Android-Apps Programmierung

Komplexe Leistung

von David Tiede

in Informatik

Marienberg, den 7. Oktober 2013

Komplexe Leistung

Thema:	Android-Apps Programmierung
Verfasser:	David Tiede
Schule:	Gymnasium Marienberg
Schuljahr:	2012/2013
Klassen:	11/4 und 12/4
Fach:	Informatik
Fachlehrer:	Herr Buttke
Ausgabetermin:	30.11.2012
Abgabetermin:	18.10.2013

_____ _____

Unterschrift des Schülers Unterschrift des Fachlehrers

Inhaltsverzeichnis

1 Vorwort

Das Thema Programmierung hat mich, seit ich mit Rechnern zu tun habe, interessiert. Dabei habe ich zuerst in Basic geschnuppert. Aufgrund der Betreuung von mehreren Webseiten habe ich PHP in der Praxis genutzt. In einem Praktikum habe ich mit meinem Programm in Python den Einsatz im Produktivsystem gesehen.

Gerade durch die Verbreitung der Smartphones hat mich das Thema App-Programmierung immer mehr interessiert. Nachdem ich nun mein eigenes Android-Handy hatte und ein Thema für die Komplexe Leistung brauchte, stand dem nichts mehr im Weg.

Diese Arbeit setzt ein gewisses Grundverständnis vom Betriebssystemaufbau und der Programmierung voraus.

An dieser Stelle möchte ich mich bei meinem betreuenden Lehrer Herrn Buttke für die Begleitung bei der Erstellung der Komplexen Leistung bedanken. Auch meinen Eltern sei herzlich gedankt, für jegliche Unterstützung bei meiner über 60-stündigen Arbeit.

1.1 These und Aufbau

Der MIT AppInventor kann eine professionelle Programmierumgebung wie das Android SDK ersetzen. Mit dieser These habe ich mich in dieser Arbeit auseinander gesetzt.

Mit dem MIT AppInventor werden Apps für die mobile Plattform Android erstellt. Um den Aufbau und die Funktionsweise einer App unter Android zu verstehen, ist es deshalb wichtig, Android als Plattform näher kennen zu lernen.

Um den MIT AppInventor gut vergleichen zu können, habe ich an mehreren Beispielen die Erstellung einer Android-App nachvollzogen. Dabei habe ich als Vergleichsprogramm das von Google empfohlene Android SDK benutzt.

2 Android – eine offene, mobile Plattform

2.1 Entstehung

Der US-amerikanische Softwareentwickler Andrew Rubin gründete 2003 die Firma Android Inc. für die Entwicklung eines freien mobilen Betriebssystem. Google-Gründer Larry Page war von der Idee begeistert und übernahm 2005 Android für 50 Mio. USD.

Am 5. November 2007 kündigte Google an, dass es ein Mobiltelefon-Betriebssystem gemeinsam mit den Mitgliedern der Open Handset Alliance entwickle. Die Open Handset Alliance (OHA) ist ein Konsortium aus Mobiltelefonherstellern, Netzbetreibern und Softwareentwicklern. Das Ziele waren eine deutlichen Verbesserung der Benutzerfreundlichkeit von mobilen Geräten, eine kostengünstige Entwicklung innovativer Produkte und einer schnellere Markteinführung. Um diese Ziele zu verwirklichen, setzte man auf Offenheit und Flexibilität, sodass alle Teile des Android-Softwarestapels als Open Source veröffentlicht wurden.

Im Herbst 2008 wurde das erste Gerät mit Android auf den Markt gebracht. Das HTC Dream oder T-Mobile G1 war ein Smartphone mit Touchscreen, das für das Internet und insbesondere für die Dienste von Google optimiert war.

Google hat mit beachtlichen Tempo an Android weitergearbeitet und dabei stetig Anwendungen verbessert und neue Funktionen eingeführt. Oft lagen zwischen den neuen Versionen nur wenige Monate.

Das die entsprechenden Aktualisierungen oft erst nach langer Zeit herausgegeben werden, liegt daran, dass die Gerätehersteller das bestehende System oft erheblich anpassen und verändern. Änderungen müssen somit in jede neue Android-Version eingearbeitet werden.

2.2 Versionsverlauf

Die neuen Haupt-Versionen tragen neben der Versionsnummer jeweils noch den Namen einer Süßspeise, deren Anfangsbuchstabe im Alphabet aufsteigend ist.

Cupcake (Android 1.5) ermöglichte erstmals die Verwendung von Geräten ohne Hardwaretastatur. Auch wurde bei dieser Version schon die Spracherkennungsfunktion integriert.

Donut (Android 1.6) unterstützte bereits die Gestenerkennung. Android konnte jetzt auch bei unterschiedlichen Bildschirmauflösungen ausgeführt werden. Eine Neuerung war auch die

Sprachsynthesesoftware (Text-to-Speech) für Deutsch, Italienisch, Englisch, Französisch und Spanisch.

Eclair (Android 2.0.x / 2.1) bot eine neue Programmierschnittstelle für den Zugriff auf Kontakt-und Kontodaten an. Somit konnte man die Daten jetzt auch mit anderen Diensten z.B. Facebook, Microsoft synchronisieren. Neu waren auch die animierten Hintergrundbilder.

Mit **Froyo** (Android 2.2.x) konnte man nun endlich die Apps auf der SD-Karte speichern (App2SD) und auch mehr als 265MB Arbeitsspeicher nutzen.

Seit **Gingerbread** (Android 2.3.x) kann man die Near Field Communnication (NFC) und auch die Internettelefonie über VoIP nutzen. Die Geschwindigkeit des Systems wurde unter anderen durch die Dual-Core-Unterstützung verbessert.

