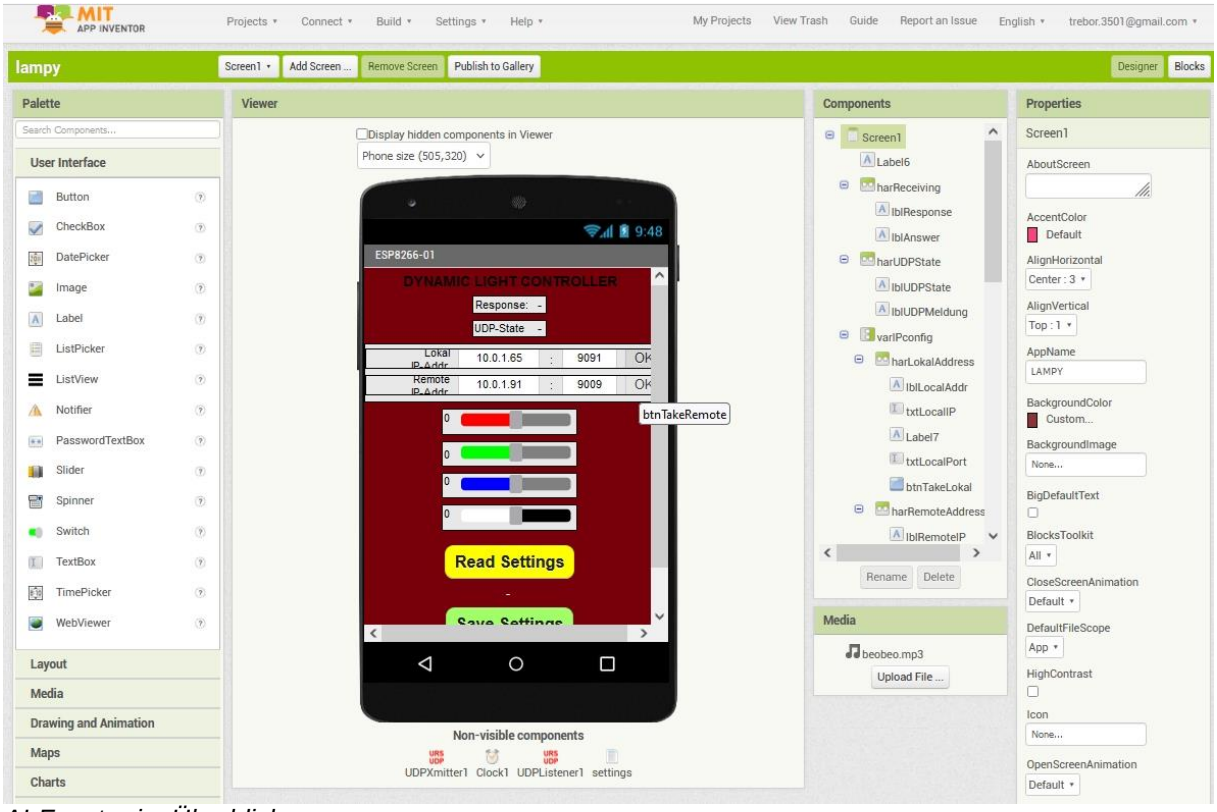

AI-Fenster im Überblick

Diesen Beitrag gibt es auch als PDF-Dokument.

So sieht der fertige Entwurf für die App zur Decolampe im MIT App-Inventor 2 (AI2) aus. AI2 ist eine Anwendung, die im Browser läuft. Dort stellen Sie die App-Oberfläche mit den Werkzeugen in der linken Spalte, der **Palette**, zusammen. Die Elemente werden einfach in den **Viewer** gezogen und dort an der Zielposition abgesetzt. In **Components** erscheinen dann die vorläufigen Bezeichner der Objekte, die Sie am besten gleich mit sprechenden Namen ersetzen, das geht über den Button **Rename**. Ich habe mir dafür ein System ausgedacht, das sowohl den Typ als auch die Aufgabe der Objekte verrät. Das erleichtert später beim Zusammenbau des Programms das Auffinden und Zuordnen. Drei Kleinbuchstaben kennzeichnen den Typ. Der erste Buchstabe des darauffolgenden Namens wird großgeschrieben. In der **Properties**-Spalte können die Eigenschaften der Objekte eingestellt werden. Da ich schon öfter Apps für meine Projekte erstellt habe, und nicht jedes Mal knapp 20 Seiten Basisanleitung in den Post mit einbauen möchte, habe ich das alles in das Dokument mit app_Inventor_2_ger.pdf hineingepackt. Dort erfahren Sie, wie sie mit AI2 arbeiten, woher Sie die App AI2-Companion für das Handy und die UDP-Erweiterung von Ullis Roboterpage bekommen. Wenn Sie bisher noch nichts mit dem AI2 gebastelt haben, lege ich Ihnen dringend nahe, diese Anleitung durchzuarbeiten.

Gut – dann starten wir durch mit

MicroPython auf dem ESP32 und ESP8266

heute

Eine App für die Tischlampe

Die Anleitung für den Bau und die Programmierung der Lampe finden Sie in diesem Beitrag dekolampe1_ger.pdf.

Den AI Companion bekommen Sie über den Google Play Store. Haben Sie den AI Companion auf dem Handy installiert? Dann starten Sie ihn jetzt bitte. Im AI2-Fenster verbinden Sie Handy und PC mit einem Klick auf **Connect** – **AI Companion**. Scannen Sie den QR-Code, kurz darauf können Sie alles, was an Design-Aktionen auf dem PC läuft, auf dem Handy in Echtzeit mitverfolgen.

