Institut für Parallele und Verteilte Systeme

Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Bachelorarbeit

# Bereitstellung und Erweiterung einer visuellen Entwicklungsplattform für Apps

Katharina Gönner

| | |
|---|---|
| **Studiengang:** | Medieninformatik |
| **Prüfer/in:** | Prof. Dr. Bernhard Mitschang |
| **Betreuer/in:** | Dr. Christoph Stach |
| **Beginn am:** | 21. Mai 2018 |
| **Beendet am:** | 21. November 2018 |

## Kurzfassung

E-Learning hat in den letzten Jahren immer mehr an Bedeutung gewonnen. Dabei kann der Lernprozess durch Chats, Foren oder Anwendungen unterstützt werden. Ein Beispiel für solche Anwendungen sind visuelle Entwicklungsplattformen, welche dazu verwendet werden können Animationen oder auch mobile Anwendungen zu erstellen. Diese visuellen Entwicklungsplattformen bieten eine grafische Oberfläche zur Erstellung von Programmen, sodass mithilfe grafischer Elemente anstatt mit Text gearbeitet werden kann. Gerade im Bereich der Informatik geben visuelle Entwicklungsplattformen die Möglichkeit, Schüler dafür zu begeistern und ihnen den Einstieg in die Programmierung zu erleichtern, da aufgrund der grafischen Elemente die korrekte Syntax des Programms immer gewährleistet ist. Die folgende Arbeit dreht sich um die Bereitstellung und Erweiterung einer solchen Plattform. Bei dieser Anwendung handelt es sich um den App Inventor, welcher für die Erstellung mobiler Anwendungen verwendet werden kann. Die Verwendung des App Inventors bei Workshops an der Universität bringt verschiedene Probleme mit sich, wie eventuelle Verbindungsabbrüche, Sprachprobleme der Teilnehmer, fehlende Funktionalitäten oder Anpassbarkeit an bestimmte Nutzergruppen. Um diese Probleme zu adressieren, werden zunächst einige Grundlagen behandelt, wozu E-Learning, Anforderungen an E-Learning Plattformen und eine Einführung in visuelle Entwicklungsplattformen gehören. Daraufhin folgt eine Beschreibung des App Inventors mit einer Auswertung bezüglich der zuvor beschriebenen Anforderungen an E-Learning Plattformen. Als Resultat der Evaluation wird beschrieben wie eine lokale Installation des App Inventors durchgeführt werden kann, wie man die Sprachunterstützung erweitert und wie es möglich ist, den App Inventor um zusätzliche Komponenten zu erweitern. Abschließend findet eine erneute Evaluation, anhand zweier mit dem App Inventor erstellter Applikationen und anhand der zuvor definierten Anforderungen, statt.

# Inhaltsverzeichnis


**1 Einleitung** **13**

**2 E-Learning** **15**
2.1 Definition . . . . . 15
2.2 Arten von E-Learning . . . . . 16
2.3 Relevante Ziele . . . . . 17

**3 Anforderungen an E-Learning Plattformen** **19**

**4 Visuelle Entwicklungsplattformen** **21**
4.1 Definitionen . . . . . 21
4.2 Anwendungen . . . . . 22

**5 App Inventor 2** **27**
5.1 Designer . . . . . 28
5.2 Blocks Editor . . . . . 30
5.3 Technische Grundlagen . . . . . 32
5.4 Probleme des App Inventors . . . . . 37

**6 Lokale Installation einer App Inventor Instanz** **39**

**7 Sprachunterstützung** **43**

**8 Eine neue App Inventor Komponente** **47**
8.1 Eine sichtbare Komponente - ToggleButton . . . . . 48
8.2 Eine unsichtbare Komponente - euklidischer Algorithmus . . . . . 54

**9 Evaluation** **57**

**10 Zusammenfassung und Ausblick** **63**

**Literaturverzeichnis** **65**

# Abbildungsverzeichnis


2.1 E-Learning Typen und dazugehörige Beispiele . . . . . 17

4.1 Benutzeroberfläche von RAPTOR . . . . . 23
4.2 Benutzeroberfläche von Alice 3 . . . . . 23
4.3 Benutzeroberfläche von Scratch . . . . . 25

5.1 Startschirm der einfachen App mit Textfeld und Knopf . . . . . 28
5.2 Zweiter Screen mit einem ListPicker-Button . . . . . 29
5.3 ListPicker zeigt nach Touch auf den Button den Inhalt der Liste an . . . . . 30
5.4 Screenshot des Designers des ersten Screens mit der TextBox, dem Button und dem Player (Startbildschirm) . . . . . 31
5.5 Screenshot des Designers des zweiten Screens mit dem ListPicker . . . . . 31
5.6 Screenshot des Blocks Editors mit dem Code für die TextBox, Liste, den Button und den Player (Startbildschirm) . . . . . 33
5.7 Screenshot Blocks Editors mit dem Code für den ListPicker . . . . . 33
5.8 Übersicht über die Projekte und wo sie ausgeführt werden . . . . . 36
5.9 Anforderungen an den AI2 und welche Probleme durch Erfüllung dieser Anforderungen gelöst werden . . . . . 38

6.1 Komponenten der lokalen Installation einer App Inventor Instanz, auf einer Maschine 40
6.2 Komponenten der aktuell verfügbaren App Inventor Instanz, getrennt zwischen der Google Infrastruktur und der Maschine, auf der sich der Buildserver befindet . . 40

8.1 Arbeitsschritte für die Erstellung einer Komponente und deren Aufbau . . . . . 48
8.2 Designer mit verwendeter ToggleButton-Komponente . . . . . 51
8.3 Blocks Editor mit verwendetem Methoden-Block der neuen ToggleButton-Komponente . . . . . 54
8.4 Designer mit verwendeter euklidischer Algorithmus-Komponente . . . . . 55
8.5 Blocks Editor mit verwendetem Methoden-Block der neuen euklidischer Algorithmus-Komponente . . . . . 56

9.1 Screenshots einer Anwendung, welche den ToggleButton verwendet um zwischen zwei Bildern einer Glühbirne hin und her zu wechseln . . . . . 58
9.2 Screenshots einer Anwendung, welche den ggT zweier Zahlen unter Verwendung der euclidean Algorithm Komponente berechnet . . . . . 59
9.3 Blöcke der App, welche den ToggleButton verwendet . . . . . 59
9.4 Blöcke der App, die den ToggleButton nicht verwendet . . . . . 60
9.5 Blöcke der App, welche die ggT-Komponente verwendet . . . . . 60
9.6 Blöcke der App, welche die ggT-Komponente nicht verwendet . . . . . 61

# Tabellenverzeichnis

8.1 Kurzbeschreibung der Annotationen . . . . . 50

# Verzeichnis der Listings

5.1 Ausschnitt des onClickListeners als Android-Code . . . . . 34
7.1 Ausschnitt aus „OdeMEssages_de_DE.properties“ . . . . . 43
7.2 Ergänzungen in „TopPanel.java“ . . . . . 44
7.3 Ausschnitt aus „OdeMessages.java“ . . . . . 44
7.4 Ergänzungen in „YaClient.gwt.xml“ . . . . . 44
7.5 Ergänzungen in „language_switch.js“ . . . . . 45
7.6 Ergänzungen in „ploverConfig.js“ . . . . . 45
8.1 Ausschnitt aus „ToggleButton.java“: Verwendung der Annotationen `Designer-Component` und `SimpleObject` . . . . . 49
8.2 Ausschnitt aus „Images.java“: Import des neuen Icons für den Designer . . . . . 50
8.3 Ausschnitt aus „SimpleComponentDescriptor.java“: Ergänzung in der If-Abfrage der `createMockComponent`-Methode . . . . . 50
8.4 Ausschnitt aus „ToggleButton.java“: TextUp-Methoden . . . . . 52
8.5 Ausschnitt aus „MockToggleButton.java“: setTextUpProperty-Methode . . . . . 52
8.6 Ausschnitt aus „OdeMessages.java“: Ergänzungen für ToggleButton . . . . . 53
8.7 Ausschnitt aus „MockVisibleComponent.java“: Ergänzungen für ToggleButton . 53
8.8 Ausschnitt aus „SimpleComponentDescriptor.java“: Ergänzung in der `initBundledImages()` . . . . . 56

# Verzeichnis der Algorithmen

8.1 Euklidischer Algorithmus (größter gemeinsamer Teiler) . . . . . 56

# Abkürzungsverzeichnis

**adb** Android Debug Bridge. 40

**AI2** App Inventor 2. 27

**AIS** App Inventor Service. 39

**APK** Android Package. 35

**GAE** Google App Engine. 32

**ggT** größter gemeinsamer Teiler. 53

**GUI** Graphical User Interface. 27

**GWT** Google Web Toolkit. 32

**RPC** Remote Procedure Call. 32

**VPE** visuelle Programmierumgebung. 21

**VPL** visuelle Programmiersprache. 21

**YAIL** Young Android Intermediate Language. 32

# 1 Einleitung

E-Learing ist heutzutage ein viel verwendeter Begriff, der sehr heterogen eingesetzt wird. E-Learning kann mithilfe von Chats, Foren, Videokonferenzen oder speziellen E-Learning Anwendungen stattfinden. Für die Informatik im Speziellen ist die Verwendung von visuellen Entwicklungsplattformen zu betrachten. Diese Art von E-Learning Plattformen sind grafische Tools für die Programmierung. Bekannte Anwendungen in diesem Bereich sind vor allem Scratch[1], zur Entwicklung von 2D Animationen, oder der App Inventor[2], zur Entwicklung von mobilen Anwendungen für Android-Geräte. Eines der Ziele der visuellen Entwicklungsplattformen ist es, Menschen beim Programmierenlernen zu unterstützen [KP05]. Dies geschieht zum einen vor allem über das Reduzieren der Syntax, um es den Lernenden zu ermöglichen sich mehr auf die Logik der Programmieraufgabe zu konzentrieren [Pap+16]. Die Reduktion der Syntax geschieht durch die grafischen Elemente der Programmierung. So verwenden sowohl Scratch als auch der App Inventor sogenannte Blöcke für die Programmierung. Diese Blöcke werden in der Entwicklungsplattform wie Puzzle-Teile aneinander gehaftet und bilden dann den Quellcode der vom Nutzer erstellten Anwendung. Zum anderen können Anwendungen, wie zum Beispiel der App Inventor, die Schüler und Studenten durch die Entwicklung mobiler Anwendungen begeistern [Hon13]. Somit hat der App Inventor in diesem Gebiet einen weiteren Vorteil, da ohne die Hürde der Syntax schnell Applikationen mit einem Nutzen für die reale Welt gebaut werden können, was die Lernenden zusätzlich dazu motiviert sich mehr mit der Logik auseinander zu setzen [Wol11].

Aufgrund der motivierenden Kraft des App Inventors liegt in der Arbeit der Fokus auf dieser Anwendung. Da es verschiedene Anforderungen an E-Learning Plattformen gibt und der App Inventor an der Universität Stuttgart bei einigen Workshops für SchülerInnen[3] verwendet worden ist und somit ein Erfahrungswert besteht, welche Probleme auftreten könnten, beschäftigt sich die Arbeit hauptsächlich mit der Bereitstellung und der Erweiterung des App Inventors. Im Rahmen der Workshops hat sich gezeigt, dass es zu Systemausfällen oder Verbindungsabbrüchen kommen kann, dass Schüler mit Sprachproblemen zu kämpfen haben, da der App Inventor nicht auf deutsch zur Verfügung steht, dass verschiedene Funktionalitäten fehlen und dass in der Anwendung bisher keine Möglichkeit zur Anpassung an heterogene Nutzergruppen besteht. Dabei soll unter Zuhilfenahme der Grundlagen des E-Learnings eine Liste der Anforderungen entwickelt werden, anhand derer der App Inventor schließlich ausgebaut werden soll. Nach der Erweiterung wird abschließend eine erneute Evaluation stattfinden, welche bestimmen soll ob die zuvor gestellten Anforderungen an die Verbesserung des App Inventors durch die Erweiterungen erfüllt worden sind.

Die Ausarbeitung ist folgendermaßen gegliedert:
Zunächst beginnt die Arbeit mit der Erläuterung des Begriffs E-Learning in Kapitel 2. Dabei findet nicht nur eine Definition, sondern auch eine Beschreibung der relevanten Ziele von E-Learning

[1] Zu finden unter: [MIT16]
[2] Zu finden unter: [Mas18a]
[3] siehe [Ber+15; Ste+13]

Anwendungen statt. Im Zusammenhang mit den relevanten Zielen der E-Learning Plattformen werden in Kapitel 3 Anforderungen definiert, mit deren Hilfe später eine Evaluation und die Erweiterung einer Plattform stattfinden soll. Anschließend gibt Kapitel 4 einen Einblick darin, was visuelle Entwicklungsplattformen sind und zeigt einige Beispiele auf. Darauf folgt die ausführliche Beschreibung des App Inventors in Kapitel 5. Darin werden sowohl die Anwendung selbst, als auch die technischen Grundlagen besprochen, die eine Basis für die weitere Arbeit an der Anwendung bilden. Nach einer Bewertung anhand der Anforderungen in Kapitel 3 folgt in Kapitel 6 bis 8 die Arbeit am App Inventor zur Bearbeitung der zuvor gefundenen Kritikpunkte. In Kapitel 6 wird beschrieben wie der App Inventor lokal installiert werden kann und Kapitel 7 beschreibt wie die Sprachunterstützung erweitert werden kann. Darauffolgend wird in Kapitel 8 erläutert wie im App Inventor neue Komponenten hinzugefügt werden können. Anschließend kommt es in Kapitel 9 zu einer erneuten Evaluation, diesmal des erweiterten App Inventors, anhand Kapitel 3 und mit dem App Inventor erstellten mobilen Anwendungen. Zum Abschluss folgt in Kapitel 10 eine Zusammenfassung und ein Ausblick auf die mögliche Fortsetzung der Arbeit.

# 2 E-Learning

In diesem Kapitel werden zunächst einige Grundlagen der Arbeit erörtert. Hierzu gehört eine Einführung in das Thema E-Learning, eine Herleitung einer Definition für die weitere Arbeit und die Betrachtung sowohl der Arten, als auch der Ziele von E-Learning Plattformen.

## 2.1 Definition

> „E-learning is an approach to teaching and learning, representing all or part of the educational model applied, that is based on the use of electronic media and devices as tools for improving access to training, communication and interaction and that facilitates the adoption of new ways of understanding and developing learning.“
>
> Sangrà et al. [San+12]

Nach eingehender Recherche wird schnell klar, dass *E-Learning* in der Literatur unterschiedlich definiert wird. Aus diesem Grund werden im Folgenden mehrere Definitionen aus verschiedenen wissenschaftlichen Publikationen betrachtet.
In ihrem Beitrag zur Differenzierung zwischen E-Learning und Fernunterricht, definiert Guri-Rosenblit E-Learning „als die Nutzung elektronischer Medien, für eine Auswahl von Lernabsichten, welche von erweiternden Funktionen in konventionellen Klassenräumen bis hin zur vollen Substitution persönlicher Treffen durch online Treffen reichen.“ [Gur05, aus dem Englischen übersetzt].
Koohang et al. definieren E-Learning wiederum als „die Lieferung von Bildung (alle Aktivitäten, die relevant zur Anleitung, zum Lehren und Lernen sind) durch verschiedene elektronische Medien“ [KH05, aus dem Englischen übersetzt].
E-Learning ist laut Naidu „ein bildender Prozess, welcher Informations- und Kommunikationstechnologien nutzt, um asynchrone und synchrone Lern- und Lehraktivitäten zu vermitteln“ [Nai02, aus dem Englischen übersetzt].
Nach Carapeto ist E-Learning „Lernen, das durch die Verwendung von digitalen Hilfsmitteln und Inhalt gefördert wird. Typischerweise beinhaltet es eine Form von Interaktivität, was online Interaktion zwischen dem Lernenden und dessen Lehrer oder Gleichaltrigen beinhalten kann“ [Car02, aus dem Englischen übersetzt].

Der Artikel von Sangrà et al. beschäftigt sich mit den verschiedenen Definitionen und der Findung einer allgemeinen Definition des Terms. Die Definitionen werden im Laufe des Artikels in vier Kategorien eingeteilt. Diese vier Arten von Definitionen werden als *Technology-Driven*,

*Delivery-System-Oriented*, *Educational-Paradigm-Oriented* und *Communication-Oriented* bezeichnet. Definitionen, welche als Technology-Driven beschrieben werden, betonen die technologischen Aspekte des E-Learning. Definitionen, die hingegen E-Learning als eine Möglichkeit Zugriff auf Wissen zu erhalten beschreiben, werden als Delivery-System-Oriented bezeichnet. Educational-Paradigm-Oriented bezeichnet Definitionen, die E-Learning als neuen Weg des Lernens oder Verbesserung eines existierenden Bildungsparadigmas beschreiben. Communication-Oriented bedeutet, dass E-Learning als Anwendung zur Kommunikation, Interaktion und Kollaboration definiert wird [San+12].

Somit lassen sich die oben genannten Definitionen auf die vier Typen abbilden. Dabei sind sie passend der Reihenfolge der beschriebenen Typen angeordnet: Technology-Driven, Delivery-System-Oriented, Educational-Paradigm-Oriented [San+12] und Communication-Oriented.

Sangrà et al. erarbeiten ebenfalls eine allgemeine Definition unter Verwendung einer Studie, welche im Laufe dieser Arbeit nun verwendet wird. Damit wird E-Learning von ihnen definiert als „ein Ansatz zum Lehren und Lernen, der das gesamte oder Teile des angewendeten Bildungsmodells repräsentiert, der auf elektronischen Medien und Geräten als Hilfsmittel für die Verbesserung des Zugangs zu Training, Kommunikation und Interaktion basiert und der die Einführung neuer Wege zum Verstehen und der Entwicklung von Lernen fördert“ [San+12, aus dem Englischen übersetzt].

Wie in der Definition von Naidu schon angesprochen, gibt es asynchrone und synchrone Lernaktivitäten [Nai02]. In dem Bericht von Welsh et al. wird E-Learning in eben diese diese zwei Arten eingeteilt, welche sich *synchrones* und *asynchrones E-Learning* nennen [Wel+13].

## 2.2 Arten von E-Learning

Somit existieren zwei Basistypen von E-Learning. Wie in Abbildung 2.1 zu sehen, sind das asynchrones und synchrones E-Learning [Hra08; Wel+13], wobei diese durch einen weiteren Typ, *blended E-Learning*, ergänzt werden [Rui+06; Wel+13]. Von diesen drei Typen wird asynchrones E-Learning am häufigsten verwendet [Wel+13]. Diese Art von E-Learning wird zum Beispiel mithilfe von E-Mail oder Diskussionsboards durchgeführt, somit müssen die Schüler und Lehrer nicht gleichzeitig online sein [Hra08]. In Gegensatz dazu findet synchrones E-Learning unmittelbar statt [Wel+13]. Synchrones E-Learning kann unter Verwendung von Medien wie Videokonferenzen oder Chats stattfinden [Hra08]. Als die einfachste Art von synchronem E-Learning wird ein Chat in Realzeit angeführt, wobei das Beispiel für kompliziertes synchrones E-Learning ein solcher Chat mit mehreren Schülern, an verschiedenen Orten, welche sich zur gleichen Zeit einloggen, ist [Wel+13]. Des Weiteren gibt es blended Learning. Es kombiniert E-Learning Technologien mit traditionellem Lernen in einem Klassenzimmer durch einen Lehrer [Rui+06; Wel+13]. Hierbei führen Welsh et al. als Beispiel die Lösung einer Firma an, bei der die Teilnehmer des Kurses zunächst verschiedene Themen mit einer E-Learning Anwendung bearbeiten und anschließend für einen interaktiven Teil zusammen treffen [Wel+13]. Um zu bestimmen welche der oben genannten Klassen von E-Learning für diese Arbeit relevant sind, ist eine Betrachtung der Vorteile, Ziele und Grenzen von E-Learning notwendig.

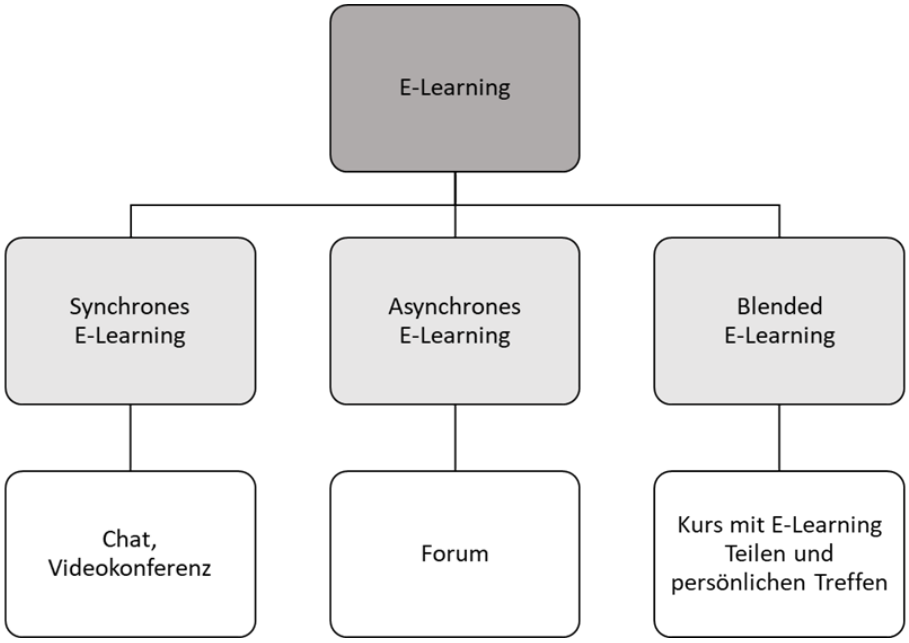



**Abbildung 2.1:** E-Learning Typen und dazugehörige Beispiele

## 2.3 Relevante Ziele

E-Learning hat verschiedene Vorteile, welche der Grund dafür sein könnten, dass es immer mehr Verwendung findet, sowohl in seiner Reinform als auch in der Form von blended E-Learning.
In der Literatur werden verschiedene Vorteile von E-Learning behandelt. Allgemein für alle Arten von E-Learning gilt, dass die Anzahl der Verwender nicht beschränkt ist [KW03]. Außerdem ist E-Learning nicht vom Standort des Lernenden abhängig, da die Plattformen von überall aus erreicht werden können [Cap01]. Des Weiteren ermöglichen online Angebote neue Strategien für das Lernen und die Einbindung eines Computers in das Lernangebot, da die Teilnehmer sich für das E-Learning schon an ihrem Computer befinden [Cap01].
Es gibt weitere Vorteile, die sich allerdings lediglich auf asynchrones E-Learning beziehen. Eines davon ist die ständige Erreichbarkeit der Plattform. So kann die Plattform zu jeder beliebigen Zeit verwendet werden [Cap01; KW03]. Klein und Ware führen in ihrer Arbeit an, dass E-Learning flexibel ist. Damit können die Verwender der Plattform nicht nur selbst entscheiden wann sie lernen, sondern auch wie schnell sie lernen [KW03]. Da die Kommunikation für E-Learning keines Falls synchron sein muss, können asynchrone Unterhaltungen zwischen den Lernenden mit mehr Bedachtsamkeit geführt werden, da die Antworten nicht sofort gegeben werden müssen [Cap01]. Die Verwendung von elektronischen Nachrichten, ermöglicht es Gruppen auf eine andere Art und Weise zusammen zu arbeiten, da wie bereits im vorigen Punkt erwähnt die Nachrichten bedachter und außerdem von permanenterer Natur sind [Cap01].