Honycomb (Android 3.x.x) wurde speziell für Tablets entwickelt. Es wurde besonders Wert auf eine benutzerfreundliche Oberfläche gelegt.

Ice Cream Sandwich (Android 4.0.x) führte die Entwicklungslinien von 2.x und 3.x wieder zusammen. Die mitgelieferten Anwendungen wurden überarbeitet und benutzerfreundlich gestaltet. Neue Features waren das entsperren per Gesichtserkennung und der Zugriff auf Kalendereinträge.

Mit **Jelly Bean** (Android 4.1.x / 4.2.x / 4.3) wurde Googe Now, ein persönlicher Assistent mit Spracherkennung, integriert. Nur bei den Tablets konnte man jetzt Benutzerkonten anlegen. Jelly Bean ist die zur Zeit am weitesten verbreitete Android-Version.[1]

Zu **Key Lime Pie** (Android 5.0) wurde offiziell noch nichts veröffentlicht. Die Vorstellung wird wahrscheinlich am 29. Oktober 2013 sein.

2.3 Systemarchitektur

Die Grundlage von Android ist ein Vier-Schichtenmodell:

1 Stand: Juli 2013

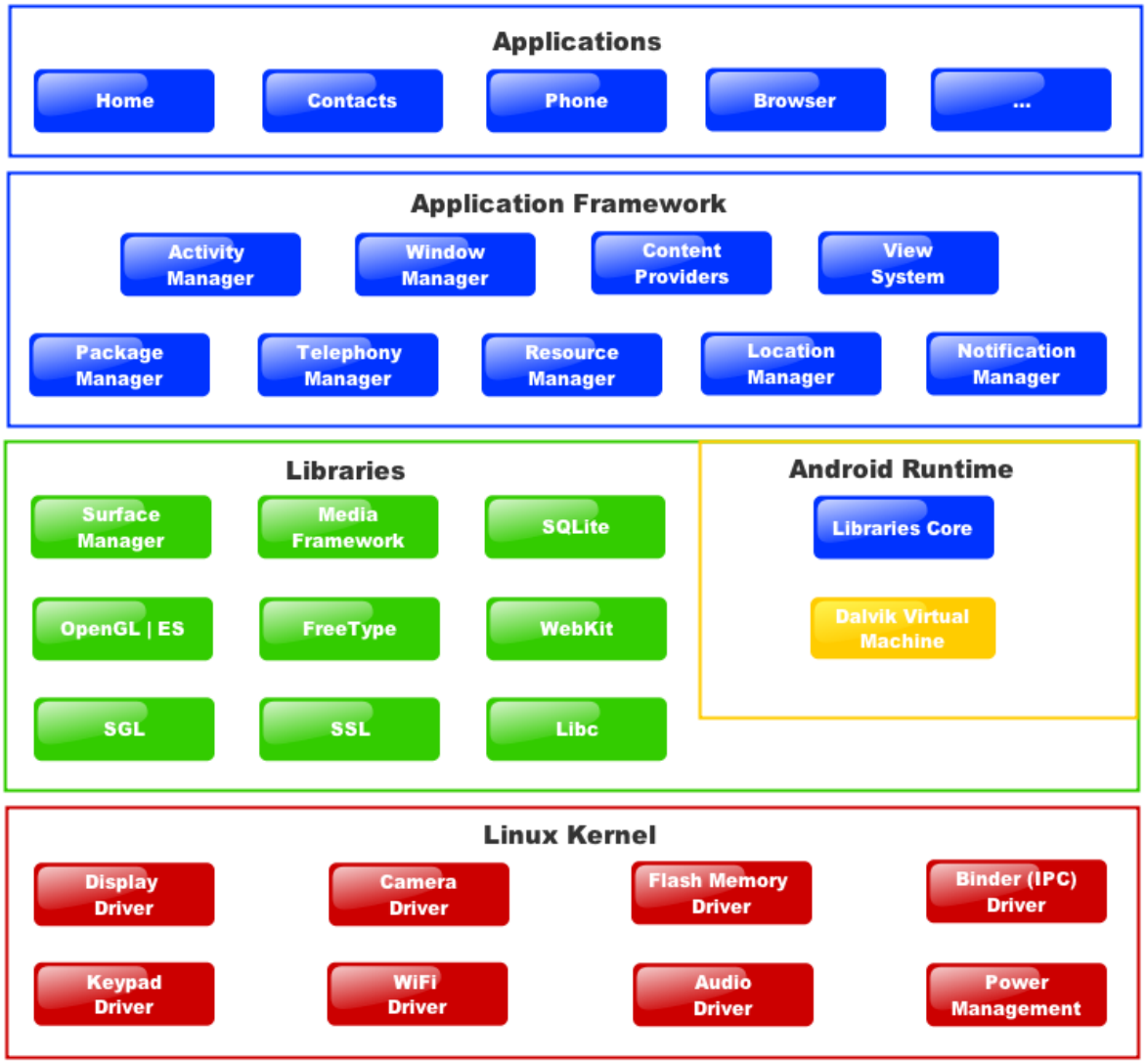

Abbildung 1: Aufbau der Android-Plattform: upload.wikimedia.org, 14.8.2013

2.3.1 Linux Kernel

Als Betriebssystem, das Systemdienste wie Prozess-und Speicherverwaltung, Teibermodell und Netzwerkstabel mitbringt, wird bei Android der Linux Kernel 2.6 verwendet. Es ist gleichzeitig auch die Abstraktionsschicht, die die Hardware von den übrigen Teilen der Plattform abkapselt.