Die App-Oberfläche

Starten Sie gleich mit dem AI2 in einem Browser Ihrer Wahl. Melden Sie sich mit Ihrem Google-Konto an und klicken Sie auf den Button **Create apps**. Das Meldefenster quittieren Sie mit **I accept the terms of service**. Das Fenster **Welcome to App Inventor** schließen Sie mit **continue**. Die Tutorial-Angebote übergehen Sie jetzt besser, sonst kommen Sie die nächste Zeit nicht dazu, ihre Lampen-App fertigzustellen. Klicken Sie auf den freien Hintergrund des AI2-Fensters und dann auf **Start new project** ganz links oben. Der Name des Projekts ist **lampy** oder Sie es nennen wollen – OK.

Abbildung 1: Neues Projekt starten

Sie landen im **Designer** mit den vier Spalten **Palette**, **Viewer**, **Components** und **Properties**. Die Eigenschaften von Screen1, der Name wurde nicht verändert, weil es nur einen Screen gibt, habe ich wie folgt gewählt.
AppName: Lampy
BackgroundColor: Custom #78000bcc
Scollable: Haken
Title: Lamp controller

So soll der fertige Entwurf aussehen, den wir jetzt Stück für Stück zusammenstellen werden.

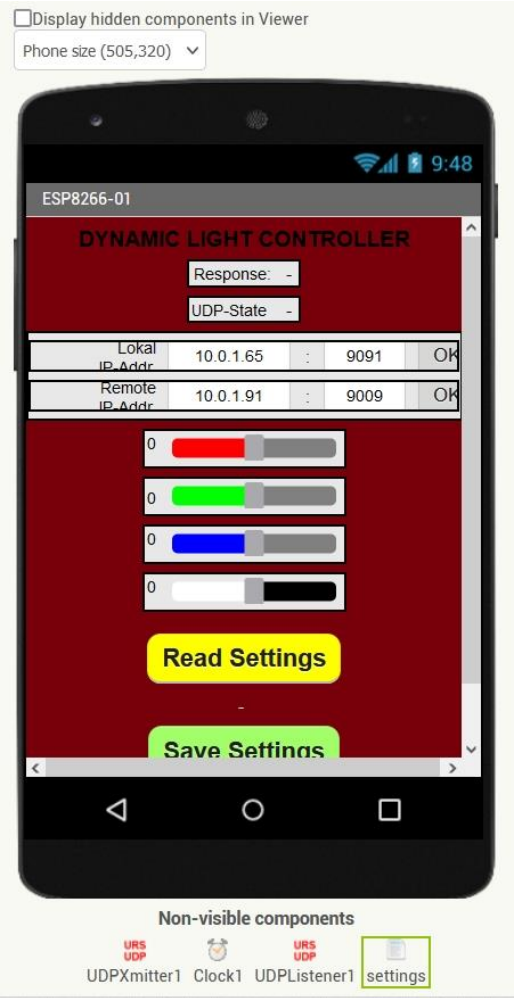

Abbildung 2: Entwurf im Viewer

1. Für die Überschrift holen wir ein Label aus dem **User interface**, das wir in **lblHeadline** umtaufen.

FontSize: 18
Width: Fill parent
Text: Dynamic Light Controller
TextAlignment: center

2. Damit in der zweiten Zeile die beiden geplanten Labels nebeneinander zu liegen kommen, sperren wir sie in ein **HorizontalArrangement**, das den Namen **harReceiving** erhält. Das Arrangement finden Sie in **Palette.Layout**.

In harReceiving ziehen wir zwei Labels. Das erste heißt **lblResponse**, das zweite **lblAnswer**. Text wird auf **Response:** und "**–"** gesetzt.

3. Den Schritt 2. wiederholen wir noch einmal mit:

harUDPState
lblUDPState: UDPstate:
lblUDPState: -

Die nächsten beiden Zeilen erhalten als Käfig ein VerticalArrangement, **varIPConfig** in welchem zwei HorizontalArrangements, **harLocalAddress** und **harRemoteAddress**, platziert werden.

varIPConfig
AlignHorizontal: Left
AlignVertical: Top
Width: Fill parent

harLocalAddress, **harRemoteAddress**
AlignHorizontal: Center
AlignVertical: Center
Width: Fill parent

Das Innenleben der HorizontalArrangements zeigt Abbildung 2.

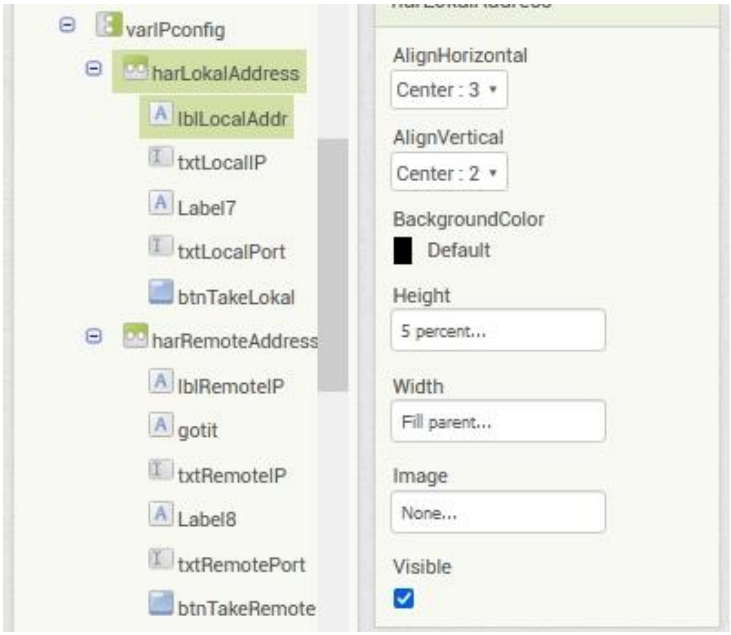

Abbildung 3: IP-Konfiguration

4. Jetzt folgen vier Sliders, also Schieberegler, welche die Farbintensität für rot, grün und blau beeinflussen sowie die Gesamthelligkeit, Sie sind zusammen mit einem führenden Label in HorizontalArrangements verpackt. Am Beispiel rot sieht das so aus.

