TfK Technik für Kinder e.V.
Begeistern durch Machen

UNIVERSITÄT PASSAU
Fakultät für Informatik und Mathematik

Modul:

App-Entwicklung auf Mobile Devices

ANDROID

Version: 2.0

Dieses Modul wurde an der Lehr- und Forschungseinheit Mathematik und Informatik (LMI) der Universität Passau in Kooperation mit dem Verein „TfK - Technik für Kinder e.V.“ im Rahmen des Projekts Technikgruppen an Schulen - Kinder für Technik begeistern erarbeitet.
Ein Großteil der Aufgaben wurde von der Didaktik der Informatik unter Leitung von Frau Ute Heuer an der Universität Passau entwickelt.
Mehr zu diesem Projekt erfahren Sie auf der Internetseite

http://www.fim.uni-passau.de/fim/fakultaet/lehrstuehle-professuren-und-fachgebiete-der-fim/didaktik-der-informatik/projekte/technikgruppen.html .

Autoren:

Ute Heuer

Didaktik der Informatik
ute.heuer@uni-passau.de

Wolfgang Pfeffer

Didaktik der Informatik / Mathematik
wolfgang.pfeffer@uni-passau.de

Modulübersicht

Im Rahmen des Kooperationsprojekts zwischen der Universität Passau und dem Verein Technik für Kinder e.V. sind folgende Module entstanden:

Modul Roboter-Programmierung

Dieses Modul bietet einen sehr kurzen Einstieg in die Programmierung eines Lego EV3 Roboters mit der Lego Mindstorms Education Software. Neben Hardware- und Software Anforderungen wird die Entwicklungsumgebung vorgestellt und anschließend Einstiegsaufgaben zu Motoren, Sensoren sowie vermischte Aufgaben bereitgestellt.

Modul App-Entwicklung auf Mobile Devices

Dieses Modul thematisiert umfassend die App-Entwicklung auf Mobile Devices mit dem AppInventor, einer graphischen Programmierumgebung. Ein einfacher und handlungsorientierter Zugang zu dieser Thematik steht dabei im Vordergrund. Neben Hardware- und Softwareanforderungen wird eine ausführliche Installationsanleitung angeboten. Die Einführungsaufgabe 'Schritt für Schritt zur ersten eigenen App' ermöglicht eine Einführung in den Umgang mit dem AppInventor. Gleichzeitig werden fast alle wichtigen Elemente behandelt. Anschließend umfasst das Modul 10 Aufgaben mit unterschiedlichem Schwierigkeitsgrad sowieso ausführlichen Lösungen. Exkurse zu Themen wie GPS und Dijkstra-Algorithmus runden die Handreichung ab.

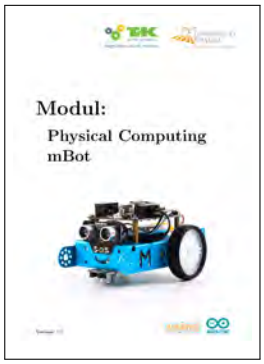
Modul App-Entwicklung mit Java Android

Dieses Modul baut thematisch auf dem Modul 'App-Entwicklung auf Mobile Devices' auf, anstelle der graphischen Programmierung werden die Applikationen nun mit Java Android entwickelt. Da die Einstiegshürde in Java Android vergleichsweise hoch ist, sind grundlegende Java-Kenntnisse unumgänglich. Es wird ein sehr ausführlicher sowie kleinschrittiger Einstieg in die Programmierung mit Java Android geboten. Anschließend stehen verschiedene Projekte mit umfangreichen Lösungen zur Auswahl (z.B. Vokabel-App, Quiz-App, 2048-App, Datenspeicher oder LegoPilot-App). Dabei werden wichtige Konzepte der Sekundarstufe II behandelt.

Modul Physical Computing

Dieses Modul kombiniert Elemente aus der Physik und der Informatik und basiert auf dem Einplatinencomputer Raspberry Pi. Zunächst geht es darum, (einfache) Schaltungen zu erstellen und mit konstanter Spannungsquelle zu arbeiten. Hierbei wird mit verschiedenen Schaltungselementen experimentiert (z.B. LED, Widerstand, LDR, Poti, Servomotor). Anschließend soll an manchen Pins gezielt Spannung angelegt oder nicht angelegt werden. Dazu werden die GPIO-Pins mit der Programmiersprache Python konfiguriert und angesteuert. Die Handreichung beinhaltet ein eigenes Kapitel zu Python, in welchem man sich anhand kleiner Aufgaben mit der Syntax der Programmiersprache vertraut machen kann.

Modul Physical Computing mBot



Dieses Modul kann als Schnittstellenmodul zwischen Roboter-Programmierung und Physical Computing gesehen werden. Bei mBot handelt es sich um ein handliches Roboterfahrzeug, das mit verschiedenen Sensoren und Motoren ausgestattet ist, die mit dem Mikrocontroller Arduino verbunden werden können. Im Bezug auf die Programmierung bietet das Modul zwei Zugangswege: Ähnlich zur App-Entwicklung mit dem AppInventor kann der mBot mithilfe der graphischen Programmierumgebung Scratch angesteuert werden. Ein zweiter Teil des Moduls bietet die Programmierung des mBot mit Hilfe von Arduino-C.

Quelle des Bildes: https://www.colourbox.de/vektor/buch-vektor-5814145

Inhaltsverzeichnis

Kapitel 1

Über das Modul

App-Entwicklung auf Mobile Devices - Wie der Titel schon vorwegnimmt, beschäftigen wir uns in diesem Modul damit, Schülerinnen und Schülern einen einfachen und handlungsorientierten Zugang zur Entwicklung von kleinen Applikationen für Smartphones bzw. Tablets zu ermöglichen.

Hierfür eignet sich der App Inventor von MIT (Massachusetts Institute of Technology) hervorragend:

Abbildung 1.1: MIT App Inventor - siehe auch http://appinventor.mit.edu/.

Diese Anwendung wurde im Zeitraum von 2009-2011 zunächst von Google entwickelt, um Nutzern den Einstieg in die (Android-)Programmierung zu erleichtern.
MIT übernahm 2012 das Projekt und entwickelt es stetig weiter - siehe hierzu http://mitmobilelearning.org. Auf der oben genannten Webseite des App Inventors gibt es viele Anleitungen und Tutorials, welche sowohl für Schüler als auch Lehrer interessant sein können.

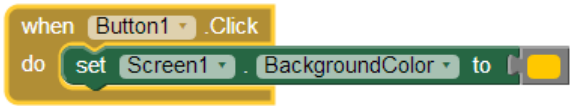


Abbildung 1.2: Beispiel für die Entwicklungsumgebung des App Inventor.

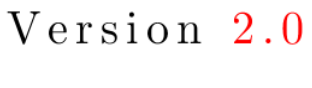
In Abbildung 1.2 ist ein Ausschnitt der Entwicklungsumgebung zu sehen. Unabhängig von einer konkreten Programmiersprache und -syntax lassen sich durch Zusammenfügen von vorgefertigten Bausteinen Android-Apps erstellen. Anhand dieser Bausteine (nicht alles lässt sich zusammenfügen) können die Schüler wichtige Vorerfahrungen im Bezug auf Programmiersprachen sammeln.

Der App Inventor ist im weitesten Sinne eine Webanwendung und es sind nur wenige Schritte erforderlich, um mit dem Erstellen der ersten eigenen Applikation beginnen zu können. In Kapitel 3 zeigen wir, wie Sie ihr Smartphone oder Tablet (im Folgenden: Mobile Device) mit dem App Inventor verbinden können. In Kapitel 4 gehen wir kurz genauer auf die Funktionsweise der Anwendung ein, bieten Ihnen anschließend einen schrittweisen Einstieg in die Entwicklungsumgebung und machen Sie mit den wichtigsten Werkzeugen vertraut.

In Kapitel 5 folgen einige Aufgaben mit verschiedenen Schwierigkeitsgraden, die von den Schülern selbständig bearbeitet werden sollen. Einige Aufgaben in diesem Dokument enthalten noch zusätzliche Hinweise für Sie als Betreuer. Für die Schüler gibt es diese auf Übungsblättern in separaten PDF-Dateien. Bei manchen Aufgaben wird zusätzliches Material benötigt, wie etwa Bild- oder Musik-Dateien. Diese finden Sie im entsprechenden Aufgabenordner.
Ausführliche Lösungsvorschläge folgen dann in Kapitel 6.
Im Anhang in Kapitel 7 haben wir weitere wichtige Informationen, Exkurse und Installationsanleitungen für Sie bereitgestellt.
Zunächst gehen wir in Kapitel 2 auf die Hardware und Software ein, die für die Verwendung des App Inventor benötigt wird.

Um Ihnen den Umgang mit diesem Modul zu erleichtern, folgt auf der nächsten Seite ein grafischer Leitfaden, in dem wir die Inhalte des Moduls noch einmal visualisiert haben.
In dieser Grafik sind insbesondere Abhängigkeiten unter den Aufgaben dargestellt, d.h. ob es sich empfiehlt, vor der Bearbeitung einer bestimmten Aufgabe bereits andere Aufgaben gelöst zu haben.
Weiter zeigen wir Ihnen, bei welchen Aufgaben sich Exkurse anbieten würden.
Die Reihenfolge der Aufgaben in diesem Modul ist nicht zwingend. Aufgaben, die im Leitfaden untereinander abgebildet sind können unabhängig voneinander bearbeitet werden.

Die aktuelle Version 1.0 ist sozusagen noch ein Prototyp. Wir möchten das Modul im Laufe der Zeit durch Evaluation weiterentwickeln - dazu ist uns Ihre Meinung als Betreuer der Technikgruppen sehr wichtig. Wir würden uns freuen, wenn Sie Anregungen oder etwaige Verbesserungsvorschläge an uns weiterleiten würden. Vorschläge für neue Aufgaben und Erfahrungen aus dem Einsatz des Moduls in den Technikgruppen sind für uns ebenfalls sehr hilfreich!



Seit Anfang 2014 gibt es eine neue Version des AppInventors - den AppInventor 2.
AppInventor 2 ermöglicht nun das Definieren von lokalen Variablen. Zudem wird der Blocks-Editor nicht mehr als `JNLP`-Datei auf den Computer heruntergeladen, sondern ist auch im Internetbrowser integriert. In Version 2.0 haben wir die Bilder, Aufgaben und Installationsanleitung an den AppInventor 2 angepasst.

Bitte senden Sie Ihre Erfahrungen und Anregungen an `wolfgang.pfeffer@uni-passau.de`.

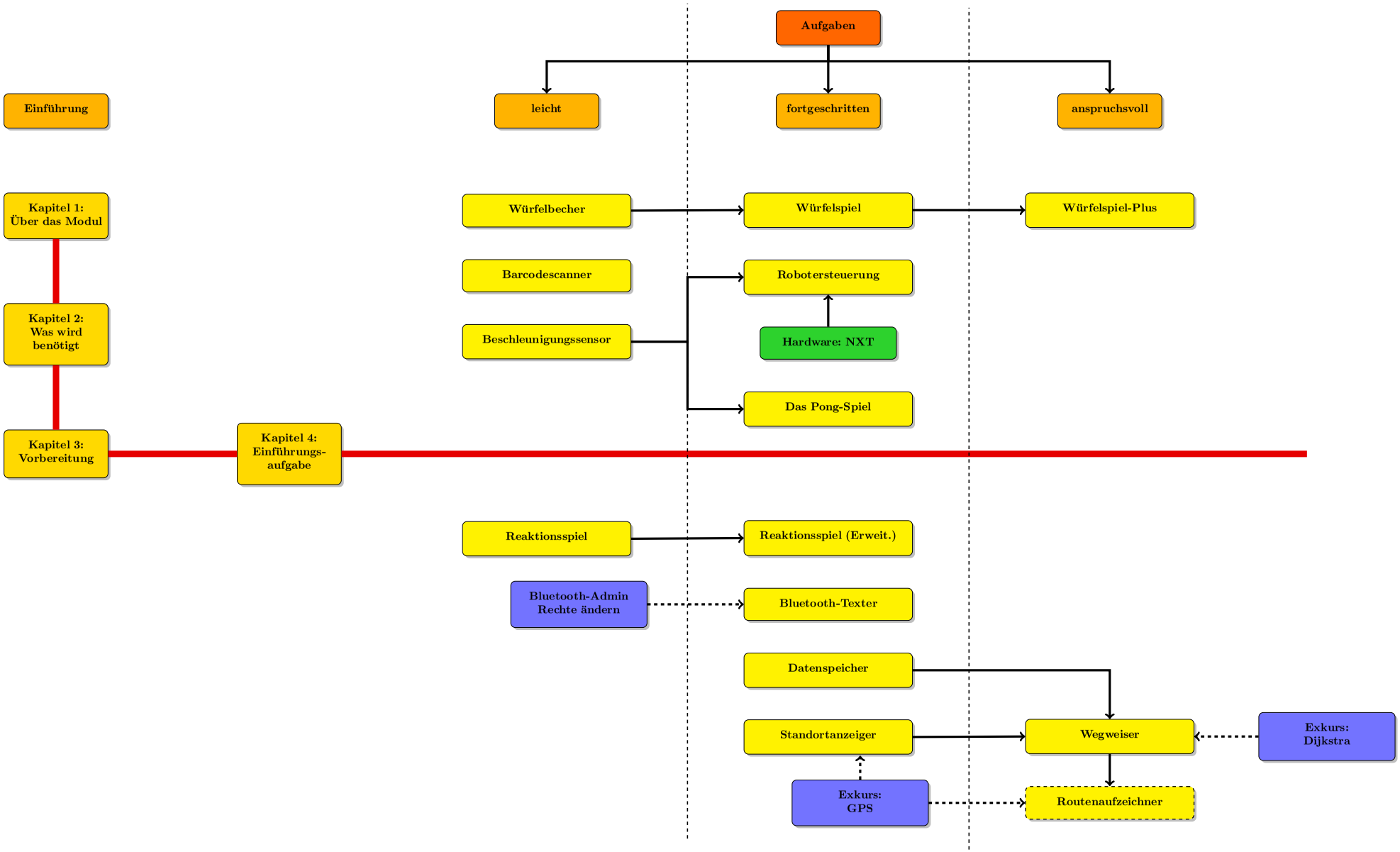


Abbildung 1.3: Modul-Leitfaden mit Abhängigkeiten zwischen Aufgaben, besonderen Hardwareanforderungen und möglichem Einsatz von Exkursen

Kapitel 2

Was wird benötigt?

Überblick:

Zunächst möchten wir auf die technischen Voraussetzungen eingehen. Wir unterscheiden dabei erforderliche Hardware und erforderliche Software:

2.1 Hardware

Jeder Arbeitsplatz benötigt ein Smartphone bzw. Tablet mit einem Android-Betriebssystem. Mit iPhone, iPad und Mobile Devices mit anderen Betriebssystemen (Windows, o.ä.) funktioniert der AppInventor nicht. Neben dem Mobile Device wird noch ein USB-Kabel benötigt, damit man es mit einem Laptop bzw. Desktop-PC verbinden kann:

Abbildung 2.1: Mobile Device mit Android-Betriebssystem und USB-Verbindungskabel

Um den AppInventor verwenden zu können, benötigen wir noch einen Laptop oder Desktop-PC mit Internetzugang.

2.2 Software

Für den `Blocks Editor` ist auf dem Laptop bzw. Desktop-PC die Software Java erforderlich. Ist Java auf ihrem Laptop bzw. Desktop-PC nicht installiert, können Sie die Software unter folgendem Link herunterladen:

`http://www.java.com/de/download/`

2.3 Optional: Lego Mindstorms NXT

In einer Aufgabe aus Abschnitt 5.7 aus Kapitel 5 werden wir einen Lego Mindstorms NXT Roboter mit dem Mobile Device über Bluetooth steuern.
Für alle anderen Aufgaben sind die Lego Mindstorms NXT Roboter nicht erforderlich.

Kapitel 3

Vorbereitung - Mobile Device mit App Inventor verbinden

Überblick:

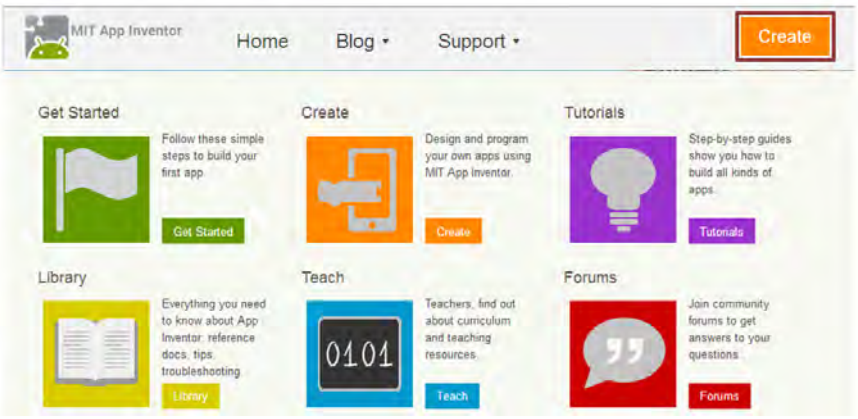
Dieses Kapitel befasst sich damit, das Mobile Device mit dem AppInventor zu verbinden. Hierfür sind nur wenige Schritte erforderlich.
Wie bereits erwähnt, ist der AppInventor eine Art Webanwendung. Um diese nutzen zu können, ist eine Anmeldung mit einem Google-Konto erforderlich. Nach der Anmeldung kann das Problem auftreten, dass das Mobile Device von der AppInventor-Anwendung nicht erkannt wird. Dieses Problem kann man durch Installation eines geeigneten Treibers beheben. Mehr dazu in Abschnitt 3.6.
Zunächst benötigen wir jedoch ein Google-Konto, um uns beim AppInventor anmelden zu können:

3.1 Erstellen eines Google-Kontos und Anmeldung

Wir wollen nun zum ersten Mal den App Inventor benutzen und rufen dazu die Internet-Seite

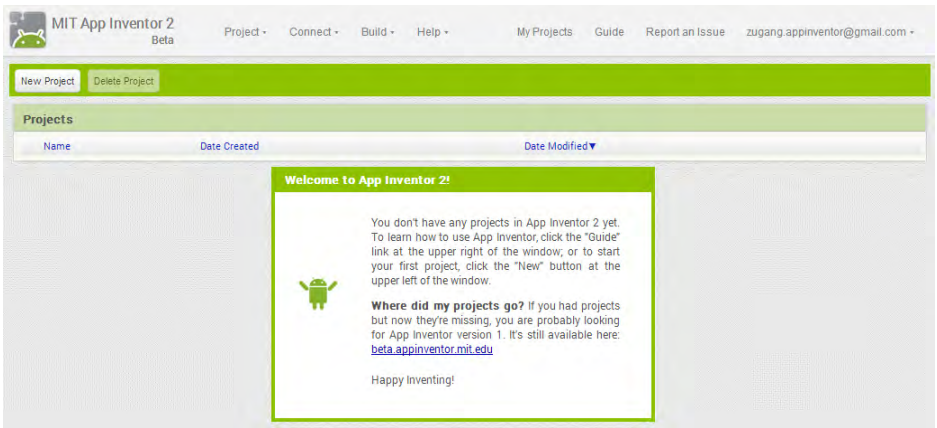
`http://www.appinventor.mit.edu/`

auf. Es erscheint folgende Seite:



Nach Auswahl der orangen Schaltfläche `Create` werden wir aufgefordert, uns mit einem GOOGLE-Konto anzumelden. Besitzt man bereits eine GOOGLE-Mailadresse, kann man diese für den AppInventor nutzen; ansonsten ist eine Registrierung über die Schaltfläche „Konto erstellen“ möglich.
Zu Demonstrationszwecken verwenden wir im Folgenden die Mail-Adresse zugang.appinventor@gmail.com.
Anschließend müssen wir für die Anwendung noch eine Zugriffs-Berechtigung erlauben:

Nach der Bestätigung landen wir zum ersten Mal auf der Seite des AppInventors:



Um in Zukunft einfacher auf den App Inventor zugreifen zu können, empfiehlt es sich, für diese Seite ein Lesezeichen anzulegen und dieses in der Lesezeichenleiste abzulegen.

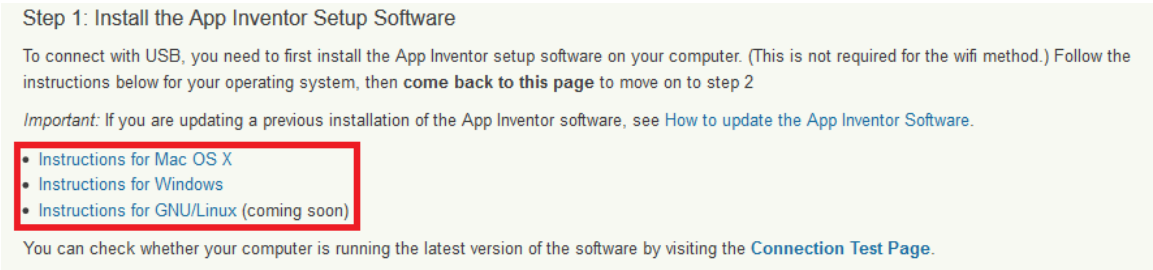
Der AppInventor informiert uns darüber, dass wir aktuell noch kein Projekt angelegt haben. Um testen zu können, ob die Anwendung unser Mobile Device ohne Installation eines zusätzlichen Treibers erkennt, legen wir nun ein Testprojekt an.
Dazu klicken wir links oben auf `New Project` und wählen anschließend als Projektname *Testprojekt*. Es öffnet sich die Designer-Ansicht des AppInventors. Bevor wir unser Mobile Device mit dem AppInventor verbinden können, sind noch weitere Konfigurationsschritte notwendig, welche im Folgenden beschrieben werden.

3.2 Installieren der AppInventor-Software

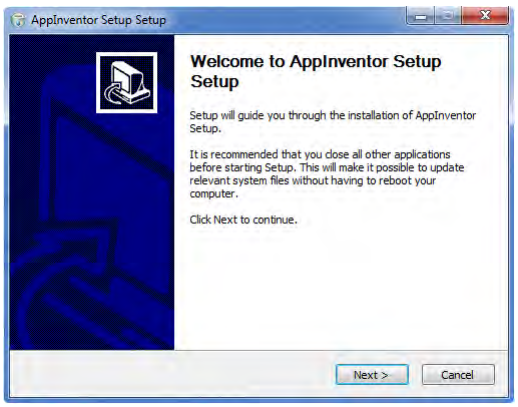
Auf der Internetseite:

`http://appinventor.mit.edu/explore/ai2/setup-device-usb.html`

können wir unter dem Abschnitt `Step 1` das passende Betriebssystem auswählen und auf der Folgeseite unter dem Link `Download the installer` die zugehörige AppInventor Software herunterladen:

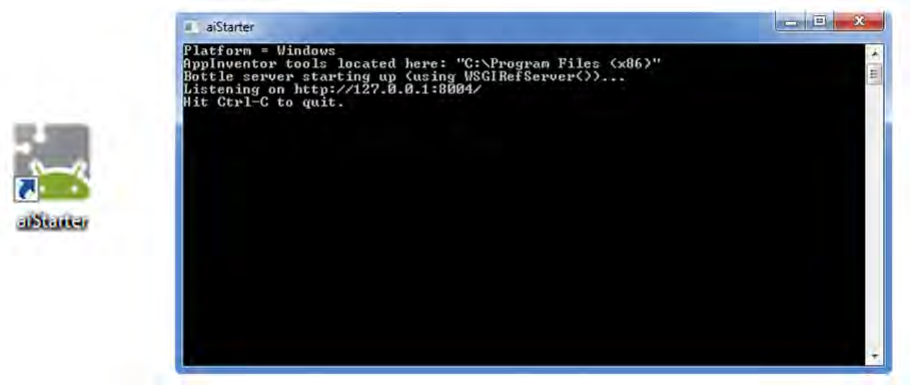


Anschließend kann diese unter einem beliebigen Dateipfad installiert werden. Bei der Installation müssen wir darauf achten, über Administratorrechte zu verfügen, da eine Installation ohne entsprechende Rechte von den Entwicklern derzeit noch nicht unterstützt wird und später zu Kompatibilitätsproblemen führen kann.



Nach erfolgreichem Abschluss der Installation erscheint auf dem Desktop automatisch das Symbol `aiStarter`.

Wir starten nun die Software mit einem Doppelklick auf das `aiStarter`-Symbol, worauf sich unter Windows ein Konsolenfenster öffnet. Die Software ist damit erfolgreich installiert und funktionsbereit. Bevor wir also mit dem Programmieren mit dem AppInventor beginnen können, müssen wir stets sicherstellen, dass auf unserem System die AppInventor Software ausgeführt wird:



3.3 Konfiguration des Mobile Device

Auf Rechnerseite haben wir die zur Programmierung mit dem AppInventor nötigen Installations- und Konfigurationsschritte in den vorangehenden Abschnitten vorgenommen. Nun wenden wir uns der Konfiguration des Mobile Device zu.
Dazu müssen wir die zugehörige `Companion App` auf unserem Mobile Device installieren. Um die `Companion App` installieren zu können, ist es nötig, einige Entwickleroptionen auf unserem Gerät zu aktivieren. Wir folgenden hier den Anweisungen der MIT-Seite:

`http://appinventor.mit.edu/explore/content/setup-device-usb.html`

Unter der Rubrik Einstellungen befindet sich der Menüpunkt *Entwickleroptionen*. Diesen wählen wir aus und aktivieren USB-Debugging und Wach bleiben (ggf. auch Aktiv lassen):

Abbildung 3.1: Entwickler-Optionen und entsprechende Menüpunkte aktivieren.

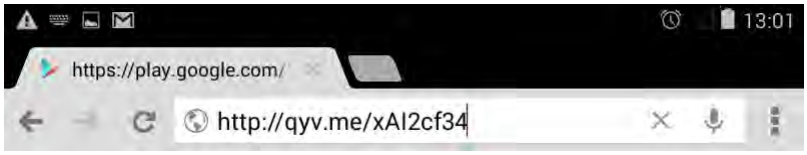
Bei manchen Modellen ist der Menüpunkt *Entwickleroptionen* unter Werkseinstellungen nicht sichtbar. Um diesen freizuschalten wählen wir unter der Rubrik *Einstellungen* den Menüpunkt *Über* aus und Tippen das Feld *Build-Nummer* solange an, bis die Meldung *Sie sind jetzt Entwickler* erscheint. Der Menüpunkt *Entwickleroptionen* ist nun in der Rubrik Einstellungen sichtbar. Sollte dies bei ihrem Gerät nicht funktionieren, können Sie im Internet nachlesen, wie man auf ihrem Mobile Device die Entwickleroptionen aktiviert.

Bei manchen Aufgaben kann es erforderlich sein, Apps ohne Google Play-Konto auf dem Mobile Device zu installieren. Dabei kommt es zu Problemen mit den Sicherheitseinstellungen. Für diesen Fall aktivieren Sie in den Handyeinstellungen unter *Sicherheit* die Rubrik *Unbekannte Quellen* - Installation von Apps von anderen Quellen als Play Store erlauben. Aus Sicherheitsgründen deaktivieren Sie bitte diese Rubrik nach Installation wieder.

3.4 Installation der Companion App

3.4.1 Installation ohne Barcodescanner

Zunächst müssen wir sicherstellen, dass unser Mobile Device über eine funktionsfähige Verbindung mit dem Internet verfügt. Je nach Mobile Device gibt es dafür verschiedene Möglichkeiten, wobei es sich anbietet, eine Internetverbindung über das lokale WLAN herzustellen.
Um die Companion App zu installieren, tippen wir den folgenden Link in die Adressleiste des Browsers des Mobile Device ein und starten den Download mit Bestätigen der `Return`-Taste:



Die Datei `MITAI2Companion.apk` wird nun heruntergeladen. Nach Abschluss des Downloadvorgangs tippen wir die neu heruntergeladene Datei an und installieren diese mit mehrmaligem Betätigen der *Weiter*-Schaltfläche im Installationsmenü. Der erfolgreiche Abschluss der Installation der `Companion App` zeigt sich, indem das zugehörige App-Symbol auf unserem Mobile Device angezeigt wird. Die Companion App ist nun erfolgreich auf unserem Mobile Device installiert. Anders als bei der AppInventor Software auf unserem Rechner, muss die `Companion App` auf unserem Mobile Device nicht explizit ausgeführt werden. Die `Companion App` öffnet sich beim Programmieren mit dem AppInventor automatisch.

Abbildung 3.2: Icon der `Companion App` auf dem Mobile Device.

3.4.2 Installation mit Barcodescanner

Sollte der in Abschnitt 3.4.1 beschriebene Weg nicht funktionieren, kann auch ein alternativer Installationsweg gewählt werden. Wir installieren dazu zuerst die App *Barcodescanner* auf unserem Mobile Device - siehe dazu Abschnitt 7.1 im Anhang.

Öffnen Sie nun den *Barcodescanner* auf ihrem Mobile Device und scannen folgenden QR-Code ein:

Im darauf folgenden Auswahlmenü wählen wir *Im Browser öffnen*, wodurch der Downloadvorgang der `Companion App` auf unserem Mobile Device gestartet wird.

3.5 Mobile Device mit App Inventor verbinden

Wir versuchen nun die Verbindung zwischen Mobile Device und App Inventor herzustellen. Dazu verbinden wir das Mobile Device mit dem USB-Verbindungskabel mit dem Laptop bzw. Desktop-PC. Wählen Sie auf dem Handy je nach Hersteller entweder *Nur laden* oder *Als Kamera (PTP)* aus: [1]

Um die Verbindung unseres Mobile Devices zu testen, rufen wir folgende Internetseite im Browser unseres Rechners auf:

`http://appinventor.mit.edu/test/`

Sollte das angezeigte Ergebnis des Verbindungstests wie folgt aussehen, ist das Mobile Device erfolgreich mit dem Rechner verbunden und es kann mit dem nächsten Kapitel fortgefahren werden:

[1] Dies ist vor allem bei Entwicklergeräten erforderlich. Bei Geräten von anderen Herstellern kann es sein, dass Mediengerät (MTP) ausgewählt werden muss. Besitzen Sie kein Entwicklergerät und wird Ihr Mobile Device nicht automatisch angezeigt, müssen Sie nach der Intallation des OEM Treiber (s. nächster Abschnitt) ggf. mit den Einstellungen experimentieren, um ihr Mobile Device mit dem AppInventor verbinden zu können.

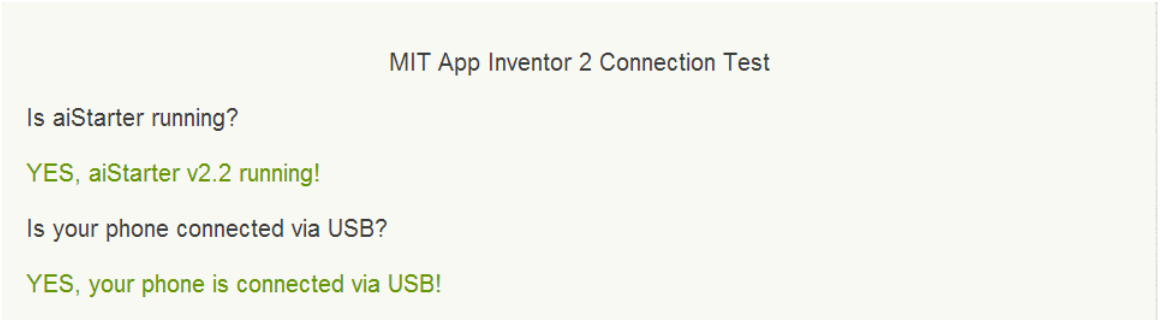


Abbildung 3.3: Testen, ob das Mobile Device korrekt mit dem AppInventor verbunden ist

Erscheint beim Verbindungstest die Fehlermeldung *NO, aiStarter is not running!*, wurde die AppInventor-Software auf dem Rechner nicht gestartet. Starten Sie in diesem Fall die AppInventor-Software über das `aiStarter`-Symbol auf dem Desktop Ihres Rechners und nehmen Sie einen erneuten Verbindungstest vor.

Wird hingegen die Fehlermeldung *NO, your phone is not connected via USB!* ausgegeben, wurde der benötigte USB-Treiber für das Mobile Device nicht oder nicht korrekt installiert. Wir installieren deshalb die benötigten Treiber manuell.

3.6 Manuelle Installation des USB-Treibers

Wir folgen bei diesem Schritt wieder den Anweisungen der ANDROID-Developer Seite

`http://developer.android.com/sdk/win-usb.html` .

Falls Ihr Mobile Device ein Entwickler-Gerät ist, wie z.B. das Nexus One, Nexus S oder die Nexus-Reihe von Asus, benötigen Sie den `Google USB Treiber`. Besitzen Sie ein Mobile Device von einem anderen Hersteller, benötigen Sie einen `OEM Treiber`, der je nach Hersteller variieren kann. Wir werden die Installation des `Google USB Treiber` im folgenden exemplarisch durchführen. Für den Fall, dass Sie kein Entwicklergerät besitzen, verweisen wir auf die Seite

`http://developer.android.com/tools/extras/oem-usb.html` ,

auf der Sie im unteren Seitenbereich gezielt für Ihr Gerät einen `OEM Treiber` installieren können [2].

Den `Google USB Treiber` können Sie auf folgender Internetseite downloaden:

`http://developer.android.com/sdk/win-usb.html#download` .

Extrahieren Sie die heruntergeladene `.zip`-Datei.

Mit Hilfe des heruntergeladenen Treibers können wir nun das Mobile Device mit dem Laptop bzw. Desktop-PC verbinden: Schließen Sie das Mobile Device an Ihren Laptop bzw. Desktop-PC an und öffnen Sie den `Geräte-Manager`[3]. Mit Rechtsklick auf das angeschlossene Mobile Device haben Sie die Möglichkeit, die Treibersoftware zu aktualisieren:

[2] Bei HTC-Geräten kann dieser Treiber über den Sync-Manager installiert werden
[3] Öffnen Sie dazu das Startmenü und tippen Sie in das Suchfeld **devmgmt.msc**.

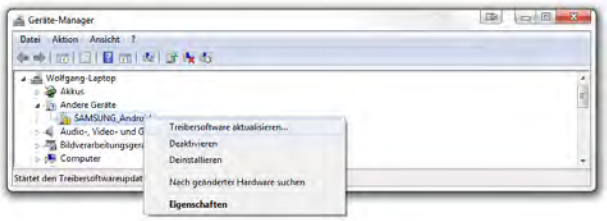
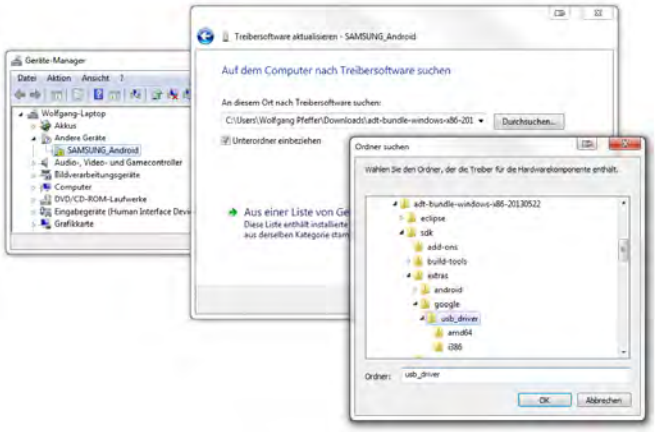


Abbildung 3.4: Manuelles Aktualisieren der Treiber-Software im Geräte-Manager.

Wählen Sie anschließend *Auf dem Computer nach Treibersoftware suchen* und navigieren Sie im Dateibaum zu dem heruntergeladenen Treiber:



Das gelbe Warndreieck sollte im Gerätemanager verschwunden sein obiger Test in Abbildung 3.3 erfolgreich sein. Behebt der `Google USB Treiber` das Problem nicht, kann die Installation eines `OEM Treibers` erforderlich sein. Wir verweisen hier auf den Anfang dieses Abschnitts.

Kapitel 4

Die ersten Schritte mit dem App Inventor

Überblick:

Dieses Kapitel bietet Ihnen einen schrittweisen Einstieg in die Entwicklungsumgebung des AppInventors. In Abschnitt 4.1 schauen wir zunächst kurz „unter die Motorhaube“ und geben eine Übersicht über das Zusammenspiel von Mobile Device, Computer und der „Cloud“. In Abschnitt 4.2 entwickeln wir unsere erste eigene App mit dem AppInventor und gehen dabei auf die wichtigsten Werkzeuge und Möglichkeiten ein.

4.1 App-Inventor aus der Entwickler-Sicht

Wir betrachten den AppInventor zunächst aus der Entwickler-Perspektive und orientieren uns dabei an der Seite

`http://appinventor.mit.edu/explore/content/what-app-inventor.html` .