2.3.2 Applications (Anwendungen)

Android ist aber viel mehr als nur ein Betriebssystem. Zu Android gehören beispielsweise noch eine Reihe von Standartanwendungen, wie Browser, E-Mail-Client, Anwendungsstarter (Launcher). Auch die Plattform Google Play ist ein fester Bestandteil von Android.

2.3.3 Android Runtime (Laufzeitumgebung)

Ein wichtiger Bestandteil von Android ist die Dalvik Virtual Machine. Diese führt fast alle

Anwendungen aus, die der Nutzer auf einem Android-System startet. Ausgenommen sind Programmteile, die mit dem Android Native Development Kit geschrieben wurden.

Die Anwendungen werden so nicht direkt von dem Prozessor abgearbeitet, sondern von einer Zwischenschicht, ähnlich wie bei Java, interpretiert. Bei Java wird der Quelltext durch den Compiler in den Bytecode umgewandelt, der später von der Java Virtual Machine ausgeführt wird. Bei Android wird der Quelltext, der auch in der Sprache Java geschrieben wird, in Dalvik Executables umgewandelt. Diese werden dann von der Dalvik Virtual Machine ausgeführt.

Der zweite Teil der Android Runtime sind die Core Libaries. Sie enthalten die von Java bekannten Pakete wie java.lang, java.io, java.math, java.net, java.security, java.util und java.text.

2.3.4 Application Framework (Anwendungsrahmen)

Für die Benutzeroberfläche, die Steuerung einer Anwendung, die Telefoniefunktionen und Multimedia gibt es Android-eigene Programmierschnittstellen, die man im Application Framework findet.

Mit dem Application Framework gestaltet sich die Erstellung von Apps wesentlich einfacher. Es bietet Zugriff auf die Gerätehardware, wie Kamera, Netzwerk oder Sensoren, und steuert auch den Zugriff auf Kontakt-oder Kalenderdaten. Dafür hat es ein integriertes Rechtesystem, das die Zugriffe entsprechend überwacht.

Beispielsweise steuert der Activity Manager die Anwendungen und verwaltet deren Lebenszyklus. Der Content Provider regelt dabei den Austausch von Daten, z.B. Kontakten mit anderen Anwendungen. Das View System bildet die Basis der Benutzeroberfläche. Es liefert zahlreiche Standartelemente, wie etwa Textfelder, Schaltfläche oder Listen, während man mit dem Resource Manager auf Grafiken-und Layoutdateien zugreifen kann.

Der Notification Manager regelt die Benachrichtigungen in der Statusleiste. Weiterhin gibt es den Telephony Manager und den Location Manager.

2.3.5 Libraries (Bibliotheken)

Die C/C++-Bibliotheken werden von verschiedenen Komponenten der Plattform genutzt. So greifen Entwickler indirekt über den Application Framework auf sie zu.

Das Media Framework ermöglicht die Aufnahme und Wiedergabe von zahlreichen Medien.

3 MIT App Inventor

Der MIT App Inventor[2] von Google ist für das Erstellen von Apps über eine grafische Benutzeroberfläche. Dabei werden per Drag and Drop grafische Code-Blöcke ähnlich wie bei Scratch miteinander verbunden.

Seit Dezember 2010 hat die Anwendung das Ziel, den Einstieg in die Programmierung von Apps zu vereinfachen und dadurch vor allem junge Menschen mehr an Android zu binden. Seit März 2012 ist der App Inventor für die Öffentlichkeit verfügbar.

Für den App Inventor gibt es viele englischsprachige Tutorials[3], die den Einstieg in das Programm gut ermöglichen.

3.1 Setup

Zuerst muss der App Inventor eingerichtet werden.[4] Dafür wird ein Google Konto benötigt und Java Runtime Environments (JRE) muss installiert sein.

Der App Inventor besteht aus zwei Teilen, zum einen ist das der App Inventor Designer im Internet und zum anderen der Blocks Editor. Diese Java-Programm muss heruntergeladen und installiert werden.[5]

Anschließend erstellt man im Designer unter http://beta.appinventor.mit.edu ein neues Projekt und öffnet über dieses dann den Blocks Editor. Daraufhin wird die Projektdatei heruntergeladen, die man anschließend ausführt, sodass sich der Blocks Editor öffnet.

Beide Programmteile werden in Echtzeit synchronisiert und alle Projekt-Daten werden sofort im Internet gespeichert.

Um die App zu testen, kann man zum einen den integrierten Emulator verwenden. Die Einrichtung von diesen erfolgt automatisch. Zum anderen kann man das eigene Android-Handy verwenden, das wahlweise wieder über W-Lan oder USB verbunden werden kann.

Bei der USB-Verbindung muss ein entsprechender Treiber installiert werden und im Handy muss die Optionen „Andere App-Quellen“ und „USB debugging“ aktiviert werden. Für die W-Lan-Verbindung reicht es aus, die App MIT AICompanion[6] zu installieren.

---

2 http://appinventor.mit.edu/
3 http://appinventor.mit.edu/explore/tutorials.html
4 http://appinventor.mit.edu/explore/setup-mit-app-inventor.html
5 http://appinventor.mit.edu/explore/content/windows.html
6 https://play.google.com/store/apps/details?id=edu.mit.appinventor.aicompanion2