harRot:
AlignHorizontal: Center

lblRed:
FontSize: 14
Text: 0
TextAlignment: left

sldRed:
ColorLeft: Red
ColorRight: Default
MaxValue: 100
MinValue: 0
ThumbPosition: 30

Für grün und blau wird lediglich ColorLeft entsprechend angepasst. **harLuminosity** ist analog aufgebaut, mit drei Änderungen beim Slider.

sldRed:
ColorLeft: White
ColorRight: Black
MaxValue: 255
MinValue: 0
ThumbPosition: 30

5. Für Alarmmeldungen folgt als Nächstes ein nacktes Label, **lblAlarm,**
FontSize: 14.

6. Die Einstellungen können in einer Datei gespeichert werden. Der Button **btnRead** initiiert den Lesevorgang.

Backgroundcolor: Yellow
Shape: Rounded
Text: Read Settings

Eín Label **lblRead** schafft zum einen Platz zwischen den Buttons und informiert über den Lesevorgang.

FontSize: 14
Text: -
TextColor: White

Der Schritt 6 wir für das Speichern analog wiederholt.

7. Aus **Palette.Extension** holen wir **UDPXmitter1** und **UDPListener1** in den Viewer. Die Elemente werden nicht auf dem Screen, sondern darunter angeordnet.

8. Den Timer **Clock1** holen wir aus **Palette.Sensors**. Auch der Timer wird unter dem Screen abgelegt, ebenso wie das File-Objekt aus **Palette.Storage**, das wir in **settings** umtaufen. Wir erteilen **ReadPermission** und **WritePermission**.

Damit ist das Einrichten der Oberfläche abgeschlossen, und wir gehen an den Zusammenbau der einzelnen Funktionalitäten.

Blockiade

Wir wechseln in den Blockeditor mit Klick auf **Blocks** ganz rechts oben im Fenster des AI2. Dort finden wir in der Spalte Blocks eine Reihe eingebauter Blockschubladen wie **control**, **Logic**, **Math** und so weiter. Darunter kommen in der gleichen Reihenfolge wie in **Designer.Components** unsere Objekte von der Bedienoberfläche. Klicken Sie auf eines der Elemente, damit die zur Verfügung stehenden Operationen angezeigt werden. Von hier zieht man sie auf die Fläche des Viewers.

Gleich ein Tipp im Voraus. Wenn Elemente mehrfach gebraucht werden, kann man mit Rechtsklick auf das Element das Kontextmenü aufrufen und das Element mit Duplicate kopieren.

Wie bei einem MicroPython-Programm beginnen wir mit der Deklaration der globalen Variablen, die wir aus **Blocks.Variables** holen. Die Wertzuweisungen stammen aus Math, Text, Lists, und UDPXmitter1.

Abbildung 4: Variablen deklarieren

Abbildung 5: Konstanten zuweisen

Klinken Sie jetzt den Zahlen-Block in den Initialisierungs-Block ein und geben Sie der Variablen den Namen **max**. Die weiteren Deklarationen laufen ähnlich. Alternativ bietet sich die Methode Duplizieren an, Variable neu benennen, fertig.

Abbildung 6: Duplizieren geht schneller

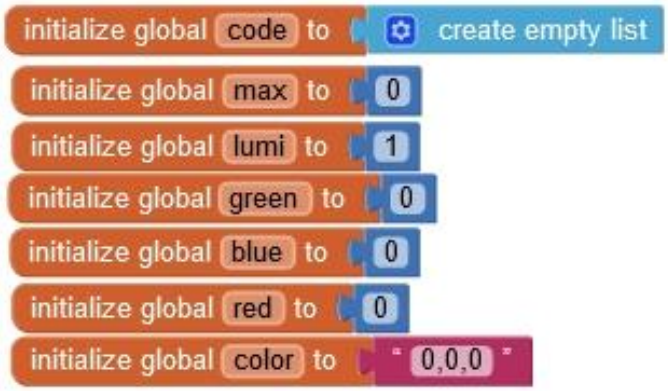

Abbildung 7: Farben deklarieren

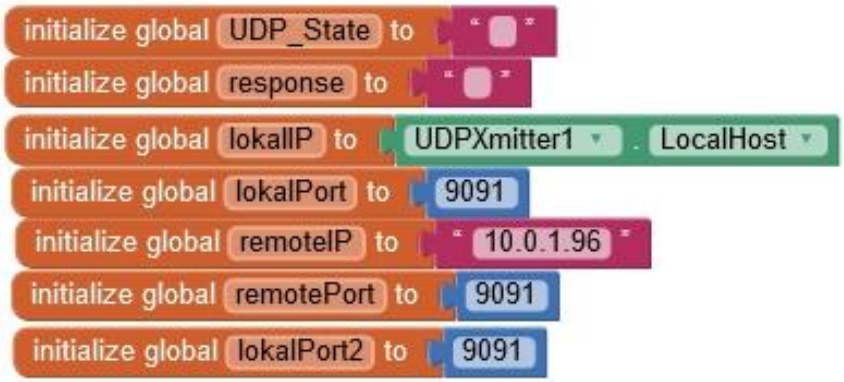

Abbildung 8: Netzwerk-Variablen deklarieren

Alle Strukturen und Anweisungen, die aus einem, der von uns im Designer deklarierten Objekte abgeleitet werden, haben als "Vornamen" den Namen des Objekts. Der "Nachname", nach dem Punkt, beschreibt die Aktion.

Abbildung 9: Zur Nomenklatur

sldRed findet man in **Blocks.sldRed**. Markiert man den Block, dann klappt die Auswahl an möglichen Aktionen auf, in der man **ThumbPosition** findet. Analog findet man **Text** im Auswahlmenü von **Blocks.lblRed**.

