbunden. Mit `CGContextStrokePath(context);` wird das Häkchen anschließend gezeichnet. Bei den Koordinaten der Punkte handelt es sich offensichtlich um relative Werte, die sich an die Größe der View orientieren und damit sicherstellen, dass dieses Häkchen immer maßstabsgetreu skaliert wird.

4.2.5 Animationen

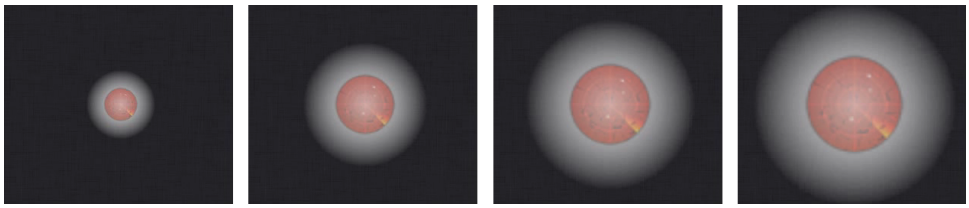
Sergiy Krutykov

Unter iOS lassen sich alle Views beliebig transformieren: Sie können skaliert, verschoben, gespiegelt etc. werden. Jede solche Transformation kann man auch animiert machen. Die folgende Abbildung zeigt einige Schritte einer Verschiebung, die animiert wurde:



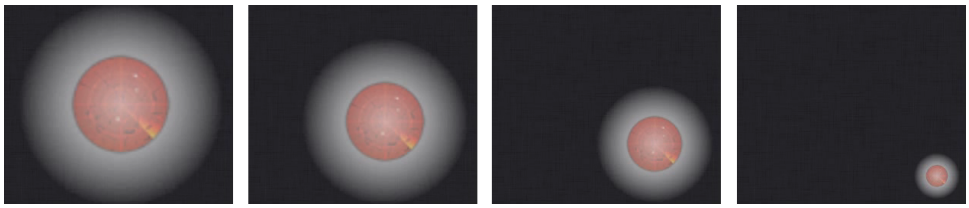
Dabei kann man bei jeder solchen Animation ihre Dauer festlegen.

Eine Vergrößerung lässt sich auch leicht animieren:



Allerdings kann man nicht jede Animation durch eine einzige Transformation beschreiben. Wenn man will, dass, wie in der obigen Abbildung, eine View zuerst vergrößert wird und anschließend wieder zum Originalzustand zurückkehrt, also eine Veränderung erfährt, die durch keine Transformation beschreiben werden kann (man kann nicht gleichzeitig vergrößern und verkleinern), dann muss man zwei Animationen unmittelbar hintereinander ausführen. Die Frage ist dabei, wie man erfährt, ob die erste Animation zu Ende ist, damit man die zweite starten kann. Dafür gibt es Möglichkeit bei einer Animation eine Listener-Methode anzugeben, die aufgerufen wird, sobald die Animation endet. Damit lässt sich beliebige Serie von Transformationen realisieren, wenn man eine Reihe von Methoden bereitstellt, die hintereinander aufgerufen werden.

Man kann eine Animation auch aus unterschiedlichen Transformationen kombinieren, indem man die entsprechenden Transformationsmatrizen miteinander multipliziert. In der folgenden Abbildung wird die View kleiner und wird gleichzeitig verschoben:



4.3 Datenhaltung

Phillipp Bertram, Peer Wagner

Die Entwicklung auf einem Smartphone bedeutet im allgemeinen eine große Umstellung im Gegensatz zur herkömmlichen Softwareentwicklung. Soviel moderne Smartphones auch zu leisten im Stande sind, so beschränkt sind sie dennoch aus der Sicht von Desktopcomputern, die eine Vielzahl an Möglichkeiten bieten oder anders gesagt kaum noch einer Einschränkung unterliegen. Die Datenhaltung und auch der Zugriff und die Zugriffskontrolle sind jedoch auf einem beschränkten Mobilgerät ein wichtiger Aspekt. Dies bedeutet, dass bereits der Entwurf einer vernünftigen Datenhaltung gut strukturiert werden sollte und das Datenmodell dementsprechend an bestimmte mobile Konzepte angepasst werden muss.

Für den Entwickler präsentieren sich direkt zwei unterschiedliche Lösungsmöglichkeiten, die im folgenden näher beleuchtet werden sollen.

Datenhaltung in einer Datenbank

Der Ansatz Daten in einer relationalen Datenbank zu halten ist für die meisten Entwickler sicherlich einer der offensichtlichsten Möglichkeiten. Diese Möglichkeit bietet sich auch auf einem Android-System mit der SQLite[31] Bibliothek, welche eine vollständige Datenbankimplementierung darstellt. SQLite behauptet von sich selbst besonders klein, schnell und zuverlässig zu sein. Dies sind natürlich Eigenschaften, wie sie auf einem mobilen System absolut wünschenswert sind.

31 http://www.sqlite.org Offizielle SQLite Homepage

Android spezifische Datenhaltung mit Content Providern

Um in einer Applikation eine Datenbank aufzusetzen und zu benutzen sind nur wenige Schritte notwendig. Die Anweisungen unterscheiden sich nur in wenigen Kleinigkeiten von Anweisungen die für andere Datenbanken benutzt werden. Android-spezifische Konzepte kommen eher bei der Darstellung von Daten aus einer Datenbank, als auch bei der Bereitstellung der Daten zum Einsatz. Bei Ersterem wird oftmals direkt vom System ein Cursor einer Abfrage weiterverarbeitet und dargestellt. Die Bereitstellung der Daten ist in einem Android-System dann oftmals modular gehalten. Diese Modularität wird durch den Einsatz von Content Providern erzeugt. Hierbei handelt es sich um eigenständige Activities oder sogar Applikationen, die eine fest definierte Schnittstelle auf die zur Verfügung gestellten Daten umsetzen. Damit kann jede andere Activity oder Applikation, die an die Daten heran möchte, eine Anfrage an den Content Provider stellen. Genau dieses Prinzip wird zum Beispiel bei hinterlegten Kontaktdaten wie dem Telefon- oder auch Adressbuch gemacht. Diese Art des Zugriffs ist jedoch nicht nur wegen seiner Einfachheit beliebt, er ist zudem auch die einzige Möglichkeit Daten zwischen Applikationen auszutauschen. Das sonst verwendete Rechtemanagment auf einem Android-System lässt den Zugriff auf Daten fremder Applikationen aus Sicherheitsgründen nämlich nicht zu.

Gegenüber den Vorteilen und der Einfachheit eine Datenbank auf einem Android-System einzusetzen, gibt es, besonders für unsere Applikation, einen entscheidenden Nachteil. Daten in einer Datenbank sind nicht ohne Weiteres dafür geeignet Echtzeitdaten vorzuhalten. Alles andere würde auch keinen Sinn ergeben, denn gerade wenn sich die Spieler über die Spielfläche bewegen oder auch einen Tresor lösen möchten, sollten die Daten, die diesen Schritten zugrunde liegen nicht mehrere Minuten alt sein. Es kommt außerdem hinzu, dass ein Spieler, der das Spiel unterbricht und zu einem späteren Zeitpunkt fortführt, einen komplett neuen Datensatz benötigt.

Datenhaltung in Objekten

Die zweite und ebenfalls augenscheinliche Option, Daten in Objekten und damit im Speicher zu halten, ist auf einem Android-System in keiner Weise vom Standardverhalten anderer Programmiersprachen und Konzepte zu unterscheiden. Einzig ist auf den Lebenszyklus und den damit möglichen Datenverlust in anderen Activities zu achten. Dieses Thema ist jedoch in der Einführung der Android-Konzepte genauer beschrieben.

Das wir uns in unserer Applikation für die Datenhaltung in Objekten/Speicher entschieden haben, ist bereits im oberen Abschnitt beschrieben worden. Es sei noch einmal erwähnt, dass die gesamte Spielidee darauf beruht, dass die Spieler in Echtzeit miteinander das Spiel spielen und sich daher im Sekundentakt Änderungen ergeben können. Der Datenhaltung im Speicher würde als einziger Punkt entgegensprechen, dass selbst moderne Smartphones einen begrenzten Platz an Arbeitsspeicher zur Verfügung haben. Diesem Punkt kann jedoch von unserer Seite die geringe Datenmenge entgegen gesetzt werden, womit die Speicherbegrenzung keine Rolle mehr spielt. Auch wenn sich im Laufe der Entwicklung verschiedene Speicherprobleme ergeben haben, so waren dies in allen Fällen fehlerhafte Freigaben des Systems und keine Engpässe auf Grund der vorzuhaltenden Datenmenge.

Das unten stehende Bild zeigt einige Strukturen, wie sie für das Modell verwendet werden und wird im weiteren genauer beschrieben.

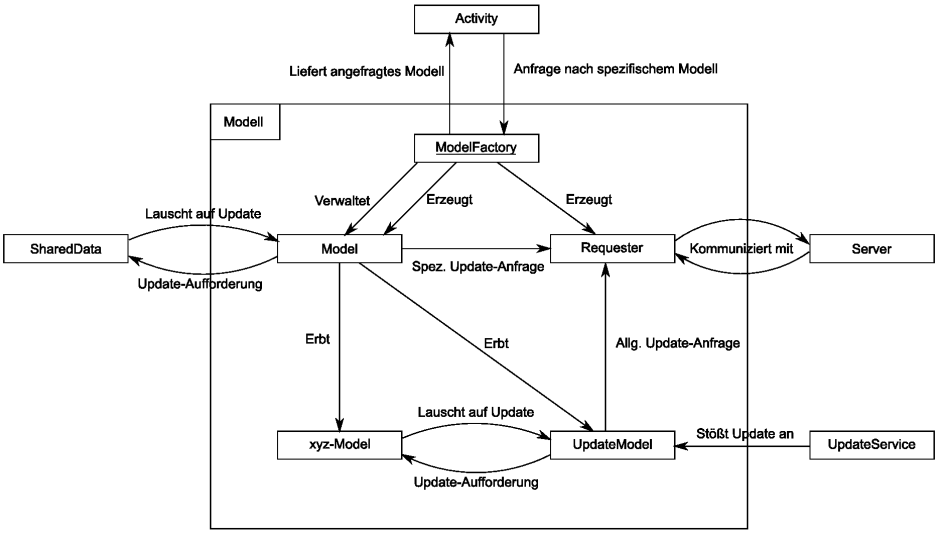


Abb. 41: Skizze der Datenhaltung in der Applikation auf dem Android-System

Die Datenhaltung hat ihren eigenen Bereich bekommen. So wurde versucht die Applikation nach dem Model-View-Controller Paradigma zu entwickeln, um den logischen Teil von den Daten und den Ansichten zu trennen. Wie in der allgemeinen Beschreibung des Android-Systems zu lesen ist, ist diese Trennung jedoch nicht vorgesehen und damit an manchen Stellen schwierig umzusetzen.

Jede `Activity` ist in diesem Bild grob als Controller zu verstehen. Sie steht damit ganz oben und regelt was und wie etwas angezeigt werden soll. Die View-Ebene ist für diese Betrachtung allerdings ohne Bedeutung und findet sich daher nicht weiter im gezeigten Bild. Eine `Activity` ist hier also als nicht näher spezifizierter Controller zu verstehen, der irgendwie an die Daten gelangen muss. Hierfür wird die `ModelFactory` eingesetzt. Sie ist ein Singleton und dient dazu, alle `Model`-Implementierungen zu erzeugen und zu verwalten. Die verschiedenen Modelle speichern logische Datenblöcke. Es gibt zum Beispiel ein Modell für den Chat, eines für die Tresore und eines für das Inventar. Sie werden alle unter dem Bezeichner `xyz-Model` in der Grafik ausgedrückt. Die `ModelFactory` erstellt außerdem ein `Requester`-Objekt. Dieses Objekt beinhaltet die Methoden für die Kommunikation mit dem Server und ist die einzige Schnittstelle zu diesem. Alle Modelle müssen wohlgeformte Anfragen an den `Requester` übergeben und bekommen dann bei Anfragen eine Antwort mitgeteilt. Die Kompression des Datenstroms ist ebenfalls eine Aufgabe des `Requesters`.

Nun gibt es noch drei weitere Objekte, die direkt oder auch indirekt mit dem Modell interagieren. Zunächst sei das `SharedData` Objekt beschrieben. Diese Klasse beinhaltet einige eindeutige Daten, die Modellübergreifend Gültigkeit haben. Darunter fallen die aktuelle Session-ID, der Spielername, die Teamzugehörigkeit, und eine Spiel-ID. Alle Modelle, die an Änderungen dieser Felder interessiert sind, können sich über ein Observer-Observable-Pattern am `SharedData`-Objekt registrieren.

Die beiden letzten unbeschriebenen Objekte, das `UpdateModel` und der `UpdateService` gehören gewissermaßen zusammen. Der `UpdateService` ist eine eigene Activity, die in einem getrennt von der restlichen Applikation laufendem Thread ausgeführt wird. Es handelt sich dabei im Grunde genommen um einen Timer, der zu festgelegten Zeiten das `UpdateModel` zu einer Update-Anfrage anregt. Dies hat seinen Grund darin, dass regelmäßige Updates nicht den Hauptthread blockieren sollen. Das `UpdateModel` ist hierbei erst einmal als normales `Model` zu betrachten. Die Besonderheit ist, dass sich bei dem `UpdateModel` jedes `xyz-Model` registrieren kann, dies ebenfalls im Observer-Observable-Pattern, um über allgemeine Updates informiert zu werden. Die Modelle sollten dann in ihren Update-Methoden die eigenen Daten verändern. Die Modelle selbst sind dann ebenfalls observable, so dass sich Activities an ihnen registrieren und ein Update der Views einleiten können.

Datenhaltung auf dem iPhone und Notifications

Das Grundkonzept stimmt größtenteils mit dem Konzept der Datenhaltung beim Android überein. Lediglich ein paar iPhone spezifische Änderungen mussten vorgenommen werden. Während die Kommunikation zwischen den Objekten beim Android weitgehend über das Observable-Pattern implementiert wurde, stellt Apple von Haus aus ein sehr nützliches *Notification* Feature zur Verfügung. *Notification* ist ein System, welches registrierte

Objekte über Änderungen irgendwo innerhalb der Applikation benachrichtigt und die entsprechenden Methoden aufruft. Üblicherweise bekommen Objekte ihre Informationen über die an sie geschickten *Messages*. Das bedeutet auch, dass das Objekt, welches die *Message* verschickt auch wissen muss, welche Objekte diese Änderungen mitkriegen wollen und welche Methode aufgerufen werden soll. *Notification* ist ein Broadcast-Modell, wo man Objekte registrieren kann, die benachrichtigt werden sollen. *Notifications* werden von einer Singleton-Klasse `NSNotificationCenter` verwaltet. Das Objekt dieser Klasse ist über `defaultCenter` zu erreichen und mit folgendem Code-Beispiel kann sich ein Objekt registrieren.

```
[[NSNotificationCenter defaultCenter] addObserver:self
                                      selector:@selector(updateAreaVisibilities)
                                      name:@"notif_effects" object:nil];
```

In diesem Fallt registriert sich ein Objekt selber (`addObserver:self`) und gibt an, dass die Methode `updateAreaVisibilities` aufgerufen werden soll, wenn die Notification `notif_effects` gepostet wird. Sobald irgendwo in der Applikation in etwa dieser Code aufgerufen wird

```
[[NSNotificationCenter defaultCenter] postNotificationName:@"notif_effects" object:nil];
```

werden die Methoden der Objekte, die auf die *Notification*`notif_effect` registriert sind, aufgerufen und ausgeführt. Im iPhone wird dieses Konzept vor allem in den Models verwendet. Wenn das `UpdateModel` vom Server z.B. eine Benachrichtigung bekommt, dass ein Spieler eine Kachel aufgedeckt hat, postet es diese *Notification*.

5. Sensoren

Peer Wagner

Bereits durch den Grundgedanken dieses Projektes war vorgegeben, die mobilen Möglichkeiten der modernen Telefone zu nutzen um den entscheidenden Unterschied zu einem gewöhlichen Computerspiel zu verdeutlichen. Die Mobilität äußert sich dabei in den Sensoren, die ein Smartphone mit sich bringt. Dieser Abschnitt beschreibt daher die große Herausforderung, die Sensoren der Geräte in das Spielgeschehen zu integrieren. Um einen Einstieg in das Thema zu finden sind hier kurz die verschiedenen Sensoren und die Problematik beim Umgang mit diesen erklärt.

Sensoren ermöglichen es, die reale Welt in digitaler Form wahrzunehmen. Zu den Sensoren die sich oftmals in einem Smartphone finden lassen, gehören ein GPS-Empfänger, drei Achsen Beschleunigungssensoren und drei Achsen Magnetfeldsensoren. In höherwertigen Smartphones kommen in neuester Zeit ab und an auch Gyrosensoren zum Einsatz, die eine erhöhte Genauigkeit beziehungsweise im Zusammenspiel mit den beiden letztgenannten Sensorarten eine Ausgleichsmöglichkeit für ungenaue Sensorwerte bieten.

Die Genauigkeit, besser gesprochen die Auflösung, aber auch die Fehlerrate der Sensoren spielen eine entscheidende Rolle, wenn man sich in einem Spiel auf diese verlassen will. Die Auflösung der Magnetfeld- und Beschleunigungssensoren lässt sich in den beiden betrachteten Smartphones durchaus beeinflussen, während die Fehlerrate durch mehrere äußere Umstände beeinflusst wird. Bei einem GPS-Empfänger hingegen bekommt man die Auflösung von außen vorgegeben und man kann nur auf die Fehlerrate indirekt Einfluss nehmen. Dieser Einfluss ist jedoch auch nur theoretischer Natur, da die Fehlerrate, wie bei den anderen Sensoren auch, mit unterschiedlicher Umgebung sich unterschiedlich verhält. Die folgenden Abschnitte beschreiben die Sensoren, deren Genauigkeit und deren Benutzung mit den verschiedenen Smartphones.

5.1 Android

Peer Wagner

Magnetfeldsensoren

Für die Sensorik gilt es zunächst einige Gegebenheiten zu klären. Wichtig ist dabei als erstes das verwendete Koordinatensystem der Smartphones, ohne das keine vernünftige Beschreibung möglich ist. Das unten stehende Bild zeigt ein Smartphone mit den drei Raumachsen eingezeichnet. Der Referenzrahmen für die Achsen ist der Bildschirm des Smartphones. Die Achsen bilden ein Rechtssystem, in der die x-Achse horizontal liegt und nach rechts zeigt, die y-Achse liegt dann vertikal und zeigt nach oben. Die z-Achse ist damit festgelegt und zeigt aus dem Bildschirm nach vorne hinaus. Das Koordinatensystem ist dabei immer nur durch den Standard-Ausgangszustand bestimmt und die Achsen werden nicht neu definiert, sollte sich der Bildschirm bei einer Drehung des Gerätes ebenfalls drehen. Der aufmerksame Leser wird auch festgestellt haben, dass diese Definition der Achsen nicht wie in einer 2D-API ist, sondern den Ursprung in der "Mitte" des Grätes hat. Für alle 2D-API Zugriffe gilt wie auch bei sonstigen Angaben in Bildschirmkoordinaten, dass der Ursprung in der linken oberen Ecke des Bildschirms sitzt.

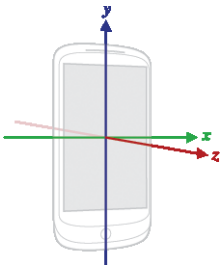


Abb. 42: Koordinatensystem eines Handies

Das neben stehende Bild[32] zeigt die Koordinatenachsen, wie sie für die Lageberechnung bei einem Android-Smartphone festgelegt sind. Das Koordinatensystem wird in Referenz zum Bildschirm festgelegt und bildet ein Rechtssystem.

Wie der Name Magnetfeldsensor bereits verrät, wird mit ihnen das lokale Magnetfeld gemessen. Für den Laien ist ein Magnetfeld beziehungsweise seine Stärke eher eine sehr abstrakte Größe, da sie im Alltag kaum an Bedeutung zu haben scheint. Dem ist allerdings nicht so. Ein Magnetfeld beziehungsweise die magnetische Flussdichte, die ein Maß für die Stärke ist, wird allgemein in der Einheit Tesla [T] gemessen, wobei das Erdmagnetfeld in etwa 30-60 µT beträgt. Jedes elektrisch betriebene Gerät erzeugt oftmals unbeabsichtigt ein magnetisches Feld. Dies geht von Fernsehern über den Toaster oder auch ein Ladegerät eines Smartphones bis hin zu Hochspannungsleitungen, Autos und auch Eisenbahnen. Es gibt also heutzutage kaum noch ein Umfeld das frei von elektromagnetischer Strahlung ist. Das Problem dabei ist, dass alle diese Quellen durchaus in die Größenordnung des Erdmagnetfeldes gelangen können. Auf einer Internetseite der TU Graz[33] zum magnetischen Feld lassen sich sehr schön die Größen solcher Einflüsse mit dem Erdmagnetfeld vergleichen.

Die Funktion des Kompass wird demnach von vielen verschiedenen äußeren Quellen beeinflusst, welche auch noch in Größenordnungen liegen, die dem Erdmagnetfeld, an dem sich der Kompass eigentlich ausrichten sollte, gleich kommen. Es ist also wenig verwunderlich, dass die Sensorwerte der Magnetfeldsonsoren nicht immer die zuverlässigsten sind.

Beschleunigungssensoren

Die Beschleunigungssensoren in einem Smartphone sind ähnlich den Megnetfeldsensoren. Es handelt sich jedoch nicht um Gyrosensoren, sondern tatsächlich "nur" um einen drei Achsen Beschleunigungssensor, denn die Beschleunigung wird entlang der gleichen Achsen wie das Magnetfeld gemessen. Auf Grund dieser Gegebenheit ergibt sich eine Besonderheit, die man bedenken sollte, wenn man mit den Werten der Sensoren arbeitet. Und zwar ist dies die Lage des Sensors im Gerät. Wie das oben stehende Bild zeigt, liegt der Ursprung des Koordinatensystems im Mittelpunkt des Smartphones. Dies ist aber natürlich nur eine Annahme und kann in keinem Fall von allen Sensoren des Smartphones erfüllt werden. Diese Abweichung vom idealen Mittelpunkt des Gerätes kann dazu führen, dass Drehungen des Smartphones eine Scheinkraft auf die Sensoren auswirken, die jedoch keinesfalls gewünscht ist. Für die Detektion von Drehungen ist somit ein Gyrosensor unerlässlich. Anders sieht es bei der Bestimmung der Lage des Gerätes aus. Hierbei wird nicht die Drehung selbst, sondern der momentane Zustand des Gerätes erfasst und mit Hilfe der Magnetfeldsensoren in eine Lagebeschreibung umgerechnet. Bevor wir uns jedoch der Lageberechnung zuwenden soll auf einen weiteren Punkt hingewiesen werden. Beschleunigung ist eigentlich als Änderung der Geschwindigkeit eines Gegenstandes definiert. Oder in einer Formel ausgedrückt:

$$a_{device} = -\sum \frac{F_{sensor}}{m}$$

32 http://developer.android.com/reference/android/hardware/SensorEvent.html - Android API, Beschreibung der SensorEvent Klasse
33 http://www.igte.tugraz.at/archive/leoben97/grundlagen/mag_feld.htm

Da wir uns jedoch zu jedem Zeitpunkt im Gravitationsfeld der Erde befinden, scheint ein ruhendes Smartphone mit der Gravitationskonstanten g=9,81m/s^2 beschleunigt zu werden. Es wird somit tatsächlich folgendes von den Sensoren gemessen:

$$a_{device} = -g - \sum \frac{F}{m}$$

Aus dieser Beschreibung ergibt sich, dass man den Vektor für die Gravitation am besten durch die Verarbeitung der Sensorwerte mit einem Tiefpassfilter erhält. Dies hat zusätzlich den Vorteil, dass ungewollte 'Ausreißer' aus den Werten gemittelt werden und so eine einigermaßen zuverlässige Messung stattfinden kann. Die Anwendung eines Tiefpassfilters wird noch einmal ausführlicher bei den Messungen der Sensortoleranzen beschrieben.

Lageberechnung

Nachdem die Grundlagen der Sensorik für Magnetfeld- und Beschleunigungsberechnung erläutert wurden, lässt sich nun die eigentliche Lageberechnung beschreiben. Auch wenn man oftmals von Lagesensoren spricht, so sind solche Art der Sensoren in kaum einem modernen Smartphone tatsächlich verbaut. Vielmehr wird die Lage aus vorhandenen Magnetfeldsensoren und Beschleunigungssensoren gewonnen. Dies mag darin begründet sein, dass so gut wie jedes Smartphone inzwischen eine Kompassfunktionalität liefert. Diese Funktionalität setzt selbstverständlich die Anwesenheit von Magnetfeldsensorik voraus. Die Verwendung von Accelerometern ist dann nur noch ein kleiner Schritt. So erschließen sich mit diesen größere Einsatzmöglichkeiten als es mit nur auf die Lage beschränkten Sensoren der Fall wäre. Aus diesem Grund wurde in den beiden vorangegangenen Punkten zuerst einmal getrennt auf die beiden Sensorarten und ihre Verwendung eingegangen.

Auch für die Lageberechnung muss als erstes wieder einmal auf die verwendeten Koordinatensysteme eingegangen werden. Da bei der Bestimmung des Magnetfeldes oder der Beschleunigung bisher immer nur der Vektore relativ zum Gerät wichtig war, ist nun die Ausrichtung des Gerätes relativ zur Beschreibung der Vektoren innerhalb eines Weltkoordinaten-/Referenzsystems nötig. Dieses Referenzsystem ist für die Beschreibung der Rotationsmatrix R des Systems verschieden von der Beschreibung der Rotation des Gerätes zu einem Referenzsystem. Die nachstehenden beiden Bilder zeigen beide Systeme.