3.2 Benutzeroberfläche

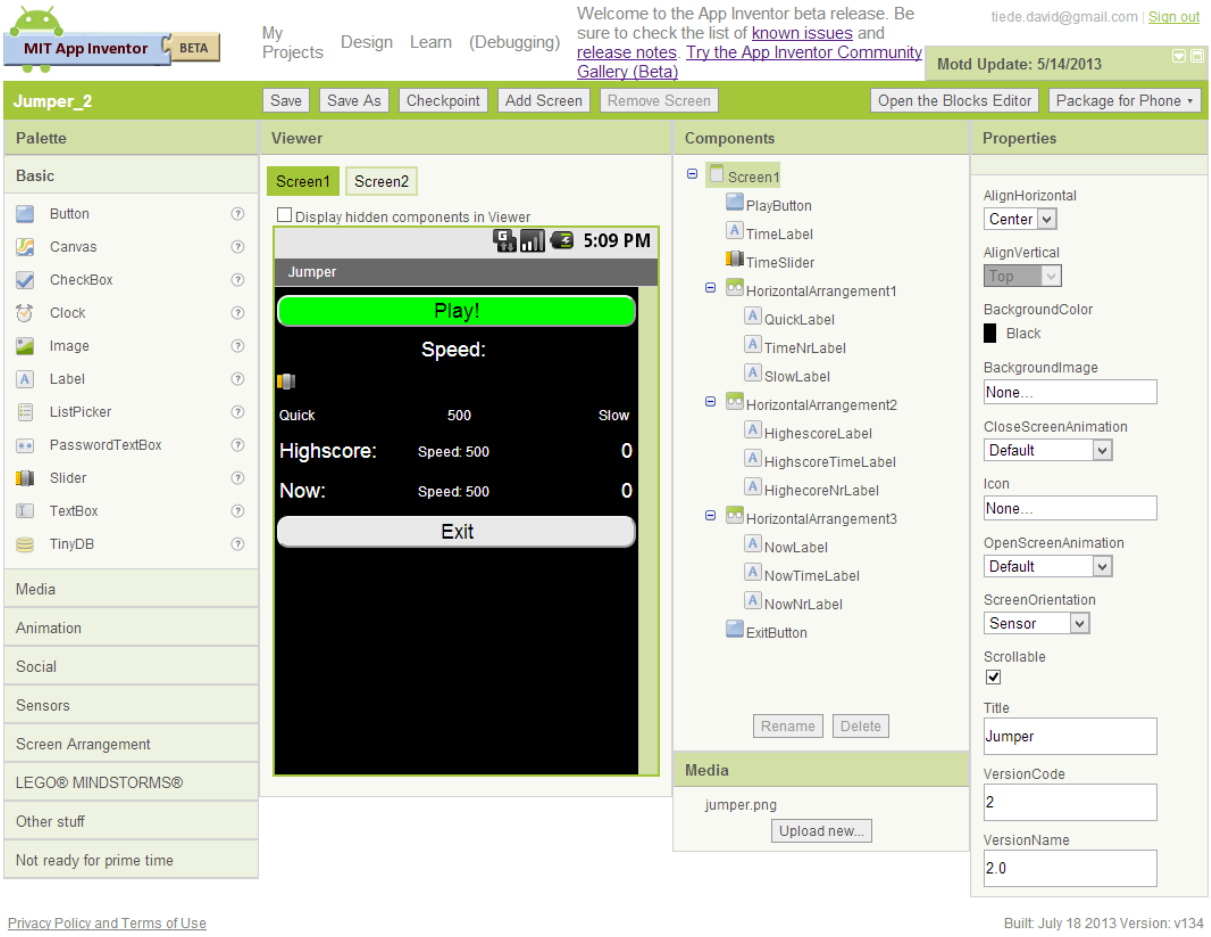

Abbildung 2: MIT App Inventor Designer, 19.8.2013

Der App Inventor Designer ist in 4 große Bereiche untergliedert. In der Palette findet man verschiedene Komponenten, die die Basiselemente einer App sind. Per Drag and Drop werden diese in den Viewer geschoben und dort entsprechend angeordnet. Alle verwendeten Komponenten findet man in der Spalte Components. In der rechten Spalte Properties werden die Eigenschaften der gerade gewählten Komponente angezeigt.

Der App Inventor Blocks Editor hat ein Menü mit den verschiedenen Code-Puzzleteilen, die per Drag and Drop auf die Puzzlefläche geschoben werden und dort zusammen gesteckt werden. Dabei wird durch die Ausbuchtungen deutlich, was zusammen passt. Sollte ein Puzzleteil trotz gleicher Ausbuchtung nicht zusammenpassen, liefert der Blocks Editor eine entsprechende Meldung mit dem Fehler.

