# {code}.racer

…the story continues!

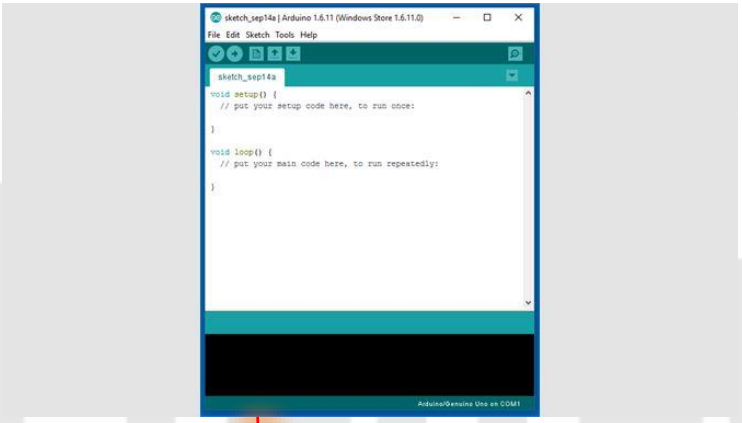
Steuerung der {code}.racer per Smartphone

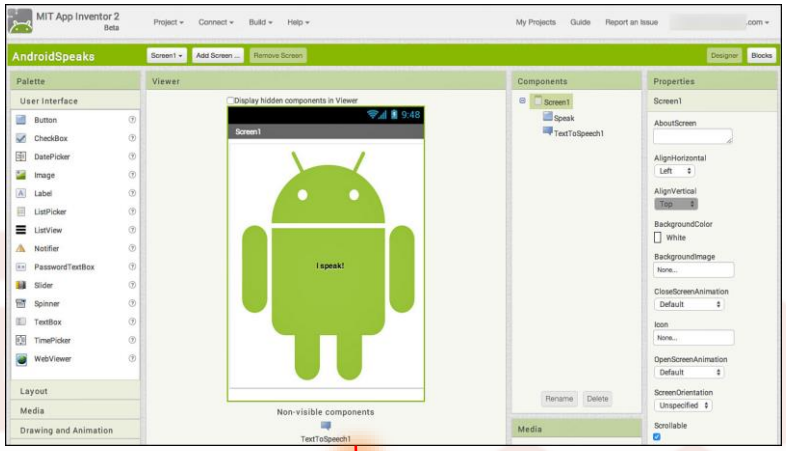
# Was wir brauchen:

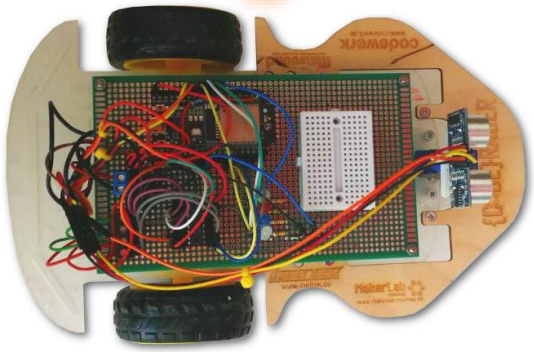
- Den fertig gebauten {code}.racer
- Ein Konto beim MIT App-Inventor haben wir für Euch angelegt
- Ein Smartphone mit Android (iOS evtl. später)
- Die AppInventor App
- Die Arduino-Umgebung

# Wie wir es machen:

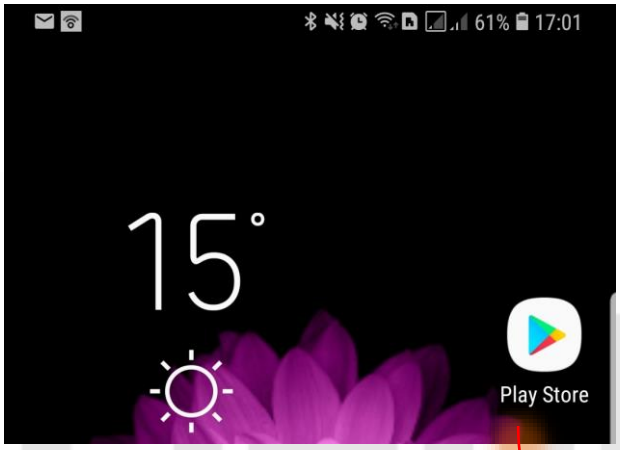
- Mit dem MIT AppInventor eine App für euer Handy machen
- Mit Arduino das Programm für den {code}.racer machen

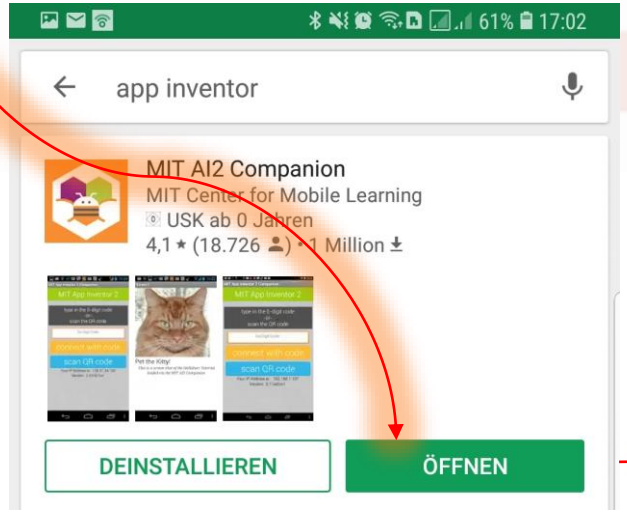


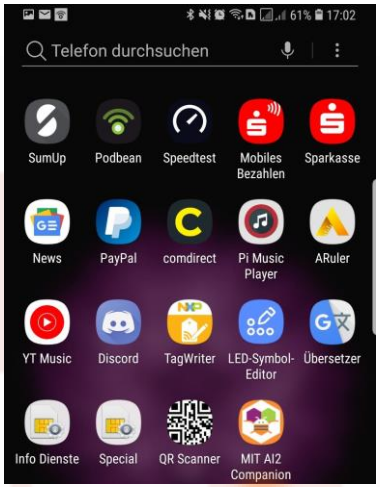



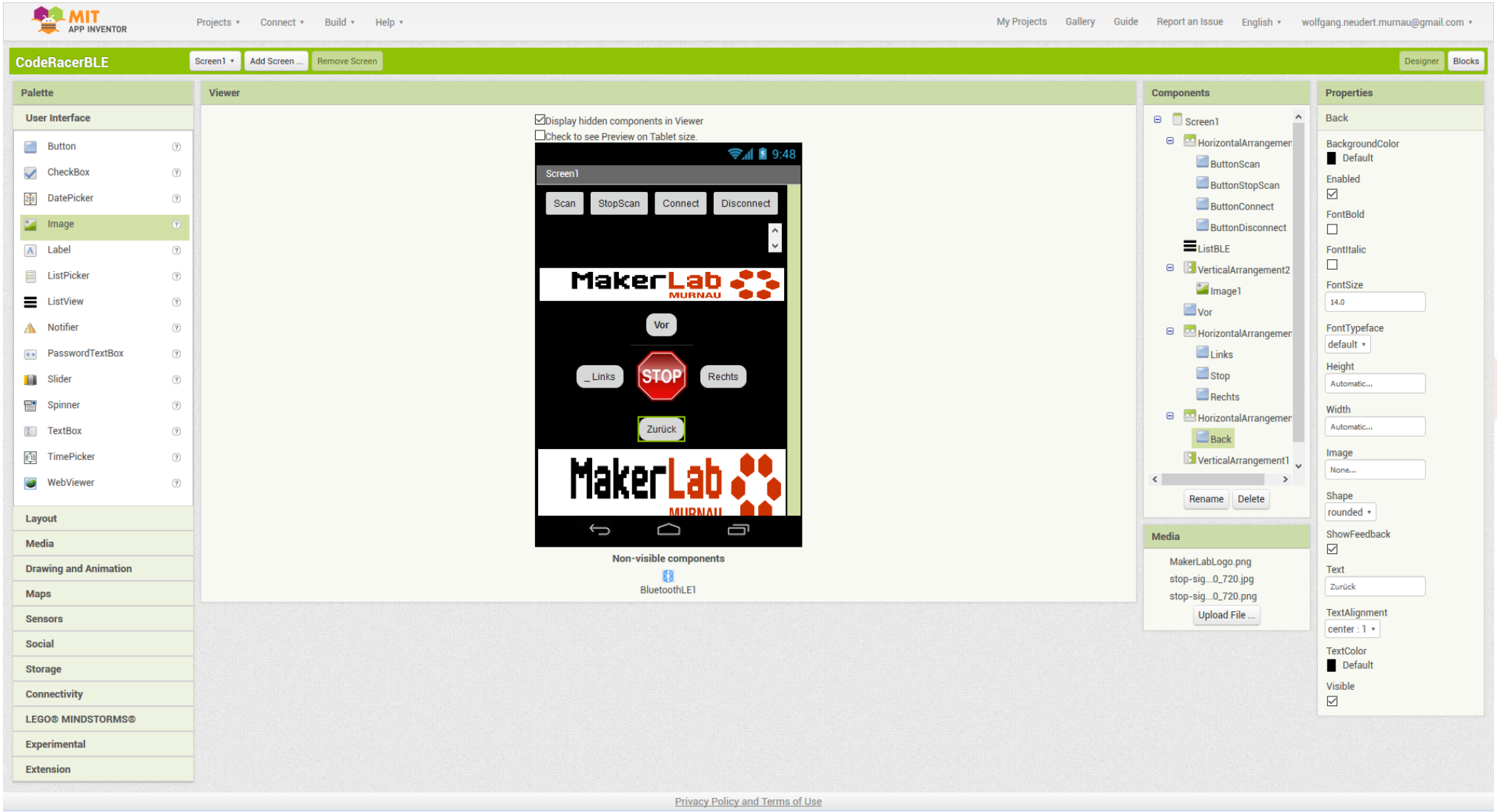
# MIT AppInventor App aus PlayStore laden und auf dem Smartphone installieren



# MIT AppInventor



# MIT AppInventor

Der Link zum AppInventor: http://ai2.appinventor.mit.edu
Eure Zugangsdaten beim AppInventor sind folgende:

eMail: coderacerX@itsblue.de
Passwort: MakerLab0X#

Das „X“ bitte durch die Nummer Eures Code-Racers ersetzen.