Laut einer Untersuchung über synchrone und asynchrone online Seminare von Hrastinski sind die meisten Sätze einer Unterhaltung in asynchroner Form auf das Thema bezogen, wobei mögliche andere Einteilungen die Planung und soziale Unterstützung sind. Diese Themenbezogenheit kann sich allerdings auch negativ auf die Studenten auswirken, in Form eines Gefühls von Isolation [Hra08]. Robert und Dennis *kognitives Modell zur Wahl von Medien* (*„Cognitive Model*

*of Media Choice"*) [RD05] besagt, dass die Leistung gleichzeitig sowohl negativ als auch positiv beeinflusst wird, sollte das Medium vielfältig und reich an sozialer Präsenz sein, da einerseits die Motivation steigt, aber die Fähigkeit Informationen zu verarbeiten sinkt. Im Rückschluss darauf verhält sich das Ganze bei eintönigen Medien mit wenig sozialer Präsenz genau gegenteilig. Dies wird von Hrastinskis Untersuchung unterstützt, da den Teilnehmern einer asynchronen Gruppe mehr Zeit bleibt um Fragen zu beantworten und sie somit mehr Zeit haben sich mit dem Thema der Frage zu beschäftigen [Hra08]. In synchronen E-Learning Seminaren beschäftigen sich Studenten nach Hrastinski zwar immer noch zum größeren Teil mit dem Thema, aber auch mit der Planung, da die Dauer beschränkt ist. Unter Verwendung der oben genannten Theorie sind die Studenten motivierter, andererseits waren Teile der Gespräche der Teilnehmer auch in den Bereich sozialer Unterstützung einzuordnen und die Diskussionen handelten von weniger komplexen Themen [Hra08].

Da es im späteren Verlauf um das Lernen von Programmieren gehen soll, ist der Fokus auf die Verwendung von asynchronen E-Learning Plattformen zu setzen. Diese Entscheidung ist vor allem darauf basiert, dass es sich beim Programmieren um ein eher kompliziertes Thema handelt und daher ein wichtiger Teil des Lernens auf der Verarbeitung von Informationen basiert. Damit haben die Lernenden mehr Zeit für die Verarbeitung der Informationen und können ihre Arbeitszeit flexibel wählen.
Um das Lernen mit einer solchen Plattform zu optimieren und alle Vorteile nutzen zu können, muss ein E-Learning Angebot die im folgenden Kapitel aufgestellten Anforderungen erfüllen.

# 3 Anforderungen an E-Learning Plattformen

Um das Lernen mit einer E-Learning Plattform so effektiv wie möglich zu gestalten, werden in diesem Kapitel einige Anforderungen zusammen getragen.

Unter der Verwendung des Modells von Robert und Dennis [RD05], welches in Abschnitt 2.3 genauer beschrieben wird, kann man schlussfolgern, dass ein guter Ausgleich gefunden werden muss, um eine möglichst hohe Motivation und Informationsverarbeitung zusammen zu erreichen. Dementsprechend muss die Vielfältigkeit des Mediums mit der sozialen Präsenz in Einklang gebracht werden.
Im Folgenden werden deshalb Anforderungen an E-Learning Plattformen vorgestellt und mit den zuvor besprochenen relevanten Zielen von E-Learning in Zusammenhang gebracht.

Das Paper von Haake et al. beschäftigt sich mit der Entwicklung eines Unterrichtskonzepts zum Erwerb der Fähigkeit des *„Collaborative Authoring“*. Dabei wird die Wichtigkeit eben dieser Fähigkeit, gemeinsam in der Gruppe Dokumente zu erstellen, betont [Haa+13].
Bezieht man sich noch einmal auf die Theorie von Robert und Dennis, so führt fehlende Vielfältigkeit des Mediums und niedrige soziale Präsenz unter anderem zu einer geringeren Motivation [RD05]. Somit könnte die Motivation durch hinzufügen einer sozialen Komponente, wie Collaborative Authoring gesteigert werden. Des Weiteren könnte dem von Hrastinski [Hra08] beschriebenen Gefühl von Isolation entgegengewirkt werden, welches unter der Verwendung asynchroner Plattformen auftreten kann, da Collaborative Authoring ein Gefühl von Zusammenarbeit generieren könnte.

Des Weiteren erläutert Herper in seinem Bericht, dass auch an Grundschulen computer-basiertes Lernen eingeführt werden soll, da der Erwerb von Kompetenzen im Bereich der Informatik immer mehr an Bedeutung gewinnt. Dazu sei es wichtig, dass keine technischen Störungen auftreten, um den Verlauf des Unterrichts nicht zu stören [HL13]. Eine technische Störung, welche den Lernprozess eines Nutzers unterbricht, ist selbstverständlich auch bei der privaten Nutzung eines E-Learning Angebots nicht wünschenswert, da eines der Vorteile von E-Learning, wie in Abschnitt 2.3 aufgezeigt, die ständige Erreichbarkeit der Plattform darstellt.

Eine weitere Anforderung besteht an das Verstehen der Plattform in Bezug auf die verwendete Sprache. So ist bei der Durchführung von Workshops für Schüler, welche an der Universität Stuttgart stattgefunden haben[1] aufgefallen, dass es den Schülern schwer fällt mit einer E-Learning Plattform zu arbeiten, wenn die grafische Oberfläche nicht auf deutsch verfügbar ist.

Darüber hinaus beschreibt Herper in seinem Bericht die Verwendung von Computern für alle Fächer. Dabei spricht er davon, dass die Lehrkräfte auf dem neusten Stand sein müssen, um so für jedes Fach die Verwendung von Computern auf das Thema anpassen zu können [HL13]. Somit muss im Rückschluss also eine gewisse Flexibilität in der Lösung vorhanden sein, um den Schülern unterschiedliche Themen nahe bringen zu können.

[1] siehe [Ber+15; Ste+13]

Auch diese Anforderung kann in Bezug auf das kognitive Modell zur Wahl von Medien [RD05] bezogen werden. Durch eine flexible Lösung besteht die Möglichkeit, unterschiedliche Themen zu behandeln, wodurch die Vielfältigkeit des Mediums erhöht wird, was sich wiederum auf die Motivation und Informationsverarbeitung auswirken kann.

Aus der Anforderung der Flexibilität des Inhalts kann man eine weitere Anforderung ableiten: Das Anpassen nicht nur an das Thema, sondern auch an den Schüler selbst. In diesem Zusammenhang beschreiben Louhab et al. einen Ansatz für *adaptives Lernen* (aus dem Englischen „Adaptive Learning“), welchen sie in eine existierende E-Learning Plattform integrieren wollen. Dabei soll die Schwäche der Plattform verbessert werden, dass es keine Möglichkeit gibt den Lernprozess zu kontrollieren. Dabei soll den Lernenden ihren Leistungen entsprechender Inhalt präsentiert werden und den Lehrern Möglichkeit gegeben werden den Lernprozess zu beeinflussen [Lou+17]. Auch Alotaiby et al. beschreiben die Wichtigkeit eines adaptiven Systems, welches den Inhalt an die Leistung der Lernenden anpasst und die Entscheidungen des Lehrers miteinbezieht [Alo+04]. Somit sollte es die Möglichkeit geben, den Inhalt der Anwendung auf das Wissen der Schüler anzupassen beziehungsweise den Inhalt adaptieren zu können.

Zusammenfassend werden also folgende Anforderungen an eine Plattform zum Erlernen von Programmieren gestellt:

A.1 Collaborative Authoring / Zusammenarbeit
A.2 Verfügbarkeit
A.3 Verständnis
A.4 Flexibilität des Inhalts
A.5 Adaptives Lernen

Programmierenlernen kann unter Verwendung sogenannter visueller Entwicklungsplattformen stattfinden. Deshalb wird im nächsten Kapitel erörtert, was eine visuelle Entwicklungsplattform ist und außerdem werden einige der zur Verfügung stehenden Plattformen vorgestellt.

# 4 Visuelle Entwicklungsplattformen

Um die zuvor aufgezählten Vorteile von E-Learning auch für das Lernen von Programmieren nutzen zu können, sind visuelle Entwicklungsplattformen verwendbar.
Da diese unter anderem zum Ziel haben Menschen beim Programmieren lernen zu unterstützen [KP05]. Dies geschieht zum Beispiel über die Reduktion von Syntax, damit die Programmierenden sich mehr auf die Logik konzentrieren können [Pap+16]. Im Laufe dieses Kapitels werden verschiedene visuelle Entwicklungsplattformen vorgestellt und kritisch betrachtet. Ziel ist es zum Ende des Kapitels eine Plattform zu wählen, welche im späteren Verlauf der Arbeit anhand der Anforderungen in Kapitel 3 analysiert und verbessert werden soll.

## 4.1 Definitionen

Auch im Bereich der visuellen Entwicklungsplattformen gibt es verschiedene Definitionen.
Nach Burnett wird *visuelle Programmierung* definiert als die Programmierung, die mehrere Dimensionen verwendet, um Semantik zu übermitteln. Hierbei sind Beispiele für zusätzliche Dimensionen, mehrdimensionale Objekte oder räumliche Beziehungen. Objekte oder Beziehungen bilden sogenannte Tokens, welche bei textuellen Programmiersprachen Wörter darstellen, wobei ein oder mehrere Tokens in der visuellen Programmierung „visual expressions“ (visuelle Ausdrücke) bilden. Ein solcher visueller Ausdruck kann ein Diagramm, ein Icon oder auch eine Skizze sein. Enthält die Syntax einer Programmiersprache visuelle Ausdrücke, so wird die Programmiersprache auch *visuelle Programmiersprache (VPL)*, aus dem englischen „visual programming language“, genannt. Laut Burnett ist eine *visuelle Programmierumgebung (VPE)*, aus dem englischen von „visual programming environment“, eine Entwicklungsumgebung, welche visuelle Ausdrücke verwendet um Code schneller schreiben zu können, wobei der Code in Form der visuellen Ausdrücke nicht die gleiche Syntax haben muss wie die der textuellen Programmierung. Somit bezieht Burnett den Begriff VPE auf Plattformen, die auf bereits existierenden Programmiersprachen basieren [Bur95].
Eine andere Definition bezieht sich rein auf den Aspekt, dass die Programmierung über grafische Elemente stattfindet, ohne einen Bezug auf textuelle Programmiersprachen zu nehmen [Bel+14].
Im weiteren Verlauf der Arbeit werden wir die Begriffe VPL und VPE auf die zweite, allgemeinere Weise definieren.

## 4.2 Anwendungen

Wie oben beschrieben, gibt es genau wie bei E-Learning auch bei den VPEs ein sehr heterogenes Verständnis und damit unterschiedliche Einsatzbereiche. Angefangen von Fußball-Training [Bel+14] über die Programmierung von animierten Figuren [Mal+10] bis hin zur Entwicklung von mobilen Anwendungen [Dic12].
Im Bereich der visuellen Programmiersprachen gibt es zum Beispiel RAPTOR[1] oder Alice[2].

RAPTOR verwendet Symbole für die visuelle Programmierung. Dabei werden Teile aus verschiedenen Programmiersprachen verwendet, zum Beispiel die Struktur einer Schleife von Ada oder auch Teile aus C- oder Pascal-Sprachen. Carlisle et al. beschreiben, dass RAPTOR speziell dafür entwickelt worden ist, Studenten dabei zu helfen sich ihre Algorithmen vorzustellen, ohne dass sie auf die Syntax achten müssen. Dabei können die Programme visuell ausgeführt werden, indem die Ausführung durch das Programm nachverfolgt wird, das bedeutet RAPTOR zeigt an welches Symbol aktuell ausgeführt wird [Car+05].
In Abbildung 4.1 ist die Benutzeroberfläche von RAPTOR zu sehen. Das Programm besteht aus zwei Fenstern, wobei in dem Größeren die grafischen Elemente verwendet werden und in dem Kleineren werden sowohl die Eingaben für die grafischen Elemente getätigt als auch die Ausgaben dargestellt. Beispielhaft ist in dem Screenshot ein Programm zu sehen, welches den Text „Hello World“ ausgibt. Hierfür ist die Verwendung eines „Output“-Symbols notwendig, woraufhin in dem kleineren Fenster „Hello World“ in Anführungszeichen eingegeben werden muss, damit der Text bei der Ausführung des Programms, ebenfalls in den kleineren Fenster, ausgegeben wird.
Alice ist eine Programmiersprache bei der zur Programmierung Blöcke nach dem Drag-and-Drop-Prinzip verwendet werden. Im Moment sind zwei Versionen von Alice erhältlich, Alice 2 und Alice 3, wobei Alice 3 zusätzlich Objektorientierung betont. Alice ist als Anwendung für den Einstieg in die Programmierung gedacht und soll dabei helfen einen Übergang in die Nutzung von Java zu schaffen. Unter der Verwendung von Alice 3 kann man animierte Szenen oder Spiele in 3D programmieren [Car17]. Als Beispiel ist auf Abbildung 4.2 der Code für eine kleine Szene zu sehen, in der ein Alien erst denkt, sich dann auf die Kamera zubewegt und schließlich „hello“ sagt. Wie oben beschrieben besteht der Code aus Blöcken. Diese können über Drag-and-Drop von der linken Seite oder vom unteren Bildschirmrand in den Programmierbereich gezogen werden. Außerdem kann man nicht nur, wie in der Abbildung zu sehen ist, den zugehörigen Java-Code anzeigen lassen [Car17], man kann auch Java als Sprache für die Blöcke auswählen.
Eine visuelle Entwicklungsumgebung zur Entwicklung von mobilen Anwendungen ist Cabana. Cabana ist browser-basiert und ist für die Entwicklung von Android und iOS Applikationen gedacht. Dabei basiert die graphische Oberfläche auf einer Art Schaltplan, auf dem verschiedene Module, in diesem Fall Programmierkonstrukte, miteinander verbunden werden [Dic12].

Die Verwendung von diesen Entwicklungsumgebungen hat viele Vorteile beim Lernen.
Im Bereich der visuellen Programmierung gibt es bereits einige Untersuchungen, die sich damit beschäftigen, wie effizient oder auch effektiv die Verwendung einer VPE ist, um Kindern, Jugendlichen oder Studenten den Einstieg in die Programmierung zu erleichtern und sie dazu zu motivieren in den Bereich der Informatik einzusteigen.

[1] Zu finden unter: [Wil+18]
[2] Zu finden unter: [Car17]

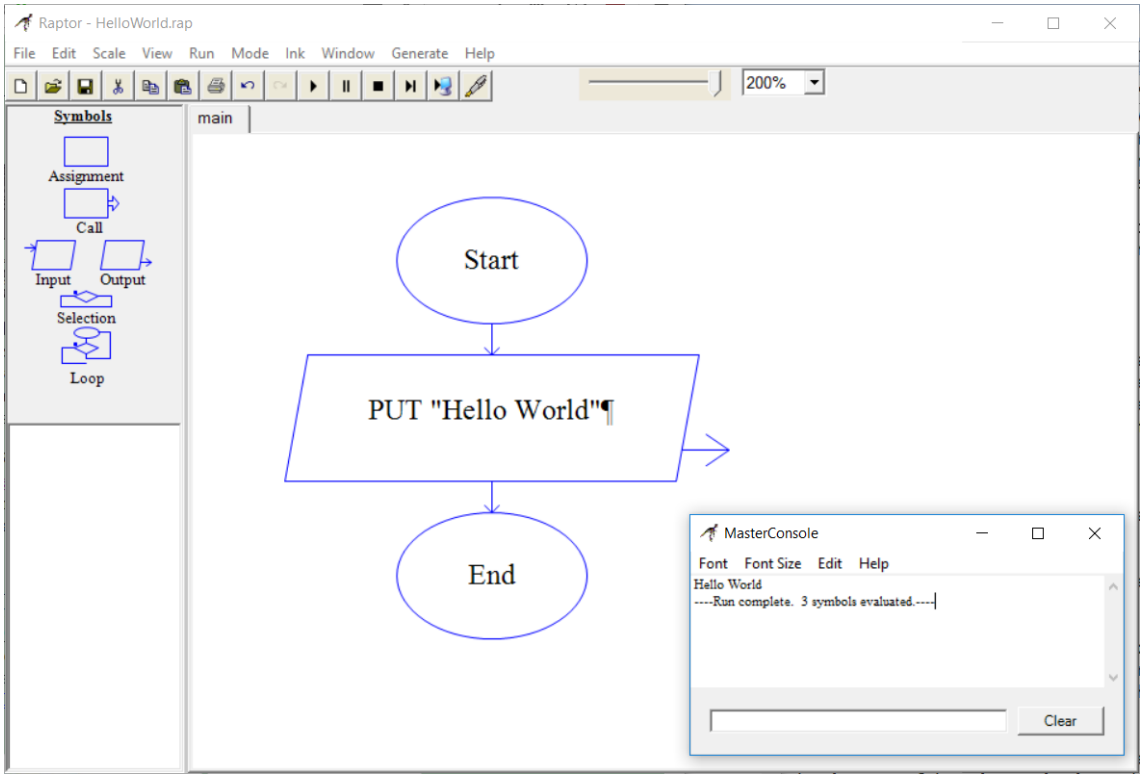



**Abbildung 4.1:** Benutzeroberfläche von RAPTOR

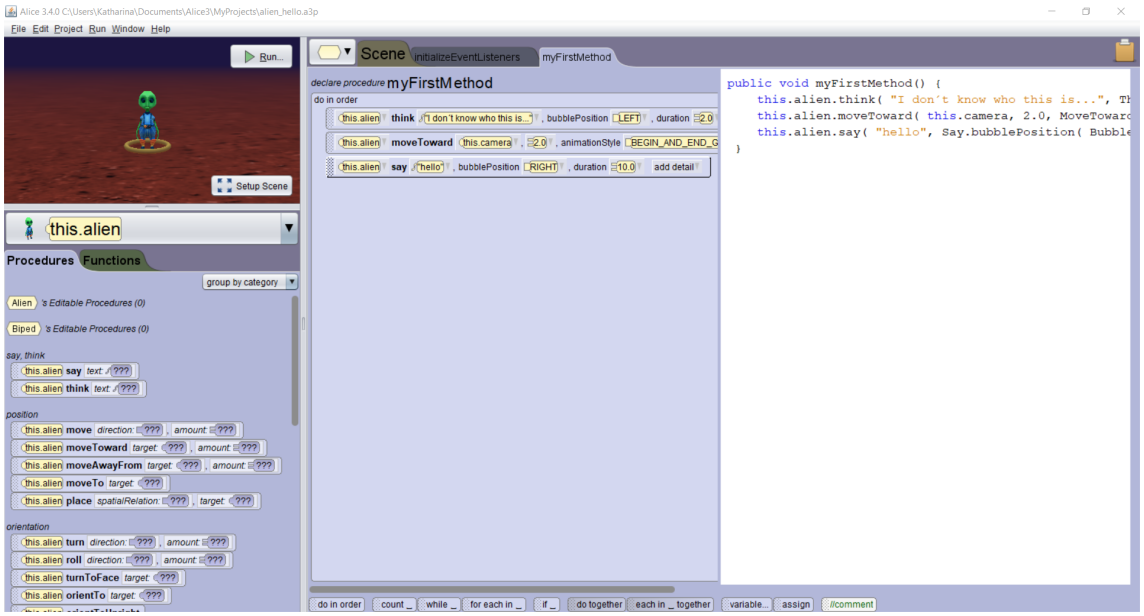



**Abbildung 4.2:** Benutzeroberfläche von Alice 3

Eine Studie von Dann et al. vergleicht die Prüfungsergebnisse von Studenten, welche Programmieren unter Verwendung von Java gelernt haben mit den Ergebnissen von Studenten, welche Programmieren durch einen Transfer von der VPE Alice 3 auf Java gelernt haben. Die Untersuchung zeigt, dass die Noten der Studenten mindestens um eine Note nach oben gegangen sind. Mit den Ergebnissen der Studie folgern die Autoren einen statistisch signifikanten positiven Effekt auf das Lernen von Studenten, wenn das Lernen über einen Transfer von Alice auf Java erfolgt [Dan+12].
Eine Untersuchung zeigt dass High-School Schüler Block-basiertes Programmieren einfacher finden, als text-basiertes Programmieren. Dabei wurden einige Gründe für die Einfachheit genannt, wozu zum Beispiel die natürliche Sprache der Blöcke und die Verwendung von Drag-and-Drop gehören [WW15].
Die Arbeit von Morelli et al. befasst sich damit, ob der App Inventor (eine VPE welche in Kapitel 5 beschrieben wird) dazu verwendet werden kann Schülern ins rechnerische Denken einzuführen. Laut den Autoren ist es einfach, auf schnelle Weise, kompliziertere Apps zu erstellen und außerdem führt die Verwendung des App Inventors gut in die Objektorientierung ein. Anstatt mit Syntax, können sich die Schüler mit dem Lösen von Problemen beschäftigen, wobei der App Inventor, laut den Autoren, Schüler durch seine Relevanz motivieren könnte [Mor+11].
Eine andere Studie von Nikou et al. vergleicht die Motivation von Schülern vom Anfang bis zur Mitte eines Kurses, in welchem verschiedene VPEs verwendet werden. Im Vergleich stehen Scratch und der App Inventor. Nikou et al. beschreiben, dass sich die Schüler im Verlauf des Kurses mehr an intrinsischen Zielen zur Motivation orientieren, die Schüler glauben stärker an den Wert der Aufgabe und daran dass sie im Kurs gut abschneiden werden [NE14].