Das neben stehende Bild[34] zeigt die Koordinatenachsen, wie sie für die Lageberechnung als Referenz-Koordinatensystem festgelegt sind. Die Achsen beschreiben dabei ein orthonormales Rechtssystem auf der gedachten Erdoberfläche. Tangential nach Norden zeigend befindet sich die y-Achse, senkrecht auf der Oberfläche in den Himmel zeigen die z-Achse und auf Grund der Definition als

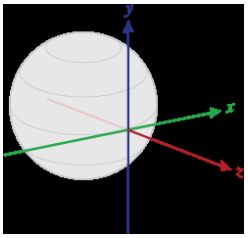


Abb. 43: Weltkoordinatensystem

34 http://developer.android.com/reference/android/hardware/SensorManager.html#getRotationMatrix%28float[],%20float[],%20float[],%20float[]%29 - Android API, Beschreibung des Koordinatensystems und der Rotationsmatrix in der SensorManager Klasse

Orthonormalensystem die x-Achse ebenfalls tangential nach Osten zeigend.

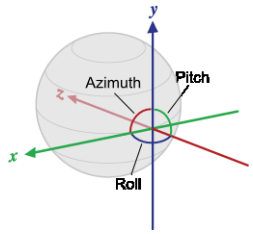


Abb. 44: Referenzsystem

Das neben stehende Bild[35] zeigt die Koordinatenachsen, wie sie für die Orientierungsberechnung festgelegt sind. Sie sind um 180° gedreht zum eigentlichen Referenzsystem für die Rotationsmatrix. Zusätzlich zu sehen sind die Beschreibung der Werte. Der Azimuth ist die Drehung des Smartphones um die z-Achse und gibt damit den Winkel zum geomagnetischen Norden an, der Pitchwinkel ist der Anstellwinkel des Gerätes als Drehung um die x-Achse und der Roll die seitliche Drehung um die z-Achse.

Alle Winkel aus der Berechnung der Orientierung werden im Radialmaß gespeichert, wobei die übliche Umrechnung gilt:

$$1\,rad = \frac{360\,^\circ}{2\pi}$$

Die so errechneten Werte für die Orientierung liegen in verschiedenen Intervallen. Der Azimuth läuft von -pi bis pi, der Pitch von -pi/2 bis pi/2 und der Roll von -pi bis pi. Diese Werte stellen auch einen der großen Stolpersteine dar, da die Intervalle in denen die Winkel laufen nur unzureichend und an der falschen Stelle in der Android-API dokumentiert sind. Für die von uns entwickelte Applikation ist somit der Azimuth für den Kompass zuständig, während über die Pitch- und Rollwinkel definiert wird in welche Ansicht der Benutzer wechseln möchte.

Global Positioning System

Vom Global Positioning System, kurz GPS, hat bestimmt jeder schon einmal etwas gehört. Es handelt sich bei diesem System um ein satellitengestütztes Ortungssystem, das mittlerweile weltweit in der zivilen Technik Anwendung findet. Zur Ortung eines GPS-Empfängers werden in der Regel vier Satelliten benötigt. Dies erklärt

35 http://developer.android.com/reference/android/hardware/SensorManager.html#getOrientation%28float[],%20float[]%29 - Android API, Beschreibung des Koordinatensystems und der Orientierungsberechnung in der SensorManager Klasse

sich dadurch, dass drei Satelliten für die Laufzeitmessung, also die eigentliche Entfernungsbestimmung benötigt werden, und ein weiterer, der eine exakte Uhrzeit sendet. Eine exakte Uhrzeit ist deshalb nötig, da sonst die Laufzeitmessung falsche Werte ergibt und sich alleine durch wenige Sekunden Unterschied die Genauigkeit drastisch reduziert. Derzeit sind in etwa Genauigkeiten von 10 Metern möglich, wobei die vom Militär eingesetzte Technik sogar wenige Zentimeter erlaubt. Für eine Vertiefung eignet sich der Wikipedia Artikel[36], der die verschiedenen Techniken von und Einflüsse auf das GPS sehr genau erklärt.

5.1.1 Sensormessungen

Sascha Henke, Peer Wagner

Die vorangegangenen Punkte haben deutlich werden lassen, dass die Sensorik in Smartphones in keinem Fall ohne Überprüfung eingesetzt werden sollte. Da wir während unserer Projektarbeit über manche der angesprochenen Punkte gestolpert sind haben wir die Sensorwerte auf ihre Genauigkeit und auch Zuverlässigkeit getestet. Die nachfolgenden Bilder geben diese Tests der Sensoren wie GPS und die Lageberechnung auf Grund der Beschleunigungs- und Magnetfeldsensoren wieder. Die Angaben der Lageberechnung erfolgen dabei in Grad. Die so erhaltenen Daten wurden in ihrer rohen Form mit einer Glättung der Daten verglichen. Die Glättung ist dabei der im Abschnitt Beschleunigungssensoren beschriebene Tiefpassfilter, welcher sich auch als exponentieller Glättungsansatz 1. Ordnung sehen lässt. Die Formel für die Glättung der Werte ist damit wie folgt:

$$x_n^* = (1 - \alpha) \cdot x_{n-1} + \alpha \cdot x_n$$

Hierbei ist das Ergebnis x der geglättete Wert und das Alpha der sogenannte Gegenwartswert und liegt für diese Glättung bei einem Wert von alpha=0,1. Dieser Gegenwartswert ist in der Literatur oftmals beschrieben und hat auch für unsere Messungen brauchbare Ergebnisse geliefert.

Zur Messung lässt sich sagen, dass alle hier gezeigten Ergebnisse entstanden sind aus Messungen, die in einer möglichst störfreien Umgebung aufgenommen wurden. Das Smartphone wurde dabei auf einer Apparatur fixiert, wobei hier der Pitch- beziehungsweise Roll-Winkel eingestellt wurden. Die Messungen haben im Schnitt 100 Messpunkte, von denen jeweils die ersten und letzten 20 Werte aus der Bewertung genommen wurden da hierbei noch Einstellung am Smartphone selbst gemacht werden mussten um die Programme zu starten.

Kompass

Im Folgenden sei der Kompass getestet, indem das Gerät in einer gleichbleibenden Position Messreihen aufgenommen hat, welche später dann von einem selbstgeschriebenen Tool ausgewertet wurden. Hierbei beschreibe "Difference" die Differenz zwischen Maximum und Minimum, wobei der "0-Durchgang" zu beachten ist. Weiterhin beschreibe "Average" den durchschnittlichen Wert des Kompasses und "Deviation" die Ablenkung von diesem.

36 https://secure.wikimedia.org/wikipedia/de/wiki/GPS-Technik

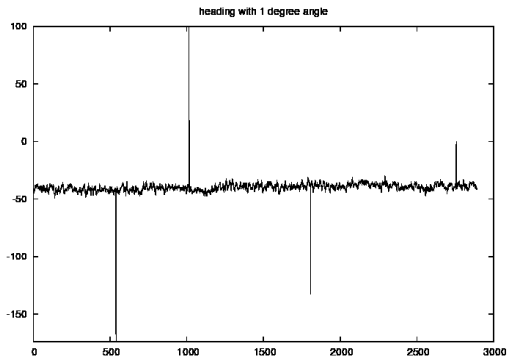


Abb. 45: Ungeglättet

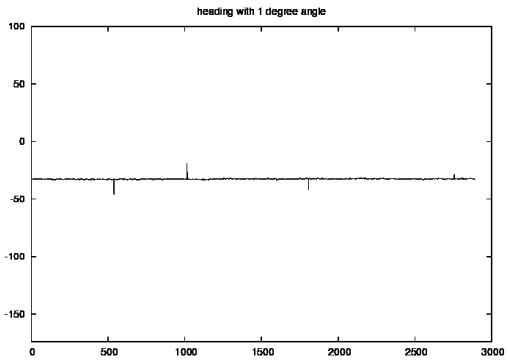


Abb. 46: Nach Glättung mit Exponentiellen Glättungsansatz

Maximum: 149.626071815
Minimum: -172.962682918
Median: -39.9645533473
Average: -40.5885227469
Deviation: 8.40884361703
Difference: 37.411245267

Maximum: -13.647577082
Minimum: -45.9064525553
Median: -32.6066395982
Average: -32.6690365382
Deviation: 0.840884361703
Difference: 32.2588754733

Beispielhaft ein Datensatz aufgenommen innerhalb eines grossen Stahlbeton-Gebaeudes (AVZ):

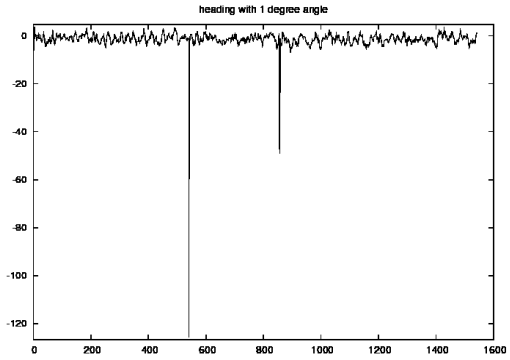


Abb. 47: Ungeglättet

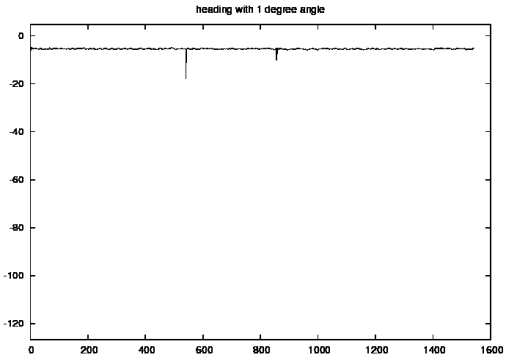


Abb. 48: Geglättet

Maximum: 3.68465869271
Minimum: -125.676739646
Median: -1.30598309597
Average: -1.47843925891
Deviation: 3.94275255656
Difference: 129.361398339

Maximum: -4.97066873204
Minimum: -17.9068085659
Median: -5.46973291091
Average: -5.4869785272
Deviation: 0.394275255656
Difference: 12.9361398339

Man erkennt das beim Kompass immer ein gewisses Rauschen und einige Peaks vorhanden sind, was bei der rohen Nutzung in der *AR* zu einer Unruhe bzw. Ruckeln führt. Die exponentielle Glättung hingegen bietet hier eine gut Möglichkeit das Grundrauschen zu minimieren und die Peaks zu verkleinern.

GPS Daten

Zu Messung der GPS Daten wurden vom Handy immer ohne Bewegung Daten aufgezeichnet. Danach wurde das Handy fünf Meter bewegt und dann wieder eine Datenreihe aufgenommen. So kann nun anhand der durchschnittlichen Position und dem gemessenen Abstand zwischen den Messreihen ein Fehler berechnet werden. Bei jeder Messung wurden etwa 40 GPS Daten aufgezeichnet, wenn die einzelnen Punkte nun im Diagramm nicht zu erkennen sind liegt dies daran, dass diese auf den selben Koordinaten liegen.

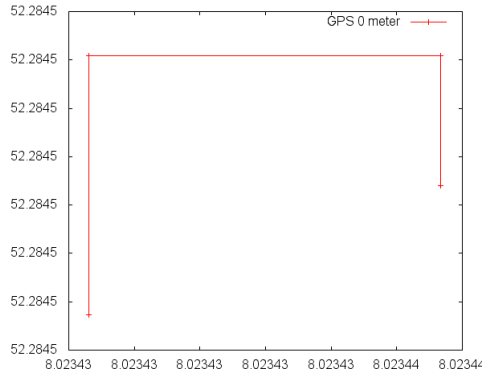


Abb. 49: GPS - 1.Datenreihe

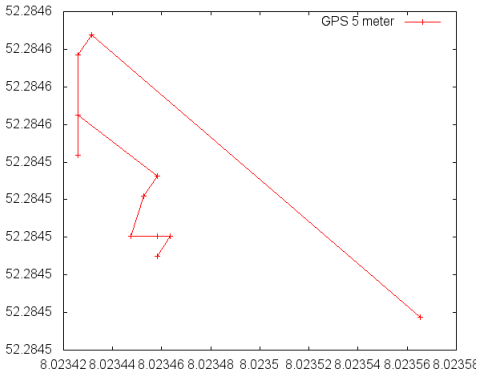


Abb. 50: GPS - 2.Datenreihe

Durchschnitt Longitude: 8.02343612909
Durchschnitt Latitude: 52.2845016003
obere Abw. longitude -4.82797622681e-06
untere Abw. longitude -5.36441802979e-07
obere Abw. latitude -8.15391540243e-06
untere Abw. latitude -2.57492065714e-06

Durchschnitt Longitude: 8.02345765264
Durchschnitt Latitude: 52.2845186685
obere Abw. longitude -3.17159451928e-05
untere Abw. longitude -0.000107758923582
obere Abw. latitude -1.98577579695e-05
untere Abw. latitude -5.52440944475e-05

Hier ist, besonders in der linken Abbildung, zu sehen, dass die GPS Daten nicht allzu häufig streuen. Vergleicht man beide Grafiken, so sieht man, dass der Unterschied zwischen den durchschnittlichen Positionen 1,86 Meter beträgt, was einen Fehler von etwa 3.2 Metern in der durchschnittlich aufgenommenen Position anzeigt. Weitere Messreihen zeigen, dass diese Fehlerzahl ungefähr gleich ist bzw. sich etwa in einem Intervall von etwa 3,2 bis 3,7 Metern Fehlerabweichung zwischen den Messungen möglich ist.

5.2 iPhone

Sergiy Krutykov

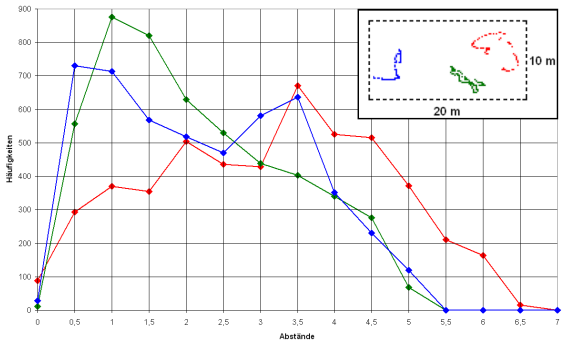
Dieser Abschnitt umfasst einige Tests der Genauigkeit und Zuverlässigkeit von GPS, Accelerometer und Kompass auf iPhones.

5.2.1 GPS

Sergiy Krutykov

Messungen an bestimmten Stellen

Zuerst wurden GPS-Koordinaten an drei bestimmten Stellen (ohne jede Bewegung) unabhängig von einander gemessen. Diese drei Punkte lagen auf einer Linie, wobei der zweite Punkt sich in einem Abstand von 5 Metern von dem ersten und der dritte in einem Abstand von 10 Metern von dem zweiten befand, also in unmittelbarer Nähe. An jedem Punkt wurden je 100 Messungen in einem Navigation-Modus vorgenommen, also in dem Modus, der maximal häufig die Position updatet (1 mal pro Sekunde) und versucht maximal genau zu sein. Aber auch in diesem Modus schwankte die Position während dieser 100 Updates bis zum gewissen Grad. In der folgenden Abbildung

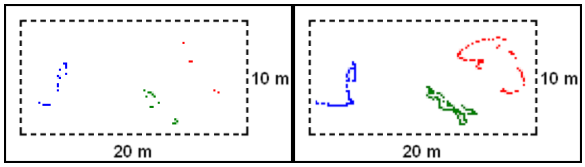


oben rechts ist eine schematische Darstellung von diesen Schwankungen: Die gestrichelte Linie umrandet das "Test-Gebiet" der Größe 10x20 m², die roten Punkte entsprechen den Messungen an der ersten Stelle, die grünen denen an der zweiten und die blauen denen an der dritten. Zur Erinnerung: Die Stellen selbst befanden auf einer horizontalen Linie, was man auf der Abbildung kaum sieht. Das Diagramm selbst versucht, die Tests visuell zu verfassen. Da es unklar ist, welcher Punkt als Referenzpunkt (etwa als Zentrum) für die jeweiligen Stellen ist, wurden alle Abstände zwischen allen Wertepaaren, bezogen auf jeweilige Stelle gemessen und ihre Häufigkeiten im Diagramm abgetragen. Z.B., liegt die Mehrheit der Paare bei den Messungen an der zweiten Stelle (grün) im Abstand von einem Meter voneinander. Man sieht, dass die erste Messreihe (rot) die schlechtesten Ergebnisse liefer, so dass die Abstände zwischen Punkten über 6 Meter hinaus gehen. Der einzige Unterschied dieser Messreihe von den anderen, dass sich hier beim Messen einige Menschen in unmittelbarer Nähe von dem Gerät befanden. Dass dies in der Tat Einfluss auf die Messungen hatte, und nicht einfach ein Zufall vorliegt, ist nur eine Vermutung.

Zu jeder GPS-Messung gehört eine **horizontale Genauigkeit**, die meistens einen der Werte (hier aufgerundet) enthält: 10 Meter, 20 Meter, 50 Meter oder 80 Meter. Diese Genauigkeit ist der Radius des Kreises um den angegebenen Punkt, innerhalb von welchem sich das Gerät tatsächlich befindet. Diese Werte sind in gewissem Sinne nur Worst Cases. Die Genauigkeit von 10 Metern ist am besten. Diese Genauigkeiten sind meistens nicht konstant sondern schwanken ständig. In der folgenden Tabelle ist die prozentuale Verteilung von solchen Genauigkeiten während der oben beschriebenen drei Messreihen mit je 100 Messungen.

	10 Meter	20 Meter	50 Meter
1. Stelle	5%	70%	25%
2. Stelle	15%	55%	30%
3. Stelle	25%	40%	35%

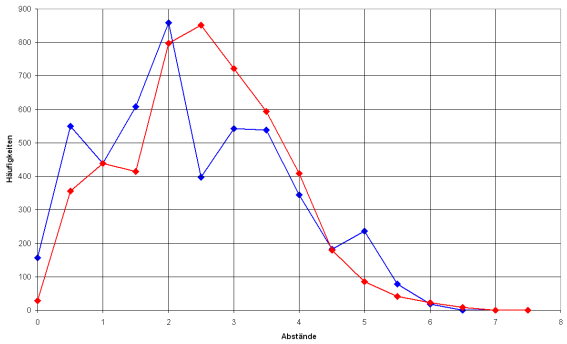
Man sieht, dass GPS bei roten Punkten auch selten eine gute Genauigkeit versprochen hat. Allerdings muss man an dieser Stelle beachten, dass diese Genauigkeiten nur relativ sind. In der folgenden Abbildung links sind nur die Punkte mit der "besten" Genauigkeit von 10 Metern dargestellt (rechts nochmal alle Punkte):



Die Punkte mit dieser (relativ) "guten" Genauigkeit scheinen kaum besser zu sein als die restlichen und eine Linie, auf welcher aller drei Positionen des Geräts in Wirklichkeit lagen, ist auch in diesem Fall kaum zu erkennen.

Vergleich zwischen ruhendem und bewegten Gerät

Es wurde zusätzlich untersucht, ob die Vibrationen des Geräts selbst die Messungen beeinflussen. Dafür wurde eine Messreihe (wieder 100 Messungen) mit einem auf dem Boden liegenden Gerät durchgeführt und eine andere, während welcher das Gerät permanent geschüttelt wurde. Das folgende Diagramm zeigt, dass dies für die Messungen kaum einen Unterschied ergibt (die blauen Punkte entsprechen dem ruhenden Gerät und die roten dem geschüttelten):

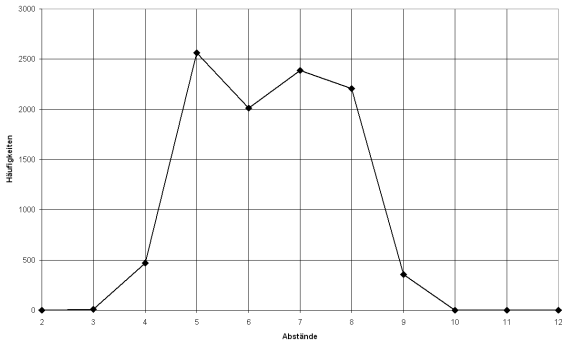


Allerdings sind die horizontalen Genauigkeiten bei dem geschüttelten Gerät besser:

	10 Meter	20 Meter	50 Meter
ruhend	10%	70%	25%
geschüttelt	40%	50%	10%

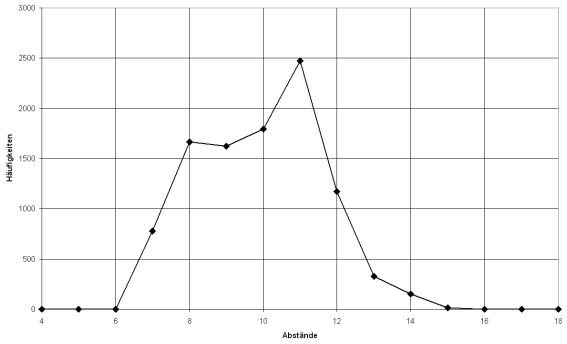
Abstände zwischen je zwei Punkten

Da die Abstände zwischen diesen drei Stellen bekannt sind, kann man jetzt versuchen auch die Statistik über die Abstände zwischen unterschiedlichen Stellen zu machen. Dafür werden die Punktepaare nicht einer Messung sondern jeweils zweier Messungen in Betracht gezogen. Vergleich der ersten und der zweiten Messreihen (die den beiden Stellen im Abstand von 5 Metern entsprechen) ergibt das folgende Diagramm:

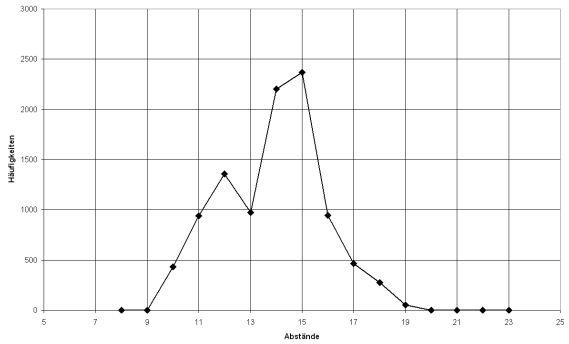


Man sieht, dass obwohl die meisten Paare in der Tat einen Abstand von 5 Metern liefern und kaum welche unter 3 Meter kommen, es jedoch sehr viele Paare mit einem Abstand von 7 und 8 Metern gibt.

Das Diagramm für die Abstände von 10 Metern



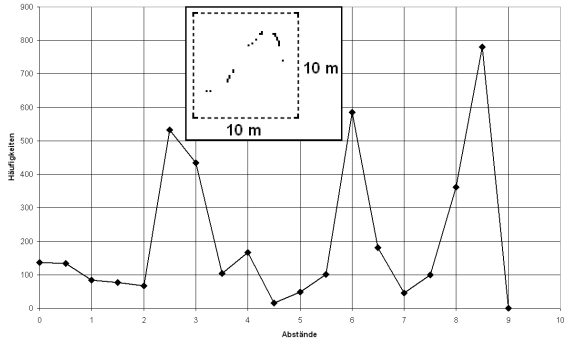
und das Diagramm für die Abstände von 15 Metern



sprechen für sich.

Messungen beim schlechten Wetter

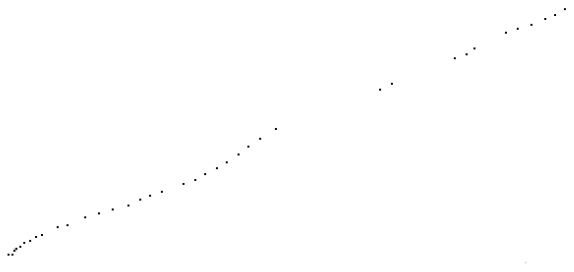
An dieser Stelle muss man erwähnen, dass die Bedingungen für die oben beschriebenen Experimente nahezu ideal waren: Das Wetter war heiter, kein Gebäude war im Umkreis von 100 Metern. Es ist aber bekannt, dass das Wetter die Genauigkeit von GPS beeinflussen kann. Deshalb wurde noch eine Messreihe auch unbewegt an einer Stelle durchgeführt unter dem Regen. Das Diagramm zeigt in der Tat eine etwas schlechtere Genauigkeit:



Die Mehrheit der Paare liefert hier 8.5 Meter als Abstand. Die horizontale Genauigkeit von allen Werten war dem Wetter entsprechend nur 50 Meter.

Messungen beim Laufen

Im Gegensatz zu all dem oben beschriebenen scheint GPS beim Gehen etwas bessere Ergebnisse zu liefern. Die nächste Abbildung zeigt eine 65 Meter lange Strecke, die mit GPS "aufgenommen" wurde:



Die horizontale Genauigkeit war fast durchgehend 20 Meter. Das war eine Gerade Strecke und die Bewegung gleichmäßig. Während das Kurvenverhalten der Strecke auf die Ungenauigkeit von GPS zurückzuführen ist, bedeuten die "Lücken", dass GPS manchmal "hängen bleibt" und liefert "veraltete" Werte, allerdings weiterhin 1 mal pro Sekunde (manche Punkte werden in der Abbildung mehrmals übereinander gezeichnet).

Fazit

Als positiv kann man bei diesen Tests die Tatsache betrachten, dass in **keiner** der Messungen sich irgendwelche "Ausreißer" ergeben haben, die in den Diagrammen nicht berücksichtigt wurden. Nach diesen Tests kann man relativ sicher sagen, dass GPS (zumindest in dem Navigations-Modus) zwar um bis zu 10 Meter schwanken kann, aber nicht plötzlich um 100 Meter springen.

Die horizontale Genauigkeit bekommt man zwar mit, kann sie aber kaum gebrauchen, denn man kann sie nicht beeinflussen. Aufgrund der kleinen Geschwindigkeit der Updates (1 mal pro Sekunde) scheidet praktisch jeder Ansatz aus, der versucht einen Mittelwert zu bilden (dann könne man die Genauigkeiten als Gewichte benutzen).