Beispiel:

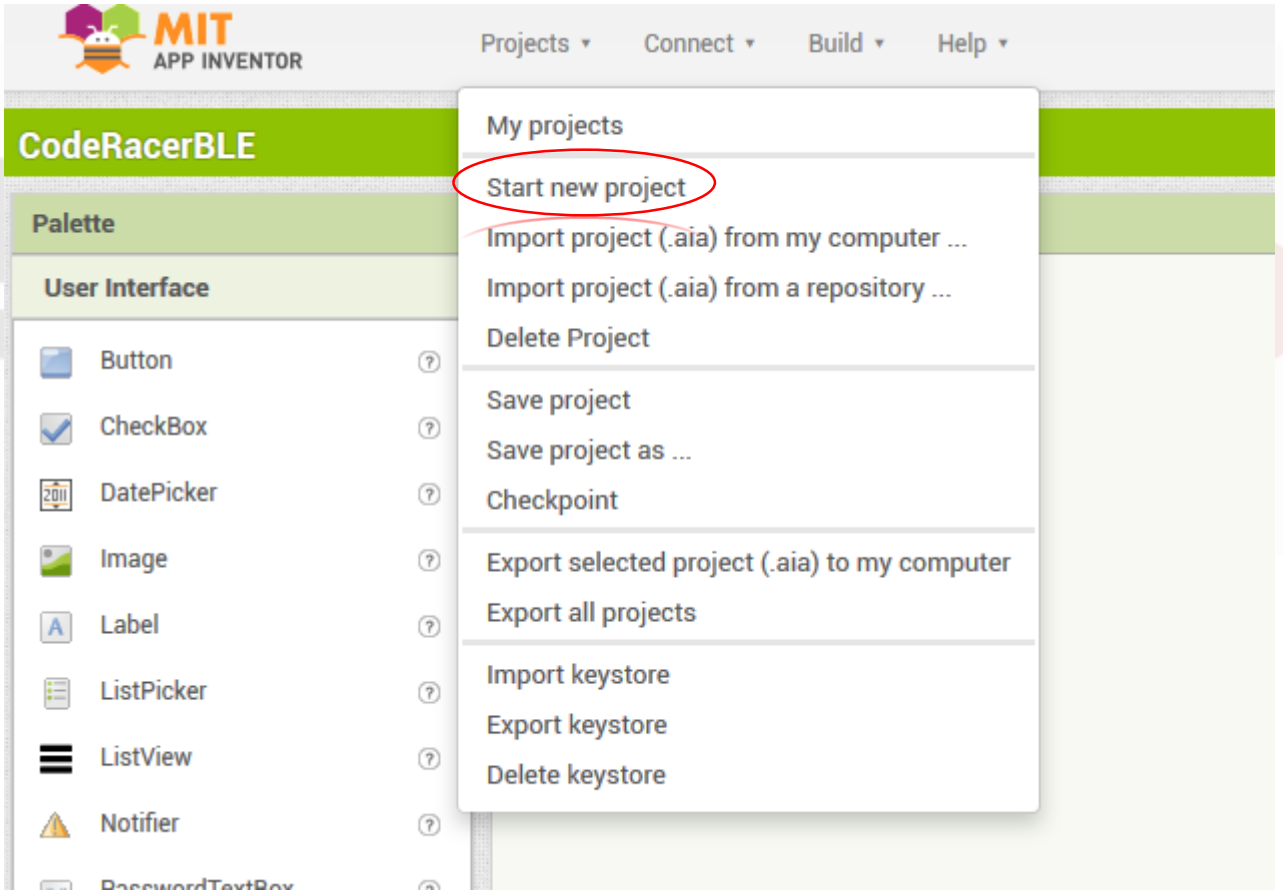
coderacer2@itsblue.de
MakerLab02#

Danach ein Projekt anlegen. Dann loslegen ☺

Siehe die Seiten danach…

# MIT AppInventor

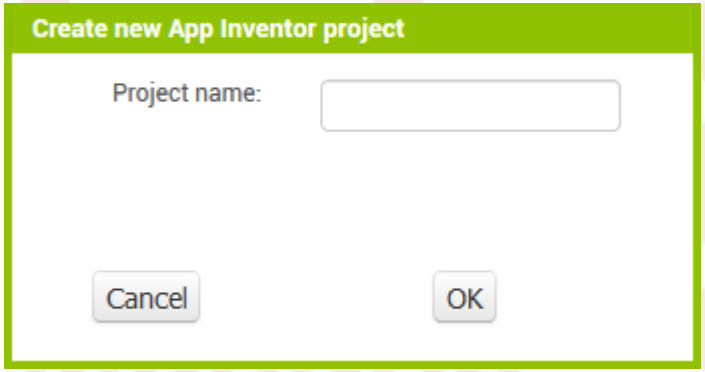
Zuerst ein neues Projekt anlegen:



# MIT AppInventor

Dem Projekt einen Namen geben:

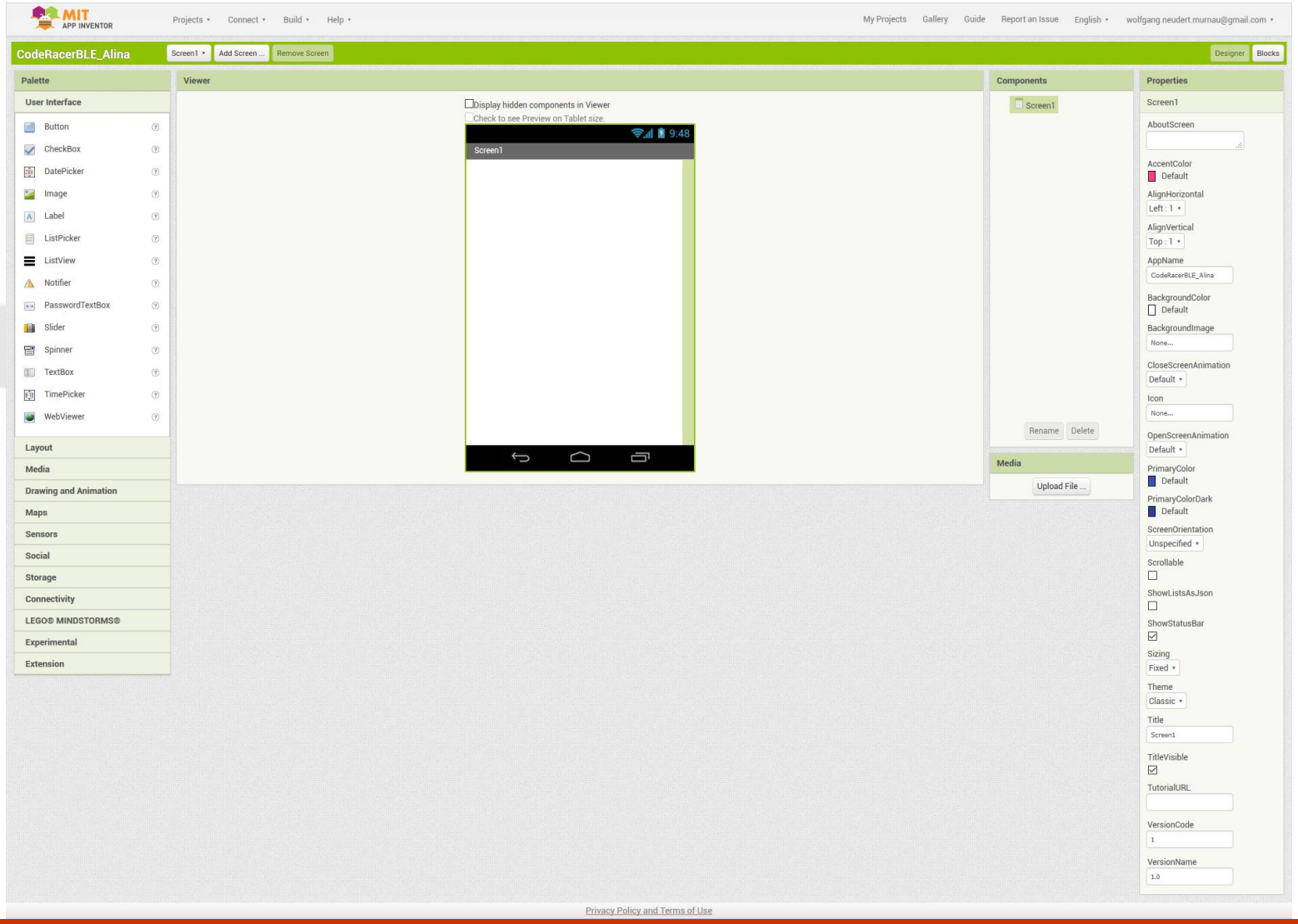
Zum Beispiel: CodeRacerBLE_Alina



# MIT AppInventor

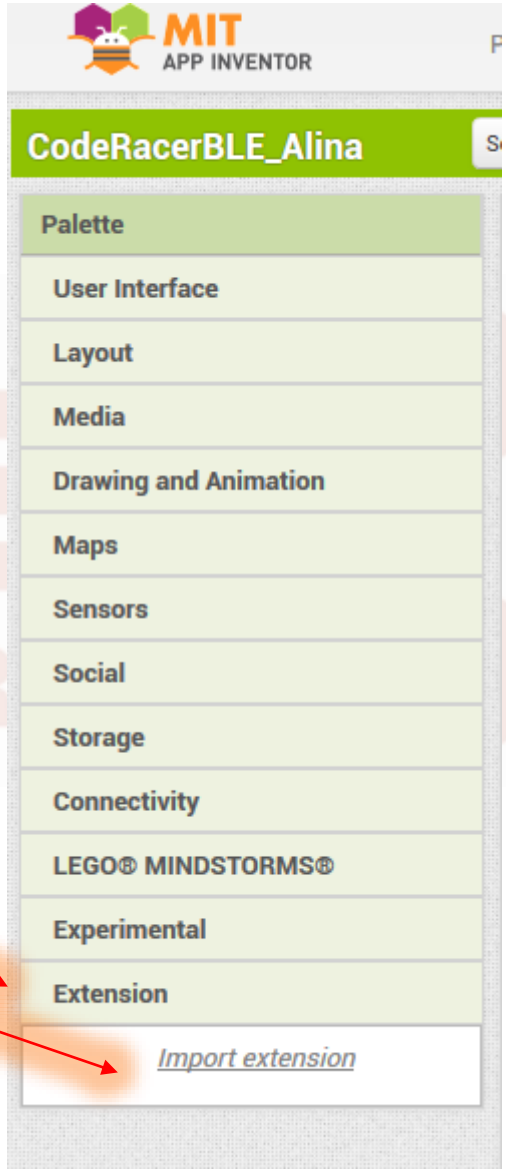
Es öffnet sich ein neuer leerer Arbeitsbereich

In der Mitte wird angezeigt, wie Eure App auf dem Smartphone (ungefähr) aussehen wird.