Am Bekanntesten und Weitverbreiteten in diesem Bereich ist allerdings das in den Untersuchungen bereits angesprochene Scratch[3]. Diese Anwendung soll es Nutzern ermöglichen Programmieren zu lernen, während sie für sich selbst bedeutende Projekte entwerfen. Dabei richtet sich diese Entwicklungsumgebung primär an Kinder und Jugendliche im Alter von acht bis sechzehn Jahren. Mithilfe der Scratch Programming Language können zum Beispiel animierte Geschichten, Spiele, Grußkarten oder Simulationen erstellt werden. Scratch verwendet dabei eine sogenannte *„visual blocks language“* [Mal+10].
Diese Sprachen sollen von Lehrern verwendet werden, um neue Versionen von beliebten Programmiersprachen für ihre Schüler verfügbar zu machen. Hierbei sollen sie zwei Probleme lösen, welche für Schüler auftreten. Zum einen das Merken von Kommandos und zum anderen das Merken der Syntax der jeweiligen Programmiersprache [Blo18].
Da Scratch auf einer Blocks Language basiert, wird die Funktionalität des Programms über Puzzle-Teile, welche Code-Ausschnitte darstellen, zusammengestellt. Dabei passen nur bestimmte Teile aneinander, womit eine korrekte Syntax gewährleistet ist.
Die Scratch Blöcke für die Programmierung basieren auf Blockly [MIT16]. Blockly ist eine von Google entwickelte Bibliothek zur Erstellung von Editoren für die visuelle Programmierung [Goo18b]. Scratch ist sowohl über das Internet auf einem Server des MIT [MIT18d] verfügbar, als auch in einer Offline-Version [MIT17] zum Herunterladen. Die aktuelle Scratch Version ist 2.0, wobei die online Version im Browser unter der Verwendung des Adobe Flash Players läuft [MIT16].
Scratch gibt die Möglichkeit gibt über die „Online Community“ Projekte zu teilen und über Foren mit andern zu diskutieren [MIT16]. Im Editor können Projekte somit über den „Share“-Button oben rechts auf der Website geteilt werden. Geteilte Projekte können daraufhin von an deren Nutzern ausprobiert und sogar weiterentwickelt werden. Weiterentwickelte Projekte werden als Remix

[3] Zu finden unter: [MIT16]

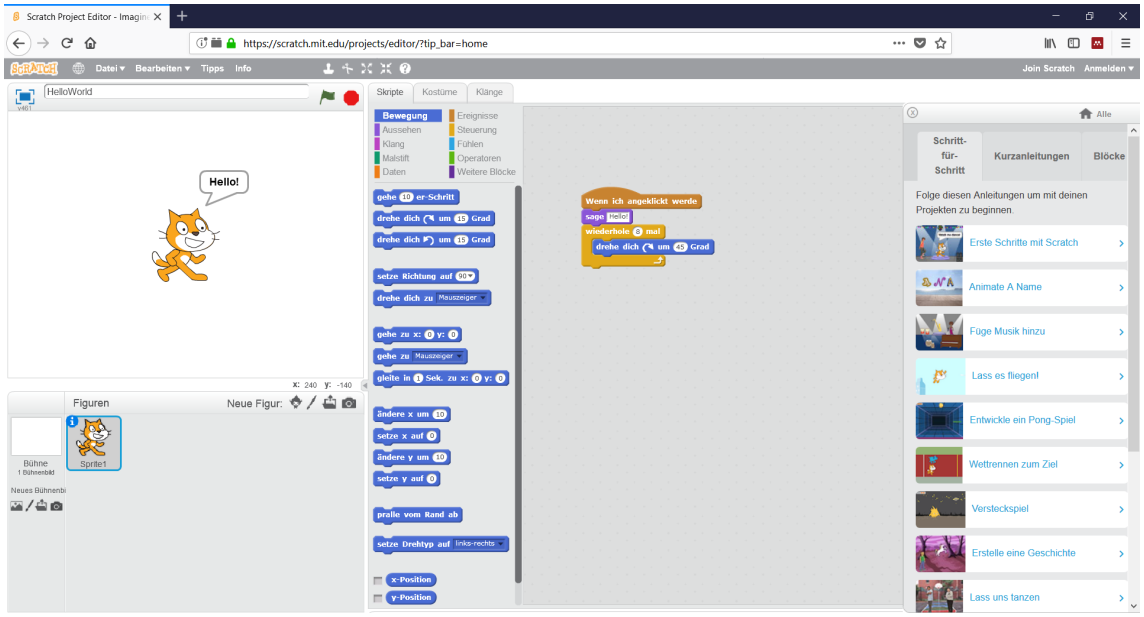



**Abbildung 4.3:** Benutzeroberfläche von Scratch[4]

bezeichnet. Das Entwicklerteam sieht es als wichtig an Projekte zu teilen und modifizierbar zu machen, da das beim Lernen und der Ideenfindung hilft [MIT16].
Lehrer, welche diese soziale Komponente für den Unterricht nicht verwenden wollen, werden darauf verwiesen den Offline-Editor zu verwenden. Außerdem hat nur unter der Verwendung des Offline-Editors oder der alten Scratch Version niemand Zugriff auf unveröffentlichte Projekte, da das Scratch-Team selbst auch auf die unveröffentlichten Projekte zugreifen kann [MIT16].
In Abbildung 4.3 ist ein Screenshot der online Version der Anwendung zu betrachten. Rechts oben im Fenster kann man die Auswirkungen des Codes sehen, darunter kann man die zu verwendenden Figuren für das Programm wählen. Mittig sind die Blöcke für die Programmierung zu finden, welche per Drag-and-Drop in das linke Fenster gezogen werden können, um den Code zu bilden. Auf dem Bild ist Code zu sehen, welcher bei einem Klick auf die Katzenfigur weitere Aktionen auslöst. So „sagt“ die Katze in Form einer Sprechblase „Hallo!“ während sie sich einmal im Kreis dreht.
Wie auf dem Screenshot der Anwendung zu sehen, ist die Entwicklungsumgebung auf deutsch verfügbar. Insgesamt steht Scratch auf mehr als vierzig Sprachen zur Verfügung, dazu gehören neben Englisch und Deutsch, zum Beispiel auch Spanisch, Italienisch, Bulgarisch, Rumänisch und Finnisch [MIT16].

Betrachten wir nun die oben vorgestellten Entwicklungsplattformen kritisch, so fällt auf, dass bei der ersten Anwendung von RAPTOR, die Verwendung nicht sehr intuitiv ist. Außerdem wird trotz der visuellen Programmierung zusätzlich ein textueller Teil benötigt, welcher voraussetzt, dass gewisse Grundkenntnisse über Befehle vorhanden sind. Hier haben Alice und Scratch aufgrund ihrer Blöcke einen Vorteil, da man sich schnell einen Überblick über alle möglichen Befehle schaffen kann, was die Anwendungen intuitiv macht. Wenn man betrachtet, was mit Hilfe von Alice und Scratch erstellt werden kann, erkennt man eine große Gemeinsamkeit. Mit beiden Programmen

[4] Scratch wird von der Lifelong Kindergarten Group am MIT Media Lab entwickelt. Es steht kostenfrei unter http://scratch.mit.edu zur Verfügung

werden Animationen generiert. Dies gibt den Nutzern eine besondere Möglichkeit das Ergebnis ihrer Arbeit zu betrachten, im Gegensatz zu den Textausgaben von RAPTOR. Besonders bei Scratch fällt die ansprechende Nutzeroberfläche auf, aufgrund der Farbcodierung der Programmierblöcke entsteht.

Es gibt verschiedene Scratch-ähnliche visuelle Programmiersprachen, wie zum Beispiel Ro-Block [FR18] und Pocket Code [Sla14] oder sogar Derivate. Da allerdings Smartphones für Kinder und Jugendliche sehr spannend sind, ist der Ableger App Inventor[5] besonders interessant, da mit dessen Hilfe Apps für Android Geräte programmiert werden können. So war schon bei der Entwicklung des App Inventors von der Motivationskraft der Smartphones auf die Bildung die Rede [Wol+14]. Auch in der Untersuchung von Nikou, welche die Motivation im Laufe eines Kurses in Bezug auf die Verwendung von Scratch und dem App Inventor vergleicht, wird der Bezug des App Inventors zu Smartphones als Grund dafür genannt, dass die Schüler Wert in ihren Aufgaben in der Arbeit mit der Anwendung sehen. Dieser Wert ist im Laufe des Kurses höher gestiegen, als bei der Verwendung von Scratch [NE14].
Dieser wird im Folgenden deshalb genauer betrachtet, da der Erfolg eines solchen E-Learning Systems besonders hoch zu sein scheint.

[5] Zu finden unter: [Mas18a]

# 5 App Inventor 2

Um den App Inventor im Verlauf der Arbeit erweitern zu können, gibt dieses Kapitel eine Einführung in die Arbeit mit dem App Inventor 2 (AI2) und seine technischen Grundlagen.
Der AI2 ist eine visuelle Entwicklungsplattform für Smartphone-Applikationen, um genau zu sein für Android-Smartphones.
Ursprünglich war der App Inventor ein Projekt von Google, welches 2012 von einem Team beim MIT übernommen wurde. Seitdem wird diese neue Version des App Inventors als AI2 bezeichnet [Wol+14].
Die Anwendung basiert auf einer visuellen Programmiermethode, welche Blöcke verwendet, die sich auch bei Kindern als erfolgreich herausgestellt hat. Die Erstellung von Apps funktioniert über die Anwendung nach dem Drag-and-Drop Prinzip, ähnlich zu dem in Abschnitt 4.2 beschriebenen Scratch. Hierbei wird das User Interface der App über ein web-basiertes Graphical User Interface (GUI) erstellt, welches *Designer* genannt wird und das Verhalten der App wird über sogenannte „Blöcke“ im *Blocks Editor* ergänzt. Dabei werden die Blöcke wie Puzzle-Teile aneinander gehaftet [Wol+14].
In Abbildungen 5.4 bis 5.7 sind sowohl der Designer als auch der Blocks Editor zu sehen, welche in Abschnitte 5.1 und 5.2 noch genauer beleuchtet werden.
Da der App Inventor frei verfügbar und online über jeden Browser erreichbar ist, kann er von jedem verwendet werden. Somit wird er nicht nur von professionellen Programmierern, sondern auch vermehrt von Menschen ohne Programmiererfahrung verwendet. Des Weiteren ist auch die Verwendung eines Smartphones nicht notwendig, da die Anwendung einen Android Emulator enthält [Wol+14]. Will man jedoch seine App parallel zum Programmieren testen, kann man auf seinem mobilen Gerät den MIT App Inventor Companion [1] verwenden, welcher als App im Google Play Store erhältlich ist, da dieser direktes Testen ermöglicht [Mas18a].
Des Weiteren ist die visuelle Programmiersprache des App Inventors mit der (in Kapitel 4 bereits erwähnten) MIT Scratch Programmiersprache verwandt [Wol+14], womit er auch für Anfänger besonders geeignet ist.

Zur Veranschaulichung der Funktionalität wird im Folgenden der Bau einer kleinen Applikation beschrieben. Die App speichert Eingaben des Nutzers durch Klicken eines Buttons, in eine Liste. Der dafür zuständige Teil der Applikation ist in Abbildung 5.1 zu sehen. Hier ist in einem Textfeld „Hello World“ eingegeben worden. Durch Drücken des Buttons wird neben dem Speichern außerdem das Abspielen eines kurzen Tons ausgelöst und zu einem neuen Screen gewechselt. In diesem Screen, welcher in Abbildung 5.2 zu sehen ist, wird der Inhalt der Liste in Form eines sogenannten ListPickers angezeigt. Dies ist zunächst der zu sehende Button, welcher auf einen Klick die Liste mit den Eingaben des Nutzers anzeigt. Die geöffnete Liste ist in Abbildung 5.3 zu betrachten. Wählt

[1] Zu finden im Google PlayStore: [Goo18c]

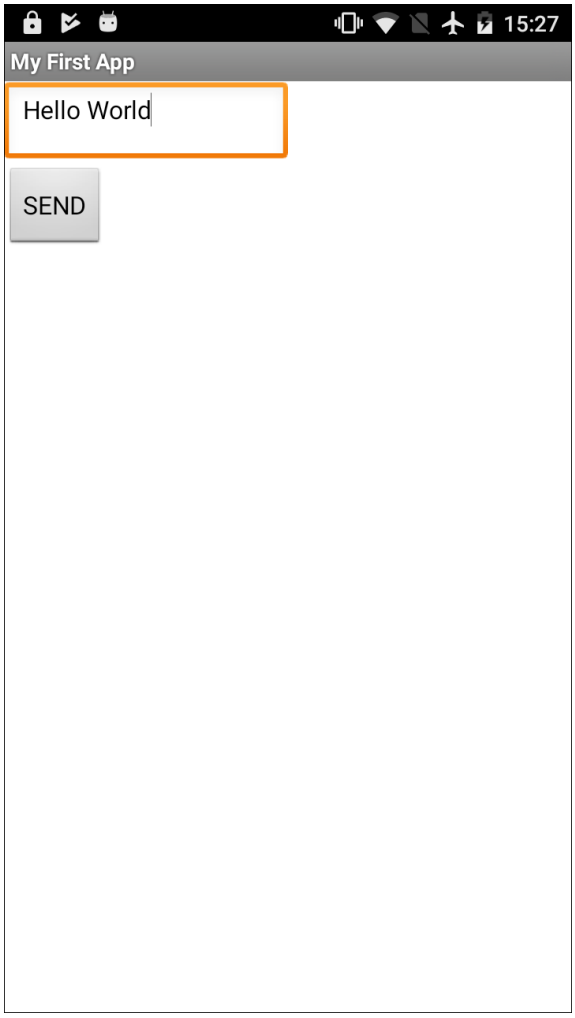



**Abbildung 5.1:** Startschirm der einfachen App mit Textfeld und Knopf

der Nutzer eines der Listenelemente aus, so kehrt er zum ersten Teil der App zurück und kann neue Eingaben tätigen. Mithilfe dieser Applikation soll in den nächsten beiden Unterkapiteln besprochen werden, wie der AI2 aufgebaut ist und wie man eine kleine App schreiben kann.

## 5.1 Designer

Nach der Erstellung des Projekts im Hauptmenü des App Inventors wird man in den Designer-Bildschirm weitergeleitet. Hier kann man mittels Drag-and-Drop eine GUI erstellen. Das ist in Abbildung 5.4 zu sehen. In dem Beispiel soll zuerst eine TextBox erstellt werden. Eine TextBox ist ein Textfeld, durch das der Nutzer Text-Eingaben tätigen kann. Im linken Bereich sind die verfügbaren Komponenten zu finden. Zu den Komponenten gehören GUI Elemente, wie Buttons oder CheckBoxen, aber auch unsichtbare Komponenten, wie Sensoren oder Datenbanken. Hat man eine Komponente ausgewählt, kann man sie einfach auf den Bereich in der Mitte mit dem Smartphone ziehen, denn hier wird die Oberfläche für die App gestaltet. Komponenten ohne graphische

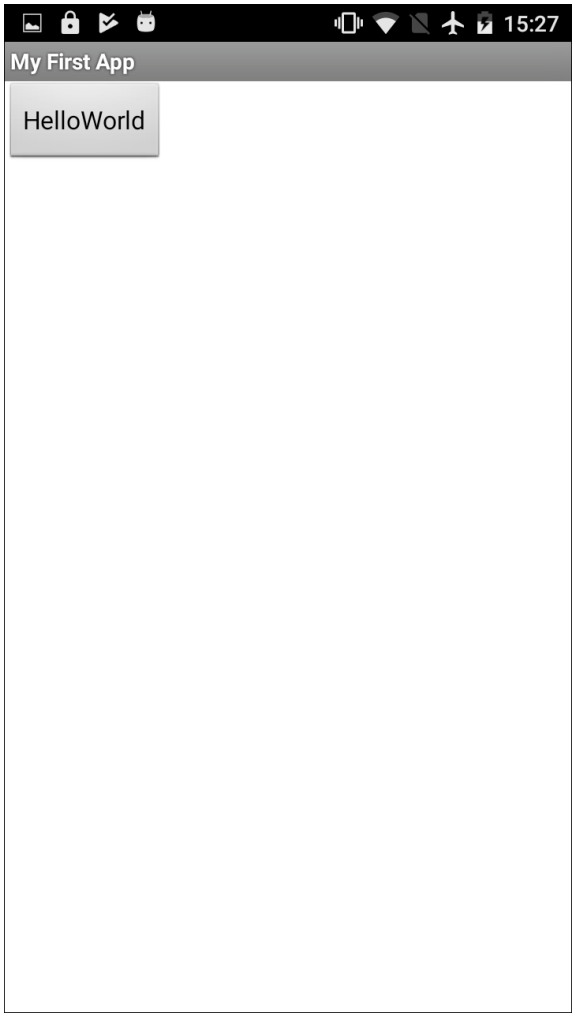



**Abbildung 5.2:** Zweiter Screen mit einem ListPicker-Button

Repräsentation erscheinen unterhalb des „Smartphones“ als „nicht-sichtbare“ Komponente, wie der Player zum Abspielen von Tönen in Abbildung 5.4. Möchte man die grafischen Elemente anders anordnen als untereinander, gibt es die Möglichkeit, in der Kategorie Layout, verschiedene Gruppierungen auszuwählen. So kann man Elemente nebeneinander, übereinander, in Form einer Tabelle, also als Viererblock, oder auch in einem Bereich zum Scrollen anzuordnen.
In den beiden Kästen rechts sind erst die verwendeten Komponenten („Components“) aufgelistet und daneben die Eigenschaften der gerade unter „Components“ ausgewählten Komponente („Properties“). Unter „Properties“ können alle Eingenschaften der Komponente bearbeitet werden. Für die App wird neben der TextBox noch ein Knopf benötigt. Dieser kann genauso über Drag-and-Drop hinzugefügt werden. Die Beschriftung des Knopfs soll in „SEND“ umgeändert werden. Hierzu wählt man unter „Components“ den Knopf aus und ändert unter „Properties“ die Eigenschaft namens „Text“ ab. Diese Änderung ist in Abbildung 5.4 mit einem blauen Rechteck markiert. Für die spätere Funktion, die vom Nutzer gefüllte Liste anzeigen zu können, wird außerdem ein zweiter Screen, welcher in Abbildung 5.5 zu sehen ist, mit einem ListPicker benötigt. Ein Screen zeigt die GUI an, will man etwas anderes darstellen, braucht man einen zweiten Screen. Der ListPicker ist ein Knopf, welcher

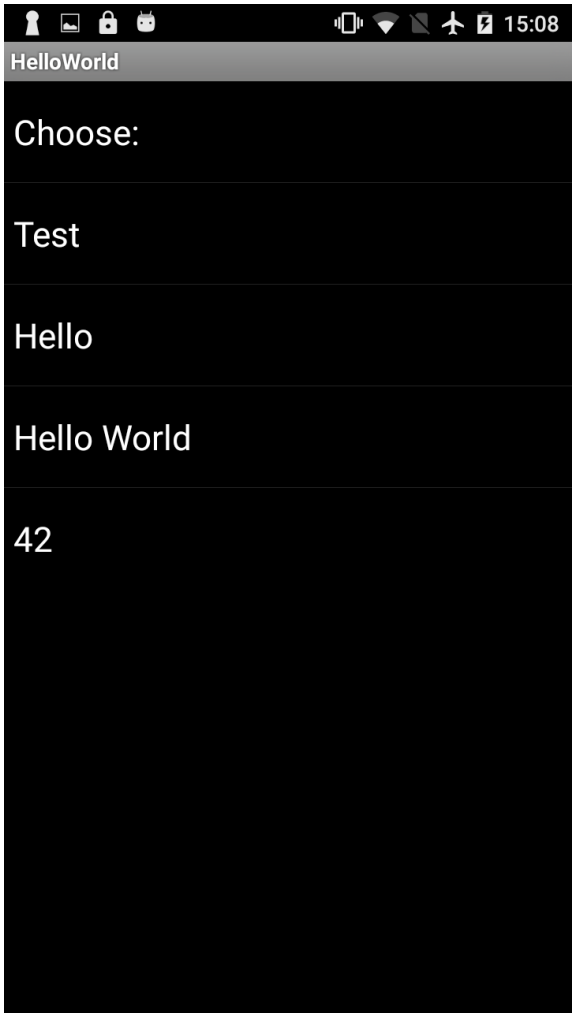



**Abbildung 5.3:** ListPicker zeigt nach Touch auf den Button den Inhalt der Liste an

eine Liste anzeigt, aus der der Nutzer ein Item auswählen kann.
Jetzt hat man die grafische Oberfläche der App. Um die Logik hinzuzufügen muss der Blocks Editor geöffnet werden.

## 5.2 Blocks Editor

Wie auf den beiden Bildern in Abbildungen 5.6 und 5.7 zu erkennen, besteht der Code aus verschiedenfarbigen Puzzle-Teilen, die auch nur an bestimmten anderen Teilen haften.
Dabei spricht man von einer sogenannten Block Language. Wie bereits zuvor im Zusammenhang mit Scratch erwähnt, sollen solche Sprachen von Lehrern verwendet werden, um neue Versionen von beliebten Programmiersprachen für ihre Schüler verfügbar zu machen [Blo18]. Hierbei sollen sie zwei Probleme lösen, welche für Schüler auftreten: Zum einen das Merken von Kommandos und zum anderen das Merken der Syntax der jeweiligen Programmiersprache [Blo18]. Die Puzzle-Teile stellen dabei die Instruktionen dar und bilden zusammengesetzt das Skript für die Anwendung [MIT18c].

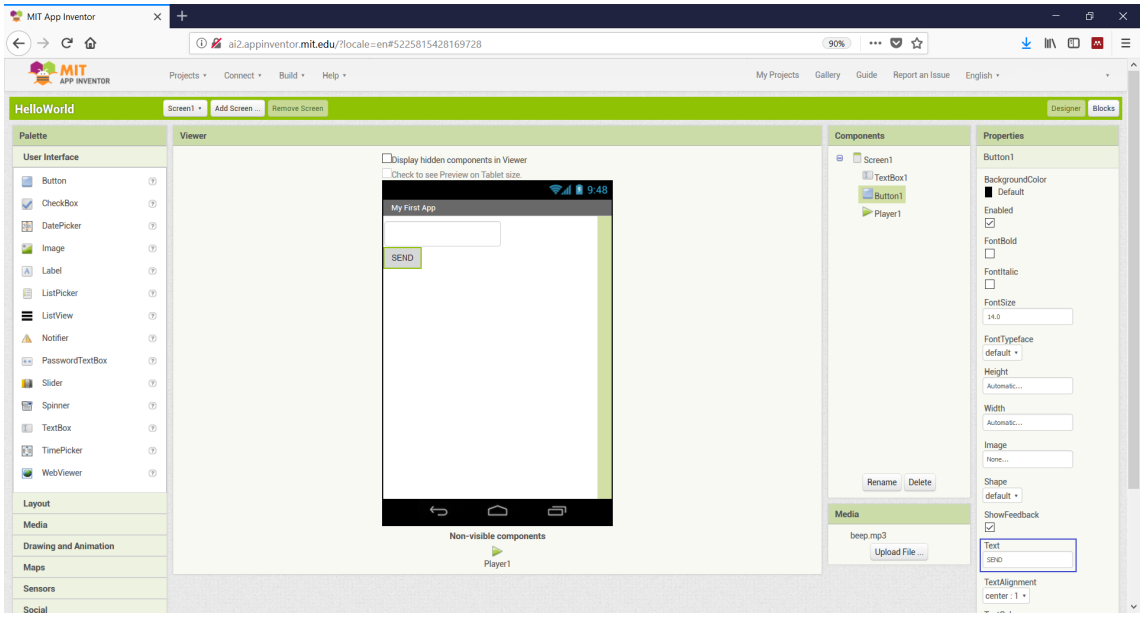



**Abbildung 5.4:** Screenshot des Designers des ersten Screens mit der TextBox, dem Button und dem Player (Startbildschirm)

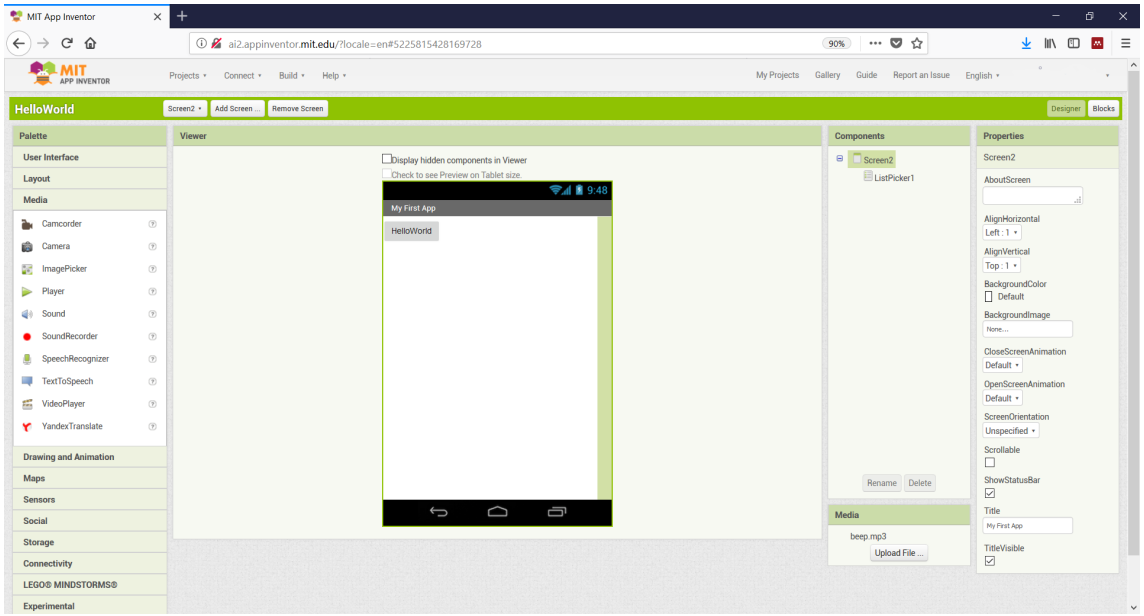



**Abbildung 5.5:** Screenshot des Designers des zweiten Screens mit dem ListPicker

Im Blocks Editor sind im linken Bereich der Anwendung die verwendbaren Code-Blöcke in verschiedene Kategorien unterteilt. Die Kategorien enthalten Kontrollstrukturen, logische Operationen, String-Operationen, mathematische Operationen, Operationen für Listen, Farben, Variablen und zur Erstellung und dem Aufruf von Prozeduren. Neben diesen allgemeinen Blöcken, welche in jedem Programm verwendbar sind, gibt es auch Blöcke für die Komponenten aus dem Designer, welche nur angezeigt werden, sollten die entsprechenden Komponenten im Designer hinzugefügt worden sein.