Auch wenn Updates für die Bildung des Mittelwertes zu langsam sind, sind sie für den Akku gerade zu schnell, denn mit dem Tempo kann der Akku kaum eine Stunde durchhalten.

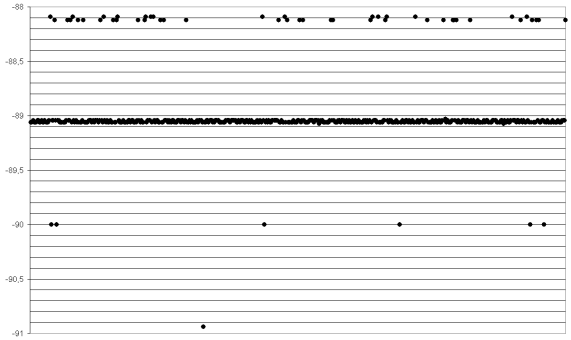
5.2.2 Accelerometer

Sergiy Krutykov

Pitch

Beim Accelerometer wurde nur Pitch untersucht, da Roll schwer zu messen ist. Auf einer verstellbaren Bank wurde Pitch flach und mit Neigungen um 10°, 20° und 30° gemessen. Jede Messreihe enthielt 500 Messungen, die einen zeitlichen Abstand von 1/30 einer Sekunde hatten. Die Werte des Accelerometers wurden dann in Winkel umgerechnet: Wenn das Gerät sich in der Porträt-Orientierung senkrecht zum Boden befindet, ist der Winkel 0° und wenn das Gerät flach auf einer horizontalen Oberfläche liegt, dann ist der Winkel -90°.

Bei dem flach liegenden Gerät hat sich die folgende Messreihe ergeben:

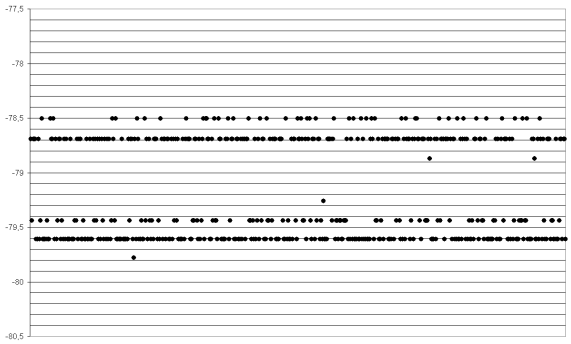


Man sieht, dass die Werte sich nicht beliebig verteilen, sondern aus einer kleinen Auswahl, insgesamt 8 Zahlen, bestehen. In der folgenden Tabelle ist die Häufigkeit jeder Zahl in Prozent ausgedrückt:

-89,060815°	49%
-89,045167°	39,6%
-88,122142°	6,4%
-88,090867°	3%
-90°	1,2%
-90,939178°	0,2%
-89,075964°	0,4%
-89,028986°	0,2%

Die Mehrheit der Werte spricht für ungefähr -89°. Dass dies nicht -90° ist (was bei einem flach liegenden Gerät erwartet wäre), kann daran liegen, dass die Bankoberfläche nicht ganz senkrecht zum Boden war. Eine positive Folgerung ist, dass bis auf den einzigen Ausreißer -90,939178° die Zahlen um weniger als plus-minus ein Grad schwanken.

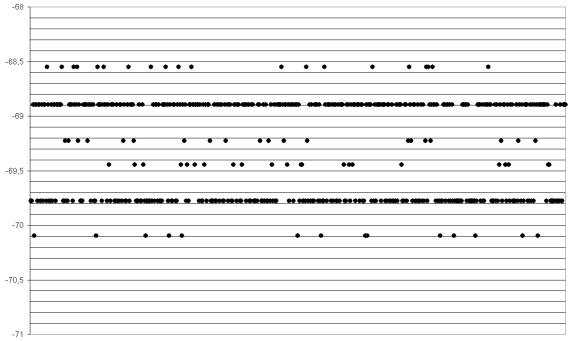
Bei dem 10°-Winkel (-80° beim Gerät) sind die Ergebnisse in dem Sinne schlechter, dass sich die Mehrheit der Zahlen in zwei Gruppen unterteilt hat, die einen Abstand von 1° haben:



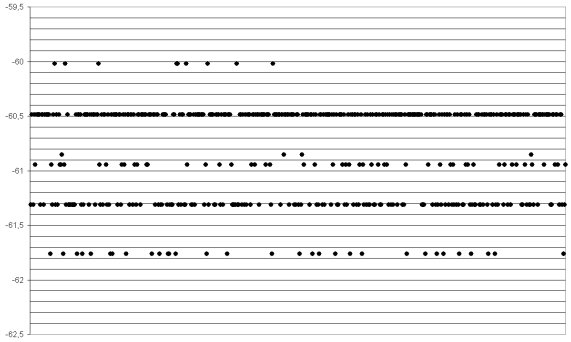
Die Tabelle macht das nur noch deutlicher:

-79,611139	38,8%
-78,690074	34,2%
-79,438991	16,8%
-78,503446	9,4%
-78,870815	0,4%
-79,77783	0,2%
-79,261106	0,2%

Bei dem 20°-Winkel (-70° beim Gerät) und dem 30°-Winkel (-60° beim Gerät) sind die Ergebnisse kaum schlechter:



-68,895153	45,8%
-69,775121	39%
-69,443938	4,4%
-68,552257	4%
-69,227749	4%
-70,096249	2,8%



-60,488529	53,8%

-61,313869	27,2%
-60,945421	10,2%
-61,762571	6,2%
-60,018366	1,8%
-60,851929	0,8%

Fazit

Die Ergebnisse sind relativ erfreulich, denn bis auf den oben genannten Ausreißer (einem unter 2000) sind die Schwankungen unter plus-minus 1 Grad. Ein Nachteil, welchen man in den oberen Diagrammen und Tabellen nicht sieht, ist dass die Zahlen sehr schnell "springen", zwar nicht mehr als um Grad, aber bei manchen Anwendungen könnte das störend sein. Allerdings kann man mit einigen "Glättung-Ansätzen" wie z.B. Mittelwertbildung gut dagegen wirken.

5.2.3 Compass

Sergiy Krutykov

Bemerkungen

Eine Besonderheit von iPhone ist, dass, wenn das Gerät unberührt flach liegt, dann ändern sich die Heading-Werte nicht (es sei denn man bewegt einen Magneten neben dem Gerät). Damit fällt bei iPhone die Berechnung der Schwankungen in der Ruhelage wie bei Accelerometer weg.

Noch zu beachten ist, dass Heading-Werte zwar Fließkommazahlen sind, steht bei jeder Messreihe immer das gleiche nach dem Komma, so dass die Daten eigentlich als diskret angesehen werden können.

Drehungen um bestimmte Winkel

Auf einem Blatt Papier wurden vier sich unter 45° kreuzende Linien gezeichnet. Mit der Hand wurde das Gerät um die Winkel, die das Vielfache von 45° sind, gedreht und beobachtet, welchen Drehwinkel das Gerät dabei registriert.

Für das flach liegende Gerät ergaben sich die folgenden Abweichungen:

Winkel	Abweichung
45°	+6°
90°	-1°
135°	-1°
180°	+2°
225°	-13°
270°	-22°
315°	-11°
360°	-3°

Die Abweichungen sind teilweise ziemlich stark, insbesondere wenn man berücksichtigt, dass die Genauigkeit, die vom Gerät angegeben wurde 15° betrug. Analog zu GPS bedeutet diese Genauigkeit, dass sich das Gerät nicht mehr als um 15° verfehlt. Diese Abweichungen können aber durch die Ungenauigkeit der Messung (mit der Hand) um ein paar Grad verstärkt (oder auch verringert) werden.

Für das senkrecht im Landscape-Modus gehaltene Gerät ergaben sich die folgenden Abweichungen:

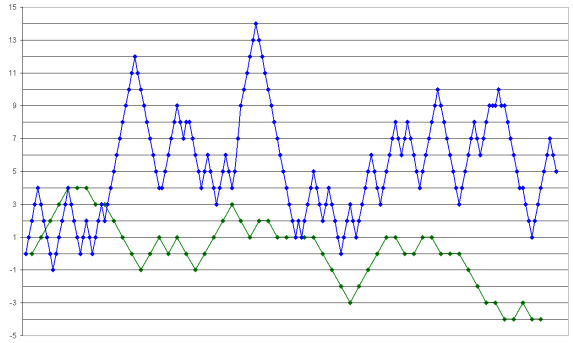
Winkel	Abweichung
45°	+16
90°	+29
135°	+30
180°	+22

225°	+9
270°	-1
315°	-3
360°	-9

Bei dieser Messungen dürften die Ungenauigkeit dieser "Handmessung" noch stärker sein. In diesem Fall war jedoch die Genauigkeit besser: 10°.

Messungen beim Gehen

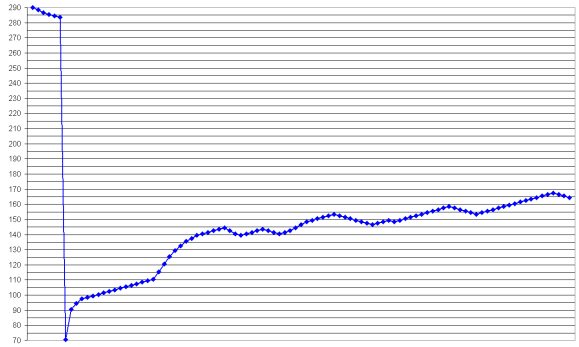
Es wurde eine gerade Strecke abgelaufen mit dem direkt nach vorne gerichteten Kompass und die Abweichungen wurden gemessen. Dabei wurde das Gerät einmal flach und einmal senkrecht zur Erdoberfläche gehalten:



Im Diagramm entsprechen die grünen Punkte dem flachen und die blauen dem senkrecht orientierten Gerät. Der Gang dauerte in beiden Fällen 35 Sekunden. Man sieht, dass beim senkrechten Gerät die Abweichungen größer sind. Außerdem hat das Gerät in diesem Fall auch mehr Werte (in der gleichen Zeit) geliefert: 60 Messungen beim flachen gehaltenen Gerät gegen 180 beim senkrecht gehaltenem. Die Genauigkeiten schwankten relativ gleichmäßig in beiden Fällen von 10° bis 30°.

Messungen beim Kippen

Auf einer Bank wurde das Gerät, welches 290° angab, ruckartig aus dem flachen Zustand in den senkrechten gekippt und die Heading-Werte gemessen:



Das sind 100 Werte, die innerhalb von 5 Sekunden gemessen wurden. Man sieht, dass die Kompass-Richtung überhaupt nicht mit der flachen Lage übereinstimmt, und dass der Kompass auch in Ruhe ständig Werte nachliefert. Die Genauigkeit war durchgehend 25°.

6. Karten

Sergiy Krutykov, Sebastian Stock

Laut der Spielidee orientiert sich der Spieler anhand einer Karte, auf der seine Position und weitere Spielinformationen angezeigt werden. Zur Darstellung der Karten gibt es eine Reihe von Services, die am meisten verbreiteten sind GoogleMaps[37] und OpenStreetMap[38]. Diese haben jeweils unterschiedliche Vor- und Nachteile. Da auf das Kartenmaterial bereits im Abschnitt OpenStreetMaps eingegangen wird, wird sich hier auf deren Darstellung auf den Smartphones beschränkt.

OpenStreetMap ist vor allem dadurch attraktiv, dass es sich dabei um freies Kartenmaterial handelt und man sich nicht von Lizenzbedingungen abhängig macht, wie dieses bei *GoogleMaps* der Fall ist. Andererseits hat *GoogleMaps* den Vorteil, dass es weit verbreitet ist und sowohl für iOS als auch für Android jeweils eine zuverlässige Bibliothek zur Anzeige des Kartenmaterials gibt. Für *OpenStreetMap* gibt es hingegen zwar einige OpenSource-Projekte, die sich damit beschäftigen, diese befinden sich allerdings noch in der Entwicklung und sind daher als etwas unzuverlässiger anzusehen. Zudem würde man sich von diesen Projekten abhängig machen. Ein weiterer entscheidender Vorteil von *GoogleMaps* ist dessen Satellitenansicht, die für ein Spiel deutlich passender und interessanter ist, als die normale Kartenansicht.

6.1 Android

Sebastian Stock

GoogleMaps-Karten können mit Hilfe der *GoogleMaps API*[39] eingebunden werden. Diese enthält die Klasse `com.google.android.maps.MapView`, die bei der Definition der Benutzeroberfläche wie gewöhnliche *Views* in der entsprechenden Layout-Datei platziert werden kann. Dort muss auch ein *GoogleMaps API-Key* eingetragen werden. Um diesen zu erhalten ist eine Registrierung notwendig, bei der auch der MD5-Fingerabdruck des Zertifikats angegeben werden muss, mit dem die Applikation später signiert wird. Ohne gültigen API-Key kann die Applikation zwar ausgeführt werden, die Karte wird jedoch nicht angezeigt.

6.1.1 MapActivity

Sebastian Stock

Immer wenn eine `MapView` verwendet wird, muss die betreffende *Activity* die Klasse `MapActivity` erweitern. Die `MapActivity`, welche wiederum von `Activity` abgeleitet ist, sorgt eigenständig für das Nachladen und Caching der Kartenausschnitte und ist daher zwingend erforderlich.

Mit Methoden der `MapView` können die Eigenschaften der dargestellten Karten festgelegt werden. So kann beispielsweise eine Satellitenansicht aktiviert oder deaktiviert werden und es können die Straßen mit dazugehörigen Namen eingezeichnet werden. Die Satellitenansicht hat zwar den Nachteil, dass das zu übertragende Datenvolumen größer ist, dennoch wurde diese Darstellung gewählt, da diese zu einem realistischeren Spielerlebnis führt und mehr Details der Umgebung sichtbar werden.

Der Benutzer möchte den Kartenausschnitt natürlich nicht immer in der gleichen Größe sehen, sondern ihn auch vergrößern und verkleinern können. Dafür hat die `MapView` unterschiedliche Zoomstufen im Bereich 1 bis 23, zwischen denen der Benutzer mit *Touch-Gesten* oder eingebauten *Zoom-Buttons* wechseln kann. Letztere können mit `MapView.setBuiltInZoomControls(true)` hinzugefügt werden. Natürlich kann die Zoomstufe aber auch im Programmcode gesetzt werden. Dafür gibt es in der *GoogleMaps API* die Klasse `com.google.android.maps.MapController`, die unter anderem auch eine Methode zur animierten Bewegung der Karte zu einer bestimmten Geo-Koordinate bereitstellt.

37 http://maps.google.de
38 http://www.openstreetmap.de
39 http://http://code.google.com/intl/de-DE/android/add-ons/google-apis

Von diesen Möglichkeiten wird in der folgenden Initialisierung der `MapView` Gebrauch gemacht. Die `MapView` wird dabei zur GPS-Position des Benutzers bewegt, welche im `PositionModel` gegeben ist:

```
MapView mapView = (MapView) findViewById(R.id.game_mapview);
MapController mapController = mapView.getController();

mapView.setBuiltInZoomControls(true);
mapView.setSatellite(true);
mapView.setStreetView(true);

// set Zoom Level
int maxZoomLevel = mapView.getMaxZoomLevel();
mapController.setZoom(maxZoomLevel - 3);

// Go to the Start Position
mapController.animateTo(positionModel.getPosition().asAndroidFormat());
```

6.1.2 Overlays

Sebastian Stock

Auf der Karte sollen die für den Spieler relevanten positionsbezogenen Informationen, also Tresore, Geschenke, andere Spieler, das Spielfeld mit seiner Unterteilung in einzelne Gebiete und natürlich auch der Benutzer selbst, angezeigt werden . Dies geschieht mittels sogenannter *Overlays*, welche von der Klasse `com.google.android.maps.Overlay` aus der *GoogleMaps API* abgeleitet und zu der `MapView` hinzugefügt werden. Für jede Art von Gegenstand wurde dabei ein eigenes *Overlay* implementiert. So gibt es beispielsweise ein `SafeOverlay` zur Darstellung der Tresore und ein `PlayerOverlay` zur Anzeige der anderen Spieler. Die `MapView` verfügt über eine Liste ihre *Overlays*. Diese erhält man mit `MapView.getOverlays()` und zu ihr können einfach die zusätzlichen *Overlays* hinzugefügt werden. Diese werden dabei als Ebenen übereinander angeordnet und können sich daher auch gegenseitig überdecken.

Anzeigen

Overlays haben eine `draw`-Methode, die bei jedem Zeichnen der `MapView` aufgerufen wird. Das ist beispielsweise beim Zoomen oder Bewegen der Karte der Fall. Die notwendigen Schritte zum Zeichnen sind für die meisten *Overlays* ähnlich: Gegeben ist eine Liste der zu zeichnenden Gegenstände, welche als Attribut auch ihre GPS-Koordinaten enthalten. Diese GPS-Koordinaten müssen jeweils in Bildschirm-Pixel umgerechnet werden, was mit Hilfe der Klasse `com.google.android.maps.Projection` aus der *GoogleMaps API* geschieht. So erhält man die Positionen auf dem Bildschirm, an denen die Gegenstände eingezeichnet werden müssen. Die `draw`-Methode verfügt über ein `Canvas`, auf der direkt einfache 2D-Objekte oder Bitmaps gezeichnet werden können. Für die eigene Position, die Tresore, die Geschenke und die anderen Spieler werden Icons im png-Format benutzt. Diese können allerdings nicht direkt eingezeichnet werden, sondern müssen an die aktuelle Zoomstufe angepasst werden. Andernfalls sähe man bei starkem Herauszoomen auf dem Spielfeld nur noch eine Ansammlung von Icons, aber nicht mehr die zugrunde liegende Karte oder beim Hineinzoomen wären die Icons zu klein. Daher werden sie im Konstruktor einmalig als Bitmap geladen und in der `draw`-Methode passend zur aktuellen Zoomstufe skaliert.

Der folgende Programmcode zeigt exemplarisch die `draw`-Methode des `PositionOverlay`. Dort wird zunächst die Größe des Bildes in Abhängigkeit von der Zoomstufe berechnet, wobei ein Minimal- und Maximalwert von 8 bzw. 128 Pixeln nicht überschritten werden sollte. Mit der `draw`-Methode des `Canvas` wird das Bild in ein Rechteck um den zur GPS-Koordinate korrespondierenden Bildschirmpunkt gezeichnet, wodurch gleichzeitig auch die Skalierung vorgenommen wird. Der Rückgabewert `false` bewirkt, dass das *Overlay* nur dann neu gezeichnet wird, wenn dieses durch erneutes Zeichnen der `MapView` auch wirklich notwendig ist. Mit `true` würde das *Overlay* ständig neu gezeichnet werden, was durch die große Anzahl dargestellter Objekte zu Geschwindigkeitseinbußen führen könnte.

```
public boolean draw(Canvas canvas, MapView mapView, boolean shadow, long when) {
        super.draw(canvas, mapView, shadow, when);
```

```
        int zoom = mapView.getZoomLevel();
        float size = (float) (Math.exp(0.015 * zoom * zoom - 0.185 * zoom + 1.64));
        if(size < 4.0) {
                size = 4.0f;
        } else if(size > 64.0) {
                size = 64.0f;
        }

        // Transform position to pixels
        Point screenPointPosition = new Point();
        GeoPoint currentPosition = positionModel.getPosition().asAndroidFormat();
        Projection projection = mapView.getProjection();
        projection.toPixels(currentPosition, screenPointPosition);
        // Rectangle to scale bitmap
        RectF rect = new RectF(screenPointPosition.x - size, screenPointPosition.y - size,
                               screenPointPosition.x + size, screenPointPosition.y + size);
        canvas.drawBitmap(bmpPlayer, null, rect, null);
        return false;
}
```

Sechs *Overlays* wurden implementiert:

- `PositionOverlay` : Zeichnet die Position des Spielers als grauen Punkt ein.
- `PlayerOverlay` : Zeigt alle anderen sichtbaren Spieler als Polizisten bzw. Diebe an.
- `SafeOverlay`: Zeigt die für den Benutzer sichtbaren Tresore an.
- `GiftOverlay` : Zeichnet die Geschenke ein.
- `FogOverlay` : Dieses *Overlay* zeichnet die Spielfeldbegrenzung als rote Linie ein und verdunkelt nicht sichtbare Gebiete des Spielfeldes, die der Benutzer erst noch aufdecken muss.
- `FakepositionTapOverlay` : Zeigt nichts auf der Karte an, sondern dient zu Demonstrationszwecken lediglich dazu, dem Benutzer bei deaktiviertem GPS-Sensor durch langes Drücken auf die Karte einen Wechsel zu der so ausgewählten Geo-Position zu ermöglichen.

In den folgenden Bildern der Kartenansicht lassen sich die *Overlays* und der Einfluss der Zoomstufen auf die Größe der gezeichneten Bilder gut erkennen:

Abb. 51: Overlays bei geringer Zoomstufe

Abb. 52: Overlays bei mittlerer Zoomstufe

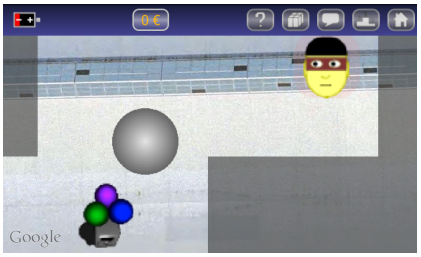


Abb. 53: Overlays bei hoher Zoomstufe

Wenn sich etwas an den zugrunde liegenden Daten in den Modellen etwas ändert, also beispielsweise neue Tresore hinzugekommen sind oder der Benutzer ein bisher verdecktes Gebiet betreten hat und dieses nun aufgedeckt wird, müssen auch die entsprechenden *Overlays* neu gezeichnet werden. Wie im Kapitel *Datenhaltung* beschrieben, wird das Observer-Observable-Pattern verwendet. Die `GameActivity`, welche auch die `MapView` enthält, implementiert dazu das Interface `Observer` und registriert sich an den einzelnen Modellen. Liegen in diesen Änderungen vor, wird die auf Basis des `Observer`-Interface implementierte `update`-Methode aufgerufen. Dort wird zunächst geprüft, wo die Änderungen vorlagen und anschließend die Daten der relevanten *Overlays* aktualisiert und die `MapView` mit deren `postInvalidate()`-Methode zum erneuten Zeichnen veranlasst.

Tap Events

Wenn der Benutzer auf die angezeigten Objekte klickt werden noch weitere Informationen zu diesen angezeigt. Dies kann in *Overlays* durch Überschreiben der Methode `boolean onTap(GeoPoint p, MapView mapView)` erreicht werden. Die *Overlays* sind als Ebenen über der `MapView` angeordnet. Bei jedem auf der `MapView` ausgelösten *Tap Event* werden, beginnend mit dem obersten *Overlay*, deren `onTap`-Methoden aufgerufen, bis dieses Event von einem der *Overlays* behandelt wird, was durch den Rückgabewert `true` signalisiert wird. Die *Overlays* müssen dafür in der `onTap`-Methode zunächst ermitteln ob dieses *Tap Event* für sie relevant ist und behandelt werden kann. Dafür ist der übergebene `GeoPoint` nützlich. Der `GeoPoint` repräsentiert die bereits in Geo-Koordinaten umgerechnete Position derjenigen Stelle auf dem Bildschirm, an der der Benutzer den *Tap Event* ausgelöst hat. `onTap` wurde im `PositionOverlay`, `PlayerOverlay`, `SafeOverlay` und `GiftOverlay` überschrieben werden. In allen vier Fällen wird die Distanz des übergebenen `GeoPoints` zu den Geo-Koordinaten der den *Overlays* zugrunde liegenden Objekte berechnet. Ist diese Distanz hinreichend gering, werden je nach *Overlay* nähere Informationen zu dem betreffenden Objekt angezeigt und der *Tap Event* somit behandelt. Andernfalls wird `false` zurückgegeben und der *Tap Event* an die nächste Ebene weitergereicht. Beim `PositionOverlay` muss lediglich mit der Geo-Position des Benutzers verglichen und gegebenenfalls dessen *Längen-* und *Breitengrad* ausgeben werden. Beim `PlayerOverlay`, `SafeOverlay` und `GiftOverlay` muss hingegen jeweils eine ganze Liste von Objekten durchlaufen und gegebenenfalls dasjenige mit dem geringsten Abstand verwendet werden. Hier ist zu beachten, dass nur diejenigen Objekte berücksichtigt werden, die auch für den Benutzer sichtbar sind. Ansonsten könnte man wahllos auf die Karte klicken um zu schauen, ob sich dort ein Gegenstand befindet. Für Tresore wird dessen Wert und die direkte Distanz des Benutzers zu diesem ausgegeben, für Geschenke hingegen nur die Distanz, da nicht verraten werden soll welcher Gegenstand sich in diesem befindet. Beim Anklicken eines anderen Spielers wird schließlich dessen Name und Punktanzahl angezeigt. Die Anzeige geschieht mit einem `Toast`. Dies ist auf Android ein kleines Textfeld, das für eine kurze Zeit eingeblendet wird und automatisch wieder verschwindet. Auf den folgenden beiden Bildern wird auf der Karte jeweils nach dem Anklicken eines Tresors bzw. eines Spielers ein `Toast` angezeigt:

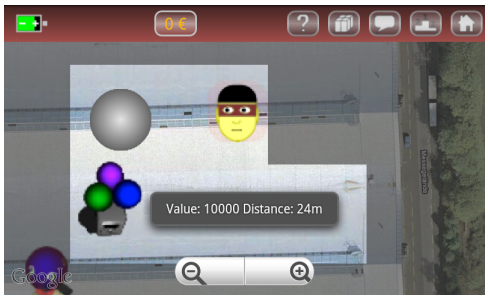


Abb. 54: Nach dem Anklicken eines Tresors angezeigter `Toast`, mit dem Wert des Tresors und der Distanz des Benutzers zu diesem.