# MIT AppInventor

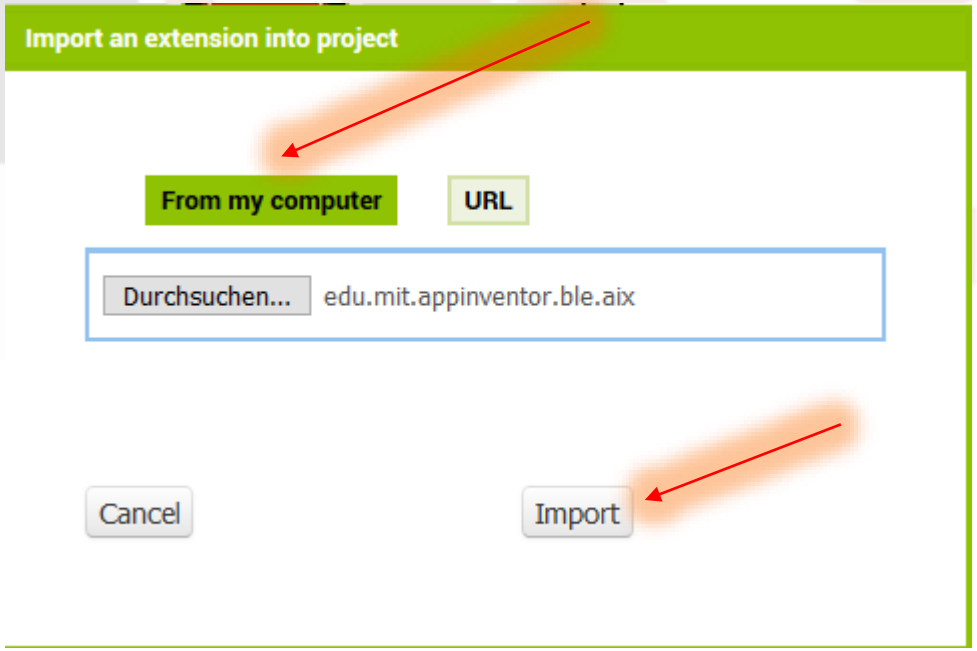
Zur Vorbereitung müssen wir noch eine „Extension“ hinzufügen. Dazu erst links auf „Extension“ klicken und dann auf „Import extension“



# MIT AppInventor

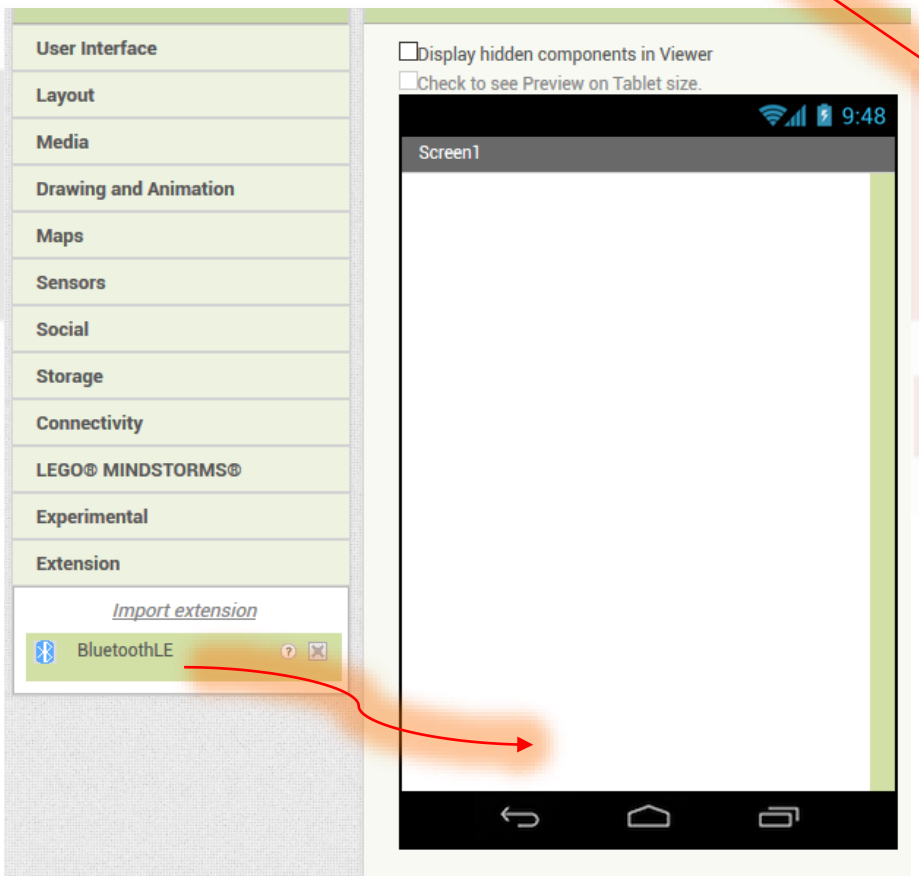
Wir importieren die Extension vom Computer. Die Datei sollte schon auf euren Rechner heruntergeladen worden sein.
Wenn nicht, könnt ihr sie hier runterladen:

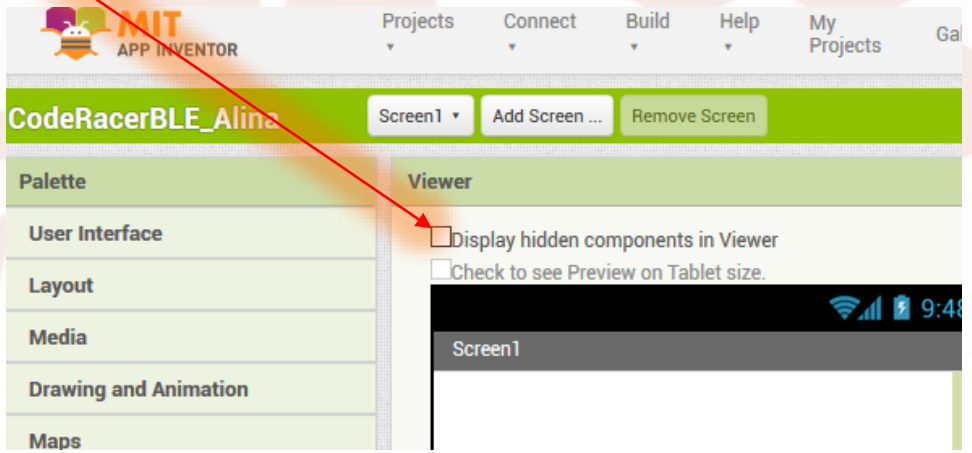
http://appinventor.mit.edu/extensions/



# MIT AppInventor

Die BluetoothLE Extension jetzt noch auf den „Bildschirm“ eures Smartphones ziehen.
Wenn nichts zu sehen ist,  dieses Kästchen anklicken.

