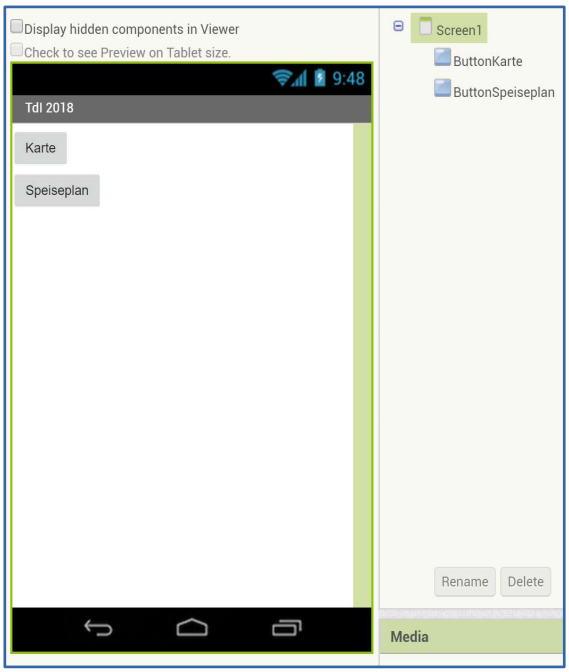
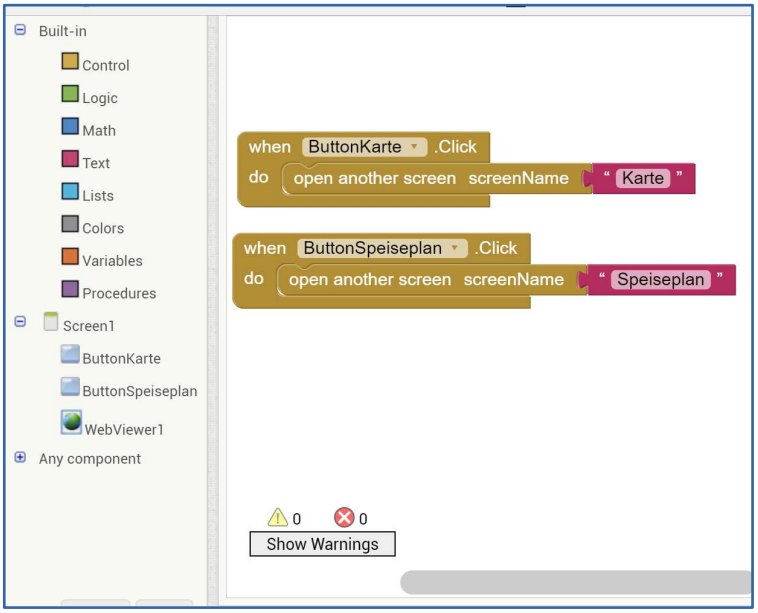
Gehen Sie zu http://ai2.appinventor.mit.edu/ und melden Sie sich an – zum Beispiel mit Ihren Google-Account. Legen Sie ein neues Projekt an, der Name ist egal.

Aufgabe 1: Erste einfache Screen-Navigation.

Beschreibung: Hier legen Sie eine einfache Navigation für eine App an, um von der Startseite aus die zwei anderen Screens erreichen zu können

- Legen Sie 3 Screens an namens Screen1, Speiseplan, Karte. Obacht: Die Namen können später nicht mehr geändert werden.
- Screen1: Platzieren Sie dort zwei Buttons, deren Beschriftung „Speiseplan“ und „Karte“ lauten. Das geht einfach mit Drag & Drop im Designer-Modus von Screen1.
- Screen 1: Die Objektbezeichnet für die Buttons lauten automatisch Button1 und Button2, ändern Sie das auf ButtonSpeiseplan und ButtonKarte.
- Screen 1: Legen Sie im *Blocks*-Modus fest, was bei dem Event „when ButtonKarte.Click“ geschehen soll, nämlich: es soll zum entsprechenden anderen Screen gewechselt werden. Die Bausteine dazu finden Sie im Menü.