Das kleine Dreieck, rechts neben dem Namen, listet die Bezeichner der deklarierten Blocks, beziehungsweise die Bezeichner für mögliche Aktionen auf. Dadurch ist es leicht möglich, über Strg + C und Strg + V Blöcke zu vervielfältigen und dann die Bezeichner anzupassen. Das geht schneller als alles über die Blocks-Spalte zu setzen und funktioniert auch für zusammengesetzte Blöcke.

Was bei der Arduino-IDE in **setup** passiert, wird bei AI2 in den Block **when Screen1 initialize** verpackt, die Vorbereitungen zum Programmlauf. Das ist hier eine ganze Menge. Der **when**-Block stammt natürlich aus **Blocks.Screen1**. Der UDP-Client wird angehalten, um die Konfiguration vorzunehmen, dann wird er neu gestartet. Dann werden Labels und Textfelder aktualisiert, **txtLocalIP**, **txtLocalPort** und **lblUDPMeldung**. Der **if**-Block kommt aus **Blocks.Control**. Wir setzen und starten den Timer. Die Slider werden auf 90% der Screen-Breite gesetzt und deren Höhe auf 10%, letzteres schafft mehr Platz zwischen den Schiebern. Die Anfangspositionen gehen auf 32, 8 und 0.

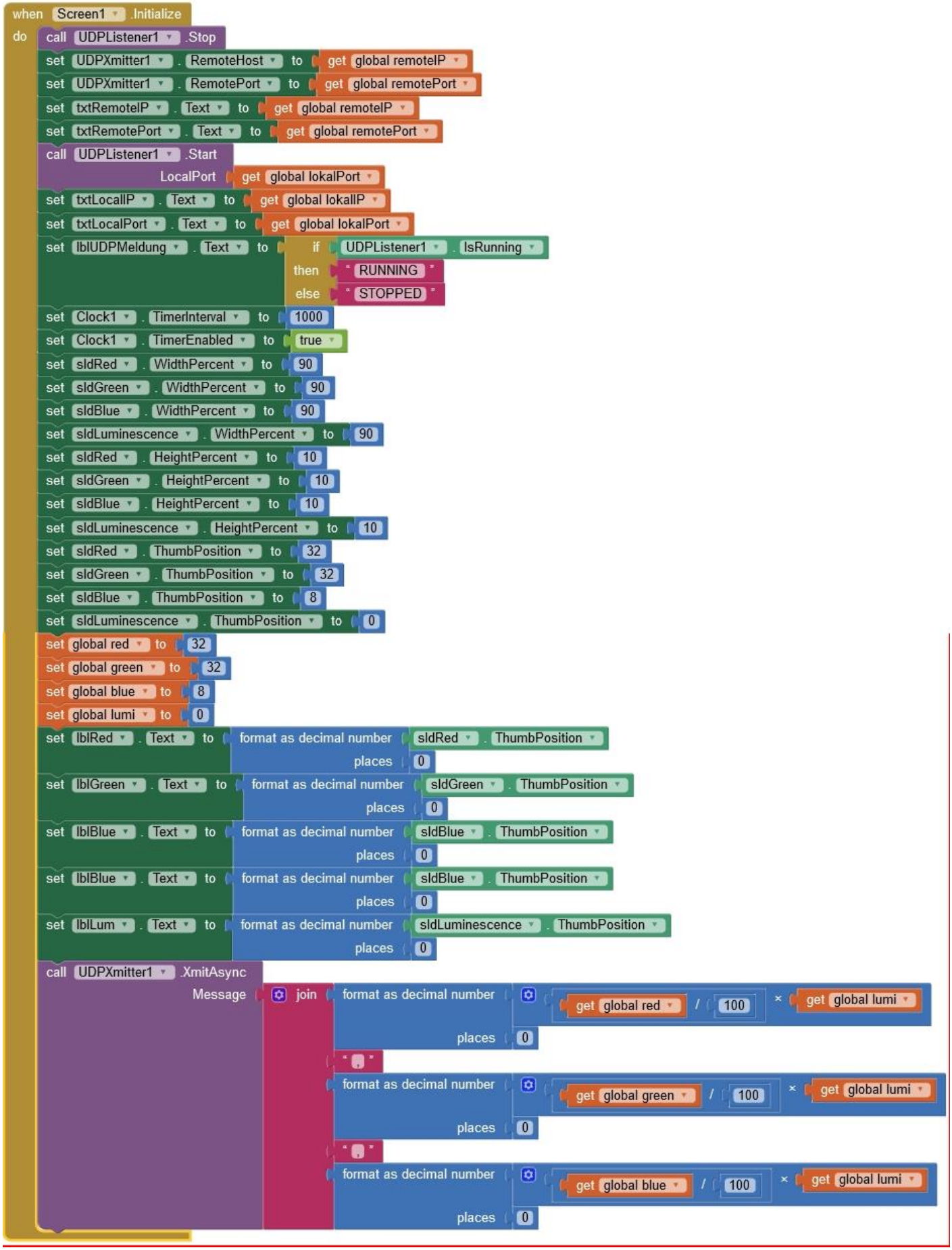

Abbildung 10: when-Scree1-initialize

Die globalen Variablen werden angeglichen, ebenso die Labels in den horizontalen Arrangements der Regler. Abschließend senden wir den zusammengesetzten Steuer-String an den Controller. Die Werte ergeben sich aus den Stellungen der Regler. Wir teilen den Stand durch den Maximalwert 100 und multiplizieren mit dem Stand des Lumineszenz-Reglers. Das Ergebnis runden wir auf eine Ganzzahl. Die blauen Blöcke kommen aus **Blocks.Math**, **join** kommt aus **Blocks.Text**.

Der Timer hat die Aufgabe den Zustand des UDP-Clients zu überwachen und den Status zu melden. Außerdem muss er sich nach Ablauf neu starten.