In Abbildung 5.6 ist der Code für den Startbildschirm der App zu sehen. Hier kann man die Färbung und Form der Kommandos betrachten. So sind zum Beispiel die Blöcke für das Erstellen und bearbeiten einer Liste blau, die Puzzle-Teile, welche sich auf Variablen beziehen sind orange und alle Kontrollstrukturen sind gelb eingefärbt. Die Form der Teile bestimmt sich durch ihre Funktion, sodass nur Teile zusammen geklickt werden können, die syntaktisch korrekt sind. Auch Teile, die passend aussehen aber nicht den passenden Rückgabewert haben, können nicht aneinander geheftet werden. Als Beispiel kann man Abbildung 5.7 betrachten. Die if-Bedingung muss einen Wahrheitswert angeheftet bekommen, allerdings wird trotz der passenden Form eines Teils, vom AI2 darauf geachtet, dass ein falscher Wert nicht angebracht werden kann. So kann das pinkfarbene Puzzle-Teil, welches einen String mit dem Wert „Screen1“ enthält, trotz passender Form nicht an die if-Bedingung angeheftet werden.

Im Blocks Editor wird die Logik ereignisorientiert erstellt. Tritt also ein bestimmtes Ereignis ein, so soll die angegebene Reaktion ausgeführt werden.
Betrachten wir deshalb nun beispielhaft den Code in Abbildung 5.6 für den Klick auf den Button. Dabei handelt es sich um einen onClickListener speziell für den Button, der bei einem Klick auf „Button1“ testet, ob der Nutzer etwas in die TextBox geschrieben hat. Sollte das der Fall sein wird erst überprüft, ob der Screen einen Startwert hat, falls ja, wird der Startwert, der eine Liste enthält, zu „list“ hinzugefügt (den Startwert erhält Screen1 nur über Screen2, was in Abbildung 5.7 zu erkennen ist). Ist dies nicht der Fall, wird als erstes Element der String „Choose:“ in die Liste gespeichert. Daraufhin wird die Eingabe des Nutzers in die Liste geschrieben und ein neuer Screen mit dem Startwert „list“ wird geöffnet. Im Vergleich dazu ist in Listing 5.1 der onClickListener für eine fast identische App mit gleicher Funktionalität zu sehen. Der Ausschnitt ist nur fast identisch, da der Code des App Inventors zu keinem Zeitpunkt in Java-Code umgewandelt wird. Bei der Kompilation hat das Programm lediglich eine Repräsentation in der Young Android Intermediate Language (YAIL), was in Abschnitt 5.3 näher erläutert wird [Sch+14]. Zu der in Android Studio geschriebenen App besteht der größte Unterschied darin, dass es in Android keinen ListPicker gibt und dass der Test über die Existenz eines Startwertes, wie in Listing 5.1 zu erkennen, über eine zusätzliche Variable namens `testIntent` gemacht werden muss.

## 5.3 Technische Grundlagen

In diesem Kapitel werden erst die Infrastruktur und dann die verschiedenen Teile des App Inventors betrachtet, da der App Inventor selbst aus mehreren Unterprojekten besteht.
Der AI2 ist ein Java Service, welcher seit der Übernahme durch das MIT nicht mehr auf der Infrastruktur von Google, sondern auf Google App Engine (GAE)[2] läuft [MIT15]. GAE ist eine

[2] Zu finden unter: [Goo18a]

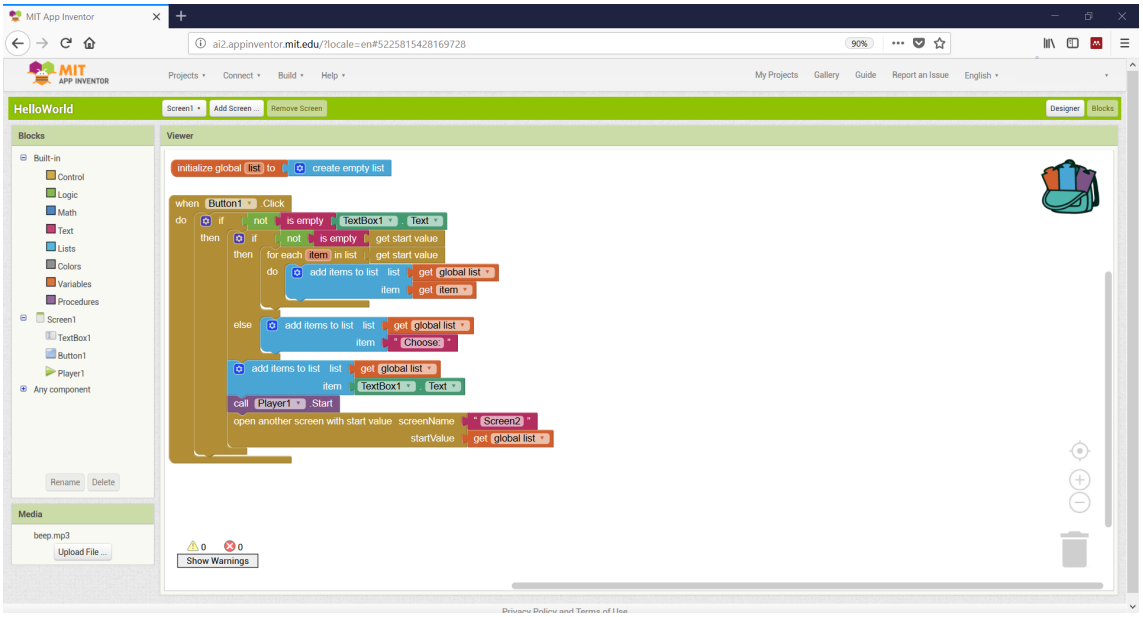



**Abbildung 5.6:** Screenshot des Blocks Editors mit dem Code für die TextBox, Liste, den Button und den Player (Startbildschirm)

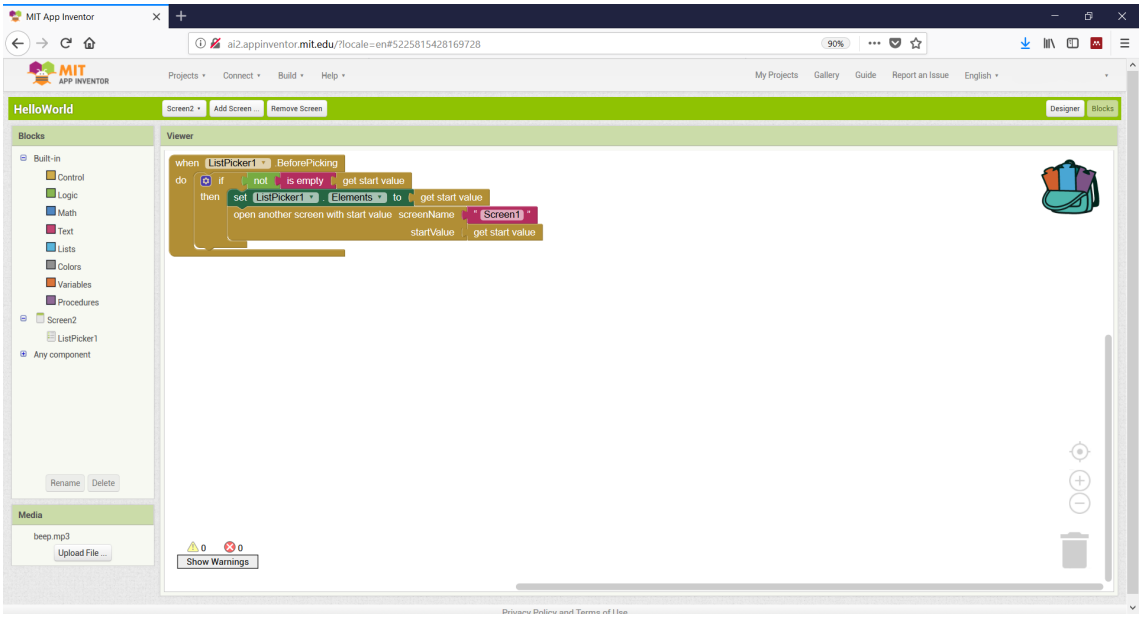



**Abbildung 5.7:** Screenshot Blocks Editors mit dem Code für den ListPicker

**Listing 5.1** Ausschnitt des onClickListeners als Android-Code

```
button1.setOnClickListener(new View.OnClickListener() {
  public void onClick(View v) {
    String text = textBox.getText().toString();
    if(!text.isEmpty()) {

      //test if Intent of Main2Activity already exists
      if(testIntent != null){

        list.addAll((ArrayList<String>)getIntent()
          .getSerializableExtra("arrayList"));

      } else {
        list.add("Choose:");
      }

      list.add(text);

      try {

        player = MediaPlayer.create(MainActivity.this, R.raw.beep);
        player.start();
      }
      catch (NullPointerException e) {

      }

      Intent i = new Intent(MainActivity.this, Main2Activity.class);
      startActivity(i.putExtra("arrayList", list));
   }
  }
});
```

Cloud-Computing Plattform, unter deren Verwendung unter anderem Java-Programme Daten auf Servern von Google ausführen oder verwalten können [SD18]. Der App Inventor verwendet GAE zusammen mit Google Web Toolkit (GWT)[3] [SD18]. GWT ist ein Open Source Toolkit für die Entwicklung und Optimierung von komplexen, browser-basierten Anwendungen [GWT18]. Das darin enthaltene GWT SDK enthält Java APIs und Widgets mit deren Hilfe AJAX Anwendungen in Java geschrieben werden können, die wiederum in JavaScript kompiliert werden und dann in jedem Browser laufen [GWT18]. Die Verwendung von GWT erlaubt das Programmieren von Client-Server Anwendungen in Java, ohne auf die Details von Remote Procedure Calls (RPCs) achten zu müssen. Hierbei kompiliert GWT den Code des Clients in JavaScript, welches im Web Browser läuft [SD18]. Die RPCs wiederum laufen als Java-Code auf dem Server, wobei die Kommunikation über HTTP abläuft [SD18].
Der App Inventor verwendet GWT, um den App Inventor Client und Server zu erstellen [SD18]. Auf der Seite des Clients wird der Code für das Frontend über die GWT Client Bibliothek im

[3] Zu finden unter: [GWT18]

Browser ausgeführt, nachdem GWT den Code in JavaScript umwandelt hat [SD18]. Serverseitig läuft das Backend auf der GWT Server Bibliothek, welche zur Datenspeicherung die Objectify API verwendet, als Google App Engine Service [SD18].

Der folgende Teil des Textes beschäftigt sich mit der Struktur des App Inventor Projekts. Auf dieser Grundlage kann in Abbildung 5.8 später noch einmal betrachtet werden, wo die für die Ausführung des App Inventors wichtigen Teile der Anwendung ausgeführt werden. Dabei besteht der App Inventor aus acht verschiedenen Unterprojekten, die sich alle im Hauptverzeichnis des AI2 unter „appinventor“ befinden: aimerger, aimerger_for_appinv_classic, aiplayapp, appengine, blocklyeditor, buildserver, common und components. Die weiteren Verzeichnisse sind misc, rendezvous, docs und lib. Wobei docs und lib statische Dateien, wie Tutorials und verschiedene externe Bibliotheken, enthalten [SD18]. Die für die spätere Ausführung wichtigen Teile des App Inventors sind die Ordner aiplayapp, appengine, blocklyeditor und buildserver.
Der aimerger ist dafür gedacht zwei von Nutzern erstellte Programme zu kombinieren [Fee+15]. Das Pendant dazu für den App Inventor Classic, also den ursprünglichen App Inventor bevor er vom MIT übernommen worden ist [Mas18a], ist aimerger_for_appinv_classic.
Weiter ist aiplayapp der MIT App Inventor Companion, welcher ein Interpreter ist, der auf einem Emulator oder einem mit dem Computer verbundenen mobilen Gerät läuft [SD18]. Das Unterprojekt appengine ist eine GWT Anwendung. Diese stellt dem Browser des Nutzers den JavaScript-Code für den Designer bereit. Außerdem werden serverseitige Funktionalitäten wie das Speichern und Wiederherstellen von Projekten und das Senden einer Compilierungsanfrage an den Buildserver über dieses Unterprojekt umgesetzt [SD18].
Der blocklyeditor enthält den Code für den in Abschnitt 5.2 beschriebenen Blocks Editor, welcher im Browser eingebettet ist [SD18]. Hierfür wird eine von Google entwickelte Open Source Bibliothek namens Blockly[4] verwendet [SD18].
Blockly ist eine Bibliothek zum Bauen von Editoren für visuelle Programmierung. Hierbei fügt Blockly der Applikation einen Editor hinzu, welcher Code-Konzepte als miteinander verknüpfbare Blöcke darstellt [Goo18b]. Außerdem generiert Blockly syntaktisch korrekten Code aus der visuellen Repräsentation, welcher schließlich von der App verwendet werden kann [Goo18b].
Eine weitere Komponente ist der buildserver, welcher ein http Server/Servlet ist [SD18]. Auch der buildserver verwendet Blockly [SD18]. Der Buildserver wird dafür verwendet, ein mit dem App Inventor geschriebenes Programm zu kompilieren und dabei in eine sogenannte Android Package (APK)-Datei zu verpacken [Sch+14]. Eine APK-Datei enthält eine Applikation für Android und muss manuell installiert werden [Fil18]. Zur Erstellung der APK-Datei werden die Komponenten und Blöcke vorübergehend in YAIL umgewandelt [Sch+14; Spe18]. YAIL-Programme bestehen aus S-Expressions, welche mit der Hilfe eines Scheme Compilers oder Interpreters übersetzt werden können [Spe18].
Dabei steht S-Expression für „symbolic expression“ und wird zum Beispiel in Lisp und Variationen von Lisp wie Scheme verwendet, um das Programm und die Daten zu repräsentieren [Com17]. YAIL ist eine in Kawa definierte Sprache [Sch+14]. Wobei Kawa ein in Java geschriebenes Framework zur Implementierung von höheren und dynamischen Programmiersprachen ist, die in Java-Bytecode kompiliert werden [KAW18b]. Die Kawa Scheme Sprache ist eine Erweiterung von Scheme, welche sich in der Familie von Lisp befindet. Außerdem ist Kawa eine Allzwecksprache, die auf der Java Plattform läuft. Sie soll die Vorteile von dynamischen Skript-Sprachen und von traditionellen kompilierten Sprachen kombinieren [KAW18a]. Dabei kann die Sprache nützlich für

[4] Zu finden unter: [Goo18b]

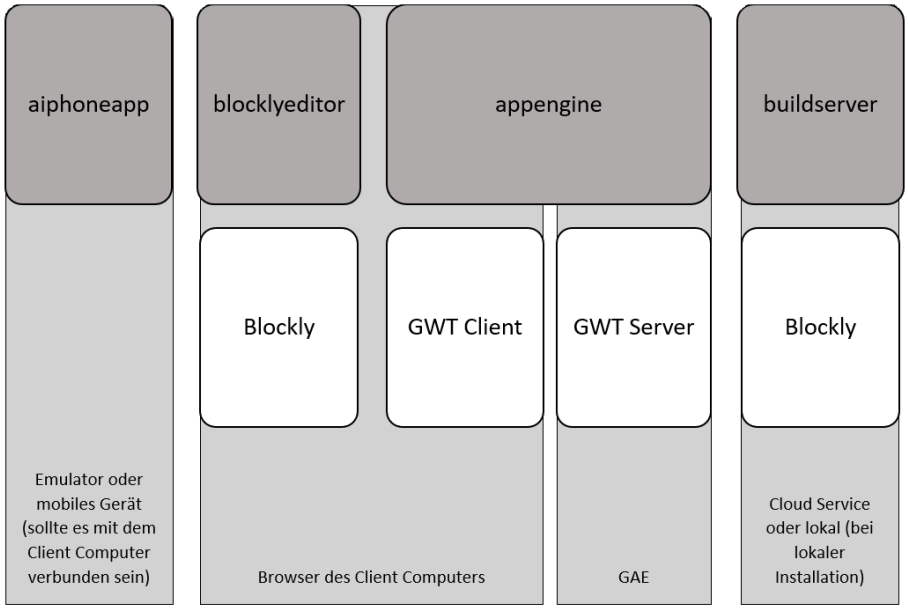



**Abbildung 5.8:** Übersicht über die Projekte und wo sie ausgeführt werden (basierend auf [SD18])

die Implementierung anderer Programmiersprachen auf einer Java Plattform sein [KAW18a].
Somit wird die visuelle Block Language durch einen Compiler übersetzt, welcher das Kawa Language Framework und den Kawa's Dialekt der Scheme Programmiersprache verwendet [Wol+14]. Der generierte YAIL-Code wird daraufhin erst zu Java Bytecode und dann zu Dalvik Virtual Machine Bytecode kompiliert [Sch+14; SD18]. Wobei die Dalvik Virtual Machine eine Virtual Machine für Android ist [Jav18].
In common befinden sich von den anderen Unterprojekten verwendete Konstanten und Utility Klassen [SD18]. Das components Projekt ist vor allem für die Erweiterung des AI2 um eine Komponente interessant. Hier befindet sich Code, welcher die Komponenten des App Inventors unterstützt. Dazu gehören zum Beispiel die Annotationen (vgl. Kapitel 8) und Implementierungen [SD18].

Um das oben beschriebene etwas übersichtlicher zu gestalten sind die wichtigsten Projekte des App Inventors und wo sie ausgeführt werden, in Abbildung 5.8 noch einmal zusammengefasst zu betrachten. Hierbei beschreiben die Lanes, wo die Komponenten ausgeführt werden, die dunkelgrauen Boxen beschreiben, in welchem Unterprojekt sich der entsprechende Code für die Anwendung befindet und die weißen Boxen beschreiben die vom App Inventor verwendeten Komponenten zur Verwirklichung der Funktionen der Unterprojekte. Hierbei ist auch der zu Beginn des Kapitels beschriebene Zusammenhang zwischen GAE und GWT dargestellt.

## 5.4 Probleme des App Inventors

Da nun die technischen Grundlagen für die Bearbeitung des App Inventors bekannt sind, wird in diesem Abschnitt analysiert, welche Schritte notwendig sind, um die in Kapitel 3 aufgelisteten Anforderungen zu erfüllen und ob einige der Anforderungen vielleicht bereits erfüllt sind. Für die Durchführung dieser Evaluation wird Bezug auf die bereits durchgeführten Workshops[5] für SchülerInnen an der Universität Stuttgart genommen, bei welchen der AI2 verwendet worden ist.

**A.1 Collaborative Authoring:**
Collaborative Authoring könnte durch die gemeinsame Erstellung von Projekten unter der Nutzung des App Inventors gefördert werden. Die gemeinsame Erstellung von Projekten ist unter der Verwendung der App Inventor Instanz des MIT nicht möglich. Wird der App Inventor lokal auf einem Server installiert, so teilen sich alle Nutzer den selben Speicherplatz, da keine Authentifizierung stattfindet [MIT15], womit eine gemeinsame Zusammenarbeit an einem Projekt möglich wäre.

**A.2 Verfügbarkeit:**
Da der App Inventor vom MIT zur Verfügung gestellt wird und somit auf deren Servern aufgesetzt ist, ist die Durchführung der Workshops davon abhängig, dass die Server des MIT Erreichbar sind und zur Verfügung stehen. Hierfür gibt es bei Scratch, wie in Abschnitt 4.2 erläutert, bereits eine Lösung, da die Anwendung nicht nur online, sondern auch als Download für den offline Gebrauch verfügbar ist. Anstatt eine offline Komponente zu erstellen, gibt es beim App Inventor, wie oben vermerkt die Möglichkeit der Installation einer lokalen App Inventor Instanz. Mithilfe einer solchen Instanz auf einem Server der Universität besteht keine Abhängigkeit zur Instanz des MITs mehr. Somit kann das Verfügbarkeitsproblem auch gelöst werden, indem eine lokale Instanz auf einem Server der Universität installiert wird.

**A.3 Verständlichkeit:**
Die in Kapitel 3 angesprochenen Schwierigkeiten mit der Verständlichkeit der Sprache bei Workshops beziehen sich auf die oben bereits erwähnten Veranstaltungen, welche unter Verwendung des App Inventors stattgefunden haben. Im Gegensatz zu Scratch verfügt der App Inventor im Moment nur über eine Auswahl von zwölf verschiedenen Sprachen. Dieses Problem kann also nur gelöst werden, indem bei der Anwendung eine weitere Sprachunterstützung hinzugefügt wird.

**A.4 Flexibilität:**
Die Flexibilität in Bezug auf verschiedene Themen, muss auch beim AI2 behandelt werden. Bei den Veranstaltungen für Schüler an der Universität ist aufgefallen, dass bei der Verwendung von Robotern, dies mit dem App Inventor zwar gut und einfach funktioniert, sofern der App Inventor eine Komponente für die Verwendung und Steuerung des Roboters besitzt. Sollte die Anwendung jedoch keine entsprechende Komponente für den gewünschten Roboter besitzen, so ist die Einbindung durch Nutzung des Block Editors sehr aufwendig.