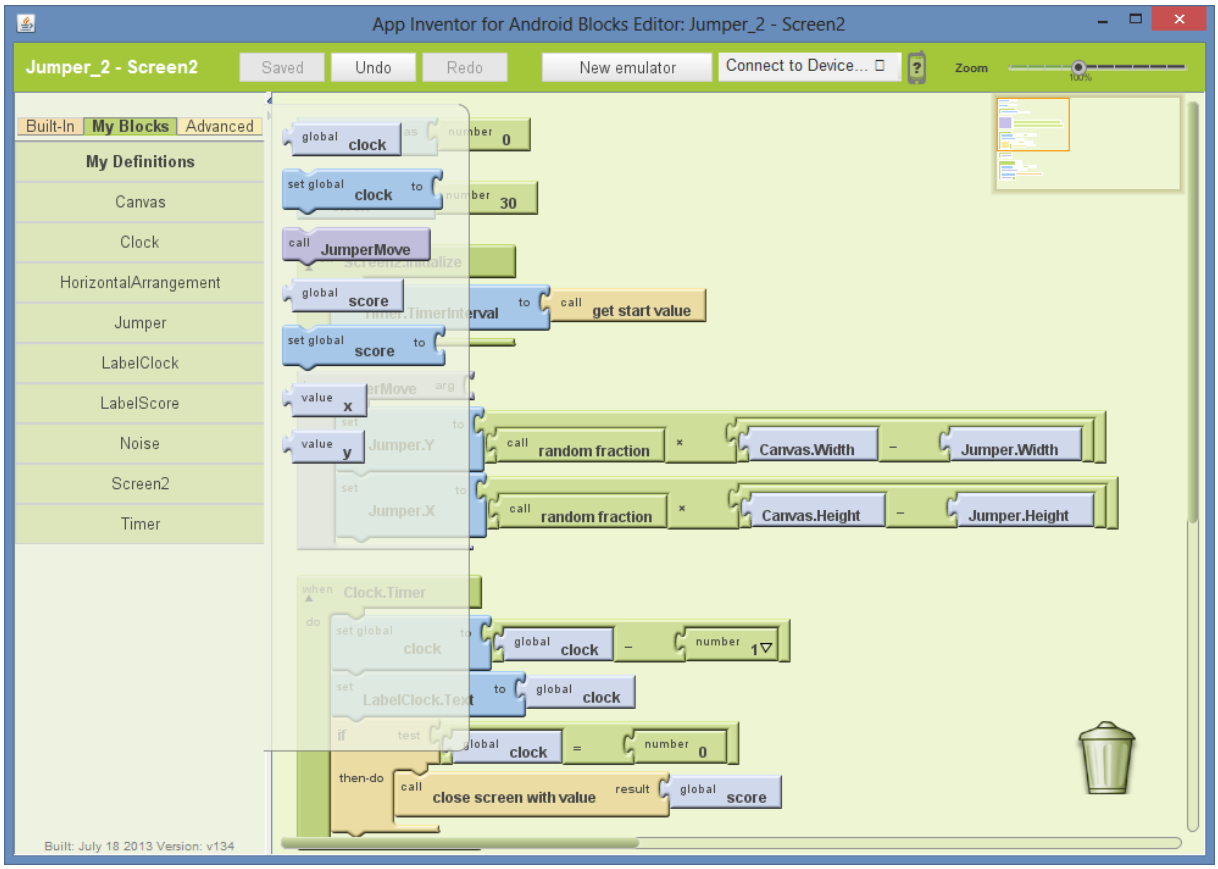

Abbildung 3: MIT App Inventor Blocks Editor, 19.8.2013

Ist der Blocks Editor mit einem Emulator oder Testhandy verbunden, wird das Desigen und auch der Programmcode sofort übertragen und man kann es sofort testen.

3.3 Erste App: HelloWorld

Als erste App habe ich HelloWorld erstellt. Diese App begrüßt einen mit „Hello World!“ und kann nach Namenseingabe auch persönlich begrüßen.

Die App besteht nur aus 5 Componenten: das Label mit „Hello World!“, das Label und die Textbox für die Namenseingabe, die in einer Horizontalkomponente angeordnet sind, und einen Button.

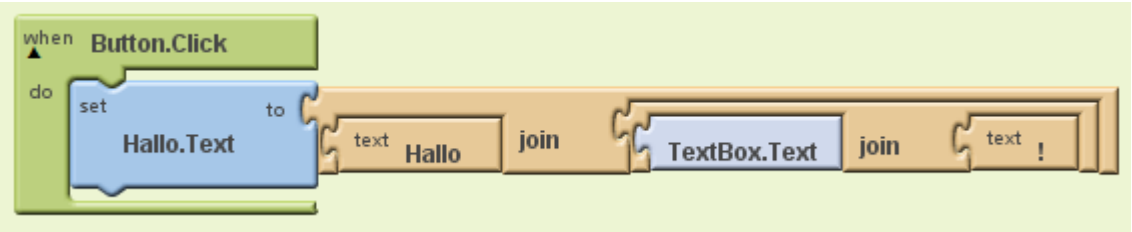

Abbildung 4: Bocks Editor "HelloWorld", 19.8.2013

Der Code ändert bei einem Klick auf den Button das „World“ in den Namen der TextBox um.

3.4 App: Jumper

Die Idee zu der Spiele-App kam beim Durchsuchen der Tutorials.[7] Ziel das Spieles ist es, den herumspringenden „Jumper“ zu erwischen.

Bei Jumper_1 gab es nur eine MainActivity (Oberfläche), die ich größtenteils nach der Anleitung gestaltet. Die Canvas-Komponente bildet mit dem bewegten Bild von „Jumper“ den zentralen Teil der Spieleoberfläche.

Was noch fehlte war ein kleines Menü und eine Highscore. Da es bei AppInventor keine Möglichkeit gab, die MainActivity, dass ist die Oberfläche die standardmäßig aufgerufen wird, zu ändern, habe ich mir den Quelltext heruntergeladen und diesen manuell bearbeitet. Anschließend habe ich das Spiel und das Menü weiter zur Version 2 ausgearbeitet.

3.5 Veröffentlichung

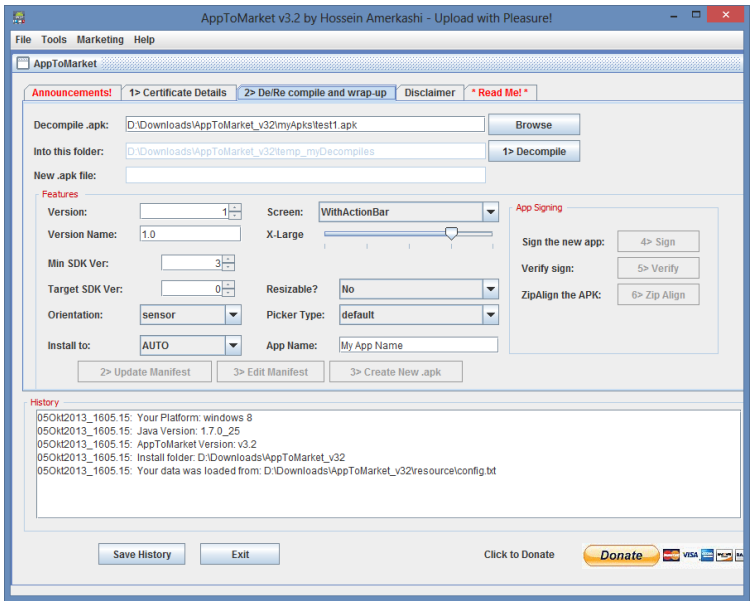

Abbildung 5: AppToMarket, 19.8.2013

Die Veröffentlichung einer App von AppInventor ist ohne Weiteres nicht möglich. Die App muss dazu erst mit entsprechender Drittsoftware verändert werden.