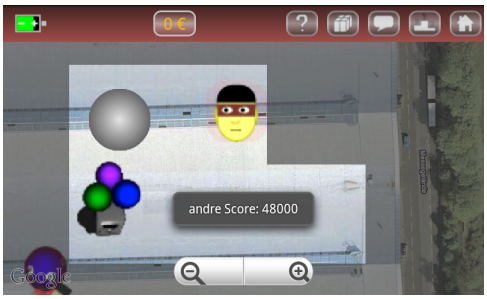


Abb. 55: Nach dem Anklicken eines Spielers angezeigter `Toast`, mit dem Namen des Spielers und dessen Punktanzahl

Ein Sonderfall ist das `FakepositionTapOverlay`, welches nicht dazu dient Gegenstände auf der Karte anzuzeigen. Stattdessen kann der Spieler, wenn der GPS-Sensor deaktiviert ist, mit diesem seine Position im Spiel durch einen langen Druck auf der Karte zu einer gewünschten Position bewegen. Diese Funktion wurde allerdings nur für Demonstrationszwecke vorgesehen und sollte vor der Auslieferung des Spiels an Kunden deaktiviert werden. Während normale *Tap Events* bereits eigenständig von der `MapView` erkannt werden und für die Behandlung lediglich die `onTap`-Methode in den *Overlays* überschrieben werden muss, gestaltet sich das Erkennen eines langen Drückens etwas schwieriger. Es muss zunächst die Methode `boolean onTouchEvent(android.view.MotionEvent event, MapView mapView)` überschrieben werden, welche bei jeder Berührung der `MapView` aufgerufen wird. Eine Berührung des Bildschirms löst eine Reihe von *MotionEvents* aus. So sind das Herunterdrücken, Bewegen und Hochnehmen des Fingers einzelne oder im Falle von Bewegungen sogar mehrere *MotionEvents*. Ein Objekt der Klasse `MotionEvent` enthält unter anderem Informationen zur Art der Aktion, wie z.B. `MotionEvent.ACTION_DOWN` oder `MotionEvent.ACTION_MOVE`, und an welcher Position am Bildschirm der *MotionEvent* ausgelöst wurde.

Gesten sind bestimmte Sequenzen von *MotionEvents*. Hierunter fällt auch das lange Drücken, welches erkannt werden soll. Einfache *Gesten* können mit einem `GestureDetector` erkannt werden. Um auf die erkannten *Gesten* zu reagieren, muss diesen im Konstruktor ein *Listener* übergeben werden, der das Interface `GestureDetector.OnGestureListener` implementiert. Da nur ein langes Drücken erkannt werden soll

genügt es in diesem Falle die Klasse `GestureDetector.SimpleOnGestureListener` zu erweitern und die Methode `onLongPress(MotionEvent e)` zu überschreiben. In dieser Methode wird der Benutzer nun gefragt, ob er sich zu der gewählten Geo-Position teleportieren möchte, wodurch ein unbeabsichtigter Positionswechsel verhindert wird. Schließlich wird die neue Position im `PositionModel` gesetzt und an den Server gesendet.

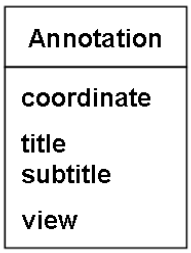
6.2 iPhone

Sergiy Krutykov

Die Anzeige der Karten auf iPhone ist grundsätzlich sehr einfach: Man muss lediglich eigene Klasse von einer speziellen Klasse (`MKMapView`) ableiten. Wenn man dann die Satellitenansicht aktiviert und die Karte zu dem Landscape um -3/2 PI dreht, dann bekommt man bereits die Ansicht, wie in der Abbildung:

Annotationen

Zum Anzeigen verschiedener Gegenstände auf der Karte können spezielle Objekte, die so genannten Annotationen verwendet werden. Die von Google Maps API bekannten Stecknadeln, sind ein Beispiel für solche Annotationen. Der Umgang mit Annotationen auf iPhone ist etwas umständlich. Als eine Annotation gilt jedes Objekt einer Klasse, die das Protokoll `MKAnnotation` adoptiert. Laut diesem Protokoll besteht eine Annotation aus einer Koordinate (in Längen- und Breitengrad), einem Titel und einem Untertitel. Darüber hinaus beinhaltet sie noch als Attribut eine spezielle `UIView` die `MKMapAnnotationView`, die ihr beim Hinzufügen zu einer Karte von dieser Karte zugewiesen wird:



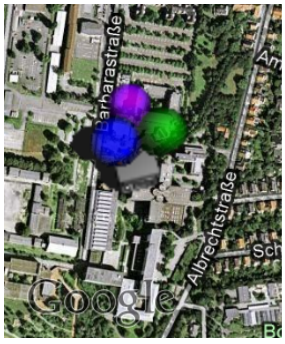
Es gibt Möglichkeit eigene Views für Annotationen zu schreiben. Es gibt aber eine bereits implementierte Klasse, `MKPinAnnotationView`, die nämlich die oben erwähnte berühmte Stecknadel repräsentiert. Dabei kann eine solche Stecknadel eine der drei Farben, rot, grün oder violett haben. In der folgenden Abbildung sind beispielhaft drei Nadeln in unterschiedlicher Farbe dargestellt:

In der Abbildung sieht man, wofür die Attribute Titel und Untertitel stehen: Das sind die Aufschriften in der Sprechblase der entsprechenden Annotation (der Titel ist "Pin" und der Untertitel ist "Ich bin rot.").

Das Platzieren von einer Annotation auf der Karte geschieht nach dem folgenden Szenario: Nachdem eine Annotation erzeugt wurde, werden ihre Koordinaten mit gewünschten Werten gesetzt und zu einer Karte (also einer Unterklasse von `MKMapView`) mit dem Aufruf von `addAnnotation:` auf dieser Karte hinzugefügt. Kurz danach (kann aber durchaus eine Sekunde dauern) wird die Instanzmethode dieser Karte

```
- (MKAnnotationView *) mapView:(MKMapView *)mapView
                      viewForAnnotation:(id <MKAnnotation>)mapAnnotation
```

automatisch aufgerufen, die eine spezielle `UIView`, nämlich `MKAnnotationView` zurück liefert (das Argument der Methode `mapView` ist dabei die Karte, auf welcher sich alles ereignet, und `mapAnnotation` ist die hinzugefügte Annotation). Man muss also innerhalb von dieser Methode die View zurück liefern, die das Aussehen der Annotation bestimmen soll. Wenn man für diese View eine Instanz von `MKPinAnnotationView` benutzt, dann bekommt die Annotation das Aussehen einer Stecknadel wie oben. Alternativ kann man eine Instanz von `MKAnnotationView` direkt benutzen. Allerdings ist diese View initial leer. Sie hat aber ein Image als Instanzvariable, welchem Pixelgrafiken zugewiesen werden können, wie in den folgenden zwei Abbildungen dargestellt ist.



Skalierung der Bitmap-Annotationen

Wie aus den obigen Abbildungen ersichtlich ist, wird die Größe der Annotation nicht automatisch an die Zoomstufe angepasst, sondern bleibt immer konstant, was optisch (und gerade für ein Spiel) sehr unpassend ist. Um dies zu ändern, muss man das Image der Annotation-View selbst skalieren. Dies geschieht dadurch, dass das Image in einen graphischen Kontext gezeichnet wird, der eine andere Größe hat und dann der Inhalt dieses Kontextes wieder zu einem RGBA-Bild gerendert wird. Danach reicht es, einfach das alte Image der View durch das neue zu ersetzen. Diese Skalierung und Ersetzung der View kann man einfach mit Hilfe von einem Timer in regelmäßigen kurzen Abständen immer wieder wiederholen, damit die Größe der Annotation zu der aktuellen Zoomstufe immer passt. Aber die Skalierung ist ein aufwändiger Prozess, deshalb ist eine Lösung ziemlich unpraktikabel. Viel bessere Lösung wäre, die Größe *nur* dann zu ändern, wenn die Zoomstufe sich ändert. Abhilfe schafft hier eine Listener-Methode der Mapview, die gerade dann aufgerufen wird, wenn das Zoom sich ändert. Die Mapview hat eine Auswahl an solchen Methoden: Die Methode `mapView:viewForAnnotation:` von oben ist auch von dieser Art. In Bezug auf das Zoomen werden vor allem die beiden Methoden aufgerufen:

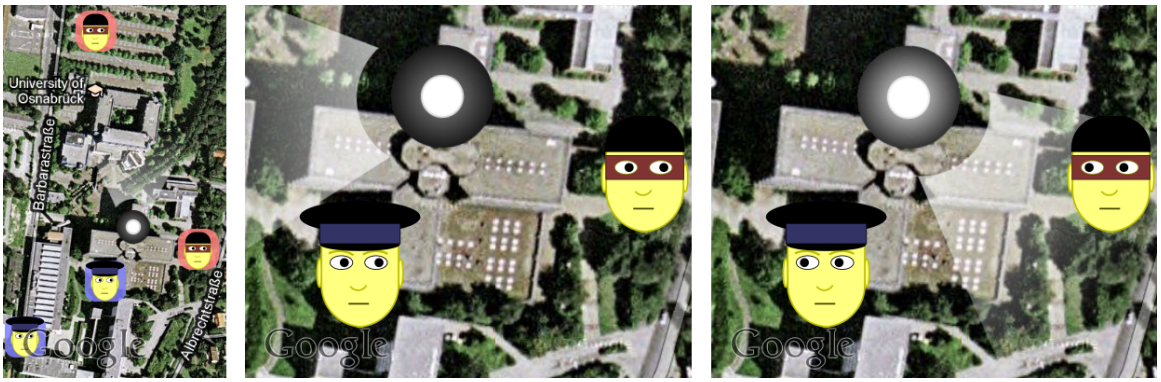
```
- (void)mapView:(MKMapView *)mapView regionWillChangeAnimated:(BOOL)animated
- (void)mapView:(MKMapView *)mapView regionDidChangeAnimated:(BOOL)animated
```

`mapView:regionWillChangeAnimated:` wird aufgerufen kurz bevor die Region (also die Größe und/oder die Position des auf dem Bildschirm sichtbaren Kartenausschnitts) sich ändert und `mapView:regionDidChangeAnimated:` direkt danach. Damit ist garantiert, dass, wenn sich die Zoomstufe ändert, die zweite Methode aufgerufen wird und dass sie aber nicht permanent ausgerufen wird, sondern nur wenn der Benutzer die Karte scrollt oder zoomt. Es bleibt dann nur zu bestimmen, auf welche Weise die Größe der Region die Skalierung der Annotation-View beeinflussen soll. Dafür gibt es keine fest definierte Formel. Was die Karte diesbezüglich zur Verfügung stellt, ist eben nur diese Region mit ihrer Position und der Größe. Die Größe wird dabei in Längen- und Breitengrad angegeben. Da die Länge sich von Polen bis hin zum Äquator sehr stark ändert, bietet sich an, die Breite zum Bestimmen der Zoomstufe zu benutzen. Man kann zum Beispiel die Breite eine Region, bei welcher die Annotaiton-View ohne Skalierung von der Größe her passt, als Ausgangslänge definieren, und dann jedes Mal die neue Breite durch diese fest definierte teilen, um den Skalierungsfaktor zu bestimmen. Auf diese Weise bekommt man das Ergebnis wie in der folgenden Abbildung:

Bei der Skalierung ist es wichtig darauf zu achten, dass wenn die Zoomstufe zu klein ist, dann werden die Annotationen entsprechend viel zu klein und verschwindet eventuell gänzlich. Dafür muss man die minimale und maximale Größe definieren, die nicht überschritten werden dürfen.

Annotationen als Vektorgrafik

Die oben beschriebene Skalierung der Images hat die Nachteile, dass sie immer noch nicht ganz effizient ist und auch starke Pixeleffekte aufweist. Da aber die Views der Annotationen auch `UIView` sind, kann man sie mit Hilfe von Vektorgrafik zeichnen. Dies ermöglicht nicht nur verlustfreie Skalierung der Views sondern auch das Animieren von diesen. In den folgenden drei Abbildungen sind einige Vorteile der Vektorgrafik beispielhaft vorgestellt:

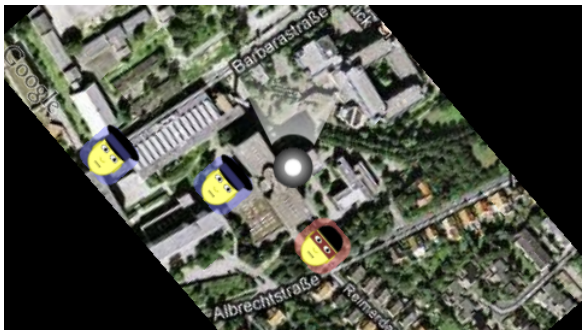


Beim Vergleich des linken Bildes mit den beiden anderen, sieht man, dass die Skalierung verlustfrei verläuft. Und der Vergleich des mittleren Bildes mit dem rechten aufweist, dass die Polizisten und die Diebe ihre Augen bewegen, und dass die Scheibe, die den Spieler selbst darstellen soll, leuchtend blinkt. Während die Bewegung der Augen nur dazu dient, die Applikation "lebhafter" zu gestalten, ist das blinkende Leuchten hilfreich, um eine sonst unauffällige Annotation auf der Karte besser erkennbar zu machen. Aus dem gleichen Grund erscheint hinter den Köpfen der Polizisten und Diebe ein farbliches halbdurchsichtiges abgerundetes Rechteck, welches um so weniger transparent wird, je kleiner die Annotationen werden. Beim Vergleich des mittleren Bildes mit

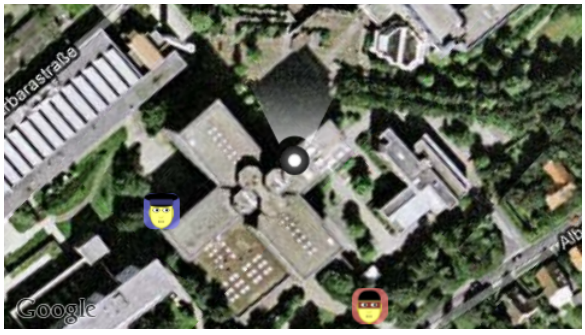
dem rechten sieht man sonst das der Licht-Keil, der den Blickwinkel des Spielers (gemäß der Richtung des Kompasses) symbolisieren soll, drehbar ist, womit nicht nur Position des Spielers sondern auch seine Orientierung visualisiert wird.

Automatische Drehung der Karte

Die Orientierung des Spielers kann man nicht nur, wie oben beschrieben, mit einem Keil visualisieren, sondern man kann auch die Karte selbst drehen, so dass die Orientierung der Karte genau zu der Umgebung passt. Offensichtlich macht dieses Vorgehen nur dann Sinn, wenn der Spieler sich exakt in dem Zentrum der Karte befindet. Die Karte kann man als jede `UIView` beliebig transformieren, zum Beispiel um ihren Mittelpunkt drehen. Allerdings reicht das einfache Drehen nicht aus, wie die folgende Abbildung zeigt:



Damit man die schwarzen Ränder nicht sieht, muss man zu einem Trick greifen, nämlich die Karte so vergrößern, dass sie auch beim Drehen immer noch den ganzen Bildschirm ausfüllt. Problematisch ist dann nur der Google-Logo, der dann am Rande dieser vergrößerten Karte bleibt und nicht mehr sichtbar wird. Dies ist natürlich ein Urheberrecht-Verstoß, der möglichst umzugehen ist. Da hilft noch ein Trick. Es stellt sich heraus, dass der Google-Logo einfach eine (und dabei die einzige) Subview auf der Karte ist, so dass man diese einfach manipulieren kann. Dafür muss man nur das Array von Subviews `subvies` von der Karte mit Hilfe von einer for-each-Schleife durchgehen, und bereits den Verweis auf die erste Subview (die gerade der Logo ist) in eine Instanzvariable speichern. Dieser Logo ist dann auf eine passende Stelle zu platzieren. Man sollte dies allerdings nicht auf der Karte selbst machen, da sie ja immer wieder gedreht wird und dadurch der Logo sich immer wieder verschieben würde, sondern auf eine Parent-View wie zum Beispiel auf der View des Viewcontrollers, der diese Karte verwaltet. Es bleibt dann nur noch eine Kleinigkeit: Man sieht auf der Abbildung, dass die Annotationen mit Dieben und Polizisten auch mitgedreht werden. Deshalb muss man diese um den gleichen Winkel in die umgekehrte Richtung (in Bezug auf die Drehung der Karte) drehen, damit sie vertikal ausgerichtet sind, wie in der folgenden Abbildung:

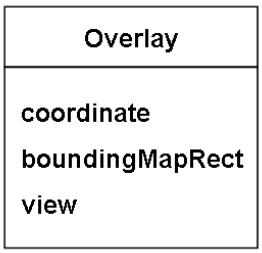


Im Vergleich zu den oberen Abbildungen sieht man, dass die Karte zwar gedreht ist, aber es keine schwarzen Ränder gibt, Google-Logo an dem richtigen Platz ist und die Bilder von Polizisten und Dieben vertikal ausgerichtet sind.

Overlays

Außer einzelne Gegenstände auf der Karte zu platzieren, besteht noch die Möglichkeit, eine Fläche auf der Karte auf eine besondere Weise zu färben, zum Beispiel, wenn man analog zu den Strategiespielen einen "Fog

of War" über einen Bereich der Karte legen will. Dies kann man auch mit Hilfe von einer (dann aber ziemlich großen) Annotation machen, indem man diese in den Mittelpunkt des Bereichs legt. Man muss dann die Größe der Annotation zu der Zoomstufe sehr genau anpassen, was sehr umständlich ist. Abhilfe schaffen hier die Overlays, die zwar zu Annotationen sehr ähnlich sind, aber für diese Probleme besser geeignet sind. Analog zu Annotationen gelten als Overlays alle Objekte, die das Protokoll `MKOverlay` adoptieren, laut welchem ein Overlay wie auch jede Annotation eine Koordinate aber keinen Titel oder Untertitel haben muss, dafür aber das Attribut `boundingMapRect`, welches die Position und die Größe des Overlays aber nicht in Längen- und Breitengrad sondern in einem speziellen 2D-projizierten Koordinatensystem beschreibt:



Das Hinzufügen eines Overlays geschieht nach dem gleichen Schema wie auch bei Annotationen: Nach dem Aufruf von der Instanzmethode `addOverlay:` auf einem Kartenobjekt wird in kurzer Zeit die folgende Methode aufgerufen:

```
(MKOverlayView *)mapView:(MKMapView *)mapView viewForOverlay:(id <MKOverlay>)overlay
```

In dieser Methode muss man analog zu `mapView:viewForAnnotation` dem Overlay eine View, die eine Unterklasse von `MKOverlayView` ist, zuweisen. Im Gegensatz zu Annotationen beinhalten allerdings Overlayviews keine Images, so dass man keine Pixelgrafiken mit einem Overlay verknüpfen kann und man *muss* mit der Vektorgrafik arbeiten. Allerdings wird dabei nicht die übliche Methode `drawRect:` (siehe Unterabschnitt Vektorgrafik in dem Abschnitt Smartphones) benutzt, sondern in einer von `MKOverlayView` abgeleiteten Klasse die Methode

```
- (void)drawMapRect:(MKMapRect)mapRect zoomScale:(MKZoomScale)zoomScale inContext:(CGContextRef)context
```

implementiert. Diese Methode wird ebenfalls automatisch aufgerufen, sobald, die View die Aufforderung, sich zu erneuern, bekommt. Dies passiert normalerweise beim Zoomen und Scrollen auf der Karte. Im Gegensatz zu `drawRect:` bekommt sie offensichtlich als Parameter die Zeichnungsfläche nicht in Bildschirm-Koordinaten sondern in speziell projizierten Kartenkoordinaten und auch einen grafischen Kontext, welchen man zum Zeichnen auch benutzen muss. In diesem Kontext kann man alle Zeichnungen wie etwa in dem Unterabschnitt Vektorgrafik machen, wie die folgende Abbildung zeigt:

In der linken Abbildung wird einfach die Zeichnungsroutine eines Toggle-Buttons benutzt, wobei sich das Bild natürlich an alle Zoomänderungen und Kartenverschiebungen automatisch anpasst. Rechts wird der "Fog of War" gezeichnet, der aus vielen abgedunkelten Plättchen besteht und einer etwas dunkleren Umrandung des

Gebiets. Wenn die Plättchen ihre Koordinaten und ihre Größe in Längen- und Breitengrad haben, dann müssen diese noch in die oben genannte Kartenkoordinaten und diese dann in die Bildschirmkoordinaten umgewandelt werden. Dabei ist der erste Schritt sehr aufwendig, weshalb, solange sich die Plättchen nicht permanent ändern, diese Berechnung vorab passieren muss und danach schon die konvertierten Kartenkoordinaten in der Routine `drawMapRect:zoomScale:inContext:` selbst verwendet werden sollen.

Beschränkungen der Kartenanzeige auf die vorgegebene Fläche

Da häufig, wie zum Beispiel bei einem Navigationsspiel, die Spielfläche begrenzt ist, der Benutzer aber dir Möglichkeit hat, beliebig zu zoomen und zu scrollen (dies kann auch ungewollt durch versehentliches Berühren des Bildschirms geschehen), so dass er das Spielgebiet aus den Augen verlieren kann, wenn er dies zu weit treibt, stellt sich die Frage, wie man diese Aktionen von Benutzer kontrollieren und gegebenenfalls einschränken könnte. Als erste Hilfe gibt es die Möglichkeit der Karte Gesten-Listener zuzuweisen, so dass zum Beispiel per Doppeltap die Karte auf dem Spieler zentriert wird. Damit kann der Benutzer auch bei viel zu wildem Scrollen immer noch zu seiner eigenen Position schnell springen. Allerdings ist die beste Lösung, dem Benutzer ganz zu verbieten, zu weit zu scrollen oder zu zoomen. Gerade in Bezug auf das Zoomen, könnte damit das Problem vermieden werden, dass bei Google Maps in der Satellitenansicht bei zu großen Zoomstufen manchmal anstatt von Satellitenfotos einfach nur grauer Hintergrund mit den Aufschriften "No Images" darauf erscheint, was für manche Applikationen, vor allem Spiele, inakzeptabel ist:

Die Gesten des Benutzers für das Zoomen und Scrollen werden durch die Karte selbst verarbeitet und man kann auf sie kaum Einfluss nehmen. Allerdings könnten hier die oben genannten Methode behilflich sein:

```
- (void)mapView:(MKMapView *)mapView regionWillChangeAnimated:(BOOL)animated
- (void)mapView:(MKMapView *)mapView regionDidChangeAnimated:(BOOL)animated
```

Die Idee ist dabei, dass man in der Methode `mapView:regionWillChangeAnimated:`, also noch bevor die Änderung der Ansicht passiert, die Parameter der Region zwischenspeichert und danach in `mapView:regionDidChangeAnimated:` zuerst überprüft, ob die neue Region noch zulässig ist und wenn ja, dann wird nichts gemacht, sonst wird die Region der Karte auf den alten Wert zurückgesetzt.

Diese Methode hat den Nachteil, dass der Benutzer beim Scrollen keine zu langen Züge macht, denn die Methode `mapView:regionDidChangeAnimated:` wird aufgerufen, nur wenn der Benutzer den Finger loslässt und wenn er durch zu langes Ziehen in eine nicht erlaubte Region "gerät", dann springt die Ansicht zu der Ausgangsregion, so dass der Benutzer praktisch an der gleichen Stelle bleibt. Das gleiche Problem hat man auch bei Zoomen: Es besteht nach wie vor die Möglichkeit, dass der graue Hintergrund mit den Aufschriften "No Images" zwischendurch erscheint und erst bei Loslassen der Finger wieder verschwindet.

Gegen diese Probleme kann man nur etwas unternehmen, wenn man die Gesten selbst verarbeitet. Dafür muss man allerdings in erster Linie in der Karte das Zoomen und Scrollen ausschalten. Dann muss man in den entsprechenden Listener-Methoden analog zu dem oben Beschriebenen überprüfen, ob die neue Region noch passt. Schwierig ist dabei diese neue Region zu bestimmen. Währen bei der "pinch-to-zoom" Geste der Skalierungsfaktor zur Verfügung steht, welcher dazu benutzt werden kann, in Verbindung mit der alter Zoomstufe die neue Zoomstufe zu berechnen, muss man beim Scrollen die neue Region anhand der Bewegung des Fingers auf dem Bildschirm bestimmen, wobei die Bildschirmkoordinaten zuerst zu den geographischen und dann zu den Kartenkoordinaten konvertiert werden müssen.

7. AR

Sergiy Krutykov, Peer Wagner

Zur Darstellung der Gegenstände als 3D-Objekte im Raum (siehe Spielidee) wurde der Prinzip von Augmented Reality (AR) benutzt.

Augmented Reality ist die Überlagerung der realen Welt mit virtuellen Informationen, welche die reale Welt erweitern und damit direkt mit der realen Welt zu interagieren scheinen. Auch wenn der Begriff der Augmented Reality oft nur mit der visuellen Darstellung von Objekten assoziiert wird ist damit in seinen Grundzügen im allgemeinen die Anreicherung aller Sinneswahrnemungen mit künstlichen Reizen gemeint.

