Deutsches Jungforscher Netzwerk
juFORUM e.V.

MIT App Inventor

Apps entwickeln - wie DU willst!

Mit dem MIT App Inventor 2

Deutsches Jungforschernetzwerk – juFORUM e.V.

Julian Bopp

www.juforum.de

facebook.com/juforum

Stand: 24. März 2015

Inhaltsverzeichnis

1 Einführung

Der MIT App Inventor[1] bietet die Möglichkeit, eigene Apps für Android zu erstellen.

Zielgruppe:	8. - 12. Klasse
Interaktiv:	ja
Vorführdauer:	mindestens 20 Minuten für ein Projekt bis zu 4 Stunden für einen gesamten Workshop

Es gibt Millionen von Apps zum Download, aber das, was man braucht, ist trotzdem nie dabei. Das Massachusetts Institute of Technology hat ein Programm erstellt, mit dem sich ohne Vorkenntnisse durch graphisches Programmieren eigene Applications für das Betriebssystem Android entwickeln lassen, ganz nach eigenem Geschmack.

[1] http://appinventor.mit.edu/

2 Vorschlag eines Workshops für den Unterricht

An Schulen gehört Informatikunterricht vielerorts nicht zum Pflichtcurriculum. Deshalb ist es nicht ungewöhnlich, dass Informatik an deutschen Schulen etwas zu kurz kommt. Umso wichtiger ist es, bei den Schülerinnen und Schülern ein Bewusstsein zu schaffen, wie Computerprogramme im Allgemeinen oder im Speziellen Smartphone-Apps funktionieren, wie die „Programmierung“ überhaupt vonstatten geht und warum ein Programm eine bestimmte Aufgabe lösen kann. Uns ist es ein Anliegen, einen Einstieg in die für den Anfänger schwer zugängliche Welt der Programmierung zu eröffnen. Ist ein Einstieg geschafft, daher die erste Hürde überwunden, fällt die weitere selbstständige Einarbeitung in die jeweilige Thematik deutlich leichter.

Der folgende Plan soll einen Vorschlag bieten, wie ein Workshop bzw. Unterrichtsmodul zur App-Entwicklung mit dem MIT App Inventor aussehen könnte. Gerne führen Mitglieder des Deutschen Jungforschernetzwerks gemeinsam mit Ihnen im Rahmen unseres Mentorenprogramms[2] entsprechende Workshops an Ihrer Schule durch. Scheuen Sie sich nicht, mit uns in Kontakt zu treten oder uns Ihre Erfahrungen mit dem MIT App Inventor mitzuteilen, sollten Sie ein entsprechendes Modul unterrichtet haben.

Musterablaufplan

Bestenfalls arbeiten die Schülerinnen und Schüler gemeinsam in Zweierteams an einem Computer. Der Umfang der Bearbeitung der im Folgenden aufgeführten Abschnitte kann selbstverständlich abhängig von der Bearbeitungsgeschwindigkeit der jeweiligen Schülergruppe variieren.

1. Begrüßung und Einführung in den App Inventor

(Dauer ca. 30 Minuten)

- Workshopleiter stellt sich, juFORUM e.V. und die Themen des Workshops vor
- Hinweise zur Installation des MIT App Inventor 2 und zur Nutzung des Emulators
- Einführung in die Oberfläche und Benutzung des App Inventors

[2] https://www.juforum.de/mentoren/

2. Get started! So funktioniert's...

(Dauer ca. 45 Minuten)

Unter Anleitung des Workshopleiters programmieren die Schülerteams eine erste App (beispielsweise den Zähler – siehe Abschnitt 5) selbst. Diese App wird anschließend getestet. Es wird auf das Erlernen folgender Konzepte Wert gelegt:

- Steuerelemente platzieren und Verhalten der ersten App programmieren
- Buttons, Event-Handler für Klick-Events, Eigenschaften von Objekten lesen bzw. schreiben

3. Auf die Menge kommt es an! – Selbstständige Vertiefung

(Dauer ca. 60 Minuten)

Anhand eines Arbeitsblatts programmieren die Schülerteams eine geringfügig anspruchsvollere App selbst. Hierfür eignet sich der Pizzarechner (siehe Abschnitt 6). Die Workshopleiter beantworten Fragen der Schülerteams. Folgendes soll erlernt werden:

- Verwendung weiterer Standardsteuerelemente und Verwendung deren Event-Handler
- Durchführen einfacher Berechnungen
- Aufzeigen der Notwendigkeit, Benutzereingaben auf Gültigkeit zu überprüfen und entsprechende Eingabeprüfung implementieren (hierzu ist eine kleine Einführung in die Logik erforderlich)

4. Zusatzaufgabe: Meine erste eigene App

Schülerteams, die mit den vorangehenden Abschnitten schnell fertig werden (beispielsweise aufgrund von Vorkenntnissen) sollen in einer freien Phase Zeit finden, fortgeschrittene Aufgaben zu bearbeiten oder eine eigen App zu programmieren. Eine freie Phase ist vorteilhaft, da eigenes Interesse hauptsächlich durch Ausprobieren entsteht. Werden interessante Steuerelemente entdeckt, tendieren Schülerinnen und Schüler dazu, diese aus eigenem Interesse selbstständig kennenlernen zu wollen. Am Ende dieser Phase sollten die in der freien Phase erarbeiteten Apps zur Motivation der anderen Schülerteams präsentiert werden.

3 Los geht's

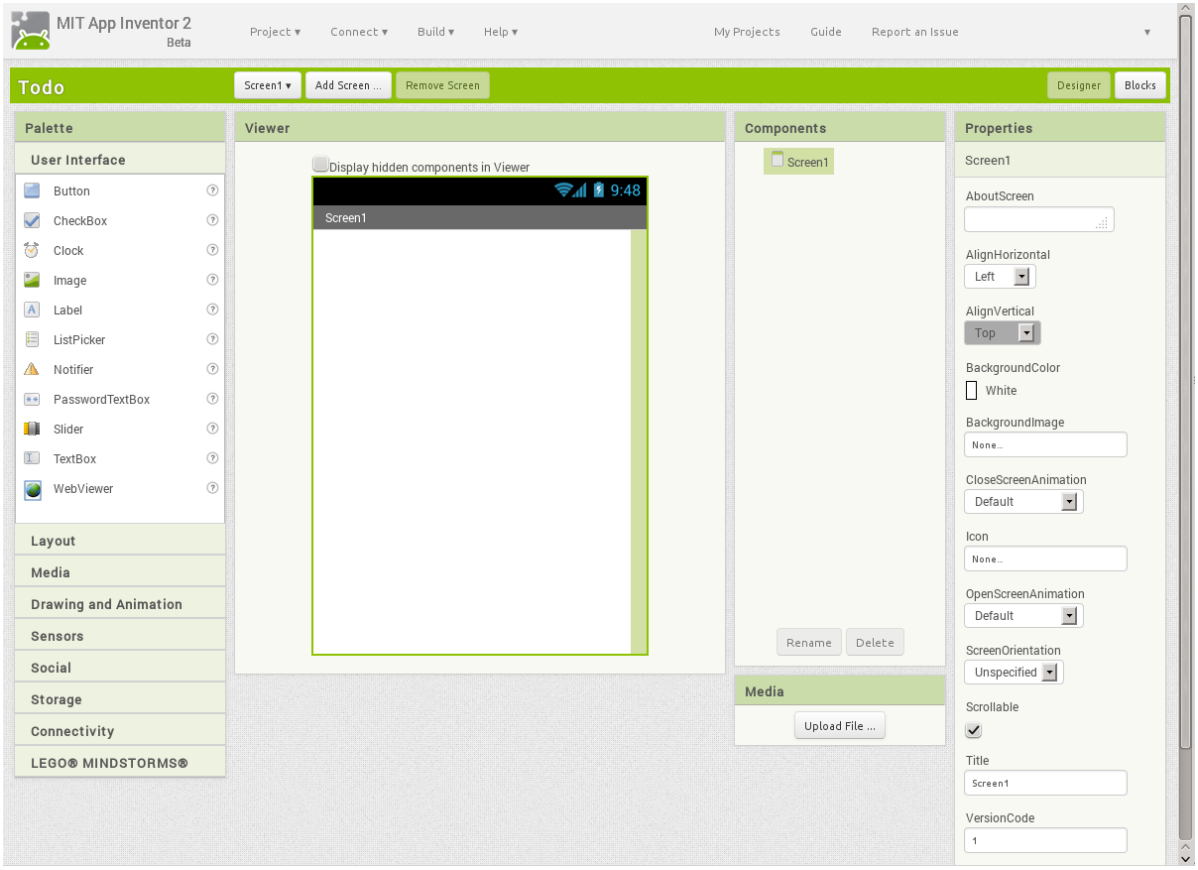
a) Überblick über den MIT App Inventor 2

Der MIT App Inventor 2 findet sich unter http://ai2.appinventor.mit.edu. Es muss ein aktueller Browser (bestenfalls Mozilla Firefox in Version 3.6 oder höher oder Google Chrome in Version 4.0 oder höher) verwendet werden. Der Internet Explorer wird bis dato nicht unterstützt! Nach dem Anmelden über einen Google-Account, einem Klick auf „My Projects" und dem Anlegen eines neuen Projekts über die Schaltfläche „New Project" kann schon mit der Gestaltung der eigenen App begonnen werden. Um ggf. später Apps simulieren zu können (d.h. eine Vorschau der programmierten App, wie sie auf einem Smartphone dargestellt werden würde zu ermöglichen), muss der aiStarter installiert werden. Dieser ist ein *Emulator*. Er kann unter http://appinventor.mit.edu/explore/ai2/setup-emulator.html heruntergeladen werden. Im Abschnitt „Step 1" finden sich auf dieser Seite nähere Informationen und die Downloadlinks der Software für verschiedene Betriebssysteme.

Im Gegensatz zur ersten Version des MIT App Inventors bietet die zweite Version im Wesentlichen abgesehen vom neuen Design in zwei Bereichen Neuerungen: Zum einen wurde der gesamte Entwicklungsworkflow optimiert. Der Blocks Editor, der zuvor ein Java-Programm war, läuft nun ebenfalls online. Mittels des AI Companions ist das Testen von Apps in Echtzeit deutlich angenehmer, da das Resultat einer kleinen Änderung sofort auf dem Smartphone oder im Emulator sichtbar wird. Die zweite wichtige Neuerung findet sich ebenfalls im Blocks Editor. Viele Blöcke sind deutlich flexibler anwendbar (Bedingungen, Schleifen und Funktionen können auf einfache Weise (Klick auf das blaue Zahnrad in der linken oberen Ecke mancher Blöcke) individualisiert werden) als zuvor. Gleichzeitig kann durch die intuitiver gestaltete Oberfläche schneller gearbeitet werden. Kurzum, wer sich zuvor mit der ersten Version des App Inventors beschäftigte, dem dürfte der Umstieg nicht allzu schwer fallen. Das Arbeiten mit der ersten Version des App Inventors ist keinesfalls eine Voraussetzung zum Verständnis dieses Tutorials. In den folgenden Kapiteln wird jeder Aspekt von Grund auf erklärt werden.

Das Fenster (siehe Abbildung 1) ist in vier Teile geteilt: In der Mitte („Viewer") befindet sich ein virtuelles Smartphonedisplay. Dieses kann mit Steuerelementen wie beispielsweise Tasten („Button"), Text („Label"), Eingabefeldern („TextBox") oder Elementen, auf die gezeichnet werden kann („Canvas"), gefüllt werden. Diese Steuerelemente befinden sich im

Abbildung 1: leeres Projekt

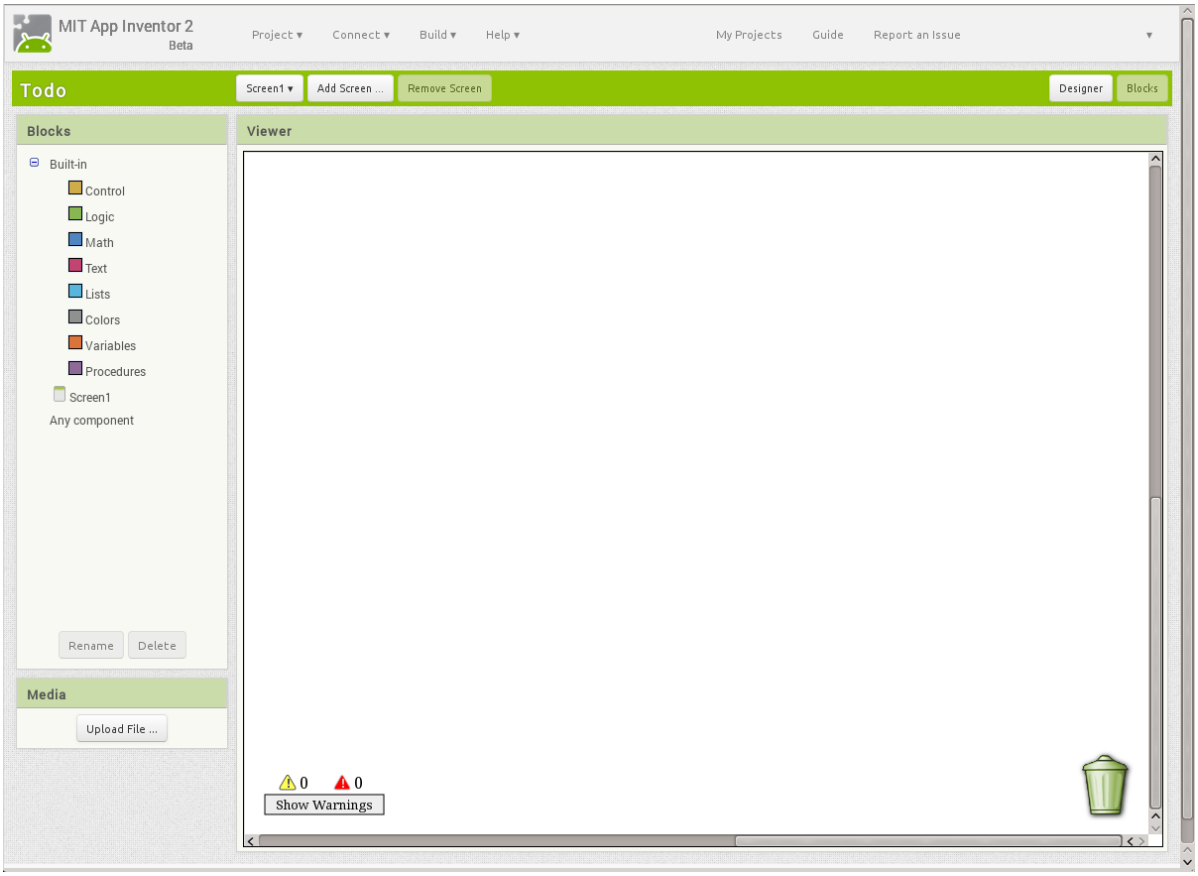


Bereich „Palette“ und können mit der Maus an die gewünschte Position im Viewer gezogen werden. Der Unterpunkt „Layout“ der Palette enthält Steuerelemente, die beim zeilen- oder spaltenweisen Anordnen von Steuerelementen helfen.

Wurde ein Steuerelement platziert, ist es ratsam, ihm einen bezeichnenden Namen zu geben, der beschreibt, welchem Zweck das Element dient (soll ein Button beispielsweise das Programm beenden, könnte ihm der Name „Beenden“ verliehen werden). Hierzu wird das jeweilige Steuerelement angeklickt. Anschließend wird unterhalb der Liste „Components“ auf „Rename“ geklickt, der neue Name eingegeben und bestätigt. „Delete“ löscht das ausgewählte Steuerelement.

Jedes Steuerelement besitzt eine Reihe von Eigenschaften („Properties“), die sein Aussehen und Verhalten bestimmen. Wird ein Steuerelement ausgewählt, können dessen Eigenschaften auf der rechten Seite angepasst werden. Bei einem Button legt die Eigenschaft „Text“ fest, welche Beschriftung er trägt. Die Eigenschaften „Width“ und „Height“ legen die Größe eines Steuerelements fest.

Abbildung 2: Blocks Editor



Auf der rechten Seite, über den „Properties“, befinden sich zwei Tasten „Designer“ und „Blocks“. Mit diesen Tasten kann zwischen der Designer-Ansicht (diese ist aktuell ausgewählt und dem „Blocks Editor“ gewechselt werden (siehe Abbildung 2). Während im Designer nur das Aussehen der App definiert werden kann, kann im Blocks Editor das Verhalten der App programmiert werden. Um eine bestimmte Funktion zu programmieren müssen verschiedene Blöcke wie in einem Puzzle aneinandergereiht werden. Jeder Block beschreibt eine Handlung, die die App durchführt, sobald der jeweilige Block erreicht wird (wie z.B. einen Text anzeigen, zwei Zahlen addieren, einen Ton abspielen, etc.). Nachdem die Handlung eines Blocks ausgeführt wurde, wird die des folgenden Blocks ausgeführt. Durch das Aneinanderreihen verschiedener Blöcke und damit verschiedener Einzelhandlungen wird eine bestimmte Funktion programmiert.

Im Blocks Editor sind die verfügbaren Blöcke in Kategorien eingeteilt (siehe Spalte „Blocks“ auf linker Seite): „Built-In“, eine Kategorie für jeden vorhandenen Screen (standardmäßig „Screen 1“) und „Any component“. Die erste Kategorie enthält Blöcke, mit denen

die Programmlogik definiert wird. Sie ist in Unterkategorien gegliedert: Es sind beispielsweise Blöcke vorhanden, die Text verarbeiten (Unterkategorie „Text“), Blöcke, die Zahlen verarbeiten (Unterkategorie „Math“) und Blöcke, die den Programmablauf steuern (Unterkategorie „Control“). Die Kategorien der vorhandenen Screens enthalten Blöcke, mit welchen auf im Designer platzierte Steuerelemente des jeweiligen Screens zugegriffen werden kann (z.B. abfragen, welcher Text in eine Textbox eingegeben wurde oder auf Drücken eines Buttons reagieren).

Wird eine Unterkategorie angeklickt erscheint rechts daneben eine Liste von Blöcken. Blöcke können ebenfalls mit der Maus in den Arbeitsbereich (Viewer) gezogen werden und dort aneinandergereiht werden. Überflüssige Blöcke werden gelöscht, indem sie auf den Mülleimer unten rechts gezogen und dort fallengelassen werden.

Die folgenden Tutorials gliedern sich jeweils in drei Unterabschnitte: Im ersten Unterabschnitt wird die Gestaltung der jeweiligen App im Designer vorgenommen. Die dazugehörigen Programmierungen werden im jeweils zweiten Unterabschnitt behandelt, während der jeweils dritte Unterabschnitt Aufgaben zur Vertiefung enthält. Diese Aufgaben können oft auf verschiedenen Wegen gelöst werden. Sie sollen zum Reflektieren und Vertiefen des Gelernten anregen. *Kursiv* gedruckte Wörter sind wichtige Begriffe der Informatik, die auf jeden Fall verinnerlicht werden sollten.

b) Testen und Installieren von Apps

Nachdem der aiStarter gestartet wurde, kann die programmierte App durch Klick auf „Emulator“ im Menü „Connect“ (oben neben dem Logo des App Inventors) in einem virtuellen Smartphone, das das Betriebssystem Android emuliert, simuliert werden. Der Emulator startet automatisch. In diesem wird die App gestartet, was einen Augenblick dauern kann. Ist die App einmal gestartet, werden alle Änderungen am Design oder im Blocks Editor automatisch sofort synchronisiert, so dass der Emulator nicht erneut gestartet werden muss. Ist das virtuelle Smartphone gesperrt, muss das Schlosssymbol mit der Maus nach rechts gezogen werden, um die Entsperrung vorzunehmen.