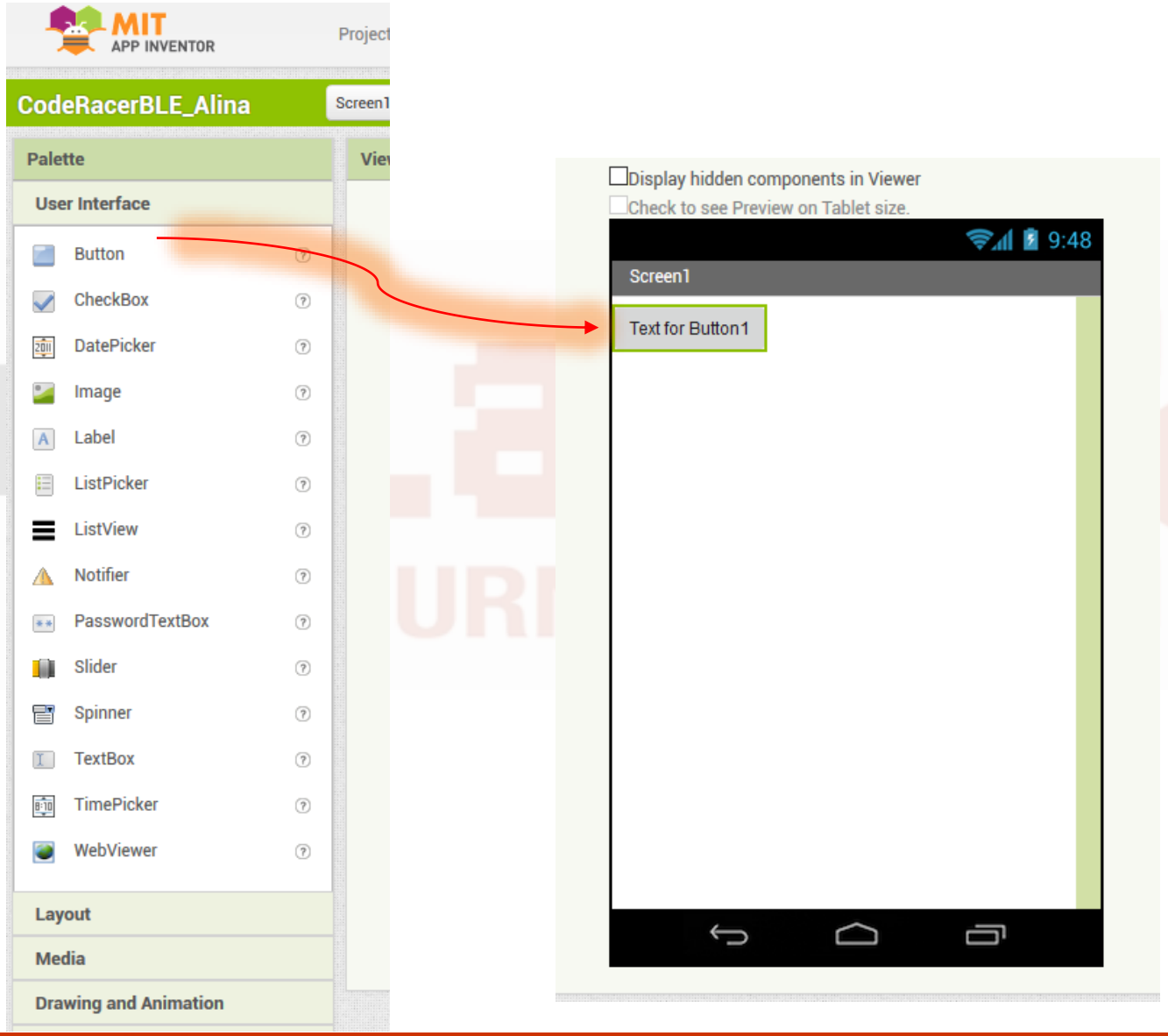


# MIT AppInventor

Jetzt können wir loslegen.
Auf der linken Seite ist unser „Werkzeugkasten“

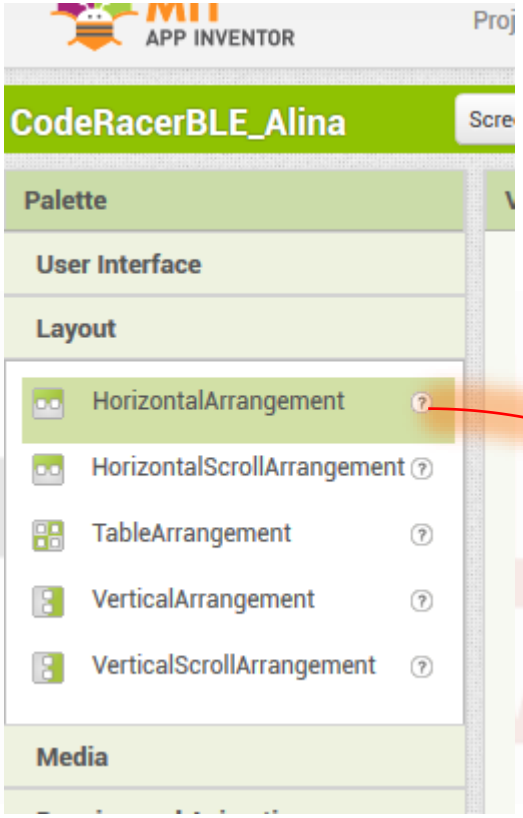
Diese Elemente wie Buttons, Bilder etc. können wir auf den Bildschirm des Telefons ziehen.
Einfach mit der Maus draufklicken, und auf den Bildschirm ziehen.

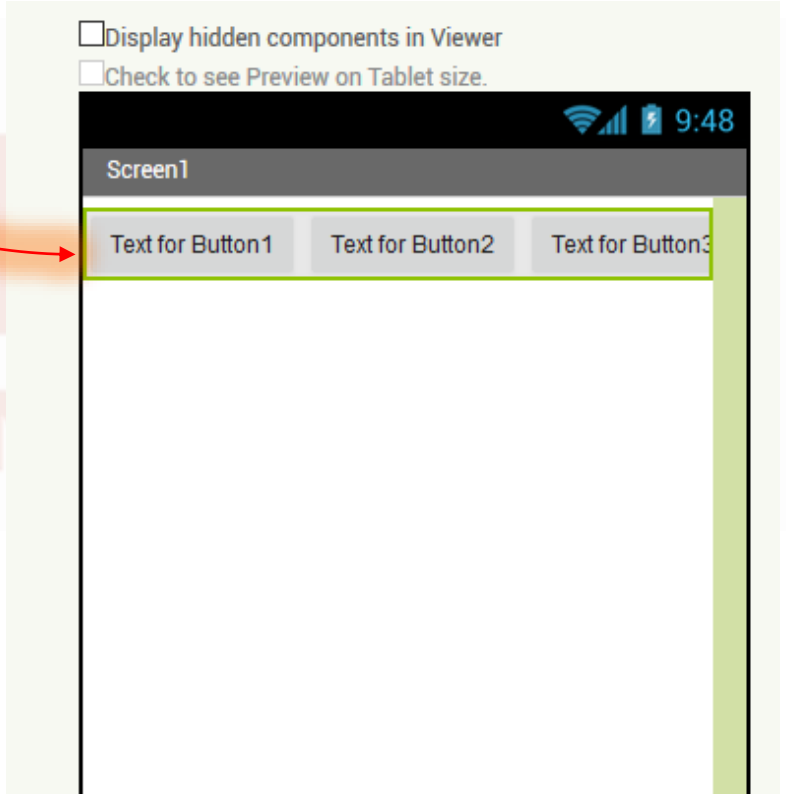


# MIT AppInventor

Diese Elemente sind etwas eigenwillig, und bleiben nicht unbedingt da wo man sie hinhaben will.

Deswegen gibt es die „Arrangements“, rechteckige Kästen, in die man zum Beispiel mehrere Buttons ablegen kann.



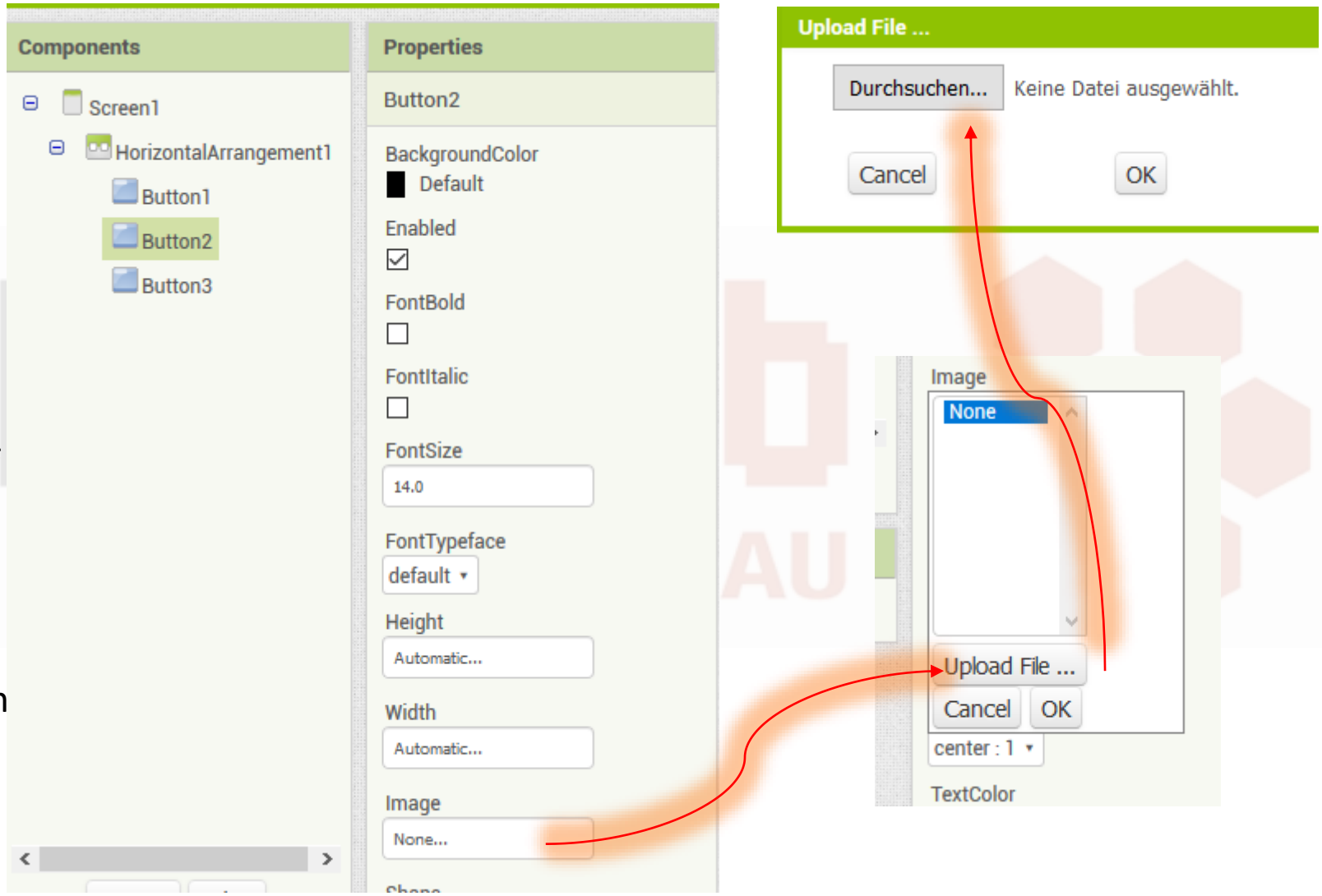



# MIT AppInventor

Buttons kann man schöner gestalten, wenn man zum Beispiel ein Bild hinterlegt. Dazu einen Button anklicken.

Dann auf Image klicken und dann auf upload file.

Auf „Durchsuchen“ klicken und das gewünschte Bild vom PC hochladen.