Mit dem AppInventor können wir eigene Applikationen erstellen und auf unserem verbunden Mobile-Device oder Emulator laufen lassen. Der AppInventor läuft dabei in unserem Webbrowser. Die AppInventor-Server speichern unsere Projekte. Folgende Übersicht auf oben genannter Internetseite visualisiert die Funktionsweise des AppInventors:

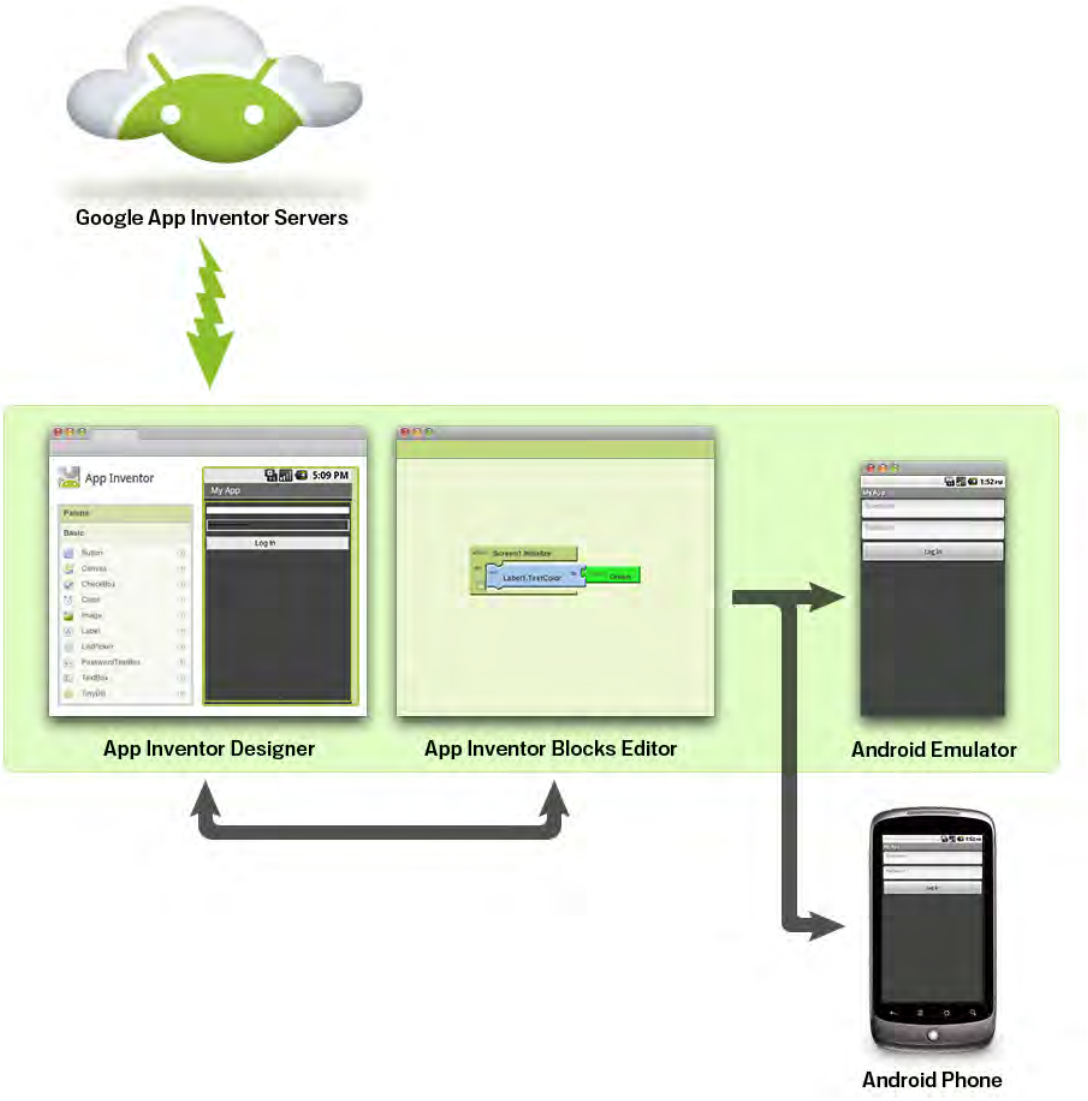


Abbildung 4.1: Der AppInventor aus der Entwickler-Sicht: Das Zusammenspiel von AppInventor Server, Webbrowser und Mobile Device (Quelle: oben angef. Internetseite).

Für uns als Anwender sind generell nur die Perspektiven `Designer-Sicht` und `Blocks-Sicht` interessant - vom Ablauf „in the cloud“ bekommen wir nichts mit. Daher betrachten wir diese beiden Sichten genauer:

4.1.1 Die Designer-Sicht

Die Designer-Sicht (s. Abb. 4.2) kennen wir bereits aus Kapitel 3, haben ihr während der Installation allerdings noch keine Bedeutung zugewiesen.

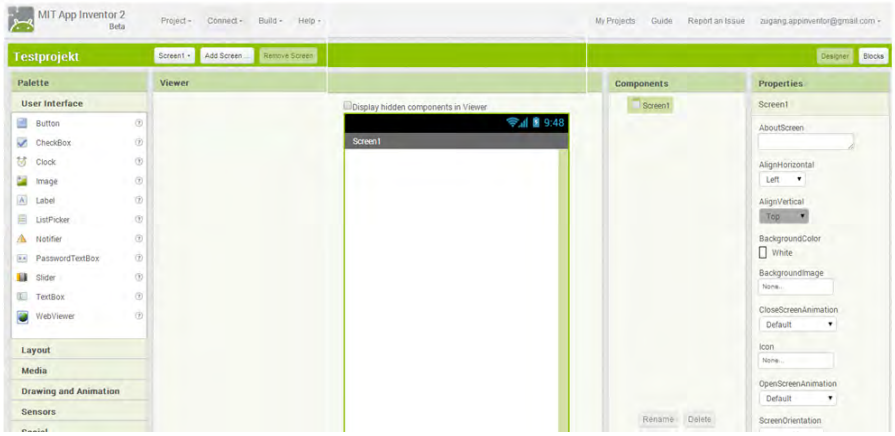


Abbildung 4.2: Die Designer-Sicht des AppInventors - Vordefinierte Klassen und Erzeugen von Objekten.

Aus dem Bereich der objektorientierten Programmiersprachen ist Ihnen das Konzept von Klassen, Objekten und Methoden bekannt. In der Designer-Sicht gibt es eine übersichtliche Zahl vordefinierter Klassen, die in verschiedene Kategorien aufgeteilt sind - in Abb. 4.2 ist die Kategorie Basic mit einigen Klassen, wie z.B. Button, Clock und Slider, zu sehen. Weitere Kategorien sind beispielsweise Media, Animation und Sensors. Die Definition eigener Klassen ist nicht möglich.
Das Mobile Device ist in der Mitte grafisch dargestellt. Per Drag & Drop auf den `Screen1` können Sie Objekte erzeugen und initialisieren.

4.1.2 Die Blocks-Sicht

Nachdem wir in der Designer-Sicht Objekte erstellt haben, müssen wir diesen noch „Leben einhauchen“. Dies geschieht in der Blocks-Sicht.
Die erzeugten Objekte bieten Methoden (z.B. Auslesen und Schreiben von Attributwerten, Starten von anderen Apps, Reagieren auf bestimmte Ereignisse) in Form von Bausteinen an.
Neben den Objekt-Methoden gibt es auch statische Methoden (z.B. Zufallszahlen erzeugen oder Texte konkatenieren), deren Bausteine in Kategorien geordnet sind. Die Deklaration lokaler sowie globaler Variablen ist ebenso möglich wie die Definition eigener Prozeduren und Funktionen.
Aus diesen vorgefertigten Bausteinen lassen sich nun eigene Skripte zur Ereignissteuerung zusammenbauen.

4.1.3 Übersicht

Um diesen Abschnitt abzurunden, fassen wir die wichtigsten Eigenschaften beider Sichten zusammen:

Designer-Sicht

- es gibt eine übersichtliche Zahl vordefinierter Klassen
- es ist nicht möglich, eigene Klassen zu definieren
- man sucht sich vordefinierte Klassen aus, erzeugt und initialisiert Objekte

Blocks-Sicht

- Objekte bieten
  - Methoden, die Attributwerte auslesen
  - Methoden, die Attributwerte (über)schreiben
  - ggf. Methoden, mit denen andere Apps gestartet werden können
  - Ereignisverarbeiter

  in Form von Bausteinen an
- es gibt auch statische Methoden, deren Bausteine in Kategorien geordnet sind
- es können eigene Prozeduren und Funktionen definiert, sowie Variablen deklariert werden
- eigene Skripte werden im Blocks-Editor aus diesen Bausteinen zusammengebaut

4.2 Schritt für Schritt zur ersten eigenen App

Nachdem wir im letzten Abschnitt den Aufbau des AppInventors genauer unter die Lupe genommen haben, wollen wir uns nun der Entwicklungsumgebung zuwenden. Wir stellen dabei die wichtigsten Funktionen anhand einer Beispiel-Aufgabe vor.
Wir wechseln dazu in die `Designer-Sicht` und klicken auf `My Projects`:

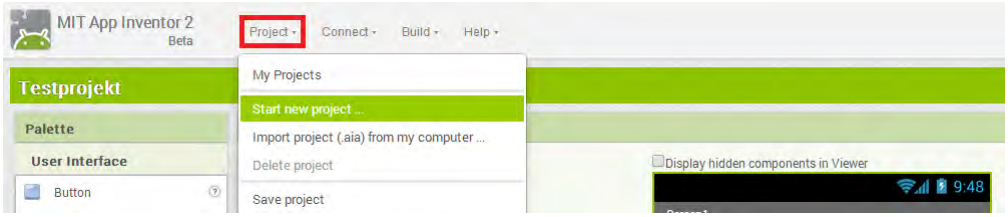


Abbildung 4.3: Über die Schaltfläche `My Projects` kann man sich eine Übersicht der erstellten Projekte anzeigen lassen.

Mit der Schaltfläche `Start new project` kann man ein neues Projekt anlegen - wir nennen unser erstes Projekt *Schuetteln*:

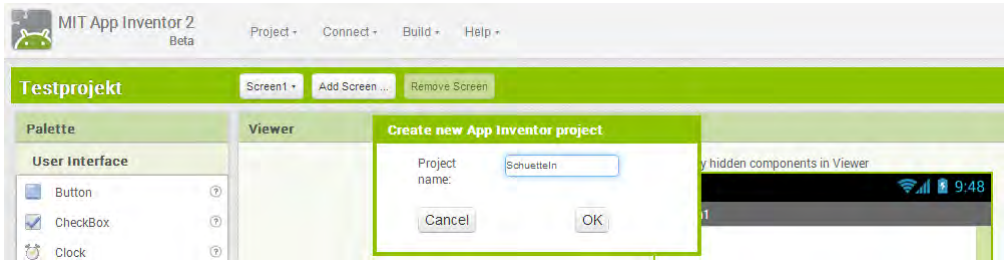


Abbildung 4.4: Jedes neue Projekt bekommt einen eindeutigen Namen.

Die Webanwendung öffnet anschließend automatisch die `Designer-Sicht` des neu angelegten Projekts.

Wir gehen im Folgenden davon aus, dass unser Mobile Device erfolgreich mit dem AppInventor verbunden ist (bei Problemen siehe Kapitel 3).

4.2.1 Hintergrundfarbe ändern

Unser erstes Ziel ist es, die Hintergrundfarbe unserer App von weiß nach orange zu ändern, wenn wir das Mobile Device schütteln.
Um auf Schütteln reagieren zu können, brauchen wir ein Objekt der Klasse `Beschleunigungssensor`. Dieses müssen wir in der `Designer-Sicht` erzeugen. Wir wählen dazu links in der Liste die Kategorie *Sensors* und ziehen per Drag & Drop den *AccelerometerSensor* auf das virtuelle Mobile Device. Unter dieser virtuellen Ansicht müsste nun ein Objekt *AccelerometerSensor1* angezeigt werden. Die gerade durchgeführten Schritte sind in Abbildung 4.5 noch einmal grafisch dargestellt. In dem Bereich `Components` können wir das erzeugte Objekt umbenennen - wir nennen es in unserem Beispiel *Beschleunigungssensor*.

Das eben erzeugte Objekt wollen wir nun mit Funktionalität ausstatten. Dazu wechseln wir in die `Blocks-Sicht`.

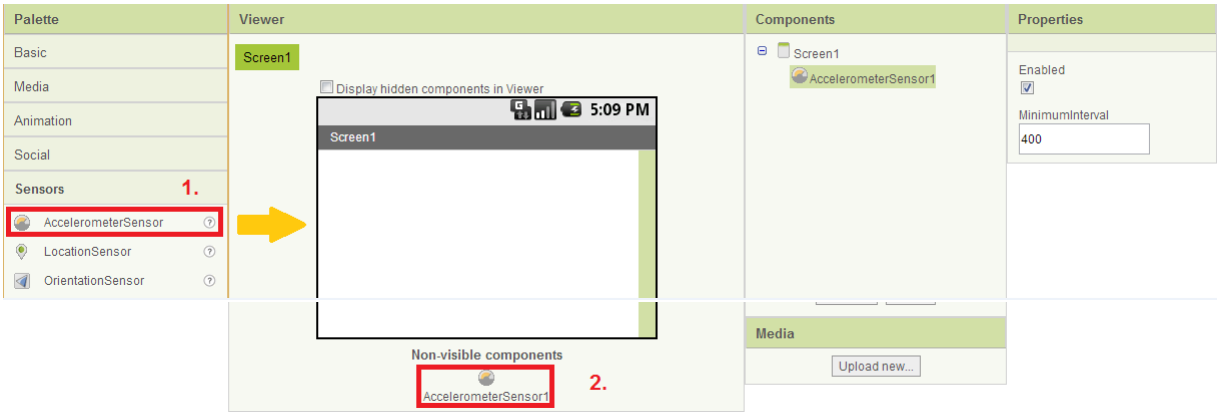


Abbildung 4.5: Mit Drag & Drop können wir ein Objekt der Klasse Beschleunigungssensor hinzufügen

In der linken Spalte ist nun unter der Rubrik `Screen1` unser `Beschleunigungssensor` aufgelistet. Klicken wir auf diesen, öffnet sich ein PopUp-Menü mit allen Ereignissteuerungen und Methoden des Objekts Beschleunigungssensor.
Um auf das Ereignis Schütteln reagieren zu können, ziehen wir den Baustein *when Beschleunigungssensor.Shaking* nach rechts auf die weiße Fläche. Je nachdem wie wir auf das Ereignis Schütteln reagieren wollen, müssen wir nun das Innere des Bausteins füllen. Zunächst wollen wir die Hintergrundfarbe der App auf orange setzen. Dies erreichen wir, indem wir auf `Screen1` klicken und den Baustein *set Screen1.BackgroundColor to* auswählen. Die Farbpalette befindet sich in dem Tab `Built-In` unter `Colors`.

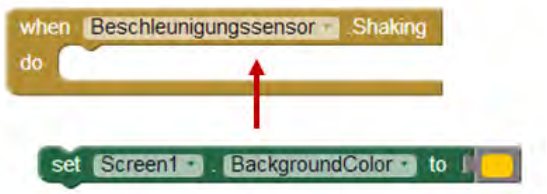


Abbildung 4.6: Was durch Schütteln ausgeführt werden soll, muss in den Ereignisverarbeiter gezogen werden.

Schütteln wir nun unser angeschlossenes Mobile Device! Die Hintergrundfarbe ändert sich von weiß nach orange.

4.2.2 Text ändern

Als nächstes wollen wir ein Objekt der Klasse Label erzeugen und mit den Attributwerten dieses Objekts experimentieren.
Die Klasse Label befindet sich in der `Designer-Sicht` in der Kategorie `User Interface` und kann wieder per Drag & Drop hinzugefügt werden. Anders als zuvor beim Hinzufügen des Beschleunigungssensors wird das Label-Objekt *Label1* direkt auf dem virtuellen Bildschirm angezeigt. Wir nennen dieses *Textausgabe*.

Abbildung 4.7: Das Objekt *Textausgabe* wird nach dem Hinzufügen auf dem virtuellen Bildschirm angezeigt.

Im `Properties`-Bereich können wir nun mit den Attributwerten des Objektes experimentieren, etwa Schriftgröße, Font oder Schriftart verändern. Auffällig ist, dass es bei dem Objekt *Textausgabe* kein Attribut gibt, um es auf dem virtuellen Bildschirm zu zentrieren. Wo muss man suchen?

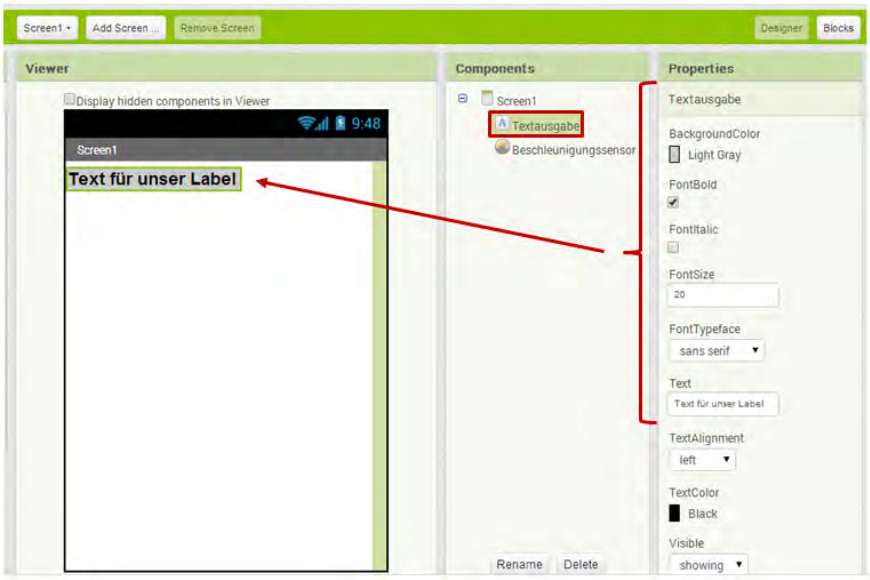


Abbildung 4.8: Attributwerte des Label-Objektes können initial beliebig verändert werden.

Unser Label hat ein Vater-Objekt! Wer beispielsweise schon mit Java Swing oder einer anderen GUI[1]-Programmiersprache in Kontakt gekommen ist, dürfte dies nicht verwundern. Falls nicht, ist dies auch kein Problem. Die Schülern werden bis auf wenige Ausnahmen auch kein Hintergrundwissen haben.
Für uns reicht es an dieser Stelle aus, zu verstehen, dass das Label-Objekt auf dem Screen-Objekt liegt und somit das Screen-Objekt Vater des Label-Objektes ist. Ganz nach dem Leitsatz

Alle Kinder haben sich nach dem Vater (oder der Mutter) zu richten!

bestimmt also das Vater-Objekt die vertikale Ausrichtung des Kind-Objektes *Textausgabe* auf „sich“. Ähnlich zu dieser Analogie verwenden GUI-Programmiersprachen für die Organisation oft eine sog. *Containersicht*. Die Elemente eines Containers sind gleichzeitig seine lokalen Variablen. Über diese Referenz legt der Container einige Eigenschaften, wie z.B. die Ausrichtung oder Position fest.

Manche stellen sich bestimmt die Frage, warum das Vater-Objekt nicht auch Schriftgröße, Schriftart und Font bestimmt. Anders als die vertikale Ausrichtung betreffen die anderen Attribute (wie Schriftgröße und Font) den Vater nicht direkt. Was ihn nicht direkt betrifft, hat ihn auch nicht zu interessieren.[2]

In Abbildung 4.9 ist noch einmal grafisch dargestellt, wie man die vertikale Ausrichtung des Label-Objektes verändern kann.

[1] *Graphical User Interface* - grafische Benutzeroberfläche, die dem Benutzer die Interaktion mit der Maschine über grafische Symbole erlaubt.

[2] Greifen wir an dieser Stelle unsere Vater-Kind-Analogie noch einmal auf: Der Vater bestimmt zwar die Lage des Kinderzimmers, aber die Einrichtung ist dem Kind überlassen!

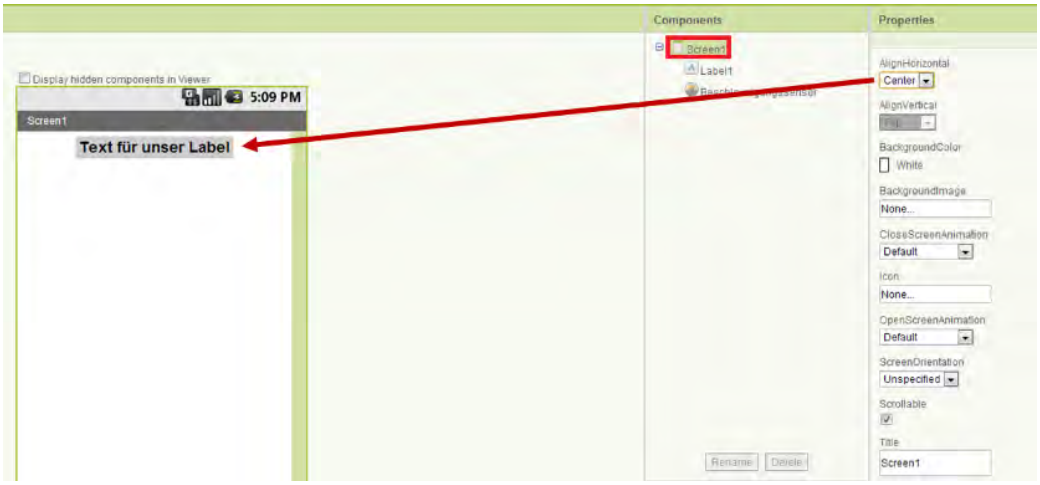


Abbildung 4.9: Die vertikale Ausrichtung des Label-Objektes *Textausgabe* wird von seinem Vater-Objekt *Screen1* festgelegt.

Wir möchten wie bisher die Hintergrundfarbe des Bildschirms auf orange setzen und zusätzlich den Inhalt des Labels auf *Hallo Welt* ändern. Versuchen Sie diese Aufgabe zunächst selbstständig zu lösen, bevor Sie weiterlesen.

Um den Inhalt des Labels bei einem Schüttel-Ereignis verändern zu können, wechseln wir wieder in die `Blocks-Sicht`. Wir fügen unseren bisherigen Bausteinen den Baustein *set Textausgabe.Text to* und ein Text-Objekt *Hallo Welt!* hinzu:

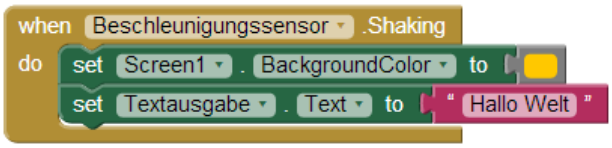


Abbildung 4.10: Beim Schütteln ändert sich die Hintergrundfarbe und der Inhalt der Textausgabe!

4.2.3 Schriftgröße während Schütteln variieren

Wir erweitern unsere Aufgabenstellung dahingehend, dass sich die Schriftgröße der Textausgabe während des Schüttelns verändern und dabei beliebige Werte[3] im Bereich 1 bis 100 annehmen soll.

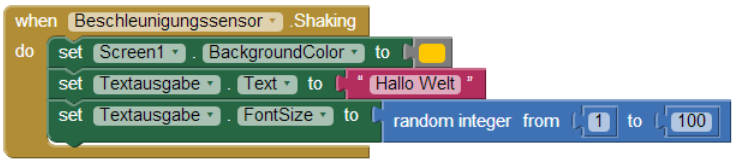


Abbildung 4.11: Während des Schüttelns ändert sich die Schriftgröße der Textausgabe und nimmt beliebige Werte im Bereich 1 bis 100 an.

[3]Hinweis: In der `Blocks-Sicht` gibt es hierfür einen vorgefertigten Baustein *random integer*. Wo könnte dieser zu finden sein?

4.2.4 Einfache Kontrollstrukturen

In diesem Unterabschnitt gehen wir exemplarisch auf die Kontrollstrukturen des AppInventors ein.

Wir möchten dazu gerne während dem Schütteln die Hintergrundfarbe auf orange setzen, wenn sie weiß ist und die Hintergrundfarbe auf weiß setzen, wenn sie orange ist. Das Ändern der Schriftgröße aus vorherigem Unterabschnitt 4.2.3 möchten wir beibehalten.

Hinweis: Die verfügbaren Kontrollstrukturen sind im Tab `Built-In` unter der Kategorie `Control` zu finden.

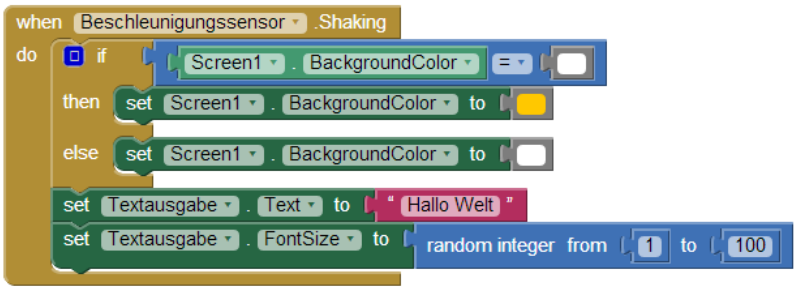
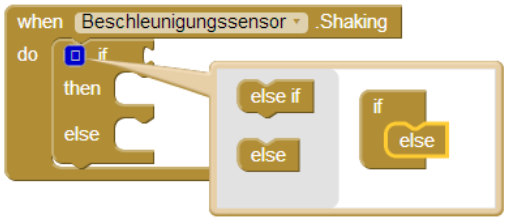
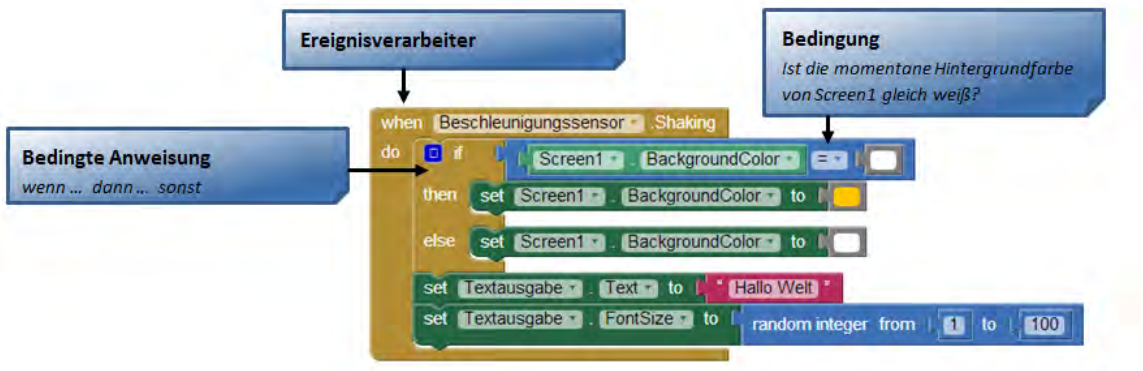


Abbildung 4.12: Der AppInventor bietet auch Kontrollstrukturen an, um auf bestimmte Gegebenheiten entsprechend reagieren zu können.

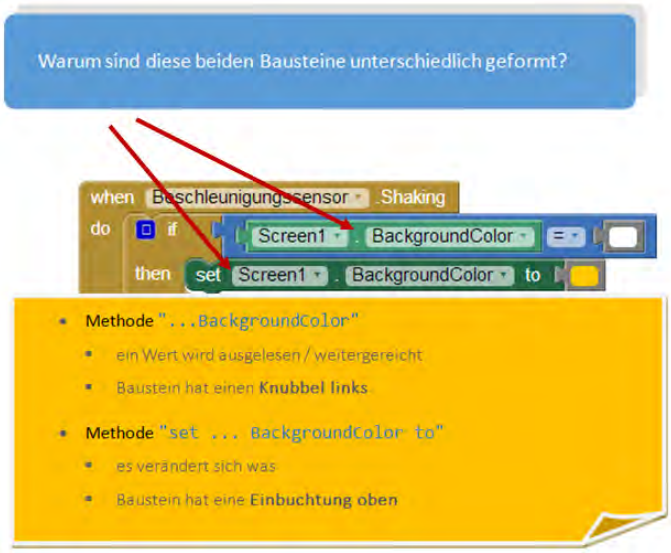
Einen `else`-Fall kann man bei einem `if`-Baustein über die blaue Schaltfläche hinzufügen:



Begriffe:



Exkurs: `Get-` und `Set-`Methoden



Im Bezug auf eine einzelne Eigenschaft eines Objektes gibt es zwei unterschiedliche Typen von Zugriffsmethoden:

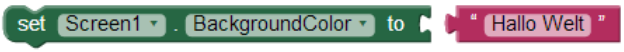
- **sondierende Methoden** (Getter)

  Zugriffsmethode, die eine Eigenschaft eines Objektes erfragt. Meist wird dabei der Wert eines Attributs ausgelesen.

- **verändernde Methdoen** (Setter)

  Zugriffsmethode, die eine Eigenschaft eines Objektes ändert. Im Gegensatz zu einer Get-Methode muss bei einer Set-Methode der neue Wert eines Attributs festgelegt werden, deshalb ist bei `"set ... BackgroundColor to"` rechts eine Einbuchtung, bei der man den neuen Wert festlegen kann.

  Wichtig ist, dass der neue Wert den Anforderungen der Set-Methode gerecht wird. Es ist nicht möglich, der Set-Methode `"set ... BackgroundColor to"` ein Text-Objekt zuzuweisen:

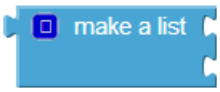


  Eine solche Zuweisung ist nicht möglich! Der AppInventor lässt uns diese beiden Bausteine nicht zusammenfügen. Kannst du dir vorstellen warum?

4.2.5 Zufällige Farbwahl

In obiger Aufgabe haben wir mit der Kontrollstruktur `ifelse` die Hintergrundfarbe abwechselnd auf weiß und orange gesetzt. Nun möchten wir die Hintergrundfarbe auf eine zufällig aus einer Liste ausgewählte Farbe setzen.
Der AppInventor bietet uns dazu eine Listenstruktur mit Zugriffsfunktionen an. Diese sind unter dem Tab

`Built-In` in der Kategorie zu finden.
Mit dem Baustein



lässt sich eine Liste aus verschiedenen Elementen erstellen. Mit der blauen Schaltfläche kann man festlegen, wie viele Elemente die Liste haben soll:

Mit einem weiteren Baustein lässt sich aus einer Liste ein beliebiges Element entnehmen.

Ändern Sie nun die bisherige App dahingehend ab, dass beim Schütteln ein beliebiger Hintergrund aus einer Liste von mindestens fünf verschiedenen Farben ausgewählt wird.

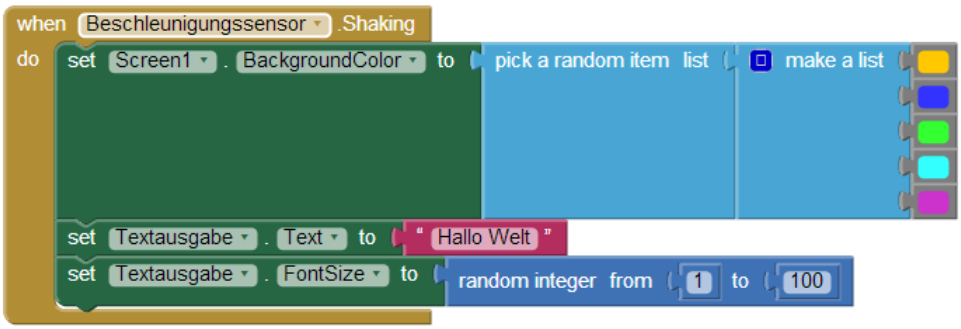
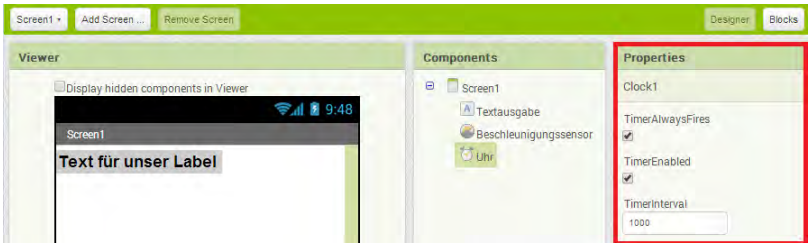


Abbildung 4.13: Beim Schütteln wird zufällig eine Farbe aus einer „Farb-Liste“ ausgewählt.

4.2.6 Schluss mit Schütteln

Bislang haben wir die Ereignisse selber ausgelöst, indem wir das Mobile Device geschüttelt haben. Dies möchten wir nun automatisieren - unsere Anweisungen sollen jede Sekunde ausgeführt werden.
Dazu stellt uns der AppInventor die Klasse `Clock` zur Verfügung. Wir können davon in der `Designer-Sicht` ein Objekt mit dem Namen *Uhr* erstellen. Unter `Properties` können wir das Timer-Intervall festlegen:



Die Einheit des Timer-Intervalls ist Millisekunden - wir müssen also den Wert 1000 nicht verändern.

Versuchen Sie, das Ereignis und die entsprechende Reaktion darauf (Hintergrundfarbe setzen und Schriftgröße verändern) nicht mehr durch Schütteln sondern sekündlich auszulösen.

4.2.7 Schriftgröße aus einer Liste auswählen

Bisher haben wir die Schriftgröße auf einen beliebigen Wert zwischen 1 und 100 gesetzt. In diesem Unterabschnitt wollen wir dem App-Benutzer nur eine begrenzte Anzahl an Möglichkeiten für die Schriftgröße zur Verfügung stellen. Um einen Wert für die Schriftgröße auszuwählen, bietet uns der AppInventor die Klasse `ListPicker`. Diese Klasse setzt einen speziellen Button um. Er hat zwei Zustände: eine Schaltfläche und eine Auswahlliste. Bei der Ereignisverarbeitung muss man dabei zwei Ereignisse unterscheiden:

- `BeforePicking` - hier wird festgelegt, was nach dem Drücken der Schaltfläche angezeigt werden soll
- `AfterPicking` - hier wird festgelegt, was mit dem aus der Auswahlliste gewählten Wert passieren soll

Folgendes Zustandsdiagramm soll die Funktionsweise des `ListPicker`s noch einmal verdeutlichen:

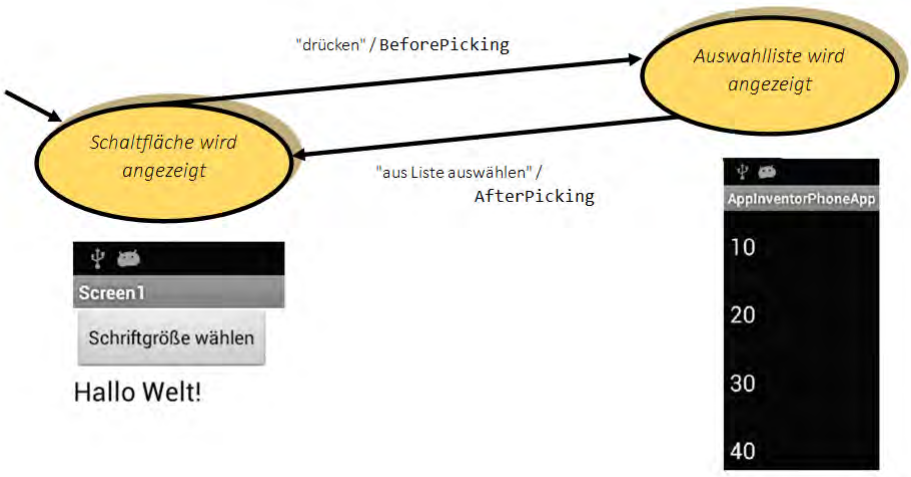
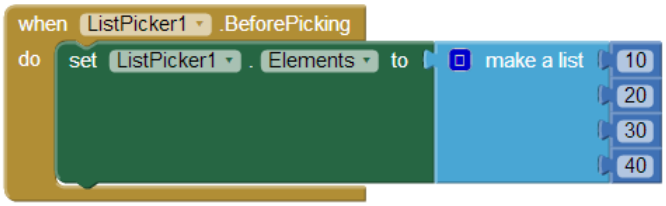


Abbildung 4.14: Zustandsdiagramm der Klasse `ListPicker`.

Ändern Sie mit Hilfe der Klasse `ListPicker` die App dahingehend ab, dass man für die Schriftgröße des Textausgabe-Objekts nur noch die Werte 10,20,30 und 40 wählen kann.

Hinweis: Man kann dem `ListPicker`-Objekt auch eine Liste an Elementen zuweisen.



Programmierumgebung erkunden - Worum ging es?