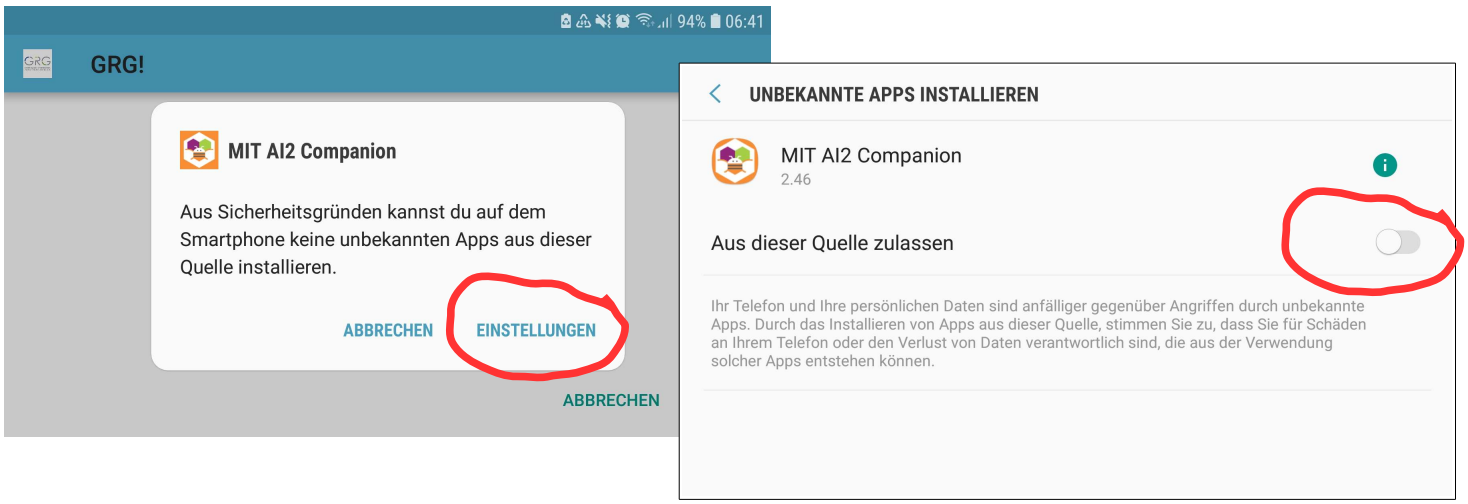




Aufgabe 2: Übertragung auf das Smartphone

Beschreibung: Sie lassen den MIT-Server die App compilieren und scannen den entstandenen QR.-Code ein, um sie auf Ihr Smaprtphone zu kriegen. Dazu müssen Sie zuerst die Begleit-App MIT AI2 Companion installieren.

- Installieren Sie die App MIT AI2 Companion aus dem Google Play Store (eventuell bereits zu Hause geschehen).
- Wählen Sie im Browser im App Inventor „Build App (provide QR code for .apk)“, woraufhin der MIT-Server Ihr Projekt compiliert und eine .apk-Datei erzeugt. Danach wird ein QR-Code angezeigt, den Sie mit der App MIT AI2 Companion einscannen.
- Standardmäßig kann man unter Android nur Apps aus dem Play Store installieren. Die Companion-App wird Sie beim ersten Einsatz fragen, ob Sie auch Apps aus anderen Quellen zulassen wollen, das müssen Sie bestätigen. Je nach Android-Version sieht das etwas anders aus, zum Beispiel so:



- Dann wird die neue App installiert,
  und zwar unter dem Namen ddes Projektes, alternativ unter dem Namen, der im Designer-Modus unter „Title“ angegeben ist.
- Testen Sie Ihre App, indem Sie mit den Buttons zu den beiden (noch leeren) Screens Speiseplan und Karte navigieren. Dort gibt es zwar noch keine weiteren Buttons zur Navigation, aber mit der Zurück-Taste Ihres Smartphones können Sie wieder zurück zu Screen 1 springen.