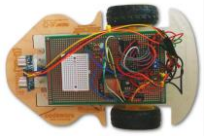
# MIT AppInventor

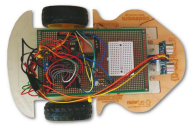
Damit haben wir den mittleren Button schon viel schöner gemacht.
So könnt ihr mit den andern Buttons auch weitermachen, zum Beispiel Pfeile nach rechts, links etc.

Wir haben ein paar Grafiken vorbereitet, ihr könnt aber auch selber welche aus dem Netz suchen oder euch welche designen…


Bilder aus dem Netz sind oft zu groß, ihr müsst sie dann verkleinern. Das geht zum Beispiel ganz einfach mit dem Programm „Irvanview“

IrfanView





# MIT AppInventor


Ihr könnt jetzt also die Oberfläche eurer App schön gestalten.

Aber bisher sind das ja nur bunte Bilder.

Damit etwas passiert wenn ihr auf so einen Button tippt, muss das Handy ja erstmal  wissen was überhaupt passieren soll.

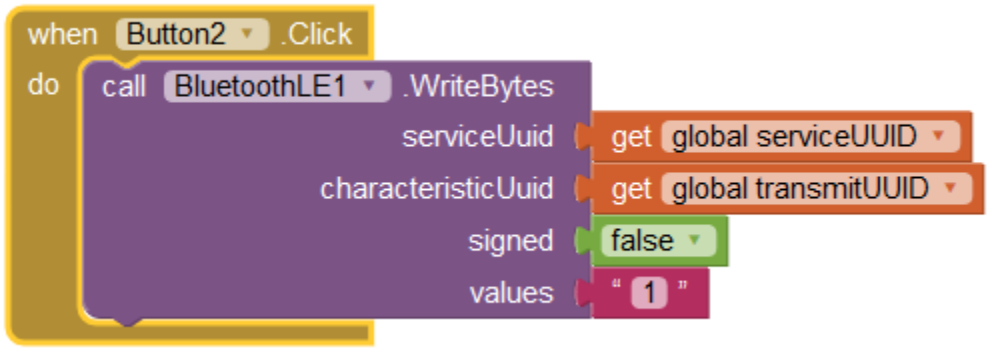
Es muss also wieder etwas Programmcode gemacht werden.

```
//*******************************************************
//* Binärzähler mit softwaremäßiger Tastenentprellung *
//* Wenn Taste gedrückt, PORTC hochzählen             *
//*******************************************************

#include <avr/io.h>
#define F_CPU 8000000    // Oszillator mit 8MHz
#include <util/delay.h>

int main (void)
{
    int i=0;

    DDRB = (0<<DDB6);   // PORTB Pin6 als Eingang
    DDRC = 0xFF;        // PORTC als Ausgang

    while(1)
    {
        if (PINB & (1<<PINB6))  // Wenn PINB6 = High
        {
            _delay_ms(10);      // Taste = gedrückt, 10ms warten

            if (!(PINB & (1<<PINB6)))   // Wenn PINB6 = Low
            {
                _delay_ms(10);  // Taste = losgelassen, 10ms warten
                PORTC = 0x00;   // PORTC löschen
                PORTC = i++;    // Neuer Wert am PORTC ausgeben
            }
        }
    }
}
```

# MIT AppInventor

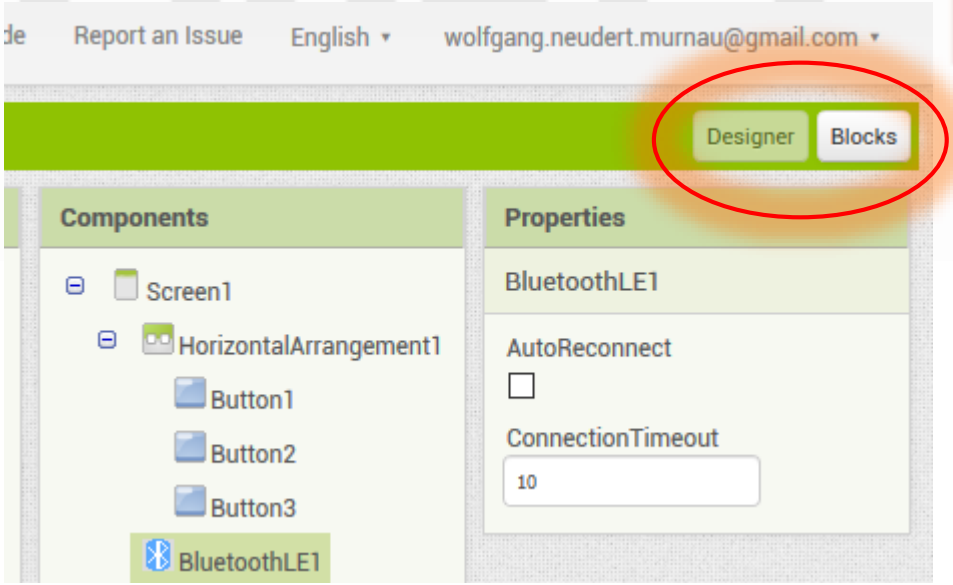

Diesmal müsst ihr allerdings nicht viel selbst tippen. Im AppInventor wird graphisch programmiert.

Das heißt, ihr zieht die Steuerelemente mit der Maus auf die Arbeitsfläche, und AppInventor kümmert sich um den Code.



# MIT AppInventor

Um zu programmieren, müsst ihr die Ansicht umschalten. Das macht ihr oben links mit den Buttons „Designer“ und „Blocks“

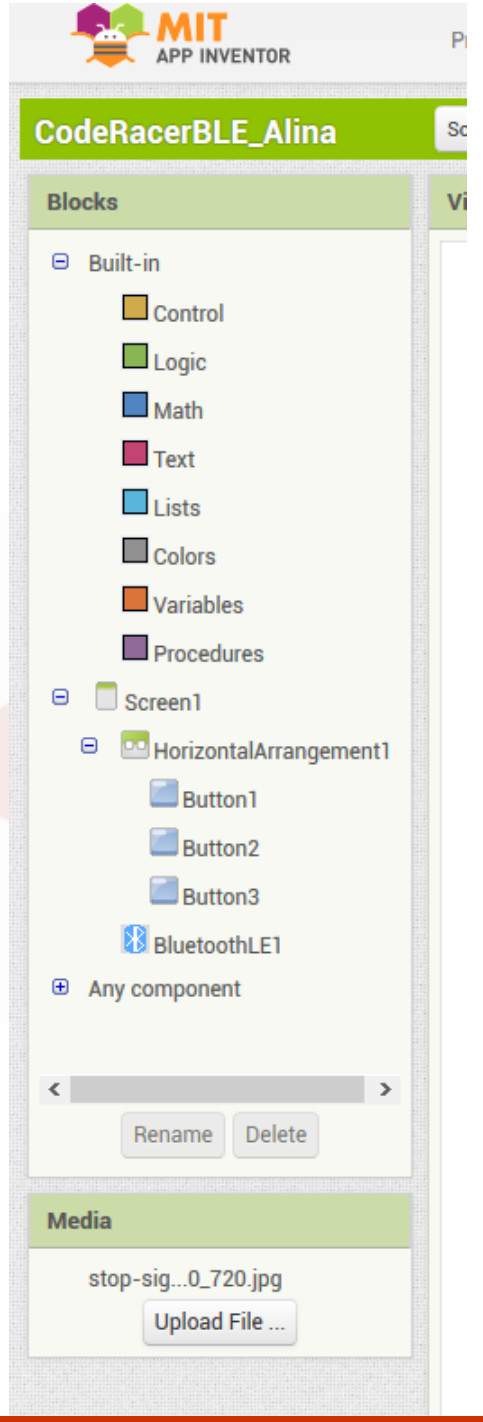



Auf der linken Seite gibt es wieder einen „Werkzeugkasten“

Ihr findet hier auch alle Elemente, die ihr in der Design-ansicht erzeugt habt.

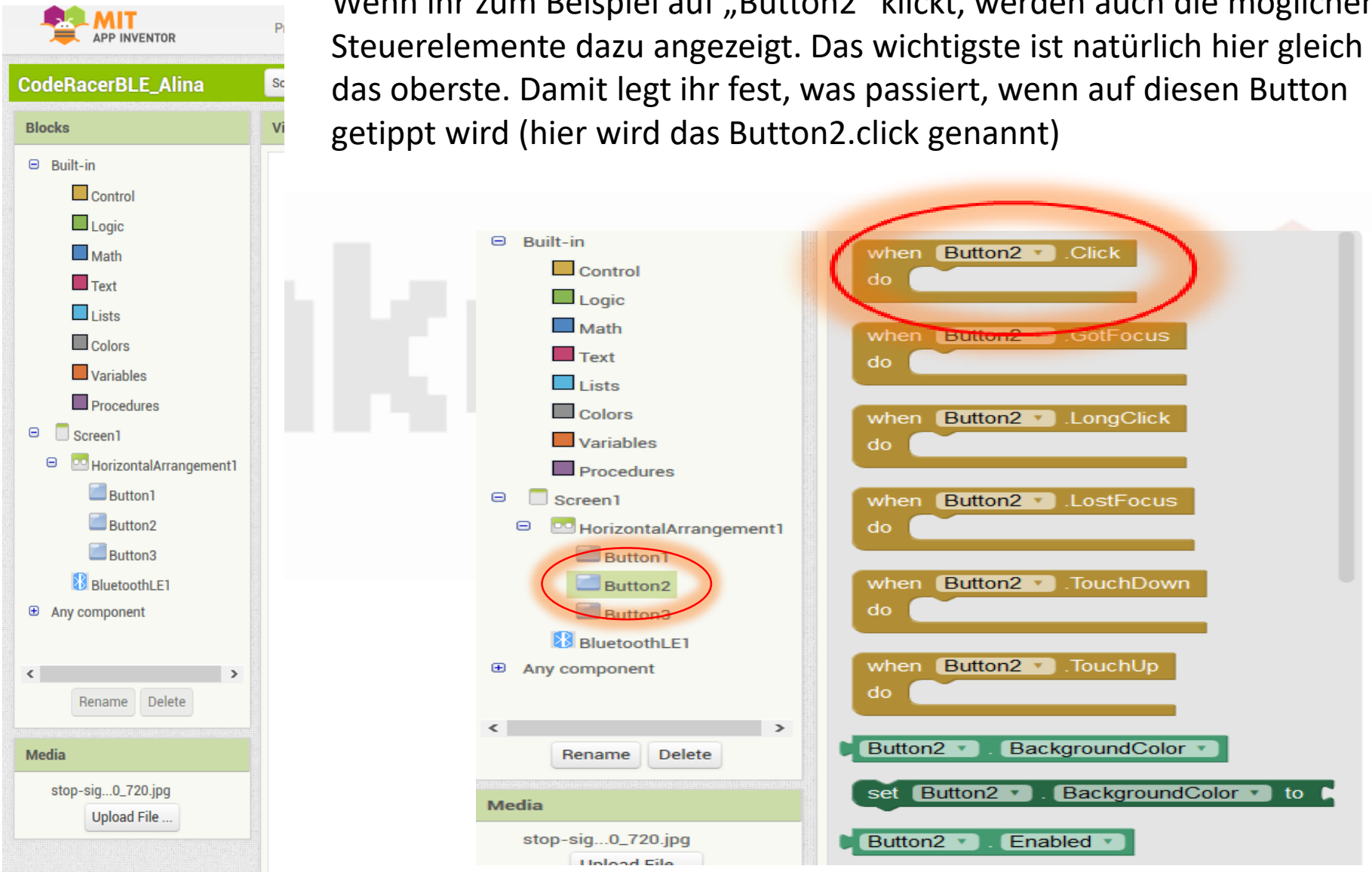
Wenn ihr auf eines dieser Element draufklickt, dann könnt ihr dafür nun den „code“ erzeugen.