Designer-Sicht

- Objekte erzeugen, Objekte umbenennen
- Objekte betrachten: Attribute haben Attributwerte
- Anfangswerte der Attribute anpassen (initialisieren)

Blocks-Sicht

- Ereignisverarbeiter nutzen (sind bei einem Objekt angemeldet)
- Methoden nutzen (Objekte bieten Methoden an)
- Auf Gleichheit testen
- Bedingte Anweisungen verwenden
- Attributwerte auslesen (Knubbel links) und setzen (`set ... to`)
- Eine Liste anlegen und zufällig auf ein Listenelement zugreifen
- Mit Objekten vom Typ Screen, Label, Beschleunigungssensor, Uhr, ListPicker arbeiten

Anhand der verschiedenen Aufgaben in diesem Kapitel haben Sie einen Einblick in die Programmierumgebung der AppInventor-Software erhalten und sich bereits mit verschiedenen Klassen und der Ereignissteuerung in der `Blocks-Sicht` vertraut gemacht. Mit diesem Vorwissen sollten Sie die Aufgaben aus Kapitel 5 bearbeiten können.

Kapitel 5

Aufgaben für Schülerinnen und Schüler

Überblick:

Dieses Kapitel beinhaltet die Aufgaben für die Schülerinnen und Schüler. Das Aufgabenmaterial liegt noch einmal in einer eigenen PDF-Datei und einer WORD-Datei bei, so dass Sie ggf. Änderungen und Erweiterungen am Aufgabenmaterial vornehmen können.

Haben Sie Verbesserungsvorschläge oder Ideen zu neuen Aufgaben? Bitte senden Sie uns Ihre Anregungen per Mail an wolfgang.pfeffer@uni-passau.de .

5.1 Würfelbecher-App (leicht)

Zwei Würfel sollen gewürfelt werden, indem man das Handy schüttelt. Während man würfelt, sieht man nur den Würfelbecher und man hört ein Würfelgeräusch. Wenn man den Würfelbecher „hochhebt“ (berührt), dann erscheinen die gewürfelten Zahlen.

Nähere dich der Lösung schrittweise, indem du zuerst die folgenden Teil-Probleme löst:

Arbeite mit einem Ereignisverarbeiter, der auf Schütteln reagiert

Verwende zwei Label-Objekte, um zwei Ziffernanzeigen darzustellen. Kümmere dich noch **nicht** um **Becher** und **Würfelgeräusch**. Während man das Handy schüttelt, soll man sehen, wie sich die Ziffernanzeigen zufällig verändern.

Hinweis: Wie du in der Einführungsaufgabe bereits gesehen hast, kann man die Zufallszahlen durch Zahlenwerte nach oben und unten eingrenzen.

Arbeite mit zwei Ereignisverarbeitern

Nimm ein Button-Objekt als Würfelbecher hinzu. Du hast zwei Zustände deiner Anzeige zu verwalten:

- Button-Objekt (später: Becher) ist zu sehen, die Ziffernanzeigen nicht.
- Button-Objekt ist nicht zu sehen, die Ziffernanzeigen schon.

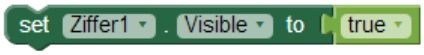
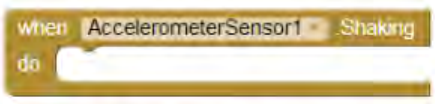


Abbildung 5.1: Sichtbar machen und unsichtbar machen geht über den Baustein `Visible` und den entsprechenden logischen Baustein `true` oder `false`.

Hinweis: Die festen Wahrheitswerte `true` und `false` befinden sich in der `Blocks-Sicht` in der Kategorie `Logic`. Überlege dir, welche Aufgaben die beiden Ereignisverarbeiter `...Click` bzw. `...Shaking` erledigen sollen.

Beachte, dass nur Anweisungen, die sich „in einem Ereignisverarbeiter befinden“ ausgeführt werden!



Hat der Würfelbecher / Haben die Ziffernanzeigen sichtbar zu sein, wenn das Mobile Device geschüttelt wird?

Was passiert mit dem Becher / den Ziffernanzeigen, wenn auf den Becher gedrückt wird?

Abbildung 5.2: Mach dir Gedanken, welche Anweisungen du in welchem Ereignisverarbeiter ausführen musst!

Bild und Ton hinzufügen

Du kannst dem Attribut `Image` des Buttons ein Würfelbecher-Bild zuweisen. Dazu muss das Bild zunächst auf den AppInventor-Server hochgeladen werden. In der `Designer-Sicht` gibt es dazu unter der Rubrik `Media` das

Feld *Upload new.*
Lade mit diesem Feld die Bild-Datei *Becher.png* hoch.

Wenn du während des Schüttelns eine Ton-Datei abspielen willst, musst du diese Ton-Datei in ähnlicher Weise auch erst auf den App Inventor-Server hochladen und in ein Sound-Objekt einfüllen. Das Sound-Objekt kannst du in der `Designer-Sicht` unter `Palette` → `Media` → `Sound` erzeugen.

Du sollst deine App dahingehend erweitern, dass während du das Mobile Device schüttelst, nur der Würfelbecher zu sehen ist und du ein Würfelgeräusch hörst. Nach dem Schütteln soll durch Klick auf den Becher das Würfel-Ergebnis angezeigt werden.

Erweiterung: Sprechender Würfelbecher

Nun soll der Würfelwurf beim „Anheben des Bechers“ vom Mobile Device vorgelesen werden.

Erzeuge dazu ein Objekt der Klasse `TextToSpeech`. Die Klasse `TextToSpeech` findest du in der `Designer-Sicht` im Bereich `Palette` unter `Media`. Dein erzeugtes Objekt bietet die Methode *Speak* an. Verwende diese Methode und erweitere dein bisheriges Programm.

Hinweis: In der `Blocks-Sicht` kannst du veranlassen, dass Textschnipsel „zusammengeklebt“ werden. So kannst du zwischen den Ziffern ein elegantes „und“ einfügen und dann alles auf einmal vorlesen lassen (vgl. Abbildung 5.3).

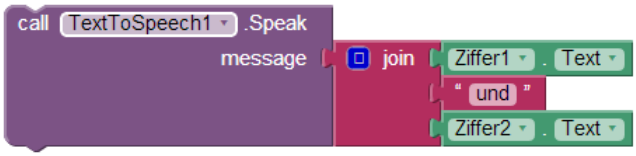


Abbildung 5.3: Mit der Methode *make text* kann man einzelne Wörter „zusammenkleben“.

Pasch

Der Würfelbecher soll um einen Pasch-Prüfer und einen Pasch-Zähler erweitert werden.

Bei Pasch Applaus

Deine Würfelapplikation soll bei jedem Wurf überprüfen, ob du einen Pasch gewürfelt hast. Wenn deine beiden Ziffernanzeigen (Label) *Ziffer1* und *Ziffer2* heißen, dann könnte das so aussehen:

Einem Pasch könnte ein Applaus folgen. Eine Sound-Datei mit Applaus haben wir dir auch zur Verfügung gestellt.

Es soll mitgezählt werden, wie oft ein Pasch gewürfelt wurde

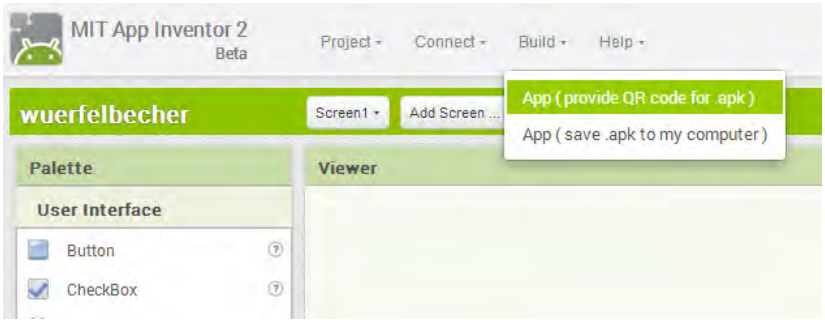
Du kannst mit einem neuen Label arbeiten, dem du den Namen *Zaehler* gibst. Den Text des Labels kannst du

auslesen (`Zaehler.Text`) und du kannst das Label neu beschriften (`set Zaehler.Text to ...`).
Vergiss nicht, deinem Zähler bei Programmstart einen sinnvollen Anfangszahlenwert zuzuweisen. Das ist z.B. via `Designer-Sicht` möglich.

Fertige Applikation auf das Mobile Device herunterladen

Bisher können wir die Applikation auf dem Mobile Device nur dann ausführen, wenn es mit dem USB-Kabel an den Computer angeschlossen und mit dem AppInventor verbunden ist.
Wir möchten das eben erstellte Würfelspiel aber gerne auch auf dem Mobile Device ausführen können, ohne dass es an den Computer angeschlossen ist. Dazu müssen wir die Applikation auf das Mobile Device herunterladen.

In der `Designer-Sicht` gibt es oben die Schaltfläche `Build`. Nach Klicken auf diese Schaltfläche können wir auswählen, ob wir die `.apk`-Datei über einen QR-Code downloaden, oder auf dem PC speichern wollen. Indem wir den QR-Code mit unserem Mobile Device einscannen, wird die App auf unserem Gerät installiert.



Die Applikation können wir ab jetzt auch ohne App Inventor und Computer auf unserem Mobile Device ausführen. Nehmen wir in der `Designer-` oder `Blocks-Sicht` Änderungen vor, müssen wir die Applikation erneut auf das Mobile Device herunterladen, um dort auch die aktuellste Version zu haben.

5.1.1 Würfelspiel (Optional, Fortgeschritten)

Das Programm soll so umgebaut werden, dass 2 Spieler gemeinsam spielen können. Dabei sollen die Spieler abwechselnd je mit **einem** Würfel spielen. Der Spieler mit der größeren Zahl bekommt einen Punkt.

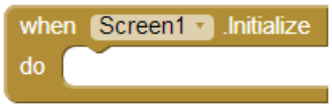
Punkteanzeige

Um eine schöne Punktetafel zu bekommen, kannst du ein Objekt der Klasse `TableArrangement` benutzen. Die Klasse findest du in der Kategorie `Layout`. Erstelle weiter die Label *Spieler1* und *Spieler2* und zur Punktanzeige *AnzahlSiege1* und *AnzahlSiege2* und füge diese in deine Punktetafel ein (Drag & Drop ausprobieren). Deinen Zähler brauchst du nun nicht mehr. Wie die Anzeige aussehen kann, zeigt dir Abbildung 5.4.

Abbildung 5.4: Punktetafel für zwei Spieler

Abwechselnd würfeln

Benutze zur Unterscheidung, wer an der Reihe ist, die Hintergrundfarbe der Spieler. Der würfelnde Spieler kann zum Beispiel gelb hinterlegt werden. Vergiss dabei nicht, zu Beginn des Spiels den Punktestand und die Hintergrundfarbe festzulegen, damit die Spieler in der richtigen Reihenfolge spielen:

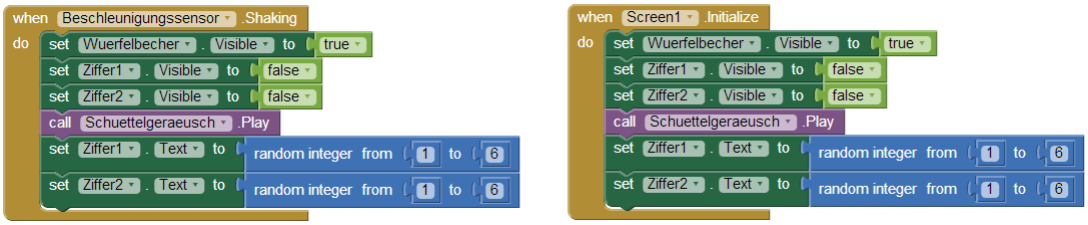


Die Hintergrundfarbe deiner Spieler kannst du ähnlich wie die Ziffern beim Pasch-Prüfer vergleichen.

Nun wird immerhin schon angezeigt, wer an der Reihe ist. Als nächstes musst du noch dafür sorgen, dass sich die Spieler abwechseln. Überlege dir auch, wann und wie die Punkte vergeben werden sollen. Folgende Stichpunkte können dir bei deinen Überlegungen helfen:

- Welcher Würfel muss „gewürfelt“ werden, wenn Spieler 1 bzw. Spieler 2 an der Reihe sind? Dürfen beide immer gewürfelt werden?
- Wann muss die Hintergrundfarbe der Label-Objekte `Spieler1` und `Spieler2` geändert werden?
- Wann muss verglichen werden, welcher Spieler die größere Zahl gewürfelt hat?
- Die Kontrollstruktur `ifelse` könnte dir helfen!

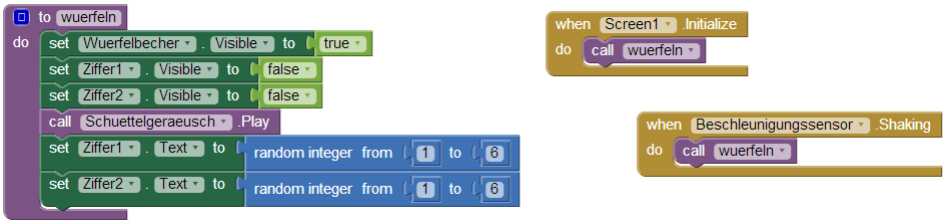
Exkurs: Eigene Methoden schreiben



Betrachten wir obigen Code-Ausschnitt genauer, stellen wir fest, dass in beiden Ereignisverarbeitern die gleiche Abfolge von Bausteinen vorkommt.
Beim Programmieren ist es oft so, dass an manchen Stellen im Programm-Code gleiche Code-Sequenzen auftreten. Neben viel Schreibarbeit kann dies zu weiteren Problemen führen: Was ist, wenn man z.B. ein anderes Schüttelgeräusch abspielen möchte? Hier ist eine Änderung in den Ereignisverarbeitern `Beschleunigungssensor.Shaking` und `Screen1.Initialize` nötig. Bei unserem kleinen Beispiel mag das nicht problematisch erscheinen. Stell dir aber einmal ein großes Programm vor, bei dem du an vielen Stellen die gleiche Änderung vornehmen musst! Dabei kann es schnell vorkommen, dass du vergisst, eine Stelle zu ändern.

Aus diesem Grund **kapselt** man gleiche Programmteile in eine eigenen Methode (auch Prozedur genannt). Der große Vorteil dabei ist, dass man die Änderung nur einmal vornehmen muss, ohne berücksichtigen zu müssen, wo diese Methode überall aufgerufen wird.

Der AppInventor bietet uns die Möglichkeit, eigene Methoden zu erstellen. Wir könnten beispielsweise obige Methode folgendermaßen abändern:



Versuche bei deinen Programmen Code-Teile, die öfters vorkommen, in einer eigenen Methode zu kapseln. Eine eigene Methode kannst du in der `Blocks-Sicht` unter `Built In` → `Procedures` erstellen.

5.1.2 Würfelspiel-Plus (Optional, Fortgeschritten)

Bis jetzt war bei gleicher Augenzahl keine Punktevergabe nötig. Ab jetzt soll die Punktezahl bei gleicher Zahl in der nächsten Runde verdoppelt werden. Die Spielregeln sollen zu Beginn für jeden Spieler durch eine Nachricht angezeigt werden, in der er auch seinen Namen angeben kann.

Bonus benutzen

Erstelle also zuerst eine Variable mit dem Namen `bonus`. Dies kannst du via `Built-In` → `Variables` machen:

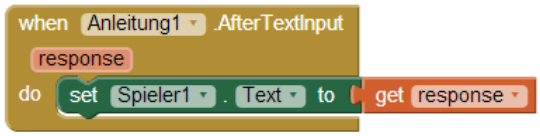
Der Wert soll im Normalfall 1 betragen. Nur bei gleicher Augenzahl soll er verdoppelt werden. Vergiss nicht,

dass der Wert, nachdem Punkte vergeben wurden, wieder auf 1 gesetzt werden muss.

Spielanleitung

Um die Spielregeln zu Beginn anzuzeigen, benötigst du wieder ein neues Objekt. Es ist von der Klasse `Notifier`. Diese findest du in der Kategorie `User Interface` in der `Designer-Sicht`.

Erstelle für jeden Spieler ein `Notifier`-Objekt, das du dann *Anleitung1* und *Anleitung2* nennst. Lasse den Text mit `Anleitung1.ShowTextDialog` zu Beginn des Spiels erscheinen.
Erst nachdem der Spieler den Namen eingegeben hat, wird der zum `Notifier-Objekt` zugehörige Ereignisverarbeiter aufgerufen. Er heißt `After-TextInput` und erwartet eben diesen Namen (wir wählen `Name1`) als Eingabe. Gib den Namen von Spieler 1 auf dem Phone-Display aus[1]:



Lasse als nächstes die Anleitung für Spieler 2 erscheinen und gib auch seinen Namen auf dem Phone-Display aus, bevor das Spiel beginnt.

Folgende Stichpunkte können dir bei deinen Überlegungen helfen:

- Wann muss das Objekt `Anleitung1` angezeigt werden? In welchen Ereignisverarbeiter muss der Baustein `Anleitung1.ShowTextDialog` also?
- Was muss im Ereignisverarbeiter `Anleitung1.After-TextInput` gemacht werden?
- Braucht man den Ereignisverarbeiter `Anleitung2.After-TextInput` auch?
- Wann muss der Wert der Variable `bonus` verdoppelt werden?
- Wann muss der Wert der Variable `bonus` wieder auf 1 zurückgesetzt werden?

[1] Den Baustein `get response` findest du, wenn du den Mauszeiger auf das orange `response`-Feld bewegst

5.2 Barcodescanner (leicht)

Wir wollen eine App entwickeln, die einen Barcode einscannt und uns den decodierten Text auf dem Mobile Device ausgibt. Zusätzlich soll uns der Text noch vorgelesen werden.

Mit Hilfe des Barcodescanner-Objekts kann man eine App (ZXING Barcodescanner) aufrufen und Rückgabewerte dieser App entgegennehmen. Diese App ist nicht im Android-Betriebssystem integriert. Überprüfen Sie zunächst, ob diese App bereits auf den Mobile Devices installiert wurde. Benutzen die Schüler ihr eigenes Mobile Device, können diese die App kostenlos aus dem GOOGLE PLAY STORE herunterladen. Verwenden Sie Mobile Devices der Schule ohne GOOGLE KONTO, zeigen wir Ihnen in Abschnitt 7.1, wie Sie die App ohne GOOGLE PLAY auf dem Mobile Device installieren können.

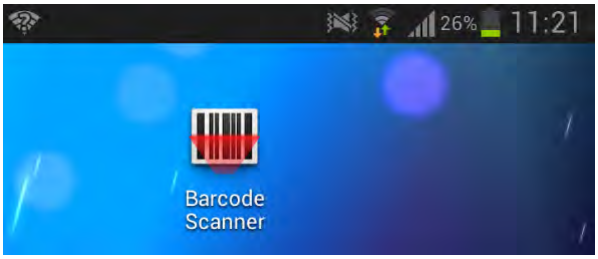


Abbildung 5.5: Erfolgreich installierte Barcodescanner-App auf dem Mobile Device.

Erstelle für diese Aufgabe im `Browser Editor` ein neues Projekt mit dem Namen *Barcode_Scanner*.

Barcode anzeigen

Bei dieser Aufgabe wirst du zum ersten Mal mit einem Barcodescanner-Objekt arbeiten, das du zunächst erst einmal erzeugen musst. Die Klasse `Barcodescanner` findest du in der Kategorie `Sensors` in der `Designer-Sicht`. Überlege dir als nächstes, wie du das Display deines Mobile Devices gestalten möchtest. Du benötigst vielleicht eine Schaltfläche mit der Aufschrift „Scannen“. Den decodierten Text kannst du dir in einem Label ausgeben lassen.

Wenn der Nutzer die Schaltfläche „Scannen“ anwählt, soll `call Barcodescanner.DoScan` ausgelöst werden. Damit wird die Kontrolle an das Betriebssystem abgegeben.

Erst nachdem der Barcode oder QR-Code erkannt wurde, wird der zum Barcodescanner-Objekt zugehörige Ereignisverarbeiter (vom Betriebssystem) aufgerufen. Dieser Ereignisverarbeiter heißt `AfterScan` und erwartet den decodierten Text als Eingabe. Entsprechend füllt das Betriebssystem den decodierten Text in den Parameter `result` ein.

Wie kommt man nun an den decodierten Text? Den Wert von `result` kannst du auslesen (`get result` findest du, indem du mit der Maus auf `result` gehst) und dann in einem Label am Display ausgeben.

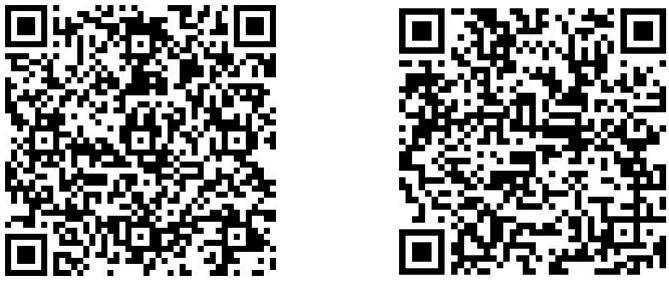
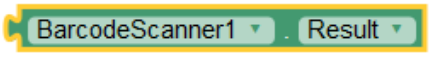


Abbildung 5.6: Teste dein Programm an den beiden QR-Codes. Lies erst den linken QR-Code ein.

Erweiterung: Sprechender Barcodescanner

Erzeuge ein Objekt der Klasse `TextToSpeech`. Setzte bei Programmstart `TextToSpeech.Country` auf `Germany` und `TextToSpeech.Language` auf `german`.
Dein erzeugtes Objekt bietet die Methode `Speak` an. Verwende diese Methode und erweitere dein Programm.

Erzeuge eine weitere Schaltfläche mit der Aufschrift „Vorlesen“. Innerhalb des zugehörigen Ereignisverarbeiters muss auf den eingescannten Text zugegriffen werden. Du kannst folgende Methode verwenden, die dein Barcodescanner-Objekt anbietet:



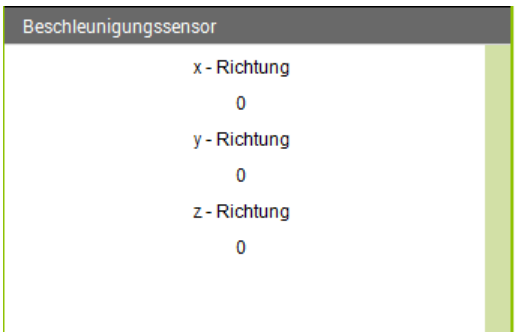
Sorge dafür, dass nur dann etwas vorgelesen wird, wenn `BarcodeScanner.Result` auch mindestens ein Textzeichen enthält. Die Methode `length` leistet hierfür gute Dienste. Sie steht immer zur Verfügung und findet sich deshalb in der `Blocks-Sicht` unter `Built In` und dort in der Kategorie `Text`.

5.3 Beschleunigungssensor-Anzeige (leicht)

Beschleunigungen können gemessen werden. Die Daten können beispielsweise zum Steuern eines Roboters oder zum Bewegen virtueller Objekte verwendet werden. Mit der Steuerung von Objekten werden wir uns in späteren Aufgaben auseinandersetzen. Bevor wir solche Steuerungsaufgaben angehen, werden wir uns zunächst die Daten, die der Beschleunigungssensor misst, anzeigen lassen.
Eine entsprechende kleine App soll nun gebaut werden.

GUI - Grafical User Interface: Anzeige gestalten

Erzeuge einige Label-Objekte. Wähle ein einfaches Layout. Du kannst dich an nachfolgender Abbildung orientieren:

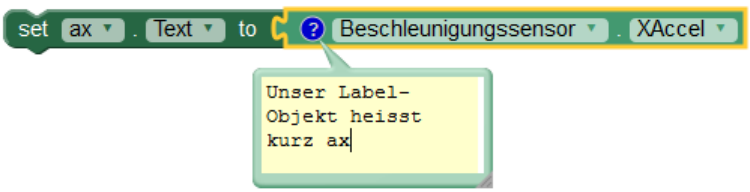


Endlose Wiederholungen

Ein Clock-Objekt kann in regelmäßigen Abständen Ereignisse feuern. Der zugehörige Ereignisverarbeiter des Clock-Objekts heißt `Timer`. Verwende diesen Ereignisverarbeiter, um deine Anzeige ständig zu aktualisieren. Via Attribut des Clock-Objekts kannst du einstellen, ob häufig oder weniger häufig aktualisiert werden soll.

Sensorwerte auslesen

Mit einem Beschleunigungssensor-Objekt hast du schon gearbeitet[2]. Nun interessieren uns die Messdaten selbst. Diese kann man über Methoden auslesen, die das Beschleunigungssensor-Objekt anbietet. Die ausgelesenen Daten können in passende `Label`-Objekte „eingefüllt“ werden.



Kippe dein Gerät nach links, nach rechts, nach vorne, ... Beschleunige dein Gerät vorsichtig nach oben, nach

[2] siehe „Würfelbecher schütteln“

unten, ... Welche Anzeigewerte erhältst bzw. erwartest du? Welche Beschleunigung stellt die x-Richtung bzw. y-Richtung bzw. z-Richtung dar? Wann sind die Werte positiv, wann negativ?

5.4 Reaktionsspiel (leicht)

Spielidee

Es gibt zwei Spieler. Wer am schnellsten reagiert, wenn der Bildschirm rot wird, bekommt einen Punkt. Wer auf andere Farben des Bildschirms reagiert, schenkt dem Spielpartner einen Punkt. Wer zuerst 10 Punkte erreicht, der hat gewonnen.

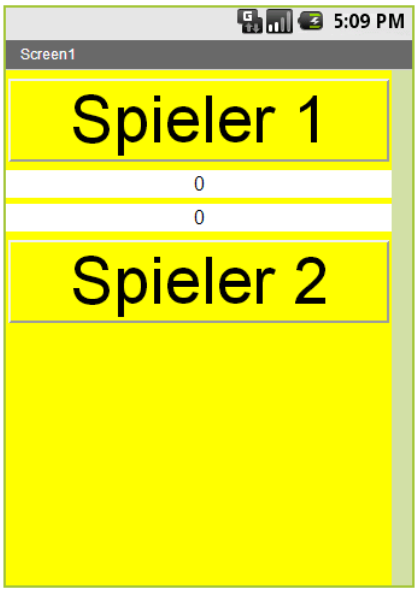


Abbildung 5.7: Grafische Benutzeroberfläche des Reaktionsspiels.

GUI - Das Spielfeld auf dem Mobile Device erstellen

Verwende zwei `Button`-Objekte, mit denen du jeweils Reaktionen (also hier `Button`-Clicks) der Spieler detektieren kannst. Wenn du keine Hintergrundfarbe setzt (`None`), dann sehen die Spieler durch den Button hindurch auf die Farbe des Bildschirms. Das ist für dieses Spiel praktisch.

Mit zwei `Label`-Objekten kannst du den momentanen Spielstand festhalten. Wir haben in der Abbildung für die `Label`-Objekte einen weißen Hintergrund gewählt und beide `Label`-Objekte mit der Zahl 0 initialisiert.

Ständiger Farbwechsel: Mit einem Clock-Objekt arbeiten

Ein `Clock`-Objekt kann in regelmäßigen Abständen Ereignisse feuern. Der zugehörige Ereignisverarbeiter des `Clock`-Objekts heißt `Timer`. Erstelle eine Liste aus einigen Farben. Wähle aus dieser Farbliste zufällig eine Bildschirm Hintergrundfarbe aus. Teste deinen `Timer` und probiere in der `Designer-Sicht` verschiedene Zeitintervalle aus.

Rot detektieren: Was muss sich ändern, wenn ein Spieler seinen Button drückt?

Wenn der Bildschirm rot ist, dann bekommt der Spieler selbst einen Punkt, ansonsten bekommt der Partner einen Punkt.

Anmerkung: Während einer „Rotphase“ kann man auch versuchen, mehr als einen Punkt zu sammeln. Dumm nur, wenn man nicht rechtzeitig aufhört, bevor die nächste Farbe erscheint - dann bekommt nämlich der Gegner den Punkt.

Teste deine bisherige Umsetzung: Spiele mit einem Partner ein paar Züge.

Punktestand analysieren: Hat jemand schon gewonnen?

Immer, wenn sich der Punktestand ändert, muss geprüft werden, ob der Spieler oder sein Partner jetzt gewonnen hat.

Wenn du diese Prüfung mit passender Reaktion in deinem Programm an mehreren Stellen vornehmen musst, ist es sinnvoll, eine eigene Methode zu schreiben.

Falls jemand gewonnen hat, kannst du das mit einem `Notifier`-Objekt gut mitteilen. Die Klasse `Notifier` findest du in der Kategorie `User Interface`. Wir haben die Methode `ShowMessageDialog` eines `Notifier`-Objekts verwendet. Vergiss nicht, den Punktestand zurückzusetzen, wenn eine neue Runde beginnt.

5.4.1 Alternative Punktevergabe (Optional, fortgeschritten)

Bis jetzt ist es möglich, dass beide Spieler pro „Rotphase“ mehrere Punkte bekommen können. Wir möchten nun erreichen, dass pro Rotphase nur noch ein Spieler einen Punkt bekommen kann. Gleichzeitig soll auch bei einer anderen Farbe nur noch ein Punkt an den anderen Spieler vergeben werden.
Dies kannst du z.B. durch eine eigene Variable und geeignete Abfragen erreichen, indem du überprüfst, ob während einer „Farbphase“ bereits ein Punkt vergeben worden ist.

Hinweis: Setze das Timer-Intervall deines `Clock`-Objekts auf einen höheren Wert, z.B. 1500 ms, um die alternative Punktevergabe gut testen zu können.

5.4.2 Gewinner verkünden (Optional, fortgeschritten)

Um nach Spielende den Gewinner nicht nur als Spieler 1 oder Spieler 2 verkünden zu können, kann man zu Beginn des Spiels die Namen der Spieler einlesen lassen. Die passende Gewinnmeldung kann dann auch noch mittels `TextToSpeech` vorgelesen werden.

Benutze zum Einlesen der Spielernamen zwei `Notifier`-Objekte und suche in der `Blocks-Sicht` nach der Methode `ShowTextDialog`, die alle `Notifier`-Objekte anbieten.

Hinweis: Falls du noch Probleme mit der Funktionsweise des `Notifier`-Objekts hast, kannst du bei der Aufgabe Würfelspiel-Plus auf Seite 40 mehr über das Verhalten der `Notifier`-Objekte erfahren.

5.5 Das Pong-Spiel (fortgeschritten)

Spielidee

Bei Pong bewegt sich ein Ball von Wand zu Wand. Der Spieler kann durch Neigen des Smartphones den Schläger beschleunigen und muss versuchen, den Ball nicht nach unten fallen zu lassen. Geschieht dies trotzdem, soll ein Ton ausgegeben werden:

Abbildung 5.8: So in etwa könnte dein Pong-Spiel aufgebaut sein.

Browser-Sicht

GUI

Bei diesem Spiel sind keine Schaltflächen oder Label notwendig, da der Schläger nur über den Beschleunigungssensor gesteuert werden soll.
Man benötigt ein `Canvas`-Objekt, also eine Art Leinwand. Auf dieser Leinwand können sich Schläger und Ball bewegen. Die Leinwand kann z.B. 260 Pixel hoch sein und die Breite deines Screens ausfüllen - setze dazu die passenden Attributwerte (`... Fill parent ...`). Wir haben das Spiel im Querformat fixiert. Dazu kannst du das Attribut `ScreenOrientation` deines Screen-Objekts geeignet setzen.

Zur Leinwand werden jetzt Ball und Schläger hinzugefügt. Der AppInventor macht es uns hier einfach, weil in der Palette `Drawing and Animation` bereits ein Ball vorgefertigt ist und wir für den Schläger ein `ImageSprite`-Objekt benutzen können. Als Schläger-Bild kannst du die Datei „Rechteck.png“ verwenden und den Schläger an das untere Ende des `Canvas`-Objektes ziehen. Für den Ball musst du eigentlich nur noch die Geschwindigkeit festlegen (wir haben mal „6“ verwendet) und den Start-Winkel („Heading“) zum Beispiel auf 60 ändern, damit der Ball nicht einfach nur von links nach rechts fliegt. Du kannst natürlich auch noch Farbe und Radius des Balls verändern.

Bei `Media` findest du noch die Klasse `Sound`. Erzeuge ein Sound-Objekt, für das du gleich eine Audio-Datei hochladen kannst (z.B. AlienCreak2.wav). Der Sound soll später abgespielt werden, wenn der Ball ins „Aus“ fliegt.

Steuerung

Zur Steuerung des Schlägers verwenden wir einen Beschleunigungssensor. Außerdem arbeiten wir mit einem `Clock`-Objekt. In regelmäßigen Abständen (wir haben den entsprechenden Attributwert auf 20 ms gesetzt) soll die Beschleunigung ausgelesen werden[3].

Editor-Sicht

Der Schläger

Der Schläger wird mit Hilfe der gemessenen Werte des Beschleunigungssensors gesteuert. Jetzt kommt Physik ins Spiel. Erstelle Variablen[4] `v` und `deltaT`. Setze den Startwert für `v` auf 0, da sich der Schläger zu Beginn des Spiels nicht bewegen soll. Und `deltaT` bekommt den Startwert 0.02 - diesen Wert haben wir im letzten Absatz festgelegt, 20 ms entspricht 0.02 s.

Jedes Mal, wenn der Timer feuert, muss jetzt der neue x-Wert des Schlägers berechnet werden. Hier hilft der Physiker gerne: das Ergebnis kann wie in Abbildung 5.9 aussehen.

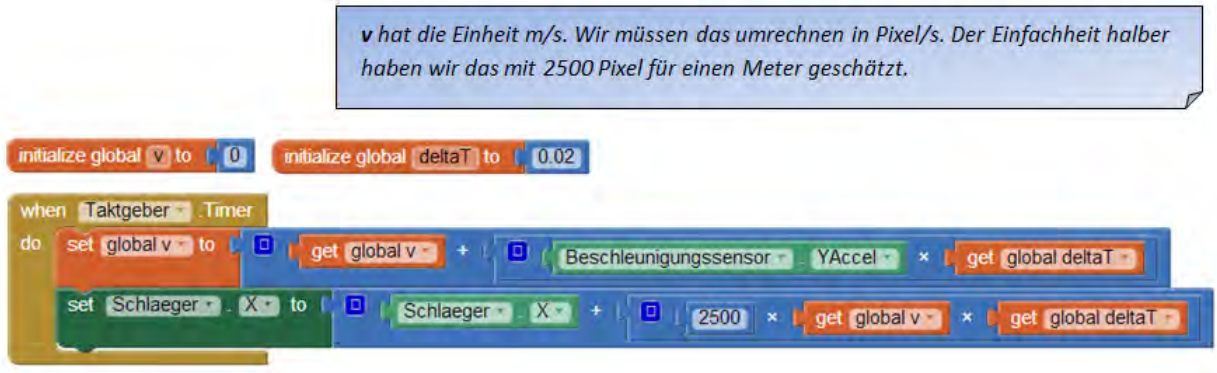


Abbildung 5.9: Festlegen der neuen x-Position des Schlägers alle 20 ms

Hinweis: Die meisten unserer Geräte arbeiten standardmäßig im Längsformat. Das Koordinatensystem der Sensoren bezieht sich immer auf die Standardorientierung. Aus diesem Grund müssen wir für die Änderung von `v` den Wert von `Beschleunigungssensor.YAccel` auslesen.
Das Koordinatensystem von Ball und Schläger wird „automatisch“ mitgedreht, wenn man von Längsformat auf Querformat wechselt. Wir arbeiten im Querformat(Landscape), weil wir dann mehr Platz für die Schlägerbewegung haben.

Weiter sollte noch sichergestellt werden, dass der Schläger nicht über den Rand des Displays hinausschießt, sondern anhält. Benutze dazu den Ereignisverarbeiter „`... EdgeReached`“ deines Schlägers. Setze bei Kollision mit dem Rand `v` auf 0 und schiebe den Schläger einfach um ein Pixel vom Rand weg, damit er dort nicht „kleben“bleibt[5].

[3] Anmerkung: 20 ms werden in keiner Weise garantiert, das ist hier aber auch nicht wichtig. Man kann messen, in welchen Abständen tatsächlich ausgelesen wird, darauf verzichten wir aber in dieser Aufgabe.
[4] Variablen erstellst du via `Built-In → Variables → initialize global name to`
[5] `EdgeReached` wird sonst ständig aufgerufen und hält den Schläger am Rand.

Der Ball

Der Ball soll von den Wänden abprallen. Mit dem Schläger musst du den Ball von der unteren Wand fernhalten.

Das Abprallen von den Wänden kannst du mit Hilfe des Ball-Ereignisverarbeiters „`... EdgeReached`“ umsetzen. Weiter gibt es bereits eine Methode, die einen Ball von einer Wand abprallen lässt („`Bounce`“). Vergiss nicht zu überprüfen, ob es sich bei der Wand um die untere Wand handelt. Wenn das der Fall ist, soll z.B. `AlienCreak2.wav` abgespielt werden.

Hinweis: Die Wände, bzw. Kanten („`edge`“) haben eine Nummer. Die untere Wand hat -1.