Alternative: Auf dem Rechner **App Inventor Setup/ MIT_Appinventor_Tools_2.3.0** (~80 MB) herunterladen und als Admin installieren: http://appinventor.mit.edu/explore/ai2/windows.html
Dann können Sie a) den Emulator verwenden und b) die .apk-Datei per Kabel auf Ihr Handy übertragen statt über QR-Code/Internet.

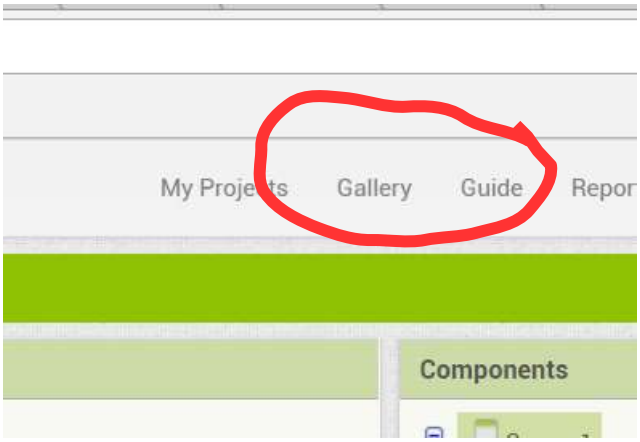
.aia und .apk

Sie können Ihr Projekt als compilierte .apk-Datei auf einem Smartphone installieren, oder als .aia-Datei exportieren – das ist der Quellcode, den der App Inventor 2 verwendet. Sie können ebenfalls .aia-Projekte von anderen importieren, zum Beispiel aus der Galerie.

Aufgabe 3: Ein Projekt importieren

Beschreibung: Sie laden eine .aia-Datei mit einem Spiel aus der AI2-Galerie auf Ihr Handy.

- Gehen Sie zur Gallery, suchen Sie nach „snakegame_ai2“ und laden Sie eine Kopie davon zu Ihren Projekten. Übertragen Sie das Projekt dann wie in Aufgabe 2 auf Ihr Smartphone.



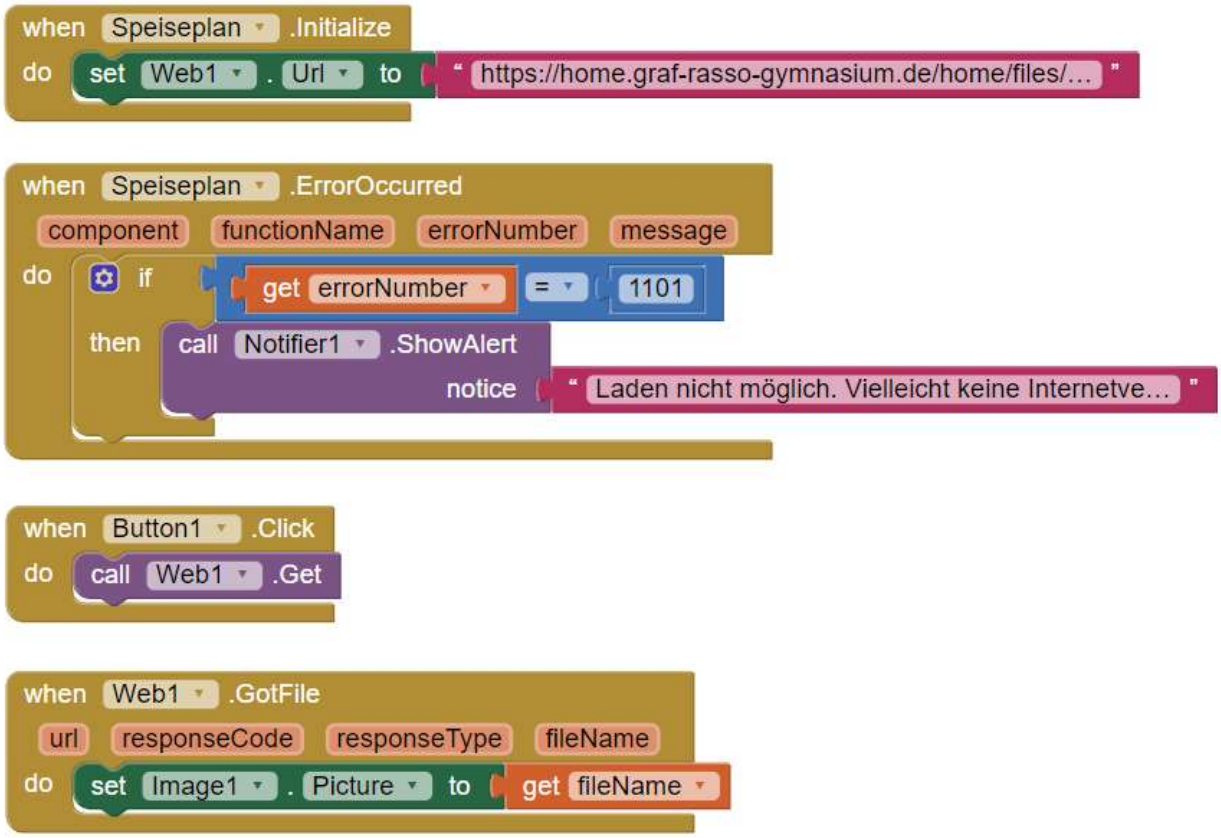
- Alternativ und für später: Suchen Sie in der Gallery nach „TdI2018“ - dort finden Sie die fertigen Lösungen für die Projekte in diesem Workshop