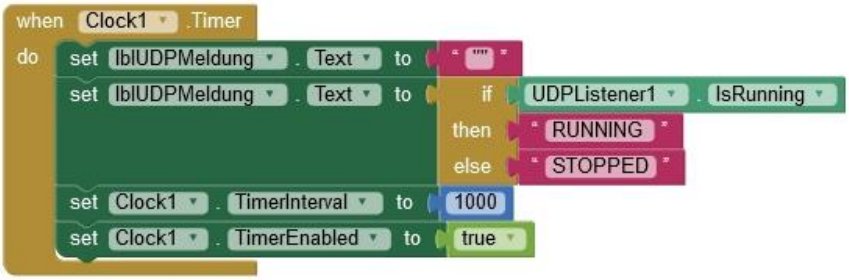

Abbildung 11: when-Clock1-Timer

Wurde die Ziel-IP beziehungsweise die entfernte Portnummer geändert, muss die Änderung mit Tippen auf **OK** übernommen werden. Das mach der Block **btnTakeRemote**.

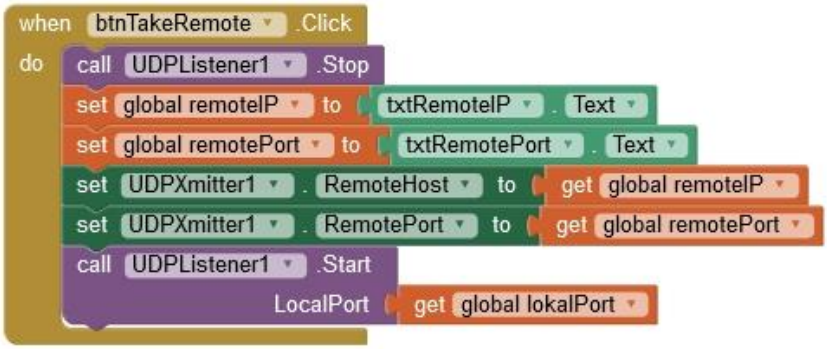

Abbildung 12: Button-TakeRemote-Click

Analog läuft das mit der lokalen Portnummer.

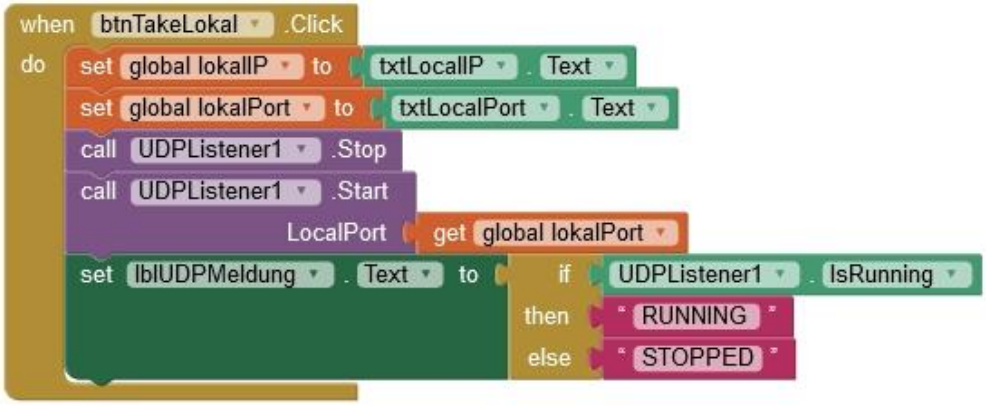

Abbildung 13: Button-TakeLocal-Click

Sollte ein Fehler in der UDP-Einheit auftreten, dann unterrichtet uns der nächste Block darüber.

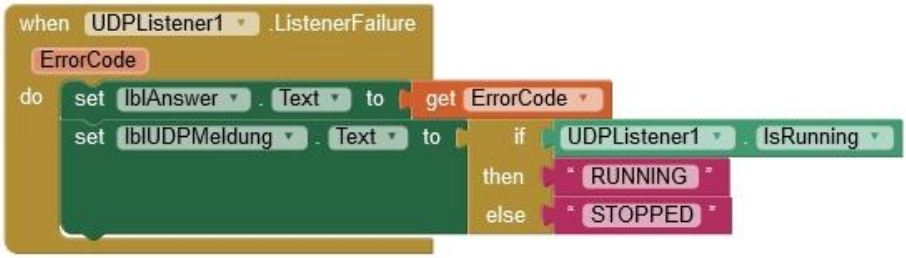

Abbildung 14: when-UDPListener1-Failure

Immer dann, wenn sich die Position eines Reglers verändert hat, muss der neue Wert verarbeitet und an den ESP8266 gesendet werden. Wir holen den Wert, schreiben ihn als Ganzzahl ins Label, setzen den String zusammen und senden ihn an den Controller. Für grün und blau geht das analog. Die when-Blöcke mit Füllung habe ich mit Duplizieren vervielfacht und nur die beiden set-Anweisungen angepasst.