Eine elegantere Methode, die App während der Entwicklung im eigenen Smartphone zu testen (funktioniert nur, wenn Computer und Smartphone dasselbe Netzwerk nutzen oder per USB verbunden sind), besteht darin, auf dem Smartphone die App MIT AI2 Companion zu installieren (verfügbar im Google Play Store) und im Connect-Menü den Eintrag „AI Companion“ zu wählen. Nach dem Abscannen des angezeigten QR-Codes

mit dem MIT AI2 Companion startet die im App Inventor geöffnete App auf dem Smartphone. Alle vorgenommenen Änderungen werden hier ebenfalls sofort synchronisiert, was die Entwicklung der eigenen App ziemlich komfortabel macht. Die Nutzung des Emulators bietet den Vorteil, dass keine Smartphone zum Testen notwendig ist. Allerdings können Sensoren wie der Beschleunigungssensor und spezielle Module wie der Barcode-Scanner im Emulator nicht ausgewertet werden.

Im Menü „Build“ (rechts neben dem Connect-Menü) befinden sich zwei Menüpunkte, über die die entwickelte App entweder lokal auf dem Computer gespeichert („App (save .apk to my computer)“) oder direkt über den MIT AI2 Companion oder einen anderen QR-Code-Scanner auf einem Android-Smartphone installiert werden kann („App (provide QR code for .apk)“). Zumindest temporär muss hierzu die Installation von Apps aus fremden Quellen auf dem Smartphone zugelassen werden.

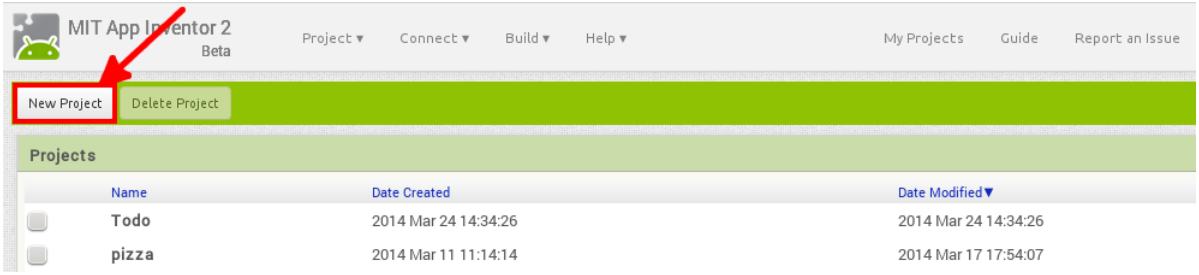
4 Tutorial: Würfel

Ziel dieses sehr einfachen Tutorials ist es, eine kleine App zu programmieren, die beim Schütteln des Smartphones zufällige Werte zwischen 1 und 6 ausgibt. Es wird die grundlegende Funktionsweise des MIT App Inventors demonstriert.

a) Steuerelemente platzieren

Nachdem der App Inventor geöffnet wurde, muss zuerst ein neues Projekt erstellt werden. Es wird auf „New Project“ geklickt, ein passender Name (wie „Wuerfel“) eingegeben und bestätigt (siehe Abbildung 3).

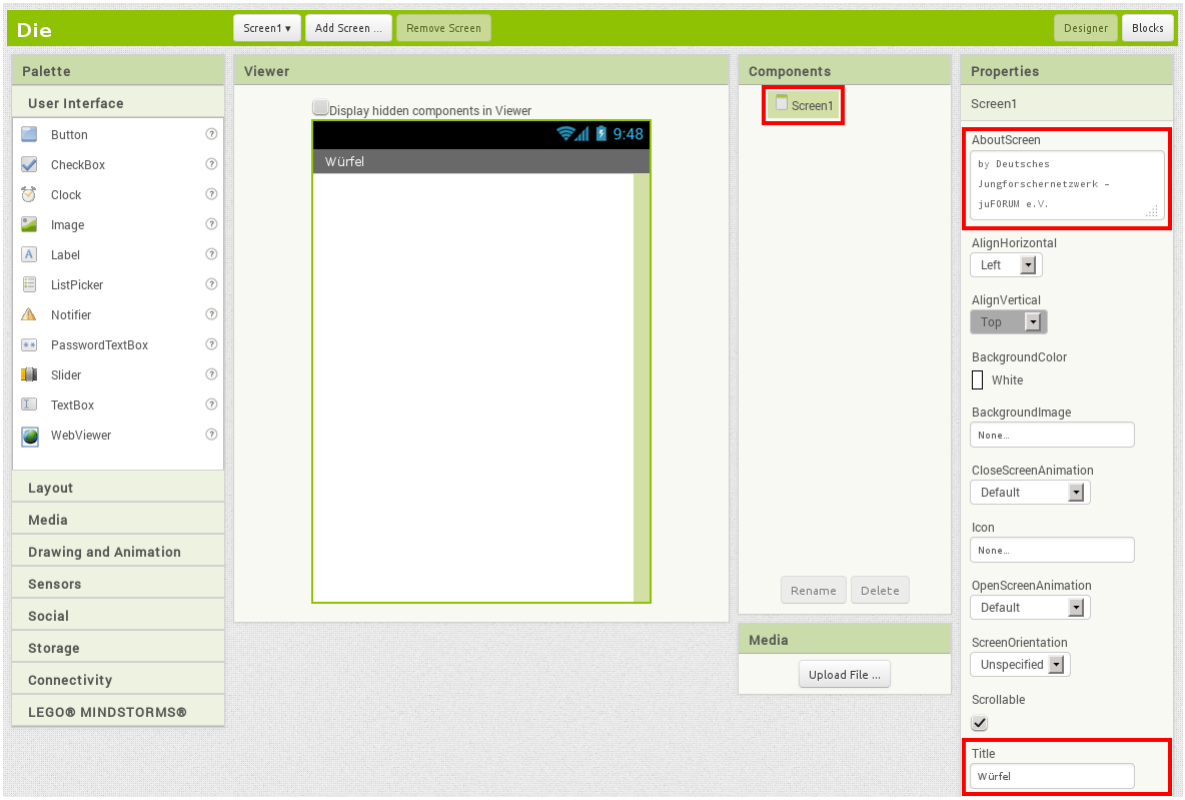
Abbildung 3: Neues Projekt erstellen



Zuerst werden der Titel der App und ein kleiner Kommentar festgelegt. Der Titel erscheint als Überschrift über der App, der Kommentar findet sich in den Informationen zur App („About“) wieder. Dazu wird „Screen1“ im Components-Bereich ausgewählt und enstprechende Texte für die Eigenschaften „Title“ und „AboutScreen“ im Properties-Bereich festgelegt. Der Titel sollte folglich „Würfel“ lauten, um die App treffend zu beschreiben. Der Kommentar ist frei wählbar (siehe Abbildung 4).

Nun kann damit begonnen werden, die benötigten Steuerelemente zu platzieren und deren Eigenschaften festzulegen. Als erstes wird ein „Label“-Steuerelement aus der Palette (Kategorie „User Interface“) mit der Maus in das virtuelle Smartphonedisplay im Viewer gezogen und dort fallen gelassen. Hiermit ist es platziert und erscheint unter Components. Ein Label ist ein einfaches Steuerelement, das einen Text anzeigt. Ein paar weitere Eigenschaften (Properties-Bereich) legen fest, wie der anzuzeigende Text dargestellt werden soll. Das Häkchen bei der Eigenschaft „FontBold“ wird gesetzt, um den Text fett darzustellen. Die Schriftgröße wird über die Eigenschaft „FontSize“ festgelegt. Sie sollte hier 35 betragen. Der initial anzuzeigende Text steht in der Eigenschaft „Text“. Dieser wird später von der

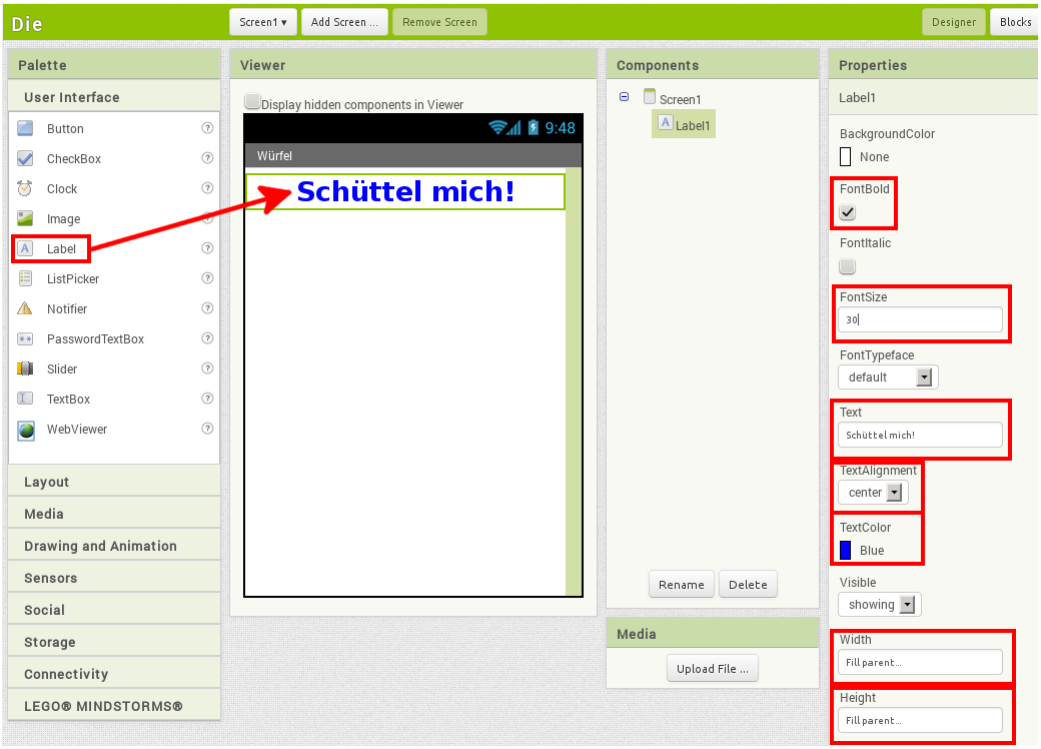
Abbildung 4: Titel und Kommentar der App festlegen



App beim Würfeln auf das Würfelergebnis gesetzt. Zu Beginn lautet er schlicht „Schüttel mich!“. „TextAlignment“ richtet den Text aus. Wenn die Eigenschaft auf „center“ gesetzt wird, wird der Text zentriert. „TextColor“ legt seine Farbe fest (hier blau). Die Eigenschaften „Width“und „Height“ bestimmen die Breite und Höhe des Labels. Sie werden auf „Fill parent...“ gesetzt, um das Label dazu zu bringen, das gesamte Display auszufüllen (siehe Abbildung 5).

Wie bereits erwähnt, ist es für die spätere Programmierung hilfreich, alle Steuerelemente aussagekräftig zu benennen, da sich insbesondere in größeren Apps schnell eine Vielzahl an Steuerelementen gleichen Typs ansammeln können. Nun sollte es bestenfalls so sein, dass der Name eines Steuerelements schon erkennen lässt, um welches konkrete Steuerelement es sich handelt. Es hat sich eingebürgert, den Namen eines Steuerelements aus dem/den Anfangsbuchstaben seiner Typenbezeichnung gefolgt von seinem Zweck zusammenzusetzen. Denkbar für das eben platzierte Label wäre demnach der Name „LErgebnis“ oder der Einheitlichkeit wegen englisch „LResult“. Zum Umbenennen wird zuerst das Label angeklickt (Schritt 1 Abbildung 6). Anschließend wird im Components-Bereich auf Rename geklickt (Schritt 2), der neue Namen („new name“) in das enstprechende Textfeld der sich öffnenden Dialogbox

Abbildung 5: Label platzieren und dessen Eigenschaften festlegen



eingegeben (Schritt 3) und mit einem Klick auf den OK-Button bestätigt (Schritt 4).

Das zweite und in diesem Beispiel letzte benötigte Steuerelement ist der Beschleunigungssensor. Dieser erlaubt das Schütteln des Smartphones zu erkennen. Der Beschleunigungssensor ist ein virtuelles Steuerelement. Es ist nicht sichtbar, stellt jedoch bestimmte Funktionen zur Verfügung. In der Unterkategorie „Sensors“ der Palette findet sich der „AccelerometerSensor“. Dieser wird mit der Maus an einer beliebigen Stelle über dem Smartphonedisplay fallengelassen. Er erscheint als nicht sichtbare („non-visible“) Komponente (siehe Abbildung 7).

Abbildung 6: Label umbenennen

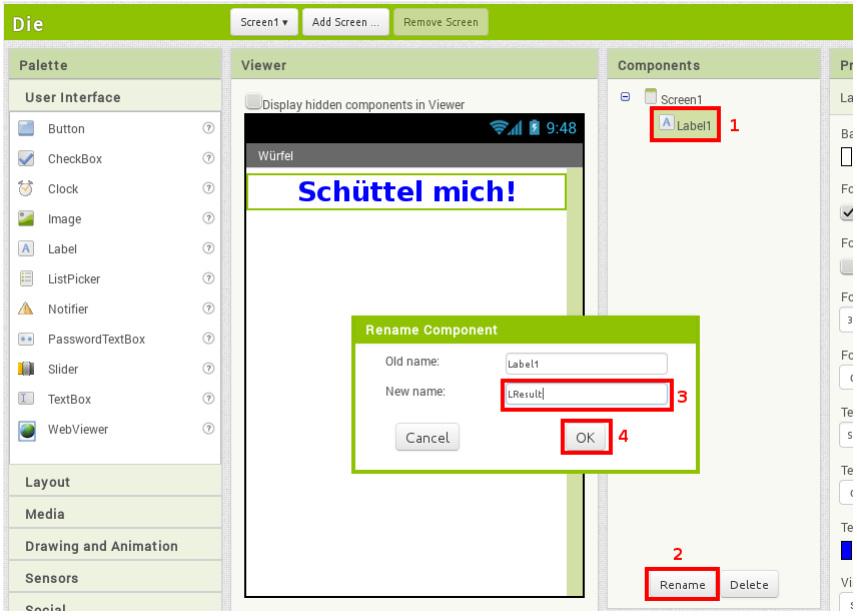


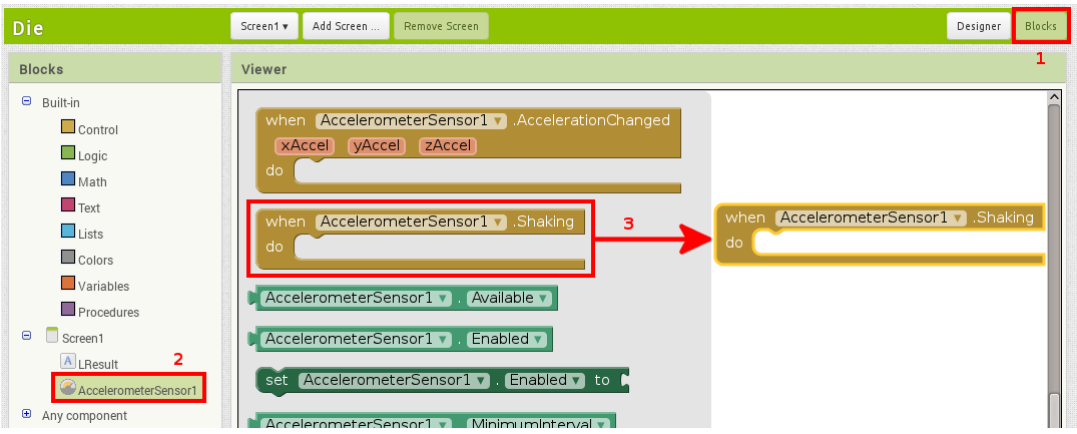
Abbildung 7: Beschleunigungssensor platzieren

b) Verhalten programmieren

Das Aussehen der App ist hiermit festgelegt. Verschiedene Steuerelemente sind vorhanden und einsatzbereit. In diesem Teil des Tutorials wird beschrieben, wie der Blocks Editor dazu verwendet wird, die Programmlogik sowie das Verhalten der App zu programmieren. Der Blocks Editor wird über den Button „Blocks“ rechts oben geöffnet.

Der erste Schritt besteht darin, das Schütteln des Smartphones zu behandeln. Im Blocks-Bereich auf der linken Seite sind in der Kategorie „Screen1“ alle platzierten Steuerelemente aufgelistet. Der Beschleunigungssensor „AccelerometerSensor1“ befindet sich ebenfalls in dieser Liste. Ein Klick auf ihn öffnet eine Liste von Blöcken. Schon der zweite Block dieser Liste ist der Benötigte. Mit der Maus wird er im Arbeitsbereich (Viewer) abgelegt (siehe Abbildung 8). Der Block spannt eine Klammer auf. Er trägt den Namen „when AccelerometerSensor1.Shaking“. Wenn das Smartphone geschüttelt wird, werden die eingeklammerten weiteren Blöcke von oben nach unten ausgeführt. Blöcke der Art des gerade Platzierten folgen dem Benennungsschema „when [Name_des_Steuerelements].[Event]“. Wenn das *Event* eines entsprechenden Steuerelements eintritt, werden die eingeklammerten Blöcke ausgeführt. Das Event entspricht hier dem Schütteln des Smartphones (engl. Shaking). Beispielsweise würde das Event „Click“ eines Buttons genau dann ausgeführt werden, wenn der Button angeklickt wird. Das Event „AccelerationChanged“ des Beschleunigungssensors wird ausgeführt, wenn sich die Beschleunigung des Smartphones ändert. Dieses Event stellt die aktuellen Beschleunigungen in die drei Raumrichtungen als *Variablen* zur Verfügung (es übernimmt sie als *Parameter*) und kann u.a. dazu genutzt werden, angezeigte Objekte in Spielen über Bewegen des Smartphones zu steuern.