Aufgabe 4: Ein Bild aus dem Web laden (Speiseplan)

Beschreibung: Der Screen „Speiseplan“ erhält einen Knopf, klickt man darauf, wird von einer Webadresse ein Bild geladen und angezeigt.

- Screen Speiseplan: Legen Sie im Designer-Modus an
  - **1 Button** (im Menü User Interface), darunter
  - **1 Image** (im Menü User Interface) – **wichtig: setzen Sie die Attributwerte für Height und Width jeweils auf „Fill parent...“**
  - **1 Notifier** (im Menü User Interface), ein nicht sichbares Element, das Sie irgendwohin ziehen können, es wird dann unterhalb des Bildschirms angezeigt
  - **1 Web** (im Menü Connectivity), ebenfalls nicht sichtbar, steht für eine Web-Ressource – **wichtig: setzen Sie unbedingt das Häkchen bei „SaveResponse“,** nur dann wird das Ergebnis auch gespeichert.

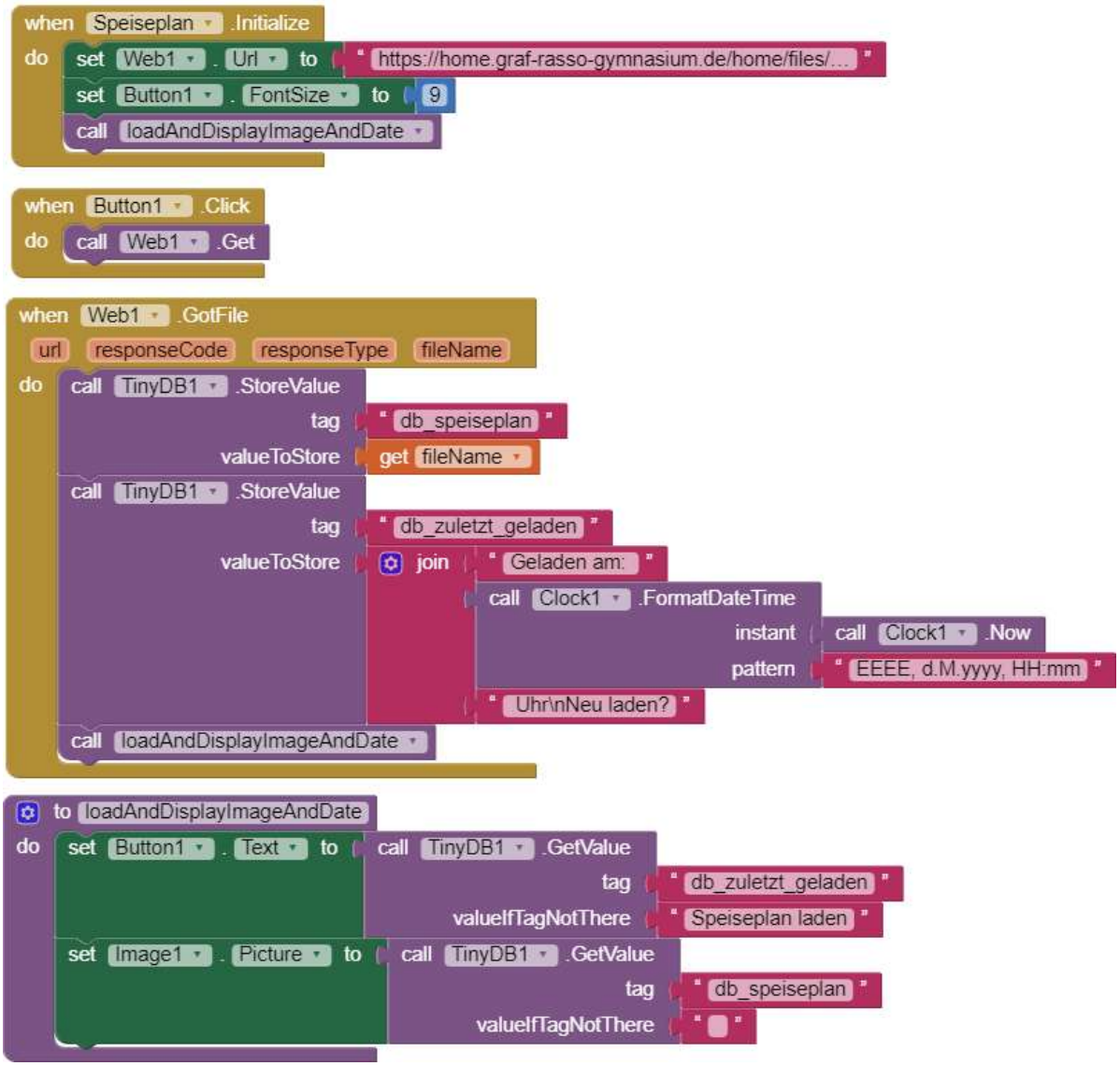
- Screen Speiseplan: Bauen Sie im Blocks-Modus folgende Blöcke nach:
  - Der erste Block legt die URL für die Webressource fest. Das könnte man auch bei den Attributen im Designer-Modus machen. Die Adresse lautet: https://home.graf-rasso-gymnasium.de/home/files/mensa/speiseplan.jpg
  - Der zweite Block ist optional.
  - Der dritte Block besagt, dass beim Klick auf den Knopf das Web-Objekt aufgerufen wird, die Ressourde auch tatsächlich zu laden.
  - Der vierte Block besagt, dass bei einem erfolgreichen Laden der Ressource diese dem Bildobjekt als Wert des Picture-Attributs zugewiesen wird.



Aufgabe 5: Speiseplan mit Bildspeicherung

Beschreibung: Wird die App neu gestartet oder auch nur der Screen neu initialisiert, verschwindet wird das geladene Bild nicht angezeigt, obwohl es sogar noch auf dem Handy gespeichert ist.
Lösung: Das Bild (bzw. der Pfad dazu) wird nach dem Laden in einer Datenbank gespeichert und beim Starten des Screens von dieser Daten aufgerufen.
Neu ist die Datenbank, die screenübergreifen wie eine Hashmap mit Strings als Key funktioniert.