Mit der Anwendung AppToMarket wird die eigentlich fertige App dekompiliert und Änderungen z.B. der Signatur vorgenommen.

Danach kann die App veröffentlicht werden.

7 http://appinventor.mit.edu/explore/content/molemash.html

4 Android Software Development Kit (SDK)

4.1 Setup

Seit ein paar Monaten gibt es das Android Developer Tools Bundle, das die Einrichtung von Android SDK wesentlich vereinfacht. In diesem Paket befindet sich nun Eclipse mit dem ADT Plugin, Android SDK Tools und Android Platform Tools. Wenn Java Runtime Environments (JRE) installiert ist, muss man das Paket nur noch entpacken.

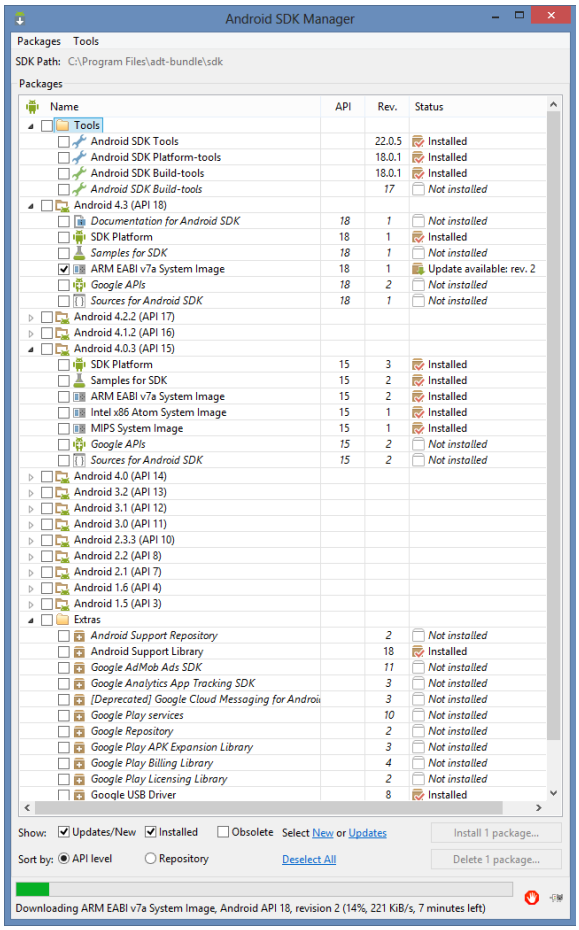

Abbildung 6: Android SDK Manager, 19.8.2013

Anschließend lädt man im Android SDK Manager die notwendigen Packages für die App-Entwicklung und den Test. Spätestens wenn man die App für GooglePlay vorbereitet, sollte man diese unter möglichst vielen Versionen testen.

Nach dem Download startet man den AVD Manager um die Android Virtual Devices (AVD) zu erstellen. AVD's fassen Einstellungen der Emulatoren, die bestimmtes physikalisches Gerät nachbilden, zusammen. Dazu gehört das Hardwareprofile, wie Kamera oder Tastatur, das Systemabbild, also die Android-Version, und der zu verwendende Emulator-Skin. Verschiedene Geräte, wie die Nexus-Reihe von Google, sind schon vorgegeben, es könne aber auch beliebige Geräte erstellt werden.