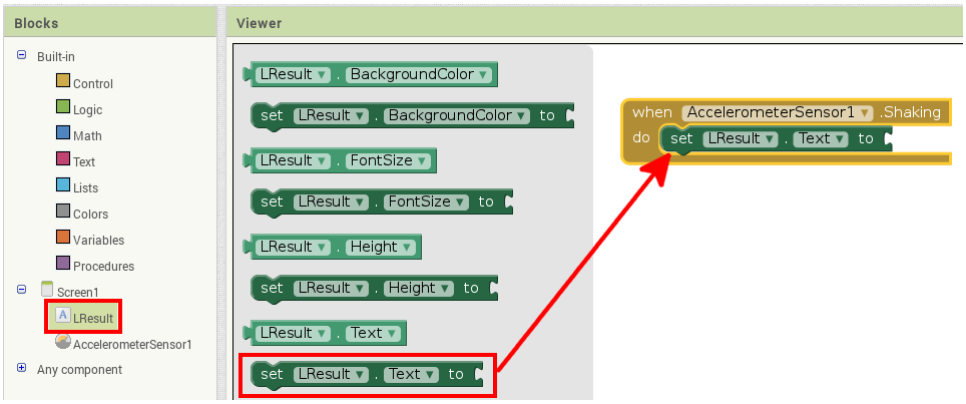
Abbildung 8: Event zum Erkennen des Schüttelns einfügen



Nun kann das Schütteln des Smartphones behandelt werden. Wenn das Schütteln erkannt

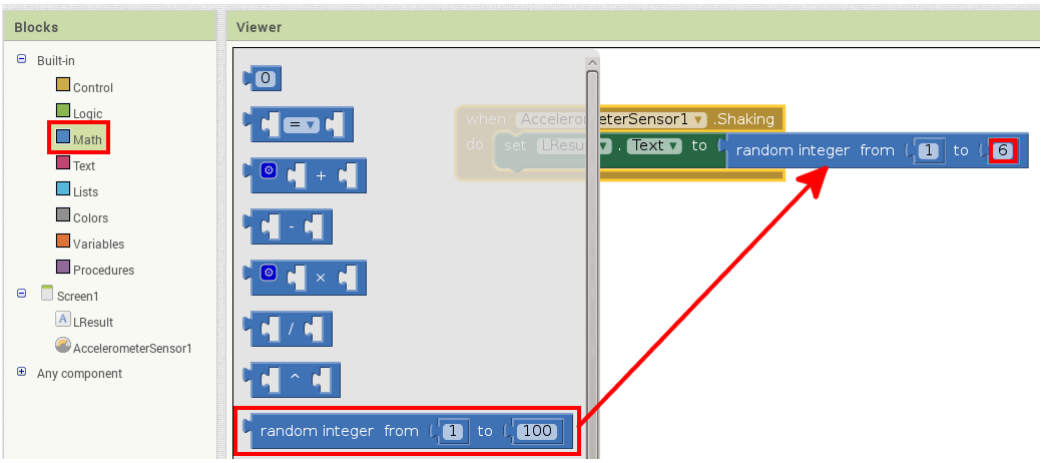
wird, soll der Text des platzierten Labels auf eine Zufallszahl gesetzt werden. Der angezeigte Text kann mittels des Befehls „set LResult.Text to“ geändert werden. set-Befehle überschreiben im Wesentlichen die zur Entwicklung im Properties-Bereich festgelegten Eigenschaften zur Laufzeit. Diese Eigenschaften können zur Laufzeit mit den gleichnamigen Befehlen ohne „set“ am Anfang zur weiteren Verarbeitung gelesen werden. Der gesuchte Block befindet sich unter Screen1 → LResult. Der Block wird innerhalb des ersten Blocks platziert (siehe Abbildung 9). Er hat rechts eine kleine Einbuchtung, in die ein weiterer Block eingefügt werden muss, der den Wert liefert, auf den die Eigenschaft (hier „Text“) gesetzt werden soll.

Abbildung 9: Block zum Setzen des Textes des Labels einfügen



Der letzte Schritt zur Fertigstellung der ersten kleinen App mit dem MIT App Inventor besteht darin, eine Zufallszahl zu generieren und diese in den vorherigen Block einzuspeisen. Der gesuchte Zufallszahlengenerator findet sich unter Screen1 → Math → „random integer fom 1 to 100“. Der Block wird mit seiner Nase in die Einbuchtung des vorhergehenden set-Befehls eingefügt. Da ein Würfel nur Zahlen zwischen 1 und 6 liefern kann, muss die Zahl 100 durch die Zahl 6 ersetzt werden. Hierzu wird die Zahl 100 einfach angeklickt und der Wert editiert (Abbildung 10).

Abbildung 10: Zufallszahlengenerator einfügen



c) Aufgaben

Anmerkung: Die folgenden Aufgaben können eventuell erst nach dem Durcharbeiten des zweiten Projekts, das unten in diesem Tutorial beschrieben wird, beabeitet werden.

Zum Festigen des Gelernten soll dem Würfel eine Animation hinzugefügt werden:

1. Das Label, das das Würfelergebnis enthält, soll durch sechs „Image“-Steuerelemente ersetzt werden, die jeweils ein Bild eines Würfels mit den Augenzahlen 1 bis 6 enthalten. Diese Bilder sollen angezeigt und versteckt werden, je nach dem, welche Zahl angezeigt werden soll. Hinweis: „Visible“-Eigenschaft verwenden.
2. Nach dem Schütteln sollen verschiedene Zufallszahlen nacheinander je für eine bestimmte Zeit angezeigt werden. Nach einer gewissen Zeit soll die Animation beim eigentlichen Würfelergebnis stehen bleiben. Hinweis: Die Komponente „Clock“ bietet einen Timer, der für diese Aufagbe erforderlich ist.
3. Die Animation soll mit der Zeit langsamer werden.

5 Tutorial: Zähler

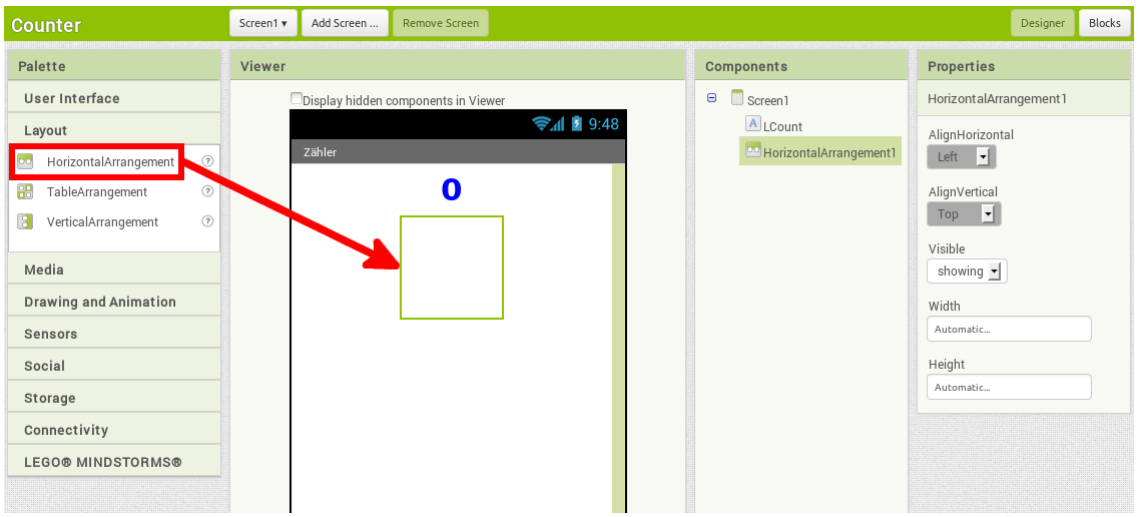
Auf Grundlage des ersten Tutorials soll ein einfacher Zähler programmiert werden, der einen Zählerstand beim Drücken eines Buttons um 1 erhöht (inkrementiert) und beim Drücken eines anderen Buttons auf 0 zurücksetzt.

a) Steuerelemente platzieren

Wie im ersten Tutorial erläutert, wird zuerst das Grundgerüst einer App aufgebaut (Projekt erstellen, Titel festlegen, etc.). Zusätzlich wird die Eigenschaft „AlignHorizontal“ von „Screen1“ auf „Center“ gesetzt, damit alle Steuerelemente zentriert werden. Es wird analog zur ersten App ein Label mit denselben Eigenschaften platziert, sein Name auf „LCount“ gelegt und der Initialtext auf „0“ gesetzt.

Um die zwei Buttons zum Hochzählen und Rücksetzen nebeneinander platzieren zu können, ist ein „HorizontalArrangement“-Container erforderlich. Dieser befindet sich in der Kategorie „Layout“ der Palette. Er wird mit der Maus unterhalb des Labels platziert (Abbildung 11).

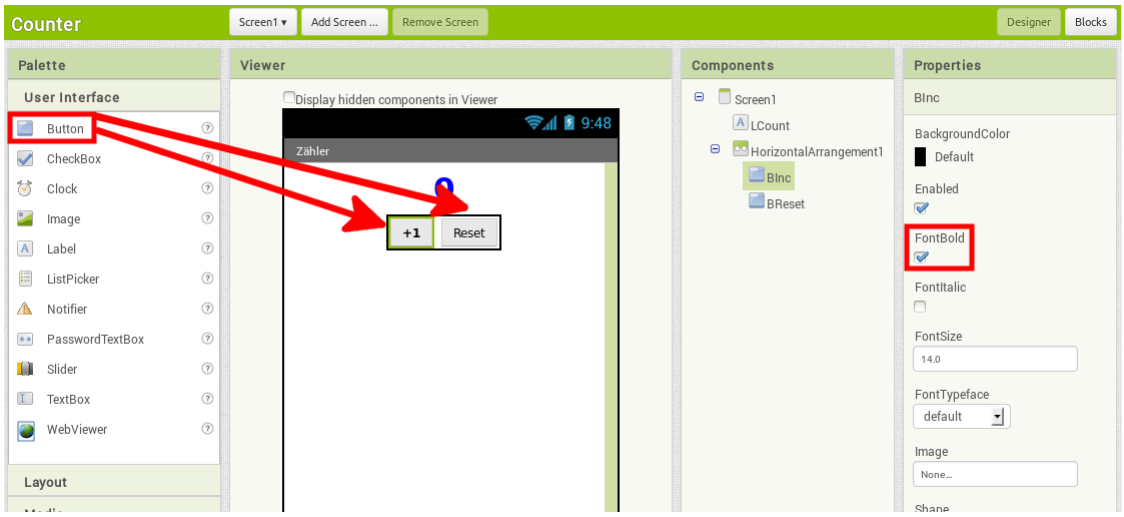
Abbildung 11: HorizontalArrangement einfügen



Die beiden benötigten Buttons finden sich wieder in der Kategorie „User Interface“. Sie werden nebeneinander im HorizontalArrangement angeordnet (Abbildung 12). Der erste Button wird „BInc“ (**B**utton **Inc**rement) und der zweite „BReset“ genannt. Die Beschriftungen (Eigenschaft „Text“) werden im Properties-Bereich entsprechend festgelegt (beispielsweise „+1“

und „Reset“). Falls gewünscht kann das Aussehen der Buttons variiert werden. Hierzu kann mit den weiteren Eigenschaften gespielt werden. Hier wird die Beschriftung des +1-Buttons fett dargestellt (Eigenschaft „FontBold“).

Abbildung 12: Buttons platzieren



b) Verhalten programmieren

Die Gestaltung ist fertiggestellt, es wird zum Blocks Editor gewechselt. Es müssen Klicks auf die beiden Buttons behandelt werden. Wird ein Button gedrückt, wird der im Label angezeigte Text geändert. Um die Klicks abfangen zu können werden wieder Events benötigt. Es handelt sich hier um „when [Button].Click“-Events, die auf dieselbe Weise wie das Shaking-Event aus dem ersten Tutorial arbeiten, allerdings beim Anklicken eines Buttons ausgelöst werden. Sie finden sich im Blocks-Bereich unter Screen1 → HorizontalArrangement1 → BInc → „when BInc.Click“ und Screen1 → HorizontalArrangement1 → BReset → „when BReset.Click“. Beide Blöcke werden in den Arbeitsbereich eingefügt (siehe Abbildung 13).

Da jeweils der Text des Labels „LCount“ modifiziert werden muss, wird in beide Events erneut der Block Screen1 → LCount → „set LCount.Text to“ eingefügt.

Beim Klick auf „BReset“ soll der Zählerstand auf 0 gesetzt werden. Dementsprechend muss die Zahl 0 an die Einbuchtung des entsprechenden set-Blocks angefügt werden. Ein Block, der die Zahl 0 darstellt, findet sich unter Built-in → Math → 0 (Abbildung 14). Um den Zähler inkrementieren zu können, wird eine Addition benötigt (Built-in → Math → +). Diese wird an den anderen set-Block angefügt.

Abbildung 13: Click-Events für „BInc“ und „BReset“ einfügen

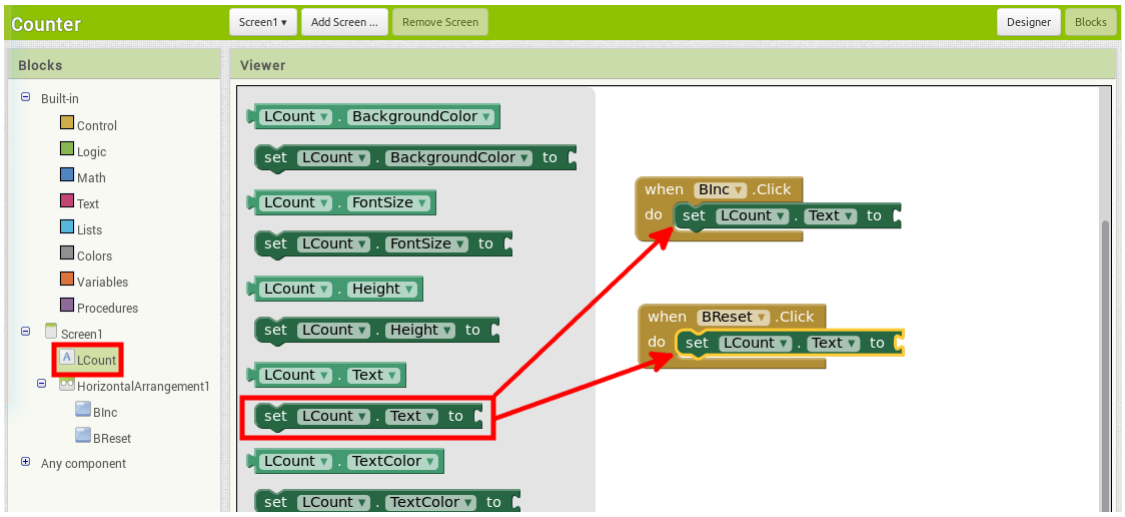
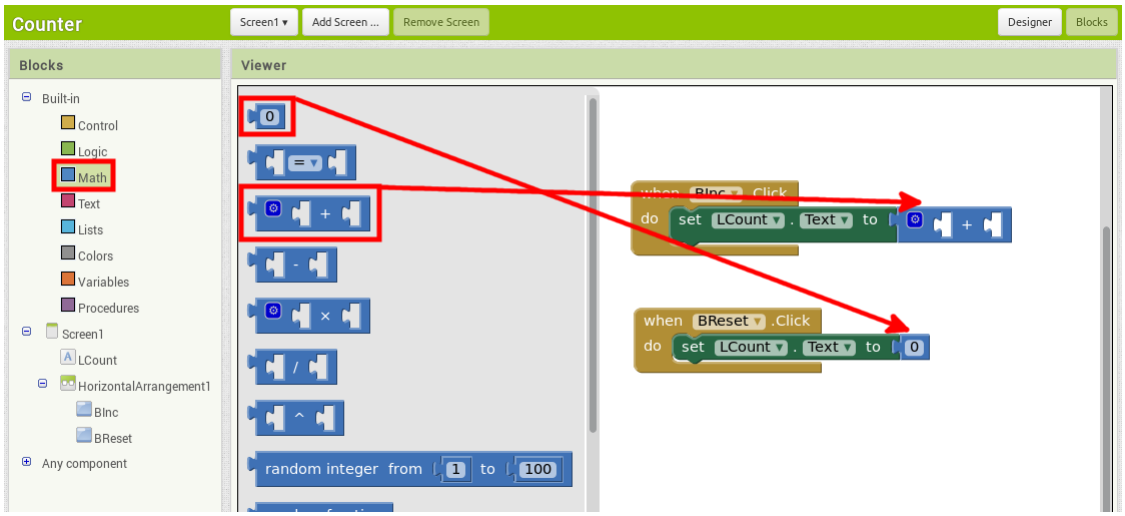
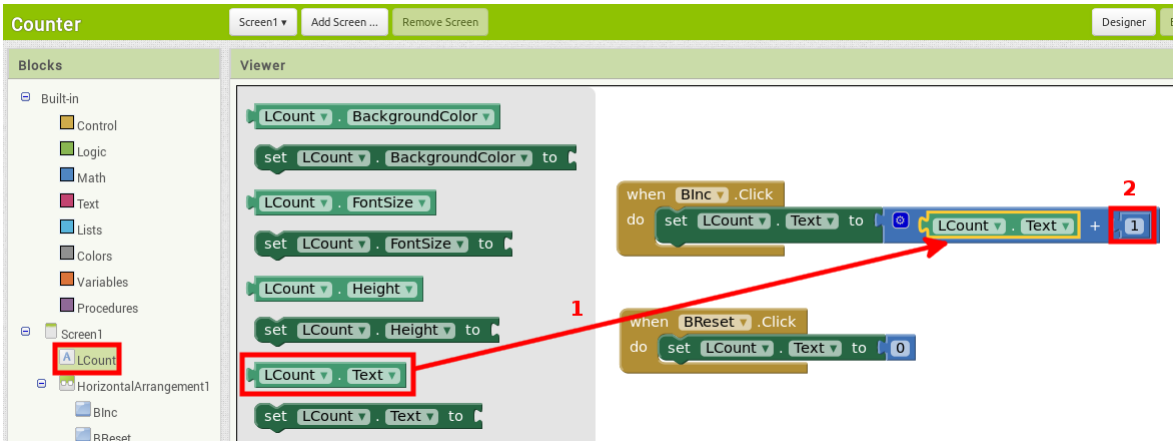


Abbildung 14: Labeltext setzen – einfache mathematische Operationen verwenden



Der Additions-Block besitzt zwei Fehlstellen, in die jeweils die Operanden (das, das addiert werden soll) eingesetzt werden müssen. Es soll eins auf den aktuellen Zählerstand aufaddiert werden. Folglich muss einer der Operanden der aktuelle Zählerstand sein. Da dieser im Label „LCount“ steht, wird er mittels des Blocks Screen1 → LCount → „LCount.Text“ ausgelesen. Dieser Block wird in eine der Fehlstellen des Additions-Blocks eingesetzt (Schritt 1 Abbildung 15). In die zweite Fehlstelle muss eine 1 eingestezt werden, um den Zähler um eins zu erhöhen. Diese findet sich unter Built-in → Math → 0. Die 0 wird angeklickt und editiert, um die 1 zu erhalten (Schritt 2).

Abbildung 15: Addition vervollständigen



c) Aufgabe

Der Zähler soll insofern erweitert werden, dass er einen „-1“-Button besitzt, der den Zählerstand um 1 erniedrigt (dekrementiert).

6 Pizzarechner

Zur Vertiefung des bisher Gelernten soll nun eine kleine App programmiert werden (fertige App vgl. Abbildung 16), die berechnet, wie viele Partypizzen bestellt werden müssen, damit eine bestimmte Anzahl an Personen satt wird und gleichzeitig keine Pizzen übrig bleiben.