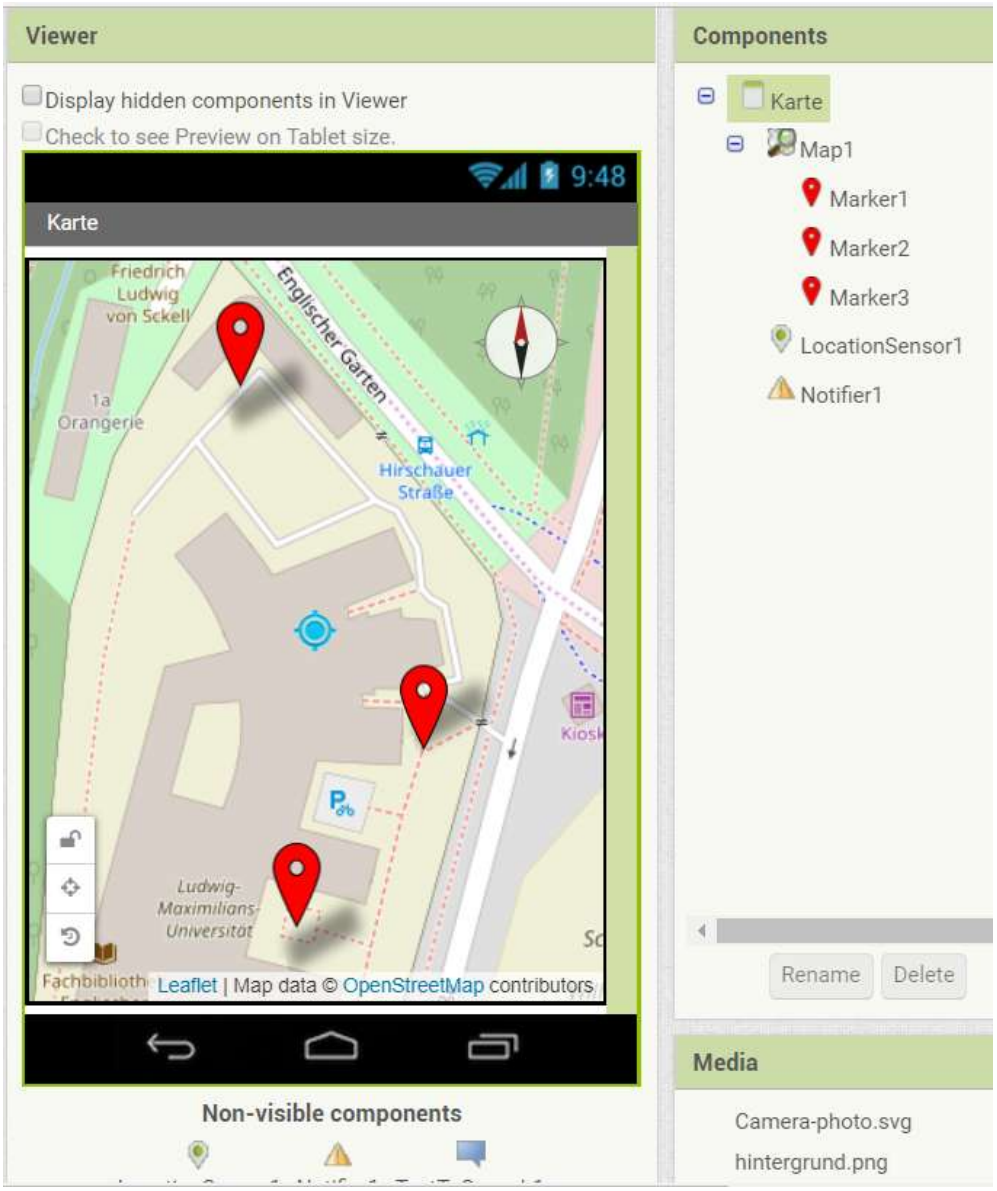
- Screen Speiseplan: Ergänzen Sie im Designer-Modus
  - **1 Clock** (im Menü Sensors), ein nicht sichbares Element und
  - **1 TinyDB** (im Menü Storage), ebenso
- Im Blocks-Modus werden jetzt zwei Blöcke erweitert und ein dritter neu angelegt:
  - die Methode loadAndDisplayImageAndDate setzt das Bild und den Buttontext auf die in der Datenbank gespeicherten Werte
  - im Initialisierungsblock wird diese Methode aufgerufen
  - beim Event „Web1.GotFile“ geschieht am meisten: das Bild wird in der DB gespeichert, das aktuelle Datum wird in der DB gspeichert, die Methode loadAndDisplayImageAndDate wird aufgerufen