Jetzt musst du noch dafür sorgen, dass der Ball auch vom Schläger abprallen kann. Verwende den Ereignisverarbeiter `...CollidedWith`. Da es keine vordefinierte Methode der Art `...Bounce (other)` gibt, die den Ball von einem anderen Objekt abprallen lässt, musst du nun „nachbauen“, was via Methode „`...Bounce`“ für die Wände schon bereitstand. Überlege dir dafür, in welchem Winkel der Ball sich bewegen soll, wenn er vom Schläger abprallt.

Hinweis: Mache dir ggf. eine Skizze mit den Winkeln und denke an das Reflexionsgesetz aus der Physik: „*Einfallswinkel = Aufallswinkel*“.

Abschließend möchten wir noch sicherstellen, dass der Ball nur dann vom Schläger abprallt, wenn er von oben kommt. Dazu müssen wir zunächst die Werte von `Ball.Heading` beachten:

- bei `Ball.Heading = 0` schaut der Ball nach rechts
- bei `Ball.Heading = 90` schaut er nach oben
- bei `Ball.Heading = 180` schaut er nach links
- usw.

Betrachten wir Abbildung 5.10, stellen wir fest, dass `Ball.Heading` zwischen den Werten 180 und 360 liegt, wenn der Ball von oben auf den Schläger trifft:

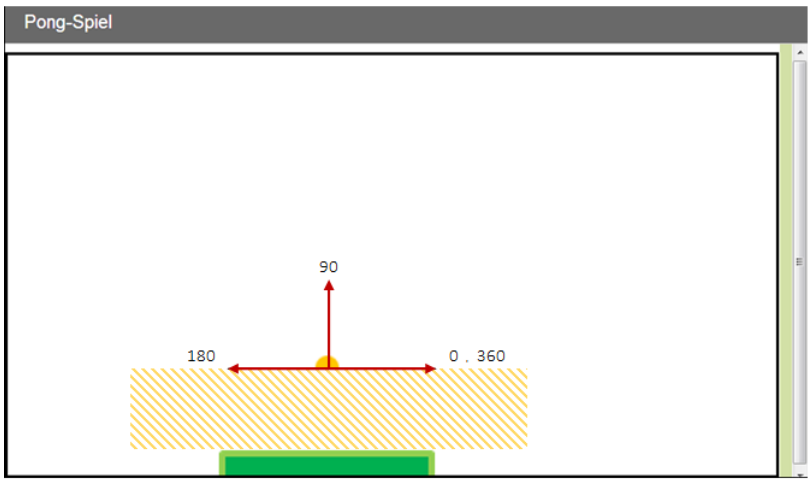


Abbildung 5.10: Werte von `Ball.Heading` liegen zwischen 180 und 360, wenn der Ball von oben auf den Schläger trifft.

Versuche nun die folgende bedingte Anweisung zu verstehen und füge anschließend deine Umsetzung des Ball-Abprallens in den „`then-do`-Teil ein:

Abbildung 5.11: Durch die bedingte Anweisung wird sichergestellt, dass der Ball nur dann vom Schläger abprallt, wenn er von oben auf ihn trifft.

Viel Spaß mit dem Pong-Spiel!

5.5.1 Das Pong-Spiel mit Zähler und Bestleistung (Optional, Fortgeschritten)

Erweitere dein Pong-Spiel um einen Zähler, der angibt, wie oft du den Ball gespielt hast, bevor er zum ersten Mal den Boden berührt. Gib in einem weiteren Feld den größten Zählerwert an, den du bisher erreicht hast.

Abbildung 5.12: Das Pong-Spiel mit Zähler und Bestleistung.

5.6 Bluetooth-Texter (fortgeschritten)

Spielidee

Es werden zwei Programme benötigt, mit deren Hilfe Text-Nachrichten von einem Telefon auf ein anderes Telefon übertragen werden können. Die Nachrichten werden über eine Bluetooth-Verbindung verschickt. Dabei übernimmt eines der beiden Programme immer die Rolle des Nachrichtenversenders, das andere die Rolle des Nachrichtenempfängers. Ein Rollentausch ist in der Grundversion dieser Aufgabe nicht vorgesehen.

Arbeite bei dieser Aufgabe mit einem Partner zusammen.

Das Programm des Empfängers (`Bluetooth-Server`) haben wir schon umgesetzt (`BluetoothServer.apk`). Beide Partner laden sich diesen `Bluetooth-Server` auf ihr Telefon und starten den Server.[6] Falls beim Starten des Servers eine Fehlermeldung zu Bluetooth-Admin Rechten kommt, lösche die BluetoothServer-App und installiere stattdessen `BluetoothServerAdmin.apk` auf deinem Mobile Device. Siehe hierzu Abschnitt 7.4 in Kapitel 7.

Vorbereiten der Telefone

Die nachfolgenden Dienste werden vom Betriebssystem bereitgestellt. Deshalb musst du die beschriebenen Einstellungen im Android-Betriebssystem deines Telefons vornehmen.

- `Bluetooth` einschalten
- Einer der Partner sorgt dafür, dass sein Gerät für kurze Zeit sichtbar ist, der andere „sucht nach Geräten“.
- Aus Sicherheitsgründen müssen beide Partner einander in eine Art Positivliste aufnehmen. Sie stimmen damit prinzipiell einem Datenaustausch mit dem jeweiligen Partner zu. Erst wenn dies geschehen ist, kann später von unserem selbst geschriebenen Programm ein Verbindungsaufbau angestoßen werden. Das Betriebssystem verwaltet diese Liste.

  Hinweis: Das Zustimmen (Pairing) ist so gelöst: Beide Partner akzeptieren (mündlich) eine Zahl (`passkey`) und bestätigen diese.

Anmerkung: Ein Telefon kann der Kommunikation mit mehreren Geräten zugestimmt haben. Diese Liste der Zustimmungen kann man sich anzeigen lassen (Geräte mit bestehendem Pairing).

Den Sender programmieren

Was erwartet man von einem einfachen Sender-Programm? Man muss...

(1) Verbinden

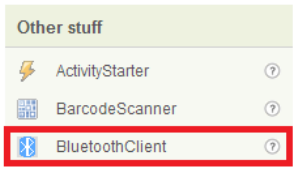
(2) Text schreiben

(3) geschriebenen Text senden

(4) Verbindung trennen

können.

[6] Falls du noch nicht weißt, wie man ein Programm auf den App-Inventor hochlädt bzw. eine fertige App auf dein Mobile Device herunterlädt, schau in den entsprechenden Abschnitten in Kapitel 7 nach.



Die technischen Details des Verbindungsaufbaus, der Nachrichtenübermittlung und des Verbindungsabbaus werden von einem Objekt der Klasse `BluetoothClient` gekapselt. Wir nennen es kurz `client`. Dieser `client` ist am Display des Telefons nicht zu sehen (*non-visible component*). Die Dienste dieses `clients` wirst du im Folgenden immer wieder nutzen.

5.6.1 GUI und Verbinden - Version 0

`Ohne Auswahlmöglichkeit (der Geräte mit bestehendem Pairing)`

Erzeuge entsprechend obiger Überlegungen einige GUI-Objekte. Wähle ein einfaches Layout und überlege dir, wann welche dieser Objekte deaktiviert bzw. aktiviert werden sollen.

Tipp: Ein Objekt, welches du im Skript eines Ereignisverarbeiters deaktiviert hast, sollte in einem anderen Skript wieder aktiviert werden.

Verbinden

Ermittle die Adresse (genauer: Bluetooth Medium Access Control-Adresse, kurz: Bluetooth MAC-Adresse) deines Partners (bei den meisten Geräten geht dies über Einstellungen → Über das Gerät → Status).
Wird die entsprechende Schaltfläche gedrückt, soll eine Verbindung zu dieser Adresse (deinem Partner) hergestellt werden.
Die MAC-Adresse könnte beispielsweise so aussehen:

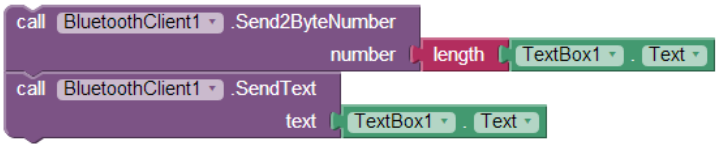
```
30:85:A9:4D:31
```

Text schreiben

In ein Textfeld (`Textbox`) kann Text eingetippt werden. Finde heraus, welche Methoden ein Textfeld anbietet. Für uns ist die Methode zum Auslesen von Text interessant. Die Methode zum Aktivieren / Deaktivieren eines Textfeldes hast du beim Umsetzen deiner GUI schon verwendet.

Geschriebenen Text senden

Wenn die Bluetooth-Verbindung erfolgreich aufgebaut wurde, dann kann gesendet werden. Das Protokoll, welches wir verwenden werden, verlangt, dass erst die Länge des zu sendenden Textes im Format `2 Byte` übertragen wird und dann der Text selbst. Entsprechende Methoden stellt der `client` bereit:



Trennen

Wenn der passende Button gedrückt wird, dann soll die bestehende Verbindung getrennt werden.

Teste dein Programm mit einem Partner und sende einige Nachrichten.

5.6.2 GUI und Verbinden - Version 1

`Mit Auswahlmöglichkeit (der Geräte mit bestehendem Pairing)`

Wie unterscheiden sich die beiden GUI-Klassen `Button` und `ListPicker`? Wähle zwei Unterschiede aus und erläutere diese jeweils in einem Satz.

Tipp: Wenn du bisher noch nicht oder schon lange nicht mehr mit einem `ListPicker`-Objekt gearbeitet, dann bearbeite zunächst die entsprechende Einführungsaufgabe in Abschnitt 4.2.7 aus Kapitel 4.

Dein `client` bietet eine Methode `AdressesAndNames` an. Diese Methode gibt eine Liste der Geräte zurück, denen du eine Kommunikation mit deinem Telefon erlaubt hast. Aus dieser Liste soll der Nutzer deines Programms nun auswählen können. Was soll passieren, nachdem der Nutzer ein Element aus der Liste per Berührung ausgewählt hat?

Teste mit deinem und mit einem anderen Partner deine überarbeitete Applikation. Vergiss nicht, den neuen Partner via Betriebssystem in deine „Positivliste“ aufzunehmen.

5.7 Robotersteuerung (fortgeschritten)

In diesem Abschnitt wollen wir eine Applikation erstellen, mit der wir einen Roboter (Lego Mindstorms NXT) mit unserem Mobile Device steuern können. Wir nutzen dazu eine Bluetooth-Verbindung zwischen Mobile Device und Roboter. Über diese Verbindung schicken wir Steuersignale und bauen somit eine Art Fernsteuerung.

Verbindung mit dem Roboter herstellen

Leider ist es nicht ganz einfach, das Mobile Device mit dem Roboter zu verbinden. Gehe schrittweise vor:

Vorbereiten der Geräte

(1) Überprüfe, ob Bluetooth am Roboter aktiviert ist und der Roboter sichtbar ist. Im Display muss links oben sowohl das Bluetooth-Symbol zu sehen sein als auch das Symbol für Sichtbarkeit, ein <.

(2) Auf dem Mobile Device musst du via Einstellungen Bluetooth aktivieren. Suche sodann nach Geräten und wähle aus der Liste der Geräte **deinen** Roboter aus und stimme prinzipiell einer Verbindung mit dem Roboter zu (Pairing). Das Zustimmen ist so gelöst: Beide Partner akzeptieren eine Zahl (`passkey`, z.B. 1234) und bestätigen diese.

Findet dein Mobile Device den Roboter nicht? Zurück zu Schritt (1).

Softwareeinstellungen

Mit unserer Steuerungs-App wollen wir nun eine Verbindung zwischen Mobile Device und Roboter herstellen und trennen können.

Du brauchst dazu auf deinem Gerät einen Bluetooth-Client und entsprechende Schaltflächen „Verbinden“ und „Trennen“, wobei durch Klicken auf „Verbinden“ eine Liste mit einer Auswahl an Bluetooth-Geräten anzeigt werden soll[7]. Welches Objekt bietet sich hierfür an?
Nach Auswahl eines Bluetooth-Partners soll versucht werden, eine Bluetooth-Verbindung aufzubauen. Dies erreichst du in einem geeigneten Ereignisverarbeiter mit folgendem Baustein:

Das Innere des `if`-Kontrollblocks musst du noch mit geeigneten Bausteinen füllen.

Durch Klicken auf Trennen soll eine bestehende Bluetooth-Verbindung beendet werden.

Achte noch darauf, dass deine Schaltflächen nur dann anklickbar sind, wenn dies auch wirklich Sinn macht!

Bekanntes Problem 1: Fehlermeldung der Art „Verbindung kann nicht hergestellt werden“. Es könnte sein, dass die App mitgeteilt bekommt, dass die Verbindung noch besteht, da kein Verbindungsabbau stattgefunden hat. Die App kann die Verbindung jedoch nicht mehr nutzen.

[7] Das Objekt `Bluetooth-Client` stellt eine Methode zur Verfügung, die alle Geräte zurückgibt, mit denen irgendwann einmal Pairing erfolgt ist. Diese müssen allerdings weder verfügbar, noch sichtbar sein.

Mögliche Abhilfe: Schalte Bluetooth am Roboter aus und wieder ein und versuche, via App die Verbindung erneut herzustellen. Teste ein paar Mal, ob du dich nun mit Hilfe deiner App erfolgreich mit dem Roboter verbinden kannst. Wenn es noch nicht klappt, dann wähle am Phone via `Einstellungen` → `Apps` → `alle` die Bluetooth-Freigabe aus, dort „Stoppen erzwingen“ und „Alle Daten löschen“. Teste erneut einige Male, ob du dich verbinden kannst.

Diesen Fehler kannst du vermeiden: Trenne die Verbindung zwischen Mobile Device und Roboter („Trennen“ in der App anklicken), bevor du via AppInventor Änderungen an der Applikation vornimmst.

Bekanntes Problem 2: Bluetooth-Verbindung kann nicht hergestellt werden und auf dem Roboter wird die Fehlermeldung `BT Storage full error` angezeigt.

Mögliche Abhilfe: Auf dem Roboter haben sich schon zu viele Geräte via Bluetooth angemeldet. Navigiere auf dem Roboter in das Bluetooth-Menü und lösche unter der Rubrik `My Contacts` einige Geräte.

Steuern des Roboters mit Tasten

Füge deiner Applikation weitere Schaltflächen hinzu, um den Roboter steuern zu können (vorwärts, rückwärts, rechts, ..., stoppen). Versuche vielleicht zuerst nur vorwärts und rückwärts zu fahren. Füge später auch Buttons zum Stoppen und Lenken hinzu.

Die Programmierumgebung AppInventor stellt Klassen bereit, die die Kommunikation mit dem Roboter erleichtern. Du findest diese Klassen in der `Designer-Sicht` in der Kategorie `Lego Mindstorms`. Erzeuge zunächst ein oder zwei `NXTDrive`-Objekte. Du kannst den Attributwert des Attributs `DriveMotors` entweder auf **CB** setzen (dann werden beide Ausgänge gleich angesteuert) oder nur auf **B** bzw. **C** (dann wird nur ein Ausgang angesteuert).

Wichtig ist, den Attributwert des Attributs `BluetoothClient` zu setzen, siehe Abbildung 5.13.

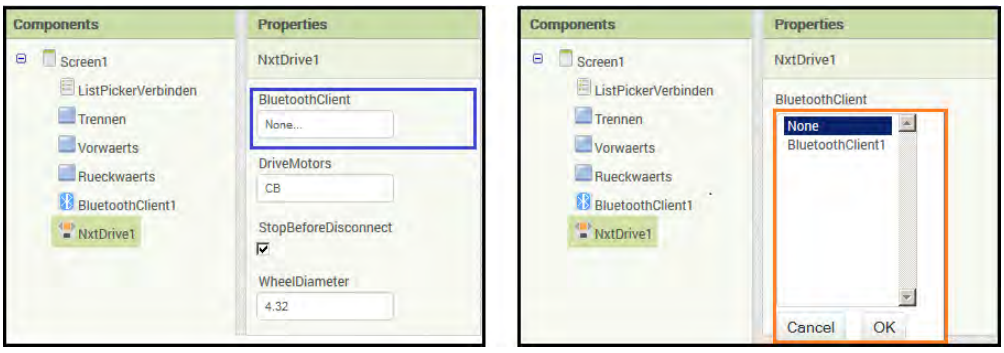


Abbildung 5.13: Der Attributwert des Attributs `BluetoothClient` des `NXTDrive`-Objekts muss auf den erstellen Bluetooth-Client gesetzt werden.

Nur so kann das `NXTDrive`-Objekt die Bluetooth-Verbindung nutzen, die das Bluetooth-Objekt bereitstellt.[8]

Hinweis: Setze das Attribut `power` der Motoren auf einen Wert 80 (bis 100). Bei 80 ist der Roboter nicht zu schnell und es gibt nicht das Problem, dass sich die Motoren nicht bewegen und nur ein leises Summen zu hören ist.

Um nach rechts und links fahren zu können, kannst du die folgenden beiden Bausteine verwenden:

`Clockwise` ist dabei der Uhrzeigersinn, `CounterClockwise` der Gegenuhrzeigersinn.

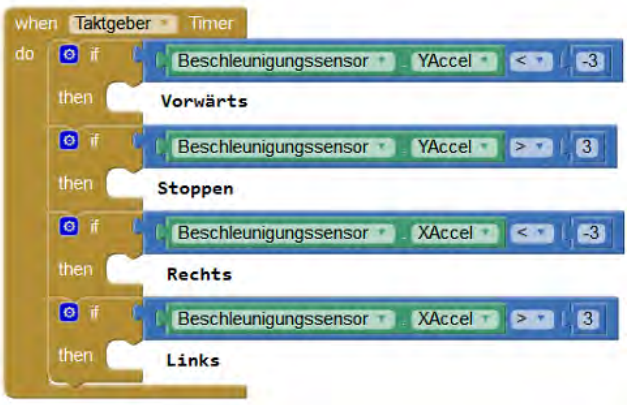
5.7.1 Steuern mittels Beschleunigungssensor (Optional, Fortgeschritten)

Du kannst dein Mobile Device auch benutzen, um den Roboter (wie von manchen Spielen bekannt) mittels Beschleunigungssensor zu steuern. Je nach Neigung des Smartphones soll sich der Roboter in die unterschiedlichen Richtungen bewegen. Kippen des Tablets oder Smartphones nach vorne bedeutet gerade fahren, neigen nach links oder rechts soll eine Drehung veranlassen.

Du benötigst sowohl einen Beschleunigungssensor als auch einen Timer. Der Timer sorgt dabei für eine angemessene Weiterleitung der Steuerungsbefehle via Bluetooth-Verbindung an den Roboter, typischerweise alle 150 ms. Setze das entsprechende Attribut deines Clock-Objekts via `Designer-Sicht`.

Um Kurven zu fahren haben wir mit zwei `NXTDrive`-Objekten umgesetzt, wobei wir dem entsprechenden Attribut jeweils nur einen Ausgang (B bzw. C) zugewiesen haben. Weiter haben wir die Methoden `MoveBackwardIndefinetely` und `Stop` verwendet.

Folgender Baustein-Block kann dir bei deinen Überlegungen hilfreich sein:



[8] Diese Einstellung muss später auch bei allen anderen Objekten vom Typ „Lego Mindstorms“ vorgenommen werden.

5.7.2 Weitere Funktionen einbauen (Optional, Schwer)

Der Roboter hat nicht nur Motoren, die angesteuert werden können. Man kann auch Sensorwerte auslesen. Benutze zum Beispiel den Ultraschallsensor des Roboters, um die Entfernung zu Gegenständen zu messen. Sobald der Abstand zu einem Gegenstand kurz genug ist, lasse den Roboter beispielsweise kurz zurückfahren und einen Ton abspielen.

Denkbar wäre auch, die beiden obigen Reaktionen durch eine Checkbox optional zu machen. Der Benutzer könnte somit selbst entscheiden, ob der Roboter zurückfahren bzw. einen Ton abspielen soll. Natürlich sind auch noch viele weitere Funktionen denkbar!!

Hinweis: Vergiss nicht, für jedes `NXT...`-Objekt den Attributwert des Attributs `BluetoothClient` zu setzen.

Überlege dir, welches Objekt du für das Abspielen einer Ton-Datei auf dem Roboter brauchen könntest. Finde heraus, welche Ton-Dateien auf dem Roboter gespeichert sind!

5.8 Datenspeicher (fortgeschritten)

Bisher gingen unsere Daten immer verloren, wenn wir die App geschlossen und wieder geöffnet haben - wie etwa die Highscore-Anzeige beim Pong-Spiel. Es wäre natürlich viel schöner, wenn die bisherige Bestleistung auch nach wiederholtem Beenden und Starten angezeigt werden würde, wie man das aus vielen anderen Spielen kennt.
Programme, bei denen gewisse Daten dauerhaft (persistent) gespeichert werden müssen, benötigen einen Datenspeicher. Bei dieser Aufgabe wirst du Schritt für Schritt einige wichtige Eigenschaften eines Datenspeichers kennenlernen.

Datenspeicher = Datenbank?

Wir verwenden hier bewusst den Begriff Datenspeicher anstelle von Datenbank. `TinyDB` heißt zwar übersetzt 'winzige Datenbank', hat aber in ihrer Funktionsweise relativ wenig mit einer Datenbank zu tun. Die Bezeichnung 'Datenbank' beruht wahrscheinlich auf der persistenten Speicherung von Daten. Die Umsetzung erinnert allerdings mehr an eine Hash-Map. Um die Schüler im Bezug auf die Datenbank-Thematik in der 9. Jahrgangsstufe nicht zu verwirren, haben wir den Begriff Datenbank nicht explizit erwähnt.

GUI - Ein Formular erstellen

Wir möchten eine Applikation erstellen, bei der man den Namen, das Geburtsdatum und die E-Mail-Adresse von Freunden abspeichern und bei Bedarf anzeigen kann. Erstelle dazu geeignete Label- und TextBox-Objekte! Dein Kontaktformular könnte ungefähr so aussehen:

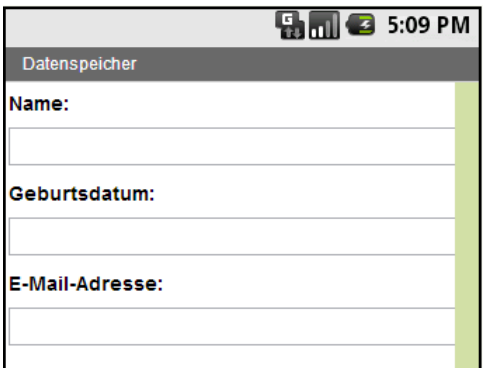


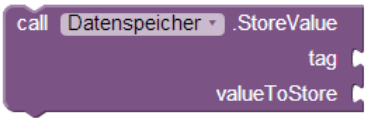
Abbildung 5.14: Kontaktformular, um die Daten von Freunden speichern zu können.

Um die eingegebenen Daten abspeichern zu können, benötigst du noch ein Objekt der Klasse `TinyDB` (Kategorie `Storage`) - wir nennen dieses Objekt *Datenspeicher*. Zudem benötigen wir noch eine Schaltfläche mit der Beschriftung *Daten speichern*.

Daten abspeichern

Wir wollen die in die entsprechenden TextBox-Objekte eingegebenen Daten bei Klicken auf die Schaltfläche in unserem Datenspeicher ablegen. Dazu benötigst du an geeigneter Stelle folgenden Baustein des Datenbank-

Objekts:



Dieser Baustein hat die zwei Dock-Stellen `tag` und `valueToStore`. Überlege dir, welche Bedeutung die beiden Dock-Stellen haben könnten. Wie können abgespeicherte Daten wiedergefunden werden?

Stark vereinfacht können wir unseren Datenspeicher mit einer Lagerhalle vergleichen, in der Tonnen abgestellt werden. Möchten wir einen neuen Datensatz in der Halle ablegen, packen wir die Werte an der Dock-Station `valueToStore` in eine Tonne. Um diese Tonne anschließend von den anderen Tonnen unterscheiden zu können, wenn wir die Daten wiederfinden wollen, beschriften wir unsere neue Tonne mit dem Wert der Dock-Station `tag`.

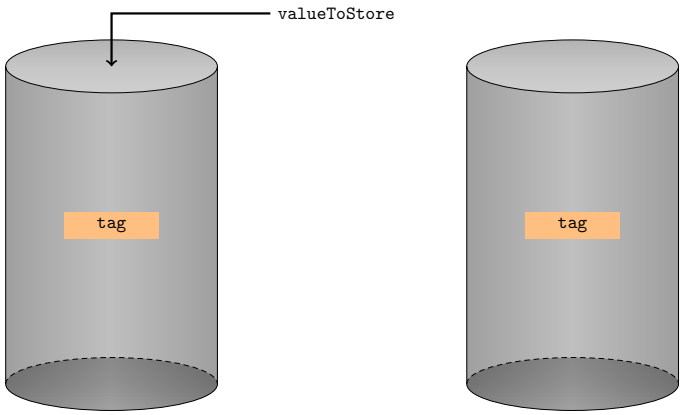


Abbildung 5.15: Vereinfachte Darstellung der Funktionsweise unseres Datenspeichers.

Suche nach einem geeigneten Baustein, mit dem du die drei Daten Name, Geburtsdatum und E-Mail-Adresse zusammen an die Dock-Stelle `valueToStore` anhängen kannst - verwende **nicht** den Baustein `makeText`.
Zur Identifizierung benutzen wir den Namen. In unserer Lagerhallen-Analogie lagern wir also einen Tonne mit dem Inhalt Name, Geburtsdatum und E-Mail-Adresse und mit der Aufschrift Name in die Halle ein.

Speichere nun einen Eintrag in deinem Datenspeicher! Wähle hierzu als Name Christian! Das Geburtsdatum und die E-Mail-Adresse darfst du frei wählen.

Daten auslesen

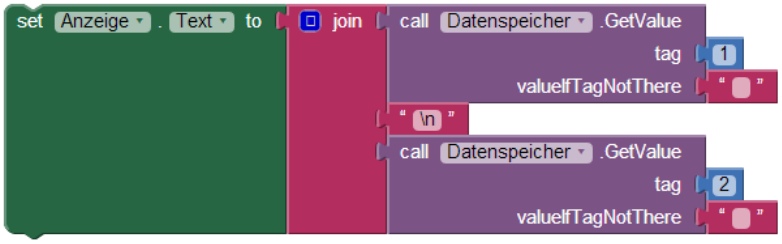
Erstelle nun eine neue Schaltfläche *Daten anzeigen*, mit der man den gespeicherten Eintrag in einem Label anzeigen kann.
Teste deinen Datenspeicher, indem du festlegst, dass bei Klicken auf die Daten anzeigen Schaltfläche die Daten des Tags `Christian` ausgegeben werden.

Speichere einen weiteren Eintrag mit Namen Christian und anderen Werten für das Geburtsdatum und die E-Mail-Adresse. Was passiert, wenn du erneut auf die Daten anzeigen Schaltfläche klickst?

Funktionsweise des Datenspeichers

Der Datenspeicher kann zu einem `tag` immer nur einen Datensatz speichern. Die Daten von Christian, den du zuerst eingespeichert hast, gehen verloren.
Um dieses Problem zu vermeiden, nummerieren wir die Einträge fortlaufend durch! Erstelle dazu eine geeignete Variable `idNummer` und ändere deine App entsprechend ab.

Erstelle zwei neue Einträge und gib diese aus. Das schaffst du erst mal über folgende Bausteine, falls du mit der Nummerierung bei 1 begonnen hast:

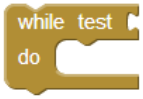


Das `\n` erzeugt einen Zeilenumbruch und somit eine schönere Darstellung der Einträge.

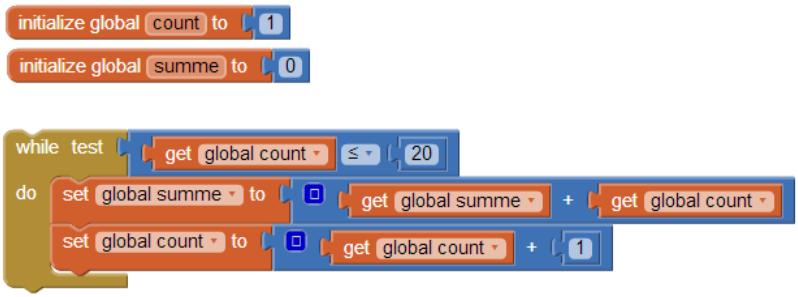
Alle Daten auslesen (schwer)

Obige Form der Ausgabe der Daten ist problematisch. Wir müssten unsere Applikation immer ändern, falls wir einen neuen Eintrag hinzufügen. Es wäre natürlich deutlich schöner, die Anzeige flexibel zu der Anzahl der Einträge zu halten.
Mit der Variable `idNummer` kannst du herausfinden, wie viele Einträge momentan in deinem Datenspeicher vorhanden sind. Um alle Daten auszugeben, musst du nun alle `tags` von 1 bis du der Anzahl der Einträge durchlaufen und ausgeben.

Dieses Problem kannst du mithilfe des Bausteins `while` lösen:



`while` führt das innere seines Bausteins solange aus, wie die Bedingung `test` an der Dock-Stelle erfüllt ist. Folgendes Beispiel zeigt dir noch einmal die Funktionsweise des `while`-Bausteins:



Solange der Wert von `count` kleiner gleich 20 ist, wird zum Wert von `summe` der Wert von `count` addiert und `count` anschließend um 1 erhöht. Folglich werden innerhalb des Bausteins die Zahlen 1 bis 20 addiert und in der Variable `summe` abgespeichert.

Überlege dir nun, wie du mithilfe eines `while`-Blocks die Ausgabe aller Daten umsetzen kannst.

Natürlich muss nun die Variable `idnummer` auch dauerhaft gespeichert werden, damit die App weiß, wie viele Einträge sich bereits im Datenspeicher finden. Du kannst die Variable etwa unter dem `tag` id speichern. Überlege dir, an welcher Stelle du die `idnummer` abspeichern und wann du den gespeicherten Wert wieder auslesen musst!

Abbildung 5.16: So könnte deine App ausschauen, wenn du auf die Daten anzeigen Schaltfläche klickst.

5.9 Standortanzeiger (fortgeschritten)

Die Aufgabe Standortanzeiger ähnelt in vielen Punkten der nachfolgenden Aufgabe Wegweiser, ist von der Schwierigkeit aber (deutlich) leichter einzustufen. Für viele Schülerinnen und Schüler kann diese Aufgabe hilfreich für die Bearbeitung der Wegweiser-Aufgabe sein. Tüftler und Cracks kann man natürlich auch gleich auf die Wegweiser-Aufgabe loslassen!

In dieser Aufgabe arbeitest du zum ersten Mal mit GPS-Daten und Google Maps. Dabei geht es darum, dich mit den hierfür benötigten Objekten vertraut zu machen und dir im Folgenden die Bearbeitung der Wegweiser-Aufgabe zu erleichtern.
Du sollst dir deine aktuelle Position als Breiten- und Längengrad ausgeben und ggf. auf der Karte anzeigen lassen.

GUI - Anzeige gestalten

Für diese Anzeige brauchst du zwei Schaltflächen und einige Label-Objekte[9]. Für die Anordnung der Label-Objekte sind `HorizontalArrangement`-Objekte hilfreich.
Deine Anzeige könnte ungefähr wie in Abbildung 5.17 dargestellt aussehen.

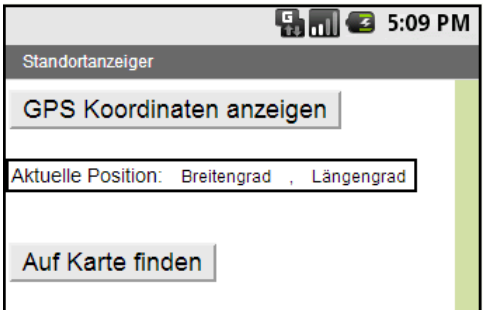


Abbildung 5.17: Mögliches Layout deiner Standortanzeiger-Applikation.

Nicht-Sichtbare Objekte

Für den Standortanzeiger benötigst du auch noch nicht-sichtbare Objekte - einen LocationSensor, um die aktuellen Standortkoordinaten auslesen zu können und einen ActivityStarter, um den Standort mit Google Maps anzeigen zu können.
Um mit dem ActivityStarter Google Maps aufrufen zu können, musst du in den Eigenschaften des ActivityStarters noch folgende Einstellungen vornehmen:

- Action: `android.intent.action.VIEW`
- ActivityClass: `com.google.android.maps.MapsActivity`
- ActivityPackage: `com.google.android.apps.maps`

[9] Erstelle je ein eigenes Label für Breiten- und Längengrad.

GPS-Koordinaten anzeigen

Nach Klick auf die Schaltfläche *GPS Koordinaten anzeigen*, soll der Breiten- und Längengrad deiner aktuellen Position in den entsprechenden Label-Objekten angezeigt werden. Du kannst ggf. noch geeignet abfragen, ob bereits ein Breiten- bzw. Längengrad ermittelt wurde, bevor du die Label-Objekte aktualisierst.

Es kann ein bisschen dauern, bis der LocationSensor das erste Mal Daten liefert!

Hinweis: Der LocationSensor gibt den aktuellen Längen- und Breitengrad mit einem Komma aus. Für Google Maps benötigen wir anstatt dem Komma einen Punkt. Dies schaffst du mit dem Baustein `replace all` in der Kategorie `Text`. Dieser ersetzt in einem gegebenen Text ein Segment (`segment`) durch ein anderes (`replacement`).

Auf Karte finden

Nach Klick auf die Schaltfläche *Auf Karte finden*, soll der aktuelle Standort auf Google Maps angezeigt werden. Dazu brauchst du die Bausteine `ActivityStarter.DataUri` und `ActivityStarter.StartActivity`.

Die in Abbildung 5.18 dargestellten Bausteine führen dazu, dass der Standort des Eiffelturms mit Google Maps angezeigt wird und können dir als Hilfe dienen.

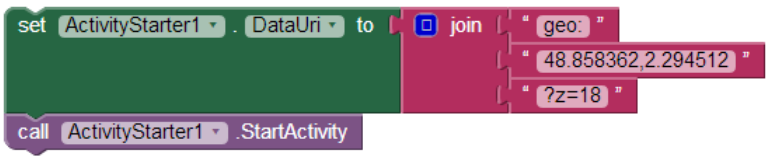


Abbildung 5.18: Mit diesen Bausteinen wird der Standort des Eiffelturms mit Google Maps angezeigt.

Finde heraus, was das Textsegment `?z=18` leistet, indem du mit dem Zahlenwert experimentierst.

5.10 Wegweiser (schwer)

In dieser Aufgabe sollst du dir deinen eigenen Wegweiser erstellen. Ziel ist es dabei, dass dir der Weg von deinem aktuellen Standort zu einer gespeicherten Adresse angezeigt werden soll:

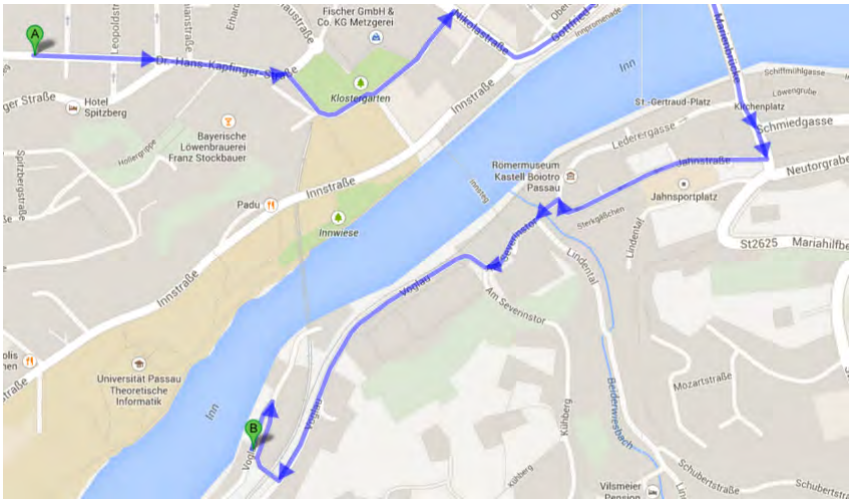


Abbildung 5.19: Route von A nach B dargestellt mit GOOGLE MAPS.

GUI - Anzeige gestalten

Für unsere Aufgabe brauchen wir einige Label-Objekte für den aktuellen Standort und den gespeicherten Standort. Zudem benötigen wir noch zwei Schaltflächen. Für die Anordnung der Label-Objekte sind `HorizontalArrangement`-Objekte hilfreich.
Deine Anzeige könnte ungefähr wie in Abbildung 5.20 dargestellt aussehen.