Es werden weitere Standardsteuerelemente genutzt und etwas tiefer in die Programmlogik vorgedrungen, um beispielsweise eine einfache Berechnung durchzuführen und um auf ungültige Benutzereingaben mittels des Anzeigens eines entsprechenden Hinweises reagieren zu können.

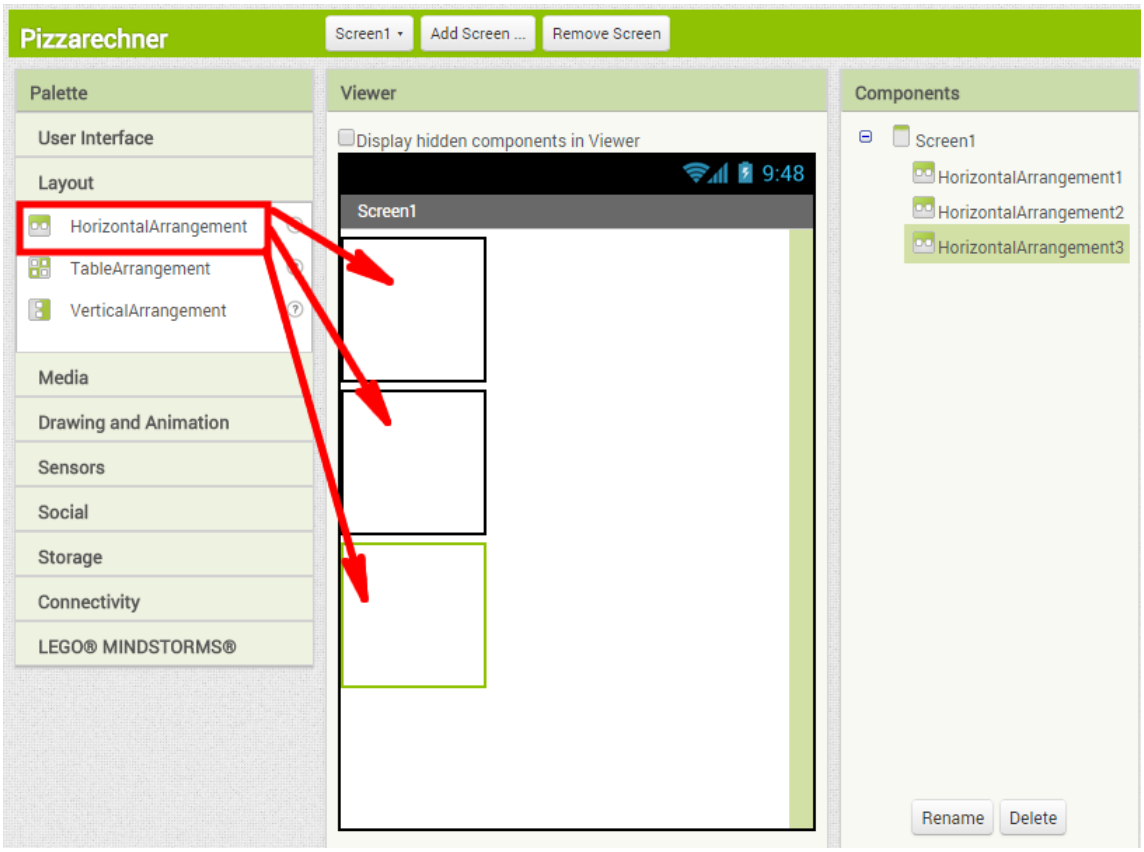
Abbildung 16: Pizzarechner – Die fertige App

a) Steuerelemente platzieren

Zuerst wird ein neues Projekt angelegt und der Titel der App sowie ggf. die About-Information festgelegt (wie zu Beginn des ersten Tutorials erläutert).

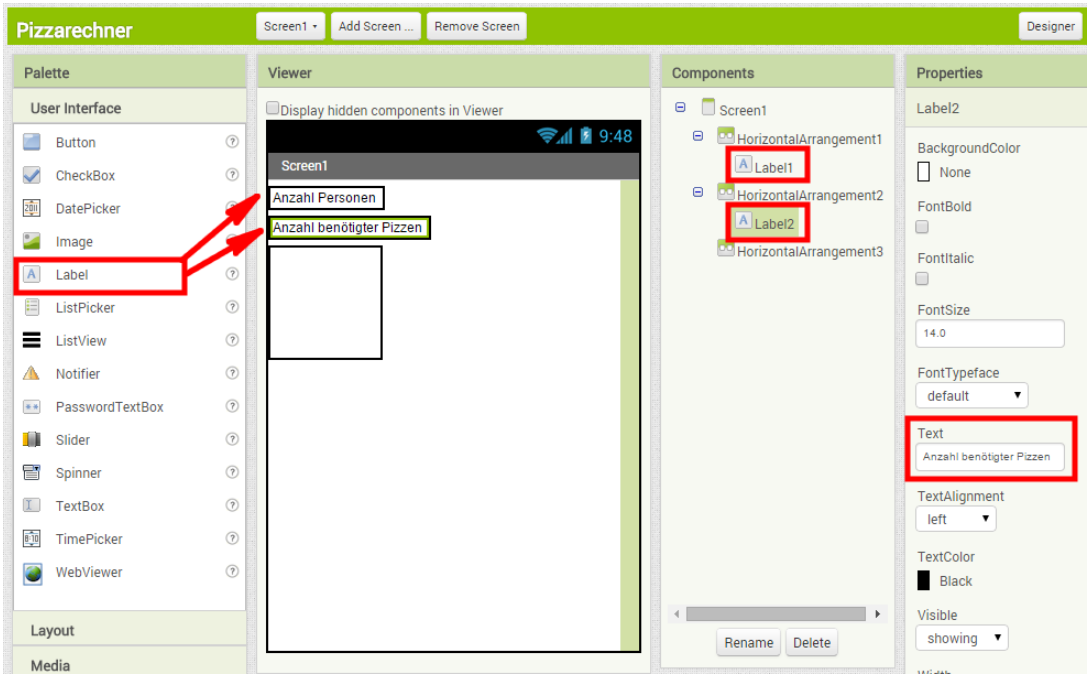
Nun können die Steuerelemente zur Eingabe der Personenzahl, zur Ausgabe der Anzahl der benötigten Pizzen und zur Steuerung der App platziert werden. Um Steuerelemente nebeneinander platzieren zu können (z.B. eine Beschreibung neben einer Textbox), werden Containerelemente des Typs „HorizontalArrangement“ benötigt. Es werden drei dieser Elemente aus der Palette (Subkategorie „Layout“) mit der Maus auf das Smartphonedisplay (Viewer) gezogen und dort fallengelassen (siehe Abbildung 17). Die Steuerelemente sind hiermit platziert und erscheinen unter Components.

Abbildung 17: HorizontalArrangements platzieren



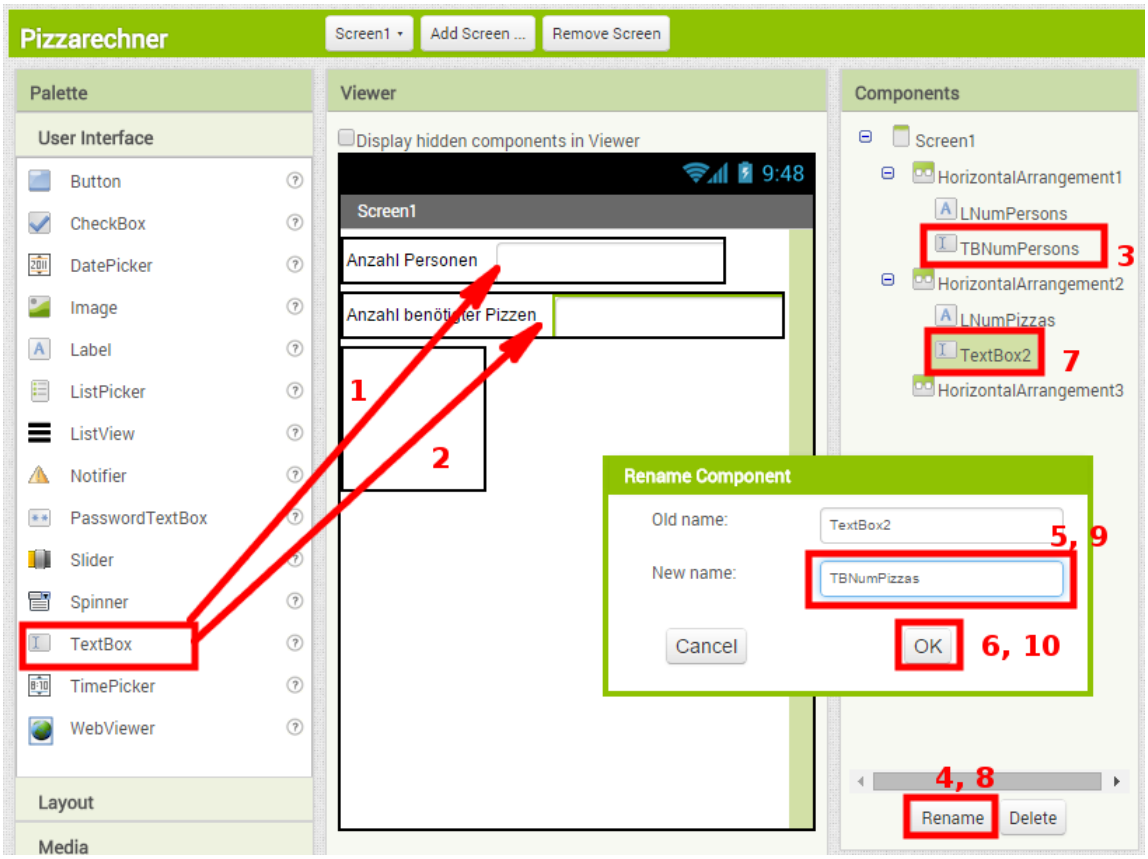
Jeweils ein „Label“ (Subkategorie „User Interface“) wird in den ersten beiden Containern platziert. Die Labels sollen kurze Infotexte darstellen. Die angezeigten Texte können, nachdem das zu bearbeitende Label angeklickt wurde, unter den Properties (Eigenschaft „Text“) bearbeitet werden (siehe Abbildung 18).

Abbildung 18: Labels einfügen und deren Beschriftung anpassen



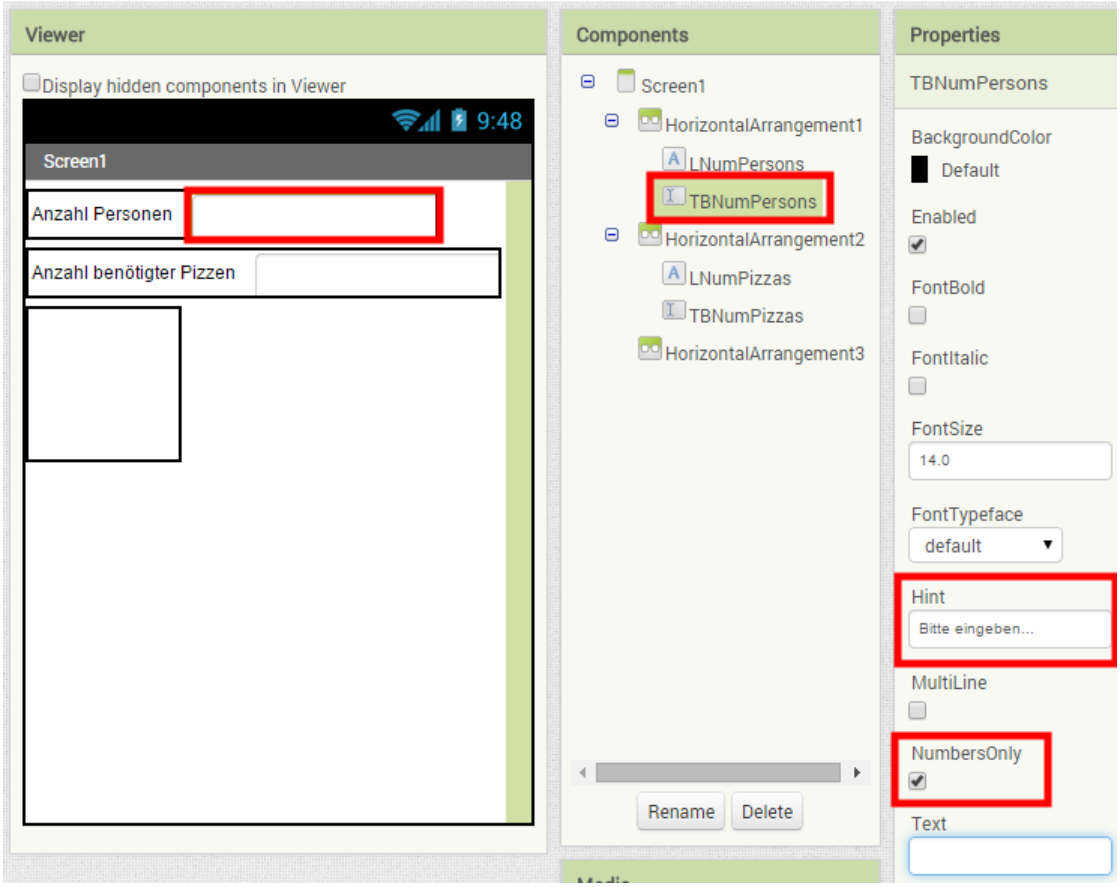
Als nächstes werden Textboxen neben die zuvor platzierten Labels gesetzt (Schritte 1 und 2 Abbildung 19). Es ist sinnvoll, Steuerelementen aussagekräftige Namen zu geben, um später beim Programmieren zu wissen, welches Element sich unter welchem Namen verbirgt. Es wird wie folgt vorgegangen: Die erste Textbox wird ausgewählt, indem der entsprechende Eintrag der Components-Liste angeklickt wird (Schritt 3). Nach einem Klick auf „Rename“ (Schritt 4) öffnet sich eine Dialogbox. Unter „New name“ wird der neue Name eingegeben (Schritt 5) und bestätigt (Schritt 6). Im Beispiel setzt sich dieser aus der Abkürzung des Steuerelements („TextBox“ → „TB“) und dessen Verwendungszweck (hier beispielsweise „NumPersons“ für die Personenanzahl) zusammen. Anschließend wird mit der anderen Textbox analog verfahren (Schritte 7 bis 10).

Abbildung 19: Textboxen platzieren und umbenennen



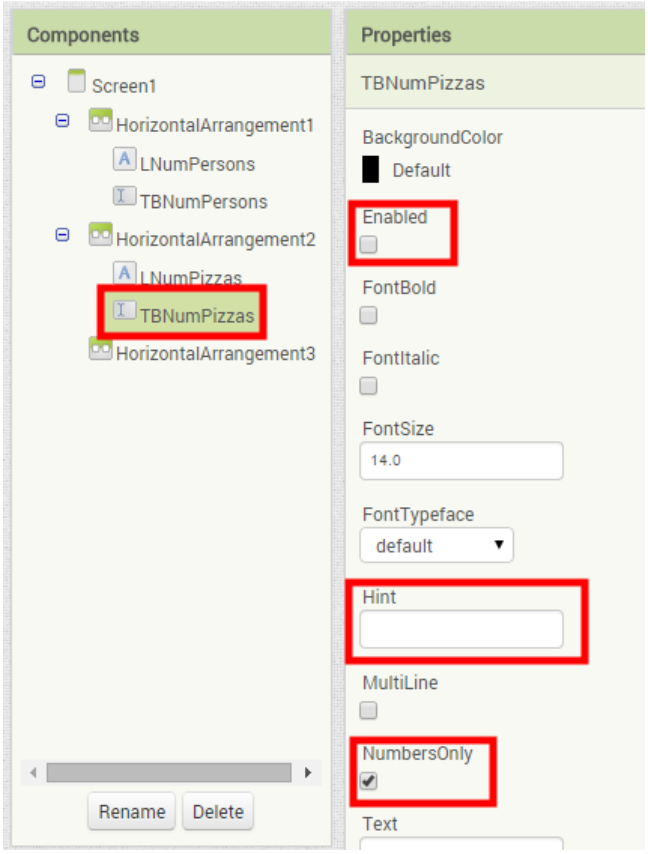
Bezüglich der Textboxen gilt es, ein paar weitere Eigenschaften anzupassen: Die Textbox „TBNumPersons“ soll der Eingabe der Personenzahl dienen. Demnach dürfen hier nur Zahlen eingegeben werden. Dies kann gewährleistet werden, indem die Eigenschaft „NumbersOnly“ unter den Properties gesetzt wird. Android wird zur Eingabe in eine mit dieser Eigenschaft versehenen Textbox eine Tastatur einblenden, die keine Buchstaben enthält. Ferner kann die Eigenschaft „Hint“ modifiziert werden. Der Hinweis (engl. Hint) ist ein kleiner Infotext, der so lange in der Textbox angezeigt wird, bis eigener Text eingegeben wird (siehe Abbildung 20). Zum Ändern von Eigenschaften muss das entsprechende Steuerelement angeklickt worden sein.

Abbildung 20: 1. Textbox „TBNumPersons“ anpassen



Bei der Textbox „TBNumPizzas“ müssen im Wesentlichen dieselben Eigenschaften angepasst werden (siehe Abbildung 21). Der Hint kann hier leer bleiben, da diese Textbox nur zur Ausgabe der berechneten Daten dient. Es muss also kein Hinweistext angezeigt werden, der erklärt, welche Art von Eingaben erwartet werden. Um die Eingabe von Daten zu verbieten (die Textbox soll schreibgeschützt sein), wird das Häkchen bei der Eigenschaft „Enabled“ entfernt. Hiermit reagiert das Steuerelement nicht mehr auf Benutzereingaben. Diese Eigenschaft ist bei beinahe jedem sichtbaren Steuerelement verfügbar.

Abbildung 21: 2. Textbox „TBNumPizzas“ anpassen



Um die App steuern zu können, werden zwei Buttons benötigt. Einer soll die App schließen, der andere soll die Berechnung der Pizzenanzahl starten. Die Buttons werden nebeneinander im untersten „HorizontalArrangement“ platziert (Schritt 1 Abbildung 22) und entsprechend benannt (Schritt 2). Die Beschriftung der Buttons wird festgelegt, indem ihre Eigenschaft „Text“ geändert wird (Schritt 3).

Der letzte Schritt besteht im Hinzufügen eines unsichtbaren Steuerelements: Die „Notifier“-Komponente erlaubt das Anzeigen von aufpoppenden Meldungen (sog. Message Boxes). Sie befindet sich ebenfalls in der Subkategorie „User Interface“ der Palette und wird zum Platzieren an einer beliebigen Stelle des virtuellen Displays fallengelassen (siehe Abbildung 23).