Dieser „code“ wird dann ausgeführt, wenn ihr am Bildschirm auf den Button tippt.



# MIT AppInventor

Wenn ihr zum Beispiel auf „Button2“ klickt, werden auch die möglichen Steuerelemente dazu angezeigt. Das wichtigste ist natürlich hier gleich das oberste. Damit legt ihr fest, was passiert, wenn auf diesen Button getippt wird (hier wird das Button2.click genannt)



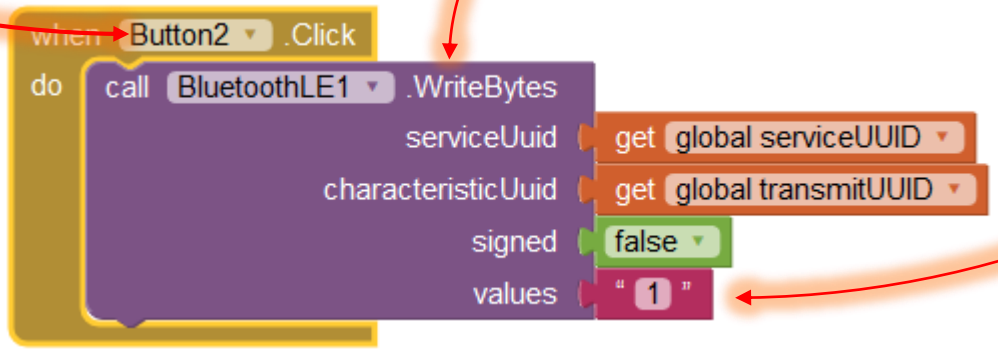
# MIT AppInventor

Wir wollen ja, dass eine Nachricht an den {code}.racer geschickt wird, wenn wir auf einen Button tippen.
Also müssen wir dem Telefon sagen, dass es unsere Nachricht über Funk, in unserem Fall über Bluetooth, an den {code}.racer schickt.

Hier kann man die vorher angelegten Buttons auswählen

Bei einem „click“ (=drauftippen) wird das violette Bluetooth-Modul im Handy aktiviert.

Die „1“ wollen wir an den racer senden



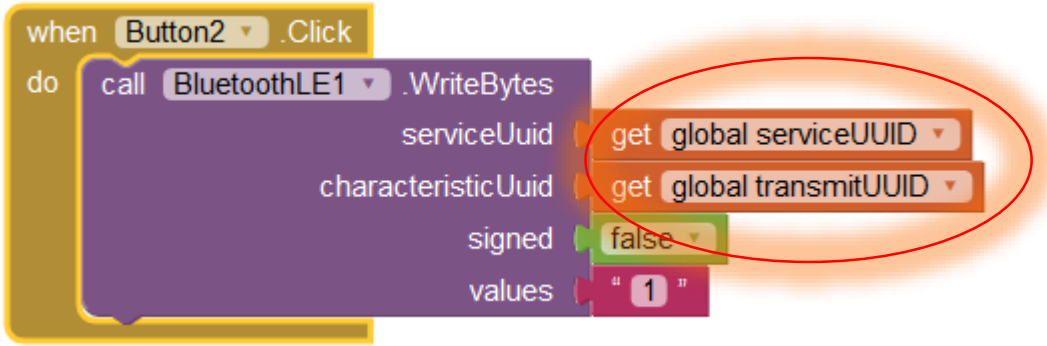
# MIT AppInventor

Ein bischen was zu Bluetooth
Wenn wir funken, müssen wir ja wissen, mit wem.
Deswegen müssen uns erst mal mit einem Gerät, oder hier mit einem CodeRacer, verbinden.

Das alleine reicht aber nicht. So modernes Funkzeug wie unser Bluetooth kann nämlich sehr viele verschiedene Dinge. Zum Beispiel ja auch Musik streamen auf einen Lautsprecher.

Und auch einfach nur Daten übertragen. Und auch mehrere Daten „gleichzeitig“ für verschiedene Anwendungen.

Das brauchen wir alles nicht, aber wir müssen dem Bluetooth sagen, wer gemeint ist.
Das macht man mit eindeutigen Zahlen, die hier UUID heißen.



# MIT AppInventor

Diese Zahlen oder diese UUIDs könnten wir uns natürlich selber ausdenken.

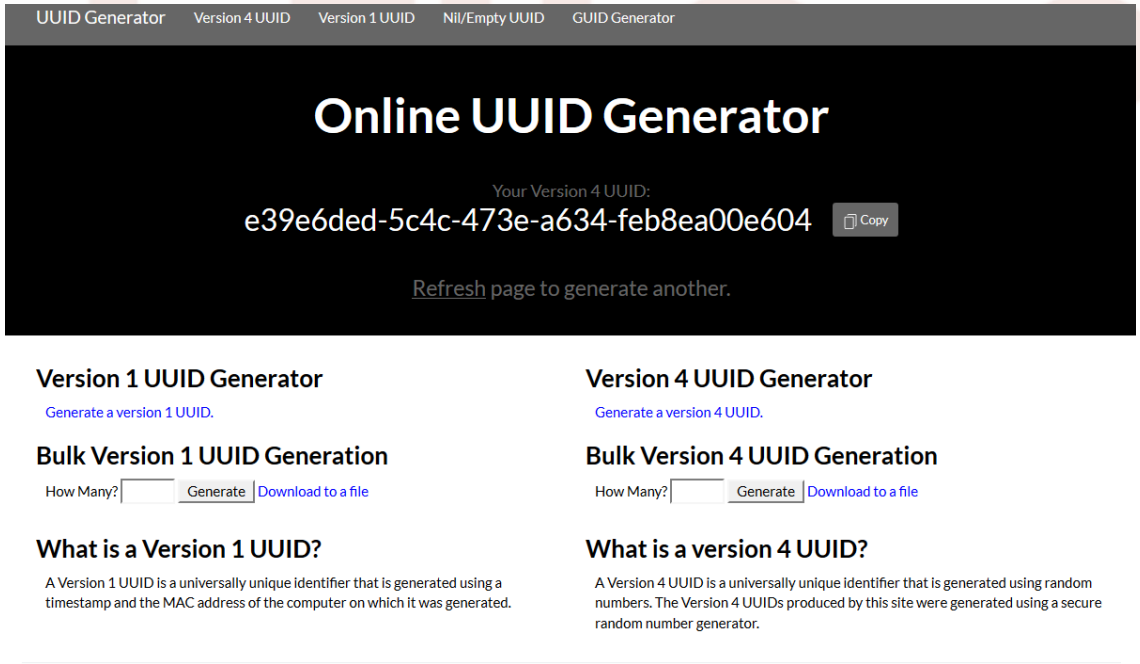
Zum Beispiel: unsere Musikübertragung erhält die Zahl 1, die erste Datenübertragung die Zahl 2, die zweite Datenübertragung die Zahl 3, und so weiter…

Aber wie Ihr Euch sicher schon gedacht habt, macht man es nicht so einfach. ☺

Man will Nummern, die garantiert nur einmal vorhanden sind.

Die kann man sich im Netz hier holen:

https://www.uuidgenerator.net/

# Das nächste {code}.racer - Rennen

Wir haben viele Ideen und planen mit euch folgendes:

- Mit Ultraschall durch die Rennstrecke kommen ( Wie bisher )
- Mit dem Smartphone den {code}.racer steuern
- Einen Gegenstand in der Rennstrecke finden mit Kamera (kommt noch)
- Mit Infrarot einer Linie durch die Rennstrecke folgen (kommt noch)
- Steuerung mit Alexa (kommt noch)

Beim nächsten {code}.racer – Rennen können wir dann davon zwei oder drei Disziplinen auswählen und damit mehrer Durchgänge beim Rennen machen.