**A.5 Adaptives Lernen:**
Adaptives Lernen steht in engem Zusammenhang mit der Flexibilität und kann somit stattfinden, indem Erweiterungen beziehungsweise Komponenten für den App Inventor erstellt werden. Gerade bei Workshops, wie dem Girls'Day[6], ist die Altersspanne der Teilnehmer groß und somit auch der

[5] siehe [Ber+15; Ste+13]
[6] siehe [Kom18]

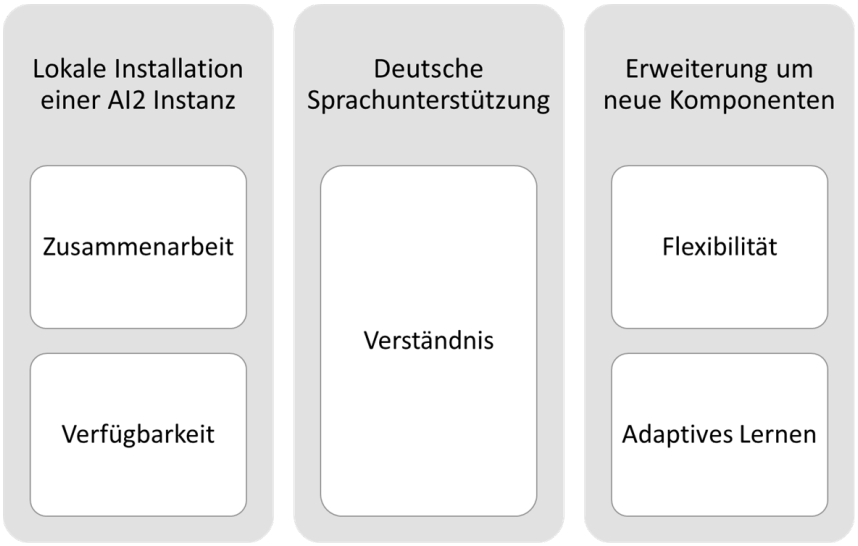



**Abbildung 5.9:** Anforderungen an den AI2 und welche Probleme durch Erfüllung dieser Anforderungen gelöst werden

Wissensstand unterschiedlich. Dieses Problem könnte man beim App Inventor lösen, indem man ihn um ein Modul erweitert, welches von den jüngeren Schülern beim Lösen der Aufgabe verwendet werden darf. Die Schüler mit höherem Wissensstand müssen ohne diese Komponente arbeiten.

Im Rahmen dieser Bachelorarbeit wird daher der AI2 in drei Schritten erweitert: Zunächst wird eine lokale Instanz des App Inventors aufgesetzt, um die Verfügbarkeit zu erhöhen und es den Schülern zu ermöglichen zusammen an einem Projekt zu arbeiten (Kapitel 6). Daraufhin folgt die Ergänzung der deutschen Sprachunterstützung, um die Sprachprobleme der Schüler zu beseitigen (Kapitel 7). Schließlich folgt die Erweiterung um zwei neue Komponenten, auf deren Grundlage gezeigt werden soll, dass sowohl die Flexibilität des Inhalts, als auch adaptives Lernen mithilfe des App Inventors gewährleistet werden kann, sollten entsprechende Komponenten hinzugefügt werden (Kapitel 8). Für die bessere Übersichtlichkeit welches Problem durch welche Anforderung an die Erweiterung des App Inventors gelöst wird, ist dies in Abbildung 5.9 visualisiert.

# 6 Lokale Installation einer App Inventor Instanz

In diesem Kapitel wird beschrieben, wie eine lokale Installation des App Inventors aufgesetzt werden kann, um dadurch die zuvor beschriebenen Anforderungen der Zusammenarbeit und Verfügbarkeit erfüllen zu können. Das MIT selbst hat bereits eine Anleitung[1] veröffentlicht, die beschreibt wie man eine App Inventor Instanz aus dem Quellcode erstellen kann. Hierbei gibt es dann die Möglichkeit eine lokale Installation auf dem eigenen Rechner oder eine Installation auf der App Engine Infrastruktur von Google aufzusetzen. Erfolgt die Installation auf der App Engine Infrastruktur von Google, so ist die erstellte AI2-Instanz für jeden über das Internet erreichbar. Neben der Instanz wird ebenfalls ein Buildserver benötigt, welcher die vom Nutzer erstellten Apps kompiliert und verpackt. Erfolgt die Installation lokal, kann die Maschine, auf der der App Inventor Service läuft, selbst diese Aufgabe übernehmen [MIT15].

Eine AI2-Installation besteht aus zwei Komponenten: Dem App Inventor Service (AIS) und dem Buildserver. Der AIS ist für die Ausführung der AI2-Anwendung da, während der Buildserver die im AIS erstellten Projekte kompiliert und ausführbare Anwendungen erstellt [MIT15]. Wie in Abbildung 6.1 dargestellt, können diese beiden Komponenten bei einer lokalen Installation auf einer gemeinsamen GAE Instanz ausgeführt werden [MIT15]. Aus Performanz- / Skalierungsgründen sollten diese beiden Komponenten bei einer Cloud-Installation jedoch auf zwei separate Server aufgeteilt werden (siehe Abbildung 6.2) [MIT15]. Die derzeitig verfügbare Instanz vom MIT ist eine solche, auf zwei Server aufgeteilte, Instanz. In Abbildung 6.2 ist zu erkennen, dass bei der Verwendung der Instanz des MIT eine Abhängigkeit zu GAE auf der Infrastruktur von Google besteht, da dort der App Inventor als Service[2] läuft. Weiterhin besteht aufgrund des Buildservers eine Abhängigkeit zum MIT, beziehungsweise dem Server oder dem Cloud-Service, den das MIT verwendet. Durch die Verwendung einer lokalen Instanz wird die Abhängigkeit, bezüglich der Verfügbarkeit, auf eine Maschine begrenzt. Der zweite Unterschied ist ebenfalls in Abbildungen 6.1 und 6.2 vermerkt, da die Verwendung einer lokalen Instanz dafür sorgt, dass alle Nutzer sich einen Speicher teilen [MIT15].

Für die lokale Installation einer Instanz des App Inventors werden zunächst folgende Programme benötigt [MIT15]:

- App Inventor Setup Software
- Java
- Apache Ant
- Git
- Google AppEngine Java SDK

[1] siehe [MIT15]
[2] vgl. [MIT15]

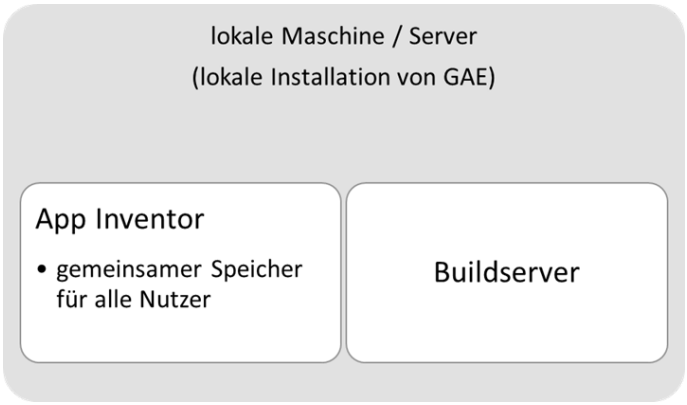



**Abbildung 6.1:** Komponenten der lokalen Installation einer App Inventor Instanz, auf einer Maschine (nötige Informationen aus [MIT15])

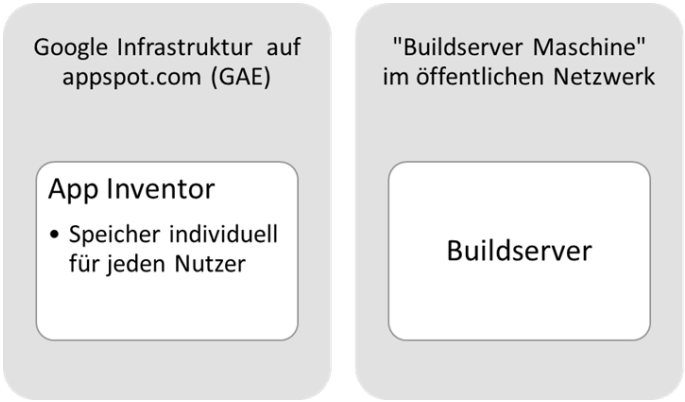



**Abbildung 6.2:** Komponenten der aktuell verfügbaren App Inventor Instanz, getrennt zwischen der Google Infrastruktur und der Maschine, auf der sich der Buildserver befindet (nötige Informationen aus [MIT15])

- Python
- PhantomJS
- Android SDK

Außerdem wird je nach Betriebssystem weitere Software benötigt. So werden für eine Installation bei Ubuntu verschiedene zusätzliche Pakete benötigt, um eine fehlende Bibliothek zu ersetzen [MIT15]. Des Weiteren werden einige Anmerkungen bezüglich der Versionen bestimmter Software gemacht. So wird empfohlen Java in der Version 1.7.0_72 zu nutzen und Apache Ant in Version 1.8.1 oder 1.8.2 [MIT15]. Im Projekt in der „README.md“ ist vermerkt, dass die Verwendung von Java 8 bei einer lokalen Installation möglich ist. Da Java 7 jedoch nicht mehr ohne Weiteres bei Oracle erhältlich ist [Ora18], könnte dies bei der Installation auf der Infrastruktur von Google zu Problemen führen.
Die Software PhantomJS wird benötigt, um Unit Tests verwenden zu können [MIT15]. Git wird verwendet, um den Quellcode des App Inventors, welcher sich im GitHub Repository[3] des MIT

[3] siehe [MIT18b]

befindet, zu clonen und die Submodule, wie Blockly upzudaten [MIT15]. Für den Build der Anwendung, den Build und das Installieren des AI Companions auf einem mobilen Gerät und das Starten des Buildservers wird Apache Ant verwendet [MIT15]. Für die Installation der mit dem AI2 generierten APKs auf ein mobiles Gerät, werden die Plattform-Tools des Android SDK benötigt, um genau zu sein das enthaltene Kommandozeilentool Android Debug Bridge (adb). Dieses Tool dient zur Kommunikation mit dem mobilen Gerät [And18a].

Nach Abschluss der Installation aller benötigten Software, müssen vor dem Starten des App Inventors und des Buildservers die Umgebungsvariablen angepasst werden, damit die installierte Software von überall aufgerufen werden kann. Dazu gehört die Erstellung einer Variable `JAVA_HOME` in der der Pfad zur Java Installation gespeichert ist, eine Variable `ANT_HOME` in der der Pfad zur Ant Installation gespeichert ist und die `PATH` Variable muss erweitert werden, sodass die Pfade zu der Installierten Software (bis auf App Inventor Setup Software) angehängt werden. Daraufhin müssen einige Kommandos in der Konsole ausgeführt werden. Dazu gehört das zuvor erwähnte Update der Submodule des App Inventors mithilfe von `git submodule update --init` [MIT15]. Ein Build kann mithilfe des Kommandos `ant` stattfinden, wobei zuvor ein geheimer Schlüssel für den AI2 unter Verwendung von `ant MakeAuthKey` generiert werden muss [MIT15]. Über das Kommando `ant clean` lassen sich unnötige Dateien aus dem Ordner entfernen, welcher die Dateien für den Build enthält [MIT15]. Nach der Ausführung dieses Kommandos findet der Build statt, wobei danach für die Ausführung der App Inventor Instanz die Datei web.xml im Verzeichnis unter appinventor/appengine/build/war/WEB-INF fehlt. Diese kann einfach wieder aus appinventor/appengine/war/WEB-INF eingefügt werden. Nach Abschluss des Builds, können der App Inventor und der Buildserver mit folgenden Kommandos gestartet werden [MIT15]: `[Pfad zum App Engine SDK]/appengine-java-sdk-[Versionsnummer]/bin/dev_appserver[.sh, bei Windows .cmd] --port=8888 --address=0.0.0.0 [Pfad zum App Inventor Quellcode]/appinventor/appengine/build/war/` und im Verzeichnis [Pfad zum App Inventor Quellcode]/buildserver muss das Kommando `ant RunLocalBuildServer` ausgeführt werden. Daraufhin ist der App Inventor komplett einsatzbereit. Will der Nutzer nun eine App erstellen und diesen Prozess parallel auf dem eigenen Gerät mitverfolgen und ausprobieren, muss der beim Build erzeugte AI Companion über das Ausführen von `ant installplay`, am Speicherort des Companions (appinventor/build/buildserver), auf das mobile Gerät gespielt werden [MIT15]. Anzumerken ist, dass für die Verbindung mit dem Smartphone, in den Einstellungen des Geräts die Entwickleroptionen aktiviert werden müssen. Des Weiteren ist zu beachten, dass unter Verwendung von Windows die aiStarter.exe in der App Inventor Setup Software nicht automatisch gestartet wird. Möchte man eine Verbindung über USB herstellen, ohne dass der aiStarter läuft, wird darauf in einer Fehlermeldung hingewiesen.

# 7 Sprachunterstützung

Dieses Kapitel beschreibt die nötigen Änderungen und Ergänzungen am Quellcode für die Unterstützung einer neuen Sprache, um so das Verständnis der Anwendung zu verbessern.
Die Übersetzungen für den Designer und die Methoden im Blocks Editor, welche Teil einer wie in Kapitel 8 erstellten Komponente sind (Dateien im appengine-Directory), und für den Blocks Editor (blocklyeditor-Directory) müssen getrennt in zwei Dateien angelegt werden.
Die Sprachdateien befinden sich somit an zwei Stellen im AI2 Projekt. Zum Einen befindet sich eine Datei unter appinventor/appengine/src/com/google/appinventor/client/ mit den Namen OdeMessages_[Kürzel für die Sprache].properties. Hier befinden sich die Übersetzungen für den Designer. Und zum Anderen ist unter appinventor/blocklyeditor/src/msg jeweils ein Ordner mit dem Kürzel der Sprache als Name angelegt, in dem sich eine Datei namens _messages.js mit den Übersetzungen für den Blocks Editor befindet. Um neue Sprachen hinzufügen zu können, müssen also zunächst diese beiden Übersetzungsdateien angelegt werden.
Will man die deutsche Sprachunterstützung hinzufügen, so benötigt man eine OdeMessages_de_DE.properties und dementsprechend einen Ordner mit dem Namen „de_de“, der die entsprechende _messages.js enthält.
Speziell die Java .properties-Datei besteht aus Schlüssel-Wert-Paaren, bei der jede Zeile aus einem Schlüssel und ihrem entsprechenden Wert besteht [Tra18]. Dies ist beispielhaft in Listing 7.1 zu sehen.
In der Datei appinventor/appengine/src/com/google/appinventor/client/TopPanel.java muss die Funktion `private String getDisplayName(String localeName)` ergänzt werden, sodass zur neu angelegten Sprache gewechselt werden kann (Listing 7.2).
Die Funktion, welche zum Wechseln der Sprache aufgerufen wird, muss nun ebenfalls in der Datei appinventor/appengine/src/com/google/appinventor/client/OdeMessages.java ergänzt werden. In dieser Datei sind Strings für alle Funktionen und Variablen enthalten. Hierbei werden in dieser Datei dann auch die default Texte (in englischer Sprache) angegeben, welche dann für das GUI verwendet werden, sofern keine andere Sprache ausgewählt wurde oder der Variable in einer der Übersetzungsdateien kein anderer Wert zugewiesen wurde.
Zwei dieser Strings sind beispielhaft in Listing 7.3 dargestellt. Betrachtet man nun Listings 7.1 und 7.3, so ist zu erkennen dass dem String `AboutScreenProperties`, welcher in Listing 7.3 zu

**Listing 7.1** Ausschnitt aus „OdeMEssages_de_DE.properties“

```
AICompanionMenuItem = AI Companion
AboutScreenProperties = ÜberDenBildschirm
AcceptConnectionMethods = AkzeptiereVerbindung
AccuracyProperties = Genauigkeit
ActionProperties = Aktion
ActivityClassProperties = Klasse
ActivityPackageProperties = Packet
```

**Listing 7.2** Ergänzungen in „TopPanel.java“

```
private String getDisplayName(String localeName){
[...]
  if (localeName == "zh_CN") {
  [...]
  } else if (localeName == "de_DE") {
    nativeName = MESSAGES.SwitchToGerman();
  }
[...]
```

**Listing 7.3** Ausschnitt aus „OdeMessages.java“

```
@DefaultMessage("AboutScreen")
@Description("")
String AboutScreenProperties();
[...]
@DefaultMessage("Deutsch")
@Description("")
String SwitchToGerman();
```

sehen ist, in Listing 7.1 eine Übersetzung zugewiesen wird, nämlich `ÜberDenBildschirm`. Unter appinventor/appengine/build/war/WEB-INF/classes/com/google/appinventor/YaClient.gwt.xml muss nun noch die Spracheigenschaft erweitert werden, sodass passend zur ergänzten Methode in TopPanel.java der Wert (in diesem Fall „de_DE“) für das Wechseln der Sprache auch verfügbar ist. (Listing 7.4)
Des Weiteren muss die Datei appinventor/blocklyeditor/src/language_switch.js ergänzt werden, damit auch der Blocks Editor übersetzt wird. Dabei muss zunächst das JavaScript File geladen werden, welches die Übersetzungen enthält, damit die Methoden, mit den Übersetzungen, in der `Blockly.language_switch` aufgerufen werden können. Dabei wird im Fall der deutschen Sprachunterstützung zum einen die Übersetzung aus der automatisch generierten Datei de.js unter appinventor/lib/blockly/msg/js/ und zum anderen die eigene Übersetzung in _messages.js aufgerufen. Der zu ergänzende Quellcode für die deutsche Sprachunterstützung ist in Listing 7.5 zu sehen.
Zuletzt fehlt lediglich der Import der verwendeten Übersetzungsfiles in der Datei appinventor/

**Listing 7.4** Ergänzungen in „YaClient.gwt.xml“

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE module PUBLIC "-//Google Inc.//DTD Google Web Toolkit 2.3.0//EN" "http
://google-web-toolkit.googlecode.com/svn/tags/2.3.0/distro-source/core/src/gwt-
module.dtd">
<module rename-to='ode'>
  [...]
  <extend-property name="locale" values="de_DE"/>
  [...]
</module>
```

**Listing 7.5** Ergänzungen in „language_switch.js“

```
goog.require('AI.Blockly.Msg.de_de')
[...]
Blockly.language_switch = {
[...]
  case 'de_DE':
          Blockly.Msg.de.switch_blockly_language_to_de.init();
          Blockly.Msg.de.switch_language_to_german.init();
          break;
```

**Listing 7.6** Ergänzungen in „ploverConfig.js“

```
'../lib/blockly/msg/js/de.js',
[...]
'./src/msg/de_de/_messages.js',
```

blocklyeditor/ploverConfig.js, welches für die Kompilation von Blockly verwendet wird (zu sehen in Listing 7.6).

# 8 Eine neue App Inventor Komponente

Um die Grundlagen für die Flexibilität des Inhalts und für adaptives Lernen zu schaffen, wird in diesem Kapitel beschrieben, wie dem AI2 neue Komponenten hinzugefügt werden können. Auch für das Hinzufügen einer Komponente stellt das MIT auf seiner Internetseite Anleitungen[1] bereit. Eine dieser Anleitungen beschreibt die Erstellung einer neuen Komponente und die Andere zeigt das Ergänzen von neuen Eigenschaften zu einer bestehenden Komponente [Fee18; Spe18]. Wie schon zuvor sind diese als Grundlage für die Arbeit verwendet worden.
Für das Hinzufügen einer neuen Komponente muss Code in Java (für den App Inventor) und für Android mithilfe des Android SDK in Java (für die Funktionen der von den Nutzern erstellten Apps) geschrieben werden. Für die Nutzeroberfläche des AI2 können ebenfalls Ergänzungen in einer css Datei notwendig werden, sollte eine grafische Komponente hinzugefügt werden.
Des Weiteren ist zu beachten, dass der AI2 schon Smartphones mit Android 2.3 (API Level 9) oder niedriger unterstützt [Fee18; Mas18b]. Das bedeutet, dass eventuell nicht alle Endgeräte den erstellten Android-Code unterstützen können, sollte ein Module verwendet worden sein, dass erst in einer späteren Android Version ergänzt worden ist. Weiter ist bei der Erstellung von sichtbaren Komponenten wichtig, dass die grafischen Elemente der Komponenten im Designer auf der Grundlage von GWT Widgets basieren [Fee18]. Dementsprechend können auf diese Weise eventuell nicht alle in Android verfügbaren, grafischen Komponenten im App Inventor korrekt visualisiert werden. Für beide dieser Punkte ist der Switch ein gutes Beispiel. Der Switch ist die neuere Version des in Abschnitt 8.1 hinzugefügten ToggleButtons, welcher ein Knopf ist, der zwischen zwei Zuständen hin und her wechseln kann. Im Gegensatz zum ToggleButton stellt der seit Android 4.0 verfügbare Switch einen Schieberegler dar [And18b]. Da der Slider aber erst ab einer späteren Android Version verfügbar ist, kann dieser nicht von allen Geräten verwendet werden, die vom App Inventor unterstützt werden. Des Weiteren gibt es in GWT kein Switch-Widget [GWT18], was bedeutet, dass dieser so nicht im Designer dargestellt werden kann.

Allgemein besteht die Erstellung einer neuen Komponente aus drei Schritten:

- Erstellung der eigentlichen Komponente mit Klassen für den App Inventor und den Android-Code
- Definition der Eigenschaften
- Anpassung der Version der Anwendung

Der Zusammenhang zwischen den verschiedenen Schritten ist in Abbildung 8.1 zu sehen, welche von oben nach unten und von links nach rechts gelesen werden muss. Wie dargestellt besteht die Komponente im Prinzip aus zwei Teilen. Zum Einen wird der Code für die grafische Oberfläche im App Inventor angegeben und zum Anderen der Code, also die Logik, für die spätere Android-Applikation. Darin befinden sich die Eigenschaften, wobei die Methoden für die Definition und Abänderung der Eigenschaften, sowohl im Code für den AI2, als auch im Android-Code angegeben

[1] siehe [Fee18; MIT18a; Spe18]

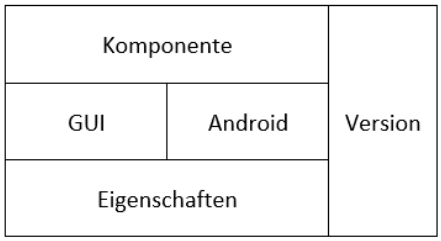



**Abbildung 8.1:** Arbeitsschritte für die Erstellung einer Komponente und deren Aufbau

werden müssen. Der rechts zu sehende Punkt „Version“, bezieht sich auf die ganze Komponente und dehnt sich deshalb über die ganze Höhe des Diagramms aus, da die Version beim Hinzufügen oder Abändern einer Komponente modifiziert werden muss.

Da zwischen zwei Arten von Komponenten unterschieden wird, sichtbare und nicht sichtbare, wird im Folgenden je eine Komponente der entsprechenden Art im AI2 ergänzt, um somit die Erstellung beider Arten zu beschreiben. Dabei besteht der Unterschied darin, dass sichtbare Komponenten ein GUI-Element besitzen. Die sichtbare Komponente wird ein ToggleButton darstellen, den man wie einen Schalter verwenden kann, um etwas zu aktivieren und zu deaktivieren, wie zum Beispiel eine Einstellung. Mithilfe der nicht sichtbaren Komponente soll es möglich sein zu betrachten, ob adaptives Lernen angewendet werden kann. Aus diesem Grund wird in dieser Komponente die rekursive Version des euklidischen Algorithmus implementiert, welcher gerade auch für jüngere Schüler schwierig zu verstehen sein könnte.

## 8.1 Eine sichtbare Komponente - ToggleButton

Als sichtbare Komponente soll ein ToggleButton implementiert werden. Dieser ähnelt der bereits vorhandenen Komponente der CheckBox. Ein ToggleButton ist ein Button, welcher an und aus geschaltet werden kann [And18b]. Im Vergleich zur CheckBox hat der ToggleButton allerdings nicht nur einen Text, sondern zwei, jeweils für den gedrückten und für den ungedrückten Button. Aber genauso wie die CheckBox verändert sich sein Zustand und er kann ein Event auslösen, indem der CompoundButton.OnCheckedChangeListener verwendet wird [And18b; And18c]. Dementsprechend findet in diesem Fall die Implementierung anhand der Implementierung der CheckBox statt.

Für die Erstellung einer neuen Komponente, müssen drei Dateien hinzugefügt werden. Zum einen appinventor/appengine/src/com/google/appinventor/client/editor/simple/components/Mock[Name der Komponente].java, in welcher die sogenannte Mock Komponente für den AI2 erstellt wird. Die Mock Komponente bestimmt die Anzeige im Designer auf der Website. Die Methoden in dieser Klasse verändern das Erscheinungsbild des Widgets, sofern im Designer die Eigenschaft einer Komponente angepasst worden ist. Zugehörig dazu wird noch eine Klasse appinventor/components/src/com/google/appinventor/components/runtime/[Name der Komponente].java benötigt, die zum einen den Code für das Android-Gerät enthalten muss und zum anderen die Annotationen für die Eigenschaften und Methoden, welche für den Benutzer im App Inventor sichtbar sein sollen.

**Listing 8.1** Ausschnitt aus „ToggleButton.java“: Verwendung der Annotationen `DesignerComponent` und `SimpleObject`