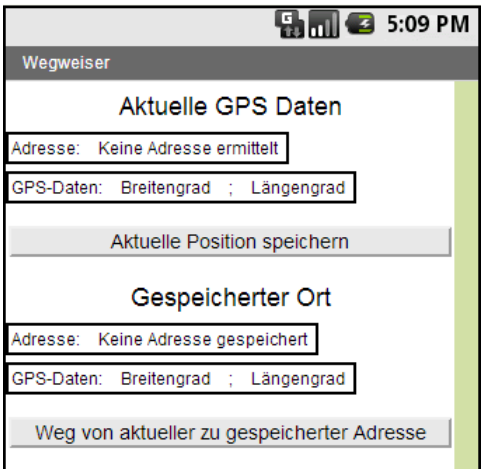


Abbildung 5.20: Mögliches Layout deiner Wegweiser-Applikation.

Gib deinen erstellten Objekten geeignete Namen, um nachher in der `Blocks-Sicht` die Übersicht zu behalten.

Nicht-Sichtbare Objekte

Für den Wegweiser brauchst zu auch noch nicht-sichtbare Objekte - einen Datenspeicher, mit dem du einen gewünschten Standort abspeichern kannst, einen LocationSensor, um die aktuellen Standortkoordinaten ausle-

sen zu können und einen ActivityStarter, um die Anfrage an Google Maps weitergeben zu können.
Um mit dem ActivityStarter Google Maps aufrufen zu können, mussst du in den Eigenschaften des ActivityStarters noch folgende Einstellungen vornehmen:

- Action: `android.intent.action.VIEW`
- ActivityClass: `com.google.android.maps.MapsActivity`
- ActivityPackage: `com.google.android.apps.maps`

Aktuelle Standortdaten auslesen

Zunächst sollen die aktuellen Standortdaten ausgelesen werden. Immer wenn sich der aktuelle Standort verändert, sollen die neuen Werte in die dafür vorgesehenen Felder geschrieben werden.

Hinweis: Der LocationSensor gibt den aktuellen Längen- und Breitengrad mit einem Komma aus. Für Google Maps benötigen wir anstatt dem Komma einen Punkt. Dies schaffst du mit dem Baustein `replace all` in der Kategorie `Text`. Dieser ersetzt in einem gegebenen Text ein Segment (`segment`) durch ein anderes (`replacement`).

Aktuellen Standort speichern

Durch Klicken auf die Schaltfläche 'Aktuelle Position speichern' sollen die aktuellen Standortdaten in die entsprechenden Felder von 'Gespeicherter Ort' geschrieben werden. Zudem sollen sie im Datenspeicher abgelegt werden, um auch bei Neustart der Applikation wieder zur Verfügung zu stehen.
Da immer nur die Daten eines Standortes gespeichert werden, bietet es sich an, die drei Daten unter dem jeweiligen Tag `adresse`, `breitengrad` bzw. `laengengrad` in den Datenspeicher zu legen.

Weg mit Hilfe von Google Maps anzeigen

Durch Klicken auf die Schaltfläche 'Weg von aktueller zu gespeicherter Adresse' soll dieser mit Hilfe von Google Maps angezeigt werden. Du hast hierfür bereits im `Browser Editor` entsprechende Einstellungen im ActivityStarter vorgenommen.
In der `Blocks-Sicht` sind zudem noch die Bausteine `ActivityStarter.DataUri` und `ActivityStarter.StartActivity` notwendig. Wir haben dir hierzu einen Ausschnitt aus unserem Programm abgebildet[10]:

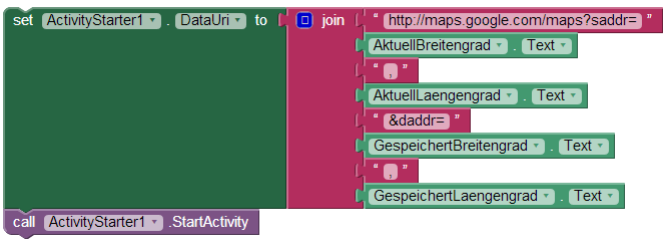


Abbildung 5.21: Verwenden von Google Maps, um den Weg vom aktuellen zum gespeicherten Standort anzeigen zu lassen.

[10] Beachte, dass Google Maps zuerst den Breitengrad und dann den Längengrad erwartet.

Gespeicherte Position bei Starten der Applikation auslesen

Haben wir in unserem Datenspeicher eine Position gespeichert, möchten wir diese bei Starten unserer Applikation in den entsprechenden Label-Objekten anzeigen. Überprüfe dazu in einem geeigneten Ereignisverarbeiter[11], ob im Datenspeicher unter dem Tag `adresse` ein Eintrag abgelegt ist oder nicht - dies kannst du elegant über den Baustein `length` der Kategorie `Text` erreichen.
Falls ein Eintrag vorhanden ist, sollen Adresse, Breitengrad und Längengrad in den Label-Objekten von 'Gespeicherter Ort' angezeigt werden.

Diese Aufgabe ähnelt sehr stark dem Tutorial „Android, Where's My Car“ der App Inventor Seite, wurde aber unabhängig davon erstellt.

[11] Welcher Ereignisverarbeiter wird immer bei App-Start aufgerufen?

Kapitel 6

Lösungsvorschläge zu den Aufgaben

Überblick:

Hier bieten wir Ihnen ausführliche Lösungsvorschläge zu den Aufgaben aus Kapitel 5.
Es handelt sich hierbei nur um Lösungsvorschläge, die Ihnen und den Schülern bei Schwierigkeiten und Problemen als Anregung dienen sollen. Natürlich sind bei allen Aufgaben auch andere Lösungen denkbar.

Die Lösungsvorschläge zu den Aufgaben gibt es auch als AppInventor-Dateien. Die `.aia`-Dateien kann man in der `Designer-Sicht` unter `Projects` → `Import project ...` hochladen. Die Lösungsvorschläge werden dann importiert.

6.1 Würfelbecher-App

Wie in der Aufgabenstellung beschrieben erzeugen wir zunächst in der `Designer-Sicht` zwei Label-Objekte *Ziffer1* und *Ziffer2*. Um auf Schütteln reagieren zu können, erzeugen wir gleich noch ein Objekt *Beschleunigungssensor* der Klasse `AccelerometerSensor`.
Wir ändern die horizontale Ausrichtung des Objekts *Screen1* auf `Center` und passen Schriftgröße (100.0) und Font der Label-Objekte entsprechend dem Beispiel-Bild aus der Aufgabenstellung an. Zudem initialisieren wir beide Label-Objekte mit den Ziffern 1:

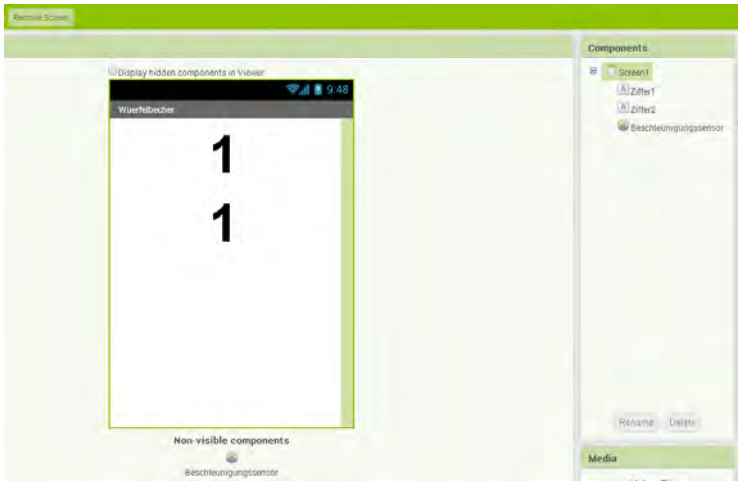


Abbildung 6.1: Erzeugen und Initialisieren der Label-Objekte in der `Designer-Sicht`.

Als nächstes wechseln wir in der `Blocks-Sicht` und setzen den Inhalt der Label-Objekte *Ziffer1* und *Ziffer2* während dem Schütteln auf beliebige Zahlen zwischen 1 und 6:

Abbildung 6.2: Ändern des Inhalts der Label-Objekte während des Schüttelns auf einen zufälligen Wert zwischen 1 und 6.

Als nächstes erzeugen wir ein Objekt der Klasse `Button` und nennen es *Wuerfelbecher*. Für das Attribut `Text` wählen wir „Ergebnis anzeigen“.
In der `Blocks-Sicht` passen wir die Ereignisverarbeitung und die Sichtbarkeitseinstellungen gemäß der Aufgabenstellung an:

Abbildung 6.3: Anpassen der Ereignisverarbeitung und der Sichtbarkeitseinstellungen.

Um Bild und Ton hinzuzufügen, laden wir zunächst die drei bereitgestellten Dateien hoch und weisen dem Button-Objekt `Wuerfelbecher` das Bild Becher.png zu, den Text „Ergebnis anzeigen“ löschen wir.
Wir erstellen noch zwei Sound-Objekte `Wuerfeln` und `Applaus` und weisen ihnen die entsprechende Sound-Datei zu.

Abbildung 6.4: Erstellen von Sound-Objekten, um in der `Blocks-Sicht` Zugriff auf die Sound-Dateien zu haben.

In der `Blocks-Sicht` müssen wir nur kleine Änderungen vornehmen:

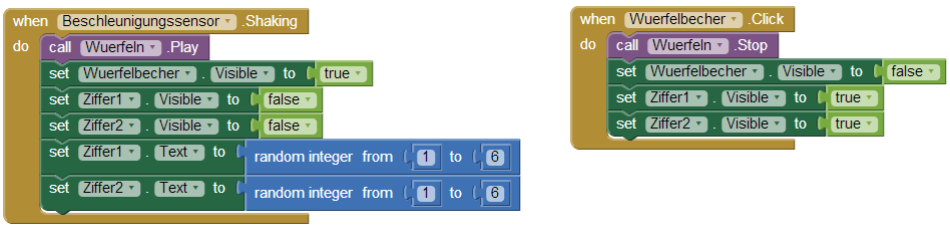


Abbildung 6.5: Abspielen und Pausieren der Sound-Datei Würfeln mit Hilfe des Objekts `Wuerfeln`.

Um uns das Würfel-Ergebnis vorlesen lassen zu können, erstellen wir, wie in der Aufgabenstellung beschrieben, ein Objekt der Klasse `TextToSpeech` mit dem Namen *Vorlesen* und erweitern unsere Ereignisverarbeitung in der `Blocks-Sicht`:

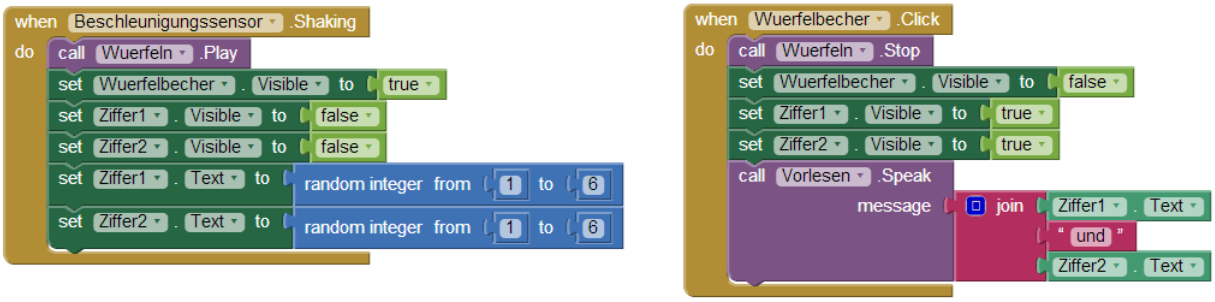


Abbildung 6.6: Mit dem Objekt `Vorlesen` der Klasse `TextToSpeech` können wir uns das Würfel-Ergebnis vorlesen lassen.

Für den Pasch-Zähler fügen wir ein neues Label-Objekt mit dem Namen *Zaehler* in der `Designer-Sicht` hinzu und initialisieren es mit dem Wert 0. Nun müssen wir nur noch im Falle, dass ein Pasch gewürfelt wurde, die Sound-Datei *Applaus* abspielen und den Wert des Objekts `Zaehler` um eins erhöhen. Wir könnten unsere App auch noch dahingehend abändern, dass bei einem Pasch nicht „3 und 3“ vogelesen wird, sondern „3-er Pasch“:

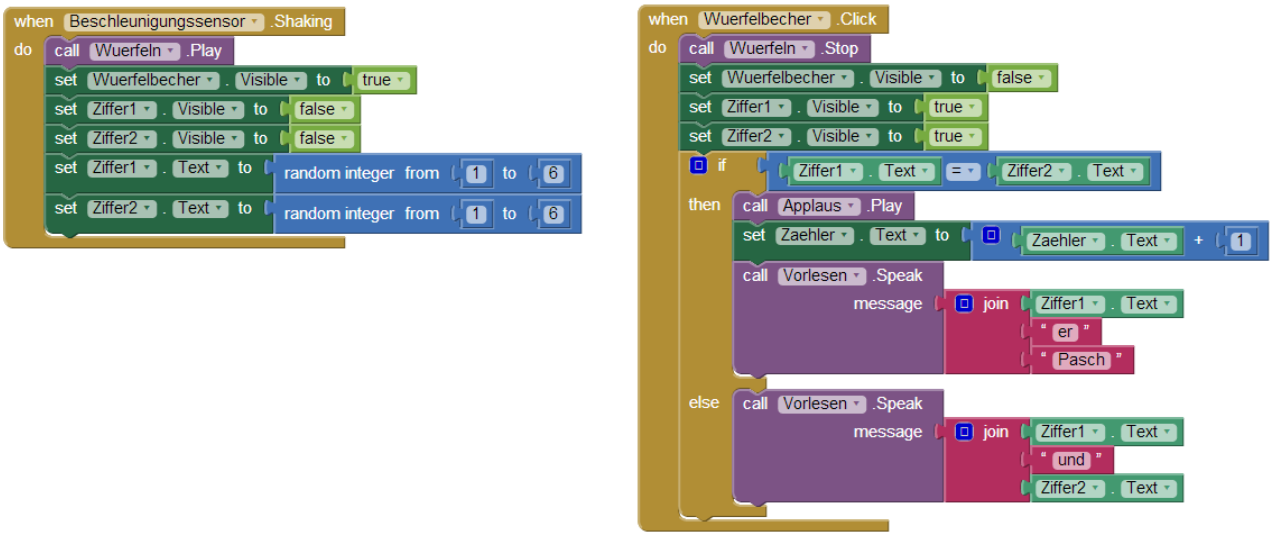


Abbildung 6.7: Mit der Kontrollstruktur `if-then` kann man auf einen Pasch gesondert reagieren.

6.1.1 Würfelspiel (Optional)

Zunächst fügen wir entsprechend der Aufgabenstellung die Label-Objekte `Spieler1, Spieler2, Ergebnis1` und `Ergebnis2` in einem Objekt der Klasse `TableArrangement` hinzu, um die Punktetafel anzeigen zu können:

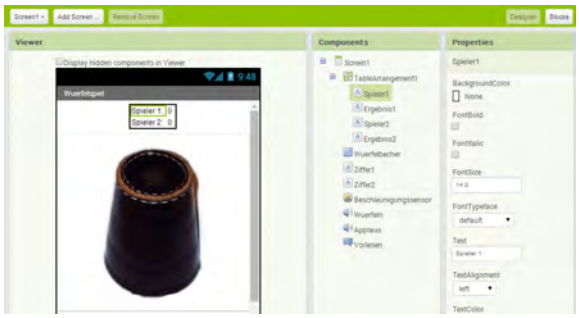


Abbildung 6.8: Hinzufügen von vier Label-Objekten für die Punktetafel.

Wir setzen initial die Hintergrundfarbe des Objekts `Spieler1` auf gelb. Anhand der Hintergrundfarbe unterscheiden wir, welcher Spieler aktuell an der Reihe ist. Ist Spieler 1 an der Reihe, verändern wir beim Schütteln nur den Wert von `Ziffer1` und zeigen anschließend beim „Aufdecken“ nur den Wert von `Ziffer1` an. Ist Spieler 2 an der Reihe, verändern wir beim Schütteln nur den Wert von `Ziffer2` und zeigen anschließend beide Würfelergebnisse an.
Der Spieler mit der größeren Zahl bekommt einen Punkt, d.h. wir erhöhen den Wert des entsprechenden `Ergebnis`-Objekts um eins:

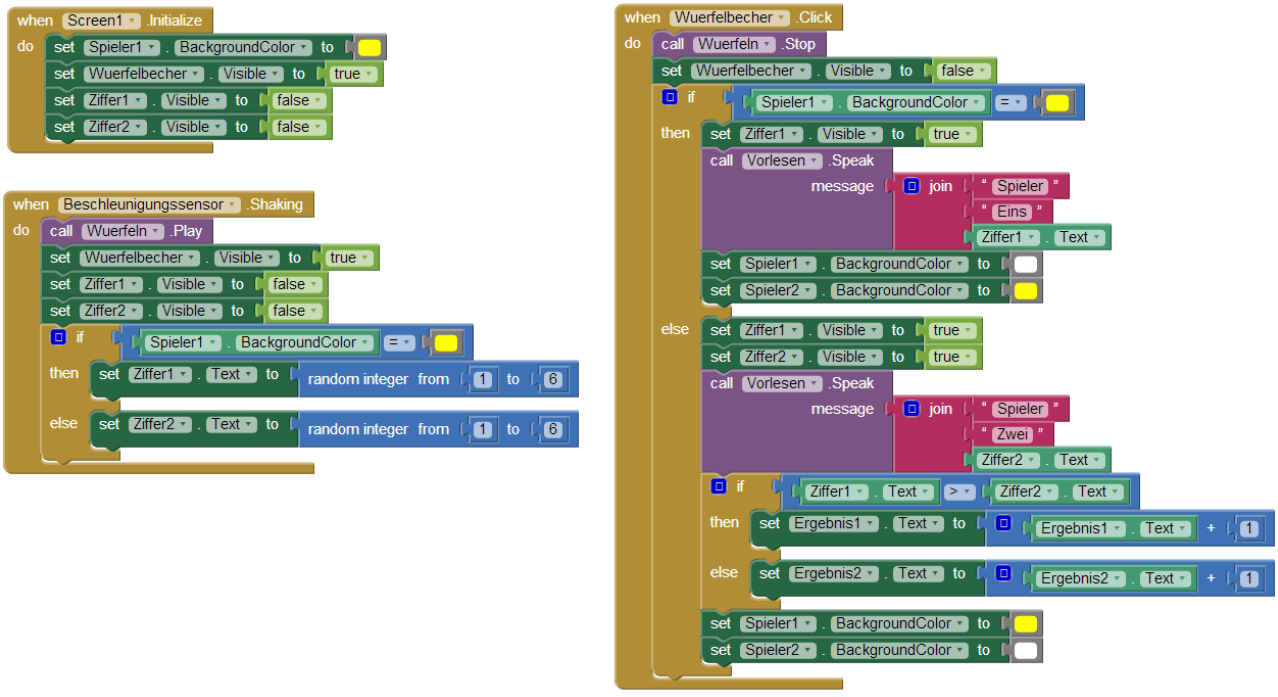


Abbildung 6.9: Das fertige Würfelspiel.

6.1.2 Würfelspiel-Plus (Optional)

Wir definieren uns zunächst in der `Blocks-Sicht` eine Variable mit dem Namen `bonus` und initialisieren sie mit dem Wert 1:

Abbildung 6.10: Definieren und Initialisieren einer Variable mit dem Namen `bonus`.

Als nächstes kümmern wir uns um die Spielanleitung und Namenseingabe. Dazu benötigen wir wie in der Aufgabenstellung angegeben zwei Objekte der Klasse `Notifier` - wir nennen sie *Anleitung1* und *Anleitung2*.

Die Anleitung mit Namenseingabe für Spieler 1 soll nach Aufrufen der App erscheinen. Aus diesem Grund packen wir diesen Baustein in die Ereignisverarbeitung `Screen1.Initialize`:

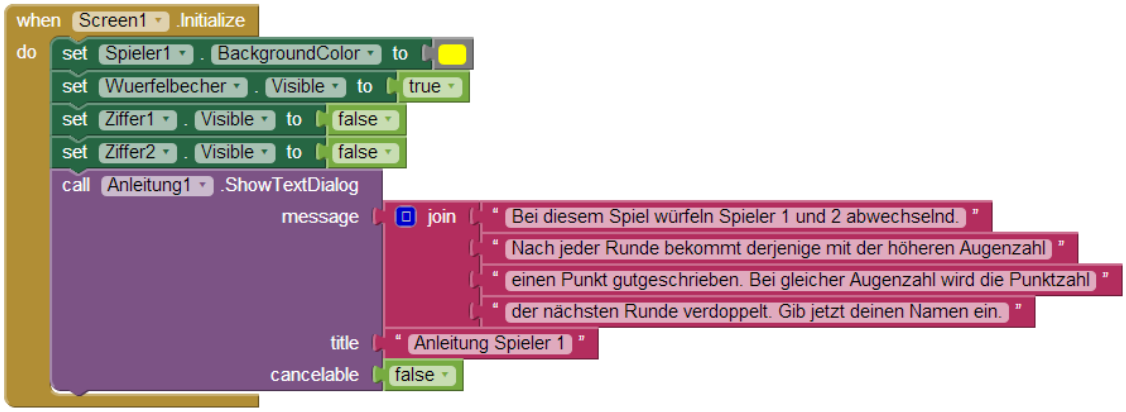


Abbildung 6.11: Anleitung für Spieler 1 soll gleich nach Starten der Applikation angezeigt werden.

Die Anleitung mit Namenseingabe für Spieler 2 soll direkt nach der erfolgreichen Namenseingabe für Spieler 1 erscheinen. Aus diesem Grund packen wir diesen Baustein in die Ereignisverarbeiutng `Anleitung1.AfterTextInput`. Gleichzeitig setzen wir hier noch den Wert des Objekts `Spieler1` auf den eingegebenen Namen:

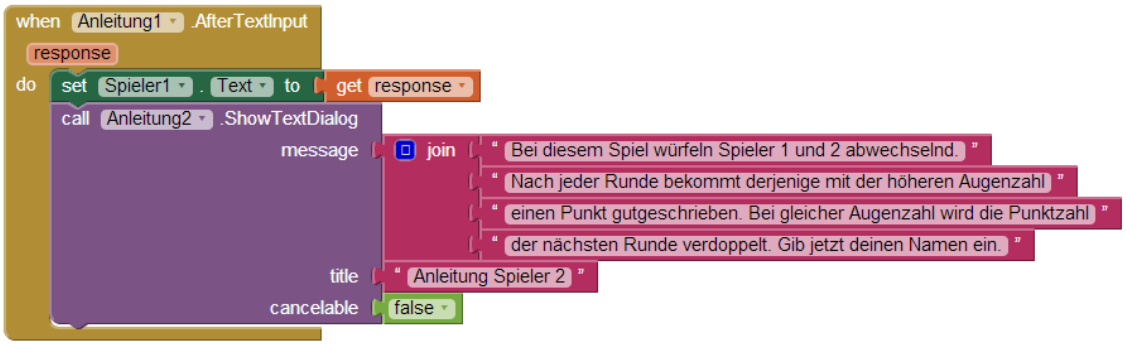


Abbildung 6.12: Anleitung für Spieler 2 soll im Anschluss an die Anleitung von Spieler 1 angezeigt werden.

Den Baustein `join` haben wir nur der Übersichtlichkeit wegen benutzt - man könnte den gesamten Text auch in einen einzigen `text`-Baustein packen.
Wir müssen auch noch den Wert des Objekts `Spieler2` neu setzen:

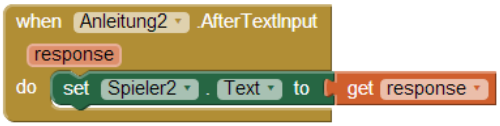


Abbildung 6.13: Setzen des Wertes des Objekts `Spieler2` auf den eingegeben Namen.

Als letztes müssen wir unsere Punktevergabe noch dahingehend anpassen, dass wir im Fall von gleichen Würfelergebnissen in der nächsten Runde den doppelten Wert vergeben. Wir haben uns dazu bereits eine Variable `bonus` definiert und müssen diese nun noch an den richtigen Stellen verdoppeln bzw. zurück auf den Wert 1 setzen:

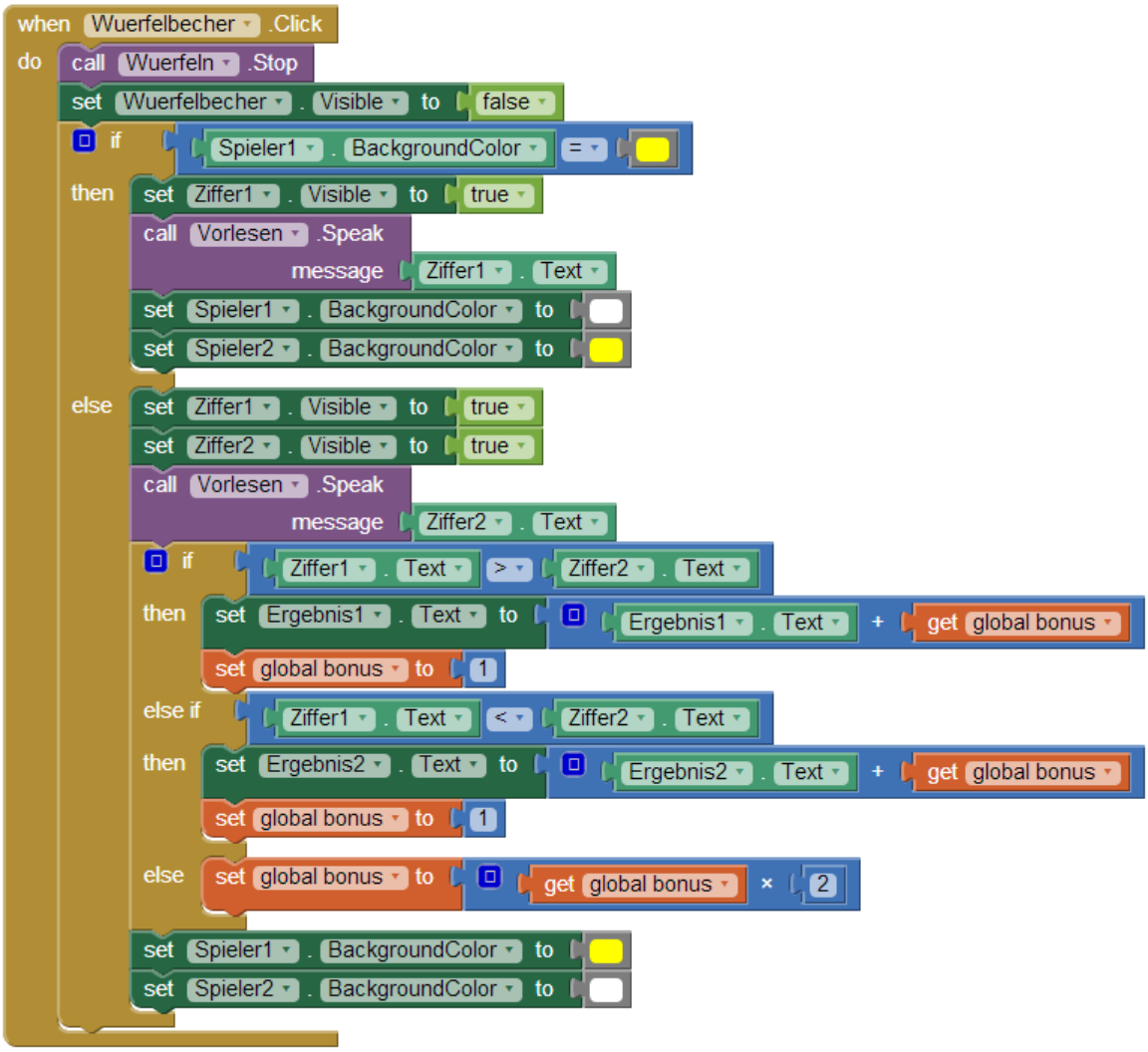


Abbildung 6.14: Punktevergabe mit der Variable `bonus`.

In Abbildung 6.15 ist das fertige Würfelspiel-Plus abgebildet:

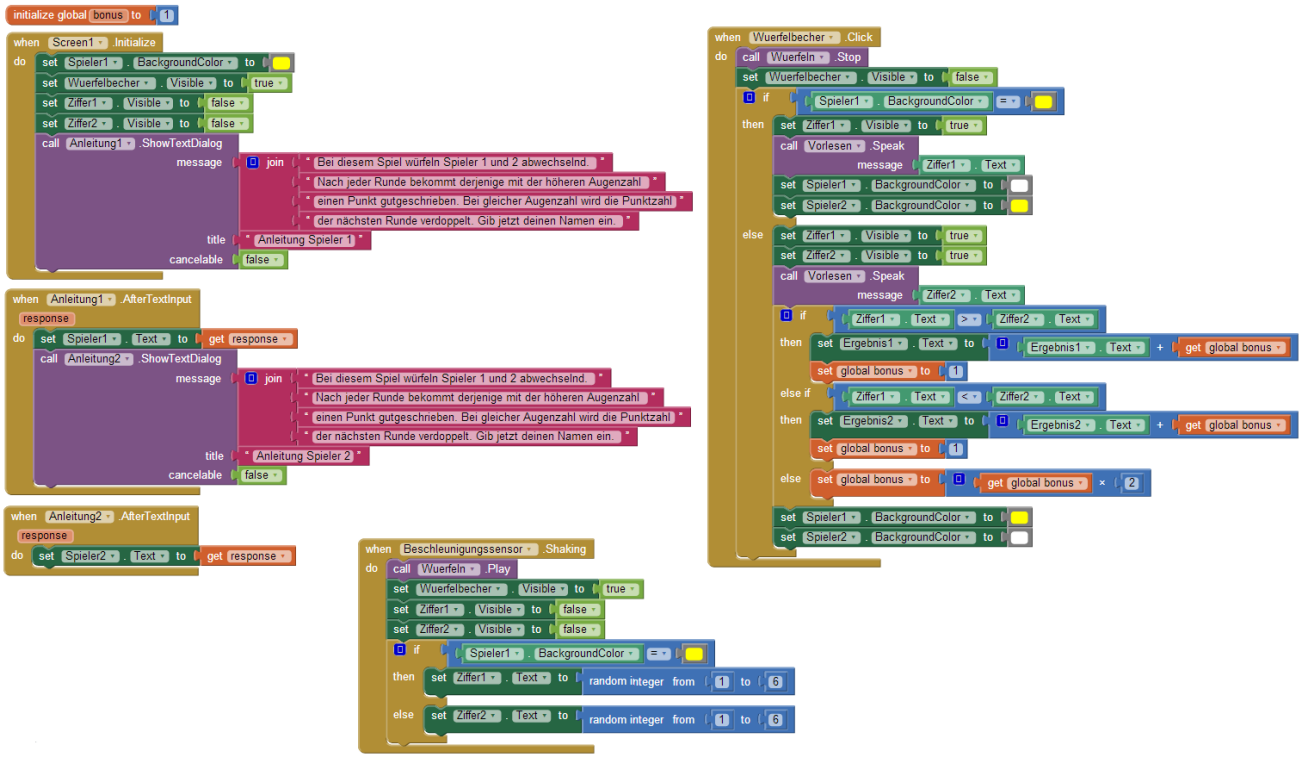


Abbildung 6.15: Das fertige Würfelspiel-Plus.

6.2 Barcodescanner

Wir erzeugen zunächst mit Drag & Drop ein Objekt der Klasse `Barcodescanner`, sowie ein Objekt der Klasse `Button` und ein Objekt der Klasse `Label`. Wir nennen das Button-Objekt *Scannen* und ändern den Text auf *Barcode einscannen*. Das Label-Objekt nennen wir *DecodierterText* und löschen den Inhalt.

Beim Klick auf die Schaltfläche `Barcode einscannen` möchten wir den Barcodescanner aufrufen. Dies müssen wir im Ereignisverarbeiter `Scannen.Click` festlegen. Nach erfolgreichem Einscannen müssen wir nun noch das Ergebnis im Label-Objekt anzeigen. Dies legen wir im Ereignisverarbeiter `BarcodeScanner1.AfterScan` fest:

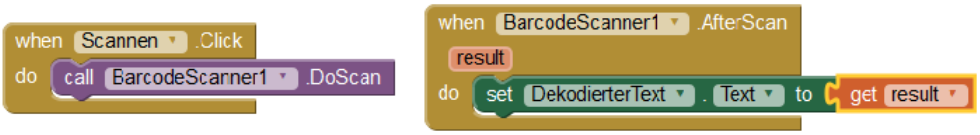


Abbildung 6.16: Das fertige Programm des Barcodescanners.

Erweiterung: Sprechender Barcodescanner

Beim Starten der Applikation soll Land und Sprache des `TextToSpeech`-Objekts festgelegt werden. Dies erledigen wir in der Methode `Screen1.Initialize`:

Als nächstes müssen wir noch geeignet auf das Anklicken der neuen Schaltfläche `Vorlesen` reagieren. Wir überprüfen, ob überhaupt ein Text eingelesen wurde. Ist dies der Fall, rufen wir die `Speak`-Methode des `TextToSpeech`-Objekts auf, um uns den Text vorlesen zu lassen.

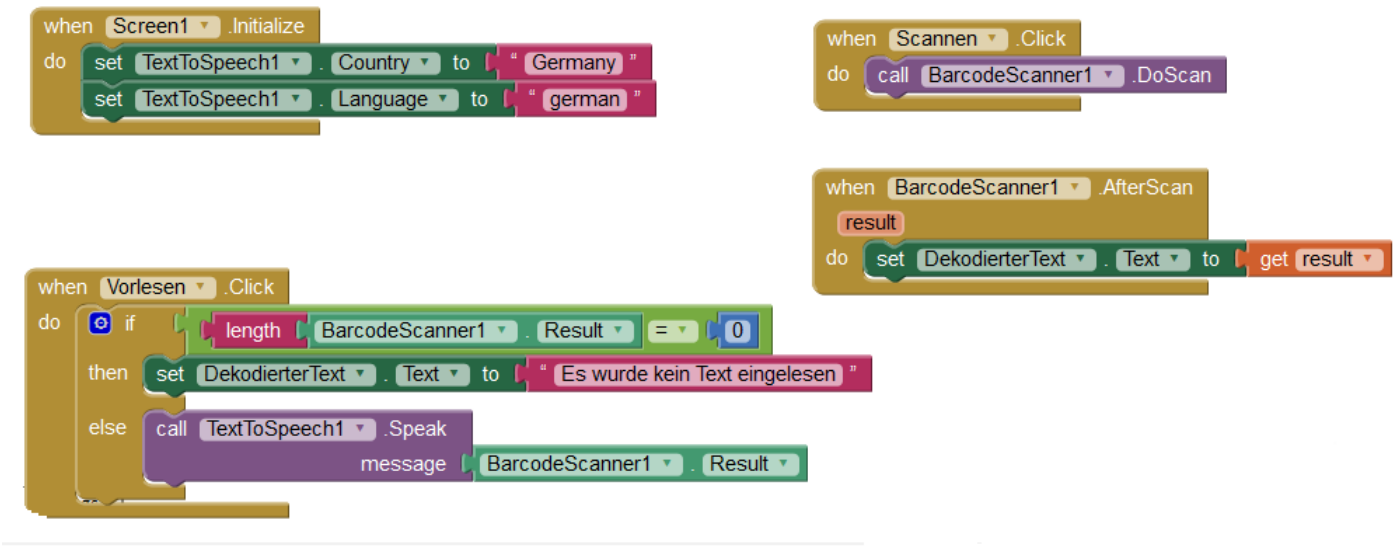


Abbildung 6.17: Das fertige Programm des sprechenden Barcodescanners.

6.3 Beschleunigungssensor-Anzeige

Wir erzeugen für die Anzeige 6 Label-Objekte, wobei wir die Objekte für die gemessenen Werte `ax` bzw. `ay` bzw. `az` nennen. Die Namen der Label-Objekte mit den zugehörigen Bezeichnungen (z.B. *x-Richtung*) sind für uns im Folgenden unwichtig.
Neben den Label-Objekten benötigen wir noch ein Clock-Objekt und wählen als Timer-Intervall 1000 ms.

Um die Sensorwerte auszulesen und anzuzeigen benötigen wir in der `Blocks-Sicht` nur folgenden Baustein-Block:

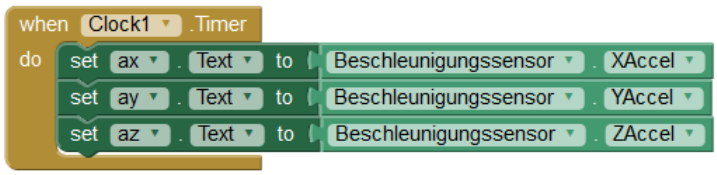


Abbildung 6.18: Die fertige Beschleunigungssensor-Anzeige.

6.4 Reaktionsspiel

Browser-Sicht

Wir legen zunächst in der `Designer-Sicht` entsprechend der Aufgabenstellung die benötigten `Button`- und `Label`-Objekte an. Um den Farbwechsel realisieren zu können, benötigen wir ein `Clock`-Objekt - wir setzen das Timer-Intervall auf 600 ms. Zusätzlich benötigen wir noch ein `Notifier`-Objekt, um mitteilen zu können, wenn ein Spieler 10 Punkte erreicht hat.