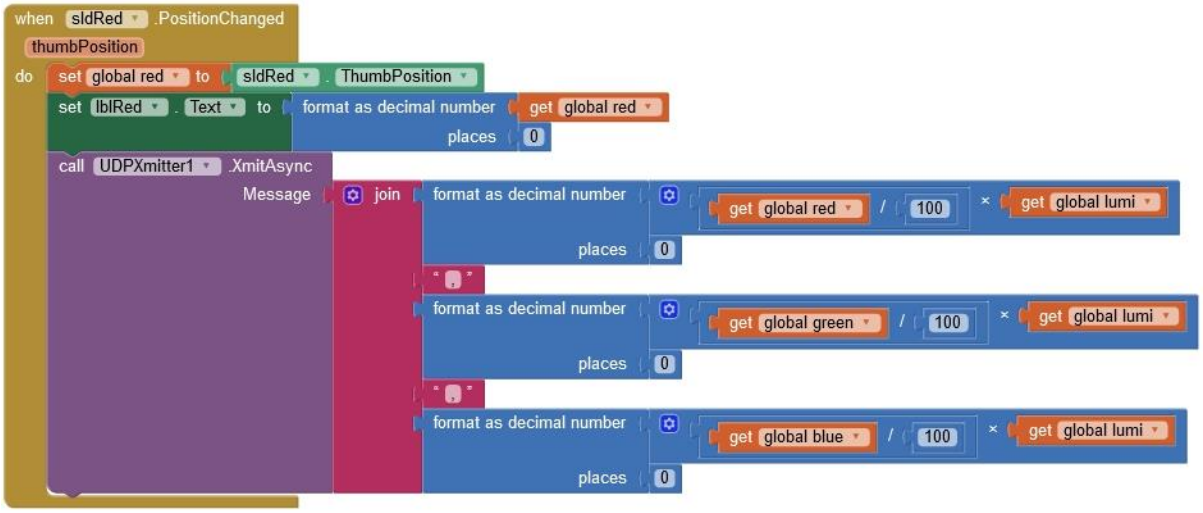

Abbildung 15: Schieber-slideRed

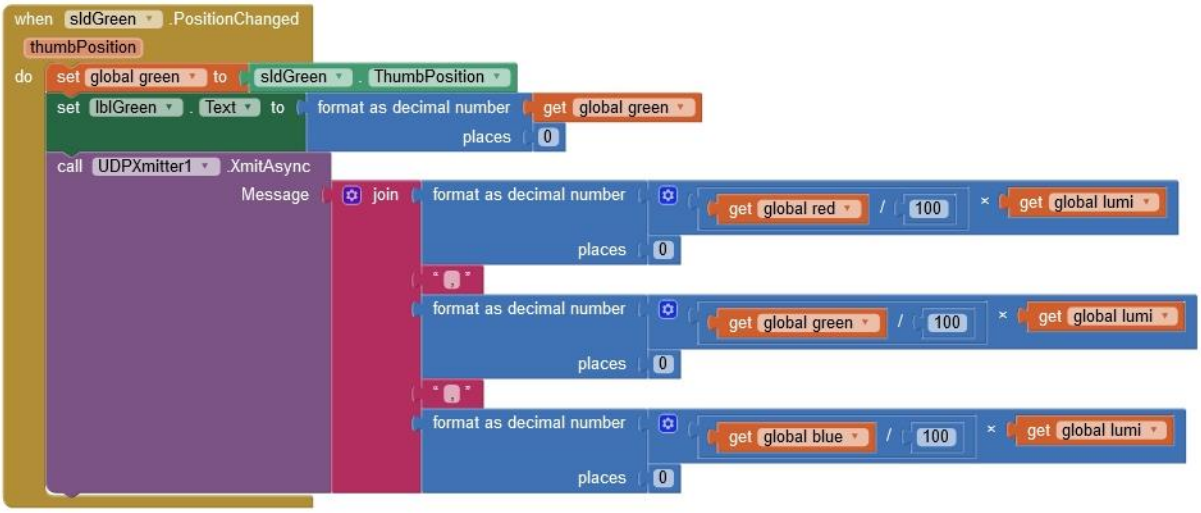

Abbildung 16: Schieber-slideGreen

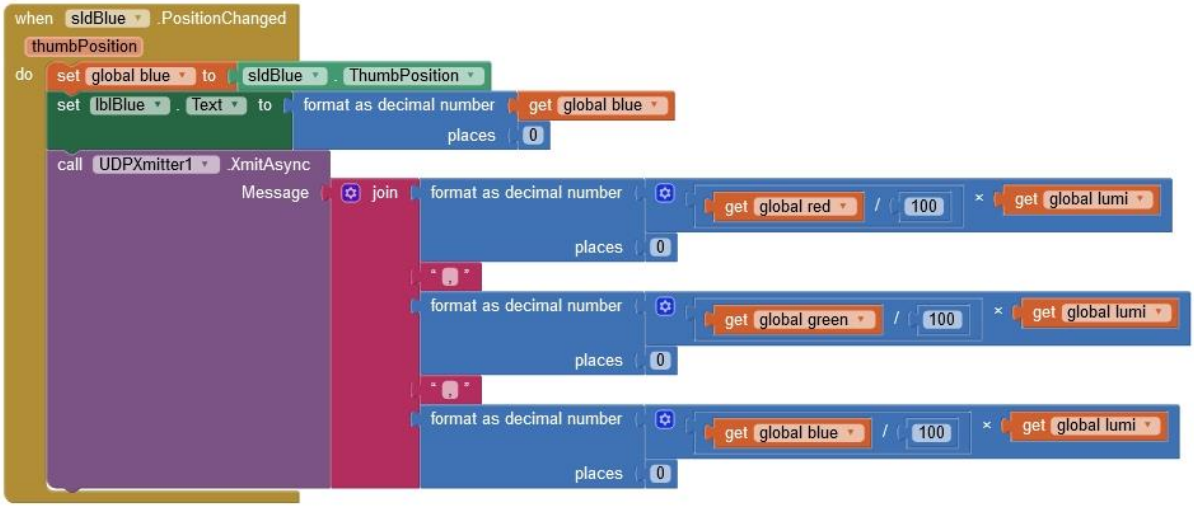

Abbildung 17: Schieber-slideBlue

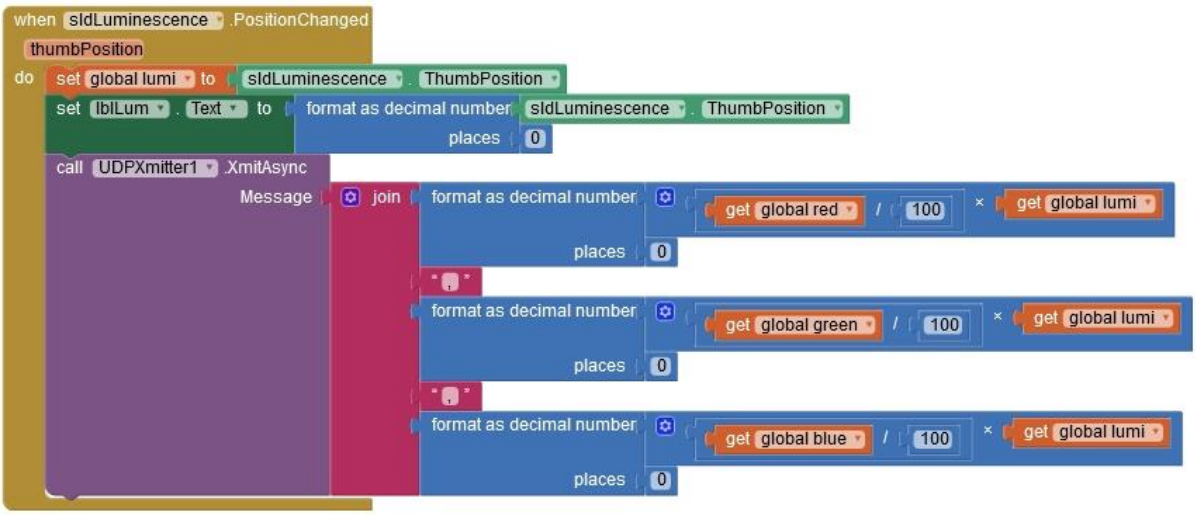

Abbildung 18: Schieber-slideLuminescence

Der Button Read Settings stößt das Auslesen der Datei **lightsettings.txt** an.

Abbildung 19: when-ButtonRead-click

Anschließend kann der Text an den Kommas in einzelne Strings aufgeteilt werden. Das Ergebnis ist, wie in MicroPython eine Liste, die wir der Listen-Variablen **code** übergeben. Wir lesen die Elemente der Liste aus und weisen die Werte den globalen