```
[...]
@DesignerComponent(version = YaVersion.TOGGLEBUTTON_COMPONENT_VERSION,
    description = "ToggleButton that changes its state and can raise an event
                   when the user clicks on it.",
    category = ComponentCategory.USERINTERFACE,
    iconName = "images/togglebutton.png")
@SimpleObject
public final class ToggleButton extends AndroidViewComponent
    implements OnCheckedChangeListener, OnFocusChangeListener {
[...]
```

Hierbei gibt es sieben verschiedene Annotationen. Eine Komponenten-Klasse, die im Designer angezeigt werden soll, muss zunächst mit der `DesignerComponent` Annotation versehen werden. `DesignerComponent` hat mehrere Elemente, dazu gehören die Version der Komponente, die Kategorie in der diese im Designer eingeordnet werden soll, eine Beschreibung, der letzte Teil des Pfads zum Icon, welches im Designer angezeigt werden soll, ob die Komponente sichtbar ist und ob diese in der Anwendung angezeigt werden soll. Daraufhin folgt die `SimpleObject` Annotation, welche bei Klassen verwendet werden muss, die entweder eine Superklasse einer Komponente definieren oder sind. Zur Veranschaulichung, wie die Annotationen verwendet werden, ist die Verwendung der oben genannten Annotationen `DesignerComponent` und `SimpleObject`, für die neue ToggleButton Komponente, in Listing 8.1 zu betrachten. Weitere Annotationen sind: `UsesPermissions`, `SimpleProperty`, `DesignerProperty`, `SimpleFunction` und `SimpleEvent`. Die `UsesPermissions` Annotation muss nur verwendet werden, wenn eine Komponente bestimmte „Permissions“, also Berechtigungen für den Zugriff auf beispielsweise Sensoren des Geräts benötigt [Spe18]. `SimpleProperty` wird verwendet, um festzulegen, ob die Eigenschaft im Blocks Editor sichtbar sein soll. Dies wird mithilfe des `userVisible` Elements der Annotation bestimmt, des Weiteren muss eine Beschreibung des Elements für die Dokumentation angegeben werden. Unter Verwendung der `DesignerProperty` wird festgelegt welche Set-Methoden im Designer zu sehen sein sollen, dabei werden in der Annotation ein Standardwert und der Typ der Eigenschaft in Form eines „PropertyEditors“ angegeben [Spe18]. Eine genauere Erklärung um was es sich bei einem „PropertyEditor“ handelt, findet sich bei der Beschreibung zum Hinzufügen von Eigenschaften im nächsten Absatz. `SimpleFunction` ist eine Annotation für Methoden und hat analog zu `SimpleProperty` zwei Elemente, in denen jeweils die Beschreibung und die Sichtbarkeit im Blocks Editor festgelegt werden. Auch `SimpleEvent` hat analog zu `SimpleProperty` die selben zwei Elemente, allerdings ist diese Annotation für Event-Handler bestimmt [Spe18]. Kurzbeschreibungen dieser Annotationen sind in Tabelle 8.1 zu finden.

Zuletzt wird ein neues Icon, im Ordner appinventor/appengine/src/com/google/appinventor/images, für die Komponenten Palette im Designer benötigt. Dabei muss die Größe des Bildes sechzehn mal sechzehn Pixel entsprechen. Darauf folgend wird das Bild in der Datei appinventor/appengine/src/com/google/appinventor/client/Images.java über eine Annotation importiert, wie in Listing 8.2 anhand der Ergänzungen für den ToggleButton zu erkennen. Für die Instantiierung der Komponente in der Palette ist zusätzlicher Code in der SimpleComponentDescriptor-Klasse (appinventor/appengine/src/com/google/appinventor/client/editor/simple/palette/SimpleComponentDescriptor

| Annotation | Beschreibung |
|---|---|
| DesignerComponent | Für die Klasse einer Komponente, die im Designer angezeigt werden soll |
| SimpleObject | Klasse, welche Superklasse einer Komponente definiert oder ist |
| UsesPermissions | Wird benötigt, sollte die Komponente gewisse Berechtigungen benötigen |
| SimpleProperty | Für jeden Getter und Setter einer Eigenschaft |
| DesignerProperty | Für Setter, welche im Designer angezeigt werden sollen |
| SimpleFunction | Für Methoden, analog zu SimpleProperty |
| SimpleEvent | Für Event-Handler |

**Tabelle 8.1:** Kurzbeschreibung der Annotationen [Spe18]

**Listing 8.2** Ausschnitt aus „Images.java“: Import des neuen Icons für den Designer

```
/**
   * Designer palette item: togglebutton component
   */
  @Source("com/google/appinventor/images/togglebutton.png")
  ImageResource togglebutton();
}
```

.java) in der if-Abfrage der `createMockComponent`-Methode notwendig. Dies ist beispielhaft für die neue ToggleButton Komponente in Listing 8.3 zu finden. Damit ist der ToggleButton mit seinem Icon im Designer wie in Abbildung 8.2 links unter der Kategorie User Interface zu finden. Zuletzt müssen Strings und deren Standard-Text, also die englischen Benennungen für die Komponente, deren Eigenschaften, Methoden und deren Hilfstext in der Klasse OdeMessages angegeben werden, damit diese wie in Kapitel 7 angegeben, übersetzt werden können. Auch hierfür ist in Listing 8.6 ein kleiner Ausschnitt der Strings für den ToggleButton zu finden.

Zum Hinzufügen von Eigenschaften und Methoden für den Designer, den Blocks Editor und den eigentlichen Android-Code, müssen die oben beschriebenen Dateien Mock[Name der Komponente].java und [Name der Komponente].java ergänzt werden. In diesem Fall sind das: MockToggleButton.java und ToggleButton.java. Da es sich um eine sichtbare Komponente handelt,

**Listing 8.3** Ausschnitt aus „SimpleComponentDescriptor.java“: Ergänzung in der If-Abfrage der `createMockComponent`-Methode

```
/**
 * Instantiates mock component by name.
 */
public static MockComponent createMockComponent(String name, String type,
SimpleEditor editor) {
  [...]
  else if (name.equals(MockToggleButton.TYPE)) {
      return new MockToggleButton(editor);
  }
  [...]
```

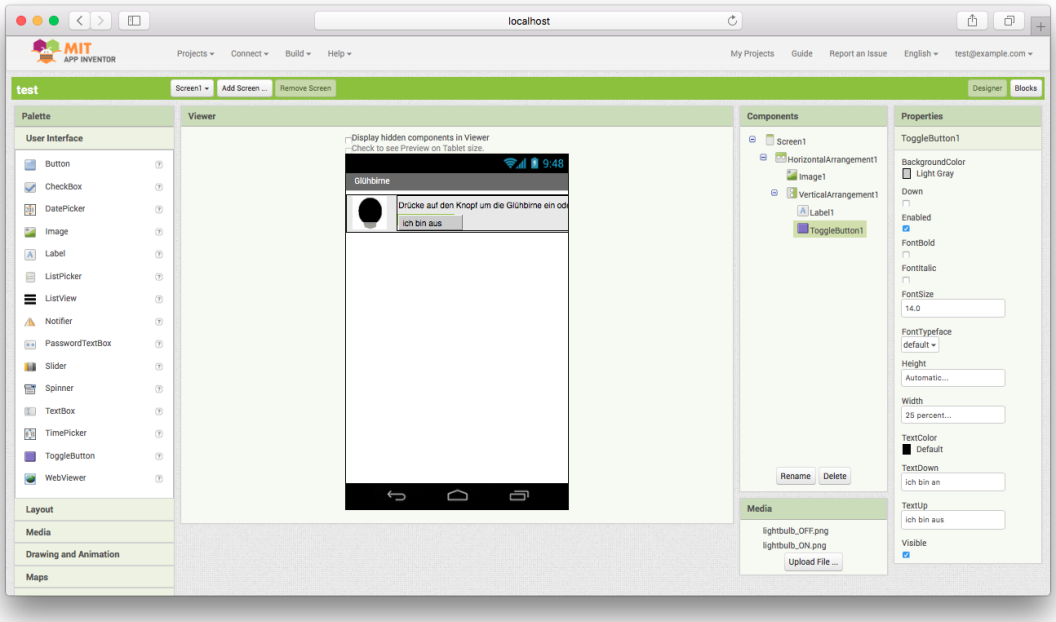



**Abbildung 8.2:** Designer mit verwendeter ToggleButton-Komponente

wird in der Mock-Klasse das entsprechende GWT-Widget hinzugefügt, genauso wie in der anderen Klasse das entsprechende Widget für Android ausgewählt wird. Um das GWT-Widget, welches für die Darstellung im Designer auf der Website verwendet wird, so gut wie möglich an das GUI-Element für die Android-App anzugleichen, können Anpassungen in einer css-Datei (appinventor/appengine/war/Ya.css) gemacht werden. Für die Eigenschaften ist, wie bei der Beschreibung der Annotationen bereits genannt, als Grundlage immer ein PropertyEditor nötig. Ein PropertyEditor bestimmt welche Werte im Designer wie festgelegt werden können [Spe18], das bedeutet der PropertyEditor legt fest, welche Werte eine Eigenschaft annehmen darf und wie die Eigenschaft im Property-Menü des AI2 dargestellt wird [Fee18]. Für viele Eigenschaften existiert bereits ein passender PropertyEditor. Diese sind im Verzeichnis unter appinventor/appengine/src/com/google/appinventor/client/widgets/properties zu finden. Zum Beispiel gibt es den BooleanPropertyEditor, welcher für Eigenschaften verwendet werden kann, die einen Wahrheitswert darstellen. Wählt man diesen PropertyEditor für eine Eigenschaft, wird diese im Property-Menü als eine CheckBox dargestellt, was in Abbildung 8.2 bei den Eigenschaften Down, Enabled, FontBold und FontItalic zu sehen ist.

Um das Ergänzen von Eigenschaften zu veranschaulichen geht es nun darum, die TextUp und die TextDown Eigenschaften hinzuzufügen. Diese bestimmen welche Texte in den beiden Zuständen des ToggleButtons angezeigt werden (gedrückt und nicht gedrückt). Für das Definieren einer Text-Eigenschaft kann man den vorhandenen „StringPropertyEditor“ verwenden. Zur Erstellung dieser Eigenschaften müssen sowohl in der ToggleButton, als auch in der MockToggleButton Klasse entsprechende Methoden ergänzt werden. Dazu gehören zwei Setter Methoden in der MockToggleButton Klasse (setTextUpProperty, setTextDownProperty) und jeweils zwei Getter und Setter für TextUp und TextDown in der Klasse ToggleButton. In Listing 8.4 und Listing 8.5 sind Code-Ausschnitte für TextUp zu sehen, wobei der Code für TextDown analog ist. Dabei sind die Methoden in der ToggleButton Klasse mit den entsprechenden Annotationen für den Designer und

**Listing 8.4** Ausschnitt aus „ToggleButton.java“: TextUp-Methoden

```
/**
 * Returns the text displayed by the togglebutton if not in down state.
 *
 * @return  togglebutton caption
 */
@SimpleProperty()
public String TextUp() {
  return (String)view.getTextOff();
}

/**
 * Specifies the text displayed by the togglebutton if not in down state.
 *
 * @param text  new caption for togglebutton
 */
@DesignerProperty(editorType = PropertyTypeConstants.PROPERTY_TYPE_STRING,
    defaultValue = "OFF")
@SimpleProperty
public void TextUp(String text) {
  textUp = text;
  if(Down() == false){
    TextViewUtil.setText(view, text);
  }
  view.setTextOff(text);
}
```

**Listing 8.5** Ausschnitt aus „MockToggleButton.java“: setTextUpProperty-Methode

```
/*
 * Sets the ToggleButtons UpText property to a new value.
 */
private void setTextUpProperty(String text) {
  if (!toggleWidget.isDown()){
    toggleWidget.setText(text);
  }
  upText = text;
  updatePreferredSize();
}
```

den Blocks Editor gekennzeichnet. Damit sind sowohl der Getter, als auch der Setter als Blöcke für das Programmieren verfügbar und der Wert für den Text kann im Designer ebenfalls über den Setter festgelegt werden. Sind alle Eigenschaften komplett, werden diese im Konstruktor des ToggleButtons angegeben. Des Weiteren ist eine Listener-Methode `onPropertyChange(String propertyName, String newValue)` in der Klasse MockToggleButton notwendig, um die Eigenschaften nach Veränderung über den Designer anzupassen. Die Eigenschaften des ToggleButtons sind in Abbildung 8.2 auf der rechten Seite des Screenshots zu finden.
Schließlich müssen sowohl in der Datei appinventor/appengine/src/com/google/appinventor/client/editor/simple/components/MockVisibleComponent.java, als auch in OdeMessages.java Strings

**Listing 8.6** Ausschnitt aus „OdeMessages.java“: Ergänzungen für ToggleButton

```
@DefaultMessage("ToggleButton")
@Description("")
String toggleButtonComponentPallette();

@DefaultMessage("ToggleButton that changes its state and can raise an event
                 when the user clicks on it.")
@Description("")
String ToggleButtonHelpStringComponentPallette();

@DefaultMessage("TextUp")
@Description("")
String TextUpProperties();

@DefaultMessage("TextDown")
@Description("")
String TextDownProperties();

@DefaultMessage("Down")
@Description("")
String DownProperties();
```

**Listing 8.7** Ausschnitt aus „MockVisibleComponent.java“: Ergänzungen für ToggleButton

```
protected static final String PROPERTY_NAME_DOWN = "Down";
protected static final String PROPERTY_NAME_TEXTDOWN = "TextDown";
protected static final String PROPERTY_NAME_TEXTUP = "TextUp";
```

ergänzt werden. Dabei stellen die Strings in MockVisibleComponent die Benennung der Eigenschaften dar, welche auch in der Listener-Methode in MockToggleButton in der Abfrage verwendet werden. In Listing 8.6 und Listing 8.7 sind einige der benötigten Strings zu finden. Wenn alle Eigenschaften und Methoden für den ToggleButton vollständig sind, sind die Eigenschaften und Methoden im Blocks Editor und in Designer verfügbar. Der Blocks Editor ist in Abbildung 8.3 zu sehen. Hier kann man einen Ausschnitt der Puzzle-Teile für die Methoden des ToggleButtons erkennen, genauso wie die Verwendung zweier dieser Teile in einem Code.

Der letzte Schritt ist die Anpassung der Versionsnummern. In der folgenden Datei müssen Änderungen vorgenommen werden, sollten sich die Young Android System, Blocks Language oder Komponenten Versionsnummern ändern. Dabei müssen bei Modifikationen der Blocks Language oder einer Komponente, deren Versionsnummern, genauso wie die des Young Android Systems, erhöht werden [Fee18]. Dabei ist in der Datei appinventor/components/src/com/google/appinventor/components/common/YaVersion.java die Zeile `public static final int TOGGLEBUTTON_COMPONENT_VERSION = 1;` zu ergänzen und die `YOUNG_ANDROID_VERSION` um eins zu inkrementieren [Fee18].

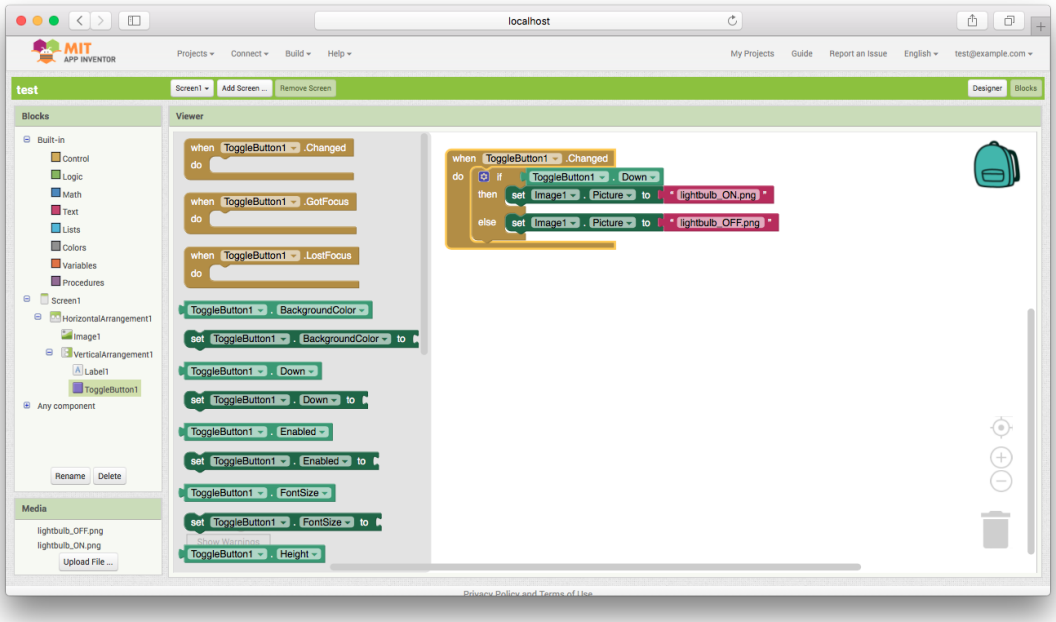



**Abbildung 8.3:** Blocks Editor mit verwendetem Methoden-Block der neuen ToggleButton-Komponente

## 8.2 Eine unsichtbare Komponente - euklidischer Algorithmus

Das Hinzufügen einer nicht sichtbaren Komponente funktioniert nach dem selben Prinzip wie das Hinzufügen einer sichtbaren Komponente, bis auf drei Ausnahmen: Es wird keine Mock[Name der neuen Komponente].java benötigt, in appinventor/appengine/src/com/google/appinventor/client/editor/simple/palette/SimpleComponentDescriptor.java wird die Methode `initBundledImages()` erweitert, anstatt `createMockComponent(String name, String type, SimpleEditor editor)` und die Klasse MockVisibleComponent bleibt unverändert. Beispielhaft sind für eine Komponente mit dem euklidischen Algorithmus, welcher den größter gemeinsamer Teiler (ggT) zweier Zahlen berechnet, also eine Datei im Ordner appinventor/components/src/com/google/appinventor/components/runtime/ mit dem Namen Euclid.java und ein Icon im Ordner appinventor/appengine/src/com/google/appinventor/images/ notwendig. Wie in Abschnitt 8.1 beschrieben befindet sich in Euclid.java die Logik für die Android Applikation und somit auch der in Algorithmus 8.1 dargestellte Algorithmus. Wie auch bei der visuellen Komponente werden in der Euclid.java die im Blocks Editor verfügbaren Puzzle-Teile über Annotationen festgelegt. Da die euklidischer Algorithmus Komponente lediglich eine Funktion erfüllen soll, ist auch nur ein Puzzle-Teil nötig. Dieses soll „greatestCommonDivisor“ (dt.: größterGemeinsamerTeiler) heißen und den ggT zweier ganzer Zahlen, als eine ganze Zahl zurück geben. Hierfür muss für im Code die `SimpleFunction` Annotation verwendet werden. Das fertige Puzzle-Teil, welches in Abbildung 8.5 bereits zu sehen ist, benötigt einen Stecker auf der linken Seite, damit der berechnete Wert für den ggT zurück gegeben werden kann. Zusätzlich werden für die anzugebenden Parameter, also die zwei Zahlen für die der ggT berechnet werden soll, zwei Aussparungen an der rechten Seite des Puzzle-Teils benötigt. Die Stecker und Aussparungen werden automatisch über die

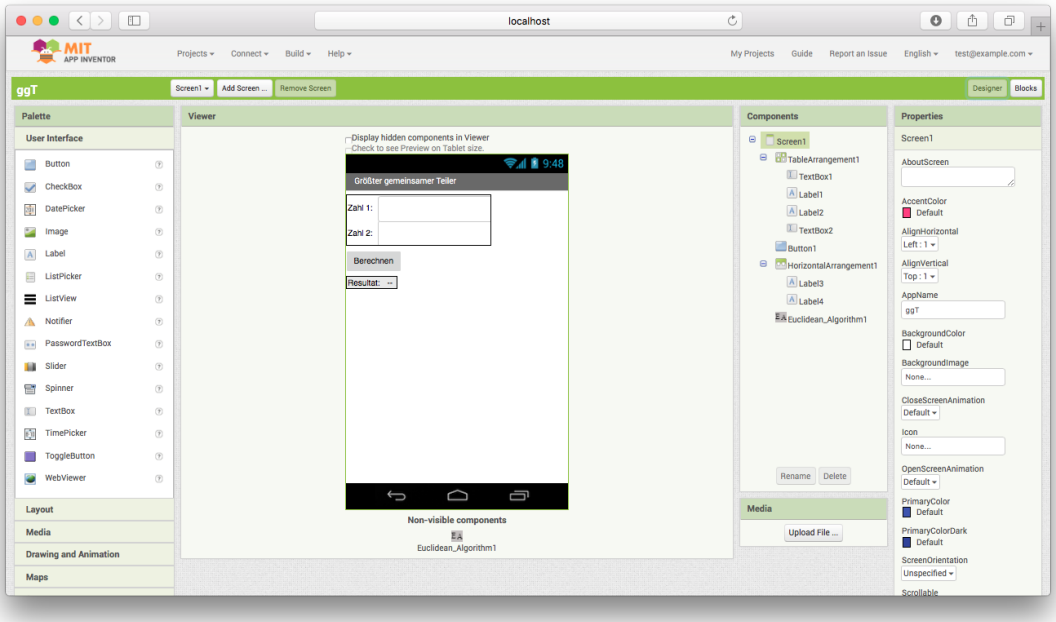



**Abbildung 8.4:** Designer mit verwendeter euklidischer Algorithmus-Komponente

Deklaration der Methode generiert. So bekommt ein Puzzle-Teil, einer mit der `SimpleFunction` Annotation versehenen Methode, einen Stecker auf der linken Seite, sollte im Methodenkopf ein Rückgabedatentyp angegeben worden sein, wird kein Datentyp angegeben, so bekommt das Teil eine dreieckige Aussparung am oberen linken Rand, wie der if-Block oder die Setter in Abbildung 8.3. Auch die Aussparungen für die Parameter entsprechen den in der Methoden-Deklaration angegeben Parametern. Des Weiteren werden in diesem Fall keine Eigenschaften für die Komponente benötigt, weshalb nur der Konstruktor und die Methode mit der `SimpleFunction` Annotation für den Blocks Editor enthalten sind. Außerdem muss in der `DesignerComponent` Annotation angegeben werden, dass die Komponente nicht sichtbar ist.
Daraufhin wird die Komponente beim Verwenden des Designer, wie in Abbildung 8.4 erkennbar, unterhalb des „Smartphone Bildschirms“ dargestellt und der Methoden-Block ist im Blocks Editor, wie in Abbildung 8.5 zu sehen, verfügbar. Die Modifikationen der Dateien appinventor/appengine/src/com/google/appinventor/client/Images.java für den Import des Icons, appinventor/appengine/src/com/google/appinventor/client/OdeMessages.java für die spätere Übersetzung der Komponente und appinventor/components/src/com/google/appinventor/components/common/YaVersion.java für der Version der Anwendung, sind analog zu denen für die sichtbare Komponente.
Die zuvor erwähnte Methode `initBundledImages()` ist für die Optimierung des Ladens der statischen Dateien der Icons für die nicht sichtbaren Komponenten. Hier muss lediglich eine Zeile hinzugefügt werden, welche ein Paar aus dem Name des Bildes und dem Bild selbst in eine Map speichert, damit nicht für jedes einzelne Bild eine Anfrage an den Server gesendet werden muss. Diese Zeile ist für das Beispiel des euklidischen Algorithmus in Listing 8.8 zu finden.

Die beiden oben beschriebenen Module können ebenfalls unter Verwendung zusätzlicher Blöcke im Blocks Editor erstellt werden. Wie das im Vergleich zur Verwendung der Komponenten aussieht wird in Kapitel 9 aufgezeigt.

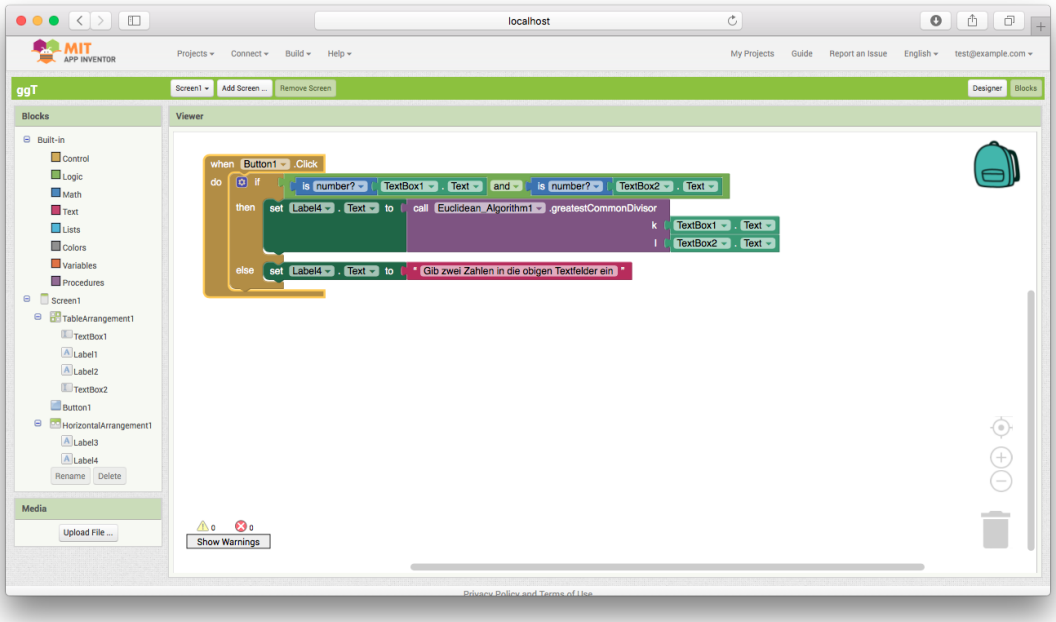



**Abbildung 8.5:** Blocks Editor mit verwendetem Methoden-Block der neuen euklidischer Algorithmus-Komponente

**Algorithmus 8.1** Euklidischer Algorithmus (größter gemeinsamer Teiler) entnommen aus: [Die+13]

**Require:** $k \geq 0$ und $l \geq 0$ mit $k \leq l$
  **procedure** GGT($k$,$l$)
    **if** $k = 0$ **then**
      **return** $l$
    **else**
      **return** ggT($l \mod k$,$k$)
    **end if**
  **end procedure**

**Listing 8.8** Ausschnitt aus „SimpleComponentDescriptor.java“: Ergänzung in der `initBundledImages()`