Abbildung 22: Buttons platzieren, benennen und anpassen

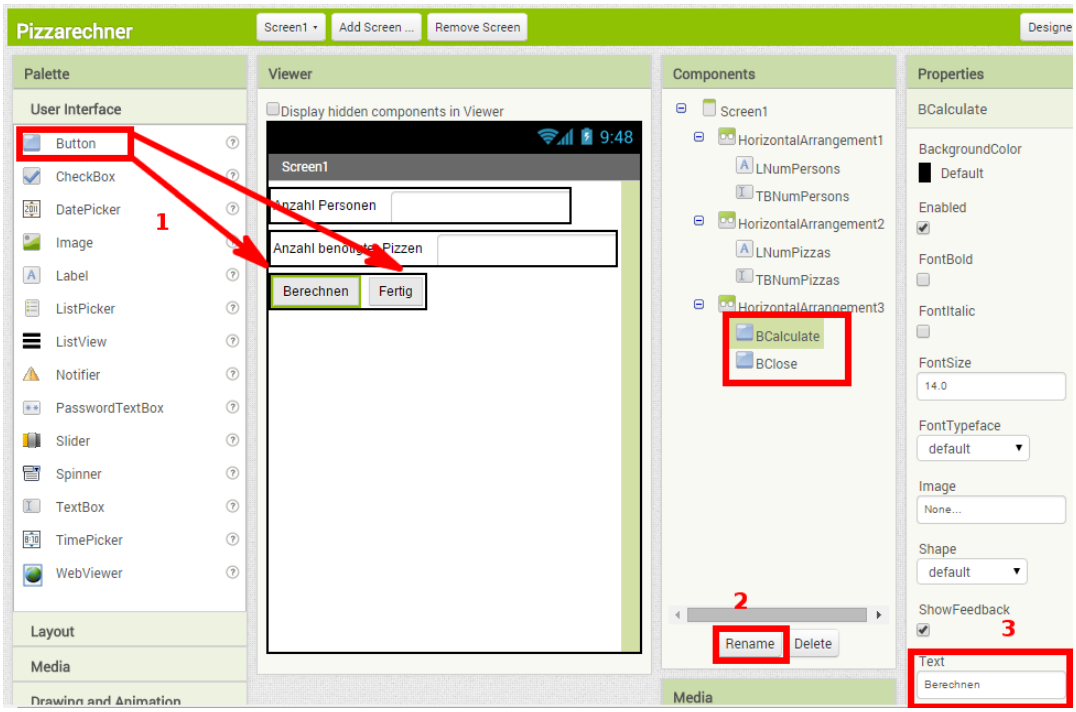
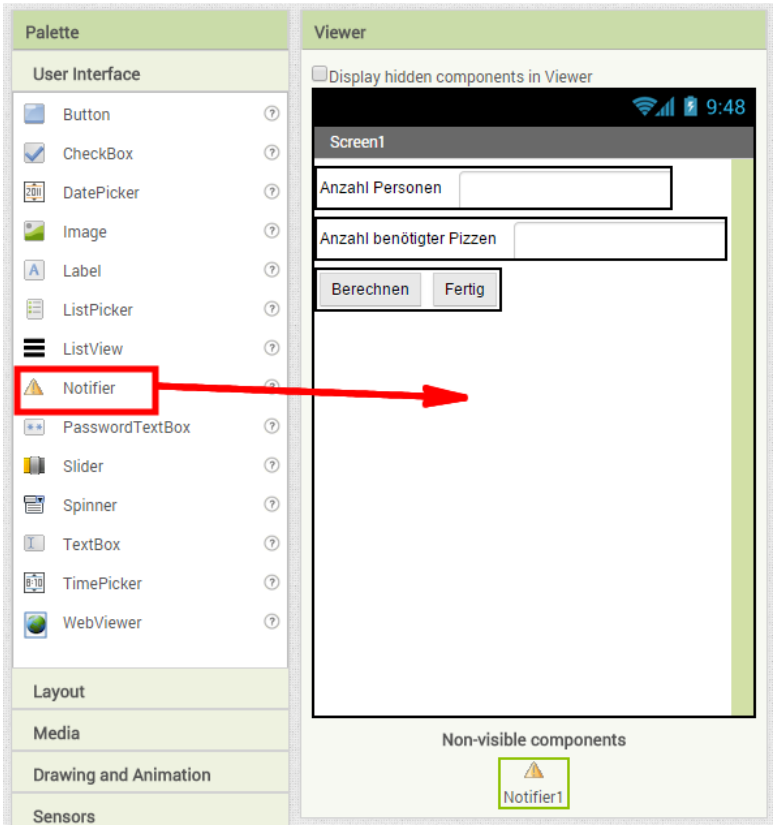


Abbildung 23: Notifier einfügen



Ferner kann der Titel der App, der momentan „Screen1“ lautet, geändert werden, indem im Components-Bereich der entsprechende Eintrag ausgewählt und im Properties-Bereich die Eigenschaft „Title“ modifiziert wird.

b) Verhalten programmieren

Das Aussehen der App ist hiermit festgelegt. Verschiedene Steuerelemente sind vorhanden und einsatzbereit. In diesem Teil des Tutorials wird beschrieben, wie der Blocks Editor dazu verwendet wird, die Programmlogik sowie das Verhalten der App zu programmieren. Zur Erinnerung: Der Blocks Editor wird über den Button „Blocks“ rechts oben geöffnet.

Der erste Schritt besteht darin, einen Klick auf den Button zu behandeln, der die App schließt (hier „BClose“ genannt). Wie schon bekannt, befinden sich im Blocks-Bereich auf der linken Seite in der Kategorie, die den Titel der App trägt, alle platzierten Steuerelemente (siehe Abbildung 24). In der Baumstruktur ist BClose unter dem übergeordneten Container (ein HorizontalArrangement) zu finden. Wird dieser angeklickt, öffnet sich eine Liste von Blöcken. Schon der erste Block dieser Liste ist der Richtige. Mit der Maus wird er im Arbeitsbereich abgelegt. Der Block spannt eine Klammer auf und trägt den Namen „when BClose.Click“. Wenn ein Klick auf BClose erfolgte, werden die vom Block eingeklammerten weiteren Blöcke von oben nach unten ausgeführt. Blöcke dieser Art folgen dem Benennungsschema „when Name_des_Steuerelements.Event“ Wenn das Event eines entsprechenden Steuerelements eintritt, werden die eingeklammerten Blöcke ausgeführt. Das Event entspricht hier einem Klick (engl. Click). Beispielsweise würde das Event „LongClick“ genau dann eintreten, wenn ein lange andauernder Klick auf das zugehörige Steuerelement erfolgt.

In der Kategorie „Built-in“, Subkategorie „Control“ befindet sich weit unten der Block „close application“. Während der zuerst eingefügte Block der Ablaufsteuerung des Programms dient (wie alle eine Klammer aufspannende Blöcke), ist dieser Block ein Befehl. Er schließt die App. Der Block wird wird in die Klammer des „when BClose.Click“-Blocks eingefügt, womit das Klicken auf BClose schon vollständig behandelt wäre (siehe Abbildung 25).

Abbildung 24: Eventhandler für Klickevent von „BClose“ hinzufügen

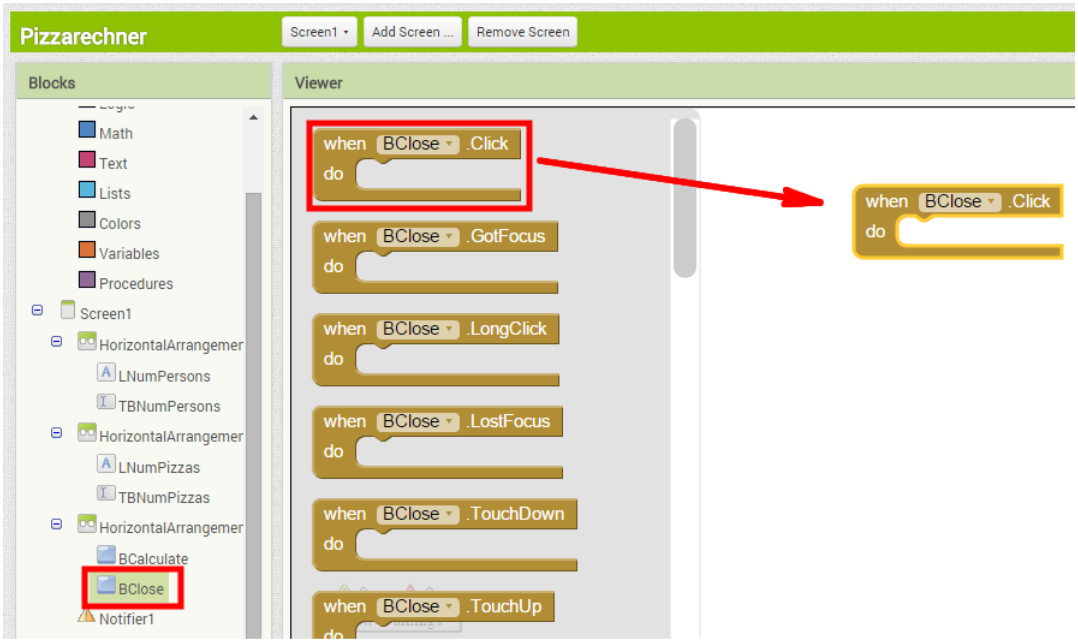
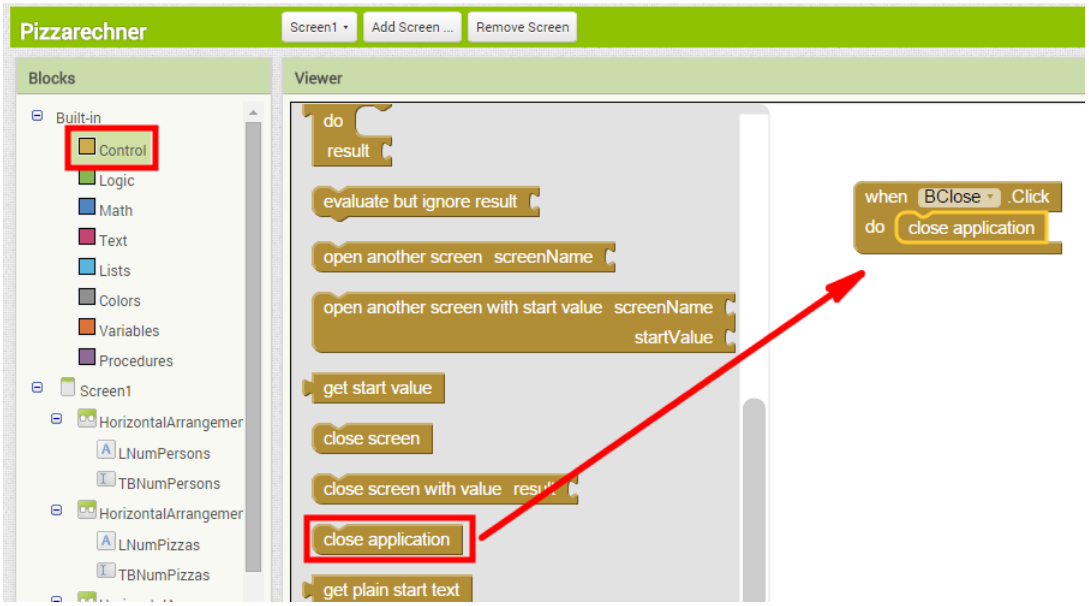


Abbildung 25: Eventhandler für Klickevent von „BClose“ programmieren



Um das Anklicken von BCalculate zu behandeln, wird analog zu „when BClose.Click“ der Block „when BCalculate.Click“ eingefügt. Bevor berechnet wird, wie viele Pizzen nötig sind, muss zuerst überprüft werden, ob die Personenanzahl einen sinnvollen Wert enthält. Zwar wurde bei der Textbox TBNumPersons die Eigenschaft „NumbersOnly“ gesetzt, die ga-

rantiert, dass keine Buchstaben eingegeben werden, dennoch muss überprüft werden, ob überhaupt eine Zahl eingegeben wurde und ob diese größer als 0 ist.[3] Zu diesem Zweck wird in den gerade eben platzierten Block der ebenfalls in der Kategorie Built-in → Control verfügbare „if"-Block (engl. für „falls") gesetzt (siehe Abbildung 26). (Wichtig: Es existieren zwei ähnliche „if"-Blöcke. Es muss derjenige mit dem blauen Quadrat in der linken oberen Ecke gewählt werden.)

Abbildung 26: Eventhandler für Klickevent von „BCalculate" programmieren – Schritt 1: Bedingung einfügen

Der „if"-Block stellt eine Bedingung dar. Von diesem Block umschlossene Blöcke werden nur dann ausgeführt, wenn der Ausdruck, der sich an den „if"-Block rechts andocken lässt, wahr ist. Die Berechnung der Anzahl benötigter Pizzen soll nur dann ausgeführt werden, wenn die Prüfung der Eingabedaten erfolgreich ist. Anderenfalls soll eine Fehlermeldung ausgegeben werden. Damit keine zwei Bedingungen notwendig sind, kann der „if"-Block zu einem „if-else"-Block (engl. für „falls-sonst") erweitert werden. Dieser besitzt zwei Zweige: Der erste

[3] Es kann nicht zu oft daran erinnert werden, dass alle Benutzereingaben stets auf Gültigkeit zu prüfen sind, da sie anderenfalls eine App zum Absturz bringen können. Um dies zu illustrieren kann eine App programmiert werden, die zwei Zahlen durcheiannder teilt. Wird aufgrund einer Benutzereingabe durch 0 geteilt, wird die App ohne sinnvolle Fehlermeldung vom Smartphone beendet. Etwaige geöffnete Daten können nicht mehr gespeichert werden und sind verloren. Schlimmstenfalls kann eine nicht überprüfte Benutzereingabe ein Sicherheitsrisiko darstellen, das Hackern erlaubt, die Kontrolle über ein Gerät zu übernehmen.

Zweig wird genau dann ausgeführt, wenn der angedockte Ausdruck wahr ist, der zweite Zweig genau dann, wenn der angedockte Ausdruck falsch ist. Wörtlich: Falls der zu testende Ausdruck wahr ist, dann führe den ersten Zweig aus, sonst führe den zweiten Zweig aus. Um dies zu erreichen wird das blaue Quadrat in der linken oberen Ecke des „if“-Blocks angeklickt. Es öffnet sich ein Feld, das es erlaubt, „else if“- und „else“-Bausteine in den „if“-Block einzufügen (siehe Abbildung 27). Der „else“-Baustein wird in den „if“-Block gezogen (siehe Abbildung 28). Durch entsprechendes Erweitern des Blocks mit den genannten Bausteinen können sich Fallunterscheidungen mit mehr als zwei Fällen (verschachtelte Bedingungen) einfach auswerten lassen.

Die hier benötigte Bedingung setzt sich aus zwei Teilbedingungen zusammen. Die erste Teilbedingung testet, ob in TBNumPersons **keine** Zahl eingegeben wurde, die Zweite testet, ob die eingegebene Zahl (sofern eine eingegeben ist) negativ ist. Insgesamt soll, sobald die eine **oder** die andere Bedingung erfüllt ist, ein Hinweistext eingeblendet werden, der den Benutzer auffordert, eine gültige Zahl einzugeben. Um beide Bedingungen testen zu können ist demnach ein „or“-Block (engl. für „oder“) erforderlich. Dieser wird an den „if-else“-Block angehängt (Schritt 1 Abbildung 29). Der „or“-Block befindet sich unter Built-in → Logic. Der erste Zweig des Blocks testet, ob **keine** Nummer eingegeben wurde. Ein „not“-Block (engl. für „nicht“) gefolgt von einem „is a number?“-Block (engl. für „ist eine Zahl?“) werden demnach benötigt. Der erste Block befindet sich ebenfalls unter Built-in → Control (Schritt 2), der Zweite unter Built-in → Math (Schritt 3). Der „is a number?“-Block soll den in TBNumPersons eingegebenen Text überprüfen. Dieser kann über den Block „TBNumPersons.Text“ verarbeitet werden (Schritt 4). Der Block folgt dem Benennungsschema „Name_des_Steuerelements.Eigenschaft“.

Die zweite Teilbedingung überprüft, ob der Inhalt von TBNumPersons kleiner ($<$) oder gleich ($=$) 0 ist. Dazu wird der „$=$“-Block (Built-in → Math) in die zweite Aussparung des „or“-Blocks eingesetzt. Es wird auf das $=$-Zeichen des Blocks geklickt und aus dem erscheinenden Menü $\leq$ ausgewählt (Schritt 1 Abbildung 30). In die zweite Aussparung des „$\leq$“-Blocks wird über den „0“-Block (Built-in → Math) die Zahl 0 eingefügt (Schritt 2). Durch Klicken auf die Zahl 0 kann diese editiert werden. So stehen selbstverständlich für andere Anlässe beliebige andere Zahlen ebenso zur Verfügung. Dies wird unter anderem weiter unten benötigt. In die erste Aussparung wird erneut der „TBNumPersons.Text“-Block eingefügt (Schritt 3).

