universität wien

DIPLOMARBEIT

Titel der Diplomarbeit

Die Entwicklung von Mobile Apps im Informatikunterricht am Beispiel des MIT App Inventor

verfasst von

Markus Dorfstätter

angestrebter akademischer Grad

Magister der Naturwissenschaften (Mag. rer. nat.)

Wien, 2015

Studienkennzahl lt. Studienblatt:	A 190 313 884
Studienrichtung lt. Studienblatt:	Lehramt UF Geschichte, Sozialkunde & Polit. Bildg., UF Informatik und Informatikmanagement
Betreuer:	Univ.-Prof. i.R. Dr. Wilfried Grossmann

Eidesstattliche Erklärung

Hiermit versichere ich, dass ich die vorliegende Diplomarbeit ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Hilfsmittel und Quellen angefertigt, die den benutzten Quellen inhaltlich oder wörtlich entnommenen Stellen als solche kenntlich gemacht und die Arbeit bisher in keiner Form als Prüfungsarbeit vorgelegt habe.

Ich habe mich bemüht, falls erforderlich, sämtliche Bildrechte ausfindig zu machen und die Zustimmung der Inhaber zur Verwendung der Bilder in dieser Arbeit einzuholen bzw. die verwendeten Quellen entsprechend ihrer Nutzbedingungen und Lizenzen zu zitieren. Sollte dennoch eine Urheberrechtsverletzung bekannt werden, ersuche ich um Meldung bei mir.

Penk, am 30. Juni 2015

Meinen Eltern

Hannes und Rosemarie Dorfstätter

für ihre Unterstützung und ihren Rückhalt gewidmet

Inhaltsverzeichnis

Vorwort

Die vorliegende Diplomarbeit beschäftigt sich grundsätzlich mit der Bedeutung, sowie der Entwicklung von „Mobile Apps“ – Anwendungssoftware für Mobilgeräte – im Informatikunterricht der AHS-Oberstufe. Um eine angemessene Bearbeitung dieser Thematik zu gewährleisten, gliedert sich die Arbeit in drei wesentliche Bereiche:

Der erste Teil der Diplomarbeit dient als Einführung bzw. Heranführung an die verschiedenen Themenbereiche. Hier werden die wesentlichen Begriffe erläutert und der Stellenwert des mobilen Marktes, sowie des Programmierunterrichts an der österreichischen AHS angeführt. Außerdem wird der noch mehrheitlich vorhandene „traditionelle Programmierunterricht“ exemplarisch mit der Programmiersprache C++ untersucht. In diesem Teil finden sich daher folgende Forschungsfragen: Warum lernen SchülerInnen programmieren in der Schule? Warum fällt vielen SchülerInnen das Programmieren so schwer? Warum wählen viele LehrerInnen keine visuellen Programmiersprachen für den Einstieg? Warum orientiert sich der Programmierunterricht an Schulen kaum an mobilen Geräten? Wie kann ein Einstieg in die Programmierung möglichst effektiv erfolgen?

Der zweite Teil beschäftigt sich genauer mit der Entwicklung von Mobile Apps im Informatikunterricht. Dazu wird als Entwicklungsumgebung für die App-Programmierung das kostenlose Tool „MIT App Inventor“ (http://appinventor.mit.edu/explore/) herangezogen, welches in Folge der Arbeit genauer vorgestellt und untersucht wird. Welche Möglichkeiten hat der „App Inventor“ und wo befinden sich seine Grenzen? Wo liegen die wesentlichen Vor- und Nachteile dieser Entwicklungsumgebung im Programmierunterricht? Kann dieses Tool den „traditionellen Programmierunterricht“ mit einer textbasierten Sprache wie C++ ersetzen? Diese Fragen werden unter anderem mit Hilfe von Code-Gegenüberstellungen an konkreten Beispielen veranschaulicht und besser zum Ausdruck gebracht.

Abschließend entstehen im dritten und letzten Teil der Diplomarbeit, unter Berücksichtigung der Lehrpläne und Kompetenzmodelle, praktische Unterrichtsszenarien und Aufgaben zur Entwicklung von Mobile Apps für den Programmierunterricht der AHS-Oberstufe. Als Zielgruppen werden ProgrammieranfängerInnen im Basisunterricht, sowie fortgeschrittene EntwicklerInnen im Schwerpunktfach Informatik Bedeutung finden.

Teil I.
Einleitung und Grundlegendes zum Programmierunterricht

Gründe für den Programmierunterricht an der Schule

Bei einer pragmatischen Vorgehensweise findet man die unterschiedlichen Gründe für einen schulischen Programmierunterricht in den verschiedenen Lehrplänen, Kompetenzmodellen und Bildungsaufträgen der Allgemeinbildenden höheren Schulen (AHS) in Österreich. Diese Richtlinien und Vorgaben sind vom Lehrpersonal weitgehend zu befolgen.

Dabei fällt zunächst auf, dass das Unterrichtsfach Informatik in der AHS-Unterstufe kein Teil der Pflichtgegenstände ist, es aber durchaus Gymnasien gibt, welche dieses Fach in Form von Freigegenständen oder unverbindlichen Übungen bereits in der Unterstufe anbieten. Die SchülerInnen sollen dabei in der Lage sein, „grundlegende Kompetenzen im Umgang mit neuen Technologien“ zu erwerben und „interessensorientierte Arbeiten mit neuen Technologien“ durchzuführen. (vgl. BMBF, Lehrplan AHS-Unterstufe, S. 3)

In der 5. Klasse der AHS-Oberstufe lässt sich Informatik auf der Stundentafel der Pflichtgegenstände finden. Dieser Gegenstand hat dabei die Lehr- und Bildungsaufgabe, den „Schülerinnen und Schülern informatische und informationstechnische Grundkenntnisse zu vermitteln, um sie zu befähigen, diese zur Lösung einer Problemstellung sicher und kritisch einzusetzen.“ Des Weiteren sollen die SchülerInnen auch „Grundprinzipien von Automaten, Algorithmen und Programmen kennenlernen.“ (vgl. BMBF, Lehrplan AHS-Oberstufe, Pflichtgegenstand, S. 1f)

In der 6., 7. und 8. Klasse der Gymnasien wird das Unterrichtsfach im Regelfall zu einem Wahlpflichtgegenstand. Diesen wählen die SchülerInnen bei Interesse freiwillig, um in diesem Bereich ihr Wissen zu vertiefen. Hier finden sich im vorgegebenen Lehrstoff Bereiche wie „Konzepte von Programmiersprachen“ oder „grundlegende Algorithmen und Datenstrukturen“. (vgl. BMBF, Lehrplan AHS-Oberstufe, Wahlpflichtgegenstand, S. 1)

Abgesehen von den Informatik-Lehrplänen finden sich im Bildungsbereich „Natur und Technik“ im allgemein Teil des AHS-Lehrplans Ziele, wie zum Beispiel das Verständnis für Problemstellungen aus dem Bereich Technik. Die Lehrpersonen sind außerdem angehalten, den SchülerInnen „Formalisierung, Modellbildung, Abstraktions- und Raumvorstellungsvermögen“ zu vermitteln. (vgl. BMBF, Lehrplan AHS-Oberstufe, Allgemeiner Teil, S. 4)

Aktuellere und genauere Informationen über die Kompetenzen, welche im Informatikunterricht zu fördern sind, sind auf der Webseite www.digikomp.at, der „Education Group“, welche mit dem Bundesministerium für Bildung und Frauen zusammenarbeitet, zu finden. Dabei sind für den Basisunterricht Informatik in der 5. Klasse der AHS unter anderem folgende Kompetenzen aufgelistet, welche die SchülerInnen innerhalb des Schuljahres erwerben sollten (übernommen von Andraschko, Kompetenzmodell Informatik 5. Klasse):

- „Ich kann Konzepte der Informatik bei der Lösung konkreter Aufgaben anwenden“
- „Ich kann den Algorithmusbegriff erklären“
- „Ich kann einfache Algorithmen nachvollziehen und erklären“
- „Ich kann die Umsetzung von Algorithmen mit einem Computer erklären“
- „Ich kann einfache Algorithmen entwerfen, diese formal darstellen, implementieren und testen“
- „Ich kann an Hand von einfachen Beispielen die Korrektheit von Programmen bewerten“

Auf derselben Webseite finden sich für den vertiefenden Informatikunterricht, in Form eines Wahlpflichtgegenstands in der Oberstufe, Informationen zur Kompetenzförderung der SchülerInnen (übernommen von Andraschko, Kompetenzmodell Wahlpflichtfach Informatik):

- „Ich kann bei der Lösung konkreter Aufgaben Heuristiken, Grundprinzipien und Konzepte der Informatik anwenden und informatische Modelle gestalten.“

- „Ich kann unterschiedliche Lösungsansätze in Bezug auf zugrunde liegende Konzepte reflektieren und in konkreten Handlungssituationen bewerten."
- „Ich kann Algorithmen entwerfen, diese formal darstellen, implementieren und testen."
- „Ich kann ein Softwareprojekt planen und durchführen."
- „Ich kann die Schritte der Softwareentwicklung reflektieren."
- „Ich kann die Angemessenheit der Entwicklungswerkzeuge grob einschätzen."
- „Ich kann die Effizienz von Algorithmen bewerten."
- „Ich kann gezielt nach Programmfehlern suchen und diese korrigieren."

Meiner Meinung nach stellen die beiden angeführten Aufzählungen der übernommenen Kompetenzen von www.digikomp.at die Lernziele der SchülerInnen sehr gut dar. In den Lehrplänen selbst werden nur Überbegriffe wie zum Beispiel „Konzepte von Programmiersprachen" (vgl. BMBF, Lehrplan AHS-Oberstufe, Wahlpflichtgegenstand, S. 1) angeführt und lassen im Unterricht daher sehr viel Freiraum: Wie lange soll programmiert werden? Wie soll programmiert werden? Was sollen die SchülerInnen nach dem Unterricht beherrschen? etc. Im Kompetenzmodell der „Education Group" hingegen sieht man direkt die einzelnen Bereiche und kann diese als Teilziele laufend ergänzend zum Lehrstoff im Lehrplan betrachten, um einen möglichst guten Programmierunterricht in der AHS-Oberstufe zu erreichen.

Abseits der rechtlichen Grundlagen finden sich weitere Gründe für den Programmierunterricht: Programmieren kann dazu verwendet werden, um Probleme zu lösen und kann daher die Problemlösungsfähigkeit der SchülerInnen, sowie das logische Denkvermögen fördern. (vgl. Bruderer, 2009, S. 2) Die Vorgehensweise beim Lösen eines Programmierproblems wird dabei durch drei grobe Schritte charakterisiert: Zunächst muss ein Problem vorliegen, für welches eine Lösung gesucht wird. Danach wird durch Problemanalyse und Abstraktion ein Algorithmus zur Problemlösung entworfen.

Dieser Algorithmus wird nachfolgend in ein Programm transferiert, welches auf dem Rechner ausgeführt wird und die Lösung des Problems darstellt. (vgl. Müller & Weichert, 2005, S. 15f) Dabei zeichnet sich eine gute Programmiersprache vor allem dadurch aus, dass sie dem menschlichen Denkprozess durch „geeignete Sprachelemente entgegen kommt, sodass der menschliche Programmierer bei der Umsetzung der Lösungsidee in die Programmiersprache weitgehend entlastet wird." (vgl. Balzert, 2005, S. 76) Bei diesem schrittweisen Konzept ist vor allem auffällig, dass der Algorithmusentwurf bzw. das algorithmische Denken vor dem Schreiben eines Programms geschieht. Die Wahl der Sprache oder der Entwicklungsumgebung ist somit sekundär und das algorithmische Denken der SchülerInnen der erste und wichtigste Schritt zur Problemlösung. Dieser Aspekt muss von der Lehrperson berücksichtigt werden und sollte dementsprechende Beachtung finden.

Es ist außerdem möglich, dass der Programmierunterricht einen fächerübergreifenden Beitrag leistet und z.B. die SchülerInnen in den Hauptfächern Englisch, Deutsch oder Mathematik fördert. Bei vielen Programmiersprachen sind die Anweisungen und Kontrollstrukturen in englischer Sprache und festigen daher die Fremdsprachenkenntnisse der SchülerInnen. Außerdem ist es notwendig, die Argumente und Anweisungen in einer präzisen, klaren Sprache zu formulieren (vgl. Bruderer, 2009, S. 2), was beispielsweise bei diversen Aufsätzen oder Referaten in den Sprachfächern der Fall ist. Obwohl zum Lernen einer Programmiersprache kein überdurchschnittlich guter Mathematiker erforderlich ist, können die Programmierkenntnisse auch die Mathematikkenntnisse verbessern, da – wie bereits angesprochen – das Programmieren die allgemeine logische Denkfähigkeit, welche auch im Mathematikunterricht gebraucht wird, verbessern kann. (vgl. Perry, 2002, S. 26)

Ein weiterer Vorteil des Programmierunterrichts ist die individuelle Förderung der SchülerInnen durch modular aufgebaute Programme und ergänzende Zusatzaufgaben.

Die SchülerInnen programmieren beispielsweise ein Spiel, welches sie optional durch Elemente wie Musik, Grafiken, Highscore-Tabellen oder weitere Ideen ergänzen. Damit werden leistungsschwächere SchülerInnen nicht über- und leistungsstärkere SchülerInnen im Unterricht nicht unterfordert. Die Orientierung am Individuum und dessen Fähigkeiten ist im Rahmen des „guten Unterrichts“ von zentraler Bedeutung und muss ebenfalls von der Lehrperson berücksichtigt werden. (vgl. Meyer, 2011, S. 18)

Zuletzt werden auch die beruflichen Chancen für die SchülerInnen genannt. Wir leben in einem Informationszeitalter, welches von digitalen Medien und zunehmender Technik geprägt ist. Es ist Aufgabe des Informatikunterrichts, einen Beitrag dazu zu leisten, dass Interesse in diesen Bereichen geweckt wird und dass den SchülerInnen mögliche Berufswege und Forschungsrichtungen aufgezeigt werden. Ferner kann der Informatik- und Programmierunterricht auch helfen, hinter die Fassaden der elektronischen Geräte und Abläufe im Alltag zu blicken, damit diese und die damit verbundenen Berufe kein Mysterium bleiben. (vgl. Geisler, 2010, S. 198)

Schwierigkeiten der SchülerInnen beim Programmieren

Vorab muss erwähnt werden, dass es keine eindeutige Antwort auf die Frage, warum vielen SchülerInnen das Programmieren schwer fällt, gibt und lediglich Gründe gesucht und aufgezählt werden können, warum manche SchülerInnen mehr Probleme und Schwierigkeiten im Programmierunterricht haben als andere. In diesem Zusammenhang erschien bei der „International Conference on Engineering Education“ in Coimbra (Portugal) ein interessanter Artikel, in welchem viele Schwierigkeiten von studierenden ProgrammieranfängerInnen gelistet wurden. (vgl. hierzu Gomes & Mendes, 2007, S. 1ff) Da sich sehr viele der genannten Gründe im schulischen Alltag wiederspiegeln, wurden diese und einige Erläuterungen aus dem Paper übernommen und durch weitere Literatur und eigene Erfahrungen ergänzt. Dabei ist zu beachten, dass sich viele Schwierigkeiten bzw. Probleme nicht genau abgrenzen lassen und ineinander greifen.

Die Unterrichtsmethoden

Trotz der Option der individuellen Förderung erhalten viele SchülerInnen zu wenig persönliches Feedback zu ihren Leistungen. Die SchülerInnen erkennen und verstehen manche Programmierfehler teilweise nicht oder haben Probleme, die Anforderungen bzw. die Aufgabenstellungen vollständig zu erfassen. Es besteht außerdem auch die Möglichkeit, dass die Lehrmethoden der Lehrperson für bestimmte Kapitel schlecht gewählt wurden und zu wenige SchülerInnen ansprechen. Um den verschiedenen Aufgaben gerecht zu werden, ist es erforderlich, dass die Lehrerin oder der Lehrer die unterschiedlichen Interessen der SchülerInnen beachtet und auf ein Repertoire an Unterrichtsmethoden zurückgreifen kann. (vgl. Meyer, 2011, S. 74) Auch ist der Fokus vieler LehrerInnen falsch gesetzt, da sie versuchen eine Programmiersprache und die damit verbundene Syntax zu lehren, anstatt die Problemlösungsfähigkeit und das algorithmische Denken der SchülerInnen mittels einer Programmiersprache als ergänzendes Medium zu trainieren. Gerade am Anfang kann eine höhere Programmiersprache wie C++ für die SchülerInnen überfordernd wirken und viele Fragen aufwerfen. Das Abschreiben fertiger Algorithmen wird auch noch heute von vielen Lehrpersonen praktiziert und hat kaum einen Lerneffekt, da viele SchülerInnen diese auswendig lernen und nicht hinter die Fassade blicken.

Dieser Umstand wird am Beispiel eines klassischen Anfängerprogramms deutlich – einem „Hello-World-Programm“ in der Programmiersprache C++:

```
#include <stdio.h>

int main (int argc, const char* argv[])
{
    printf("\nHello World!\n");
}
```

Abbildung 1: Hello-World in C++

Betrachtet man die einzelnen Elemente dieser wenigen Code-Zeilen genauer, wird einer erfahrenen Entwicklerin bzw. einem erfahrenen Entwickler relativ schnell klar: Im Regelfall wissen ProgrammieranfängerInnen in der ersten Einheit kaum etwas über Datentypen, Funktionen, Parameter, Arrays, Zeiger, Bibliotheken oder Escape-Sequenzen. Sie verstehen die Logik und die Bedeutung dieser Elemente nicht auf Anhieb und haben bei so einem Beispiel aufgrund der Menge an unbekannten Inhalten überhaupt keine Möglichkeit, diesen Code-Auszug zur Gänze zu verstehen.

Hinzu kommt, dass die SchülerInnen je nach Entwicklungsumgebung die Programmteile auch kompilieren müssen, um überhaupt greifbare Ergebnisse zu sehen.

Die Lernmethoden

Viele SchülerInnen haben eigene „Lernmethoden“ entwickelt, mit denen sie den Schulalltag bewältigen, ohne spezifische Inhalte zu verstehen oder Fähigkeiten bzw. Kompetenzen in manchen Bereichen zu erwerben. Oftmals reicht es aus, über ein Stoffgebiet nur oberflächlich Bescheid zu wissen und viele Inhalte auswendig zu lernen. In Unterrichtsfächern wie Mathematik oder Informatik ist das allerdings nur selten der Fall, da das Verstehen der Inhalte eine vergleichsweise hohe Priorität hat.

Man kann nicht ausschließlich durch das Lesen eines Buches lernen, wie man Auto fährt, Basketball oder Klavier spielt – man muss es üben. Genauso wenig kann man programmieren lernen, ohne nicht selbst Code zu lesen und zu schreiben. (vgl. Stroustrup, 2010, S. 29)

Die Fähigkeiten der SchülerInnen

Eine weitere Schwierigkeit im Programmierunterricht ist die unzureichende Problemlösefähigkeit der SchülerInnen bzw. die nicht trainierte Herangehensweise an Problemstellungen. Letzteres möglicherweise als Folge dessen, dass die SchülerInnen nicht sinnerfassend lesen bzw. diesen Aspekt zu wenig geübt haben. Vielen SchülerInnen gelingt es außerdem nicht, einen Zusammenhang zwischen bereits bearbeiteten Problemen herzustellen. Ihnen fehlt die Relation zwischen verschiedenen Problemstellungen und sie merken nicht, dass sie vielleicht ein ähnliches Problem schon gelöst haben. Auch programmieren sie oftmals nur eine gültige Lösung für einen konkreten Anwendungsfall, ohne dabei spezielle Grenzfälle zu berücksichtigen oder den Versuch für eine allgemein gültige Lösung zu wagen. (vgl. Lobnig, 2008, S. 4) Bei einigen Problemstellungen ist das fehlende mathematische oder logische Verständnis ein Hindernis bei der Lösung. Leider ist auch der fehlende Ehrgeiz vieler SchülerInnen zu nennen: Sie geben zu schnell auf, wenn sie ein Problem nicht direkt lösen können.

Das Wesen der Programmierung

Programmieren lernen ist keine leichte Disziplin und verlangt den SchülerInnen einige Fähigkeiten ab. Vorstellungs- und Abstraktionsvermögen sind essentiell zum Lösen vieler Probleme und gerade für ProgrammieranfängerInnen oft schwierig, wenn es um das Verständnis von Objektorientierung, Kontrollstrukturen oder Konzepten, wie z.B. das Zeigerkonzept in C++ geht. Dabei ist auch die Syntax von C++ für viele AnfängerInnen sehr hinderlich, da sie den eigentlichen Denkprozess und die Problemlösungsfähigkeit nicht unterstützt. Dieser Umstand zeigt sich an diversen repräsentativen Umfragen, bei welchen studierende ProgrammieranfängerInnen angegeben haben, im klassischen Unterricht von C++ vor allem mit Rekursionen, Zeigern und dem Klassenkonzept (z.B. abstrakten Datentypen, Konstruktoren, Überladen, etc.) Schwierigkeiten zu haben. (vgl. hierzu Lahtinen, 2005 oder Milne & Rowe, 2002)

Ich bin der Auffassung, dass solche Umfragen an Schulen zu ähnlichen Ergebnissen geführt hätten. Der Programmierunterricht an einer AHS geht zwar nicht in die gleiche Tiefe wie ein Informatikstudium, dennoch werden SchülerInnen teilweise durchaus mit komplexeren Konzepten, wie Zeigern oder Objektorientierung während ihrer Schulzeit konfrontiert.

Psychologische Effekte

SchülerInnen haben oftmals nicht die notwendige Motivation im Programmierunterricht, da er aufgrund seiner Komplexität und teilweisen unzureichenden Vermittlung abschreckend wirken kann. Guter Informatikunterricht geht von der Interessenslage der SchülerInnen aus, da Neugier einer der größten Motivationsfaktoren ist. (vgl. Mittermeir, 2003, S. 13)

Michael Kölling (University of Kent) behauptet sogar, dass das Erzeugen von Motivation das Geheimnis guten Programmierunterrichts ist. Demnach haben alle Menschen das Verlangen, kreativ zu sein und Dinge zu schaffen – sei es beim Spielen mit Legosteinen, beim Kuchen backen oder beim Haus bauen. Im Informatikunterricht lässt sich diese Kreativität am besten mit dem Programmieren von Anwendungen mit einfachen Entwicklungsumgebungen, wie z.B. Greenfoot oder Scratch fördern. Erst wenn die SchülerInnen gefesselt und motiviert sind, haben sie ein Verlangen nach mehr Informationen und mehr Wissen, wodurch komplexere Programmierkonzepte Einzug in das Klassenzimmer finden können. (vgl. Kölling, 2010, S. 33f)

Viele SchülerInnen werden außerdem durch das häufig verbundene Image von motivierten, ehrgeizigen SchülerInnen als „Streber“ oder guten Informatikern als „Nerds“ konfrontiert und nehmen sich daher im Unterricht zurück. Auch Probleme im sozialen Umfeld abseits der Schule wie z.B. der Familie, beeinflussen das Verhalten der SchülerInnen im Unterricht und können Auswirkungen auf ihre Motivation und ihre Leistungen haben.

Visuelle Programmiersprachen – eine Alternative?

Das Unterrichtsfach Informatik wurde als verbindliche Übung in der AHS seit dem Schuljahr 1985/86 vorgeschrieben und drei Jahre später als Pflichtfach im Lehrplan verankert. Ab dem Schuljahr 1989/90 wurde es den AHS-SchülerInnen ermöglicht, Informatik schließlich auch als Wahlpflichtgegenstand von der 6. bis zu der 8. Klasse zu belegen. (vgl. Reiter, 2005, S. 6 & S. 12) Die Erscheinung der Programmiersprache C++ im Jahr 1985 ging mit den österreichischen Reformen zum Informatikunterricht einher und gab daraufhin die Möglichkeit eines an einer neuen und modernen Programmiersprache orientierten Programmierunterrichts mit einer höheren, hybriden Sprache.

Rüdeger Baumann kommt zum Entschluss, dass das Beherrschen einer einzigen Programmiersprache nicht ausreicht, um die Konzepte und Methoden der Informatik zu verstehen und stellt daher folgende Thesen auf (übernommen von Baumann, 2000, S. 28f):

- „Die Schüler sollen zwei höhere Programmiersprachen kennen lernen“
- „Eine dieser Sprache sollte imperativisch und objektorientiert sein“
- „Die andere Sprache sollte prädikativ (logikorientiert sein)“

Ich kann diesen Thesen teilweise zustimmen, wobei ich zwischen einführendem und fortgeschrittenem Programmierunterricht differenzieren möchte. Für SchülerInnen mit etwas Programmiererfahrung halte ich es durchaus vorstellbar und je nach Schulschwerpunkt auch sinnvoll, eine objektorientiere Sprache (z.B. C++) und eine logikorientierte Sprache (z.B. Prolog) zu lernen. Betrachtet man aber die möglichen Programmierschwierigkeiten bei SchülerInnen, so frage ich mich, warum LehrerInnen nicht versuchen, diese Anfängerprobleme mit komplexen Kontrollstrukturen oder der Syntax zu vermeiden und erst programmiererfahrenere SchülerInnen in höheren Sprachen unterrichten? Die Intention bei der Wahl einer Programmiersprache ist eben von Lehrperson zu Lehrperson bzw. von Schule zu Schule verschieden.

Es gibt zwar viele Gründe, welche für C++ als Programmiersprache in der Schule sprechen, wie exemplarisch angeführt:

- effizient und schnell im Vergleich mit vielen anderen Sprachen
- weit verbreitete und anerkannte Programmiersprache – womöglich wichtig für Studium und zukünftiges Berufsleben der SchülerInnen
- C++ ist eine hybride Sprache, wodurch objektorientiert oder rein prozedural programmiert werden kann (vgl. Balzert, 2005, S. 830) – dadurch kann man die Sprache im Unterricht flexibel einsetzen
- viele Programmierkonzepte sind direkt auf andere Sprachen z.B. C#, Java oder C übertragbar (vgl. Stroustrup, 2010, S. 28)
- es gibt einen Compiler für praktisch jedes Betriebssystem, welches an Schulen und anderen Bildungseinrichtungen eingesetzt wird
- kann in der Schule kostenlos verwendet werden und braucht kaum Zusatzsoftware
- und vieles mehr…

Dennoch findet man auch sehr viele Punkte, welche dagegen sprechen, wie:

- komplexe und umständlichere Syntax im Vergleich zu einigen anderen Sprachen, daher sehr fehleranfällig für EinsteigerInnen
- trotz verbreiteter Verwendung kein innovativer Ansatz im Programmierunterricht (C++ erschien vor 40 Jahren)
- wird in vielen Schulen nur textbasiert in Konsolen entwickelt, wodurch die Sprache für SchülerInnen nicht wirklich motivierend und greifbar ist
- undefiniertes Verhalten kann zu Abstürzen oder Fehlern führen, welche die SchülerInnen und LehrerInnen nicht immer nachvollziehen können
- etc.

Visuelle Programmiersprachen können dabei als Möglichkeit gesehen werden, die Einstiegsprobleme im „traditionellen Programmierunterricht“ zu minimieren und den SchülerInnen verständlichere und spannendere Informatikstunden zu ermöglichen.

Dabei erleichtern grafische Notationen die Programmierung bzw. erhöhen die Verständlichkeit von Programmen bei der Programmerstellung. Dadurch können auch AnfängerInnen komplexere Applikationen erstellen, für welche teilweise professionelle SoftwareentwicklerInnen benötigt werden würden. (vgl. Schiffer, 1999, S. 619)

Historisch gesehen hat die graphisch-orientierte Programmiersprache „Logo“ mit ihrer eingebauten Turtle-Grafik sicherlich teilweise Bedeutung im Informatikunterricht gefunden. Dabei steuern die ProgrammierInnen in einem Spielfeld Schildkröten, welche bunte Linien hinter sich herziehen, wodurch verschiedene Muster und Zeichnungen dargestellt werden können. Viele dieser Muster, wie z.B. die Kochsche Schneeflocke, haben wiederkehrende Merkmale, damit die SchülerInnen Iterationen und Rekursionen mittels grafischer Ergebnisse üben können. Seit der Jahrtausendwende sind einige neue, mächtigere „visuelle Programmiersprachen“ auf den Markt gekommen, welche ebenfalls Jugendliche und Kinder als Zielgruppe haben und sich daher perfekt für den Schuleinsatz eignen. Einen hohen Bekanntheitsgrad haben Scratch und dessen Erweiterung namens „Snap! (BYOB)“ – letztere wird mehrmals in der Diplomarbeit für Vergleiche herangezogen. Diesen grafischen Umsetzungen gelingt es, die Syntax, also die korrekte Art und Weise sprachliche Elemente zusammenzufügen und daraus gültige Sätze zu ordnen (vgl. Küchlin, 2005, S. 120), dahin gehend zu vereinfachen, dass die Programmteile mittels Puzzle-Konzept aneinander gesteckt werden. Dabei werden die verschiedenen Anweisungen mit Hilfe von Drag & Drop aneinander gereiht, wodurch Syntaxfehler weitgehend vermieden werden und die SchülerInnen intuitiv richtige Puzzle-Elemente aneinander stecken. (vgl. Modrow, 2013, 5f) Einige vordefinierte Funktionen helfen den SchülerInnen bei der Lösung ihrer Aufgaben und vereinfachen die Programme:

Abbildung 2: Hello-World in Snap!

Gerald Futschek (Technische Universität Wien, Institute of Software Technology & Interactive Systems) befasst sich seit Jahren mit dem Bildungssektor innerhalb der Informatik und hält zu dieser Thematik diverse Vorträge und Lehrveranstaltungen. Auch er ist der Auffassung, dass programmieranfangende SchülerInnen vor allem mit der Syntax der Sprachen, sowie den Fehlermeldungen zu kämpfen haben. Er hat daher einige Vor- und Nachteile von der graphisch-orientierten Sprache BYOB am Informatiktag 2012 zusammengefasst (vgl. Futschek, 2012):

Vorteile:

- die ersten Programmschritte können leicht erlernt werden
- rasche Fortschritte beim Erlernen von Programmierkonzepten
- für die Benutzung der Entwicklungsumgebung ist kaum ein Lehraufwand erforderlich, d.h. man hat mehr Zeit, sich auf das Erlernen der Programmierkonzepte zu konzentrieren
- die Programme werden oftmals strukturierter

Nachteile:

- Programmierfehler werden teilweise nicht erkannt (es gibt keine Fehlermeldungen)
- mühsames Fehlerfinden
- der Umstieg auf Programmiersprachen wie C++ ist eher schwierig
- die Entwicklungsumgebung und die Ausführung der Programme ist eher langsam

Er empfiehlt den Einsatz von BYOB im Unterricht, wenn Programmierkonzepte relativ zügig behandelt werden, aber nicht, „wenn es auf die Vermittlung professioneller SW-Entwicklungswerkzeuge ankommt“ (vgl. Futschek, 2012). Da letzteres offensichtlich nicht zu den direkten Aufgaben eines einführenden Programmierunterrichts an der österreichischen AHS gehört, kann die Vermittlung einer visuellen Programmiersprache wie BYOB als Chance gesehen werden, den SchülerInnen algorithmisches Denken näher zu bringen bzw. deren Problemlösungsfähigkeit zu stärken.