Editor-Sicht

Beginnen wir damit, den ständigen Farbwechsel zu realisieren. Wir haben dazu bereits in der Browser-Sicht ein `Clock`-Objekt angelegt. In der `Blocks-Sicht` müssen wir nun noch den entsprechenden Ereignisverarbeiter „füllen“. Wir erstellen uns dazu eine Liste mit 7 Farben und wählen daraus eine beliebige Farbe aus, die wir als neue Hintergrundfarbe setzen:

Abbildung 6.19: Wird der Timer des `Clock`-Objekts „gefeuert“, wird zufällig ein Element der Farbliste als neue Hintergrundfarbe gesetzt.

Als nächstes füllen wir die Ereignisverarbeiter `...Click` der beiden Schaltflächen `Spieler1` und `Spieler2`: Ist die Hintergrundfarbe Rot, erhält der Spieler, der auf seine Schaltfläche gedrückt hat, einen Punkt - falls nicht, erhält der andere Spieler einen Punkt:

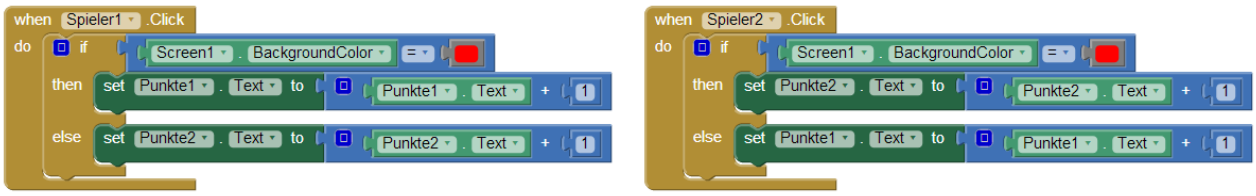


Abbildung 6.20: Festlegen der Punktevergabe, falls ein Spieler auf seine Schaltfläche drückt.

Bei den zwei Ereignisverarbeitern `...Click` haben wir noch vergessen zu überprüfen, ob ein Spieler bereits gewonnen hat. Da wir diese Überprüfung in beiden Ereignisverarbeitern brauchen, erstellen wir uns eine eigene Methode, um gleiche Bausteinabfolgen zu vermeiden. Hat ein Spieler 10 Punkte erreicht, geben wir über das `Notifier`-Objekt eine entsprechende Nachricht aus und setzen den Punktestand zurück auf 0:

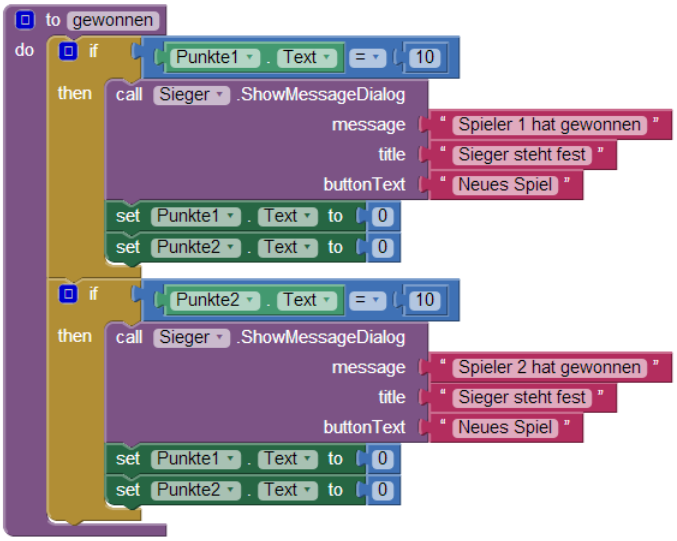


Abbildung 6.21: Eigene Methode `gewonnen` - Sieger wird ausgegeben und Punktestand zurückgesetzt.

Bei beiden `if`-Abfragen müssen wir am Ende die Punktetafel auf 0 zurücksetzen. Auch hier könnte man eine eigene Methode erstellen:

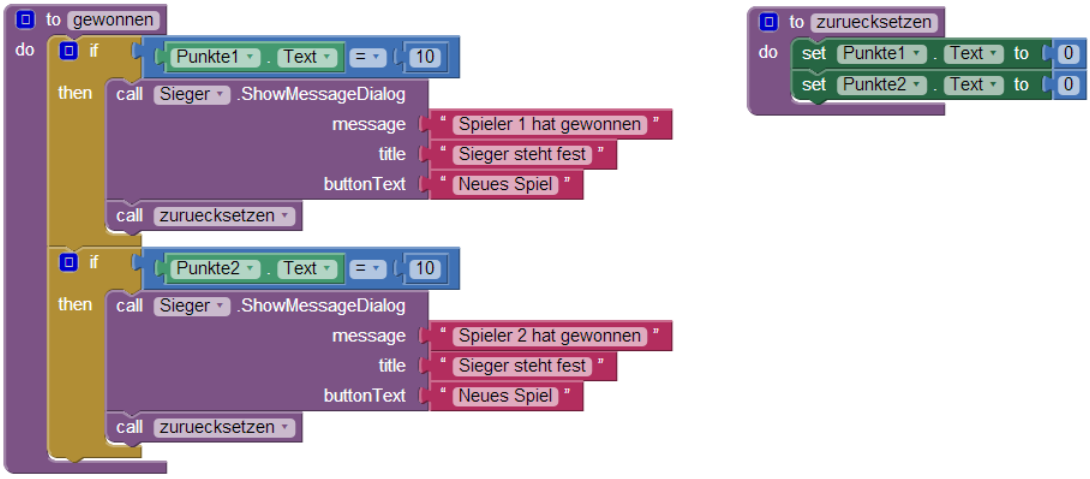


Abbildung 6.22: Eigene Methode für das Zurücksetzen der Punktetafel.

Wir müssen unsere `gewonnen`-Methode noch an geeigneter Stelle aufrufen:

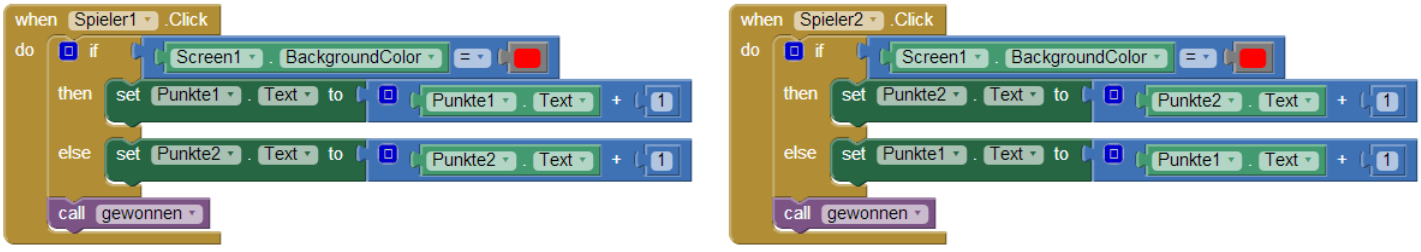


Abbildung 6.23: Überprüfen, ob ein Spieler bereits gewonnen hat.

Wir sind nun mit der Standard-Variante unseres Reaktionsspiels fertig:

Abbildung 6.24: Das fertige Reaktionsspiel.

6.4.1 Alternative Punktevergabe (Optional)

Um die alternative Punktevergabe realisieren zu können, benötigen wir eine Variable, mit der wir überprüfen können, ob in einer „Rotphase“ bereits ein Punkt vergeben worden ist. Dazu eignet sich eine `boolean`-Variable. Wir nennen diese *sperren*.

Erfolgt ein Farbwechsel, setzen wir unsere Variable auf `false`, d.h. wir möchten die Punktevergabe nicht sperren. Drückt ein Spieler auf seine Schaltfläche, müssen wir nun überprüfen, ob die Punktevergabe gesperrt ist oder nicht. Ist sie noch nicht gesperrt, vergeben wir einen Punkt und setzen dabei unsere Variable auf `true`, d.h. bis zum nächsten Farbwechsel wird die Punktevergabe gesperrt.

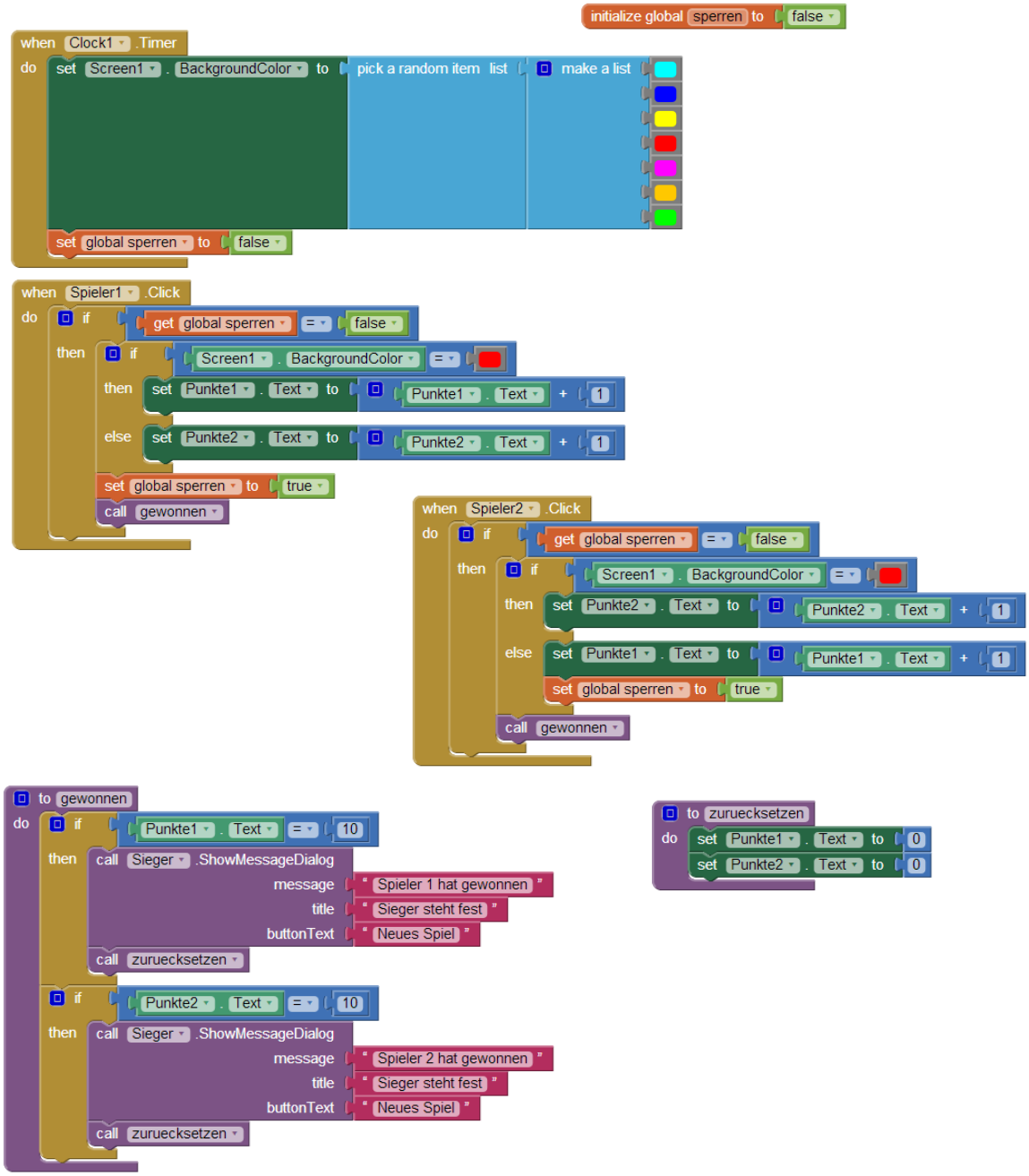


Abbildung 6.25: Das fertige Reaktionsspiel mit alternativer Punktevergabe.

6.4.2 Gewinner verkünden (Optional)

Um die Namen der Spieler einlesen zu können, erstellen wir zwei `Notifier`-Objekte *Name1* und *Name2* - zum Vorlesen benötigen wir ein `TextToSpeech`-Objekt.

Bei Spielbeginn sollen die Spieler zunächst ihre Namen eingeben. Dazu rufen wir im Ereignisverarbeiter `Screen1.Initialize` die Methode `Name1.ShowTextDialog`. Der zugehörige Ereignisverarbeiter, der aufgerufen wird, wenn der Name eingegeben worden ist, lautet `Name1.AfterTextInput`. Hier setzen wir den eingegebenen Namen als Namen für Spieler 1 und rufen `Name2.ShowTextDialog` auf. Den von Spieler 2 eingegebenen Namen müssen wir auch noch setzen - dies erledigen wir im Ereignisverarbeiter `Name2.AfterTextInput`:

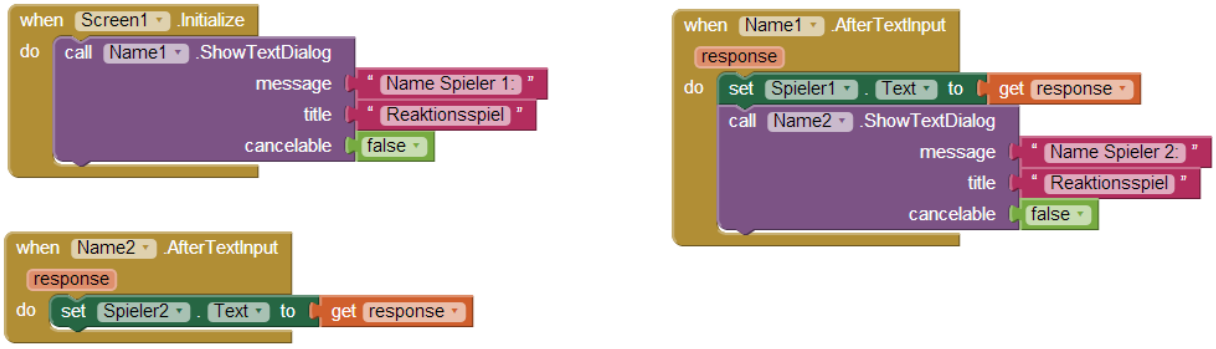


Abbildung 6.26: Die Spieler können zu Beginn ihren Namen eingeben.

Wir wollen nun den Sieger nicht mehr mit dem `Notifier`-Objekt ausgeben lassen, sondern vorlesen lassen. Dazu ändern wir die Methode `gewonnen` wie folgt ab:

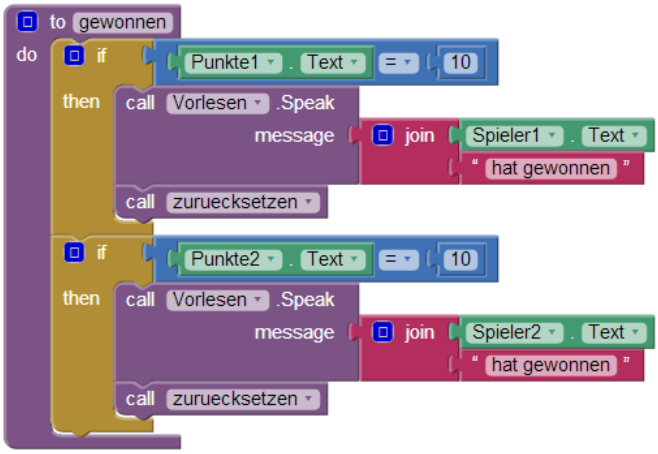


Abbildung 6.27: Der Sieger wird vorgelesen.

Am restlichen Programm müssen wir nichts verändern.

6.5 Das Pong-Spiel

Wir erzeugen und initialisieren analog zur Aufgabenstellung in der `Designern-Sicht` alle geforderten Objekte:

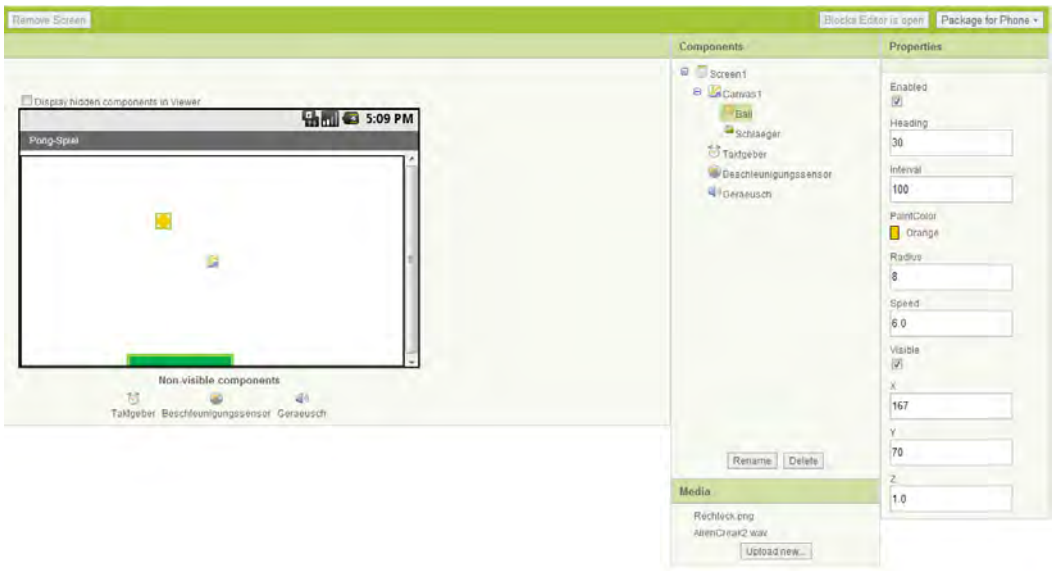


Abbildung 6.28: Erzeugen und initialisieren der in der Aufgabenstellung aufgeführten Objekte.

Wir erzeugen und initialisieren die zwei Variablen `v` und `deltaT`. Bei der physikalischen Steuerung des Schlägers mittels Kippen des Mobile Devices halten wir uns an die Vorgaben der Aufgabenstellung:

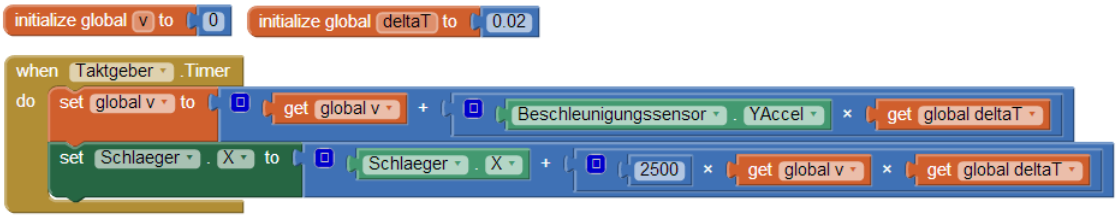


Abbildung 6.29: Erzeugen der Variablen und Festlegen der physikalischen Steuerung des Schlägers.

Als nächstes legen wir fest, dass der Schläger nicht über den Rand des Displays hinausschießt, sondern anhält. Wir setzen dabei, wie gefordert, `v` auf 0 und schieben den Schläger um ein Pixel vom Rand weg. Der rechte Rand hat den Wert +3, der linke Rand den Wert -3:

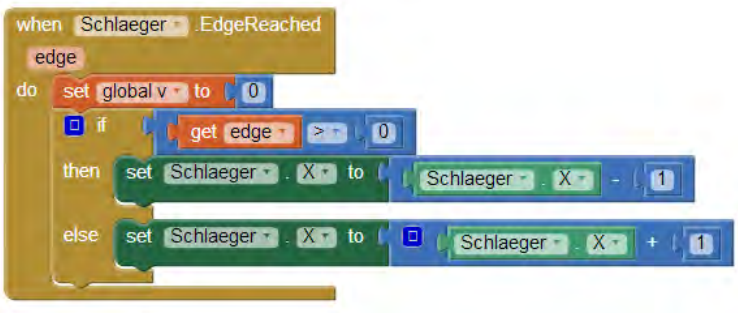


Abbildung 6.30: Schläger darf nicht über den Rand hinausschießen.

Der Ball soll ebenfalls von den Wänden abprallen. Wir benutzen dabei die vorgefertigte Methode `Bounce` und spielen für den Fall, dass der Ball die untere Wand berührt, die Sound-Datei ab:

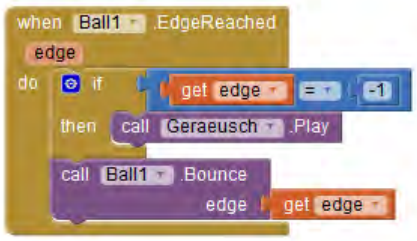
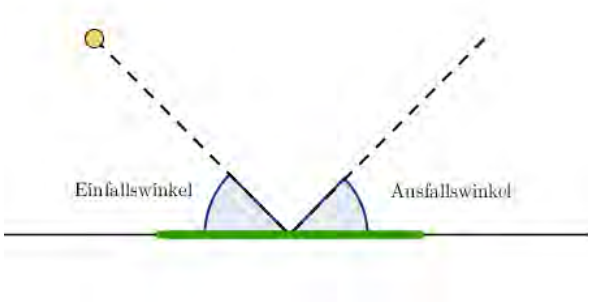
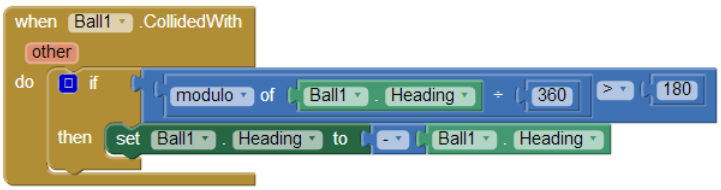


Abbildung 6.31: Der Ball soll von den Wänden abprallen. Berührt er die untere Wand wird die Sound-Datei abgespielt.

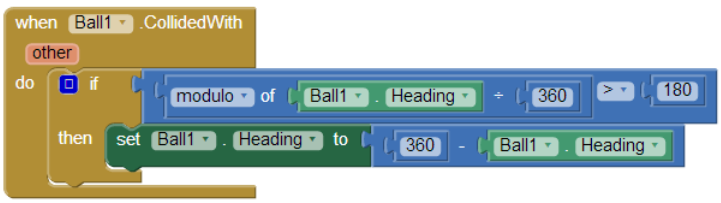
Wir müssen nun noch das Abprallen des Balls vom Schläger realisieren. In der Aufgabenstellung wurde dabei darauf hingewiesen, dass der Ball nur dann vom Schläger abprallen soll, wenn er von oben auf den Schläger trifft - d.h, `Ball.Heading` zwischen den Werten 180 und 360 liegt.
Betrachten wir zunächst folgende Skizze:



Durch die Skizze und das Reflexionsgesetz der Physik stellen wir fest, dass wir für den richtigen Ausfallswinkel nur den Einfallswinkel negieren müssen, d.h. wir ändern einfach das Vorzeichen von `Ball.Heading`:



Möchte man keine negativen Werte für `Ball.Heading` ist folgende Lösung natürlich auch richtig:



Unser Pong-Spiel ist nun fertig:

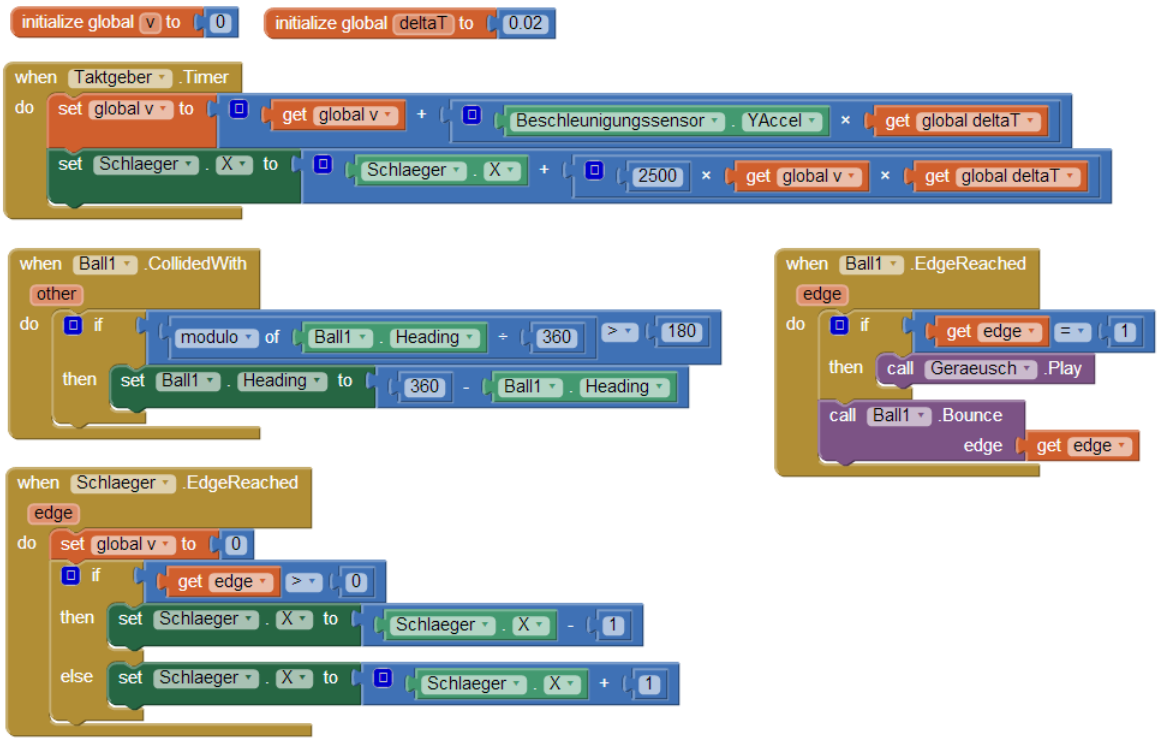


Abbildung 6.32: Das fertige Pong-Spiel.

6.5.1 Das Pong-Spiel mit Zähler und Bestleistung (Optional)

Wir benötigen für die Zähleranzeige 4 Label-Objekte, die wir in der Browser-Sicht erstellen. Wir möchten die Label-Objekte gerne nebeneinander anordnen. Dies erreichen wir, indem wir ein `HorizontalTableArrangement`-Objekt verwenden. In der Editor-Sicht müssen wir nun noch die Zählerstände entsprechend verwalten. Berührt der Ball die untere Kante, muss der aktuelle Zählerstand wieder auf 0 zurückgesetzt werden:

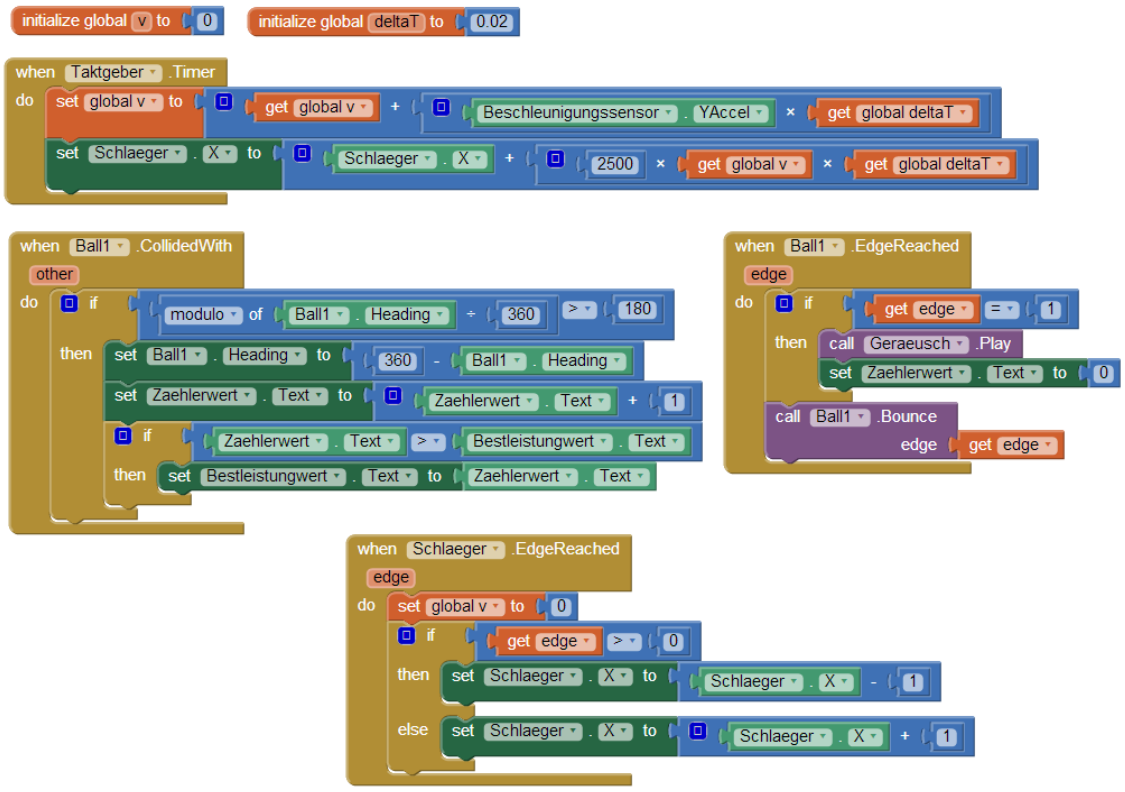


Abbildung 6.33: Das fertige Pong-Spiel mit Zähler und Bestleistung.

6.6 Robotersteuerung

Verbindung mit dem Roboter herstellen

Wir folgen der Beschreibung der Aufgabenstellung und erstellen ein Pairing zwischen Roboter und Mobile Device.

Als nächstes versuchen wir nun mit unserer Applikation eine Bluetooth-Verbindung zwischen Roboter und Mobile Device herzustellen. Wir erstellen dazu in der `Designer-Sicht` ein Objekt der Klasse Bluetooth-Client und jeweils ein Objekt der Klassen `ListPicker` (für die Verbinden-Schaltfläche) und Button (für die Trennen-Schaltfläche) mit dem Namen *ListPickerVerbinden* bzw. *Trennen*. Bei dem Trennen-Objekt entfernen wir den Haken bei `Enabled`, da es keinen Sinn macht, dieses anzuklicken, solang noch keine Verbindung besteht:

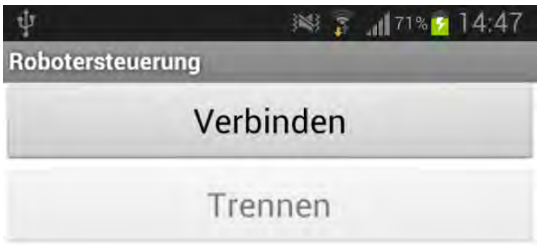


Abbildung 6.34: Schaltflächen zum Verbinden und Trennen mit dem Roboter.

Im `Blocks Editor` füllen wir nun die Ereignisverarbeiter `ListPickerVerbinden.BeforePicking`, `ListPickerVerbinden.AfterPicking` und `Trennen.Click`. Bei Klicken auf die Verbinden-Schaltfläche soll uns die Liste mit den Pairing-Geräten des Mobile Device angezeigt werden. Dies erreichen wir wie folgt:

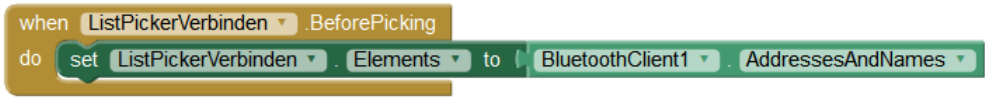


Abbildung 6.35: Anzeigen der Liste aller Pairing-Geräte des Mobile Device.

Nach dem Auswählen des Roboters soll ein Verbindungsaufbau gestartet werden. War dieser erfolgreich, deaktivieren wir die Verbinden-Schaltfläche und aktivieren die Trennen-Schaltfläche:

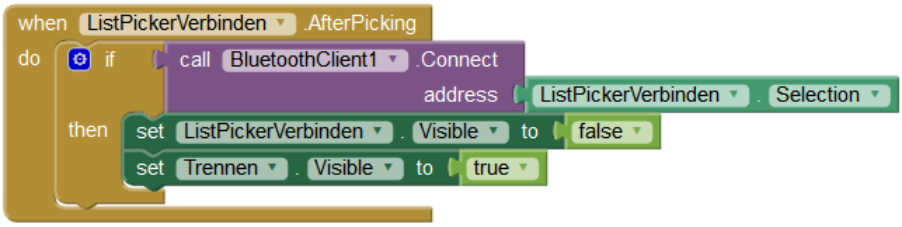


Abbildung 6.36: Verbindungsaufbau mit dem Roboter und aktivieren bzw. deaktivieren der Trennen- bzw. Verbinden-Schaltfläche.

Bei Klicken auf die Trennen-Schaltfläche beenden wir die bestehende Verbindung und aktivieren bzw. deaktivieren die Verbinden- bzw. Trennen-Schaltfläche:

Abbildung 6.37: Trennen der Verbindung und deaktivieren bzw. aktivieren der Trennen- bzw. Verbinden-Schaltfläche.

Steuern des Roboters mit Tasten

Wir erstellen zunächst weitere Schaltflächen für Vorwärts- und Rückwärtsfahren:

Abbildung 6.38: Schaltflächen zum Vorwärts- und Rückwärtsfahren.

Für diese Schaltflächen müssen wir noch die entsprechende Ereignisverarbeitung in der `Blocks-Sicht` festlegen. Durch Klicken auf die Vorwärts-Schaltfläche soll unser Roboter vorwärts fahren. Wir haben hier zwei mögliche Ereignisverarbeiter zur Auswahl:

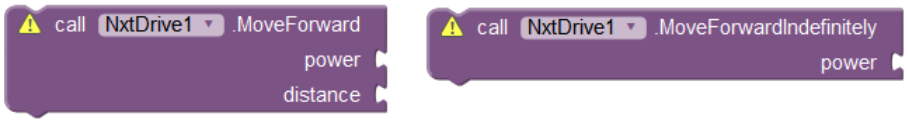


Abbildung 6.39: Mögliche Bausteine, um das Vorwärtsfahren festzulegen.

Mit Hilfe des linken Bausteins können wir den Roboter eine bestimmte Distanz zurücklegen lassen, etwa 40 cm mit „Geschwindigkeit“ 80:

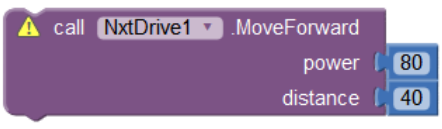


Abbildung 6.40: Mit dem Baustein `MoveForward` kann man „Geschwindigkeit“ und Distanz festlegen.

Wir entscheiden uns für den rechten Baustein, d.h. unsere Ereignisverarbeiter zum Vorwärts- und Rückwärtsfahren schauen wie folgt aus:

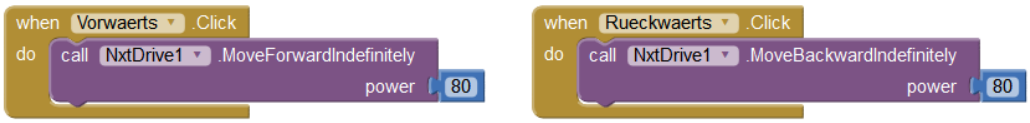


Abbildung 6.41: Ereignisverarbeiter zum Vorwärts- und Rückwärtsfahren.

Wir können unseren Roboter nun bereits vorwärts und rückwärts fahren lassen. Allerdings haben wir das Problem, dass wir den Roboter, sobald er einmal fährt, nur noch stoppen können, indem wir die Bluetooth-Verbindung kappen. Wir erstellen also neben den Schaltflächen *Rechts* und *Links* noch eine Schaltfläche *Stoppen* und dazu wieder geeignete Ereignisverarbeiter:

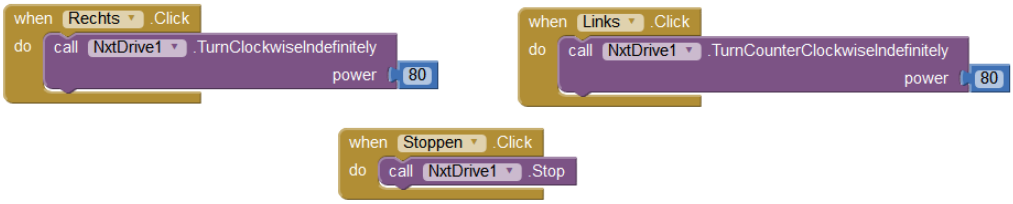


Abbildung 6.42: Ereignisverarbeiter zum Vorwärts- und Rückwärtsfahren.