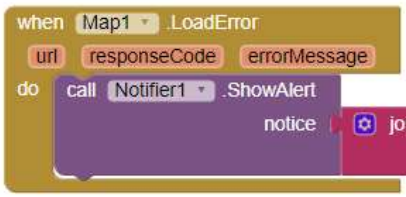
Aufgabe 6: Eine Karte mit Markern, die erscheinen

Beschreibung: Auf einer Karte sind drei Marker verteilt, nähert man sich diesen auf eine bestimmte Distanz, werden sie sichtbar.

- Screen Karte: Fügen Sie im Designermodus eine **Map** hinzu (Menü Maps), setzen Sie die Height und Width auf „Fill parent...“ und platzieren Sie drei Marker (Menü Maps) darauf.
- Ebenso 1 **Location Sensor** (Menü Sensors) und 1 **Notifier** (Menü User Interface).
- Setzen Sie für dieMarker-Attribute „Tiitle“ und „Description“ sinnvolle Werte
- Setzen Sie für den LocationSensor „DistanceInterval“ auf 1 (das heißt, ein Event wird ausgelöst, wenn sich die Position um 1 Meter geändert hat.)
- Setzen Sie für den LocationSensor „TimeInterval“ auf 1 (das heißt, ein Event wird ausgelöst, wenn sich die Zeit um 1000 Millisekunden geändert hat.)
- Alternativ: Laden Sie das Beispielprojekt.



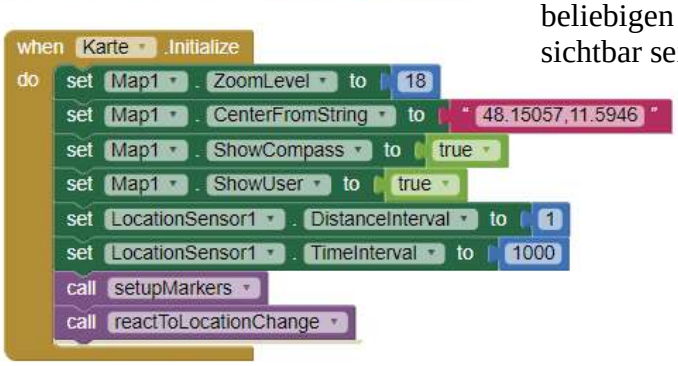
Bauen Sie den Quellcode auf der folgenden Seite nach, oder übernehmen Sie ihn aus dem Beispielprojekt.



optional, gibt Fehlermeldung aus, wenn etwas nicht funktioniert

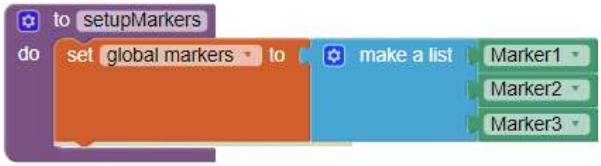
Deklaration einer globalen Variablen, initialisiert auf leere Liste – wird später gefüllt mit den Markern

Deklaration einer globalen Variablen, initialisiert auf einen beliebigen hohen Wert – ab dieser Distanz sollen Marker sichtbar sein

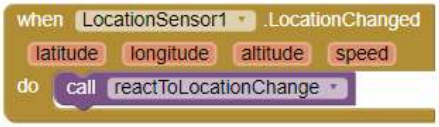


Initialisierung von verschiedenen Attributen auf sinnvolle Werte, kann auch im Designer-Modus vorgenommen werden

Aufruf einer Methode, um die Marker zur Liste hinzuzufügen
Aufruf der zentralen Methode, die am hier am Anfang und später bei jeder Positionsänderung aufgerufen wird



Im Moment recht leere Methode, die die Marker in eine leere Liste setzt, später aber erweitert werden kann.