Zum Stellenwert mobiler Geräte und Anwendungen

In den letzten Jahrzehnten wurde die Entwicklung von mobilen Endgeräten stets vorangetrieben, sodass wir heute auf eine Vielzahl verschiedener „Mobile Devices“ zurückblicken können. Der Kommunikationsaspekt steht bei jedem dieser Geräte als Grundfunktion im Mittelpunkt, allerdings lassen sich diese neuen technischen Erzeugnisse nach den Eigenschaften Lokalisierbarkeit, Ortsunabhängigkeit und Erreichbarkeit subjektiv klassifizieren (vgl. Tschersich, 2010):

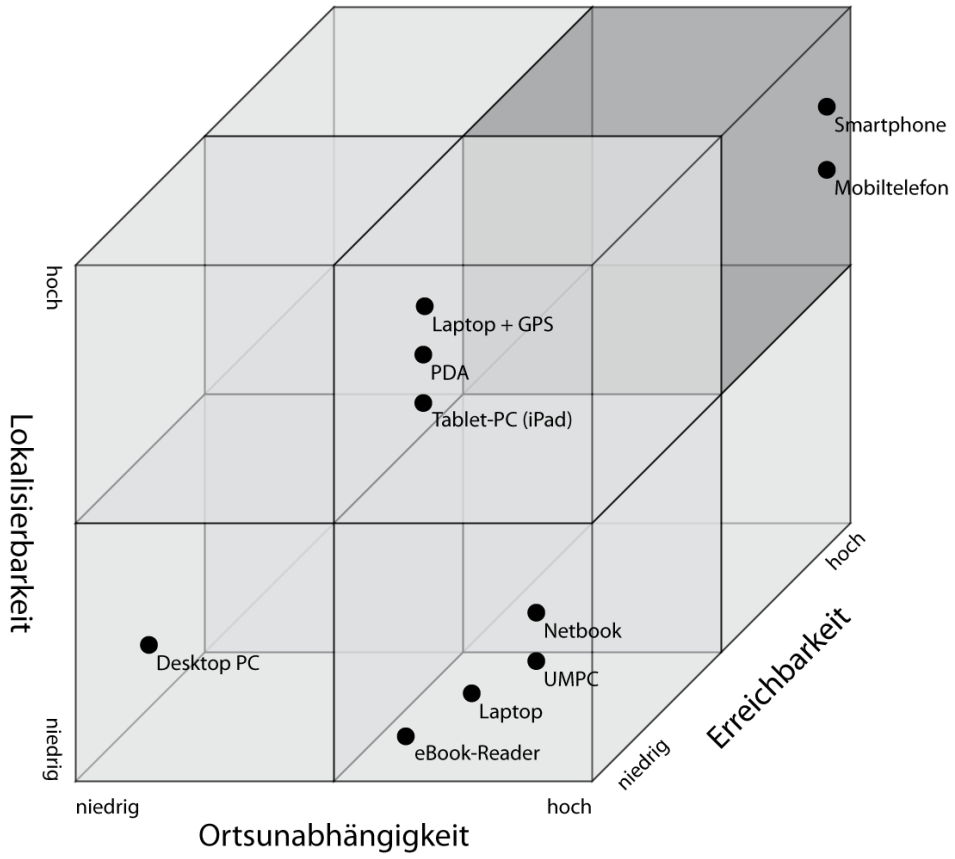


Abbildung 3: Klassifizierung nach Tschersich (2010)

Betrachtet man diese Klassifikation unter Berücksichtigung der historischen Entwicklung, so lässt sich ein klarer Trend zu ortsunabhängigen, flexiblen Geräten erkennen.

Viele SchülerInnen sind daher mittlerweile im Besitz von unterschiedlichen Smartphones, wobei diese Geräte keine eindeutige Beschreibung bzw. Definition besitzen. Smartphones haben die grundlegenden Funktionen von Mobiltelefonen wie z.B. telefonieren oder SMS senden, verfügen allerdings oftmals über größere Bildschirme, verwenden geläufige Betriebssysteme und sind in der Lage, eine kabellose Internetverbindung aufzubauen. (vgl. Fling, 2009, S. 8) Gängige Geräte sind heute außerdem mit einem Touch-Screen ausgestattet, mit welchem der Benutzer interagiert und verzichten auf eine eigene Tastatur direkt am Gerät. (vgl. Fling, 2009, S. 10) Deutlich wird die alltägliche Bedeutung von mobilen Geräten vor allem bei der Betrachtung diverser Nutzerstatistiken: Demnach nutzten im Jahre 2014 bereits 82% der ÖsterreicherInnen ein Smartphone (vgl. Mobile Marketing Association, 2014, S. 24) und beachtliche 93,3% der 16- bis 24-Jährigen nutzten ihre tragbaren Geräte für den mobilen Internetzugang. (vgl. Statistik Austria, 2014) Über 90% der österreichischen Smartphone-NutzerInnen haben bereits Apps aus dem Store heruntergeladen, wobei junge Menschen mehr downloaden als ältere Menschen. Fast 40 Prozent haben außerdem mehr als zwanzig Apps auf ihrem Handy installiert. (vgl. Mobile Marketing Association, 2014) Hinter der Kurzform „App“ versteckt sich das englische Wort „Application“ – auf Deutsch also „Anwendung“. Dabei ist zu beachten, dass eine „App“ keine vom System benötigten Eigenschaften erfüllt und eher Dienste für die individuellen Bedürfnisse der NutzerInnen bereitstellt. (vgl. Achten & Pohlmann, 2012, S. 161 zitiert nach Vogel, 2013, S. 12) Der Designer und Entwickler Josh Lehman charakterisiert sie außerdem als möglichst benutzerfreundlich, indem sie leicht verständlich und ohne großartiges Lesen des Benutzerhandbuches und langes Einarbeiten genutzt werden können. (vgl. Lehman, 2012) Zweifelsohne stellt die rasche technische Entwicklung eine besondere Art der Herausforderung im Informatikunterricht dar, da die LehrerInnen und SchülerInnen immer wieder mit neuen Geräten konfrontiert werden. Dennoch muss man dieser Entwicklung Beachtung schenken, indem man sie im Unterricht einbezieht und die SchülerInnen befähigt, zwischen langlebigen Informatikprinzipen und kurzlebigen Soft- bzw. Hardwareprodukten zu unterscheiden. (vgl. Andraschko, Kompetenzmodell Wahlpflichtfach Informatik)

Effektiver Einstieg für ProgrammieranfängerInnen

Betrachtet man nun die Erkenntnisse aus den vorhergegangenen Kapiteln, so lassen sich verschiedene Thesen für einen „optimalen“ Einstieg in die Programmierung finden:

Zum einen sind die SchülerInnen laut den Lehrplänen und Kompetenzmodellen vorrangig mit dem grundlegenden Algorithmenkonzept vertraut zu machen und haben diverse Algorithmen selbstständig zu entwerfen, implementieren und zu testen. (vgl. z.B. Andraschko, Kompetenzmodell Wahlpflichtfach Informatik) Es wurde allerdings bereits darauf hingewiesen, dass die Wahl der Programmiersprache bzw. die Wahl der Entwicklungsumgebung sekundär ist und das algorithmische Denken und die Problemlösungsfähigkeit der SchülerInnen mehr Priorität hat. Dabei kann eine verständliche, einfache Programmiersprache bzw. Entwicklungsumgebung einen Beitrag dazu leisten, dass die SchülerInnen einen Teil der anfänglichen Schwierigkeiten wie z.B. das Erlernen der Syntax einer Programmiersprache oder das Einarbeiten in eine komplexe Entwicklungsumgebung weitgehend vermieden werden. Gerade das „Auswendiglernen“ der Syntax und der Sprachelemente gehört mit modernen, komplexen IDEs (=Integrierte Entwicklungsumgebungen) eher der Unterrichtsvergangenheit an, da die Bedienung und das Verständnis dieser immer mehr in den Vordergrund gerückt ist. Visuelle Programmiersprachen unterstützen diesen Trend und stellen das logische Denkvermögen der SchülerInnen, sowie die Interface-Bedienung in den Fokus. Auch wenn diese Sprachen im späteren Wirtschafts- bzw. Berufsleben der SchülerInnen keinen oder kaum Stellenwert haben werden, eignen sie sich für den anfänglichen Programmierunterricht und können später, je nach Schul- und Interessenschwerpunkt, durch eine „höhere Programmiersprache“ ersetzt werden. Entscheidet man sich im Informatikunterricht für eine visuelle Programmiersprache, so fällt die Wahl mitunter nicht immer leicht. Jede Sprache hat Vor- und Nachteile, welche man gegeneinander aufwiegen kann, wobei man sich möglichst auch am Interessensgebiet der SchülerInnen orientieren sollte.

Doch welche technischen Geräte gewinnen zunehmend an Bedeutung und beschäftigen die heutigen Jugendlichen, indem sie es schaffen, diese über Stunden hinweg zu begeistern und zu fesseln? Richtig! Smartphones. Warum dann nicht bei dieser technischen Entwicklung ansetzen und als Lehrperson den SchülerInnen vermitteln, wie sie selbstständig Anwendungen für ihre Smartphones erschaffen können? Ein vergleichsweise einfaches Tool, mit dessen Hilfe im Programmierunterricht „Mobile Apps“ entwickelt werden können, ist der MIT App Inventor, welcher im nächsten Teil der Diplomarbeit genauer analysiert wird.

Teil II.
Der MIT App Inventor
(in der Version 2, Beta)

Über das Tool [1]

Der MIT App Inventor ist eine block-basierte, graphische Entwicklungsumgebung, welche für ProgrammieranfängerInnen als auch für fortgeschrittene EntwicklerInnen geeignet ist. Ursprünglich von Google entwickelt, zeichnet sich der MIT App Inventor dadurch aus, dass mit Hilfe dieses Open-Source-Tools relativ rasch Smartphone-Anwendungen für das Betriebssystem Android entwickelt und zum Laufen gebracht werden können. Mittlerweile werden durch die weltweite Community von etwa drei Millionen Benutzern rund 195 Staaten repräsentiert, wobei das Tool mit über sieben Millionen programmierten Apps mehr als 100.000 aktive wöchentliche User verzeichnet. Die Arbeit des App Inventor Teams ist primär geleitet von folgenden fünf Aspekten: (1) Vorantreiben bzw. Weiterentwicklung des Tools (2) Zusammenarbeit mit Smartphone-Herstellern wie z.B. Motorola (3) Entwicklung von Unterrichtsmaterial und Handbüchern (4) Förderung des Informatikunterrichts (5) Verwaltung und Unterstützung der gemeinschaftlichen Forschung an der Entwicklungsumgebung.

Bei den Anwendern fokussiert sich das Team auf folgende Zielgruppen:

- PädagogInnen – Zum einen um den SchülerInnen das Programmieren beizubringen, zum anderen aber auch für die eigene Entwicklung von Unterrichtswerkzeug bzw. Unterrichtsmaterial
- Privatpersonen und Interessierte – Personen, welche den MIT App Inventor nutzen, um sich fortzubilden oder um hobbymäßig ihren Interessen nachzugehen
- UnternehmerInnen – Kreative Köpfe, welche mit innovativen App-Ideen versuchen, Fuß zu fassen, ohne dabei einen langwierigen Lernprozess zu benötigen
- ForscherInnen in unterschiedlichen Bereichen - Zur Entwicklung eigener Anwendungen, um Daten zu verwalten oder diese zu analysieren

[1] entnommen von der MIT App Inventor Webseite mit eigenen Ergänzungen, online unter http://appinventor.mit.edu/explore/about-us.html, letzter Aufruf: 10.05.2015

Voraussetzungen [2]

Betriebssystem

- Windows: Windows XP, Windows Vista, Windows 7
- GNU/Linux: Debian 5 oder höher, Ubuntu 8 oder höher
- Macintosh mit Intel Prozessor: Mac OS X 10.5 oder höher

Browser

- Mozilla Firefox 3.6 oder höher
- Google Chrome 4.0 oder höher
- Apple Safari 5.0 oder höher
- Microsoft Internet Explorer wird nicht unterstützt

Mobiles Gerät (Smartphone / Tablet)

- Android Betriebssystem 2.3 („Gingerbread“) oder höher

Sonstiges

- Der User benötigt ein Google Konto
- Kostenlose Zusatzsoftware für Geräteverbindung & Emulator erforderlich

Aufbau der Entwicklungsumgebung

Im Wesentlichen besteht der MIT App Inventor aus zwei verschiedenen Editoren, mit denen die Benutzerin bzw. der Benutzer interagiert und zwischen denen umgeschaltet werden kann. Beide Editoren werden benötigt, um eine funktionierende Anwendung für ein Smartphone oder Tablet erstellen zu können. Eine Besonderheit und Stärke des App Inventors ist die Verbindung mit dem Gerät oder dem zur Verfügung gestellten Emulator direkt während der Entwicklungsphase.

[2] entnommen von der MIT App Inventor Webseite mit eigenen Ergänzungen, online unter http://appinventor.mit.edu/explore/ai2/setup.html, letzter Aufruf: 11.05.2015

Damit können die Ergebnisse der Programmierung in Echtzeit betrachtet werden und SchülerInnen ohne entsprechende Geräte können ebenfalls am Unterricht teilnehmen, indem sie ihre Programme mittels Emulator zum Laufen bringen. Ein mögliches Unterrichtsszenario im Informatikunterricht wäre daher, dass alle SchülerInnen die Anwendungen am PC programmieren und diese laufend mittels Emulator überprüfen. Nach der Fertigstellung der Programmierung können jene SchülerInnen mit Android-Betriebssystem die Anwendungen auf ihre Smartphones laden.

Design Editor

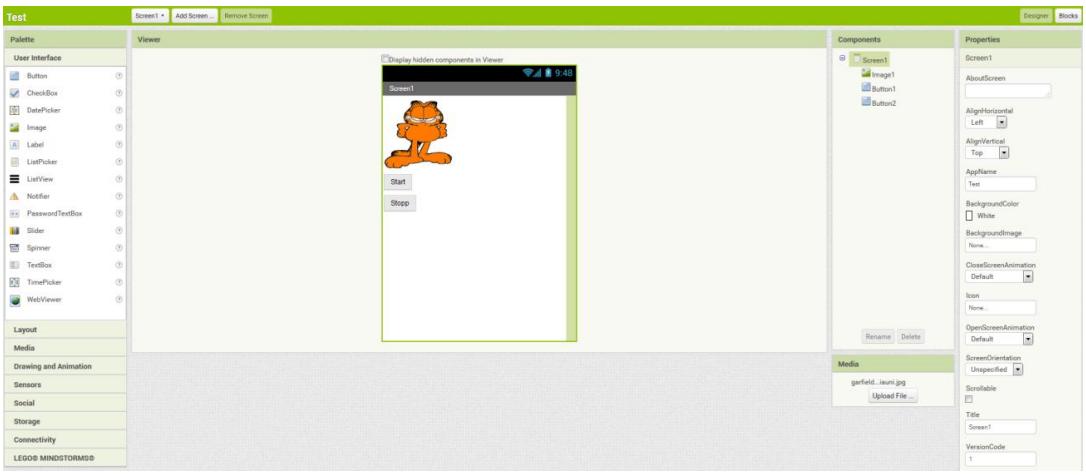


Abbildung 4: Design Editor

Der Design Editor ist notwendig, damit die Anwendung ein bestimmtes Aussehen bekommt bzw. dafür, dass die einzelne Elemente in der Darstellung bearbeitet werden können. Die *Palette* beinhaltet die unterschiedlichen Komponenten und Bauelemente für die Anwendung. Diese enthält neben den „Standardelementen“ wie Buttons, Textfeldern oder Labels, auch die gerätespezifischen Elemente wie Audioausgabe, Barcode Scanner oder die Möglichkeit, das Gerät zu lokalisieren (GPS). Mittels Drag & Drop lassen sich die gewünschten Elemente einfach in den *Viewer*, welcher aus beliebig vielen *Screens* besteht, ziehen. Ein *Screen* ist dabei immer eine Ansammlung diverser Elemente und deren Darstellung auf dem Gerät.

Das mobile Gerät kann danach aufgrund verschiedener Ereignisse wie z.B. Wischen oder Drücken den aktuellen Screen wechseln und dadurch die Darstellung am Bildschirm ändern. Unter *Components* befindet sich eine praktische Übersicht aller Komponenten auf dem aktuellen Screen. Aus dieser Liste kann man direkt die gewünschten Elemente anklicken und bearbeiten, wodurch einem die Suche der Controls innerhalb des *Viewers* erspart bleibt. Jedes Bauelement wird dabei durch seine *Properties* beschrieben. Die verschiedenen Eigenschaften des Elements gehen je nach Typ von Größe, Farbe, Name über Bilderdateien oder Positionierung bis hin zu Startwerten, Zeitintervallen, Telefonnummern und vielem mehr. Unter *Media* befindet sich schließlich eine Übersicht aller verwendeten externen Dateien wie z.B. Bilder, Videos oder Musik.

Blocks Editor

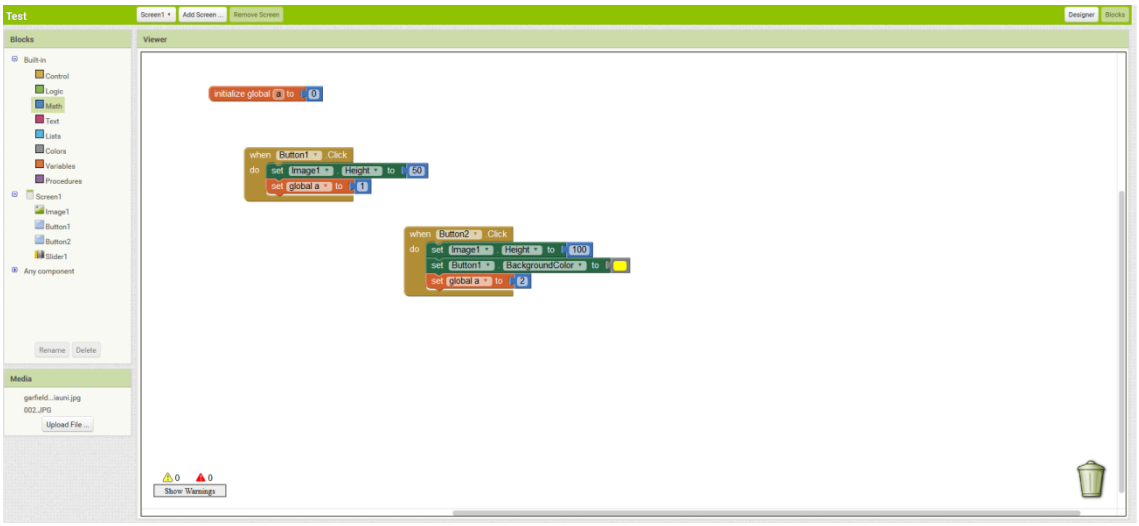


Abbildung 5: Blocks Editor

Der Blocks Editor beinhaltet den eigentlichen Quellcode der Anwendung in Form von bunten Puzzleteilen / Blöcken. Hier können unabhängig vom Aussehen der einzelnen Elemente Entscheidungen, Anweisungen oder bestimmte Verhaltensmuster getroffen werden.

Unter *Blocks* befinden sich die unterschiedlichen Kontrollstrukturen und Anweisungen, welche auch aus anderen Programmiersprachen bekannt sind. Hier können z.B. Schleifen, Verzweigungen, Listen, Variablen, Berechnungen, usw. bequem mittels Drag & Drop in den *Viewer* gezogen werden. Der *Viewer* besteht aus den *Screens* des Design Editors. Jeder *Screen* wird im Design Editor im Aussehen und im Blocks Editor in der Logik bearbeitet. Auch *Media* als Übersicht aller verwendeten externen Dateien ist erneut vorhanden. Die einzelnen Blöcke im *Viewer* können verschoben und bearbeitet werden. Dabei ist zu beachten, dass fast nur syntaktisch-korrekter Quellcode entstehen kann, da die verschiedenen Elemente als Puzzleteile fungieren und nur jene Elemente zusammenpassen, welche auch zusammengehören können.

Zur Gestaltung der Anwendungen

Den EntwicklerInnen stehen im Design Editor eine Vielzahl an Möglichkeiten zur Verfügung, um ihre Anwendungen zu gestalten. Einige wichtige und häufig verwendete Controls werden in diesem Abschnitt der Diplomarbeit kurz vorgestellt. Jedes dieser Elemente verfügt über spezifische, individuelle Eigenschaften, welche vor und/oder während der Laufzeit verändert werden und daher unterschiedliche Werte annehmen können. Dabei ist zu beachten, dass sich die Controls verschachtelt in Containern befinden können, welche die Ausrichtung der Elemente am Bildschirm festlegen:

 HorizontalArrangement	Die Elemente in diesem Container werden waagrecht bzw. nebeneinander dargestellt
 TableArrangement	Die Elemente werden in Tabellenform angeordnet, wobei Spalten- und Zeilenanzahl konfigurierbar ist
 VerticalArrangement	Die Elemente in diesem Container werden senkrecht bzw. untereinander dargestellt

Tabelle 1: Layout im MIT App Inventor

In der Palette unter dem Abschnitt *User Interface* finden sich wichtige Controls, welche teilweise auch aus anderen Entwicklungsumgebungen bekannt sind:

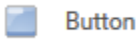
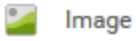
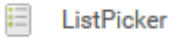
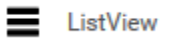
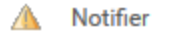
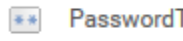
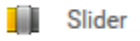
	Klickbare Schaltfläche, welche verschiedene Klickmethoden auf das Element unterscheidet und entsprechende Ereignisse auslösen kann z.B. einfaches Klicken & Loslassen oder gedrückt halten
	Klickbares Element, welches ausgewählt oder nicht ausgewählt werden kann und daher zu diversen Überprüfungen eingesetzt wird
	Öffnet beim Klick einen Dialog zur Auswahl eines Datums (Tag, Monat & Jahr) und kann das Ergebnis zurückliefern
	Dient zur Anzeige hochgeladener Bilder und kann das dargestellte Bild auch während der Laufzeit ändern
	Wird für eine einfache Textanzeige verwendet, beispielweise zum Gestalten von Überschriften oder Beschreibungen
	Klickbare Schaltfläche, welche beim Klick eine Liste anzeigt von welcher Elemente ausgewählt werden können
	Direkte Anzeige einer Liste mit auswählbaren Elementen
	Wird für die Anzeige unterschiedlicher Meldungen benötigt z.B. Fehlermeldungen, Nachrichten, Hinweise, etc.
	Dient zur Texteingabe über die Gerätetastatur und versteckt die Eingabe hinter Symbolen
	Schieberegler, welcher während der Laufzeit manuell verschoben werden kann und das aktuelle Zahlenergebnis innerhalb eines festgelegten Intervalls zurückliefert
	Beim Klick kann der User eines von mehreren vordefinierten Elementen auswählen. Zusätzlich wird das Ergebnis dieser Auswahl direkt im Element angezeigt

Tabelle 2: User Interface Elemente 1/2 im MIT App Inventor

TextBox	Dient zur Texteingabe über die Gerätetastatur
 TimePicker	Öffnet beim Klick einen Dialog zur Auswahl einer Uhrzeit (Stunde, Minute, AM/PM) und kann das Ergebnis zurückliefern
 WebViewer	In diesem Element kann eine Internetseite dargestellt werden, deren URL vor oder während der Laufzeit einstellbar ist

Tabelle 3: User Interface Elemente 2/2 im MIT App Inventor

Für den Unterricht besonders spannend scheinen einige Controls aus der Rubrik *Media*, durch welche das multimediale Angebot des mobilen Geräts für die weitere Verarbeitung zugänglich gemacht wird. Dabei ist zu beachten, dass einige dieser Controls nicht durch den Emulator unterstützt werden und sich daher vor allem für den Einsatz direkt am Gerät eignen.

 Camcorder	Öffnet die Videofunktion des mobilen Geräts und ermöglicht die weitere Verwendung des aufgenommenen Videos
 Camera	Öffnet die Kamerafunktion des mobilen Geräts und ermöglicht die weitere Verwendung des aufgenommen Bildes
 ImagePicker	Öffnet die Bildergalerie des mobilen Geräts und ermöglicht die Auswahl mehrere Bilder aus dieser
 Player	Ermöglicht das Abspielen von Musik oder der Vibrationsfunktion des mobilen Geräts – wird für längere Musikstücke empfohlen z.B. Lieder
 Sound	Ermöglicht das Abspielen von Musik oder der Vibrationsfunktion des mobilen Geräts – wird für kürzere Musikstücke empfohlen z.B. Klingeltöne

Tabelle 4: Media Elemente 1/2 im MIT App Inventor

SoundRecorder	Dient zur Sprachaufzeichnung und der Speicherung der aufgenommenen Geräusche als Soundfiles
SpeechRecognizer	Dient zur Sprachaufzeichnung und der Speicherung der aufgenommenen Geräusche als Text / in Textform
TextToSpeech	Zur Audiowiedergabe von einem gegebenen Text – dabei können einige Sprachen und Länder ausgewählt werden, um bei der Aussprache unterschiedliche Akzente zu berücksichtigen
VideoPlayer	Ermöglicht die Wiedergabe von Videos mit einem Standardinterface (Play, Pause, Nächstes, usw.)
YandexTranslate	Ermöglicht die Übersetzung eines Textes in eine gewünschte Zielsprache

Tabelle 5: Media Elemente 2/2 im MIT App Inventor

Ebenfalls auf das mobile Gerät sind die Elemente aus der Kategorie *Social* zugeschnitten, welche Controls für die zwischenmenschliche Kommunikation bereitstellen:

ContactPicker	Zeigt die Kontaktliste und liefert spezifische Eigenschaften der Kontakte z.B. Name, E-Mail
EmailPicker	Eine Textbox, welche bei der Eingabe einer E-Mail-Adresse mögliche Vorschläge zur Vervollständigung von gespeicherten Adressen (in der Kontaktliste) liefert
PhoneCall	Kann bei Zuweisung einer gültigen Telefonnummer, diese mittels Methodenaufruf anrufen
PhoneNumberPicker	Zeigt die Kontaktliste und liefert spezifische Eigenschaften der Kontakte – liefert allerdings weniger Informationen als der ContactPicker

Tabelle 6: Social Elemente 1/2 im MIT App Inventor

 Sharing	Kann ein File aus der entwickelten App auf eine andere App übertragen. Dazu wird als Transferziel eine Liste der kompatiblen Apps vorgeschlagen.
 Texting	Kann bei Zuweisung einer gültigen Telefonnummer, diese per einstellbarer Textnachricht (SMS) kontaktieren
 Twitter	Stellt Dienste zur Interaktion mit dem Kurznachrichtendienst „Twitter“ zur Verfügung

Tabelle 7: Social Elemente 2/2 im MIT App Inventor

Auch unter *Sensors* befinden sich viele interessante Steuerungselemente, deren Anwendung sich eher direkt am mobilen Gerät und nicht auf die Simulation im Emulator fokussiert. Einige dieser Elemente werden in der darunterliegenden Tabelle vorgestellt:

 Clock	Zeitobjekt, durch welches Zeitberechnungen durchgeführt oder bestimmte Events nach Ablauf einer Zeit ausgeführt werden können. Ebenfalls interessant für spielerische Elemente, da man z.B. Objekte nach Ablauf einer gewissen Zeit automatisch bewegen kann (vgl. Bhagi, 2012, S. 7)
 BarcodeScanner	Zum Einscannen von Strichcodes und QR-Codes. Es besteht die Möglichkeit, den internen Scanner vom App Inventor zu verwenden oder eine andere App als Scanner anzugeben
 LocationSensor	Bietet eine Vielzahl von Lokalisierungsoptionen wie z.B. GPS-Koordinaten von einem bestimmten Ort oder dem aktuellen Standort des mobilen Geräts
 AccelerometerSensor	Erkennt Erschütterungen am mobilen Gerät wie z.B. Wackeln in unterschiedlichen Stärken und kann dadurch verschiedene Ereignisse auslösen

Tabelle 8: Sensors Elemente im MIT App Inventor

Unter *Drawing and Animation* befinden sich Controls, welche vor allem für Anwendungen mit spielerischem Charakter geeignet sind. Es ist kein Geheimnis, dass gerade spielerische Inhalte im Informatikunterricht positive Effekte auf den Lernerfolg der SchülerInnen haben können (vgl. Braun, 2013, S. 94), weshalb die folgenden Elemente durchaus Gegenstand im AHS-Programmierunterricht sein können:

 Canvas	Dieser Container nimmt die Interaktion des Users am Bildschirm intensiver wahr und wird benötigt, um darin Animationen zu erzeugen. Die unteren zwei Elemente in dieser Tabelle „Ball“ und „ImageSprite“ können nur in diesem Container erzeugt und gesteuert werden. Zusätzlich kann in diesem Panel auch gezeichnet werden, wobei die Farbe, Strichstärke, usw. konfigurierbar sind
 Ball	Erzeugt einen Ball auf dem Spielfeld, welcher mit dem User interagiert und z.B. beim Ziehen oder Drücken auf dem Bildschirm bewegt werden kann
 ImageSprite	Äquivalent zum oberen Element „Ball“, allerdings kann hier das zu bewegende Element (= Bild) selbst gewählt werden und sich während der Laufzeit auch ändern

Tabelle 9: Drawing and Animation Elemente im MIT App Inventor

Abgesehen von den vorgestellten Controls bietet der MIT App Inventor noch eine Vielzahl an weiteren Steuerungselementen. So ist es z.B. auch möglich, Verbindungen über Bluetooth aufzubauen oder Dateiverarbeitungen und kleinere Datenbanken für eine Anwendung zu entwickeln, um Inhalte wie Spielstände bzw. Highscore-Tabellen zu speichern und zu verarbeiten. Es ist sogar möglich, eigene Apps zur Steuerung von *LEGO Mindstorms* im MIT App Inventor zu entwickeln. Damit gelingt eine Verbindung zwischen den mobilen Geräten der SchülerInnen und den programmierbaren Legosteinen, deren Anwendung im Informatikunterricht zwar nicht kostengünstig, aber ebenfalls ertragreich scheint.

Kontrollstrukturen und andere Konzepte

Der MIT App Inventor kann dazu verwendet werden, um die grundlegenden Programmierkonzepte und Kontrollstrukturen zu lernen und diese zu verinnerlichen bzw. zu üben. Im weiteren Verlauf der Diplomarbeit werden diese vorgestellt und die Entwicklung dieser mittels App Inventor mit einer möglichen textbasierten Umsetzung in der Programmiersprache C++ verglichen. Doris Mayr vergleicht in ihrer Diplomarbeit zwei Programmiersprachen miteinander, indem sie diese zeilenweise in Tabellen gegenüberstellt. (vgl. hierzu Mayr, 2014) Aufgrund der Übersichtlichkeit wurde ebenfalls dieses Konzept zur Code-Gegenüberstellung verwendet. Dabei ist zu beachten, dass die Kontrollstrukturen und Konzepte nicht nach Schwierigkeit oder Vorwissen der SchülerInnen geordnet sind und daher nicht zwingenderweise in dieser Reihenfolge im Programmierunterricht behandelt werden müssen. Wie bereits angesprochen, erfolgt durch das Ziehen der gewünschten Elemente mittels Drag & Drop und durch diverse Verschachtelungen, Zuweisungen, Aufrufe, etc. dieser, die eigentliche Programmierung der Anwendungen im Block Editor. Viele der Blöcke benötigen einen dynamischen Charakter, welcher mittels *Mutatoren* (Zahnradsymbol) erreicht werden kann, wodurch mittels Drag & Drop die gewünschten Blöcke erzeugt werden können:

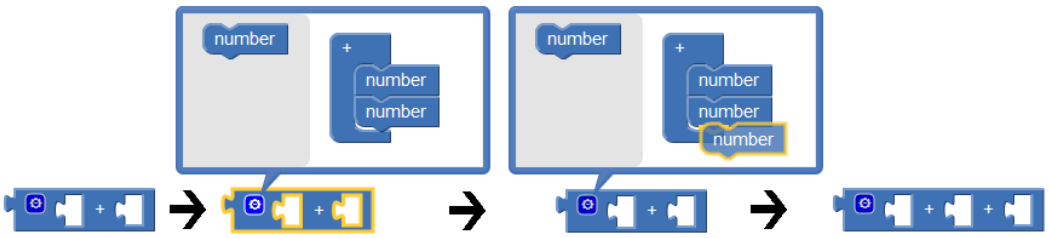


Abbildung 6: Mutator im MIT App Inventor

Variablen und Datentypen

Variablen sind ein essentieller Bestandteil einer Programmiersprache und werden benötigt, um die unterschiedlichen Daten in den Speicher abzulegen.