```
private static void initBundledImages() {
[...]
bundledImages.put("images/euclid.png", images.euclid());
[...]
}
```

# 9 Evaluation

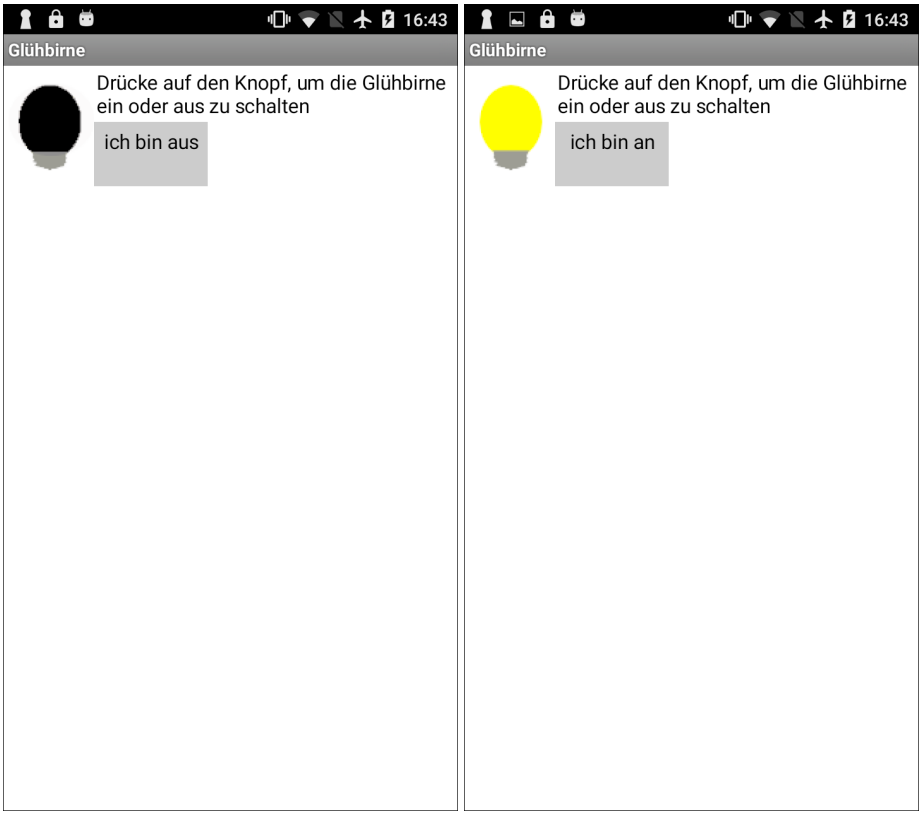
Im folgenden Kapitel wird evaluiert, ob der App Inventor durch die im Rahmen dieser Arbeit erstellten Erweiterungen (siehe Kapitel 6 bis 8) alle in Kapitel 3 aufgestellten Anforderungen an eine asynchrone E-Learning Plattform erfüllen, da diese wie in Abschnitt 5.4 aufgezeigt für die aktuelle Version des AI2 nicht erfüllt sind. Diese Bewertung findet zum Teil anhand von Applikationen statt, welche mithilfe des App Inventors implementiert worden sind. Dabei handelt es sich um zwei Anwendungen, welche einmal mit den Erweiterungen aus Kapitel 8 und einmal ohne die neuen Komponenten implementiert worden sind. Im Speziellen werden die verwendeten Blöcke zweier Paare an Applikationen miteinander verglichen. Des Weiteren soll mithilfe dieser Apps aufgezeigt werden, ob die Ergänzung neuer Komponenten eine Ersparnis an benötigten Blöcken mit sich bringt. Nachfolgend werden zunächst die beiden Anwendungen vorgestellt.

Die erste Applikation verwendet den ToggleButton, um eine „Glühbirne“ an und wieder auszuschalten, je nachdem ob sich der Button im gedrückten Zustand befindet oder nicht[1]. Dabei ist eine Image, eine Label und eine ToggleButton Komponente im Designer notwendig. Des Weiteren sind für die Anordnung der Komponenten noch Horizontal- und VerticalAlignments verwendet worden. Dies ist in Abbildung 8.2, im Screenshot des Designers, zu sehen. Im Blocks Editor, welcher in Abbildung 8.3 zu sehen ist, wird die Methode Change verwendet, welche dann je nach Zustand des Buttons das angezeigte Bild über eine Methode der Image-Komponente, abändert. Die fertige Anwendung ist in Abbildungen 9.1a und 9.1b zu sehen, wobei einmal der ToggleButton nicht gedrückt ist, also das Bild der Glühbirne schwarz ist und das zweite Bild zeigt den Button in seinem gedrückten Zustand mit den abgeänderten Bild einer „leuchtenden“ Glühbirne.

Die zweite Applikation verwendet die zuvor hinzugefügte, unsichtbare Komponente, welche den ggT berechnet. Diese nimmt zwei Zahlen als Eingabeparameter und berechnet auf Knopfdruck den größten gemeinsamen Teiler der vom Nutzer eingebenenen Zahlen. Hierbei werden neben der euclidean Algorithm Komponente mehrere Labels, zwei TextBoxen, ein Button und ein TableAlignment genauso wie ein HorizontalAlignment benötigt, um die Oberfläche wie in Abbildungen 8.4 und 9.2 zu gestalten. Im Blocks Editor, der in Abbildung 8.5 gezeigt wird, wird die Button.Click Methode verwendet, um anschließend die Eingaben des Nutzers aus den TextBoxen auszulesen und entweder einen Fehler wie in Abbildung 9.2b auszugeben, falls in die TextBoxen keine Zahlen eingegeben worden sind, oder die Methode der euclidean Algorithm Komponente mit der Nutzereingabe als Parameter aufzurufen (siehe Abbildung 9.2a).

Für die Bewertung der Komponenten findet der Vergleich zwischen den Anwendungen und ihrem jeweiligen Pendant ohne die neue Komponente statt. Der Code aus dem Blocks Editor der Apps kann anhand der Abbildungen 9.3 und 9.4 verglichen werden. Im Designer unterscheidet sich lediglich die Verwendung des Buttons statt des ToggleButtons. Um die Zustandsänderung des ToggleButtons zu

[1] Die Glühbirne „simuliert“ die Nutzung der Taschenlampenfunktion des Smartphones. Dies ist aktuell mit dem AI2 nicht möglich, da diese Funktion nicht unterstützt wird.



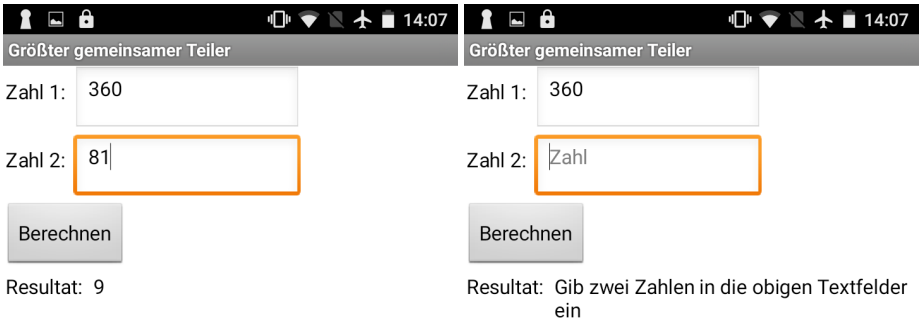

**(a)** ToggleButton nicht gedrückt und Glühbirne schwarz

**(b)** ToggleButton gedrückt und Glühbirne gelb

**Abbildung 9.1:** Screenshots einer Anwendung, welche den ToggleButton verwendet um zwischen zwei Bildern einer Glühbirne hin und her zu wechseln

simulieren, wird im Unterschied zum Code der ToggleButton-App in Abbildung 9.3, eine zusätzliche Variable benötigt, die aussagt ob der Button gerade gedrückt ist oder nicht. Hierfür bedeuten die Wahrheitswerte „wahr“, dass der Button im gedrückten Zustand ist und „falsch“ bedeutet, dass der Button nicht gedrückt ist. Diese Variable wird bei einem Button-Klick auf ihren negierten Wert gesetzt, um den Zustand zu ändern. Damit muss in der if-Abfrage dann dieser Wert abgefragt werden, um das richtige Bild auszuwählen und den Text des Buttons entsprechend seines Zustands zu anzupassen.
Somit spart das Hinzufügen des ToggleButtons in diesem Fall eine zusätzliche Variable, das wiederholte setzen der Variable und die zwei „Zeilen“ Code, beziehungsweise die vier Blöcke, für das Verändern des Textes auf dem Button. Außerdem folgt daraus ebenfalls eine Reduktion der Fehleranfälligkeit.

Die App zur Berechnung des größten gemeinsamen Teilers ist ohne die unsichtbare Komponente aufwendiger zu erstellen, obwohl im Designer bis auf die Verwendung der euclidean Algorithm Komponente kein Unterschied besteht. Der Code im Blocks Editor für die App ohne die neue Komponente ist länger, wie man im Vergleich der Abbildungen 9.5 und 9.6 erkennen kann. Der Teil des Codes in der Button.Click Methode ist bei beiden Anwendungen beinahe identisch. Unter Verwendung der unsichtbaren Komponente, wird lediglich, wie in Abbildung 9.5 zu erkennen, der




**(a)** Berechnung des ggT nach der Eingabe des Nutzers und pressen des Buttons **(b)** Fehlermeldung nach fehlerhafter Eingabe des Nutzers

**Abbildung 9.2:** Screenshots einer Anwendung, welche den ggT zweier Zahlen unter Verwendung der euclidean Algorithm Komponente berechnet

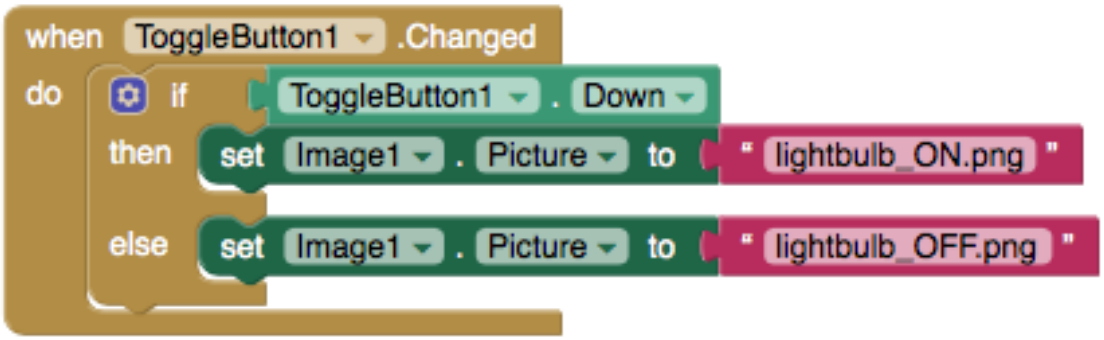



**Abbildung 9.3:** Blöcke der App, welche den ToggleButton verwendet

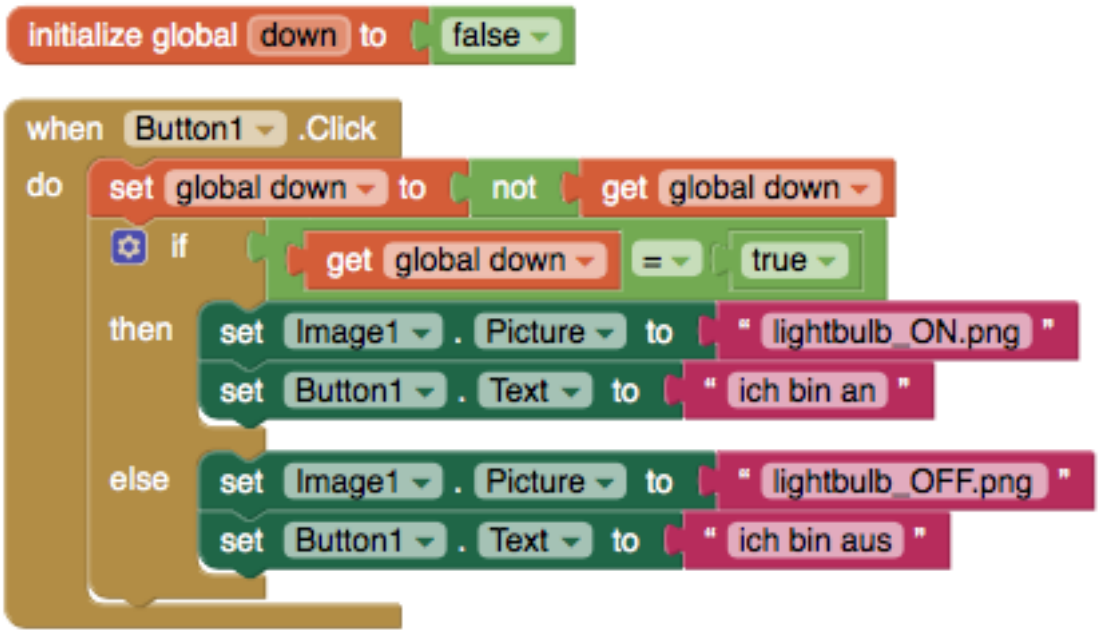



**Abbildung 9.4:** Blöcke der App, die den ToggleButton nicht verwendet

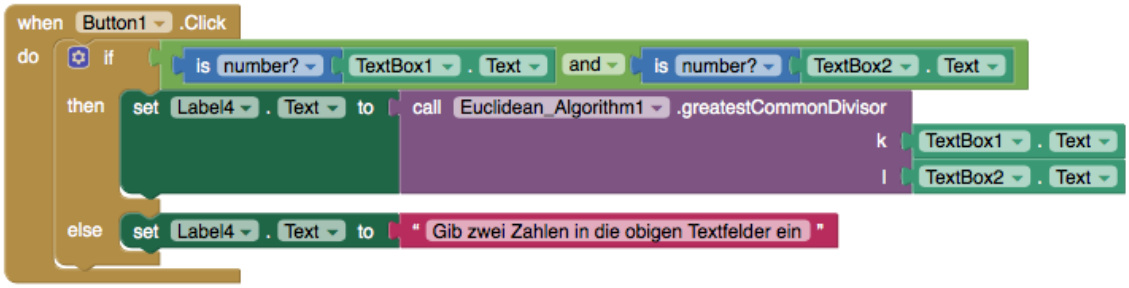



**Abbildung 9.5:** Blöcke der App, welche die ggT-Komponente verwendet

vorhandene Block mit der Methode „greatestCommonDivisor“ aus der Komponente zur Berechnung des ggT aufgerufen. Ohne die Komponente wird, wie in Abbildung 9.6 zu sehen, eine selbst geschriebene Methode aufgerufen, welche weiter unten, mithilfe eines Prozeduren-Blocks, definiert wird. Hier ist die selbe rekursive Methode implementiert, wie in Algorithmus 8.1 abgebildet, welche ebenfalls in der Komponente implementiert worden ist. Für die Definition des Algorithmus wird ein Prozeduren-Block mit Rückgabewert benötigt. Dieser gibt eine Variable zurück, in der das Resultat der Berechnung für den ggT der beiden Nutzereingaben gespeichert ist.
Somit erhöht die Implementierung des euklidischen Algorithmus im Blocks Editor, genauso wie bei der Implementierung des ToggleButtons, die Anzahl der zu verwendenden Blöcke. Des Weiteren wird die Komplexität der Implementierung reduziert, da der Algorithmus nicht selbst unter Verwendung der verfügbaren Blöcke zusammen geklickt werden muss.

**A.1 Collaborative Authoring** und **A.2 Verfügbarkeit:**
Sowohl die Zusammenarbeit der Lernenden, als auch die Verfügbarkeit der Plattform sollten nach der Ausführung der in Kapitel 6 beschriebenen Schritte kein Problem mehr sein. Mit einer

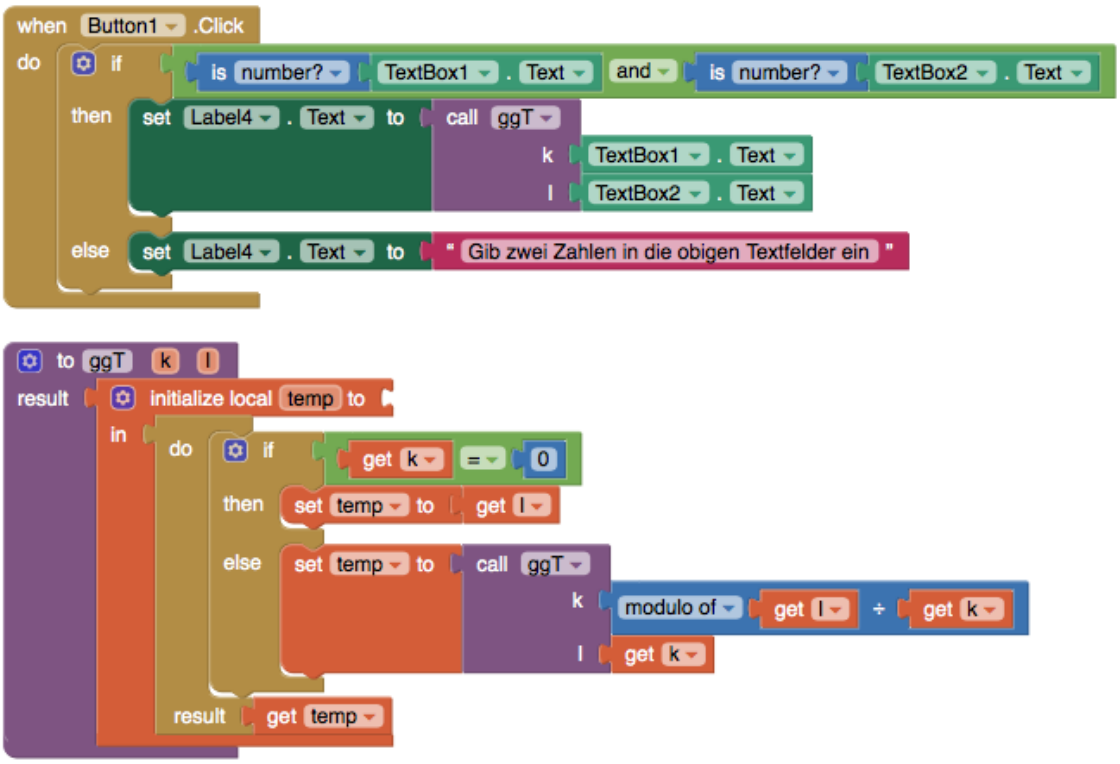



**Abbildung 9.6:** Blöcke der App, welche die ggT-Komponente nicht verwendet

lokal verfügbaren Instanz des App Inventors, wären zukünftige Workshops nicht mehr von der Erreichbarkeit der Server des MIT oder Google abhängig. Die einzige Abhängigkeit besteht dann zu den eigenen Systemen. Außerdem könnten die Schüler zusammen an einem Projekt arbeiten ohne das Projekt erst zu exportieren und dann auf dem anderen Account wieder importieren zu müssen, da sie sich auf der lokalen Installation alle den selben Speicher teilen [MIT15].

**A.3 Verständnis:**
Auch das Verstehen der grafischen Oberfläche sollte sich nach der Erweiterung in Kapitel 7 verbessert haben, da die Schüler mit ein paar einfachen Klicks in der Webanwendung die verwendete Sprache, nicht mehr nur auf die bisher verfügbaren Sprachen, wie Englisch, Italienisch, Französisch oder Koreanisch [Mas18a], sondern auch auf Deutsch abändern können. Weitere Sprachen können analog zu Kapitel 7 hinzugefügt werden.

**A.4 Flexibilität:**
In Kapitel 8 ist gezeigt worden, wie Komponenten hinzugefügt werden können. Damit kann ebenfalls die Flexibilität des Inhalts des AI2 gewährleistet werden. Dies zeigen beide der oben beschriebenen, mit dem AI2 erstellten Apps, da beide einen unterschiedlichen Inhalt haben. So ist die erste App, mit der Glühbirne, eher zum Austesten der Funktionen des App Inventors und die zweite App, zur Berechnung des ggT, behandelt Mathematik. Des Weiteren behandelt die zweite App in Verbindung mit ihrem Pendant, ohne die neue Komponente, das Konzept von Rekursion in der Programmierung. Auf diese Weise kann man durch das Erstellen bestimmter Komponenten verschiedene Inhalte behandeln. Als weiteres Beispiel könnte eine Komponente, welche einfache Verschlüsselungsalgorithmen, wie die Caesar-Verschlüsselungen enthält, eine Einführung in die

Kryptographie geben. Eine Komponente zum Steuern von Robotern, ähnlich zur vorhandenen LEGO MINDSTORMS Komponente, kann Interesse für den Bereich Robotics wecken. Zum Beispiel für die mBots[2], die aktuell nicht vom AI2 unterstützt werden und nur aufwendig über die generische Bluetooth-Verbindung angesteuert werden können.

**A.5 Adaptives Lernen:**
Für das Bieten einer adaptiven Lernumgebung, kann man als Beispiel die Verwendung der euclidean Algorithm-Komponente aus dem obigen Beispiel angeführt werden. Die rekursive Definition der Methode für die Berechnung des ggT stellt für Schüler möglicherweise eine Herausforderung dar, da die Rekursion als eines der am schwersten beizubringenden Konzepte gehandhabt wird [GH98]. Sollte die Altersspanne bei der Durchführung eines Workshops groß sein, wie es zum Beispiel beim Girls'Day[3] der Fall ist (Klassenstufe 5–10), so ist es möglich zwei Gruppen zu bilden. So kann man, aufgrund der Schwierigkeit der Rekursion im euklidischen Algorithmus, bei der Erstellung einer App, welche den ggT berechnen soll, die jüngeren Teilnehmer das ggT-Modul verwenden lassen, während die älteren den Algorithmus selbst implementieren. Damit kann die Erweiterbarkeit des App Inventors gut von den Lehrenden verwendet werden, um den Inhalt und die Aufgabe an den individuellen Wissenstand der Lernenden anzupassen.

Damit sind alle der zuvor gestellten Anforderungen an die Software erfüllt worden.

[2] Zu finden unter: [Mak18]
[3] siehe [Kom18]

# 10 Zusammenfassung und Ausblick

E-Learning hat in den letzten Jahren immer mehr an Bedeutung gewonnen. Des Weiteren wird dieser Begriff in der Literatur heterogen verwendet. So werden eine Vielzahl an Anwendungen als E-Learning Plattformen bezeichnet. Im Rahmen dieser Arbeit liegt der Fokus auf asynchronen E-Learning Plattformen. Daraufhin wird ermittelt, welche Anforderungen ein solches System erfüllen muss. Dazu gehören die Zusammenarbeit, die Verfügbarkeit, das Verständnis der Sprache, die Flexibilität des Inhalts und die Möglichkeit adaptives Lernen zu integrieren. Ebenso werden visuelle Entwicklungsplattformen vorgestellt, welche dafür genutzt werden können Menschen einen besseren Einstieg ins Programmierenlernen zu gewährleisten. Hierbei wird das Hauptaugenmerk auf den App Inventor gesetzt, welcher für die Entwicklung von mobilen Applikationen verwendet werden kann. Da der App Inventor in den Bereichen Zusammenarbeit, Verfügbarkeit, Verständlichkeit der Sprache, Flexibilität des Inhalts und adaptives Lernen, die Anforderungen nicht erfüllt, wurden im Kontext dieser Arbeit mehrere Schritte zur Verbesserung dieser Punkte durchgeführt. Dazu gehört die Installation einer lokalen Instanz des App Inventors, um eine bessere Zusammenarbeit zwischen Schülern zu ermöglichen und die Verfügbarkeit zu verbessern. Die Erweiterung der Sprachunterstützung um die deutsche Sprache, hilft den Schülern mit ihren Sprachproblemen und die Erweiterung um neue Komponenten ermöglicht die Flexibilität und adaptives Lernen. Abschließend folgt aus einer erneuten Evaluation, nicht nur die mögliche Reduktion an der Anzahl zu verwendender Blöcken, sondern auch die Chance auf eine Minderung der Komplexität einer Aufgabe durch die Verwendung von neu erstellten Komponenten.

Um den praktischen Nutzen der gemachten Änderungen am AI2 zu untersuchen, muss eine Studie durchgeführt werden. Eine solche Evaluation wäre möglich, indem man mit einer Gruppe von Schülern arbeitet. Die Schüler müssten daraufhin in zwei Gruppen aufgeteilt werden. Dabei verwendet die eine Gruppe die erweiterte und die andere Gruppe verwendet als Kontrollgruppe die ursprüngliche Version des App Inventors, damit ein direkter Vergleich in der Arbeit mit den beiden App Inventor Versionen gezogen werden kann. Hierbei sollte dann sowohl die Zeit zum lösen der Aufgabe, als auch die objektive Meinung der Schüler über das Sprachverständnis und die Schwierigkeit der Aufgabe betrachtet werden.