Die AR Szene in dem Spiel besteht aus einem Kamera-Bild mit darauf projizierten 3D-Objekten. Dabei wird das Kamera-Bild von dem System als Array von Pixeln bereitgestellt und man muss diese noch rendern. Die 3D-Objekte müssen mit einer entsprechenden Bibliothek gezeichnet werden. Als Standrad-Bibliothek für 3D-Grafik steht auf beiden Plattformen (iOS und Android) "OpenGL ES" zur Verfügung. Damit der Code um einige spezielle Effekte erweiterbar bleibt, wurde beschlossen die aktuell neuste OpenGL Version zu nehmen, nämlich "OpenGL ES 2.0". Darüber hinaus bat sich an, den 3D-Teil des Programms (3D-Engine) in C zu schreiben, damit man diesen Code sowohl für iPhones als auch für Android-Smartphones praktisch ohne Anpassungen benutzt, um Redundanz an dieser Stelle zu vermeiden.

7.1 3D-Engine

Sergiy Krutykov

Die C-Bibliotheck, die zum Zeichnen der 3D-Gegenstände in Augmented Reality benutzt werden soll, beinhaltet nicht nur die Routinen zum Zeichnen sondern auch die Werkzeuge zum Parsen, Verwaltung von Objekten, Behandlung von Touch Events und sogar eine simple Logik für Minispiele. Jedoch sind die Methoden, die zum Zeichnen benutzt werden, zentral für sie und ausschließlich diese werden im folgenden diskutiert.

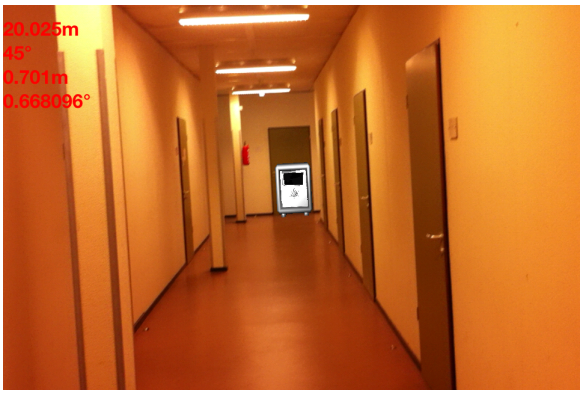
7.1.1 Integration der 3D-Objekte in das Kamerabild

Sergiy Krutykov

Eine der wichtigsten Aufgaben der Engine ist, die 3D-Gegenstände in OpenGL maßstabsgetreu und an richtiger Stelle zu zeichnen. Dafür stehen die Orientierung des Smartphones als Pitch-, Roll- und Kompass-Winkel in Grad zur Verfügung.

Maßstabsgetreue Darstellung der 3D-Objekte

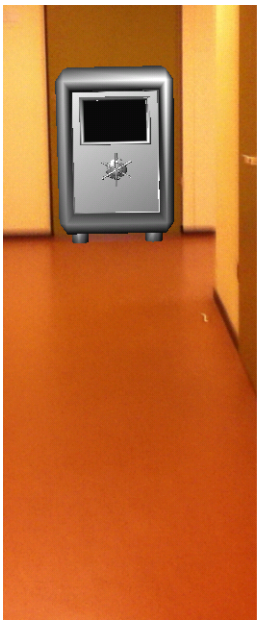
Damit man von "maßstabsgetreu" sprechen kann, muss man zuerst einen Maßstab definieren. Es liegt nah, die Einheit von OpenGL gleich einem Meter zu definieren. Demnach sollte z.B. ein virtueller Würfel mit der Kantenlänge 1.0 in OpenGL in der Entfernung 20.0 genauso groß erscheinen wie ein reeller Würfel mit der Kantenlänge 1m in der Entfernung 20m. Dass die 3D-Objekte in OpenGL in der (virtuellen) Entfernung kleiner werden wird dadurch beeinflusst, dass sie mit einer Projektionsmatrix multipliziert werden. Genauer gesagt beeinflusst ein Faktor dieser Matrix, der Blickwinkel der Projektion (Field of View). Die Änderung des Abstand eines 3D-Objekts geschieht dadurch, dass seine Matrix mit einer Translationsmatrix multipliziert wird. In den beiden folgenden Abbildung sind die Screenshots von einem Test-Programm auf iPhone, welches unter anderem auch dafür geschrieben wurde, um den Blickwinkel zu "justieren":

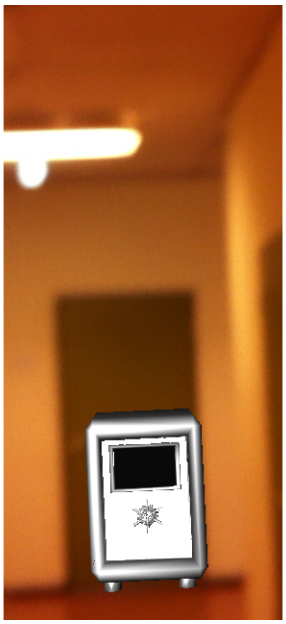


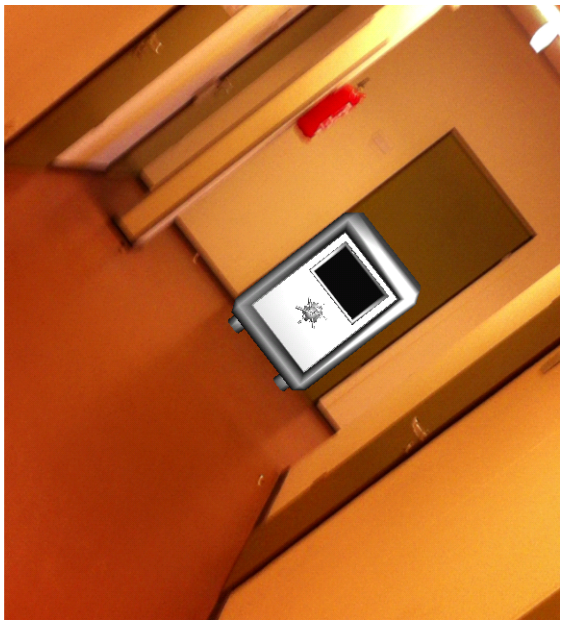
Auf beiden Bildern ist der gleiche Tresor in unterschiedlichen Abständen zu dem Betrachter zu sehen. Der Tresor soll die gleiche Breite wie die Tür hinter ihm haben. Im linken Bild hat der Tresor sowie die Tür hinter ihm den Abstand von 20 Metern zu dem Betrachter, in der rechten 10 Meter. Die Darstellung scheint (im Vergleich mit der Tür) maßstabsgetreu zu sein: Der Blickwinkel der Projektionsmatrix wurde auf 45° gesetzt. Man muss beachten, dass dieser Blickwinkel von der Auflösung der Kamera abhängt, so dass für andere Geräte eventuell nachjustiert werden muss.

Anpassungen an die Pitch-, Roll- und Kompass-Winkel

Bei den Änderungen der Orientierung des Smartphones müssen die 3D-Objekte in OpenGL so verschoben werden, dass ihre Bewegung die Bewegung des Geräts ausgleicht. Bei jeder Drehung des Smartphones muss also die ganze Szene in entgegengesetzte Richtung gedreht werden. Dafür muss die Projektionsmatrix mit drei Drehmatrizen, jeweils um *x*-, *y*- oder *z*-Achse nacheinander multipliziert werden. Man muss nur beachten, dass ein Objekt zuerst verschoben werden muss und danach mit dieser Produktmatrix multipliziert werden muss, sonst dreht es sich um eigene Achse und nicht um den Betrachter. Die Reaktion auf die Änderungen der Pitch- und Roll-Winkel erscheint bei dem Testprogramm auch ziemlich passend zu sein:







Auf dem linken und dem mittleren Bild sieht man, dass der Tresor sich dem Kippen des Geräts "anpasst". Das rechte Bild ist dadurch entstanden, dass das Gerät zur Seite geneigt wurde, wobei der Tresor

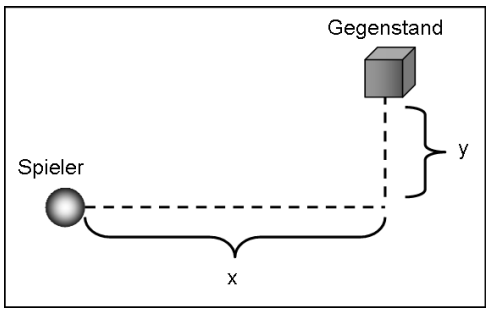
Allerdings macht sich die Ungenauigkeit des Accelerometers (und damit der Pitch- und Roll-Winkel) insbesondere für die weit entfernten Gegenstände bemerkbar, denn die Verschiebung des in 100 Metern liegenden Gegenstandes um 1° ergibt visuelle Erhöhung oder Erniedrigung von ihm um 2m: Wenn man zum Beispiel einen virtuellen Torwart an der anderen Seite des Stadions in einem Tor platzieren möchte, dann schwankt

dieser ab und zu über dem Tor. Und wie im Abschnitt Sensoren steht, ist diese Schwankung um 1° vorprogrammiert.

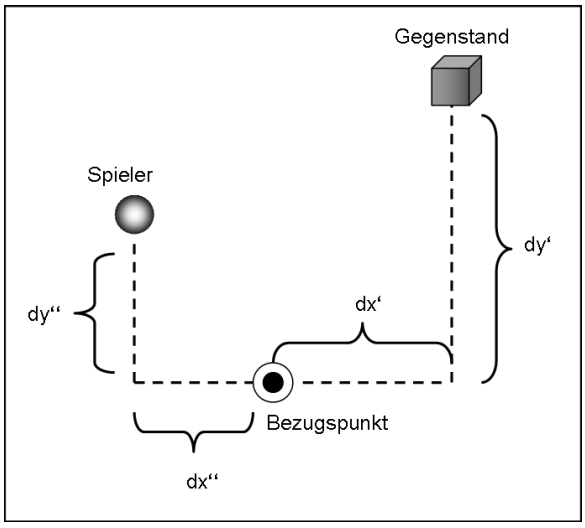
Viel zu große Ungenauigkeit des Kompasses (siehe Abschnitt Sensoren) ruiniert aber die Idee, einen virtuellen Gegenstand so zu platzieren, dass er wie "eingegossen" zu der Umgebung passt, komplett. Daher wurde im Laufe des Projekt entschieden, die Ungenauigkeiten der Sensoren dadurch auszugleichen, dass die Gegenstände auf Luftballons in der Luft schweben, womit die Schwankungen in der Kamera vertuscht werden.

Koordinaten der Objekte in OpenGL

Die Position der Objekte in der Engine wird durch zweidimensionale Koordinaten beschrieben: (*x*, *y*). Dabei ist geht die *x*-Achse von links (-) nach rechts(+) und die *y*-Achse von hinten (-) nach vorne (+). Wenn der Spieler zum Beispiel im Punkt (0, 0) steht und nach Norden (also mit Drehung der Szene 0°) schaut, dann befindet sich ein Objekt mit den Koordinaten (0, 4) direkt vor im (im Abstand von 4 Metern) und ein Objekt mit den Koordinaten (-3, 4) vorne links (4 Meter nach vorne und 3 nach links). In der folgenden Abbildung ist die schematische Darstellung diesen Sachverhalts:



Da der Spieler nicht immer im (0,0) steht, sondern seine Position ständig ändert, die Gegenstände jedoch an der gleichen Stelle bleiben, kann der Spieler selbst nicht als Bezugspunkt für die Messung der Abstände gelten. Als Lösung dafür wird ganz zum Anfang ein beliebiger Punkt in der Nähe des Spielers als Bezugspunkt genommen und alle Positionen werden anhand von den *x*- und *y*-Komponenten der Abstände zu ihm bestimmt. Die Vorgehensweise ist in dem folgendem Diagramm anhand von einem Beispiel dargestellt:



Die *y*-Komponente des Abstandes vom Spieler zu dem Gegenstand ist gleich `dy' - dy` und die *x*-Komponente gleich *dx' - dx* (da der Spieler sich links von dem Bezugspunkt befindet, ist `dx' - dx = |`*dx'*`| - (-|`*dx*`|) = |dx'| + |dx''|`).

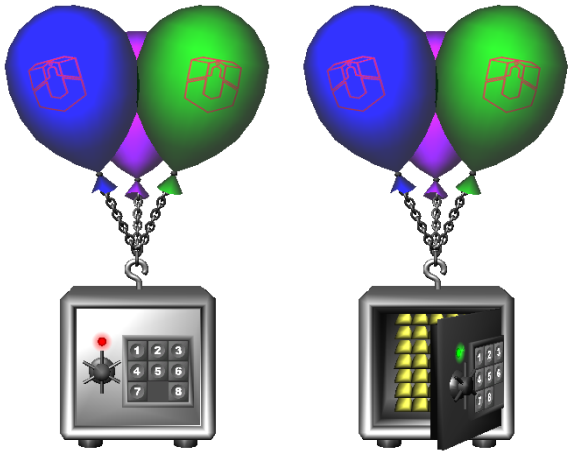
Diese Differenzen werden bei Zeichnen der Objekte berücksichtigt: Die Translationsmatrizen die auf jedes Objekt angewandt werden, beziehen sich auf diese Werte. Es wurde oben angenommen, dass der Spieler zum Norden schaut. Es ändert sich jedoch nichts, wenn er seine Blickrichtung ändert: Wenn man die

Drehmatrizen, Translationsmatrizen und die Projektionsmatrix in richtiger Reihenfolge miteinander multipliziert, werden die Objekte immer an passenden Stellen angezeigt.

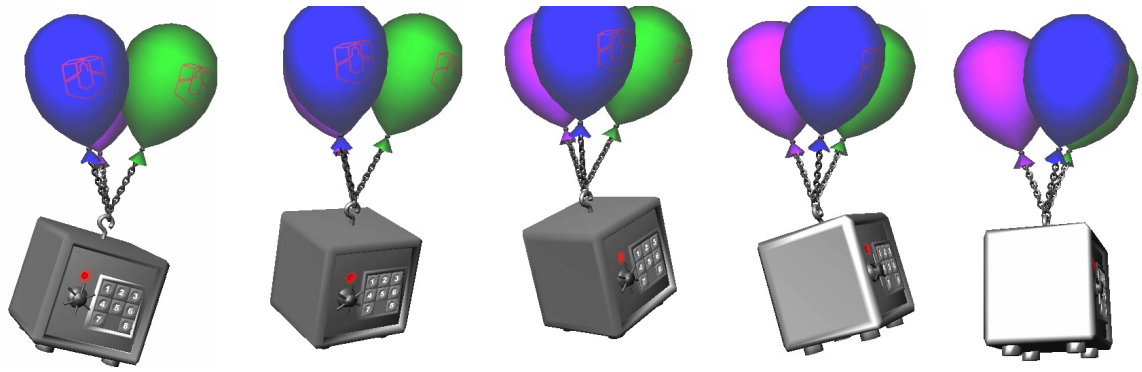
7.1.2 Erstellung der 3D-Objekte

Sergiy Krutykov

Zum Zeichnen der dreidimensionaler Objekte in OpenGL, müssen diese Objekte zuerst in einer passenden Form vorliegen. Sie bestehen häufig aus vielen Elementen, die unterschiedliche Farben, unterschiedliche Texturen und eventuell sogar unterschiedliche Orientierung im Raum haben (z.B., wenn nur ein Teil des Objekts animiert wird). In der folgenden Abbildung ist ein 3D-Tresor dargestellt, der an Luftballons hängt und dessen Tür sich öffnen lässt:



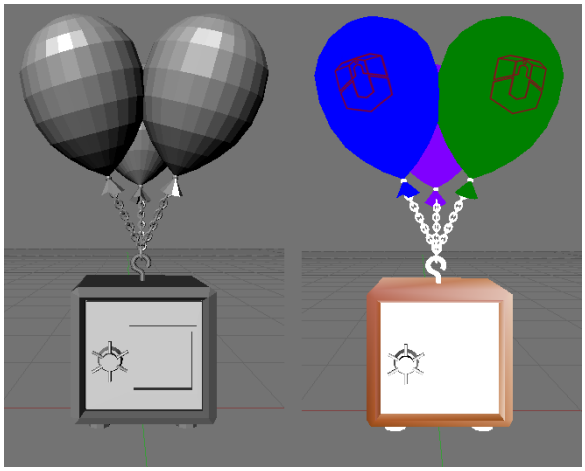
Die Bestandteile des Tresors sind: drei Luftballons(einzeln), Kette, Körper des Tresors, Tür, Klinke an der Tür, Platte mit 8-Puzzle darauf, Lämpchen, Leuchtkreis über dem Lämpchen und die Goldbarren. Diese Unterteilung in Bestandteile hat den Grund, dass sie erstens teilweise unterschiedlich gezeichnet werden sollen (z.B. sind Luftballons texturiert und die Kette ist nur grau gefärbt) und zweitens kann man die Bestandteile unabhängig voneinander im Raum transformieren. Zum Beispiel ermöglicht die Tatsache, dass die Tür nicht zu dem Tresor gehört, diese zu rotieren, ohne die Orientierung der anderen Teile vom Tresor zu ändern, was den Effekt der sich öffnenden Tür ermöglicht. Die Unabhängigkeit der Kette von dem Körper des Tresors ermöglicht zum Beispiel die folgende Animation:



Obwohl sich der Tresor als Ganzes um seine vertikale Achse dreht, verhalten sich seine Teile unterschiedlich: Die Kette und die Luftballons bleiben immer vertikal, während der Körper des Tresors selbst schwankt. Dies geschieht dadurch, dass auf unterschiedliche Teile des Tresor verschiedene Transformationsmatrizen angewandt werden.

Einige der Bestandteile dieses Tresors lassen sich mehr oder weniger leicht in dem Programm "on the fly" erzeugen, solche wie Lämpchen und Leuchtkreis, als auch Platte mit 8-Puzzle. Das Erstellen von anderen Bestandteilen, die etwas kompliziertere Strukturen haben, benötigt einen 3D-Editor. In diesem Projekt wurde

der kostenloser Editor Blender[40] eingesetzt. Mit Hilfe von diesem Editor wurde unter anderem der Tresor auf den Luftballons gemacht. So sieht er in Blender aus:



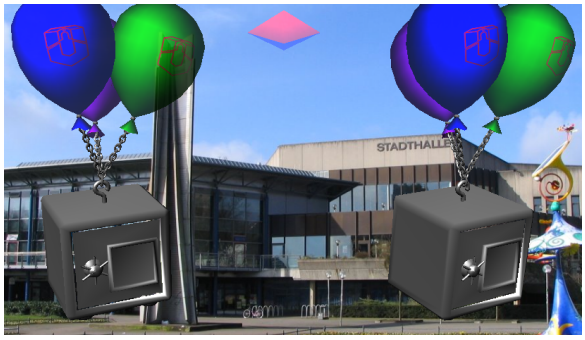
Links ist er in einfacher Beleuchtung und rechts ist er texturiert.

Ein Vorteil vom Blender, ist die Möglichkeit, die Objekte, die mit ihm gemacht worden sind, in Wavefront-Format (als obj-Dateien) zu exportieren, welches leicht zu parsen ist.

7.1.3 Interaktionen mit OpenGL-Objekten

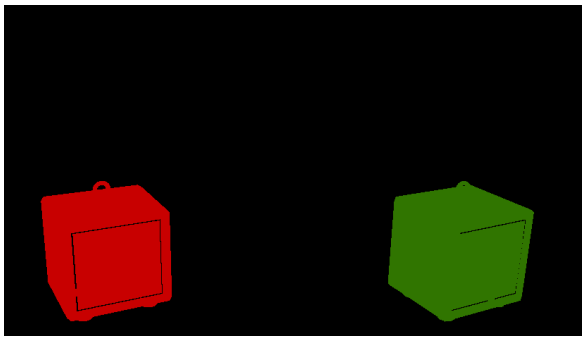
Sergiy Krutykov

Eine der Anforderungen an die 3D-Objekte in OpenGL ist, dass sie darauf reagieren, wenn man auf sie tippt. Das schwierige ist dabei, herauszufinden, auf welchem Objekt getippt wurde, wenn überhaupt. Die einzige Angabe, die man dabei hat, sind die Koordinaten des Fingers auf dem Touchscreen. Es gibt die OpenGL-Funktion `glReadPixels`, die zu den Bildschirmkoordinaten die Farbe der entsprechenden Pixels auf dem Bildschirm liefert. Man kann sie insbesondere auf einen einzelnen Pixel anwenden. Die Farbe steht dann in Form von drei Bytes (jeweils eines für rot, grün und blau - alpha kann meistens nicht ausgelesen werden). Aus der folgenden Abbildung wird ersichtlich, dass dies zu keinem Erfolg führt, wenn man diese Methode direkt auf die gerenderte Szene anwenden würde:



Die Tresors haben zu viele unterschiedliche Farben, die eventuell auch im Hintergrund vorkommen und insbesondere ist es problematisch, die Tresore selbst auseinander zu halten. Allerdings wenn man die Tresore mit einem einfarbigen Shader Programm rendert und dabei unterschiedliche Farben zuweist und den Hintergrund schwarz hält, dann kann man anhand von der Ausgabe von `glReadPixels` die Tresore identifizieren. Wenn der Benutzer auf einem leeren Feld getippt hat, dann liefert die Funktion (R,G,B) = (0,0,0) zurück, so dass man erkennen kann, dass kein Objekt getroffen wurde. Sonst wird die entsprechende Farbe zurückgeliefert. In folgender Abbildung ist dargestellt, wie die obigen Tresore in dieser einfarbigen Fassung aussehen:

40 http://www.blender.org/



Aus Effizienzgründen wird hier nur der Körper des Tresors gezeichnet, weil es ausreichend ist, wenn nur beim Tippen darauf der Tresor reagiert. Das Rendern braucht auch nur dann stattzufinden, wenn auf dem Touchscreen getippt wird. Das alles kann man auch auf dem Framebuffer machen, der auch zum Anzeigen benutzt wird. Allerdings ist viel sauberer einen alternativen, unsichtbaren Framebuffer dafür zu nehmen. Dann kann man auch die Perfomance etwas verbessern, indem man die Auflösung dieses Framebuffer niedriger wählt, wie in der folgenden Abbildung:

Da der Buffer unsichtbar ist, stört das unschöne Aussehen niemanden, aber das Rendern geht noch schneller. Allerdings muss man dann die Bildschirmkoordinaten etwas transformieren, da der Framebuffer nicht den ganzen Bereich des Screens ausfüllt.

Diese Methode funktioniert ganz gut (sobald der Client saubere Koordinaten der Finger auf dem Touchscreen liefert), denn das Rendern von einfarbigen Objekten sehr schnell geht. Die Funktion `glReadPixels` ist zwar ziemlich langsam, aber, da es sich hier immer um einen einzigen Pixel handelt, fällt das auch nicht ins Gewicht. Das Identifizieren anhand von Farben ist auch sehr einfach: Man muss einfach jede Zahl, die man einem Objekt zuweisen will, in 3 Bytes Spalten z.B. nach der folgenden Formel:

```
int objectID = 12345;
int B = objectID/65536;
int G = (objectID-B*65536)/256;
int R = (objectID-B*65536)-G*256;
```

Beim Erkennen der Objekte muss man die Farbe des Objekts wieder in einen Integer konvertieren `int objectID = B*65536 + G*256 + R`. Zum Beispiel war die ID von dem linken (roten) Tresors von oben gleich 200 und die des rechten (grünen) gleich 30000.

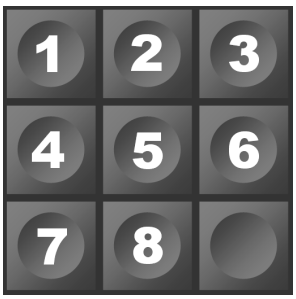
Die einzige Einschränkung dieser Methode ist, dass die Anzahl der zu verwendeten Farben und entsprechenden der unterschiedlichen IDs auf 256*256*256 = 16'777'216 beschränkt ist.

7.1.4 Minispiel

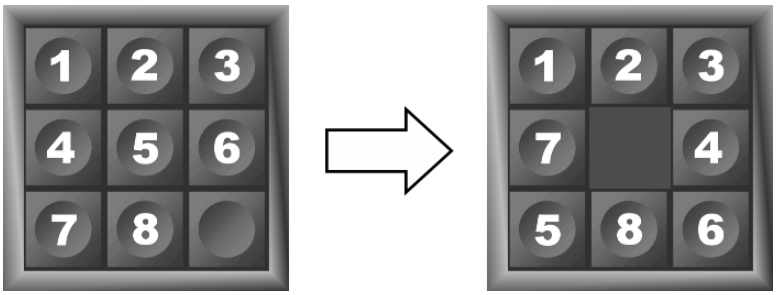
Sergiy Krutykov

Anzeige

In den oberen Abbildungen sieht man, dass der Tresor eine Platte mit 8-Puzzle auf sich hat, das zu lösen ist, wenn man den Tresor zu sich nimmt. Dabei können die Plättchen mit den Nummern mit einem Finger verschoben werden. Das Interessante ist, dass diese Plättchen gar nicht existieren, sondern dass das ganze Puzzle eine Textur ist, die über ein Quadrat gezogen ist. Als Ausgang für dieses Puzzle dient ein einziges Bild, das in der folgenden Abbildung zu sehen ist:



Man sieht sogar ein Dummy-Plättchen, welches so im Spiel gar nicht auftaucht. Also ist dieses Verschieben von Plättchen in OpenGL so implementiert, dass aus einer Textur eine andere generiert wird, die etwas andere Ordnung der Fragmente der ersten Textur aufweist, wie in der folgenden Abbildung dargestellt ist:

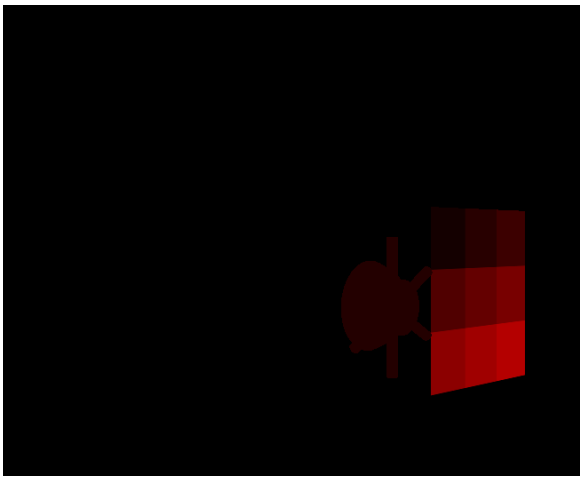


Bei diesem Vorgehen wird die Methode benutzt, die *Render-to-Texture* heißt, bei welcher ein gerendertes Bild direkt in eine Textur geschrieben wird.