Sie besitzen in der Regel eine Bezeichnung und einen Datentyp, welcher den Typ der zu speichernden Inhalte einer spezifischen Variable festlegt – sie sind also benannte „Orte“, in denen Daten gespeichert werden. (vgl. Stroustrup, 2010, S. 96) Variablen können auch manipuliert werden, indem diese durch Berechnungen und Zuweisungen andere Werte als den ursprünglich, initialisierten Wert annehmen. Im App Inventor stehen folgende Blöcke zur Behandlung von Variablen zur Verfügung:

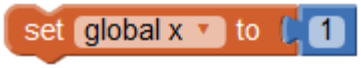
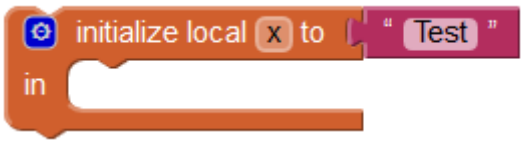
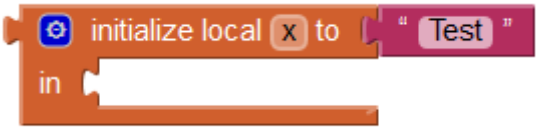
	Deklaration der globalen Variable x (ist dadurch im ganzen Screen verfügbar) und Zuweisung des Wertes 0
	Rückgabe des Wertes der globalen Variable x
	Zuweisung des Wertes 1 an die globale Variable x
	Deklaration der lokalen Variable x und Zuweisung des Wertes „Test“. Als lokale Variable ist x nur in den darunterliegenden Blöcken, welche aus Anweisungen bestehen, enthalten
	Deklaration der lokalen Variable x und Zuweisung des Wertes „Test“. Als lokale Variable ist x nur in dem darunterliegenden Blöcken enthalten. Im Unterschied zum oberen Block wird hier allerdings ein Wert zurückgegeben (siehe die Unterschiede in der Puzzleform)

Tabelle 10: Variablen im MIT App Inventor

Eine Besonderheit im Bezug auf das Variablenkonzept des App Inventors stellt sicherlich die Tatsache dar, dass die Variablen auf keinen Datentyp beschränkt werden. Es ist dadurch möglich, dass eine Variable mit einer Zahl initialisiert wird und im nächsten Schritt einen Text zugewiesen bekommt. Daher spielen im Programmierunterricht mit dem MIT App Inventor die Datentypen eine eher untergeordnete Rolle, wodurch sich die Lehre dieses Konzepts wahrscheinlich mit einer „traditionellen“, textbasierten Programmiersprache wie C++ vereinfachen lässt. Die Datentypen sind im App Inventor lediglich bei der Zuweisung durch verschiedene Blöcke in unterschiedlichen Farben unterscheidbar. Blaue Blöcke repräsentieren Zahlen und mathematische Operationen, wobei zwischen ganzen Zahlen und Gleitkommazahlen kein Unterschied besteht. Grüne Blöcke repräsentieren die Operationen der booleschen Algebra und liefern die Ausdrücke „wahr“ oder „falsch“ als Rückgabe. Rosa Blöcke wiederum stellen die unterschiedliche Operationen für Strings (= Zeichenketten) dar, wobei es im Gegensatz zu C++ keine Längenbeschränkung gibt und man ohne Umwege direkt mit längeren Zeichenketten arbeiten kann.

	`int a = 12;`

	`double b = 151.2;`
	oder
	`float b = 151.2;`

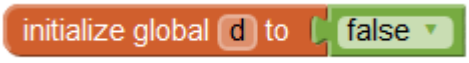
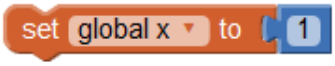
	`bool c = true;`
	`bool d = false;`
	`char e[] = "Beispieltext";`
	`char f = 'a';`

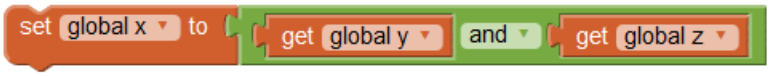
Tabelle 11: Datentypen Vergleich App Inventor / C++

Wie bereits angesprochen, lassen sich Variablen während der Laufzeit bearbeiten und können dadurch unterschiedliche Werte annehmen, z.B.

	`x = 1;`

	`x = x + 1;`
	oder
	`x += 1;`
	oder
	`x++;`

	`x = sqrt(4);`
	allerdings muss hier vorher eine Bibliothek eingebunden werden:
	`#include <math.h>`

	`x = !y;`
	`x = y && z;`

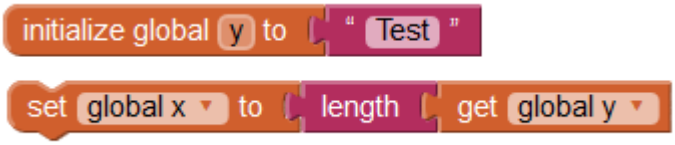
	`char y[] = "Test";`
	`for (;y[x]!='\0';x++);`
	oder
	`string y ("Test");`
	`x = y.size();`
	aber Datentyp „string“ nur mit einer Bibliothek:
	`#include <string>`

Tabelle 12: Beispielhafte Zuweisungen App Inventor / C++

Betrachtet man die Beispiele in der vorherigen Tabelle, so sind einige Dinge anzumerken. Vorab muss allerdings betont werden, dass alle Beispiele nur exemplarisch gewählt wurden, um Unterschiede zwischen den Sprachen zu zeigen und nicht immer den einzigen Lösungsweg für das Beispiel zeigen müssen. Auffällig ist vor allem, dass eine Anweisung beim MIT App Inventor in der Regel selbst erklärend ist, da sie über eine textuelle Beschreibung direkt im Blockelement verfügt. Möchte man beispielweise einer Variable namens *var* einen Wert zuweisen, so lautet der Block *„set var to …“* – im Gegensatz zur Zuweisung mit dem Gleichheitszeichen in C++, welches womöglich von ProgrammieranfängerInnen fehlinterpretiert werden könnte. Da sehr viele Funktionen bereits direkt im App Inventor verfügbar sind, erspart man sich auch das Einbinden von externen Libraries und das Wissen darüber, welche Funktion in welcher Library überhaupt enthalten ist. Dieser Umstand wird vor allem bei der Arbeit mit Strings deutlich, da man ohne externe Libraries diesen Datentyp bei der Arbeit mit C++ über ein eigenes Char-Array simuliert und dann auch auf Dinge wie das Nullzeichen in Form einer Escape-Sequenz als Markierung für das Ende der Zeichenkette achten muss oder bei Vergleichen und Bearbeitungen von Zeichenketten meistens mit Zeigern arbeiten muss.

Bedingungen und Verzweigungen

Oftmals ist es notwendig, dass in bestimmten Programmstellen Entscheidungen über den weiteren Verlauf getroffen werden müssen. Dabei kommen bedingte Anweisungen ins Spiel, welche es ermöglichen, dass Anweisungen nur dann ausgeführt werden, wenn eine bestimmte Bedingung erfüllt wird. Optional lassen sich alternative Anweisungen beim Nichterfüllen der Bedingung festlegen oder man kann mehrere Bedingungen und Entscheidungen angeben. Ob bestimmte Anweisungen innerhalb der Verzweigungen ausgeführt werden, hängt ausschließlich vom booleschen Rückgabewert – „wahr“ oder „falsch“ - der Bedingungen ab. Folgendes Beispiel soll die Anwendung von Bedingungen und Verzweigungen demonstrieren:

Falls die Variable *a* einen kleineren Wert als die Variable *b* gespeichert hat, wird der Wert der Variable *c* auf den von *b* gesetzt. Andernfalls wird der Wert der Variable *c* auf den von *a* gesetzt. Somit wird in der Variable *c* immer der größte Wert von den anderen beiden Variablen gespeichert. Im zweiten Beispiel erfolgt eine Erweiterung um eine Verzweigung, dass bei gleichem Wert von *a* und *b*, die Variable *c* auf den Wert 1 gesetzt wird.

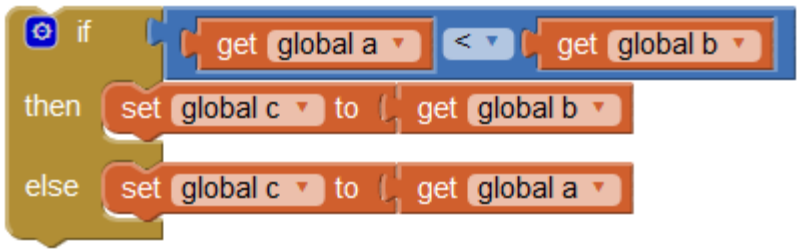
<table>
<tr>
<td> 
</td>
<td><pre>
if (a < b)
{
    c = b;
}
else
{
    c = a;
}
</pre></td>
</tr>
<tr>
<td> 
</td>
<td><pre>
if (a < b)
{
    c = b;
}
else if (a == b)
{
    c = 1;
}
else
{
    c = a;
}
</pre></td>
</tr>
</table>

Tabelle 13: Bedingungen Vergleich App Inventor / C++

Im zweiten Beispiel können die zuvor erwähnten Schwierigkeiten in der unterschiedlichen Bedeutung der Gleichheitszeichen in der Programmiersprache C++ auftreten. Je nach Anzahl ist die Bedeutung entweder eine Wertzuweisung oder ein Vergleich der beiden Werte. Diese Schwierigkeit kann beim App Inventor nicht entstehen, da das Gleichheitszeichen ausschließlich als Wertevergleich eingesetzt wird. Die Vergleichsoperatoren im MIT App Inventor verfügen über die gewohnte Trennung in den Farben blau, grün und rosa für die jeweiligen Datentypen.

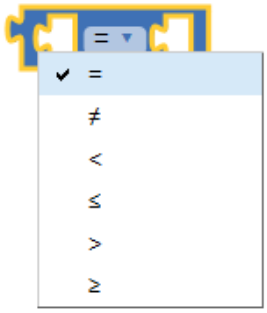
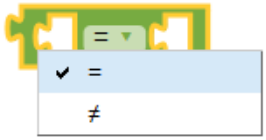
	Mathematische Vergleichsoperatoren
	Boolesche Vergleichsoperatoren
	Textbasierte Vergleichsoperatoren

Tabelle 14: Vergleichsoperatoren im MIT App Inventor

Dabei ist anzumerken, dass die Vergleichsoperatoren nicht in ihrem definierten Gültigkeitsbereich – ihrer „Farbe“ - eingeschränkt sind. Der liberale Umgang mit Datentypen kann erneut die Fehlerquellen von ProgrammieranfängerInnen vermindern, sodass folgende Vergleiche, unabhängig vom Datentyp, das gleiche Resultat (*false*) liefern:

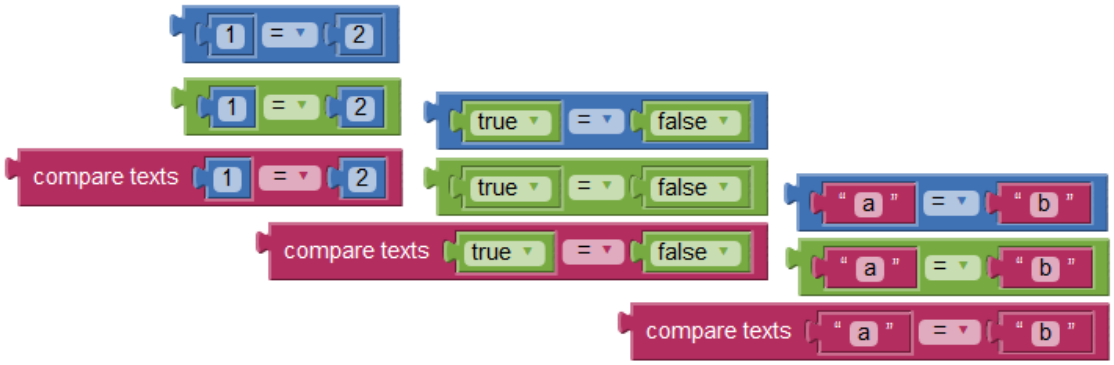
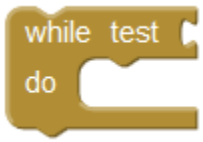


Abbildung 7: Vergleichsmöglichkeiten

Schleifen und Wiederholungen

Diese Kontrollstrukturen werden benötigt, damit man unter einer bestimmten Bedingung gewünschte Anweisungen mehrmals ausführen kann. Dabei gibt es verschiedene Arten von Schleifen und die Schleifen können auch untereinander verschachtelt werden, wodurch ein zusätzlicher Komplexitätsfaktor entsteht, welcher im regulären Programmierunterricht bei einigen SchülerInnen zu Schwierigkeiten führen kann. Im folgenden Beispiel wird mittels kopfgesteuerter Schleife die Summe der natürlichen Zahlen vom Wert der Variable *a* bis zum Wert der Variable *c* durchgeführt und in der Variable *b* gespeichert.

	Führt die Anweisungen im „do-Bereich“ so lange aus, solang die Bedingungen im „test-Bereich“ erfüllt sind. Dabei ist zu beachten, dass die Bedingungen vor einem Durchgang geprüft werden und es daher sein kann, dass die Anweisungen innerhalb der Schleife nicht ausgeführt werden

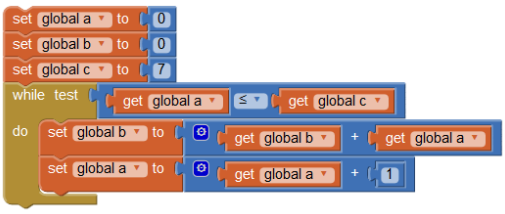
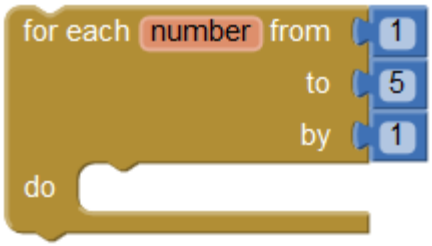
	`a = 0;`
	`b = 0;`
	`c = 7;`
	`while (a <= c)`
	`{`
	`  b = b + a;`
	`  a++;`
	`}`

Tabelle 15: Kopfgesteuerte Schleife Vergleich App Inventor / C++

In diesem Beispiel ist wieder die textuelle Beschreibung der Schleife für AnfängerInnen ganz hilfreich. Beim MIT App Inventor wird die Bedingung zuvor „getestet“, während auf dieses Schlüsselwort in C++ verzichtet wurde.

Das nächste Beispiel zeigt eine Zählerschleife, welche alle ungeraden Zahlen von 1 bis 10 aufsummiert und in der Variable *a* speichert. Die Variable *i* fungiert dabei als Zähler, wird mit dem Startwert 1 initialisiert und erhöht sich in Zweier-Schritten, bis sie maximal den Wert 10 erreicht.

	Deklariert eine lokale Variable innerhalb der Schleife, welche standardmäßig mit dem Wert 1 initialisiert wird und bei jedem Durchgang um 1 erhöht wird, bis sie den Wert 5 erreicht, wodurch die Schleife beendet wird. Für die Zählvariable wird standardmäßig der Name „number“ vergeben

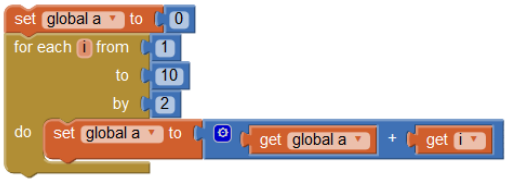
	`a = 0;`
	`for (int i = 1; i <= 10; i = i + 2)`
	`{`
	`  a = a + i;`
	`}`

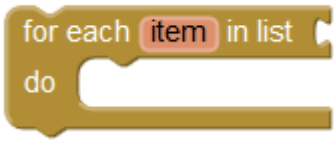
Tabelle 16: Zählerschleife Vergleich App Inventor / C++

Praktischerweise werden auch in dieser Schleife verschiedene Schlüsselwörter als Hilfestellung verwendet. Bei der Zählerschleife in C++ hingegen muss man die Reihenfolge der Anweisungen in den runden Klammern und die unterschiedliche Bedeutung dieser kennen. So enthält z.B. der erste Teil einmalige Initialisierungen vor dem Schleifendurchgang, der zweite Teil eine oder mehrere Bedingungen, welche vor einem Durchgang erfüllt sein müssen und der dritte Teil eine oder mehrere Anweisungen, welche am Ende der Schleife durchgeführt werden. Es können aber auch alle drei Teile leer bleiben, wodurch folgende Schleife eine syntaktisch-korrekte Endlosschleife in C++ wäre:

```
for (;;) a++;
```

Abbildung 8: Endlosschleife C++

Dass dieser Umstand für ProgrammieranfängerInnen nicht immer logisch und verständlich erscheint, ist eher weniger überraschend. Der App Inventor umgeht diese Probleme durch seine Schlüsselwörter, denn die SchülerInnen erkennen direkt, an welcher Stelle sie die Werte einfügen müssen, um das gewünschte Verhalten zu erzielen. Abgesehen davon werden kaum Syntaxfehler zugelassen: Die Strichpunkte als Markierung des Anweisungsendes sind nie notwendig und grüne (= boolesche) Blöcke bzw. rosa (= textbasierte) Blöcke passen nicht in den Schleifenkopf, wodurch gewährleistet wird, dass dort nur Zahlen vorkommen dürfen. Als letzte Schleife steht den EntwicklerInnen eine „For-Each-Schleife“ zur Verfügung, welche sämtliche Elemente eines bestimmten Containers durchläuft. Im nächsten Beispiel werden alle Elemente einer Liste (dieses Konzept wird im nächsten Teil der Diplomarbeit erläutert) aufsummiert und das Ergebnis in der Variable *a* gespeichert.

	Deklariert eine lokale Variable „item“ innerhalb der Schleife, welche für jeden Durchgang den Wert eines Listenelements annimmt

	`int liste[] = {1,10,7};`
	
	`a = 0;`
	`for (int item: liste)`
	`{`
	`  a = a + item;`
	`}`

Tabelle 17: For-Each-Schleife Vergleich App Inventor / C++

Abgesehen von den zusätzlichen Schlüsselwörtern im App Inventor sind beide Konzepte sehr ähnlich. Ein Vorteil des App Inventors ist allerdings erneut der relative freie Umgang mit Datentypen, da diese weder bei der Initialisierung der Liste, noch im Schleifenkopf angeben werden müssen.

Arrays und Listen

Grundsätzlich besteht ein Unterschied zwischen Listen und Arrays, vor allem im Bezug auf Speicherplatzreservierung und dynamischen Verhalten während der Laufzeit. Der MIT App Inventor kennt allerdings nur das Listenkonzept und erlaubt damit die Speicherung mehrerer Werte unterschiedlicher Datentypen unter einem benannten Objekt. Das Listenkonzept in C++ ist allerdings in der Umsetzung weitaus komplizierter, benötigt viel Programmiererfahrung und wird nur in seltenen Fällen Gegenstand des Programmierunterrichts an einer AHS ohne besonderen Programmierschwerpunkt sein. Statische Arrays in C++ hingegen können und sollten durchaus Gegenstand des Unterrichts sein, da sie eben auch zu den grundlegenden Konzepten der Programmierung zählen. Daher werden im folgenden Teil das Listenkonzept des MIT App Inventors mit dem Arraykonzept von C++ verglichen, um mehr Aussagekraft über die Unterschiede zu erhalten. Diese Vergleichsform wird auch von Mayr gewählt, da in der einen Programmiersprache Arrays und in der anderen Programmiersprache Listen das Basiskonzept „für die Zusammenfassung mehrerer Variablen darstellen“. (vgl. Mayr, 2014, S. 46) Für Listen gibt es MIT App Inventor Blöcke in einer eigenen Farbe (hellblau) und viele vordefinierte Funktionen, welche standardmäßig in C++ implementiert werden müssen. Die Speicherung von Elementen mit unterschiedlichen Datentypen ist im MIT App Inventor möglich und je nach Anwendungsfall auch sinnvoll:

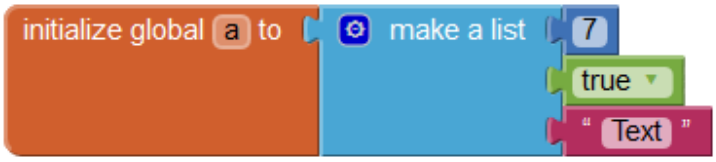
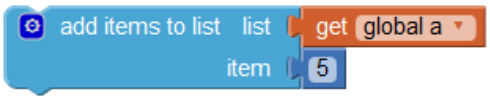
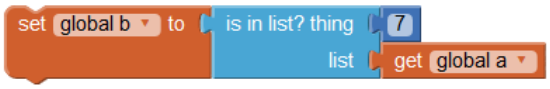


Abbildung 9: Liste im MIT App Inventor

In der Programmiersprache C++ muss man allerdings einen geeigneten Datentyp und eine geeignete Größe für sein Array wählen. Dadurch ist dieses Konzept eher statisch und für die EntwicklerInnen kaum flexibel und anpassungsfähig.

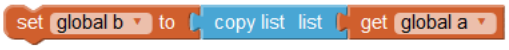
	In C++ für Programmier-anfängerInnen nicht ohne weiteres möglich, da Wissen über Speicherverwaltung bzw. Speicherallokation erforderlich ist.
	`a[0] = 11;`

	`b = false;`
	`for (int i = 0; i < groesse - 1; i++)`
	`  if (a[i] == 7)`
	`  b = true;`

	`for (int i = 2; i < groesse - 1; i++)`
	`  a[i] = a[i+1];`
	allerdings passt sich durch diese Verschiebung das Array nicht an die neue Größe an

	Nachdem Arrays in C++ mit einer statischen Größe erzeugt werden, ist diese den EntwicklerInnen im Regelfall bekannt. Ist es allerdings möglich, dass sich die Größe während der Laufzeit ändert, so liefert die Berechnung unter Berücksichtigung des Speicherverbrauchs in Bytes: `sizeof(a) / sizeof(a[0])` zuverlässige Ergebnisse.

Tabelle 18: Container Vergleich 1/2 App Inventor / C++

	Für Arrays wird ein bestimmter Speicherbereich durch die Belegung unterschiedlicher Werte reserviert, wodurch es eigentlich nie ohne Inhalt existieren kann
	`for (int i = 0; i < groesse; i++)` `  b[i] = a[i];`

	`int c[] = {1, 2, 3, 4};`
	`a = c[rand() % 4];`
	allerdings muss für die Funktion rand() eine Standardbibliothek eingebunden werden:
	`#include <cstdlib>`

Tabelle 19: Container Vergleich 2/2 App Inventor / C++

Gerade wenn ein Programm die Speicherung mehrerer Werte unter einem benannten Objekt erfordert, könnte C++ für viele AnfängerInnen zu maschinennahe sein. Der MIT App Inventor kann hier überzeugen, da er eine grundlegende Dynamik für Listen mitbringt und häufig verwendete Funktionen bereits standardmäßig zur Verfügung stellt. Viele dieser Funktionen sind in C++ mit großem Implementierungsaufwand verbunden und erfüllen auch dann womöglich nicht immer die Problemstellung zu vollster Zufriedenheit.

In der Gegenüberstellung mit C++ wurde deutlich, dass häufige Anwendungsfälle wie z.B. das Entfernen eines Listenelements oder die Überprüfung, ob ein bestimmtes Element im Container enthalten ist, standardmäßig nicht ohne Weiteres realisierbar sind. Gerade wenn man diese Funktionen nicht „ausprogrammieren“, sondern unterstützend für andere grundlegende Konzepte und Algorithmen verwenden möchte, ist die simple Anwendung dieser Funktionen im App Inventor ein Vorteil.

Des Weiteren besteht sowohl beim App Inventor, als auch in C++ die Möglichkeit, mehrdimensionale Felder anzulegen und zu bearbeiten. Die Anzahl der Dimensionen scheint unbegrenzt, allerdings erhöht sich die Komplexität bei der Arbeit mit dem Objekt relativ rasch. Bereits eine einfache Matrix, bestehend aus Zeilen und Spalten, verlangt ProgrammieranfängerInnen sehr viel Abstraktionsvermögen ab und erfordert daher viel Übung und Geduld. Folgendes Beispiel legt ein 2x3 Feld, welches mit Zahlen initialisiert wird, an, iteriert durch dieses zeilenweise und gibt alle Werte innerhalb der Matrix aus:


```
int c[2][3] = { { 1, 2, 3 },
                { 1, 2, 3 } };

for (int i = 0; i < 2; i++)
    for (int j = 0; j < 3; j++)
        cout << c[i][j];
```

gibt *((1 2 3)(1 2 3))* aus	gibt *123123* aus

Tabelle 20: Mehrdimensionale Felder Vergleich App Inventor / C++

Abschließend sollen auch Aspekte der Fehlerbehandlung im Bezug auf Listen beim App Inventor genannt werden. Der App Inventor erkennt Indexverletzungen und gibt diese sowohl am Smartphone, als auch am Entwickler-PC aus:

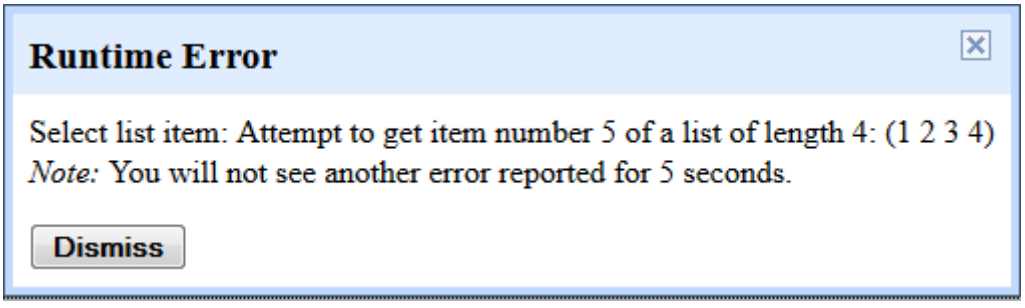


Abbildung 10: Fehlermeldung Liste im MIT App Inventor

Dabei ist auch die erfreuliche Eigenschaft zu nennen, dass der Index der Listen – wie in der Vielzahl an Beispielen ersichtlich – nicht bei 0, sondern bei 1 beginnt und daher, gerade für ProgrammieranfängerInnen, logischer erscheint.

Funktionen und Methoden

Funktionen werden dazu verwendet, um mehrere Anweisungen unter einem bestimmten Namen zusammenzufassen. Damit kann der Programmcode verständlicher werden, das Testen erleichtert werden, die Funktion an mehreren Programmstellen verwendet werden und man kann die eigentliche Programmieraufgabe von anderweitig erforderlichen Berechnungen logisch trennen. (vgl. Stroustrup, 2010, S. 142) Da Funktionen als isoliert vom restlichen Programmcode angesehen werden können, besteht die Möglichkeit, den Funktionen Werte von außen per Parameter zu übergeben oder Funktionen mit Rückgabewert zu definieren, damit die Ergebnisse einer Funktion nach außen gelangen.

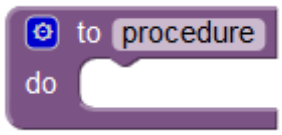
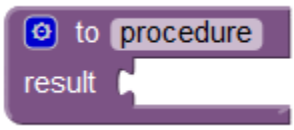
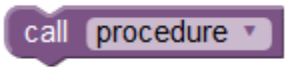
	Fasst eine oder mehrere Anweisungen unter einem Namen als Funktion zusammen, kann optional über Parameter verfügen und liefert keinen Wert zurück
	Fasst eine oder mehrere Anweisungen unter einem Namen als Funktion zusammen, kann optional über Parameter verfügen und liefert einen Wert zurück
	Aufruf einer Funktion ohne Rückgabewert mit dem Namen und optional (falls definiert) mit Parameter
	Aufruf einer Funktion mit Rückgabewert mit dem Namen und optional (falls definiert) mit Parameter

Tabelle 21: Funktionen im MIT App Inventor

Im folgenden Beispiel wird zunächst eine globale Variable *erg* mit dem Wert 0 angelegt. Dieser Variable wird anschließend der Wert aus dem Ergebnis der Funktion *Produkt* zugewiesen, welches durch die Parameter *x*, *y* und *z* berechnet wird.

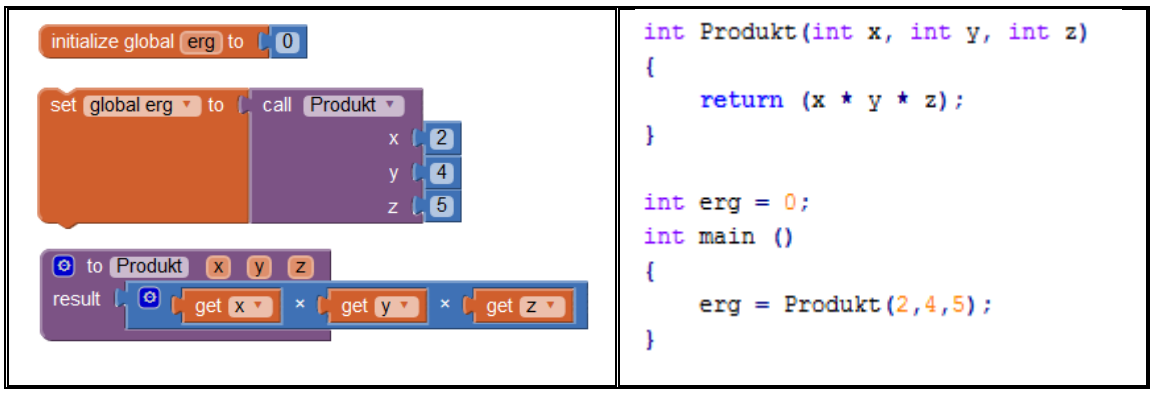


Tabelle 22: Funktionen Vergleich App Inventor / C++

Im oberen Beispiel fällt auf, dass die Geltungsbereiche im MIT App Inventor kaum eine Rolle spielen. Es gibt keine Main-Funktion, da die Anweisungen aufgrund bestimmter Events ausgeführt werden. So kann man z.B. die Wertzuweisung der globalen Variable *erg* vornehmen, wenn ein Screen umgeschaltet wird, ein Button gedrückt wird oder ein Slider bewegt wird. Außerdem spielt die Reihenfolge der Funktionen im Gegensatz zu C++ keine Rolle: Würde man im oberen Beispiel die Funktion *Produkt* unterhalb der Main-Funktion in C++ deklarieren, bekommt man die Fehlermeldung, dass die Funktion *Produkt* der Main-Funktion unbekannt ist. Als nächster Punkt ist auch die globale Variable *erg* zu beachten. Die Deklaration als globale Variable erfolgt im App Inventor aufgrund eines bestimmten Blockelements bzw. Puzzleteils. In C++ hingegen unterscheiden sich lokale und globale Variablen nicht aufgrund verschiedener Schlüsselwörter, sondern durch die Deklaration in verschiedenen Geltungsbereichen. Um effizienten Programmcode in C++ zu erzeugen, benötigen die SchülerInnen daher das Wissen über verschiedene Geltungsbereiche und das Wissen über die syntaktische Kennzeichnung dieser. Eine falsche Klammersetzung reicht aus, damit das Programm nicht mehr startet oder nicht ordnungsgemäß funktioniert. Dieser Umstand wird beim App Inventor durch das Blockprinzip vermieden.

Klassen und Objekte

Im Gegensatz zu der Programmiersprache C++ ist es in der Entwicklungsumgebung des MIT App Inventor nicht möglich, eigene Klassen zu erstellen und somit vernünftig das Klassenkonzept bzw. Objektorientierung im Unterricht mit diesem Tool zu lehren. Dennoch besteht ein großer Teil der Controls des App Inventor auch aus Objekten und Methoden – allerdings benötigt man die notwendige Expertise, um diese auch als solche zu erkennen und wahrzunehmen. Folgende Controls und ihre Eigenschaften wurden daher ausgewählt, um die Objektorientierung im MIT App Inventor exemplarisch zu demonstrieren:

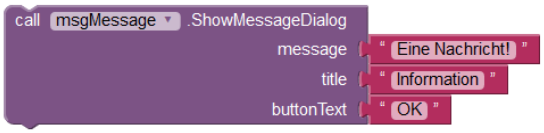
	Zugriff auf eine Eigenschaft des Objekts *btnButton*: Höhe wird auf 150 Pixel gesetzt
	Zugriff auf eine Eigenschaft des Objekts *btnButton*: Inhalt von *Text* wird zurückgeliefert
	Aufruf einer Methode des Objekts *msgMessage*: Erzeugung eines Message-Dialogs auf Basis der Parameter *message*, *title* und *buttonText*
	EventHandler im Objekt *chkCheck*: Ausgelöst durch das Ändern des Wertes werden die darunterliegenden Anweisungen ausgeführt

Tabelle 23: Objektorientierung im MIT App Inventor

Persönliches Fazit

Nachdem ich mich im Rahmen dieser Arbeit intensiv mit dem MIT App Inventor auseinandergesetzt habe, erlaube ich mir an dieser Stelle eine subjektive Zusammenfassung der wesentlichen Vor- und Nachteile, um die Grenzen und Möglichkeiten dieses Tools zusammenzufassen und übersichtlich wiederzugeben.