Unser fertiges Programm zur Robotersteuerung sieht also wie folgt aus:

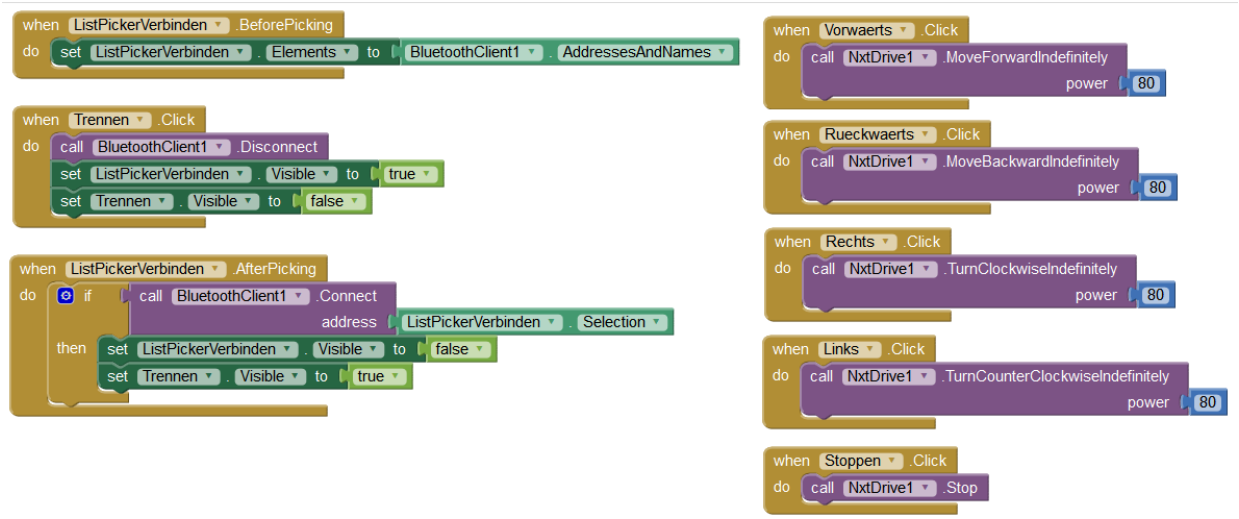


Abbildung 6.43: Das fertige Programm zur Robotersteuerung.

6.6.1 Steuern mittels Beschleunigungssensor (Optional)

Wir wollen unseren Roboter nicht mehr durch entsprechende Tasten steuern, sondern durch das Mobile Device selbst, indem wir dieses in die jeweilige Richtung neigen. Wir benötigen dazu einen Beschleunigungssensor und einen Timer. Als Timer-Intervall wählen wir 150 ms.

Wir haben bei unserem Roboter den Motor für das (in Fahrtrichtung) rechte Rad mit B und den Motor für das linke Rad mit C verbunden. Wollen wir vorwärts fahren, legen wir an beiden Motoren Power 80 an. Um anzuhalten stoppen wir beiden Motoren. Bei den Kurven muss man ein bisschen rumtüfteln: um nach rechts zu fahren stoppen wir das linke Rad und fahren mit dem rechten Rad rückwärts. Für eine Linkskurve gilt das gleiche nur vertauscht. Natürlich sind hier auch andere Lösungen denkbar!

Unser fertiges Programm zur Robotersteuerung mittels Beschleunigungssensor sieht wie folgt aus:

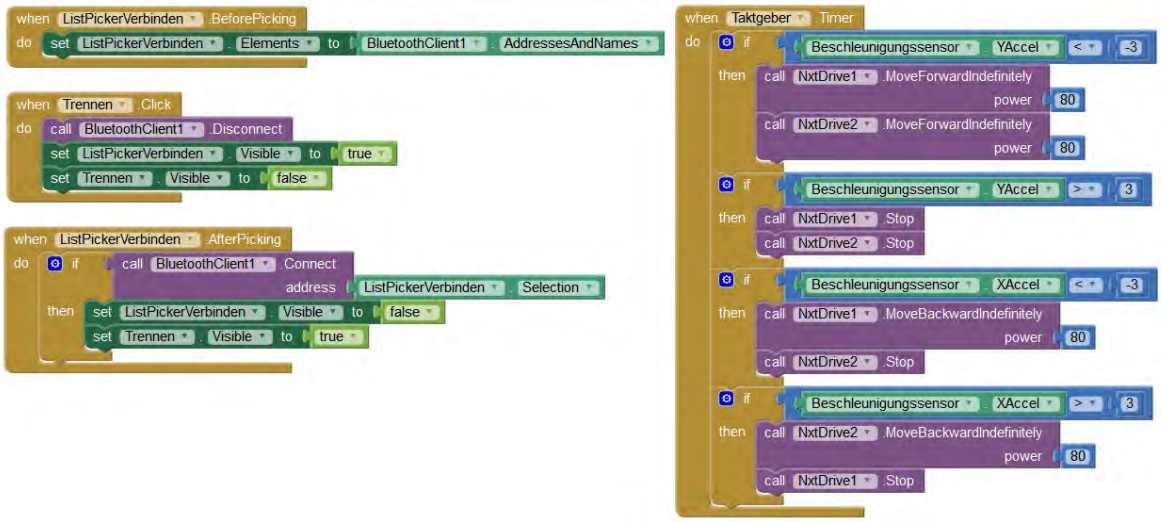


Abbildung 6.44: Das fertige Programm zur Robotersteuerung mittels Beschleunigungssensor.

6.6.2 Weitere Funktionen einbauen (Optional)

Wir haben für diese Aufgabe einen Roboter mit Ultraschallsensor verwendet:

Abbildung 6.45: Lego NXT Roboter mit Ultraschallsensor.

In der `Designer-Sicht` erstellen wir, zusätzlich zu den Objekten der vorangegangenen Aufgabe aus Unterabschnitt 6.6.1, zwei Label-Objekte für die Distanzanzeige und zwei CheckBox-Objekte, jeweils eins um auswählen zu können, ob der Roboter bei einem Hindernis rückwärts fahren soll, oder eine Sound-Datei abspielen soll. Zudem brauchen wir noch jeweils ein Objekt der Klassen `NxtUltrasonicSensor` und `NxtDirectCommands`. Bei beiden müssen wir wieder die Bluetooth-Schnittstelle einstellen.

Unsere Applikation hat nun folgende grafische Benutzeroberfläche:

Abbildung 6.46: Grafische Benutzeroberfläche unserer Robotersteuerung mit erweiterten Funktionen.

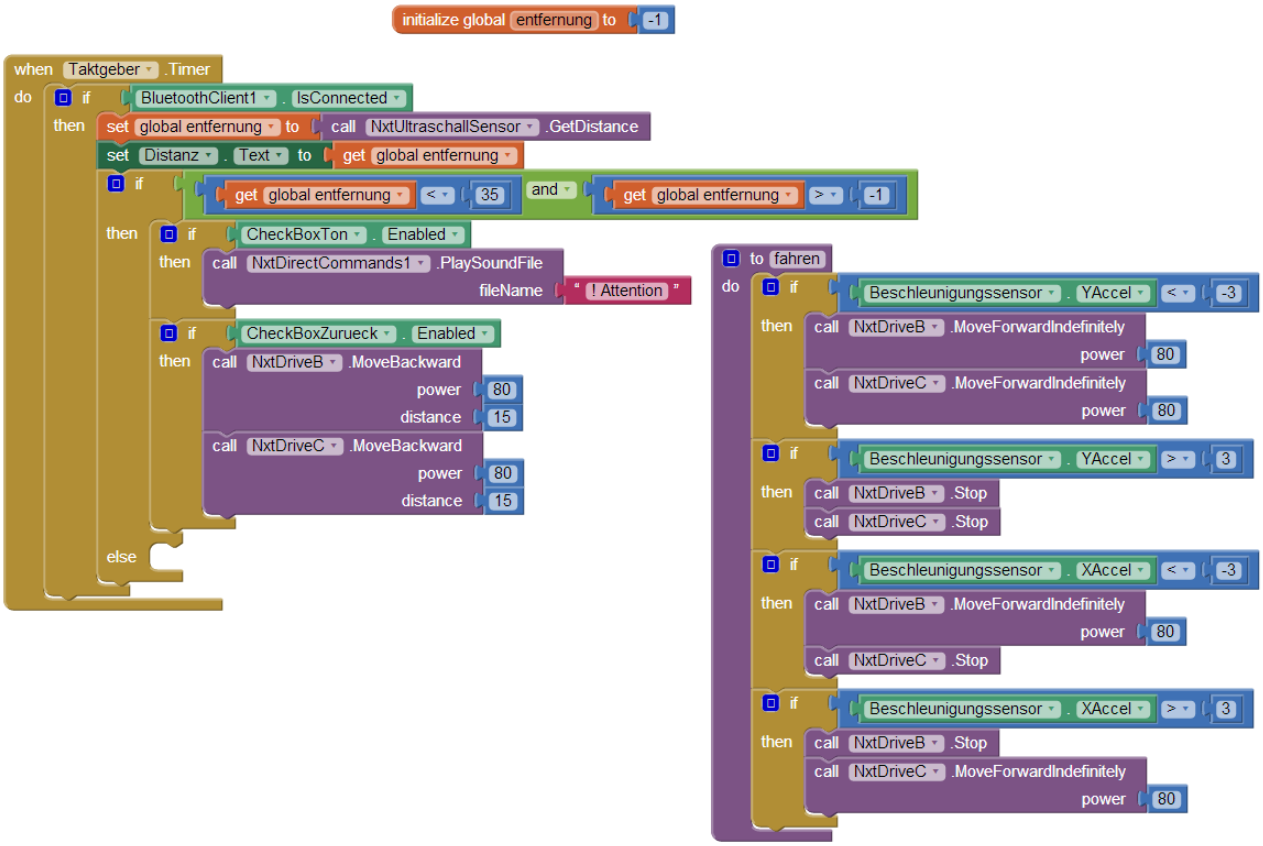
Als initialen Wert für die Distanz haben wir -1 gewählt, um anzuzeigen, dass noch keine Distanz gemessen wurde. Der Ultraschallsensor wählt auch den Wert -1, wenn er in seiner Reichweite keinen Gegenstand findet.

Als nächstes kümmern wir uns um die Ereignisverarbeitung in der `Blocks-Sicht`. Wir müssen dabei folgende Eigenschaften umsetzen:

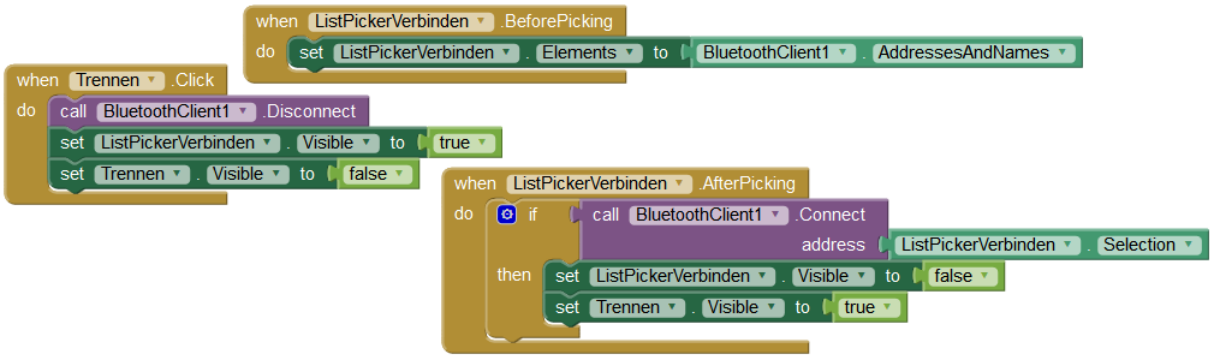
- Überprüfen, ob die Bluetooth-Verbindung erfolgreich aufgebaut wurde.
- Wert des Ultraschallsensors auslesen und in das dafür vorgesehen Label schreiben.
- Überprüfen, ob wir nahe an einem Hindernis sind. Wir haben hier einen Abstand von weniger als 35 gewählt. Gleichzeitig muss noch sicher gestellt werden, dass eine Entfernung gemessen wurde, sprich der Wert größer ist als -1.

- Wenn der entsprechende Haken gesetzt wurde, wird eine Sound-Datei abgespielt (hier: `! Attention`).
- Wenn der entsprechende Haken gesetzt wurde, lassen wir den Roboter rückwärtsfahren (hier: um den Wert 15).
- Ist man nicht nahe an einem Hindernis, soll man den Roboter ganz normal steuern können.

Setzt man diese Punkte mehr oder weniger 1 zu 1 in Bausteine um, ergibt sich folgende Bausteinsequenz:



Die Ereignisverabeiter zum Erstellen und Trennen der Bluetooth-Verbindung bleiben unverändert:

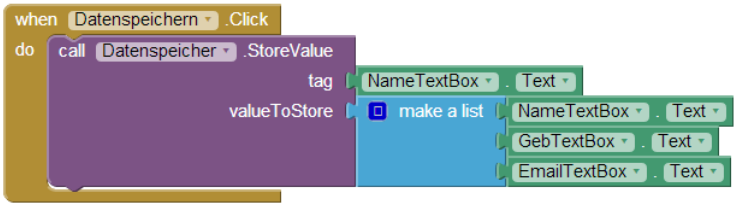


6.7 Datenspeicher

Die GUI ist denkbar einfach - wir orientieren uns an der Grafik aus der Aufgabenstellung und erzeugen die dort angegebenen Objekte. Wir wählen charakteristische Namen, so dass Sie diese im Folgenden ohne Probleme den entsprechenden Objekten zuordnen können.

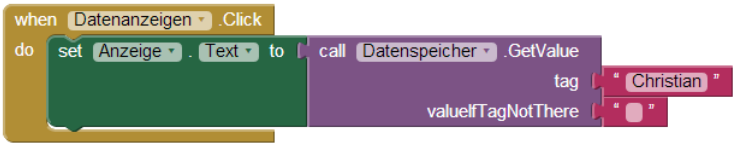
Daten speichern

Wir wollen nun die eingegebenen Daten nach Klicken auf die Schaltfläche *Daten speichern* in unseren Datenspeicher einfügen - dies müssen wir im Ereignisverarbeiter der Schaltfläche festlegen. Als `tag` benutzen wir laut Aufgabenstellung den Namen, die Daten speichern wir in einer Listenstruktur ab:



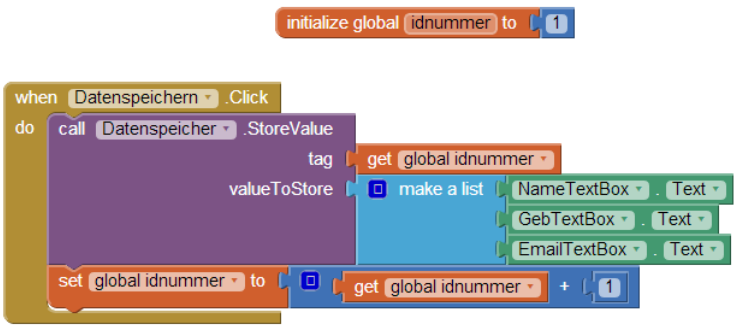
Daten auslesen

Wir erstellen eine weitere Schaltfläche und ein Label, um den gespeicherten Wert ausgeben zu können. Im zugehörigen Ereignisverarbeiter müssen wir noch festlegen, dass der Eintrag mit `tag` Christian im neu erstellten Label angezeigt wird:

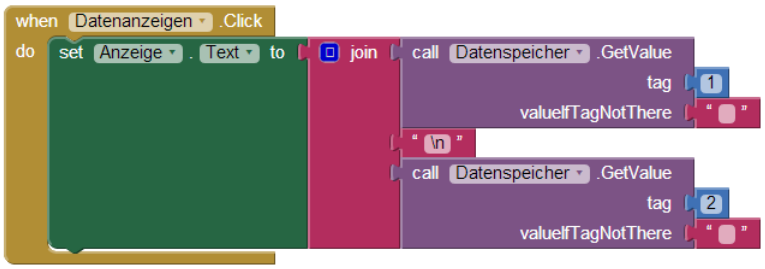


Funktionsweise des Datenspeichers

Wir erstellen uns eine eigene Variable `idnummer` und verwenden diese als `tag`. Immer wenn ein neuer Wert in den Datenspeicher eingefügt wurde, müssen wir unsere Variable um 1 erhöhen:



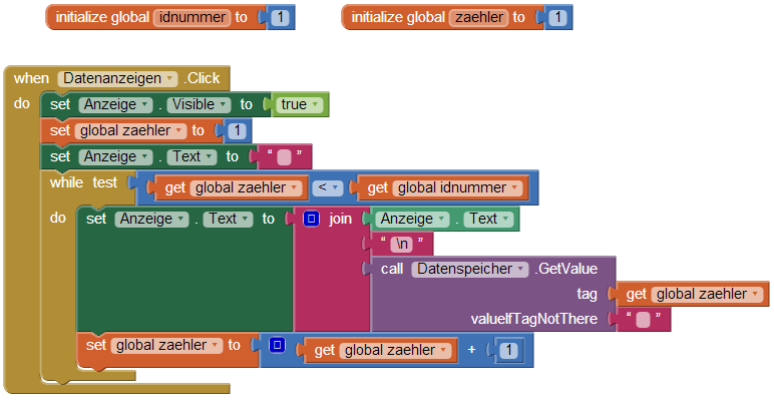
Für die Ausgabe verwenden wir den in der Aufgabenstellung angegebenen Codeblock:



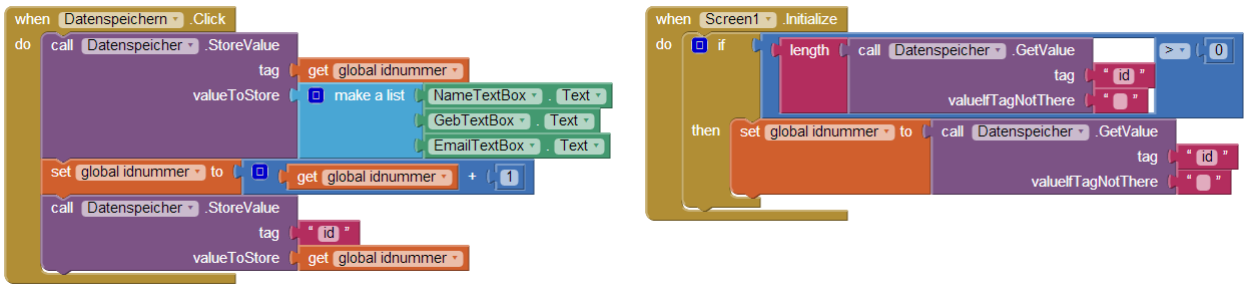
Alle Daten auslesen

Um alle Einträge zu durchlaufen, könnten wir unsere Variable `idnummer` verwenden - wir ziehen 1 ab [1], lesen die zu dem `tag` mit dieser Nummer gehörenden Daten aus, ziehen 1 ab, geben die zu dem `tag` mit dieser Nummer gehörenden Daten aus, ziehen 1 ab, ... , bis der Wert von `idnummer` 1 ist.
Das Problem bei dieser Vorgehensweise ist allerdings, dass wir dann die Information der Variable `idnummer` verlieren, d.h. wir wissen nicht mehr, wie viele Elemente in unserem Datenspeicher abgelegt sind.

Aus diesem Grund ist es notwendig, eine eigene Zählvariable zu definieren, die alle Werte von 1 bis zum Wert von `idnummer` durchläuft und die Daten der zugehörigen Tags ausgibt:
Nun müssen wir noch das Speichern der Variable `idnummer` umsetzen. Jedes Mal, wenn sich der Wert der

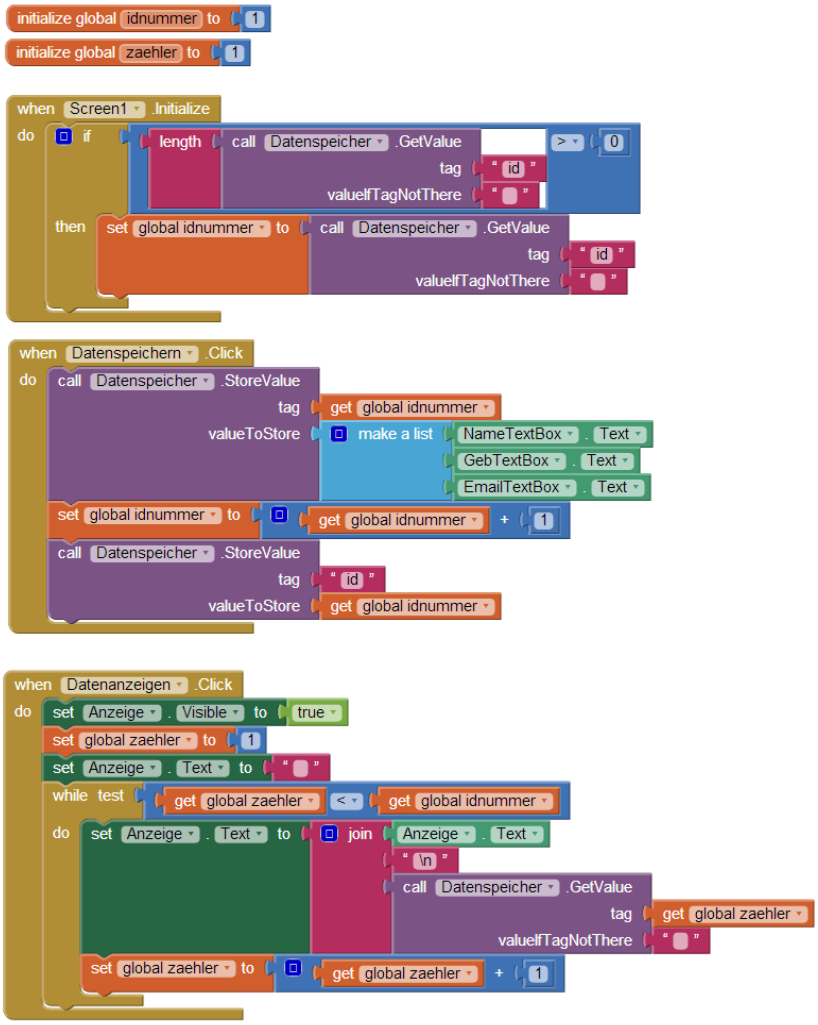


Variable verändert, müssen wir diesen speichern. Wird die App gestartet, müssen wir den aktuellen Wert von `idnummer` auslesen. Dafür eignet sich der Baustein `Screen1.Initialize`:



[1] `idnummer` ist immer um 1 größer, als der größte zugewiesene `tag`

Unsere Datenspeicher-App ist nun fertig:



6.8 Standortanzeiger

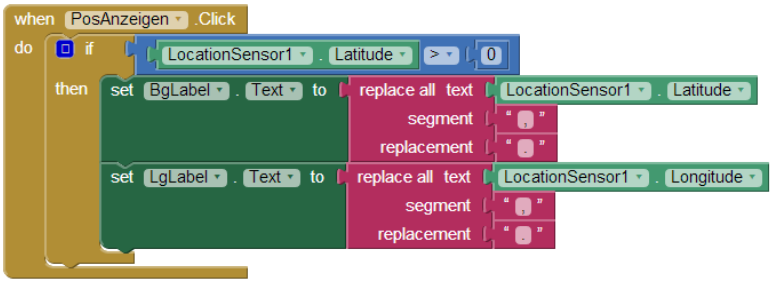
Die GUI erzeugen wir wieder anhand der Grafik in der Aufgabenstellung und bezeichnen dabei die jeweiligen Objekte aussagekräftig.

Nicht-Sichtbare Objekte

Wir erstellen je ein Objekt der Klasse LocationSensor und ActivityStarter und nehmen die angegebenen Einstellungen in den Eigenschaften des ActivityStarter-Objekts vor.

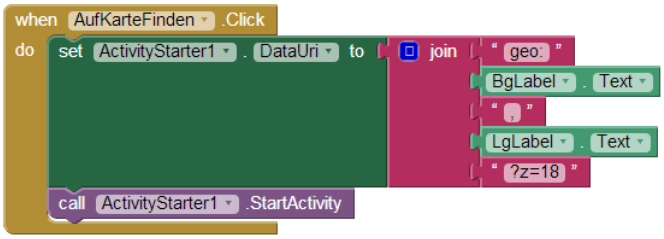
GPS-Koordinaten anzeigen

Die folgenden Code-Bausteine dürften selbsterklärend sein:



Auf Karte finden

Auch hier dürften die folgenden Code-Bausteine selbsterklärend sein:



Je größer der Zahlenwert im Textsegment `?z=18` ist, umso näher zoomt Google Maps an den gesuchten Punkt heran.

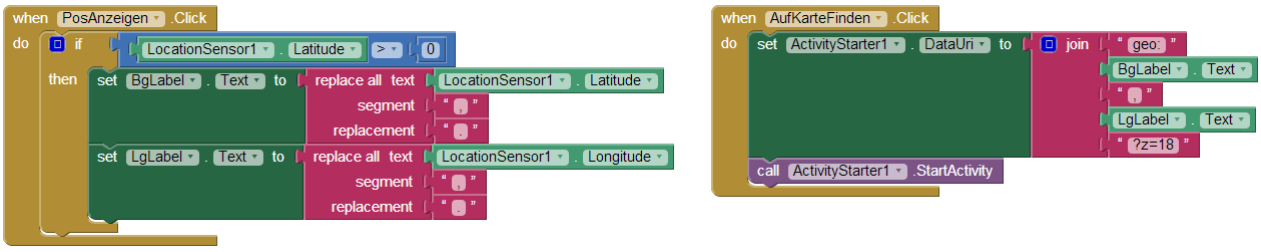


Abbildung 6.47: Unsere fertige App Standortanzeiger

6.9 Wegweiser

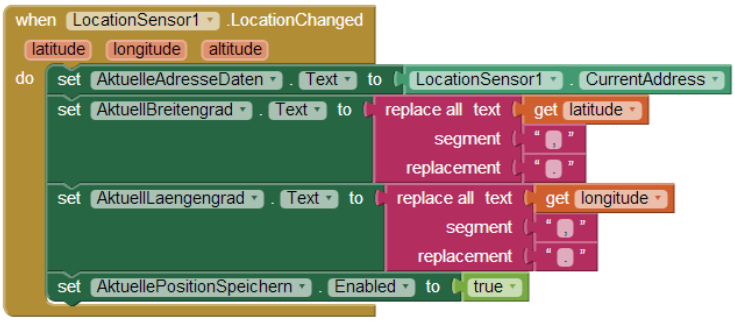
Die GUI erzeugen wir wieder anhand der Grafik in der Aufgabenstellung und bezeichnen dabei die jeweiligen Objekte aussagekräftig. Wir verwenden dabei vier HorizontalArrangement-Objekt für die Anordnung der Adresse bzw. der GPS-Daten.
Für den Breiten- bzw. Längengrad verwenden wir eigene Label-Objekte - ebenso für die Darstellung des Strich-Punkt.
Weiter benötigen wir noch zwei Button-Objekte für die Schaltflächen. Wir setzen beide initial auf Disabled.

6.9.1 Nicht-Sichtbare Objekte

Wir erstellen je ein Objekt der Klasse LocationSensor und ActivityStarter und nehmen die angegebenen Einstellungen in den Eingeschaften des ActivityStarter-Objekts vor.
Zudem erzeugen wir ein Objekt der Klasse TinyDB und nennen es Datenspeicher.

6.9.2 Aktuelle Standortdaten auslesen

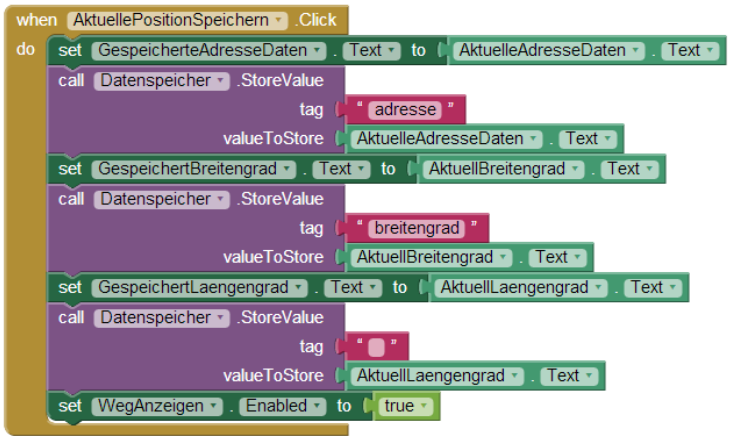
Auslesen des aktuellen Standorts und Auslesen der Daten in die entsprechenden Label-Objekte:



Zudem aktivieren wir die Schaltfläche *Aktuelle Position speichern*, nachdem wir zum ersten Mal GPS-Daten gefunden haben.

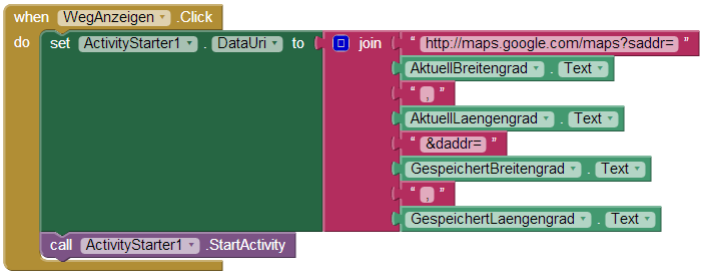
Aktuellen Standort speichern

Speichern des aktuellen Standorts im Datenspeicher und in den entsprechenden Feldern der Rubrik *Gespeicherter Ort* - zudem aktivieren wir die Schaltfläche *Weg von aktueller zu gespeicherter Adresse*:



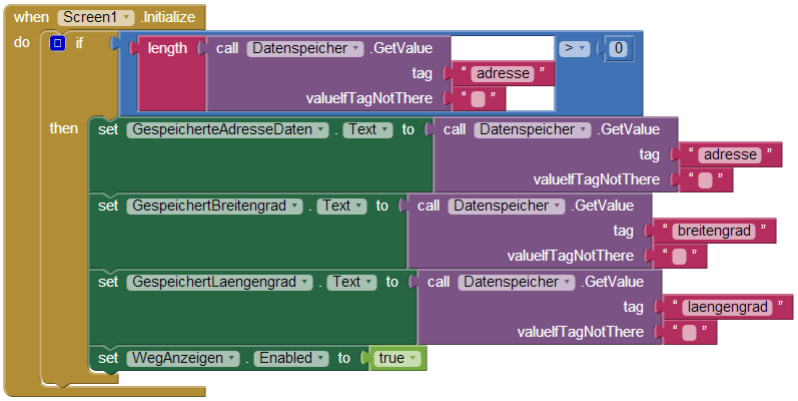
Weg mit Hilfe von Google Maps anzeigen

Der Codeblock zum Aufrufen von Google Maps ist in der Aufgabenstellung bereits vorgegeben, dabei steht `saddr` für *start address* und `daddr` für *destination address*:

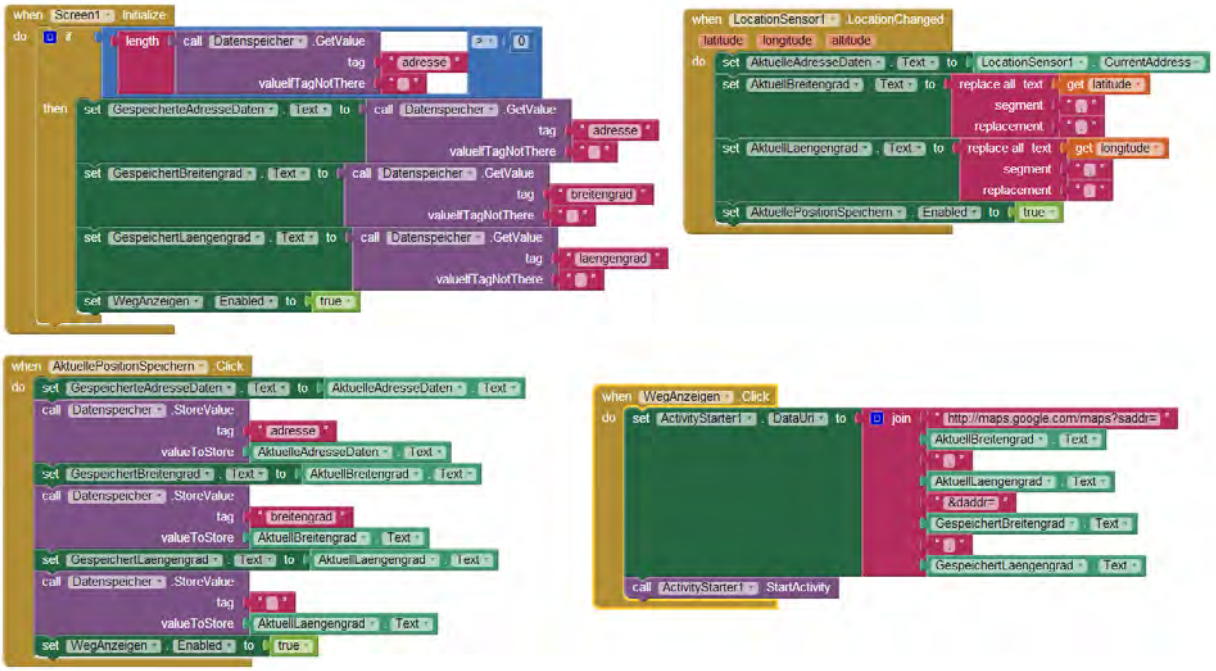


Gespeicherte Position bei Starten der Applikation auslesen

Der Ereignisverarbeiter `Screen1.Initialize` wird immer bei Programmstart aufgerufen. Wir überprüfen hier, ob eine Adresse im Datenspeicher abgelegt wurde und laden diese ggf. in die entsprechenden Label-Objekte:



Unsere App ist nun fertig:



Kapitel 7

Anhang

Überblick:

In diesem Kapitel bieten wir Ihnen zusätzliche Features, wie Installationsanleitungen, nützliche Informationen und Exkurse.

7.1 Installation Barcode-Scanner ohne Google-Play

7.1.1 Installation mit WLAN

Am schnellsten geht die Installation über eine WLAN-Verbindung. Verbinden Sie das Mobile Device mit dem WLAN und laden Sie auf der Internetseite

`https://code.google.com/p/zxing/downloads/list`

die `.apk`-Datei für Ihr Betriebssystem herunter:[1]

[1] Beim anschließenden Installieren der `.apk`-Datei kann es zu Problemen mit den Sicherheitseinstellungen des Mobilce Device kommen. Wir verweisen hier auf den entsprechenden Abschnitt auf Seite **??**.

Abbildung 7.1: Herunterladen der `.apk`-Datei von der ZXING-Seite.

Wählen Sie dann in der Kopf-Leiste oder unter der App `Downloads` die heruntergeladene Datei aus und installieren Sie diese:

Abbildung 7.2: Wählen Sie die heruntergeladene Datei aus ...

Abbildung 7.3: ... und installieren Sie diese.

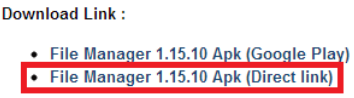
Die Applikation ist nun auf Ihrem Mobile Device installiert.

7.1.2 Installation mit USB

Alternativ können Sie die `.apk`-Datei auch auf Ihrem Computer downloaden und dann via USB (Verbinden Sie Ihr Mobile Device als Mediengerät) auf Ihr Mobile Device ziehen. Dort können Sie die Applikation dann wie in Unterabschnitt 7.1.1 beschrieben installieren.