Abbildung 27: Eventhandler für Klickevent von „BCalculate“ programmieren – Schritt 2: Bedingung anpassen

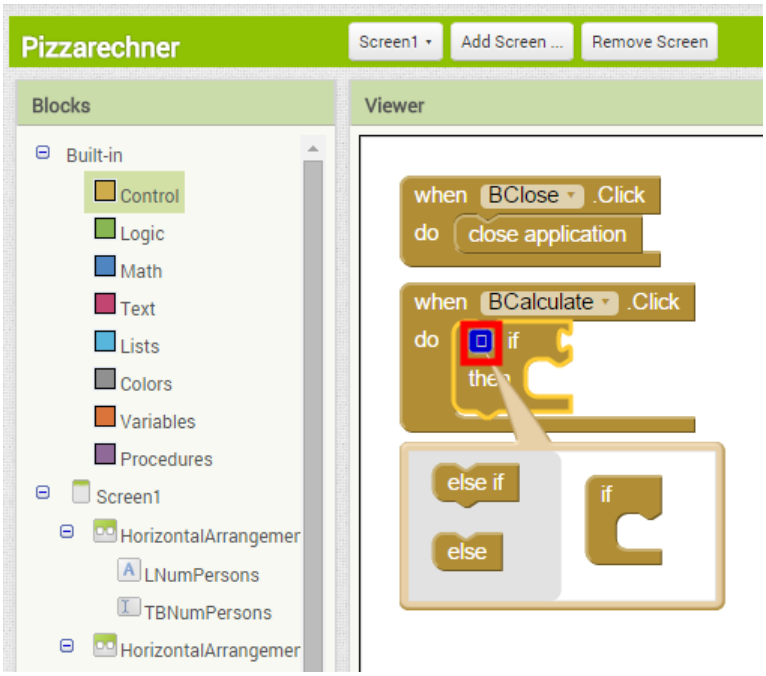


Abbildung 28: Eventhandler für Klickevent von „BCalculate“ programmieren – Schritt 3: Bedingung anpassen

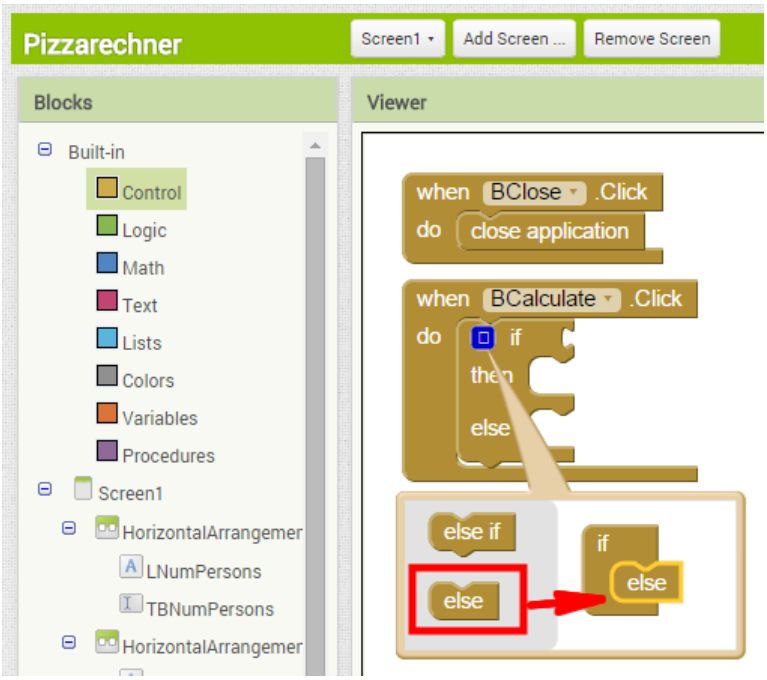


Abbildung 29: Eventhandler für Klickevent von „BCalculate“ programmieren – Schritt 4: Prüfen, ob Eingabe numerisch ist

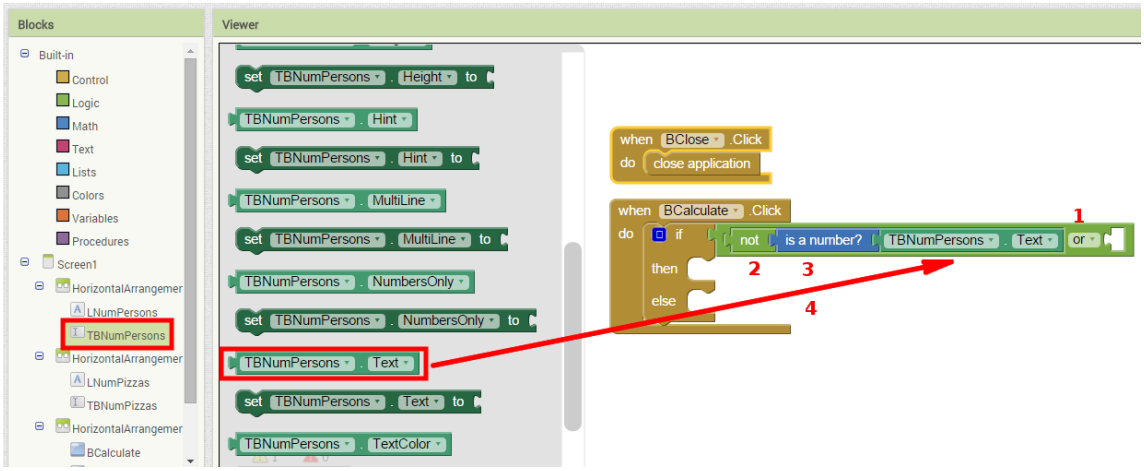
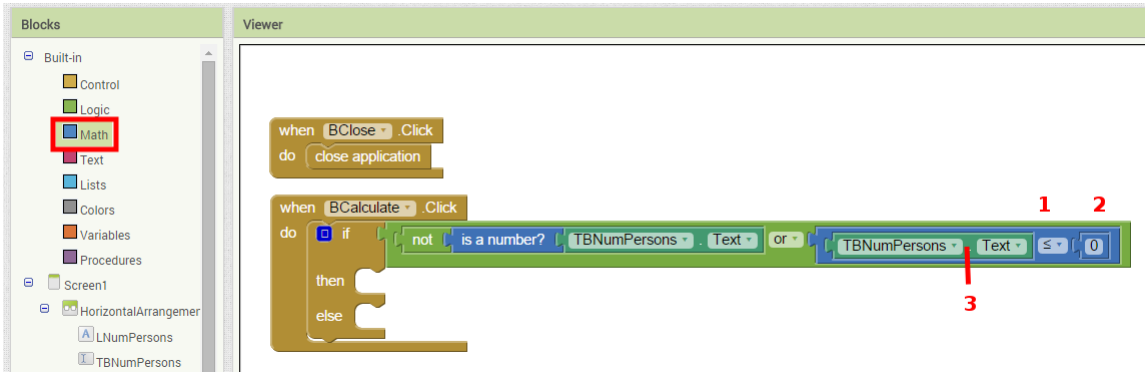


Abbildung 30: Eventhandler für Klickevent von „BCalculate“ programmieren – Schritt 5: Prüfen, ob Eingabe positiv



Ist mindestens eine der oben beschriebenen Teilbedingungen erfüllt, soll eine Meldung an den Benutzer ausgegeben werden. Dazu bietet sich die im letzten Schritt des Platzierens der Steuerelemente eingefügte „Notifier“-Komponente an. Es findet sich nach einem Klick auf diese Komponente der Block „call Notifier1.ShowAlert“. Dieser wird in der ersten Klammer des „if-else“-Blocks eingefügt (siehe Abbildung 31). Sobald der Block ausgeführt wird, lässt er den in seinen Eingang „notice“ eingegebenen Text aufpoppen. Ein „ “-Block (Built-in → Text) stellt Text bereit. Dieser Block wird demnach an den Eingang gehängt. Der auszugebende Text lässt sich durch Klicken auf den Bereich zwischen der Anführungszeichen setzen (siehe Abbildung 32).

Abbildung 31: Eventhandler für Klickevent von „BCalculate“ programmieren – Schritt 6: Fehlermeldung bei ungültiger Eingabe ausgeben

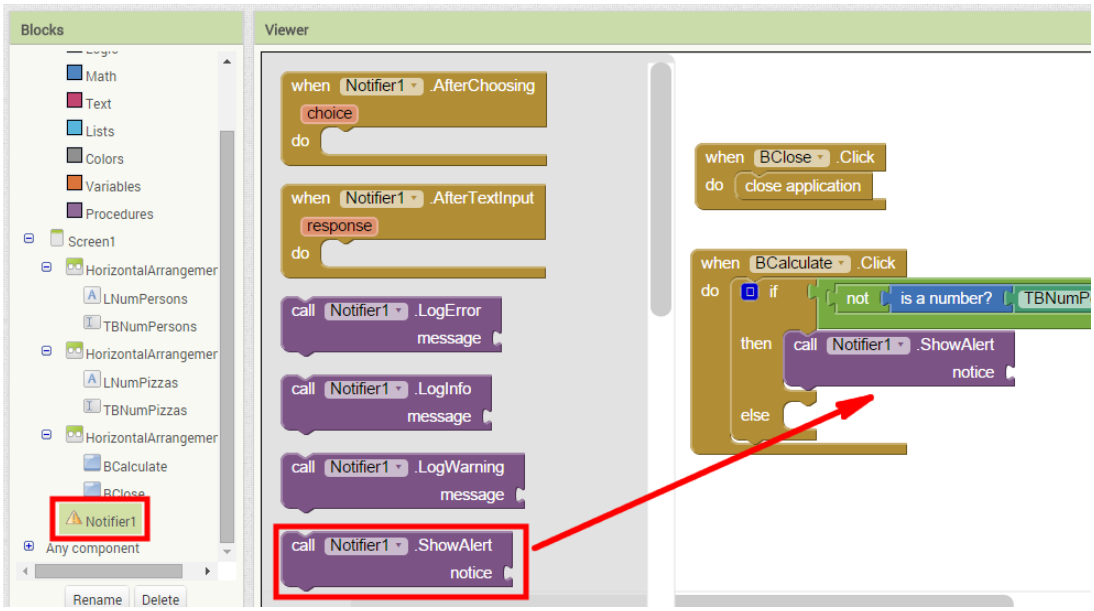
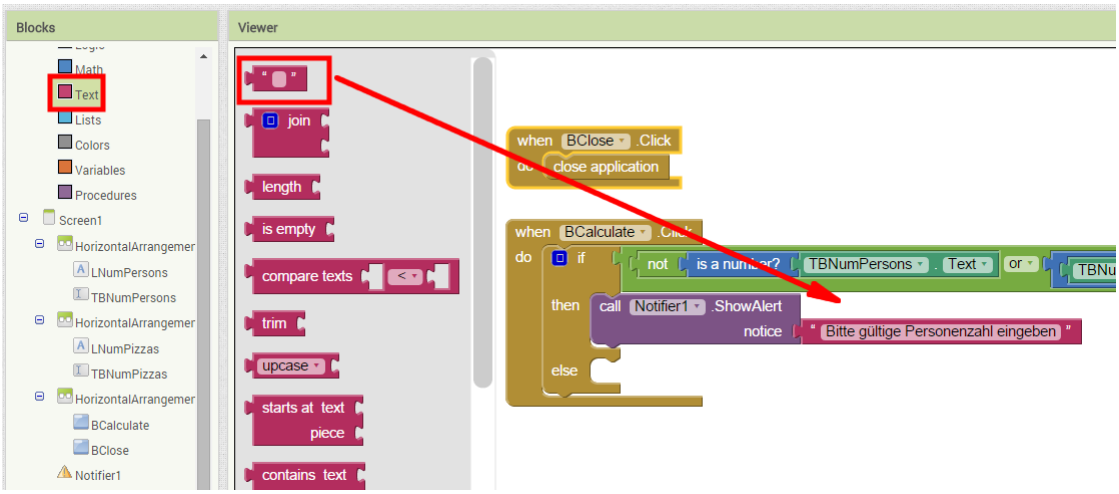


Abbildung 32: Eventhandler für Klickevent von „BCalculate“ programmieren – Schritt 7: Fehlertext ausgeben



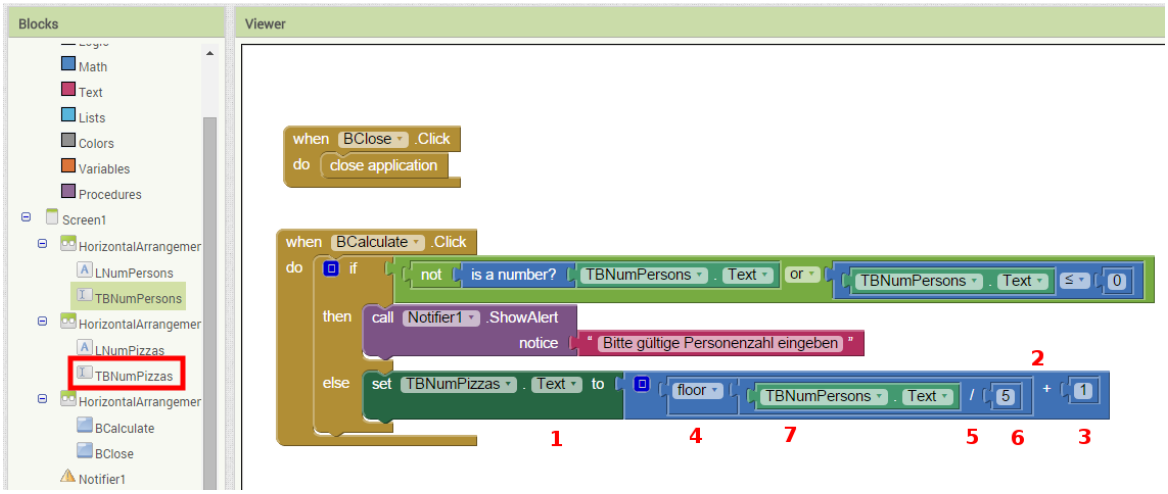
Ist keine der Teilbedingungen erfüllt, kann endlich die Anzahl der benötigten Pizzen berechnet und in TBNumPizzas ausgegeben werden. Zuvor wurde die Eigenschaft „Text“ der Textbox TBNumPersons gelesen. Nun soll die gleichnamige Eigenschaft von TBNumPizzas geschrieben werden. Hierzu steht im Blocks-Bereich nach Klick auf TBNumPizzas der Befehl „set TBNumPizzas.Text“ zur Verfügung. Dieser wird in den zweiten Zweig des „if-else“-Blocks eingefügt (Schritt 1 Abbildung 33). Mit entsprechenden Blöcken aus Built-in → Math wird

folgende Formel zur Berechnung der Pizzenanzahl umgesetzt:

$$AnzahlPizzen = floor\left(\frac{AnzahlPersonen}{5}\right) + 1$$

Die Funktion $floor$ rundet dabei immer ab. Es wird ein „+“-Block an den zuvor eingefügten Block angefügt (Schritt 2). In dessen zweite Aussparung wird ein Zahlenblock der Zahl 1 eingesetzt (Schritt 3). In die linke Aussparung wird ein „floor“-Block gelegt (Schritt 4). Dessen Aussparung wird mit einem „/“-Block gefüllt, der wiederum selbst zwei Aussparungen besitzt (Schritt 5). Ein Zahlenblock der Zahl 5 wird in die rechte Aussparung (Schritt 6) und der „TBNumPersons.Text“-Block in die linke Aussparung (Schritt 7) gelegt.

Abbildung 33: Eventhandler für Klickevent von „BCalculate“ programmieren – Schritt 8: Berechnung der Pizzenanzahl



Jetzt ist die App fertiggestellt. Sie kann nun simuliert oder besser noch auf das Smartphone übertragen werden und bei nächster Gelegenheit unter realen Bedingungen getestet werden. Wie dies funktioniert ist in Testen und Installieren von Apps nachzulesen.

Obwohl das Überprüfen der Gültigkeit von Benutzereingaben oft einen großen Teil des eigentlichen Programms ausmacht, gehört dies zum guten Programmierstil. Es gewährleistet, dass eine App durch fehlerhafte oder manipulative Benutzereingaben nicht abstürzt oder gar Schaden auf dem Smartphone anrichtet.

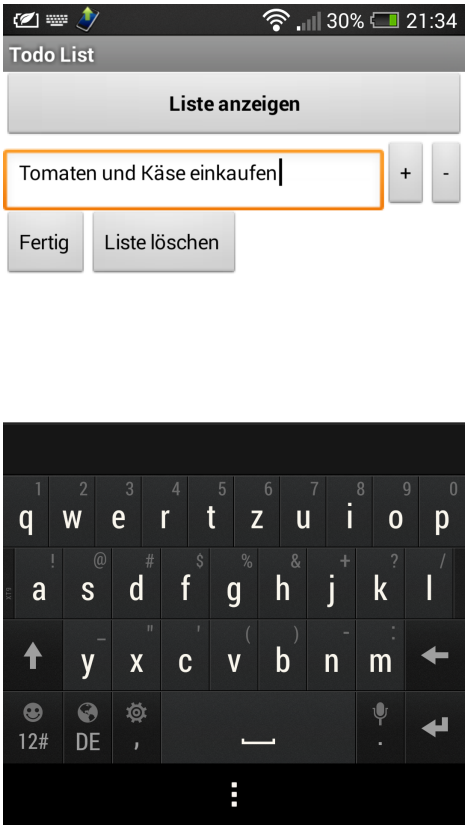
c) Aufgaben

1. Der Pizzarechner soll dahingehend erweitert werden, dass er den Endpreis der benötigten Pizzen anzeigt. Hierzu müssen zwei weitere Textfelder implementiert werden. In das erste Feld wird der Stückpreis pro Pizza eingetragen. Im anderen Feld soll der Gesamtpreis ausgegeben werden. Es sollte unbedingt auf gültige Nutzereingaben geachtet werden.
2. Ferner sollen die pro Person anfallenden Kosten berechnet und angezeigt werden.
3. Die zur Berechnung der benötigten Pizzen angewandte Formel ist relativ einfach gehalten und nicht immer präzise. Kann eine bessere Methode zur Berechnung der Anzahl der nötigen Pizzen gefunden und implementiert werden?

7 Tutorial: Todo-Liste

Im Folgenden soll eine App programmiert werden (fertige App vgl. Abbildung 34), die als einfache Todo-Liste dient. Einträge können zur Liste hinzugefügt und wieder entfernt werden. Alle Listeneinträge werden auf dem Smartphone gespeichert und bleiben demnach beim Beenden der App erhalten. Es wird gezeigt, wie verschiedene Standardsteuerelemente verwendet werden und wie mit einfachen Datenstrukturen gearbeitet wird. Dieses Tutorial ist recht anspruchsvoll und setzt voraus, dass das bisher Erklärte restlos verstanden wurde. Falls dies bis zu dieser Stelle nicht der Fall ist, sollten entsprechende Abschnitte dieses Tutorials wiederholt, anderweitig Informationen eingeholt und geübt werden. Es werden einige neue Blöcke verwendet, deren Bedeutung knapper erklärt wird.

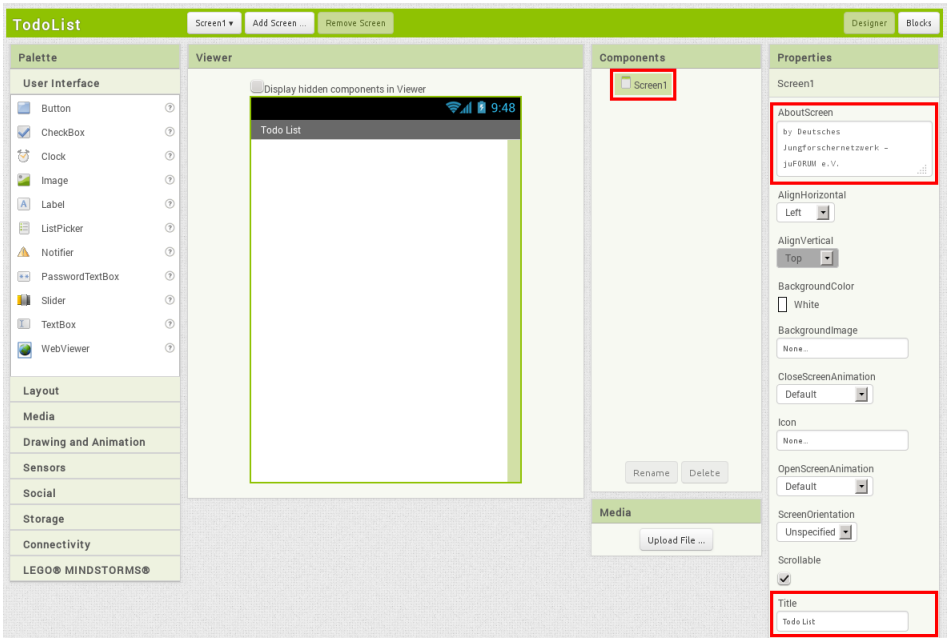
Abbildung 34: Todo-Liste – Die fertige App



a) Steuerelemente platzieren

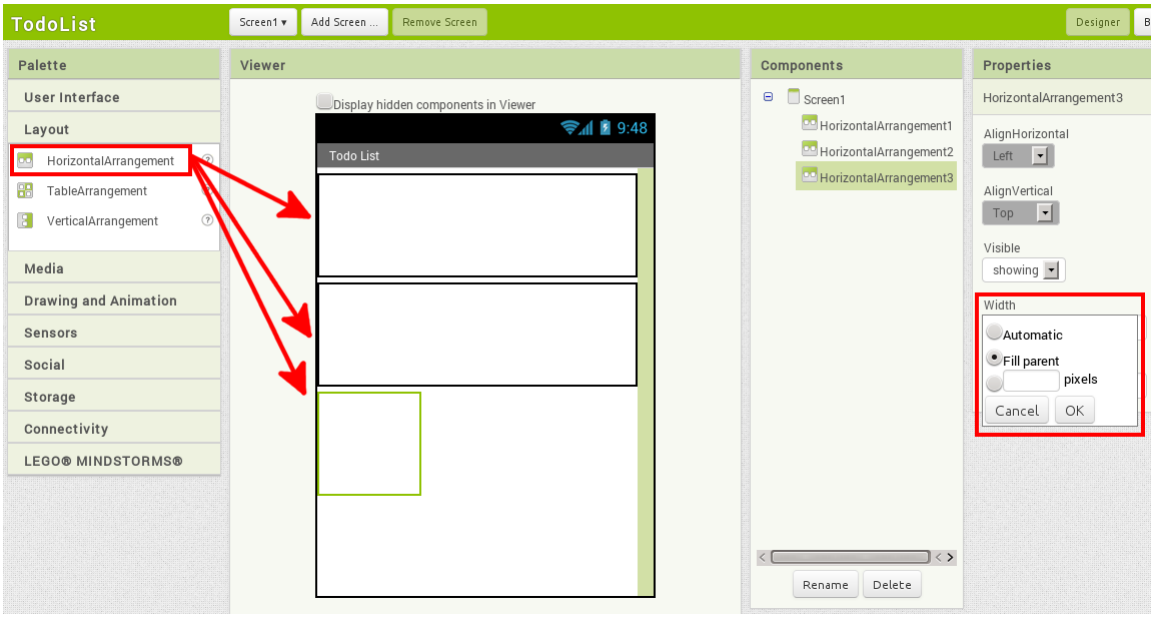
Zu Beginn wird ein neues Projekt erstellt, der Titel des Hauptdialogs „Screen 1“ und der Kommentar werden beispielsweise auf „Todo List“ festgelegt (Abbildung 35).