„This course helped me see how powerful modern technology really is. Before this class, I never in a million years thought that creating apps for phones was so accessible and simple to create/test." (Auszug der Meinung eines Studierenden nach der Programmiererfahrung mit dem MIT App Inventor in Wolber, 2011, S. 2)

Vorteile

- Kostenlos verfügbar
- Neues, modernes und intuitives Tool
- Kann Scratch bzw. SNAP! ersetzen, um den Fokus auf mobile Geräte zu legen
- Wird derzeit ständig weiterentwickelt
- Kompatibel mit einer breiten Palette von Betriebssystemen & Browsern
- Visuelle Programmiersprache / Entwicklungsumgebung, um den Schwierigkeiten von ProgrammieranfängerInnen entgegenzuwirken
- Emulator erlaubt auch den Unterricht mit SchülerInnen, welche keine mobilen Geräte mit Android-Betriebssystem besitzen
- Obwohl der Fokus bei der App-Entwicklung in der Lehre bzw. dem Bildungsbereich liegt, kann jeder Apps entwickeln und diese sogar über den Google Play Store zum Verkauf stellen
- Es werden sehr viele mächtige Controls bereitgestellt, welche es relativ einfach ermöglichen, komplexere und spannende Apps zu gestalten z.B. durch GPS-Lokalisierung, Barcode Scanner, etc.
- Cloud-Speicherung ermöglicht bequemen Zugriff auch von zu Hause aus
- Es wird ein Forum für Fragen der Community bereitgestellt

- Es gibt weiterführende Literatur und eine gute Online-Dokumentation für die Bedienung der Entwicklungsumgebung

Nachteile

- EntwicklerInnen benötigen einen Google-Account
- Zur App-Entwicklung benötigt man einen Internetzugang
- Spannende Apps mit vollem Funktionsumfang benötigen Android-Geräte, da der Emulator nicht alle Funktionen ordentlich simulieren kann z.B. GPS, Wischen am Bildschirm, Bewegungen am mobilen Gerät, etc.
- Professionelle App-Entwicklung ist nur beschränkt möglich
- Objektorientierung im Unterricht ist nicht wirklich möglich
- Der Wechsel auf eine Programmiersprache mit strengeren Umgang mit Datentypen ist womöglich teilweise problematisch
- SchülerInnen lernen nicht den Umgang mit einem Debugger
- Derzeit ausschließlich in englischer Sprache verfügbar
- Größere Programme können schnell unübersichtlich werden, da es z.B. nicht die Möglichkeit gibt, einen Zeilenumbruch bei Bedingungen vorzunehmen, etc.
- Programmieren auf mehreren Screens ist trotz weniger Codezeilen sehr langsam und führt dadurch teilweise zu Abstürzen

Teil III.
Beispiele für den AHS-Programmierunterricht

Grundlegende Informationen zu den Beispielen

Im Folgenden sind zehn Beispiele für den anfänglichen Programmierunterricht im AHS-Informatikunterricht und zehn Beispiele für den fortgeschrittenen Programmierunterricht im AHS-Schwerpunktfach Informatik gelistet und entsprechend gekennzeichnet.

Jedes der Beispiele enthält

- die Information über die Zielgruppe
- eine Auflistung von Zielen (Lehr- und/oder Lernzielen), welche ich bei einem konkreten Beispiel erreichen möchte
- eine kurze Aufgabenstellung über den zu programmierenden Sachverhalt
- sämtliche Verweise auf verwendete Quellen wie z.B. Bilder
- meine Beispiellösung zu der Aufgabenstellung
- Exemplarische Vorschläge für zusätzliche Programmieraufgaben für schnellere bzw. leistungsstärkere SchülerInnen
- Bildmaterial zur Veranschaulichung von möglichem App-Design und zur Demonstration des Praxiseinsatzes einer Anwendung

Dabei ist zu beachten, dass ich sämtliche Aufgabenstellungen eigenständig entwickelt habe und falls Ideen zu bestimmten Aufgaben von anderen Quellen entnommen wurden, diese entsprechend gekennzeichnet sind. Beim Material, welches für die Programmierung von einigen Beispielen erforderlich ist, habe ich darauf geachtet, dass dieses die entsprechenden Nutzungsrechte besitzt und daher direkt von Lehrpersonen verwendet werden kann. Meine Lösung der Aufgaben ist bei vielen Beispielen nur eine Möglichkeit von vielen, kann aber von Lehrpersonen als Musterlösung zur Hand genommen werden. Die Programmieraufgaben sind zwar nach Zielgruppen, allerdings nicht zwingenderweise aufbauend nach Schwierigkeit geordnet. Diese Entscheidung obliegt den Lehrerinnen und Lehrern, unter der Berücksichtigung von etwaigen Vorkenntnissen der SchülerInnen, selbst.

Einfacher Taschenrechner

Zielgruppe: SchülerInnen im Pflichtgegenstand Informatik

Ziele - Die SchülerInnen sollen:

- Ein- und Ausgabe implementieren und anwenden können
- Einfache Rechenalgorithmen implementieren und testen können

Aufgabenstellung:

Entwicklung eines einfachen Taschenrechners: Zunächst sollen zwei Zahlen mittels Textfeldern eingegeben werden. Danach wird je nach Button-Klick eine der vier Grundrechnungsarten (Addition, Subtraktion, Multiplikation, Division) auf die beiden Zahlen angewandt und das Ergebnis ausgegeben. Ein zusätzlicher Button namens „C“ soll die Eingaben und das Ergebnis löschen.

Material: wird für die grundlegende Anwendung nicht benötigt

Mögliche Beispiellösung:

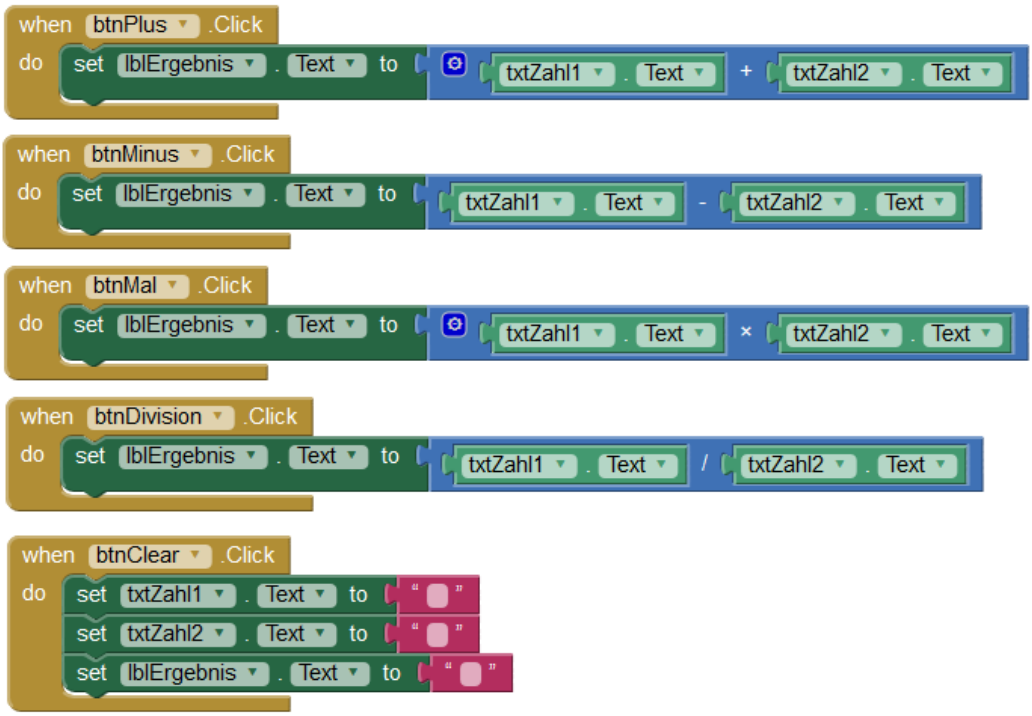


Abbildung 11: Lösung von Einfacher Taschenrechner

Einige Erweiterungen für leistungsstärkere SchülerInnen:

- Eingabeüberprüfung implementieren
- Weitere mathematische Funktionen implementieren z.B. Sinus
- Optische Verbesserungen im Design vornehmen
- Die Möglichkeit, mit dem Ergebnis weiterzurechnen implementieren

Praktische Anwendung:

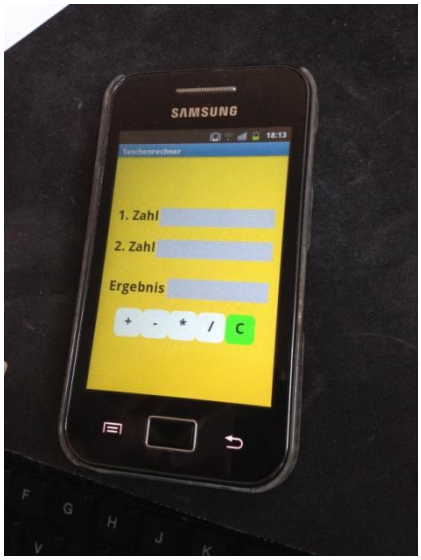


Abbildung 12: Praxis 1/2 von Einfacher Taschenrechner

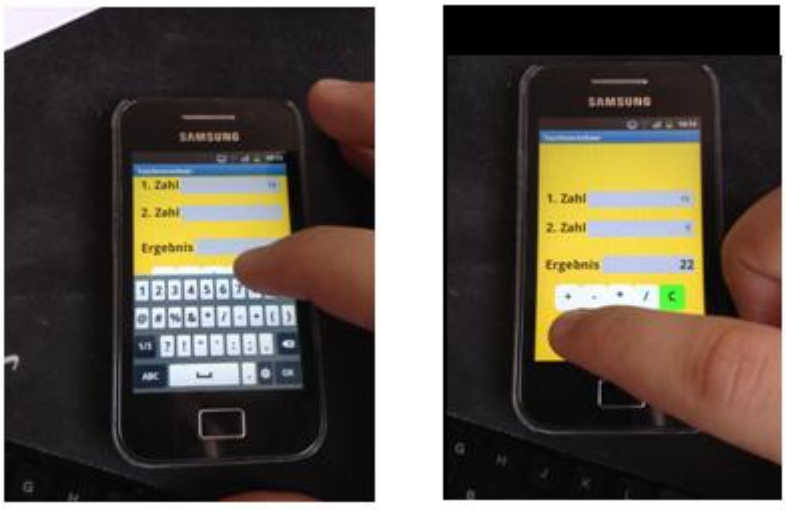


Abbildung 13: Praxis 2/2 von Einfacher Taschenrechner

Magic Ball

Zielgruppe: SchülerInnen im Pflichtgegenstand Informatik

Ziele - Die SchülerInnen sollen:

- Verschiedene Controls kennen und bedienen lernen
- Eine multimediale App erstellen können
- Einfache Zufallsfunktionen und Listen implementieren

Aufgabenstellung:

Die Idee stammt direkt vom Team des MIT App Inventors (http://appinventor.mit.edu/explore/teach/magic-8-ball.html, letzter Aufruf: 10.06.2015) und wurde damals noch für die erste Version des App Inventors entwickelt. Beim Klick auf den Magic Ball soll ein zufälliger Vorschlag auf eine Entscheidungsfrage erscheinen. Zusätzlich soll dabei Musik ertönen.

Material: [3]

Abbildung 14: Magic Ball

Als Sound-File wurde folgende Datei von Marianne Gagnon verwendet: http://soundbible.com/1671-Mystic-Chanting-1.html, letzter Aufruf: 10.06.2015

[3] Magic Ball ist online unter http://upload.wikimedia.org/wikipedia/commons/9/90/Magic8ball.jpg, letzter Aufruf: 10.06.2015

Mögliche Beispiellösung:

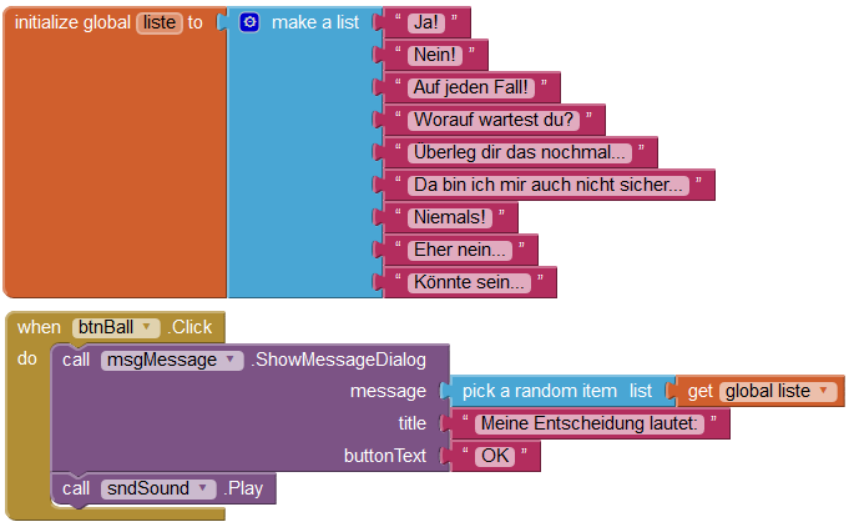


Abbildung 15: Lösung von Magic Ball

Einige Erweiterungen für leistungsstärkere SchülerInnen:

- Statt Klick auf den Magic Ball soll bei der Bewegung des Smartphones eine Antwort erscheinen
- Mehrere Sound-Files sollen eingebunden werden und ein zufälliges Geräusch bei einer Antwort ertönen

Praktische Anwendung:

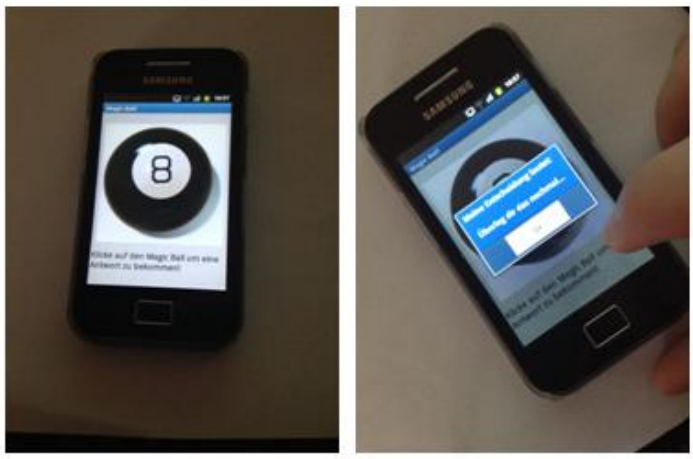


Abbildung 16: Praxis von Magic Ball

Zahlenraten

Zielgruppe: SchülerInnen im Pflichtgegenstand Informatik

Ziele - Die SchülerInnen sollen:

- Unterschiedliche Notifier-Controls kennenlernen und anwenden können
- Einfache Verzweigungen und bedingte Anweisungen implementieren

Aufgabenstellung:

Beim Zahlenraten spielt der Spieler gegen den Computer. Dazu soll der Spieler eine ganze Zufallszahl zwischen 0 und 100 erraten. Nach jedem Rateversuch bekommt der Spieler einen Hinweis, ob er die Zahl erraten hat oder ob seine geratene Zahl zu niedrig bzw. zu hoch war. Der Spieler hat eine unbegrenzte Anzahl an Rateversuchen.

Material: [4]

Abbildung 17: Zahlenfeld

Diese Abbildung wurde als Hintergrundbild für die Anwendung verwendet, um ein ansprechenderes Design zu ermöglichen.

[4] Zahlenfeld ist online unter http://pixabay.com/static/uploads/photo/2013/05/30/09/16/pay-114649_640.jpg, letzter Aufruf: 11.06.2015

Mögliche Beispiellösung:

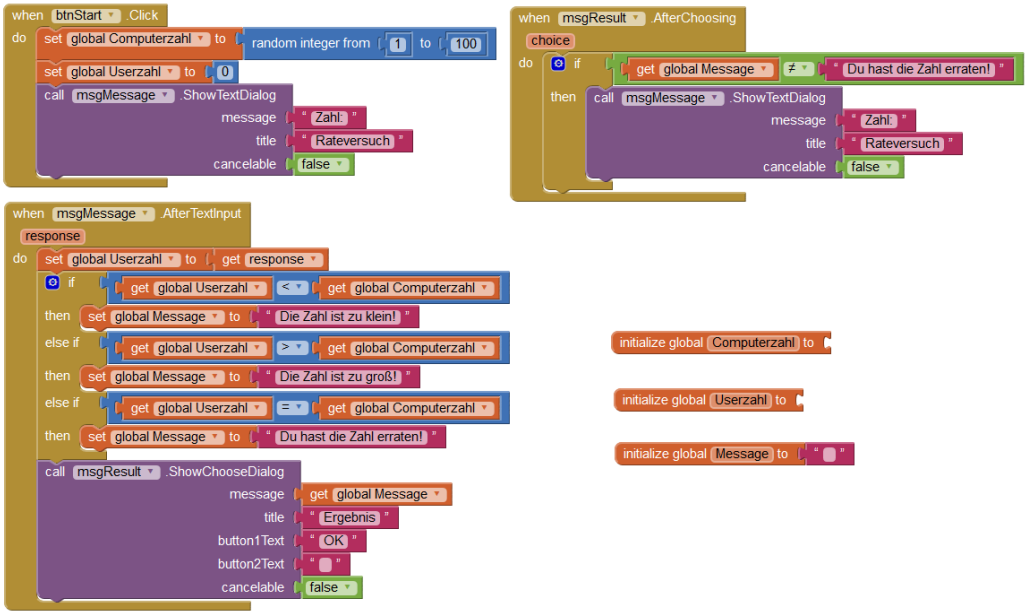


Abbildung 18: Lösung von Zahlenraten

Einige Erweiterungen für leistungsstärkere SchülerInnen:

- Eingabeüberprüfung implementieren
- Die Anzahl der Versuche soll mitgezählt und ausgegeben werden
- Es soll eine wählbare, begrenze Anzahl an Rateversuchen geben
- Spieß umdrehen: Der Computer soll eine gedachte Zahl des Users erraten

Praktische Anwendung:

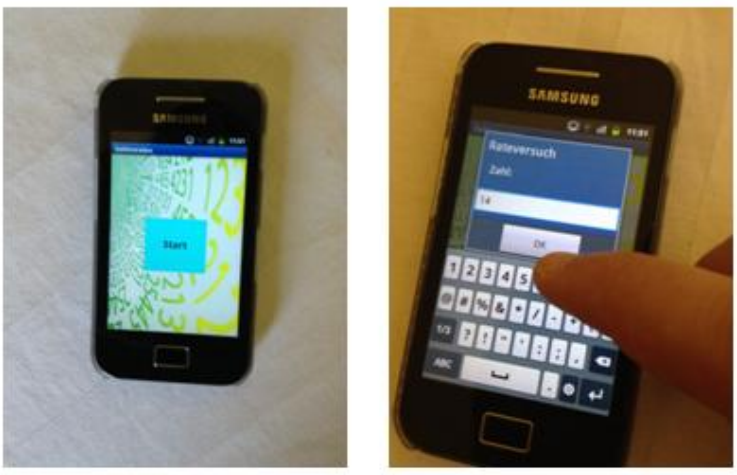


Abbildung 19: Praxis von Zahlenraten

Zeichenprogramm

Zielgruppe: SchülerInnen im Pflichtgegenstand Informatik

Ziele - Die SchülerInnen sollen:

- Das Canvas-Control kennenlernen und sinnvoll anwenden können
- Einfache Zeichenanwendungen erstellen können

Aufgabenstellung:

Die Anwendung soll eine Bewegung mit dem Finger auf dem Bildschirm grafisch darstellen können. Es soll die Option bestehen, die aktuelle Zeichenfarbe (Rot, Blau, Grün, Schwarz oder Gelb), sowie die aktuelle Linienstärke (5, 10, 15, 20 oder 25) während der Laufzeit umzustellen. Zusätzlich soll ein Reset-Button implementiert werden, welcher die aktuelle Zeichnung löscht, die Anwendung auf die Standardfarbe (=schwarz) und Standard-Linienstärke (=15) zurücksetzt.

Material: wird für diese Anwendung nicht benötigt

Mögliche Beispiellösung:

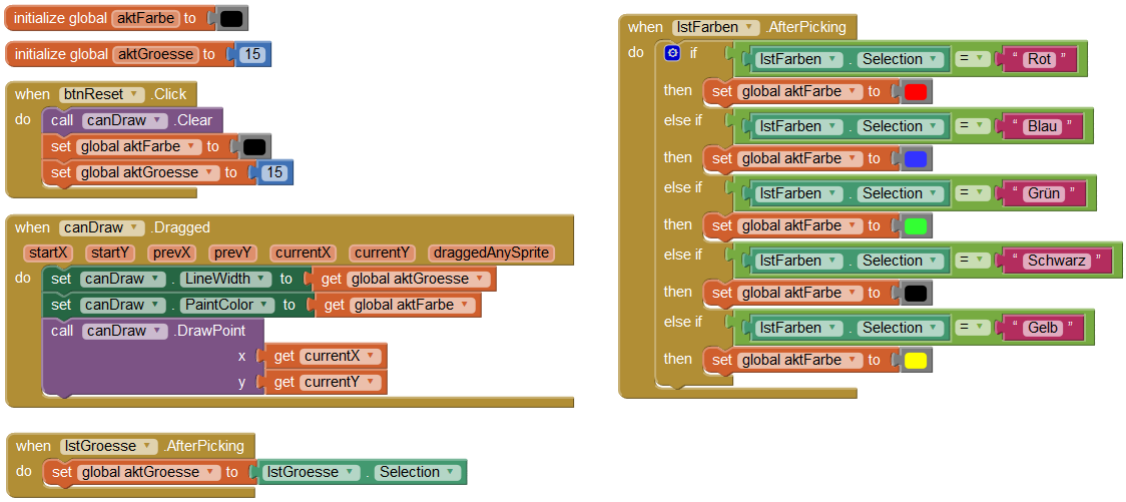


Abbildung 20: Lösung von Zeichenprogramm

Einige Erweiterungen für leistungsstärkere SchülerInnen:

- Erweiterung um weitere Farben und Strichstärken
- Stärkeauswahl in Form eines Slider implementieren
- Durch die Eingabe von RGB-Werten soll eine Zeichenfarbe erstellt werden können
- Die Hintergrundfarbe der Zeichenfläche soll einstellbar sein
- Bilder sollen in den Hintergrund geladen werden können

Praktische Anwendung:

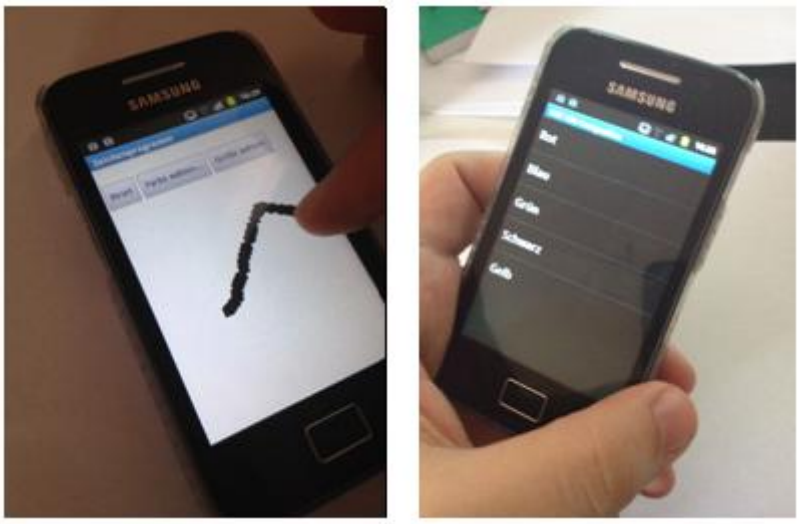


Abbildung 21: Praxis 1/2 von Zeichenprogramm

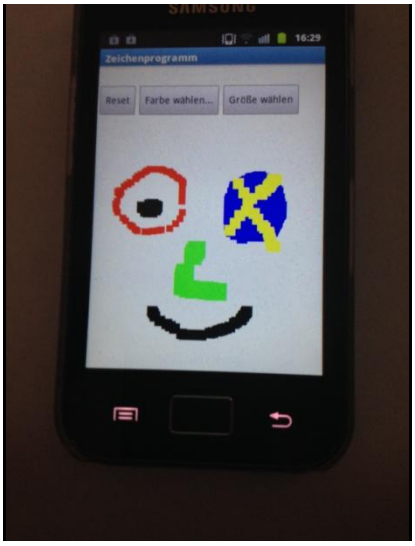


Abbildung 22: Praxis 2/2 von Zeichenprogramm

Fakultät

Zielgruppe: SchülerInnen im Pflichtgegenstand Informatik

Ziele - Die SchülerInnen sollen:

- Einfache Funktionen verstehen und anwenden können
- Einfache Rekursionen verstehen und anwenden können

Aufgabenstellung:

Die Anwendung ermöglicht die Berechnung der Fakultät der Zahlenwerte von 1 bis 9. Zur Berechnung der Fakultät soll eine rekursive Funktion implementiert werden. Beim Klick auf eine Zahl wird das entsprechende Ergebnis in Form eines Message-Dialogs angezeigt.

Material: wird für die grundlegende Anwendung nicht benötigt

Mögliche Beispiellösung:

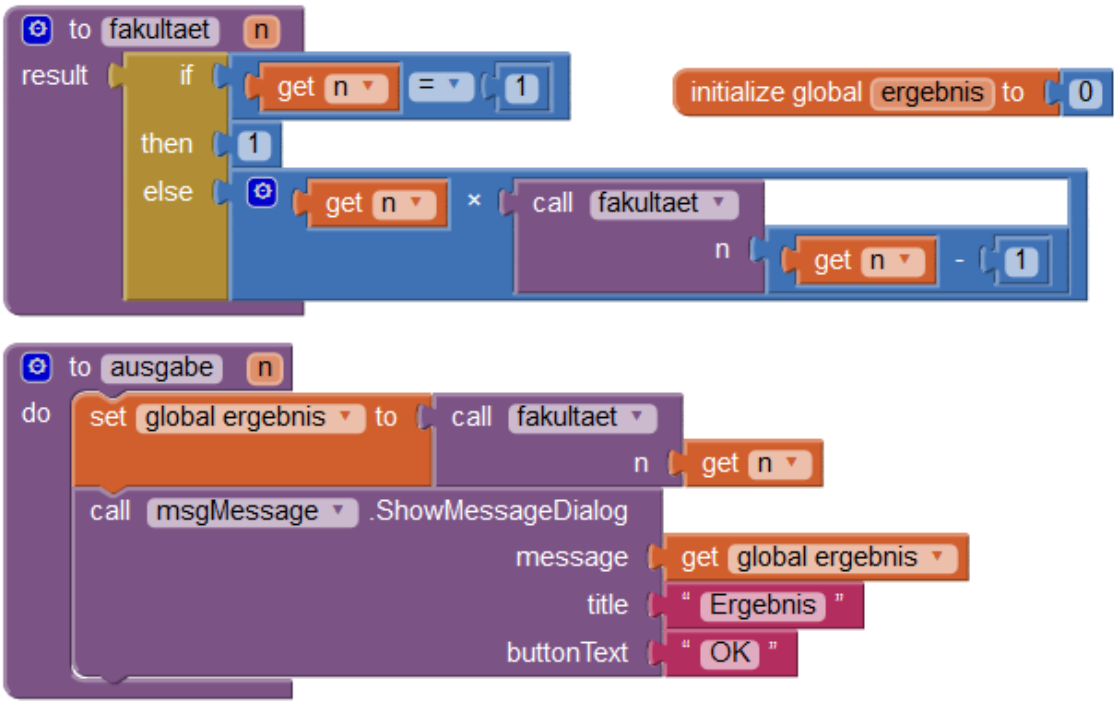


Abbildung 23: Lösung 1/2 von Fakultät

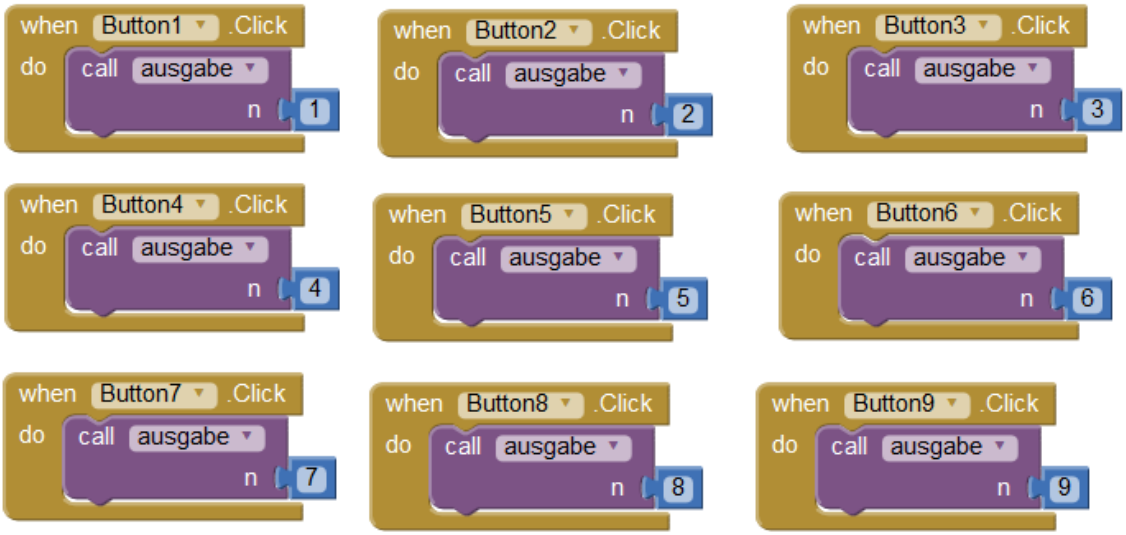


Abbildung 24: Lösung 2/2 von Fakultät

Einige Erweiterungen für leistungsstärkere SchülerInnen:

- Eingabe zusätzlicher Zahlen soll möglich sein
- Hilfe-Funktion soll die Formel zur Berechnung der Fakultät anzeigen

Praktische Anwendung:

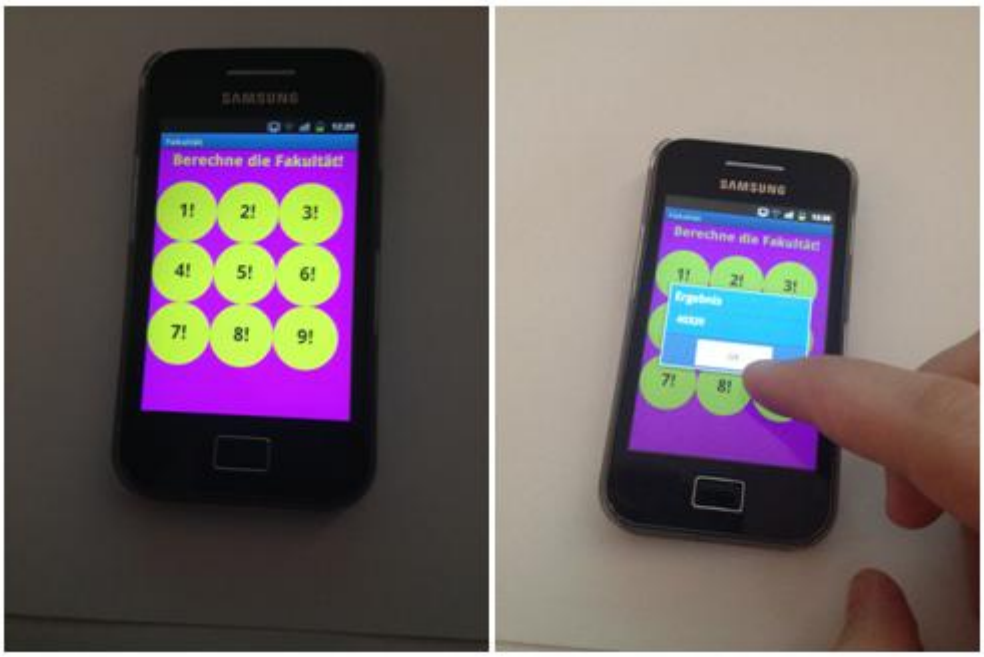


Abbildung 25: Praxis von Fakultät

Gerade und ungerade Zahlen

Zielgruppe: SchülerInnen im Pflichtgegenstand Informatik

Ziele - Die SchülerInnen sollen:

- Die Restdivision (Modulo) kennenlernen und anwenden können
- Einfache Zählerschleifen verstehen und implementieren können

Aufgabenstellung:

Die Anwendung besteht aus einem einfachen Interface mit zwei Buttons und einem Ausgabefeld. Beim Klick auf den ersten Button sollen alle geraden Zahlen zwischen 1 und 100 ausgegeben werden. Beim Klick auf den zweiten Button sollen alle ungeraden Zahlen zwischen 1 und 100 ausgegeben werden. Zur Lösung der Aufgabe soll die Modulo-Funktion angewandt werden.