Zum Umordnen von Plättchen wird ein unsichtbarer Framebuffer genommen (so wie im Unterabschnitt Interaktionen mit OpenGL-Objekten), der allerdings so definiert ist, dass sein Inhalt direkt in eine Textur geschrieben wird. In diesem Buffer wird sein *Viewport* auf seine volle Größe gelegt und auf ihm 8 gleich große (entsprechend jeweils genau ein Drittel von der Höhe und ein Drittel von der Breite des Viewports) Quadrate gezeichnet, die den ganzen Viewport ausfüllen, wobei jedes Quadrat die Ausgangstextur von oben bindet, allerdings jedes Mal in dem *Fragment Shader* auf eine andere Stelle von dieser zugreift. Wenn man die Quadrate an entsprechenden Stellen zeichnet, dann bekommt man automatisch am Ende des Vorgangs eine Textur mit neuem Zustand des Puzzles. Danach reicht es einfach die so generierte Textur wieder zu binden und beim Rendern von der Platte auf dem Tresor zu benutzen. Der graue Hintergrund an der Stelle in dem Puzzle von kein Plättchen vorhanden ist, entsteht dadurch, dass man `glClearColor` mit der grauen Farbe (vor dem Zeichnen der Plättchen) auf dem Framebuffer aufruft.

Interaktion mit dem Benutzer

Das Reagieren auf das Tippen oder das Ziehen mit dem Finger über den Touchscreen, wenn er das Puzzle löst, geschieht nach dem in Interaktionen mit OpenGL-Objekten beschriebenen Prinzip. Hier müssen offensichtlich die Plättchen auf einem unsichtbaren Framebuffer unterschiedlich gefärbt werden. Aber da diese Plättchen, wie oben bereits erklärt wurde, nicht existieren, muss hier eine etwas andere Methode angewandt werden. Analog zu der Darstellung des Puzzles selbst wird über die gleiche Platte des Puzzles eine Textur gezogen, die in neun gleich große Bereiche mit jeweils unterschiedlicher Farbe unterteilt ist. Die folgende Abbildung zeigt wieder den Tresor in normaler und "einfarbiger" Variante:

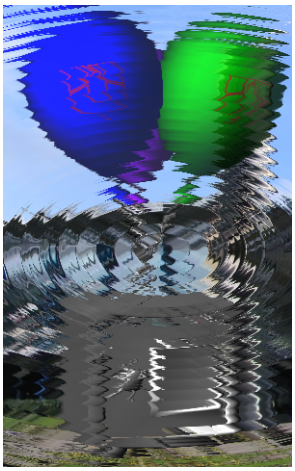


Außerdem ist die Klinke des Tresor auch gefärbt, damit der Benutzer beim Tippen darauf den Tresor öffnen könnte. Man sieht gleich den Vorteil dieser Methode: Man kann auf einem Tresor auch dann spielen, wenn er nicht exakt auf den Benutzer gerichtet ist. Allerdings sieht man auch, dass die Quadrate der Textur doch nicht ganz einfarbig sind, weil OpenGL diese interpoliert. Allerdings ist dies bei so kleiner Anzahl von Quadraten kein Problem, denn man kann die Farben so weit von einander liegend wählen, dass sie sich auch bei großer Ungenauigkeit, trotzdem nicht "überschneiden". Wenn aber eine große Genauigkeit gewünscht wird, kann man alternativ die Platte nicht texturieren sondern mit einem Gradient färben, welcher zum Beispiel von der linken oberen Ecke anfängt und nach rechts immer "roter" (also läuft die rote Komponente von 0 bis 255) wird und nach unten immer "grüner" (auch von 0 bis 255) wird. Damit könnte man die Koordinaten auf der Platte an der Stelle des Tippens gut erkennen. Wenn man zum Beispiel weiß, dass die Farbe (R,G,B) = (127, 127, 0) von `glReadPixels` zurückgeliefert wird, dann weiß man dass in der Mitte der Platte getippt wurde.

7.1.5 Spezielle Effekte

Sergiy Krutykov

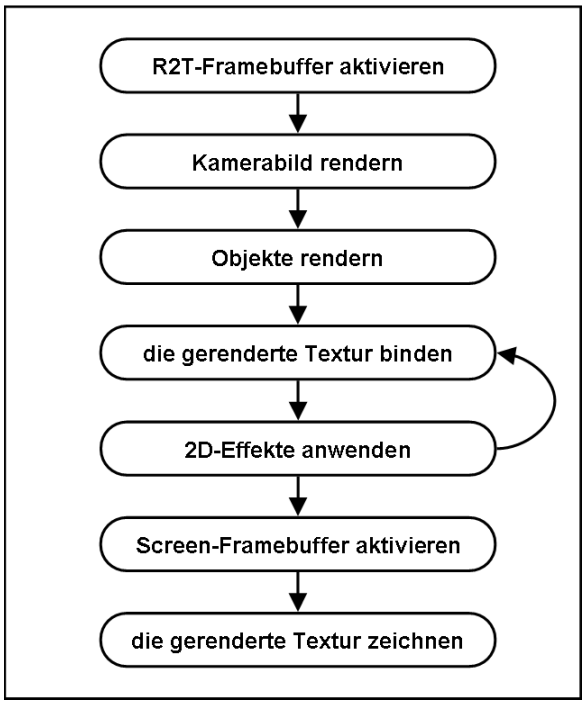
Die Methode Render-to-Texture, die bereits in dem Unterabschnitt Minispiel erläutert wurde, ermöglicht viele 2D Textur-Effekte, die eine OpenGL-Applikation verschönern können. Die Idee ist, dass man die ganze Szene nicht direkt auf dem Framebuffer rendert, der mit dem Bildschirm verknüpft ist (Screen-Framebuffer), sondern zuerst zu einer Textur, dann einige Effekte auf dieser Textur anwendet und erst diese Textur anzeigt. Mit dieser Methode lassen sich unter anderem die folgenden Effekte erzeugen:



Links ist ein Effekt des krummen Spiegels, in der Mitte der Effekt des Schwindelanfalls, und über der Szene rechts laufen Wasserwellen. Natürlich können diese Effekte auch animiert werden und diese Bilder sind nur Screenshots von den laufenden Animationen.

Offensichtlich machen diese Effekte nur dann Sinn, wenn man schafft, das Kamerabild als Textur in OpenGL zu bekommen.

Schematisch sieht der Vorgang folgendermaßen aus:



Mir R2T wird dabei "Render-to-Texture" abgekürzt. Besonderes zu beachten ist der Pfeil, der von "2D-Effekte anwenden" wieder auf "die gerenderte Textur binden" zurück zeigt. Er symbolisiert, dass man die Effekte mehrmals pro Durchlauf anwenden kann. Somit lassen sich die Effekte auch miteinander kombinieren. Dies wird vor allem bei dem bekannten Effekt "Blurring" verwendet, welches in der folgenden Abbildung dargestellt ist:

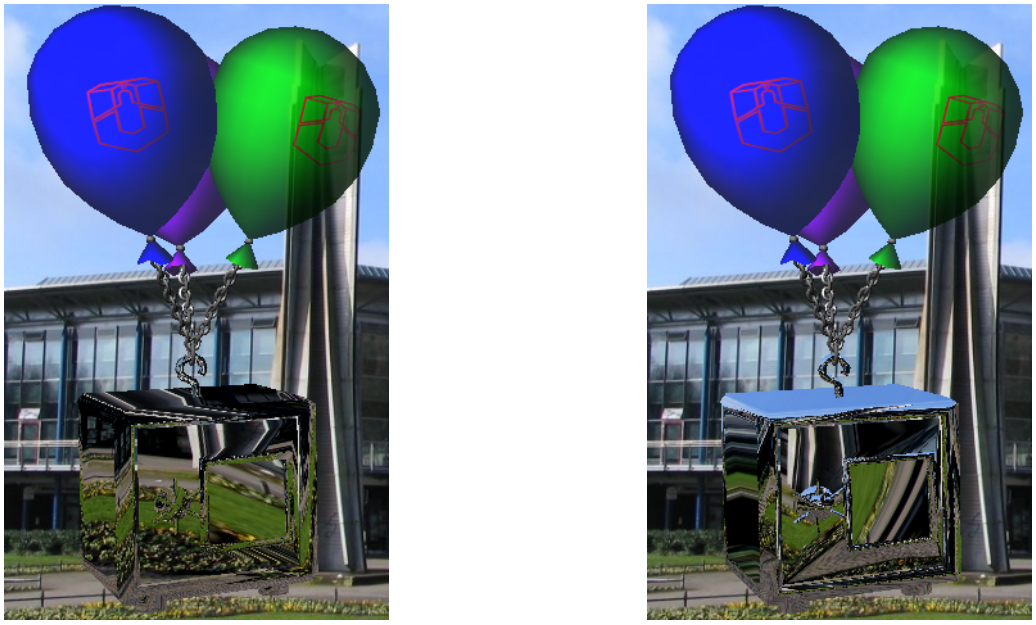
Das Ziel des Blurrings ist es, in jeden Pixel die Farbe ihn umgebender Pixels einfließen zu lassen, damit die Farben verwischt werden. Da alle Pixels einzeln betrachtet werden müssen, benötigt dieser Vorgang Parallelität, weshalb OpenGL ES 2.0 dafür gut geeignet ist. Sehr großen Performance-Gewinn bringt die Aufspaltung des Prozesses in 2 Unterprozesse: horizontales Blurring (nur die Pixels links und rechts von dem betroffenen Pixel werden betrachtet) und vertikales Blurring (entsprechend nur oben und unten stehende Pixels werden betrachtet). Wenn man diese Effekte kombiniert, dann ergibt sich das Ergebnis wie oben.

Je mehr man das Blurring hintereinander anwendet, desto verschwommener wirkt die Szene. Allerdings ist dieser Prozess auch mit der Aufspaltung in die vertikale und horizontale Komponente sehr aufwendig, so dass er sich so, in dieser Form auf einem Kleingerät kaum realisieren lässt. Als Abhilfe kann man an dieser Stelle einen Trick benutzen. Man kann die Auflösung heruntersetzen und dann das Blurring anwenden. Erstens gibt

es dann viel weniger Pixels und der Prozess läuft deshalb schneller und außerdem stellt sich heraus, dass bei niedrigeren Auflösungen die Szene viel schneller verschwommen wird, so dass man doppelten Vorteil dabei hat. In der folgenden Abbildung ist der Blurring-Effekt bei einer relativ niedrigen Auflösung zu sehen, bei welchem nur die nächsten Nachbarn der Pixels benutzt werden (also eigentlich sehr schwaches Blurring, bei welchem in einer normalen Auflösung kaum Unterschied zum Original festzustellen wäre):

Links ist die Szene (zum Vergleich) in der normalen Auflösung, rechts in etwas niedrigeren und rechts ist die Szene mit dem Blurring-Effekt, angewandt auf die Szene in der Mitte. Dieses Heruntersetzten der Auflösung kann man dadurch erzwingen, dass man einfach kleineren Render-to-Texture-Framebuffer (mit kleinerem ViewPort) verwendet aber den normalen Screen-Framebuffer: Dadurch wird am Ende des Vorgangs die kleinere Textur auf einen größeren Bereich aufgezogen.

Die Textur mit dem Kamerabild kann nicht nur für Render-to-Texture verwendet werden, sondern auch ganz normal zum Texturieren der Objekte. Dies ermöglicht einige spiegelnde oder durchsichtige Stoffe zu imitieren. In der folgenden Abbildung ist ein Tresor dargestellt, der auf diese Weise aus zwei etwas unterschiedlichen Arten texturiert ist:



Links scheint er aus Glas zu bestehen und rechts wirkt er metallisch.

7.1.6 OpenGL auf der Karte

Sergiy Krutykov

In dem Unterabschnitt Interaktionen mit OpenGL-Objekten wurde die Funktion `glReadPixels` erwähnt, die aus OpenGL die Farben eines bestimmten Bereichs eines Framebuffers lesen kann. Diese Methode kann man auch dafür nutzen, dass man auch außerhalb von dem OpenGL-Kontext die von OpenGL gerenderten Bilder als Pixel-Arrays benutzt. Wenn man diese Funktion permanent anwendet, dann haben diese Bilder einen Vorteil wie bei der Vektorgrafik: Sie sind beliebig verlustfrei skalierbar und können animiert werden. Dies macht insbesondere Sinn, wenn man die Objekte, die in OpenGL-Teil des Programm bereits vorliegen, diese auch auf der Karte wiederverwenden will, wobei diese auf der Karte genauso animiert sein müssen wie in OpenGL selbst. In der folgenden Abbildung sieht man die Tresore und Geschenke, die von der OpenGL auf die Karte "geholt" werden:

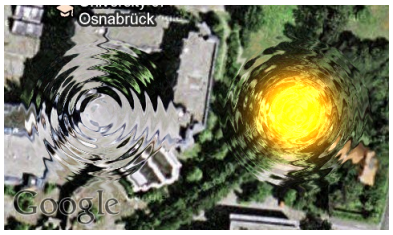
Diese Gegenstände auf der Karte können beliebig (solange der Speicher ausreicht) verlustfrei skaliert werden. Das sind einfach Annotationen, die mit Hilfe von dem Pixel-Array, welches von der Funktion `glReadPixels` geliefert wird, gezeichnet werden. Dieses Zeichnen geschieht unter Benutzung der Vektorgrafik, was ermöglicht, die halbdurchsichtigen abgerundeten Rechtecke auf dem Hintergrund der Bitmaps aus OpenGL bei kleineren Zoomstufenzu platzieren. (Siehe Unterabschnitt iPhone in dem Abschnitt Karten.)

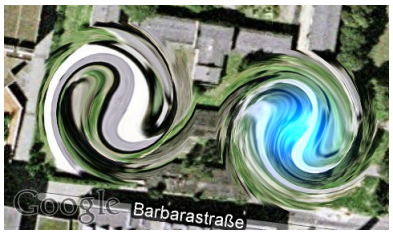
Damit diese Bilder in passender Größe gerendert werden, wird in OpenGL dafür ein Framebuffer benutzt, bei welchem jedes Mal der Viewport in der passenden Größe gesetzt wird und ein Tresor oder ein Geschenk aus einer bestimmten Blickrichtung darauf gerendert wird. Daraufhin braucht man nur `glReadPixels` auf den ganzen Bereich des Framebuffers anzuwenden.

Wenn man die Tresore in OpenGL animiert und `glReadPixels` permanent anwendet, dann lassen sich die Gegenstände auf der Karte ebenfalls animieren, was die folgende Abbildung zeigt:

Textur-Effekte auf der Karte

Es lassen sich auf der Karte auch die 2D-Effekte aus dem Abschnitt Spezielle Effekte anwenden, wenn die Karte als Textur vorliegt. Die folgende Abbildung zeigt zwei Effekte:



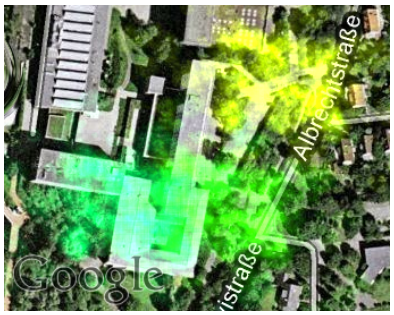


Links sind die Wellen, die über die Oberfläche der Karte laufen, wobei eine Welle aus der Mitte gelb leuchtet, und rechts sind zwei Wirbel dargestellt.

Im Gegensatz zu der Methode von oben, die die Bilder aus OpenGL einfach den Annotationen auf der Karte zuordnet, muss man bei diesem Vorgehen genau wissen, wo auf der Karte ein Effekt stattfindet, damit die passende Stelle der Textur als Basis für den Effekt genommen wird. Dafür muss man die reellen, geographischen Koordinaten der Annotation in die Bildschirmkoordinaten transformieren und diese dann in OpenGL verwenden.

Bei diesen Effekten muss man beachten, dass sie sehr aufwendig sind, so dass wenn eine Karte voll von Annotationen ist, das Prgramm sehr langsam werden kann. Außerdem ist das Vorgehen aus der urheberrechtlichen Gründen etwas zweifelhaft, da dadurch Teile der Karte verändert werden, die das Eigentum von Google sind.

Als Zugabe zu dem Unterabschnitt zeigt die folgende Abbildung, dass man mit OpenGL auch die Rauchwolken mit der Methode Particle Effects über die Karte ziehen lassen kann:



7.1.7 Schnittstelle

Sergiy Krutykov

Die Aufgabe der C-Bibliothek, die Routinen zum Zeichnen, Parser, Mathematik und Logik beinhaltet und auf unterschiedlichen Geräten möglichst gleich aussieht, ist eine Schnittstelle zu schaffen, die von der inneren Implementierung möglichst abstrahiert und die Benutzung von der Bibliotheck möglichst einfach gestaltet.

Die Schnittstelle, die im Rahmen des Projekts erarbeitet wurde, bietet, grob skizziert, folgende Funktionen an:

- Initialisierungsroutine
- Verwalten von Tresoren und Geschenken
- Zeichnen
- Behandlung von Touch Events

Die Schnittstelle mit ihren meisten Funktionen wird im folgenden wegen der bessern Übersicht vollständig aufgelistet. Im Weiteren wird die Applikation, die diese Bibliothek benutzt, als *Client* bezeichnet.

Initialisierungsroutine

Zu der Initialisierungsroutine gehören die folgenden Funktionen der Schnittelle, die in der angegebenen Reihenfolge beim Starten des Programms (bis auf `geTearDown`) ausgeführt werden (das "ge" am Anfang der Funktionsnamen steht etwa für Game Engine):

```
void geSetup(int screenWidth, int screenHeight, int cameraWidth, int cameraHeight);

void geCreateProgramsFromStrings(char *solidObjectShaders, char *quadShaders, char *planeShaders);

void geLoadSafeObjectFromStrings(char *body, char *door, char *knob, char *chaines,
                                 char *balloon1, char* ballon1_texture,
                                 char *balloon2, char* ballon2_texture,
                                 char *balloon3, char* ballon3_texture,
                                 char *eight_puzzle_texture);

void geLoadGiftObjectFromStrings(char *box, char *ribbon, char *rope,
                                 char *balloon, char* ballon_texture);

void geTearDown();
```

`geSetup` setzt OpenGL auf, wobei alle Framebuffer-Objekte, einige simple Vertexbuffer-Objekte initialisiert und in den Speicher von OpenGL geladen werden, wobei der Client nur die Auflösungen des Bildschirms (Höhe und Breite in Pixel) und der Kamera angeben muss.

`geCreateProgramsFromStrings` parst und lädt alle Shader in Speicher, die in Form von einem char-Array von dem Client geliefert werden müssen.

`geLoadSafeObjectFromStrings` parst alle Teile des Tresors und Texturen, initialisiert und lädt anschließend die Texturen in den Speicher und erzeugt Vertexbuffer-Objekte, die zu dem Tresor gehören. Alle Bestandteile müssen als char-Arrays von dem Client geliefert werden.

`geLoadGiftObjectFromStrings` ist das Analogon zu `geLoadSafeObjectFromStrings` für die Geschenke.

`geTearDown` befreit allen Speicher, der durch OpenGL belegt wurde.

Verwalten von Tresoren und Geschenken

Zum Verwalten der Tresoren und Geschenken müssen die folgenden Funktionen benutzt werden:

```
void geAddSafe(int ID, GLfloat x, GLfloat y, int value);

void geRemoveSafe(int ID);

void geRemoveAllSafes();

void geAddGift(int ID, GLfloat x, GLfloat y, int type);

void geRemoveGift(int ID);

void geRemoveAllGifts();

void geGetSafeMapIconAsBitmap(char *result, GLint safeIconSize, long absoluteTime);

void geGetGiftMapIconAsBitmap(char *result, GLint safeIconSize, long absoluteTime);
```

Die Methoden `geAddSafe` und `geAddGift` fügen einen Tresor bzw. ein Geschenk mit der angegebenen ID und mit angegebenen Koordinaten zu einer internen Liste. Die Bezeichnungen der Methoden `geRemoveSafe`, `geRemoveAllSafes`, `geAddGift`, `geRemoveGift` und `geRemoveAllGifts` sprechen für sich.

`geGetSafeMapIconAsBitmap` und `geGetGiftMapIconAsBitmap` schreiben in das mittels eines Pointers übergebene Array `result` das Bild für Karten-Annotationen zu der gegebenen Größe. Außerdem erwartet die Funktion die absolute Zeit in Millisekunden, um den Schritt der Animation zu ermitteln.

Zeichnen

Zum unmittelbaren Zeichnen sollen die beiden Funktionen benutzt werden:

```
void geSetBackground(GLint width, GLint height, char* pixels);

void geRender(long timeDelta, GLfloat angleX, GLfloat angleY, GLfloat angleZ,
                              GLfloat centerX, GLfloat centerY, GLfloat centerZ);
```

`geSetBackground` setzt das Kamerabild der angegebenen Auflösung, wobei die Pixels des Bildes als ein char-Array von dem Client geliefert werden muss.

`geRender` zeichnet alles, was auf dem Bildschirm auftauchen muss. Das Argument `timeDelta` ist die Zeitdifferenz in Millisekunden, die seit dem letzten Aufruf von `geRender` vergangen ist. Die Argumente `angleXangleYangleZ` stehen für die Orientierung des Smartphones im Raum (Pitch-, Roll- und Kompass-Winkel) stehen und die Argumente `centerX`, `centerY`, `centerZ` die relative Position des Spielers (und damit der OpenGL-Kamera) im Raum.

Behandlung von Touch Events

Um die Logik der Bibliotheck möglichst zu kapseln, werden die Touch Events des Benutzers von dieser Bibliothek selbst behandelt, wobei die Bibliothek selbst entscheidet, wie sie darauf reagiert:

```
void geTapScreen(GLfloat x, GLfloat y);

void gePanScreen(GLfloat x, GLfloat y, GLfloat tx, GLfloat ty);

void gePinchScreen(GLfloat scaleFactor);
```

`geTapScreen` reagiert auf das Tippen auf dem Touchscreen durch den Benutzer und erwartet die Bildschirmkoordinaten als Argumente und reagiert darauf je nach dem Zustand des Spiels. Diese Reaktion kann das Nehmen von einem Geschenk oder einem Safe, das Öffnen der Tür eines Tresors oder das Nehmen der Goldbarren sein.

`gePanScreen` reagiert auf das Ziehen mit einem Finger über den Touchscreen durch den Benutzer und bekommt wie auch bei `geTapScreen` die Bildschirmkoordinaten als Argumente und außerdem die Richtung und Länge der Verschiebung des Fingers. Wird wirksam nur beim Lösen von dem 8-Puzzle, indem die Plättchen dadurch bewegt werden.

`gePinchScreen` reagiert auf die "Pinch-to-Zoom" Geste des Benutzers und bekommt als Argument den entsprechenden Skalierungsfaktor. Wird ausschließlich für das Zoomen benutzt.

Callbacks

Da die Bibliothek völlig gekapselt ist und der Client nichts über ihren Zustand weiß und kann deshalb nicht entsprechend reagieren, wenn seine Reaktion für den Verlauf der Applikation notwendig ist, zum Beispiel bei den Sachen, die OpenGL selbst nicht kann, wie etwa einen Sound abspielen. Deshalb muss die Bibliotheck in der Lage sein, dem Client eine Zustandsänderung mitzuteilen. Dafür wird die Idee der Callbacks benutzt: Der Client bietet auch eine (sehr kleine) Schnittstelle an, die der Bibliothek zur Verfügung steht. Hier sind ein paar Funktionen aus dieser Schnittelle:

```
void gcAGiftBoxHasBeenPicked(int screenX, int screenY);

void gcChosenSafeCausedSound(int soundID);

void gcChosenSafeHasBeenSolved(int safeID);

void gcChosenSafeCouldNotBeSolvedInTime(int safeID);
```

`gcAGiftBoxHasBeenPicked` wird aufgerufen, sobald beim Tippen auf dem Touchscreen ein Geschenk erwischt wurde. Die Funktion liefert die Bildschirmkoordinaten des Fingers als Argument, damit der Client weiß, an welcher Stelle die Animation zu starten ist.

`gcChosenSafeCausedSound` teilt dem Client mit, dass ein Tresor ein Geräusch verursacht hat, wobei die ID des Geräusches als Argument auch mitgeteilt wird, damit der Client weiß, welcher Sound abgespielt werden kann.

`gcChosenSafeHasBeenSolved` signalisiert, dass der Tresor mit der ID `safeID` gerade durch den Benutzer gelöst wurde.

`gcChosenSafeCouldNotBeSolvedInTime` teilt dem Client mit, dass der Tresor mit der ID `safeID` nicht gelöst werden konnte, weil die dafür gegebene Zeit abgelaufen ist.

Portierung

Diese Bibliotheck wurde in erster Linie für zwei Plattformen entwickelt: iPhone und Android. Auf iPhone lässt sie sich einfach als Bestandteil des Projekts kompilieren und direkt benutzen. Auf Android benötigt man das Java Native Interface (JNI), um diese zu benutzen. Die Callbacks müssen in beiden Fällen implementiert werden.

7.2 Android

Peer Wagner

Besonders wenn sich eine Applikation einer solch fortschrittlichen Bibliothek wie OpenGL ES 2.0 bedient, gibt es immer verschiedene Punkte, die nicht auf Anhieb funktionieren oder auch auf eine andere Art eine ähnliche und von anderen als besser angepriesene Umsetzung funktionieren. Es soll daher im folgenden Abschnitt auf Schwierigkeiten bei der Verwendung der Kamera, als eine der Grundlagen für die AR, als auch auf zwei Möglichkeiten für die Umsetzung OpenGL ES 2.0 zu nutzen eingegangen werden.