Wenn das Event „Position geändert“ eintritt, wird die entsprechende Methode aufgerufen

für alle Marker in der Liste wird überprüft, ob ihre Entfernung von der aktuellen Position kleiner als die Sichtbarkeitsdistanz ist, wenn ja, wird der Marker auf sichtbar gesetzt, sonst auf unsichtbar. Siehe Java-Äquivalent unten. (Und ja, das ginge noch kürzer.)

```
void reactToLocationChange() {
  for (Marker currentMarker : globalMarkers) {
    if (currentMarker.distanceToPoint(locationSensor1.getLatitude(), locationSensor1.getLongitude()) <
sichtbarkeitsradius) {
      currentMarker.setVisible(true);
    } else {
      currentMarker.setVisible(false);
    }
  }
}
```

Programmieraufgaben ohne Lösung

Das Schüttelorakel: Man stellt dem Handy eine Frage, schüttelt es, und danach erscheint zufällig ein „Nein“, „Ja“, „Vielleicht“ un so weiter. Optional: Die Antwort wird auch laut gesagt.

- Designer-Modus:
  - 1 Accelerometer (Menü: Sensors)
  - 1 Label (Menü: User Interace, beliebige Größe usw)
  - Optional: 1 Text-to-Speech (Menü: Media)
- Blocks-Modus:
  - Wenn der Accelerometer geschüttelt wird, dann wird aus einer Liste von mindestens vier Strings, die Varianten von „Ja“, „Nein“, „Vielleicht“ enthalten, ein zufälliges Element ausgesucht (dafür gibt es eine Listenmethode) und auf den Labeltext geschrieben
    - Optional: auch noch vom Text-to-Speech-Objekt laut sagen lassen – dann muss das zufällige Element allerdings in einer lokalen Variablen gespeichert werden

Eine Bildergalerie

- Design-Modus:
  - 1 horizontales Layout (Menü: Layout; Height & Width: „Fill Parent...“), darin
  - 1 Button
  - 1 Image (Height und Width: „Fill Parent...“)
  - 1 Button
  - Hochladen von zwei, drei, vier Bildern
- Blocks-Modus:
  - Variable anlegen und als gefüllte Liste initialisieren; die Liste besteht aus Strings mit den Namen der hochgeladenen Bilder
  - Eine zweite Variable (Name z.B. „index“) anlegen und als Zahl 1 initialisieren
  - Den Image-Wert des Bilds mit dem vom index angegebenen Listenelement füllen
  - Die zwei Knöpfe benutzen, um den Index zu erhöhen und zu verringern, und jeweils das aktualisierte Bild anzeigen

Alternative: Statt der Buttons zwei Canvas-Objekte verwenden (Menü: Drawing and Animation). Darauf kann man zeichnen, aber vor allem erkennen sie auch Wischgesten. Das Event heißt „when Canvas.Flung“.

Man kann auch auf Knöpfe und Image verzichten und stattdessen nur 1 großen Canvas benutzen. Das Canvas.Flung-Event stellt als Parameter unter anderem „speed“ und „heading“ (Richtung) zur Verfügung:

- Wischen nach rechts: Betrag(heading) <45
- Wischen nach links: Betrag(heading)>135
- Wischen nach oben: heading zwischen +45 und +135
- Wischen nach unten: heading zwischen -45 und -135

Damit kann man dann bestimmen, welches Bild als Nächstes erscheinen soll.

Schnitzeljagd: Man lädt von einem Webserver eine Datei mit (mehreren?) Koordinaten, speichert diese, zeigt jeweils die erste an, und muss dorthin gehen, umd das nächste Koordinatenpaar zu erhalten.