Material: wird für diese Anwendung nicht benötigt

Mögliche Beispiellösung:

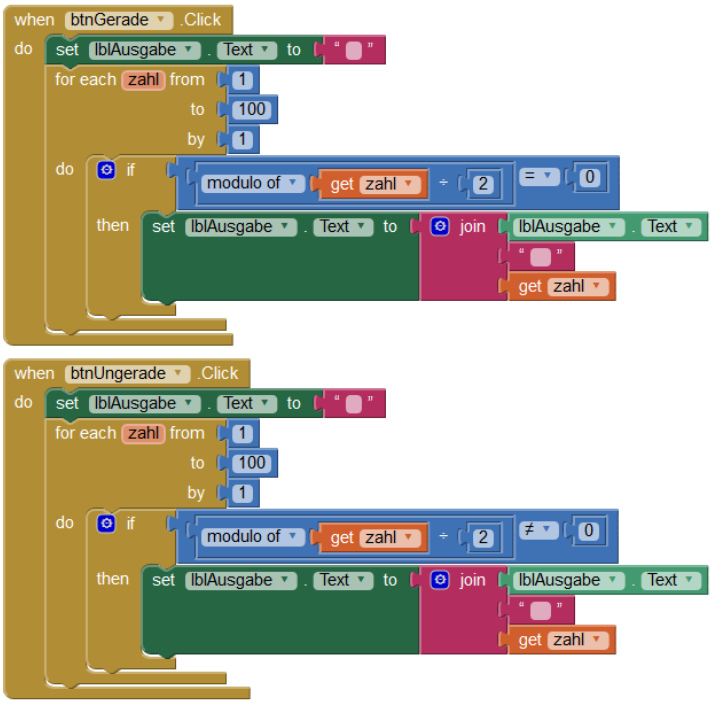


Abbildung 26: Lösung von Gerade und ungerade Zahlen

Einige Erweiterungen für leistungsstärkere SchülerInnen:

- Das Intervall für die Ausgabe von geraden und ungeraden Zahlen soll nicht mehr ausschließlich zwischen 0 und 100 liegen, sondern individuell einstellbar sein. Dabei können unterschiedliche Controls zur Anwendung kommen, wie z.B. ein Slider oder eine Textbox
- Es soll auch die Möglichkeit geben, alle Primzahlen in einem bestimmten Intervall auszugeben
- Die Eingabe der Intervalle soll über den SpeechRecognizer erfolgen. Dieser liefert praktischerweise die Zahlenwörter über das Mikrofon des mobilen Geräts direkt als Zahlen zurück. Spricht der User z.B. „Hundert“ ins Mikrofon wird 100 zurückgeliefert.

Praktische Anwendung:

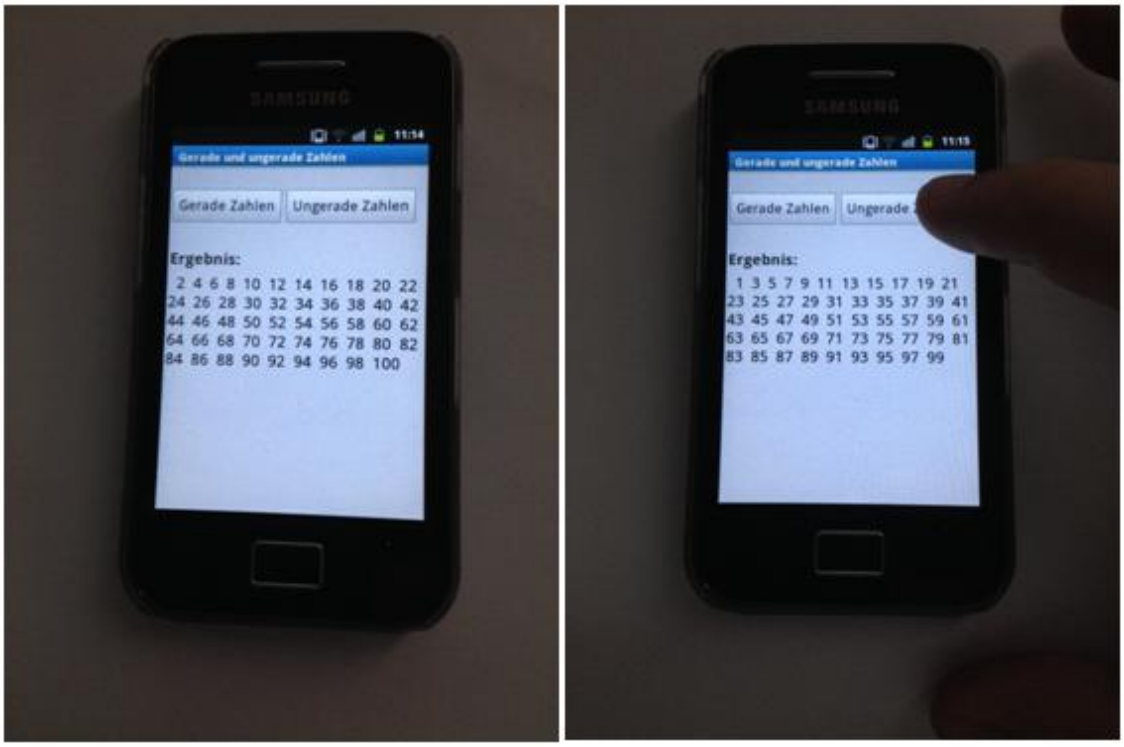


Abbildung 27: Praxis von Gerade und ungerade Zahlen

Der lachende Alien

Zielgruppe: SchülerInnen im Pflichtgegenstand Informatik

Ziele - Die SchülerInnen sollen:

- Verschiedene Controls kennen und bedienen lernen
- Einfache Verzweigungen anwenden können
- Eine multimediale App erstellen

Aufgabenstellung:

Inspiriert von der App „Lachsack“ (vgl. Stadtmann, 2013, S. 12ff) werden die SchülerInnen eine mobile Anwendung entwickeln, welche einen kleinen Außerirdischen abbildet. Klickt man auf den Alien, so wechselt dieser je nach aktueller Hautfarbe seine Farbe (Rot wird Grün, Grün wird Blau und Blau wird wieder Rot). Zusätzlich wird beim Farbwechsel ein vorgegebenes Sound-File abgespielt, wodurch ein lachendes Geräusch aus dem Gerät ertönt. Ein zusätzlicher Button erlaubt das Beenden der Anwendung.

Material: [5]

Abbildung 28: Aliens

Als Sound-File für das Lachgeräusch wurde folgende Datei von Mike Koenig verwendet: http://soundbible.com/1123-Kid-Giggle-2.html, letzter Aufruf: 08.06.2015

[5] Alien ist online unter http://pixabay.com/static/uploads/photo/2012/04/12/19/27/alien-30287_640.png, letzter Aufruf: 08.06.2015 - wurde dreimal eingefärbt und verändert

Mögliche Beispiellösung:

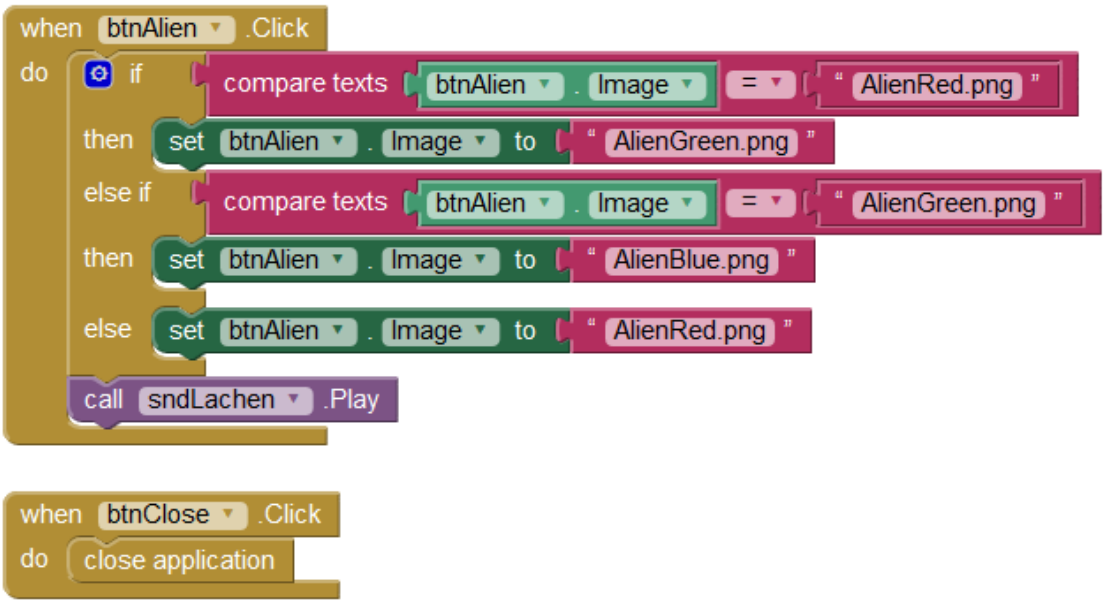


Abbildung 29: Lösung von Der lachende Alien

Einige Erweiterungen für leistungsstärkere SchülerInnen:

- Unterschiedliche Sound-Files je nach Farbe einbinden
- Bilder bearbeiten, sodass der Alien am Bildschirm seine Position wechselt

Praktische Anwendung:

Abbildung 30: Praxis von Der lachende Alien

Countdown

Zielgruppe: SchülerInnen im Pflichtgegenstand Informatik

Ziele - Die SchülerInnen sollen:

- Zeitliche Steuerung mittels Clock-Control vornehmen können
- Time-Picker kennenlernen und implementieren
- Einfache Funktionen verstehen und implementieren

Aufgabenstellung:

Die SchülerInnen programmieren einen einfachen Countdown. Dazu soll nach Auswahl einer Startzeit mittels Time-Picker die Anwendung im Sekundentakt runter zählen. Ist die Uhr bei 00:00:00 angelangt, soll ein kurzes Alarmsignal einmalig ertönen. Da der Time-Picker nur die Auswahl von Stunden und Minuten ermöglicht, soll zusätzlich das Feature implementiert werden, dass man direkt von einer Minute runter zählen und einen Alarm auslösen kann.

Material:

Als Sound-File für den Alarm wurde folgende Datei von Mike Koenig verwendet: http://soundbible.com/1251-Beep.html, letzter Aufruf: 15.06.2015

Mögliche Beispiellösung:

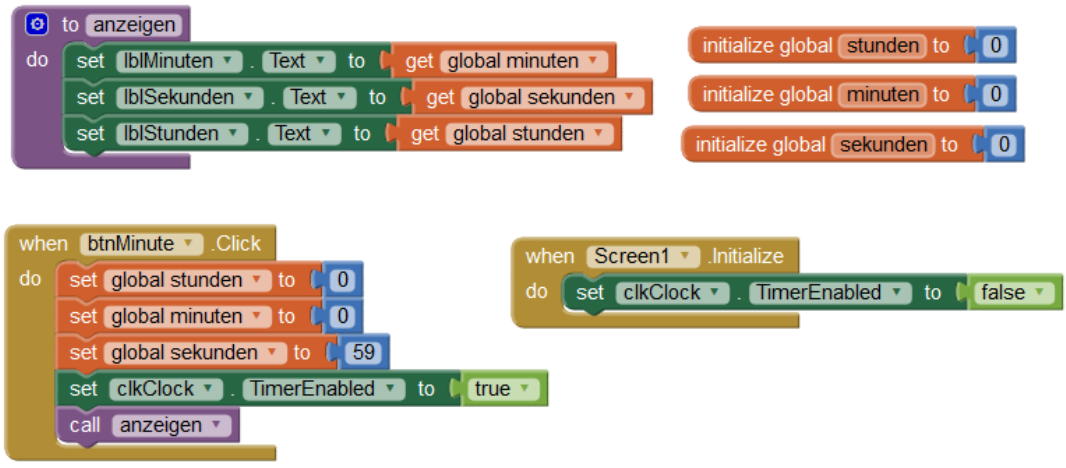


Abbildung 31: Lösung 1/2 von Countdown

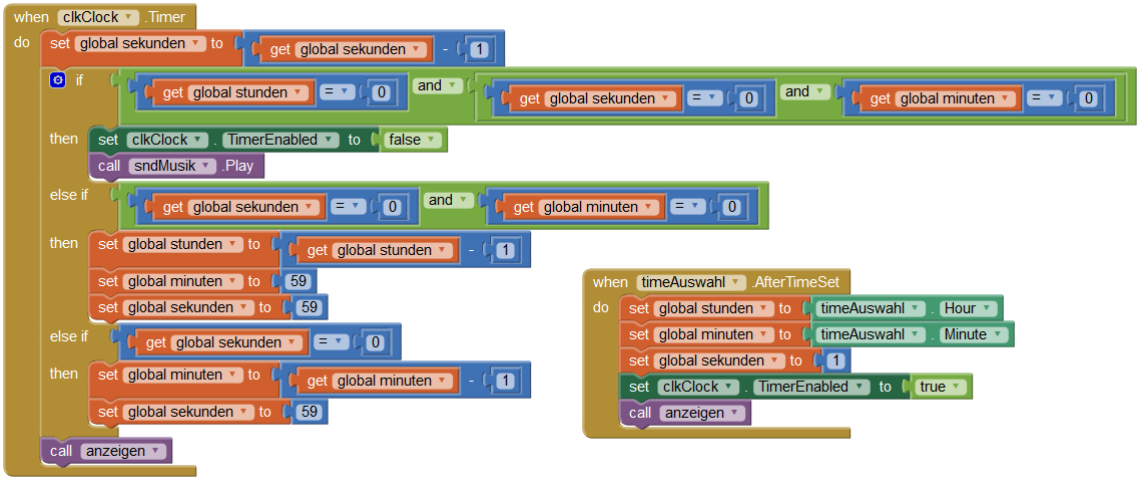


Abbildung 32: Lösung 2/2 von Countdown

Einige Erweiterungen für leistungsstärkere SchülerInnen:

- Anzeige von Hundertstel implementieren
- Stopp- bzw. Fortsetzen-Funktion implementieren
- Interface-Verbesserungen vornehmen

Praktische Anwendung:

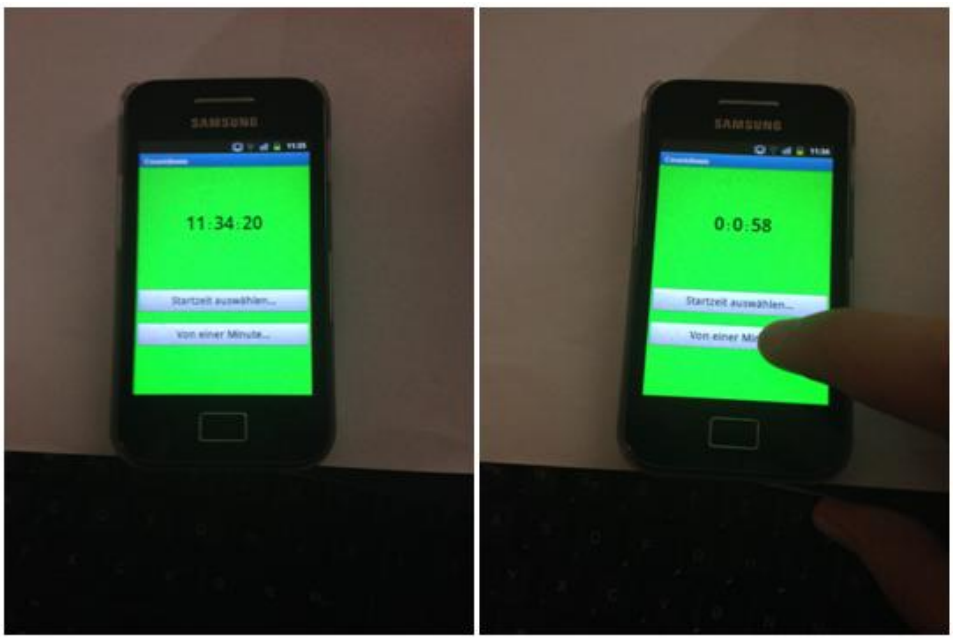


Abbildung 33: Praxis von Countdown

Monsterjagd

Zielgruppe: SchülerInnen im Pflichtgegenstand Informatik

Ziele - Die SchülerInnen sollen:

- Animationen in einem Canvas-Control erstellen können
- Verschiedene Objekte über das Clock-Control zeitlich steuern können

Aufgabenstellung:

Die Anwendung besteht aus einem Spielball und einem Monster, welches alle drei Sekunden irgendwo zufällig am Spielfeld erscheint. Der Spieler muss den Ball mit seinem Finger steuern und dabei versuchen, das Monster zu treffen. Gelingt ihm ein Treffer, verschwindet das Monster kurz und der Spieler bekommt einen Punkt. Zusätzlich soll die Möglichkeit bestehen, die Punkte wieder auf 0 zurückzusetzen.

Material: [6]

Abbildung 34: Monster

[6] Monster ist online unter http://pixabay.com/static/uploads/photo/2014/04/03/10/22/monster-310261_640.png, letzter Aufruf: 12.06.2015

Mögliche Beispiellösung:

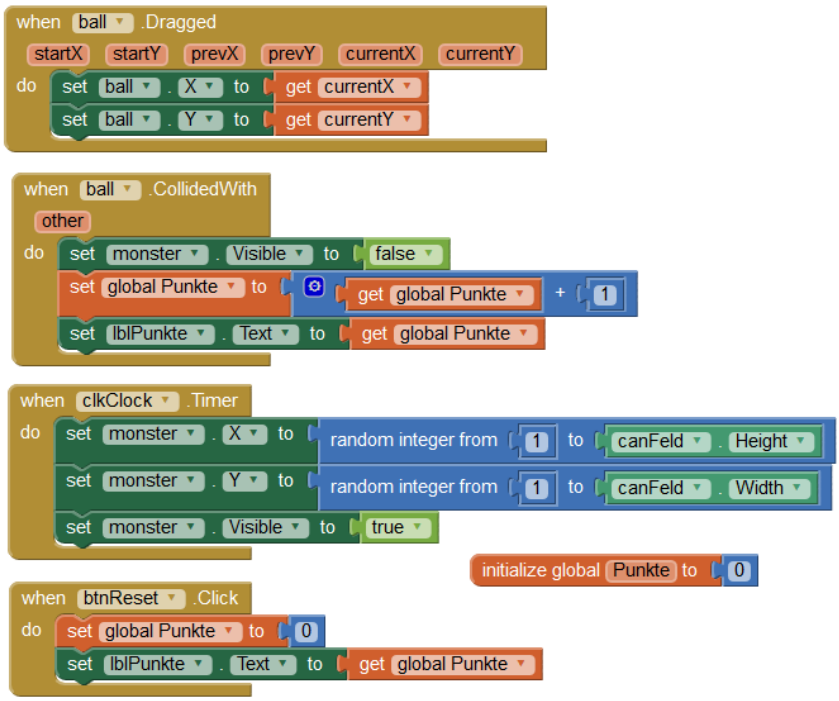


Abbildung 35: Lösung von Monsterjagd

Einige Erweiterungen für leistungsstärkere SchülerInnen:

- mehrere Monster sollen gleichzeitig am Spielfeld sichtbar sein
- Sound-Files einbinden

Praktische Anwendung:

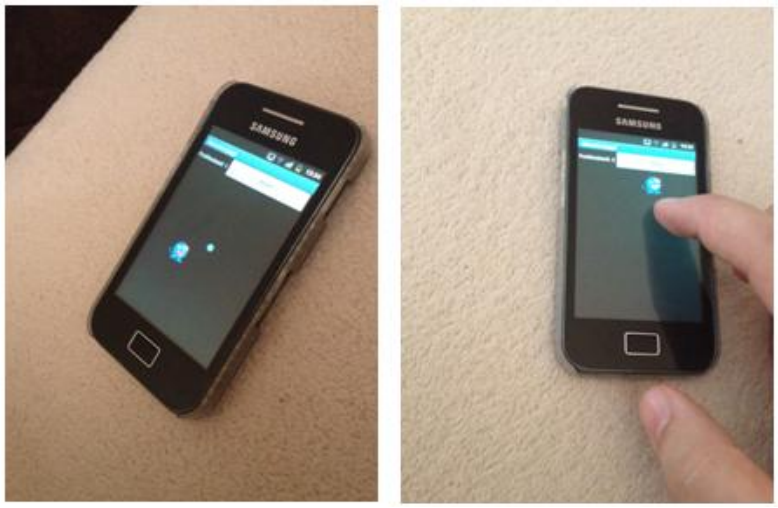


Abbildung 36: Praxis von Monsterjagd

Pfeile

Zielgruppe: SchülerInnen im Pflichtgegenstand Informatik

Ziele - Die SchülerInnen sollen:

- einfache Anwendungen mit mehreren Screens erstellen können

Aufgabenstellung:

Im Startbildschirm (Screen 1) werden vier Pfeile in Richtung der vier Himmelsrichtungen dargestellt. Bei einem Klick auf einen Pfeil erfolgt ein Wechsel des Screens. Auf diesem Screen wird ein großer Pfeil in Herkunftsrichtung angezeigt. Klickt man auf diesen Pfeil kommt man wiederum auf den Startbildschirm.

Material: [7]

Abbildung 37: Pfeile

Mögliche Beispiellösung:

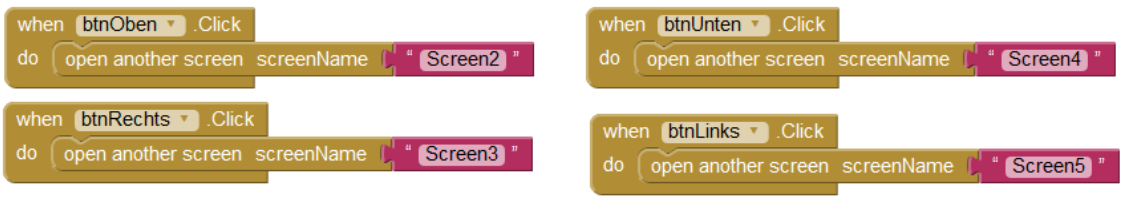


Abbildung 38: Lösung von Pfeile (Screen 1)

[7] Pfeil ist online unter http://pixabay.com/static/uploads/photo/2012/04/11/10/22/arrow-27315_640.png, letzter Aufruf: 15.06.2015 – wurde zusätzlich dreimal gedreht um die anderen Bilder zu erhalten

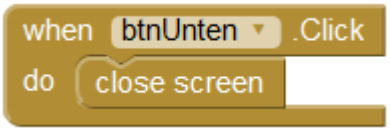


Abbildung 39: Lösung von Pfeile (Screen 2)

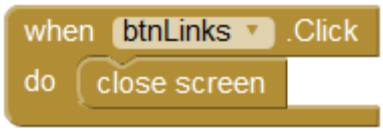


Abbildung 40: Lösung von Pfeile (Screen 3)

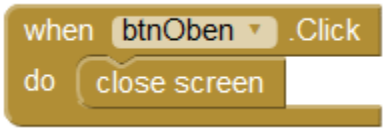


Abbildung 41: Lösung von Pfeile (Screen 4)

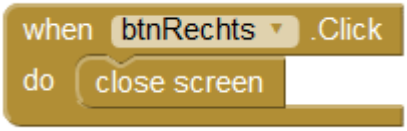


Abbildung 42: Lösung von Pfeile (Screen 5)

Anmerkung: Beim Arbeiten auf mehreren Screens bekommt jeder Screen einen eigenen Design-Editor und einen eigenen Blocks-Editor. Jeder Screen besteht daher aus der Anzeige von Controls und der damit verbundenen Logik.

Einige Erweiterungen für leistungsstärkere SchülerInnen:

- Sprachsteuerung der Richtungen z.B. beim Sprechen von „Links“ durch das Mikrofon soll auf den entsprechenden Screen gewechselt werden
- Der Titel der Screens soll sich während der Laufzeit dynamisch anpassen

Praktische Anwendung:

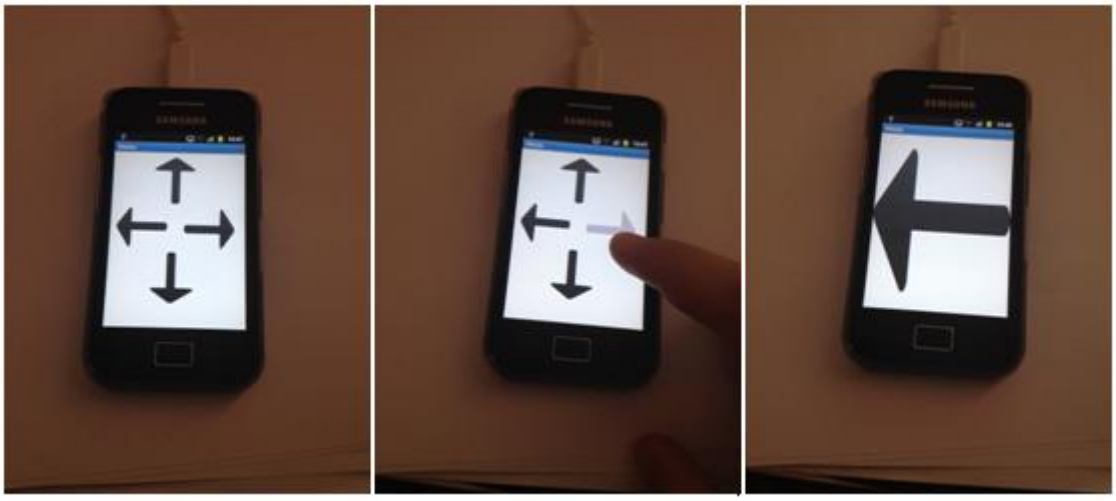


Abbildung 43: Praxis von Pfeile

Schere, Stein, Papier

Zielgruppe: SchülerInnen im Wahlpflichtgegenstand Informatik

Ziele - Die SchülerInnen sollen:

- Die Random-Funktion für eine einfache KI nutzen können
- Verschachtelte Verzweigungen und bedingte Anweisungen üben
- Einfache Handy-Funktionen (Bewegungen, Nachrichten) benutzen

Aufgabenstellung:

Die SchülerInnen entwickeln ein einfaches „Schere, Stein, Papier – Spiel". Dazu sollen sie das vorgebende Bildmaterial verwenden und sowohl dem Spieler, als auch dem Computer bei der Bewegung des Smartphones ein zufälliges Bild zuweisen. Anschließend soll der Sieger ermittelt werden: Schere schlägt Papier und wird von Stein geschlagen, Papier schlägt Stein und wird von Schere geschlagen, Stein schlägt Schere und wird von Papier geschlagen. Treffen zwei gleiche Objekte aufeinander, so wird die Spielrunde mit einem Unentschieden bewertet. Das Ergebnis nach einer Spielrunde soll in Form einer Nachricht am Bildschirm mitgeteilt werden.

Material: [8]

Abbildung 44: Schere, Stein, Papier

[8] Schere ist online unter http://pixabay.com/static/uploads/photo/2014/04/03/10/20/scissors-310134_640.png, letzter Aufruf: 08.06.2015
Stein ist online unter http://pixabay.com/static/uploads/photo/2012/04/16/11/07/rock-35522_640.png, letzter Aufruf: 08.06.2015
Papier ist online unter http://pixabay.com/static/uploads/photo/2012/04/24/17/40/document-40599_640.png, letzter Aufruf: 08.06.2015

Mögliche Beispiellösung:

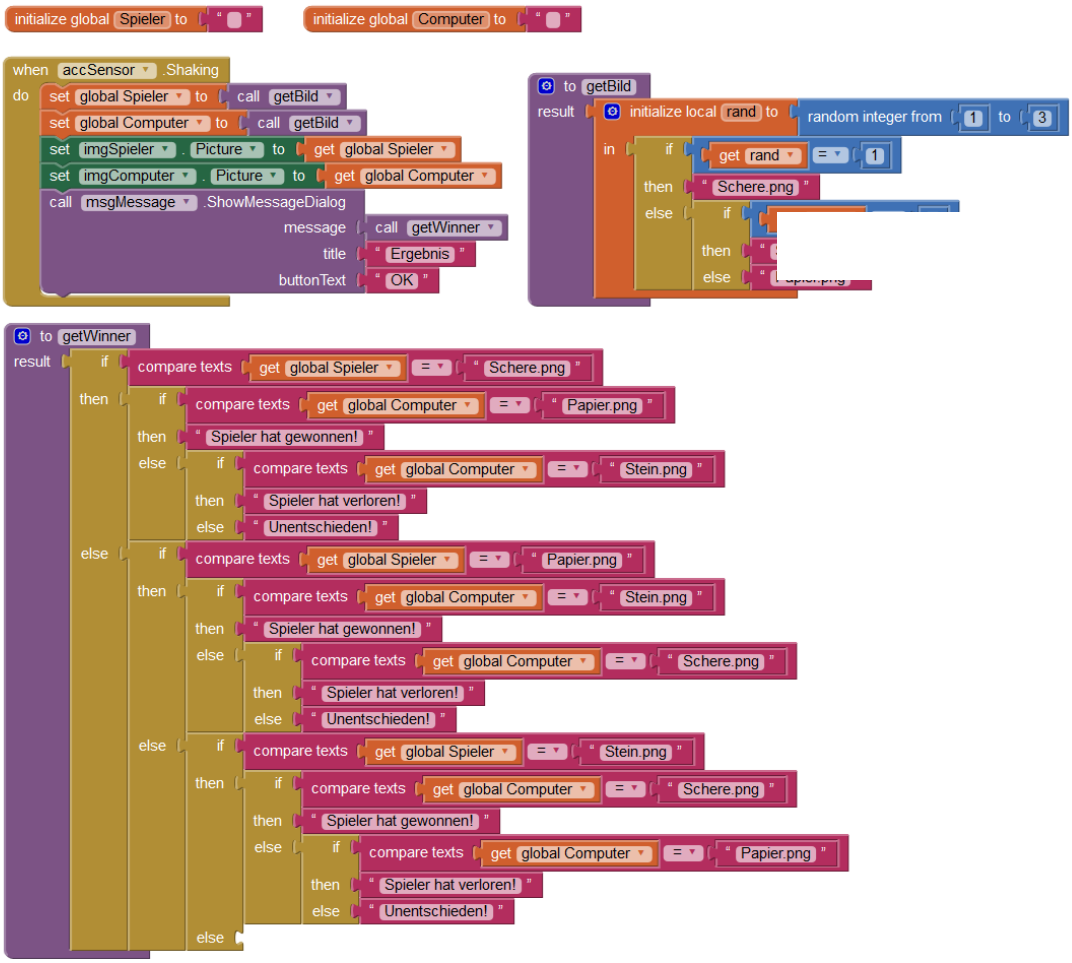


Abbildung 45: Lösung von Schere, Stein, Papier

Einige Erweiterungen für leistungsstärkere SchülerInnen:

- Einbindung von passenden Sound-Files
- Punktestand mitzählen und optional zurücksetzen
- Objekt-Auswahl für den Spieler ermöglichen
- Beliebige Design-Verbesserungen

Praktische Anwendung:

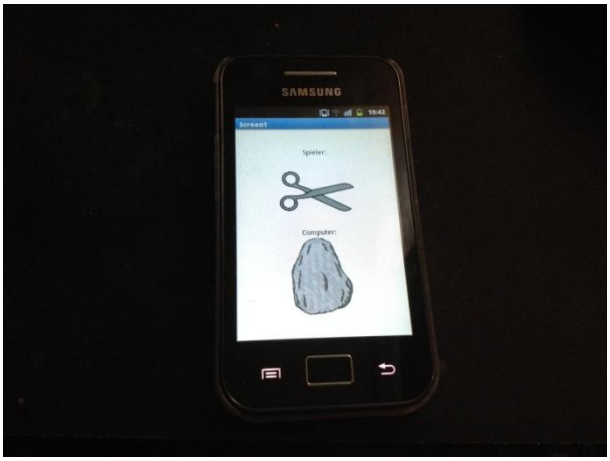


Abbildung 46: Praxis 1/3 von Schere, Stein, Papier

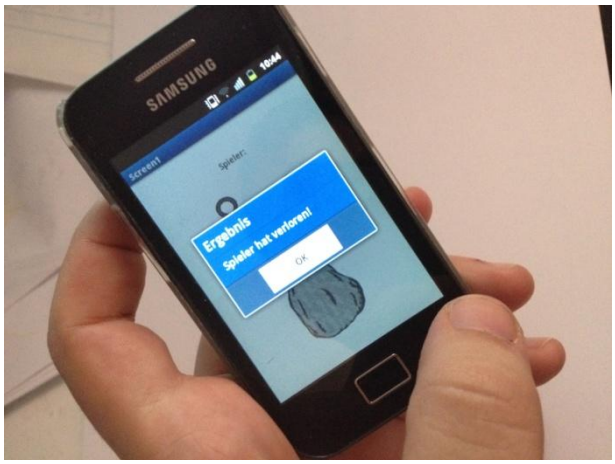


Abbildung 47: Praxis 2/3 von Schere, Stein, Papier

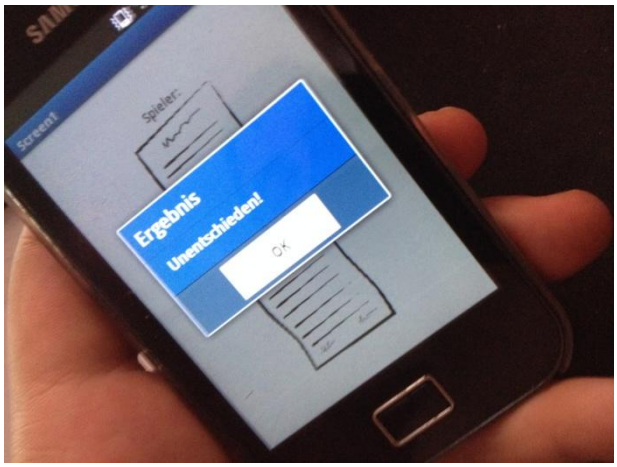


Abbildung 48: Praxis 3/3 von Schere, Stein, Papier

Tic Tac Toe

Zielgruppe: SchülerInnen im Wahlpflichtgegenstand Informatik

Ziele - Die SchülerInnen sollen:

- Passende Funktionen erstellen und diese effizient verwenden
- Verschachtelte Verzweigungen und bedingte Anweisungen üben
- Geeignete Lösungsalgorithmen finden und implementieren

Aufgabenstellung:

Die SchülerInnen programmieren ein „Tic Tac Toe – Spiel“. Dabei wird das Spielfeld in ein 3x3 Feld aus Button-Controls unterteilt. Durch den Klick auf einen Button wird dieser abwechselnd entweder mit „X“ für den Spieler 1 oder „O“ für den Spieler 2 beschriftet. Nach jedem Klick soll ermittelt werden, ob das Spiel von einem Spieler gewonnen wurde. Dazu muss dieser entweder in einer Reihe, einer Spalte oder einer Diagonale drei gleiche Symbole erzeugen. Sind alle Felder besetzt und ging kein Spieler als Sieger hervor, wird die aktuelle Spielrunde mit Unentschieden beendet. Das Spiel soll so ausgelegt sein, dass zwei reale Spieler auf einem mobilen Gerät gegeneinander spielen können.