7.2.1 Kamera und Musik

Peer Wagner

Kamera und Sound sind ein wichtiger Teil einer Augmented Rality Anwendung. Jedoch nehmen diese auch eine gesonderte Stellung bei der Programmierung auf dem Android-Smartphone ein. Dem ist so, da die Video- wie auch die Soundausgabe auf beschränkte Hardwareressourcen zugreifen. Es sind somit ein paar wenige Konzepte zu beachten wenn man auf dem Gerät mit jenen Ressourcen arbeiten möchte.

Kamera

Wie im allgemeinen Teil zu Android Smartphones bereits beschrieben, setzt sich die Ansicht einer Applikation aus verschiedenen Views zusammen. Die Kamera benötigt für die Darstellung des Bildes jedoch eine spezielle

Art der View, nämlich eine *SurfaceView*. Eine solche *SurfaceView* ist jedoch außer zu den Standardaktionen einer normalen View auch in der Lage den Zugriff auf das Canvas zu synchronisieren. Sie bietet sich daher besonders für eben solche Zwecke, wie das Zeichnen eines Kamerabildes etc. an. Die Ressourcen die für die Kamera benötigt werden, sind jedoch nur ein einziges mal verfügbar, was jedoch auch verständlich ist.

Um den Prozess der Benutzung zu verdeutlichen zeigt das folgende Bild ein Zustandsdiagramm.

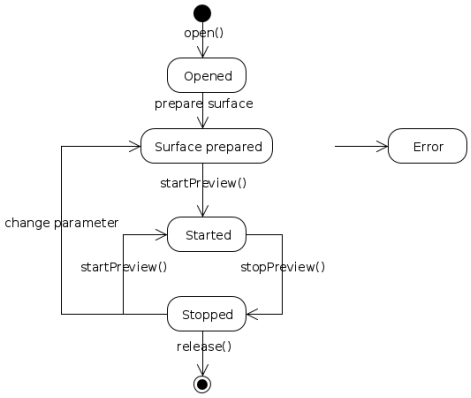


Abb. 56: Zustandsdiagramm der `CameraPreview`

Als erstes muss der Zugriff auf die Kamera durch den Methodenaufruf `open()` erfolgen. Dann ist die *SurfaceView* vorzubereiten um anschließend die Preview starten zu können. Beendet wird das Ganze durch den Aufruf von `stopPreview()`. Hierbei ist auf jeden Fall die Reihenfolge zu berücksichtigen und vor allem muss der Aufruf der Methode `stopPreview()` getätigt werden. Um auch wirklich auf der sicheren Seite zu sein sollte dieser Aufruf bei jedem Pausieren der Applikation durchgeführt werden, damit die Systemressourcen bei einem Absturz nicht weiterhin blockiert bleiben. Entsprechend ist bei einem Fortsetzen der Anwendung dann die `startPreview()` aufzurufen.

Für unsere Anwendung speziell, aber auch im allgemeinen bei Applikationen die einen Augmented Reality Modus nutzen, wäre es hilfreich wenn die View, welche das Kamerabild anzeigt und diejenige welche das OpenGL-Layer darstellt, ein und dieselbe View wären. Dies ist jedoch aus ungeklärten Umständen nicht machbar gewesen. Auch die Verwendung des Kamerabildes als OpenGL Textur ist nicht durchführbar gewesen, da sich dies als sehr unperformant erwiesen hat.

Sound

Der Sound, oder vielmehr die Soundausgabe, greift ebenfalls auf beschränkte Ressourcen des Android Smartphones zu. Aus diesem Grund muss mit einer vergleichbaren Sorgfalt programmiert werden um bei einem Absturz oder anderen unvorhersehbaren Zuständen nicht den Sound des Gerätes zu blockieren. Für die Benutzung der Soundausgabe gibt es jedoch weitere Beschränkungen zu beachten, da sich an einem `MediaPlayer` noch mehr Einstellungen vornehmen lassen als an der Kameraausgabe.

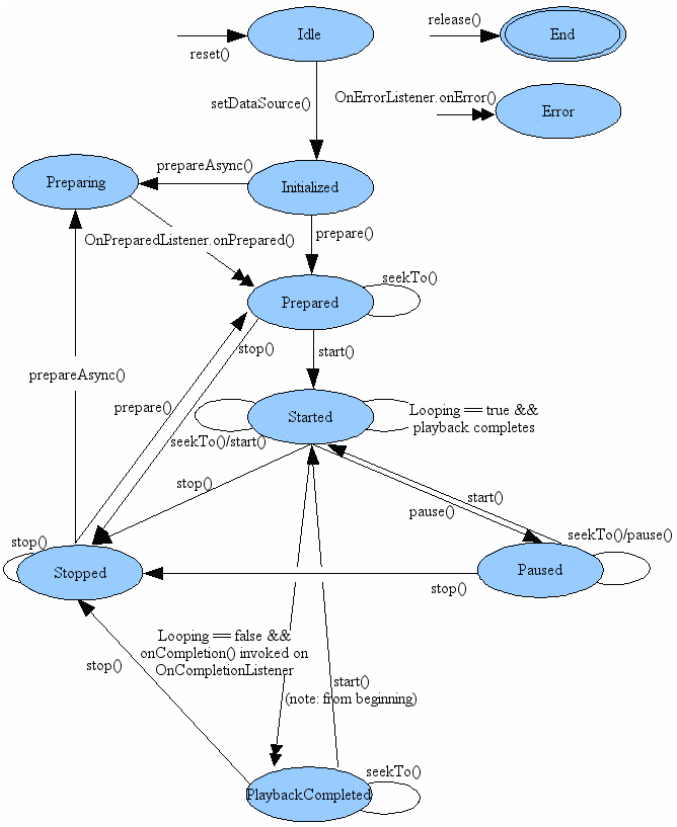


Abb. 57: Das Bild zeigt ein Zustandsdiagramm für die Verwendung des `MediaPlayer`. (http://developer.android.com/reference/android/media/MediaPlayer.html#StateDiagram)

Trotz dieser ausführlichen Dokumentation und vielen Tutorials ist es uns jedoch nicht gelungen die Soundausgabe auf dem Android-Smartphone zufriedenstellend umzusetzen.

7.2.2 OpenGL-Anbindung

Sascha Henke, Peer Wagner

Das Native Development Kit

Es gibt bereits seit der frühen Version 1.5 des Android-Systems eine elegante Möglichkeit OpenGL oder auch andere Performance kritische Applikationsteile zu programmieren. Dies geschah und geschieht teilweise immer noch mit dem *Native Development Kit*[41] (NDK), welches ebenfalls von Google zur Entwicklung bereitgestellt wird. Es ist allerdings nicht möglich mit dem NDK vollständige Applikationen zu entwickeln, sondern es dient lediglich als Erweiterung zum Android-SDK. Die Anbindung erfolgt dabei über das *Java Native Interface*[42], kurz JNI.

Die Applikationen lassen sich, wie der Name schon vermuten lässt, in C oder auch C++ schreiben und haben sogar durch verschiedene Header Zugriff auf Hardware wie die Kamera oder die Soundausgabe. Es wird jedoch davon abgeraten Anwendungen nur in nativem Code zu schreiben weil einem die Sprache C besser gefällt, da sich hierbei zum Beispiel eine erhöhte Gefahr von Speicherzugriffsfehlern ergeben oder auch die Sandbox in der die Applikationen immer laufen kompromittieren lassen.

41 http://developer.android.com/sdk/ndk/overview.html
42 http://java.sun.com/docs/books/jni/

Das NDK eigent sich somit für Operationen die nicht zu viel Speicher allokieren und stellt daher eine gute Erweiterung für die Entwicklung von Applikationen bereit auch wenn sich manche Dinge dadurch kaum vereinfachen oder gar umgehen lassen.

Java-Bindings

Mit der Android-Version 2.2 (Froyo) gibt es neuerdings außer dem NDK auch Java-Bindings um OpenGL ES 2.0 zu verwenden. Diese sind jedoch noch im Aufbau und stellen keine vollständige Implementation der OpenGL ES 2.0 Schnittstelle bereit, so dass man hierauf nur zurückgreifen sollte wenn man sich sicher ist, dass man die nicht implementierte Funktionalität in keinem Fall benötigt. Mit dem Erscheinen neuerer Android-Versionen werden jedoch die Bindings immer weiter vervollständigt, so dass mit einer der kommenden Versionen eventuell keine Programmierung mit dem NDK mehr nötig ist um aufwendige Spielegrafiken in eine Applikation einzubauen.

Da wir jedoch in den Anfängen schon an die Grenzen der damaligen Bindings gestoßen waren haben wir unsere Augmented Reality komplett über das NDK programmiert. Das NDK hat außerdem den Vorteil der Wiederverwendbarkeit von Code gebracht, denn der OpenGL-Code ist wie in den vorigen Kapiteln beschrieben fast Plattformunabhängig.

7.2.3 Animationen beim Einsammeln der Geschenke

Sebastian Stock

Wenn der Benutzer im *Augmented Reality Modus* ein Geschenk anklickt, um dieses einzusammeln, wird dessen Inhalt mit Hilfe einer Animation angezeigt. Der Gegenstand, der sich in ihm befindet, wird zunächst größer und bewegt sich dann in das Inventar in der Statusleiste, wobei er wieder kleiner wird. So soll dem Benutzer deutlich gemacht werden, dass dieser Gegenstand nun eingesammelt wurde und im Inventar darauf zugegriffen werden kann.

Zum Anzeigen der Gegenstände wird eine `ImageView` verwendet. Dies ist eine einfache *View* zur Darstellung von Bildern. Die Sichtbarkeit wird zunächst auf *unsichtbar* gesetzt. Unter Android können *Views* mit der Klasse `Animation` zweidimensional animiert werden [43]. Wie so häufig unter Android können auch Animationen in XML oder Java geschrieben werden. Folgendes Beispiel zeigt die in XML definierte Animation für das Einsammeln eines Geschenkes:

```
<set xmlns:android="http://schemas.android.com/apk/res/android"
android:interpolator="@android:anim/accelerate_interpolator">

   <scale
         android:fromXScale="0.5"
         android:toXScale="1"
         android:fromYScale="0.5"
         android:toYScale="1"
         android:duration="1000"
         android:pivotX="50%"
         android:pivotY="50%"
         android:startOffset="0000" />

   <scale
         android:fromXScale="1"
         android:toXScale="0.2"
         android:fromYScale="1"
         android:toYScale="0.2"
         android:duration="2000"
         android:pivotX="50%"
         android:pivotY="50%"
         android:startOffset="1000" />

   <translate
         android:interpolator="@android:anim/accelerate_interpolator"
```

43 http://developer.android.com/guide/topics/graphics/view-animation.html

```
        android:fromXDelta="0"
        android:toXDelta="18%"
        android:fromYDelta="00%"
        android:toYDelta="-40%"
        android:duration="3000"
        android:startOffset="1000"
        android:fillBefore="false"
        android:fillAfter="false"
       />
</set>
```

Drei Teilanimatonen - zwei Skalierungen und eine Translation - werden hier mit dem Element `<set>` zu einer Animation zusammengefasst. Die erste Skalierung vergrößert das Bild. Dazu wird es beim Start um den Faktor 0.5 verkleinert und nimmt am Ende wieder seine volle Größe ein. Mit der zweiten Skalierung wird das Bild wieder auf den Faktor 0.2 verkleinert, sodass es in etwa der Größe der Buttons in der Statusleiste entspricht und so angedeutet werden kann, dass es im Inventar verschwindet. Dazu muss allerdings noch mit dem `<translate>`-Element eine Bewegung des Bildes durchgeführt werden. Deren Ziel lässt sich mit den Attributen `android:toXDelta` bzw. `android:toYDelta` angeben und wurde so gewählt, dass die Animation auf dem Inventar-Button in der Statusleiste endet. Damit allerdings nicht alle Teilanimationen gleichzeitig ausgeführt werden, kann jeweils mit `android:duration` die Dauer in Millisekunden und mit `android:startOffset` eine Verzögerung angegeben werden. So wird zunächst die Vergrößerung durchgeführt, welche eine Sekunde dauert. Anschließend wird das Bild drei Sekunden lang gleichzeitig verschoben und verkleinert. Mit dem Attribut `android:interpolator` kann zusätzlich noch die Geschwindigkeit beeinflusst werden. Mit dem hier verwendeten `AccelerateInterpolator` beginnt die Animation langsam und wird dann schneller. Die in der Datei `gift_animation.xml` im Ressource-Ordner `res/anim` definierte Animation kann mit folgendem Aufruf aus den Ressourcen geladen werden:

```
Animation anim = AnimationUtils.loadAnimation( context, R.anim.gift_animation );
```

Wenn im *Augmented Reality Modus* ein Geschenk angeklickt wird, muss zunächst in der `ImageView` das dazugehörige Bild gesetzt werden, da nur eine `ImageView` verwendet wird, aber sechs unterschiedliche Gegenstände und somit auch Bilder vorhanden sind. Die unterschiedlichen Gegenstände und ihre Bilder sind im Kapitel *Spielanleitung* unter dem Punkt *Inventar, Chat, High-Score* aufgeführt. Das Starten der Animation wird dann folgendermaßen vorgenommen, wobei davor und danach noch die Sichtbarkeiten angepasst werden müssen:

```
imageView.setVisibility(View.VISIBLE);
imageView.startAnimation(animation);
imageView.setVisibility(View.GONE);
```

Die folgenden drei Bilder zeigen Ausschnitte aus der Animation eines Gegenstandes. In dem linken Bild ist der Gegenstand direkt nach dem Start der Animation noch relativ klein und erreicht in dem mittleren Bild seine maximale Größe. Auf dem rechten Bild sieht man wie der Gegenstand schließlich wieder verkleinert wird und sich in Richtung des Inventars bewegt.

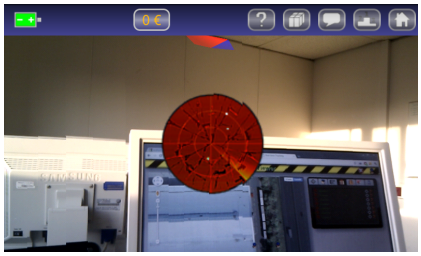


Abb. 58: Start der Animation.

Abb. 59: Vergrößerung des Bildes.

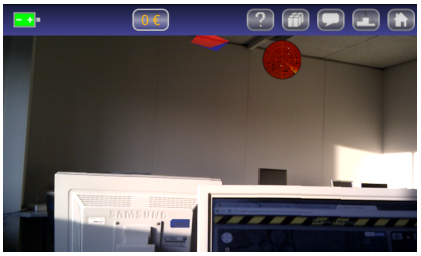


Abb. 60: Translation in Richtung des Inventars.

7.2.4 Aufgetretene Probleme

Sebastian Stock, Peer Wagner

Das Zusammenspiel der einzelnen für die *Augmented Reality* benötigten *Views* und der Wechsel zwischen verschiedenen *Activites* hat einige Schwierigkeiten und Schwächen bei deren Programmierung zum Vorschein gebracht. Das Kamerabild und die beiden *Views* zur Darstellung der *OpenGL*-Objekte und der Geschenk-Animation müssen in einem `FrameLayout` vereinigt und übereinander gelegt werden, um alle gleichzeitig anzeigen zu können. Erst so wird eine *Augmented Reality* erzeugt. Dafür müssen die *Views* in einer bestimmten Reihenfolge in das `FrameLayout` eingefügt werden, da das Kamerabild nicht die anderen beiden Views überdecken darf.

Aber selbst mit der scheinbar richtigen Reihenfolge traten Probleme auf. So wurde beim ersten Wechsel in die *Augmented Reality* zunächst alles richtig angezeigt. Nach einem Wechsel in eine andere *Activity*, wie beispielsweise das *Inventar*, war nur noch das Kamerabild sichtbar. Wie sich später herausstellte, ist dies verschuldet durch eine falsche Implementation der Klasse, von der alle Views abstammen. Es hat also nicht gereicht auf verschiedene Views auszuweichen und auch ein zweifaches Initialisieren der Oberflächen brachte keine Besserung. Das Problem ist nämlich, dass die Reihenfolge und damit die Tiefeninformationen beim Erstellen der Augmented Reality von großer Bedeutung sind. Die Tiefeninformation wird jedoch beim initialen Aufbau der Views anders herum getätigt als es bei einem erneuten Aufbau der Oberflächen der Fall ist. Seltsam ist dabei jedoch nicht, dass es diesen Fehler gibt, sondern dass er bereits seit der ersten Android-Version existiert und auch seitdem bekannt ist und trotzdem noch nicht behoben wurde. Hier stellt man sich natürlich die Frage ob ein solches System dann überhaupt für einen stabilen Betrieb verschiedener Applikationen geeignet ist.

Die Soundausgabe ist, wie im vorigen Kapitel beschrieben, ebenfalls nicht zu benutzen gewesen. Der Fehler der hierbei auftrat war, dass der Sound in manchen Fällen gar nicht und in manchen verzerrt wiedergegeben wurde. Dies ist natürlich nicht zumutbar wenn man ein soches Spiel vertreiben möchte. Es mag allerdings auch an der Anforderung liegen, den Sound in Echtzeit zu wechseln oder auch zu wiederholen, denn es sind dabei immer noch die beschriebenen Grenzen einzuhalten und nur eine Instanz des `MediaPlayer` vorzuhalten.

8. Agent

Sebastian Stock

Zusätzlich zu realen Spielern gibt es auch Agenten. Dieses sind Computergegner mit denen zwei Ziele verfolgt werden. Zum einen kann mit ihnen der Server getestet werden und zum anderen können sie auch als Gegner in realen Spielen eingesetzt werden. Sie führen in einem Spiel alle Aktionen durch, die auch menschliche Spieler nutzen können. So senden auch sie in regelmäßigen Abständen Updates ihrer Position, lösen Tresore mit einer bestimmten Wahrscheinlichkeit, nutzen das Inventar und chatten, um sich über nächste Zielpunkte auszutauschen.

Beim Testen des Servers können effizient reale Spielsituationen überprüft werden, ohne dass das Spiel von menschlichen Testern gespielt werden muss. Auch lässt sich mit ihnen das Verhalten des Servers bei Spielen mit sehr vielen Spielern testen, was ansonsten mit einer begrenzten Anzahl an Smartphones nur schwer möglich wäre.

Da die Agenten allerdings nicht nur zufällige Aktionen durchführen, sondern sich wie rationale Spieler verhalten, bietet es sich an, diese auch in realen Spielen einzusetzen. So haben auch Einzelspieler und kleinere Gruppen die Möglichkeit in größeren Teams zu spielen.

8.1 Ablauf

Sebastian Stock

Für die Datenhaltung und Kommunikation mit dem Server wird bis auf wenige notwendige Modifikationen das Modell der Android-Smartphones verwendet, welches im Kapitel *Datenhaltung* vorgestellt wird. Daher stehen dem Agenten nur die Informationen zur Verfügung, die auch den Spielern auf den Smartphones angezeigt werden. Somit hat er keinen künstlichen Vorteil gegenüber anderen Spielern, welcher z.B. durch zusätzliche Schnittstellen zum Server denkbar gewesen wäre, allerdings auch von anderen Spielern als unfair angesehen werden könnte. Einzig bei der Routenplanung wird auf einen externen Webservice zurückgegriffen, worauf im Unterkapitel *Routenberechnung* genauer eingegangen wird. Aber auch dies ist mit der auf den Smartphones angezeigten *GoogleMaps*-Karte vergleichbar, mit der auch die Spieler eine Möglichkeit der Orientierung und Routenplanung zur Verfügung haben.

Parameter

Agenten wurden als Threads implementiert, diese können direkt vom Server heraus oder als eigenes Java-Programm gestartet werden. Im Konstruktor der Klasse `Agent` werden alle benötigten Parameter mitgegeben. Diese werden im Folgenden kurz vorgestellt:

- `AgentHandler handler`: Wenn der Agent vom Server aus gestartet wurde, können mit diesem Handler Log-Ausgaben und Exceptions geschickt werden. Auf diese Weise kann der Server informiert werden, wenn beispielsweise ein nicht behandelbarer Fehler aufgetreten ist.
- `String host`: Adresse des Servers, zu dem sich der Agent verbinden soll.
- `String agentname`: Benutzername mit dem der Agent einem Spiel beitritt.
- `String password`: Zum Benutzernamen gehörendes Passwort mit dem sich der Agent einloggen kann.
- `int gameId`: Gibt das Spiel an dem der Agent beitreten soll.
- `GeoPoint startposition`: Die Geo-Koordinaten der Startposition des Agenten im Spiel.
- `int safesToClaim`: Gibt an, ob und nach welcher Anzahl geöffneter Tresore der Agent ein Spiel verlässt. Bei dem Wert 0 ist diese Anzahl unbegrenzt.
- `double speedfactor`: Als Basis bewegt sich der Agent mit einer Geschwindigkeit von 4 km/h. Diese Geschwindigkeit kann mit dem hier angegebenen Faktor beeinflusst werden, um den Agenten beispielsweise für Tests schneller oder im realen Spiel evtl. auch langsamer laufen zu lassen.
- `int maxDelay`: Gibt den maximalen Zeitabstand in Millisekunden an, nach dem der Agent ein Update seiner Geo-Position sendet.

- `boolean cops`: Gibt an, welchem Team der Agent beitreten soll. Bei dem Wert `true` ist er ein Polizist und andernfalls ein Dieb.

Hauptschleife

In der `run`-Methode des Agenten wird sein eigentliches Verhalten realisiert. Dieses wird im folgenden Diagramm verdeutlicht:

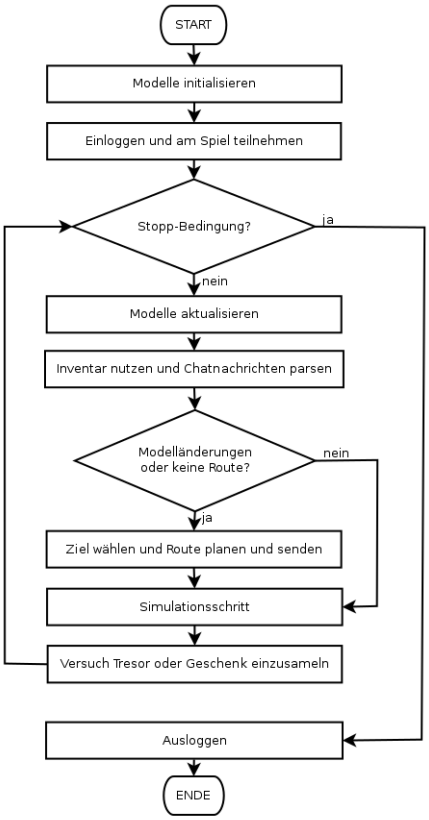


Abb. 61: Programmblaufplan der `run`-Methode des Agenten

Zunächst müssen die Modelle initialisiert werden und der Agent versucht, sich mit seinem Benutzernamen und Passwort einzuloggen, einem Spiel beizutreten und seine Startposition an den Server zu senden. War dies erfolgreich, werden in einer Hauptschleife seine Aktionen im Spiel realisiert. Die Stopp-Bedingung setzt sich dabei aus mehreren Einzelbedingungen zusammen. Sie ist erfüllt, wenn von Außen ein Signal zum Beenden gesendet oder die vorgegebene maximale Anzahl gelöster Tresore erreicht wurde.

In der Schleife wird als Erstes mithilfe des `UpdateModel` eine Aktualisierung der Modelle durchgeführt. Anschließend werden außer der Blaupause alle im Inventar vorhandenen Gegenstände genutzt und die Chatnachrichten geparst, worauf im Unterkapitel *Zielauswahl* noch eingegangen wird.

Falls notwendig, wird eine neue Zielposition berechnet. Dies ist genau dann der Fall, wenn sich am Modell etwas geändert hat und z.B. Tresore hinzugekommen sind oder von einem anderen Spieler gelöst wurden, oder wenn derzeit keine Route vorhanden ist. Letzteres ist direkt nach dem Start oder nachdem im letzten Schritt das Ziel erreicht wurde erfüllt. Dann wird mit dem `PathPlanner` eine Zielposition und die zugehörige Route ermittelt. Auf diesen wird im Kapitel Zielauswahl noch genauer eingegangen. Wenn sich der neu berechnete Zielpunkt von dem bisherigen unterscheidet, wird die Route an den Server gesendet, damit diese im Gameviewer angezeigt werden kann.

Anschließend wird die Bewegung des Agenten simuliert. Die Route besteht aus einer Liste von Geo-Koordinaten. Diese wird schon bei der Berechnung der Route so generiert, dass zwei aufeinander folgende Punkte, bei

gegebener Bewegungsgeschwindigkeit, in einer Zeit, die die ebenfalls gegebene maximale Zeitdifferenz nicht überschreitet, erreicht werden. Auf Details der Generierung der Route wird im Unterkapitel Routenberechnung noch genauer eingegangen. In diesem Simulationsschritt muss also zunächst der nächste Geo-Punkt von der Route genommen und die Distanz zur aktuellen Position berechnet werden. Mit dieser lässt sich nun die Zeit ermitteln, die der Agent für diese Teilstrecke benötigt. Für diese Zeit wird der als Thread implementierte Agent daher auch pausiert.

Abschließend muss noch überprüft werden, ob mit dem letzten Schritt das Ziel erreicht wurde. In diesem Fall wird versucht, den Tresor zu öffnen bzw. zu sichern oder das gefundene Geschenk einzusammeln. Da Geschenke auch auf den Smartphones durch einfaches Anklicken eingesammelt werden können und dieser Vorgang daher für den Benutzer keinen großen Aufwand darstellt, sendet auch der Agent beim Erreichen der entsprechenden Geo-Koordinate sofort die entsprechende Anfrage an den Server.