4.2 Benutzeroberfläche

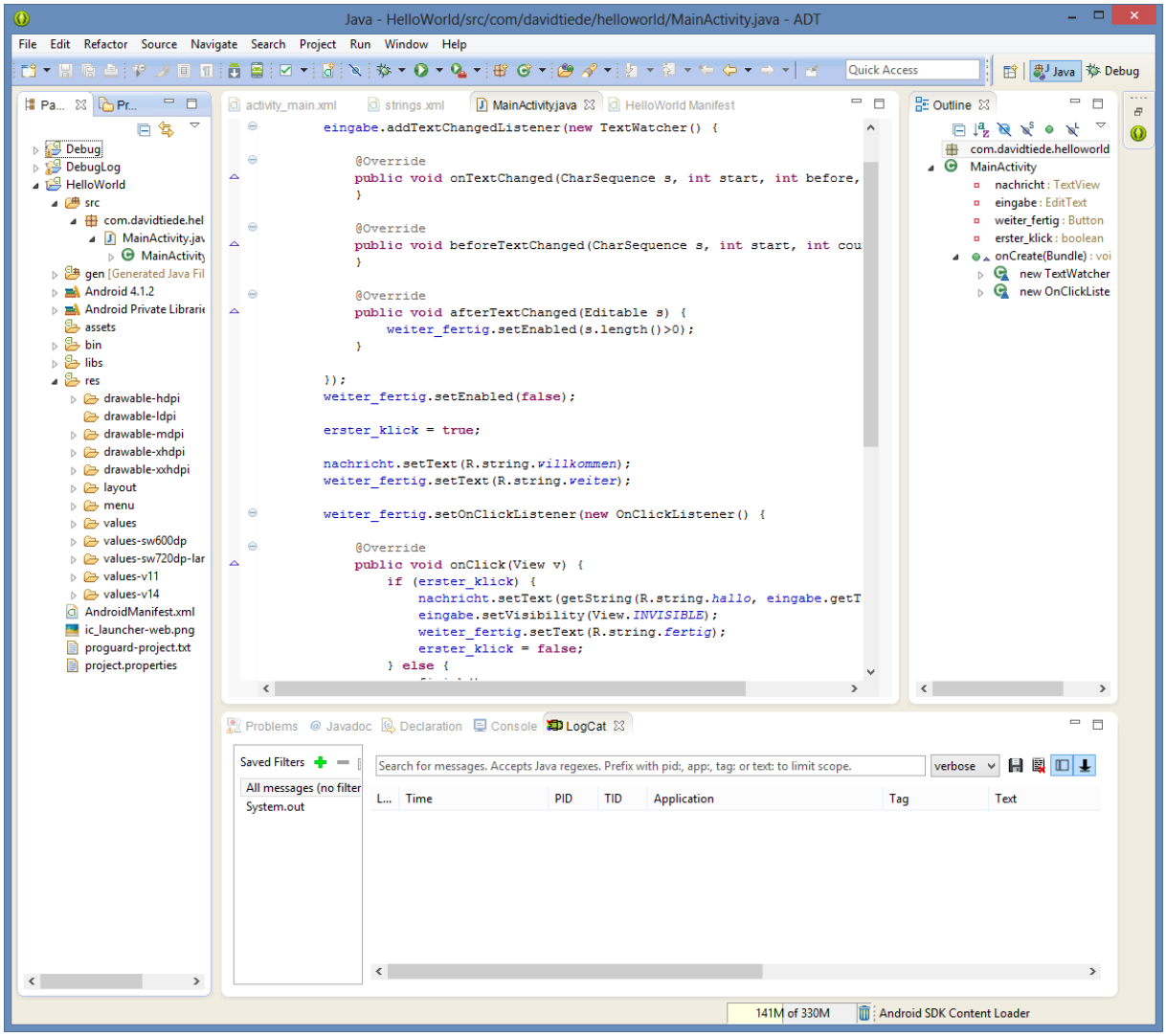

Abbildung 7: Benutzeroberfläche Eclipse, 19.8.2013

Für die eigentliche Programmierung verwendet man das Programm Eclipse. Ursprünglich war es für die Entwicklung mit Java gedacht, doch durch zahlreiche Erweiterungen wird es heute für verschiedene Softwareprojekte verwendet. Die Oberfläche besteht aus vielen einzelnen Fenstern, die sich beliebig anordnen lassen.

In der Standard-Perspektive „Java“ findet man links den Projektmanager mit der Übersicht über die Projektdaten. Rechts findet man das Outline, das den Programmcode in der Mitte strukturell visualisiert. Unter dem Programmcode findet man in den Tabs Problems, Javadoc und Declaration Infos zu dem aktuellen Programmcode. Auch die Konsole und der LogCat, der die Log-Meldungen vom Emulator oder Handy ausgibt, lässt sich dort finden.

Eine zweite wichtige Perspektive ist Debug. Hier findet man viele Infos, z.B. den aktuellen

Variabelinhalt, zur Laufzeit der App um Fehler zu finden.

4.3 Erste App: HelloWorld

Wie bei dem AppInvetor habe ich auch mit Eclipse eine ähnliche HelloWorld App programmiert.

Das Layout kann entweder per Drag and Drop oder mit einer XML-Datei erstellt werden. Der eigentliche Programmcode befindet sich in der MainActivity, der zuerst aufgerufenen Activity. Als Activity wird eine sichtbar Oberfläche bezeichnet. Activitys sind Klassen die unter anderen die Methode „onCreate“ haben. Mit dieser Methode wird das Layout aus der XML-Datei erstellt und die entsprechenden Beschriftungen aus einer zentralen String-Datei geladen.

4.4 App: ListsHelper

Der ListHelper ist eine einfache App, mit der man Listen, z.B. Einkaufslisten, erstellen und verwalten kann. Die App besteht aus zwei Activitys, die „MainActivity“ zeigt die Übersicht der Listen und die „EditActivity“ zeigt die Listeneinträge.

Für die App-Erstellung bietet Android eine ganze Mengen an Klassen, die die Erstellung von Apps erleichtern sollen, indem sie wichtige Grundfunktionen liefern. Für den ListsHelper habe ich unter anderen Gebrauch von der Klasse „ContentProvider“ gemacht. Diese verwaltet die Speicherung von Daten einer App in einer Datenbank, meistens in einer SQL-Datenbank.

4.5 Veröffentlichung

Damit die App auch unter Android installiert werden kann, muss man die App mit einem Zertifikat des Entwicklers signieren. Dieses System dient der Identifizierung von Entwicklern und damit auch dem Schutz der App und des Android-Systems.

Die Erstellung des Privaten Schlüssels für das Zertifikat erfolgt separat mit einem Programm oder beim ersten Export einer App mit Eclipse. Das Zertifikat enthält die persönlichen Daten des Entwicklers und legt auch den Gültigkeitszeitraum fest. Abgespeichert wird das Ganze in einer passwortgeschützten Keystore-Datei, die bei weiteren Exporten nur noch aufgerufen werden muss.

Nach dem signierten Export kann die App weitergegeben werden oder auf einen App-Market hochgeladen werden. Dabei belaufen sich die Kosten bei GooglePlay einmalig auf 25$ (19€), während es bei alternativen Markets (z.B. AndroidPit) kostenlos ist.