Material: zusätzliches Material wird für das Grundspiel nicht benötigt

Mögliche Beispiellösung:

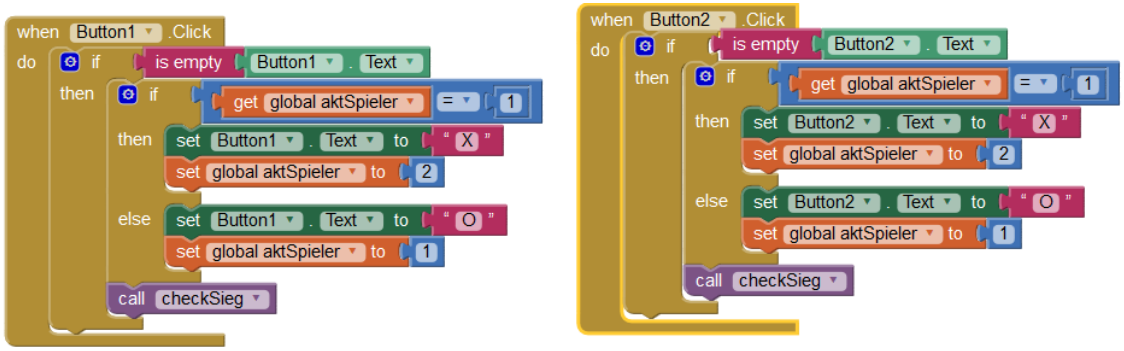


Abbildung 49: Lösung 1/4 von Tic Tac Toe

Abbildung 50: Lösung 2/4 von Tic Tac Toe

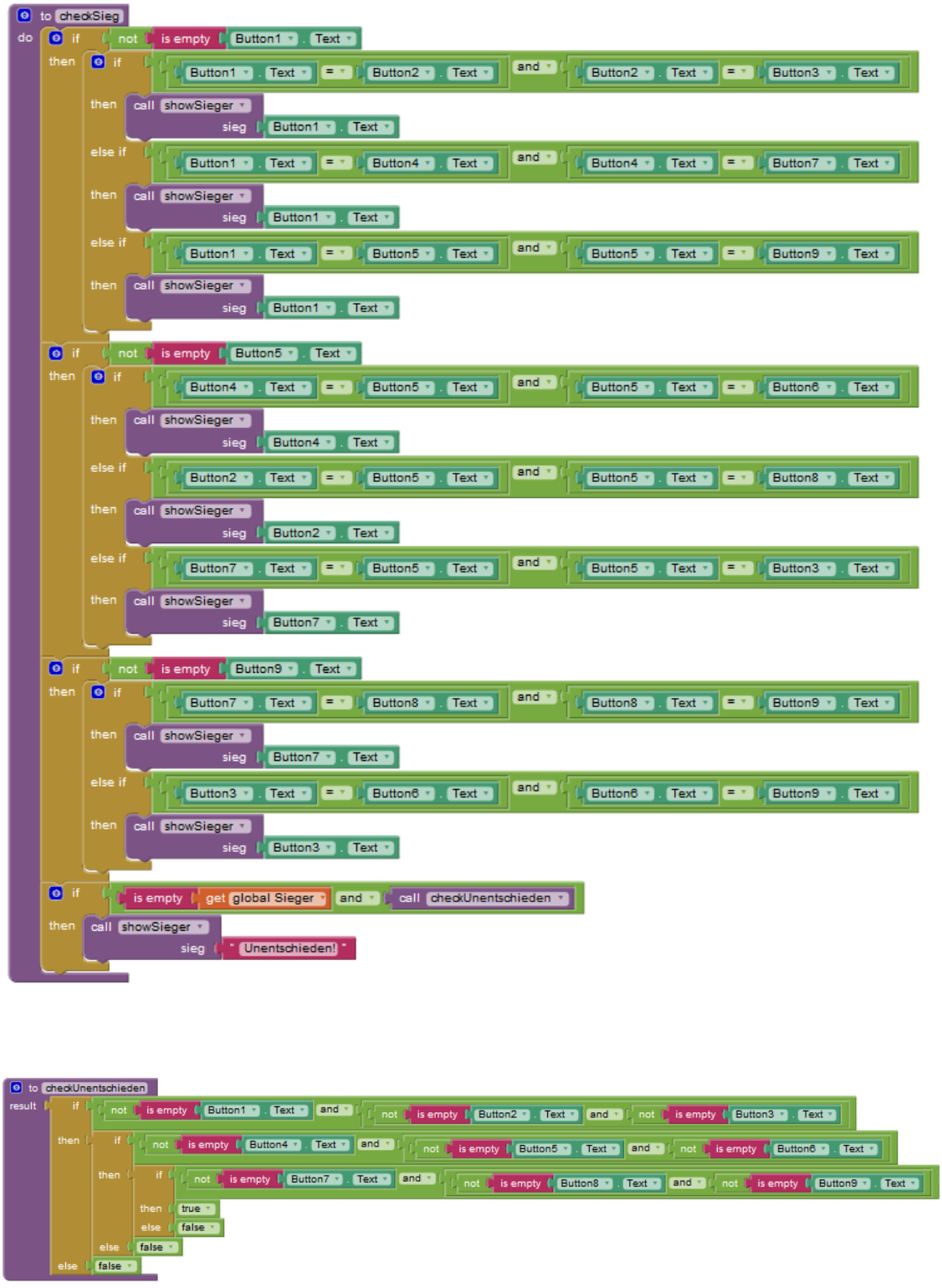


Abbildung 51: Lösung 3/4 von Tic Tac Toe

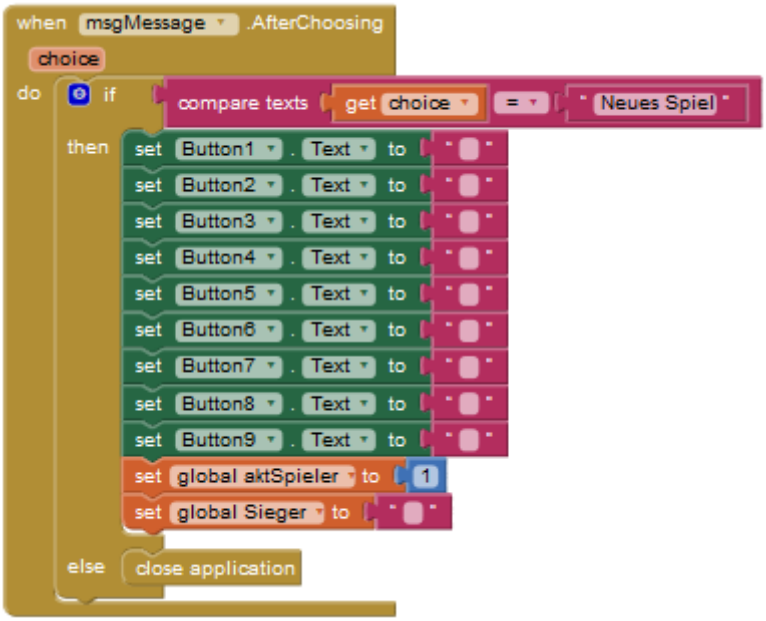


Abbildung 52: Lösung 4/4 von Tic Tac Toe

Einige Erweiterungen für leistungsstärkere SchülerInnen:

- Punktestand mitzählen und optional zurücksetzen
- Einstellbare, individuelle Spielernamen
- Einstellbare, individuelle Symbole
- Entwicklung einer KI, um auch gegen den Computer spielen zu können
- Beliebige Design-Verbesserungen
- Code-Optimierung: eventuell mehr Funktionen verwenden, etc.

Praktische Anwendung:

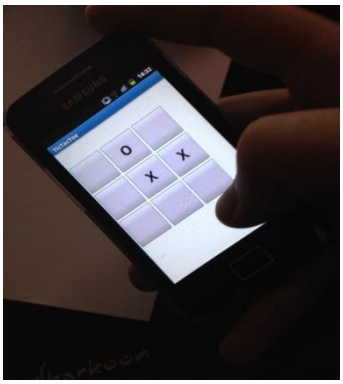


Abbildung 53: Praxis 1/3 von Tic Tac Toe

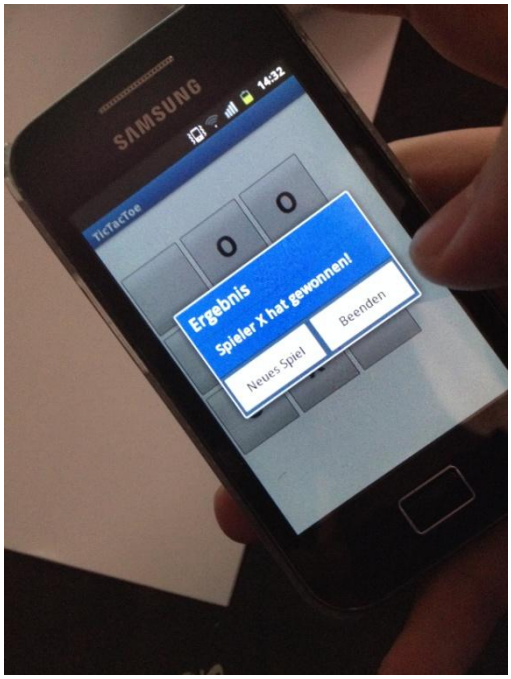


Abbildung 54: Praxis 2/3 von Tic Tac Toe

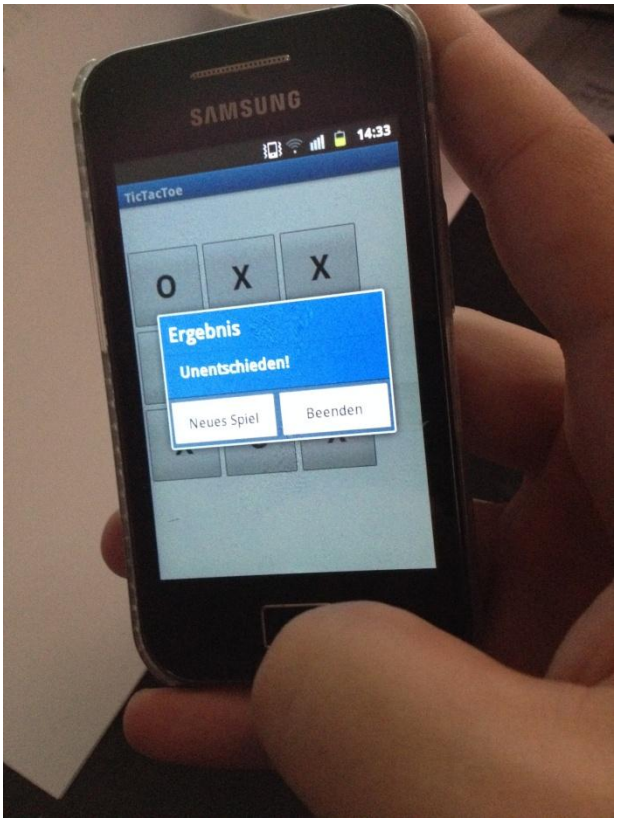


Abbildung 55: Praxis 3/3 von Tic Tac Toe

Kryptographie

Zielgruppe: SchülerInnen im Wahlpflichtgegenstand Informatik

Ziele - Die SchülerInnen sollen:

- Einfache Kryptographie-Verfahren kennen und verstehen lernen
- Listen implementieren und den Umgang mit diesen üben
- Einfache Schleifen anwenden und verstehen können
- Verschiedene String-Operationen anwenden und verstehen können

Aufgabenstellung:

Es soll eine App mit zwei Funktionen entwickelt werden: Zum einen soll es möglich sein, einen Klartext einzugeben und diesen verschlüsselt als Geheimtext auszugeben. Zum anderen soll es möglich sein, einen Geheimtext einzugeben und diesen entschlüsselt als Klartext auszugeben. Für das Kryptographie-Verfahren wird eine einfache Zeichenersetzung (monoalphabetische Substitution) mittels Listen verwendet. Dazu wird einem Zeichen aus dem Klartext immer genau das gleiche Zeichen aus dem Geheimtext zugewiesen und umgekehrt, so wird z.B. jedes A als R verschlüsselt, jedes B als D, usw.

Material: wird für diese Anwendung nicht benötigt

Mögliche Beispiellösung:

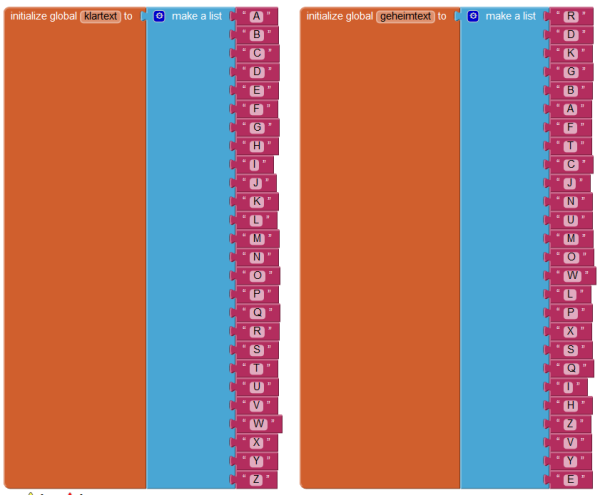


Abbildung 56: Lösung 1/2 von Kryptographie

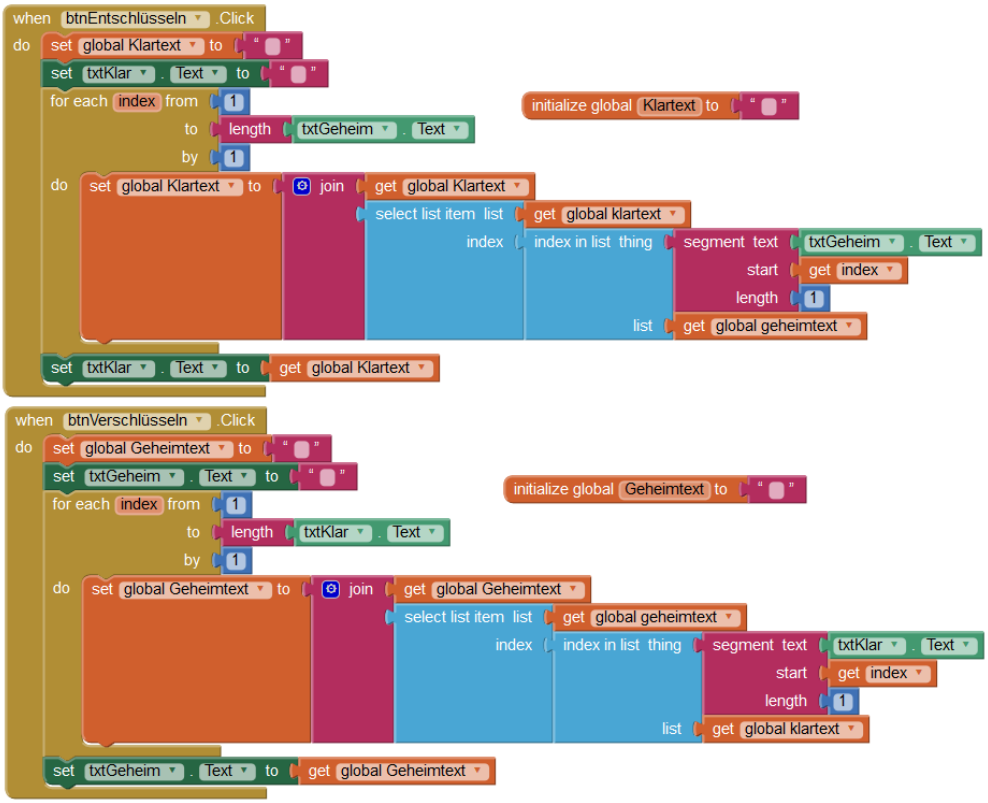


Abbildung 57: Lösung 2/2 von Kryptographie

<u>Einige Erweiterungen für leistungsstärkere SchülerInnen:</u>

- Erweiterung um Kleinbuchstaben und Zahlen
- Andere Form der Verschlüsselung implementieren z.B. Caesar

<u>Praktische Anwendung:</u>

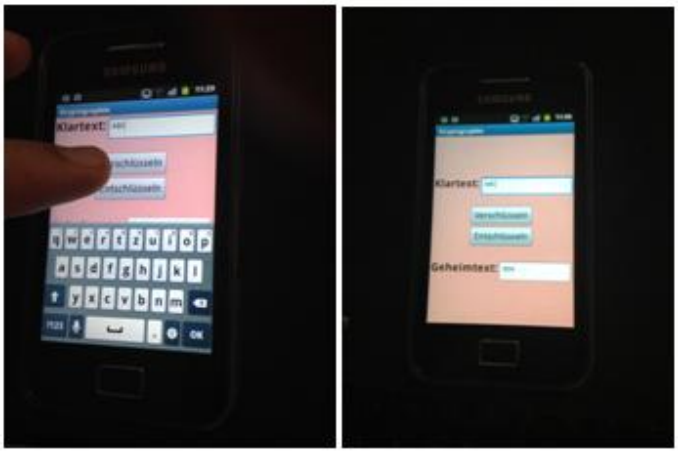


Abbildung 58: Praxis von Kryptographie

Selection Sort

Zielgruppe: SchülerInnen im Wahlpflichtgegenstand Informatik

Ziele - Die SchülerInnen sollen:

- Einfache Sortieralgorithmen kennenlernen und implementieren
- Verschachtelte Schleifen verstehen und implementieren

Aufgabenstellung:

Eine Liste mit zehn zufälligen Elementen zwischen 1 und 100 soll mittels Selection Sort aufsteigend sortiert und ausgegeben werden. Vor jeder Anwendung des Sortieralgorithmus soll die unsortierte Liste mit neuen zufälligen Werten initialisiert werden. Textuelle Beschreibung des anzuwendenden Algorithmus:

„Finde die kleinste Zahl aus den unsortierten Zahlen und vertausche diese mit der Zahl an der ersten Stelle der unsortierten Liste. Wiederhole diesen Vorgang für alle unsortierten Zahlen in der Liste.“

Material: wird für diese Anwendung nicht benötigt

Mögliche Beispiellösung:

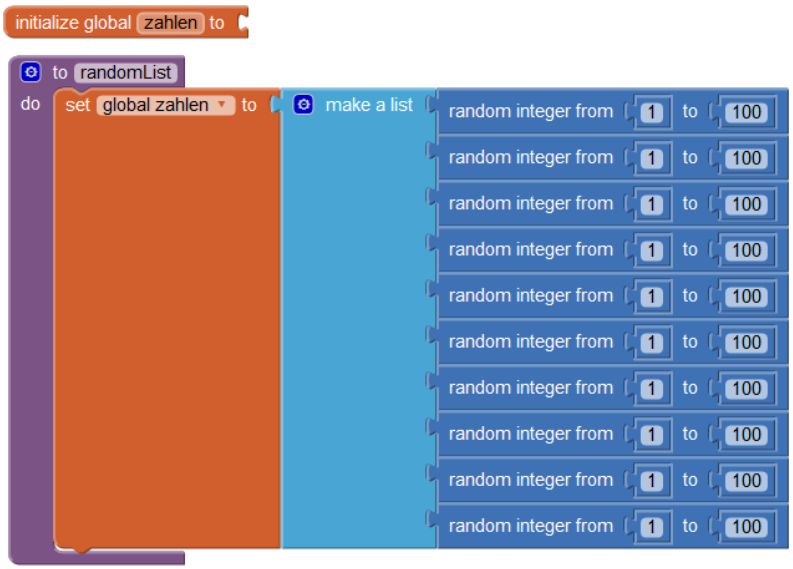


Abbildung 59: Lösung 1/2 von Selection Sort

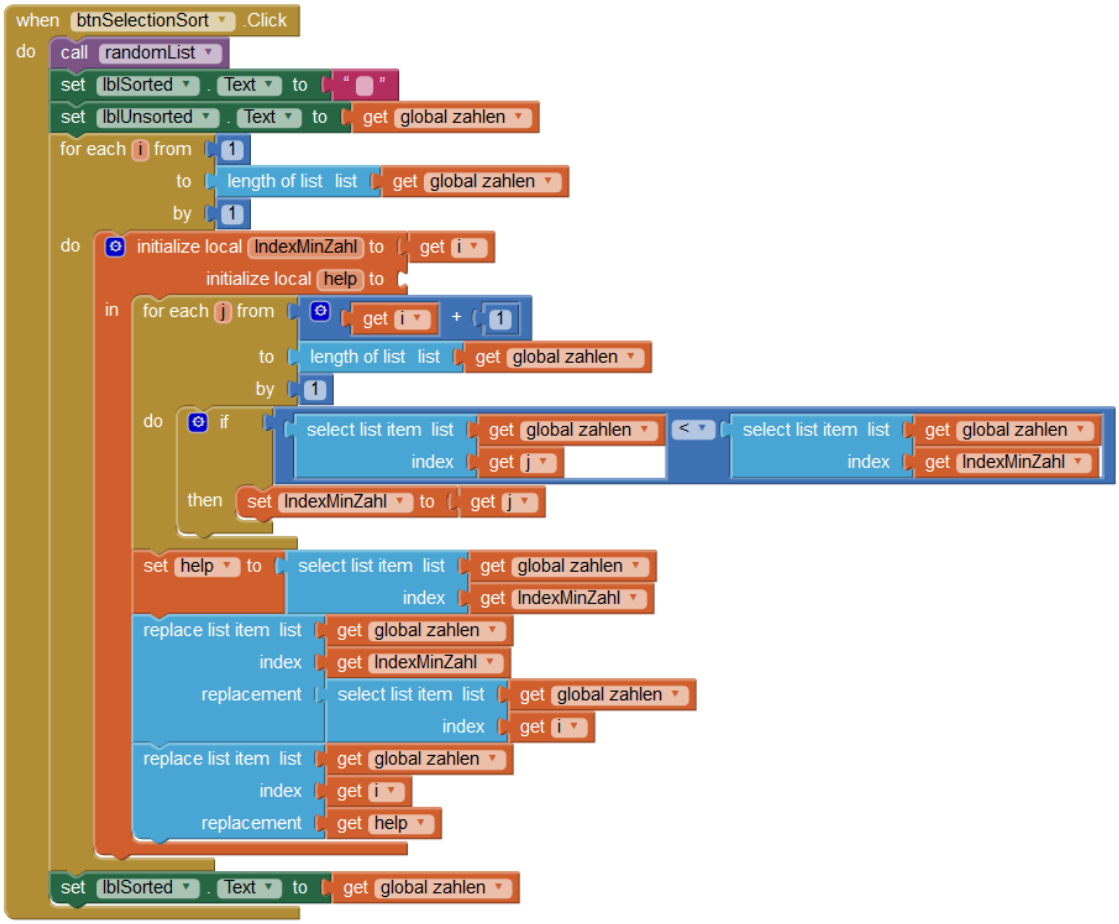


Abbildung 60: Lösung 2/2 von Selection Sort

Einige Erweiterungen für leistungsstärkere SchülerInnen:

- Andere Sortieralgorithmen wie z.B. Bubble Sort zusätzlich implementieren
- Aufwand unterschiedlicher Algorithmen festhalten z.B. in Form von Anzahl der Vertauschungen oder der benötigen Zeit
- Überlegungen für Best-Case-Szenarios und Worst-Case-Szenarios anstellen
- Buchstabensortierung nach Alphabet implementieren

Praktische Anwendung:

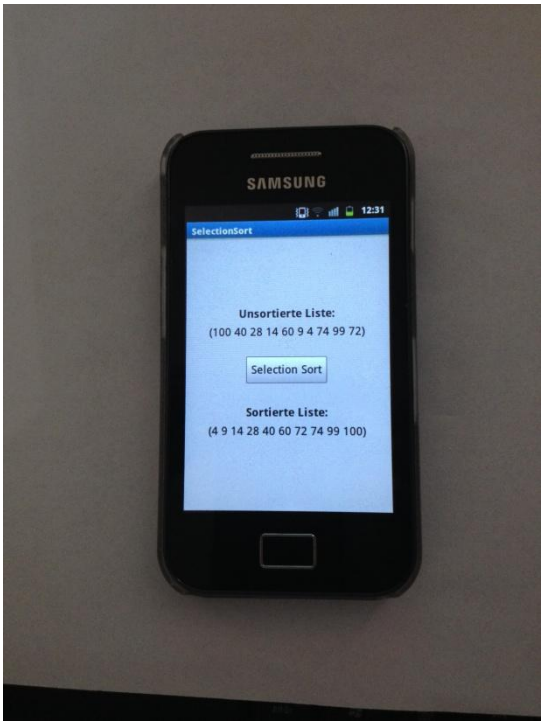


Abbildung 61: Praxis 1/2 von Selection Sort

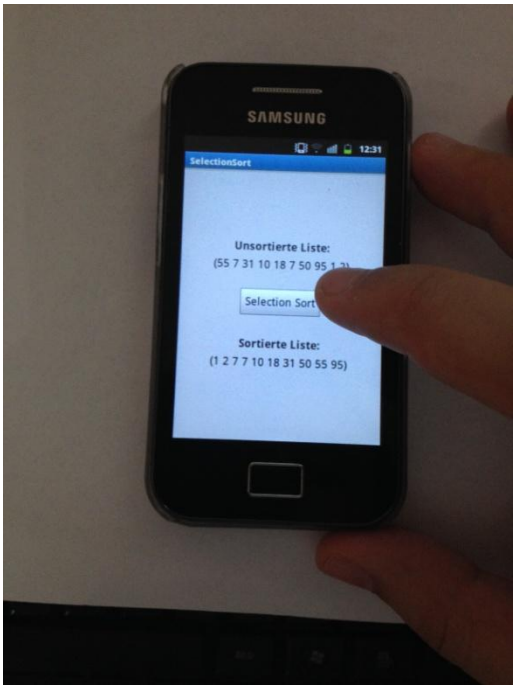


Abbildung 62: Praxis 2/2 von Selection Sort

Space Invaders

Zielgruppe: SchülerInnen im Wahlpflichtgegenstand Informatik

Ziele - Die SchülerInnen sollen:

- Animationen in einem Canvas-Control erstellen können
- Verschiedene Objekte über das Clock-Control zeitlich steuern können

Aufgabenstellung:

Die Ideen und einige Materialien stammen direkt vom Team des MIT App Inventors (http://explore.appinventor.mit.edu/ai2/space-invaders, letzter Aufruf: 12.06.2015) und wurden um eigene Aspekte ergänzt. Am oberen Bildschirmrand bewegt sich automatisch ein gegnerisches Ufo. Der Spieler steuert manuell eine Rakete am unteren Bildschirmrand und kann beim Klick auf die Rakete ein Schuss abfeuern. Für jedes getroffene Ufo erhält der Spieler Punkte. Zusätzlich werden beim Abschuss des Ufos und beim Feuern aus der Rakete Sound-Files abgespielt.