Abbildung 35: Hauptdialog einrichten



Um Steuerelemente nebeneinander platzieren zu können, werden drei HorizontalArrangements (Kategorie „Layout“) untereinander angeordnet. Ihre Breite („Width“ wird auf „Fill parent“) gesetzt (Abbildung 36).

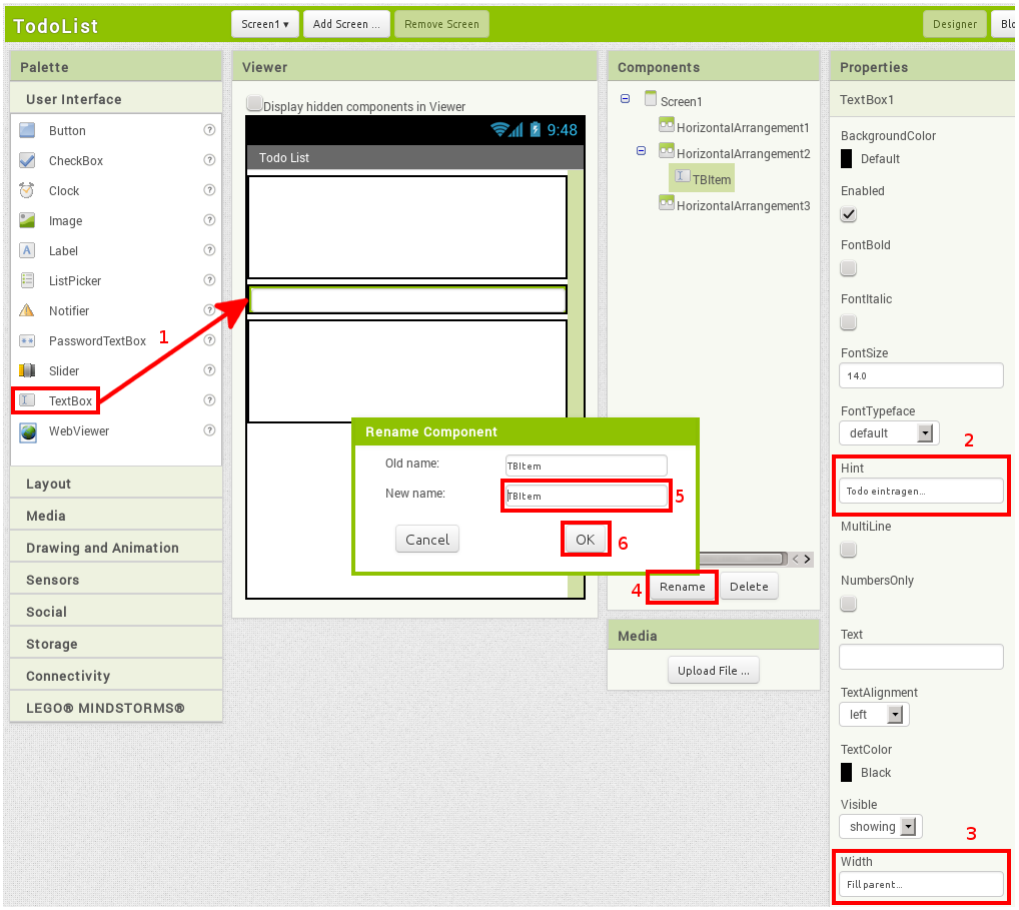
Abbildung 36: HorizontalArrangements platzieren



In das mittlere HorizontalArrangement wird eine TextBox (Kategorie „User Interface“) ein-

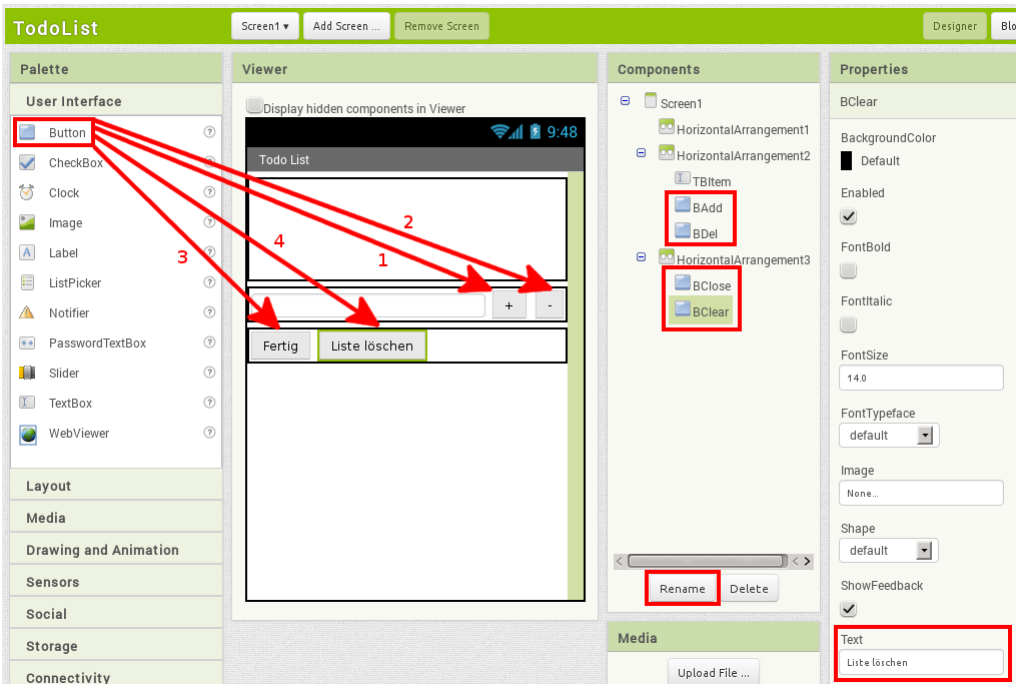
gefügt (Schritt 1, Abbildung 37). Ihr werden folgende Eigenschaften zugewiesen: Hint: „Todo eintragen...“ (Schritt 2), Width: „Fill parent...“ (Schritt 3), Name: „TBItem“ (Schritte 4-6). Eine TextBox ist ein einfaches Steuerelement zur Eingabe unformatierten Textes. Die Eigenschaft „Hint“ gibt einen Platzhaltertext an, der angezeigt wird, sollte die TextBox keinen Text enthalten.

Abbildung 37: TextBox platzieren



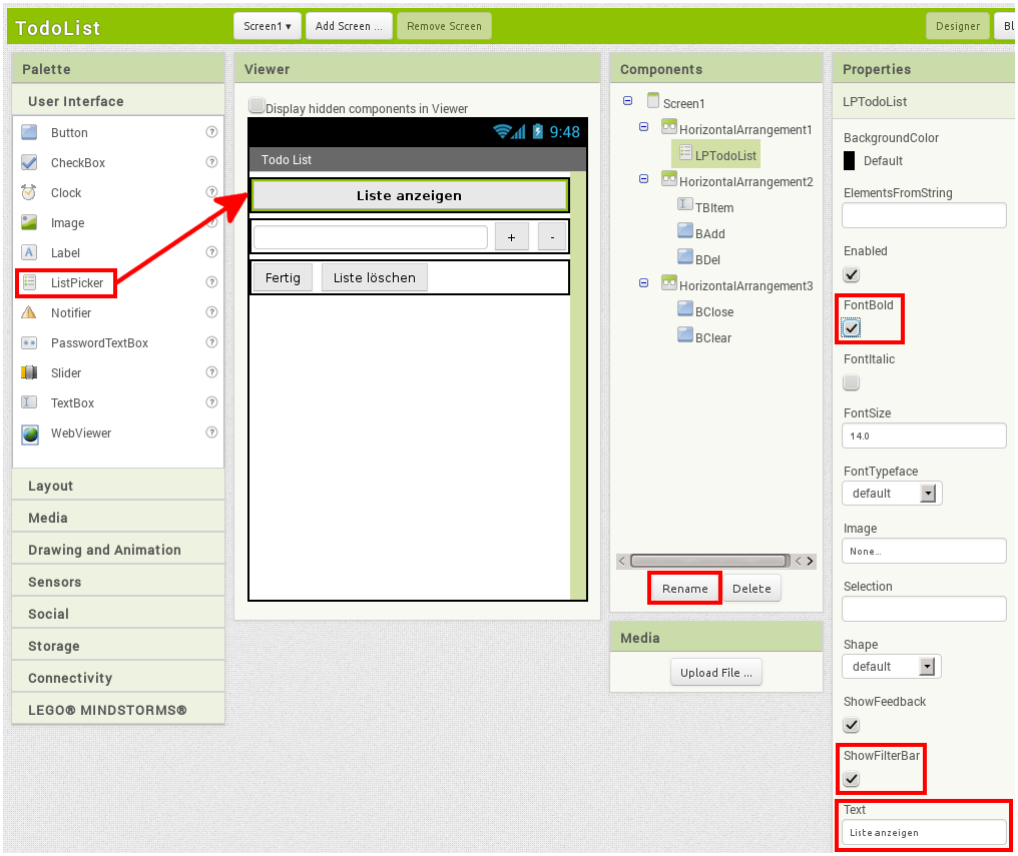
Neben der TextBox werden zwei Buttons mit Eigenschaften Name: „BAdd“, Text: „+“ und Name: „BDel“, Text: „-“ platziert. Diese dienen zum Hinzufügen bzw. Löschen eines Eintrags der Todo-Liste. In das untere HorizontalArrangement werden ebenfalls zwei Buttons mit Eigenschaften Name: „BClose“, Text: „Fertig“ und Name: „BClear“, Text: „Liste löschen“ eingefügt. Diese werden später die App beenden bzw. die Todo-Liste leeren (siehe Abbildung 38).

Abbildung 38: Buttons platzieren



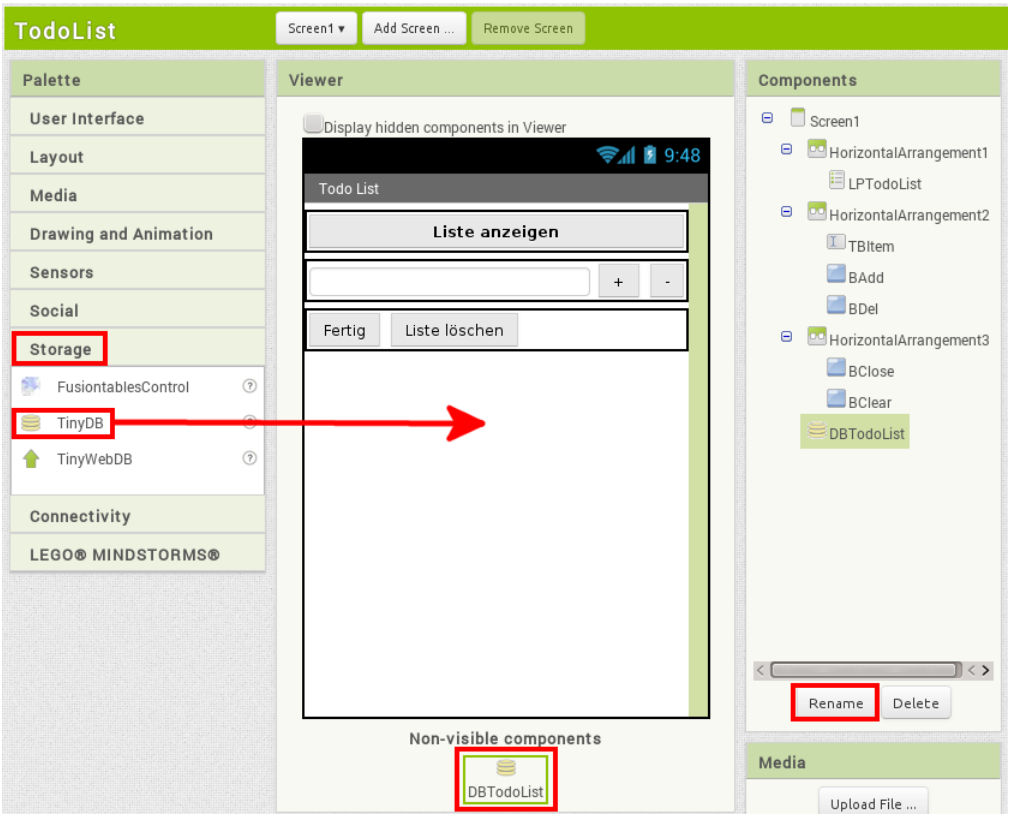
In das oberste HorizontalArrangement wird ein ListPicker eingefügt. Hierbei handelt es sich um eine Art Button, der beim Anklicken eine Liste öffnet, aus der ein Element ausgewählt werden kann. Folgende Eigenschaften werden gesetzt: Name: „LPTodoList“, FontBold: ja, ShowFilterBar: ja, Text: „Liste anzeigen“ (siehe Abbildung 39). Die Eigenschaft „ShowFilterBar“ bewirkt, dass die geöffnete Liste ein Suchfeld enthält, das es erlaubt, nach Listeneinträgen zu suchen.

Abbildung 39: ListPicker platzieren



Im letzten Schritt wird der App das virtuelle Steuerelement „TinyDB“ (Kategorie „Storage“) hinzugefügt und „DBTodoList“ genannt (Abbildung 40). TinyDB stellt Funktionen bereit, mittels welchen Daten (hier Listeneinträge) auf dem Smartphone gespeichert werden können (konkret im App-Cache). Die auf diese Weise gespeicherten Daten bleiben erhalten, wenn die App geschlossen wird. Sie werden erst beim Deinstallieren der App oder bei manueller Löschung entfernt.

Abbildung 40: TinyDB einfügen



b) Verhalten programmieren

Nun wird die Programmierung im Blocks Editor vorgenommenen. Um Klicks auf die vier Buttons behandeln zu können, wird für jeden Button ein *Eventhandler* hinzugefügt: Screen 1 → HorizontalArrangement[n] → [Buttonname] → „when [Buttonname].Click“ (siehe Abbildung 41).

Ein Klick auf „BClose“ soll die App beenden. Der hierzu nötige Block Built-in → Control → „close application“ wird im enstprechenden Eventhandler platziert (Abbildung 42).

Da es relativ aufwändig ist, alle Listeneinträge der Todo-Liste aus dem Speicher zu lesen, wird eine Kopie dieser Liste als Variable parat gehalten. Variablen erlauben es, temporär Daten an bestimmten Stellen im (RAM-)Speicher des Smartphones abzulegen bzw. zwischenzuspeichern, um an einer anderen Stelle im Programm Zugriff auf diese zuvor bereitgestellten Daten zu haben. Variablen können bestimmte Typen haben: Es gibt beispielsweise Variablen, die Text enthalten können, Variablen, die ausschließlich Zahlenwerte enthalten können und Listen, die in gewisser Weise selbst ein eigener Datentyp sind. Es wird nun eine Variable vom

Abbildung 41: Eventhandler für Buttons einfügen

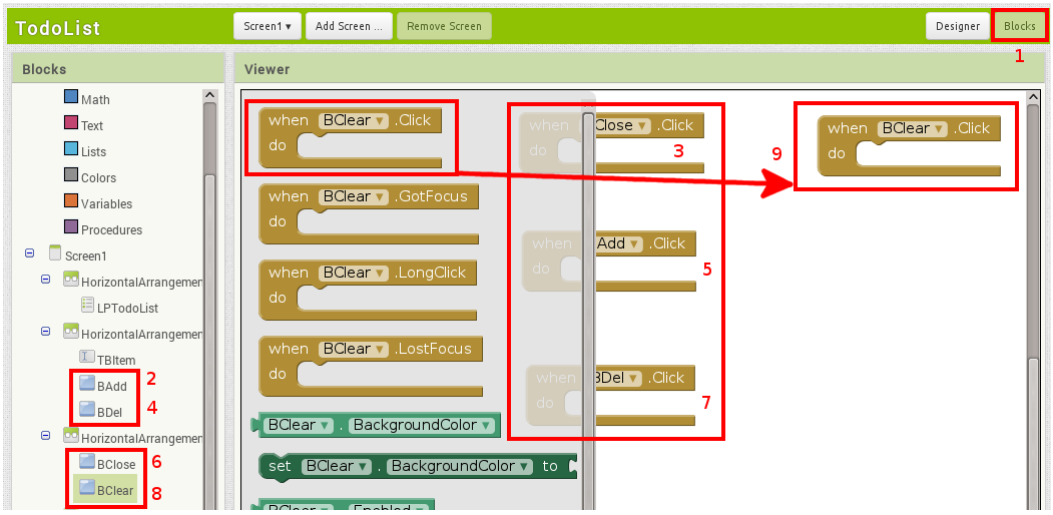
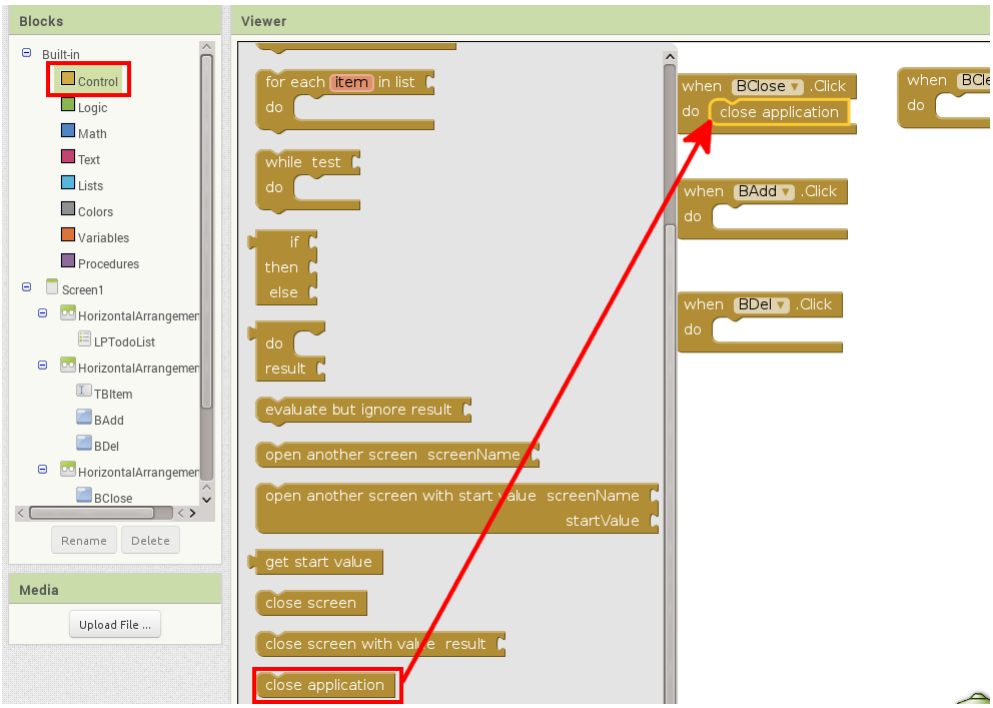


Abbildung 42: Block zum Beenden der App einfügen



Typ einer Liste angelegt: Der Block Built-in → Variables → „initialize global [name] to“ erzeugt eine Variable, die an jeder Stelle im Programm verwendet werden kann, d.h. sie ist *global*. Dieser Block wird im Arbeitsbereich abgelegt. Der Name der Variable wird auf „Elements“ gesetzt. In die Einbuchtung dieses Blocks muss der Startwert der Variable eingesetzt werden. Dieser impliziert den Typ der Variable. Zu Beginn soll „Elements“ eine leere Liste

sein. Aus diesem Grund wird der Block Built-in → Lists → „create empty list“ angefügt (Abbildung 43).