5 Einschätzung

5.1 MIT AppInventor

Die Einrichtung des AppInventors erfolgt nur mit der Anleitung ohne Probleme, da die Zweiteilung des Programms in Designer und BlocksEditor zuerst verwirrt.

Die Einarbeitung in den AppInventor erfolgt dann auch ohne große Vorkenntnisse schnell und zügig. Auch die umfassende Dokumentation hilft bei vielen Problemen weiter und die ersten Ergebnisse sind schnell sichtbar.

Die Zweiteilung des Programm stört gerade dann, wenn man gleichzeitig designend und programmiert, da es bei der Synchronisierung eine Verzögerung von wenigen Sekunden gibt.

Die Bedienung ist einfach gehalten, was aber gerade beim Programmieren von größeren Apps eine Geduldsprobe ist, da jeder Code-Blog an die entsprechende Stelle geschoben werden muss und jede Eigenschaft der Komponenten manuell eingetragen werden muss.

Bei dem Designer vermisst man Einstellungen für ein ordentliches Design, wie zum Beispiel den Rand oder Abstand der Komponenten. So wurden die Buttons im Designer mit Abstand zum Rand eingefügt, während dieser Rand aber im Emulator und auf dem Handy fehlte.

Auch die Veröffentlichung über Drittsoftware fällt negativ auf.

Alles in allem ist es aber ein ausgereiftes Programm, was sehr gut für Programmieranfänger und zum „Schnuppern“ in der App-Programmierung geeignet ist.

5.2 Android Software Development Kit (SDK)

Die Einrichtung des Android SDKs ist auch ohne Anleitung gut zu bewältigen. Negativ fällt lediglich die Downloadgröße auf, die mit einigen Emulatoren schnell auf mehrere Gigabyte wächst.

Beim ersten öffnen von Eclipse werden mit Hilfe eines Assistenten die Grundeinstellungen vorgenommen. Nach dem dieser beendet ist, hat man sich mit sehr vielen Funktionsmöglichkeiten herumzuschlagen. Ohne eine entsprechende Anleitung kommt man an dieser Stelle auch mit einiger Programmiererfahrung nicht mehr weiter.

Die Dokumentation über Eclipse und die Android-Klassen ist aber sehr umfangreich. Nur die englische Fachsprache hat mir ab und zu Probleme bereitet.

Die Eingabe des Programmcodes erfolgt durch verschiedene Helfer, wie die automatische

Codehervorhebung oder die Autovervollständigung zügig. Auch wird man in Echtzeit auf Fehler oder mögliche Probleme hingewiesen.

Durch die Objektorientierung fiel es mir etwas schwer die Abarbeitung des Codes schnell zu verstehen.

Der Export für die Veröffentlichung erfolgt mit Hilfe eines Assistenten zügig und einfach.

Insgesamt macht das Android SDK einen sehr guten Eindruck, aber ohne Dokumentation brauch man gar nicht erst anfangen.

5.3 Fazit

„Der MIT AppInventor kann eine professionelle Programmierumgebung wie das Android SDK ersetzen.“ Diese These kann ich nicht befürworten, da der MIT AppInventor zu wenig Möglichkeiten bietet, um eine ordentliche App zu erstellen.

In dem MIT AppInventor findet man lediglich die Standard-Komponenten und noch wenige weiter. Es gibt keine Möglichkeit, das Standartdesign zu ändern und eine Mutilanguage-Unterstützung sucht man beim AppInventor auch vergeblich.

Die Apps vom MIT AppInventor sind auch um das fünffache größer als die Apps mit ähnlichen Codeumfang von Eclipse. Auch ist der Arbeitsspeicherverbrauch dieser AppInventor Apps höher.

Gerade die fehlende Funktion für die Veröffentlichung ist ein weiteres eindeutiges Indiz dafür, dass der MIT AppInventor für das „Schnupper“ in der App-Programmierung gedacht ist.

6 Quellen

6.1 Primärliteratur

- **Künneth, Thomas:** Android 4. Bonn 2012

6.2 Sekundärliteratur

- **Dr. Joachim Weiß:** Meyers Taschenlexikon in 10 Bänden. Band 1. Karlsruhe 1999

6.3 Internetadressen

- http://developer.android.com/sdk/installing/bundle.html, 26.01.2013
- http://www.basic4ppc.com/forum/german-tutorials/8505-b4a-bridge-eine-neue-moeglichkeiten-ihr-geraet-anzuschliessen.html#post47506, 27.01.2013
- http://de.wikipedia.org/wiki/Andy_Rubin, 13.8.2013
- http://de.wikipedia.org/wiki/Android_(Betriebssystem), 13.8.2013
- http://de.wikipedia.org/wiki/HTC_Dream, 13.8.2013
- http://de.wikipedia.org/wiki/Liste_von_Android-Versionen, 13.8.2013
- http://de.wikipedia.org/wiki/Google_Now, 13.8.2013
- http://nena.se/nenamark/view?version=2&device_id=4245, 13.8.2013
- http://www.techradar.com/news/phone-and-communications/mobile-phones/android-5-0-key-lime-pie-release-date-news-and-rumours-1091500, 13.8.2013
- http://developer.android.com/tools/publishing/app-signing.html, 02.10.2013

6.4 Abbildungsverzeichnis

7 Selbstständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und ohne fremde Hilfe verfasst und keine anderen als die im Literaturverzeichnis angegebenen Hilfsmittel verwendet habe. Insbesondere versichere ich, dass ich alle wörtlichen und sinngemäßen Übernahmen aus anderen Werken und Quellen als solche kenntlich gemacht habe.

Marienberg, den 7. Oktober 2013