Material: [9]

Abbildung 63: Rakete

Abbildung 64: Ufo

Als Sound-Files wurden folgende Dateien von Mike Koenig verwendet: http://soundbible.com/1769-Laser-Gun.html & http://soundbible.com/538-Blast.html, letzter Aufruf: 12.06.2015

[9] Rakete und Ufo sind online unter http://explore.appinventor.mit.edu/ai2/space-invaders, letzter Aufruf: 12.06.2015

Mögliche Beispiellösung:

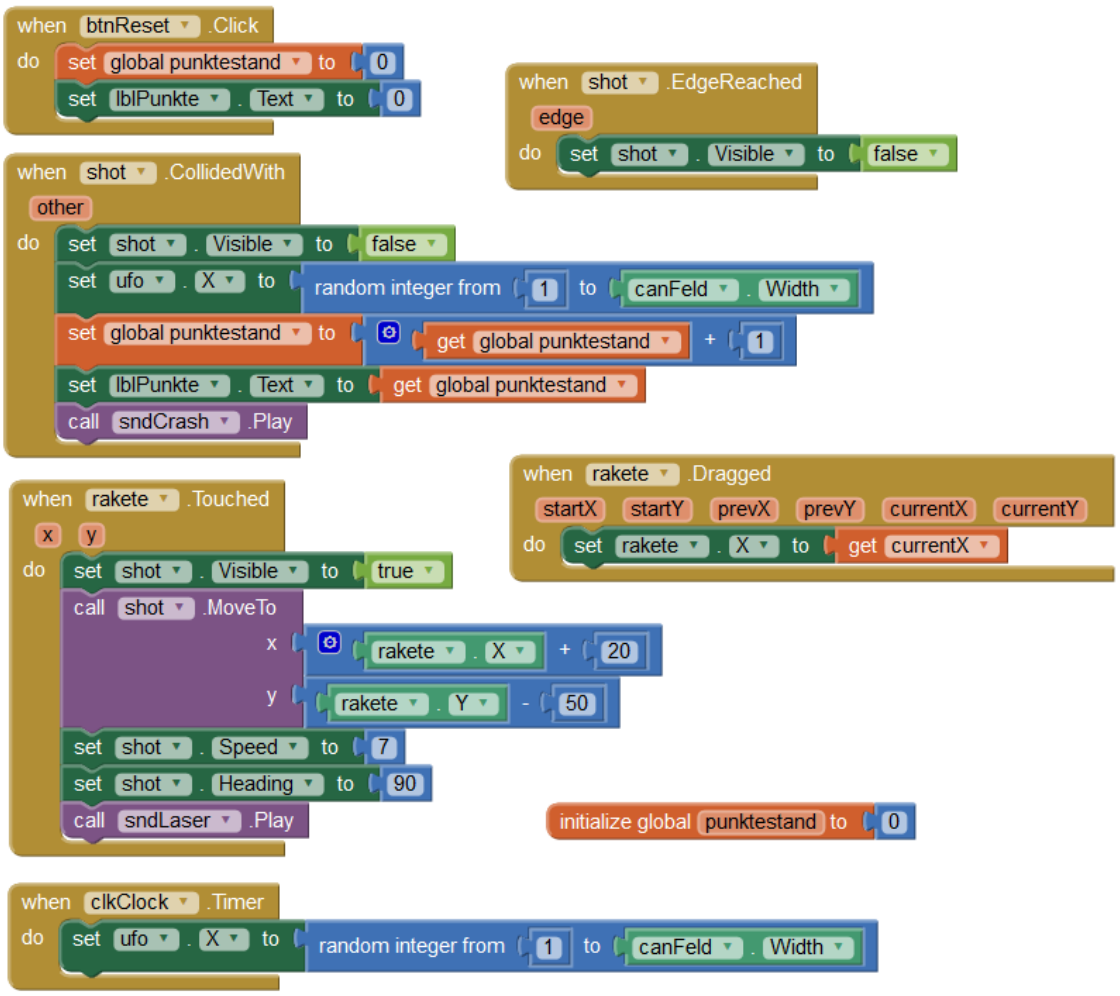


Abbildung 65: Lösung von Space Invaders

Einige Erweiterungen für leistungsstärkere SchülerInnen:

- Das Ufo soll in zufälligen Zeitabständen ebenfalls Schüsse abfeuern können. Wird die Rakete des Spielers getroffen, ist die Spielrunde vorbei.
- Erweiterung um die Auswahl von Schwierigkeitsgraden für das Spiel (leicht, mittel, schwer) – je nach Schwierigkeitsgrad haben die Objekte unterschiedliche Geschwindigkeiten
- Die gegnerischen Ufos sollen zufällig in unterschiedlichen Formen und Farben erscheinen (Erweiterung um eigene Bilder)

Praktische Anwendung:

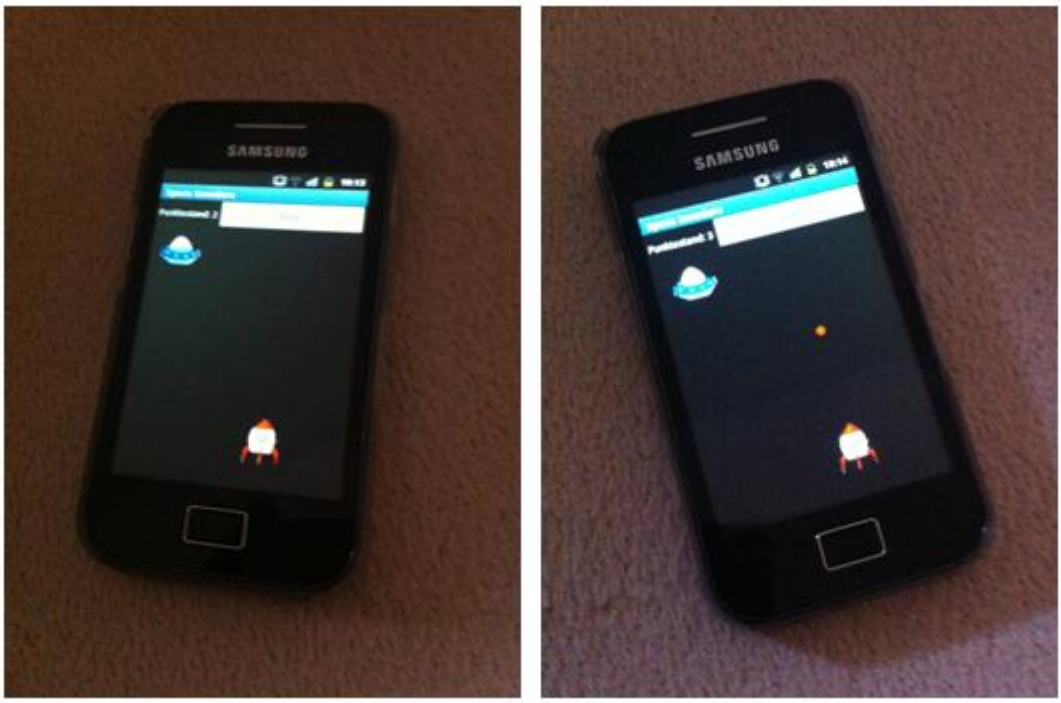


Abbildung 66: Praxis 1/2 von Space Invaders

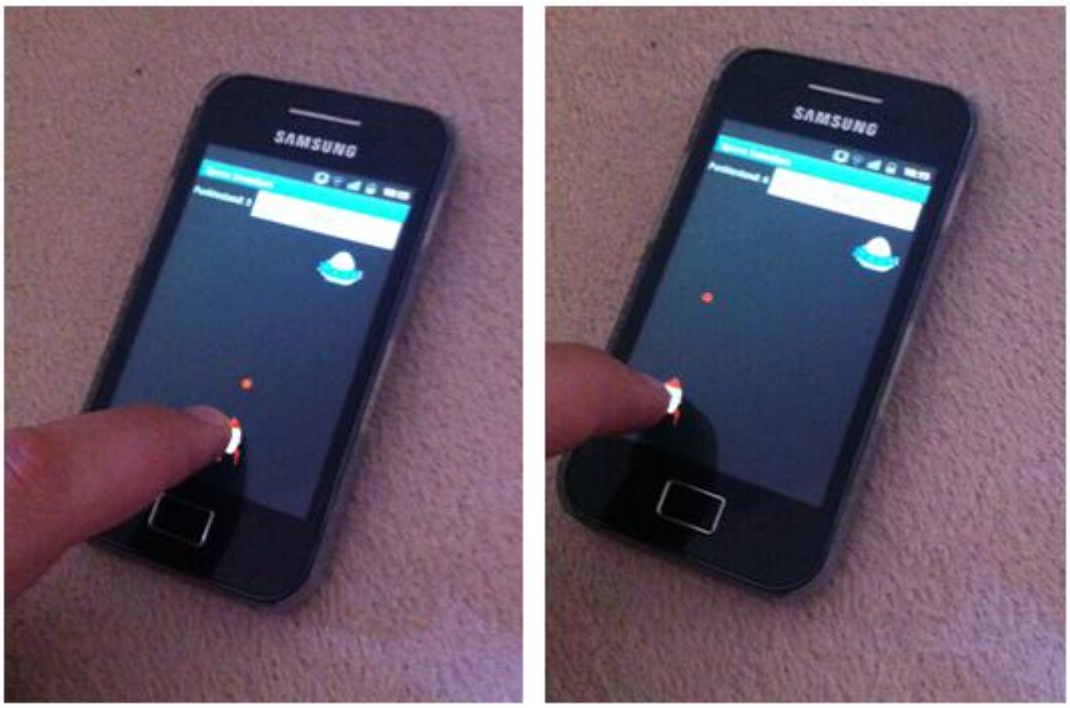


Abbildung 67: Praxis 2/2 von Space Invaders

Quiz

Zielgruppe: SchülerInnen im Wahlpflichtgegenstand Informatik

Ziele - Die SchülerInnen sollen:

- Eindimensionale Listen verstehen und anwenden können
- Mehrdimensionale Listen verstehen und anwenden können
- List-Picker kennenlernen und implementieren

Aufgabenstellung:

Die Anwendung besteht aus einem einfachen Quiz, bei dem eine von vier Antwortmöglichkeiten richtig ist. Beim Drücken auf einen Button soll eine zufällige Frage erscheinen. Zur Beantwortung der Frage soll aus vorgegebenen Antworten die richtige aus einer Liste (List-Picker) gewählt werden können. Nach der Auswahl einer Antwort soll eine Meldung erscheinen, ob die Frage richtig oder falsch beantwortet wurde.

Material: wird für die grundlegende Anwendung nicht benötigt

Mögliche Beispiellösung:

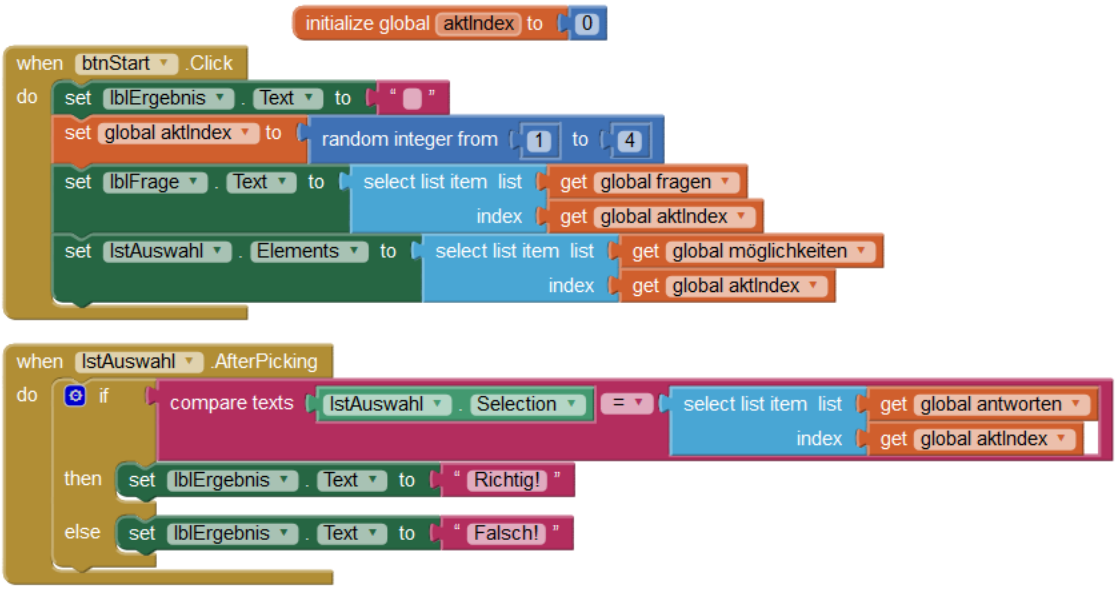


Abbildung 68: Lösung 1/2 von Quiz

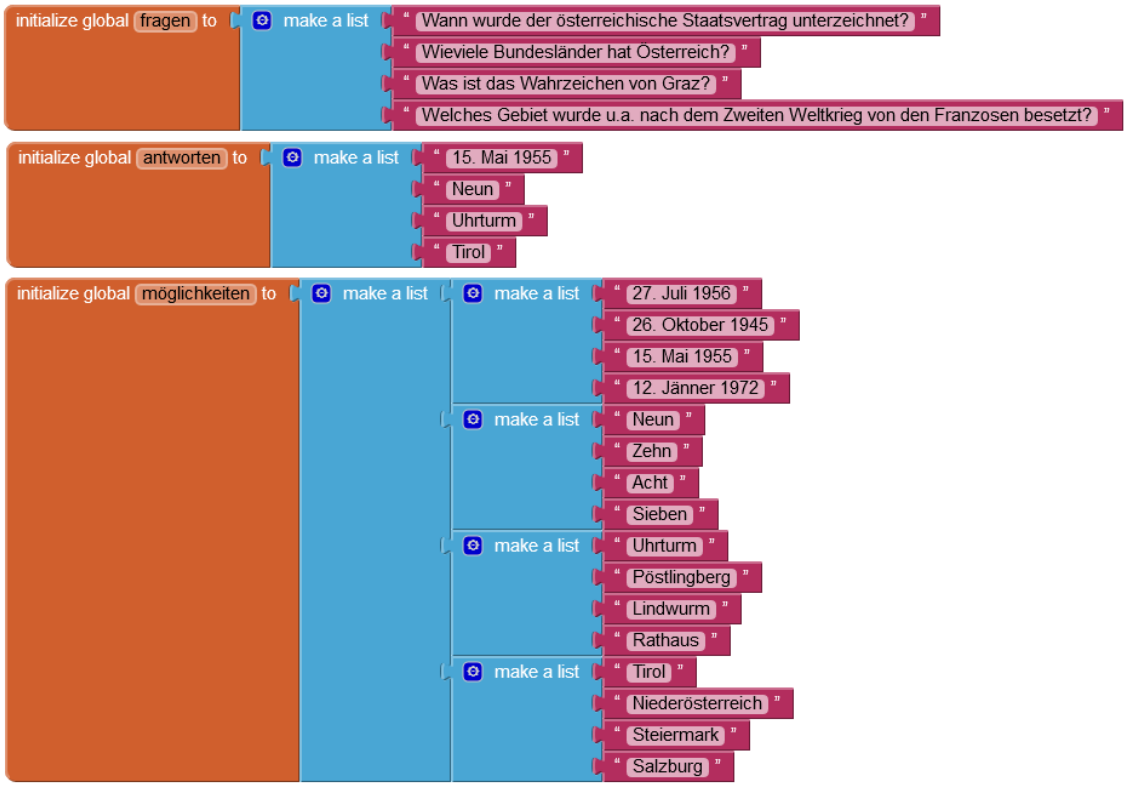


Abbildung 69: Lösung 2/2 von Quiz

Einige Erweiterungen für leistungsstärkere SchülerInnen:

- Anwendung um mehrere Fragen bzw. Antworten ergänzen
- Punktestand mitzählen
- Design-Verbesserungen vornehmen
- Sound-Files einbinden
- In der derzeitigen Version werden die Antwortmöglichkeiten auf eine Frage immer in der gleichen Reihenfolge in den List-Picker geladen. In einer erweiterten Version könnten die SchülerInnen eine zufällige Reihenfolge der Antwortmöglichkeiten implementieren

Praktische Anwendung:

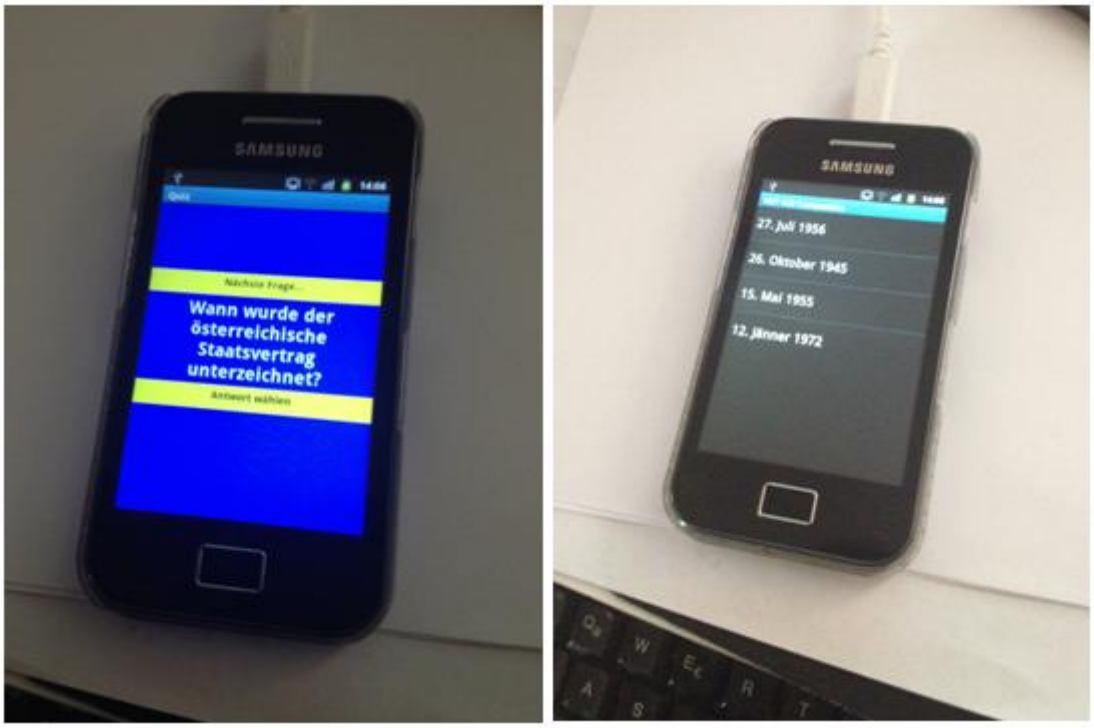


Abbildung 70: Praxis 1/2 von Quiz

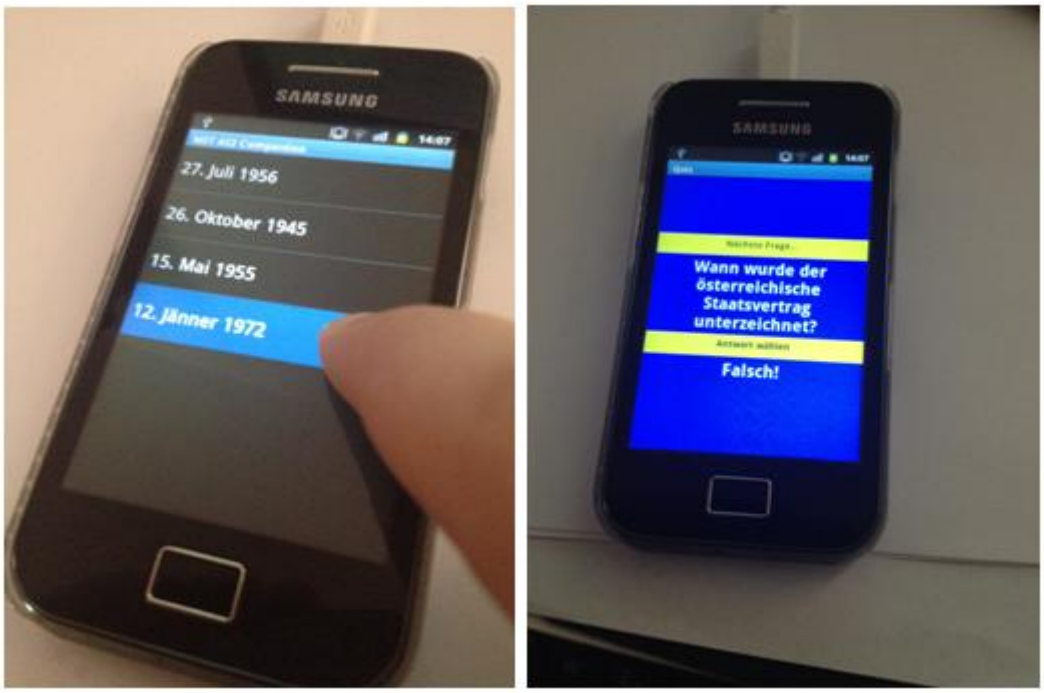


Abbildung 71: Praxis 2/2 von Quiz

Hangman

Zielgruppe: SchülerInnen im Wahlpflichtgegenstand Informatik

Ziele - Die SchülerInnen sollen:

- Verschiedenen Listen-Funktionen verstehen und anwenden können
- Verschiedene String-Funktionen verstehen und anwenden können
- Umgang mit Bedingungen und Schleifen festigen

Aufgabenstellung:

Die SchülerInnen entwickeln eine einfache Version des Hangman-Spiels. Dabei spielen zwei SpielerInnen auf einem mobilen Gerät gegeneinander. Der erste Spieler überlegt sich ein Wort und gibt dieses in eine Password-Textbox ein, wodurch der zweite Spieler das Wort nicht lesen kann. Der zweite Spieler hat beliebig viele Versuche, um das eingebende Wort zu erraten, darf aber immer nur einen Buchstaben verwenden. Danach können die SpielerInnen tauschen und Spieler 1 muss das Wort von Spieler 2 erraten, usw. Die Anzahl der Fehlversuche von einer Raterunde soll mitgezählt werden.

Material: wird für das Grundspiel nicht benötigt

Mögliche Beispiellösung:

Abbildung 72: Lösung 1/2 von Hangman

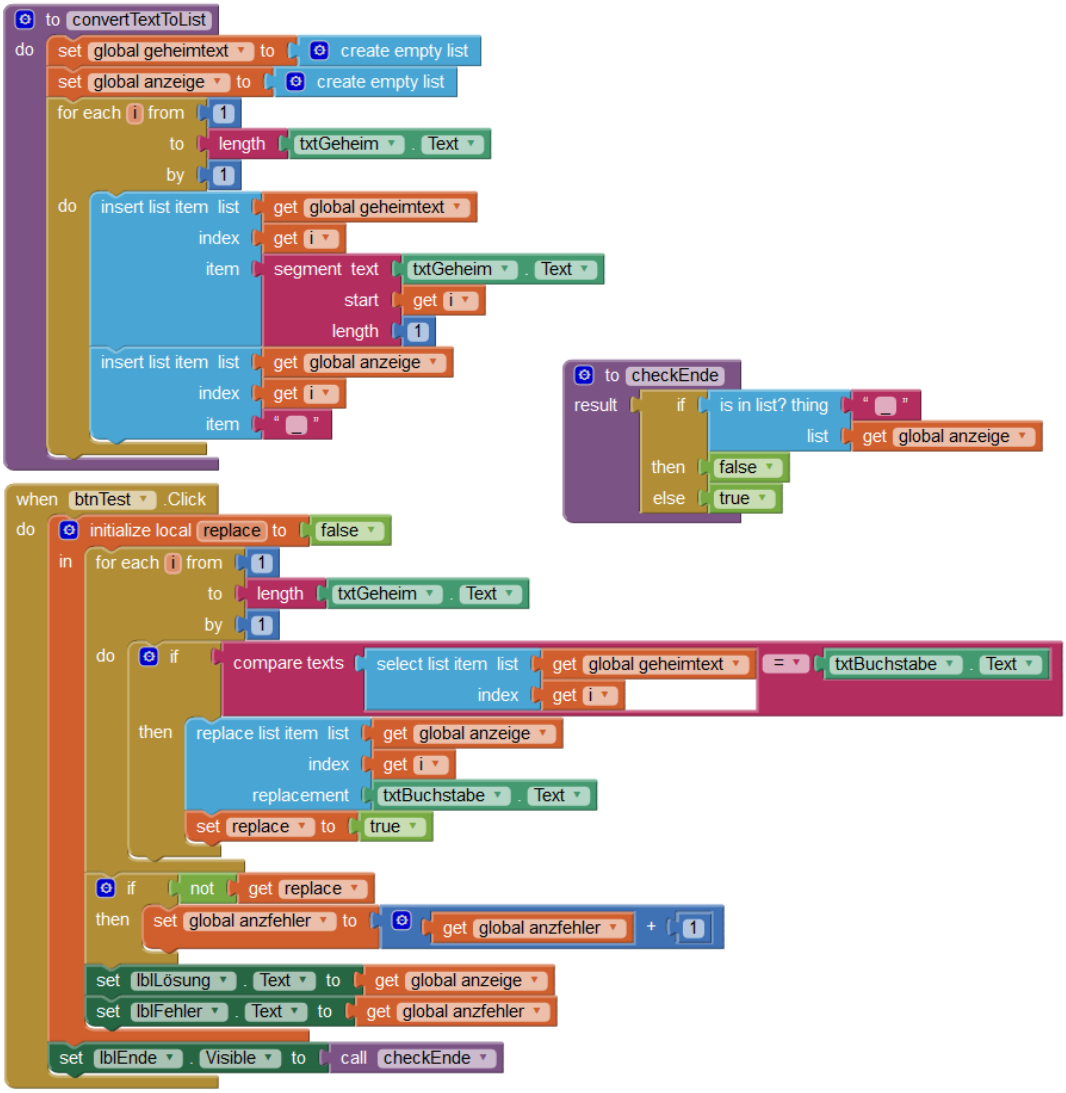


Abbildung 73: Lösung 2/2 von Hangman

<u>Einige Erweiterungen für leistungsstärkere SchülerInnen:</u>

- Begrenzte Anzahl an Fehlversuchen implementieren
- Ordentliche Eingabeüberprüfung überlegen und implementieren
- In der derzeitigen Version wird zwischen Groß- und Kleinbuchstaben unterschieden, diese Unterscheidung soll nicht mehr passieren
- Design-Verbesserungen z.B. statt der Anzeige der begrenzten Fehlversuchen für jeden Fehlversuch ein anderes Bild einfügen, um damit das Galgenmännchen zu simulieren

Praktische Anwendung:

Abbildung 74: Praxis 1/2 von Hangman

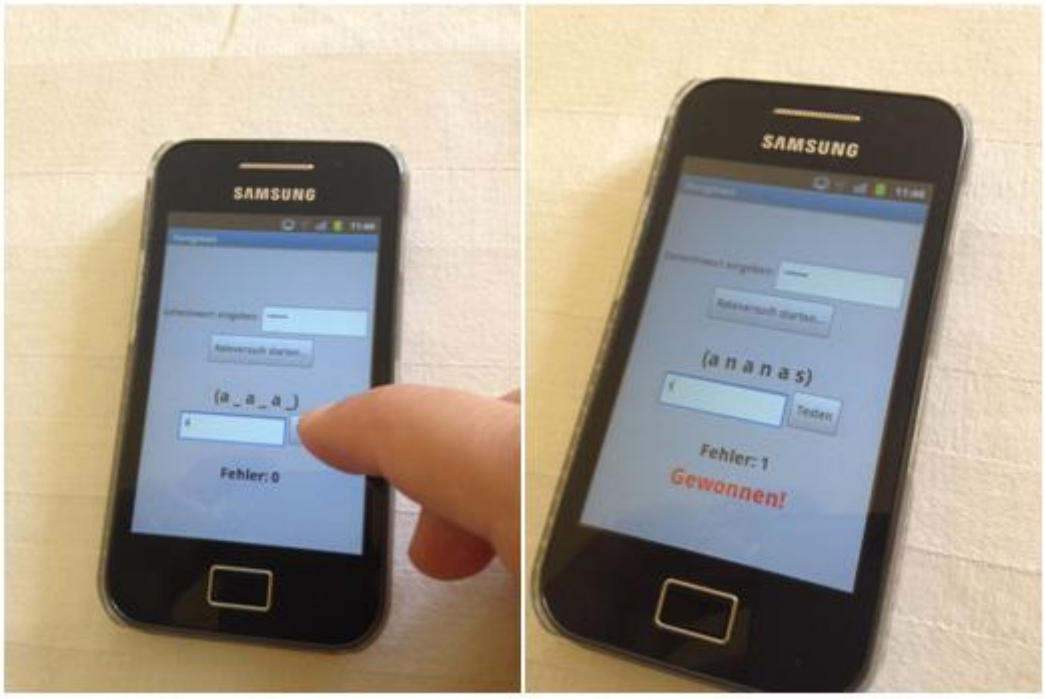


Abbildung 75: Praxis 2/2 von Hangman

Tonstudio

Zielgruppe: SchülerInnen im Wahlpflichtgegenstand Informatik

Ziele - Die SchülerInnen sollen:

- Eine multimediale App erstellen und die verschiedenen Objekte koordinieren können. Dazu müssen sie in dieser Anwendung folgende Controls verstehen und implementieren können: SoundRecorder, Camcorder, Player, VideoPlayer

Aufgabenstellung:

Die Anwendung soll die grundlegenden Multimedia-Funktionen des mobilen Geräts verwenden und diese miteinander verknüpfen. Dabei sollen über das Mikrofon beliebig lange Geräusche aufgenommen werden können und über eine andere Funktion wiedergegeben werden. Zusätzlich soll der Status, ob man sich gerade im Aufnahmemodus oder Wiedergabemodus befindet, ausgegeben werden. Als letzte Funktion soll die Möglichkeit bestehen, über die Handykamera ein Video aufzunehmen und dieses in einen Player zwischen zu speichern. Von dort aus kann das Video jederzeit aufgerufen werden, bis es durch ein anders überschrieben wird.

Material: [10]

Abbildung 76: Sound-Wave

[10] Sound-Wave von Luis Lima89989 ist online unter
https://en.wikipedia.org/wiki/Acoustic_network#/media/File:Sound_wave.jpg
letzter Aufruf: 16.06.2015 – wurde als Hintergrundbild für die Anwendung verwendet

Mögliche Beispiellösung:

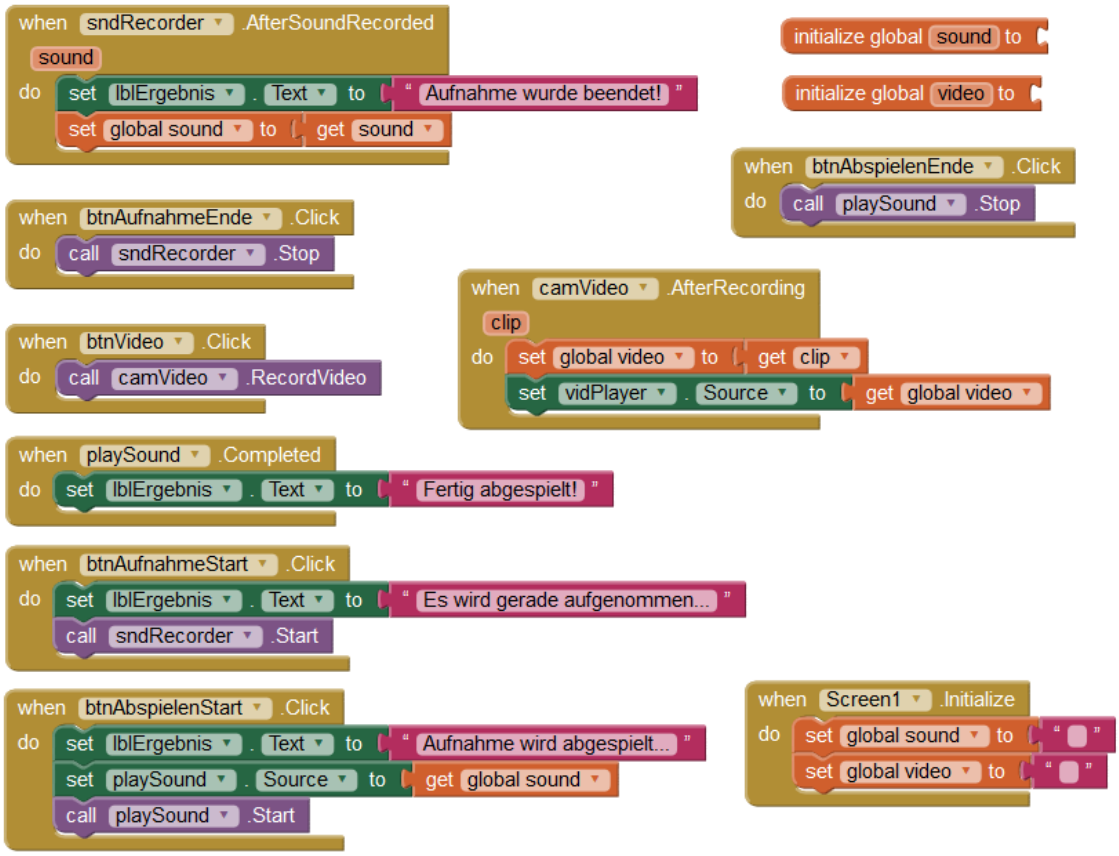


Abbildung 77: Lösung von Tonstudio

Einige Erweiterungen für leistungsstärkere SchülerInnen:

- Lautstärke-Regelung mittels Slider implementieren
- Ergänzung der Anwendung um weitere Multimedia-Elemente
- Speicherung mehrere Aufnahmen durch das Datenbanksystem der Entwicklungsumgebung (TinyDB)

Praktische Anwendung:

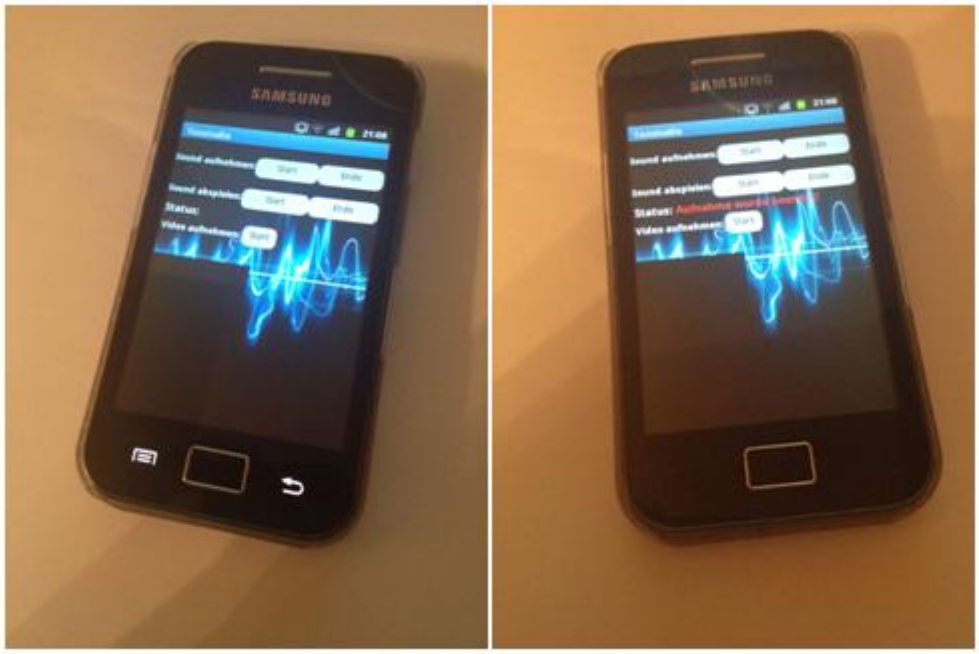


Abbildung 78: Praxis 1/2 von Tonstudio

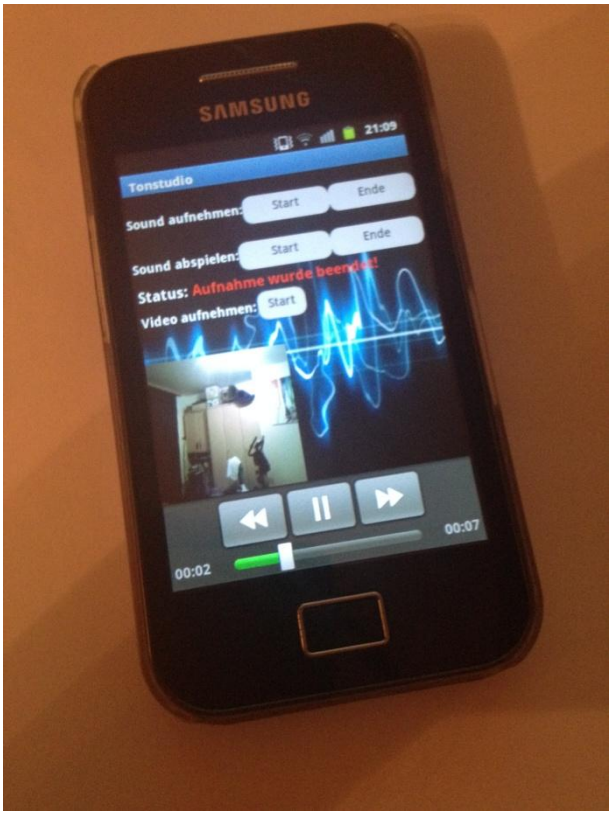


Abbildung 79: Praxis 2/2 von Tonstudio

Mein GPS

Zielgruppe: SchülerInnen im Wahlpflichtgegenstand Informatik

Ziele - Die SchülerInnen sollen:

- Die Funktionsweise von GPS verstehen
- GPS-Lokalisierung an mobilen Geräten implementieren können
- Das ActivityStarter-Control kennenlernen und verstehen

Aufgabenstellung:

Die Anwendung soll die aktuelle GPS-Position des mobilen Gerätes in Form der geographischen Breite und der geographischen Länge anzeigen. Zusätzlich zu der Ausgabe in Koordinaten-Form, soll auch die Adresse des mobilen Geräts angezeigt werden. Der Benutzer hat danach die Möglichkeit, selbst eine geographische Breite und eine geographische Länge in zwei Textboxen einzugeben. Beim Klick auf „Karte anzeigen“ öffnet sich der Routenplaner von Google Maps und zeigt die Route zwischen der aktuellen Position des mobilen Geräts, sowie der vom Benutzer manuell festgelegten Position.