Hinweis: Für diese Vorgehensweise muss auf Ihrem Mobile Device ein Datei-Manager installiert sein, damit Sie auf die mit USB übertragene `.apk`-Datei zugreifen können. Bei dem Nexus 7 ist standardmäßig kein Dateimanager installiert. Diesen müssen Sie einmal über eine WLAN-Verbindung installiert haben. Ohne Google Play Konto können Sie einen Dateimanager z.B. unter folgendem Link herunterladen:

`http://apkcute.blogspot.de/2013/05/file-manager-11510-apk.html`



7.2 App Inventor Projekte exportieren und importieren

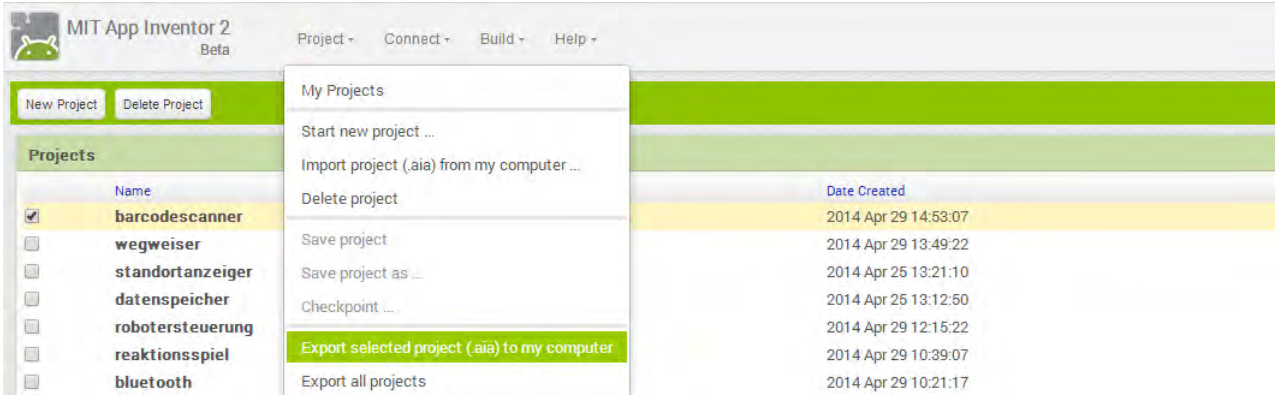
Es gibt grundsätzlich zwei Möglichkeiten, ein mit dem App Inventor erstelltes Projekt zu exportieren:

(1) Als *.aia-Datei auf den Computer

(2) Als *.apk-Datei zum Download via QR-Code oder auf dem Computer (`Build`)

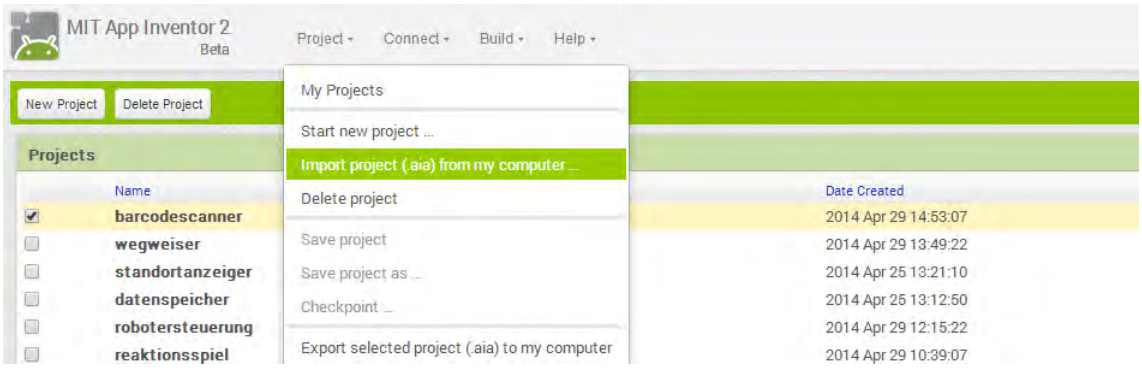
In diesem Abschnitt behandeln wir Punkt (1). Für Punkt (2) verweisen wir auf nachfolgenden Abschnitt 7.3.

Möchte man sein Programm anderen zur Verfügung stellen, so dass diese es auf ihren App Inventor hochladen und ggf. editieren können, bietet sich der Export als *.aia-Datei an.
Dies erreicht man in der `Desigern-Sicht` unter `Project` → `My Projects`. Man wählt dann das gewünschte Projekt aus und wählt unter der Schaltfläche `Project` den Befehl `Export selected Project to my computer` aus:



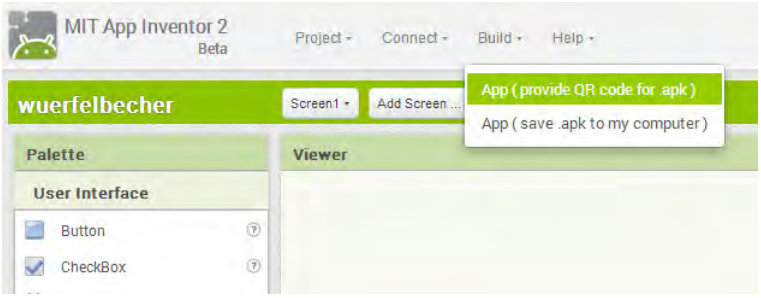
Das Projekt wird dann als *.aia-Datei exportiert.

Über `Project` → `Import project from my computer` kann man ein exportiertes Projekt in den AppInventor importieren:



7.3 Erstellte Applikation auf dem Mobile Device installieren

Möchte man die erstellte App auch ohne Verbindung mit dem AppInventor auf dem Mobile Device nutzen, kann man via `Build` die zugehörige *.apk-Datei entweder via QR-Code direkt auf dem Handy installieren, oder zunächst auf den Computer herunterladen. Dies geht in der `Designer-Sicht` mittels der Schaltfläche `Build`:



Die heruntergeladene App können Sie ab jetzt auch ohne AppInventor und Computer auf Ihrem Mobile Device ausführen. Beachten Sie, dass Sie nach jeder Änderung im AppInventor diesen Schritt erneut ausführen müssen, um die aktuellste Version auch auf dem Mobile Device zu haben.

Man kann das Projekt als *.apk-Datei auch auf den Computer herunterladen, um diese dann ggf. auf weitere Smartphones oder Tablets zu laden.

7.4 Bluetooth-Admin Rechte einer APK-Datei ändern

Bei neueren Android-Versionen kann es zu Problemen mit den Bluetooth-Rechten kommen. Möchten wir beispielsweise den Bluetooth-Server auf unserem Google Nexus 7 mit Android Version 4.2.1 starten, erhalten wir folgende Fehlermeldung:

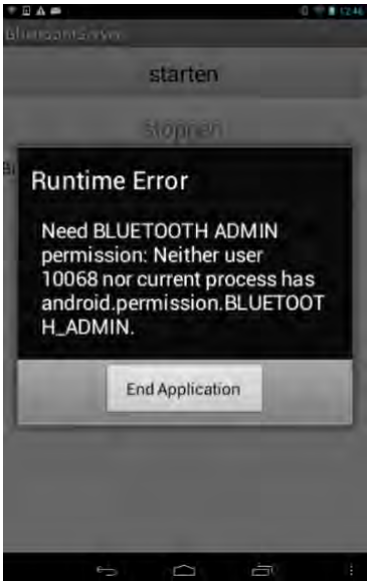


Abbildung 7.4: Fehlermeldung, dass Bluetooth-Admin Rechte nicht vergeben wurden.

Diese Admin-Rechte lassen sich mit dem AppInventor nicht direkt vergeben - wir müssen die Rechte nachträglich hinzufügen.
Uns ist dieses Problem bisher nur bei der Bluetooth-Texter Aufgabe bekannt. Wir haben Ihnen hierfür eine `.apk`-Datei mit den nötigen Rechten bereitgestellt. Für Interessierte beschreiben wir in den folgenden Abschnitten, welche Schritte notwendig sind, um selbst Bluetooth-Admin-Rechte festzulegen.

7.4.1 APK-Datei auf den Computer laden und AppToMarket installieren

Um Bluetooth-Admin Rechte zu vergeben, müssen wir in der `AndroidManifest.xml` Datei folgende Zeile einfügen:

```
1   <uses-permission android:name="android.permission.BLUETOOTH_ADMIN" />
```

Das Hauptproblem ist hierbei, an diese Datei heranzukommen.

Zuerst laden wir unser Projekt, wie in Abschnitt 7.2 beschrieben, als `.apk`-Datei auf unseren Computer. Diese Datei müssen wir nun „in ihre Einzelteile“ zerlegen, um `AndroidManifest.xml` editieren zu können. Dazu benötigen wir das Tool `AppToMarket`. Dieses können Sie auf der folgenden Seite herunterladen:

`https://code.google.com/p/apptomarket/downloads/detail?name=AppToMarket_v32.zip`.

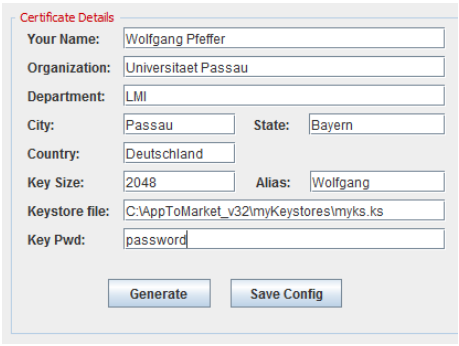
Bei `AppToMarket` handelt es sich um ein Java-Programm, für das keine weitere Installation notwendig ist. Wichtig ist, dass Sie den extrahierten Ordner im Verzeichnis `C:` ablegen.

7.4.2 APK-Datei mit AppToMarket editieren

Mit dem Programm AppToMarket können wir die `.apk`-Datei zunächst in einzelne Teile zerlegen, diese Teile editieren und anschließend wieder zu einer `.apk`-Datei zusammenfügen.
Dazu sind folgende Schritte nötig:

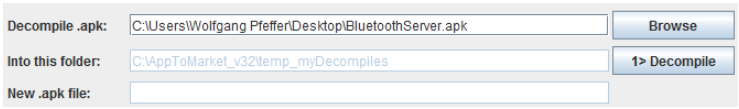
(1) Führen Sie die Jar-Datei `AppToMarket_v32` aus.

(2) Klicken Sie auf den Tab `1> Certificate Details` und füllen die Felder geeignet aus. Dies ist erforderlich, um später die Schritte `Sign` und `Zip Align` auszuführen.

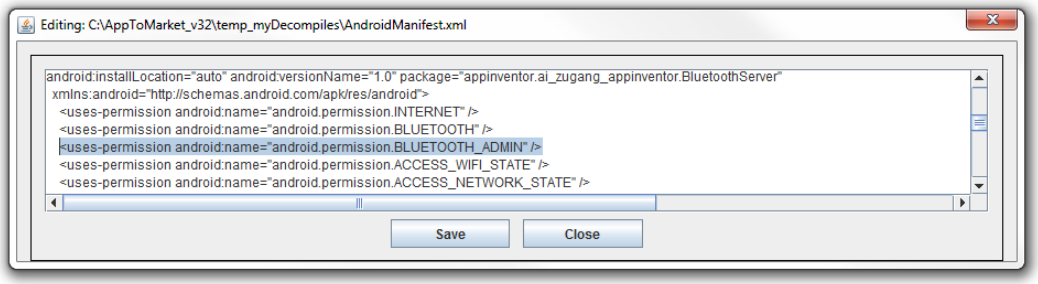


(3) Klicken Sie auf `Generate` und wechseln Sie anschließend in den Tab `2> De/Re compile and wrap up`.

(4) Wählen Sie die heruntergeladene `.apk`-Datei aus und klicken Sie auf `1> Decompile`.



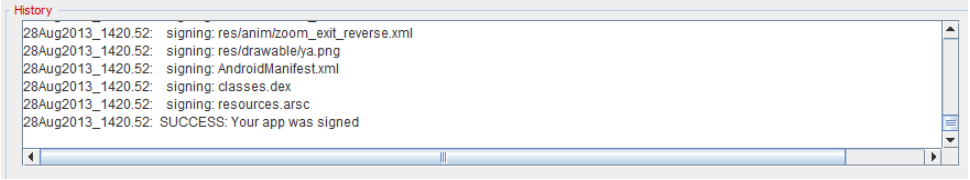
(5) Klicken Sie auf `2> Update Manifest` und anschließend auf `3> Edit Manifest` und fügen Sie oben abgebildete Zeile hinzu.



(6) Speichern Sie die Änderungen und schließen das Bearbeitungsfenster

(7) Klicken Sie auf `3> Create New .apk`. In der History wird angezeigt, wo die neue `.apk`-Datei gespeichert wurde.

(8) Klicken Sie auf `4> Sign`. In der History sollte folgender Text erscheinen:



(9) Klicken Sie auf `5> Verify` und anschließend auf `6> Zip Align`.

In Schritt (9) wurde in unserem Fall die Datei `BluetoothServer_new_zipAlign.apk` erstellt. Für uns ist ab jetzt nur noch diese Datei von Bedeutung.

7.4.3 APK-Datei auf dem Mobile Device installieren

Die neu erstellte `.apk`-Datei können wir nun mittels USB-Verbindung auf unser Mobile Device übertragen. Beachten Sie, dass sie ggf. bei der USB-Verbindung *Als Mediengerät angeschlossen* auswählen und auf Ihrem Mobile Device ein Dateimanager installiert ist (siehe ggf. Abschnitt 7.1.2).

Wichtig ist nun, dass Sie vor der Installation dieser `.apk`-Datei alle bisher installierten Apps des gleichen Programms deinstallieren. In unserem Fall müssen wir alle Bluetooth-Server Applikationen, die wir ggf. im Vorfeld schon einmal installiert haben, deinstallieren.

Sie können nun die `.apk`-Datei auf Ihrem Mobile Device installieren.[2] In unserem Fall hat unsere Bluetooth-App nun die benötigten Rechte und lässt sich ohne Fehlermeldung ausführen

Warum muss man die Applikation mit einem Zertifikat unterschreiben?

Das „Unterschreiben“ der Applikation beruht auf der Idee der *Digitalen Unterschriften*. Wir verweisen hier auf eine Lernaufgabe zu diesem Thema:

http://swisseduc.ch/informatik/daten/digitale_unterschriften/

Verändert man etwas an der App, muss man für diese Änderungen unterschreiben; man erklärt sich verantwortlich für diese App. Es kann niemand etwas an der App verändern, ohne seine eigene Unterschrift zu hinterlassen.

Ein hierzu verwandtes Thema ist *Public-Key-Kryptographie*. Zu dieser Thematik gibt es ein sehr schönes Kapitel aus dem Buch *Abenteuer Informatik* von Jens Gallenbacher, das kostenlos im Netz unter folgendem Link verfügbar ist:

http://www.springer.com/cda/content/document/cda_downloaddocument/Kapitel-9_inkl-Schablone+aus+2008_01_28_druckdaten-niedrig.pdf?SGWID=0-0-45-586798-p174325162

[2] Hier kann es zu Problemen mit den Sicherheitseinstellungen des Mobile Device kommen. Wir verweisen hier auf den entsprechenden Abschnitt auf Seite **??**.

7.5 Exkurs: Positionsbestimmung mit GPS

Wie der Name bereits vorwegnimmt, beschäftigen wir uns in diesem Abschnitt mit der Funktionsweise der Positionsbestimmung mittels GPS. Dieser Exkurs eignet sich gut vor oder nach der Bearbeitung der Aufgaben `Standortanzeiger` und / oder `Wegweiser`. In diesem Exkurs versuchen wir, den Schülern einen Einblick in die grundsätzliche Funktionsweise von GPS zu geben. Für weiterführende und detailliertere Beschreibungen verweisen wir auf die PDF-Datei

`http://zogg-jm.ch/Dateien/Update_Zogg_Deutsche_Version_Jan_09_Version_Z4x.pdf`

GPS ist heute jedem ein Begriff - nahezu jedes Smartphone besitzt mittlerweile eine Schaltfläche zum (De-)Aktivieren von GPS und oft fällt der Name beim Thema Navigation.
Doch wofür steht GPS überhaupt? Was leistet es und in welchem Zusammenhang steht es mit Satelliten?

Abbildung 7.5: Bis zu 31 Satelliten wurden in den Orbit gebracht, um den flächendeckenden Einsatz von GPS gewährleisten zu können.

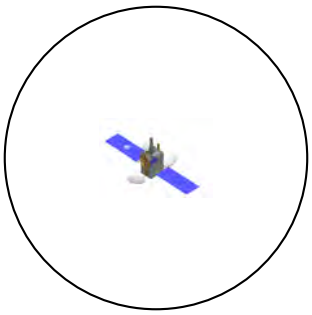
GPS steht für **Global Positioning System** und ist ein globales Satellitensystem, das zur Positionsbestimmung und Zeitmessung eingesetzt wird. Ursprünglich kam GPS nur im militärischen Bereich, etwa in Kampfflugzeugen, Kriegsschiffen und U-Booten, zum Einsatz. Heutzutage kommt es bekanntermaßen auch im zivilen Bereich zum Einsatz und wird meist mit dem Navigationssystem in Assoziation gesetzt.

Aber wie funktioniert GPS eigentlich? Woher weiß das Navigationssystem, wo ich aktuell bin?

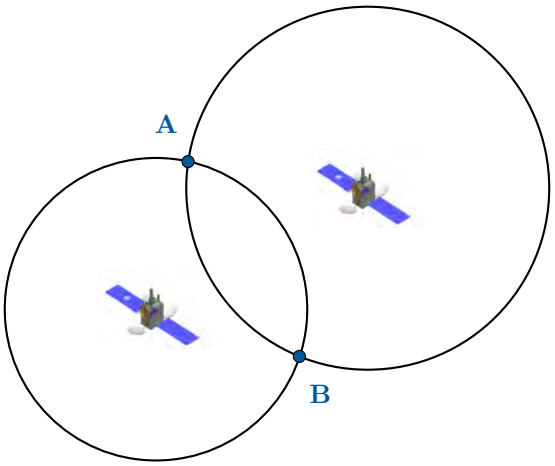
Hier möchten wir zunächst als Einstieg auf einen gut gemachten Beitrag zum Thema „Navigationssystem“ aus der Serie *Die Sendung mit der Maus* verweisen. Diesen findest du auf der Internetseite `http://www.wdrmaus.de/sachgeschichten/sachgeschichten/sachgeschichte.php5?id=2798`.
Falls du keine Internetverbindung hast, haben wir den Film auch als Video-Datei `maus_navi.avi` bereitgestellt. Frage ggf. bei deinem Betreuer nach dem Video.
Bei obigen Link erfährst du ab Minute 3 mehr zu unseren Fragen, bei dem mitgelieferten Video ca. ab 03:35.

Stellen wir noch einmal kurz vereinfacht die wichtigsten Erkenntnisse aus dem Film zusammen:

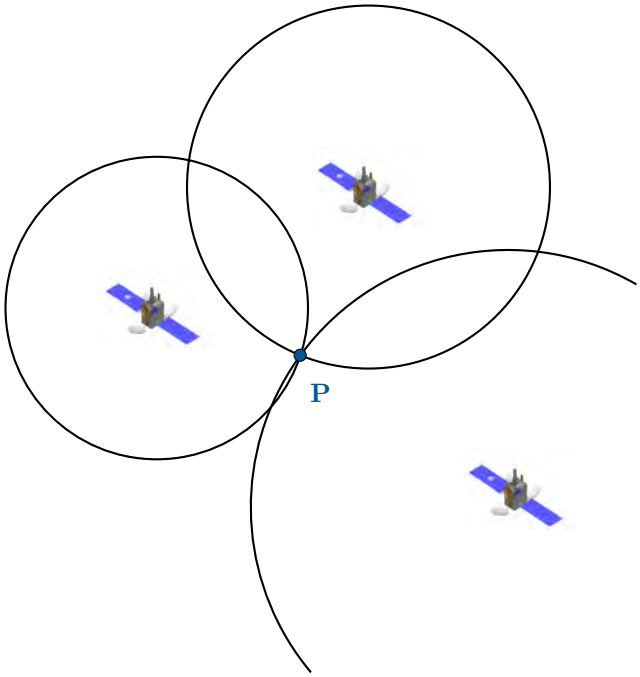
- Erhält ein Navigationssystem nur Signale von einem Satelliten, kann es daraus noch nicht seinen aktuellen Standort errechnen. Es gibt auf der Erde viele Punkte, die die gleiche Entfernung von diesem Satelliten haben. Diese Punkte liegen alle auf einer Kreislinie - dies haben wir im zweidimensionalen in untenstehender Abbildung noch einmal dargestellt:



- Erhält ein Navigationssystem Signale von zwei Satelliten, kann es daraus immer noch nicht seinen aktuellen Standort errechnen. Die Möglichkeiten für seine Position schränken sich allerdings auf zwei ein:



- Erhält ein Navigationssystem Signale von drei Satelliten, kann es daraus seinen eindeutigen Standort errechnen:



Bei dieser Vorgehensweise gibt es allerdings ein Problem, dass die Sendung mit der Maus nicht anspricht:

- **das GPS-Empfangsgerät hat keine mit dem Satelliten synchrone Atomuhr**

Warum ist das ein Problem?

- Signale werden mit Lichtgeschwindigkeit übertragen.
  Angenommen, die Uhr im GPS-Empfangsgerät geht nur 1 ms falsch, ergäbe sich eine um $0.001\,s \cdot 300000\,\frac{\text{km}}{s} = 300\,\textbf{km}$ falsche Entfernungsberechnung Standpunkt - Satellit.

  In diesem Fall wäre GPS völlig unbrauchbar!

Elegante Lösung:

- GPS verwendet die Information eines vierten Satelliten

  Die vier Satelliten besitzen jeweils eine (oder mehrere) Atomuhren und haben somit identische Zeit. Folglich ist der Zeitunterschied zwischen Empfänger und jedem Satelliten gleich groß - diesen Zeitunterschied bezeichnen wir mit $\Delta\, t_{\text{Fehler}}$. Geht die Empfängeruhr z. Bsp. 0,003 Sekunden vor, wird die Entfernung zu allen Satelliten um 900 Kilometer zu groß gemessen.
  Mit einem vierten Satelliten lässt sich dieser Fehler beheben.

  * Das GPS-Empfangsgerät kennt die Koordinaten $X_{\text{Sat}i}$, $Y_{\text{Sat}i}$, $Z_{\text{Sat}i}$ des jeweiligen Satelliten i
  * Satelliten senden Signale mit genauem Sendezeitpunkt t
  * GPS-Empfänger haben eine Uhr, die um $\Delta\, t_{\text{Fehler}}$ falsch geht
  * Der GPS-Empfänger muss von der gemessenen Laufzeit $\Delta\, t_{\text{gemessen}_i}$ den Fehler $\Delta\, t_{\text{Fehler}}$ abziehen
  * Die Entfernung zum Sendesatelliten kann somit wie folgt berechnet werden:

$$\left(\Delta\, t_{\text{gemessen}_i} - \Delta\, t_{\text{Fehler}}\right) \cdot c$$

  * Gesucht: Wo bin ich mit meinem GPS-Empfänger (X_0, Y_0, Z_0)?

Das GPS-Empfangsgerät hat mit den Werten von vier Satelliten ein System von vier Gleichungen mit vier Unbekannten zu lösen:

$$
\begin{aligned}
\left(\Delta t_{\text{gemessen}_1} - \Delta t_{\text{Fehler}}\right) \cdot c &= \sqrt{\left(X_{\text{Sat1}} - X_0\right)^2 + \left(Y_{\text{Sat1}} - Y_0\right)^2 + \left(Z_{\text{Sat1}} - Z_0\right)^2} \\
\left(\Delta t_{\text{gemessen}_2} - \Delta t_{\text{Fehler}}\right) \cdot c &= \sqrt{\left(X_{\text{Sat2}} - X_0\right)^2 + \left(Y_{\text{Sat2}} - Y_0\right)^2 + \left(Z_{\text{Sat2}} - Z_0\right)^2} \\
\left(\Delta t_{\text{gemessen}_3} - \Delta t_{\text{Fehler}}\right) \cdot c &= \sqrt{\left(X_{\text{Sat3}} - X_0\right)^2 + \left(Y_{\text{Sat3}} - Y_0\right)^2 + \left(Z_{\text{Sat3}} - Z_0\right)^2} \\
\left(\Delta t_{\text{gemessen}_4} - \Delta t_{\text{Fehler}}\right) \cdot c &= \sqrt{\left(X_{\text{Sat4}} - X_0\right)^2 + \left(Y_{\text{Sat4}} - Y_0\right)^2 + \left(Z_{\text{Sat4}} - Z_0\right)^2}
\end{aligned}
$$

Dieses Gleichungssystem ist eindeutig lösbar, was zur Synchronisation der Empfängeruhr mit den Atomuhren der Satelliten und somit zur genaueren Bestimmung der Position führt.

Quellenangaben und verwendete Literatur:

- Glühbirne: http://gluehbirne.ist.org/images/gluehbirne/500px-Dialog-information_on.svg.png
- Weltraum: http://www.tcs.ch/de/test-sicherheit/testberichte/assistenzsysteme-navigation/so-funktioniert-es.php
- Maus: http://www.wall-art.de/img/Die_Maus_neugierig.jpg
- Satellitenbild: OHB Technology GmbH
- Welt der Physik: http://www.weltderphysik.de/de/5946.php
- Zogg, Jean-Marie : Grundlagen der Ortung und Navigation mit Satelliten : http://zogg-jm.ch/Dateien/Update_Zogg_Deutsche_Version_Jan_09_Version_Z4x.pdf

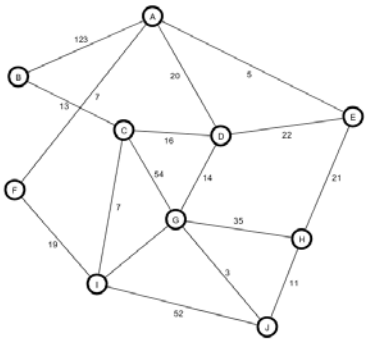
7.6 Exkurs: Kürzeste Wege - Der Dijkstra-Algorithmus

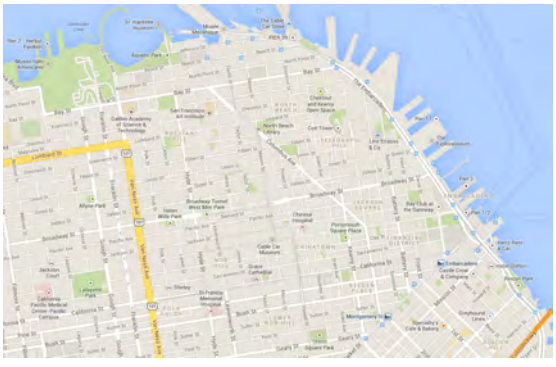
In diesem Exkurs beschäftigen wir uns mit dem Kürzesten Wege Problem und dessen Lösung mittels dem Dijkstra-Algorithmus. Dieser Exkurs eignet sich gut vor oder nach der Bearbeitung der `Wegweiser`-Aufgabe.
In seinem Buch „Abenteuer Informatik“ beschäftigt sich Jens Gallenbacher neben weiteren Themen ausführlich mit dieser Thematik - wir können sein Buch nur weiterempfehlen!

Auf nebenstehender Karte (erstellt mit Google Maps) ist ein Ausschnitt des Bayerischen Waldes zu sehen. Wir betrachten dazu folgendes Szenario:
Martin wohnt in Regen, seine Freunde Heinrich und Willi in Frauenau. Einmal in der Woche treffen sich die drei zum Skatabend. Martin überlegt nun, wie er am schnellsten nach Frauenau kommt.
Die Entfernung von Regen nach Zwiesel ist 9 km, von Regen nach Langdorf 4 km, von Regen nach Rinchnach 6 km, von Langdorf nach Zwiesel 7 km, von Rinchnach nach Zwiesel 8 km, von Rinchnach nach Frauenau 17 km und von Zwiesel nach Frauenau 7 km.

Welchen Weg wird Martin nehmen?

Wie bist du bei der Lösung der Aufgabe vorgegangen? Möglicherweise hast du auf die maßstabsgetreue Karte geschaut und geschlussfolgert, dass der Weg von Regen über Zwiesel nach Frauenau der Kürzeste ist. Ein anderer Lösungsweg wäre, alle Möglichkeiten durchzuprobieren.
Doch was ist, wenn die Karte nicht maßstabsgetreu ist? Was ist, wenn die Karte unübersichtlich wird?



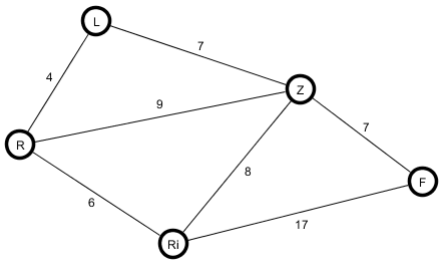


Alle Wege durchzuprobieren und uns anschließend für den kürzesten Weg zu entscheiden, wäre für uns bei größeren und komplexeren Karten (wie etwa der Stadtplan oben rechts) unmöglich - dies schaffen nicht einmal moderne Computer.

Unser Navigationssystem findet bekanntlich dennoch den kürzesten Weg.

Welche Strategie liegt hier zugrunde? Wie wird eine Karte vom Navigationssystem intern betrachtet? Welche Informationen der Karte sind für das Navigationssystem überhaupt relevant?

Das Navigationssystem arbeitet mit *Knoten* (das sind auf unserer Landkarte die Städte und Kreuzungen) und *Kanten* (Straßen). Zu jeder Kante speichert es ein *Gewicht* (hier: die Länge der Straße zwischen zwei Knoten). Vereinfacht dargestellt, wird die Karte aus unserem Einführungsbeispiel wie folgt intern repräsentiert:



Edsger Wybe Dijkstra
(* 1930 - † 2002)

Um nun den kürzesten Weg berechnen zu können, benutzt das Navigationssystem den `Dijkstra-Algorithmus`.

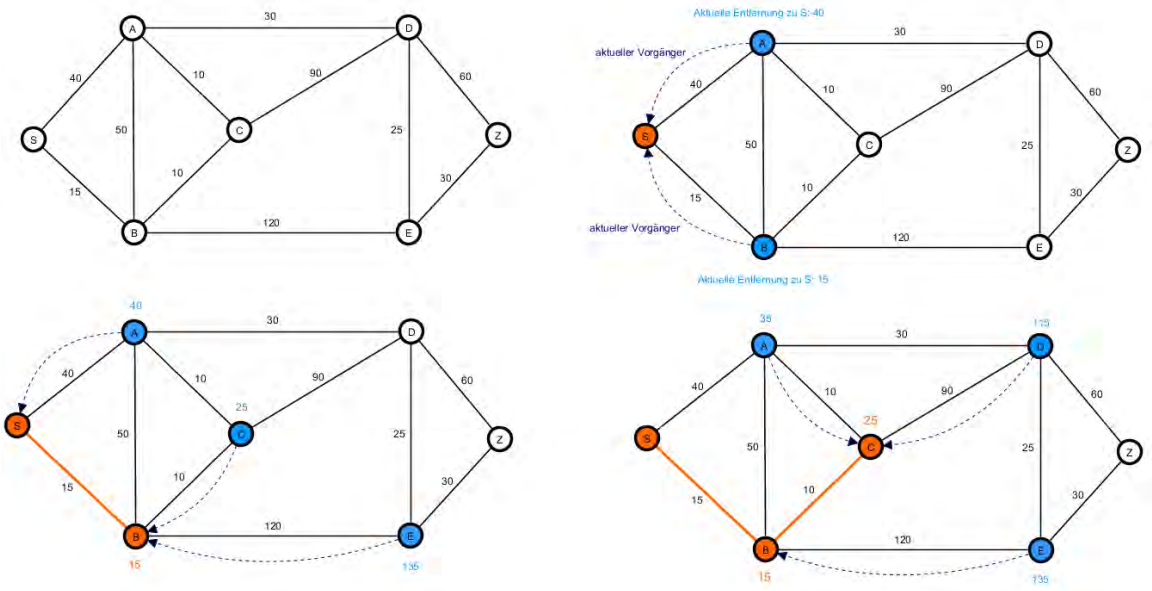
Klären wir vorab noch den Begriff `Algorithmus`:

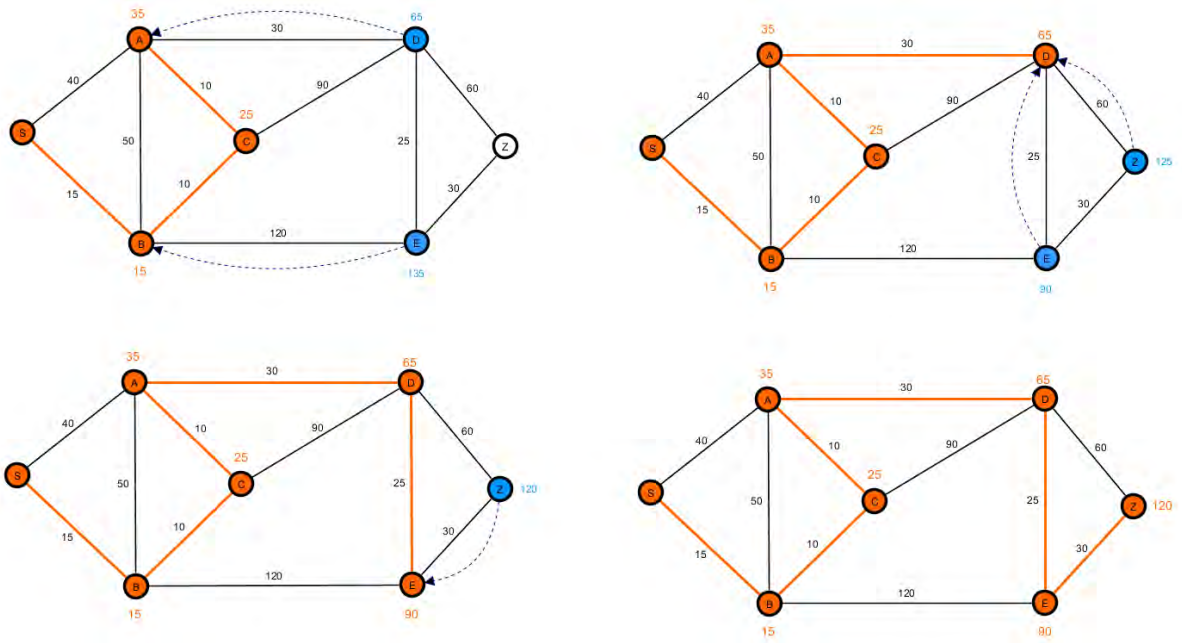
Algorithmus

Ein Algorithmus ist eine Handlungsvorschrift zur Lösung eines Problems bzw. einer Kategorie von Problemen. Diese Handlungsvorschriften lassen sich im Allgemeinen in ein Computerprogramm umsetzen. Hierfür muss sie hinreichend genau formuliert sein.

Beim Dijkstra-Algorithmus werden, beginnend bei einem Startknoten S, die kürzesten Wege zu allen anderen Knoten berechnet.

Im Folgenden haben wir die Funktionsweise des Dijkstra-Algorithmus an einem Beispiel schrittweise visualisiert. Versuche anhand dieses Beispiels Dijkstras Ideen mit eigenen Worten zu beschreiben - versuche auch eine Handlungsvorschrift / einen Algorithmus zu erstellen!





Vergleiche deine Beschreibung und deine Handlungsvorschrift mit Dijkstras-Algorithmus. Wir haben den Algorithmus leicht abgeändert, so dass er konform mit der Darstellung und den Farben unseres Beispiels ist!

Der Dijkstra-Algorithmus

(1) Auswahl eines beliebigen Startknotens S

(2) Schritte vorab:

- (a) Auswahl des Startknotens S (orange markiert)
- (b) Alle von S aus direkt erreichbaren Knoten werden gekennzeichnet (blau markiert)
- (c) Entfernung der blau-markierten Knoten von S wird gespeichert (blaue Zahlen)
- (d) Der aktuelle Vorgänger der blau-markierten Knoten wird auf S gesetzt

(3) Solange es noch nicht-fertig-abgearbeitete (nicht-orange) Knoten gibt, führe folgende Schritte aus:

- (a) Wähle den blauen Knoten aus, der die kleinste Entfernung zu S hat - wir bezeichnen diesen Knoten mit K
- (b) Kennzeichne alle unmarkierten, von K aus direkt erreichbaren Knoten (hellblau markiert)
- (c) Aktualisiere bei allen blau-markierten, von K aus erreichbaren Knoten die Entfernung zu S und den Vorgänger, falls sich über K ein kürzerer Weg ergibt
- (d) K ist fertig abgearbeitet (wird orange markiert)

Quellenangaben und verwendete Literatur:

- Glühbirne: http://gluehbirne.ist.org/images/gluehbirne/500px-Dialog-information_on.svg.png
- Stadtplan und Landkarte: Google Maps
- Abbildung Edsger W. Dijkstra: http://en.wikipedia.org/wiki/Edsger_W._Dijkstra
- Gallenbacher, Jens (^{3}2008) : Abenteuer Informatik. Spektrum Akademischer Verlag. Heidelberg.

Kapitel 8

Quellennachweise

In diesem Abschnitt sind die Quellen verwendeter Bilder angeführt, falls dies nicht bereits auf der jeweiligen Seite geschehen ist:

- Android Logo auf der Titelseite:

http://www.tagmotion.de/wp-content/uploads/2009/09/google_android_logo.jpg

- Lego Mindstorms NXT auf den Seiten 12 und 93 wurden erstellt mit

Lego Digital Designer

- Email Logo auf Seite 35:

http://thinkprogress.org/wp-content/uploads/2013/03/email.jpg

- QR-Codes auf den Seiten 2 und 43 wurden erstellt mit

http://www.qrcode-generator.de/newgen/?utm_expid=64850221-14&utm_referrer=https%3A%2F%2Fwww.google.com%2F

- Glühbirne auf den Seiten 15, 56, 59, 63, 42, 69 und 27:

http://gluehbirne.ist.org/images/gluehbirne/500px-Dialog-information_on.svg.png

Kapitel 9

Haftung für Links zu Webseiten

In diesem Mdoul sind Querverweise (Links) zu Webinhalten fremder Betreiber angegeben. Auf die Inhalte dieser Webseiten haben wir keinen Einfluss.

Bei der erstmaligen Angabe haben wir den fremden Inhalt daraufhin überprüft, ob durch ihn eine mögliche zivilrechtliche oder strafrechtliche Verantwortlichkeit ausgelöst wird.
Rechtswidrige Inhalte waren zum Zeitpunkt der Verlinkung nicht erkennbar.

Trotz sorgfältiger inhaltlicher Kontrolle übernehmen wir keine Haftung für die Inhalte externer Links.
Für den Inhalt der verlinkten Seiten sind ausschließlich deren Betreiber verantwortlich.

Passau, im August 2016

Ute Heuer, Wolfgang Pfeffer