Abbildung 43: Liste als globale Variable erstellen

Im Folgenden wird eine *Funktion* definiert, die die gerade eben erzeugte Liste mit Listeneinträgen aus dem Speicher (aus „DBTodoList“) füllt. Eine Funktion verhält sich ähnlich wie die mittlerweile schon häufig genutzten Events. Sie wird im Gegensatz zu Events jedoch vom Programmierer direkt selbst aufgerufen. Auf diese Weise kann u.a. vermieden werden, dass häufig genutzte Programmteile an jeder Stelle, wo sie gebraucht werden, als Duplikate eingefügt werden müssen. Dies trägt erheblich zur Übersichtlichkeit bei. Änderungen in diesen häufig genutzten Abschnitten müssen außerdem nur einmal an zentraler Stelle vorgenommen werden. Funktionen können *Parameter* übernehmen. Dies sind *lokale* Variablen, die nur innerhalb der Funktion selbst sichtbar sind (beispielsweise sind die Werte, die in Einbuchtungen verschiedener schon genutzter Blöcke „eingegeben“ wurden, Parameter).

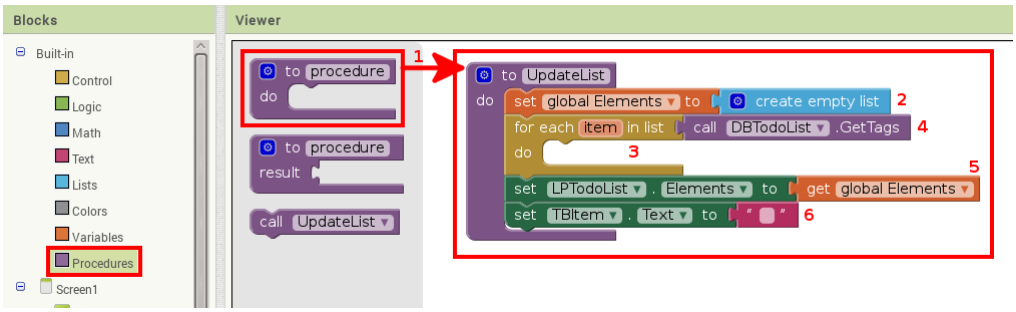
Um eine Funktion zu definieren, wird der Block Built-in → Procedures → „to procedure do“ eingefügt. Der Name der Funktion wird in „UpdateList“ geändert (Schritt 1 Abbildung 44). In der Funktion wird die Liste „Elements“ zuerst gelöscht: Built-in → Lists → „create empty list“ wird als Parameter an Built-in → Variables → „set global Elements to“ übergeben. Letzterer Block wird oben in die Funktion eingefügt (Schritt 2).

Als dritter Schritt wird der Block Built-in → Control → „for each item in list“ unten in die Funktion eingefügt. Als Parameter wird ihm Screen1 → DBTodoList → „call DBTodoList.GetTags“ übergeben (Schritt 4). Dieser Block liefert eine Liste von sog. Tags unter denen Werte in „DBTodoList“ gespeichert sind (dazu später mehr). Es handelt sich bei dem „for each“-Block um eine *for-each-Schleife*. Die Schleife spannt ebenfalls eine Klammer auf (Inhalt wird später folgen). Für jedes Element („item“), das sich in der Liste, die „GetTags“ zurückgibt, befindet, wird der *Schleifenrumpf* (die Klammer) einmal von oben nach unten durchlaufen. Auf diese Weise kann jedes Element einer Liste verarbeitet werden.

Nachdem die Schleife beendet ist, sollen die Elemente, die der ListPicker „LPTodoList“ enthält, auf die Listenvariable gesetzt werden. Hierzu wird der Befehl Screen1 → HorizontalArrangement1 → LPTodoList → „set LPTodoList.Elements to“ mit Parameter Built-in → Variables → „get global Elements“ direkt unter der Schleife in die Funktion eingefügt (Schritt 5).

Ferner soll der Inhalt der TextBox „TBItem“ gelöscht werden. Dazu wird der Block Screen1 → HorizontalArrangement2 → TBItem → „set TBItem.Text to“ mit Parameter Built-in → Text → „ “ unten in die Funktion eingefügt (Schritt 6).

Abbildung 44: Funktion „UpdateList“ erstellen



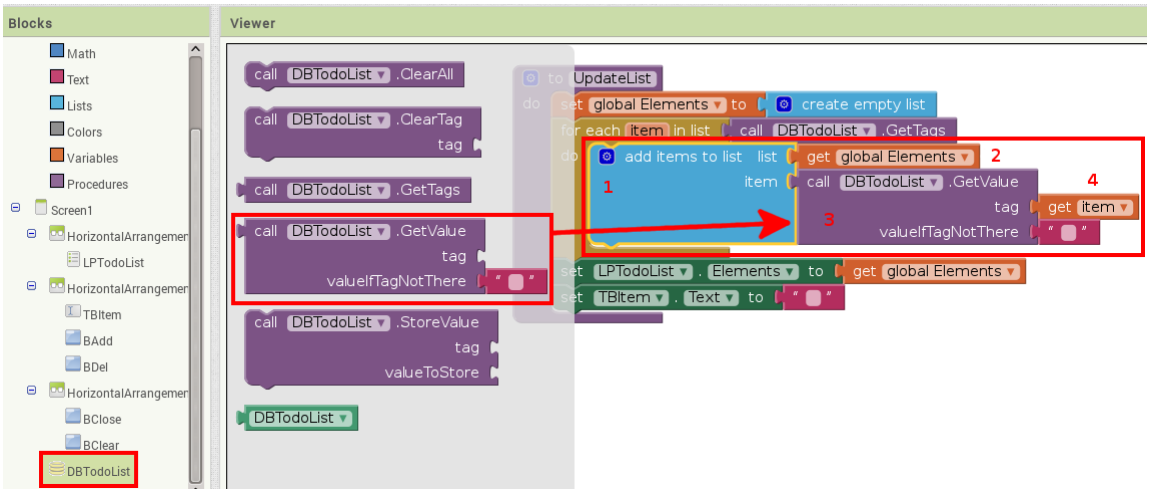
An dieser Stelle ist zu erklären, wie TinyDB Daten intern abspeichert, um verstehen zu können, wie mit „DBTodoList“ gearbeitet wird. TinyDB verwaltet intern eine Liste von Schlüssel-Wert-Paaren. Jeder Eintrag dieser Liste besteht aus einem Schlüssel (tag), dem ein Wert (value) zugeordnet ist. Hier werden die Tags einfach hochgezählt, es ist allerdings möglich, beliebige Bezeichnungen zu wählen. In diesem Fall könnte der Inhalt von „DBTodoList“ wie folgt aussehen:

Schlüssel (tag)	**Wert (value)**
1	Todo 1
2	Todo 2
3	Todo 3

In der Schleife werden alle Tags durchlaufen. Jeder zugehörige Wert soll in der Liste „Elements“ abgelegt werden. Um dieser Liste Einträge hinzuzufügen, wird der Block Built-in → Lists → „add items to list“ im Schleifenrumpf platziert (Schritt 1 Abbildung 45). Der erste Parameter („list“) bezeichnet die Liste, der ein Eintrag hinzugefügt werden soll. Hier ist dies Built-in → Variables → „get global Elements“ (Schritt 2). Der Parameter „item“ bezeichnet den Inhalt des hinzuzufügenden Eintrags. Dies ist der zum aktuellen Tag gehörende Wert.

Mittels Screen1 → DBTodoList → „call DBTodoList.GetValue“ wird dieser Wert geholt (Schritt 3). Diesem Block muss allerdings noch der aktuelle Tag übergeben werden. Dies geschieht mit Built-in → Variables → „get item“ (Schritt 4).

Abbildung 45: Funktion „UpdateList“ fertigstellen



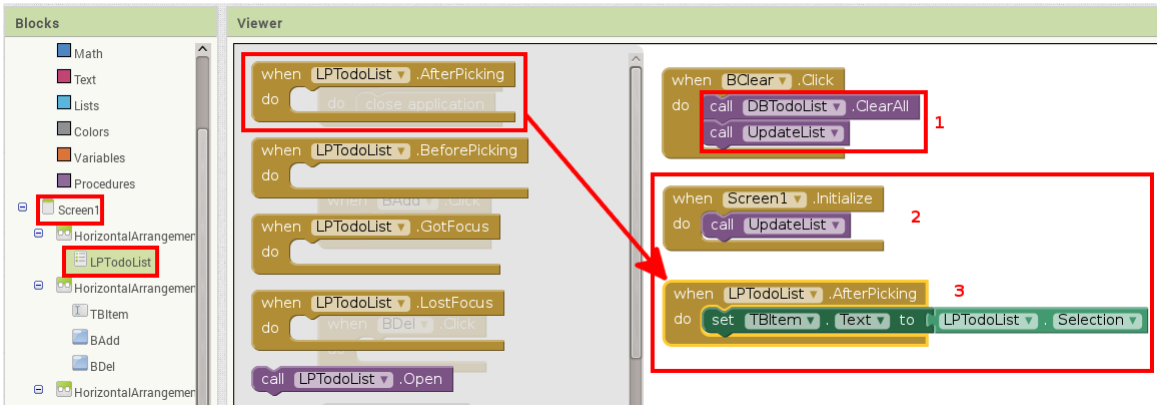
Wenn „BClear“ gedrückt wurde, soll die gesamte Todo-Liste gelöscht werden. Dies wird durch den Aufruf von Screen1 → DBTodoList → „call DBTodoList.ClearAll“ (aus Speicher löschen) gefolgt von Built-in → Procedures → „call UpdateList“ (Anzeige aktualisieren) bewerkstelligt (Schritt 1 Abbildung 46).

Die Anzeige soll ebenfalls aktualisiert werden, nachdem die App gestartet wurde. Dazu ist das Event Screen1 → „when Screen1.Initialize“ erforderlich. Hierin wird Built-in → Procedures → „call UpdateList“ ebenso aufgerufen (Schritt 2).

Es wird das Event Screen1 → LPTodoList → HorizontalArrangement1 → „when LPTodoList.AfterPicking“ eingefügt. Dieses Event wird ausgelöst, wenn im ListPicker „LPTodoList“ ein Eintrag ausgewählt wurde. Ist dies der Fall, soll das Textfeld „TBItem“ auf den ausgewählten Eintrag gesetzt werden. Dies geschieht mittels Screen1 → HorizontalArrangement1 → LPTodoList → „LPTodoList.Selection“ als Parameter für Screen1 → HorizontalArrangement2 → TBItem → „set TBItem.Text to“ (Schritt 3).

Der wichtigste Schritt besteht im Hinzufügen von Elementen zur Todo-Liste. Dies soll beim Klicken von „BAdd“ geschehen, sofern „TBItem“ Text enthält. Wie an dieser Beschreibung schon ersichtlich ist, wird eine *Bedingung* in Form einer Fallunterscheidung benötigt: Falls (engl. if) „TBItem.Text“ nicht leer ist, soll ein Eintrag angelegt werden. Der if-Block (Built-in → Control → „if then“) erfüllt diese Anforderung (Schritt 1 Abbildung 47). Alles, was die Klammer beinhaltet, die von diesem Block aufgespannt wird, wird genau dann

Abbildung 46: weitere Events behandeln



ausgeführt, wenn die Bedingung (das, das an der Einbuchtung des if-Blocks hängt), wahr ist. Da hier auf Ungleichheit überprüft werden soll, ist der entsprechende logische Operator nötig: Built-in → Logic → „≠“ (vgl. Addition im 2. Tutorial). Einer der Operanden ist der zu überprüfende Text (Screen1 → HorizontalArrangement2 → TBItem → „TBItem.Text“), der zweite Operand ist der leere Text (Built-in → Text → „ “).

Ist die Bedingung wahr, wird ein Schlüssel-Wert-Paar in „DBTodoList“ abgespeichert. Hierzu wird Screen1 → DBTodoList → „call DBTodoList.StoreValue“ benötigt (Schritt 2). Der Tag setzt sich aus der aktuellen Anzahl an Einträgen plus 1 zusammen, um fortlaufend nach oben zu zählen. Entsprechend wird für diesen Parameter eine Addition (Built-in → Math → „+“) benötigt (Schritt 3). Der erste Operand ist die Länge der Liste „Elements“. Diese wird mittels Built-in → Lists → „length of list“ bestimmt. Dieser Befehl erwartet die Liste, deren Länge bestimmt werden soll, als Parameter. Es wird wieder Built-in → Variables → „get global Elements“ verwendet. Der zweite Operand ist die Zahl 1: Built-in → Math → „0“. Die 0 muss zur 1 abgeändert werden (Schritt 4). Der zu speichernde Wert ist der aktuell in „TBItem“ eingegebene Text. Dieser wird mittels Screen1 → HorizontalArrangement2 → TBItem → „TBItem.Text“ gelesen und an „valueToStore“ als Parameter angebunden (Schritt 5).

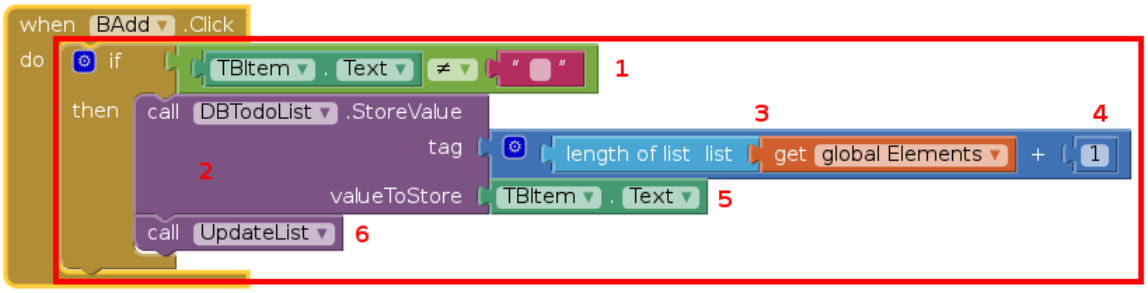
Es sei an dieser Stelle angemerkt, dass sich ifelse-Blöcke erzeugen lassen, indem auf das blaue Zahnrad in der linken oberen Ecke des if-Blocks geklickt wird und ein else-Baustein im sich öffnenden Bereich in den if-Baustein eingefügt wird.

Obwohl das Überprüfen der Gültigkeit von Benutzereingaben oft einen großen Teil des eigentlichen Programms ausmacht, gehört dies zum guten Programmierstil. Es gewährleistet, dass eine App durch fehlerhafte oder manipulative Benutzereingaben nicht abstürzt oder gar

Schaden auf dem Smartphone anrichtet und ist darum essentiell!

Schließlich wird die Anzeige aktualisiert, indem Built-in → Procedures → „call UpdateList“ in den if-Block unten eingefügt wird (Schritt 6). Damit ist die App vorerst fertiggestellt.

Abbildung 47: Einfügen Speichern von Listeneinträgen ermöglichen



c) Aufgaben

1. Wie vielleicht bemerkt wurde, wurde das Event „when BDel.Click“ nicht implementiert. Diese Aufgabe soll darin bestehen, dieses Event zu vervollständigen und damit das Löschen von Einträgen aus der Todo-Liste zu ermöglichen. Achtung: Dies ist nicht damit getan, den entsprechenden Tag aus „DBTodoList“ zu löschen, da dies dazu führen wurde, dass die Tags nicht mehr durchgehend nummeriert sind. Wenn diese Aufgabe erfolgreich bearbeitet wurde, ist gezeigt, dass wichtige Konzepte der Informatik verstanden wurden.
2. Die App soll dahingehend erweitert werden, dass mehrere verschiedene Listen verwaltet werden können. Dazu könnten beispielsweise weitere Screens zum Anlegen und Löschen weiterer Listen verwendet werden.

8 Ausblick

Die hier vorgestellten Apps nutzen die Möglichkeiten des MIT App Inventors lange nicht aus. Der App Inventor kann beinahe die gesamte Hardware eines Smartphones ansteuern. Das Ermitteln des aktuellen Standorts über den im Smartphone integrierten GPS-Empfänger, das Versenden und Auswerten von SMS und selbst das Steuern von LEGO® MINDSTORMS® ist ohne große Einarbeitung möglich. Tutorials hierzu finden sich im Internet unter http://appinventor.mit.edu/explore/tutorials.html. Ebenso ist online eine Referenz vorhanden, die die Funktionsweise aller verfügbaren Blöcke und Steuerelemente erklärt.

9 Lizenzbestimmungen

[4] http://creativecommons.org/licenses/by-nc-nd/4.0/deed.de

Abbildungsverzeichnis