Material: wird für die grundlegende Anwendung nicht benötigt

Mögliche Beispiellösung:

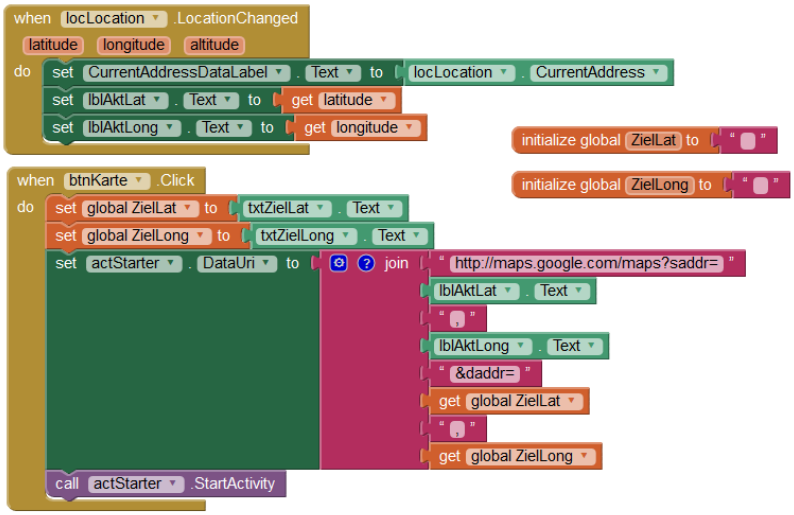


Abbildung 80: Lösung von Mein GPS

Anmerkung: Obwohl die grundlegenden Funktionen in der Aufgabenstellung mit relativ wenig Programmierung verbunden sind, würde ich dieses Beispiel trotzdem nur im Wahlpflichtgegenstand Informatik durchnehmen. Dies liegt vor allem daran, dass das Verständnis für Geoinformatik-Anwendungen mit einer entsprechenden Vorbereitung und Sorgfalt aufzubauen ist. Man kann somit Einheiten rund um das Thema Lokalisierung und GPS gestalten und als Abschluss die SchülerInnen selbst diese App entwickeln lassen, mit welcher sie den Grundstein für ein einfaches Navigation-System oder für eine Anwendung zur Freizeitbeschäftigung, wie z.B. Geocaching schaffen.

Einige Erweiterungen für leistungsstärkere SchülerInnen:

- Adresse des Ziels soll ebenfalls angezeigt werden
- Es soll möglich sein, mehrere verschiedene Standorte abzuspeichern
- Eingabeüberprüfung implementieren
- Man soll eine Rückmeldung erhalten, wenn man das Ziel erreicht hat z.B. in Form einer selbst aufgenommenen Sound-Wiedergabe

Praktische Anwendung:

Abbildung 81: Praxis von Mein GPS

Pong

Zielgruppe: SchülerInnen im Wahlpflichtgegenstand Informatik

Ziele - Die SchülerInnen sollen:

- Mathematische Berechnungen verstehen und sinnvoll einsetzen
- Auf Wissen aus der Physik zurückgreifen und dieses anwenden
- Verschiedene Objekte in einer Anwendung koordinieren können

Aufgabenstellung:

Die SchülerInnen dürfen eine einfache Version des Spieles „Pong" auf Basis eines Online-Tutorials von der Seite http://www.appinventor.org/content/CourseInABox/drawAnimate/PongTutorial (letzter Aufruf: 17.06.2015) mit eigenen Ergänzungen programmieren. Dabei steuert der Spieler einen Balken in der Nähe des unteren Bildschirmrandes und muss versuchen, mit diesem einen Spielball solange wie möglich im Spielfeld zu bleiben. Der Ball prallt von den Seiten, dem oberen Bildschirmrand und dem Balken des Spielers im richtigen Winkel ab (Einfallswinkel = Ausfallswinkel). Berührt der Ball allerdings den unteren Bildschirmrand, so ist das Spiel vorbei. Bei jeder Berührung des Balls mit dem Balken bekommt der Spieler einen Punkt. Beim Spielstart soll der Spielball an einer zufälligen Position in einem zufälligen Winkel starten, der Balken des Spielers soll möglichst mittig platziert werden.

Material: [11]

Abbildung 82: Balken

[11] es wird nur der Balken für den Spieler benötigt. Dabei kann auf diese erstellte Vorlage zurückgegriffen werden oder die SchülerInnen erstellen den Balken rasch selbst im Unterricht

Mögliche Beispiellösung:

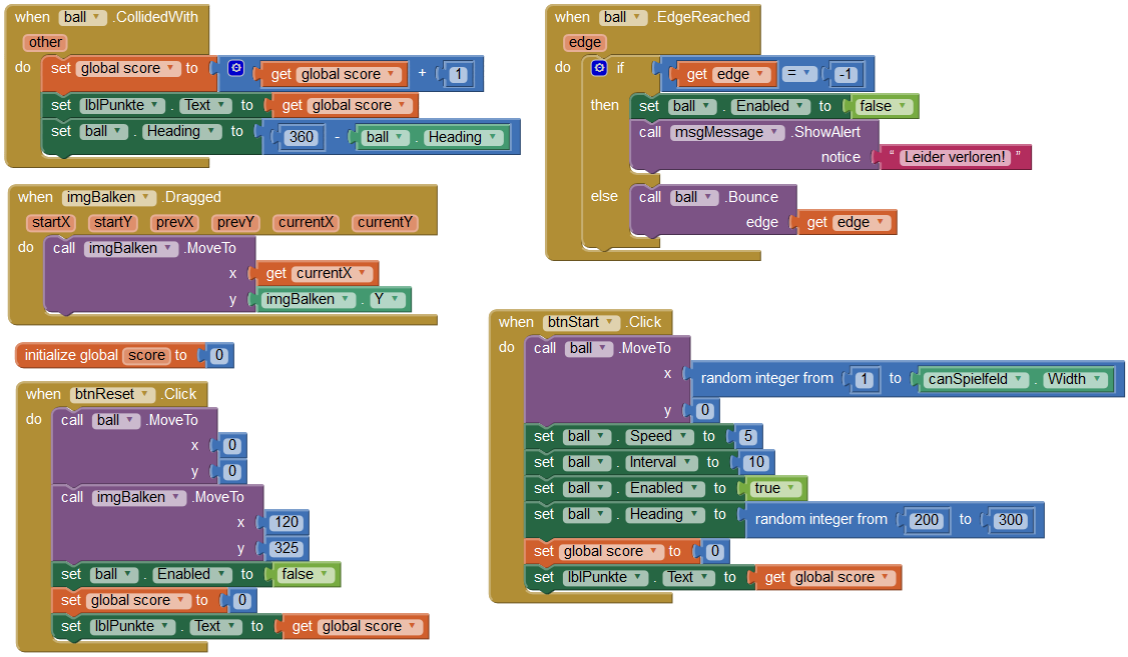


Abbildung 83: Lösung von Pong

Einige Erweiterungen für leistungsstärkere SchülerInnen:

- Grafische Design-Verbesserungen, eventuell eigene Grafiken einbinden
- Sound-Files für die verschiedenen Aktionen verwenden, wie z.B. wenn der Spieler einen Punkt bekommt, das Spiel verloren wurde oder der Ball abprallt
- Bei einer bestimmten Punkteanzahl soll der Ball immer schneller werden
- Speicherung der besten Spielstände in Form einer Highscore-Tabelle durch das Datenbanksystem der Entwicklungsumgebung (TinyDB)
- Schwierigkeitsgrad soll manuell einstellbar sein, wodurch die Balkengröße, die Ballgröße und die Ballgeschwindigkeit beeinflusst wird

Praktische Anwendung:

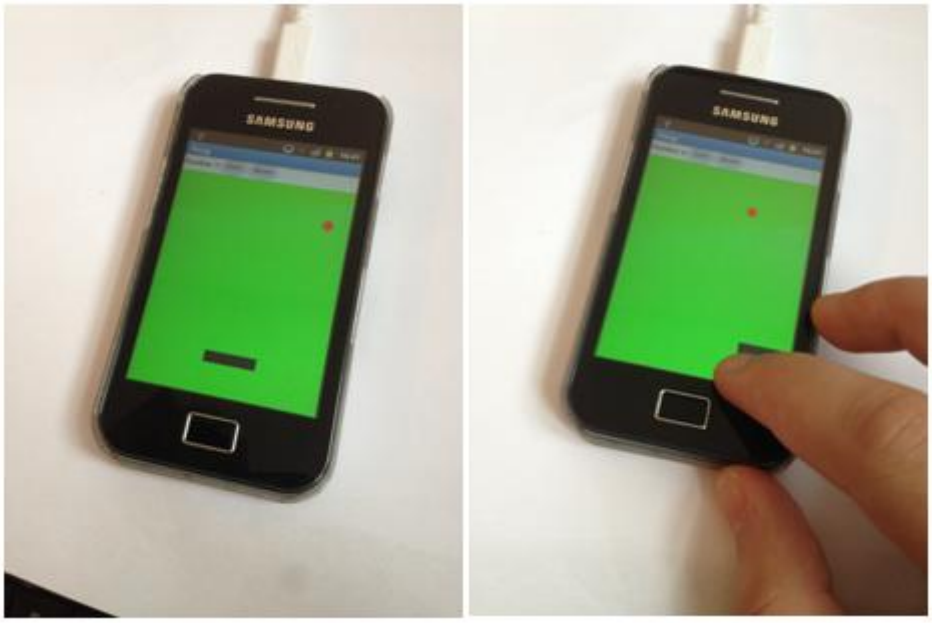


Abbildung 84: Praxis 1/2 von Pong

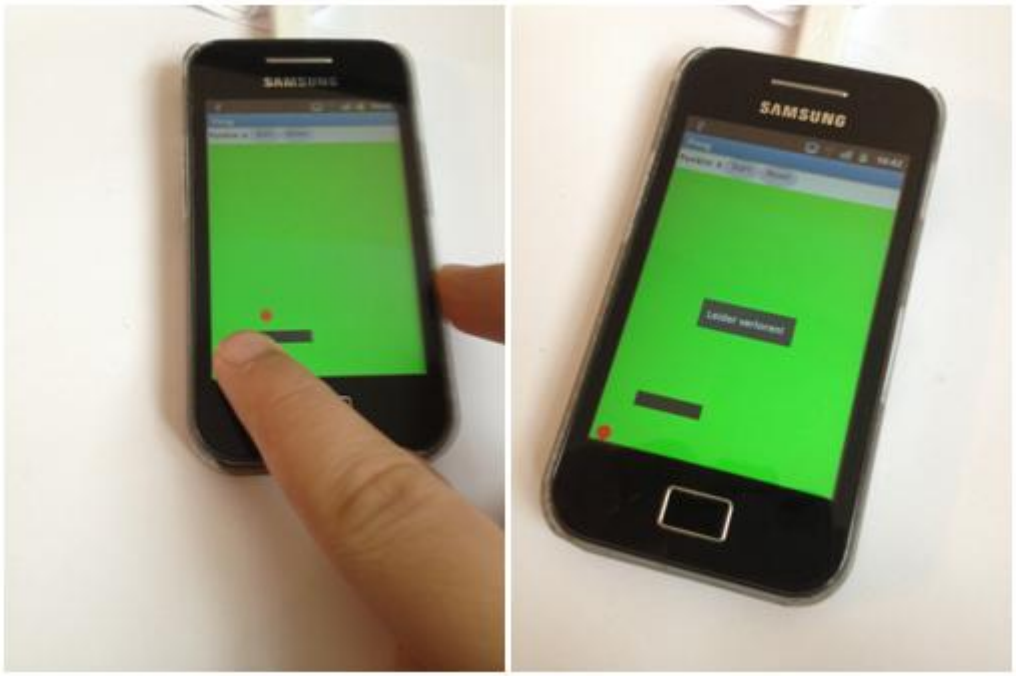


Abbildung 85: Praxis 2/2 von Pong

Rückschau und Ausblick

Es wurde eine Vielzahl an Gründen und Schwierigkeiten von ProgrammieranfängerInnen angeführt und erläutert. Vor allem die Unterrichtsmethoden der Lehrperson, die Lernmethoden und Fähigkeiten der SchülerInnen, das komplexe Wesen der Programmierung, sowie andere psychologische Effekte können zu Problemen führen. Visuelle Programmiersprachen können dabei als Chance gesehen werden, einige der Schwierigkeiten wie z.B. das Erlernen der Syntax einer Programmiersprache, zu vermindern und sind durchaus eine Alternative, wenn die Vermittlung der grundlegenden Konzepte sowie die Förderung des algorithmischen Denkens der SchülerInnen im Fokus liegen soll. Eine Orientierung an der Interessenslage der Jugendlichen kann zu einer zusätzlichen Motivation und dadurch zu besseren Leistungen im Unterricht führen. Der Unterricht mit mobilen Geräten geht dabei als besonders attraktive Option hervor, da diese technische Innovation einen besonderen Stellenwert in der heutigen Gesellschaft und vor allem auch bei Kindern und Jugendlichen einnimmt.

Es hat sich herausgestellt, dass mit der kostenlosen Entwicklungsumgebung „App Inventor“ des MIT relativ einfach und rasch Anwendungen für mobile Geräte entwickelt werden können. Viele komplexe Funktionen sind bereits in der Entwicklungsumgebung vorhanden und benötigen keine eigene Umsetzung. Dadurch kann ein effizientes Arbeiten während des Unterrichts stattfinden, indem die grundlegenden Kontrollstrukturen und deren Verständnis im Vordergrund stehen können. Auch die Gestaltungsmöglichkeiten der Anwendungen befinden sich in einem Rahmen, der weder zu komplex, noch zu simpel scheint, da die Bedienung intuitiv erfolgt und nach einem genaueren Blick kaum Design-Elemente fehlen. Abgesehen von einigen speziellen Konzepten, wie z.B. Objektorientierung, kann die Entwicklung mittels App Inventor als Ersatz für eine andere visuelle Programmiersprache wie SNAP! (BYOB) oder als Ersatz für eine textbasierte Programmiersprache wie C++ fungieren.

Im Bezug auf die im Beginn der Arbeit erwähnten und im Informatikunterricht zu vermittelnden Kompetenzen von der Webseite www.digikomp.at wurde allerdings deutlich, dass auch der Einsatz des MIT App Inventor nicht ohne Weiteres alle Kompetenzen fördern und daher auch nicht als „Heiliger Gral der Entwicklungsumgebungen und Programmiersprachen“ gesehen werden kann. So ist z.B. fraglich, ob die SchülerInnen nach etwas Programmiererfahrung und ohne Debugger tatsächlich in der Lage sind, gezielt nach Programmierfehlern zu suchen und diese zu korrigieren (vgl. Andraschko, Kompetenzmodell Wahlpflichtfach Informatik) oder, ob die SchülerInnen tatsächlich die Angemessenheit von Entwicklungswerkzeugen einschätzen können (vgl. Andraschko, Kompetenzmodell Wahlpflichtfach Informatik), wenn sie womöglich nur dieses eine Tool kennenlernen? Meiner Meinung nach können die SchülerInnen in der Oberstufe viele der zu vermittelnden Kompetenzen nur dadurch erwerben, indem sie verschiedene Blickwinkel und Perspektiven im Informatikunterricht vermittelt bekommen und in der Schule ein vielfältiges Softwareangebot kennenlernen.

Bei der Entwicklung der eigenen Beispiele für den Unterricht war ich allerdings erstaunt, wie einfach sich Elemente wie GPS, Kamera, Sound & Co. einbinden lassen. Auch im Vergleich mit „höheren Programmiersprachen“ stellen Herausforderungen, wie z.B. die zeitliche Koordination und Abstimmung von verschiedenen Objekten während der Laufzeit für fortgeschrittene EntwicklerInnen im App Inventor kaum Schwierigkeiten dar. Diese Umstände führen unter anderem dazu, dass die ProgrammierInnen schneller zu möglichst spannenden, greifbaren Ergebnissen kommen – keine unwesentliche Eigenschaft im Unterricht mit Jugendlichen, welche mitunter nicht immer einfach motivierbar sind und ihre Schwerpunkte in unterschiedlichen Interessensbereichen haben.

Mobile Geräte haben teilweise bereits den Einzug in viele Klassenzimmer geschafft und es scheint, als ob dieser Trend aufgrund unserer heutigen, zunehmend technologisierten Gesellschaft weiterverfolgt wird. Als angehender Informatiklehrer bin ich der Auffassung, dass ein innovativer Informatikunterricht diesen Aspekt berücksichtigen und die SchülerInnen auf eine einfache Art und Weise an die neue Technologien und deren Programmierung heranführen soll.

Teil IV.
Anhang

Quellen- und Literaturverzeichnis

Achten Oliver M. & Pohlmann Norbert, Sichere Apps. Vision oder Realität? In: Datenschutz und Datensicherheit – DuD, Vol. 36, Nr. 3/2012, S. 161-164

Andraschko Monika, digi.komp12 – Das Kompetenzmodell Informatik 5. Klasse. online unter http://www.edugroup.at/praxis/portale/digitale-kompetenzen/digikomp12ahs/kompetenzmodelle/informatik-5-klasse.html, letzter Aufruf: 16.03.2015

Andraschko Monika, digi.komp12 – Das Kompetenzmodell Wahlpflichtfach Informatik. online unter http://digikomp.at/praxis/portale/digitale-kompetenzen/digikomp12ahs/kompetenzmodelle/wahlpflichtfach-informatik.html, letzter Aufruf: 16.03.2015

Balzert Helmut, Lehrbuch Grundlagen der Informatik. Konzepte und Notationen in UML 2, Java 5, C++ und C#. Algorithmik und Software-Technik Anwendungen. München: Elsevier Spektrum Akademischer Verlag, 2005

Baumann Rüdeger, Computernetze und Telekommunikation. Herausforderung für die Informatik-Didaktik. In: Anton Reiter (Hrsg.) Was ist Informatik-Didaktik in der Informationsgesellschaft. Tagungsunterlage zur Seminarveranstaltung in der Österreichischen Computer Gesellschaft am 3. Mai 1999. Wien: Bundesministerium für Bildung, Wissenschaft und Kultur (BMBWK), 2000, S. 19-32

Bhagi Anshul, Android Game Development with AppInventor. Cambridge: Massachusetts Institute of Technology, 2012. online unter http://appinventor.mit.edu/explore/sites/all/files/Resources/Thesis_FINAL_AnshulBhagi.pdf, letzter Aufruf: 27.05.2015

Braun Andreas, Spieledidaktik – Einstieg in die Programmierung mit Hilfe von Spielen und Game-Design, Diplomarbeit. Wien: Universität, 2013

Bruderer Herbert, Plädoyer für den Programmierunterricht. Programmieren fördert die Problemlösefähigkeit. Zürich: Eidgenössische Technische Hochschule, 2009. online unter http://e-collection.library.ethz.ch/eserv/eth:433/eth-433-01.pdf?pid=eth:433&dsID=eth-433-01.pdf, letzter Aufruf: 16.03.2015

Bundesministerium für Bildung und Frauen (BMBF), Lehrpläne der AHS-Oberstufe. Lehrpläne für die Pflichtgegenstände. Informatik. online unter https://www.bmbf.gv.at/schulen/unterricht/lp/lp_neu_ahs_14_11866.pdf?4dzgm2, letzter Aufruf: 16.03.2015

Bundesministerium für Bildung und Frauen (BMBF), Lehrpläne der AHS-Oberstufe. Lehrpläne für die Wahlpflichtgegenstände. Informatik. online unter https://www.bmbf.gv.at/schulen/unterricht/lp/lp_neu_ahs_21_11876.pdf?4dzgm2, letzter Aufruf: 16.03.2015

Bundesministerium für Bildung und Frauen (BMBF), Lehrpläne der AHS-Unterstufe. Freigegenstände. online unter https://www.bmbf.gv.at/schulen/unterricht/lp/ahs20_795.pdf?4dzgm2, letzter Aufruf: 16.03.2015

Bundesministerium für Bildung und Frauen (BMBF), Lehrpläne der AHS-Oberstufe. Allgemeiner Teil. online unter https://www.bmbf.gv.at/schulen/unterricht/lp/11668_11668.pdf?4dzgm2, letzter Aufruf: 16.03.2015

Fling Brian, Mobile Design and Development. Practical concepts and techniques for creating mobile sites and web apps. Sebastopol: O'Reilly, 2009

Futschek Gerald, Programmieren lernen mit BYOB. Vortrag auf der Tagung Informatiktag 2012 der Österreichischen Computer Gesellschaft (OCG). online unter http://www.ocg.at/sites/ocg.at/files/medien/pdfs/futschek_5_11.pdf, letzter Aufruf: 08.04.2015

Geisler Uwe, Digitale Zaubereien im Informatikunterricht. Komplexe Informatik Themen als Science Show für die Sekundarstufe I. In: Brandhofer Gerhard u.a. Hrsg., 25 Jahre Schulinformatik in Österreich. Zukunft mit Herkunft. Wien: Österreichische Computer Gesellschaft, 2010, S. 198-201

Gomes Anabela & Mendes Antonio Jose, Learning to program – difficulties and solutions. Coimbra: International Conference on Engineering Education, 2007. online unter http://ineerweb.osanet.cz/Events/ICEE2007/papers/411.pdf, letzter Aufruf: 18.03.2015

Kölling Michael, The Secret of Programming Education. Creating Motivation. In: Brandhofer Gerhard u.a. Hrsg., 25 Jahre Schulinformatik in Österreich. Zukunft mit Herkunft. Wien: Österreichische Computer Gesellschaft, 2010, S. 33-34

Küchlin Wolfgang & Weber Andreas, Einführung in die Informatik. Objektorientiert mit Java. Berlin: Springer, 2005

Lahtinen Essi, Ala-Mutka Kirsti & Jarvinen Hannu-Matti, A Study of the Difficulties of Novice Programmers. Monte de Caparica: Innovation and Technology in Computer Science Education, 2005. online unter https://student.brighton.ac.uk/mod_docs/cmis/past%20papers/ci_modules/level%20_2/2007_08/ci215_cs2%20iticse2005_novice_programmers.pdf, letzter Aufruf: 22.03.2015

Lehman Josh, Software vs. Apps. am 14. August 2012. online unter http://www.joshlehman.com/thoughts/software-vs-apps/, letzter Aufruf am 23.04.2015

Lobnig Tanja, Probleme von Programmieranfänger/inne/n. Klagenfurt: Alpen-Adria Universität, 2008. online unter: https://www.ddi.edu.tum.de/fileadmin/tueds10/www/material/Lehre/Seminare/BestOf/08-KL-Lobnig-Programmieranfaenger.pdf, letzter Aufruf: 19.03.2015

Mayr Doris, Schülergerechte Einführung in die Programmierung und in das algorithmische Denken anhand von visuellen Programmiersprachen, Diplomarbeit. Wien: Universität, 2014

Meyer Hilbert, Was ist guter Unterricht? Berlin: Cornelsen Scriptor, 2011

Milne Iain & Rowe Glenn, Difficulties in Learning and Teaching Programming – Views of Students and Tutors. Printed in the Netherlands: Education and Information Technologies, 2002. online unter: http://www.swisseduc.ch/informatik-didaktik/programmieren-lernen/docs/milne.pdf, letzter Aufruf: 22.03.2015

Massachusetts Institute of Technology (MIT), MIT App Inventor. online unter http://appinventor.mit.edu/explore/, letzter Aufruf: 19.06.2015

Massachusetts Institute of Technology (MIT), MIT App Inventor. Space Invaders. online unter http://explore.appinventor.mit.edu/ai2/space-invaders, letzter Aufruf: 12.06.2015

Mittermeir Roland, Schulinformatik – ein Fach oder ein Gebiet? In: Reiter Anton u.a. Hrsg., Schulinformatik in Österreich. Erfahrungen und Beispiele aus dem Unterricht. Wien: Ueberreuter, 2003, S. 5–14

Mobile Marketing Association (MMA), Mobile Communication Report – MMA Umfrage 2014. online unter http://www.mmaaustria.at/html/img/pool/Mobile_Communications_Report_2014.pdf, letzter Aufruf: 22.04.2015

Modrow Eckart, Informatik mit BYOB / Snap! Göttingen: Universität, 2013. online unter http://www.uni-goettingen.de/de/document/download/6dcfce31e8f072cd0a280153d6efdd20.pdf/Informatik%20mit%20BYOB.pdf, letzter Aufruf: 20.04.2015

Müller Heinrich & Weichert Frank, Vorkurs Informatik. Der Einstieg ins Informatikstudium. Wiesbaden: Teubner, 2005

Perry Greg, Jetzt lerne ich Programmieren. Von Basic, Java, C++ bis zu .NET. Alle wichtigen Sprachen und Techniken im Überblick. München: Markt & Technik-Verlag, 2002

Pixabay. online unter http://pixabay.com, letzter Aufruf: 19.06.2015

Reiter Anton, 20 Jahre Schulinformatik in Österreich und IKT-Einsatz im Unterricht. Perg, Wien: CDA Verlag, 2005

Schiffer Stefan, Visuelle Programmierung. In: Rechenberg Peter & Pomberger Gustav (Hrsg.), Informatik-Handbuch. München, Wien: Hanser, 1999

Soundbible.com, Free Sound Clips, Sound Bites, and Sound Effects. online unter http://soundbible.com/, letzter Aufruf: 19.06.2015

Stadtmann Cornelia, App-Programmierung. online unter https://www.imst.ac.at/files/projekte/929/berichte/929_Langfassung_Stadtmann.pdf, letzter Aufruf: 08.06.2015

Statistik Austria, Personen mit Nutzung tragbarer Geräte für den mobilen Internetzugang außerhalb des Haushalts oder der Arbeit 2014. online unter http://www.statistik.at/web_de/statistiken/informationsgesellschaft/ikt-einsatz_in_haushalten/022210.html, letzter Aufruf: 22.04.2015

Stroustrup Bjarne, Einführung in die Programmierung mit C++. München: Pearson Studium, 2010

Tschersich Markus, Was ist ein mobiles Endgerät? am 09. März 2010. online unter http://www.mobile-zeitgeist.com/2010/03/09/was-ist-ein-mobiles-endgeraet/, letzter Aufruf: 20.04.2015

Vogel Dieter, Smartphones. Nutzung während der face-to-face Kommunikation. Wien: Magisterarbeit an der Fakultät für Sozialwissenschaften, 2013

Wikimedia Commons. online unter https://commons.wikimedia.org/wiki/Hauptseite, letzter Aufruf: 19.06.2015

Wolber David, App Inventor and Real-World Motivation. San Francisco: Universität, Department of Computer Science. online unter http://cs.millersville.edu/~webster/cs406MDD/stuff/ACMSIGCSE2011Papers/p601.pdf, letzter Aufruf: 05.06.2015

Wolber David, Appinventor.org. App building for everyone. online unter http://www.appinventor.org/content/CourseInABox/drawAnimate/PongTutorial letzter Aufruf: 17.06.2015

Tabellenverzeichnis

Abbildungsverzeichnis

Lebenslauf

Persönliche Daten

Vor- und Zuname:	Markus Dorfstätter
Geburtsdatum:	14. 12. 1988
Geburtsort:	Neunkirchen
Staatsangehörigkeit:	Österreich

Bildungslaufbahn

1995 bis 1999	Volksschule Grafenbach – St. Valentin
1999 bis 2003	Hauptschule Neunkirchen Augasse
2003 bis 2008	HTBLuVA Wiener Neustadt, Abteilung EDV und Organisation
2011 bis 2015	Universität Wien, Lehramtsstudium Geschichte / Informatik

Berufserfahrung

Jun. 2009 – Jan. 2011:	Softwareentwickler bei der Smarter Business Solutions GmbH
Okt. 2012 – Okt. 2013:	Softwareentwickler bei der dvo Software Entwicklungs- und Vertriebs-GmbH
Dez. 2013 – Jan. 2015:	Softwareentwickler bei der Smarter Business Solutions GmbH

Zusammenfassung

Die Entwicklung von mobilen Geräten und deren Software wurde in den letzten Jahren stets vorangetrieben. Der Großteil der SchülerInnen besitzt mittlerweile eigene Smartphones oder Tablets und verbringt bis zu mehreren Stunden täglich mit der Bedienung dieser Geräte. Gerade in der heutigen Zeit fällt es im Informatikunterricht schwer, mit den technischen Innovationen am Markt mitzuhalten. Daher scheint es notwendig, dass die SchülerInnen vor allem mit den langlebigen Prinzipien der Informatik vertraut werden und die grundlegenden Programmierkonzepte, sowie das algorithmische Denken im Unterricht lernen und üben können. Dabei geht hervor, dass die Wahl der Programmiersprache sekundär ist und vor allem visuelle Programmiersprachen und Entwicklungsumgebungen einen Beitrag dazu leisten können, die Schwierigkeiten von ProgrammieranfängerInnen zu vermindern. Der vom MIT entwickelte App Inventor stellt exakt diese Kombination aus der Orientierung am mobilen Markt und dem Erlernen einer visuellen Programmiersprache dar, indem er ProgrammieranfängerInnen und fortgeschrittenen ProgrammiererInnen die Möglichkeit gibt, selbst Anwendungen für ihre mobilen Geräte zu entwickeln. In dieser Arbeit findet sich daher neben allgemeinen, grundlegenden Informationen und Fragestellungen zum AHS-Programmierunterricht, die Vorstellung der Entwicklungsumgebung „MIT App Inventor“ und dessen Vergleich mit anderen Programmiersprachen. Abschließend wurden mit diesem Tool 20 Mobile Apps selbst entwickelt, um eine mögliche Umsetzung im AHS-Programmierunterricht zu demonstrieren.

Abstract

The development of mobile devices and their software continuously increased in the last years. The majority of the kids in the schools already own smartphones or tablets and use them for several hours each day. Especially in our generation it's hard for our computer science education in schools to hold on with the technical improvements nowadays. Because of this fact it seems that the students need to know about the durable principles of computer science, the basic programming concepts and of course the possibility to improve their algorithmic thinking at school. The choice which programming language is used at school is secondary and graphic-oriented programming languages and development environments can reduce the programming difficulties of novice programmers. The Massachusetts Institute of Technology (MIT) provides a development environment which is called "App Inventor", which is exactly the combination of the work with mobile devices and the usage of a graphic-oriented programming language. In this thesis there is some basic information and questions for the computer science education in schools as well as the introduction of the development environment "App Inventor" and his comparison with other programming languages. Finally, 20 mobile apps were created with this tool to show a possible realization in school.