Das Öffnen der Tresore hingegen wird den Spielern auf den Smartphones durch das zusätzliche 8-Puzzle deutlich erschwert. Zum einen wird dies eine gewisse Zeit dauern und zum anderen besteht auch die Gefahr, das Rätsel nicht zu lösen und somit einen Punktabzug zu erhalten. Daher ist es sinnvoll diese beiden Aspekte auch in das Verhalten des Agenten einzubeziehen. So öffnen auch die Agenten einen Tresor nur mit einer bestimmten Wahrscheinlichkeit und bekommen andernfalls Geld abgezogen. Hier wurde eine Wahrscheinlichkeit von 90% für erfolgreiches Öffnen bzw. Sichern angesetzt. Die Zeit die der Agent für das Puzzle benötigt ist nicht fest vorgegeben, sondern auch vom Wert des Tresors abhängig. Diese Werte liegen im Bereich 10000ms bis 40000ms. Im Falle eines erfolgreichen Lösens des Puzzles, wird die benötigte Zeit nun zufällig aus dem Intervall [10000 , 10000 + Tresorwert] ms ausgewählt. Andernfalls beträgt die Zeit 10000ms + Tresorwert ms. Der Agent wird um diese Zeit pausiert, bevor er mit der entsprechenden Anfrage an den Server fortfährt. Ein Sonderfall tritt auf, wenn eine Blaupause im Inventar vorhanden ist. Diese wird sofort genutzt und ein entsprechendes Warten bzw. Lösen des 8-Puzzles ist nicht notwendig.

Anschließend wird die Schleife gegebenenfalls erneut durchlaufen und die neue Position des Agenten gesendet. Ein Abbruch der Schleife erfolgt falls der Server dies mit einem Aufruf der `logout`-Methode veranlasst hat oder die vorgegebene maximale Anzahl an geöffneten Tresoren erreicht ist.

Serveranbindung

Wie bereits erwähnt, kann der Agent als Thread vom Server aus gestartet werden. Um dieses zu ermöglichen waren einige Anpassungen notwendig. So muss der Server beispielsweise über ein Beenden des Agenten informiert werden und es muss dem Server auch möglich sein, den Agenten zu beenden. Hier genügt es nicht lediglich den Thread zu beenden, sondern es muss sichergestellt werden, dass der Agent vorher noch einen Logout durchführt. Um dieses zu erreichen muss der Agent das Interface `IAgent` implementieren, in welchem die folgenden vier Methoden enthalten sind.

- `public void validateLogin() throws AgentException` : Überprüft, ob sich der Agent mit den im Konstruktor angegebenen Nutzernamen und Passwort einloggen kann. Ist dieses nicht möglich, versucht er sich neu zu registrieren. Wenn beides fehlschlägt, wird eine `AgentException` geworfen um dem Server diesen Fehler mitzuteilen. Anschließend loggt er sich wieder aus.
- `public void start()` : Startet den Agenten und wird bereits von der Oberklasse `Thread` geerbt. Dadurch wird die `run()`-Methode ausgeführt, deren Verhalten oben bereits beschrieben wurde.
- `public Integer getPlayerID() throws AgentException` : Gibt die eindeutige *ID* des Agenten zurück, die er vom Server erhält, wenn er sich einloggt. Da diese Methode häufig direkt nach dem Starten des Agenten aufgerufen wird, er zu diesem Zeitpunkt allerdings nicht immer bereits eingeloggt ist, wird hier gewartet bis die geforderte ID vorliegt. Existiert diese nach einer vorgegebenen Zeitspanne immer noch nicht, wird eine `AgentException` geworfen.
- `public void logout() throws AgentException` : Veranlasst den Agenten dazu sich zu beenden und auszuloggen. Wenn er aktiv ist, wird die aktuelle Iteration der Hauptschleife noch vollständig ausgeführt und anschließend ein Logout durchgeführt. Andernfalls loggt er sich neu ein und direkt wieder aus. So wird sichergestellt, dass er nicht im Spiel verbleiben und für die anderen Spieler weiterhin angezeigt werden kann.

8.2 Zielauswahl

Sebastian Stock

In diesem Unterkapitel wird etwas genauer beschrieben, nach welchen Kriterien die nächste Zielposition ausgewählt wird. Dies geschieht in der Klasse `Pathplanner`. Als Basis dienen dabei natürlich die Tresore, Geschenke und die eigene Position. Aber auch die Positionen und evtl. bekannten Zielpositionen anderer Spieler werden berücksichtigt.

Um sich keinen künstlichen Vorteil gegenüber anderen Spielern zu verschaffen, verwendet auch der Agent nur diejenigen Tresore und Geschenke, die für ihn auch sichtbar sind. Diesen wird jeweils mit Hilfe einer Bewertungsfunktion ein Wert zugeteilt und der Tresor oder das Geschenk mit dem höchsten Wert wird als Ziel ausgewählt.

Chat der Agenten untereinander

Bevor auf die Bewertungsfunktion eingegangen werden kann, muss noch kurz ein weiteres Detail angesprochen werden. Ein Problem, das sich zunächst bei mehreren Agenten schnell gezeigt hat, war, dass sie nach einer gewissen Zeit alle das gleiche Ziel gewählt haben. Dieses kann verhindert werden, indem sie den Chat nutzen, um sich gegenseitig ihre Ziele mitzuteilen. Jedesmal wenn ein neues Ziel ausgewählt wurde, wird im Team-Chat `"Next goal: "`, gefolgt von der eindeutigen *ID* des Tresors oder Geschenkes, gesendet. Ebenso werden die anderen Spieler auch darüber informiert, wenn der Agent zu seinem bisherigen Ziel nicht mehr läuft, er einen Tresor geöffnet hat, das 8-Puzzle nicht lösen konnte oder ein Geschenk eingesammelt hat.

Bewertungsfunktion

Um den Tresoren und Geschenken einen Nutzen zuzuweisen wird zunächst deren Punkte-Wert durch die euklidische Distanz, der aktuellen Position des Agenten zu diesen, dividiert. Da man für das Öffnen der Tresore zwischen 10000 und 40000 Punkten erhält, die Geschenke jedoch keinen festen Wert haben, wird für diese ein Wert von 2000 Punkte verwendet. Man erhält auf diese Weise also jeweils die Anzahl der Punkte pro Meter. Dies ist zwar eine gute Basis kann durch einige andere Faktoren jedoch noch verbessert werden:

- Wenn ein Mitspieler aus dem gleichen Team im Chat bereits angekündigt hat, zu diesem Ziel zu laufen, wird der Nutzenwert stark reduziert.
- Die Positionen der anderen Spieler werden herangezogen, damit sich die Spieler besser verteilen: Wenn ein Mitspieler aus demselben Team näher am Ziel ist, wird dieses bestraft und der Nutzen mit Hilfe des Faktors *`0.8 * teamDistanz / Distanz`* verringert, wobei *teamDistanz* die minimale Distanz eines Spielers aus demselben Team zum Ziel ist. Im Falle eines Geschenkes wird statt `0.8` der Faktor `0.2` verwendet, da es im Vergleich zu den Tresoren weniger sinnvoll ist, wenn sich zwei Spieler aus dem gleichen Team nur wegen eines Geschenkes in das gleiche Gebiet bewegen.

Die Positionen der Gegner gehen ausschließlich in die Bewertung der Geschenke ein und verringern auch hier den Nutzen, wenn sich ein Gegner näher an einem Geschenk befindet als der Agent. Dies hat mehrere Gründe. Wenn sich ein Gegner näher an einem Geschenk befindet und zu diesem läuft, wird er es wahrscheinlich auch früher einsammeln, wodurch der Weg umsonst wäre. Bei den Tresoren muss hingegen noch das 8-Puzzle gelöst werden. Auch wenn der Gegner den Tresor vor dem Agenten erreicht, besteht dennoch die Möglichkeit, dass er das 8-Puzzle schneller löst und den Gegnern so auch noch Punkte wegnimmt. Daher soll der Nutzen des Tresores in der Bewertungsfunktion auch nicht auf Grund eines Gegners verringert werden.

Beispiel

Das folgende Bild zeigt ein Beispiel für die Zielauswahl der Agenten und deren gewählte Route: Man sieht zwei Agenten aus demselben Team und deren berechneten Routen, an deren Ende jeweils ein Tresor liegt. Obwohl

es Geschenke gibt, die näher an den Startpunkten liegen, haben sich beide Agenten auf Grund des höheren Wertes für Tresore entschieden.

Wie man an der Länge der eingezeichneten Routen sieht, hat der obere, blaue Agent bereits eine längere Strecke zurückgelegt. Daher muss er auch sein Ziel bereits vor seinem Mitspieler ausgewählt haben, da sich beide mit der gleichen Geschwindigkeit fortbewegen. Der blaue Agent hat sich also für den Tresor in der Mitte entschieden und ihn fast erreicht. Für den roten Agenten ist dieser Tresor auch deutlich näher als der Tresor links oben. Dennoch wird letzterer Tresor als Ziel gewählt, weil der blaue Agent im TeamChat bereits mitgeteilt hat, dass er zu dem Tresor in der Mitte läuft und dessen Nutzen für den roten Agenten dadurch deutlich sinkt.

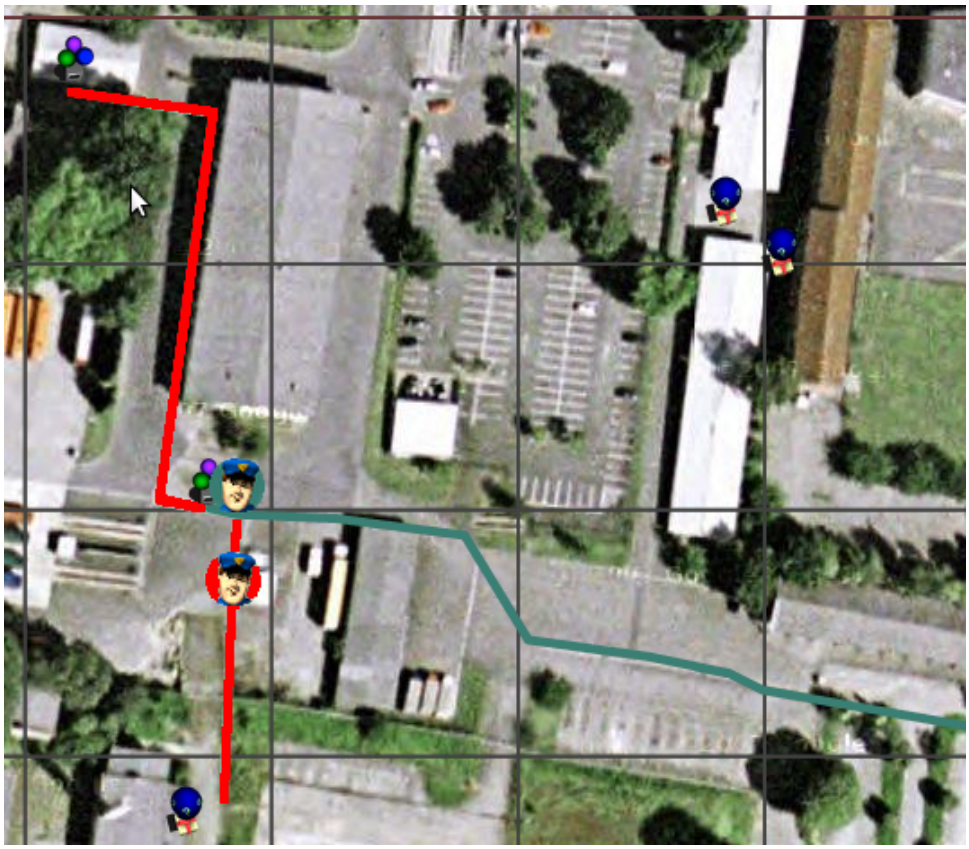


Abb. 62: Zielauswahl zweier Agenten

8.3 Routenberechnung

Sebastian Stock

Hat der Agent ein Ziel ausgewählt, muss als nächstes eine Route berechnet werden. Diese soll möglichst so verlaufen, wie auch menschliche Spieler gehen könnten und würden. Daher bietet sich auch hier das Kartenmaterial von *OpenStreetMaps* an, welches auch schon vom Server für die Verteilung der Tresore und Geschenke verwendet wird. Es wird allerdings nicht direkt mit diesen Daten die Route geplant, sondern ein von *CloudMade*[44] bereitgestellter Webservice zur Routenplanung benutzt.

CloudMade

CloudMade ist ein Unternehmen, das APIs, gerenderte Karten und positionsbasierte Webservices auf Basis von *OpenStreetMaps* anbietet. Einer dieser Webservices führt eine Routenberechnung zwischen zwei

44 http://cloudmade.com

Geo-Koordinaten durch [45]. Der Dienst ist kostenlos. Allerdings ist eine Registrierung notwendig, um einen *API-Key* zu erhalten, der bei jeder Anfrage mitgesendet werden muss. Eine Anfrage muss die Start- und Endpunkte als Geo-Koordinaten, die Art der Fortbewegung (Auto, Fahrrad oder Fußgänger) und das Ausgabeformat (JSON oder GPX) enthalten. Optional sind außerdem Zwischenpunkte, die Sprache der Anweisungen, die Längeneinheit und eine Angabe ob die kürzeste oder die schnellste Route bevorzugt werden soll. Beispielsweise liefert folgende Anfrage eine sehr kurze Route für einen Fußgänger von der Startposition (52.286585, 8.024641), welcher auf der Sedanstraße in Osnabrück liegt, zur Zielposition (52.286547, 8.024300) auf der Barbarastraße, im Datenformat JSON:

```
http://routes.cloudmade.com/BC9A493B41014CAABB98F0471D759707/api/0.3/52.286585,8.024641,52.286547,8.024300/foot.js
```

Die Antwort besteht aus einer Liste von Zwischenpunkten als Geo-Koordinaten, die auf der Route liegen, Fahranweisungen und allgemeinen Routeninformationen wie Gesamtlänge, geschätzte Dauer und Straßennamen des Start- und Zielpunktes. Die obige Anfrage liefert folgendes Ergebnis:

```
{   "version":0.3,
    "status":0,
    "route_summary": {
        "total_distance":28,
        "total_time":20,
        "start_point":"Sedanstraße",
        "end_point":"Barbarastraße"
    },
    "route_geometry": [
        [52.28656,8.02464], [52.286591,8.02429], [52.286549,8.02428]
    ],
    "route_instructions":[
        ["Richtung West auf Sedanstraße", 24, 0, 17, "24 m", "W", 277.2],
        ["Bei Barbarastraße links abbiegen", 4, 1, 3, "4 m", "S", 190.4, "TL", 273.3]
    ]
}
```

Zusätzlich gibt es von *CloudMade* auch eine *Java-API*[46], welche das Erstellen der Anfrage aus Java heraus noch weiter vereinfacht und die Antwort bereits verarbeitet. Damit wird die obige Routenberechnung folgendermaßen durchgeführt:

```
Point start = new Point(52.286585, 8.024641);
Point goal = new Point((52.286547, 8.024300);
CMClient cmClient = new CMClient(APIKEY);
Route route = cmClient.route( start, goal, RouteType.FOOT, null, null, "en", MeasureUnit.KM);
```

Verarbeitung der Route und aufgetretene Probleme

Die so erhaltene Route kann nicht direkt übernommen werden, sondern muss noch an die Bedürfnisse des Agenten angepasst werden. Ausgangsbasis ist eine Liste von Zwischenpunkten vom Typ `Point`, welche in `Route` enthalten ist. Diese besteht aus allen Eckpunkten der Route, zwischen denen sie gradlinig verläuft. Kurven der Straße werden so bereits von *CloudMade* durch eine größere Anzahl von Zwischenpunkten interpoliert, wodurch auch Routen auf nicht geradlinigen Straßen gut eingezeichnet werden können, indem diese Zwischenpunkte durch Linien verbunden werden, wie sich später in der Darstellung im *Gameviewer* zeigen wird.

Eine Route als Liste von Punkten ist für den Agenten vorteilhaft, da er so für die Simulation seiner Bewegnung nur über diese iterieren muss. Für eine flüssige Bewegung liegen die von *CloudMade* berechneten Punkte allerdings teilweise noch zu weit auseinander. Ziel ist es, dass zwischen zwei Updates der eigenen Position eine vorgegebene Dauer nicht überschritten wird. Daher werden zwischen zwei Punkten der Route weitere eingefügt, deren Abstand der Agent in der vorgegebenen Zeit zurücklegen kann.

45 http://developers.cloudmade.com/projects/show/routing-http-api
46 http://developers.cloudmade.com/documentation/java-lib/index.html

Es sind teilweise Probleme mit den Startpunkten der von *CloudMade* berechneten Route aufgetreten. Der übergebene Startpunkt lag bei Tests mehrmals nicht am Anfang der berechneten Route, sondern in der Mitte. Dadurch lief der Agent in diesen Fällen erst zurück in die falsche Richtung. Um dieses abzufangen muss zunächst in der Liste der von *CloudMade* berechneten Punkte derjenige Punkt gesucht werden, der den geringsten Abstand zum Startpunkt hat und die vorherigen Punkte müssen verworfen werden. Die auf diese Weise generierte Route wird an den Server gesendet um sie im *Gameviewer* anzuzeigen. So kann einfach verfolgt werden, welchen Plan der Agent verfolgt.

9. Gameviewer

Andre Schemschat

Der Gameviewer basiert auf verschiedenen Technologien, die ineinandergreifen um alle Aspekte aufzubereiten. An unterster Stelle steht eine Django-Applikation die sich um das Anzeigen der Webseiten kümmert und die Callbacks für die Datenanfragen liefert.

Django[47] ist ein in Python geschriebenes Webframework, dass sehr stark auf dem Model-View-Controller-Konzept beruht. Dabei liefert es eine All-in-One-Lösung die es dem Entwickler erlaubt sehr schnell vorhandene Komponenten zu verwenden und in wenigen Schritten eine Webapplikation zusammenzustellen. Zu diesen Komponenten gehören unter anderem eine Template-Engine, ein Datenbank-ORM, eine Formular-Verwaltung, automatisierte Daten-Backends und teilautomatisierte Authentifizierung neben dem Kern der Anfrage-Verwaltung.

Nachdem die Seite geladen worden ist wird die Kontrolle hauptsächlich durch mehrere Javascript-Objekte ausgeübt. Diese initialisieren alle nötigen Hierarchien und bauen eine interne Nachrichtenschnittstelle auf, über die zentralisiert Events ausgetauscht werden können. Jedes einzelne der Objekte ist dabei für einen eigenen Bereicht zuständig und ermittelt mittels der Callbacks auf dem Server und einem JSON-Austausch-Format die nötigen Daten um alle Informationen anzuzeigen. Zur Darstellung und Anfrage setzen diese Skripte sehr stark auf das frei verfügbare Framework jQuery[48]. Es kapselt oft wiederkehrende Funktionen und abstrahiert von dem zugrundeliegenden Browser, so dass z.B. Ajax-Requests sehr einfach werden. Die gesendeten Daten und die Schnittstellen des Gameviewers sind dabei unabhängig von dem aktuellen Modi, lediglich die Callback-Implementierung unterscheidet sich in ihrer Funktionsweise. Dazu jedoch in den zwei Unterabschnitten mehr.

Zur Anzeige des Kartenmaterials wird die Google-Maps-Api[49] in der Version 3 verwendet. Diese API ermöglicht die einfache Anbindung von Karten und bietet Methoden um Polygone und Icons in Ebenen darüber anzulegen.

```
var latlng = new google.maps.LatLng(52.23, 8.07);
var myOptions = {
    zoom: 8,
    center: latlng,
    mapTypeId: google.maps.MapTypeId.ROADMAP
};
var map = new google.maps.Map(document.getElementById("map"),
myOptions);

var position = new google.maps.LatLng(52.23, 8.07);
var marker = new google.maps.Marker({
    position: position,
    map: map,
    flat: true,
    title: "My Marker"
});
```

Ausschnitt einer Javascript-Funktion und der Google-Maps-API

```
def live_get(request):
    try:
        type = request.GET['type']
        time = request.GET.get('time', None)
        game_id = request.GET.get('game_id', None)
        player_id = request.GET.get('player_id', None)

        func = {
            'game_list' : s_live.game_list,
            'game_time' : s_live.game_time,
            'game' : s_live.game,
            'area_visibilities': s_live.area_visibilities,
            'players' : s_live.players,
```

47 http://www.djangoproject.com
48 http://jquery.com
49 http://code.google.com/intl/de-DE/apis/maps/documentation/javascript/

```
            'teams': s_live.teams,
            'safes' : s_live.safes,
            'tracks' : s_live.tracks,
            'gifts': s_live.gifts,
            'inventory': s_live.inventory,
            'chat': s_live.chat,
            'agents': s_live.agents}[type]

        # calling function and dumping result
        res = func(time, game_id, player_id, request.GET)
        return HttpResponse(simplejson.dumps(res))

    except KeyError:
        return HttpResponse("null")
    except SpectatorException:
        return HttpResponse("null")
```

Django(Python)-Funktion, die die Anfragen verarbeitet

9.1 Realtime-Tracking

Andre Schemschat

Einer der Modi des Gameviewers ist das *Realtime-Tracking*. Dieser holt mit Hilfe eines speziellen Sets von Callbacks immer den aktuellen Spielstand. Dabei werden alle nötigen Parameter zwar mitgeschickt, doch der Zeitstempel wird jeweils ignoriert. Die Daten werden in beiden Modi gleich aufbereitet und dann an die spezielle Implementation weitergereicht. Diese kontaktiert den Server über die normale Netzwerkschnittstelle, die ebenfalls von den Smartphone-Klienten genutzt wird. Die gesendete Anfrage unterscheidet sich nur geringfügig, doch der Gameviewer sendet weitere Meta-Informationen mit, die dem Server erlauben das betrachtete Spiel und den Spieler zu ermitteln. Die Antwort des Servers ist im gewohnten JSON-Format und wird über den Gameviewer an die Webseite weitergeliefert, wie das unten stehende Beispiel zeigt.

```
def game_time(time, game_id, player_id, get=None):
    gameinfo = request("get_gameinfo", gameid=game_id)

    return simplejson.dumps({
            'start_time' : gameinfo['startingtime'],
            'end_time' : int(time.time()*1000),
            'live' : True
            })
```

9.2 History-Tracking

Daniel Künne

Der zweite Modus des Gameviewers dient zum Betrachten von bereits abgelaufenen Spielen. Da es in diesem Fall nicht möglich ist, Anfragen direkt an den Server zu stellen, muss eine Zwischenauswertung erfolgen. Diese übernimmt die *Django*-Applikation, von wo aus Anfragen direkt an die Datenbank gestellt, aufbereitet und dann im gleichen Format wie beim Realtime-Tracking an den Gameviewer zurückgeliefert werden.

```
def game_time(time, game_id, player_id, get=None):
    c = ocpgdb.connect(**webadmin_db)
    curs = c.execute("""SELECT round(date_part('epoch',start_time)),
round(date_part('epoch',end_time))
                        FROM games_history
                        WHERE id = %d"""
                        % (int (game_id)))
    c.close
    if len(curs) == 1:
        return {
```

```
            'start_time' : curs[0][0] * 1000,
            'end_time' : curs[0][1] * 1000,
            'live' : False,
        }
    return None
```

Obige Funktion erfüllt die gleiche Aufgabe wie das Code-Beispiel im Realtime-Tracking allerdings werden die Unterschiede schnell deutlich. Hier muss eine SQL-Anfrage an die Datenbank gestellt werden, welche einen Datensatz zurückliefert. Anschließend müssen die erhaltenen Werte noch in das vorgegebene JSON-Format gebracht werden.

10. Fazit

Daniel Künne, Andre Schemschat

Der ursprüngliche Gedanken der Projektgruppe war eine Evaluation der Fähigkeiten von aktuellen Smartphones. Es sollten die verschiedenen Komponenten genutzt, kombiniert und an ihre Grenzen gebracht werden um die Möglichkeiten für zukünftige Projekte auszuloten. Im Verlauf des Projektes wurden dazu nach und nach Spielideen diskutiert und verfeinert. Viele dieser Ideen wurden wieder verworfen, zum Teil sprengten sie den Rahmen der Arbeit, wieder andere waren noch nicht umsetzbar, da die Hardware der Smartphones noch nicht weit genug war.

Je weiter die Zeit voranschritt, umso deutlicher kristallisierte sich eine Idee heraus. Es formte sich immer mehr ein Spiel, auch wenn immer wieder Probleme unterschiedlichster Art auftauchten. Viele der Probleme, die bei der Umsetzung auf den Handies auftauchten, konnten behoben werden, einige vorher vermutete Grenzen der Systeme stellten sich als noch nicht überwindbar heraus und wieder andere waren zu Beginn gar nicht sichtbar und zeigten sich erst spät in der Entwicklung.

Nichtsdestotrotz haben sich in einem Jahr mehrere Applikationen gebildet, die die verschiedenen Komponenten in sich vereinen und darüber hinaus erlauben ein bekanntes, klassisches Spiel in neuem Gewand zu spielen. War die Resonanz auf dem Technologietag, mit einer frühen Beta-Version als Demo, noch gemischt, so hat sich nach dieser anfänglichen Lernphase eine Richtung herausgebildet. Dementsprechend haben wir auf der Cebit ein durchweg positives Feedback von verschiedenen Personengruppen erhalten. Darüber hinaus zeigten sich einige Interessenten für das zugrundeliegende Framework des Spiels und dem angesammelten Know-How der Entwickler.

Abschließend lässt sich sagen, dass das Projekt zu einer guten Basis, auch für kommerzielle Produkte, geworden ist.

Abb. 63: Foto vom Technologietag

Abb. 64: Foto von der CeBIT