# Was hat und was kann ein {code}.racer

Teil

| Antriebsmotoren | Servo | Ultraschallsensor |
|---|---|---|

Funktionen

**RACER_ANHALTEN**
- Motor links und rechts aus

**SERVO_MITTE**
- Ultraschallsensor nach vorn

**ABSTAND MESSEN**
- Abstand vorn (Servo in der Mitte)

**RACER_VORWAERTS**
- Motor links und rechts vorwärts

**SERVO_LINKS**
- Ultraschallsensor nach links drehen

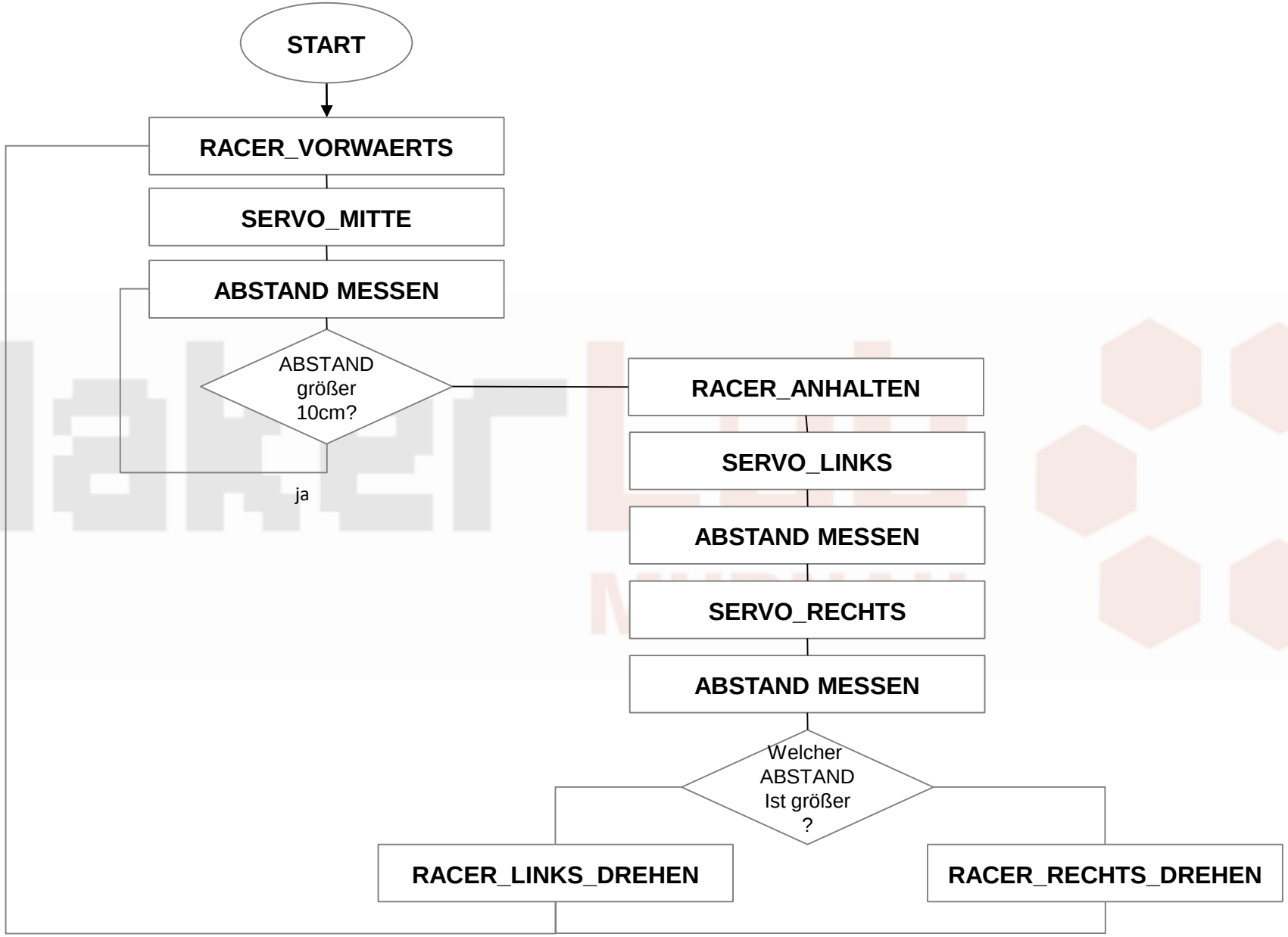
**RACER_RUECKWAERTS**
- Motor links und rechts rückwärts

**SERVO_RECHTS**
- Ultraschallsensor nach rechts drehen

**RACER_RECHTS_DREHEN**
- beide Motoren gleichzeitig:
  . links kurz vorwärts
  . rechts kurz rückwärts

**RACER_LINKS_DREHEN**
- beide Motoren gleichzeitig:
  . links kurz rückwärts
  . rechts kurz vorwärts

# Wie fährt ein {code}.racer ohne anzuecken?



# {code}.racer

Was man immer machen muss:

Richtigen Controller in Arduino auswählen:

- ESP32 DevKit

Richtigen Port in Arduino auswählen:

- Zum Beispiel COM9

Pins richtig einstellen:

- `pinMode(Pinnummer, OUTPUT);`
  Oder
- `pinMode(Pinnummer, INPUT);`

Die Pinnummer vom Modul ablesen.
Anstatt die Pinnummer zu verwenden, kann man dem Pin auch einen Namen geben:

```
#define US_TRIG 13
```

# {code}.racer

Wie steuere ich den Servo an?

```
#include <ESP32Servo.h>
```

Ein Servo-objekt erzeugen

```
Servo myservo;
```

Pin für Steuerung festlegen:

```
myservo.attach(SERVOPIN);
```

servo_angle ist der Winkel, auf den gedreht werden soll
( 0° - 180°)

```
myservo.write(servo_angle);
```

# {code}.racer

Wie steuere ich den Ultraschall an?

Der Ultraschall wird mit 2 Pins angeschlossen:

```
#define US_TRIG 13
#define US_ECHO 12
```

Diese Pins müssen Ausgänge werden:

```
pinMode(US_TRIG, OUTPUT);
pinMode(US_ECHO, INPUT);
```

Die Messung wird gestartet, in dem man auf dem TRIG-Pin einen kurzen Puls erzeugt:

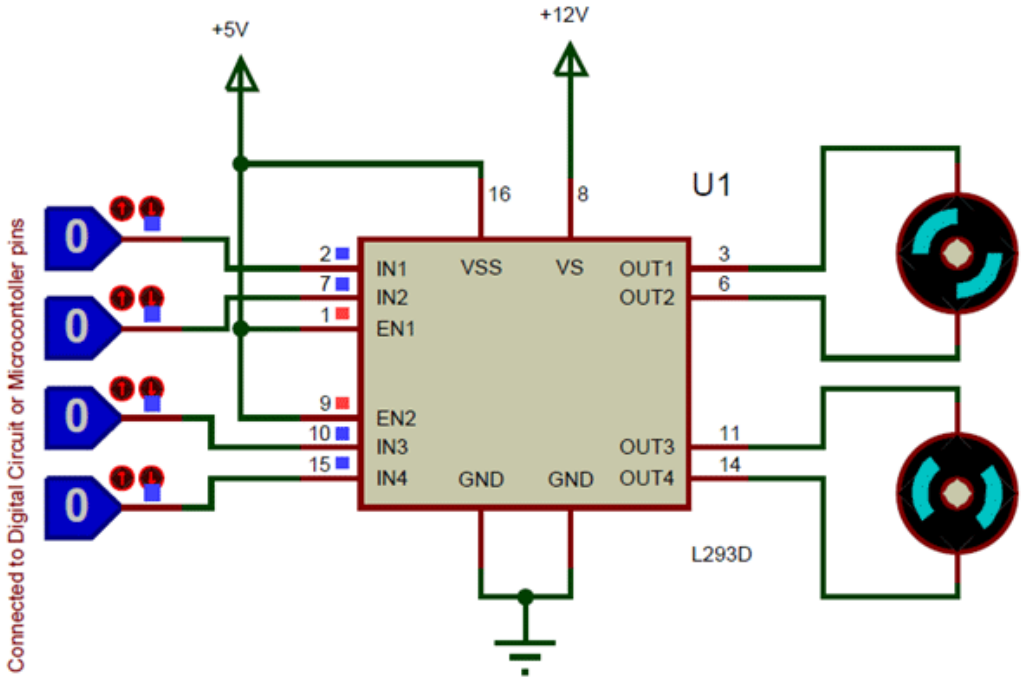
```
digitalWrite(US_TRIG,LOW);
delayMicroseconds(2);
digitalWrite(US_TRIG,HIGH);
delayMicroseconds(10);
digitalWrite(US_TRIG,LOW);
```

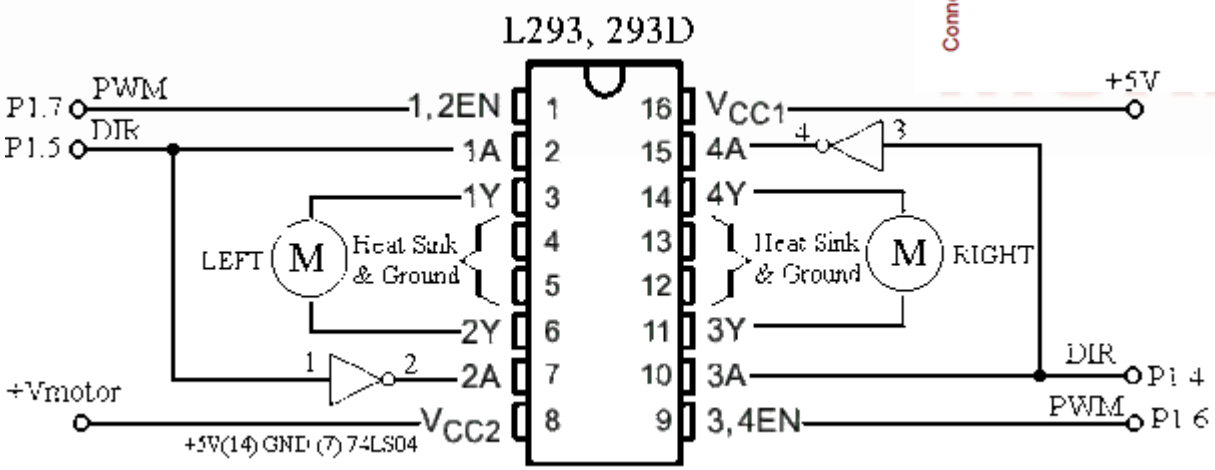
Dann wird gemessen, wie lange es dauert bis der ECHO-Pin auf „HIGH“ geht:

```
echo_duration = pulseIn(US_ECHO,HIGH);
```