Variablen für Farbe und Helligkeit zu. Die Indizierung von Listen läuft hier ab 1, nicht wie in MicroPython ab 0.

Wir aktualisieren die Labels und die Schieberstellungen und senden den zusammengesetzten Steuer-String an den ESP8266. Die Blöcke **split** und **join** stammen aus **Blocks.Text**, **select list item** aus **Blocks.Lists**. **format decimal number** und die Rechenoperatoren wohnen in **Blocks.Math**.

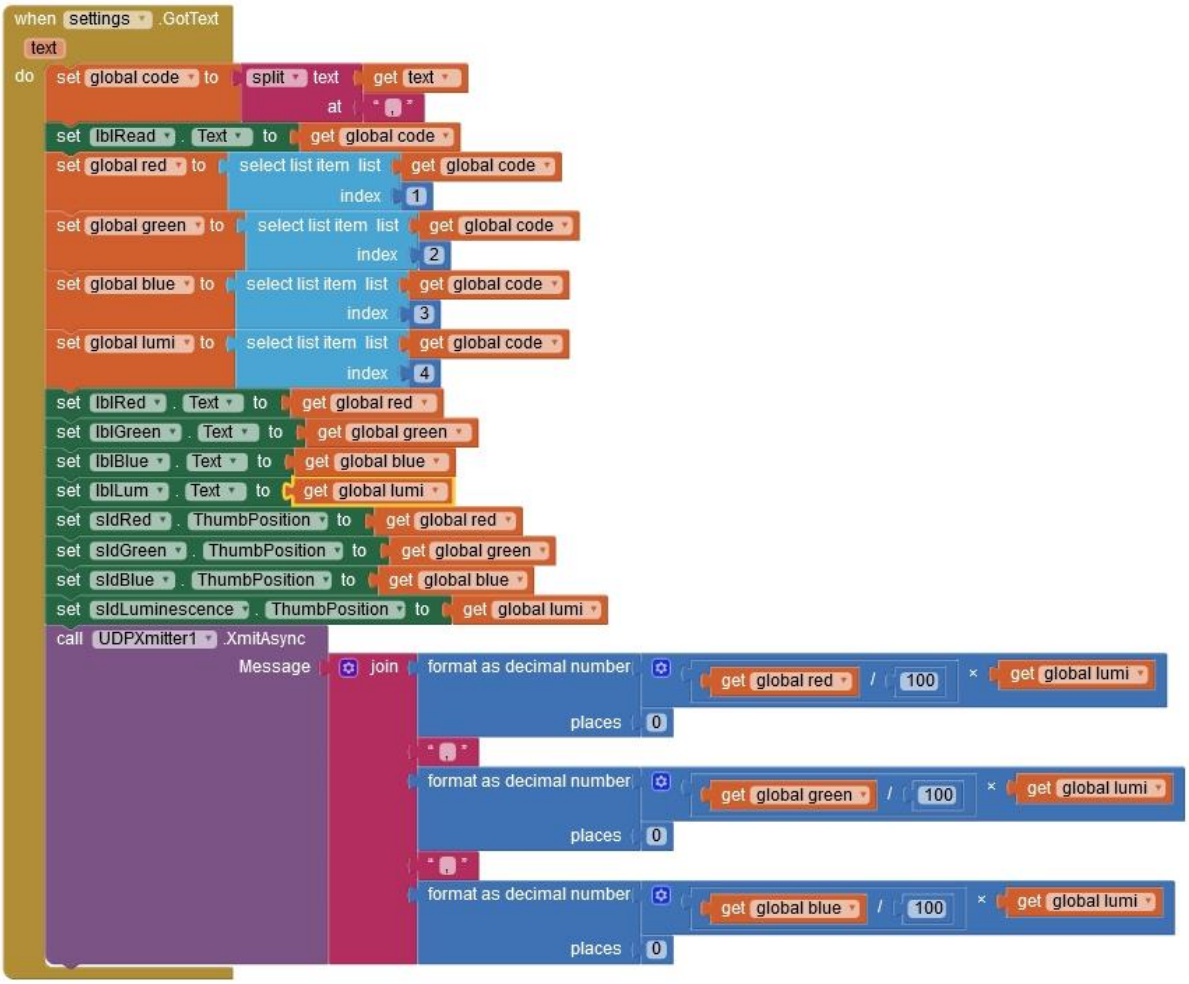

Abbildung 20: when settings-GotText

Zum Speichern der Einstellungen dient der Button Save Settings. Wir setzen den String zusammen und speichern ihn in der Datei **lightsettings.txt**, die im Speicher-Kontext der App wohnt.