# Literaturverzeichnis


[Alo+04] F. T. Alotaiby, J. X. Chen, E. J. Wegman, H. Wechsler, D. Sprague. „Teacher-driven: Web-based Learning System“. In: *Proceedings of the 5th Conference on Information Technology Education*. CITC5 ’04. 2004, S. 284–284 (zitiert auf S. 20).

[And18a] Android Developers. *Android Debug Bridge (adb)*. 2018. URL: https://developer.android.com/studio/command-line/adb (zitiert auf S. 41).

[And18b] Android Developers. *Toggle Buttons*. 2018. URL: https://developer.android.com/guide/topics/ui/controls/togglebutton (zitiert auf S. 47, 48).

[And18c] Android Developers. *ToggleButton*. 2018. URL: https://developer.android.com/reference/android/widget/ToggleButton (zitiert auf S. 48).

[Bel+14] A. Belguinha, P. Rodrigues, P. J. Cardoso, J. M. Rodrigues, D. Paciência. „A visual programming language for soccer“. In: *Proceedings of the 9th International Conference on Software Paradigm Trends*. ICSOFT-PT 2014. 2014, S. 121–127 (zitiert auf S. 21, 22).

[Ber+15] A. Berchtold, T. Kieselmann, P. Wagner. *When Androids Control Robots*. Projekt-INF. Universität Stuttgart, Juni 2015. 8 S. (zitiert auf S. 13, 19, 37).

[Blo18] Blocklanguages. *What’s a Block Language?* 2018. URL: http://www.blocklanguages.org/ (zitiert auf S. 24, 30).

[Bur95] M. M. Burnett. „Visual programming“. In: *Computer* 28.3 (1995), S. 14–16 (zitiert auf S. 21).

[Cap01] J. Capper. „E-Learning Growth and Promise For the Developing World“. In: *TechKnowLogia* 2.2 (2001), S. 7–10 (zitiert auf S. 17).

[Car02] C. Carapeto. „Environmental Sciences and Distance Education“. In: *Encyclopedia of Networked and Virtual Organizations*. IGI Global, 2002, S. 492–498 (zitiert auf S. 15).

[Car+05] M. C. Carlisle, T. A. Wilson, J. W. Humphries, S. M. Hadfield. „RAPTOR: a visual programming environment for teaching algorithmic problem solving“. In: *Acm Sigcse Bulletin* 37.1 (2005), S. 176–180 (zitiert auf S. 22).

[Car17] Carnegie Mellon University. *Alice – Tell Stories. Build Games. Learn to Program.* 2017. URL: https://www.alice.org/ (zitiert auf S. 22).

[Com17] Computer Hope. *S-expression*. 2017. URL: https://www.computerhope.com/jargon/s/s-expression.htm (zitiert auf S. 35).

[Dan+12] W. Dann, D. Slater, D. Cosgrove, D. Culyba. „Mediated transfer: Alice 3 to java“. In: *Proceedings of the 43rd ACM technical symposium on Computer Science Education*. SIGCSE ’12. 2012, S. 141–146 (zitiert auf S. 24).

[Dic12] P. E. Dickson. „Cabana: a cross-platform mobile development system“. In: *Proceedings of the 43rd ACM technical symposium on Computer Science Education*. SIGCSE '12. 2012, S. 529–534 (zitiert auf S. 22).

[Die+13] V. Diekert, M. Kufleitner, G. Rosenberger. *Elemente der Diskreten Mathematik: Zahlen und Zählen, Graphen und Verbände*. Berlin, Boston: De Gruyter, 2013. 246 S. ISBN: 9783110277678 (zitiert auf S. 56).

[Fee18] K. Feeney. *How to Add a Property to a Component*. 2018. URL: https://docs.google.com/document/d/1ukpZXcd0fIn1fV0cNp4J00wrfwwA1daswU1yY6dqqfs/pub (zitiert auf S. 47, 51, 53).

[Fee+15] K. Feeney, N. Nolte, A. Esmaili. *AIMerger Documentation AI2*. 2015. URL: https://docs.google.com/document/d/15cEkliGmGxLX2muxysEOrMbPJnjmmglMtMVhbDpLZFE/edit#heading=h.uqq1hmq6y3rx (zitiert auf S. 35).

[FR18] P. G. Feijóo-García, F. De la Rosa. „Instructional Application for Programming and Algorithmic Self-Learning - A Didactic Approach with Mobile Robotics as Pedagogical Context“. In: *Proceedings of the 10th International Conference on Computer Supported Education*. CSEDU. 2018, S. 230–237 (zitiert auf S. 26).

[Fil18] FileInfo. *.APK File Extension*. 2018. URL: https://www.alice.org/ (zitiert auf S. 35).

[GWT18] GWT Project. *Google Web Toolkit*. 2018. URL: http://www.gwtproject.org (zitiert auf S. 34, 47).

[GH98] J. Gal-Ezer, D. Harel. „What (else) should CS educators know?“ In: *Communications of the ACM* 41.9 (1998), S. 77–84 (zitiert auf S. 62).

[Goo18a] Google Cloud. *Google App Engine*. 2018. URL: https://cloud.google.com/appengine/ (zitiert auf S. 32).

[Goo18b] Google Developers. *Blockly Basics*. 2018. URL: https://developers.google.com/blockly/ (zitiert auf S. 24, 35).

[Goo18c] Google Play. *MIT AI2 Companion*. 2018. URL: https://play.google.com/store/apps/details?id=edu.mit.appinventor.aicompanion3 (zitiert auf S. 27).

[Gur05] S. Guri-Rosenblit. „'Distance education' and 'e-learning': Not the same thing“. In: *Higher Education* 49.4 (2005), S. 467–493 (zitiert auf S. 15).

[Haa+13] J. M. Haake, T. Erdbrügge, C. Rietzscher. „Förderung von Kompetenzen für “Collaborative Authoring“ in der Sekundarstufe II.“ In: *GI-Jahrestagung*. INFORMATIK. 2013, S. 181–195 (zitiert auf S. 19).

[HL13] H. Herper, Lehramtsausbildung, AG. „Informatische Bildung in der Primarstufe-Voraussetzung für den Einsatz digitaler Unterrichtsmedien.“ In: *GI-Jahrestagung*. INFORMATIK. 2013, S. 196–207 (zitiert auf S. 19).

[Hon13] W. L. Honig. „Teaching and assessing programming fundamentals for non majors with visual programming“. In: *Proceedings of the 18th ACM conference on Innovation and technology in computer science education*. ITiCSE '13. 2013, S. 40–45 (zitiert auf S. 13).

[Hra08] S. Hrastinski. „Asynchronous and Synchronous E-Learning“. In: *Educause Quarterly* 31.4 (2008), S. 51–55 (zitiert auf S. 16–19).

[Jav18] Javatpoint. *Dalvik Virtual Machine | DVM*. 2018. URL: https://www.javatpoint.com/dalvik-virtual-machine (zitiert auf S. 36).

[KAW18a] KAWA. *The Kawa Scheme language*. 2018. URL: https://www.gnu.org/software/kawa/index.html (zitiert auf S. 35, 36).

[KAW18b] KAWA. *The Kawa language framework*. 2018. URL: https://www.gnu.org/software/kawa/Framework.html (zitiert auf S. 35).

[KP05] C. Kelleher, R. Pausch. „Lowering the barriers to programming: A taxonomy of programming environments and languages for novice programmers“. In: *ACM Computing Surveys (CSUR)* 37.2 (2005), S. 83–137 (zitiert auf S. 13, 21).

[KW03] D. Klein, M. Ware. „E-learning: new opportunities in continuing professional development“. In: *Learned Publishing* 16.1 (2003), S. 34–46 (zitiert auf S. 17).

[Kom18] Kompetenzzentrum Technik-Diversity-Chancengleichheit e.V. *Girls'Day*. 2018. URL: https://www.girls-day.de/ (zitiert auf S. 37, 62).

[KH05] A. Koohang, K. Harman. „Open source: A metaphor for E-learning“. In: *Proceedings of the 2005 InSITE Conference*. I$^{n}$SITE 2005. 2005, S. 75–86 (zitiert auf S. 15).

[Lou+17] F. E. Louhab, A. Bahnasse, M. Talea. „Smart Adaptive Learning Based on Moodle Platform“. In: *Proceedings of the Mediterranean Symposium on Smart City Application*. SCAMS '17. 2017, 13:1–13:5 (zitiert auf S. 20).

[MIT15] MIT Center for Mobile Learning. *How to build App Inventor from the MIT sources*. 2015. URL: https://docs.google.com/document/d/1Xc9yt02x3BRoq5m1PJHBr81OOv69rEBy8LVG_84j9jc/pub (zitiert auf S. 32, 37, 39–41, 61).

[MIT18a] MIT Center for Mobile Learning. *MIT App Inventor*. 2018. URL: http://appinventor.mit.edu/appinventor-sources/ (zitiert auf S. 47).

[MIT18b] MIT Center for Mobile Learning. *Welcome to MIT App Inventor*. 2018. URL: https://github.com/mit-cml/appinventor-sources (zitiert auf S. 40).

[MIT16] MIT. *Scratch*. 2016. URL: https://scratch.mit.edu/ (zitiert auf S. 13, 24, 25).

[MIT17] MIT. *Offline Editor für Scratch 2.0*. 2017. URL: https://scratch.mit.edu/download (zitiert auf S. 24).

[MIT18c] MIT. *Block-Based Coding*. 2018. URL: https://en.scratch-wiki.info/wiki/Block-Based_Coding (zitiert auf S. 30).

[MIT18d] MIT. *Scratch*. 2018. URL: https://scratch.mit.edu/projects/editor/?tip_bar=home (zitiert auf S. 24).

[Mak18] Makeblock Co., Ltd. *Makeblock: Global STEAM Education Solution Provider*. 2018. URL: http://www.makeblock.com/ (zitiert auf S. 62).

[Mal+10] J. Maloney, M. Resnick, N. Rusk, B. Silverman, E. Eastmond. „The scratch programming language and environment“. In: *ACM Transactions on Computing Education (TOCE)* 10.4 (2010), S. 1–16 (zitiert auf S. 22, 24).

[Mas18a] Massachusetts Institute of Technology. *MIT App Inventor*. 2018. URL: http://appinventor.mit.edu/explore/ (zitiert auf S. 13, 26, 27, 35, 61).

[Mas18b] Massachusetts Institute of Technology. *System Requirements*. 2018. URL: http://appinventor.mit.edu/explore/content/system-requirements.html (zitiert auf S. 47).

[Mor+11] R. Morelli, T. De Lanerolle, P. Lake, N. Limardo, E. Tamotsu, C. Uche. „Can android app inventor bring computational thinking to k-12“. In: *Proceedings of the 42nd ACM technical symposium on Computer Science Education*. SIGCSE '11. 2011, S. 1–6 (zitiert auf S. 24).

[Nai02] S. Naidu. „Designing and Evaluating Instruction for e-Learning“. In: *Designing Instruction for Technology-Enhanced Learning*. IGI Global, 2002, S. 134–159 (zitiert auf S. 15, 16).

[NE14] S. A. Nikou, A. A. Economides. „Transition in student motivation during a scratch and an app inventor course“. In: *Proceedings of Global Engineering Education Conference (EDUCON)*. 2014 IEEE. 2014, S. 1042–1045 (zitiert auf S. 24, 26).

[Ora18] Oracle. *Java SE Downloads*. 2018. URL: http://www.oracle.com/technetwork/java/javase/downloads/index.html (zitiert auf S. 40).

[Pap+16] S. Papadakis, M. Kalogiannakis, N. Zaranis, V. Orfanakis. „Using Scratch and App Inventor for teaching introductory programming in secondary education. A case study“. In: *International Journal of Technology Enhanced Learning* 8.3/4 (2016), S. 217–233 (zitiert auf S. 13, 21).

[RD05] L. P. Robert, A. R. Dennis. „Paradox of richness: A cognitive model of media choice“. In: *IEEE Transactions on Professional Communication* 48.1 (2005), S. 10–21 (zitiert auf S. 18–20).

[Rui+06] J. G. Ruiz, M. J. Mintzer, R. M. Leipzig. „The impact of e-learning in medical education“. In: *Academic medicine* 81.3 (2006), S. 207–212 (zitiert auf S. 16).

[San+12] A. Sangrà, D. Vlachopoulos, N. Cabrera. „Building an inclusive definition of e-learning: An approach to the conceptual framework“. In: *The International Review of Research in Open and Distributed Learning* 13.2 (2012), S. 145–159 (zitiert auf S. 15, 16).

[Sch+14] J. Schiller, F. Turbak, H. Abelson, J. Dominguez, A. McKinney, J. Okerlund, M. Friedman. „Live programming of mobile apps in App Inventor“. In: *Proceedings of the 2nd Workshop on Programming for Mobile & Touch*. PROMOTO '14. 2014, S. 1–8 (zitiert auf S. 32, 35, 36).

[Sla14] W. Slany. „Pocket code: a scratch-like integrated development environment for your phone“. In: *Proceedings of the companion publication of the 2014 ACM SIGPLAN conference on Systems, Programming, and Applications: Software for Humanity*. SPLASH '14. 2014, S. 35–36 (zitiert auf S. 26).

[Spe18] E. Spertus. *How to Add a Component*. 2018. URL: https://docs.google.com/document/d/1xk9dMfczvjbbwD-wMsr-ffqkTlE3ga0ocCE1KOb2wvw/pub (zitiert auf S. 35, 47, 49–51).

[SD18] E. Spertus, J. Dominguez. *App Inventor Developer Overview*. 2018. URL: https://docs.google.com/document/d/1hIvAtbNx-eiIJcTA2LLPQOawctiGIpnnt0AvfgnKBok/pub (zitiert auf S. 34–36).

[Ste+13] M. Steinert, P. Gerth, S. Mick. *LEGO Mindstorms – Spielend programmieren lernen*. Projekt-INF. Universität Stuttgart, Okt. 2013. 9 S. (zitiert auf S. 13, 19, 37).

[Tra18] Transifex. *Java Properties*. 2018. URL: https://docs.transifex.com/formats/java-properties (zitiert auf S. 43).

[WW15] D. Weintrop, U. Willensky. „To block or not to block? That is the question“. In: *Journal of Thoracic and Cardiovascular Surgery*. IDC '15. 2015, S. 199–208 (zitiert auf S. 24).

[Wel+13] E. T. Welsh, C. R. Wanberg, K. G. Brown, M. J. Simmering. „E-learning: emerging uses, empirical results and future directions“. In: *International Journal of Training and Development* 7.4 (2013), S. 245–258 (zitiert auf S. 16).

[Wil+18] T. Wilson, M. Carlisle, J. Humphries, J. Moore. *Welcome to the RAPTOR home page*. 2018. URL: https://raptor.martincarlisle.com/ (zitiert auf S. 22).

[Wol11] D. Wolber. „App inventor and real-world motivation“. In: *Proceedings of the 42nd ACM technical symposium on Computer science education*. SIGCSE '11. 2011, S. 601–606 (zitiert auf S. 13).

[Wol+14] D. Wolber, H. Abelson, E. Spertus, L. Looney. *App Inventor 2*. Sebastopol: O'Reilly Media, Inc., 2014. 362 S. ISBN: 9781491906842. URL: http://www.appinventor.org/book2 (zitiert auf S. 26, 27, 36).

Alle URLs wurden zuletzt am 05. 11. 2018 geprüft.

**Erklärung**

Ich versichere, diese Arbeit selbstständig verfasst zu haben. Ich habe keine anderen als die angegebenen Quellen benutzt und alle wörtlich oder sinngemäß aus anderen Werken übernommene Aussagen als solche gekennzeichnet. Weder diese Arbeit noch wesentliche Teile daraus waren bisher Gegenstand eines anderen Prüfungsverfahrens. Ich habe diese Arbeit bisher weder teilweise noch vollständig veröffentlicht. Das elektronische Exemplar stimmt mit allen eingereichten Exemplaren überein.

---

Ort, Datum, Unterschrift