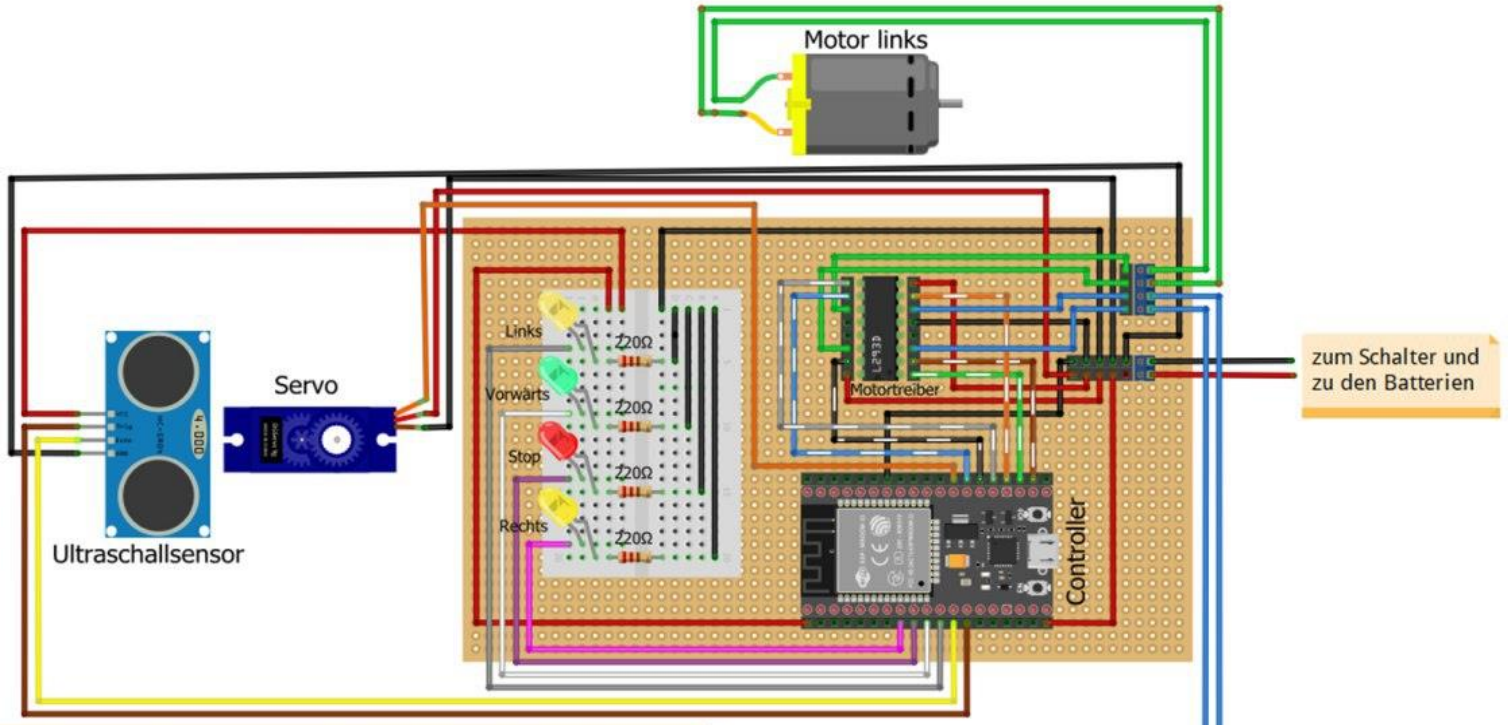
# {code}.racer

Wie steuere ich die Motoren an?



# {code}.racer




Anschlüsse ULTRASCHALLSENSOR -> CONTROLLER, Drahtfarbe im Bild, Name im Programm:

| | | | | |
|---|---|---|---|---|
| Trig | -> | 13 (GPIO12) | braun | US_TRIG |
| Echo | -> | 12 (GPIO14) | gelb | US_ECHO |

Anschluss SERVO -> CONTROLLER, Drahtfarbe im Bild, Name im Programm:

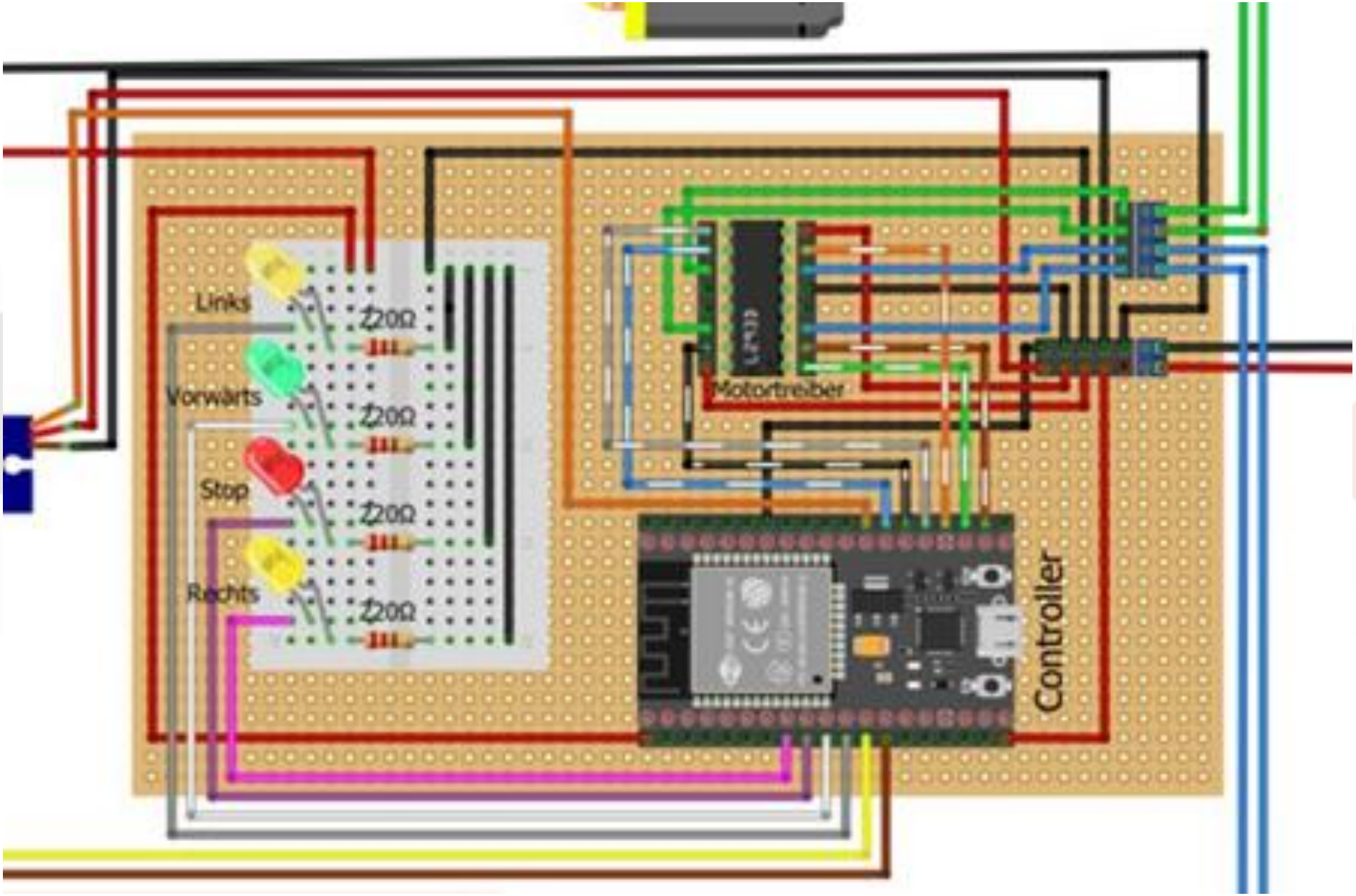
| | | | | |
|---|---|---|---|---|
| Pulse (orange)-> | | 27 (GPIO 16) | orange | SERVOPIN |

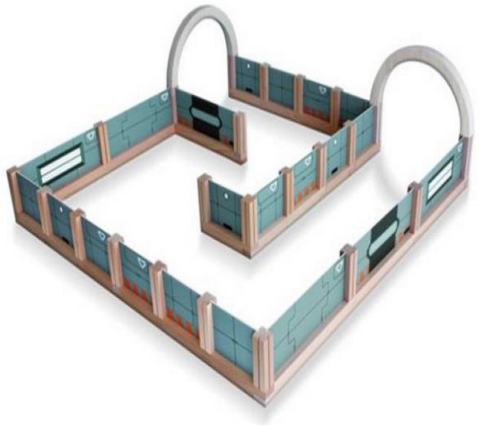
Anschlüsse MOTORTREIBER -> CONTROLLER, Drahtfarbe im Bild, Name im Programm:

| | | | | |
|---|---|---|---|---|
| Enable1 | -> | 22 (GPIO8) | grün-weiß | MOTORRE_SPEED |
| In1 | -> | 21 (GPIO7) | braun-weiß | MOTORRE_FWRD |
| In2 | -> | 23 (GPIO15) | orange-weiß | MOTORRE_BACK |
| Enable2 | -> | 24 (GPIO2) | grau-weiß | MOTORLI_SPEED |
| In3 | -> | 26 (GPIO4) | blau-weiß | MOTORLI_BACK |
| In4 | -> | 25 (GPIO0) | schwarz-weiß | MOTORLI_FWRD |

Anschlüsse LEDs -> CONTROLLER, Drahtfarbe im Bild, Name im Programm:

| | | | | |
|---|---|---|---|---|
| Vorwärts | -> | 10 (GPIO26) | weiß | LED_VORWAERTS |
| Stop | -> | 9 (GPIO25) | rosa | LED_STOP |
| Links | -> | 11 (GPIO27) | grau | LED_LINKS |
| Rechts | -> | 8 (GPIO33) | pink | LED_RECHTS |


Links
Vorwärts
Stop
Rechts
220Ω
220Ω
220Ω
220Ω
Motortreiber
Controller

{CODE} RACE!

MakerLab