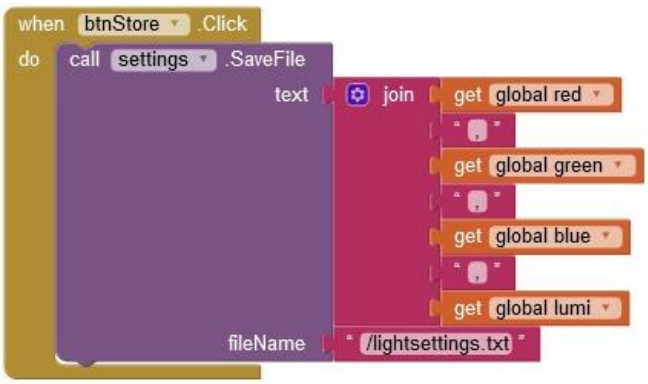

Abbildung 21: when-ButtonStore-click

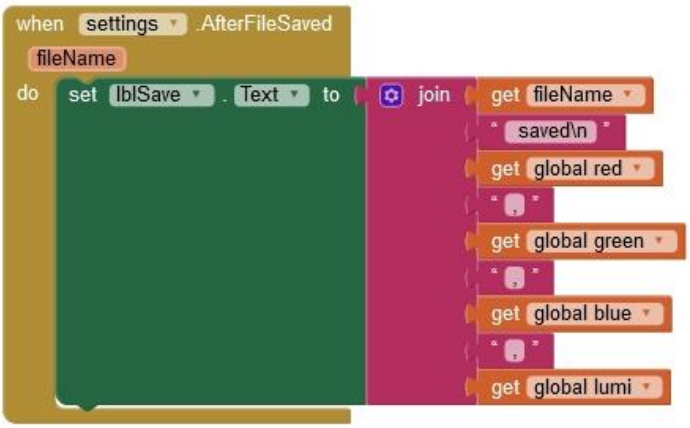

Abbildung 22: settings-after File Saved

Nachdem der String gespeichert wurde, wird der Text im Label **lblSave** angezeigt.

Wenn nun alles zusammengebaut ist, geben wir den Auftrag, unser Werk zu compilieren. Im AI2-Fenster klicken wir aus **Build – Android App (apk)**. Der Fortschritt wird angezeigt und am Ende poppt ein Fenster auf.

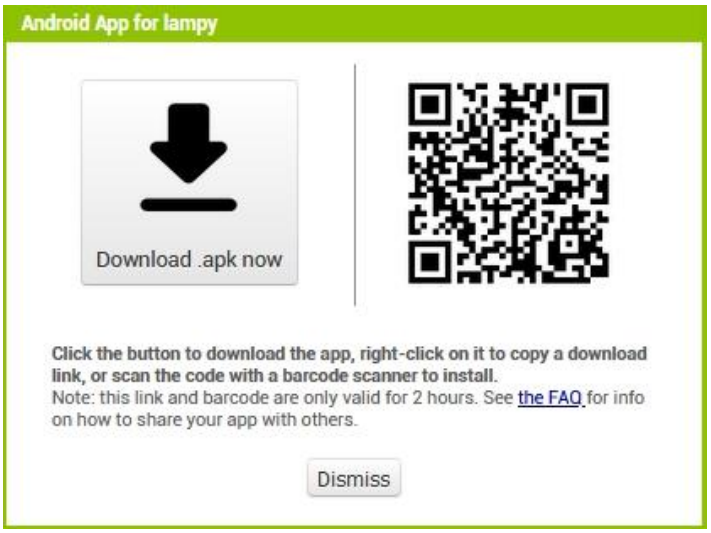

Abbildung 23: App herunterladen

Sie können nun die apk-Datei auf ihrem PC speichern oder/und über den QR-Code direkt aufs Handy übertragen und dort ausführen. Damit starten Sie die Installation von **lampy.apk**. Die App kann nun ohne den AI Companion ausgeführt werden.

Damit auch das MicroPython-Programm auf dem ESP8266 autonom läuft, öffnen wir das Programm **tischlampe.py** in Thonny und schieben es mit **File – Save as…** oder einfach **Strg + Shift + S** als **main.py** zum ESP8266 (MicoPython device) hoch.

Starten Sie jetzt die Lampe. Während die Verbindung zum Router aufgebaut wird, blinkt die LED 0 im Sekundentakt grün. Jetzt starten Sie die App. Die LED hört auf zu blinken, wenn die Verbindung steht, und Sie können jetzt mit den Reglern am Handy Farbe und Gesamthelligkeit steuern.

Für ganz Eilige gibt es hier die Projektdatei lampy.aia zum Download. Vergessen Sie aber nicht, IP und Portnummern an Ihr WLAN anzupassen. Das gleiche gilt natürlich auch für das MicroPython-Programm.

Die Projektdatei speichern Sie in einem beliebigen Verzeichnis. Im AI2 können Sie diese über **Projects – Import Project (aia) from my computer …** öffnen und nach den Änderungen compilieren.

Viel Spaß mit lampy!